向量数据库

SDK 参考

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

SDK 参考
Python SDK
SDK 准备
Python SDK Demo
新建 Client
Database 操作
新建 Database
删除 Database
查询 DataBase
检查 Database 是否存在
Collection 操作
新建 Collection
删除 Collection
查询 Collection 列表
查询指定 Collection
检查 Colleciton 是否存在
清空 Collection
CollectionView 操作
新建 CollectionView
删除 CollectionView
查询 CollectionView 列表
查询指定 CollectionView
清空 CollectionView 数据
Alias 操作
设置别名
删除别名
Document 操作
插入数据
精确查询
基于向量数据相似度检索
基于 Doc ID 相似度检索
基于原始文本相似度检索
单路稀疏向量相似度检索
稠密向量与稀疏向量高阶混合检索
删除 Document

版权所有：腾讯云计算（北京）有限责任公司

更新 Document
查询文档数量
上传文件
获取文件图片 URL
DocumentSet 操作
上传文件
获取文件内容
获取 Chunks
精确查询文件信息
文件内容检索
删除指定文件
更新文件
Index 操作
重建索引
新增索引
删除索引
修改向量索引
User 管理
新建用户
为用户授权
撤销用户权限
查询指定用户权限
查看用户列表
删除用户
重置用户密码
Java SDK
SDK 准备
Java SDK Demo
新建 Client
Database 操作
新建 Database
检查 Database 是否存在
删除 Database
查询 Database
Collection 操作
新建 Collection
检查 Collection 是否存在
删除 Collection

查询 Collections 列表
查询指定 Collection
清空 Collections 数据
CollectionView 操作
创建 CollectionView
删除 CollectionView
查询 CollectionView 列表
查询指定的 CollectionView
清空 CollectionView 数据
Alias 操作
设置别名
删除别名
Document 操作
插入数据
精确查询
向量数据相似性检索
Doc ID 相似性检索
文本信息相似性检索
单路稀疏向量相似性检索
稠密向量与稀疏向量高阶混合检索
删除 Document
更新数据
查询文档数量
上传文件
获取文件图片 URL
DocumentSet 操作
上传文件
获取文件内容
获取 Chunks
精确查询文件
文件内容相似性检索
删除指定文件
更新文件
Index 操作
重建索引
新建索引
删除索引
修改向量索引

版权所有：腾讯云计算（北京）有限责任公司

User 管理
新建用户
为用户授权
撤销用户授权
查询指定用户
查看用户列表
删除用户
重置用户密码
Go SDK
SDK 准备
GO Demo
连接数据库实例
管理 Database
检查 Database 是否存在
管理 Base 类 Database
管理 AI 类 Database
查询集群数据库
管理 Collection
新建 Collection
检查 Collection 是否存在
查询 Collection 配置
查询指定 Collection 配置
清空 Collection 数据
管理 Collection 别名
删除指定 Collection
管理 CollectionView
新建 CollectionView
查询 CollectionView 配置
查询指定 CollectionView
清空 CollectionView 数据
管理 CollectionView 别名
删除 CollectionView
管理 Document 数据
写入数据
精确查询
基于 ID 相似性检索
基于向量数据相似性检索
基于原始文本相似性检索

版权所有：腾讯云计算（北京）有限责任公司

单路稀疏向量相似性检索
稠密向量与稀疏向量高阶混合检索
更新 Document 数据
删除 Document 数据
查询文档数量
上传文件
获取文件图片 URL
管理 DocumentSet 数据
基于文件写入数据
获取文件内容
获取 Chunks
查询文件信息
相似性检索文件内容
更新文件信息
删除指定文件
管理 Index
重构索引
新增索引
删除索引
修改向量索引
User 管理
新建用户
为用户授权
撤销用户授权
查询指定用户
查看用户列表
删除用户
重置用户密码
C++ SDK
HTTP API
使用前阅读
Database
create
drop
list
Collection
create
drop

- list
- describe
- truncate

CollectionView
- create
- drop
- list
- describe
- truncate

Alias
- set
- delete

Document
- upsert
- query
- search
- hybridSearch
- fullTextSearch
- delete
- update
- count
- uploadUrl
- getImageUrl

DocumentSet
- uploadUrl
- get
- getChunks
- query
- search
- delete
- update

Index
- rebuild
- add
- drop
- modifyVectorIndex

User
- create

- grant
- revoke
- describe
- list
- drop
- changePassword

错误码

SDK 参考
Python SDK
SDK 准备

最近更新时间：2024-10-24 19:13:12

腾讯云向量数据库（Tencent Cloud VectorDB）的 Python SDK 是将 HTTP API 封装成易于使用的 Python 函数或类。开发者可以通过 Python SDK 更加方便地操作数据库。

> **说明：**
> 使用 Python SDK 之前，请您先了解腾讯云向量数据库产品设计的逻辑结构。具体信息，请参见 逻辑结构简介 。

语言	语言版本	SDK 下载	SDK 源码
Python	Python 3.6 及以上版本 推荐使用 3.8 及以上版本	• 执行如下命令，可直接安装最新版本。 `pip3 install tcvectordb` • 执行如下命令，可获取历史版本。 `pip3 install tcvectordb==`*历史版本*	vectordatabase-sdk-python

Python SDK Demo

最近更新时间：2025-05-26 19:31:32

类别	场景描述	SDK Demo
写入向量数据并检索	操作 Base 类数据库，直接写入向量数据并进行精确查询与相似度检索。	example.py
写入原始文本并检索（Embedding）	操作 Base 类数据库，应用 Embedding 模型写入或更新原始文本，并进行精确查询或相似度检索。	exampleWithEmbedding.py
使用 AI 套件上传文件	基于 Collection 或 CollectionView 上传文件并进行内容相似性检索。	• ai_db_example.py • upload_file.py
使用稀疏向量	操作 Base 类数据库，应用 BM25 写入稀疏向量，使用语义检索（稠密向量）和关键字检索（稀疏向量）两路混合召回。	sparse_vector_example.py

版权所有：腾讯云计算（北京）有限责任公司

新建 Client

最近更新时间：2025-07-01 10:19:42

接口定义

- **RPCVectorDBClient()** 用于通过 RPC（Remote Procedure Call，远程过程调用）的请求方式创建一个向量数据库的客户端对象。
- **VectorDBClient()** 为了兼容旧版本而保留，用于通过 HTTP I/O 请求方式创建一个向量数据库的客户端对象。

说明：

- Python SDK 为 1.3.3 及以上版本支持 RPC（Remote Procedure Call，远程过程调用）通信，在多线程场景，RPC 可以提供比 HTTP I/O 更好的并发请求能力。同时，为了确保兼容性和性能，Python SDK 需要在 Python 3.8 或更高版本的环境下运行。
- RPCVectorDBClient() 支持创建连接池，特别适合客户端节点较少但 collection 副本较多（尤其是单 Shard 多副本）的场景。

RPC 请求（推荐）

```
class RPCVectorDBClient(
    url: Any | None = None,
    username: str = '',
    key: str = '',
    read_consistency: ReadConsistency =
ReadConsistency.EVENTUAL_CONSISTENCY,
    timeout: int = 10,
    adapter: HTTPAdapter = None,
    pool_size: int = 10
)
```

HTTP IO 请求（兼容 Restful 旧版本）

```
class VectorDBClient(
    url: Any | None = None,
```

```
    username: str = '',
    key: str = '',
    read_consistency: ReadConsistency =
ReadConsistency.EVENTUAL_CONSISTENCY,
    timeout: int = 10,
    adapter: HTTPAdapter = None,
    pool_size: int = 10
)
```

使用示例

RPC 请求（推荐）

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency

client = tcvectordb.RPCVectorDBClient(
            url='http://10.0.X.X:80',
            username='root',
            key='eC4bLRy2va******************************',
            read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY,
            timeout=30
          )
```

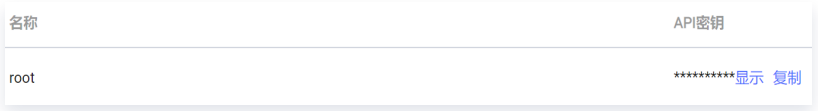
HTTP IO 请求（兼容 Restful 旧版本）

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency

client = tcvectordb.VectorDBClient(url='http://10.0.X.X:80',
            username='root',
            key='eC4bLRy2va******************************',
            read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY,
timeout=30
```

```
)
```

入参描述

参数名	参数含义	是否必选	获取方式
url	客户端所需连接的向量数据库服务端访问地址。	是	获取向量数据库实例内网 IP 地址与端口，请登录 向量数据库控制台，在**实例详情**页面**网络信息**区域直接复制**访问地址**。具体操作，请参见 查看实例信息 。默认端口为80。
username	客户端访问向量数据库服务端的账号。	是	数据库当前仅支持 root 账号。
key	客户端访问向量数据库服务端的 API 密钥，用于进行身份认证。	是	请登录 向量数据库控制台，在**密钥管理**页面直接复制密钥。具体操作，请参见 密钥管理 。 
read_consistency	设置读一致性。	否	取值如下所示，默认为 EVENTUAL_CONSISTENCY。 • ReadConsistency.STRONG_CONSISTENCY：强一致性。 • ReadConsistency.EVENTUAL_CONSISTENCY：最终一致性。
pool_size	设置连接池最大连接数	否	正整数，建议设置为数据节点个数的2-5倍。
timeout	请求超时时间。	是	• 单位：秒。 • 默认值：10。

			• 取值范围：大于等于0。若 timeout 设置为小于0或为 null，系统会自动赋值为默认值。

版权所有：腾讯云计算（北京）有限责任公司

Database 操作
新建 Database

最近更新时间：2024-09-27 18:00:11

接口定义

- **create_database() 用于创建一个 Base 类的向量数据库，存储向量数据或写入文本。**
- **create_ai_database() 用于创建一个 AI 类向量数据库，上传并存储文件。**

create_database()

```
def create_database(
    database_name: str,
    timeout: float | None = None
) -> Database
```

create_ai_database()

```
def create_ai_database(
    database_name: str,
    timeout: float | None = None
) -> AIDatabase
```

使用示例

基于 新建 Client 创建的客户端对象，创建数据库。

创建 Base 类数据库

版权所有：腾讯云计算（北京）有限责任公司

```
db = client.create_database(database_name='db-test')
```

创建 AI 类数据库

```
db = client.create_ai_database(database_name='db-test-ai')
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database_name	是	设置 Database 名称。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
timeout	否	请求超时时长。	• 单位：秒 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

删除 Database

最近更新时间：2024-09-27 18:00:11

接口定义

- **drop_database()** 用于删除一个 Base 类向量数据库。
- **drop_ai_database()** 用于删除一个 AI 类向量数据库。

drop_database()

```
def drop_database(
    self,
    database_name: str,
    timeout: Optional[float] = None) -> dict
```

drop_ai_database()

```
def drop_ai_database(
    database_name: str,
    timeout: float | None = None
) -> Dict[str, Any]
```

使用示例

> **警告：**
> 执行 drop 操作将会彻底删除指定数据库下所有数据。在操作之前，请务必谨慎考虑。

基于 新建 Client 创建的客户端对象，创建数据库。

删除 Base 类数据库

```
db = client.drop_database(database_name='db-test')
```

删除 AI 类数据库

```
db = client.drop_ai_database(database_name='db-test-ai')
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database_name	是	设置需删除的 Database 名称。	使用 list_databases() 查找需删除的数据库名。
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

```
{
    'code': 0,
    'msg': 'Operation success, requestId:
56f864f55a0b9c17acc840515d93e4f4',
    'affectedCount': 1
 }
```

参数名	参数含义
affectedCount	影响行数，即为删除数据库的数量。

查询 DataBase

最近更新时间：2024-09-27 18:00:11

接口定义

list_databases() 用于查询集群中所有的向量数据库，包括 Base 类与 AI 类数据库。

```
def list_databases(timeout: float | None = None) -> List[Database]
```

使用示例

```
db_list = client.list_databases()
for db in db_list :
    print("name={}, type={}".format(db.database_name,
db.__class__.__name__))
```

入参描述

参数名	是否必选	参数含义	配置方法
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

输出已创建的所有的数据库名，如下所示。

```
name=db-test, type=Database
name=db-test-ai, type=AIDatabase
```

参数名	参数含义
name	数据库名。

type	数据库类型。 • 带有 AI 字样说明数据库为 AI 套件类数据库，用于上传文件。 • 不带 AI 字样，则为 Base 类数据库。

版权所有：腾讯云计算（北京）有限责任公司

检查 Database 是否存在

最近更新时间：2024-10-14 18:01:21

接口定义

- **exists_collection()**用于检查指定的数据库是否已经存在。
- **create_collection_if_not_exists()**用于检查指定的数据库是否已经存在，若不存在直接创建 Database。

exists_db()

```
def exists_db(self, database_name: str) -> bool
```

create_database_if_not_exists()

```
def create_database_if_not_exists(self, database_name: str, timeout:
Optional[float] = None) -> Database
```

使用示例

exists_db()

```
# 创建 Database
# exists_db 使用示例
db_name = "db-test"
db_exists = client.exists_db(database_name=db_name)
print(f'Database {db_name} exists={db_exists}')
```

create_database_if_not_exists()

```
# create_database_if_not_exists 使用示例
db_name = "db-test"
db = client.create_database_if_not_exists(database_name=db_name)
print(f'Database {db_name} exists=
{client.exists_db(database_name=db_name)}')
```

参数名	是否必须	参数含义
database_name	是	指定检查的数据库名。

输出描述

```
Database db-test exists=True
```

参数名	返回值
db_exists	• True：数据库存在 • False：数据库不存在

Collection 操作
新建 Collection

最近更新时间：2025-07-09 15:17:22

接口定义

create_collection() 用于为已创建的 Base 类向量数据库创建 Collection。

```
def create_collection(self,
                      database_name: str,
                      collection_name: str,
                      shard: int,
                      replicas: int,
                      description: str = None,
                      index: Index = None,
                      embedding: Embedding = None,
                      timeout: float = None,
                      ttl_config: dict = None,
                      filter_index_config: FilterIndexConfig = None,
                      indexes: List[IndexField] = None,
                      ) -> RPCCollection:
```

使用示例

说明：

- **当前版本一个数据库实例下，不支持创建同名的 Collection。**
- **不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。**
- **插入数据时动态增加的标量字段暂不支持创建索引。创建 Collection 时，需规划好 Filter 索引的字段。**

向量读写

在 Base 类数据库 db-test 下，创建一个名为 book-vector 的集合，不配置 Embedding 模型相关参数，写入3维向量数据。

```
import tcvectordb
```

```
from tcvectordb.model.enum import FieldType, IndexType, MetricType
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams, SparseVector, SparseIndex

# -- index config
index = Index(
    FilterIndex(name='id', field_type=FieldType.String,
index_type=IndexType.PRIMARY_KEY),
    FilterIndex(name='author', field_type=FieldType.String,
index_type=IndexType.FILTER),
    FilterIndex(name='bookName', field_type=FieldType.String,
index_type=IndexType.FILTER),
    FilterIndex(name='tags', field_type=FieldType.Array,
index_type=IndexType.FILTER),
    VectorIndex(name='vector', field_type=FieldType.Float16Vector,
dimension=3, index_type=IndexType.HNSW,
    metric_type=MetricType.COSINE, params=HNSWParams(m=16,
efconstruction=200))
)
# create a collection
coll = client.create_collection(
    database_name='db-test',
    collection_name='book-vector',
    shard=1,
    replicas=1,
    description='this is a collection of test vector',
    index=index
)
print(vars(coll))
```

Embedding 文本读写

在 Base 类数据库 db-test 下，创建一个名为 book-emb 的集合，配置 Embedding 模型相关参数，写入原始文本。Embedding 模型自动将原始文本进行向量化。

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
EmbeddingModel
from tcvectordb.model.collection import Embedding
```

```
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams, SparseVector, SparseIndex

# -- index config
index = Index(
    FilterIndex(name='id', field_type=FieldType.String,
index_type=IndexType.PRIMARY_KEY),
    VectorIndex(name='vector', dimension=768,
index_type=IndexType.HNSW,
        metric_type=MetricType.COSINE, params=HNSWParams(m=16,
efconstruction=200)),
    # 稀疏向量索引
    # SparseIndex(name='sparse_vector',
    #             field_type=FieldType.SparseVector,
    #             index_type=IndexType.SPARSE_INVERTED,
    #             metric_type=MetricType.IP),
    FilterIndex(name='author', field_type=FieldType.String,
index_type=IndexType.FILTER),
    FilterIndex(name='tags', field_type=FieldType.Array,
index_type=IndexType.FILTER),
    FilterIndex(name='bookName', field_type=FieldType.String,
index_type=IndexType.FILTER))

# Embedding config
ebd = Embedding(vector_field='vector', field='text',
model_name='bge-base-zh')

# create a collection
coll = client.create_collection(
    database_name='db-test',
    collection_name='book-emb',
    shard=1,
    replicas=1,
    description='this is a collection of test embedding',
    embedding=ebd,
    index=index
)
print(vars(coll))
```

仅稀疏向量集合

在 Base 类数据库 db-test 下，创建一个名为 book-vector 的集合，不配置 Embedding 模型相关参数，写入3维向量数据，支持稀疏向量写入。

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams, SparseVector, SparseIndex

# -- index config
index = Index(
    FilterIndex(name='id', field_type=FieldType.String,
index_type=IndexType.PRIMARY_KEY),
    SparseIndex(name='sparse_vector',
        field_type=FieldType.SparseVector,
        index_type=IndexType.SPARSE_INVERTED,
        metric_type=MetricType.IP))
# create a collection
coll = client.create_collection(
    database_name='db-test',
    collection_name='book-vector',
    shard=1,
    replicas=1,
    description='this is a collection of test vector',
    index=index)
print(vars(coll))
```

二进制集合

在 Base 类数据库 db-test 下，创建一个名为 book-vector 的集合，不配置 Embedding 模型相关参数，写入3维向量数据。

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType
from tcvectordb.model.index import Index, VectorIndex, FilterIndex
from tcvectordb.toolkit.binary import binary_to_uint8
# -- index config
index = Index(
    FilterIndex(name='id', field_type=FieldType.String,
index_type=IndexType.PRIMARY_KEY),
```

```
    FilterIndex(name='bookName', field_type=FieldType.String,
index_type=IndexType.FILTER),
    VectorIndex(name='vector', dimension=16,
index_type=IndexType.BIN_FLAT,
                metric_type=MetricType.HAMMING,
field_type=FieldType.BinaryVector)
)
# create a collection
coll = client.create_collection(
    database_name='db-test',
    collection_name='bin-vector',
    shard=1,
    replicas=1,
    description='this is a collection of test bin vector',
    index=index
)
print(vars(coll))
```

支持 Json 与 AutoID

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams
coll = client.create_collection_if_not_exists(
    database_name="db-test",
    collection_name="coll-json-autoid",
    shard=1,
    replicas=0,
    description='test collection json and autoid',
    indexes=[
        FilterIndex('id', field_type=FieldType.String,
index_type=IndexType.PRIMARY_KEY, auto_id='uuid'),
        VectorIndex('vector', 1, IndexType.HNSW, MetricType.IP,
HNSWParams(m=16, efconstruction=200)),
        FilterIndex(name='bookInfo', field_type=FieldType.Json,
index_type=IndexType.FILTER),
    ],
)
```

```
# print collection
print(vars(coll))
```

写入文件集合

> **注意：**
> 在创建存储文件内容的 Collection 时，索引字段的设计需遵循以下限制与建议，否则可能导致上传文件失败。
> - 文件名字段（file_name）：该字段**必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。字段名可以自定义，在上传文件时，需通过参数 field_mappings 的 filename 映射自定义的字段名。
> - 文件内容字段（text）：该字段用于存储知识点的原始文本内容。由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。字段名可以自定义，但需通过参数 field_mappings 的 text 映射自定义的字段名。
> - 文件图片信息字段（image_list）：该字段存放 PDF 文件中图片的 Key 列表。在原始检索出的文本块中，图片位置将以 {key} 的形式进行占位。通过调用接口 getImageUrl，可以获取图片 URL 地址列表，这些地址与 Key 一一对应，从而实现将原始 PDF 文档恢复并展示为 HTML 格式。
>   - **不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。
>   - 字段名可以自定义，但需要在上传文件时通过参数 field_mappings 的 imageList 映射到自定义的字段名。

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams
coll = client.create_collection_if_not_exists(
    database_name="db-test",
    collection_name="coll-file-test",
    shard=1,
    replicas=0,
    description='test collection file upload',
    indexes=[
```

```
        FilterIndex('id', field_type=FieldType.String,
index_type=IndexType.PRIMARY_KEY),
        VectorIndex('vector', 768, IndexType.HNSW, MetricType.IP,
HNSWParams(m=16, efconstruction=200)),
        FilterIndex(name='file_name', field_type=FieldType.String,
index_type=IndexType.FILTER)
    ],
)

# print collection
print(vars(coll))
```

入参描述

创建集合之前，需要创建主键索引、向量索引、标量索引、稀疏向量索引。

Index 参数

参数名	子参数	是否必选	参数配置
FilterIndex	name	否	配置可作为 Filter 索引的自定义扩展的标量字段名。 **说明：** • Filter 索引（Filter Index）对**标量字段**建立的索引。标量字段名称、类型均由用户自定义，不限制标量字段数量. • 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询以此来匹配相似向量。 • 建立 Filter 索引时，选取需要使用 Filter 表达式高效过滤数据的标量字段。不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。 • 必须构建唯一的 Document id 为主键的 Filter 索引，配置 name 固定为 id 。 • 配置其他自定义扩展的可作为 Filter 索引的标量字段，例如：author、page 等。该标量字段名称、类型均由用户自定义，且不限制字段数量。

	field_type	否	指定 Filter 字段的数据类型。取值如下：
			• **String**：字符型。若 **name** 为 **id**，则该参数固定为 **FieldType.String**。
			• **Uint64**：无符号整数，该参数可设置为 **FieldType.Uint64**。
			• **Array**：数组类型，该参数可设置为 **FieldType.Array**，数组元素默认为 **String**。
			• **Json**：由键值对组成的数据对象。具体规则，请参见 Json 类型 。为 json 类型的字段创建 Filter 索引，如下所示。
			⚠ 注意：
			开启 动态标量字段（动态 Schema ）时，即使用 filterIndexConfig 开启所有标量字段全索引模式，现已支持配置 json 类型的 Filter 索引。
			`FilterIndex(name='bookInfo',`
			`field_type=FieldType.Json,`
			`index_type=IndexType.FILTER)`

	index_type	否	指定 Filter 字段的索引类型。 • 若 **name** 为 **id**，该参数固定配置为 **IndexType.PRIMARY_KEY**，即以 id 为主键构建索引。 • 若 **name** 为其他自定义的标量字段，预在检索时，依据该字段过滤数据，可配置该字段索引类型为 **IndexType.FILTER**。
	auto_id	否	自动生成 ID。指定 "autoId": "uuid"，即可开启自动生成 ID 功能。 **ⓘ 说明：** • 当启用 autoId 功能后，用户在写入或更新数据时，无需手动传入 id 字段。若用户在启用 autoId 的情况下仍传入了 id 字段，则系统将优先采用用户传入的 id 值作为数据的唯一标识。 • 在执行批量写入（batch upsert）时，如果部分数据包含 id 字段，而另一部分数据未包含 id 字段，则系统仅对未传入 id 的数据生成默认 id。

Vector Index	name	是	指定向量索引字段名，固定为 vector。
	field_type	是	向量字段的数据类型。支持不同类型、不同精度的向量数据，以适用不同业务场景。更多信息，请参见 数据类型 。 • Vector：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位），适用于需最高精度的计算任务或对误差敏感的向量检索。使用示例：`field_type=FieldType.Vector` 。 • Float16Vector：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位 的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 BGE-large 等模型生成的嵌入向量。使用示例：field_type=FieldType.Float16Vector。 • BFloat16Vector：bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，适用于向量数值范围较大（更多bit位表达整数）的场景。 • BinaryVector：每个维度仅用 1 个比特（bit）表示0或1，无指数位、尾数位和符号位，直接存储二进制数据，适用于存储稀疏向量。
	index_type	是	指定索引类型，取值如下所示。更多信息，请参见 索引类型 。 • FLAT：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。 • HNSW：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区间数据规模。 • BIN_FLAT：二进制索引，暴力检索，召回率100%。 • IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。
	dimension	否	指定向量维度。 • 取值类型：uint64。 • 取值范围：[1,4096]。 **说明：** • 若 indexType 为 BIN_FLAT，向量维度大于等于8，且必须为8的倍数。 • 开通 Embedding 功能，则无需配置该字段，该字段将自动配置为 Embedding 模型对应的向量维度。

	metric_type	是	指定向量之间距离度量的算法。取值如下： • L2：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • IP：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 • COSINE：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度。所得值越大越与搜索值相似。 • Hamming：汉明距离（Hamming Distance），计算两个二进制字符串对应位置上不同字符的数量，如果字符不同，两字符串的汉明距离就会加一。汉明距离越小，表示两个字符串之间的相似度越高。
	params	否	• 指定索引类型 indexType 为 HNSW，需配置如下参数。 ○ m：每个节点在检索构图中可以连接多少个邻居节点。 ○ 取值类型：uint64。 ○ 取值范围：[4,64]。默认为16。 ○ efconstruction：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。 ○ 取值类型：uint64。 ○ 取值范围：[8,512]。默认为200。 • 索引类型 indexType 为 IVF_FLAT、IVF_PQ、IVF_SQ4、IVF_SQ8、IVF_SQ16，需配置如下参数。 ○ nlist：指索引中的聚类中心数量。 ○ 取值类型：uint64。 ○ 取值范围：[1,65536]。 ○ m：指乘积量化中原始数据被拆分的子向量的数量，仅 IVF_PQ 索引类型需配置。 ○ 取值要求：原始数据的向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。 ○ 取值范围：[1,向量维度]。
SparseIndex	name	是	可选择构建稀疏向量索引，字段名固定为 sparse_vector。每个集合仅支持创建1个稀疏向量索引，无需指定稀疏向量维度。
	field_type		稀疏向量字段类型固定为 FieldType.SparseVector。

	index_type		稀疏向量索引类型固定为 IndexType.SPARSE_INVERTED。
	metric_type		稀疏向量相似性计算仅支持设置为 IP。

Embedding 参数

集合开启 Embedding，需配置 Embedding 模型相关参数，写入原始文本。

子参数	是否必选	参数配置
field	否	指定文本字段名称。取值类型：String。 **说明：** 通过 upsert() 写入数据或通过 update() 更新数据时，Embedding 模型会自动将该字段的文本内容转换成向量数据。
vector_field	否	指定向量字段。通过 Embedding 模型生成的向量会自动存储在该字段中。固定为 `vector` 。
model_name	否	指定使用的 Embedding 模型的名称。您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，参见 Embedding 介绍 。取值如下所示： • bge-large-zh-v1.5：适用中文，1024维，推荐使用。 • bge-base-zh-v1.5：适用中文，768维。 • bge-large-zh：适用中文，1024维。 • bge-base-zh：适用中文，768维。 • m3e-base：适用中文，768维。 • e5-large-v2：适用英文，1024维。 • text2vec-large-chinese：适用中文，1024维。 • multilingual-e5-base：适用于多种语言类型，768维。 • BAAI/bge-m3：适用于多种语言类型，1024维。

Collection 参数

参数	参数含义	是否必选	参数配置

database_name	指定数据库名	是	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	指定 Collection 的名称。	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
replicas	指定 Collection 的副本数。副本数是指每个主分片有多个相同的备份，用来容灾和负载均衡。	是	• 取值类型：uint64。 • 取值范围如下所示。搜索请求量越高的索引，建议设置越多的副本数，避免负载不均衡。 ○ 单可用区实例：0。 ○ 两可用区实例：[1,节点数-1]。 ○ 三可用区实例：[2,节点数-1]。
shard	指定 Collection 的分片数。分片是把大数据集切成多个子数据集。	是	• 取值类型：uint64。 • 取值范围：[1,100]。例如：5。 • 配置建议：在搜索时，全部分片是并发执行的，分片数量越多，平均耗时越低，但是过多的分片会带来额外开销而影响性能。 ○ 单分片数据量建议控制在300万以内，例如500万向量，可设置2个分片。 ○ 如果数据量小于300万，建议使用1分片。系统对1分片有特定优化，可显著提升性能。
description	指定 Collection 的描述信息。	否	• 取值类型：string。 • 字符长度要求：[1,256]。 • 示例：this is the collection description。
index	指定 Collection 存储文档的索引属性。	是	将已创建的 Index 参数赋值给 index。
embedding	Embedding 相关参数。	否	将已创建的 Embedding 参数赋值给 **embedding** 。

ttl_config	标识数据库是否开启 TTL 配置，指定存储数据过期时间戳字段名。	否	**enable**：标识数据库是否开启 TTL 属性。 • True：开启。 • False：关闭。 **说明：** TTL（生存周期）功能规定了数据自创建或更新后将被自动删除的时间期限。向量数据库默认每小时轮询检查一次数据是否过期，超期即清理，且插入数据时允许存在最多1小时的过期时间误差。具体信息，请参见 TTL 。 **timeField**：指定存储数据过期时间戳的字段名。插入数据时，以标准的 Unix 时间戳指定该字段的值。字段名要求如下： • 数据类型：uint64。 • 索引类型：Filter 索引。 • 示例：expired_at。 `ttl_config={ 'enable': True, 'timeField': 'expire_at' }`
filter_index_config	支持开启所有标量字段全索引模式，默认关闭	否	**filter_all**：控制标量字段全索引模式的开启和关闭。 • true：开启。 • false：关闭。默认值为 fasle。 **fields_without_index**：当 filterAll 为 true 时，支持通过 fieldWithoutFilterIndex 参数指定不创建索引的字段。 • 若 filterAll 为 false 时不支持配置。 • 默认值：NULL。 **max_str_len**：当 filterAll 为 true 时，支持通过 maxFieldLength 参数设定单条文档中创建索引的标量字段的最大字符数。如果字段字符数超出此限制，将按照设定的 maxStrLen 值进行截断创建索引。 • 默认值：32。 • 取值范围：[1,65536]。 `filter_index_config=FilterIndexConfig( filter_all=True,`

			<pre>fields_without_index=['author'],
			max_str_len=32,
			)</pre>

timeout	请求超时时间。	否	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

删除 Collection

最近更新时间：2025-07-01 10:19:42

接口定义

```
def drop_collection(self,
                    database_name: str,
                    collection_name: str,
                    timeout: Optional[float] = None) -> Dict:
```

使用示例

警告：
执行 drop 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

说明：
如果您为集合设置了别名，删除集合操作仅支持使用集合的原始名称，无法通过别名来删除对应的集合。

```
res = client.drop_collection(database_name='db-test',
collection_name='book-emb')
print(res, flush=True)
```

入参描述

参数名	是否必选	参数含义	配置方法
database_name	是	指定数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定 Collection 的名称。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
timeout	否	请求超时时间。	• 单位：秒。

			• 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

```
{
    'code': 0,
    'msg': 'operation success',
    'affectedCount': 1
 }
```

参数名	参数含义
affectedCount	影响行数，即为删除集合数量。

查询 Collection 列表

最近更新时间：2025-07-01 10:19:42

接口定义

list_collections() 接口用于查询指定 Database 中所有的 Collection。

```
def list_collections(self, database_name: str, timeout: Optional[float]
= None) -> List[RPCCollection]:
```

使用示例

```
coll_list = client.list_collections(database_name='db-test')
for col in coll_list:
        print(vars(col))
```

入参描述

参数名	是否必选	参数含义	配置方法
database_name	是	指定数据库名	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定 Collection 的名称。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

```
{
  'database': 'db-test',
```

```
  'collection': 'book-emb',
  'replicaNum': 0,
  'shardNum': 1,
  'indexes': [
    {
      'fieldName': 'vector',
      'fieldType': 'vector',
      'indexType': 'HNSW',
      'metricType': 'COSINE',
      'dimension': 768,
      'params': {
        'M': 16,
        'efConstruction': 200
      },
      'indexedCount': 0
    },
    {
      'fieldName': 'bookName',
      'fieldType': 'string',
      'indexType': 'filter'
    },
    {
      'fieldName': 'tags',
      'fieldType': 'string',
      'indexType': 'filter'
    },
    {
      'fieldName': 'id',
      'fieldType': 'string',
      'indexType': 'primaryKey'
    },
    {
      'fieldName': 'author',
      'fieldType': 'string',
      'indexType': 'filter'
    },
    {
      'fieldName': 'sparse_vector',
      'fieldType': 'sparseVector',
      'indexType': 'inverted',
      'metricType': 'IP'
    }
  ],
```

```
  'embedding': {
    'status': 'enabled',
    'field': 'text',
    'model': 'bge-base-zh',
    'vectorField': 'vector'
  },
  'description': 'this is a collection of test embedding',
  'createTime': '2024-09-09 17:28:10',
  'documentCount': 0,
  'alias': [
    'alias-book-emb'
  ],
  'indexStatus': {
    'status': 'ready',
    'startTime': ''
  }
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 Collection 所在的 Database 名称。
collection	-	-	显示 Collection 的名称。
replicaNum	-	-	显示 Collection 的副本数。
shardNum	-	-	显示 Collection 的分片数。
createTime	-	-	显示 Collection 的创建时间。
description	-	-	显示 Collection 的描述信息。
documentCount	-	-	返回 Collection 中存储的 Document 数量。
indexes	主键索引	fieldName	标识索引对象为 id。

		filedType	显示该索引对象的数据类型，固定为 string。
		indexType	该参数固定显示为 **primaryKey**。即该索引对象以 id 为主键构建索引。
	向量索引	fieldName	标识索引对象为 vector 字段名，固定为 **vector**。
		filedType	指定索引对象为 vector 字段的数据类型，固定为 **vector**。
		indexType	显示索引类型。具体信息，请参见 索引类型 。
		indexedCount	索引对象为 vector 字段的文档数量。
		dimension	显示索引对象为 vector 的向量维度。
		metricType	显示索引象为 vector 的向量之间的距离度量的算法。
		params	显示索引类型对应的参数。
	Filter 索引	fieldName	自定义扩展的可设置 Filter 表达式的字段名。例如：page、author。
		filedType	显示字段的数据类型。
		indexType	标识自定义字段是否可以设置 Filter 表达式，固定为 **filter**。
	稀疏向量索引	fieldName	稀疏向量字段名，固定为：sparse_vector。
		fieldType	稀疏向量数据类型，固定为：`sparseVector` 。
		indexType	索引类型，固定为：`inverted` 。
		metricType	稀疏向量间的距离度量的算法。

embedding	Embedding 相关参数	status	说明该 Collection 是否配置 Embedding 模型。 • **enabled**：已配置。 • **disabled**：未配置。
		field	显示 Embedding 模型输入文本的字段名。
		model	Embedding 模型的名称。
		vectorField	显示 Embedding 模型向量字段名。
alias	集合别名	-	集合别名。
indexStatus	集合重建索引相关参数	status	标识当前 Collection 是否在重建索引。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

查询指定 Collection

最近更新时间：2025-07-01 10:19:42

接口定义

describe_collection() 接口用于查询指定 Collection 的信息。

```
def describe_collection(
    database_name: str,
    collection_name: str,
    timeout: float | None = None
) -> RPCCollection
```

使用示例

```
# 查询 Base 类数据库下的集合
res = client.describe_collection(database_name='db-test',
collection_name='book-emb')
print(vars(res))
```

入参描述

参数名	是否必选	参数含义	配置方法
database_name	是	指定数据库名	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定 Collection 的名称。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
timeout	否	请求超时时长。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

版权所有：腾讯云计算（北京）有限责任公司

出参描述


```
{
    "database": "db-test",
    "collection": "book-emb",
    "replicaNum": 2,
    "shardNum": 1,
    "description": "this is a collection of test embedding",
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "dimension": 768,
            "metricType": "COSINE",
            "params": {
                "M": 16,
                "efConstruction": 200
            },
            "indexedCount": 3
        },
        {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "tags",
            "fieldType": "string",
            "indexType": "filter"
        }
    ],
    "embedding": {
```

```
            "status": "enabled",
            "field": "text",
            "model": "bge-base-zh",
            "vectorField": "vector"
        },
        "documentCount": 3,
        "alias": [
            "alias-book-emb"
        ],
        "indexStatus": {
            "status": "ready",
            "startTime": ""
        }
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 Collection 所在的 Database 名称。
collection	-	-	显示 Collection 的名称。
replicaNum	-	-	显示 Collection 的副本数。
shardNum	-	-	显示 Collection 的分片数。
createTime	-	-	显示 Collection 的创建时间。
description	-	-	显示 Collection 的描述信息。
documentCount	-	-	返回 Collection 中存储的 Document 数量。
indexes	主键索引	fieldName	标识索引对象为 id。
		filedType	显示该索引对象的数据类型，固定为 string。

		indexType	该参数固定显示为 **primaryKey**。即该索引对象以 id 为主键构建索引。
	向量索引	fieldName	标识索引对象为 vector 字段名，固定为 **vector**。
		filedType	指定索引对象为 vector 字段的数据类型，固定为 **vector**。
		indexType	显示索引类型。具体信息，请参见 索引类型 。
		indexedCount	索引对象为 vector 字段的文档数量。
		dimension	显示索引对象为 vector 的向量维度。
		metricType	显示索引象为 vector 的向量之间的距离度量的算法。
		params	显示索引类型对应的参数。
	Filter 索引	fieldName	自定义扩展的可设置 Filter 表达式的字段名。例如：page、author。
		filedType	显示字段的数据类型。
		indexType	标识自定义字段是否可以设置 Filter 表达式，固定为 **filter**。
embedding	Embedding 相关参数	status	说明该 Collection 是否配置 Embedding 模型。 • **enabled**：已配置。 • **disabled**：未配置。
		field	显示 Embedding 模型输入文本的字段名。
		model	Embedding 模型的名称。
		vectorField	显示 Embedding 模型向量字段名。
alias	集合别名	-	集合别名。

indexStatus	集合重建索引相关参数	status	标识当前 Collection 是否在重建索引。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

检查 Colleciton 是否存在

最近更新时间：2025-07-01 10:19:42

接口定义

- exists_collection() 用于检查指定的数据库及其对应的集合是否已经存在。
- create_collection_if_not_exists() 用于检查指定的数据库及其对应的集合是否已经存在，若不存在直接创建集合。

exists_collection()

```
def exists_collection(self,
                      database_name: str,
                      collection_name: str) -> bool
```

create_collection_if_not_exists()

```
def create_collection_if_not_exists(self,
                                    database_name: str,
                                    collection_name: str,
                                    shard: int,
                                    replicas: int,
                                    description: str = None,
                                    index: Index = None,
                                    embedding: Embedding = None,
                                    timeout: float = None,
                                    ttl_config: dict = None,
                                    filter_index_config:
FilterIndexConfig = None,
                                    indexes: List[IndexField] =
None,
                                    ) -> RPCCollection:
```

使用示例

检查 Collection 是否存在

```
db_name = "db-test"
coll_name = "book-vector"
coll_exists = client.exists_collection(database_name=db_name,
collection_name=coll_name)
print(f'Collection {coll_name} exists={coll_exists}')
```

检查 Collection 是否存在，若不存在创建集合

```
coll = client.create_collection_if_not_exists(
    database_name="db-test"e,
    collection_name="book-vector",
    shard=1,
    replicas=0,
    description='test collection',
    index=index,
)
print(f'Collection {coll_name} exists=
{client.exists_collection(database_name=db_name,
collection_name=coll_name)}')
```

说明：
create_collection_if_not_exists()若集合不存在，根据集合相关参数新建集合。具体参数，请参见 新建 Collection 。

参数名	是否必须	参数含义
database_name	是	指定检查的数据库名。
collection_name	是	指定查询的集合。

输出描述

```
Collection book-vector exists=True
```

参数名	返回值
coll_exists	• True：集合存在 • False：集合不存在

版权所有：腾讯云计算（北京）有限责任公司

清空 Collection

最近更新时间：2025-07-01 10:19:42

接口定义

truncate_collection() 用于清空 Collection 中所有的数据与索引，仅保留 Collection 配置信息，例如索引类型及参数、分片等设置，减少用户的操作成本。

```
def truncate_collection(self,
                        database_name: str,
                        collection_name: str) -> Dict:
```

使用示例

> **警告：**
> 执行 truncate 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

```
res = client.truncate_collection(database_name='db-test',
collection_name='book-emb')
print(res, flush=True)
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database_name	是	指定数据库名	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定 Collection 的名称。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

```
{
```

```
    'code': 0,
    'msg': 'operation success',
    'affectedCount': 1
}
```

参数名	参数含义
affectedCount	影响行数，即为清空的集合数量。

CollectionView 操作
新建 CollectionView

最近更新时间：2024-12-05 20:37:02

接口定义

create_collection_view() 用于为已创建的 AI 类向量数据库创建 CollectionView。

```
def create_collection_view(
    name: str,
    description: str = '',
    embedding: Embedding | None = None,
    splitter_process: SplitterProcess | None = None,
    index: Index | None = None,
    timeout: float | None = None,
    expected_file_num: int | None = None,
    average_file_size: int | None = None,
    shard: int | None = None,
    replicas: int | None = None
) -> CollectionView
```

请求示例

在 AI 类数据库 db-test-ai 下，创建一个名为 coll-ai-files 的集合视图，用于上传并存储文件。

```
import tcvectordb
from tcvectordb.model.ai_database import AIDatabase
from tcvectordb.model.enum import FieldType, IndexType, ReadConsistency
from tcvectordb.model.index import Index, FilterIndex
from tcvectordb.model.collection_view import Embedding, SplitterProcess,
Language, CollectionView, ParsingProcess

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

db = client.database('db-test-ai')

# create a collection
```

```
# 配置文件 Meta 信息，构建为 Filter 索引
index = Index()
index.add(FilterIndex('author', FieldType.String, IndexType.FILTER))
index.add(FilterIndex('tags', FieldType.Array, IndexType.FILTER))
# 创建集合，文件预处理相关参数
# 1. Embedding：语言类型、是否开启词（Words）向量精排
# 2、splitter_process：拆分时，是否需要追加 Title 或 keywords
# 3、Index：为文件 meta 信息的标量字段设置 Filter 索引
coll_view = db.create_collection_view(name='coll-ai-files',
                              description='this is a collection
description',
                              index=index,
                              embedding=Embedding(
                                  language=Language.ZH.value,
                                  enable_words_embedding=True,
                                ),
                              splitter_process=SplitterProcess(
                                  append_title_to_chunk=True,
                                  append_keywords_to_chunk=True,
                                  #  chunk_splitter 参数以正则表达式的方式配置
文档拆分方式，如下：\n{2,} 代表以两个及以上的换行进行拆分，常用在QA对文件拆分中。
                                  # chunk_splitter="\n{2,}"
                              ),
                              parsing_process=ParsingProcess(
                                  parsing_type='AlgorithmParsing',
                              ),
                              timeout=100)
print(vars(coll_view))
```

入参描述

参数	参数含义	子参数	是否必选	配置方法
name	指定 CollectionView 的名称。	-	是	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
description	指定 CollectionVie	-	否	• 取值类型：string。 • 字符长度要求：[1,256]。

	w 的描述信息			• 示例：this is the collection description。
embedding	Embedding 相关配置	language	否	取值如下所示： • ZH：中文 • EN：英文 • MULTI：多语言
		enable_words_embedding	否	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • True：开启。 • False：不开启，默认为 False。
splitter_process	文件预处理方式配置	append_title_to_chunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • True：将段落 Title 追加到切分后的段落。 • False：不追加。默认值为 False。
		append_keywords_to_chunk	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • True：将全文的 keywords 追加到切分后的段落。默认值为 True。 • False：不追加。
		chunk_splitter	否	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsing_process	PDF 文件解析方式	parsing_type	否	指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

index	配置需使用 Filter 索引的标量字段，以便使用该字段的 Filter 条件表达式过滤查找文件。	fieldName	是	配置可作为 Filter 索引的自定义扩展的标量字段名。 **说明：** • Filter 索引（Filter Index）是建立在标量字段的索引。该标量字段名称、类型均由用户自定义，不限制标量字段数量。 • 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询以此来匹配相似向量。 • 建立 Filter 索引时，选取文件 Metadata 信息需要使用 Filter 表达式高效过滤数据的标量字段。不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。
		FieldType	是	指定自定义字段的数据类型。取值如下： • **String**：字符型。 • **Uint64**：指无符号整数（unsigned integer）。 • **Array**：数组类型，默认数组元素为 string。
		IndexType	否	该参数固定设置为 `FILTER`。

出参描述

```
{
  "database": "db-test-ai",
  "collectionView": "coll-ai-files-sdk",
```

```
  "description": "this is a collection description",
  "embedding": {
    "language": "zh",
    "enableWordsEmbedding": true
  },
  "splitterPreprocess": {
    "appendTitleToChunk": true,
    "appendKeywordsToChunk": true
  },
  "indexes": [
    {
      "fieldName": "tags",
      "fieldType": "array",
      "indexType": "filter"
    },
    {
      "fieldName": "author",
      "fieldType": "string",
      "indexType": "filter"
    }
  ]
}
```

版权所有：腾讯云计算（北京）有限责任公司

删除 CollectionView

最近更新时间：2024-09-27 18:00:11

接口定义

drop_collection_view() 用于删除已创建的 CollectionView。

```
def drop_collection_view(
    collection_view_name: str,
    timeout: float | None = None
) -> Dict[str, Any]
```

使用实例

> **警告：**
> 执行 drop 操作将会永久删除指定 CollectionView 下的所有 DocumentSet。在操作之前，务必谨慎考虑。

```
# 指定 AI 类数据库
db = client.database('db-test-ai')
# 删除 AI 类数据库下的指定集合视图
db.drop_collection_view(collection_view_name ='coll-ai-files')
```

入参描述

参数名	是否必选	参数含义	配置方法
collection_view_name	是	所需删除的 CollectionView 名称。	• 请使用 list_collection_view() 查找需删除的集合视图。
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

```
{
    'code': 0,
    'msg': 'Operation success, requestId:
88eeb1748ebc9b1715bbef**********',
    'affectedCount': 1
 }
```

参数名	参数含义
affectedCount	影响行数，即为删除的集合视图数量。

查询 CollectionView 列表

最近更新时间：2024-12-05 20:37:02

功能介绍

list_collection_view() 接口用于查询指定 AI 类 Database 中所有的 CollectionView。

```
def list_collection_view(timeout: float | None = None) ->
List[CollectionView]
```

使用示例

```
# 指定 AI 类数据库
db = client.database('db-test-ai')
# 查询 AI 类据库下的集合列表
coll_list = db.list_collection_view()
for col in coll_list:
        print(vars(col))
```

入参描述

参数名	是否必选	参数含义	配置方法
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

```
{
  "database": "db-test-ai",
  "collectionView": "coll-ai-files",
  "description": "this is a collection description",
  "embedding": {
    "language": "zh",
    "enableWordsEmbedding": false
  },
  "splitterPreprocess": {
```

```
      "appendTitleToChunk": true,
      "appendKeywordsToChunk": true
    },
    "parsingProcess": {
      "parsingType": "AlgorithmParsing"
    },
    "indexes": [
      {
        "fieldName": "author",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "documentSetId",
        "fieldType": "string",
        "indexType": "primaryKey"
      },
      {
        "fieldName": "documentSetName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "tags",
        "fieldType": "array",
        "indexType": "filter"
      }
    ],
    "createTime": "2023-11-27 17:16:54",
    "stats": {
      "indexedDocumentSets": 0,
      "totalDocumentSets": 0,
      "unIndexedDocumentSets": 0
    },
    "alias": [
      "alias-coll-ai-files"
    ]
}
```


参数	子参数	子参数	参数含义
database	-	-	显示 CollectionView 所在的 AI 类 Database 名称。

collectionView	–	–	显示 CollectionView 的名称。
embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • mutil：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
alias	–	–	CollectionView 的所有别名。创建别名，请参见 set_alias() 。
createTime	–	–	显示 CollectionView 的创建时间。
description	–	–	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。
		unIndexedDocumentSets	未处理的文件数量。

splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • True：将段落 Title 追加到切分后的段落。 • False：不追加。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
		chunk_splitter	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsingProcess	指定 PDF 类型文件的解析方式	parsingType	取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 documentSetId 文件 ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 documentSetName 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式检索文件。

版权所有：腾讯云计算（北京）有限责任公司

	其他自定义需建立 Filter 索引的标量字段	fieldName	自定义扩展字段，例如：author、tags。
		filedType	显示自定义字段的数据类型。
		indexType	显示自定义字段索引类别为 `filter` 。

查询指定 CollectionView

最近更新时间：2024-12-05 20:37:02

接口定义

describe_collection_view() 接口用于查询指定 CollectionView 的信息。

```
def describe_collection_view(
    collection_view_name: str,
    timeout: float | None = None
) -> CollectionView
```

使用示例

```
# 指定 AI 类数据库
db = client.database('db-test-ai')
# 查询 Ai 类数据库下的集合
res = db.describe_collection_view(collection_view_name ='coll-ai-files')
print(vars(res))
```

入参描述

参数名	是否必选	参数含义	配置方法
collection_view_name	是	指定所需查询的 Collection 名称。	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
timeout	否	请求超时时长。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

返回消息

```
{
  "database": "db-test-ai",
  "collectionView": "coll-ai-files",
```

```
    "description": "this is a collection description",
    "embedding": {
      "language": "zh",
      "enableWordsEmbedding": false
    },
    "splitterPreprocess": {
      "appendTitleToChunk": true,
      "appendKeywordsToChunk": true
    },
    "parsingProcess": {
        "parsingType": "AlgorithmParsing"
    },
    "indexes": [
      {
        "fieldName": "author",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "documentSetId",
        "fieldType": "string",
        "indexType": "primaryKey"
      },
      {
        "fieldName": "documentSetName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "tags",
        "fieldType": "array",
        "indexType": "filter"
      }
    ],
    "createTime": "2023-11-27 17:16:54",
    "stats": {
      "indexedDocumentSets": 0,
      "totalDocumentSets": 0,
      "unIndexedDocumentSets": 0
    },
    "alias": [
      "alias-coll-ai-files"
    ]
```

```
}
```

参数	参数含义	子参数	参数含义
database	-	-	显示 CollectionView 所在的 AI 类 Database 名称。
collectionView	-	-	显示 CollectionView 的名称。
embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • mutil：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
alias	-	-	CollectionView 的所有别名。
createTime	-	-	显示 CollectionView 的创建时间。
description	-	-	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。
		unIndexedDocument	未处理的文件数量。

		Sets	
splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • false：不追加。 • true：将段落 Title 追加到切分后的段落。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
		chunk_splitter	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsingProcess	指定 PDF 类型文件的解析方式	parsingType	取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 documentSetId 文件ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 documentSetName 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式检索文件。

	其他自定义需建立 Filter 索引的标量字段	fieldName	自定义扩展字段，例如：author、tags。
		filedType	显示自定义字段的数据类型。
		indexType	显示自定义字段索引类别为 `filter` 。

清空 CollectionView 数据

最近更新时间：2024-09-27 18:00:11

接口定义

truncate_collection_view() 用于清空 CollectionView 中所有的数据与索引，仅保留 CollectionView 配置信息，例如索引类型及参数等设置，减少用户的操作成本。

```
def truncate_collection_view(
    collection_view_name: str,
    timeout: float | None = None
) -> Dict[str, Any]
```

使用示例

> **警告：**
> 执行 truncate 操作将会永久删除指定 CollectionView 下的所有数据。在操作之前，务必谨慎考虑。

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# Specify the database name
db = client.database('db_test')

db.truncate_collection_view(collection_view_name ='coll-ai-files')
```

入参描述

参数	是否必选	参数含义	配置方法及要求
collection_view_name	是	指定需清空数据的 CollectionView 名。	使用 list_collection_view() 获取指定数据库名下的 CollectionView 列表，复制需清空数据的集合视图。

出参描述

```
{
    'code': 0,
    'msg': 'Operation success, requestId:
3610f94a80bc9b1714bbef**********',
    'affectedCount': 1
}
```

参数名	参数含义
affectedCount	影响行数，即为清空的集合数量。

Alias 操作
设置别名

最近更新时间：2024-09-27 18:00:11

功能介绍

set_alias() 接口用于为 Collection 或 CollectionView 设置别名。别名可以是一个简短的字符串，方便标识和访问对应的集合。通常，别名会替换 Collection 或 CollectionView 的名称用于业务切换等场景。

为 Collection 设置别名

```
def set_alias(
    collection_name: str,
    collection_alias: str
) -> Dict[str, Any]
```

为 CollectionView 创建别名

```
def set_alias(
    collection_view_name: str,
    alias: str
) -> Dict[str, Any]
```

使用实例

说明：

- **通过集合的别名做业务迁移时，仅需通过 set_alias() 接口将同一别名指向新的集合，别名与集合的映射关系将自动更新为新集合，可直接通过别名访问新集合。**
- **DB 和 Collection 级别（包含 AI 类数据库的 CollectionView）的 drop 操作会同时删除库表下的所有别名。**
- **Document 与 DocumentSet 层级的访问优先访问别名，其余级别仅支持原 Collection 或 CollectionView 名操作。**

版权所有：腾讯云计算（北京）有限责任公司

- 集合或集合视图的别名可以和名称重复，一个集合或集合视图的多个别名之间不能重复。

为 Collection 设置别名

```
# Specify the database name
db = client.database('db-test')
# set alias
res = db.set_alias(collection_name='book-emb',
collection_alias='alias-book-emb')
print(res)
```

参数	是否必选	参数含义	配置方法及要求
collection_name	是	指定需创建别名的 Collection 名。	使用 list_collections() 获取指定数据库名下的 Collection 列表，复制需创建别名的集合名。
collection_alias	否	设置 Collection 别名。	Collection 别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

为 CollectionView 设置别名

```
# Specify the database name
db = client.database('db-test-ai')
# set alias
res = db.set_alias(collection_view_name='coll-ai-files',
alias='alias-sdk-test')
print(res)
```

参数	是否	参数含义	配置方法及要求

	必选		
collection_view_name	是	指定需创建别名的 CollectionView 名。	使用 list_collection_view() 获取指定数据库名下的 Collection 列表，复制需创建别名的集合名。
alias	否	设置 CollectionView 别名。	CollectionView 别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

```
{'affectedCount': 1}
```

参数名	参数含义
affectedCount	影响行数，即为创建别名的集合数量。

删除别名

最近更新时间：2024-09-27 18:00:11

功能介绍

delete_alias() 接口用于删除数据库 Collection 或 CollectionView 的别名。

```
def delete_alias(alias: str) -> Dict[str, Any]
```

使用实例

```
db = client.database('db-test')
res = db.delete_alias(alias='alias-book-emb')
print(res)
```

入参描述

参数	是否必选	参数含义	配置方法及要求
alias	否	指定需删除的 Collection 或 CollectionView 的别名。	Collection 与 CollectionView 别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

```
{'affectedCount': 1}
```

参数名	参数含义
affectedCount	影响行数，即为删除别名的集合数量。

Document 操作
插入数据

最近更新时间：2025-02-24 19:45:02

接口定义

upsert() 接口用于在 Base 类数据库创建的 Collection 中插入向量数据。

```
def upsert(
    documents: List[Document | Dict],
    timeout: float | None = None,
    build_index: bool = True,
    **kwargs: Any
) -> Any
```

使用示例

说明：

- **在插入数据时， Collection 中已经存在相同 ID 的 Document，则会删除源 Document 后插入新的 Document 数据。**
- **如果 Collection 在创建时，配置 Embedding 参数，则插入文本，Embedding 服务会自动将文本信息转换为向量数据，存入数据库。**

写入向量数据

如果您无需使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能做向量化，则可以直接写入向量数据。

```
import tcvectordb
from tcvectordb.model.document import Document

# 写入数据。
# 参数 build_index 为 True,指写入数据同时重新创建索引。
res = client.upsert(
            database_name='db-test',
            collection_name='book-vector',
            documents=[
```

```
                Document(id='0001', vector=[
                        0.2123, 0.23, 0.213], author='罗贯中',
bookName='三国演义', page=21, tags=['曹操','诸葛亮','刘备']),
                Document(id='0002', vector=[
                        0.2123, 0.22, 0.213], author='吴承恩',
bookName='西游记', page=22, tags=['孙悟空','猪八戒','唐僧']),
                Document(id='0003', vector=[
                        0.2123, 0.21, 0.213], author='曹雪芹',
bookName='红楼梦', page=23, tags=['贾宝玉','林黛玉','王熙凤'])
            ],
            build_index=True
)
print(res, flush=True)
```

写入原始文本

如果您使用 Embedding 功能，在 create_collection() 建表时，配置 Embedding 模型相关参数之后，便可以通过 **upsert()** 接口可直接传入原始文本。Embedding 模型会将原始文本转换为向量数据，并将转换后的向量数据以及原始文本一并存入数据库。具体信息，请参见 Embedding 介绍 。如下示例，基于 create_collection() 创建的集合 **book-emb**，写入原始文本。

```
import tcvectordb
from tcvectordb.model.document import Document

# 写入数据。
# 参数 build_index 为 True,指写入数据同时重新创建索引。
res = client.upsert(
            database_name='db-test',
            collection_name='book-emb',
            documents=[
                Document(id='0001', text="话说天下大势，分久必合，合久必分。", author='罗贯中', bookName='三国演义', page=21, tags=['曹操','诸葛亮','刘备']),
                Document(id='0002', text="混沌未分天地乱，茫茫渺渺无人间。", author='吴承恩', bookName='西游记', page=22, tags=['孙悟空','猪八戒','唐僧']),
                Document(id='0003', text="甄士隐梦幻识通灵，贾雨村风尘怀闺秀。", author='曹雪芹', bookName='红楼梦', page=23, tags=['贾宝玉','林黛玉','王熙凤'])
            ],
```

```
            build_index=True
        )
print(res, flush=True)
```

写入稠密向量 + 稀疏向量

如果您无需使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能做向量化，则可以直接写入向量数据与稀疏向量。

```
import tcvectordb
from tcvectordb.model.document import Document
from tcvdb_text.encoder.bm25 import BM25Encoder

texts = [
    '《三国演义》是元末明初罗贯中创作的长篇章回体历史演义小说',
    '《西游记》是明代吴承恩创作的长篇章回体神魔小说',
    '《红楼梦》是清代曹雪芹创作的长篇章回体小说',
]

bm25 = BM25Encoder.default('zh')
sparse_vectors: List[SparseVector] = bm25.encode_texts(texts)

# 写入数据。
# 参数 build_index 为 True,指写入数据同时重新创建索引。
res = client.upsert(
            database_name='db-test',
            collection_name='book-vector',
            documents=[
                Document(id='0001', vector=[
                        0.1123, 0.13, 0.113],
sparse_vector=sparse_vectors[0], author='罗贯中', bookName='三国演义',
page=21, tags=['曹操','诸葛亮','刘备']),
                Document(id='0002', vector=[
                        0.5123, 0.52, 0.513],
sparse_vector=sparse_vectors[0], author='吴承恩', bookName='西游记',
page=22, tags=['孙悟空','猪八戒','唐僧']),
                Document(id='0003', vector=[
                        0.9123, 0.91, 0.913],
sparse_vector=sparse_vectors[0], author='曹雪芹', bookName='红楼梦',
page=23, tags=['贾宝玉','林黛玉','王熙凤'])
```

```
            ],
            build_index=True
)
print(res, flush=True)
```

原始文本 + 稀疏向量

```
from typing import List
import tcvectordb
from tcvectordb.model.document import Document
from tcvdb_text.encoder.bm25 import BM25Encoder

texts = [
    '腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。',
    '作为专门为处理输入向量查询而设计的数据库，它支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，高达百万级 QPS 及毫秒级查询延迟。',
    '不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。',
    '腾讯云向量数据库（Tencent Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 ',
    '腾讯云向量数据库可以和大语言模型 LLM 配合使用。企业的私域数据在经过文本分割、向量化后，可以存储在腾讯云向量数据库中，构建起企业专属的外部知识库，从而在后续的检索任务中，为大模型提供提示信息，辅助大模型生成更加准确的答案。',
]

bm25 = BM25Encoder.default('zh')
sparse_vectors: List[SparseVector] = bm25.encode_texts(texts)
# 写入数据。
# 参数 build_index 为 True,指写入数据同时重新创建索引。
res = client.upsert(
            database_name='db-test',
            collection_name='book-emb',
            documents=[
                {'id': '0001', 'text': texts[0], 'sparse_vector':
sparse_vectors[0]},
                {'id': '0002', 'text': texts[1], 'sparse_vector':
sparse_vectors[1]},
```

```
                {'id': '0003', 'text': texts[2], 'sparse_vector':
sparse_vectors[2]},
                {'id': '0004', 'text': texts[3], 'sparse_vector':
sparse_vectors[3]},
            ]
)
print(res, flush=True)
```

写入二进制数据

```
import tcvectordb
from tcvectordb.model.document import Document

# 写入数据。
# 参数 build_index 为 True,指写入数据同时重新创建索引。
res = client.upsert(
            database_name='db-test',
            collection_name='bin-vector',
            documents=[
                Document(id='0001',
                         vector=binary_to_uint8([1, 1, 1, 0, 0, 1,
0, 1,  0, 0, 0, 0, 0, 0, 0, 0]),
                         bookName='西游记'),
                Document(id='0002',
                         vector=binary_to_uint8([1, 1, 0, 0, 1, 0,
1, 1,  0, 0, 0, 0, 0, 0, 0, 0]),
                         bookName='西游记'),
                Document(id='0003',
                         vector=binary_to_uint8([0, 0, 0, 1, 0, 0,
0, 0,  1, 1, 0, 1, 0, 0, 1, 0]),
                         bookName='三国演义'),
            ]
)
print(res, flush=True)
```

写入 json 格式的数据

```
import tcvectordb
from tcvectordb.model.document import Document

res = client.upsert(
    database_name="db-test-1",
    collection_name="coll-json-autoid",
    documents=[
        {
            'id': '0001',
            'vector': [0.4],
            'bookInfo': {
                'bookName': '三国演义',
                'author': '罗贯中',
            }
        },
        {
            'id': '0002',
            'vector': [0.3],
            'bookInfo': {
                'bookName': '西游记',
                'author': '吴承恩',
            }
        }
    ]
)
print(res, flush=True)
```

入参描述

参数名称	参数含义	子参数	是否必选	配置方法及要求
database_name	指定写入的数据库。	-	是	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

collection_name	指定写入的集合名。	-	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
build_index	指定是否需要更新索引	-	否	取值如下所示： • **True**：需更新索引。默认值是 **True**。 • **False**：不更新索引。 **注意：** 如果创建 Collection 选择的索引类型为 IVF 系列： • 当第一次写入时，当前集合还没有向量索引，此时 **build_index** 必须为 **False**。插入原始数据之后，需通过 rebuild_index() 训练数据并重建索引。 • 当集合已经调用过 rebuild_index() 创建索引后，集合已经存在向量索引，此时： ○ 如果 **build_index = True**，表示新写入的数据会加入到已有的 IVF 索引中，但不会更新索引结构，此时新写入的数据可以被检索到。 ○ 如果 **build_index = False**，表示新写入的数据不会加入到已有的 IVF 索引中，此时新写入的数据无法被检索到。
documents	指定要插入的 Document 数据，是一个数组，支持单次插入一条或者多条 Document，最大可插入 1000 条。	id	是	Document 主键，长度限制为[1,128]。
		vector	否	表示文档的向量值， • 集合的向量索引为FLAT/HNSW/IVF时，直接传入向量数据，以32位浮点数存储，如[0.2123, 0.9281, ..., 0.6712]。 • 集合的向量索引为二进制索引时，写入向量时可以直接写入二进制数据，也可以以十进制写入。十进制写入数据时，数据维度是创建集合时指定维度的1/8。例如：upsert 时 vector=[12, 21]，则创建集合时维度为16维，其代表的二进制数据为[00001100, 00010101]。

			说明： 如果使用 Embedding 功能做向量化，则无需写入向量数据。
		text	否	该参数为 创建集合 时，Embedding 参数 **field** 对应指定的文本字段名。该请求示例中为 text，您可以自定义其他便于识别的字段名。 **说明：** • 写入原始文本数据，系统会自动从该字段中提取原始文本信息，并将其转换为向量数据，并将原始文本以及转化后的向量数据一起存储在数据库中。 • 输入的文本长度有限制，超过512个 token，约400个字符之后会自动截断。
		sparse_vector	否	写入稀疏向量数据值。**sparse_vector** 为创建集合时自定义的稀疏向量的字段名。
		other_scalar_field	否	其他标量字段，用于存储文档的其他信息。例如：请求示例中的 bookName、author、page、tags。 **说明：** • 若字段为存储数据过期时间戳的标量字段，需以标准的 Unix 时间戳格式为该字段赋值，且数据类型为 uint64。例如：`'expire_at': int(time.time()) + 10*60`。具体示例，请参见 ttl.py 。 • 若时间戳字段为空未指定具体的过期时间，插入的该条数据将不会执行过期删除操作，数据将永久保留。 • 若时间戳字段传入 Value 的数据类型有误或者不符合 Unix 时间戳规范时，将会报出错误信息，请根据提示信息修改。

				• 若时间戳字段指定的过期时间小于当前插入数据时的时间，则在下一个定时删除操作时，直接删除该条数据。

出参描述

```
{
    'code': 0,
    'msg': 'Operation success',
    'affectedCount': 3,
    'embeddingExtraInfo': {'tokenUsed': 285}
}
```

参数名	参数含义
affectedCount	影响行数，即为插入的数据条数。
embeddingExtraInfo	参数 tokenUsed，返回使用 Embedding 功能已经消耗的 Token 数量。

版权所有：腾讯云计算（北京）有限责任公司

精确查询

最近更新时间：2025-07-01 10:19:42

接口定义

query() 用于精确查找与查询条件完全匹配的数据。

```
def query(
    document_ids: List | None = None,
    retrieve_vector: bool = False,
    limit: int | None = None,
    offset: int | None = None,
    filter: Filter | str = None,
    output_fields: List[str] | None = None,
    timeout: float | None = None,
    sort: dict | None = None
) -> List[Dict]
```

使用示例

说明：

- **支持根据主键 id（Document ID），搭配自定义的标量字段的 Filter 表达式一并查询数据。**
- **支持指定查询起始位置 offset 和返回数量 limit，精确检索查询区间内的数据，实现数据 SCAN 能力。**

```
import tcvectordb
from tcvectordb.model.document import Filter, SearchParams

# 设置标量字段的 Filter 表达式，便于过滤需查询的数据，支持直接以字符串的形式赋值
# filter_param=Filter(Filter.In("bookName",["三国演义", "西游
记"])).And(Filter.Include("tags",["曹操", "诸葛亮"]))
filter_param='bookName in ("三国演义","西游记") and (tags include ("曹
操","诸葛亮"))'
# 调用接口 query() 查询数据
doc_list = client.query(
            database_name='db-test',
            collection_name='book-vector',
            document_ids=['0001','0002','0003'],
            retrieve_vector=True,
```

```
            filter=filter_param,
            limit=3,
            offset=0,
            output_fields=['bookName','author','page']
)
for doc in doc_list:
    print(doc)
```

入参描述

参数	是否必选	参数含义	参数配置
database_name	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定检索的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
documentIds	否	表示要查询的文档的所有 ID。	每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。
retrieve_vector	否	标识是否需要返回检索结果的向量值。	取值如下： • True：需要。 • False：不需要。默认为 False。

filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中
			• <field_name>：表示要过滤的字段名。
			• <operator>：表示要使用的运算符。
			  ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
			  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如，expired _time > 1623388524。

			○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter_param='bookInfo.bookName in ("三国演义","西游记")'` • <value>：表示要匹配的值。
limit	是	每页返回的 Document 数量。	每页返回的 Document 数量。默认为10。 • 数据类型：uint 64 • 取值范围：[1,16384] ⚠ **注意：** 若使用 **query** 检索数据时，不配置 **documentIds** 和 **filter** 参数，则必须配置 **offset** 和 **limit** 参数，返回从 offset 开始的 limit 条数据，避免遍历所有数据而浪费不必要的资源。
offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。	取值要求如下： • 取值：为 **limit** 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。
output_fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 ⓘ **说明：** • **output_fields** 与 **retrieve_vector** 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。

			• 输出 Json 字段时，**output_fields** 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，**output_fields** 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。
sort	否	指定查询结果按照特定字段升序或降序排序。	数据类型为数组。 `sort={ 'fieldName': 'page', 'direction': 'desc' },` • **fieldName**：指定特定字段，查询结果将依据该字段确定输出顺序。当前排序字段仅支持 uint64 的类型，且该字段为标量字段索引，可通过接口 describe_collection() 确认该字段是否为索引字段。 **说明：** • 当满足条件的数据量小于或等于1000时，orderby 操作将正常生效，否则会引发错误。 • 目前仅支持对单个字段进行排序，暂时不支持多字段排序功能。 • **direction**：指定排序的方向。 ○ desc：降序排序。 ○ asc：升序排序。
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

检索结果，如下所示。

```
{'id': '0001', 'score': 0.0, 'vector': [0.21230000257492065,
0.23000000417232513, 0.21299999952316284], 'sparse_vector': [[2,
```

```
0.9599999785423279], [5, 0.5299999713897705], [100,
0.4429999887943268]], 'page': 21, 'author': '罗贯中', 'bookName': '三国演
义'}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
author、page、section	Document 自定义扩展的标量字段。

基于向量数据相似度检索

最近更新时间：2025-07-01 10:19:42

接口定义

基于相似度匹配的查询方式，search() 接口据输入的1到 N 向量数据，查找存储在数据库中的相似性向量，返回指定的 Top K 个最相似的文档。

```
def search(
    vectors: List[List[float]] | Any,
    filter: Filter | str = None,
    params: Any | None = None,
    retrieve_vector: bool = False,
    limit: int = 10,
    output_fields: List[str] | None = None,
    timeout: float | None = None,
    return_pd_object: bool = False,
    radius: float | None = None
) -> List[List[Dict | Document]]
```

使用示例

基于向量数据相似性检索

```
import tcvectordb
from tcvectordb.model.document import Document, Filter, SearchParams

# search topn similary documents with filter
# vectors 指定了检索的向量数据
# filter 指定了过滤条件
# params 指定索引类型对应的查询参数，HNSW 类型需要设置 ef，指定查询的遍历范围;IVF 系列需要设置 nprobe,指定查询的单位数量
# retrieve_vector 指定是否输出向量字段
# limit 指定返回最相似的 Top K 条结果。如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
# output_fields 指定输出字段
doc_lists = client.search(
            database_name='db-test',
            collection_name='book-vector',
```

```
            vectors=[[0.3123, 0.43, 0.213],[0.315, 0.4, 0.216],
[0.40, 0.38, 0.26]],
            # filter=Filter(Filter.In("bookName",["三国演义", "西游
记"])).And(Filter.Include("tags",["曹操", "孙悟空"])),
            filter='bookName in ("三国演义","西游记") and (tags include
("曹操","孙悟空"))',
            params=SearchParams(ef=200),
            retrieve_vector=True,
            limit=3,
            output_fields=['bookName','author']
        )
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
       print(doc)
```

输出信息，如下所示。

```
0
{'id': '0001', 'score': 0.9714228510856628, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.44299998879432681]], 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.9668837785720825, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591]], 'author': '吴承恩', 'bookName': '西游记'}
1
{'id': '0001', 'score': 0.9784632325172424, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.44299998879432681]], 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.9747834205627441, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591]], 'author': '吴承恩', 'bookName': '西游记'}
2
{'id': '0001', 'score': 0.9860693216323853, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.44299998879432681]], 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.9852011203765869, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
```

```
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591], 'author': '吴承恩', 'bookName': '西游记'}
```

二进制向量数据相似性检索

```
import tcvectordb
from tcvectordb.model.document import Document, Filter, SearchParams

# search topn similary documents with filter
# vectors 指定了检索的向量数据
# filter 指定了过滤条件
# params 指定索引类型对应的查询参数，HNSW 类型需要设置 ef，指定查询的遍历范
围;IVF 系列需要设置 nprobe,指定查询的单位数量
# retrieve_vector 指定是否输出向量字段
# limit 指定返回最相似的 Top K 条结果。如果插入的数据不足 K 条，则返回实际插入
的 Document 数量。
# output_fields 指定输出字段
doc_lists = client.search(
            database_name='db-test',
            collection_name='bin-vector',
            # vectors=[binary_to_uint8([0, 0, 0, 0, 0, 0, 0, 0,  1,
1, 0, 1, 1, 1, 0, 1])],
            vectors=[[12, 21],[12, 30]],
            retrieve_vector=True,
            limit=3,
            output_fields=['bookName']
        )
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
       print(doc)
```

输出信息，如下所示。

```
0
{'id': '0001', 'score': 8.0, 'vector': [229.0, 0.0], 'bookName': '西
游记'}
{'id': '0002', 'score': 8.0, 'vector': [203.0, 0.0], 'bookName': '西
游记'}
```

```
{'id': '0003', 'score': 8.0, 'vector': [16.0, 210.0], 'bookName': '三
国演义'}
1
{'id': '0003', 'score': 7.0, 'vector': [16.0, 210.0], 'bookName': '三
国演义'}
{'id': '0001', 'score': 9.0, 'vector': [229.0, 0.0], 'bookName': '西
游记'}
{'id': '0002', 'score': 9.0, 'vector': [203.0, 0.0], 'bookName': '西
游记'}
```

Json 类型数据相似性检索

```
import tcvectordb
from tcvectordb.model.document import Document, Filter, SearchParams

# search topn similary documents with filter
# vectors 指定了检索的向量数据
# filter 指定了过滤条件
# params 指定索引类型对应的查询参数，HNSW 类型需要设置 ef，指定查询的遍历范
围;IVF 系列需要设置 nprobe,指定查询的单位数量
# retrieve_vector 指定是否输出向量字段
# limit 指定返回最相似的 Top K 条结果。如果插入的数据不足 K 条，则返回实际插入
的 Document 数量。
# output_fields 指定输出字段
doc_lists = client.search(
            database_name='db-test',
            collection_name='coll-json-autoid',
            vectors=[[0.3123],[0.315],[0.40]],
            filter='bookInfo.bookName in ("三国演义","西游记")',
            params=SearchParams(ef=200),
            retrieve_vector=True,
            limit=3,
            output_fields=['bookInfo','id']
        )
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)
```

检索结果，如下所示。

```
0
{'id': 'E3AB9477-670E-3025-B9BE-189A8DBF3944', 'vector':
[0.4000000059604645], 'score': 0.12492, 'bookInfo': {'bookName': '三
国演义', 'author': '罗贯中'}}
{'id': 'D7678BA3-6485-EAC4-59FD-B4AC3D5D033A', 'vector':
[0.30000001192092896], 'score': 0.09369, 'bookInfo': {'bookName': '西
游记', 'author': '吴承恩'}}
{'id': '23D97B7E-411F-3BEC-4D91-0732FD8710D1', 'vector':
[0.30000001192092896], 'score': 0.09369, 'bookInfo': {'bookName': '西
游记', 'author': '吴承恩'}}
1
{'id': 'E3AB9477-670E-3025-B9BE-189A8DBF3944', 'vector':
[0.4000000059604645], 'score': 0.126, 'bookInfo': {'bookName': '三国
演义', 'author': '罗贯中'}}
{'id': 'D7678BA3-6485-EAC4-59FD-B4AC3D5D033A', 'vector':
[0.30000001192092896], 'score': 0.0945, 'bookInfo': {'bookName': '西
游记', 'author': '吴承恩'}}
{'id': '23D97B7E-411F-3BEC-4D91-0732FD8710D1', 'vector':
[0.30000001192092896], 'score': 0.0945, 'bookInfo': {'bookName': '西
游记', 'author': '吴承恩'}}
2
{'id': 'E3AB9477-670E-3025-B9BE-189A8DBF3944', 'vector':
[0.4000000059604645], 'score': 0.16, 'bookInfo': {'bookName': '三国演
义', 'author': '罗贯中'}}
{'id': 'D7678BA3-6485-EAC4-59FD-B4AC3D5D033A', 'vector':
[0.30000001192092896], 'score': 0.12, 'bookInfo': {'bookName': '西游
记', 'author': '吴承恩'}}
{'id': '23D97B7E-411F-3BEC-4D91-0732FD8710D1', 'vector':
[0.30000001192092896], 'score': 0.12, 'bookInfo': {'bookName': '西游
记', 'author': '吴承恩'}}
```

入参描述

参数名	是否必选	参数含义	配置方法

database_name	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定检索的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
vectors	是	表示要查询的向量列表。	数组元素数量最大为20，即每次最多允许传入20个向量。相似性检索返回每一个向量的 Top K 个相似数据。

filter	否	设置查询过滤条件。	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中
			• <field_name>：表示要过滤的字段名。
			• <operator>：表示要使用的运算符。
			  ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
			  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。
			  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
			  ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。
			`filter_param='bookInfo.bookName in ("三国演义","西游记")'`
			• <value>：表示要匹配的值。

params	否	指定索引查询参数。	索引类型不同，检索时，所需配置的参数不同。 • FLAT：无需指定参数。

			• HNSW 类型：需配置参数 **ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为10。 • IVF 系列：需设置参数 **nprobe** ，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 list_collections() 查看。
retrieve_vector	否	标识是否需要返回向量值。	取值如下所示： • **True**：需要。 • **False**：不需要。默认为 False。
limit	是	指定返回最相似的 Top K 的 K 的值。	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= radius 的文档。
output_fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • **output_fields** 与 **retrieve_vector** 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，**output_fields** 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}` ，**output_fields** 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围大于等于0。

出参描述

检索结果，如下所示。

> **说明：**
>
> - 输出结果的顺序，与搜索时设置的 vectors 配置的向量值的顺序一致。如下示例，0下面的三行结果对应 `[0.3123, 0.43, 0.213]` 向量的相似度查询结果。1下面的三行结果对应 `[0.315, 0.4, 0.216]` 的查询结果。
> - 每一个查询结果都返回 TopK 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
> - 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。

```
0
{'id': '0001', 'score': 0.9714228510856628, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.4429999887943268]], 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.9668837785720825, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591]], 'author': '吴承恩', 'bookName': '西游记'}
1
{'id': '0001', 'score': 0.9784632325172424, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.4429999887943268]], 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.9747834205627441, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591]], 'author': '吴承恩', 'bookName': '西游记'}
2
{'id': '0001', 'score': 0.9860693216323853, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.4429999887943268]], 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.9852011203765869, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591]], 'author': '吴承恩', 'bookName': '西游记'}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
score	表示查询向量与检索结果向量之间的相似性计算分数。欧式距离（L2）与汉明距离（Hamming Distance）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。
author、page、section	Document 其他自定义的标量字段。

版权所有：腾讯云计算（北京）有限责任公司

基于 Doc ID 相似度检索

最近更新时间：2025-07-01 10:19:42

接口定义

基于相似度匹配的查询方式，**search_by_id()** 根据输入的1到 N 个文档 ID，查找存储在数据库中的相应向量，并执行相似性 TopK 检索。

```
def search_by_id(self,
                 database_name: str,
                 collection_name: str,
                 document_ids: List[str],
                 filter: Union[Filter, str] = None,
                 params=None,
                 retrieve_vector: bool = False,
                 limit: int = 10,
                 output_fields: Optional[List[str]] = None,
                 timeout: Optional[float] = None,
                 ) -> List[List[Dict]]
```

使用示例

如下示例已经通过 create_database() 创建数据库 **db-test**，已通过 create_collection() 创建集合为 **book-vector**，通过 searchById() 进行数据检索。

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams
from tcvectordb.model.document import Document, Filter, SearchParams

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)


# search by document id
# document_ids 指定了检索文档 id
# filter 指定了过滤条件
```

```
# params 指定索引类型对应的查询参数，HNSW 类型需要设置 ef，指定查询的遍历范围；IVF
系列需要设置 nprobe，指定查询的单位数量
# retrieve_vector 指定是否输出向量字段
# limit 指定返回最相似的 Top K 条结果。如果插入的数据不足 K 条，则返回实际插入的
Document 数量。
# output_fields 指定输出字段
doc_lists = client.search_by_id(
                database_name='db-test',
                collection_name='book-vector',
                document_ids=['0001','0002'],
                # filter=Filter(Filter.In("bookName",["三国演义", "西游
记"])),
                filter='bookName in ("三国演义","西游记")',
                params=SearchParams(ef=200),
                limit=3,
                retrieve_vector=True,
                output_fields=['bookName','author']
            )

for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)
```

入参描述

参数	是否必选	参数含义	配置方法
database_name	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定检索的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
document_ids	是	待查询的文档 ID。	数组元素数量最大为20。即每次最多允许传入20个 ID。而每个 ID 长度限制为[1,128]。相似性检索返回每一个 ID 的 Top K

			个相似数据。
filter	否	设置查询过滤条件。	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter_param='bookInfo.bookName in ("三国演义","西游记")'` • <value>：表示要匹配的值。
params	否	指定索引查询参数。	索引类型不同，检索时，所需配置的参数不同。 • FLAT：无需指定参数。 • HNSW 类型：需配置参数 **ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为10。 • IVF 系列：需设置参数 **nprobe**，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 list_collections() 查看。
retrieve_vector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • True：需要。 • False：不需要。默认为 False。
limit	是	指定返回最相似的 Top K 的	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。

		K 的值。	
radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= radius 的文档。
output_fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • output_fields 与 retrieve_vector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，output_fields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，output_fields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

查看输出，如下所示。查询参数 document_ids = ['0001','0002']，查询结果中0下面的三行为 id 为0001进行相似度查询的结果，1下面的三行为 id 为0002进行相似度查询的结果。

说明：

- 输出的 Document ID 顺序与查询时配置的参数 document_ids 输入的顺序一致。
- 每一个查询结果都返回 TopK 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。相似程度则通过 L2（欧几里得距离）、IP（内积）或 COSINE（余弦相似度）计算得出的分数来衡量，输出参数 score 表示相似性计算分数。其中，欧式距离（L2）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。

```
0
{'id': '0001', 'score': 1.0000001192092896, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
```

```
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.44299998879432681]], 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.9997729659080505, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591]], 'author': '吴承恩', 'bookName': '西游记'}
1
{'id': '0002', 'score': 1.000000238418579, 'vector':
[0.21230000257492065, 0.2199999988079071, 0.21299999952316284],
'sparse_vector': [[2, 0.9700000286102295], [5, 0.5400000214576721],
[100, 0.44400000572204591]], 'author': '吴承恩', 'bookName': '西游记'}
{'id': '0001', 'score': 0.9997729659080505, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.44299998879432681]], 'author': '罗贯中', 'bookName': '三国演义'}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
score	表示查询向量与检索结果向量之间的相似性计算分数。
author、page、section	Document 其他自定义的标量字段。

基于原始文本相似度检索

最近更新时间：2025-07-01 10:19:42

接口定义

若在 Base 类数据库中创建 Collection 时，已配置 Embedding 模型，则可使用 **search_by_text()** 接口输入原始文本，在数据库中检索与原始文档相似的文档，返回指定的 Top K 个最相似的 Document。

```
def search_by_text(self,
                   database_name: str,
                   collection_name: str,
                   embedding_items: List[str],
                   filter: Union[Filter, str] = None,
                   params=None,
                   retrieve_vector: bool = False,
                   limit: int = 10,
                   output_fields: Optional[List[str]] = None,
                   timeout: Optional[float] = None,
                   ) -> List[List[Dict]]
```

使用示例

```
import tcvectordb
from tcvectordb.model.document import Document, Filter, SearchParams
# search by text
# embeddingItems 指定了检索的文本
# filter 指定了过滤条件
# params 指定索引类型对应的查询参数，HNSW 类型需要设置 ef，指定查询的遍历范围;IVF
系列需要设置 nprobe,指定查询的单位数量

# limit 指定返回最相似的 Top K 条结果。如果插入的数据不足 K 条，则返回实际插入的
Document 数量。
# output_fields 指定输出字段
doc_lists = client.search_by_text(
                database_name='db-test',
                collection_name='book-emb',
                embedding_items=['天下大势，分久必合，合久必分'],
                #filter=Filter(Filter.In("bookName",["三国演义", "西游
记"])),
                filter='bookName in ("三国演义","西游记")',
                params=SearchParams(ef=200),
```

```
                limit=3,
                retrieve_vector=False,
                output_fields=['bookName','author']
            )
# printf
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)
```

入参描述

参数	是否必选	参数含义	配置方法
database_name	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定检索的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
embedding_items	是	待检索的文本信息。	输入文本信息，用于检索与该文本信息相似的数据。 • 类型：字符串数组。 • 范围：数组元素最大批量为20 。 • 相似性检索返回每一个文本的 TOP K 个相似性数据。

filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中
			• <field_name>：表示要过滤的字段名。
			• <operator>：表示要使用的运算符。
			  ○ string：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
			  ○ uint64：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。

			○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter_param='bookInfo.bookName in ("三国演义","西游记")'` • <value>：表示要匹配的值。
params	否	指定索引查询参数。	索引类型不同，检索时，所需配置的参数不同。 • FLAT：无需指定参数。 • HNSW 类型：需配置参数 **ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为10。 • IVF 系列：需设置参数 **nprobe**，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 list_collections() 查看。
retrieve_vector	否	标识是否需要返回向量值。	取值如下所示： • True：需要。 • False：不需要。默认为 False。
limit	否	指定返回最相似的 Top K 的 K 的值。	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= radius 的文档。
output_fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：**

- output_fields 与 retrieve_vector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。
- 输出 Json 字段时，output_fields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，output_fields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

说明：

- 每一个查询结果都返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。相似程度则通过 L2（欧几里得距离）、IP（内积）或 COSINE（余弦相似度）计算得出的分数来衡量，输出参数 score 表示相似性计算分数。其中，欧式距离（L2）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。

```
0
{'id': '0001', 'score': 0.9792182445526123, 'author': '罗贯中',
'bookName': '三国演义'}
{'id': '0002', 'score': 0.790553867816925, 'author': '吴承恩',
'bookName': '西游记'}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
score	表示查询向量与检索结果向量之间的相似性计算分数。
author、page、section	Document 其他自定义的标量字段。

单路稀疏向量相似度检索

最近更新时间：2025-07-01 10:19:42

接口定义

本接口 `fulltext_search()` 基于稀疏向量实现全文检索，对查询词与文档内容进行匹配，并按相关度排序结果。

```
def fulltext_search(self,
                    database_name: str,
                    collection_name: str,
                    data: SparseVector,
                    field_name: str = 'sparse_vector',
                    filter: Union[Filter, str] = None,
                    retrieve_vector: Optional[bool] = None,
                    output_fields: Optional[List[str]] = None,
                    limit: Optional[int] = None,
                    terminate_after: Optional[int] = None,
                    cutoff_frequency: Optional[float] = None,
                    **kwargs) -> List[Dict]:
```

使用示例

fulltext_search() 完整的代码示例，请参见 fulltext_search.py 。

```
import json
from typing import List
from tcvdb_text.encoder.bm25 import BM25Encoder

## 初始化稀疏向量编码器
bm25 = BM25Encoder.default('zh')
## 稀疏向量相似性检索
res = client.fulltext_search(
    database_name='db-test',
    collection_name='sparse-vector-test',
    data=bm25.encode_queries('什么是腾讯云数据库'),
    field_name="sparse_vector",
    filter=None,
    retrieve_vector=False,
    # output_fields=['id', 'sparse_vector', 'text'],
    limit=10,
    # terminate_after=4000,
```

版权所有：腾讯云计算（北京）有限责任公司

```
    # cutoff_frequency=0.1,
)
print(json.dumps(res, indent=2, ensure_ascii=False))
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database_name	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定检索的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

match	否	稀疏向量检索配置。	• **field_name**：稀疏向量的字段名，例如：sparse_vector。
			• **data**：检索的稀疏向量，当前仅支持输入一个稀疏向量。
			• **terminate_after**：设置检索数量上限，即检索数量超过 terminate_after 值之后，提前终止检索行为，可以有效的控制速率，在数据量较大的情况下，推荐经验值：4000。
			• **cutoff_frequency**：设置高频词出现的频次或频率上限，即词频大于 cutoff_frequency 的值之后，该关键词被忽略。
			  ○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。
			  ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。
			说明：
			cutoff_frequency 可以排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：0.1。

filter	否	设置 Filter 表达式，过滤所需的数据。	Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中：

			• <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `filter_param='bookInfo.bookName in ("三国演义","西游记")'` • <value>：表示要匹配的值。
retrieve_vector	否	标识是否需要返回检索结果的向量值。	• **True**：需要。 • **False**：不需要。默认为 False。
limit	是	指定最终返回最相似的 Top K 条数据的 K 的值。	K 为大于0的正整数。
output_fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • **output_fields** 与 **retrieve_vector** 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，**output_fields** 仅支持指定 Json 字段的名称，而不支持直接指定 Json

			字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，output_fields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

```
[
  {
    "id": "0001",
    "score": 0.75646,
    "text": "腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。"
  },
  {
    "id": "0004",
    "score": 0.71671,
    "text": "腾讯云向量数据库（Tencent Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 "
  },
  {
    "id": "0002",
    "score": 0.26303,
    "text": "作为专门为处理输入向量查询而设计的数据库，它支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，高达百万级 QPS 及毫秒级查询延迟。"
  }
]
```

参数名	子参数	参数含义
documents	id	Document 的 ID 信息。
	sparse_vector	Document 的稀疏向量。
	score	表示查询向量与检索结果向量之间的相似性计算分数。
	other_scalar_field	Document 其他自定义的标量字段。例如：author、bookName、page 等。

稠密向量与稀疏向量高阶混合检索

最近更新时间：2025-07-01 10:19:42

接口定义

```
def hybrid_search(
    database_name: str,
    collection_name: str,
    ann: List[AnnSearch] | AnnSearch | None = None,
    match: List[KeywordSearch] | KeywordSearch | None = None,
    filter: Filter | str = None,
    rerank: Rerank | None = None,
    retrieve_vector: bool | None = None,
    output_fields: List[str] | None = None,
    limit: int | None = None,
    timeout: float | None = None,
    return_pd_object: bool = False,
    **kwargs: Any
) -> List[List[Dict | Document]]
```

使用示例

说明：

- 如需使用稀疏向量，至少需要升级 Python SDK 至1.4.5版本。推荐使用1.5.0版本使用最新的 terminate_after、cutoff_frequency 等新特性，使得召回效果和性能更佳。
- 腾讯云向量数据库团队提供了一款稀疏向量工具包，旨在帮助用户高效生成稀疏向量。具体信息，请参见 工具介绍 。

根据向量数据检索

```
from tcvectordb.model.document import Document, AnnSearch,
WeightedRerank, KeywordSearch
from tcvdb_text.encoder.bm25 import BM25Encoder

## 初始化稀疏向量编码器
bm25 = BM25Encoder.default('zh')
doc_lists = client.hybrid_search(
```

```
            database_name='db-test',
            collection_name='book-vector',
            ann=[
                AnnSearch(
                    field_name="vector",
                    data=[0.7123, 0.83, 0.913],
                ),
            ],
            match=[
                KeywordSearch(
                    field_name="sparse_vector",
                    data=bm25.encode_queries('红楼梦是什么小说'),
                    terminate_after=4000,
                    cutoff_frequency=0.1,
                ),
            ],
            rerank=WeightedRerank(
                field_list=['vector', 'sparse_vector'],
                weight=[0.3, 0.7],
            ),
            retrieve_vector=False,
            limit=1,
        )
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)
```

输出信息，如下所示。

```
0
{'id': '0003', 'score': 0.68806, 'author': '曹雪芹', 'tags': ['贾宝
玉', '林黛玉', '王熙凤'], 'page': 23, 'bookName': '红楼梦'}
```

应用 Embedding 与稀疏向量混合检索

```
from tcvectordb.model.document import Document, AnnSearch,
WeightedRerank, KeywordSearch
from tcvdb_text.encoder.bm25 import BM25Encoder
```

```
## 初始化稀疏向量编码器
bm25 = BM25Encoder.default('zh')
# search topn similary documents with filter
doc_lists = client.hybrid_search(
            database_name='db-test',
            collection_name='book-emb',
            ann=[
                AnnSearch(
                    field_name="text",
                    data='什么是腾讯云向量数据库',
                    limit=2,
                ),
            ],
            match=[
                KeywordSearch(
                    field_name="sparse_vector",
                    terminate_after=4000,
                    cutoff_frequency=0.1,
                    data=bm25.encode_queries('向量数据库'),
                    limit=2,
                ),
            ],
            rerank=WeightedRerank(
                field_list=['vector', 'sparse_vector'],
                weight=[0.9, 0.1],
            ),
            retrieve_vector=False,
            limit=3,
        )
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)
```

输出信息，如下所示。

```
0
{'id': '0004', 'score': 0.8072301149368286, 'text': '腾讯云向量数据库（Tencent Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 '}
{'id': '0001', 'score': 0.8068472146987915, 'text': '腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存
```

```
储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。'}
{'id': '0002', 'score': 0.7084043622016907, 'text': '作为专门为处理输入
向量查询而设计的数据库，它支持多种索引类型和相似度计算方法，单索引支持10亿级向量规
模，高达百万级 QPS 及毫秒级查询延迟。'}
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database_name	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定检索的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
ann	是	设置向量检索信息。	• **field_name**：检索的字段名，支持设置为：vector 或 text。 **说明：** 其中，vector 为向向量数据的字段名，不可更改；text 为创建 Collection 时，开启 Embedding 功能自定义的原始文本字段名，请以实际定义为准，二者只能选择其中之一。 • **data**：检索的数据列表，当前仅支持输入一条向量或原始文本。 • **params**：检索参数，与索引类型相关。 ○ HNSW 类型：需配置参数 **ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为10。 ○ IVF 系列：需设置参数 **nprobe**，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置。 • **limit**：指定稠密向量返回的数量。

match	否	稀疏向量检索配置。	• **field_name**：稀疏向量的字段名，例如：sparse_vector。
			• **data**：检索的稀疏向量，当前仅支持输入一个稀疏向量。
			• **limit**：指定稀疏向量返回的数量。
			• **terminate_after**：设置检索数量上限，即检索数量超过 terminate_after 值之后，提前终止检索行为，可以有效的控制速率，在数据量较大的情况下，推荐经验值：4000。
			• **cutoff_frequency**：设置高频词出现的频次或频率上限，即词频大于 cutoff_frequency 的值之后，该关键词被忽略。
			  ○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。
			  ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。
			说明：
			cutoff_frequency 可以排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：0.1。

rerank	否	检索中重排序（rerank）的配置。	参数 method 指定 Rerank 的方法，枚举如下值：
			• **WeightedRerank**：基于不同字段的加权组合来进行排序。
			  ○ field_list：列出用于加权计算的字段列表。例如，field_list=: ['vector', 'sparse_vector']表示密集型与稀疏型加权计算。
			  ○ weight：定义了 "field_list" 中每个字段的权重，例如，`weight=[0.9, 0.1]` 表示"vector" 字段的权重是0.9，"sparse_vector" 字段的权重是0.1。
			• **RRFRerank**：一种融合多个搜索方法的排名结果，通过为每个文档分配倒数排名分数并合并这些分数来生成一个新的综合排名，从而提高搜索结果的准确性和相关度。k 是一个用于计算倒数排名分数的常数，其作用是调整分数计算公式，以控制排名分数的分布。例如：`rerank=RRFRerank(k=1)` 。

filter	否	设置 Filter 表达式，过滤所需的数据。	Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中：
			• <field_name>：表示要过滤的字段名。
			• <operator>：表示要使用的运算符。
			  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
			  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：

			expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter_param='bookInfo.bookName in ("三国演义","西游记")'` • <value>：表示要匹配的值。
retrieve_vector	否	标识是否需要返回检索结果的向量值。	• **True**：需要。 • **False**：不需要。默认为 False。
limit	是	指定最终返回最相似的 Top K 条数据的 K 的值。	K 为大于0的正整数。
output_fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • **output_fields** 与 **retrieve_vector** 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，**output_fields** 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}` ，**output_fields** 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

```
0
{'id': '0001', 'score': 0.874280571937561, 'vector':
[0.21230000257492065, 0.23000000417232513, 0.21299999952316284],
'sparse_vector': [[2, 0.9599999785423279], [5, 0.5299999713897705],
[100, 0.4429999887943268]], 'page': 21, 'tags': ['曹操', '诸葛亮', '刘备'],
'author': '罗贯中', 'bookName': '三国演义'}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
sparse_vector	稀疏向量值。
score	表示查询向量与检索结果向量之间的相似性计算分数。
author、page、section	Document 其他自定义的标量字段。

版权所有：腾讯云计算（北京）有限责任公司

删除 Document

最近更新时间：2025-07-01 10:19:42

功能介绍

delete() 接口用于删除指定 id（Document ID）的文档。

```
def delete(
    database_name: str,
    collection_name: str,
    document_ids: List[str] = None,
    filter: Filter | str = None,
    timeout: float | None = None,
    limit: int | None = None
) -> Dict
```

使用示例

> **说明：**
> 索引类型为 FLAT，在1.6及之后版本，支持删除操作，而1.6之前版本，则不支持删除操作。

```
import tcvectordb
from tcvectordb.model.document import Document, Filter

# delete
# 删除指定 id（Document ID），并满足 Filter 表达式的文档。
res = client.delete(
        database_name='db-test',
        collection_name='book-vector-2',
        document_ids=['0001', '0002', '0003'],
        filter='bookName in ("三国演义","西游记")',
        limit=1
),
print(res, flush=True)
```

入参描述

参数名	是否必选	参数含义	配置方法

database_name	是	指定删除的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定删除的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
document_ids	否	表示要删除的 Document 的 ID。	• ID 长度限制为[1,128]。 • 批量删除，数据元素最大值为20。
filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter_param='bookInfo.bookName in`

			("三国演义","西游记")' • <value>：表示要匹配的值。
limit	否	控制删除文档的数量。	若不设置该参数，最大删除16384条数据。 • 数据类型：整型。 • 取值范围：[1,16384]
timeout	否	请求超时时间。	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

```
{
    'code': 0,
    'msg': 'Operation success',
    'affectedCount': 1
}
```

参数名	参数含义
affectedCount	影响行数，即为删除数据的条数。

更新 Document

最近更新时间：2025-07-01 10:19:42

接口定义

update() 接口用于对通过主键（Document ID）与 Filter 表达式过滤检索 Document，对 Document 的部分字段进行更新，同时支持新增字段。

```
def update(self,
           database_name: str,
           collection_name: str,
           data: Union[Document, Dict],
           filter: Union[Filter, str] = None,
           document_ids: Optional[List[str]] = None,
           timeout: Optional[float] = None)
```

使用示例

说明：

- 新增字段，在创建 Collection 时没有为这些字段设置索引，那么新增这些字段时，系统不会自动为其创建索引。
- 不能变更 Document ID 字段，不要求事务完整性。

更新向量数据

集合未配置 Embedding 参数，则直接更新向量数据。如下示例，在集合 **book-vector** 中，基于 upsert() 插入的向量数据，通过 documentIds 与 filter 表达式，过滤出需更新的 Document，更新其 vectors 字段的向量数据，并更新 page 字段值为30，新增字段 test_new_field。

```
import tcvectordb
from tcvectordb.model.document import Document, Filter, SearchParams

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
#设置需更新的字段，或增加新的字段
update_doc = Document(
```

```
        vector=[0.2123, 0.290, 0.213],
        page=30,
        test_new_field="new field value"
        )
# 对满足查询条件的 Document 更新字段
res = client.update(
        database_name='db-test',
        collection_name='book-vector',
        data=update_doc,
        document_ids=['0001','0002','0003'],
        #filter=Filter(Filter.In("bookName",["三国演义", "西游记"]))
        filter='bookName in ("三国演义","西游记")'
        )
print(res, flush=True)
```

更新原始文本

实例在创建 Collection 时，已配置 Embedding 模型，通过 upsert() 写入原始文本，则可输入文本信息，通过 Embedding 将数据向量化更新向量数据。如下示例，基于 upsert() 插入的原始文本，使用 update 接口，通过 documentIds 与 filter 表达式过滤数据，更新其 text 字段的文本信息，更新 page 字段值为30，并新增字段 test_new_field。

```
import tcvectordb
from tcvectordb.model.document import Document, Filter

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 设置需更新的字段
update_doc = Document(text="分久必合，合久必分", page=30,
test_new_field="new field value")
# 对满足查询条件的 Document 更新字段
res = client.update(
                database_name='db-test',
                collection_name='book-emb-1012',
                data=update_doc,
                document_ids=['0001', '0002','0003'],
```

```
                #filter=Filter(Filter.In("bookName",["三国演义", "西游
记"]))
                filter='bookName in ("三国演义","西游记")'
                )
print(res, flush=True)
```

入参描述

参数名称	参数含义	子参数	是否必选	配置方法
database_name	指定更新的数据库名。	-	是	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	指定更新的集合名。	-	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
document_ids	表示要更新的 Document 的 ID。	-	是	• ID 长度限制为[1,128]。 • 批量删除，数据元素最大值为20。

filter	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式	-	否	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中
				• <field_name>：表示要过滤的字段名。
				• <operator>：表示要使用的运算符。
				  ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
				  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。

				○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `filter_param='bookInfo.bookName in ("三国演义","西游记")'` • <value>：表示要匹配的值。
data	设置需更新的字段，或新增字段。	**vector**	否	更新 vector 字段的向量数据。 **说明：** 如果 Collection 在创建时，未配置 Embedding 参数，或者该实例并未开通 Embedding 功能，则只能配置该参数，输入向量数据，更新数据。
		text	否	Embedding 模型输入文本的字段名。该字段在创建 Collection 时定义。本示例为 text 。 • **string**：字符型。 • **float**：浮点型数据。 **说明：** 如果 Collection 在创建时，已配置 Embedding 参数，则只能配置该参数，依据输入的文本信息更新向量数据，并将文本字段与向量数据更新存于数据库。
		old_field	否	当前已存在的字段，更新字段对应的数据。 • 类型：string。 • 字符长度要求：[1,256]。

		new_field	否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。
timeout	请求超时时间。	-	否	• 单位：秒。 • 默认值：VectorDBClient() 接口配置的 timeout 时长。 • 取值范围：大于等于0。

出参描述

```
{
    'code': 0,
    'msg': 'Operation success',
    'affectedCount': 2
}
```

参数名	参数含义
affectedCount	影响行数，即为更新数据的条数。

查询文档数量

最近更新时间：2025-07-01 10:19:42

接口定义

count() 用于查询集合中的文档数量。

```
def count(
    database_name: str,
    collection_name: str,
    filter: Filter | str = None,
    timeout: float = None
) -> int
```

使用示例

说明：

- **支持根据主键 id（Document ID），搭配自定义的标量字段的 Filter 表达式一并查询数据。**
- **支持指定查询起始位置 offset 和返回数量 limit，精确检索查询区间内的数据，实现数据 SCAN 能力。**

```
import tcvectordb
from tcvectordb.model.document import Filter

doc_count = client.count(database_name='db-test',
                         collection_name='book-vector',
                         filter='bookName=\"三国演义\"')
print(doc_count)
```

入参描述

参数	是否必选	参数含义	参数配置
database_name	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

collection_name	是	指定检索的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如，expired _time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `filter_param='bookInfo.bookName in ("三国演义","西游记")'` • <value>：表示要匹配的值。

出参描述

检索结果，如下所示。

```
3
```

参数名	参数含义
count	集合中的文档数量。

上传文件

最近更新时间：2025-05-26 19:31:32

接口定义

upload_file() 接口用于上传文件于 Base 类向量数据库。文件上传后，其内容将被直接拆分并写入指定的 Collection 中，而非 CollectionView。

> **说明：**
> 当前支持导入 Collection 的文件类型包含： Markdown、PDF、Word、PPT。其中，仅 PDF 类型支持解析图片信息。

```
def upload_file(self,
                database_name: str,
                collection_name: str,
                local_file_path: str,
                file_name: Optional[str] = None,
                splitter_process: Optional[SplitterProcess] = None,
                parsing_process: Optional[ParsingProcess] = None,
                embedding_model: Optional[str] = None,
                field_mappings: Optional[Dict[str, str]] = None,
                metadata: Optional[dict] = None,
                ) -> dict:
```

约束限制

- 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
- 该接口当前不支持使用别名获取文件的上传路径与授权。

> **注意：**
> 在创建存储文件内容的 Collection 时，索引字段的设计需遵循以下限制与建议，否则可能导致上传文件失败。
>
> - 文件名字段（file_name）：该字段**必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。字段名可以自定义，在上传文件时，需通过参数 field_mappings 的 filename 映射自定义的字段名。
> - 文件内容字段（text）：该字段用于存储知识点的原始文本内容。由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。字段名可以自定义，但需通过参数 field_mappings 的 text 映射自定义的字段名。

- 文件图片信息字段（image_list）：该字段存放 PDF 文件中图片的 Key 列表。在原始检索出的文本块中，图片位置将以 {key} 的形式进行占位。通过调用接口 get_image_url，可以获取图片 URL 地址列表，这些地址与 Key 一一对应，从而实现将原始 PDF 文档恢复并展示为 HTML 格式。
  - 不建议为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。
  - 字段名可以自定义，但需要在上传文件时通过参数 field_mappings 的 imageList 映射到自定义的字段名。

使用示例

说明：
如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
import tcvectordb
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.enum import ReadConsistency
from tcvectordb.model.collection_view import SplitterProcess,
ParsingProcess

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 上传文件
# 1. local_file_path：指定当前文件在客户端的路径
# 2. file_name：配置文件存储于向量数据库的名称，可不配置，从local_file_path获取
# 3. splitter_process：配置文件拆分规则
# 4. parsing_process：指定文件解析方式
# 5. metadata：定义文件 meta 信息字段，示例中指定了author 与 tags
# 6. field_mappings：文件与数据库中存储的映射字段
# 7. embedding_model：Embedding 模型
res = client.upload_file(
            database_name='db-test',
            collection_name='coll-file-test',
            local_file_path="/data/home/********/腾讯云向量数据库.pdf",
            metadata={
                'author': 'Tencent',
                'tags': ['向量', 'Embedding', 'AI']
            },
```

```
            splitter_process=SplitterProcess(
                append_keywords_to_chunk=True,
                append_title_to_chunk=False,
                # chunk_splitter 参数以正则表达式的方式配置文档拆分方式，如下：
\n{2,} 代表以两个及以上的换行进行拆分，常用在QA对文件拆分中。
                # chunk_splitter="\n{2,}",
            ),
            parsing_process=ParsingProcess(
                parsing_type='VisionModelParsing'
            ),
            embedding_model='bge-base-zh',
            field_mappings={
                 "filename": "file_name",
                 "text": "text",
                 "imageList": "image_list"
                 "chunkNum": "chunk_num",
                 "sectionNum": "section_num",
             },
    )
print(res)
```

输出信息，如下所示。

```
{
  "database": "db-test",
  "collection": "coll-files-test",
  "fileName": "腾讯云向量数据库.pdf",
  "splitterPreprocess": {
    "appendTitleToChunk": false,
    "appendKeywordsToChunk": true
  },
  "parsingProcess": {
    "parsingType": "VisionModelParsing"
  },
  "embeddingModel": "bge-base-zh",
  "fieldMappings": {
    "filename": "file_name",
    "text": "text",
    "imageList": "image_list"
    "chunkNum": "chunk_num",
    "sectionNum": "section_num"
  },
  "id": "腾讯云向量数据库.pdf"
```

```
}
```

入参描述

参数名	子参数	是否必选	参数含义
database_name	-	是	指定 Database 名称。
collection_name	-	是	指定 Collection 的名称。
file_name	-	否	指定文件名。若不指定，从 local_file_path 获取文件名。
local_file_path	-	是	本地上传文件路径。
metadata	-	否	文件的 Metadata 元数据信息，可自定义扩展字段。例如：author、tags 等。 • 上传文件时，可为创建 Collection 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。
splitter_process	append_title_to_chunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **False**：不追加。默认值为 False。 • **True**：将段落 Title 追加到切分后的段落。
	append_keywords_to_chunk	否	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **False**：不追加。 • **True**：将全文的 **keywords** 追加到切分后的段落。默认值为 True。
	chunk_splitter	否	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsing_process	parsing_type	否	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。

版权所有：腾讯云计算（北京）有限责任公司

			• **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
embedding_model	–	是	指定使用的 Embedding 模型的名称。您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，参见 Embedding 介绍 。取值如下所示： • **bge-large-zh-v1.5**：适用中文，1024维，推荐使用。 • **bge-base-zh-v1.5**：适用中文，768维。 • **bge-large-zh**：适用中文，1024维。 • **bge-base-zh**：适用中文，768维。 • **m3e-base**：适用中文，768维。 • **e5-large-v2**：适用英文，1024维。 • **text2vec-large-chinese**：适用中文，1024维。 • **multilingual-e5-base**：适用于多种语言类型，768维。 • **BAAI/bge-m3**：适用于多种语言类型，1024维。
field_mappings	–	是	指定文件字段在数据库中映射关系。 • **filename**：指定文件名映射的字段。例如："filename": "file_name"，其中，file_name 为用户自定义字段，以在数据库中存储文件名数据。 **注意：** **必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。 • **text**：指定文件内容文本映射的字段，存储知识点的原始文本内容。例如："text": "text"，其中，text 为用户自定义字段，以在数据库中存储文件的原始文本内容。 **说明：** 由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。

- **imageList**：指定图片信息存放的字段。例如："imageList": "image_list"，其中，image_list 为用户自定义的字段，以在数据库中存储图片 Key 列表。

  > **说明：**
  > **不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。

- **chunkNum**：指定数据库中存放文件切分语块序号的字段。例如："chunkNum": "chunk_num"，其中，chunk_num 为用户自定义字段，其值为唯一标识语块片段的整数。

  > **说明：**
  > 在创建 Collection 时若为该字段（如 chunk_num）创建索引，则查询文件内容时可通过 Filter 表达式指定语块范围，快速定位目标文件内容。

- **sectionNum**：指定数据库中用于存储文件段落序号的字段名。例如："sectionNum": "section_num"，其中，section_num 为用户自定义的字段，其值为唯一标识段落的整数。

  > **说明：**
  > 在创建 Collection 时若为该字段（如 section_num）创建索引，则查询文件内容时可通过 Filter 表达式指定段落范围内的语块，快速定位目标文件内容。

获取文件图片 URL

最近更新时间：2025-03-04 20:25:12

接口定义

get_image_url() 接口用于在使用 AI 套件上传 PDF 类型的文件后，获取图片 URL 地址列表。

> **说明：**
> 当在数据库的 Collection 中上传 PDF 文件时，系统会提取 PDF 中的图片信息，并将这些信息以 Key 列表的形式存储。在检索出的原始文本块中，图片的位置会以 {key} 的形式进行占位。通过调用接口 **get_image_url()**，可以获取到与这些 Key 一一对应的图片 URL 地址列表。利用这种映射关系，可以将原始 PDF 文档的内容恢复并以 HTML 格式进行展示。

```
def get_image_url(self,
                  database_name: str,
                  collection_name: str,
                  document_ids: List[str],
                  file_name: str) -> List[List[dict]]:
```

使用示例

```
res = client.get_image_url(
                    database_name='db-test',
                    collection_name='coll-file-test',
                    document_ids=["1340988975862403135"],
                    file_name='腾讯云向量数据库.pdf',
                    )
print(res)
```

入参描述

参数	是否必选	参数含义	配置方法
database_name	是	指定查询的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

collection_name	是	指定查询的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

document_ids	是	表示要查询的文档的所有ID。	• 每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。
			• 文件上传成功之后，可使用 query 查询文件的 ID 信息。
			`res = client.query(`
			`database_name='db-test',`
			`collection_name='coll-file-test',`
			`limit=100,`
			`)`

file_name	是	指定文件名称。	-

输出信息

```
[
  [
    {
      "documentId": "1340988975862403135",
      "imageName": "ea97227d-1b12-4d35-8b67-************.png",
      "imageUrl": "https://vecterdb************.cos.ap-guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-test-6/****************************************"
    },
    {
      "documentId": "1340988975862403135",
      "imageName": "a5daca6f-a27e-4b35-89f5-************.png",
      "imageUrl": "https://vecterdb************.cos.ap-guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-test-6/****************************************"
    }
  ]
```

```
]
```

参数名	参数含义
documentId	Document 的 ID 信息。
imageName	图片名。
imageUrl	图片 URL。

版权所有：腾讯云计算（北京）有限责任公司

DocumentSet 操作
上传文件

最近更新时间：2024-12-05 20:37:02

接口定义

load_and_split_text() 接口用于上传文件于 AI 类向量数据库。

```
def load_and_split_text(
    local_file_path: str,
    document_set_name: str | None = None,
    metadata: dict | None = None,
    splitter_process: SplitterProcess | None = None,
    timeout: float | None = None
) -> DocumentSet
```

> **注意：**
> - 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
> - 该接口当前不支持使用别名替换集合视图上传文件。

使用示例

> **说明：**
> 如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
import tcvectordb
from tcvectordb.model.ai_database import AIDatabase
from tcvectordb.model.collection_view import CollectionView
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.document_set import DocumentSet
from tcvectordb.model.enum import ReadConsistency
from tcvectordb.model.collection_view import SplitterProcess,
ParsingProcess

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
```

```
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# Specify the database name
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')
# 上传文件
# 1. local_file_path：指定当前文件在客户端的路径
# 2. document_set_name：配置文件存储于向量数据库的名称
# 3. metadata：定义文件 meta 信息字段，示例中指定了author 与 tags
res = coll_view.load_and_split_text(
            local_file_path="/data/home/../腾讯云向量数据库.pdf",
            document_set_name="腾讯云向量数据库.pdf",
            metadata={
                'author': 'Tencent',
                'tags': ['向量', 'Embedding', 'AI']
            },
            splitter_process=SplitterProcess(
                append_keywords_to_chunk=True,
                append_title_to_chunk=False,
                # chunk_splitter 参数以正则表达式的方式配置文档拆分方式，如下：
\n{2,} 代表以两个及以上的换行进行拆分，常用在QA对文件拆分中。
                # chunk_splitter="\n{2,}",
            ),
            parsing_process=ParsingProcess(
                parsing_type='VisionModelParsing'
            ),
    )
print(vars(res))
```

入参描述

> **说明：**
> 如果创建 CollectionView 和上传文件时同时配置了 **chunk_splitter** 参数，将使用上传文件时配置的文件拆分方式处理。

参数名	子参数	是否必选	参数含义
local_file_path	-	是	本地上传文件路径。

document_set_name	-	否	存储在向量数据库中的文件名。若不设置该参数，则默认使用 local_file_path 中的文件名。
metadata	-	否	文件的 Metadata 元数据信息，可自定义扩展字段。例如：author、tags 等。 • 上传文件时，可为创建 CollectionView 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。
splitter_process	append_title_to_chunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • False：不追加。默认值为 False。 • True：将段落 Title 追加到切分后的段落。
	append_keywords_to_chunk	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • False：不追加。 • True：将全文的 keywords 追加到切分后的段落。默认值为 True。
	chunk_splitter	否	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsing_process	parsing_type	否	指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

出参描述

输出信息，如下所示。

```
{'documentSetId': '11800467415********', 'documentSetName': '腾讯云向量数据库.pdf', 'documentSetInfo': {'indexedProgress': 0, 'indexedStatus': 'New'}}
```

参数名	子参数	参数含义

documnetSetId	-	文件 ID。
documnetSetName	-	文件名。
documentSetInfo	indexedProgress	文件被预处理、Embedding 向量化的进度。
	indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。

获取文件内容

最近更新时间：2025-05-12 18:23:12

接口定义

get_document_set() 该接口用于获取存储于 AI 类向量数据库的文件完整内容以及系统分配的文件 ID、关键字、文件大小、预处理进度与状态等信息。

```
def get_document_set(
    document_set_id: str | None = None,
    document_set_name: str | None = None,
    timeout: float | None = None
) -> (DocumentSet | None)
```

使用示例

> **注意：**
> 使用 **get_document_set()** 获取单个文档的 chunk 数量上限为1000，若超过此限制，接口将返回错误提示信息，并提示可使用 获取 Chunks 接口，结合 offset（起始位置）和 limit（每页数量）参数分批获取文档内容。

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

res = coll_view.get_document_set(document_set_name="腾讯云向量数据库.pdf")
print(vars(res))
```

入参描述

参数名	是否必选	参数含义	获取方式

document_set_id	否	文件上传在数据库之后，系统分配的文件 ID	先使用文件名获取文件 ID 之后，可使用文件 ID 查询文件内容
document_set_name	否	文件名	-

出参描述


```
{
  'documentSetId': '1190130763145412608',
  'documentSetName': '腾讯云向量数据库.pdf',
  'textPrefix': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似',
  'text': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。\n## 关键概念\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。\n### 什么是向量？\n向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。\n### 什么是非结构化数据？\n非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。\n### 什么是 AI 中的向量表示？\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。\n### 什么是向量检索？\n向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。\n## 为什么是腾讯云向量数据库？\n腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。 \n## 腾讯云向量数据库应用示例有哪些？\n腾讯云向量数据库可进行高性能向量存储和检索，主要适用于以下应用场景。\n- [大规模知识库]：企业的私域数据存储在向量数据库中可构建外部知识库，帮助企业更好地管理和利用自己的数据资源。\n- [推荐系统]：向量数据库会基于用户特征进行向量存储与检索，最终筛选用户可能感兴趣的物品推荐给用户。\n- [问答系统]：向量数据库会基于问题信息进行向量存储与检索，并返回最相关的问题与对应的答案。\n- [文本/图像检索]：向量数据库对输入的图像和文本信息进行向量存储与检索，会找到最匹配输入信息的文本或图像结果。\n## 腾讯云向量数据库支持哪些索引类型？\n索引是数据的组织单位。您必须先声明索引类型和相似性度量，然后才能搜索或查询向量数据。目前，腾讯云向量数据库支持如下类型。\n- FLAT
```

索引：向量会以浮点型的方式进行存储，不做任何压缩处理。搜索向量会遍历所有向量与目标向量进行比较。\n- HNSW 索引：全称为 Hierarchical Navigable Small World， 是基于图的索引，适合对搜索效率要求较高的场景。\n- IVF 系列：全称为 Inverted File，IVF 系列索引的核心思想是将高维空间划分为多个聚类，并为每个聚类构建一个倒排文件。适用于高维向量数据的快速检索。\n## 腾讯云向量数据库支持哪些相似度计算方法？\n在 VectorDB 中，相似度度量用于衡量向量之间的相似度。选择良好的距离度量有助于显著提高分类和聚类性能。根据输入数据形式，选择特定的相似性度量方法，获得数据库最佳性能。\n**相似性计算方法** | **方法说明** \n:-: | :-: \n内积（IP） | 全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 | \n欧式距离（L2） | 全称为 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 | \n余弦相似度（COSINE） | 余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度。所得值越大越与搜索值相似。 | \n## 腾讯云向量数据库是如何设计的？\n- **部署架构**：腾讯云向量数据库采用分布式部署架构，每个节点相互通信和协调，实现数据存储与检索。客户端请求通过 **Load Balancer** 分发到各节点上。\n- **逻辑架构**：实例是腾讯云中独立运行的数据库环境，是用户购买向量数据库服务的基本单位。腾讯云向量数据库数据存储的一个实例集群中包括 [Database]、[Collection]、[Document]三个逻辑层级。其中，一个实例可以包含很多个 Database，一个 Database 可以包含多个 Collection，一个 Collection 可以包含多个 Document。\n- **数据安全**：腾讯云向量数据库的多副本设计、多可用区分布节点、API 密钥认证，并运行于私有网络环境，通过安全组控制访问来源，CAM 账户授权等多方面保护向量数据的完整性和隐私。\n- **鉴权方式**：腾讯云向量数据库使用账号（account）和 API 密钥（api_key）的组合进行鉴权，以验证用户身份并授权其访问。\n- **连接方式**：腾讯云向量数据库支持通过 HTTP 协议进行数据写入和查询等操作。\n- **检索方法**：腾讯云向量数据库支持通过精确检索、相似度检索、混合检索的方法。\n- 精确查询：基于标量（指一个单独的数值，例如文本字段、数值字段或日期字段，区别于向量等多维数据结构）字段精确查找数据的方式。\n- 相似度检索：基于向量相似度计算的检索方式，通过计算向量之间的相似度来找到与查询向量最相似的文档。\n- 混合检索：基于标量字段和向量字段，搭配自定义的标量字段的 Filter 表达式进行检索的方式。\n## 如何快速体验向量数据库？\n腾讯云向量数据库目前是公测阶段。免费测试版实例每个账号仅限申领1个，高可用版与单机版实例免费试用时长1个月，到期后可 [提交工单](https://console.cloud.tencent.com/workorder/category) 进行续期；若一个月内未使用实例，平台将自动回收。\n**序号** | **步骤描述** | **具体操作** \n:-: | :-: | :-: \n1 | 申请腾讯云账号并认证。 | - 如需注册腾讯云账号：请单击 [注册腾讯云账号](https://cloud.tencent.com/register?s_url=https%3A%2F%2Fcloud.tencent.com%2F)。 | - 如需完成实名认证：请单击 [实名认证](https://console.cloud.tencent.com/developer)。 | \n2 | 了解向量数据库所支持的规格与类型。 | 预估数据规模，选择合适的类型与规格。| \n3 | 确定向量数据库所部署的地域。 | 当前支持的地域信息，请参见 [发布地域] | \n4 | 规划数据库实例的私有网络与安全组。 | 具体操作，请参见 [创建私有网络](https://cloud.tencent.com/document/product/215/36515)与 [创建安全组]，并同时设置安全组入站规则。 | \n5 | 购买实例。 | 具体操作，请参见 [新建数据库实例]。购买实例中，直接选择上一步已准备的私有网络与安全组。 | \n6 | 申请与腾讯云向量数据库在同一

```
地域同一个 VPC 内的 Linux 云服务器 CVM。 | 具体操作，请参见 [快速配置 Linux 云服务器](https://cloud.tencent.com/document/product/213/2936)。 | \n7 | 连接并操作向量数据库。 | [连接并写入数据库] ，本文使用 [API 接口] 从创建 DataBase 到插入数据、检索数据到最终删除数据，均给出了具体的使用示例。您可以简单并快速体验向量数据库。 | \n8 | 管理向量数据库实例 | 您可以体验通过控制台直接管理实例，查看实例状态或销毁实例。 | \n9 | 智能运维 | 您可以在控制台查看监控数据库实例的各项指标。目前仅支持对节点信息的监控，后续还会支持更丰富的监控项目。 | \n## 开发者工具\n**开发者工具** | **API** \n:-: | :-: \nHTTP API | [API 接口](https://cloud.tencent.com/document/product/1709/98666) | \nPython SDK | [Python SDK Demo](https://cloud.tencent.com/document/product/1709/96724) | \nJava SDK | [Java SDK Demo](https://cloud.tencent.com/document/product/1709/97768) | \n',
  'documentSetInfo': {
    'textLength': 5526,
    'byteLength': 12886,
    'indexedProgress': 100,
    'indexedStatus': 'Ready',
    'createTime': '2023-12-29 11:14:45',
    'lastUpdateTime': '2023-12-29 11:14:47',
    'keywords': '向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似'
  },
  'splitterPreprocess': {
    'appendTitleToChunk': True,
    'appendKeywordsToChunk': True
  },
  'parsingProcess': {
    'parsingType': 'VisionModelParsing'
  },
  'author': 'Tencent',
  'tags': [
    '向量',
    'Embedding',
    'AI'
  ]
}
```

参数名	子参数	参数含义
documnetSetId	-	文件 ID。
documnetSetName	-	文件名。

textPrefix	-	文件内容前200个字符。
text	-	文件完整内容。
documentSetInfo	textLength	文件的字符数。
	byteLength	文件的字节数。
	indexedProgress	文件被预处理、Embedding 向量化的进度。
	indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。
	createTime	文件创建时间。
	lastUpdateTime	文件最后更新时间。
	keywords	文件关键字。
splitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • false：不追加。 • true：将段落 Title 追加到切分后的段落。
	appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
	chunk_splitter	以正则表达式的方式配置文档拆分方式，如下： `\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsingProcess	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类

		型，无需配置该参数，默认使用 AlgorithmParsing 解析。
author、tags	-	自定义扩展的文件 Metadata 信息的标量字段。

版权所有：腾讯云计算（北京）有限责任公司

获取 Chunks

最近更新时间：2025-05-26 11:14:52

接口定义

get_chunks() 接口用于获取文件切分后的语块。

> **说明：**
> Chunk 指语块，较长文本在处理时会切分为多个语块，以便于向量化和更高效地检索，多个 Chunk 组成一个 DocumentSet。

```
def get_chunks(
    limit: int | None = None,
    offset: int | None = None,
    timeout: float | None = None
) -> List[Chunk]
```

使用示例

> **说明：**
> - 支持指定具体的文件名获取文件切分后的语块。
> - 支持指定具体的 DocumentSet ID 获取文件切分后的语块。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

chunk_list = coll_view.get_chunks(document_set_name='腾讯云向量数据库.pdf',
limit=3)
```

版权所有：腾讯云计算（北京）有限责任公司

```
for chunks in chunk_list:
    print(vars(chunks))
```

入参描述

参数名	是否必选	参数含义
document_set_id	否	文件上传在数据库之后，系统分配的文件 ID。
document_set_name	否	文件名。
limit	否	每页返回的 Chunks 数量。 • 数据类型：uint 64。 • 默认值：10。 • 取值范围：[1,1000]。
offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 limit 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。

出参描述

```
{'startPos': 0, 'endPos': 122, 'text': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n'}
{'startPos': 122, 'endPos': 313, 'text': '## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。\n'}
{'startPos': 313, 'endPos': 441, 'text': '## 关键概念\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。\n'}
```

参数名	参数含义
text	获取的 Chunks 内容。

版权所有：腾讯云计算（北京）有限责任公司

startPos	每个 Chunks 在文件中偏移的起始位置。
endPos	每个 Chunks 在文件中偏移的结束位置。

版权所有：腾讯云计算（北京）有限责任公司

精确查询文件信息

最近更新时间：2024-12-05 20:37:02

接口定义

query() 用于精确查找与查询条件完全匹配的文件，可获取文件长度、向量化的进度与状态等，不包括文件内容。

```
def query(
    document_set_id: List | None = None,
    document_set_name: List[str] | None = None,
    filter: Filter | None = None,
    limit: int | None = None,
    offset: int | None = None,
    output_fields: List[str] | None = None,
    timeout: float | None = None
) -> List[DocumentSet]
```

使用示例

说明：

- 支持指定具体的文件名查找文件，或搭配文件 Metadata 信息对应字段的 Filter 表达式查询文件信息。
- 支持指定具体的 DocumentSet ID 查找文件，或搭配文件 Metadata 信息对应字段的 Filter 表达式查询文件信息。
- 支持指定查询起始位置 offset 和返回数量 limit，查找指定范围的文件信息。
- 支持根据文件 Metadata 信息对应字段 Filter 表达式，过滤需查找的文件。

使用文件名搭配 Filter 查询文件

根据存储于向量数据库的文件名，搭配标量字段 author 与 tags 的 Filter 表达式一并过滤文件。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va********************************',
```

```
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

doc_list = coll_view.query(
        document_set_name=["腾讯云向量数据库.pdf"],
        limit = 2,
        filter=Filter(Filter.In("author",
["Tencent","tencent"])).And(Filter.Include("tags",
["AI","Embedding"])),
        output_fields=['textPrefix','author', 'tags']
    )

for doc in doc_list:
    print(vars(doc))
```

查询结果，如下所示。

```
{
  'documentSetId': '11801071477********',
  'documentSetName': '腾讯云向量数据库.md',
  'textPrefix': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似',
  'author': 'Tencent',
  'tags': [
    '向量',
    'Embedding',
    'AI'
  ]
}
```

使用范围查询文件

文件上传于向量数据库之后，可以使用 limit 与 offset 参数，设定查询的范围来查询文件信息。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')
# 指定 limit 与 offset ，查询文件信息，返回从 offset 开始的 limit 条数据
doc_list = coll_view.query(limit=10, offset=0)
for doc in doc_list:
          print(vars(doc))
```

查询结果，如下所示。

```
{
  'documentSetId': '1190130763145412608',
  'documentSetName': '腾讯云向量数据库.md',
  'textPrefix': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似',
  'documentSetInfo': {
    'textLength': 5526,
    'byteLength': 12886,
    'indexedProgress': 100,
    'indexedStatus': 'Ready',
    'createTime': '2023-12-29 11:14:45',
    'lastUpdateTime': '2023-12-29 11:14:47',
    'keywords': '向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似'
  },
  'splitterPreprocess': {
    'appendTitleToChunk': True,
    'appendKeywordsToChunk': True
  },
  'parsingProcess': {
      'parsingType': 'VisionModelParsing'
```

```
  },
  'author': 'Tencent',
  'tags': [
    '向量',
    'Embedding',
    'AI'
  ]
}
```

使用文件 ID 查询文件

文件上传于向量数据库之后，系统会自动分配文件 ID，获取文件 ID 信息之后可通过文件 ID 批量查询文件信息。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')
# 指定 limit 与 offset ，查询文件信息，返回从 offset 开始的 limit 条数据
doc_list = coll_view.query(
        document_set_id=["11793516237********"],
        limit = 2,
        filter=Filter("author=\"Tencent\""),
        output_fields=['textPrefix','author', 'keywords']
    )
for doc in doc_list:
          print(vars(doc))
```

查询结果，如下所示。

```
{
  'documentSetId': '11793516237********',
```

```
  'documentSetName': '腾讯云向量数据库.md',
  'textPrefix': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似',
  'documentSetInfo': {
    'keywords': '向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似'
  },
  'author': 'Tencent'
}
```

入参描述

参数名	是否必选	配置方法及要求
document_set_name	否	表示要查询的文档的名称，支持批量查询，数组元素范围[1,20]。
document_set_id	否	表示要查询的文档的所有 ID，支持批量查询，数组元素范围[1,20]。

filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中：
		• <field_name>：表示要过滤的字段名。
		• <operator>：表示要使用的运算符。
		  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
		  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。
		  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
		• <value>：表示要匹配的值。
		示例：`Filter('author="jerry"').And('page>20')`。

limit	是	每页返回的 DocumentSet 数量。 • 数据类型：uint 64。

		• 默认值：10。 • 取值范围：[1,16384]。 **⚠ 注意：** • 若不配置任何查询条件，即 `doc_list = coll_view.query()`，则默认返回 10 个 DocumentSet。 • 若查询条件仅配置 Filter 表达式，不配置 limit，则默认返回 10 条 DocumentSet。 • 若查询条件仅设置 document_set_name 或 document_set_id，则可不配置 limit 参数，默认返回 10 条数据。
offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 limit 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。
outputFields	否	以数组形式配置需返回的字段。

出参描述


```
{
  'documentSetId': '1190130763145412608',
  'documentSetName': '腾讯云向量数据库.pdf',
  'textPrefix': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似',
  'documentSetInfo': {
    'textLength': 5526,
    'byteLength': 12886,
    'indexedProgress': 100,
    'indexedStatus': 'Ready',
    'createTime': '2023-12-29 11:14:45',
    'lastUpdateTime': '2023-12-29 11:14:47',
    'keywords': '向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似'
```


版权所有：腾讯云计算（北京）有限责任公司

```
  },
  'splitterPreprocess': {
    'appendTitleToChunk': True,
    'appendKeywordsToChunk': True
  },
  'parsingProcess': {
      'parsingType': 'VisionModelParsing'
  },
  'author': 'Tencent',
  'tags': [
    '向量',
    'Embedding',
    'AI'
  ]
}
```

返回参数

参数名	子参数	参数含义
documnetSetId	-	文件 ID。
documnetSetName	-	文件名。
textPrefix	-	文件内容前 200个字符。
documentSetInfo	textLength	文件的字符数。
	byteLength	文件的字节数。
	indexedProgress	文件被预处理、Embedding 向量化的进度。
	indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。
	indexedErrorMsg	文件解析、写入错误描述信息。 **说明：**

		当 IndexedStatus 为 Failure 时，返回 indexedErrorMsg 信息。
	createTime	文件创建时间。
	lastUpdateTime	文件最后更新时间。
	keywords	文件关键字。
splitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • false：不追加。 • true：将段落 Title 追加到切分后的段落。
	appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
	chunk_splitter	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsingProcess	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析。 • AlgorithmParsing：文件依据算法解析。
author	-	自定义的文件 Metadata 信息字段。

文件内容检索

最近更新时间：2025-07-01 10:19:42

功能介绍

search() 接口用于在指定的文件内，查找与给定文本信息相似的 Top K 条文本信息。

- **支持指定文件名称检索最相似的文本信息。**
- **支持文件名搭配文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。**
- **支持仅使用文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。**

```
def search(
    content: str,
    document_set_name: List[str] | None = None,
    expand_chunk: list | None = None,
    rerank: Rerank | None = None,
    filter: Filter | None = None,
    limit: int | None = None,
    timeout: float | None = None
) -> List[SearchResult]
```

使用示例

根据文件名搭配 Filter 检索相似数据

如下示例，在文件 `腾讯云向量数据库.pdf` 中，检索与 `什么是向量数据库` 相似的文本信息，并使用标量字段 author 与 tags 的 Filter 表达式一并过滤文件。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')
```

```
doc_list = coll_view.search(
            content='什么是向量数据库',
            document_set_name = ['腾讯云向量数据库.pdf'],
            expand_chunk=[1, 0],
            filter=Filter(Filter.In("author",
["Tencent","tencent"])).And(Filter.Include("tags",
["AI","Embedding"])),
            limit=3
        )

for doc in doc_list:
 print(vars(doc))
```

根据 Filter 表达式检索相似数据

如下示例，通过文件 meta 信息的标量字段 author 的 Filter 表达式，检索与 `什么是向量数据库` 相似的文本信息。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

doc_list = coll_view.search(
            content='什么是向量数据库',
            expand_chunk=[1, 0],
            filter=Filter("author=\"Tencent\""),
            limit=3
        )

for doc in doc_list:
```

```
print(vars(doc))
```

入参描述

参数名称	是否必选	参数含义及配置方法
content	是	以 String 类型输入检索的文本信息
expand_chunk	否	• 以数组形式配置检索的目标信息所需向前扩展的段落数量以及向后扩展的段落数。例如，输入[2,3]，指所检索到的 Chunk 返回时，同时返回其之前的 2个段落与之后的3个段落。 • 段落指文件在上传存储时，自动向量化拆分的段落。 • 默认值：[1,1]。
document_set_name	否	表示要查询的文档的名称，支持批量查询，数组元素范围[1,10]。
filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ◦ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ◦ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ◦ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`
limit	是	指定返回最相似的 Top K 的 K 的值。

出参描述

如下为检索到的相似数据，返回最相似的 Top 3 条数据。

```
[
```

```
  {
    "score": 0.855039119720459,
    "data": {
      "text": "## 腾讯云向量数据库是什么？\n\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级QPS及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP服务、计算机视觉、智能客服等AI领域。\n\n",
      "startPos": 1154,
      "endPos": 1342,
      "pre": [
        "## 向量数据库\n\n## 产品简介\n\n## 腾讯云向量数据库\n\n最近更新时间：2023-07-2714:46:31\n\n本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（TencentCloudVectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n\n"
      ],
      "next": [],
      "paragraphTitle": "腾讯云向量数据库是什么？",
      "allParentParagraphTitles": [
        "腾讯云向量数据库是什么？"
      ]
    },
    "documentSet": {
      "documentSetId": "13879853046025502 72",
      "documentSetName": "腾讯云向量数据库.pdf",
      "author": "Tencent",
      "tags": [
        "向量",
        "Embedding",
        "AI"
      ]
    }
  },
  {
    "score": 0.8471382260322571,
    "data": {
      "text": "## 什么是向量检索？\n\n向量检索是将向量与数据库进行比较以查找与查询向量最相似的向量的过程。相似的向量通常具有相近的原始数据，通过向量检索可以挖掘出原始非结构化数据之间的联系。\n\n",
      "startPos": 1708,
      "endPos": 1799,
      "pre": [
```

```
        "## 什么是AI中的向量表示？\n\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。具体上线时间，请关注产品动态。\n\n"
      ],
      "next": [],
      "paragraphTitle": "什么是向量检索？",
      "allParentParagraphTitles": [
        "什么是向量检索？"
      ]
    },
    "documentSet": {
      "documentSetId": "1387985304602550272",
      "documentSetName": "腾讯云向量数据库.pdf",
      "author": "Tencent",
      "tags": [
        "向量",
        "Embedding",
        "AI"
      ]
    }
  },
  {
    "score": 0.846755862236023,
    "data": {
      "text": "## 向量数据库\n\n## 产品优势\n\n最近更新时间：2023-08-2115:36:41\n\n腾讯云向量数据库（TencentCloudVectorDB）作为一种专门存储和检索向量数据的服务提供给用户，在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。\n\n",
      "startPos": 4991,
      "endPos": 5130,
      "pre": [
        "## 开发者工具\n\n![vdb-image](https://vecterdb-backup-gz-1318917394.cos.ap-guangzhou.myqcloud.com/embedding_file/vdb-********/db-test-ai/coll-ai-files-**********%3Aimages/**********.png?q-sign-algorithm=sha1&q-ak=**********)\n\n腾讯云向量数据库支持丰富的API接口，以促进DevOps。具体操作，请参见API文档。\n\n\n\n![vdb-image](https://vecterdb-backup-gz-**********.png?q-sign-algorithm=sha1&q-ak=**********)\n\n"
      ],
```

```
      "next": [],
      "paragraphTitle": "## 向量数据库\n\n## 产品优势\n",
      "allParentParagraphTitles": [
        "产品优势"
      ]
    },
    "documentSet": {
      "documentSetId": "1387985304602550272",
      "documentSetName": "腾讯云向量数据库.pdf",
      "author": "Tencent",
      "tags": [
        "向量",
        "Embedding",
        "AI"
      ]
    }
  }
]
```


参数名	子参数	参数含义
score	-	表示查询向量与检索结果向量之间的相似性计算分数。
data	text	检索的结果。
	endPos	检索结果在文件中偏移的结束位置。
	startPos	检索结果在文件中偏移的起始位置。
	next	根据检索时，设置的参数 expand_chunk，返回检索结果向后扩展的段落。
	pre	根据检索时，设置的参数 expand_chunk，返回检索结果向前扩展的段落。
	paragraphTitle	检索到的 chunk 所属的段落标题。
	allParentParagraphTitles	检索到的 chunk 所属的所有层级的段落标题。
documentSet	documentSetId	文件 ID。
	documentSetName	文件名。

	other_scalar_field	自定义的文件 Metadata 信息的标量字段。例如：author、page 等。 **说明：** 显示创建 CollectionView 时设置为 Filter 索引的字段，同时显示上传文件时或使用 update 新增的字段，但新增的字段不会构建索引。

删除指定文件

最近更新时间：2024-12-05 20:37:02

功能介绍

delete() 接口用于删除存储于 CollectionView 文件。

```
def delete(
    document_set_id: str | List[str] = None,
    document_set_name: str | List[str] = None,
    filter: Filter = None,
    timeout: float = None
) -> Any
```

使用示例

说明：

- **支持批量删除，文件 ID 或文件名数组元素数量最大为20。**
- **支持使用 Filter 表达式过滤所需删除的所有文件。**

根据文件名过滤需删除的文件

如下示例，删除文件名为 `腾讯云向量数据库.pdf` ，并满足 Filter 表达式的文件，二者取并集。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

res = coll_view.delete(
            document_set_name = ['腾讯云向量数据库.pdf'],
            filter=Filter("author=\"Tencent\"")
```

```
        )
print(res)
```

根据文件 ID 过滤需删除的文件

如下示例，删除指定文件 ID，并满足 Filter 表达式的文件，二者取并集。

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

res = coll_view.delete(
            document_set_id=["11793516237********"],
        )
print(res)
```

根据 Filter 表达式过滤需删除的文件

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')
```

```
res = coll_view.delete(
            filter=Filter("author=\"Tencent\"")
        )
print(res)
```

入参描述

参数名称	参数含义	是否必选	配置方法及要求
document_set_name	指定需删除的文件名。	否	支持批量删除，数据元素最大值为20。 **说明：** 同时配置 document_set_name 与 filter 参数，删除数据将会取二者的交集。
document_set_id	指定需删除的文件ID。	否	支持批量删除，数据元素最大值为20。 **说明：** 同时配置 document_set_id 与 filter 参数，删除数据将会取二者的交集。
filter	配置Filter表达式过滤需删除的文件	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ string：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ uint64：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ array：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例

			如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`

出参描述

```
{
    'code': 0,
    'msg': 'Operation success',
    'affectedCount': 1
 }
```

参数名	参数含义
affectedCount	删除的文档数量。

更新文件

最近更新时间：2024-12-17 11:51:02

功能介绍

update() 接口用于对 DocumentSet 的部分字段进行更新。

```
def update(
    data: Document,
    document_set_id: str | List[str] = None,
    document_set_name: str | List[str] = None,
    filter: Filter = None,
    timeout: float = None
) -> Any
```

使用示例

⚠ 注意：

- **支持新增字段，支持更改部分字段，但不能变更系统分配的 DocumentSet ID 字段，不要求事务完整性。**
- **不能变更已上传的文件内容，支持通过主键（DocumentSet ID）或文件名，搭配 Filter 表达式过滤需更新的文件。**
- **新增字段，在创建 CollectionView 时没有为这些字段设置索引，那么新增这些字段时，系统不会自动为其创建索引。**

根据文件名过滤需更新的文件

如下示例，修改文件名为 `腾讯云向量数据库.pdf` ，并满足 author 字段为 Tencent 的文件，新增字段 tag 更新字段 author 的值。

```
import tcvectordb
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
```

```
# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

res = coll_view.update(
            data=Document(author ='tencent', tag = '向量'),
            document_set_name = ['腾讯云向量数据库.pdf'],
            filter=Filter("author=\"Tencent\"")
        )
print(res)
```

根据文件 ID 过滤需更新的文件

如下示例，修改指定文件 ID，并满足 Filter 表达式的文件的标量字段 author 为 Tencent 的文件，并修改 author 字段为 tencent。

```
import tcvectordb
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

res = coll_view.update(
            data=Document(author ='tencent'),
            document_set_id=["11793516237********"],
            filter=Filter("author=\"Tencent\"")
        )
print(res)
```

根据 Filter 表达式过滤需更新的文件

如下示例，修改满足 author 的 Filter 表达式的文件的字段 author 为 Tencent，并修改 author 字段为 tencent，新增字段 tag。

```
import tcvectordb
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

# 指定文件上传所属的文件
db = client.database('db-test-ai')
coll_view = db.collection_view('coll-ai-files')

res = coll_view.update(
            data=Document(author ='tencent', tag = '向量'),
            filter=Filter("author=\"Tencent\"")
        )
print(res)
```

入参描述

参数名	参数含义	子参数	是否必选	配置方法及要求
document_set_name	指定需更新的文件名。	-	否	支持批量更新，数据元素最大值为20。 **说明：** 同时配置 document_set_name 与 filter 参数，更新数据将会取二者的交集。
document_set_id	指定需更新的文件ID。	-	否	支持批量更新，数据元素最大值为20。 **说明：**

				同时配置 document_set_id 与 filter 参数，更新数据将会取二者的交集。
filter	配置 Filter 表达式过滤需更新的文件		否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ string：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ uint64：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ array：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`
data	设置需更新的字段	old_field	否	当前已存在的字段，更新字段对应的数据。 • 类型：string。 • 字符长度要求：[1,256]。
		new_field	否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。

出参描述

```
{
    'code': 0,
    'msg': 'Operation success, requestId:
16f78e1d59149c1760bafd**********',
    'affectedCount': 1
 }
```

参数名	参数含义
affectedCount	更新的文档数量。如果该参数返回的值为 0，说明更新无效。

Index 操作
重建索引

最近更新时间：2025-07-01 10:19:42

接口定义

rebuild_index() 接口用于重建 Base 类 Collection 的所有索引，清除无用的索引数据，修复损坏的索引数据，优化索引结构，改善性能。

```
def rebuild_index(
    self,
    drop_before_rebuild: bool = False,
    throttle: int = 0,
    timeout: Optional[float] = None
)
```

说明：

rebuild_index 接口用于重建指定集合的向量索引，主要适用于以下两种场景：

- 首次使用 IVF 系列索引时，需要通过 rebuild_index 接口初始化索引结构。具体信息，请参见 IVF 系列索引应用指南 。
- 在离线批量数据导入场景中，建议先执行数据写入操作（通过 upsert 接口设置 build_index = False 暂停索引构建），待数据完整导入后，再调用rebuild_index 接口集中创建索引，以此减少写入过程中的索引维护开销，提升整体导入效率。

警告：

Rebuild 操作需占用额外内存且可能影响服务，执行前请评估业务场景并选择重建方式。

- 保留原索引、同时构建新索引（即设置 dropBeforeRebuild=False）：期间集合读取正常，写入被阻止，构建新索引后，替换原索引，构建过程中，旧索引和新索引都会占用内存，如果内存足够可采用该方式，建议根据实际业务高低峰选择停写时段执行。
- 先删除原索引、再构建新索引（即设置 dropBeforeRebuild=True）：期间集合读写会受到影响，先删除旧索引，再构建新索引。如果内存严重不足时，则需采用该方案。建议根据业务需求选择可中断服务的时段处理。

使用实例

```
import tcvectordb
```

```
# rebuild vector
client.rebuild_index(
    database_name=db_name,
    collection_name=coll_name,
    drop_before_rebuild=True,
    throttle=1,
    field_name='vector'
)
print(vars(vdb_client.collection(database_name=db_name,
collection_name=coll_name)))
```

入参描述

参数名	是否必选	参数含义	配置方法及要求
database_name	是	配置需重建索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection_name	是	指定需重建索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
field_name	否	指定重建索引的字段。	• vector：重建稠密向量索引。默认值为 vector。 • sparse_vector：重建稀疏向量索引。 **说明：** 同一集合，不支持并发执行 rebuild。如果集合中既有稠密向量又有稀疏向量，则需要分别进行重建。
drop_before_rebuild	否	标识在重建索引时，是否需先删除旧索引再重建新索引。 **警告：** 重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在	取值如下所示： • True：重建之前，先删除旧索引在重建索引。 **警告：** 内存资源不足时，可配置为 true，先删除旧索引，在新

		重建索引之前，您需根据实际资源情况选择是否需先删除旧索引再重建，避免引起内存占满而阻塞业务正常运行。	索引构建过程中，无法正常读写。 • False：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为 False。 **警告：** 内存资源足够的情况下，可配置为 false，可不删除旧索引。在新索引构建过程中，可读数据，禁止写入数据。
throttle	否	标识是否限制构建索引的单节点 CPU 核数。 **说明：** 重建索引会消耗 CPU 资源，为防止资源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为1。	取值如下所示： • 0：不限制 CPU 核数。在模型训练期间，会消耗大量的 CPU 资源。重建索引任务将会尽快执行，但可能会对其他集合的读写操作产生影响。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数,可避免构建索引期间对其他集合产生影响，但任务执行较慢。

相关说明

使用 describe_collection() 可查看指定集合的索引状态。返回参数 **indexStatus** 中的 **status** 标识当前 Collection 重建索引的状态，**startTime** 显示重建索引开始的时间。

- **initial**：表示首次创建索引，尚未 Rebuild 构建索引。
- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。
- **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。
- **failed**: 重建索引失败，可能会影响集合读写操作。

注意：

training data 与 **building index** 状态期间不可写入数据。若重建索引之前先删除旧索引，则集合不可进行相似性查询，只能进行精确查询。

新增索引

最近更新时间：2025-04-29 16:00:02

接口定义

add_index() 接口用于在已有集合上，新增标量字段索引。

```
def add_index(self,
              database_name: str,
              collection_name: str,
              indexes: List[FilterIndex],
              build_existed_data: bool = True,
              timeout: Optional[float] = None) -> dict:
```

说明：

- **如果集合开启了动态 schema（即创建集合时配置了 FilterIndexConfig 参数），不支持创建标量索引。**
- **当前不支持创建 json 类型的标量索引。**

使用实例

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import FieldType, IndexType
from tcvectordb.model.index import Index, FilterIndex

res = client.add_index(database_name='db-test',
                       collection_name='book-vector',
                       indexes=[
                           FilterIndex(name='bookInfo',
field_type=FieldType.String, index_type=IndexType.FILTER),
                           FilterIndex(name='page',
field_type=FieldType.Uint64, index_type=IndexType.FILTER),
                       ],
                       build_existed_data=True
                   )
print(res)
```

请求参数

参数	子参数	是否必选	参数含义	配置方法及要求
database_name	-	是	配置需重建索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection_name	-	是	指定需重建索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
indexes	name	是	指定新增标量索引的字段。	用户自定义字段名。命名规则为1-128字符，只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。
	field_type		指定字段数据类型。	指定自定义字段的数据类型。取值如下： • String：字符型。 • Uint64：指无符号整数（unsigned integer）。 • Array：数组类型，数组元素为 string。
	index_type		指定索引类型。	仅支持设置为 FILTER。
build_existed_data	-	是	用于指定新增的索引字段是否写入过数据。	• True：默认为 True，新增字段已写入过数据，需扫描历史数据并构建索引。 • False：新增索引字段为新增字段，无需扫描历史数据，可以快速创建索引。

删除索引

最近更新时间：2025-04-18 17:58:12

接口定义

drop_index() 接口用于在已有集合上，删除标量字段索引。

```
def drop_index(self,
               database_name: str,
               collection_name: str,
               field_names: List[str],
               timeout: Optional[float] = None) -> dict:
```

注意：

- 仅支持删除标量索引，不支持删除向量、稀疏向量索引。
- 如果集合开启了动态schema（即创建集合时配置了 FilterIndexConfig 参数），不支持删除标量索引。

使用实例

```
import tcvectordb
from tcvectordb.model.document import Filter
from tcvectordb.model.enum import FieldType, IndexType
from tcvectordb.model.index import Index, FilterIndex

res = client.drop_index(database_name='db-test-0314',
                        collection_name='book-vector-0314',
                        field_names=['bookInfo', 'page'],
                    )
print(res)
```

请求参数

参数	是否必选	参数含义	配置方法及要求
database_name	是	指定数据库。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。

			• 长度要求：[1,128]。
collection_name	是	指定集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
field_names	是	用于指定删除的标量索引的字段名。	以数组形式，列出需删除的字段名。

修改向量索引

最近更新时间：2025-07-01 10:19:42

接口定义

modify_vector_index() 接口用于在不重建整个索引的前提下，调整已创建向量索引的配置参数，包括：数据精度类型、距离度量方式、性能优化参数等。

说明：

- **仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。**
- **索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。**
- **HNSW 索引支持重新指定参数 M 和 efConstruction，IVF 索引支持重新指定 nlist（IVF_PQ 支持重新指定 M 和 nlist）。**
- **仅 HNSW 索引，支持修改字段数据类型，例如：从vector（标准单精度浮点数向量）可修改为 float16_vector（半精度浮点数向量）。**
- **支持重新指定向量相似度计算方法。**

```
def modify_vector_index(self,
                        database_name: str,
                        collection_name: str,
                        vector_indexes: List[VectorIndex],
                        rebuild_rules: Optional[dict] = None,
                        timeout: Optional[float] = None) -> dict:
```

使用示例

```
import tcvectordb
from tcvectordb.model.enum import MetricType
from tcvectordb.model.index import Index, VectorIndex

#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
#设置需更新的字段，或增加新的字段
res = client.modify_vector_index(
    database_name='db-test',
```

```
    collection_name='book-vector',
    vector_indexes=[
        VectorIndex(
            index_type=IndexType.HNSW,
            field_type=FieldType.Float16Vector,
            metric_type=MetricType.COSINE,
            params={
                "M": 8,
                "efConstruction": 100
            },
        ),
    ],
    rebuild_rules={
        "dropBeforeRebuild": True,
        "throttle": 1,
    }
)
print(res)
```

入参描述

参数	子参数	是否必选	参数含义	配置方法及要求
database_name	-	是	配置需重建索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection_name	-	是	指定需 Collection 的重建索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
vector_indexes	name	否	稠密向量数据字段名。	固定为 vector。 **说明：** 仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。

	field_type	否	稠密向量数据字段类型。	仅 HNSW 索引可重新指定向量索引的数据类型。默认值与当前类型一致，具体取值如下所示。 • **Vector**：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位），适用于需最高精度的计算任务或对误差敏感的向量检索。 • **Float16Vector**：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位 的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 bge-large 等模型生成的嵌入向量。 • **BFloat16Vector**：bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，适用于向量数值范围较大（更多bit位表达整数）的场景。
	index_type	否	指定调整后的目标索引类型。如果无需调整索引类型，该参数可不填。	默认与原索引类型一致，取值如下所示。 **说明：** • 索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。 • 切换索引类型时，请同步修改 params 参数，与目标索引类型的参数保持一致。 • FLAT 和 IVF 索引不支持 float16_vector 或 bfloat16_vector，当从 HNSW 索引类型切换到 FLAT 或 IVF 类型时，如果字段类型（fieldType）为 float16_vector 或 bfloat16_vector，则必须同步将字段类型修改为 vector 类型。

				• 索引切换时需要注意提前预估内存占用，尤其是在 IVF 系列索引切换成 FLAT/HNSW 时，内存差异可能较大。 • **FLAT**：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。 • **HNSW**：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区间数据规模。 • **IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16**：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。
	metric_type	否	重新指定相似度计算方法。	指定向量之间距离度量的算法。取值如下： • **L2**：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • **IP**：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 • **COSINE**：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度，所得值越大越与搜索值相似。

	params	否	重新指定向量索引参数。	• 索引类型 **indexType** 为 HNSW ，需配置如下参数。
				  ○ **M**：每个节点在检索构图中可以连接多少个邻居节点。
				  ○ 取值类型：uint64。
				  ○ 取值范围：[4,64]。一般可设置为16。
				  ○ **efConstruction**：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。

				○ 取值类型：uint64。 ○ 取值范围：[8,512]。一般可设置为200。 • 索引类型 indexType 为 IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16需配置如下参数。 ○ nlist：指索引中的聚类中心数量。 取值类型：uint64。取值范围：[1,65536]。 ○ M：指乘积量化中原始数据被拆分的子向量的数量。该参数仅 IVF_PQ 索引类型需配置。更多信息，请参见 索引与计算 。 ○ 取值要求：原始数据的向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。 ○ 取值范围：[1,向量维度]。
rebuild_rules	drop_before_rebuild	否	标识在重建索引时，是否需先删除旧索引再重建新索引。 **警告：** 重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在重建索引之前，您需根据实际资源情况选择是否需先删除旧索引再重建，避免引起内存占满而阻塞业务正常运行。	取值如下所示： • True：重建之前，先删除旧索引再重建索引。 **警告：** 内存资源不足时，可配置为true，先删除旧索引，在新索引构建过程中，无法正常读写。 • False：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为False。 **警告：** 内存资源足够的情况下，可配置为 false，可不删除旧索引。在新索引构建过程中，可读数据，禁止写入数据。

	throttle	否	标识是否限制构建索引的单节点 CPU 核数。 **说明：** 重建索引会消耗 CPU 资源，为防止资源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为 1。	取值如下所示： • **0**：不限制 CPU 核数。在模型训练期间，会消耗大量的 CPU 资源。重建索引任务将会尽快执行，但可能会对其他集合的读写操作产生影响。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数,可避免构建索引期间对其他集合产生影响，但任务执行较慢。

出参描述

```
{
  "code": 0,
  "msg": "Start rebuilding. You can use the '/collection/describe' API
to follow the progress of rebuilding."
}
```

参数名	参数含义
msg	提示启动 rebuild 重建索引，可使用 /collection/describe 接口返回的参数 indexStatus 查看集合索引状态。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。

User 管理

新建用户

最近更新时间：2025-01-21 19:52:12

接口定义

create_user() 接口用于创建自定义用户。

```
def create_user(self,
                user: str,
                password: str) -> dict:
```

使用示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# 创建用户
res = client.create_user(user='app_user', password='********')
print(res)
```

入参描述

参数名	参数含义	数据类型	是否必填	默认值	参数配置
user	自定义用户名	string	是	-	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。

password	设置密码	string	是	-	• 密码长度要求：8~128字符。
					• 密码设置支持如下类型的任意组合。
					  ○ 字母：大写 A~Z、小写 a~z。
					  ○ 数字：0~9。

					○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

输出信息

```
{
  "code": 0,
  "msg": "Operation success"
}
```

为用户授权

最近更新时间：2025-01-21 19:52:12

接口定义

grant_to_user() 接口用于对用户授予指定资源的访问权限。

```
def grant_to_user(self,
                  user: str,
                  privileges: Union[dict, List[dict]]) -> dict:
```

使用示例

如下示例，给用户 app_user 授予读取 db-0 下所有集合数据的权限，并具有数据库 db1 下 coll1 集合的读写权限。

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
res = client.grant_to_user(
        user='app_user',
        privileges=[
        {
            "resource": "db0.*",
            "actions": ["read"]
        },
        {
            "resource": "db1.coll1",
            "actions": ["readWrite"]
        }
])
print(res)
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置

user	需授权的用户名	-	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。
privileges	对用户授予的权限列表，可以配置多条权限	resource	string	是	指定用户可操作的资源对象，分为如下三个层级。 • Global：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"`。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有Collection。配置方式：`"resource": "db0.*"`，其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。 **说明：** resource 只能指向已存在的资源；若指定的资源不存在，则触发错误提示信息。
		actions	array	是	配置资源对象允许操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

输出信息

```
{
  "code": 0,
  "msg": "Operation success"
}
```

版权所有：腾讯云计算（北京）有限责任公司

撤销用户权限

最近更新时间：2025-01-21 19:52:12

接口定义

revoke_from_user() 接口用于撤销用户已拥有的权限。

```
def revoke_from_user(self,
                     user: str,
                     privileges: Union[dict, List[dict]]) -> dict:
```

说明：

- root 用户为数据库的默认用户，享有最高权限级别，无法被任何账号（包括其自身）撤销。
- 撤销权限时，权限需要和授予时完全对应，包括资源对象和允许的操作，都需要保持一致。
  - 只能针对原本层级回收权限，不能单独回收子对象的权限。例如，用户拥有 *.* 下的权限，那么不能单独回收 db.* 或者 db.coll 的权限。
  - 只能回收授予时指定的 action。例如，用户拥有某个资源对象的 readWrite 权限，那么回收时不能单独回收其 read 权限。
- **授权后，如果资源对象被删除，权限不会被撤销。后续如果该资源对象被重新创建，用户依然拥有原本的访问权限。建议用户在删除资源对象后，同时清理对应的访问权限。**

使用示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
res = client.revoke_from_user(
        user='app_user',
        privileges={
            "resource": "db0.*",
            "actions": ["read"]
        })
print(res)
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
user	用户名	-	string	是	需撤销操作权限的用户名。 • 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。
privileges	指定撤销用户的资源对象以及其权限，支持一次撤销多条权限。	resource	string	是	指定用户撤销的资源对象，必须与授权时指定的层级范围一致。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有Collection。配置方式：`"resource": "db0.*"`，其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
		actions	array	是	指定资源对象允许操作的权限，必须与授权时的权限保持一致。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

输出信息

```
{
```

```
    "code": 0,
    "msg": "Operation success"
}
```

版权所有：腾讯云计算（北京）有限责任公司

查询指定用户权限

最近更新时间：2025-01-21 19:52:12

接口定义

describe_user() 接口用于查询指定用户的信息，包括用户的创建时间、拥有的权限列表。

```
def describe_user(self, user: str) -> dict:
```

使用示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
res = client.describe_user(user='app_user')
print(res)
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
user	需查询的用户名	-	string	是	指定需查询的用户名。 **说明：** root 用户享有最高权限级别，无法被任何账号（包括其自身）删除或修改其权限。

出参描述

```
{
  "code": 0,
  "msg": "Operation success",
  "user": "app_user",
```

```
  "createTime": "2024-12-27 16:47:25",
  "privileges": [
    {
      "resource": "db0.*",
      "actions": ["read"]
    },
    {
      "resource": "db1.coll1",
      "actions": ["readWrite"]
    }
  ]
}
```

参数名	子参数	参数配置
user	-	查询的用户名。
createTime	-	用户创建时间。
privileges	resource	用户可操作的资源对象层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式： `"resource": "db0.*"`， 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

查看用户列表

最近更新时间：2025-01-21 19:52:12

接口定义

user_list() 接口用于查询实例下的所有用户。

```
def user_list(self) -> List[dict]:
```

使用示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
res = client.user_list()
print(res)
```

输出信息，如下所示：

```
[
  {
    "user": "root",
    "createTime": "",
    "privileges": [
      {
        "resource": "*.*",
        "actions": ["fullAccess"]
      }
    ]
  },
  {
    "user": "app_user",
    "createTime": "2024-12-27 16:47:25",
    "privileges": [
      {
        "resource": "db0.*",
        "actions": ["read"]
```

```
        },
        {
          "resource": "db1.coll1",
          "actions": ["readWrite"]
        }
      ]
    }
]
```

参数名	子参数	参数配置
user	-	查询的用户名，包含显示 root 用户以及所有已创建的用户。
createTime	-	用户创建时间，其中 root 用户创建时间为空。
privileges	resource	用户可操作的资源对象层级。 • Global：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

删除用户

最近更新时间：2025-01-21 19:52:12

接口定义

drop_user() 接口用于删除自定义的用户。

```
def drop_user(self, user: str) -> dict:
```

使用示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

res = client.drop_user(user='app_user')
print(res)
```

入参描述

参数名	参数含义	数据类型	是否必填	参数配置
user	需删除的自定义的用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。 **说明：** root 用户无法被任何账号（包括其自身）删除。

输出消息

```
{
```

```
    "code": 0,
    "msg": "Operation success"
}
```

重置用户密码

最近更新时间：2025-01-21 19:52:12

接口定义

change_password() 用于修改自定义用户密码。

```
def change_password(self,
                    user: str,
                    password: str) -> dict:
```

使用示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.RPCVectorDBClient(url='http://10.0.X.X:80',
username='root', key='eC4bLRy2va*****************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
res = client.change_password(user='app_user', password='########')
print(res)
```

入参描述

参数名	参数含义	数据类型	是否必填	参数配置
user	指定用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母、数字、下划线，并以英文字母开头。
password	设置新密码	string	是	• 密码长度要求：8~128字符。 • 密码字符类型要求如下： ○ 字母：大写 A~Z、小写 a~z。 ○ 数字：0~9。 ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

版权所有：腾讯云计算（北京）有限责任公司

响应消息

```
{
  "code": 0,
  "msg": "Operation success"
}
```

版权所有：腾讯云计算（北京）有限责任公司

Java SDK
SDK 准备

最近更新时间：2025-07-01 10:19:42

腾讯云向量数据库（Tencent Cloud VectorDB）的 Java SDK 是将 HTTP API 封装成易于使用的 Java 函数或类。开发者可以通过 Java SDK 更加方便地操作数据库。

SDK 信息

语言	版本	SDK 仓库	SDK 源码
Java	Java 8 或更高版本	com/tencent/tcvectordb/vectordatabase-sdk-java	vectordatabase-sdk-java

接入方式

> ⚠ **注意：**
> 如下为 Gradle 与 Maven 项目添加 SDK 依赖的不同方式，请依据实际需求添加最新版本。

Gradle 引入

请在 Gradle 项目的 build.gradle 文件中添加如下依赖。

```
com.tencent.tcvectordb:vectordatabase-sdk-java:2.5.0
```

Maven 引入

请在 Maven 项目的 pom.xml 文件中添加如下依赖。

```
<dependency>
    <groupId>com.tencent.tcvectordb</groupId>
    <artifactId>vectordatabase-sdk-java</artifactId>
    <version>2.5.0</version>
</dependency>
```

版权所有：腾讯云计算（北京）有限责任公司

Java SDK Demo

最近更新时间：2025-05-26 19:31:32

类别	场景描述	SDK Demo
写入向量数据并检索	操作 Base 类数据库，直接写入向量数据并进行精确查询与相似度检索。	VectorDBExample.java
写入原始文本并检索（Embedding）	操作 Base 类数据库，应用 Embedding 模型写入或更新原始文本，并进行精确查询或相似度检索。	VectorDBExampleWithEmbedding.java
使用 AI 套件上传文件	基于 Collection 或 CollectionView 上传文件并进行内容相似性检索。	• VectorDBExampleWithAI_doc.java • VectorDBExampleWithCollectionUploadFile.java
使用稀疏向量	操作 Base 类数据库，应用 BM25 写入稀疏向量，使用语义检索（稠密向量）和关键字检索（稀疏向量）两路混合召回。	VectorDBExampleWithSparseVector.java

新建 Client

最近更新时间：2025-07-01 10:19:42

功能介绍

- **RPCVectorDBClient()** 用于通过 RPC（Remote Procedure Call，远程过程调用）的请求方式创建一个向量数据库的客户端对象。
- **VectorDBClient()** 为了兼容旧版本而保留，用于通过 HTTP I/O 请求方式创建一个向量数据库的客户端对象。

说明：

- Java SDK 为1.3.5及以上版本支持 RPC（Remote Procedure Call，远程过程调用）通信，在多线程场景，RPC 可以提供比 HTTP I/O 更好的并发请求能力。
- RPCVectorDBClient() 支持创建连接池，特别适合客户端节点较少但 collection 副本较多（尤其是单 Shard 多副本）的场景。

RPCVectorDBClient()

```
public RPCVectorDBClient(ConnectParam connectParam,
ReadConsistencyEnum readConsistency)
```

VectorDBClient()

```
public VectorDBClient(ConnectParam connectParam, ReadConsistencyEnum
readConsistency)
```

使用示例

RPC 请求（推荐）

```
import com.tencent.tcvectordb.client.VectorDBClient;
import com.tencent.tcvectordb.model.*;


public class VectorDBExample {
    public static void main(String[] args) {
        // 创建VectorDB Client
        ConnectParam connectParam = ConnectParam.newBuilder()
                .withUrl("http://10.0.X.X:80")
                .withUsername("root")
                .withKey("eC4bLRy2va*****************************")
                .withTimeout(30)
                .build();
        VectorDBClient client = new
RPCVectorDBClient(connectParam,ReadConsistencyEnum.EVENTUAL_CONSISTE
NCY);
    }
}
```

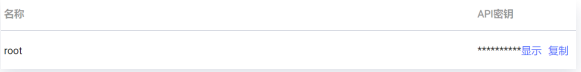
HTTP IO 请求（兼容 Restful 旧版本）

```
import com.tencent.tcvectordb.client.VectorDBClient;
import com.tencent.tcvectordb.model.*;

public class VectorDBExample {
    public static void main(String[] args) {
        // 创建VectorDB Client
        ConnectParam connectParam = ConnectParam.newBuilder()
                .withUrl("http://10.0.X.X:80")
                .withUsername("root")
                .withKey("eC4bLRy2va*****************************")
                .withTimeout(30)
                .withConnectTimeout(5)
                .withMaxIdleConnections(10)
                .withKeepAliveDuration(5 * 60)
                .build();
        VectorDBClient client = new
VectorDBClient(connectParam,ReadConsistencyEnum.EVENTUAL_CONSISTENCY
```

```
);
        }
    }
```

入参描述

参数	子参数	参数含义	是否必选	获取方式
connectParam	Url	客户端所需连接的向量数据库服务端访问地址。	是	获取向量数据库实例内网 IP 地址与端口，请登录 向量数据库控制台，在**实例详情**页面**网络信息**区域直接复制**访问地址**。具体操作，请参见 查看实例信息 。
	Username	客户端访问向量数据库服务端的账号。	是	数据库当前仅支持 root 账号。
	Key	客户端访问向量数据库服务端的 API 密钥，用于进行身份认证。	是	请登录 向量数据库控制台，在**密钥管理**页面直接复制密钥。具体操作，请参见 密钥管理 。 
	Timeout	请求超时时间。	是	• 单位：秒。 • 取值范围：大于等于0。 • 默认值：10。
	connectTimeout	指建立新连接时的超时时间。	否	• 单位：秒。 • 默认值：10。 • 取值范围：大于0。

	maxIdleConnections	指最大空闲连接数。	否	• RPC 默认值：1。 • HTTP 默认为 10。 • 正整数，建议设置为数据节点个数的2-5倍。
	keepAliveDuration	设置空闲连接的超时时间。	否	• 单位：秒 • 取值范围：大于0。 • 默认值：300。
readConsistency	-	设置读一致性。	否	取值如下所示，默认为EVENTUAL_CONSISTENCY。 • ReadConsistencyEnum.STRONG_CONSISTENCY：强一致性。 • ReadConsistencyEnum.EVENTUAL_CONSISTENCY：最终一致性。

输出说明

说明：
如果抛出异常，说明连接数据库异常。具体异常原因，可根据提示信息进行分析。无任何提示信息说明执行成功。

Database 操作
新建 Database

最近更新时间：2024-09-29 15:47:41

接口定义

- **createDatabase()** 用于创建一个 Base 类的向量数据库，存储向量数据或写入文本。
- **createAIDatabase()** 用于创建一个 AI 类向量数据库，上传并存储文件。

createDatabase()

```
public Database createDatabase(String databaseName);
```

createAIDatabase()

```
public AIDatabase createAIDatabase(String databaseName)
```

使用示例

基于已创建的 Client，创建数据库。

创建 Base 类数据库

```
// client 为 VectorDBClient() 创建的客户端对象
Database db = client.createDatabase("db-test");
```

创建 AI 类向量数据库

版权所有：腾讯云计算（北京）有限责任公司

```
// client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.createAIDatabase("db-test-ai");
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database Name	是	设置 Database 名称。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

说明：
调用该接口之后，如果抛出异常，说明创建数据库失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明创建数据库执行成功，可使用 listDatabase() 查看已经创建的数据库。

检查 Database 是否存在

最近更新时间：2024-10-16 14:54:22

接口定义

- **existsDatabase()**用于检查指定的数据库是否已经存在。
- **createDatabaseIfNotExists()**用于检查指定的数据库是否已经存在，若不存在直接创建数据库。

existsDatabase()

```
public Boolean existsDatabase(String databaseName)
```

createDatabaseIfNotExists()

```
public Database createDatabaseIfNotExists(String databaseName)
```

使用示例

existsDatabase()

```
private static final String DBNAME = "db-test";
Boolean isExisted = client.existsDatabase(DBNAME);
System.out.println(DBNAME + " exists: "+ isExisted);
```

createDatabaseIfNotExists()

```
private static final String DBNAME = "db-test";
```

```
Database db = client.createDatabaseIfNotExists(DBNAME);
```

参数名	是否必须	参数含义
databaseName	是	指定检查的数据库名。

输出描述

```
"db-test exists: True
```

参数名	返回值
isExisted	• True：数据库存在 • False：数据库不存在

删除 Database

最近更新时间：2024-09-29 15:47:41

接口定义

- **dropDatabase()** 用于删除一个 Base 类向量数据库。
- **dropAIDatabase()** 用于删除一个 AI 类向量数据库。

dropDatabase()

```
public Database dropDatabase(String databaseName);
```

dropAIDatabase()

```
public AffectRes dropAIDatabase(String databaseName)
```

使用示例

删除 Base 类数据库

```
// client 为 VectorDBClient() 创建的客户端对象
client.dropDatabase("db-test");
```

删除 AI 类数据库

```
// client 为 VectorDBClient() 创建的客户端对象
```

```
client.dropAIDatabase("db-test-ai");
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database Name	是	设置 Database 名称。	使用 listDatabase() 查询需删除的数据库。

出参描述

说明：
调用该接口之后，如果抛出异常，说明创建数据库失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明创建数据库执行成功，可使用 listDatabase() 查看已经创建的数据库。

查询 Database

最近更新时间：2024-09-29 15:47:41

接口定义

listDatabase() 用于查询集群中所有的向量数据库。

```
public List<String> listDatabase();
```

使用示例

```
// client 为 VectorDBClient() 创建的客户端对象
List<String> database = client.listDatabase();
for (String s : database) {
    System.out.println("\tres: " + s);
}
```

出参描述

```
res: db-test-base
res: db-test
res: go-sdk-test-db
res: go-sdk-test-ai-db
res: db_test-ai
```

Collection 操作
新建 Collection

最近更新时间：2025-07-09 15:17:22

接口定义

createCollection() 用于为已创建的 Base 类向量数据库创建 Collection。

```
public Collection createCollection(String databaseName,
CreateCollectionParam param)
```

> **说明：**
> 当前版本一个数据库实例下，不支持创建同名的 Collection。

使用示例

无 Embedding 模型（向量读写）

创建一个名为 book-vector 的集合，不配置 Embedding 模型相关参数，用于写入3维向量数据。

```
private static final String DBNAME = "db-test";
// 初始化 Colleciton 参数
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("book-vector")
        .withShardNum(3)
        .withReplicaNum(2)
        .withDescription("this is a collection book vector")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
        .addField(new VectorIndex("vector", 3, IndexType.HNSW,
                MetricType.COSINE, new HNSWParams(16, 200)))
        .addField(new FilterIndex("bookName", FieldType.String,
IndexType.FILTER))
        .addField(new FilterIndex("author", FieldType.String,
IndexType.FILTER))
        .addField(new FilterIndex("tags", FieldType.Array,
IndexType.FILTER))
```

```
        .build();
Collection collection = client.createCollection(DBNAME,
collectionParam);
```

Embedding 模型（文本读写）

创建一个名为 book-emb 的集合，配置 Embedding 模型相关参数，用于写入原始文本。Embedding 模型自动将原始文本进行向量化。

```
// link database, client 为 VectorDBClient() 创建的客户端对象
Database db = client.database("db-test");
// 初始化 Colleciton 参数
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
          .withName("book-emb")
          .withShardNum(3)
          .withReplicaNum(2)
          .withDescription("this is an embedding collection")
          .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
          .addField(new VectorIndex("vector", 768, FieldType.Vector,
IndexType.HNSW,
                  MetricType.IP, new HNSWParams(16, 200)))
           // 稀疏向量可根据需求添加
          //.addField(new FilterIndex("sparse_vector",
IndexType.INVERTED, MetricType.IP))
          .addField(new FilterIndex("bookName", FieldType.String,
IndexType.FILTER))
          .addField(new FilterIndex("author", FieldType.String,
IndexType.FILTER))
          .addField(new FilterIndex("tags", FieldType.Array,
IndexType.FILTER))
          .addField(new FilterIndex("bookName", FieldType.String,
IndexType.FILTER))

.withEmbedding(Embedding.newBuilder().withVectorField("vector")
                  .withField("text").withModelName("bge-base-
zh").build())
          .build();
```

```
  Collection collection = db.createCollection(collectionParam);
```

仅稀疏向量

创建一个名为 book-vector 的集合，不配置 Embedding 模型相关参数，用于写入3维向量数据，并支持稀疏向量写入。

```
// link database, client 为 VectorDBClient() 创建的客户端对象
Database db = client.database("db-test");
// 初始化 Colleciton 参数
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("book-vector")
        .withShardNum(3)
        .withReplicaNum(2)
        .withDescription("this is a collection book vector")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
        .addField(new SparseVectorIndex("sparse_vector",
IndexType.INVERTED, MetricType.IP))
        .addField(new FilterIndex("bookName", FieldType.String,
IndexType.FILTER))
        .addField(new FilterIndex("author", FieldType.String,
IndexType.FILTER))
        .addField(new FilterIndex("tags", FieldType.Array,
IndexType.FILTER))
        .build();
Collection collection = db.createCollection(collectionParam);
```

支持二进制

```
// link database, client 为 VectorDBClient() 创建的客户端对象
Database db = client.database("db-test");
// 初始化 Colleciton 参数
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("bin-vector")
```

```
        .withShardNum(3)
        .withReplicaNum(2)
        .withDescription("this is a collection bin vector")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
        .addField(new VectorIndex("vector", 16,
FieldType.BinaryVector, IndexType.BIN_FLAT,
                MetricType.HAMMING))
        .addField(new FilterIndex("bookName", FieldType.String,
IndexType.FILTER))
        .build();
Collection collection = db.createCollection(collectionParam);
```

支持 json 与 autoid

```
// link database, client 为 VectorDBClient() 创建的客户端对象
Database db = client.database("db-test");
// 初始化 Colleciton 参数
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("coll-json-autoid")
        .withShardNum(3)
        .withReplicaNum(2)
        .withDescription("this is a collection book vector")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY, AutoIdType.UUID))
        .addField(new VectorIndex("vector", 3, IndexType.HNSW,
MetricType.IP, new HNSWParams(16, 200)))
        .addField(new FilterIndex("bookInfo", FieldType.Json,
IndexType.FILTER))
        .build();
Collection collection = db.createCollection(collectionParam);
```

支持文件写入

> ⚠ **注意：**

在创建存储文件内容的 Collection 时，索引字段的设计需遵循以下限制与建议，否则可能导致上传文件失败。

- 文件名字段（file_name）：该字段**必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。字段名可以自定义，在上传文件时，需通过参数 FieldMappings 的 filename 映射自定义的字段名。
- 文件内容字段（text）：该字段用于存储知识点的原始文本内容。由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。字段名可以自定义，但需通过参数 FieldMappings 的 text 映射自定义的字段名。
- 文件图片信息字段（image_list）：该字段存放 PDF 文件中图片的 Key 列表。在原始检索出的文本块中，图片位置将以 {key} 的形式进行占位。通过调用接口 getImageUrl，可以获取图片 URL 地址列表，这些地址与 Key 一一对应，从而实现将原始 PDF 文档恢复并展示为 HTML 格式。
  - **不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。
  - 字段名可以自定义，但需要在上传文件时通过参数 FieldMappings 的 imageList 映射到自定义的字段名。

```
// link database， client 为 VectorDBClient() 创建的客户端对象
Database db = client.database("db-test");
// 初始化 Colleciton 参数
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("coll-file-test")
        .withShardNum(3)
        .withReplicaNum(2)
        .withDescription("this is a collection upload file")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
        .addField(new VectorIndex("vector", 768, IndexType.HNSW,
MetricType.IP, new HNSWParams(16, 200)))
        .addField(new FilterIndex("file_name", FieldType.String,
IndexType.FILTER))
        .build();
Collection collection = db.createCollection(collectionParam);
```

入参描述

参数	参数含义	子参数	是否必选	参数配置
Name	指定 Collection 的名称。	-	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
ReplicaNum	指定 Collection 的副本数。副本数是指每个主分片有多个相同的备份，用来容灾和负载均衡。	-	是	• 取值类型：int。 • 取值范围如下所示。搜索请求量越高的索引，建议设置越多的副本数，避免负载不均衡。 ○ 单可用区实例：0。 ○ 两可用区实例：[1,节点数-1]。 ○ 三可用区实例：[2,节点数-1]。
ShardNum	指定 Collection 的分片数。分片是把大数据集切成多个子数据集。	-	是	• 取值类型：int。 • 取值范围：[1,100]。例如：5。 • 配置建议：在搜索时，全部分片是并发执行的，分片数量越多，平均耗时越低，但是过多的分片会带来额外开销而影响性能。 ○ 单分片数据量建议控制在300万以内，例如500万向量，可设置2个分片。 ○ 如果数据量小于300万，建议使用1分片。系统对1分片有特定优化，可显著提升性能。
Description	指定 Collection 的描述信息。	-	否	• 取值类型：string。 • 字符长度要求：[1,256]。 • 示例：this is the collection description。
addField	FilterIndex	fieldName	是	配置可作为 Filter 索引的自定义扩展的标量字段名。 **说明：** • Filter 索引（Filter Index）对**标量字段**或**稀疏向量**建立的索引。标量字段名称、类型均由用户自定义，不限制标量字段数量.

			• 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询以此来匹配相似向量。 • 建立 Filter 索引时，选取需要使用 Filter 表达式高效过滤数据的标量字段。不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。 • 必须构建唯一的 Document id 为主键的 Filter 索引，配置 **name** 固定为 **id** 。 • 配置其他自定义扩展的可作为 Filter 索引的标量字段，例如：author、page 等。该标量字段名称、类型均由用户自定义，且不限制字段数量。
		filedType	是	指定 Filter 字段的数据类型。取值如下： • **String**：字符型。若 **name** 为 **id**，则该参数固定为 **FieldType.String**。 • **Uint64**：无符号整数，该参数可设置为 **FieldType.Uint64**。 • **Array**：数组类型，该参数可设置为 **FieldType.Array**，数组元素默认为 **string**。 • **Json**：由键值对组成的数据对象。具体规则，请参见 json 类型 。为 json 类型的字段创建 Filter 索引，如下所示。 **注意：** 开启 动态标量字段（动态 Schema ）时，即使用 FilterIndexConfig 开启所有标量字段全索引模式，现已支持配置 json 类型的 Filter 索引。 `.addField(new FilterIndex("bookInfo", FieldType.Json, IndexType.FILTER))`
		indexTyp	是	指定 Filter 字段的索引类型。

		e		• fieldName 为 id，该参数固定配置为 IndexType.PRIMARY_KEY，即以 id 为主键构建索引。 • fieldName 为其他自定义的标量字段，预在检索时，依据该字段过滤数据，可配置该字段索引类型为 IndexType.FILTER。
		autoIdType	否	自动生成 ID。指定为 `AutoIdType.UUID` ，即可开启自动生成 ID 功能。 **说明：** • 当启用 autoId 功能后，用户在写入或更新数据时，无需手动传入 id 字段。若用户在启用 autoId 的情况下仍传入了 id 字段，则系统将优先采用用户传入的 id 值作为数据的唯一标识。 • 在执行批量写入（batch upsert）时，如果部分数据包含 id 字段，而另一部分数据未包含 id 字段，则系统仅对未传入 id 的数据生成默认 id。
	VectorIndex	fieldName	是	指定向量索引的索引对象为 vector。
		FieldType	是	向量字段的数据类型。支持不同类型、不同精度的向量数据，以适用不同业务场景。更多信息，请参见 数据类型 。 • Vector：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位），适用于需最高精度的计算任务或对误差敏感的向量检索。 • Float16Vector：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 BGE-large 等模型生成的嵌入向量。 • BFloat16Vector：bfloat16浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，适用于向量数值范围较大（更多 bit 位表达整数）的场景。

				• **BinaryVector**：每个维度仅用1个比特（bit）表示0或1，无指数位、尾数位和符号位，直接存储二进制数据，适用于存储稀疏向量。
		indexType	是	指定索引类型，取值如下所示。更多信息，请参见 索引类型 。 • **FLAT**：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。 • **HNSW**：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区间数据规模。 • **BIN_FLAT**：二进制索引，暴力检索，召回率100%。 • **IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16**：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。
		dimension	是	指定向量维度。 • 取值类型：uint 64。 • 取值范围：[1,4096]。 • 配置建议：维度建议为4的整数倍，字节对齐有助于提升搜索性能。维度越高，存储成本越高，检索效率越低。 **说明：** 若 indexType 为 BIN_FLAT，向量维度大于等于8，且必须为8的倍数。 开通 Embedding 功能，则无需配置该字段，该字段将自动配置为 Embedding 模型对应的向量维度。
		metricType	是	指定索引对象为 vector 的向量之间距离度量的算法。取值如下： • **L2**：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • **IP**：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。

				• COSINE：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度。所得值越大越与搜索值相似。 • HAMMING：汉明距离（Hamming Distance），计算两个二进制字符串对应位置上不同字符的数量，如果字符不同，两字符串的汉明距离就会加一。汉明距离越小，表示两个字符串之间的相似度越高。
		params	否	指定索引类型 indexType 为 HNSW，则 HNSWParams 需配置如下参数。 ○ m：每个节点在检索构图中可以连接多少个邻居节点。 ○ 取值类型：uint64。 ○ 取值范围：[4,64]。默认为16。 • intefconstruction：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。 ○ 取值类型：uint64。 ○ 取值范围：[8,512]。默认为200。
			否	指定索引类型 indexType 分别为 IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16，则 IVFFLATParams/IVFPQParams/ IVFSQ8Params 需配置如下参数。 ○ NList：指索引中的聚类中心数量。 ○ 取值类型：uint64。 ○ 取值范围：[1,65536]。 ○ M：指乘积量化中原始数据被拆分的子向量的数量。该参数仅IVF_PQ 索引类型需配置。更多信息，请参见 索引与计算 。 ○ 取值要求：原始数据的向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。 ○ 取值范围：[1,向量维度]。
	SparseVectorIndex	fieldName	否	可选择构建稀疏向量索引，字段名固定为 sparse_vector。每个集合仅支持创建1个稀疏向量索引，无需指定稀疏向量维度。

		indexType		稀疏向量索引类型固定为 IndexType.INVERTED。
		metricType		稀疏向量相似性计算仅支持设置为 IP。
TtlConfig	标识数据库是否开启 TTL 配置，指定存储数据过期时间戳字段名。	Enable	否	标识数据库是否开启 TTL 属性。 • true：开启。 • false：关闭。 **说明：** TTL（生存周期）功能规定了数据自创建或更新后将被自动删除的时间期限。向量数据库默认每小时轮询检查一次数据是否过期，超期即清理，且插入数据时允许存在最多1小时的过期时间误差。具体信息，请参见 TTL。
		TimeField		指定存储数据过期时间戳的字段名。插入数据时，以标准的 Unix 时间戳指定该字段的值。字段名要求如下： • 数据类型：uint64。 • 索引类型：Filter 索引。 • 示例：expired_at。 `.withTtlConfig(TTLConfig.newBuilder().WithEnable(true).WithTimeField("expired_at").build())`
FilterIndexConfig	支持开启所有标量字段全索引模式，默认关闭。 **说明**	FilterAll		控制标量字段全索引模式的开启和关闭。 • true：开启。 • false：关闭。默认值为 fasle。
		FieldWithoutF		当 FilterAll 为 true 时，支持通过 fieldWithoutFilterIndex 参数指定不创建索引的字段。 • 若 filterAll 为 false 时不支持配置。

	：使用Json类型的字段，目前不支持创建动态索引。	ilterIndex		• 默认值：NULL。

		MaxStrLen		当 filterAll 为 true 时，支持通过 maxFieldLength 参数设定单条文档中创建索引的标量字段的最大字符数。如果字段字符数超出此限制，将按照设定的 maxStrLen 值进行截断创建索引。
				• 默认值：32。
				• 取值范围：[1,65536]。
				`.withFilterIndexConfig(FilterIndexConfig.newBuilder()`
				`.withFilterAll(true)`
				`.withFieldWithoutFilterIndex(Arrays.asList("test1", "test2"))`
				`.withMaxStrLen(64)`
				`.build())`

Embedding	设置 Embedding 相关参数	ModelName	否	指定使用的 Embedding 模型的名称。您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，参见 Embedding 介绍 。取值如下所示： • bge-large-zh-v1.5：适用中文，1024维，推荐使用。 • bge-base-zh-v1.5：适用中文，768维。 • bge-large-zh：适用中文，1024维。 • bge-base-zh：适用中文，768维。 • m3e-base：适用中文，768维。 • e5-large-v2：适用英文，1024维。 • text2vec-large-chinese：适用中文，1024维。 • multilingual-e5-base：适用于多种语言类型，768维。

				• BAAI/bge-m3：适用于多种语言类型，1024维。
		Field	否	指定文本字段名称。取值类型：String。 **说明：** 通过 upsert() 写入数据或通过 update() 更新数据时，Embedding 模型会自动将该字段的文本内容转换成向量数据。
		VectorField	否	指定向量字段。通过 Embedding 模型生成的向量会自动存储在该字段中。固定为 `vector` 。

返回结果

说明：
createCollection() 执行之后，如果抛出异常，说明创建 Collection 失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明创建 Collection 执行成功，可使用 listCollections() 查看已经创建的 Collection。

检查 Collection 是否存在

最近更新时间：2025-07-07 16:58:12

接口定义

- **existsCollection()** 用于检查指定的数据库及其对应的集合是否已经存在。
- **createCollectionIfNotExists()** 用于检查指定的数据库及其对应的集合是否已经存在，若不存在直接创建集合。

existsCollection()

```
public Boolean existsCollection(String databaseName, String
collection)
```

createCollectionIfNotExists()

```
public Collection createCollectionIfNotExists(String databaseName,
CreateCollectionParam param)
```

使用示例

existsCollection()

```
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-vector";
Boolean isExisted = client.existsCollection(DBNAME, COLL_NAME);
System.out.println(COLL_NAME+ " exists: "+ isExisted);
```

参数名	是否必须	参数含义	配置方法

databaseName	是	检查的集合所在的数据库名	数据库名可以通过listDatabase()查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	是	检查的集合名。	集合名可以通过listcollections()查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

createCollectionIfNotExists()

```
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-vector";
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("book-vector")
        .withShardNum(3)
        .withReplicaNum(2)
        .withDescription("this is a collection book vector")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
        .addField(new VectorIndex("vector", 3, IndexType.HNSW,
                MetricType.COSINE, new HNSWParams(16, 200)))
        .addField(new FilterIndex("sparse_vector",
IndexType.INVERTED, MetricType.IP))
        .addField(new FilterIndex("bookName", FieldType.String,
IndexType.FILTER))
        .addField(new FilterIndex("author", FieldType.String,
IndexType.FILTER))
        .addField(new FilterIndex("tags", FieldType.Array,
IndexType.FILTER))
        .build();
Collection coll = client.createCollectionIfNotExists(DBNAME,
collectionParam);
Boolean isExisted = client.existsCollection(COLL_NAME);
System.out.println(COLL_NAME+ " exists: "+ isExisted);
```

参数名	是否必须	参数含义	配置方法
databaseName	是	检查的集合所在的数据库名	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
param	是	集合相关配置参数	当检查的 Collection 不存在，使用该参数的配置创建 Collection。具体相关参数，请参见 新建 Collection 。

输出描述

```
book-vector exists: true
```

参数名	返回值
isExisted	• true：集合存在 • false：集合不存在

删除 Collection

最近更新时间：2025-07-07 16:58:12

接口定义

dropCollection()用于删除已创建的 Collection。

```
void dropCollection(String databaseName, String collectionName)
```

使用示例

警告：
执行 drop 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

说明：
如果您为集合设置了别名，删除集合操作仅支持使用集合的原始名称，无法通过别名来删除对应的集合。

```
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-emb";
// drop
client.dropCollection(DBNAME, COLL_NAME);
```

入参描述

参数名	是否必选	参数含义	配置方法
databaseName	是	删除的集合所在的数据库名。	数据库名可以通过 listDatabase() 查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	是	删除的 Collection 名称。	集合名可以通过 listcollections() 查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	影响行数，即为删除的集合数量。

查询 Collections 列表

最近更新时间：2025-07-01 10:19:43

接口定义

listcollections() 接口用于查询指定 Base 类 Database 中所有的 Collection。

```
public List<Collection> listCollections(String databaseName)
```

使用示例

```
// list collections
List<Collection> cols3 = client.listCollections("db-test");
for (Collection col : cols3) {
    System.out.println(col.toString());
}
```

出参描述

输出结果，如下所示。

```
{
    "database": "db-test",
    "collection": "book-emb",
    "replicaNum": 2,
    "shardNum": 3,
    "description": "this is an embedding collection",
    "indexes": [
        {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "metricType": "COSINE",
            "params": {
                "efConstruction": 200,
```

```
                "M": 16
            },
            "dimension": 768,
            "indexedCount": 0
        },
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "sparse_vector",
            "fieldType": "sparseVector",
            "indexType": "inverted",
            "metricType": "IP"
        }
    ],
    "createTime": "2023-09-26 17:30:42",
    "embedding": {
        "field": "text",
        "vectorField": "vector",
        "model": "bge-base-zh",
        "status": "enabled"
    },
    "indexStatus": {
        "status": "ready",
        "startTime": ""
    },
    "alias": [
        "book-emb-alias"
    ]
}
```


参数	子参数	子参数	参数含义
database	-	-	显示 Collection 所在的 Database 名称。

collection	-	-	显示 Collection 的名称。
replicaNum	-	-	显示 Collection 的副本数。
shardNum	-	-	显示 Collection 的分片数。
createTime	-	-	显示 Collection 的创建时间。
description	-	-	显示 Collection 的描述信息。
documentCount	-	-	返回 Collection 中存储的 Document 数量。
indexes	主键索引	fieldName	标识索引对象为 id。
		filedType	显示该索引对象的数据类型，固定为 string。
		indexType	该参数固定显示为 primaryKey。即该索引对象以 id 为主键构建索引。
	向量索引	fieldName	标识索引对象为 vector 字段名，固定为 vector。
		filedType	指定索引对象为 vector 字段的数据类型，固定为 vector。
		indexType	显示索引类型。具体信息，请参见 索引类型 。
		indexedCount	索引对象为 vector 字段的文档数量。
		dimension	显示索引对象为 vector 的向量维度。
		metricType	显示索引象为 vector 的向量之间的距离度量的算法。
		params	显示索引类型对应的参数。

	Filter 索引	fieldName	自定义扩展的可设置 Filter 表达式的字段名。例如：page、author。
		filedType	显示字段的数据类型。
		indexType	标识自定义字段是否可以设置 Filter 表达式，固定为 **filter**。
	稀疏向量索引	fieldName	稀疏向量字段名，固定为：`sparse_vector`。
		fieldType	稀疏向量数据类型，固定为：`sparseVector`。
		indexType	索引类型，固定为：`inverted`。
		metricType	稀疏向量间的距离度量的算法。
embedding	Embedding 相关参数	status	说明该 Collection 是否配置 Embedding 模型。 • **enabled**：已配置。 • **disabled**：未配置。
		field	显示 Embedding 模型输入文本的字段名。
		model	Embedding 模型的名称。
		vectorField	显示 Embedding 模型向量字段名。
alias	集合别名	-	集合别名。
indexStatus	集合重建索引相关参数	status	标识当前 Collection 是否在重建索引。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

查询指定 Collection

最近更新时间：2025-07-07 16:58:12

接口定义

describeCollection() 用于查询指定 Collection 的信息。

```
public Collection describeCollection(String databaseName, String
collectionName)
```

使用示例

```
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-emb";
// describe collection
Collection coll = client.describeCollection(DBNAME, COLL_NAME);
System.out.println(coll.toString());
```

入参描述

参数名	是否必选	参数含义	配置方法
databaseName	是	查询的集合所在的数据库名。	数据库名可以通过 listDatabase() 查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	是	查询的 Collection 名称。	集合名可以通过 listcollections() 查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

输出结果，如下所示。

```
{
```

```
    "database": "db-test",
    "collection": "book-emb",
    "replicaNum": 2,
    "shardNum": 3,
    "description": "this is an embedding collection",
    "indexes": [
        {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "metricType": "COSINE",
            "params": {
                "efConstruction": 200,
                "M": 16
            },
            "dimension": 768,
            "indexedCount": 0
        },
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
        }
    ],
    "createTime": "2023-09-26 17:30:42",
    "embedding": {
        "field": "text",
        "vectorField": "vector",
        "model": "bge-base-zh",
        "status": "enabled"
    },
    "indexStatus": {
        "status": "ready",
```

```
        "startTime": ""
    },
    "alias": [
        "book-emb-alias"
    ]
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 Collection 所在的 Database 名称。
collection	-	-	显示 Collection 的名称。
replicaNum	-	-	显示 Collection 的副本数。
shardNum	-	-	显示 Collection 的分片数。
createTime	-	-	显示 Collection 的创建时间。
description	-	-	显示 Collection 的描述信息。
documentCount	-	-	返回 Collection 中存储的 Document 数量。
indexes	主键索引	fieldName	标识索引对象为 id。
		filedType	显示该索引对象的数据类型，固定为 string。
		indexType	该参数固定显示为 primaryKey。即该索引对象以 id 为主键构建索引。
	向量索引	fieldName	标识索引对象为 vector 字段名，固定为 vector。
		filedType	指定索引对象为 vector 字段的数据类型，固定为 vector。

		indexType	显示索引类型。具体信息，请参见 索引类型 。
		indexedCount	索引对象为 vector 字段的文档数量。
		dimension	显示索引对象为 vector 的向量维度。
		metricType	显示索引象为 vector 的向量之间的距离度量的算法。
		params	显示索引类型对应的参数。
	Filter 索引	fieldName	自定义扩展的可设置 Filter 表达式的字段名。例如：page、author。
		filedType	显示字段的数据类型。
		indexType	标识自定义字段是否可以设置Filter 表达式，固定为 **filter**。
	稀疏向量索引	fieldName	稀疏向量字段名，固定为：`sparse_vector` 。
		fieldType	稀疏向量数据类型，固定为：`sparseVector` 。
		indexType	索引类型，固定为：`inverted` 。
		metricType	稀疏向量间的距离度量的算法。
embedding	Embedding 相关参数	status	说明该 Collection 是否配置 Embedding 模型。 • **enabled**：已配置。 • **disabled**：未配置。
		field	显示 Embedding 模型输入文本的字段名。
		model	Embedding 模型的名称。
		vectorField	显示 Embedding 模型向量字段名。
alias	集合别名	–	集合别名。

indexStatus	集合重建索引相关参数	status	标识当前 Collection 是否在重建索引。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

清空 Collections 数据

最近更新时间：2025-07-07 16:58:12

接口定义

truncateCollections() 用于清空 Base 类数据库的 Collection 中所有的数据与索引，仅保留 Collection 配置信息，例如索引类型及参数、分片等设置，减少用户的操作成本。

```
AffectRes truncateCollections(String databaseName, String
collectionName)
```

接口约束

> **警告：**
> 执行 truncate 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

使用示例

```
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-vector";
// truncate
AffectRes affectRes = client.truncateCollections(DBNAME, COLL_NAME);
System.out.println("\tres: " + affectRes);
```

入参描述

参数名	是否必选	参数含义	配置方法
databaseName	是	查询的集合所在的数据库名	数据库名可以通过 listDatabase() 查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	是	查询的 Collection 名称。	集合名可以通过 listcollections() 查询并复制，其命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。

			• 长度要求：[1,128]。

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	影响行数，即为清空数据的集合数量。

CollectionView 操作
创建 CollectionView

最近更新时间：2024-12-05 20:37:02

接口定义

createCollectionView() 用于为已创建的 AI 类向量数据库创建 CollectionView。

```
public CollectionView createCollectionView(@NotNull
CreateCollectionViewParam param)
```

使用示例

在 AI 类数据库 db-test-ai 下，创建一个名为 coll-ai-files 的集合视图，用于上传并存储文件。

```
// link database， client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// 初始化 ColletionView 参数
CreateCollectionViewParam collectionParam =
CreateCollectionViewParam.newBuilder()
                .withName("coll-ai-files")
                .withDescription("test create ai collection")

.withEmbedding(EmbeddingParams.newBuilder().withEnableWordEmbedding(true
).withLanguage(LanguageType.ZH).Build())
                .addField(new FilterIndex("author", FieldType.String,
IndexType.FILTER))
                .addField(new FilterIndex("tags", FieldType.Array,
IndexType.FILTER))

.withSplitterPreprocess(SplitterPreprocessParams.newBuilder()
                        .withAppendKeywordsToChunkEnum(true)
                        .withAppendTitleToChunkEnum(false)
                        // ChunkSplitter 参数以正则表达式的方式配置文档拆分方
式，如下：\n{2,} 代表以两个及以上的换行进行拆分，常用在QA对文件拆分中。
                        //.withChunkSplitter("\n{2,}")
                        .Build())

.withParsingProcess(ParsingProcessParam.newBuilder().withParsingType(Par
singTypeEnum.VisionModel).build())
```

```
                .build();
// Create CollectionView
CollectionView cv = db.createCollectionView(collectionParam);
```

入参描述

参数	参数含义	子参数	是否必选	配置方法
`Name`	指定 CollectionView 的名称。	-	是	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
`Description`	指定 CollectionView 的描述信息。	-	否	• 取值类型：string。 • 字符长度要求：[1,256]。 • 示例：this is the collection description。
`Embedding`	Embedding 相关配置。	Language	否	取值如下所示： • ZH：中文。 • EN：英文。 • MULTI：多语言。
		EnableWordEmbedding	否	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
`SplitterPreprocess`	文件预处理方式配置。	AppendTitleToChunkEnum	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • true：将段落 Title 追加到切分后的段落。 • false：不追加。默认值为 false。
		AppendKeywordsToChunkEnu	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • true：将全文的 keywords 追加到切分后的段落。默认值为 true。

		m		• **false**：不追加。
		ChunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
`ParsingProcess`	PDF 文件解析方式	**ParsingType**	否	指定 PDF 类型文件的解析方式，取值如下所示： • ParsingTypeEnum.**VisionModel**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • singTypeEnum.**AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
`Field`	配置需使用 **FilterIndex** 索引的标量字段，以便使用该字段的 Filter 条件表达式过滤查找文件。 **说明：** 文件上传之后，会为文件 ID 构建主键索引，为文件名	`fieldName`	是	配置可作为 **FilterIndex** 自定义扩展的标量字段名。请求示例中为：author、tags。 **说明：** • Filter 索引（Filter Index）是建立在标量字段的索引。该标量字段名称、类型均由用户自定义，不限制标量字段数量。 • 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询以此来匹配相似向量。 • 建立 Filter 索引时，选取文件 Metadata 信息需要使用 Filter 表达式高效过滤数据的标量字段。不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有

	构建 Filter 索引。			标量字段建立索引，导致内存资源的浪费。
		filedType	是	指定 FilterIndex 自定义字段的数据类型。取值如下： • String：字符型。 • Uint64：指无符号整数（unsigned integer）。 • Array：数组类型，默认数组元素为 string。
		indexType	是	该参数固定设置为 FILTER 。

出参说明

说明：
createCollectionView() 执行之后，如果抛出异常，说明创建 CollectionView 失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明创建 CollectionView 执行成功，可使用 describeCollectionView() 查看已经创建的 CollectionView。

删除 CollectionView

最近更新时间：2024-09-29 15:47:41

接口定义

dropCollectionView() 用于删除已创建的 CollectionView。

```
public AffectRes dropCollectionView(String collectionViewName)
```

使用示例

```
// link database, client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// drop
AffectRes affectRes = db.dropCollectionView("coll-ai-files");
System.out.println("\tres: " + affectRes);
```

入参描述

参数名	是否必选	参数含义	配置方法
collectionViewName	是	所需删除的 CollectionView 名称。	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	影响行数，即为删除的集合视图数量。

查询 CollectionView 列表

最近更新时间：2024-12-05 20:37:02

接口定义

listCollectionView() 接口用于查询指定 AI 类 Database 中所有的 CollectionView。

```
public List<CollectionView> listCollectionView()
```

使用示例

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// list all collectionViews
List<CollectionView> cols = db.listCollectionView();
for (CollectionView col : cols) {
    System.out.println("\tres: " + col.toString());
}
```

出参描述

```
res: {
  "database": "db_test-ai",
  "collectionView": "coll-ai-files",
  "description": "test create ai collection",
  "createTime": "2023-12-07 17:13:04",
  "stats": {
    "indexedDocumentSets": 0,
    "totalDocumentSets": 0,
    "unIndexedDocumentSets": 0
  },
  "splitterPreprocess": {
    "appendTitleToChunk": false,
    "appendKeywordsToChunk": true
  },
  "parsingProcess": {
    "parsingType": "AlgorithmParsing"
  },
  "embedding": {
    "language": "zh",
```

版权所有：腾讯云计算（北京）有限责任公司

```
      "enableWordsEmbedding": true
    },
    "indexes": [
      {
        "fieldName": "documentSetName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "documentSetId",
        "fieldType": "string",
        "indexType": "primaryKey"
      },
      {
        "fieldName": "tags",
        "fieldType": "array",
        "indexType": "filter"
      },
      {
        "fieldName": "author",
        "fieldType": "string",
        "indexType": "filter"
      }
    ]
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 CollectionView 所在的 AI 类 Database 名称。
collectionView	-	-	显示 CollectionView 的名称。
embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • mutil：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。

alias	-	-	CollectionView 的所有别名。创建别名，请参见 /ai/alias/set 。
createTime	-	-	显示 CollectionView 的创建时间。
description	-	-	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。
		unIndexedDocumentSets	未处理的文件数量。
splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • True：将段落 Title 追加到切分后的段落。 • False：不追加。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。

parsingProcess	指定 PDF 类型文件的解析方式	parsingType	取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 documentSetId 文件 ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 documentSetName 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式检索文件。
	其他自定义需建立 Filter 索引的标量字段	fieldName	自定义扩展字段，例如：author、tags。
		filedType	显示自定义字段的数据类型。
		indexType	显示自定义字段索引类别为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式进行 混合检索 。

查询指定的 CollectionView

最近更新时间：2024-12-17 11:54:52

接口定义

describeCollectionView() 用于查询指定 CollectionView 的信息。

```
public CollectionView describeCollectionView(String collectionView)
```

使用示例

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// describe collectionView
CollectionView descCollRes = db.describeCollectionView("coll-ai-files");
System.out.println("\tres: " + descCollRes.toString());
```

入参描述

参数名	是否必选	参数含义	配置方法
collectionViewName	是	所需查询的 CollectionView 名称。	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

```
res: {
  "database": "db_test-ai",
  "collectionView": "coll-ai-files",
  "description": "test create ai collection",
  "createTime": "2023-12-07 17:13:04",
  "stats": {
    "indexedDocumentSets": 0,
    "totalDocumentSets": 0,
    "unIndexedDocumentSets": 0
  },
  "splitterPreprocess": {
```

```
      "appendTitleToChunk": false,
      "appendKeywordsToChunk": true
    },
    "parsingProcess": {
      "parsingType": "AlgorithmParsing"
    },
    "embedding": {
      "language": "zh",
      "enableWordsEmbedding": true
    },
    "alias": [
      "alias-coll-ai-files"
    ],
    "indexes": [
      {
        "fieldName": "tags",
        "fieldType": "array",
        "indexType": "filter"
      },
      {
        "fieldName": "documentSetId",
        "fieldType": "string",
        "indexType": "primaryKey"
      },
      {
        "fieldName": "documentSetName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "author",
        "fieldType": "string",
        "indexType": "filter"
      }
    ]
}
```

返回参数

参数	子参数	子参数	参数含义
database	-	-	显示 CollectionView 所在的 AI 类 Database 名称。

collectionView	–	–	显示 CollectionView 的名称。
embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • mutil：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
alias	–	–	CollectionView 的所有别名。创建别名，请参见 setAlAlias()。
createTime	–	–	显示 CollectionView 的创建时间。
description	–	–	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。
		unIndexedDocumentSets	未处理的文件数量。

splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • True：将段落 Title 追加到切分后的段落。 • False：不追加。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
parsingProcess	指定 PDF 类型文件的解析方式	parsingType	取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 documentSetId 文件 ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 documentSetName 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式进行检索。
	其他自定义需建立 Filter 索引的标量字段	fieldName	自定义扩展字段，例如：author、tags。
		filedType	显示自定义字段的数据类型。

		indexType	显示自定义字段索引类别为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式进行 混合检索 。

清空 CollectionView 数据

最近更新时间：2024-09-29 15:47:41

接口定义

truncateCollectionView() 用于清空 AI 类数据库的 CollectionView 中所有的数据与索引，仅保留 CollectionView 配置信息，例如索引类型及参数等设置，减少用户的操作成本。

```
public AffectRes truncateCollectionView(String collectionViewName)
```

接口约束

> **警告：**
> 执行 truncate 操作将会永久删除指定 CollectionView 下的所有数据。在操作之前，务必谨慎考虑。

使用示例

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// truncate
AffectRes affectRes = db.truncateCollectionView("coll-ai-files");
System.out.println("\tres: " + affectRes);
```

入参描述

参数名	是否必选	参数含义	配置方法
`collectionViewName`	是	所需清空数据的 CollectionView 名称。	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义

affectedCount	影响行数，即为清空数据的集合视图数量。

Alias 操作
设置别名

最近更新时间：2024-09-29 15:47:41

接口定义

- **setAlias()** 接口用于为 Collection 设置别名。
- **setAIAlias()** 接口用于为 CollectionView 设置别名。

> **说明：**
> 别名可以是一个简短的字符串，方便标识和访问对应的集合。通常，别名会替换 Collection 名称用于业务切换等场景。一个Collection 或 CollectionView 可以设置一个或者多个别名。

setAlias()

```
public AffectRes setAlias(String collectionName,String aliasName)
```

setAIAlias()

```
public AffectRes setAIAlias(String collectionViewName, String
aliasName)
```

使用示例

> **说明：**
> - 通过集合的别名做业务迁移时，将同一别名指向新的集合，别名与集合的映射关系将自动更新为新集合，可直接通过别名访问新集合。
> - DB 和 Collection 级别（包含 AI 类 数据库的CollectionView）的 drop 操作会同时删除库表下的所有别名。
> - Document 与 DocumentSet 层级的访问优先访问别名，其余级别仅支持原 Collection 或 CollectionView 名操作。

- 集合或集合视图的别名可以和名称重复，一个集合或集合视图的多个别名之间不能重复。

创建 Collection 别名

```
// link database
Database db = client.database("db-test");
// set alias
AffectRes affectRes1 = db.setAlias("book-emb", "book-emb-alias");
System.out.println("\tres: " + affectRes1);
```

参数	是否必选	参数含义	配置方法及要求
`collectionName`	是	指定需创建别名的 Collection 名。	使用 listCollections() 获取指定数据库名下的 Collection 列表，复制需创建别名的集合名。
`aliasName`	否	设置 Collection 别名。	Collection 别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

创建 CollectionView 别名

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// set alias
AffectRes affectRes1 = db.setAIAlias("coll-ai-files", "alias-sdk-
test");
System.out.println("\tres: " + affectRes1);
```

参数	是否必选	参数含义	配置方法及要求

`collectionViewName`	是	指定需创建别名的 CollectionView 名。	使用 listCollectionView() 获取指定数据库名下的 CollectionView 列表，复制需创建别名的集合名。
`aliasName`	否	设置别名。	别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

输出信息，如下所示。

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	影响行数，即为创建别名的集合数量。

删除别名

最近更新时间：2024-09-29 15:47:41

功能介绍

- **deleteAlias()** 接口用于删除 Collection 的别名。
- **deleteAIAlias()** 接口用于删除 CollectionView 的别名。

deleteAlias()

```
public AffectRes deleteAlias(String aliasName)
```

deleteAIAlias()

```
public AffectRes deleteAIAlias(String aliasName)
```

使用示例

删除 Collection 别名

```
// link database
Database db = client.database("db-test");
// delete alias
AffectRes affectRes = db.deleteAlias("book-emb-alias");
System.out.println("\tres: " + affectRes);
```

删除 CollectionView 别名

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// delete alias
AffectRes affectRes = db.deleteAIAlias("alias-sdk-test");
System.out.println("\tres: " + affectRes);
```

入参描述

参数	是否必选	参数含义	配置方法及要求
`aliasName`	否	指定需删除的别名。	别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

出参描述

输出信息，如下所示。

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	影响行数，即为删除别名的数量。

Document 操作
插入数据

最近更新时间：2025-07-01 10:19:43

接口定义

upsert() 接口用于给创建的 Collection 中插入 Document。如果 Collection 在创建时，已配置 Embedding 参数，则仅需要插入文本信息，Embedding 服务会自动将文本信息转换为向量数据，存入数据库。

```
public AffectRes upsert(String database, String collection, InsertParam
param)
```

使用示例

说明：

- 在插入数据时， Collection 中已经存在相同 ID 的 Document，则会删除源 Document 后插入新的 Document 数据。
- 如果 Collection 在创建时，配置 Embedding 参数，则插入文本，Embedding 服务会自动将文本信息转换为向量数据，存入数据库。

写入原始文本

如果您使用 Embedding 功能，在 createCollection() 建表时，配置 Embedding 模型相关参数之后，便可以通过 **upsert()** 接口可直接传入原始文本。Embedding 模型会将原始文本转换为向量数据，并将转换后的向量数据以及原始文本一并存入数据库。具体信息，请参见 Embedding 介绍 。如下示例，基于 createCollection() 创建的集合 **book-emb**，写入原始文本。

```
Document doc1 = Document.newBuilder()
        .withId("0001")
        .addDocField(new DocField("text", "话说天下大势，分久必合，合久必
分。"))
        .addDocField(new DocField("bookName", "三国演义"))
        .addDocField(new DocField("author", "罗贯中"))
        .addDocField(new DocField("page", 21))
        .addDocField(new DocField("tags", Arrays.asList("曹操","诸葛
亮","刘备")))
        .build();
```

```
Document doc2 = Document.newBuilder()
        .withId("0002")
        .addDocField(new DocField("text",
                "正大光明，混沌未分天地乱，茫茫渺渺无人间。"))
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 22))
        .addDocField(new DocField("tags", Arrays.asList("孙悟空","猪八
戒","唐僧")))
        .build();
Document doc3 = Document.newBuilder()
        .withId("0003")
        .addDocField(new DocField("text", "甄士隐梦幻识通灵，贾雨村风尘怀
闺秀。"))
        .addDocField(new DocField("bookName", "红楼梦"))
        .addDocField(new DocField("author", "曹雪芹"))
        .addDocField(new DocField("page", 23))
        .addDocField(new DocField("tags", Arrays.asList("贾宝玉","林黛
玉","王熙凤")))
        .build();

InsertParam insertParam = InsertParam.newBuilder()
        .addDocument(doc1)
        .addDocument(doc2)
        .addDocument(doc3)
        .withBuildIndex(true)
        .build();

AffectRes affectRes = client.upsert("db-test", "book-emb",
insertParam);
System.out.println("\tres: " + affectRes);
```

写入向量数据

如果您无需使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能做向量化，则可以直接写入向量数据。

```
Document doc1 = Document.newBuilder()
        .withId("0001")
        .withVector(Arrays.asList(0.2123, 0.21, 0.213))
```

```
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 21))
        .addDocField(new DocField("tags", Arrays.asList("孙悟空","猪八
戒","唐僧")))
        .build();
Document doc2 = Document.newBuilder()
        .withId("0002")
        .withVector(Arrays.asList(0.2123, 0.22, 0.213))
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 22))
        .addDocField(new DocField("tags", Arrays.asList("孙悟空","猪八
戒","唐僧")))
        .build();
Document doc3 = Document.newBuilder()
        .withId("0003")
        .withVector(Arrays.asList(0.2123, 0.23, 0.213))
        .addDocField(new DocField("bookName", "三国演义"))
        .addDocField(new DocField("author", "罗贯中"))
        .addDocField(new DocField("page", 23))
        .addDocField(new DocField("tags", Arrays.asList("曹操","诸葛
亮","刘备")))
        .build();

InsertParam insertParam = InsertParam.newBuilder()
        .addDocument(doc1)
        .addDocument(doc2)
        .addDocument(doc3)
        .addDocument(doc4)
        .addDocument(doc5)
        .withBuildIndex(true)
        .build();
collection.upsert(insertParam);

AffectRes affectRes = client.upsert("db-test", "book-vector",
insertParam);
System.out.println("\tres: " + affectRes);
```

写入向量数据 + 稀疏向量

如果您无需使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能做向量化，则可以直接写入向量数据与稀疏向量。

```
Document doc1 = Document.newBuilder()
        .withId("0001")
        .withVector(Arrays.asList(0.2123, 0.21, 0.213))
        .withSparseVector(Arrays.asList(Pair.of(21, 0.96f),
Pair.of(51, 0.53f),Pair.of(1001, 0.443f)))
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 21))
        .addDocField(new DocField("tags", Arrays.asList("孙悟空","猪八
戒","唐僧")))
        .build();
Document doc2 = Document.newBuilder()
        .withId("0002")
        .withVector(Arrays.asList(0.2123, 0.22, 0.213))
        .withSparseVector(Arrays.asList(Pair.of(21, 0.97f),
Pair.of(51, 0.54f),Pair.of(1001, 0.444f)))
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 22))
        .addDocField(new DocField("tags", Arrays.asList("孙悟空","猪八
戒","唐僧")))
        .build();
Document doc3 = Document.newBuilder()
        .withId("0003")
        .withVector(Arrays.asList(0.2123, 0.23, 0.213))
        .withSparseVector(Arrays.asList(Pair.of(21, 0.98f),
Pair.of(51, 0.55f),Pair.of(1001, 0.445f)))
        .addDocField(new DocField("bookName", "红楼梦"))
        .addDocField(new DocField("author", "曹雪芹"))
        .addDocField(new DocField("page", 23))
        .addDocField(new DocField("tags", Arrays.asList("贾宝玉","林黛
玉","王熙凤")))
        .build();

InsertParam insertParam = InsertParam.newBuilder()
        .addDocument(doc1)
        .addDocument(doc2)
        .addDocument(doc3)
        .withBuildIndex(true)
```

```
        .build();

AffectRes affectRes = client.upsert("db-test", "book-vector-sparse",
insertParam);
System.out.println("\tres: " + affectRes);
```

文本写入 + 稀疏向量

```
SparseVectorBm25Encoder bm25Encoder =
SparseVectorBm25Encoder.getDefaultBm25Encoder();
List<String> texts = Arrays.asList(
        "腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。",
        "作为专门为处理输入向量查询而设计的数据库，它支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，高达百万级 QPS 及毫秒级查询延迟。",
        "不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。",
        "腾讯云向量数据库（Tencent Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 ",
        "腾讯云向量数据库可以和大语言模型 LLM 配合使用。企业的私域数据在经过文本分割、向量化后，可以存储在腾讯云向量数据库中，构建起企业专属的外部知识库，从而在后续的检索任务中，为大模型提供提示信息，辅助大模型生成更加准确的答案。");
List<List<Pair<Long, Float>>> sparseVectors =
bm25Encoder.encodeTexts(texts);
// upsert
List<Document> documentList = new ArrayList<>(Arrays.asList(
                Document.newBuilder()
                        .withId("0001")
                        .addDocField(new DocField("text",
texts.get(0)))
                        .withSparseVector(sparseVectors.get(0))
                        .build(),
                Document.newBuilder()
                        .withId("0002")
                        .withSparseVector(sparseVectors.get(1))
                        .addDocField(new DocField("text",
texts.get(1)))
                        .build(),
```

```
                Document.newBuilder()
                        .withId("0003")
                        .withSparseVector(sparseVectors.get(2))
                        .addDocField(new DocField("text",
texts.get(2)))
                        .build(),
                Document.newBuilder()
                        .withId("0004")
                        .withSparseVector(sparseVectors.get(3))
                        .addDocField(new DocField("text",
texts.get(3)))
                        .build(),
                Document.newBuilder()
                        .withId("0005")
                        .withSparseVector(sparseVectors.get(4))
                        .addDocField(new DocField("text",
texts.get(4)))
                        .build()));
System.out.println("---------------------- upsert ------------------
----");
InsertParam insertParam =
InsertParam.newBuilder().withDocuments(documentList).build();

AffectRes affectRes = client.upsert("db-test", "book-emb-sparse",
insertParam);
System.out.println("\tres: " + affectRes);
```

写入二进制数据

```
List<Document> documentList = new ArrayList<>(Arrays.asList(
        Document.newBuilder()
                .withId("0001")

.withVector(BinaryUtils.binaryToUint8(Arrays.asList(1, 1, 1, 0, 0,
1, 0, 1,  0, 0, 0, 0, 0, 0, 0, 0)))
                .addDocField(new DocField("bookName", "西游记"))
                .addDocField(new DocField("author", "吴承恩"))
                .build(),
        Document.newBuilder()
                .withId("0002")
```

```
.withVector(BinaryUtils.binaryToUint8(Arrays.asList(1, 1, 0, 0, 0,
1, 0, 1,  0, 0, 0, 0, 0, 0, 0, 0)))
                .addDocField(new DocField("bookName", "西游记"))
                .addDocField(new DocField("author", "吴承恩"))
                .build(),
        Document.newBuilder()
                .withId("0003")

.withVector(BinaryUtils.binaryToUint8(Arrays.asList(0, 1, 0, 0, 0,
1, 0, 1,  0, 0, 0, 0, 0, 0, 0, 0)))
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .build(),
        Document.newBuilder()
                .withId("0004")

.withVector(BinaryUtils.binaryToUint8(Arrays.asList(0, 1, 0, 0, 0,
1, 0, 1,  0, 0, 0, 0, 0, 1, 0, 0)))
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .build(),
        Document.newBuilder()
                .withId("0005")

.withVector(BinaryUtils.binaryToUint8(Arrays.asList(0, 1, 0, 0, 0,
1, 0, 1,  0, 0, 0, 0, 0, 1, 1, 0)))
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .build()));
System.out.println("---------------------- upsert ------------------
----");
InsertParam insertParam =
InsertParam.newBuilder().withDocuments(documentList).build();

AffectRes affectRes = client.upsert("db-test", "bin-vector",
insertParam);
System.out.println("\tres: " + affectRes);
```

支持 json 类型

```
List<Document> documentList = new ArrayList<>(Arrays.asList(
                Document.newBuilder()
                        .withId("0001")
                        .withVector(Arrays.asList(0.2123, 0.21,
0.213))
                        .addDocField(new DocField("bookInfo",
                                new JSONObject(new
HashMap<Object,Object>(){{
                                    put("bookName", "西游记");
                                    put("page", 24);
                                    put("author", "吴承恩");
                                }})))
                        .build(),
                Document.newBuilder()
                        .withVector(Arrays.asList(0.2123, 0.23,
0.213))
                        .addDocField(new DocField("bookInfo",
                                new JSONObject(new
HashMap<Object,Object>(){{
                                    put("bookName", "西游记");
                                    put("page", 22);
                                    put("author", "吴承恩");
                                    put("array",
Arrays.asList("test_6", "test_7", "test_3"));
                                }})))
                        .build(),
                Document.newBuilder()
                        .withVector(Arrays.asList(0.2123, 0.23,
0.213))
                        .addDocField(new DocField("bookInfo",
                                new JSONObject(new
HashMap<Object,Object>(){{
                                    put("bookName", "红楼梦");
                                    put("page", 26);
                                    put("author", "曹雪芹");
                                    put("array",
Arrays.asList("test_4", "test_2", "test_6"));
                                }})))
                        .build()));
System.out.println("---------------------- upsert ------------------
----");
```

```
InsertParam insertParam =
InsertParam.newBuilder().withDocuments(documentList).build();

AffectRes affectRes = client.upsert("db-test", "coll-json-autoid",
insertParam);
System.out.println("\tres: " + affectRes);
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法
database	指定插入数据的数据库名。	-	是	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	指定插入数据的集合名。	-	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
insertParam	指定要插入的Document数据，是一个数组，支持单次插入一条或者多条Document，最大可插入1000条。	Id	是	Document 主键，长度限制为[1,128]。
		Vector	否	表示文档的向量值， • 集合的向量索引为 FLAT/HNSW/IVF 时，直接传入向量数据，以32位浮点数存储，如[0.2123, 0.9281, ..., 0.6712]。 • 集合的向量索引为二进制索引时，写入向量时可以直接写入二进制数据，也可以十进制写入 `.withVector(Arrays.asList(123, 53))` 。十进制写入数据时，数据维度是创建集合时指定维度的1/8。例如：upsert 时 vector=[12, 21]，则创建集合时维度为16维，其代表的二进制数据为[00001100, 00010101]。 **说明：** 如果使用 Embedding 功能做向量化，则无需写入向量数据。

				text 字段为该参数为创建集合 时，Embedding 参数 field 对应指定的文本字段名。您可以自定义其他便于识别的字段名。 **说明：** 写入原始文本数据，系统会自动从该字段中提取原始文本信息，并将其转换为向量数据，并将原始文本以及转化后的向量数据一起存储在数据库中。
				sparse_vector 为创建集合时自定义的稀疏向量的字段名。
		DocField	否	其他标量字段，用于存储文档的其他信息。例如：bookName、author、page。 **说明：** 若字段为存储数据过期时间戳的标量字段，需以标准的 Unix 时间戳格式为该字段赋值，且数据类型为 uint64。例如：`.addDocField(new DocField("expired_at", System.currentTimeMillis()/1000 + 24*60*60))`。具体示例，请参见 example 。 • 若时间戳字段为空未指定具体的过期时间，插入的该条数据将不会执行过期删除操作，数据将永久保留。 • 若时间戳字段传入 Value 的数据类型有误或者不符合 Unix 时间戳规范时，将会报出错误信息，请根据提示信息修改。 • 若时间戳字段指定的过期时间小于当前插入数据时的时间，则在下一个定时删除操作时，直接删除该条数据。
		BuildIndex	否	指定是否需要更新索引，取值如下所示： • true：需更新索引。默认值是 true。 • false：不更新索引。

				注意： 如果创建 Collection 选择的索引类型为 IVF 系列： • 当第一次写入时，当前集合还没有向量索引，此时 BuildIndex 必须为 false。插入原始数据之后，需通过 rebuildIndex() 训练数据并重建索引。 • 当集合已经调用过 rebuildIndex() 创建索引后，集合已经存在向量索引，此时： ○ 如果 BuildIndex = true，表示新写入的数据会加入到已有的 IVF 索引中，但不会更新索引结构，此时新写入的数据可以被检索到。 ○ 如果 BuildIndex = false，表示新写入的数据不会加入到已有的 IVF 索引中，此时新写入的数据无法被检索到。

出参描述

```
res: AffectRes{affectedCount=5, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	插入的文档数量。

精确查询

最近更新时间：2025-07-01 10:19:43

接口定义

基于精确匹配的查询方式，query() 根据输入的文档 ID，或根据指定的查询起始位置和返回数量，精确查找与查询条件完全匹配的文档。

```
public List<Document> query(String database, String collection,
QueryParam param)
```

使用示例

说明：

- 支持根据主键 id（Document ID），搭配自定义的标量字段的 Filter 表达式一并查询数据。
- 支持指定查询起始位置 offset 和返回数量 limit，精确检索查询区间内的数据，实现数据 SCAN 能力。

```
QueryParam queryParam = QueryParam.newBuilder()
        .withDocumentIds(Arrays.asList("0001", "0002", "0003", "0004",
"0005"))
        // 使用 filter 过滤数据
        .withFilter("bookName=\"三国演义\"")
        // limit 限制返回行数，默认为 1，取值为 0 到 16384 之间
        .withLimit(5)
        // 偏移
        .withOffset(0)
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName", "page"))
        // 是否返回 vector 数据
        .withRetrieveVector(true)
        .build();

List<Document> qdos = client.query("db-test", "book-vector",
queryParam);
for (Document doc : qdos) {
   // System.out.println("\tres: " + doc.toString());
    System.out.print("id: " + doc.getId());
    System.out.print(" vector: " + (List<Double>)doc.getVector());
```

```
        System.out.print(" bookName: " + doc.getObject("bookName"));
        System.out.println(" page: " + doc.getObject("page"));
        System.out.println();
    }
```

入参描述

参数名	子参数	是否必选	参数含义	参数配置
database	-	是	指定检索数据的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	-	是	指定检索数据的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
queryParam	DocumentIds	否	表示要查询的文档的所有ID。	每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。
	RetrieveVector	否	标识是否需要返回检索结果的向量值。	• true：需要。 • false：不需要。默认为 false。
	Filter	否	设置 Filter 表达式。	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ string：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ uint64：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不

			等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `.withFilter("bookInfo.bookName=\"三国演义\"")` • <value>：表示要匹配的值。
Limit	是	每页返回的 Document 数量。	每页返回的 Document 数量。默认为10。 • 数据类型：uint 64 • 取值范围：[1,16384] **注意：** 若使用 **query** 检索数据时，不配置 **documentIds** 和 **filter** 参数，则必须配置 **offset** 和 **limit** 参数，返回从 offset 开始的 limit 条数据，避免遍历所有数据而浪费不必要的资源。
Offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。	取值要求如下： • 取值：为 **limit** 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。
Sort	否	指定查询结果按照特定字段	数据类型为数组。

			升序或降序排序。	`withSort(OrderRule.newBuilder().withFieldName("page").withDirection(OrderEnum.DESC).build())` • **FieldName**：指定特定字段，查询结果将依据该字段确定输出顺序。当前排序字段仅支持 uint64 的类型，且该字段为标量字段索引，可通过接口 describeCollection() 确认该字段是否为索引字段。 **说明：** • 当满足条件的数据量小于或等于1000时，orderby 操作将正常生效，否则会引发错误。 • 目前仅支持对单个字段进行排序，暂时不支持多字段排序功能。 • **Direction**：指定排序的方向。 ○ OrderEnum.DESC：降序排序。 ○ OrderEnum.ASC：升序排序。
	OutputFields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • **OutputFields** 与 **RetrieveVector** 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，**OutputFields** 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，**OutputFields** 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

```
id: 0005 vector: [0.21230000257492065,0.25,0.21299999952316284]
bookName: 三国演义 page: 25
id: 0004 vector:
[0.21230000257492065,0.23999999463558197,0.21299999952316284] bookName:
三国演义 page: 24
id: 0003 vector:
[0.21230000257492065,0.23000000417232513,0.21299999952316284] bookName:
三国演义 page: 23
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
docFields	Document 自定义扩展的标量字段存储的对象。

向量数据相似性检索

最近更新时间：2025-07-01 10:19:43

功能介绍

基于相似度匹配的查询方式，search() 接口根据输入的1到 N 向量数据，查找存储在数据库中的相似性向量，返回指定的 Top K 个最相似的文档。

```
public List<List<Document>> search(String database, String collection,
SearchByVectorParam param)
```

使用示例

基于向量数据检索

如下示例，检索与向量数据（0.3123, 0.43, 0.213）与（0.5123, 0.63, 0.413）的相似性数据。

```
// search by filter
System.out.println("-searchByFilter----------------------");
SearchByVectorParam searchByVectorParam =
SearchByVectorParam.newBuilder()

.withVectors(Arrays.asList(Arrays.asList(0.3123, 0.43, 0.213), Array
s.asList(0.5123, 0.63, 0.413)))
        // 若使用 HNSW 索引，则需要指定参数 ef，ef 越大，召回率越高，但也会影
响检索速度
        .withParams(new HNSWSearchParams(200))
        // 设置是否输出向量字段
        .withRetrieveVector(true)
         // 指定返回的最相似的 Top K 的 K 值
        .withLimit(2)
        // 过滤所需的结果
        // .withFilter(new Filter("bookName=\"三国演义\""))
        .withFilter("bookName=\"三国演义\"")
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName"))
        .build();

List<List<Document>> svDocs = client.search("db-test", "book-
vector", searchByVectorParam);
```

```
int i = 0;
for (List<Document> docs : svDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

输出信息，如下所示。

```
res: 0
res: {"id":"0001","vector":
[0.21230000257492065,0.20999999344348907,0.21299999952316284],"spars
e_vector":[[2,0.96],[5,0.53],
[100,0.443]],"score":0.9617376327514648,"bookName":"三国演义"}
res: 1
res: {"id":"0001","vector":
[0.21230000257492065,0.20999999344348907,0.21299999952316284],"spars
e_vector":[[2,0.96],[5,0.53],
[100,0.443]],"score":0.9846718311309814,"bookName":"三国演义"}
```

基于二进制数据检索

如下示例，输入二进制数据，在二进制集合，检索与其相似的数据。

```
SearchByVectorParam searchByVectorParam =
SearchByVectorParam.newBuilder()
        .addVector(BinaryUtils.binaryToUint8(Arrays.asList(0, 0, 0,
0, 0, 0, 0, 0,  1, 1, 0, 1, 1, 1, 0, 1)))
        // 指定 Top K 的 K 值
        .withLimit(10)
        .build();

List<List<Document>> svDocs = client.search("db-test", "book-
vector", searchByVectorParam);
int i = 0;
for (List<Document> docs : svDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
```

```
    }
}
```

输出信息，如下所示。

```
res: 0
res: {"id":"0004","score":8.0,"author":"罗贯中","bookName":"三国演义"}
res: {"id":"0001","score":9.0,"bookName":"西游记","author":"吴承恩"}
res: {"id":"0003","score":9.0,"author":"罗贯中","bookName":"三国演义"}
res: {"id":"0005","score":9.0,"author":"罗贯中","bookName":"三国演义"}
res: {"id":"0002","score":10.0,"author":"吴承恩","bookName":"西游记"}
```

支持 json 类型的数据检索

如下示例，输入二进制数据，在二进制集合，检索与其相似的数据。

```
SearchByVectorParam searchByVectorParam =
SearchByVectorParam.newBuilder()
        .withVectors(Arrays.asList(Arrays.asList(0.3123, 0.43,
0.213), Arrays.asList(0.5123, 0.63, 0.413)))
        // 若使用 HNSW 索引，则需要指定参数ef，ef越大，召回率越高，但也会影响
检索速度
        .withParams(new HNSWSearchParams(100))
        .withFilter("bookInfo.bookName=\"西游记\"")
        .withOutputFields(Arrays.asList("id", "bookInfo"))
        // 指定 Top K 的 K 值
        .withLimit(10)
        .build();
// 输出相似性检索结果，检索结果为二维数组，每一位为一组返回结果，分别对应 search
时指定的多个向量
List<List<Document>> svDocs = client.search("db-test", "coll-json-
autoid", searchByVectorParam);
i = 0;
for (List<Document> docs : svDocs) {
    System.out.println("\tres: " + i);
    i++;
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
```

```
}
```

输出信息，如下所示。

```
res: 0
res: {"id":"229CA235-B956-E150-630F-
FE1119B7C474","score":0.21057027578353882,"bookInfo":{"array":
["test_6","test_7","test_3"],"author":"吴承
恩","page":22,"bookName":"西游记"}}
res: {"id":"0001","score":0.20197027921676636,"bookInfo":
{"author":"吴承恩","page":24,"bookName":"西游记"}}
res: 1
res: {"id":"229CA235-B956-E150-630F-
FE1119B7C474","score":0.34163033962249756,"bookInfo":{"array":
["test_6","test_7","test_3"],"author":"吴承
恩","page":22,"bookName":"西游记"}}
res: {"id":"0001","score":0.32903027534484863,"bookInfo":
{"author":"吴承恩","page":24,"bookName":"西游记"}}
```

请求参数

参数名	子参数	是否必选	参数含义	配置方法
database	-	是	指定检索数据的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	-	是	指定检索数据的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
SearchByVectorParam	Vectors	是	表示要查询的向量列表。	数组元素数量最大为20。
	Limit	是	指定返回最相似的 Top K 的 K 的值。	大于等于0的正整数。

	Params	否	设置索引类型对应的检索参数。	• 若索引类型为 HNSW，设置检索参数 **efConstruction**，指定需要访问的候选向量的数目。取值范围[1,32768]，默认为10。 • 若索引类型为 IVF 系列，设置参数 **nprobe** ，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 listCollections() 查看。
	RetrieveVector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • **true**：需要。 • **false**：不需要。默认为 false。
	Limit	是	指定返回最相似的 Top K 的 K 的值。	大于等于0的正整数。

	Filter	否	设置标量字段的 Filter 表达式，过滤所需的数据。	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中：
				• <field_name>：表示要过滤的字段名。
				• <operator>：表示要使用的运算符。
				  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
				  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。
				  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
				  ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。

				若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `.withFilter("bookInfo.bookName=\"三国演义\"")` • <value>：表示要匹配的值。
	Radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= Radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= Radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= Radius 的文
	Output Fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。

出参描述

说明：

- 输出结果的顺序，与搜索时设置的 vectors 配置的向量值的顺序一致。
- 每一个查询结果都返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。

```
res: 0
```

```
res: {"id":"0001","vector":
[0.21230000257492065,0.20999999344348907,0.21299999952316284],"sparse_ve
ctor":[[2,0.96],[5,0.53],
[100,0.443]],"score":0.9617376327514648,"bookName":"三国演义"}
res: 1
res: {"id":"0001","vector":
[0.21230000257492065,0.20999999344348907,0.21299999952316284],"sparse_ve
ctor":[[2,0.96],[5,0.53],
[100,0.443]],"score":0.9846718311309814,"bookName":"三国演义"}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
score	表示查询向量与检索结果向量之间的相似性计算分数。欧式距离（L2）与汉明距离（Hamming Distance）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。
author、page、section	Document 其他自定义的标量字段。

版权所有：腾讯云计算（北京）有限责任公司

Doc ID 相似性检索

最近更新时间：2025-07-01 10:19:43

接口定义

基于相似度匹配的查询方式，**searchById()** 根据输入的1到 N 个文档 ID，查找存储在数据库中的相应向量，并执行相似性 Top K 检索。

```
public List<List<Document>> searchById(String database, String
collection, SearchByIdParam param)
```

使用示例

如下示例，检索与0002相似的数据。

```
SearchByIdParam searchByIdParam = SearchByIdParam.newBuilder()
        .withDocumentIds(Arrays.asList("0002"))
        // 若使用 HNSW 索引，则需要指定参数 ef，ef 越大，召回率越高，但也会影响检
索速度
        .withParams(new HNSWSearchParams(200))
        // 设置标量字段的 Filter 表达式，过滤所需查询的文档
        .withRetrieveVector(true)
         // 指定 Top K 的 K 值
        .withLimit(2)
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName"))
        .build();

List<List<Document>> siDocs = client.searchById("db-test", "book-
vector", searchByIdParam);
int i = 0;
for (List<Document> docs : siDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

入参描述

参数名	子参数	是否必选	参数含义	配置方法
database	-	是	指定检索数据的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	-	是	指定检索数据的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
searchByIdParam	DocumentIds	是	待检索的文档ID。	每个 ID 长度限制为[1,128]。数组元素数量最大为20。
	Params	否	设置索引类型对应的检索参数。	• 若索引类型为 HNSW，设置检索参数 efConstruction，指定需要访问的候选向量的数目。取值范围[1,32768]，默认为10。 • 若索引类型为 IVF 系列，设置参数 nprobe ，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 listCollections() 查看。
	RetrieveVector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • true：需要。 • false：不需要。默认为 false。
	Limit	是	指定返回最相似的 Top K 的 K 的值。	大于等于0的正整数。
	Filter	否	设置 Filter 表达式，过滤所需的数据。	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。

				• <operator>：表示要使用的运算符。
				  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
				  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。
				  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
				  ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。
				`.withFilter("bookInfo.bookName=\"三国演义\"")`
				• <value>：表示要匹配的值。

	Radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= Radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= Radius 的文档。

				• L2：取值范围 [0, +∞)，返回相似性计算得分 score <= Radius 的文
	OutputFields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}` ，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

输出参数

查看输出，如下所示。

说明：

- 输出的 Document ID 顺序与查询时配置的参数 document_ids 输入的顺序一致。
- 每一个查询结果都返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。

```
res: 0
res: {"id":"0002","vector":
[0.21230000257492065,0.2199999988079071,0.21299999952316284],"sparse_vec
```

```
tor":[[2,0.97],[5,0.54],
[100,0.444]],"score":1.000000238418579,"bookName":"西游记"}
res: {"id":"0003","vector":
[0.21230000257492065,0.23000000417232513,0.21299999952316284],"sparse_ve
ctor":[[2,0.98],[5,0.55],
[100,0.445]],"score":0.9997729063034058,"bookName":"红楼梦"}
```


参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
score	表示查询向量与检索结果向量之间的相似性计算分数。欧式距离（L2）与汉明距离（Hamming Distance）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。
author、page、section	Document 其他自定义的标量字段。

版权所有：腾讯云计算（北京）有限责任公司

文本信息相似性检索

最近更新时间：2025-07-01 10:19:43

接口定义

若在创建 Collection 时，已配置 Embedding 模型，则可使用 **searchByEmbeddingItems()** 接口输入原始文本，在数据库中检索与原始文档相似的文档，返回指定的 Top K 个最相似的 Document。

```
public SearchRes searchByEmbeddingItems(String database, String
collection, SearchByEmbeddingItemsParam param)
```

使用示例

如下示例已经通过 createDatabase() 创建数据库 **db-test**，已通过 createCollection() 创建集合为 **book-emb**，通过 **searchByEmbeddingItems()** 检索与 **EmbeddingItems** 参数的文本信息最相似，且满足 Filter 表达式的文档。

```
SearchByEmbeddingItemsParam searchByEmbeddingItemsParam =
SearchByEmbeddingItemsParam.newBuilder()
        .withEmbeddingItems(Arrays.asList("天下大势，分久必合"))
        // 若使用 HNSW 索引，则需要指定参数 ef，ef 越大，召回率越高，但也会影响检
索速度
        .withParams(new HNSWSearchParams(200))
        // 设置标量字段的 Filter 表达式，过滤所需查询的文档
        .withRetrieveVector(false)
         // 指定 Top K 的 K 值
        .withLimit(2)
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName"))
        .build();

List<List<Document>> siDocs = client.searchByEmbeddingItems("db-test",
"book-emb", searchByEmbeddingItemsParam).getDocuments();
int i = 0;
for (List<Document> docs : siDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println(doc.toString());
    }
}
```

入参描述

参数名	子参数	是否必选	参数含义	配置方法
database	-	是	指定检索数据的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	-	是	指定检索数据的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
searchByEmbeddingItemsParam	EmbeddingItems	是	待检索的文本信息。	输入文本信息，用于检索与该文本信息相似的数据。 • 类型：字符串数组。 • 范围：数组元素最大批量为20 。
	Params	否	设置索引类型对应的检索参数。	• 若索引类型为 HNSW，设置检索参数 **efConstruction**，指定需要访问的候选向量的数目。取值范围[1,32768]，默认为10。 • 若索引类型为 IVF 系列，设置参数 **nprobe** ，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 listCollections() 查看。
	RetrieveVector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • true：需要。 • false：不需要。默认为 false。
	Limit	是	指定返回最相似的 Top K 的 K 的值。	大于等于0的正整数。
	Filter	否	设置 Filter 表达式，过滤所需的数据。	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not

				（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `.withFilter("bookInfo.bookName=\"三国演义\"")` • <value>：表示要匹配的值。
	Radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= Radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= Radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= Radius 的文
	Output Fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：**

				• OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

说明：

- 输出的 Document ID 顺序与查询时配置的参数 document_ids 输入的顺序一致。
- 每一个查询结果都返回 TopK 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。相似程度则通过 L2（欧几里得距离）、IP（内积）或 COSINE（余弦相似度）计算得出的分数来衡量，输出参数 score 表示相似性计算分数。其中，L2和 IP 计算所得的分数越小，表示与搜索值越相似；而 COSINE 计算所得的分数越大，表示与搜索值越相似。

```
res: 0
res: {"id":"0001","score":0.9625580906867981,"page":21,"bookName":"三国演义","text":"话说天下大势，分久必合，合久必分。","tags":["曹操","诸葛亮","刘备"],"author":"罗贯中"}
res: {"id":"0002","score":0.7675797343254089,"text":"正大光明，混沌未分天地乱，茫茫渺渺无人间。","author":"吴承恩","tags":["孙悟空","猪八戒","唐僧"],"page":22,"bookName":"西游记"}
```

参数名	参数含义

id	Document 的 ID 信息。
vector	Document 的向量值。
score	表示查询向量与检索结果向量之间的相似性计算分数。
author、page、section	Document 其他自定义的标量字段。

单路稀疏向量相似性检索

最近更新时间：2025-07-01 10:19:42

接口定义

本接口 `fullTextSearch()` 基于稀疏向量实现全文检索，对查询词与文档内容进行匹配，并按相关度排序结果。

```
public FullTextSearchRes fullTextSearch(String database, String
collection, FullTextSearchParam param)
```

使用示例

说明：

- 如需使用稀疏向量，至少需要升级 Java SDK 至2.0.16版本，推荐使用2.1.1版本使用最新的 TerminateAfter、CutoffFrequency 等新特性，使得召回效果和性能更佳。
- 腾讯云向量数据库团队提供了一款稀疏向量工具包，旨在帮助用户高效生成稀疏向量。具体信息，请参见 工具介绍 。

```
FullTextSearchParam fullTextSearchParam =
FullTextSearchParam.newBuilder()
        .withMatch(MatchParam.newBuilder()
                .withFieldName("sparse_vector")
                .withData(bm25Encoder.encodeQueries(Arrays.asList("什么是
腾讯云向量数据库")))
                //.withCutoffFrequency(0.05)
                //.withTerminateAfter(1)
                .build())
        .withLimit(3)
        .withRetrieveVector(false)
        //.withOutputFields(Arrays.asList("id"))
        .build();
FullTextSearchRes res= client.fullTextSearch("db-test", "vector-sparse",
fullTextSearchParam);
System.out.println(JsonUtils.toJsonString(res));
int i = 0;
for (Object docs : res.getDocuments()) {
        System.out.println("\tres: " + (i++) + docs.toString());
}
```

入参描述

参数名	子参数	是否必选	参数含义	配置方法及要求
database	-	是	指定检索数据的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	-	是	指定检索数据的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
	Match	否	稀疏向量检索配置。	• **FieldName**：稀疏向量的字段名，例如：sparse_vector。 • **Data**：检索的稀疏向量列表，当仅支持输入一个稀疏向量。 • **TerminateAfter**：设置检索数量上限，即检索数量超过 TerminateAfter 值之后，提前终止检索行为，可以有效的控制速率。在数据量较大的情况下，推荐经验值：4000。 • **CutoffFrequency**：设置高频词出现的频次或频率上限，即词频大于 CutoffFrequency 的值之后，该关键词被忽略。 ○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。 ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。 **说明：** CutoffFrequency可以排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：0.1。
		否	标识是否需要返回检索结果的向量	取值如下所示： • **true**：需要。

RetrieveVector		值。	• **false**：不需要。默认为 false。
Limit	是	指定返回最相似的 Top K 的 K 的值。	大于等于0的正整数。
Filter	否	设置 Filter 表达式，过滤所需的数据。	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见Json 类型表达式 。 `.withFilter("bookInfo.bookName=\"三国演义\"")`

				• <value>：表示要匹配的值。
	Output Fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。

出参描述

```
---------------------- full text search ----------------------
    res: 0{"id":"0001","score":0.510307252407074,"text":"腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。"}
    res: 1{"id":"0005","score":0.31104958057403564,"text":"腾讯云向量数据库可以和大语言模型 LLM 配合使用。企业的私域数据在经过文本分割、向量化后，可以存储在腾讯云向量数据库中，构建起企业专属的外部知识库，从而在后续的检索任务中，为大模型提供提示信息，辅助大模型生成更加准确的答案。"}
    res: 2{"id":"0004","score":0.2857203185558319,"text":"腾讯云向量数据库（Tencent Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 "}
```

参数名	参数含义
id	Document 的 ID 信息。
sparse_vector	稀疏向量。
score	表示查询稀疏向量与检索结果向量之间的相似性计算分数。
other_scalar_field	Document 其他自定义的标量字段。

稠密向量与稀疏向量高阶混合检索

最近更新时间：2025-07-01 10:19:43

接口定义

```
public HybridSearchRes hybridSearch(String database, String collection,
HybridSearchParam param)
```

使用示例

说明：

- 如需使用稀疏向量，至少需要升级 Java SDK 至2.0.16版本，推荐使用2.1.1版本使用最新的 TerminateAfter、CutoffFrequency 等新特性，使得召回效果和性能更佳。
- 腾讯云向量数据库团队提供了一款稀疏向量工具包，旨在帮助用户高效生成稀疏向量。具体信息，请参见 工具介绍 。

稠密向量 + 稀疏向量混合检索

```
SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
HybridSearchParam hybridSearchParam = HybridSearchParam.newBuilder()

.withAnn(Arrays.asList(AnnOption.newBuilder().withFieldName("vector"
)
                .withData((Arrays.asList(0.3123, 0.43, 0.213)))
                .build()))

.withMatch(Arrays.asList(MatchOption.newBuilder().withFieldName("spa
rse_vector")
                 .withData(encoder.encodeQueries(Arrays.asList("正大
光明，忠良善果弥深")))
                 //.withData(Arrays.asList(Pair.of(21, 0.96f)))
                 .withCutoffFrequency(0.1)
                 .withTerminateAfter(4000)
                 .build()))
          // 指定 Top K 的 K 值
```

```
            .withRerank(new
WeightRerankParam(Arrays.asList("vector","sparse_vector"),
Arrays.asList(0.9, 0.1)))
            .withLimit(1)
            .withRetrieveVector(true)
            .build();

List<List<Document>> siDocs = client.hybridSearch("db-test", "book-
vector-sparse", hybridSearchParam).getDocumentsList();
int i = 0;
for (List<Document> docs : siDocs) {
    System.out.println("\tres: " + i);
    i++;
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

输出信息，如下所示。

```
res: 0
res: {"id":"0001","vector":
[0.21230000257492065,0.20999999344348907,0.21299999952316284],"spars
e_vector":[[2,0.96],[5,0.53],
[100,0.443]],"score":0.18177326023578644,"page":21,"author":"罗贯
中","tags":["曹操","诸葛亮","刘备"],"bookName":"三国演义"}
```

文本与稀疏向量混合检索

```
SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
HybridSearchParam hybridSearchParam = HybridSearchParam.newBuilder()
        .withAnn(AnnOption.newBuilder().withFieldName("text")
                .withTextData("什么是腾讯云向量数据库")
                .build())

.withMatch(MatchOption.newBuilder().withFieldName("sparse_vector")
                .withData(encoder.encodeQueries(Arrays.asList("什么是
腾讯云向量数据库")))
```

```
                .withCutoffFrequency(0.1)
                .withTerminateAfter(4000)
                .build())
        // 指定 Top K 的 K 值
        .withRerank(new
WeightRerankParam(Arrays.asList("vector","sparse_vector"),
Arrays.asList(1, 1)))
        .withLimit(2)
        .withRetrieveVector(false)
        .build();

List<Document> siDocs = client.hybridSearch("db-test", "book-emb-
sparse", hybridSearchParam).getDocuments();
int i = 0;
for (Object docs : siDocs) {
    System.out.println("\tres: " + (i++) + docs.toString());
}
```

输出信息，如下所示。

```
res: 0{"id":"0001","score":195.30913,"text":"腾讯云向量数据库（Tencent
Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检
索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。"}
res: 1{"id":"0004","score":194.4411,"text":"腾讯云向量数据库（Tencent
Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可
用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 "}
```

入参描述

参数名	子参数	是否必选	参数含义	配置方法及要求
database	-	是	指定检索数据的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

collection	_	是	指定检索数据的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
hybridSearchParam	Ann	是	设置向量检索信息	• **FieldName**：检索的字段名，支持设置为：vector 或 text。 **说明：** 其中，vector 为向向量数据的字段名，不可更改；text 为创建 Collection 时，开启 Embedding 功能自定义的原始文本字段名，请以实际定义为准，二者只能选择其中之一。 • **Data**：检索的数据列表，当前仅支持输入一条向量或原始文本。 • **Params**：检索参数，与索引类型相关。 ○ HNSW 类型：需配置参数 **ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为10。 ○ IVF 系列：需设置参数 **nprobe** ，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置。 • **Limit**：指定返回稠密向量的数量。
	Match	否	稀疏向量检索配置。	• **FieldName**：稀疏向量的字段名，例如：sparse_vector。 • **Data**：检索的稀疏向量列表，当仅支持输入一个稀疏向量。 • **Limit**：指定返回的数量。 • **TerminateAfter**：设置检索数量上限，即检索数量超过 TerminateAfter 值之后，提前终止检索行为，可以有效的控制速率。在数据量较大的情况下，推荐经验值：4000。 • **CutoffFrequency**：设置高频词出现的频次或频率上限，即词频大于 CutoffFrequency 的值之后，该关键词被忽略。

			○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。 ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。 **说明：** CutoffFrequency可以排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：0.1。
Rerank	否	检索中重排序（rerank）的配置。	参数 Method 指定 Rerank 的方法，枚举如下值： • **WeightRerankParam**：基于不同字段的加权组合来进行排序。 ○ **fieldList**：列出用于加权计算的字段列表。例如，field_list=: ['vector', 'sparse_vector']表示密集型与稀疏型加权计算。 ○ **weight**：定义了 "field_list" 中每个字段的权重，例如，weight=[0.9, 0.1]表示"vector" 字段的权重是 0.9，"sparse_vector" 字段的权重是 0.1。 • **RRFRerankParam**：一种融合多个搜索方法的排名结果，通过为每个文档分配倒数排名分数并合并这些分数来生成一个新的综合排名，从而提高搜索结果的准确性和相关度。**rrfk** 是用于计算倒数排名分数的常数，其作用是调整分数计算公式，以控制排名分数的分布。例如：`.withRerank(new RRFRerankParam(1))`。
RetrieveVector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • **true**：需要。 • **false**：不需要。默认为 false。
Limit	是	指定返回最相似的 Top K 的 K 的值。	大于等于0的正整数。

	Filter	否	设置 Filter 表达式，过滤所需的数据。	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `.withFilter("bookInfo.bookName=\"三国演义\"")` • <value>：表示要匹配的值。
	Output Fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** OutputFields 与 RetrieveVector 参数均可以配置是

				否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。

出参描述

```
res: 0
res: {"id":"0001","vector":
[0.21230000257492065,0.20999999344348907,0.21299999952316284],"sparse_ve
ctor":[[2,0.96],[5,0.53],
[100,0.443]],"score":0.18177326023578644,"page":21,"author":"罗贯
中","tags":["曹操","诸葛亮","刘备"],"bookName":"三国演义"}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
sparse_vector	稀疏向量。
score	表示查询向量与检索结果向量之间的相似性计算分数。
author、page、section	Document 其他自定义的标量字段。

版权所有：腾讯云计算（北京）有限责任公司

删除 Document

最近更新时间：2025-07-01 10:19:43

接口定义

delete() 接口用于删除指定 id（Document ID）的文档，且支持设置 Filter 表达式，删除满足 Filter 表达式的数据。

```
public AffectRes delete(String database, String collection, DeleteParam
param)
```

使用示例

> **说明：**
> 索引类型为 FLAT，在1.6及之后版本，支持删除操作，而1.6之前版本，则不支持删除操作。

```
// delete
DeleteParam deleteParam = DeleteParam.newBuilder()
        .withDocumentIds(Arrays.asList("0001", "0003"))
        .withFilter("bookName=\"三国演义\"")
        .withLimit(1)
        .build();
AffectRes deleteAffectRes = client.delete("db-test", "book-vector",
deleteParam);
System.out.println(deleteAffectRes.toString());
```

入参描述

参数名	子参数	是否必选	参数含义	配置方法
database	-	是	指定删除数据的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

collection	_	是	指定删除数据的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
deleteParam	DocumentIds	否	表示要删除的 Document 的 ID。	• ID 长度限制为[1,128]。 • 批量删除，数据元素最大值为20。
	Filter	否	设置 Filter 表达式。	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `.withFilter("bookInfo.bookNa`

				`me=\"三国演义\"")` • <value>：表示要匹配的值。
	limit	否	控制删除文档的数量。	若不设置该参数，最大删除16384条数据。 • 数据类型：整型。 • 取值范围：[1,16384]

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	删除的文档数量。如果该参数返回的值为0，说明删除无效。

更新数据

最近更新时间：2025-07-01 10:19:43

接口定义

update() 接口用于对通过主键（Document ID）与 Filter 表达式过滤检索 Document，对 Document 的部分字段进行更新，同时支持新增字段。

```
public AffectRes update(String database, String collection, UpdateParam
param, Document document)
```

使用示例

集合未配置 Embedding 参数，则直接更新向量数据。如下示例，在集合 **book-vector** 中，基于 upsert() 插入的向量数据，通过 documentIds 与 filter 表达式，过滤出需更新的 Document，更新其 vectors 字段的向量数据，并更新 page 字段值为30，新增字段 test_new_field。

写入文本更新向量数据

实例在创建 Collection 时，已配置 Embedding 模型，通过 upsert() 写入原始文本，则可输入文本信息，通过 Embedding 将数据向量化更新向量数据。如下示例，基于 upsert() 插入的原始文本，使用 **update** 接口，通过 documentIds 与 filter 表达式过滤数据，更新其 text 字段的文本信息，更新 page 字段值为100，并新增字段 test_new_field。

```
//设置 更新字段
Document updateDoc = Document.newBuilder()
        .addDocField(new DocField("page", 100))
        .addDocField(new DocField("text", "天下大势，分久必合"))
        .addDocField(new DocField("test-new-field", 50))
        .build();
// 过滤更新的文档
UpdateParam updateParam = UpdateParam
        .newBuilder()
        .addAllDocumentId(Arrays.asList("0001", "0003"))
        .withFilter(new Filter("bookName=\"三国演义\"")
        .build();
// update
AffectRes affectRes = client.update("db-test", "book-emb",
updateParam, updateDoc);
```

```
System.out.println("\tres: " + affectRes);
```

更新向量数据

集合未配置 Embedding 参数，则直接更新向量数据。如下示例，在集合 book-vector 中，基于 upsert() 插入的向量数据，通过 documentIds 与 filter 表达式，过滤出需更新的 Document，更新其 vectors 字段的向量数据，并更新 page 字段值为100，新增字段 test_new_field。

```
//设置更新字段
Document updateDoc = Document.newBuilder()
        .addDocField(new DocField("page", 100))
        .withVector(Arrays.asList(0.2123, 0.29, 0.213))
        .addDocField(new DocField("test-new-field", 50))
        .build();
// 过滤更新的文档
UpdateParam updateParam = UpdateParam
        .newBuilder()
        .addAllDocumentId(Arrays.asList("0001", "0003"))
        .withFilter(new Filter("bookName=\"三国演义\""))
        .build();
// update
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-vector";
List<Document> qdos = client.update(DBNAME, COLL_NAME, updateParam,
updateDoc);
for (Document doc : qdos) {
    System.out.println(doc.toString());
}
```

入参描述

参数	参数含义	子参数	是否必选	子参数含义
database	指定删除数据	-	是	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。

	的数据库名。			• 长度要求：[1,128]。
collection	指定删除数据的集合名。	_	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
updateParam	过滤需更新的文档	DocumentIds	是	设置需更新的文档 ID。每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。
		Filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `.withFilter("bookInfo.bookName=\"三国演义\"")` • <value>：表示要匹配的值。

updateDoc	设置需更新的字段	vector	否	更新 vector 字段的向量数据。 **说明：** 如果 Collection 在创建时，未配置 Embedding 参数，或者该实例并未开通 Embedding 功能，则只能配置该参数，输入向量数据，更新数据。
		text	否	Embedding 模型输入文本的字段名。该字段在创建 Collection 时定义。本示例为 text 。 • **string**：字符型。 • **float**：浮点型数据。 **说明：** 如果 Collection 在创建时，已配置 Embedding 参数，则只能配置该参数，依据输入的文本信息更新向量数据，并将文本字段与向量数据更新存于数据库。
		old_field	否	当前已存在的字段，更新字段对应的数据。 • 类型：string • 字符长度要求：[1,256]。
		new_field	否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='operation success'}
```

参数名	参数含义
affectedCount	更新的文档数量。如果该参数返回的值为0，说明更新无效。

相关说明

更新字段数据之后，可使用 query() 确认更新的字段或新增的字段是否已生效。

查询文档数量

最近更新时间：2025-07-01 10:19:43

接口定义

count() 接口查询集合中文档数量。

```
public BaseRes count(String database, String collection, CountQueryParam
countQueryParam)
```

使用示例

```
BaseRes res = client.count(
        "db-test",
        "book-vector",
        CountQueryParam.newBuilder()
                .withFilter("bookName=\"三国演义\"")
                .build());
System.out.println("count document: "+ res.getCount());
```

入参描述

参数	是否必选	参数含义	参数配置
database	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	是	指定检索的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

CountQueryParam	否	使用创建 Collection 时指定的 Filter 索引的字段给设置过滤表达式。	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如，expired _time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `.withFilter("bookInfo.bookName=\"三国演义\"")` • <value>：表示要匹配的值。

出参描述

检索结果，如下所示。

```
count document: 3
```

参数名	参数含义
count	集合中的文档数量。

上传文件

最近更新时间：2025-05-26 19:31:32

接口定义

UploadFile() 接口用于上传文件于 Base 类向量数据库。文件上传后，其内容将被直接拆分并写入指定的 Collection，而非 CollectionView。

> **说明：**
> 当前支持导入 Collection 的文件类型包含： Markdown、PDF、Word、PPT。其中，仅 PDF 类型支持解析图片信息。

```
public void UploadFile(String database,
                       String collection,
                       UploadFileParam collectionLoadAndSplitTextParam,
                       Map<String, Object> metaDataMap)
```

约束限制

- 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
- 该接口当前不支持使用别名获取文件的上传路径与授权。

> **注意：**
> 在创建存储文件内容的 Collection 时，索引字段的设计需遵循以下限制与建议，否则可能导致上传文件失败。
>
> - 文件名字段（file_name）：该字段**必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。字段名可以自定义，在上传文件时，需通过参数 FieldMappings 的 filename 映射自定义的字段名。
> - 文件内容字段（text）：该字段用于存储知识点的原始文本内容。由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。字段名可以自定义，但需通过参数 FieldMappings 的 text 映射自定义的字段名。
> - 文件图片信息字段（image_list）：该字段存放 PDF 文件中图片的 Key 列表。在原始检索出的文本块中，图片位置将以 {key} 的形式进行占位。通过调用接口 GetImageUrl，可以获取图片 URL 地址列表，这些地址与 Key 一一对应，从而实现将原始 PDF 文档恢复并展示为 HTML 格式。
>   - **不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。

- 字段名可以自定义，但需要在上传文件时通过参数 FieldMappings 的 imageList 映射到自定义的字段名。

使用示例

说明：
如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
// 配置文件 matadata 信息，示例中指定了author 与 tags
Map<String, Object> metaDataMap = new HashMap<>();
metaDataMap.put("author", "Tencent");
metaDataMap.put("tags", Arrays.asList("Embedding", "向量", "AI"));
// 配置文件与数据库中存储的映射字段
Map<String,String> columnMap = new HashMap<>();
columnMap.put("filename", "file_name");
columnMap.put("text", "text");
columnMap.put("imageList", "image_list");
columnMap.put("chunkNum", "chunk_num");
columnMap.put("sectionNum", "section_num");
// 上传文件
// 1. LocalFilePath: 指定当前文件在客户端的路径
// 2. FileName:指定文件名
// 3. SplitterProcess: 配置文件拆分规则
// 4. ParsingProcess: 指定文件解析方式
// 5. metadata: 定义文件 meta 信息字段，示例中指定了author 与 tags
// 6. FieldMappings: 文件与数据库中存储的映射字段
// 7. EmbeddingModel: Embedding 模型
UploadFileParam param = UploadFileParam.newBuilder()
        .withLocalFilePath("/data/home/********/腾讯云向量数据库.pdf")

.withSplitterProcess(SplitterPreprocessParams.newBuilder().withAppendKey
wordsToChunkEnum(true).Build())
        // parsingProcess is used for parsing pdf file by vision model

.withParsingProcess(ParsingProcessParam.newBuilder().withParsingType(Par
singTypeEnum.VisionModel).build())
        .withFileName("腾讯云向量数据库.pdf")
        .withFieldMappings(columnMap)
        .withEmbeddingModel("bge-base-zh")
        .Build();
client.UploadFile("db-test", "coll-file-test", param, metaDataMap);
```

入参描述

参数名	子参数（一）	子参数（二）	是否必选	参数含义
database		-	是	指定 Database 名称。
collection		-	是	指定 Collection 的名称。
UploadFileParam	LocalFilePath	-	是	本地上传文件路径。
	FileName	-	否	指定上传的文件名。
	SplitterProcess	AppendTitleToChunkEnum	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： false：不追加。默认值为 false。 • true：将段落 Title 追加到切分后的段落。
		AppendKeywordsToChunkEnum	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。默认值为 true。
		ChunkSplitter	否	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
	ParsingProcess	ParsingType	否	指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

	Embedding Model	–	是	指定使用的 Embedding 模型的名称。您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，参见 Embedding 介绍 。取值如下所示： • **bge-large-zh-v1.5**：适用中文，1024维，推荐使用。 • **bge-base-zh-v1.5**：适用中文，768维。 • **bge-large-zh**：适用中文，1024维。 • **bge-base-zh**：适用中文，768维。 • **m3e-base**：适用中文，768维。 • **e5-large-v2**：适用英文，1024维。 • **text2vec-large-chinese**：适用中文，1024维。 • **multilingual-e5-base**：适用于多种语言类型，768维。 • **BAAI/bge-m3**：适用于多种语言类型，1024维。
	FieldMappings	–	是	指定文件字段在数据库中映射关系。 • **filename**：指定文件名映射的字段。例如："filename": "file_name"，其中，file_name 为用户自定义字段，以在数据库中存储文件名数据。 **注意：** **必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。 • **text**：指定文件内容文本映射的字段，存储知识点的原始文本内容。例如："text": "text"，其中，text 为用户自定义字段，以在数据库中存储文件的原始文本内容。 **说明：** 由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过

output_fields 参数指定该字段即可返回原始文本。

- **imageList**：指定图片信息存放的字段。例如："imageList": "image_list"，其中，image_list 为用户自定义的字段，以在数据库中存储图片 Key 列表。

  说明：
  不建议为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。

- **chunkNum**：指定数据库中存放文件切分语块序号的字段。例如："chunkNum": "chunk_num"，其中，chunk_num 为用户自定义字段，其值为唯一标识语块片段的整数。

  说明：
  在创建 Collection 时若为该字段（如 chunk_num）创建索引，则查询文件内容时可通过 Filter 表达式指定语块范围，快速定位目标文件内容。

- **sectionNum**：指定数据库中用于存储文件段落序号的字段名。例如："sectionNum": "section_num"，其中，section_num 为用户自定义的字段，其值为唯一标识段落的整数。

  说明：
  在创建 Collection 时若为该字段（如 section_num）创建索引，则查询文件内容时可通过 Filter 表达式指定段落范围内的语块，快速定位目标文件内容。

	metaData Map	–	否	文件的 Metadata 元数据信息，可自定义扩展字段。例如：author、tags 等。 • 上传文件时，可为创建 Collection 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。

获取文件图片 URL

最近更新时间：2025-03-04 20:25:12

接口定义

GetImageUrl() 接口用于在使用 AI 套件上传 PDF 类型的文件后，获取图片 URL 地址列表。

> **说明：**
> 当在数据库的 Collection 中上传 PDF 文件时，系统会提取 PDF 中的图片信息，并将这些信息以 Key 列表的形式存储。在检索出的原始文本块中，图片的位置会以 {key} 的形式进行占位。通过调用接口 **GetImageUrl()**，可以获取到与这些 Key 一一对应的图片 URL 地址列表。利用这种映射关系，可以将原始 PDF 文档的内容恢复并以 HTML 格式进行展示。

```
public GetImageUrlRes GetImageUrl(String database, String collection,
GetImageUrlParam param)
```

使用示例

```
GetImageUrlRes getImageUrlRes = client.GetImageUrl("db-test", "coll-
file-test",
        GetImageUrlParam.newBuilder().setFileName("tcvdb.pdf")
                // 指定具体的文件 ID
                .setDocumentIds(Arrays.asList("1340988975862403135"))
                .build());
System.out.println("get image url res:");
System.out.println(JsonUtils.toJsonString(getImageUrlRes.getImages()));
```

入参描述

参数名	子参数	是否必选	参数含义	
database	-	是	指定查询的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线 _、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

collection	-	是	指定查询的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
GetImageUrlParam	setDocumentIds	是	表示要查询的文档的所有 ID。	• 每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。 • 文件上传成功之后，可使用 query 查询文件的 ID 信息。
	setFileName	是	指定文件名称。	-

输出信息

```
[
  [
    {
      "documentId": "1340988975862403135",
      "imageName": "ea97227d-1b12-4d35-8b67-************.png",
      "imageUrl": "https://vecterdb************.cos.ap-
guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-
test-6/****************************************"
    },
    {
      "documentId": "1340988975862403135",
      "imageName": "a5daca6f-a27e-4b35-89f5-************.png",
      "imageUrl": "https://vecterdb************.cos.ap-
guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-
test-6/****************************************"
    }
  ]
]
```

参数名	参数含义
documentId	Document 的 ID 信息。
imageName	图片名。
imageUrl	图片 URL。

DocumentSet 操作
上传文件

最近更新时间：2024-12-05 20:37:02

接口定义

loadAndSplitText() 接口用于上传文件至 AI 类向量数据库。

```
public void loadAndSplitText(LoadAndSplitTextParam
loadAndSplitTextParam, Map<String, Object> metaDataMap )
```

约束限制

- 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
- 该接口当前不支持使用别名替换集合视图上传文件。

使用示例

> **说明：**
> 如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
// link database， client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-files");
// LocalFilePath 配置上传文件的本地路径
// DocumentSetName 指定文件存储于数据库的名称
LoadAndSplitTextParam param = LoadAndSplitTextParam.newBuilder()
                .withLocalFilePath("/tmp/腾讯云向量数据库.pdf")
                .withDocumentSetName("腾讯云向量数据库.pdf")

.withSplitterProcess(SplitterPreprocessParams.newBuilder()
                    .withAppendKeywordsToChunkEnum(true)
                    .withAppendTitleToChunkEnum(false)
                     // ChunkSplitter 参数以正则表达式的方式配置文档拆分方式，
如下： \n{2,} 代表以两个及以上的换行进行拆分，常用在QA对文件拆分中。
                     //.withChunkSplitter("\n{2,}")
                    .Build())
```

```
.withParsingProcess(ParsingProcessParam.newBuilder().withParsingType(Par
singTypeEnum.VisionModel).build())
                .Build();
// 配置文件 Meatdata 标量字段的值
Map<String, Object> metaDataMap = new HashMap<>();
metaDataMap.put("author", "Tencent");
metaDataMap.put("tags", Arrays.asList("Embedding","向量","AI"));
// 调用 loadAndSplitText() 上传文件
collection.loadAndSplitText(param, metaDataMap);
```

入参描述

说明：
如果创建 CollectionView 和上传文件时同时配置了 **ChunkSplitter** 参数，将使用上传文件时配置的文件拆分方式处理。

参数名	子参数	是否必选	参数含义
`LocalFilePath`	-	是	本地上传文件路径。
`DocumentSetName`	-	否	存储在向量数据库中的文件名。若不设置该参数，则默认使用 **LocalFilePath** 中的文件名。
SplitterProcess	**AppendTitleToChunkEnum**	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **true**：将段落 Title 追加到切分后的段落。 • **false**：不追加。默认值为 false。
	AppendKeywordsToChunkEnum	否	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **true**：将全文的 **keywords** 追加到切分后的段落。默认值为 true。 • **false**：不追加。
	ChunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：

			`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
`ParsingProcess`	ParsingType	否	指定 PDF 类型文件的解析方式，取值如下所示： • ParsingTypeEnum.**VisionModel**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • singTypeEnum.**AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
`metaDataMap`	-	否	文件的 Metadata 元数据信息，可自定义扩展字段。例如：author、tags 等。 • 上传文件时，可为创建 CollectionView 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。

输出说明

> **说明：**
> **loadAndSplitText()** 执行之后，如果抛出异常，说明上传文件失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明执行成功，可使用 getFile() 查看上传的文件内容。

获取文件内容

最近更新时间：2025-05-12 18:23:12

接口定义

getFile() 该接口用于获取存储于 AI 类向量数据库文件的完整内容，以及系统分配的文件 ID、关键字、文件大小、预处理进度与状态等信息。

```
public DocumentFileContent getFile(String fileName, String fileId )
```

使用示例

说明：

- 支持根据文件名获取文件内容。
- 支持根据文件 ID 获取文件内容。

注意：

使用 **getFile()** 获取单个文档的 chunk 数量上限为1000，若超过此限制，接口将返回错误提示信息，并提示可使用 获取 Chunks 接口，结合 offset（起始位置）和 limit（每页数量）参数分批获取文档内容。

使用文件名获取文件内容

```
// link database， client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");

String fileId="";
String fileName = "腾讯云向量数据库.pdf";
System.out.println(collection.getFile(fileName,fileId).toString());
```

使用文件 ID 获取文件内容

```
// link database， client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
String fileId="11822484427********";
String fileName = "";
System.out.println(collection.getFile(fileName,fileId).toString());
```

入参请求

参数名	是否必选	参数含义	获取方式
`fileId`	否	文件上传在数据库之后，系统分配的文件 ID	第一次使用可使用文件名获取文件 ID。
`fileName`	否	文件名	-

出参描述


```
{
  "documentSetName": "腾讯云向量数据库.pdf",
  "documentSetId": "1182248442786480128",
  "text": "本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。\n## 关键概念\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。。\n### 什么是向量？\n向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。\n### 什么是非结构化数据？\n非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。\n### 什么是 AI 中的向量表示？\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量
```

表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。。\n### 什么是向量检索？\n向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。\n## 为什么是腾讯云向量数据库？\n腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户，在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。\n## 腾讯云向量数据库应用示例有哪些？\n腾讯云向量数据库可进行高性能向量存储和检索，主要适用于以下应用场景。\n- [大规模知识库]：企业的私域数据存储在向量数据库中可构建外部知识库，帮助企业更好地管理和利用自己的数据资源。\n- [推荐系统]：向量数据库会基于用户特征进行向量存储与检索，最终筛选用户可能感兴趣的物品推荐给用户。\n- [问答系统]：向量数据库会基于问题信息进行向量存储与检索，并返回最相关的问题与对应的答案。\n- [文本/图像检索]：向量数据库对输入的图像和文本信息进行向量存储与检索，会找到最匹配输入信息的文本或图像结果。\n## 腾讯云向量数据库支持哪些索引类型？\n索引是数据的组织单位。您必须先声明索引类型和相似性度量，然后才能搜索或查询向量数据。目前，腾讯云向量数据库支持如下类型。具体信息，请参见 [Index]。\n- FLAT 索引：向量会以浮点型的方式进行存储，不做任何压缩处理。搜索向量会遍历所有向量与目标向量进行比较。\n- HNSW 索引：全称为 Hierarchical Navigable Small World， 是基于图的索引，适合对搜索效率要求较高的场景。\n- IVF 系列：全称为 Inverted File，IVF 系列索引的核心思想是将高维空间划分为多个聚类，并为每个聚类构建一个倒排文件。适用于高维向量数据的快速检索。\n## 腾讯云向量数据库支持哪些相似度计算方法？\n在 VectorDB 中，相似度度量用于衡量向量之间的相似度。选择良好的距离度量有助于显著提高分类和聚类性能。根据输入数据形式，选择特定的相似性度量方法，获得数据库最佳性能。\n**相似性计算方法** | **方法说明** \n:-: | :-: \n内积（IP） | 全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 | \n欧式距离（L2） | 全称为 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 | \n余弦相似度（COSINE） | 余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度。所得值越大越与搜索值相似。 | \n## 腾讯云向量数据库是如何设计的？\n- **部署架构**：腾讯云向量数据库采用分布式部署架构，每个节点相互通信和协调，实现数据存储与检索。客户端请求通过 **Load Balancer** 分发到各节点上。\n- **逻辑架构**：实例是腾讯云中独立运行的数据库环境，是用户购买向量数据库服务的基本单位。腾讯云向量数据库数据存储的一个实例集群中包括[Database]、[Collection]、[Document]三个逻辑层级。其中，一个实例可以包含很多个 Database，一个 Database 可以包含多个 Collection，一个 Collection 可以包含多个 Document。\n- **数据安全**：腾讯云向量数据库的多副本设计、多可用区分布节点、API 密钥认证，并运行于私有网络环境，通过安全组控制访问来源，CAM 账户授权等多方面保护向量数据的完整性和隐私。。\n- **鉴权方式**：腾讯云向量数据库使用账号（account）和 API 密钥（api_key）的组合进行鉴权，以验证用户身份并授权其访问。\n- **连接方式**：腾讯云向量数据库支持通过 HTTP 协议进行数据写入和查询等操作。\n- **检索方法**：腾讯云向量数据库支持通过精确检索、相似度检索、混合检索的方法。\n- 精确查询：基于标量（指一个单独的数值，例如文本字段、数值字段或日期字段，区别于向量等多维数据结构）字段精确查找数据的方式。。\n- 相似度检索：基于向量相似度计算的检索方式，通过计算向量之间的相似度来找到与查询向量最相似的文档。。\n- 混合检索：基于标量字段和向量字段，搭配自定义的标量字段的

```
Filter 表达式进行检索的方式。\n## 如何快速体验向量数据库？\n腾讯云向量数据库目前是公测阶段。免费测试版实例每个账号仅限申领1个，高可用版与单机版实例免费试用时长1个月，到期后可 [提交工单](https://console.cloud.tencent.com/workorder/category) 进行续期；若一个月内未使用实例，平台将自动回收。\n**序号** | **步骤描述** | **具体操作** \n:-: | :-: | :-: \n1 | 申请腾讯云账号并认证。 | - 如需注册腾讯云账号：请单击 [注册腾讯云账号](https://cloud.tencent.com/register?s_url=https%3A%2F%2Fcloud.tencent.com%2F)。 | - 如需完成实名认证：请单击 [实名认证](https://console.cloud.tencent.com/developer)。 | \n2 | 了解向量数据库所支持的规格与类型。 | 预估数据规模，选择合适的类型与规格。 | \n3 | 确定向量数据库所部署的地域。 | 当前支持的地域信息，请参见 [发布地域]。 | \n4 | 规划数据库实例的私有网络与安全组。 | 具体操作，请参见 [创建私有网络](https://cloud.tencent.com/document/product/215/36515)与 [创建安全组]，并同时设置安全组入站规则。 | \n5 | 购买实例。 | 具体操作，请参见 [新建数据库实例]。购买实例中，直接选择上一步已准备的私有网络与安全组。 | \n6 | 申请与腾讯云向量数据库在同一地域同一个 VPC 内的 Linux 云服务器 CVM。 | 具体操作，请参见 [快速配置 Linux 云服务器](https://cloud.tencent.com/document/product/213/2936)。 | \n7 | 连接并操作向量数据库。 | [连接并写入数据库] ，本文使用 [API 接口] 从创建 DataBase 到插入数据、检索数据到最终删除数据，均给出了具体的使用示例。您可以简单并快速体验向量数据库。 | \n8 | 管理向量数据库实例 | 您可以体验通过控制台直接管理实例，查看实例状态或销毁实例。具体操作，请参见 [管理实例]。 | \n9 | 智能运维 | 您可以在控制台查看监控数据库实例的各项指标。目前仅支持对节点信息的监控，后续还会支持更丰富的监控项目。 | \n## 开发者工具\n**开发者工具** | **API** \n:-: | :-: \nHTTP API | [API 接口](https://cloud.tencent.com/document/product/1709/98666) | \nPython SDK | [Python SDK Demo](https://cloud.tencent.com/document/product/1709/96724) | \nJava SDK | [Java SDK Demo](https://cloud.tencent.com/document/product/1709/97768) | \n",
  "documentSetInfo": {
    "textLength": 5526,
    "byteLength": 12886,
    "indexedProgress": 100,
    "indexedStatus": "Ready",
    "createTime": "2023-12-07 17:13:14",
    "lastUpdateTime": "2023-12-07 17:13:14",
    "indexedErrorMsg": null,
    "keywords": "向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似"
  },
  "splitterPreprocess": {
      "appendTitleToChunk": true,
      "appendKeywordsToChunk": true
    },
  "parsingProcess":{
      "parsingType":"VisionModelParsing"
    },
```

```
  "docFields": [
    {
      "name": "textPrefix",
      "value": "本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似",
      "fieldType": "string",
      "stringValue": "本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似"
    },
    {
      "name": "author",
      "value": "Tencent",
      "fieldType": "string",
      "stringValue": "Tencent"
    },
    {
      "name": "tags",
      "value": [
        "Embedding",
        "向量",
        "AI"
      ],
      "fieldType": "array",
      "stringValue": "[Embedding, 向量, AI]"
    }
  ]
}
```


参数名	子参数	参数含义
documnetSetId	-	文件 ID。
documnetSetName	-	文件名。

textPrefix	-	文件内容前200个字符。
text	-	文件完整内容。
documentSetInfo	textLength	文件的字符数。
	byteLength	文件的字节数。
	indexedProgress	文件被预处理、Embedding 向量化的进度。
	indexedStatus	文件预处理、Embedding 向量化的状态。 • **New**：等待解析。 • **Loading**：文件解析中。 • **Failure**：文件解析、写入出错。 • **Ready**：文件解析、写入完成。
	createTime	文件创建时间。
	lastUpdateTime	文件最后更新时间。
	keywords	文件关键字。
splitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将段落 Title 追加到切分后的段落。
	appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 **keywords** 追加到切分后的段落。
parsingProcess	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
author	-	自定义的文件 Metadata 信息的字段。显示创建 CollectionView 时设置为 Filter 索引的字段，同时显示上传文件时或使用 update 新增的字段，但新增的字段不会构建索引。 **说明：**

		显示创建 CollectionView 时设置为 Filter 索引的字段，同时显示上传文件时或使用 update 新增的字段，但新增的字段不会构建索引。

获取 Chunks

最近更新时间：2025-05-26 11:14:52

接口定义

getChunks() 接口用于获取文件切分后的语块。

> **说明：**
> **Chunk 指语块，较长文本在处理时会切分为多个语块，以便于向量化和更高效地检索，多个 Chunk 组成一个 DocumentSet。**

```
public GetChunksRes getChunks(
  String documentSetId,
  String documentSetName,
  Integer limit,
  Integer offset
)
```

使用示例

> **说明：**
> - **支持指定具体的文件名获取文件切分后的语块。**
> - **支持指定具体的 DocumentSet ID 获取文件切分后的语块。**

```
// link database, client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-files");
System.out.println(JsonUtils.toJsonString(collection.getChunks("腾讯云向量数据库.pdf")));
```

入参描述

参数名	是否必选	参数含义
Document	否	文件上传在数据库之后，系统分配的文件 ID。 **说明：**

SetId		DocumentSetId 与 documentSetName 二者必须配置其中之一。
DocumentSetName	否	文件名。
limit	否	每页返回的 Chunks 数量。 • 数据类型：uint 64。 • 默认值：10。 • 取值范围：[1,1000]。
offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 limit 整数倍。 • 计算公式：offset = limit * (page-1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 - 1) = 10，表示从查询结果的第11条记录开始返回数据。

出参描述


```
{
    "code":0,
    "msg":"Operation success",
    "count":10,
    "requestId":"ade87e83c8d0a61790139728d0db4007",
    "documentSetId":"1192057501114437632",
    "documentSetName":"腾讯云向量数据库.pdf",
    "chunks":[
        {
            "text":"本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n",
            "endPos":122,
            "startPos":0
        },
        {
            "text":"## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。\n",
            "endPos":313,
```

```
            "startPos":122
        },
        {
            "text":"## 关键概念\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。\n",
            "endPos":441,
            "startPos":313
        },
        {
            "text":"### 什么是向量？\n向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。\n",
            "endPos":508,
            "startPos":441
        },
        {
            "text":"### 什么是非结构化数据？\n非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。\n",
            "endPos":585,
            "startPos":508
        },
        {
            "text":"### 什么是 AI 中的向量表示？\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。\n",
            "endPos":784,
            "startPos":585
        },
        {
            "text":"### 什么是向量检索？\n向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。\n",
            "endPos":876,
            "startPos":784
        },
        {
            "text":"## 为什么是腾讯云向量数据库？\n腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。\n",
            "endPos":1037,
            "startPos":876
        },
```

```
        {
            "text":"## 腾讯云向量数据库应用示例有哪些？\n腾讯云向量数据库可进行高性能向量存储和检索，主要适用于以下应用场景。\n- [大规模知识库]：企业的私域数据存储在向量数据库中可构建外部知识库，帮助企业更好地管理和利用自己的数据资源。\n- [推荐系统]：向量数据库会基于用户特征进行向量存储与检索，最终筛选用户可能感兴趣的物品推荐给用户。",
            "endPos":1302,
            "startPos":1037
        },
        {
            "text":"\n- [问答系统]：向量数据库会基于问题信息进行向量存储与检索，并返回最相关的问题与对应的答案。\n- [文本/图像检索]：向量数据库对输入的图像和文本信息进行向量存储与检索，会找到最匹配输入信息的文本或图像结果。\n",
            "endPos":1511,
            "startPos":1302
        }
    ]
}
```


参数名	参数含义
text	获取的 Chunks 内容。
startPos	每个 Chunks 在文件中偏移的起始位置。
endPos	每个 Chunks 在文件中偏移的结束位置。

版权所有：腾讯云计算（北京）有限责任公司

精确查询文件

最近更新时间：2024-12-05 20:37:02

功能介绍

query() 用于精确查找与查询条件完全匹配的文件，可获取文件长度、向量化的进度与状态等，不包括文件内容。

```
public List<DocumentSet> query(CollectionViewQueryParam param)
```

使用示例

说明：

- 支持指定具体的文件名查找文件，或搭配文件 Metadata 信息对应字段的 Filter 表达式查询文件信息。
- 支持指定具体的 DocumentSet ID 查找文件，或搭配文件 Metadata 信息对应字段的 Filter 表达式查询文件信息。
- 支持指定查询起始位置 offset 和返回数量 limit，查找指定范围的文件信息。
- 支持根据文件 Metadata 信息对应字段 Filter 表达式，过滤需查找的文件。

使用文件名搭配 Filter 查询文件

根据存储于向量数据库的文件名，搭配标量字段 author 与 tags 的 Filter 表达式一并过滤文件。

```
// link database， client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 设置查询要求
// Limit 设置每页返回的 DocumentSet 数量，默认值为 10
// Filter 设置 author 与 tags 字段的 Filter 表达式，过滤文件
// DocumentSetNames 指定需要查找的文件名，可批量设置
// OutputFields 配置需输出的字段，不配置则输出所有字段
CollectionViewQueryParam queryParam =
CollectionViewQueryParam.newBuilder()
        .withLimit(2)
        .withFilter(new Filter(Filter.in("author",
Arrays.asList("Tencent","tencent"))).
```

```
                        and(Filter.include("tags",
Arrays.asList("AI","Embedding"))))
        .withDocumentSetNames(Arrays.asList("腾讯云向量数据库.pdf"))
        .withOutputFields(Arrays.asList("textPrefix", "author",
"tags"))
        .build();
// 根据查询要求查找文件
List<DocumentSet> qdos = collection.query(queryParam);
// 输出查询结果
for (DocumentSet doc : qdos) {
    System.out.println("\tres: " + doc.toString());
}
```

使用范围查询文件

文件上传于向量数据库之后，可以使用 **limit** 与 **offset** 参数，设定查询的范围来查询文件信息。

```
// link database, client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 设置查询要求
// Limit 设置每页返回的 DocumentSet 数量，默认值为 10
// Offset 设置分页偏移量，用于控制分页查询返回结果的起始位置
CollectionViewQueryParam queryParam =
CollectionViewQueryParam.newBuilder()
        .withLimit(2)
        .withOffset(0)
        .build();
// 根据查询要求查找文件
List<DocumentSet> qdos = collection.query(queryParam);
// 输出查询结果
for (DocumentSet doc : qdos) {
    System.out.println("\tres: " + doc.toString());
}
```

查询结果，如下所示。

```
{
```

```
  'documentSetId': '11800467415********',
  'documentSetName': '腾讯云向量数据库.pdf',
  'textPrefix': '本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似',
  'documentSetInfo': {
    'textLength': 5526,
    'byteLength': 12886,
    'indexedProgress': 100,
    'indexedStatus': 'Ready',
    'createTime': '2023-12-01 15:24:27',
    'lastUpdateTime': '2023-12-01 15:24:27',
    'keywords': '向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似'
  },
  'author': 'Tencent',
  'tag': [
    '向量',
    'Embedding',
    'AI'
  ]
}
```

使用文件 ID 查询文件

文件上传于向量数据库之后，系统会自动分配文件 ID，获取文件 ID 信息之后可通过文件 ID 批量查询文件信息。

```
// link database, client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-files");
// 设置查询要求
// Limit 设置每页返回的 DocumentSet 数量，默认值为 10
// Filter 设置 author 与 tags 字段的 Filter 表达式，过滤文件
// DocumentSetIds 指定需要查找的文件ID，可批量设置
// OutputFields 配置需输出的字段，不配置则输出所有字段
```

```
CollectionViewQueryParam queryParam =
CollectionViewQueryParam.newBuilder()
        .withLimit(2)
        .withFilter(new Filter(Filter.in("author",
Arrays.asList("Tencent","tencent"))).
                        and(Filter.include("tags",
Arrays.asList("AI","Embedding"))))
        .witDocumentSetIds(Arrays.asList("11793516237********"))
        .withOutputFields(Arrays.asList("textPrefix", "author",
"tags"))
        .build();
List<DocumentSet> qdos = collection.query(queryParam);
for (DocumentSet doc : qdos) {
    System.out.println("\tres: " + doc.toString());
}
```

入参描述

子参数	是否必选	配置方法及要求
DocumentSetNames	否	表示要查询的文档的名称，支持批量查询，数组元素范围[1,20]。
DocumentSetIds	否	表示要查询的文档的所有 ID，支持批量查询，数组元素范围[1,20]。

Filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见混合检索。其中：
		• <field_name>：表示要过滤的字段名。
		• <operator>：表示要使用的运算符。
		  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
		  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。
		  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
		• <value>：表示要匹配的值。

		示例：`Filter('author="jerry"').And('page>20')`。
Limit	是	每页返回的 DocumentSet 数量。 • 数据类型：uint 64。 • 默认值：10。 • 取值范围：[1,16384]。 **⚠ 注意：** • 若不配置任何查询条件，即 `doc_list = coll_view.query()`，则默认返回 10 个 DocumentSet。 • 若查询条件仅配置 Filter 表达式，不配置 limit，则默认返回 10 条 DocumentSet。 • 若查询条件仅设置 document_set_name 或 document_set_id，则可不配置 limit 参数，默认返回 10 条数据。
Offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 limit 整数倍。 • 计算公式：offset = limit * (page-1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 - 1) = 10，表示从查询结果的第11条记录开始返回数据。
OutputFields	否	以数组形式配置需返回的字段。

出参描述


```
res: {"documentSetId":"11920675605150474 24","documentSetName":"腾讯云向量数据库.pdf","textPrefix":"本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似","documentSetInfo":"{\"textLength\":5526,\"byteLength\":12886,\"indexedProgress\":100,\"indexedStatus\":\"Ready\",\"createTime\":\"2024-01-03 19:30:53\",\"lastUpdateTime\":\"2024-01-03 19:30:59\",\"keywords\":\"向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似\"}","splitterPreprocess":"
```

{\"appendTitleToChunk\":false,\"appendKeywordsToChunk\":true}","parsingProcess":{"parsingType":"VisionModelParsing"},"author":"Tencent","tags":"[Embedding, 向量, AI]"}

参数名	子参数	参数含义
documnetSetId	-	文件 ID。
documnetSetName	-	文件名。
textPrefix	-	文件内容前 200个字符。
documentSetInfo	textLength	文件的字符数。
	byteLength	文件的字节数。
	indexedProgress	文件被预处理、Embedding 向量化的进度。
	indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。
	indexedErrorMsg	文件解析、写入错误描述信息。 **说明：** 当 IndexedStatus 为 Failure 时，返回 indexedErrorMsg 信息。
	createTime	文件创建时间。
	lastUpdateTime	文件最后更新时间。
	keywords	文件关键字。
splitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • false：不追加。 • true：将段落 Title 追加到切分后的段落。

	appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
parsingProcess	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
author、tags	-	自定义的文件 Metadata 信息字段。 **说明：** 显示创建 CollectionView 时设置为 Filter 索引的字段，同时显示上传文件时或使用 update 新增的字段，但新增的字段不会构建索引。

文件内容相似性检索

最近更新时间：2025-07-01 10:19:43

接口定义

search() 接口用于在指定的文件内，查找与给定文本信息相似的 Top K 条文本信息。

```
public List<SearchContentInfo> search(SearchByContentsParam param)
```

使用示例

说明：

- 支持指定文件名称检索最相似的文本信息。
- 支持文件名搭配文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。
- 支持仅使用文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。

根据文件名搭配 Filter 检索相似数据

如下示例，在文件 `腾讯云向量数据库.pdf` 中，检索与 `什么是向量` 相似的文本信息，并使用标量字段 author 与 tags 的 Filter 表达式一并过滤文件。

```
// link database， client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 设置查询参数
// Content 配置需检索的文本信息
// SearchContentOption，设置检索结果的输出要求，ChunkExpand 指定输出内容向
前和向后扩展的段落
// Filter 设置 author 与 tags 字段的 Filter 表达式，过滤文件
// DocumentSetNames 指定需要查找的文件名，可批量设置
SearchByContentsParam searchByContentsParam =
SearchByContentsParam.newBuilder()
                .withContent("什么是向量")

.withSearchContentOption(SearchOption.newBuilder().withChunkExpand(A
rrays.asList(1,1)))
```

```
                .withFilter(new Filter(Filter.in("author",
Arrays.asList("Tencent", "tencent")))

.and(Filter.include("tags", Arrays.asList("AI",
"Embedding"))).getCond())
                .withDocumentSetNames(Arrays.asList("腾讯云向量数据
库.pdf"))
                .build();
// 根据配置的查询条件进行内容检索
List<Document> searchRes = collection.search(searchByContentsParam);
// 输出检索结果
int i = 0;
for (Document doc : searchRes) {
    System.out.println("\tres" +(i++)+": "+ doc.toString());
}
```

根据 Filter 表达式检索相似数据

如下示例，通过文件 meta 信息的标量字段 author 的 Filter 表达式，检索与 `什么是向量数据库` 相似的文本信息。

```
// link database， client 为 VectorDBClient() 创建的客户端对象
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 设置查询参数
SearchByContentsParam searchByContentsParam =
SearchByContentsParam.newBuilder()
                .withContent("什么是向量")

.withSearchContentOption(SearchOption.newBuilder().withChunkExpand(A
rrays.asList(1,1)))
                .withFilter(new Filter(Filter.in("author",
Arrays.asList("Tencent", "tencent")))

.and(Filter.include("tags", Arrays.asList("AI",
"Embedding"))).getCond())
                .build();
// 根据配置的查询条件进行内容检索
```

```
List<Document> searchRes = collection.search(searchByContentsParam);
// 输出检索结果
int i = 0;
for (Document doc : searchRes) {
    System.out.println("\tres" +(i++)+": "+ doc.toString());
}
```

入参描述

参数名称	是否必选	参数含义及配置方法
Content	是	以 String 类型输入检索的文本信息。
ChunkExpand	否	• 以数组形式配置检索的目标信息所需向前扩展的段落数量以及向后扩展的段落数。例如，输入[2,3]，指所检索到的 Chunk 返回时，同时返回其之前的2个段落与之后的3个段落。 • 段落指文件在上传存储时，自动向量化拆分的段落。 • 默认值：[1,1]。
DocumentSetNames	否	表示要查询的文档的名称，支持批量查询，数组元素范围[1,10]。

Filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中：
		• <field_name>：表示要过滤的字段名。
		• <operator>：表示要使用的运算符。
		  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
		  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。
		  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
		• <value>：表示要匹配的值。
		示例：`Filter('author="jerry"').And('page>20')`

Limit	是	指定返回最相似的 Top K 的 K 的值。

出参描述

如下为检索到的相似数据，返回最相似的 Top 3条数据。


```
{
  "res0": {
    "score": 0.8610672354698181,
    "data": {
      "text": "### 什么是向量？\n\n向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。\n\n",
      "endPos": 515,
      "startPos": 446,
      "next": [
        "### 什么是非结构化数据？\n\n非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。\n\n"
      ],
      "pre": [
        "## 关键概念\n\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。更多名词解释，请阅读 [关键概念](https://write.woa.com/document/113467593951064064 )。\n\n"
      ],
      "paragraphTitle": "什么是向量？",
      "allParentParagraphTitles": [
        "关键概念",
        "什么是向量？"
      ]
    },
    "documentSet": {
      "documentSetName": "\"腾讯云向量数据库.pdf\"",
      "documentSetId": "\"1387770634545528832\"",
      "docFields": [
        {
          "name": "author",
          "value": "Tencent"
        },
        {
          "name": "tags",
          "value": [
            "Embedding",
            "向量",
            "AI"
```

```
          ]
        }
      ]
    }
  },
  "res1": {
    "score": 0.8356848359107971,
    "data": {
      "text": "### 什么是 AI 中的向量表示？\n\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。具体上线时间，请关注［产品动态］(https://write.woa.com/document/117908688477143040    )。\n\n",
      "endPos": 795,
      "startPos": 594,
      "next": [
        "### 什么是向量检索？\n\n向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。\n\n"
      ],
      "pre": [
        "### 什么是非结构化数据？\n\n非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。\n\n"
      ],
      "paragraphTitle": "什么是 AI 中的向量表示？",
      "allParentParagraphTitles": [
        "关键概念",
        "什么是 AI 中的向量表示？"
      ]
    },
    "documentSet": {
      "documentSetName": "\"腾讯云向量数据库.pdf\"",
      "documentSetId": "\"1387770634545528832\"",
      "docFields": [
        {
          "name": "author",
          "value": "Tencent"
        },
        {
          "name": "tags",
          "value": [
            "Embedding",
            "向量",
```

```
            "AI"
          ]
        }
      ]
    }
  },
  "res2": {
    "score": 0.7836605906486511,
    "data": {
      "text": "### 什么是向量检索？\n\n向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。\n\n",
      "endPos": 889,
      "startPos": 795,
      "next": [
        "## 为什么是腾讯云向量数据库？\n\n腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。 具体信息，请参见［产品优势］
(https://write.woa.com/document/113467541728104448    )。\n\n"
      ],
      "pre": [
        "### 什么是 AI 中的向量表示？\n\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。具体上线时间，请关注［产品动态］
(https://write.woa.com/document/117908688477143040    )。\n\n"
      ],
      "paragraphTitle": "什么是向量检索？",
      "allParentParagraphTitles": [
        "关键概念",
        "什么是向量检索？"
      ]
    },
    "documentSet": {
      "documentSetName": "\"腾讯云向量数据库.pdf\"",
      "documentSetId": "\"1387770634545528832\"",
      "docFields": [
        {
          "name": "author",
          "value": "Tencent"
        },
        {
          "name": "tags",
```

```
            "value": [
              "Embedding",
              "向量",
              "AI"
            ]
          }
        ]
      }
    }
}
```

参数名	子参数	参数含义
score	-	表示查询向量与检索结果向量之间的相似性计算分数。 **说明：** 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。相似程度则通过 L2（欧几里得距离）、IP（内积）或 COSINE（余弦相似度）计算得出的分数来衡量，输出参数 score 表示相似性计算分数。其中，欧式距离（L2）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。
data	text	检索的结果。
	endPos	检索结果在文件中偏移的结束位置。
	startPos	检索结果在文件中偏移的起始位置。
	next	根据检索时，设置的参数 expand_chunk，返回检索结果向后扩展的段落。
	pre	根据检索时，设置的参数 expand_chunk，返回检索结果向前扩展的段落。
	paragraphTitle	检索到的 chunk 所属的段落标题。
	allParentParagraphTitles	检索到的 chunk 所属的所有层级的段落标题。
documentSet	documentSetId	文件 ID。

	documentSetName	文件名。
	docFields	自定义的文件 Metadata 信息的标量字段。例如：author、tags 等。 **说明：** 显示创建 CollectionView 时设置为 Filter 索引的字段，同时显示上传文件时或使用 update 新增的字段，但新增的字段不会构建索引。 • **name**：字段名。 • **value**：字段值。 • **fieldType**：字段类型，支持string、unit64、array。

删除指定文件

最近更新时间：2024-12-05 20:37:02

功能介绍

deleteDocumentSets() 接口用于删除存储于 CollectionView 的文件。

```
public AffectRes deleteDocumentSets(CollectionViewConditionParam param)
```

使用示例

说明：

- **支持批量删除，文件 ID 或文件名数组元素数量最大为20。**
- **支持使用 Filter 表达式过滤所需删除的所有文件。**

根据文件名过滤需删除的文件

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 配置删除条件，以便检索需删除的文件
CollectionViewConditionParam build = CollectionViewConditionParam
        .newBuilder()
        .withDocumentSetNames(Arrays.asList("腾讯云向量数据库.pdf"))
        .withFilter(new Filter("author=\"Tencent\""))
        .build();
// 删除文件
AffectRes affectRes = collection.deleteDocumentSets(build);
// 输出
System.out.println("\tres: " + affectRes.toString());
```

根据文件 ID 过滤出需删除的文件

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 配置删除条件，以便检索需删除的文件
CollectionViewConditionParam build = CollectionViewConditionParam
        .newBuilder()
        .witDocumentSetIds(Arrays.asList("11793516237********"))
        .withFilter(new Filter("authore=\"Tencent\""))
        .build();
// 删除文件
AffectRes affectRes = collection.deleteDocumentSets(build);
// 输出
System.out.println("\tres: " + affectRes.toString());
System.out.println(collection.query().size());
```

根据 Filter 表达式过滤需删除的文件

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 配置删除条件，以便检索需删除的文件
CollectionViewConditionParam build = CollectionViewConditionParam
        .newBuilder()
        .withFilter(new Filter("author=\"Tencent\""))
        .build();
// 删除文件
AffectRes affectRes = collection.deleteDocumentSets(build);
// 输出
System.out.println("\tres: " + affectRes.toString());
System.out.println(collection.query().size());
```

版权所有：腾讯云计算（北京）有限责任公司

入参描述

参数名	是否必选	参数含义	配置方法
DocumentSetNames	否	表示要查询的文档的名称。	支持批量查询，数组元素范围[1,20]。 **说明：** 同时配置 DocumentSetNames 与 filter 参数，删除数据将会取二者的交集。
DocumentSetIds	否	表示要查询的文档的所有 ID。	• ID 长度限制为[1,128]。 • 批量删除，数据元素最大值为20。 **说明：** 同时配置 DocumentSetIds 与 filter 参数，删除数据将会取二者的交集。

`Filter`	否	设置 Filter 表达式。	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中：
			• <field_name>：表示要过滤的字段名。
			• <operator>：表示要使用的运算符。
			  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
			  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。
			  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
			• <value>：表示要匹配的值。
			示例：
			`Filter('author="jerry"').And('page>20')`

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='Operation success'}
```

参数名	参数含义
affectedCount	影响行数，即为删除文件数量。

更新文件

最近更新时间：2024-12-05 20:37:03

功能介绍

update() 接口用于对通过主键（DocumentSet ID）与 Filter 表达式过滤检索 DocumentSet，对 DocumentSet 的部分字段进行更新。同时，支持新增字段。

```
public AffectRes update(CollectionViewConditionParam param, Map<String,
Object> updateFieldValues)
```

使用示例

说明：

- **支持通过主键（DocumentSet ID）或文件名，搭配 Filter 表达式过滤需更新的文件。**
- **支持新增字段，支持更改部分字段。新增字段，在创建 CollectionView 时没有为这些字段设置索引，那么新增这些字段时，系统不会自动为其创建索引。**

根据文件名过滤需更新的文件

如下示例，修改文件名为 `腾讯云向量数据库.md`，并满足 author 字段 Filter 表达式的文件的字段 author 为 tencent，新增字段 tag。

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 设置查询条件，以便查询需更新的文件
CollectionViewConditionParam updateParam =
CollectionViewConditionParam
        .newBuilder()
        .withDocumentSetNames(Arrays.asList("腾讯云向量数据库.pdf"))
        .withFilter(new Filter("author=\"tencent\""))
        .build();
Map<String, Object> updateFieldValues = new HashMap<>();
// 配置更新字段
updateFieldValues.put("author", "tencent");
```

```
// 新增字段
updateFieldValues.put("array_test", Arrays.asList("1", "2","5"));
// 更新操作
AffectRes affectRes = collection.update(updateParam,
updateFieldValues);
// 输出
System.out.println("\tres: " + affectRes.toString());
```

根据文件 ID 过滤需更新的文件

如下示例，修改指定文件 ID，并满足 Filter 表达式的文件的标量字段 **author** 为 tencent。

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 设置查询条件，以便查询需更新的文件
CollectionViewConditionParam updateParam =
CollectionViewConditionParam
        .newBuilder()
        .withDocumentSetIds(Arrays.asList("11793516237********"))
        .withFilter(new Filter("author=\"tencent\""))
        .build();
Map<String, Object> updateFieldValues = new HashMap<>();
// 配置更新字段
updateFieldValues.put("author", "tencent");
// 新增字段
updateFieldValues.put("array_test", Arrays.asList("1", "2","5"));
// 更新操作
collection.update(updateParam, updateFieldValues);

System.out.println(collection.query(10).get(0).toString());
```

根据 Filter 表达式过滤需更新的文件

如下示例，修改满足 **author** 的 Filter 表达式的文件的字段 author 为 tencent，并新增字段 tag。

```
// link database
AIDatabase db = client.aiDatabase("db-test-ai");
// link collectionView
CollectionView collection = db.describeCollectionView("coll-ai-
files");
// 设置查询条件，以便查询需更新的文件
CollectionViewConditionParam updateParam =
CollectionViewConditionParam
        .newBuilder()
        .withFilter(new Filter("author=\"tencent\""))
        .build();
Map<String, Object> updateFieldValues = new HashMap<>();
// 配置更新字段
updateFieldValues.put("author", "tencent");
// 新增字段
updateFieldValues.put("array_test", Arrays.asList("1", "2","5"));
// 更新操作
collection.update(updateParam, updateFieldValues);

System.out.println(collection.query(10).get(0).toString());
```

入参描述

参数名称	参数含义	子参数	是否必选	配置方法及要求
DocumentSetNames	指定需更新的文件名。	-	否	支持批量更新，数据元素最大值为20。 **说明：** 同时配置 DocumentSetNames 与 filter 参数，更新数据将会取二者的交集。
DocumentSetIds	指定需更新的文件ID。	-	否	支持批量更新，数据元素最大值为20。 **说明：** 同时配置 DocumentSetIds 与 filter 参数，更新数据将会取二者的交集。

Filter	配置 Filter 表达式过滤需更新的文件。		否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。 其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例： `Filter('author="jerry"').And('page>20')`
updateFieldValues	设置需更新的字段。	old_field	否	当前已存在的字段，更新字段对应的数据。 • 类型：string。 • 字符长度要求：[1,256]。
		new_field	否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。

出参描述

```
res: AffectRes{affectedCount=1, code=0, msg='Operation success'}
```

参数名	参数含义
affectedCount	更新的文档数量。如果该参数返回的值为 0，说明更新无效。

Index 操作
重建索引

最近更新时间：2025-07-01 10:19:43

接口定义

rebuildIndex() 接口用于重建指定 Collection 的所有索引，清除无用的索引数据，修复损坏的索引数据，优化索引结构，改善性能。

```
public BaseRes rebuildIndex(String database, String collection,
RebuildIndexParam rebuildIndexParam)
```

说明：

rebuildIndex 接口用于重建指定集合的向量索引，主要适用于以下两种场景：

- 首次使用 IVF 系列索引时，需要通过 rebuildIndex 接口初始化索引结构。具体信息，请参见 IVF 系列索引应用指南 。
- 在离线批量数据导入场景中，建议先执行数据写入操作（通过 upsert 接口设置 BuildIndex = false 暂停索引构建），待数据完整导入后，再调用rebuildIndex 接口集中创建索引，以此减少写入过程中的索引维护开销，提升整体导入效率。

警告：

Rebuild 操作需占用额外内存且可能影响服务，执行前请评估业务场景并选择重建方式。

- 保留原索引、同时构建新索引（即设置 dropBeforeRebuild=False）：期间集合读取正常，写入被阻止，构建新索引后，替换原索引，构建过程中，旧索引和新索引都会占用内存，如果内存足够可采用该方式，建议根据实际业务高低峰选择停写时段执行。
- 先删除原索引、再构建新索引（即设置 dropBeforeRebuild=True）：期间集合读写会受到影响，先删除旧索引，再构建新索引。如果内存严重不足时，则需采用该方案。建议根据业务需求选择可中断服务的时段处理。

使用示例

```
BaseRes res = client.rebuildIndex("db-test", "book-vector",
RebuildIndexParam.newBuilder()
        .withFieldName("sparse_vector")
        .withDropBeforeRebuild(true)
        //.withThrottle(0).build());
```

```
System.out.println("rebuild response: " + JsonUtils.toJsonString(res));
```

入参描述

参数	是否必选	参数含义	配置方法及要求
database	是	配置需重建索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection	是	指定需重建索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
FieldName	否	指定重建索引的字段。	• vector：重建稠密向量索引。默认值为 vector。 • sparse_vector：重建稀疏向量索引。 **说明：** 同一集合，不支持并发执行 rebuild。如果集合中既有稠密向量又有稀疏向量，则需要分别进行重建。
DropBeforeRebuild	否	标识在重建索引时，是否需先删除旧索引再重建新索引。 **警告：** 重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在重建索引之前，您需根据实际资源情况选择是否需先删除旧索引再重建，避免引起内存占满而阻塞业务正常运行。	取值如下所示： • **true**：重建之前，先删除旧索引在重建索引。 **警告：** 内存资源不足时，可配置为 true，先删除旧索引，在新索引构建过程中，无法正常读写。 • **false**：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为 false。 **警告：** 内存资源足够的情况下，可配置为 false，可不删除旧索

			引。在新索引构建过程中，可读数据，禁止写入数据。
Throttle	否	标识是否限制构建索引的单节点 CPU 核数。 **说明：** 重建索引会消耗 CPU 资源，为防止资源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为1。	取值如下所示： • 0：不限制 CPU 核数。在模型训练期间，会消耗大量的 CPU 资源。重建索引任务将会尽快执行，但可能会对其他集合的读写操作产生影响。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数,可避免构建索引期间对其他集合产生影响，但任务执行较慢。

出参说明

说明：
rebuildIndex() 执行之后，如果抛出异常，说明重建索引失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明执行成功，可使用 query() 确认删除的 Document 已经不存在。

相关说明

使用 describeCollection() 接口查看 Collection 的索引状态，返回参数 **indexStatus** 中的 **status** 标识当前 Collection 重建索引的状态，**startTime** 显示重建索引开始的时间。

- **initial**：表示首次创建索引，尚未 Rebuild 构建索引。
- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。
- **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。
- **failed**：重建索引失败，可能会影响集合读写操作。

注意：
training data 与 **building index** 状态期间不可写入数据。若重建索引之前先删除旧索引，则集合不可进行相似性查询，只能进行精确查询。

版权所有：腾讯云计算（北京）有限责任公司

新建索引

最近更新时间：2025-07-01 10:19:43

接口定义

addIndex() 接口指用于在已有集合上，新增标量字段索引。

```
public BaseRes addIndex(
    String database,
    String collection,
    AddIndexParam addIndexParam
)
```

说明：

- 如果集合开启了动态 schema（即创建集合时配置了 filterIndexConfig 参数），不支持创建标量索引。
- 当前不支持创建 json 类型的标量索引。

使用示例

```
BaseRes baseRes = client.addIndex("db-test", "book-vector",
AddIndexParam.newBuilder()
        .withBuildExistedData(true)
        .withIndexes(Arrays.asList(new FilterIndex("bookInfo",
FieldType.String, IndexType.FILTER),
                new FilterIndex("page", FieldType.Uint64,
IndexType.FILTER)))
        .build());
```

请求参数

参数	子参数	子参数	是否必选	参数含义	配置方法及要求
database		-	是	配置需重建索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。

collection		–	是	指定需重建索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
AddIndexParam	Indexes	FieldName	是	指定新增标量索引的字段。	用户自定义字段名。命名规则为1-128字符，只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。
		FieldType		指定字段数据类型。	指定自定义字段的数据类型。取值如下： • **string**：字符型。 • **uint64**：指无符号整数（unsigned integer）。 • **array**：数组类型，数组元素为string。
		IndexType		指定索引类型。	仅支持设置为 `filter` 。
	BuildExistedData	–	是	指定新增的索引字段是否写入过数据。	• **true**：默认为true，新增字段已写入过数据，需扫描历史数据并构建索引。 • **false**：新增索引字段为新增字段，无需扫描历史数据，可以快速创建索引。

删除索引

最近更新时间：2025-04-18 17:58:12

接口定义

dropIndex() 接口指用于在已有集合上，删除标量字段索引。

```
public BaseRes dropIndex(
    String database,
    String collection,
    List<String> fieldNames
)
```

> ⚠ **注意：**
>
> - 仅支持删除标量索引，不支持删除向量、稀疏向量索引。
> - 如果集合开启了动态 schema（即创建集合时配置了 filterIndexConfig 参数），不支持删除标量索引。

使用示例

```
BaseRes baseRes = client.dropIndex("db-test", "book-vector",
Arrays.asList("bookInfo","page"));
```

请求参数

参数	是否必选	参数含义	配置方法及要求
database	是	配置需删除索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection	是	指定需删除索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
fieldNames	是	用于指定删除的标量索引的字段名。	以数组形式，列出需删除的字段名。

修改向量索引

最近更新时间：2025-07-01 10:19:43

接口定义

modifyVectorIndex() 接口用于在不重建整个索引的前提下，调整已创建向量索引的配置参数，包括：数据精度类型、距离度量方式、性能优化参数等。

说明：

- 仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。
- 索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。
- HNSW 索引支持重新指定参数 M 和 efConstruction，IVF 索引支持重新指定 nlist（IVF_PQ 支持重新指定 M 和 nlist）。
- 仅 HNSW 索引，支持修改字段数据类型，例如：从vector（标准单精度浮点数向量）可修改为 float16_vector（半精度浮点数向量）。
- 支持重新指定向量相似度计算方法。

```
public BaseRes modifyVectorIndex(String database, String collection,
ModifyVectorIndexParam modifyVectorIndexParam)
```

使用示例

```
BaseRes baseRes = client.modifyVectorIndex(
        "db-test",
        "book-vector",
        ModifyVectorIndexParam.newBuilder()
                .withVectorIndex(new VectorIndex("vector",
MetricType.IP, FieldType.BFloat16Vector, IndexType.HNSW,  new
HNSWParams(16, 200)))
                .withRebuildRules(RebuildIndexParam.newBuilder()
                        .withDropBeforeRebuild(true)
                        .withThrottle(1)
                        .build())
                .build());
System.out.println("modify res: "+ JsonUtils.toJsonString(baseRes));
Database db = client.database("db-test");
Collection collectionAfter = db.describeCollection("book-vector");
```

```
System.out.println("after");
System.out.println(JsonUtils.toJsonString(collectionAfter));
```

入参描述

参数名	子参数一	子参数二	是否必选	配置方法及要求
database	-	-	是	指定 Database 名称，命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	-	-	是	指定需 Collection 的重建索引的 Collection 名称。 • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
ModifyVectorIndexParam	VectorIndex	fieldName	是	稠密向量数据字段名。固定为 vector。 **说明：** 仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。
		metricType	否	重新指定相似度计算方法。 • L2：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • IP：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 • COSINE：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度，所得值越大越与搜索值相似。
		fieldType	否	仅 HNSW 索引可重新指定向量索引的数据类型。默认值与当前类型一致，具体取值如下所示。更多信息，请参见 数据类型 。

			• **Vector**：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位），适用于需最高精度的计算任务或对误差敏感的向量检索。 • **Float16Vector**：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位 的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 BGE-large 等模型生成的嵌入向量。 • **BFloat16Vector**：bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，适用于向量数值范围较大（更多bit位表达整数）的场景。
		indexType	指定调整后的目标索引类型。默认与原索引类型一致，如果无需调整索引类型，该参数可不填。取值如下所示。 **说明：** • 索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。 • 切换索引类型时，请同步修改 params 参数，与目标索引类型的参数保持一致。 • FLAT 和 IVF 索引不支持 float16_vector 或 bfloat16_vector，当从 HNSW 索引类型切换到 FLAT 或 IVF 类型时，如果字段类型（fieldType）为 float16_vector 或 bfloat16_vector，则必须同步将字段类型修改为 vector 类型。 • 索引切换时需要注意提前预估内存占用，尤其是在 IVF 系列索引切换成 FLAT/HNSW 时，内存差异可能较大。 • **FLAT**：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。 • **HNSW**：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区间数据规模。 • **IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16**：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。
		params	否	重新指定向量索引参数。 • 索引类型 **indexType** 为 HNSW ，需配置如下参数。

				○ M：每个节点在检索构图中可以连接多少个邻居节点。 ○ 取值类型：uint64。 ○ 取值范围：[4,64]。一般可设置为16。 ○ efConstruction：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。 ○ 取值类型：uint64。 ○ 取值范围：[8,512]。一般可设置为200。 • 索引类型 indexType 为 IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16需配置如下参数。 ○ nlist：指索引中的聚类中心数量。 取值类型：uint64。取值范围：[1,65536]。 ○ M：指乘积量化中原始数据被拆分的子向量的数量。该参数仅 IVF_PQ 索引类型需配置。更多信息，请参见 索引与计算 。 ○ 取值要求：原始数据的向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。 ○ 取值范围：[1,向量维度]。
	RebuildRules	DropBeforeRebuild	否	标识在重建索引时，是否需先删除旧索引再重建新索引。取值如下所示： **警告：** 重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在重建索引之前，您需根据实际资源情况选择是否需先删除旧索引再重建，避免引起内存占满而阻塞业务正常运行。 • true：重建之前，先删除旧索引再重建索引。 **警告：** 内存资源不足时，可配置为 true，先删除旧索引，在新索引构建过程中，无法正常读写。 • false：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为 false。 **警告：**

				内存资源足够的情况下，可配置为 false，可不删除旧索引。在新索引构建过程中，可读数据，禁止写入数据。
		Throttle	否	标识是否限制构建索引的单节点 CPU 核数。取值如下所示： **说明：** 重建索引会消耗 CPU 资源，为防止资源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为1。 • 0：不限制 CPU 核数。在模型训练期间，会消耗大量的 CPU 资源。重建索引任务将会尽快执行，但可能会对其他集合的读写操作产生影响。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数,可避免构建索引期间对其他集合产生影响，但任务执行较慢。

出参描述


```
modify res: {"code":0,"msg":"Start rebuilding. You can use the
'/collection/describe' API to follow the progress of
rebuilding.","warning":null,"count":0,"requestId":null,"affectedCount":0
}
after
{"database":"db-test","collection":"book-
vector","replicaNum":1,"shardNum":1,"description":"test
collection0","indexes":
[{"fieldName":"id","fieldType":"string","indexType":"primaryKey"},
{"fieldName":"vector","fieldType":"vector","indexType":"HNSW","metricTyp
e":"COSINE","params":
{"efConstruction":200,"M":16},"dimension":3,"indexedCount":5},
{"fieldName":"author","fieldType":"string","indexType":"filter"},
{"fieldName":"bookName","fieldType":"string","indexType":"filter"},
{"fieldName":"array_test","fieldType":"array","indexType":"filter"},
{"fieldName":"expired_at","fieldType":"uint64","indexType":"filter"}],"c
reateTime":"2024-12-16 11:24:56","documentCount":5,"indexStatus":
{"status":"ready"},"ttlConfig":{"enable":true,"timeField":"expired_at"}}
```


参数名	参数含义

msg	提示启动 rebuild 重建索引，可使用 **describeCollection()** 接口返回的参数 **indexStatus** 查看集合索引状态。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。

User 管理

新建用户

最近更新时间：2025-01-21 19:52:12

接口定义

createUser() 接口用于创建自定义用户。

```
public BaseRes createUser(String username,String password)
```

使用示例

```
// client 为 VectorDBClient() 创建的客户端对象
BaseRes res = client.createUser("app_user", "********");
System.out.println("create user res: code:" + res.getCode() + ", msg: "
+ res.getMsg());
```

输出信息，如下所示：

```
create user res: code:0, msg: Operation success
```

入参描述

参数名	参数含义	数据类型	是否必填	默认值	参数配置
username	自定义用户名	string	是	-	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。
password	设置密码	string	是	-	• 密码长度要求：8~128字符。 • 密码设置支持如下类型的任意组合。 ○ 字母：大写 A~Z、小写 a~z。 ○ 数字：0~9。 ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号

					(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

为用户授权

最近更新时间：2025-01-21 19:52:12

接口定义

grantToUser() 接口用于对用户授予指定资源的访问权限。

```
public BaseRes grantToUser(UserGrantParam param)
```

使用示例

如下示例，给用户 app_user 授予读取 db-0 下所有集合数据的权限，并具有数据库 db1 下 coll1 集合的读写权限。

```
// client 为 VectorDBClient() 创建的客户端对象
BaseRes res = client.grantToUser(UserGrantParam.newBuilder()
        .withUser("app_user")
        .withPrivileges(Arrays.asList(

PrivilegeParam.newBuilder().withResource("db0.*").withActions(Arrays.asL
ist("read")).build(),

PrivilegeParam.newBuilder().withResource("db1.coll1").withActions(Arrays
.asList("readWrite")).build())
        ).build());
System.out.println("grant user permission res: code:" + res.getCode() +
", msg: " + res.getMsg());
```

输出信息，如下所示：

```
grant user permission res: code:0, msg: Operation success
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
User	需授权的用户名	-	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字

					（0~9）、下划线(_)，并以英文字母开头。
Privileges	对用户授予的权限列表，可以配置多条权限	resource	string	是	指定用户可操作的资源对象，分为如下三个层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"`。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有Collection。配置方式：`"resource": "db0.*"`，其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，包含集合下所有文档。配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。 **说明：** resource 只能指向已存在的资源；若指定的资源不存在，则触发错误提示信息。
		actions	array	是	配置资源对象允许操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

撤销用户授权

最近更新时间：2025-01-21 19:52:12

接口定义

revokeFromUser() 接口用于撤销用户已拥有的权限。

```
public BaseRes revokeFromUser(UserRevokeParam param)
```

说明：

- root 用户为数据库的默认用户，享有最高权限级别，无法被任何账号（包括其自身）撤销。
- 撤销权限时，权限需要和授予时完全对应，包括资源对象和允许的操作，都需要保持一致。
  - 只能针对原本层级回收权限，不能单独回收子对象的权限。例如，用户拥有 *.* 下的权限，那么不能单独回收 db.* 或者 db.coll 的权限。
  - 只能回收授予时指定的 action。例如，用户拥有某个资源对象的 readWrite 权限，那么回收时不能单独回收其 read 权限。
- **授权后，如果资源对象被删除，权限不会被撤销。后续如果该资源对象被重新创建，用户依然拥有原本的访问权限。建议用户在删除资源对象后，同时清理对应的访问权限。**

使用示例

```
BaseRes res = client.revokeFromUser(UserRevokeParam.newBuilder()
        .withUser("app_user")
        .withPrivilege(

PrivilegeParam.newBuilder().withResource("db0.*").withActions(Arrays.asL
ist("read")).build()
        ).build());
System.out.println("revoke user permission res: code:" + res.getCode() +
", msg: " + res.getMsg());
```

输出信息，如下所示。

```
revoke user permission res: code:0, msg: Operation success
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
User	用户名	-	string	是	需撤销操作权限的用户名。 • 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。
Privilege	指定撤销用户的资源对象以及其权限，支持一次撤销多条权限。	resource	string	是	指定用户撤销的资源对象，必须与授权时的指定的层级范围一致。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有Collection。配置方式：`"resource": "db0.*"`，其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
		actions	array	是	指定资源对象允许操作的权限，必须与授权时的权限保持一致。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

查询指定用户

最近更新时间：2025-01-21 19:52:12

接口定义

describeUser() 接口用于查询指定用户的信息，包括用户的创建时间、拥有的权限列表。

```
public UserDescribeRes describeUser(String user)
```

使用示例

```
// client 为 VectorDBClient() 创建的客户端对象
UserDescribeRes userDescribeRes = client.describeUser("app_user");
System.out.println("describe user res: " + userDescribeRes.toString());
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
user	需查询的用户名	-	string	是	指定需查询的用户名。 **说明：** root 用户享有最高权限级别，无法被任何账号（包括其自身）删除或修改其权限。

出参描述

```
describe user res: UserDescribeRes{user='app_user', createTime='2025-01-
08 15:27:15', privileges=[{"resource":"db0.*","actions":["read"]},
{"resource":"db1.coll1","actions":["readWrite"]}]}
```

参数名	子参数	参数配置
user	-	查询的用户名。

createTime	–	用户创建时间。
privileges	resource	用户可操作的资源对象层级。 • Global：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

查看用户列表

最近更新时间：2025-01-21 19:52:12

接口定义

listUser() 接口用于查询实例下所有的用户信息。

```
public UserListRes listUser()
```

使用示例

```
// client 为 VectorDBClient() 创建的客户端对象
UserListRes userListRes = client.listUser();
System.out.println("list user res: " +
JsonUtils.toJsonString(userListRes.getUsers()));
```

输出信息，如下所示：

```
list user res: [
{"user":"root","createTime":"","privileges":
[{"resource":"*.*","actions":["fullAccess"]}]},
{"user":"app_user","createTime":"2025-01-08 15:27:15","privileges":
[{"resource":"db0.*","actions":["read"]},
{"resource":"db1.coll1","actions":["readWrite"]}]}]
```

出参描述

参数名	子参数	参数配置
user	-	查询的用户名。
createTime	-	用户创建时间。其中，root 用户的创建时间为空。
privileges	resource	用户可操作的资源对象层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。

		• Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

删除用户

最近更新时间：2025-01-21 19:52:12

接口定义

dropUser() 接口用于删除自定义的用户。

```
public BaseRes dropUser(String user)
```

使用示例

```
// rebuild
BaseRes res = client.dropUser("app_user");
System.out.println("drop user res: code:" + res.getCode() + ", msg: " +
res.getMsg());
```

```
drop user res: code:0, msg: Operation success
```

入参描述

参数名	参数含义	数据类型	是否必填	参数配置
user	需删除的自定义的用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。 **说明：** root 用户无法被任何账号（包括其自身）删除。

重置用户密码

最近更新时间：2025-01-21 19:52:12

接口定义

changePassword() 接口用于修改自定义用户密码。

```
public BaseRes changePassword(String user, String password)
```

使用示例

```
// client 为 VectorDBClient() 创建的客户端对象
BaseRes res = client.changePassword("app_user", "##########");
System.out.println("change user password res: " + res.getCode() + " " +
res.getMsg());
```

输出信息，如下所示：

```
change user password res: 0 Operation success
```

入参描述

参数名	参数含义	数据类型	是否必填	参数配置
user	指定用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母、数字、下划线，并以英文字母开头。
password	设置新密码	string	是	• 密码长度要求：8~128字符。 • 密码字符类型要求如下： ○ 字母：大写 A~Z、小写 a~z。 ○ 数字：0~9。 ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

版权所有：腾讯云计算（北京）有限责任公司

Go SDK
SDK 准备

最近更新时间：2024-11-18 14:07:22

腾讯云向量数据库（Tencent Cloud VectorDB）的 GO SDK 是将 HTTP API 封装成易于使用的 Go 函数或类。开发者可以通过 Go SDK 更加方便地操作数据库。

SDK 信息

语言	支持版本	SDK 源码	SDK 安装
GO	Go 1.17 或更高版本	vectordatabase-sdk-go	`go get -u github.com/tencent/vectordatabase-sdk-go/tcvectordb`

GO Demo

最近更新时间：2025-05-26 19:31:32

类别	场景描述	SDK Demo
写入向量数据并检索	操作 Base 类数据库，直接写入向量数据并进行精确查询与相似度检索	demo
写入原始文本并检索（Embedding）	操作 Base 类数据库，应用 Embedding 模型写入或更新原始文本，并进行精确查询或相似度检索	embedding_demo
使用 AI 套件上传文件	基于 Collection 或 CollectionView 上传文件并进行内容相似性检索	• ai_demo • collection_upload_file_demo
使用稀疏向量	操作 Base 类数据库，应用 BM25 写入稀疏向量，使用语义检索（稠密向量）和关键字检索（稀疏向量）两路混合召回 **说明：** 使用 Go SDK 通过 Bm25 tcvdbtext 将文本转化成稀疏向量的过程中，英文文本统一会转换成小写字母处理。	sparse_vector_demo_with_encoder

连接数据库实例

最近更新时间：2025-07-01 10:19:43

接口定义

- `NewRpcClient()` 用于通过 RPC（Remote Procedure Call，远程过程调用）的请求方式创建一个向量数据库的客户端对象。
- `NewClient()` 为了兼容旧版本而保留，用于通过 HTTP I/O 请求方式创建一个向量数据库的客户端对象。
- `NewRpcClientPool()` 用于创建 RPC（远程过程调用）客户端连接池。

说明：

- Go SDK 为 1.3.2 及以上版本支持 RPC（Remote Procedure Call，远程过程调用）通信，在多线程场景，RPC 可以提供比 HTTP I/O 更好的并发请求能力。
- NewRpcClientPool() 适用于高可用版本实例，特别适合客户端节点较少但 collection 副本较多（尤其是单 Shard 多副本）的场景。

NewRpcClient()

```
func tcvectordb.NewRpcClient(url string, username string, key
string, option *tcvectordb.ClientOption) (*tcvectordb.RpcClient,
error)
```

NewClient()

```
func tcvectordb.NewClient(url string, username string, key string,
option *tcvectordb.ClientOption) (*tcvectordb.Client, error)
```

NewRpcClientPool()

```
func NewRpcClientPool(url, username, key string, option
*ClientOption) (VdbClient, error)
```

请求示例

RPC 请求（推荐）

```
package main
import (
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)

func main() {
    // 初始化客户端
    var defaultOption = &tcvectordb.ClientOption{
        Timeout:             30 * time.Second,
        ReadConsistency:     tcvectordb.EventualConsistency,
    }
    _, err := tcvectordb.NewRpcClient("http://10.0.X.X:80", "root",
"eC4bLRy2va*****************************", defaultOption)
    if err != nil {
        panic(err)
    }
}
```

HTTP IO 请求（兼容 Restful 旧版本）

```
package main
import (
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)
```

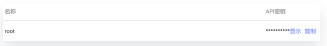
```
func main() {
    // 初始化客户端
    var defaultOption = &tcvectordb.ClientOption{
        Timeout:             30 * time.Second,
        MaxIdldConnPerHost: 2,
        IdleConnTimeout:    time.Minute,
        ReadConsistency:    tcvectordb.EventualConsistency,
    }
    _, err := tcvectordb.NewClient("http://10.0.X.X:80", "root",
"eC4bLRy2va******************************", defaultOption)
    if err != nil {
        panic(err)
    }
}
```

RPC Pool

```
package main
import (
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)

func main() {
    url := "vdb http url or ip and port"
    key := "key get from web console"
    ctx := context.Background()
    _, err := tcvectordb.NewRpcClientPool(url, "root", key,
&tcvectordb.ClientOption{
        ReadConsistency: tcvectordb.EventualConsistency,
        RpcPoolSize:     5})
    if err != nil {
        panic(err)
    }
}
```

请求参数

参数名称	子参数	参数含义	是否必选	获取方式
url	-	客户端所需连接的向量数据库服务端访问地址。	是	获取向量数据库实例内网 IP 地址与端口，请登录 向量数据库控制台，在**实例详情**页面**网络信息**区域直接复制**访问地址**。具体操作，请参见 查看实例信息。
username	-	客户端访问向量数据库服务端的账号。	是	数据库当前仅支持 root 账号。
key	-	客户端访问向量数据库服务端的 API 密钥，用于进行身份认证。	是	请登录 向量数据库控制台，在**密钥管理**页面直接复制密钥。具体操作，请参见 密钥管理。 
ClientOption	ReadConsistency	设置读一致性。	否	取值如下所示，默认为 EventualConsistency。 • StrongConsistency：强一致性。 • EventualConsistency：最终一致性。
	Timeout	请求超时时间。	否	• 单位：秒。 • 默认值：5。 • 取值范围：大于等于0。
	MaxIdldConnPerHost	设置连接池最大的空闲连接数。	否	• 默认为 2。 • 取值范围：大于0，如配置为0，或小于0，则使用默认值。
	IdleConnTimeout	设置空闲连接的超时时间。	否	• 默认值：不限制。 • 取值说明：大于0。如果配置为0，或者负数，表示不限制空闲连接的超时时间。 • 举例：time.Second*10。

	RpcPoolSize	设置连接池最大连接数	否	正整数，建议设置为数据节点个数的2-5倍。

管理 Database
检查 Database 是否存在

最近更新时间：2024-10-22 11:30:51

接口定义

- **ExistsDatabase()** 用于检查指定的数据库是否已经存在。
- **CreateDatabaseIfNotExists()** 用于检查指定的数据库是否已经存在，若不存在直接创建数据库。

ExistsDatabase()

```
func (i *implementerDatabase) ExistsDatabase(ctx context.Context,
name string) (bool, error)
```

CreateDatabaseIfNotExists()

```
func (i *implementerDatabase) CreateDatabaseIfNotExists(ctx
context.Context, name string) (*CreateDatabaseResult, error)
```

使用示例

ExistsDatabase()

```
database := "go-sdk-test-db"
dbExists, _ := client.ExistsDatabase(context.Background(), database)
log.Printf("database %v exists: %v", database, dbExists)
```

CreateDatabaseIfNotExists()

```
database := "go-sdk-test-db"
db, _ := client.CreateDatabaseIfNotExists(context.Background(),
database)
log.Printf("create database %v successfully, if it doesn't exist",
db.DatabaseName)
```

参数名	是否必须	参数含义
name	是	指定检查的数据库名。

管理 Base 类 Database

最近更新时间：2024-09-30 16:34:52

本文介绍创建、删除 Base 类 Database 的 SDK 接口。

创建 Base 类 Database

接口定义

```
CreateDatabase(ctx context.Context, name string)
(*tcvectordb.CreateDatabaseResult, error)
```

使用示例

```
database            := "go-sdk-test-db"
db, _ := client.CreateDatabase(context.Background(), database)
log.Printf("create database success, %s", db.DatabaseName)
```

参数	是否必选	参数含义	配置方法及要求
name	是	设置 Database 名称。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

删除 Base 类 Database

接口定义

```
DropDatabase(ctx context.Context, name string)
(*tcvectordb.DropDatabaseResult, error)
```

使用示例

```
database            := "go-sdk-test-db"
result, _ := client.DropDatabase(context.Background(), database)
log.Printf("DropDatabase result: %+v", result)
```

出参描述

```
result: &{AffectedCount:1}
```

参数名	参数含义
AffectedCount	影响行数。

管理 AI 类 Database

最近更新时间：2024-10-08 16:09:51

本文介绍创建、删除 AI 类 Database 的 SDK 接口。

创建 AI 类 Database

接口定义

```
CreateAIDatabase(ctx context.Context, name string) (result
*tcvectordb.CreateAIDatabaseResult, err error)
```

使用示例

```
aiDatabase := "go-sdk-demo-ai-db"
db, _ := client.CreateAIDatabase(context.Background(), aiDatabase)
log.Printf("Create AI database success, %s", db.DatabaseName)
```

参数	是否必选	参数含义	配置方法及要求
`aidatabase`	是	设置 AI 类 Database 名称。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

删除 AI 类 Database

接口定义

```
DropAIDatabase(ctx context.Context, name string) (result
*tcvectordb.DropAIDatabaseResult, err error)
```

使用示例

```
aiDatabase := "go-sdk-demo-ai-db"
result, _ := client.DropAIDatabase(context.Background(), aiDatabase)
log.Printf("DropAIDatabase result: %+v", result)
```

出参描述

```
result: &{AffectedCount:1}
```

参数名	参数含义
affectedCount	影响行数。

查询集群数据库

最近更新时间：2024-10-08 17:00:41

接口定义

ListDatabase() 用于查询集群中所有的向量数据库，包括 Base 类与 AI 类数据库。

```
ListDatabase(ctx context.Context) (result
*tcvectordb.ListDatabaseResult, err error)
```

使用示例

查询之前，您需要先 连接数据库实例 。

```
dbList, _ := client.ListDatabase(context.Background())
log.Printf("base database ======================")
for _, db := range dbList.Databases {
    log.Printf("database: %s, createTime: %s, dbType: %s",
db.DatabaseName, db.Info.CreateTime, db.Info.DbType)
}
log.Printf("ai database ======================")
for _, db := range dbList.AIDatabases {
    log.Printf("database: %s, createTime: %s, dbType: %s",
db.DatabaseName, db.Info.CreateTime, db.Info.DbType)
}
```

出参描述

```
2023/12/20 17:24:11 base database ======================
2023/12/20 17:24:11 database: go-sdk-test-db, createTime: 2023-12-20
17:24:08, dbType: BASE_DB
2023/12/20 17:24:11 AI database ======================
2023/12/20 17:24:11 database: go-sdk-test-ai-db, createTime: 2023-12-20
17:24:02, dbType: AI_DB
```

参数名	参数含义
database	数据库名。
createTime	数据库创建时间。

dbType	数据库类型。 • Base_DB：Base 类数据库。 • AI_DB：AI 类数据库。

管理 Collection
新建 Collection

最近更新时间：2025-07-09 15:17:22

接口定义

`CreateCollection()` 用于为已创建的 Base 类向量数据库创建 Collection。创建一个新的 Collection 需要指定所需的副本、分片、索引字段等关键参数，如果应用 Embedding 功能，则还需配置文本向量化模型及其相关字段。

```
CreateCollection(ctx context.Context, name string, shardNum uint32,
replicasNum uint32, description string, indexes tcvectordb.Indexes,
params ...*tcvectordb.CreateCollectionParams) (*tcvectordb.Collection,
error)
```

构建索引结构

创建 Collection 之前，需要为集合构建索引结构。

- 必须构建唯一的 Document id 为主键索引，用于快速查找特定行的数据。
- 必须构建 vector 向量数据索引，指定索引类型，当前支持 FLAT、HNSW、 IVF 系列向量索引方式。如何选择，请参见 Index 。
- 选取需要使用 Filter 表达式高效过滤数据的标量字段。

> **说明：**
> - 不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。
> - 插入数据时动态增加的标量字段暂不支持创建索引。创建 Collection 时，需规划好 Filter 索引的字段。
> - 创建写入文件的 Collection，务必为存放文件名的字段创建 Filter 索引，否则，在获取 COS 授权签名时提示错误信息。

写入向量数据构建索引

```
import (
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)
```

```
index := tcvectordb.Indexes{}
index.VectorIndex = append(index.VectorIndex,
tcvectordb.VectorIndex{
        FilterIndex: tcvectordb.FilterIndex{
            FieldName: "vector",
            FieldType: tcvectordb.Vector,
            IndexType: tcvectordb.HNSW,
        },
        Dimension:  3,
        MetricType: tcvectordb.COSINE,
        Params: &tcvectordb.HNSWParam{
            M:              16,
            EfConstruction: 200,
        },
    })
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "id", FieldType:
tcvectordb.String, IndexType: tcvectordb.PRIMARY})
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "bookName", FieldType:
tcvectordb.String, IndexType: tcvectordb.FILTER})
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "page", FieldType:
tcvectordb.Uint64, IndexType: tcvectordb.FILTER})
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "tags", FieldType:
tcvectordb.Array, IndexType: tcvectordb.FILTER})
//index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "author", FieldType:
tcvectordb.String, IndexType: tcvectordb.FILTER})
maxStrLen := uint32(32)
param := &tcvectordb.CreateCollectionParams{
    FilterIndexConfig: &tcvectordb.FilterIndexConfig{
        FilterAll:          true,
        FieldsWithoutIndex: []string{"author"},
        MaxStrLen:          &maxStrLen,
    },
}
```

写入稠密向量 + 稀疏向量构建索引

```
import (
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)
index := tcvectordb.Indexes{
    VectorIndex: []tcvectordb.VectorIndex{
        {
            FilterIndex: tcvectordb.FilterIndex{
                FieldName: "vector",
                FieldType: tcvectordb.Vector,
                IndexType: tcvectordb.HNSW,
            },
            // 如果使用 embedding ，维度要与模型的维度保持一致
            Dimension:  768,
            MetricType: tcvectordb.COSINE,
            Params: &tcvectordb.HNSWParam{
                M:              16,
                EfConstruction: 200,
            },
        },
    },
    SparseVectorIndex: []tcvectordb.SparseVectorIndex{
        {
                FieldName:  "sparse_vector",
                FieldType:  tcvectordb.SparseVector,
                IndexType:  tcvectordb.SPARSE_INVERTED,
                MetricType: tcvectordb.IP,
         },
    },

    FilterIndex: []tcvectordb.FilterIndex{
        {
            FieldName: "id",
            FieldType: tcvectordb.String,
            IndexType: tcvectordb.PRIMARY,
        },
        {
            FieldName: "bookName",
            FieldType: tcvectordb.String,
            IndexType: tcvectordb.FILTER,
        },
        {
```

```
            FieldName: "author",
            FieldType: tcvectordb.String,
            IndexType: tcvectordb.FILTER,
        },
        {
            FieldName: "tags",
            FieldType: tcvectordb.Array,
            IndexType: tcvectordb.FILTER,
        },
    },
}
```

二进制数据集合构建索引

```
import (
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)
index := tcvectordb.Indexes{}
index.VectorIndex = append(index.VectorIndex,
tcvectordb.VectorIndex{
    FilterIndex: tcvectordb.FilterIndex{
        FieldName: "vector",
        FieldType: tcvectordb.BinaryVector,
        IndexType: tcvectordb.BIN_FLAT,
    },
    Dimension:  16,
    MetricType: tcvectordb.HAMMING,
})
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "id", FieldType:
tcvectordb.String, IndexType: tcvectordb.PRIMARY})
```

仅稀疏向量

```
import (
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
```

```
)
index := tcvectordb.Indexes{}
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "id", FieldType:
tcvectordb.String, IndexType: tcvectordb.PRIMARY})
index.SparseVectorIndex  append(index.SparseVectorIndex,
tcvectordb.SparseVectorIndex{
    FieldName:  "sparse_vector",
    FieldType:  tcvectordb.SparseVector,
    IndexType:  tcvectordb.SPARSE_INVERTED,
    MetricType: tcvectordb.IP,
})
```

支持文件上传

注意：

在创建存储文件内容的 Collection 时，索引字段的设计需遵循以下限制与建议，否则可能导致上传文件失败。

- 文件名字段（file_name）：该字段**必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。字段名可以自定义，在上传文件时，需通过参数 FieldMappings 的 filename 映射自定义的字段名。
- 文件内容字段（text）：该字段用于存储知识点的原始文本内容。由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。字段名可以自定义，但需通过参数 FieldMappings 的 text 映射自定义的字段名。
- 文件图片信息字段（image_list）：该字段存放 PDF 文件中图片的 Key 列表。在原始检索出的文本块中，图片位置将以 {key} 的形式进行占位。通过调用接口 getImageUrl，可以获取图片 URL 地址列表，这些地址与 Key 一一对应，从而实现将原始 PDF 文档恢复并展示为 HTML 格式。
  - **不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。
  - 字段名可以自定义，但需要在上传文件时通过参数 FieldMappings 的 imageList 映射到自定义的字段名。

```
import (
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
```

```
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api"
    "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/ai_document_set"
)
index := tcvectordb.Indexes{}
index.VectorIndex = append(index.VectorIndex,
tcvectordb.VectorIndex{
    FilterIndex: tcvectordb.FilterIndex{
        FieldName: "vector",
        FieldType: tcvectordb.Vector,
        IndexType: tcvectordb.HNSW,
    },
    Dimension:  768,
    MetricType: tcvectordb.IP,
    Params: &tcvectordb.HNSWParam{
        M:              16,
        EfConstruction: 200,
    },
})
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "id", FieldType:
tcvectordb.String, IndexType: tcvectordb.PRIMARY})
index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "file_name", FieldType:
tcvectordb.String, IndexType: tcvectordb.FILTER})
```

参数	子参数	是否必选	参数含义
FilterIndex	FieldName	否	配置可作为 Filter 索引的自定义扩展的标量字段名。 **说明：** • Filter 索引（Filter Index）对**标量字段**建立的索引。标量字段名称、类型均由用户自定义，不限制标量字段数量. • 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询以此来匹配相似向量。 • 建立 Filter 索引时，选取需要使用 Filter 表达式高效过滤数据的标量字段。不做过滤查询、检索的标量

			字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。 • 必须构建唯一的 Document id 为主键的 Filter 索引，配置 **name** 固定为 **id** 。 • 配置其他自定义扩展的可作为 Filter 索引的标量字段，例如：author、page 等。该标量字段名称、类型均由用户自定义，且不限制字段数量。
	FiledType	是	指定 Filter 字段的数据类型。当前支持如下类型： • **String**：字符型。若 **name** 为 **id**，则该参数固定为 **tcvectordb.String**。 • **Uinit64**：指无符号整数，该参数可设置为 **tcvectordb.Uint64**。 • **Array**：数组类型。该参数可设置为 **tcvectordb.Array**，数组元素默认为 **String**。 • **Json**：由键值对组成的数据对象。具体规则，请参见 Json 类型 。为 json 类型的字段创建 Filter 索引，如下所示。 **⚠ 注意：** 开启 动态标量字段（动态 Schema ）时，即使用 filterIndexConfig 开启所有标量字段全索引模式，现已支持配置 json 类型的 Filter 索引。 `index.FilterIndex = append(index.FilterIndex, tcvectordb.FilterIndex{FieldName: "bookInfo", FieldType: tcvectordb.Json, IndexType: tcvectordb.FILTER})`

	IndexType	是	指定 Filter 字段的索引类型。 • 若 FieldName 为 id，该参数固定配置为 tcvectordb.PRIMARY，即以 id 为主键构建索引。 • 若 FieldName 为其他自定义的标量字段，该参数设置自定义字段的索引类别。该字段设置为 tcvectordb.FILTER，在后续检索数据时，才能对该字段设置 Filter 条件表达式进行 混合检索 。
	AutoId	否	自动生成 ID。指定 `AutoId: "uuid"` ，即可开启自动生成 ID 功能。 **说明：** • 当启用 autoId 功能后，用户在写入或更新数据时，无需手动传入 id 字段。若用户在启用 autoId 的情况下仍传入了 id 字段，则系统将优先采用用户传入的 id 值作为数据的唯一标识。 • 在执行批量写入（batch upsert）时，如果部分数据包含 id 字段，而另一部分数据未包含 id 字段，则系统仅对未传入 id 的数据生成默认 id。
VectorIndex	FieldName	是	指定向量数据的索引字段为 vector。
	FiledType	是	向量字段的数据类型。支持不同类型、不同精度的向量数据，以适用不同业务场景。更多信息，请参见 数据类型 。 • tcvectordb.Vector：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位），适用于需最高精度的计算任务或对误差敏感的向量检索。 • tcvectordb.Float16Vector：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位 的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 BGE-large 等模型生成的嵌入向量。 • tcvectordb.BFloat16Vector：bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，适用于向量数值范围较大（更多bit位表达整数）的场景。 **说明：** 当前仅索引类型为 HNSW 时，字段类型支持.Float16Vector与BFloat16Vector。

		• tcvectordb.BinaryVector：每个维度仅用 1 个比特（bit）表示 0 或 1，无指数位、尾数位和符号位，直接存储二进制数据，适用于存储稀疏向量。
IndexType	是	指定索引类型，取值如下所示。更多信息，请参见 索引类型 。 • FLAT：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。 • HNSW：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区间数据规模。 • BIN_FLAT：二进制索引，暴力检索，召回率100%。 • IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。
Dimension	是	指定向量维度。 • 取值类型：uint64。 • 取值范围：[1,4096]。 **说明：** • 若 indexType 为 BIN_FLAT，向量维度大于等于8，且必须为8的倍数。 • 开通 Embedding 功能，则无需配置该字段，该字段将自动配置为 Embedding 模型对应的向量维度。

	MetricType	是	指定向量之间距离度量的算法。取值如下： • **L2**：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • **IP**：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 • **COSINE**：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度。所得值越大越与搜索值相似。 • **Hamming**：汉明距离（Hamming Distance），计算两个二进制字符串对应位置上不同字符的数量，如果字符不同，两字符串的汉明距离就会加一。汉明距离越小，表示两个字符串之间的相似度越高。
	Params	否	如果索引类型 **indexType** 为 **HNSW**，则需配置如下 HNSWParams 参数。 • **M**：每个节点在检索构图中可以连接多少个邻居节点。 ○ 取值类型：uint64。 ○ 取值范围：[4,64]。默认为16。 • **EfConstruction**：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。 ○ 取值类型：uint64。 ○ 取值范围：[8,512]。默认为200。
		否	索引类型 **indexType** 分别为 **IVF_FLAT**、**IVF_PQ**、**IVF_SQ4**、**IVF_SQ8**、**IVF_SQ16**，则分别对应配置如下 **IVFFLATParams**、**IVFPQParams**、**IVFSQ8Params** 参数。 ○ **NList**：指索引中的聚类中心数量。 ○ 取值类型：uint64。 ○ 取值范围：[1,65536]。 ○ **M**：指乘积量化中原始数据被拆分的子向量的数量，仅 **IVF_PQ** 索引类型需配置。 ○ 取值要求：原始向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。 ○ 取值范围：[1,向量维度]。

SparseVectorIndex	FieldName	是	可选择构建稀疏向量索引，字段名固定为 sparse_vector。每个集合仅支持创建1个稀疏向量索引，无需指定稀疏向量维度。
	FiledType		稀疏向量字段类型固定为 tcvectordb.SparseVector。
	IndexType		稀疏向量索引类型固定为 tcvectordb.SPARSE_INVERTED。
	MetricType		稀疏向量相似性计算仅支持设置为 IP。
FilterIndexConfig	FilterAll	否	控制标量字段全索引模式的开启和关闭。 • true：开启。 • false：关闭。默认值为 fasle。
	FieldsWithoutIndex		当 FilterAll 为 true 时，支持通过 FieldWithoutFilterIndex 参数指定不创建索引的字段。 • 若 filterAll 为 false 时不支持配置。 • 默认值：NULL。
	MaxStrLen		当 FilterAll 为 true 时，支持通过 MaxStrLen 参数设定单条文档中创建索引的标量字段的最大字符数。如果字段字符数超出此限制，将按照设定的 maxStrLen 值进行截断创建索引。 • 默认值：32。 • 取值范围：[1,65536]。

（可选）配置 Embedding 参数

如果创建 Embedding 集合，则需配置其相关参数，如下示例，指定文本输入字段 text，向量数据字段为 vector，选择 Embedding 模型 BGE_BASE_ZH。

```
// 设置embedding字段和模型
param := &tcvectordb.CreateCollectionParams{
    Embedding: &tcvectordb.Embedding{
        Field:      "text",
        VectorField: "vector",
        ModelName:   "bge-base-zh",
    },
}
```

参数名	是否必选	参数含义

ModelName	否	指定使用的 Embedding 模型的名称。您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，请参见 Embedding 介绍 。取值如下所示： • bge-large-zh-v1.5：适用中文，1024维，推荐使用。 • bge-base-zh-v1.5：适用中文，768维。 • bge-large-zh：适用中文，1024维。 • bge-base-zh：适用中文，768维。 • m3e-base：适用中文，768维。 • e5-large-v2：适用英文，1024维。 • text2vec-large-chinese：适用中文，1024维。 • multilingual-e5-base：适用于多种语言类型，768维。 • BAAI/bge-m3：适用于多种语言类型，1024维。
Field	否	指定文本字段名称。取值类型：String。 **说明：** 写入数据或更新数据时，Embedding 模型会自动将该字段的文本内容转换成向量数据。
VectorField	否	指定向量字段。通过 Embedding 模型生成的向量会自动存储在该字段中。固定为 `vector` 。

创建 Collection

说明：
当前版本一个数据库实例下，不支持创建同名的 Collection。

创建 Collection 时可以根据预估数据规模和业务需求，指定集合的副本数和分片数。

- 分片数量：可根据数据规模判断，单分片数据量建议控制在300万以内，例如，若某个 Collection 有500万向量，可设置2个分片。如果数据量小于300万，建议使用1分片。系统对1分片有特定优化，可显著提升性能。当前支持的分片数量最小为1分片，最大为100分片。
- 副本数量：可以根据业务容灾以及吞吐量来判断，当前支持的副本数量两可用区最少为1个（不包含主节点），三可用区最少为2个，最大不超过实例的节点数量。搜索请求量越高的索引，建议设置越多的副本数，避免负载不均衡。并且，副本默认为可读，可以分担主节点的读压力，提升系统性能。

Embedding 模型（文本读写）集合

如下示例，创建 Embedding 的集合，指定 Embedding。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    embeddingCollection = "go-sdk-test-emcoll"
)

db := client.Database(database)
coll, _ := db.CreateCollection(ctx, embeddingCollection, 1, 1,
"desription doc", index, param)
log.Printf("CreateCollection success: %v: %v", coll.DatabaseName,
coll.CollectionName)
```

无 Embedding 模型（向量读写）集合

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "book-vector"
)

db := client.Database(database)
coll, _ := db.CreateCollection(ctx, collectionName, 1, 1, "test
collection", index)
log.Printf("CreateCollection success: %v: %v", coll.DatabaseName,
coll.CollectionName)
```

二进制数据集合

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "bin-vector"
)

db := client.Database(database)
```

```
coll, _ := db.CreateCollection(ctx, collectionName, 1, 1, "test
collection", index)
log.Printf("CreateCollection success: %v: %v", coll.DatabaseName,
coll.CollectionName)
```

参数	参数含义	是否必选	配置要求
name	指定 Collection 的名称。	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
replicaNum	指定 Collection 的副本数。副本数是指每个主分片有多个相同的备份，用来容灾和负载均衡。	是	• 取值类型：int。 • 取值范围：[2,9]。例如：5。
shardNum	指定 Collection 的分片数。分片是把大数据集切成多个子数据集。	是	• 取值类型：int。 • 取值范围：[1,100]。例如：5。
description	指定 Collection 的描述信息。	否	• 取值类型：string。 • 字符长度要求：[1,256]。 • 示例：this is the collection description。

TtlConfig	标识数据库是否开启 TTL 配置，指定存储数据过期时间戳字段名。	否	**Enable**：标识数据库是否开启 TTL 属性。 • true：开启。 • false：关闭。 **说明：** TTL（生存周期）功能规定了数据自创建或更新后将被自动删除的时间期限。向量数据库默认每小时轮询检查一次数据是否过期，超期即清理，且插入数据时允许存在最多1小时的过期时间误差。具体信息，请参见 TTL 。 **TimeField**：指定存储数据过期时间戳的字段名。插入数据时，以标准的 Unix 时间戳指定该字段的值。字段名要求如下： • 数据类型：uint64。 • 索引类型：Filter 索引。 • 示例：expired_at。 `param := &tcvectordb.CreateCollectionParams{ TtlConfig: &tcvectordb.TtlConfig{ Enable: true, TimeField: "expire_at", }, }`

返回结果

CreateCollection() 执行之后，如果抛出异常，说明创建 Collection 失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明创建 Collection 执行成功，可使用 listCollections() 查看已经创建的 Collection 配置信息。

检查 Collection 是否存在

最近更新时间：2024-12-23 21:13:13

接口定义

- **ExistsCollection()** 用于检查指定的数据库及其对应的集合是否已经存在。
- **CreateCollectionIfNotExists()** 用于检查指定的数据库及其对应的集合是否已经存在，若不存在直接创建集合。

ExistsCollection()

```
func (tcvectordb.CollectionInterface) ExistsCollection(ctx
context.Context, name string) (bool, error)
```

CreateCollectionIfNotExists()

```
func (tcvectordb.CollectionInterface)
CreateCollectionIfNotExists(ctx context.Context, name string,
shardNum uint32, replicasNum uint32, description string, indexes
tcvectordb.Indexes, params ...*tcvectordb.CreateCollectionParams)
(*tcvectordb.Collection, error)
```

使用示例

ExistsCollection()

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "book-vector"
)
db := client.Database(database)
```

```
collExists, err := db.ExistsCollection(ctx, collectionName)
printErr(err)
log.Printf("collection %v exists: %v", collectionName, collExists)
```

参数名	是否必须	参数含义	配置方法
name	是	检查的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
其他参数	是	集合相关参数。	若集合不存在，根据集合相关参数创建集合。参数含义，请参见 新建 Collection 。

CreateCollectionIfNotExists()

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "book-vector"
)
db := client.Database(database)
index := tcvectordb.Indexes{
    VectorIndex: []tcvectordb.VectorIndex{
    {
            FilterIndex: tcvectordb.FilterIndex{
                FieldName: "vector",
                FieldType: tcvectordb.Vector,
                IndexType: tcvectordb.HNSW,
            },
            Dimension:  3,
            MetricType: tcvectordb.COSINE,
            Params: &tcvectordb.HNSWParam{
                M:              16,
                EfConstruction: 200,
            },
        },
    },
    FilterIndex: []tcvectordb.FilterIndex{
```

```
        {FieldName: "id", FieldType: tcvectordb.String, IndexType:
tcvectordb.PRIMARY},
        {FieldName: "bookName", FieldType: tcvectordb.String,
IndexType: tcvectordb.FILTER},
        {FieldName: "expire_at", FieldType: tcvectordb.Uint64,
IndexType: tcvectordb.FILTER},
    },
}
param := &tcvectordb.CreateCollectionParams{
    TtlConfig: &tcvectordb.TtlConfig{
        Enable:    true,
        imeField: "expire_at",
    },
}
coll, err := db.CreateCollectionIfNotExists(ctx, collectionName, 3,
1, "test collection", index, param)
printErr(err)
log.Printf("create collection if not exists success: %v: %v",
coll.DatabaseName, coll.CollectionName)
```

参数名	是否必须	参数含义	配置方法
name	是	检查的集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。 **说明：** 其余参数含义，与 新建 Collection 一致。

查询 Collection 配置

最近更新时间：2025-02-12 17:58:52

接口定义

ListCollection() 接口可查询指定 Base 类 Database 中所有的 Collection。

```
ListCollection(ctx context.Context) (result
*tcvectordb.ListCollectionResult, err error)
```

使用示例

如下示例，查询数据库 go-sdk-test-db 下的集合配置。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
)
db := client.Database(database)
result, _ := db.ListCollection(ctx)
for _, col := range result.Collections {
    data, _ := json.Marshal(col)
    log.Printf("%+v", string(data))
}
```

出参描述

```
{
  "databaseName": "go-sdk-test-db",
  "collectionName": "go-sdk-test-coll",
  "documentCount": 0,
  "alias": null,
  "shardNum": 1,
  "replicasNum": 0,
  "indexes": {
    "VectorIndex": [
      {
        "FieldName": "vector",
        "FieldType": "vector",
        "ElemType": "",
```

```
        "IndexType": "HNSW",
        "Dimension": 3,
        "MetricType": "COSINE",
        "IndexedCount": 0,
        "Params": {
          "M": 16,
          "EfConstruction": 200
        }
      }
    ],
  "SparseVectorIndex": [
     {
       "FieldName": "sparse_vector",
       "FieldType": "sparseVector",
       "IndexType": "inverted",
       "MetricType": "IP",
     }
    ],
    "FilterIndex": [
      {
        "FieldName": "page",
        "FieldType": "uint64",
        "ElemType": "",
        "IndexType": "filter"
      },
      {
        "FieldName": "id",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "primaryKey"
      },
      {
        "FieldName": "bookName",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "filter"
      },
      {
        "FieldName": "tag",
        "FieldType": "array",
        "ElemType": "",
        "IndexType": "filter"
      }
```

```
    ]
  },
  "indexStatus": {
    "Status": "ready",
    "StartTime": "0001-01-01T00:00:00Z"
  },
  "embedding": {

  },
  "description": "test collection",
  "createTime": "2023-12-20T17:26:12Z"
}
```

参数	子参数	子参数	参数含义
databaseName	-	-	显示 Collection 所在的 Database 名称。
collectionName	-	-	显示 Collection 的名称。
replicaNum	-	-	显示 Collection 的副本数。
shardNum	-	-	显示 Collection 的分片数。
createTime	-	-	显示 Collection 的创建时间。
description	-	-	显示 Collection 的描述信息。
documentCount	-	-	返回 Collection 中存储的 Document 数量。
indexes	VectorIndex	fieldName	向量数据索引的字段名，固定为 vector。
		filedType	vector 字段的数据类型，固定为 vector。
		indexType	显示索引类型。

		indexedCount	向量索引的文档数量。
		dimension	向量数据维度。
		metricType	目标向量数据与查询的向量数据相似性计算方法。
		params	向量索引类型对应的相关参数。
	FilterIndex	fieldName	• **id**：默认对 id 构建主键 Filter 索引，对应 indexType 为 primaryKey。 • page、bookName、tag：自定义扩展的设置为 Filter 表达式的字段名。
		filedType	Filter 字段的数据类型。
		indexType	• fieldName 为 id，应 indexType 为 primaryKey。 • 其他为 Collection 配置的可以设置 Filter 表达式的字段，索引类型固定为 **filter**。
	SparseVectorIndex	fieldName	稀疏向量字段名，固定为：`sparse_vector`。
		fieldType	稀疏向量数据类型，固定为：`sparseVector`。
		indexType	索引类型，固定为：`inverted`。
		metricType	稀疏向量间的距离度量的算法。
embedding	_	status	说明该 Collection 是否配置 Embedding 模型。 • **enabled**：已配置。 • **disabled**：未配置。
		field	显示 Embedding 模型输入文本的字段名。
		model	Embedding 模型的名称。
		vectorField	显示 Embedding 模型向量字段名。

alias	-	-	集合别名。
indexStatus	-	status	标识当前 Collection 是否在重建索引。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

查询指定 Collection 配置

最近更新时间：2025-02-12 17:58:52

接口定义

DescribeCollection() 用于查找指定 Collection 的配置信息。

```
DescribeCollection(ctx context.Context, name string) (result
*tcvectordb.DescribeCollectionResult, err error)
```

使用示例

如下示例，通过接口 `DescribeCollection()` 查找 go-sdk-test-coll 的集合配置信息。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)

db := client.Database(database)
result, _ := db.DescribeCollection(ctx, collectionName)
data, _ := json.Marshal(result)
log.Printf("DescribeCollection result: %+v", string(data))
```

参数名	是否必选	参数含义	配置方法及要求
collectionName	是	指定所需查询的 Collection 名称。	使用 ListCollection() 获取指定数据库名下的 Collection 列表，复制需查询的集合名。

出参描述

```
{
    "databaseName": "go-sdk-test-db",
    "collectionName": "go-sdk-test-coll",
    "documentCount": 0,
    "alias": null,
    "shardNum": 1,
    "replicasNum": 0,
```

版权所有：腾讯云计算（北京）有限责任公司

```
    "indexes": {
        "VectorIndex": [{
            "FieldName": "vector",
            "FieldType": "vector",
            "ElemType": "",
            "IndexType": "HNSW",
            "Dimension": 3,
            "MetricType": "COSINE",
            "IndexedCount": 0,
            "Params": {
                "M": 16,
                "EfConstruction": 200
            }
        }],
       "SparseVectorIndex": [{
         "FieldName": "sparse_vector",
         "FieldType": "sparseVector",
         "IndexType": "inverted",
         "MetricType": "IP",
        }],
        "FilterIndex": [{
            "FieldName": "id",
            "FieldType": "string",
            "ElemType": "",
            "IndexType": "primaryKey"
        }, {
            "FieldName": "tag",
            "FieldType": "array",
            "ElemType": "",
            "IndexType": "filter"
        }, {
            "FieldName": "page",
            "FieldType": "uint64",
            "ElemType": "",
            "IndexType": "filter"
        }, {
            "FieldName": "bookName",
            "FieldType": "string",
            "ElemType": "",
            "IndexType": "filter"
        }]
    },
    "indexStatus": {
```

```
            "Status": "ready",
            "StartTime": "0001-01-01T00:00:00Z"
        },
        "embedding": {},
        "description": "test collection",
        "size": 0,
        "createTime": "2023-12-20T17:26:12Z"
}
```

参数	子参数	子参数	参数含义
database Name	-	-	显示 Collection 所在的 Database 名称。
collection Name	-	-	显示 Collection 的名称。
replicaNu m	-	-	显示 Collection 的副本数。
shardNu m	-	-	显示 Collection 的分片数。
createTi me	-	-	显示 Collection 的创建时间。
descripti on	-	-	显示 Collection 的描述信息。
documen tCount	-	-	返回 Collection 中存储的 Document 数量。
indexes	Vector Index	fieldNa me	向量数据索引的字段名，固定为 vector。
		filedTyp e	vector 字段的数据类型，固定为 vector。
		indexTy pe	显示索引类型。
		indexed Count	向量索引的文档数量。
		dimensi on	向量数据维度。

		metricType	目标向量数据与查询的向量数据相似性计算方法。
		params	向量索引类型对应的相关参数。
	FilterIndex	fieldName	• **id**：默认对 id 构建主键 Filter 索引，对应 indexType 为 primaryKey。 • page、bookName、tag：自定义扩展的设置为 Filter 表达式的字段名。
		filedType	Filter 字段的数据类型。
		indexType	• fieldName 为 id，应 indexType 为 primaryKey。 • 其他为 Collection 配置的可以设置 Filter 表达式的字段，索引类型固定为 **filter**。
	SparseVectorIndex	fieldName	稀疏向量字段名，固定为：`sparse_vector`。
		fieldType	稀疏向量数据类型，固定为：`sparseVector`。
		indexType	索引类型，固定为：`inverted`。
		metricType	稀疏向量间的距离度量的算法。
embedding	–	status	说明该 Collection 是否配置 Embedding 模型。 • **enabled**：已配置。 • **disabled**：未配置。
		field	显示 Embedding 模型输入文本的字段名。
		model	Embedding 模型的名称。
		vectorField	显示 Embedding 模型向量字段名。
alias	–	–	集合别名。
indexStatus	–	status	标识当前 Collection 是否在重建索引。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。

版权所有：腾讯云计算（北京）有限责任公司

			• **training**：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

清空 Collection 数据

最近更新时间：2024-09-30 16:34:52

接口定义

TruncateCollection() 用于清空 Collection 中所有的数据与索引，仅保留 Collection 配置信息，例如索引类型及参数、分片等设置，减少用户的操作成本。

```
TruncateCollection(ctx context.Context, name string) (result
*tcvectordb.TruncateCollectionResult, err error)
```

使用示例

> **警告：**
> 执行 truncate 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

如下示例，清空集合 go-sdk-test-coll 的数据。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)

db := client.Database(database)
result, _ := db.TruncateCollection(ctx, collectionName)
log.Printf("truncate collection result: %+v", result)
```

入参描述

参数	是否必选	参数含义	配置方法及要求
collectionName	是	指定需清空数据的 Collection 名。	使用 ListCollection() 获取指定数据库名下的 Collection 列表，复制需清空数据的集合名。

出参描述

```
truncate collection result: &{AffectedCount:1}
```

参数名	参数含义
affectedCount	影响行数，即为清空数据的集合数量。

版权所有：腾讯云计算（北京）有限责任公司

管理 Collection 别名

最近更新时间：2024-09-30 16:34:52

别名可以是一个简短的字符串，方便标识和访问对应的集合。一个 Collection 可以设置一个或者多个别名。

- SetAlias() 接口用于为 Collection 指定别名。
- DeleteAlias() 接口用于删除数据库指定的集合的别名。

> **说明：**
> 通过集合的别名做业务迁移时，仅需通过 setAlias() 接口将同一别名指向新的集合，别名与集合的映射关系将自动更新为新集合，可直接通过别名访问新集合。

为 Collection 创建别名

接口定义

```
SetAlias(ctx context.Context, collectionName string, aliasName string)
(result *SetAliasResult, err error)
```

使用示例

> **说明：**
> - Document 层级的访问优先访问别名，其余级别仅支持原 Collection 名操作。
> - 集合的别名可以和集合名重复，一个集合的多个别名之间不能重复。

如下示例，为集合 go-sdk-test-coll 指定别名 go-sdk-test-alias。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
    aliasName           = "go-sdk-test-alias"
)
db := client.Database(database)
colRes, _ := db.SetAlias(ctx, collectionName, aliasName)
log.Printf("SetAlias: %v", colRes)
```

参数名	是否必选	参数含义	配置方法及要求

collectionName	是	指定需创建别名的 Collection 名称。	使用 ListCollection() 获取指定数据库名下的 Collection 列表，复制需设置别名的集合名。
aliasName	是	设置别名。	别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

删除 Collection 别名

接口定义

```
DeleteAlias(ctx context.Context, aliasName string) (result
*DeleteAliasResult, err error)
```

使用示例

> **说明：**
> DB 和 Collection 级别的 drop 操作会同时删除库表下的所有别名。

如下示例，删除别名 go-sdk-test-alias。

```
colRes, _ := db.DeleteAlias(ctx, aliasName)
log.Printf("DeleteAlias: %v", colRes)
```

出参描述

```
SetAlias: &{1}
DeleteAlias: &{1}
```

参数名	参数含义
affectedCount	影响的集合数量。

使用限制

- DB 和 Collection 级别的 drop 操作会同时删除库表下的所有别名。
- Document 层级的访问优先访问别名，其余级别仅支持原 Collection 名操作。
- 集合的别名可以和集合名重复，一个集合的多个别名之间不能重复。

版权所有：腾讯云计算（北京）有限责任公司

删除指定 Collection

最近更新时间：2025-02-12 17:58:52

接口定义

DropCollection() 用于删除已创建的 Base 类 Collection。

```
DropCollection(ctx context.Context, name string) (result
*tcvectordb.DropCollectionResult, err error)
```

使用示例

> **警告：**
> 执行 drop 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

> **说明：**
> 如果您为集合设置了别名，删除集合操作仅支持使用集合的原始名称，无法通过别名来删除对应的集合。

如下示例，删除集合 go-sdk-test-coll 。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)

db := client.Database(database)
result, _ := db.DropCollection(ctx, collectionName)
log.Printf("drop collection result: %+v", result)
```

入参描述

参数名	是否必选	参数含义	配置方法及要求
collectionName	是	所需删除的 Collection 名称。	使用 ListCollection() 获取指定数据库名下的 Collection 列表，复制需清空数据的集合名。

出参描述

```
drop collection result: &{AffectedCount:1}
```

参数名	参数含义
AffectedCount	影响行数，即为删除的集合数量。

管理 CollectionView
新建 CollectionView

最近更新时间：2025-07-01 10:19:43

接口定义

CreateCollectionView() 用于为已创建的 AI 类向量数据库创建 CollectionView，以应用 AI 套件上传文件写入数据。一个新的 CollectionView 需要指定索引字段、配置文件拆分规则。

```
CreateCollectionView(ctx context.Context, name string, param
CreateCollectionViewParams) (result *CreateAICollectionViewResult, err
error)
```

构建索引结构

创建 Collection 之前，需要针对预上传的文件数据选取可作为 Filter 索引的标量字段，以便使用该字段的 Filter 条件表达式过滤查找文件。通常选取文件的 Metadata 信息字段。如下示例，预以文件的作者字段为 Filter 索引，设定字段名为 author_name。

> **说明：**
> - **不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。**
> - **插入数据时动态增加的标量字段暂不支持创建 Filter 索引，请在创建 Collection 时规划的字段。**

```
import (
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)
index := tcvectordb.Indexes{
    FilterIndex: []tcvectordb.FilterIndex{
        {
            FieldName: "author",
            FieldType: tcvectordb.String,
            IndexType: tcvectordb.FILTER,
        },
    },
}
```

参数名	是否必选	参数配置
FieldName	是	配置可作为 Filter 索引的自定义扩展的标量字段名。 **说明：** • Filter 索引（Filter Index）是建立在标量字段的索引。该标量字段名称、类型均由用户自定义，不限制标量字段数量。 • 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询以此来匹配相似向量。
FieldType	是	指定自定义字段的数据类型。取值如下： • **String**：字符型。 • **Uint64**：指无符号整数（unsigned integer）。 • **Array**：数组类型，默认数组元素为 string。
IndexType	否	该参数固定设置为 **FILTER**。

配置文件处理参数

创建 Collection 之前，需要指定文件 Embedding 与 Split 预处理文件的相关参数。

```
language              := string(tcvectordb.LanguageChinese)
enableWordsEmbedding  := true
appendTitleToChunk    := true
appendKeywordsToChunk := false
```

参数类别	参数名	是否必选	参数配置
Embedding	language	否	取值如下所示： • **LanguageChinese**：中文。 • **LanguageEnglish**：英文。 • **LanguageMulti**：多语言。

	enableWordsEmbedding	否	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
SplitterPreprocess	appendTitleToChunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • true：将段落 Title 追加到切分后的段落。 • false：不追加。默认值为 false。
	appendKeywordsToChunk	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • true：将全文的 keywords 追加到切分后的段落。默认值为 true。 • false：不追加。
	chunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。

创建 CollectionView

如下示例，基于以上配置的参数，创建集合视图 go-sdk-test-ai-coll 。

```
import (
    "context"
    "log"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api"
    collection_view "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/collection_view"
)
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

db := client.AIDatabase(aiDatabase)
coll, _ := db.CreateCollectionView(ctx, collectionViewName,
tcvectordb.CreateCollectionViewParams{
    Description: "test ai collectionView",
```

```
    Indexes:     index,
    Embedding: &collection_view.DocumentEmbedding{
        Language:             language,
        EnableWordsEmbedding: &enableWordsEmbedding,
    },
    SplitterPreprocess: &collection_view.SplitterPreprocess{
        AppendTitleToChunk:    &appendTitleToChunk,
        AppendKeywordsToChunk: &appendKeywordsToChunk,
    },
    ParsingProcess: &api.ParsingProcess{
            ParsingType: string(tcvectordb.VisionModelParsing),
   },
})
log.Printf("CreateCollectionView success: %v: %v", coll.DatabaseName,
coll.CollectionViewName)
```

参数	参数含义	是否必选	配置方法
collectionViewName	指定 CollectionView 的名称。	是	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
Indexes	索引参数	是	指 构建索引结构 中定义的索引参数。
Description	指定 CollectionView 的描述信息	否	• 取值类型：string。 • 字符长度要求：[1,256]。 • 示例：this is the collection description。
Embedding	Embedding 相关参数	是	配置文件处理参数 中指定的 Embedding 模型、以及语言类型。
SplitterPreprocess	文件 Split 预处理相关参数	否	配置文件处理参数 中指定的 Split 预处理参数，包含 Title 与 Keywords 追加的方式。

ParsingProcess	文件解析方式相关参数	否	通过参数 ParsingType 指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

查询 CollectionView 配置

最近更新时间：2024-12-05 20:37:03

接口定义

`ListCollectionViews()` 接口用于查询指定 AI 类 Database 中所有的 CollectionView。

```
ListCollectionViews(ctx context.Context) (result
*ListAICollectionViewsResult, err error)
```

请求示例

如下示例，查询数据库 go-sdk-test-ai-db 的集合视图配置信息。

```
import (
    "context"
    "encoding/json"
    "log"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    collection_view "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/collection_view"
)
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
db := client.AIDatabase(aiDatabase)
coll_view, _ := db.ListCollectionViews(ctx)
for _, col := range coll_view.CollectionViews {
    data, _ := json.Marshal(col)
    log.Printf("%+v", string(data))
}
```

返回信息

```
{
    "databaseName": "go-sdk-test-ai-db",
    "collectionViewName": "go-sdk-test-ai-coll",
    "alias": null,
```

```
    "embedding": {
        "language": "zh",
        "enableWordsEmbedding": true
    },
    "splitterPreprocess": {
        "appendTitleToChunk": true,
        "appendKeywordsToChunk": false
    },
    "parsingProcess":{
        "parsingType":"VisionModelParsing"
    },
    "indexedDocumentSets": 0,
    "totalDocumentSets": 0,
    "unIndexedDocumentSets": 0,
    "filterIndexes": [{
        "FieldName": "author_name",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "filter"
    }, {
        "FieldName": "documentSetName",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "filter"
    }, {
        "FieldName": "documentSetId",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "primaryKey"
    }],
    "description": "test ai collectionView",
    "createTime": "2023-12-20T17:48:03Z"
}
```


参数	子参数	子参数	参数含义
databaseName	-	-	显示 collectionView 所在的 AI 类 Database 名称。
collectionViewName	-	-	显示 collectionView 的名称。

embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • multi：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启。
alias	–	–	CollectionView 的所有别名。创建别名，请参见 管理 CollectionView 别名 。
createTime	–	–	显示 CollectionView 的创建时间。
description	–	–	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。
		unIndexedDocumentSets	未处理的文件数量。

splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • True：将段落 Title 追加到切分后的段落。 • False：不追加。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
Parsing Process	文件解析方式相关参数	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 documentSetId 文件 ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 documentSetName 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式检索文件。
	其他自定义需建立 Filter 索引的标量字段	fieldName	自定义扩展字段，例如：author、tags。
		filedType	显示自定义字段的数据类型。
		indexT	显示自定义字段索引类别为 `filter` 。

		ype	

查询指定 CollectionView

最近更新时间：2024-12-05 20:37:03

接口定义

DescribeCollectionView() 接口用于查询指定 CollectionView 的信息。

```
func (AICollectionViewInterface) DescribeCollectionView(ctx
context.Context, name string) (result *DescribeAICollectionViewResult,
err error)
```

使用示例

如下示例，查询集合视图 go-sdk-test-ai-coll 的配置信息。

```
import (
    "context"
    "encoding/json"
    "log"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    collection_view "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/collection_view"
)
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
db := client.AIDatabase(aiDatabase)
coll, _ := db.DescribeCollectionView(ctx, collectionViewName)
data, _ := json.Marshal(coll)
log.Printf("DescribeCollection result: %+v", string(data))
```

入参描述

参数名	是否必选	参数含义	配置方法及要求
collectionViewName	是	指定所需查询的 CollectionView 名称。	使用 ListCollectionViews() 获取指定数据库名下的 CollectionView 列表，复制需查询的集合视图。

出参描述


```
{
    "databaseName": "go-sdk-test-ai-db",
    "collectionViewName": "go-sdk-test-ai-coll",
    "alias": null,
    "embedding": {
        "language": "zh",
        "enableWordsEmbedding": true
    },
    "splitterPreprocess": {
        "appendTitleToChunk": true,
        "appendKeywordsToChunk": false
    },
    "parsingProcess": {
        "parsingType": "AlgorithmParsing"
    },
    "indexedDocumentSets": 0,
    "totalDocumentSets": 0,
    "unIndexedDocumentSets": 0,
    "filterIndexes": [{
        "FieldName": "documentSetId",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "primaryKey"
    }, {
        "FieldName": "documentSetName",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "filter"
    }, {
        "FieldName": "author_name",
        "FieldType": "string",
        "ElemType": "",
        "IndexType": "filter"
    }],
    "description": "test ai collectionView",
    "createTime": "2023-12-20T17:48:03Z"
}
```


参数	子参数	子参数	参数含义

databaseName	-	-	显示 collectionView 所在的 AI 类 Database 名称。
collectionViewName	-	-	显示 collectionView 的名称。
embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • mutil：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
alias	-	-	CollectionView 的所有别名。创建别名，请参见 /ai/alias/set 。
createTime	-	-	显示 CollectionView 的创建时间。
description	-	-	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。
		unIndexedDocumen	未处理的文件数量。

		tSets	
splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • True：将段落 Title 追加到切分后的段落。 • False：不追加。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。
parsingProcess	指定 PDF 类型文件的解析方式	parsingType	取值如下所示： • VisionModelParsing：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • AlgorithmParsing：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 documentSetId 文件ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 documentSetName 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式检索文件。
	其他自定义需建立	fieldName	自定义扩展字段，例如：author、tags。

	Filter 索引的标量字段	filedType	显示自定义字段的数据类型。
		indexType	显示自定义字段索引类别为 `filter` 。

清空 CollectionView 数据

最近更新时间：2024-09-30 16:34:53

接口定义

TruncateCollectionView() 用于清空 CollectionView 中所有的数据与索引，仅保留 CollectionView 配置信息，减少用户的操作成本。

```
TruncateCollectionView(ctx context.Context, name string) (result
*TruncateAICollectionViewResult, err error)
```

使用示例

> **警告：**
> 执行 truncate 操作将会永久删除指定 CollectionView 下的所有数据。在操作之前，务必谨慎考虑。

如下示例，清空集合视图 go-sdk-test-ai-coll 的数据。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
db := client.AIDatabase(aiDatabase)
result, err := db.TruncateCollectionView(ctx, collectionViewName)
log.Printf("TruncateCollectionView success: %+v", result)
```

入参描述

参数	是否必选	参数含义	配置方法及要求
collectionViewName	是	指定需清空数据的 CollectionView 名。	使用 ListCollectionViews() 获取指定数据库名下的 CollectionView 列表，复制需清空数据的集合视图。

出参描述

```
TruncateCollectionView success: &{AffectedCount:1}
```

参数名	参数含义
affectedCount	影响行数，即为清空数据的集合视图数量。

版权所有：腾讯云计算（北京）有限责任公司

管理 CollectionView 别名

最近更新时间：2024-09-30 16:34:53

接口定义

别名可以是一个简短的字符串，方便标识和访问对应的集合。一个 CollectionView 可以设置一个或者多个别名。

- `SetAlias()` 接口用于为 CollectionView 指定别名。
- `DeleteAlias()` 接口用于删除数据库指定的集合视图的别名。

> **说明：**
> 通过集合的别名做业务迁移时，仅需通过 **setAlias()** 接口将同一别名指向新的集合，别名与集合的映射关系将自动更新为新集合，可直接通过别名访问新集合。

SetAlias()

```
SetAlias(ctx context.Context, collectionView string, aliasName
string) (result *SetAIAliasResult, err error)
```

DeleteAlias()

```
DeleteAlias(ctx context.Context, aliasName string) (result
*DeleteAIAliasResult, err error)
```

使用示例

为 CollectionView 创建别名

> **说明：**
> - DocumentSet 层级的访问优先访问别名，其余级别仅支持原 CollectionView 名操作。
> - 集合视图的别名可以和集合视图名重复，一个集合视图的多个别名之间不能重复。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName  = "go-sdk-test-ai-coll"
    aliasName           = "go-sdk-test-alias"
)

db := client.AIDatabase(aiDatabase)
colRes, _:= db.SetAlias(ctx, collectionViewName, aliasName)
log.Printf("Set a CollectionView alias: %v", colRes)
```

参数名	是否必选	参数含义	配置方法及要求
collectionViewName	是	指定需创建别名的 CollectionView 名称。	使用 ListCollectionViews() 获取指定数据库名下的 CollectionView 列表，复制需设置别名的集合视图。
aliasName	是	设置别名。	别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

删除 CollectionView 别名

> **说明：**
> DB 和 CollectionView 级别的 drop 操作会同时删除库表下的所有别名。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    aliasName           = "go-sdk-test-alias"
)
db := client.AIDatabase(aiDatabase)
colRes, _:= db.DeleteAlias(ctx, aliasName)
log.Printf("Delete CollectionView Alias : %v", colRes)
```

出参描述

```
Set a CollectionView alias: &{1}
```

```
Delete CollectionView Alias : &{1}
```

参数名	参数含义
AffectedCount	影响的集合视图数量。

删除 CollectionView

最近更新时间：2024-09-30 16:34:53

接口定义

DropCollectionView() 用于删除已创建的 CollectionView。

```
DropCollectionView(ctx context.Context, name string) (result
*DropAICollectionViewResult, err error)
```

使用示例

> **警告：**
> 执行 drop 操作将会永久删除指定 CollectionView 下的所有 DocumentSet。在操作之前，务必谨慎考虑。

如下示例，删除集合视图 go-sdk-test-ai-coll。

```
var (
    ctx                = context.Background()
    aiDatabase         = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
db := client.AIDatabase(aiDatabase)
res, err := db.DropCollectionView(ctx, collectionViewName)
log.Printf("DropCollectionView success: %v", res)
```

参数名	参数含义
collectionViewName	所需删除的 CollectionView 名称。

出参描述

```
DropCollectionView success: &{AffectedCount:1}
```

参数名	参数含义
AffectedCount	影响行数，即为删除的集合视图数量。

版权所有：腾讯云计算（北京）有限责任公司

管理 Document 数据
写入数据

最近更新时间：2025-04-17 16:55:43

接口定义

`Upsert()` 接口用于给创建的 Collection 中插入 Document。如果 Collection 在创建时，已配置 Embedding 参数，则仅需要插入文本信息，Embedding 服务会自动将文本信息转换为向量数据，存入数据库。

```
Upsert(ctx context.Context, databaseName string, collectionName string,
documents interface{}, params ...*tcvectordb.UpsertDocumentParams)
(result *tcvectordb.UpsertDocumentResult, err error)
```

使用示例

写入文本

如下示例，给集合 go-sdk-test-emcoll 基于文本字段 segment 插入数据。segment 为创建 Collection 时指定的文本字段。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    embeddingCollection = "go-sdk-test-emcoll"
)
result, err := client.Upsert(ctx, database, embeddingCollection,
[]tcvectordb.Document{
    {
        Id: "0001",
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 21},
            "text":   {Val: "富贵功名，前缘分定，为人切莫欺心。"},
        },
    },
    {
        Id: "0002",
        Fields: map[string]tcvectordb.Field{
```

```
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 22},
            "text":  {Val: "正大光明，忠良善果弥深。些些狂妄天加谴，眼前不遇
待时临。"},
        },
    },
    {
        Id: "0003",
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 23},
            "text":  {Val: "细作探知这个消息，飞报吕布。"},
        },
    },
    {
        Id: "0004",
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 24},
            "text":  {Val: "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，可
往投之。”布从其言，竟投徐州来。有人报知玄德。"},
        },
    },
    {
        Id: "0005",
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 25},
            "text":  {Val: "玄德曰：“布乃当今英勇之士，可出迎之。”糜竺
曰：“吕布乃虎狼之徒，不可收留；收则伤人矣。"},
        },
    },
})

log.Printf("upsert result: %+v", result)
```

写入向量

如下示例，给集合 go-sdk-test-coll 直接写入向量数据。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
buildIndex := true
result, err := client.Upsert(ctx, database, collectionName,
[]tcvectordb.Document{
    {
        Id:     "0001",
        Vector: []float32{0.2123, 0.21, 0.213},
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 21},
            "segment":  {Val: "富贵功名，前缘分定，为人切莫欺心。"},
            "tag":      {Val: []string{"孙悟空", "猪八戒", "唐僧"}},
        },
    },
    {
        Id:     "0002",
        Vector: []float32{0.2123, 0.22, 0.213},
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 22},
            "segment":  {Val: "正大光明，忠良善果弥深。些些狂妄天加谴，眼前
不遇待时临。"},
            "tag":      {Val: []string{"孙悟空", "猪八戒", "唐僧"}},
        },
    },
    {
        Id:     "0003",
        Vector: []float32{0.2123, 0.23, 0.213},
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 23},
            "segment":  {Val: "细作探知这个消息，飞报吕布。"},
            "tag":      {Val: []string{"曹操", "诸葛亮", "刘备"}},
```

```
        },
    },
  }, &tcvectordb.UpsertDocumentParams{BuildIndex: &buildIndex})

log.Printf("upsert result: %+v", result)
```

写入稀疏向量

如下示例，给集合 go-sdk-test-coll 直接写入向量数据与稀疏向量，对应字段为 Vector 与 SparseVector。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
buildIndex := true
result, err := client.Upsert(ctx, database, collectionName,
[]tcvectordb.Document{
    {
        Id:     "0001",
        Vector: []float32{0.2123, 0.21, 0.213},
        SparseVector: []encoder.SparseVecItem{
            {
               TermId: 4056151844,
               Score:  0.7627807,
            },
            {
               TermId: 4293601821,
               Score:  0.7627807,
            },
            {
               TermId: 1834252564,
               Score:  0.7627807,
            },
        },
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 21},
```

```
            "segment":  {Val: "富贵功名，前缘分定，为人切莫欺心。"},
            "tag":      {Val: []string{"孙悟空", "猪八戒", "唐僧"}},
        },
    },
    {
        Id:     "0002",
        Vector: []float32{0.2123, 0.22, 0.213},
        SparseVector: []encoder.SparseVecItem{{
           TermId: 4056151844,
           Score:  0.7627807,
            },
            {
               TermId: 4293601821,
               Score:  0.7627807,
            },
            {
               TermId: 1834252564,
               Score:  0.7627807,
            },
        },
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 22},
            "segment":  {Val: "正大光明，忠良善果弥深。些些狂妄天加谴，眼前
不遇待时临。"},
            "tag":      {Val: []string{"孙悟空", "猪八戒", "唐僧"}},
        },
    },
    {
        Id:     "0003",
        Vector: []float32{0.2123, 0.23, 0.213},
        SparseVector: []encoder.SparseVecItem{{
           TermId: 4056151844,
           Score:  0.7627807,
            },
            {
               TermId: 4293601821,
               Score:  0.7627807,
            },
            {
               TermId: 1834252564,
               Score:  0.7627807,
```

```
            },
        },
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 23},
            "segment":  {Val: "细作探知这个消息，飞报吕布。"},
            "tag":      {Val: []string{"曹操", "诸葛亮", "刘备"}},
        },
    },
  },
}, &tcvectordb.UpsertDocumentParams{BuildIndex: &buildIndex})

log.Printf("upsert result: %+v", result)
```

稀疏向量工具

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
buildIndex := true
bm25, err :=
encoder.NewBM25Encoder(&encoder.BM25EncoderParams{Bm25Language:
"zh"})
if err != nil {
  log.Fatalf(err.Error())
}
segments := []string{
  "腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。",
  "作为专门为处理输入向量查询而设计的数据库，它支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，高达百万级 QPS 及毫秒级查询延迟。",
  "不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。",
}
// 查看稀疏向量文本拆分情况
```

```
tokens := bm25.GetTokenizer().Tokenize(segments[0])
fmt.Println("tokens: ", tokens)
// 如下使用 tcvdb-text 工具的接口 bm25.EncodeTexts 将segments 文本转为稀疏向量表示。
sparse_vectors, err := bm25.EncodeTexts(segments)
if err != nil {
  log.Fatalf(err.Error())
}
// 写入向量数据、稀疏向量、标量字段
result, err1 := client.Upsert(ctx,, database, collectionName,
[]tcvectordb.Document{
    {
        Id:     "0001",
        Vector: []float32{0.2123, 0.21, 0.213},
        SparseVector: sparse_vectors[0],
    },
    {
        Id:     "0002",
        Vector: []float32{0.2123, 0.22, 0.213},
        SparseVector: sparse_vectors[1],
    },
    {
        Id:     "0003",
        Vector: []float32{0.2123, 0.23, 0.213},
        SparseVector: sparse_vectors[2],
    },
  },
}, &tcvectordb.UpsertDocumentParams{BuildIndex: &buildIndex})
```

写入二进制

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/utils"
)
var (
    ctx                 = context.Background()
```

```
    database             = "go-sdk-test-db"
    collectionName       = "go-sdk-test-coll"
)
binaryVectors := [][]byte{
        {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
        {1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1},
}
insertVectors := make([][]float32, 0)
for _, binaryVector := range binaryVectors {
    res, err := utils.BinaryToUint8(binaryVector)
    if err != nil {
        panic(err)
    }
    insertVectors = append(insertVectors, res)
}
documentList := []tcvectordb.Document{
    {
        Id:     "0001",
        Vector: insertVectors[0],
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 21},
            "segment":  {Val: "富贵功名，前缘分定，为人切莫欺心。"},
        },
    },
    {
        Id:     "0002",
        Vector: insertVectors[1],
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 22},
            "segment":  {Val: "正大光明，忠良善果弥深。些些狂妄天加谴，眼前
不遇待时临。"},
        },
    },
    {
        Id:     "0003",
        Vector: insertVectors[2],
```

```
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 23},
            "segment":  {Val: "细作探知这个消息，飞报吕布。"},
        },
    },
    {
        Id:     "0004",
        Vector: insertVectors[3],
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 24},
            "segment":  {Val: "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，
可往投之。”布从其言，竟投徐州来。有人报知玄德。"},
        },
    },
    {
        Id:     "0005",
        Vector: insertVectors[4],
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 25},
            "segment":  {Val: "玄德曰：“布乃当今英勇之士，可出迎之。”糜竺
曰：“吕布乃虎狼之徒，不可收留；收则伤人矣。"},
        },
    },
}
result, _ := client.Upsert(ctx, database, collectionName,
documentList)
log.Printf("UpsertResult: %+v", result)
```

Embedding文本与稀疏向量

```
import (
    "context"
    "log"
    "strconv"
```

```
    "time"

    "github.com/tencent/vectordatabase-sdk-go/tcvdbtext/encoder"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
segments := []string{
    "腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。",
    "作为专门为处理输入向量查询而设计的数据库，它支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，高达百万级 QPS 及毫秒级查询延迟。",
    "不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。",
    "腾讯云向量数据库（Tencent Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 ",
    "腾讯云向量数据库可以和大语言模型 LLM 配合使用。企业的私域数据在经过文本分割、向量化后，可以存储在腾讯云向量数据库中，构建起企业专属的外部知识库，从而在后续的检索任务中，为大模型提供提示信息，辅助大模型生成更加准确的答案。",
}
bm25, _ :=
encoder.NewBM25Encoder(&encoder.BM25EncoderParams{Bm25Language:
"zh"})
sparse_vectors, _ := bm25.EncodeTexts(segments)
documentList := []tcvectordb.Document{
        {
            Id: "0001",
            Fields: map[string]tcvectordb.Field{
                "text":     {Val: segments[0]},
                "bookName": {Val: "腾讯云向量数据库"},
            },
            SparseVector: sparse_vectors[0],
        },
        {
            Id: "0002",
            Fields: map[string]tcvectordb.Field{
                "text": {Val: segments[1]},
                "bookName": {Val: "VectorDB"},
```

```
            },
            SparseVector: sparse_vectors[1],
        },
        {
            Id: "0003",
            Fields: map[string]tcvectordb.Field{
                "text": {Val: segments[2]},
                "bookName": {Val: "腾讯云向量数据库"}
            },
            SparseVector: sparse_vectors[2],
        },
        {
            Id: "0004",
            Fields: map[string]tcvectordb.Field{
                "text": {Val: segments[3]},
            },
            SparseVector: sparse_vectors[3],
        },
        {
            Id: "0005",
            Fields: map[string]tcvectordb.Field{
                "text": {Val: segments[4]},
            },
            SparseVector: sparse_vectors[4],
        },
    }
result, _ := client.Upsert(ctx, database, collectionName,
documentList)
log.Printf("UpsertResult: %+v", result)
```

Json 类型

```
var (
    ctx = context.Background()
    database = "go-sdk-test-db"
    collection = "go-sdk-test-coll"
)
documentList := []tcvectordb.Document{
    {
        Vector: []float32{0},
```

```
            Fields: map[string]tcvectordb.Field{
                "bookInfo": {Val: map[string]interface{}{
                    "bookName": "西游记",
                    "author":   "吴承恩",
                }},
            },
        },
        {
            Vector: []float32{0.3},
            Fields: map[string]tcvectordb.Field{
                "bookInfo": {Val: map[string]interface{}{
                    "bookName": "红楼梦",
                    "author":   "曹雪芹",
                }},
            },
        },
    }
    result, _ := client.Upsert(ctx, database, collection, documentList)
    log.Printf("UpsertResult: %+v", result)
```

入参描述

参数名称	参数含义	子参数	是否必选	配置方法
databaseName	指定需写入的数据库名	-	是	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	指定写入数据的集合	-	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
BuildIndex	是否重建索引	-	否	取值如下所示： • **true**：需更新索引。默认值是 **true**。 • **false**：不更新索引。

				注意： 如果创建 Collection 选择的索引类型为 IVF 系列： • 当第一次写入时，当前集合还没有向量索引，此时 BuildIndex 必须为 false。插入原始数据之后，需通过 rebuild_index() 训练数据并重建索引。 • 当集合已经调用过 rebuild_index() 创建索引后，集合已经存在向量索引，此时： ○ 如果 BuildIndex = true，表示新写入的数据会加入到已有的 IVF 索引中，但不会更新索引结构，此时新写入的数据可以被检索到。 ○ 如果 BuildIndex = false，表示新写入的数据不会加入到已有的 IVF 索引中，此时新写入的数据无法被检索到。
Document	指定要插入的 Document 数据，是一个数组，支持单次插入一条或者多条 Document，最大可插入 1000 条。	Id	是	Document 主键，长度限制为[1,128]。
		Vector	否	写入向量数据。 • 集合的向量索引为 FLAT/HNSW/IVF 时，直接传入向量数据，以32位浮点数存储，如[0.2123, 0.9281, ..., 0.6712]。 • 集合的向量索引为二进制索引时，写入向量时可以直接写入二进制数据，也可以十进制写入。十进制写入数据时，数据维度是创建集合时指定维度的 1/8。例如：upsert 时 vector=[12, 21]，则创建集合时维度为16维，其代表的二进制数据为[00001100, 00010101]。 **说明：** 如果使用 Embedding 功能做向量化，则无需写入向量数据。

		SparseVector	否	写入稀疏向量数据。支持写入稠密向量与稀疏向量，或使用 Embedding 写入原始文本与稀疏向量。
		Fields	否	text 字段为创建集合时，Embedding 参数 Field 对应指定的文本字段名，您可以自定义其他便于识别的字段名。 **说明：** 写入原始文本数据，系统会自动从该字段中提取原始文本信息，并将其转换为向量数据，并将原始文本以及转化后的向量数据一起存储在数据库中。 其他自定义扩展的标量字段，用于存储文档的其他信息。例如：bookName、author、page。 **说明：** 若字段为存储数据过期时间戳的标量字段，需以标准的 Unix 时间戳格式为该字段赋值，且数据类型为 uint64。具体示例，请参见 ttl_demo 。代码主要部分，如下所示。 `Fields: map[string]tcvectordb.Field{ "expire_at": {Val: time.Now().Add(10 * time.Second).Unix()} , },` • 若时间戳字段为空未指定具体的过期时间，插入的该条数据将不会执行过期删除操作，数据将永久保留。 • 若时间戳字段传入 Value 的数据类型有误或者不符合 Unix 时间戳规范时，将会报出错误信息，请根据提示信息修改。

				• 若时间戳字段指定的过期时间小于当前插入数据时的时间，则在下一个定时删除操作时，直接删除该条数据。

精确查询

最近更新时间：2025-07-01 10:19:43

接口定义

基于精确匹配的查询方式，`query()` 根据输入的文档 ID，或根据指定的查询起始位置和返回数量，精确查找与查询条件完全匹配的文档。

```
func (tcvectordb.FlatInterface) Query(ctx context.Context, databaseName
string, collectionName string, documentIds []string, params
...*tcvectordb.QueryDocumentParams) (result
*tcvectordb.QueryDocumentResult, err error)
```

使用示例

> **说明：**
> - 支持根据主键 id（Document ID），搭配自定义的标量字段的 Filter 表达式一并查找数据。
> - 支持指定查询起始位置 offset 和返回数量 limit，实现数据 SCAN 能力。

如下示例，检索满足 **documentId** 以及 **tag** 字段 Filter 条件表达式的数据。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
option := &tcvectordb.QueryDocumentParams{
    Filter: tcvectordb.NewFilter(tcvectordb.Include("tag", []string{"曹
操", "刘备"})),
    OutputFields:   []string{"id", "bookName"},
    Limit: 100,
}
documentId := []string{"0001", "0002", "0003", "0004", "0005"}
result, err := client.Query(ctx, database, collectionName, documentId,
option)
for _, doc := range result.Documents {
    log.Printf("document: %+v", doc)
}
```

参数	子参数	是否必选	参数含义	参数配置
databaseName	-	是	指定查询的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	-	是	指定查询数据的集合。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
documentIds	-	否	表示要查询的文档的所有 ID。	每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。
QueryDocumentParams	RetrieveVector	否	标识是否需要返回检索结果的向量值。	• **true**：需要。 • **false**：不需要。默认为 false。

	Filter	否	设置 Filter 表达式。	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中：
				• <field_name>：表示要过滤的字段名。
				• <operator>：表示要使用的运算符。
				  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
				  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。
				  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素

			（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter := tcvectordb.NewFilter(\`bookInfo.bookName="西游记"\`)` • <value>：表示要匹配的值。
	Limit	是	每页返回的 Document 数量。	每页返回的 Document 数量。默认为10。 • 数据类型：uint 64 • 取值范围：[1,16384] **注意：** 若使用 **query** 检索数据时，不配置 **documentIds** 和 **Filter** 参数，则必须配置 **Offset** 和 **Limit** 参数，返回从 **Offset** 开始的 **Limit** 条数据，避免遍历所有数据而浪费不必要的资源。
	Offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。	取值要求如下： • 取值：为 **Limit** 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。
	Output Fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：**

- OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。
- 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}` ，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

	Sort		指定查询结果按照特定字段升序或降序排序。数据类型为数组。	Sort:
				[]document.SortRule{
				{
				FieldName: "page",
				Direction: "asc",
				},
				}
				
				FieldName 指定特定字段，查询结果将依据该字段确定输出顺序。当前排序字段仅支持 uint64 的类型，且该字段为标量字段索引，可通过接口 DescribeCollection() 确认该字段是否为索引字段。
				
				说明：
				• 当满足条件的数据量小于或等于1000时，orderby 操作将正常生效，否则会引发错误。
				• 目前仅支持对单个字段进行排序，暂时不支持多字段排序功能。
				
				Direction 指定排序的方向。
				• desc：降序排序。
				• asc：升序排序。

出参描述

```
2024/12/31 16:19:58 QueryDocument: {Id:0004 Vector:[0.2123 0.24 0.213]
SparseVector:[] Score:0 Fields:map[bookName:三国演义 page:24]}
2024/12/31 16:19:58 QueryDocument: {Id:0003 Vector:[0.2123 0.23 0.213]
SparseVector:[] Score:0 Fields:map[bookName:三国演义 page:23]}
```

参数名	参数含义
id	Document 的 ID 信息。

Vector	Document 的向量值。
SparseVector	稀疏向量。
Score	相似性得分。查询文档信息，分值为0。
Fields	Document 自定义扩展的标量字段。例如：bookName、author 等。

版权所有：腾讯云计算（北京）有限责任公司

基于 ID 相似性检索

最近更新时间：2025-07-01 10:19:43

接口定义

基于相似度匹配的查询方式，`searchById()` 根据输入的1到 N 个文档 ID，查找存储在数据库中的相应向量，并执行相似性 TopK 检索。

```
func (tcvectordb.FlatInterface) SearchById(ctx context.Context,
databaseName string, collectionName string, documentIds []string, params
...*tcvectordb.SearchDocumentParams) (result
*tcvectordb.SearchDocumentResult, err error)
```

使用示例

如下示例，相似度检索与 **documentId** 0003 相似，且满足 **bookName** 条件表达式的 Top 2 条数据。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)

filter := tcvectordb.NewFilter(`bookName="三国演义"`)
documentId := []string{"0003"}
searchRes, err := client.SearchById(ctx, database, collectionName,
documentId, &tcvectordb.SearchDocumentParams{
    Filter:         filter,
    Params:         &tcvectordb.SearchDocParams{Ef: 100},
    RetrieveVector: false,
    Limit:          2,
})
for i, docs := range searchRes.Documents {
    log.Printf("doc %d result: ", i)
    for _, doc := range docs {
        log.Printf("document: %+v", doc)
    }
}
```

入参描述

参数	子参数	是否必选	参数含义	配置方法
databaseName	-	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	-	是	指定检索数据的集合。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
documentIds	-	是	待查询的文档 ID。	每个 ID 长度限制为[1,128]。数组元素数量最大为20。
SearchDocumentParams	Filter	否	设置查询过滤条件。	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo

			，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter := tcvectordb.NewFilter(`bookInfo.bookName="西游记"`)` • <value>：表示要匹配的值。
Params	否	指定索引查询参数。	索引类型不同，检索时，所需配置的参数不同。 • FLAT：无需指定参数。 • HNSW 类型：需配置参数 Ef，指定需要访问向量的数目。取值范围[1,32768]，默认为 10。如何设置，请参见选型指南。 • IVF 系列：需设置参数 nprobe ，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 ListCollection() 查看。
RetrieveVector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • true：需要。 • false：不需要。默认为 false。
Limit	是	指定返回最相似的 Top K 的 K 的值。	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
Radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= Radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= Radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= Radius 的文
Output Fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：**

				• OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b"。

出参描述

说明：

- 输出的 Document ID 顺序与查询时配置的参数 document_ids 输入的顺序一致。
- 每一个查询结果都返回 TopK 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。相似程度则通过 L2（欧几里得距离）、IP（内积）或 COSINE（余弦相似度）计算得出的分数来衡量，输出参数 score 表示相似性计算分数。其中，欧式距离（L2）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。

```
2023/12/20 19:10:00 doc 0 result:
2023/12/20 19:10:00 document: {Id:0003 Vector:[] Score:1
Fields:map[author:罗贯中 bookName:三国演义 page:23 segment:细作探知这个消息，
飞报吕布。 tag:[曹操 诸葛亮 刘备]]}
2023/12/20 19:10:00 document: {Id:0004 Vector:[] Score:0.999787
Fields:map[author:罗贯中 bookName:三国演义 page:24 segment:布大惊，与陈宫商
议。宫曰：“闻刘玄德新领徐州，可往投之。”布从其言，竟投徐州来。有人报知玄德。 tag:[曹
操 诸葛亮 刘备]]}
```

参数名	参数含义
Id	Document 的 ID 信息。
Vector	Document 的向量值。

Score	表示查询向量与检索结果向量之间的相似性计算分数。
Fields	Document 其他自定义的标量字段。例如：author、bookName。

版权所有：腾讯云计算（北京）有限责任公司

基于向量数据相似性检索

最近更新时间：2025-07-01 10:19:43

接口定义

基于相似度匹配的查询方式，`search()`接口用于根据输入的1到 N 向量数据，查找存储在数据库中的相似性向量，返回指定的 Top K 个最相似的文档。

```
Search(ctx context.Context, vectors [][]float32, params
...*tcvectordb.SearchDocumentParams) (result
*tcvectordb.SearchDocumentResult, err error)
```

使用示例

基于向量数据的相似性检索

如下示例，相似性检索与向量数据`{0.3123, 0.43, 0.213}`与`{0.233, 0.12, 0.97}`相似的向量数据。

```
import (
    "context"
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)

searchRes, _ := client.Search(ctx, database, collectionName, []
[]float32{
    {0.3123, 0.43, 0.213},
    {0.233, 0.12, 0.97},
}, &tcvectordb.SearchDocumentParams{
    Params:          &tcvectordb.SearchDocParams{Ef: 100},
    RetrieveVector: false,
    Limit:           10,
})
log.Printf("search by vector-----------------")
```

```
for i, docs := range searchRes.Documents {
    log.Printf("doc %d result: ", i)
    for _, doc := range docs {
        log.Printf("document: %+v", doc)
    }
}
```

基于二进制数据检索

如下示例，基于二进制数据进行相似性检索。

```
import (
    "context"
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/utils"
)
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
searchVector, _ := utils.BinaryToUint8([]byte{0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 0, 1, 1, 1, 0, 1})
searchResult, _ := client.Search(ctx, database, collectionName,
    [][]float32{searchVector}, //指定检索向量，最多指定20个
    &tcvectordb.SearchDocumentParams{
        RetrieveVector: true,
        Limit:          2,
})
// 输出相似性检索结果，检索结果为二维数组，每一位为一组返回结果，分别对应search
时指定的多个向量
for i, item := range searchResult.Documents {
    log.Printf("SearchDocumentResult, index: %d ==================",
i)
    for _, doc := range item {
        log.Printf("SearchDocument: %+v", doc)
    }
}
```

输出信息，如下所示。

```
2024/12/18 16:53:12 SearchDocumentResult, index: 0
==================
2024/12/18 16:53:12 SearchDocument: {Id:0005 Vector:[0 217]
SparseVector:[] Score:1 Fields:map[author:罗贯中 bookName:三国演义
page:25 segment:玄德曰："布乃当今英勇之士，可出迎之。"糜竺曰："吕布乃虎狼之
徒，不可收留；收则伤人矣。]}
2024/12/18 16:53:12 SearchDocument: {Id:0004 Vector:[0 228]
SparseVector:[] Score:4 Fields:map[author:罗贯中 bookName:三国演义
page:24 segment:布大惊，与陈宫商议。宫曰："闻刘玄德新领徐州，可往投之。"布从其
言，竟投徐州来。有人报知玄德。]}
```

参数	子参数	是否必选	参数含义	配置方法
databaseName	-	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	-	是	指定检索数据的集合。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
Vectors	-	是	表示要查询的向量列表。	数组元素数量最大为20。
SearchDocumentParams	Filter	否	设置查询过滤条件。	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ string：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值

			（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ json：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `filter := tcvectordb.NewFilter(`bookInfo.bookName="西游记"`)` • <value>：表示要匹配的值。
Params	否	指定索引查询参数。	索引类型不同，检索时，所需配置的参数不同。 • FLAT：无需指定参数。 • HNSW 类型：需配置参数 **Ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为 10。如何设置，请参见选型指南。 • IVF 系列：需设置参数 **nprobe**，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 ListCollection() 查看。
Retrieve Vector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • **true**：需要。 • **false**：不需要。默认为 false。

	Limit	是	指定返回最相似的 Top K 的 K 的值。	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
	Radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= Radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= Radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= Radius 的文
	OutputFields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b"。

返回信息

说明：

- 输出结果的顺序，与搜索时设置的 vectors 配置的向量值的顺序一致。
- 每一个查询结果都返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值，如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。

版权所有：腾讯云计算（北京）有限责任公司

```
search by vector----------------
2023/12/20 19:11:16 doc 0 result:
2023/12/20 19:11:16 document: {Id:0005 Vector:[] Score:0.97885
Fields:map[author:罗贯中 bookName:三国演义 page:25 segment:玄德曰："布乃当今英勇之士，可出迎之。"糜竺曰："吕布乃虎狼之徒，不可收留；收则伤人矣。 tag:[曹操 诸葛亮 刘备]]}
2023/12/20 19:11:16 document: {Id:0004 Vector:[] Score:0.975398
Fields:map[author:罗贯中 bookName:三国演义 page:24 segment:布大惊，与陈宫商议。宫曰："闻刘玄德新领徐州，可往投之。"布从其言，竟投徐州来。有人报知玄德。 tag:[曹操 诸葛亮 刘备]]}
2023/12/20 19:11:16 document: {Id:0003 Vector:[] Score:0.971423
Fields:map[author:罗贯中 bookName:三国演义 page:23 segment:细作探知这个消息，飞报吕布。 tag:[曹操 诸葛亮 刘备]]}
2023/12/20 19:11:16 document: {Id:0002 Vector:[] Score:0.966884
Fields:map[author:吴承恩 bookName:西游记 page:22 segment:正大光明，忠良善果弥深。些些狂妄天加谴，眼前不遇待时临。 tag:[孙悟空 猪八戒 唐僧]]}
2023/12/20 19:11:16 document: {Id:0001 Vector:[] Score:0.961738
Fields:map[author:吴承恩 bookName:西游记 page:21 segment:富贵功名，前缘分定，为人切莫欺心。 tag:[孙悟空 猪八戒 唐僧]]}
2023/12/20 19:11:16 doc 1 result:
2023/12/20 19:11:16 document: {Id:0001 Vector:[] Score:0.763192
Fields:map[author:吴承恩 bookName:西游记 page:21 segment:富贵功名，前缘分定，为人切莫欺心。 tag:[孙悟空 猪八戒 唐僧]]}
2023/12/20 19:11:16 document: {Id:0002 Vector:[] Score:0.754486
Fields:map[author:吴承恩 bookName:西游记 page:22 segment:正大光明，忠良善果弥深。些些狂妄天加谴，眼前不遇待时临。 tag:[孙悟空 猪八戒 唐僧]]}
2023/12/20 19:11:16 document: {Id:0003 Vector:[] Score:0.745703
Fields:map[author:罗贯中 bookName:三国演义 page:23 segment:细作探知这个消息，飞报吕布。 tag:[曹操 诸葛亮 刘备]]}
2023/12/20 19:11:16 document: {Id:0004 Vector:[] Score:0.736874
Fields:map[author:罗贯中 bookName:三国演义 page:24 segment:布大惊，与陈宫商议。宫曰："闻刘玄德新领徐州，可往投之。"布从其言，竟投徐州来。有人报知玄德。 tag:[曹操 诸葛亮 刘备]]}
2023/12/20 19:11:16 document: {Id:0005 Vector:[] Score:0.728028
Fields:map[author:罗贯中 bookName:三国演义 page:25 segment:玄德曰："布乃当今英勇之士，可出迎之。"糜竺曰："吕布乃虎狼之徒，不可收留；收则伤人矣。 tag:[曹操 诸葛亮 刘备]]}
```

参数名	参数含义
Id	Document 的 ID 信息。

Vector	Document 的向量值。
Score	表示查询向量与检索结果向量之间的相似性计算分数。欧式距离（L2）与汉明距离（Hamming Distance）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。
Fields	Document 其他自定义的标量字段。例如：author、bookName。

版权所有：腾讯云计算（北京）有限责任公司

基于原始文本相似性检索

最近更新时间：2025-07-01 10:19:43

接口定义

若在创建 Collection 时，已配置 Embedding 模型，则可使用 `SearchByText()` 接口输入原始文本，在数据库中检索与原始文档相似的文档，返回指定的 Top K 个最相似的 Document。

```
SearchByText(ctx context.Context, databaseName string, collectionName
string, text map[string][]string, params ...*SearchDocumentParams)
(result *SearchDocumentResult, err error)
```

使用示例

如下示例，相似性检索与原始文本 `吕布` ，相似的 Top 2 条数据。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    embeddingCollection = "go-sdk-test-emcoll"
)
searchRes, err := client.SearchByText(ctx, database,
embeddingCollection, map[string][]string{"segment": {"吕布"}},
&tcvectordb.SearchDocumentParams{
    Params:         &tcvectordb.SearchDocParams{Ef: 100}, // 若使用HNSW索
引，则需要指定参数ef，ef越大，召回率越高，但也会影响检索速度
    RetrieveVector: false,                                // 是否需要返回向
量字段，False：不返回，True：返回
    Limit:          2,                                    // 指定 Top K
的 K 值
})
log.Printf("searchByText-----------------")
for i, docs := range searchRes.Documents {
    log.Printf("doc %d result: ", i)
    for _, doc := range docs {
        log.Printf("document: %+v", doc)
    }
}
```

入参描述

参数	子参数	是否必选	参数含义	配置方法
databaseName	-	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	-	是	指定检索数据的集合。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
text	-	是	待检索的文本信息。	输入文本信息，用于检索与该文本信息相似的数据。 • 类型：字符串数组。 • 范围：数组元素最大批量为20 。
SearchDocumentParams	Filter	否	设置查询过滤条件。	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问

				Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `filter := tcvectordb.NewFilter(`bookInfo.bookName="西游记"`)` • <value>：表示要匹配的值。
	Params	否	指定索引查询参数。	索引类型不同，检索时，所需配置的参数不同。 • FLAT：无需指定参数。 • HNSW 类型：需配置参数 Ef，指定需要访问向量的数目。取值范围[1,32768]，默认为 10。如何设置，请参见选型指南。 • IVF 系列：需设置参数 nprobe，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 ListCollection() 查看。
	RetrieveVector	否	标识是否需要返回检索结果的向量值。	取值如下所示： • true：需要。 • false：不需要。默认为 false。
	Limit	是	指定返回最相似的 Top K 的 K 的值。	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。
	Radius	否	指定相似性检索半径。	• IP：取值范围 (−∞, +∞)，返回相似性计算得分 score >= Radius 的文档。 • COSINE：取值范围 [−1, 1]，返回相似性计算得分 score >= Radius 的文档。 • L2：取值范围 [0, +∞)，返回相似性计算得分 score <= Radius 的文
	OutputFields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。

				说明： • OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}` ，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

```
searchByText----------------
2023/12/20 19:12:18 doc 0 result:
2023/12/20 19:12:18 document: {Id:0005 Vector:[] Score:0.798083
Fields:map[author:罗贯中 bookName:三国演义 page:25 segment:玄德曰："布乃当今英勇之士，可出迎之。"糜竺曰："吕布乃虎狼之徒，不可收留；收则伤人矣。]}
2023/12/20 19:12:18 document: {Id:0003 Vector:[] Score:0.793226
Fields:map[author:罗贯中 bookName:三国演义 page:23 segment:细作探知这个消息，飞报吕布。]}
```

参数名	参数含义
Id	Document 的 ID 信息。
Vector	Document 的向量值。
Score	表示查询向量与检索结果向量之间的相似性计算分数。
Fields	segment：开启 Embedding 功能，在建表时，指定文本信息的字段名，则显示该字段。
	Document 其他自定义的标量字段。例如：author、bookName。

版权所有：腾讯云计算（北京）有限责任公司

单路稀疏向量相似性检索

最近更新时间：2025-07-01 10:19:42

接口定义

本接口 `FullTextSearch()` 基于稀疏向量实现全文检索，对查询词与文档内容进行匹配，并按相关度排序结果。

```
FullTextSearch(ctx context.Context, databaseName string, collectionName
string, params tcvectordb.FullTextSearchDocumentParams) (result
*tcvectordb.SearchDocumentResult, err error)
```

使用示例

说明：

- 如需使用稀疏向量，至少需要升级 Go SDK 至 v1.5.1 版本，推荐使用1.5.4版本使用最新的 TerminateAfter、CutoffFrequency 等新特性，使得召回效果和性能更佳。
- 腾讯云向量数据库团队提供了一款稀疏向量工具包，旨在帮助用户高效生成稀疏向量。具体信息，请参见 工具介绍 。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
bm25, err :=
encoder.NewBM25Encoder(&encoder.BM25EncoderParams{Bm25Language: "zh"})
if err != nil {
    log.Fatalf(err.Error())
}
sparseVec, err := bm25.EncodeQuery("什么是向量数据库")
if err != nil {
    log.Fatalf(err.Error())
}
keywordSearch := &tcvectordb.MatchOption{
    FieldName: "sparse_vector",
    Data:       sparseVecs,
    TerminateAfter:  4000,
    CutoffFrequency: 0.1,
}
```

```
limit := 5
searchRes, err := client.FullTextSearch(ctx, database, collectionName,
tcvectordb.FullTextSearchDocumentParams{
    Match:        keywordSearch,
    Limit:        &limit,
    OutputFields: []string{"id", "sparse_vector"},
})
if err != nil {
    return err
}

// 输出相似性检索结果，检索结果为二维数组，每一位为一组返回结果，分别对应search时指定
的多个向量
for i, item := range searchRes.Documents {
    log.Printf("SearchDocumentResult, index: %d ==================", i)
    for _, doc := range item {
        log.Printf("SearchDocumentResult: %+v", doc)
    }
}
```

入参描述

参数	子参数	是否必选	参数含义	配置方法及要求
databaseName	-	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	-	是	指定检索数据的集合。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
FullTextSearchDocumentP	Match	否	稀疏向量检索配置。	• **FieldName**：稀疏向量的字段名，例如：sparse_vector。 • **Data**：检索的稀疏向量，当前仅支持输入一个稀疏向量。 • **TerminateAfter**：设置检索数量上限，即检索数量超过 TerminateAfter 值之后，提前终止检索行为，可以有效

arams				的控制速率，在数据量较大的情况下，推荐经验值：4000。 • **CutoffFrequency**：设置高频词出现的频次或频率上限，即词频大于 CutoffFrequency 的值之后，该关键词被忽略。 ○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。 ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。 **说明：** CutoffFrequency 可以在排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：0.1。
	Filter	否	设置 filter	设置标量字段的 Filter 表达式。格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter := tcvectordb.NewFilter(`bookInfo.bookName="西游记"`)`

				• <value>：表示要匹配的值。
	RetrieveVector	否	标识是否需要返回检索结果的向量值。	• true：需要。 • false：不需要。默认为 false。
	Limit	是	指定返回最相似的 Top K 的 K 的值。	K 为大于0的正整数。
	OutputFields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

```
2025/06/17 20:00:27 ------------------------------ FullTextSearch ------
------------------------
2025/06/17 20:00:27 SearchDocumentResult, index: 0 ==================
2025/06/17 20:00:27 SearchDocument: {Id:0000 Vector:[] SparseVector:[]
Score:0.36539772 Fields:map[text:腾讯云向量数据库（Tencent Cloud VectorDB）
是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或
其他机器学习模型生成的大量多维嵌入向量。]}
2025/06/17 20:00:27 SearchDocument: {Id:0001 Vector:[] SparseVector:[]
Score:0.18224944 Fields:map[text:作为专门为处理输入向量查询而设计的数据库，它支
持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，高达百万级 QPS 及毫秒级查询
延迟。]}
```

```
2025/06/17 20:00:27 SearchDocument: {Id:0004 Vector:[] SparseVector:[]
Score:0.17610866 Fields:map[text:腾讯云向量数据库可以和大语言模型 LLM 配合使
用。企业的私域数据在经过文本分割、向量化后，可以存储在腾讯云向量数据库中，构建起企业专
属的外部知识库，从而在后续的检索任务中，为大模型提供提示信息，辅助大模型生成更加准确的
答案。]}
2025/06/17 20:00:27 SearchDocument: {Id:0003 Vector:[] SparseVector:[]
Score:0.16722518 Fields:map[text:腾讯云向量数据库（Tencent Cloud VectorDB）
作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单
易用、稳定可靠等方面体现出显著优势。 ]}
```

参数名	参数含义
id	Document 的 ID 信息。
sparse_vector	稀疏向量。
score	表示查询稀疏向量与检索结果向量之间的相似性计算分数。
other_scalar_field	Document 其他自定义的标量字段。

稠密向量与稀疏向量高阶混合检索

最近更新时间：2025-07-01 10:19:43

接口定义

本接口 `HybridSearch()` 基于稠密向量与稀疏向量，对查询向量与文档内容进行匹配，并按相关度排序结果。

```
HybridSearch(ctx context.Context, databaseName string, collectionName
string, params HybridSearchDocumentParams) (result
*SearchDocumentResult, err error)
```

使用示例

说明：

- 如需使用稀疏向量，至少需要升级 Go SDK 至 v1.5.1 版本，推荐使用1.5.4版本使用最新的 **TerminateAfter、CutoffFrequency** 等新特性，使得召回效果和性能更佳。
- 腾讯云向量数据库团队提供了一款稀疏向量工具包，旨在帮助用户高效生成稀疏向量。具体信息，请参见 工具介绍 。

将文本转换稀疏向量检索

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
bm25, err :=
encoder.NewBM25Encoder(&encoder.BM25EncoderParams{Bm25Language:
"zh"})
if err != nil {
    log.Fatalf(err.Error())
}
sparseVec, err := bm25.EncodeQuery("腾讯云向量数据库")
if err != nil {
    log.Fatalf(err.Error())
}
annSearch := &tcvectordb.AnnParam{
    FieldName: "vector",
```

```
    Data:       []float32{0.2123, 0.21, 0.213},
}
keywordSearch := &tcvectordb.MatchOption{
    FieldName: "sparse_vector",
    Data:      sparseVecs,
    TerminateAfter:  4000,
    CutoffFrequency: 0.1,
}
limit := 2
searchRes, err := client.HybridSearch(ctx, database, collectionName,
tcvectordb.HybridSearchDocumentParams{
    AnnParams: []*tcvectordb.AnnParam{annSearch},
    Match:     []*tcvectordb.MatchOption{keywordSearch},
    // rerank也支持rrf，使用方式见下
    // Rerank: &tcvectordb.RerankOption{
    //  Method:    tcvectordb.RerankRrf,
    //  RrfK: 1,
    // },
    Rerank: &tcvectordb.RerankOption{
       Method:    "weighted",
       FieldList: []string{"vector", "sparse_vector"},
       Weight:    []float32{0.6, 0.4},
    },
    Limit:        &limit,
    OutputFields: []string{"id", "sparse_vector"},
})
if err != nil {
    return err
}

// 输出相似性检索结果，检索结果为二维数组，每一位为一组返回结果，分别对应search
时指定的多个向量
for i, item := range searchRes.Documents {
    log.Printf("HybridSearchDocumentResult, index: %d
==================", i)
    for _, doc := range item {
       log.Printf("HybridSearchDocument: %+v", doc)
    }
}
```

直接写入稀疏向量检索

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
annSearch := &tcvectordb.AnnParam{
    FieldName: "vector",
    Data:      []float32{0.2123, 0.21, 0.213},
}
keywordSearch := &tcvectordb.MatchOption{
    FieldName: "sparse_vector",
    Data: []encoder.SparseVecItem{
        {TermId: 1172076521, Score: 0.71296215},
        {TermId: 3434399993, Score: 0.71296215},
    },
}
limit := 2
searchRes, err := client.HybridSearch(ctx, database, collectionName,
tcvectordb.HybridSearchDocumentParams{
    AnnParams: []*tcvectordb.AnnParam{annSearch},
    Match:     []*tcvectordb.MatchOption{keywordSearch},
    // rerank也支持rrf，使用方式见下
    // Rerank: &tcvectordb.RerankOption{
    //  Method:    tcvectordb.RerankRrf,
    //  RrfK: 1,
    // },
    Rerank: &tcvectordb.RerankOption{
        Method:    tcvectordb.RerankWeighted,
        FieldList: []string{"vector", "sparse_vector"},
        Weight:    []float32{0.1, 0.9},
    },
    Limit:        &limit,
    OutputFields: []string{"id", "sparse_vector"},
})
if err != nil {
    return err
}
// 输出相似性检索结果，检索结果为二维数组，每一位为一组返回结果，分别对应search
时指定的多个向量
for i, item := range searchRes.Documents {
```

```
    log.Printf("HybridSearchDocumentResult, index: %d
==================", i)
    for _, doc := range item {
        log.Printf("HybridSearchDocument: %+v", doc)
    }
}
```

Embedding 文本与稀疏向量检索

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
searchText := "腾讯云向量数据库"
annSearch := &tcvectordb.AnnParam{
    FieldName: "text",
    Data:      searchText,
}
sparseVec, _ := bm25.EncodeQuery(searchText)
keywordSearch := &tcvectordb.MatchOption{
    FieldName:        "sparse_vector",
    Data:             sparseVec,
    TerminateAfter:   4000,
    CutoffFrequency: 0.1,
}
limit := 2
searchRes, err := coll.HybridSearch(ctx, database, collectionName,
tcvectordb.HybridSearchDocumentParams{
    AnnParams: []*tcvectordb.AnnParam{annSearch},
    Match:     []*tcvectordb.MatchOption{keywordSearch},
    Rerank: &tcvectordb.RerankOption{
        Method:    tcvectordb.RerankWeighted,
        FieldList: []string{"vector", "sparse_vector"},
        Weight:    []float32{0.1, 0.9},
    },
    Limit:        &limit,
    OutputFields: []string{"id", "sparse_vector"},
})
if err != nil {
```

```
    panic(err)
}
// 输出相似性检索结果，检索结果为二维数组，每一位为一组返回结果，分别对应search
时指定的多个向量
for i, item := range searchRes.Documents {
    log.Printf("HybridSearchDocumentResult, index: %d
==================", i)
    for _, doc := range item {
        log.Printf("HybridSearchDocument: %+v", doc)
    }
}
```

入参描述

参数	子参数	是否必选	参数含义	配置方法及要求
databaseName	-	是	指定检索的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	-	是	指定检索数据的集合。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
HybridSearchDocumentParams	AnnParam	是	设置向量检索信息。	• **FieldName**：检索的字段名，支持设置为：vector 或 text。 **说明：** 其中，vector 为向向量数据的字段名，不可更改；text 为创建 Collection 时，开启 Embedding 功能自定义的原始文本字段名，请以实际定义为准，二者只能选择其中之一。 • **Data**：检索的数据列表，当前仅支持输入一条向量或原始文本。

				• **Params**：检索参数，与索引类型相关。
				  ○ HNSW 类型：需配置参数 **ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为10。
				  ○ IVF 系列：需设置参数 **nprobe**，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置。
				• **Limit**：指定稠密向量返回的数量。

	Match	否	稀疏向量检索配置。	• **FieldName**：稀疏向量的字段名，例如：sparse_vector。
				• **Data**：检索的稀疏向量，当前仅支持输入一个稀疏向量。
				• **Limit**：指定返回的数量。
				• **TerminateAfter**：设置检索数量上限，即检索数量超过 TerminateAfter 值之后，提前终止检索行为，可以有效的控制速率，在数据量较大的情况下，推荐经验值：4000。
				• **CutoffFrequency**：设置高频词出现的频次或频率上限，即词频大于 CutoffFrequency 的值之后，该关键词被忽略。
				  ○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。
				  ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。
				说明：
				CutoffFrequency 可以在排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：0.1。

	Rerank	否	检索中重排序（rerank）的配置。	参数 Method 指定 Rerank 的方法，枚举如下值：
				• **weighted**：基于不同字段的加权组合来进行排序。
				  ○ field_list：列出用于加权计算的字段列表。例如，field_list=: ['vector', 'sparse_vector'] 表示密集型与稀疏型加权计算。
				  ○ weight：定义了 "field_list" 中每个字段的权重，例如，`weight=[0.9, 0.1]` 表示"vector" 字段的权重是0.9，"sparse_vector" 字段的权重是0.1。
				• **rrf**：一种融合多个搜索方法的排名结果，通过为每个文档分配倒数排名分数并合并这些分数来生成一个新

				的综合排名，从而提高搜索结果的准确性和相关度。k 是一个用于计算倒数排名分数的常数，其作用是调整分数计算公式，以控制排名分数的分布。例如：`rerank=RRFRerank(k=1)` 。
	Filter	否	设置 filter	设置标量字段的 Filter 表达式。格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter := tcvectordb.NewFilter(`bookInfo.bookName="西游记"`)` • <value>：表示要匹配的值。
	RetrieveVector	否	标识是否需要返回检索结果的向量值。	• **true**：需要。 • **false**：不需要。默认为 false。

	Limit	是	指定返回最相似的 Top K 的 K 的值。	K 为大于0的正整数。
	Output Fields	否	配置需返回的字段。	以数组形式配置需返回的字段。若不配置，返回所有字段。 **说明：** • OutputFields 与 RetrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，OutputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，OutputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

```
2024/09/29 19:51:56 HybridSearchDocumentResult, index: 0
==================
2024/09/29 19:51:56 HybridSearchDocument: {Id:0003 Vector:[]
SparseVector:[{TermId:21 Score:0.9827807} {TermId:51
Score:0.5527807} {TermId:1001 Score:0.445}] Score:0.12634 Fields:map[]}
2024/09/29 19:51:56 HybridSearchDocument: {Id:0002 Vector:[]
SparseVector:[{TermId:21 Score:0.9727807} {TermId:51
Score:0.5427807} {TermId:1001 Score:0.444}] Score:0.12376 Fields:map[]}
```

参数名	参数含义
id	Document 的 ID 信息。
vector	Document 的向量值。
SparseVector	稀疏向量。
score	表示查询向量与检索结果向量之间的相似性计算分数。

author、page、section	Document 其他自定义的标量字段。

版权所有：腾讯云计算（北京）有限责任公司

更新 Document 数据

最近更新时间：2025-07-01 10:19:43

接口定义

`update()` 接口用于对通过主键（Document ID）与 Filter 表达式过滤检索 Document，对 Document 的部分字段进行更新，同时支持新增字段。

```
Update(ctx context.Context, databaseName string, collectionName string,
param UpdateDocumentParams) (result *UpdateDocumentResult, err error)
```

使用示例

如下示例，更新 **id** 为0001、0003，并满足 **bookName** 条件表达式的 Document 字段 page 的值。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
result, _ := client.Update(ctx, database, collectionName,
tcvectordb.UpdateDocumentParams{
    QueryIds:    []string{"0001", "0003"},
    QueryFilter: tcvectordb.NewFilter(`bookName="三国演义"`),
    UpdateFields: map[string]tcvectordb.Field{
        "page": {Val: 24},
    },
})
log.Printf("update affect count: %d", result.AffectedCount)
```

入参描述

参数名	子参数	是否必选	参数含义
database Name	-	是	指定更新的数据库名。Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

collectionName	-	是	指定更新数据的集合。Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
QueryIds	DocumentIds	是	设置需更新的文档 ID。每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。
QueryFilter	Filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `filter := tcvectordb.NewFilter(` `bookInfo.bookName="西游记"` `)` • <value>：表示要匹配的值。

UpdateFields	vector	否	更新 vector 字段的向量数据。 **说明：** 如果 Collection 在创建时，未配置 Embedding 参数，或者该实例并未开通 Embedding 功能，则只能配置该参数，输入向量数据，更新数据。
	text	否	Embedding 模型输入文本的字段名。该字段在创建 Collection 时定义。本示例为 text 。 • **string**：字符型。 • **float**：浮点型数据。 **说明：** 如果 Collection 在创建时，已配置 Embedding 参数，则只能配置该参数，依据输入的文本信息更新向量数据，并将文本字段与向量数据更新存于数据库。
	old_field	否	当前已存在的字段，更新字段对应的数据。 • 类型：string • 字符长度要求：[1,256]。
	new_field	否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。

出参描述

```
update affect count: &{AffectedCount:1}
```

参数名	参数含义
AffectedCount	更新的文档数量。如果该参数返回的值为0，说明更新无效。

删除 Document 数据

最近更新时间：2025-07-01 10:19:43

接口定义

`delete()` 接口用于删除指定 id（Document ID）的文档，且支持设置 Filter 表达式，删除满足 Filter 表达式的数据。

```
Delete(ctx context.Context, databaseName string, collectionName string,
param DeleteDocumentParams) (result *UpdateDocumentResult, err error)
```

接口约束

> **说明：**
> 索引类型为 FLAT，在1.6及之后版本，支持删除操作，而1.6之前版本，则不支持删除操作。

使用示例

如下示例，删除 **id** 为0001、0003，并满足 **bookName** 条件表达式的数据。

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
result, _ := client.Delete(ctx, database, collectionName,
tcvectordb.DeleteDocumentParams{
    DocumentIds: []string{"0001", "0003"},
    Filter:      tcvectordb.NewFilter(`bookName="西游记"`),
    Limit1:      1
})
log.Printf("delete affect count: %d", result.AffectedCount)
```

入参描述

DeleteDocumentParams 该参数的含义如下表。

参数名	是否必选	参数含义	配置方法

databaseName	是	指定删除的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collectionName	是	指定删除数据的集合。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
DocumentIds	否	表示要删除的 Document 的 ID。	• ID 长度限制为[1,128]。 • 批量删除，数据元素最大值为20。
Filter	否	设置 Filter 表达式。	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。

			``` filter := tcvectordb.NewFilter(`bookInfo.bookName="西游记"`) ``` • <value>：表示要匹配的值。
Limit	否	控制删除文档的数量。	若不设置该参数，最大删除16384条数据。 • 数据类型：整型。 • 取值范围：[1,16384]。

出参描述

```
delete affect count: &{AffectedCount:1}
```

参数名	参数含义
AffectedCount	影响行数，即为删除的文档数量。

查询文档数量

最近更新时间：2025-07-01 10:19:43

接口定义

`Count()` 接口用于查询集合中的文档数量。

```
Count(ctx context.Context, params ...tcvectordb.CountDocumentParams)
(*tcvectordb.CountDocumentResult, error)
```

使用示例

```
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
result, err = client.Count(ctx, database, collectionName,
tcvectordb.CountDocumentParams{
        CountFilter: tcvectordb.NewFilter(`bookName="腾讯云向量数据库"`),
})
log.Printf("affect count: %d", result.Count)
```

入参描述

参数	是否必选	参数含义	参数配置
database	是	指定数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	是	指定集合名。	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
CountDocume	否	使用创建 Collection 时指定的 Filter	Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中

ntParams		索引的字段给设置过滤表达式。	• <field_name>：表示要过滤的字段名。
			• <operator>：表示要使用的运算符。
			  ○ **string**：匹配单个字符串值（=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
			  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如，expired _time > 1623388524。
			  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
			  ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。
			``
			filter := tcvectordb.NewFilter(`bookInfo.bookName="西游记"`)
			``
			• <value>：表示要匹配的值。

出参描述

检索结果，如下所示。

```
affect count: 2
```

参数名	参数含义
count	集合中的文档数量。

上传文件

最近更新时间：2025-05-26 19:31:32

接口定义

UploadFile() 接口用于上传文件于 Base 类向量数据库。文件上传后，其内容将被直接拆分并写入指定的 Collection，而非 CollectionView。

> **说明：**
> 当前支持导入 Collection 的文件类型包含： Markdown、PDF、Word、PPT。其中，仅 PDF 类型支持解析图片信息。

```
func (tcvectordb.FlatInterface) UploadFile(ctx context.Context,
databaseName string, collectionName string, param
tcvectordb.UploadFileParams) (result *tcvectordb.UploadFileResult, err
error)
```

约束限制

- 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
- 该接口当前不支持使用别名获取文件的上传路径与授权。

> **注意：**
> 在创建存储文件内容的 Collection 时，索引字段的设计需遵循以下限制与建议，否则可能导致上传文件失败。
>
> - 文件名字段（file_name）：该字段**必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。字段名可以自定义，在上传文件时，需通过参数 FieldMappings 的 filename 映射自定义的字段名。
> - 文件内容字段（text）：该字段用于存储知识点的原始文本内容。由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。字段名可以自定义，但需通过参数 FieldMappings 的 text 映射自定义的字段名。
> - 文件图片信息字段（image_list）：该字段存放 PDF 文件中图片的 Key 列表。在原始检索出的文本块中，图片位置将以 {key} 的形式进行占位。通过调用接口 GetImageUrl，可以获取图片 URL 地址列表，这些地址与 Key 一一对应，从而实现将原始 PDF 文档恢复并展示为 HTML 格式。
>   - **不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。

- 字段名可以自定义，但需要在上传文件时通过参数 FieldMappings 的 imageList 映射到自定义的字段名。

使用示例

说明：
如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api"
    "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/ai_document_set"
)
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
appendKeywordsToChunk := false
appendTitleToChunk := false
localFilePath := "/data/home/********/腾讯云向量数据库.pdf"
param := tcvectordb.UploadFileParams{
    LocalFilePath: localFilePath,
    SplitterPreprocess: ai_document_set.DocumentSplitterPreprocess{
        AppendKeywordsToChunk: &appendKeywordsToChunk,
        AppendTitleToChunk:    &appendTitleToChunk,
    },
    ParsingProcess: &api.ParsingProcess{
        ParsingType: "AlgorithmParsing",
    },
    EmbeddingModel: "bge-base-zh",
    FieldMappings: map[string]string{
        "filename":   "file_name",
        "text":       "text",
        "imageList":  "image_list",
        "chunkNum":   "chunk_num",
        "sectionNum": "section_num",
```

```
    },
    MetaData: map[string]interface{}{
        "author": "sam",
    },
}
result, _ := client.UploadFile(ctx, database, collectionName, param)
log.Printf("UploadFile result: %+v", result)
```

入参描述

参数名	子参数	是否必选	参数含义
databaseName	-	是	指定 Database 名称。
collectionName	-	是	指定 Collection 的名称。
FileName	-	否	指定文件名。若不指定，从 local_file_path 获取文件名。
LocalFilePath	-	是	本地上传文件路径。
MetaData	-	否	文件的 Metadata 元数据信息，可自定义扩展字段。例如：author、tags 等。 • 上传文件时，可为创建 CollectionView 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。
SplitterPreprocess	AppendTitleToChunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。默认值为 false。 • **true**：将段落 Title 追加到切分后的段落。
	AppendKeywordsToChunk	否	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 **keywords** 追加到切分后的段落。默认值为 true。

	ChunkSplitter	否	以正则表达式的方式配置文档拆分方式，如下：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
ParsingProcess	ParsingType	否	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
EmbeddingModel	-	是	指定使用的 Embedding 模型的名称。您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，参见 Embedding 介绍 。取值如下所示： • **bge-large-zh-v1.5**：适用中文，1024维，推荐使用。 • **bge-base-zh-v1.5**：适用中文，768维。 • **bge-large-zh**：适用中文，1024维。 • **bge-base-zh**：适用中文，768维。 • **m3e-base**：适用中文，768维。 • **e5-large-v2**：适用英文，1024维。 • **text2vec-large-chinese**：适用中文，1024维。 • **multilingual-e5-base**：适用于多种语言类型，768维。 • **BAAI/bge-m3**：适用于多种语言类型，1024维。
FieldMappings	-	是	指定文件字段在数据库中映射关系。 • **filename**：指定文件名映射的字段。例如："filename": "file_name"，其中，file_name 为用户自定义字段，以在数据库中存储文件名数据。 **注意：** 必须定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。 • **text**：指定文件内容文本映射的字段，存储知识点的原始文本内容。例如："text": "text"，其中，text 为用户自定义字段，以在数据库中存储文件的原始文本内容。 **说明：**

由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。

- **imageList**：指定图片信息存放的字段。例如："imageList": "image_list"，其中，image_list 为用户自定义的字段，以在数据库中存储图片 Key 列表。

  说明：
  不建议为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。

- **chunkNum**：指定数据库中存放文件切分语块序号的字段。例如："chunkNum": "chunk_num"，其中，chunk_num 为用户自定义字段，其值为唯一标识语块片段的整数。

  说明：
  在创建 Collection 时若为该字段（如 chunk_num）创建索引，则查询文件内容时可通过 Filter 表达式指定语块范围，快速定位目标文件内容。

- **sectionNum**：指定数据库中用于存储文件段落序号的字段名。例如："sectionNum": "section_num"，其中，section_num 为用户自定义的字段，其值为唯一标识段落的整数。

  说明：
  在创建 Collection 时若为该字段（如 section_num）创建索引，则查询文件内容时可通过 Filter 表达式指定段落范围内的语块，快速定位目标文件内容。

获取文件图片 URL

最近更新时间：2025-03-04 20:25:12

接口定义

GetImageUrl() 接口用于在使用 AI 套件上传 PDF 类型的文件后，获取图片 URL 地址列表。

> **说明：**
> 当在数据库的 Collection 中上传 PDF 文件时，系统会提取 PDF 中的图片信息，并将这些信息以 Key 列表的形式存储。在检索出的原始文本块中，图片的位置会以 {key} 的形式进行占位。通过调用接口 **GetImageUrl()**，可以获取到与这些 Key 一一对应的图片 URL 地址列表。利用这种映射关系，可以将原始 PDF 文档的内容恢复并以 HTML 格式进行展示。

```
GetImageUrl(ctx context.Context, databaseName string, collectionName
string, param tcvectordb.GetImageUrlParams) (result
*tcvectordb.GetImageUrlResult, err error)
```

使用示例

```
mport (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api"
    "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/ai_document_set"
)
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-coll"
)
filename := "腾讯云向量数据库.pdf"
res, _ := client.GetImageUrl(ctx, database, collectionName,
tcvectordb.GetImageUrlParams{
        FileName:    filename,
        DocumentIds: []string{"1342412********9596",
"1342412********9597"},
})
```

```
for _, docImages := range res.Images {
    for _, docImage := range docImages {
        log.Printf("docImage: %+v", docImage)
    }
}
```

入参描述

参数	是否必选	参数含义	配置方法
database_name	是	指定查询的数据库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection_name	是	指定查询的集合名	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
document_ids	是	表示要查询的文档的所有ID。	• 每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。 • 文件上传成功之后，可使用 query 查询文件的 ID 信息。 `res = client.query( database_name='db-test', collection_name='coll-file-test', limit=100, )`
file_name	是	指定文件名称。	-

出参描述

```
 "docImage": {
    "DocumentId": 1342412866867179596,
    "ImageName": "aa23511c-e722-4c2c-a74f-***********.png",
    "ImageUrl": "https://vecterdb***********.cos.ap-
guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-
test-6/****************************************"
  }
  "docImage": {
    "DocumentId": 1342412866867179597,
    "ImageName": "aa23511c-e722-4c2c-a74f-***********.png",
    "ImageUrl": "https://vecterdb***********.cos.ap-
guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-
test-6/****************************************"
  }
```

参数名	参数含义
documentId	Document 的 ID 信息。
imageName	图片名。
imageUrl	图片 URL。

管理 DocumentSet 数据
基于文件写入数据

最近更新时间：2025-07-01 10:19:43

接口定义

`LoadAndSplitTextParams()` 接口用于给已创建的 AI 类集合视图中上传文件写入数据。

```
LoadAndSplitText(ctx context.Context, param
tcvectordb.LoadAndSplitTextParams) (result
*tcvectordb.LoadAndSplitTextResult, err error)
```

约束限制

- 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
- 该接口当前不支持使用别名替换集合视图上传文件。

使用示例

> **说明：**
> 如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
import (
    "context"
    "encoding/json"
    "log"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api"
    "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/ai_document_set"
    collection_view "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/collection_view"
)
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
```

```
col := client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
metaData := map[string]interface{}{
        // 元数据只支持string、uint64类型的值
          "test_str": "v1",
          "fileKey":  1024,
          "author":   "sam",
    }

appendTitleToChunk := false
appendKeywordsToChunk := true

result, _ := col.LoadAndSplitText(ctx,
tcvectordb.LoadAndSplitTextParams{
    DocumentSetName: "腾讯云向量数据库.pdf",
    // Reader:           fd,
    LocalFilePath: "../example/腾讯云向量数据库.pdf",
    MetaData:       metaData,
    SplitterPreprocess: ai_document_set.DocumentSplitterPreprocess{
            AppendTitleToChunk:    &appendTitleToChunk,
            AppendKeywordsToChunk: &appendKeywordsToChunk,
    },
    ParsingProcess: &api.ParsingProcess{
            ParsingType: string(tcvectordb.VisionModelParsing),
    },
})
log.Printf("LoadAndSplitText success: %+v", result)
```

入参描述

参数名	子参数	是否必选	参数含义
LocalFilePath	-	是	本地上传文件路径。
DocumentSetName	-	否	存储在向量数据库中的文件名。若不设置该参数，则默认使用 LocalFilePath 中的文件名。

SplitterPreprocess	AppendTitleToChunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。默认值为 false。 • **true**：将段落 title 追加到切分后的段落。
	AppendKeywordsToChunk	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 keywords 追加到切分后的段落。默认值为 true。
	ChunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
ParsingProcess	ParsingType	否	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
`MetaData`	-	否	文件的 Metadata 元数据信息，可自定义扩展字段。例如：author_name、filekey 等。 • 上传文件时，可为创建 CollectionView 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。

出参描述

```
{
  "GetCosTmpSecretResult": {
    "DocumentSetId": "131383079**********",
```

```
    "DocumentSetName": "腾讯云向量数据库.pdf",
    "CosEndpoint": "https://vecterdb-backup-gz-********.cos.ap-
guangzhou.myqcloud.com",
    "CosRegion": "ap-guangzhou",
    "CosBucket": "vecterdb-backup-gz-**********",
    "UploadPath": "embedding_file/vdb-9eyhlf2m/go-sdk-test-ai-db/go-sdk-
test-ai-coll-***********/腾讯云向量数据库.pdf",
    "TmpSecretID":
"************************************************************",
    "TmpSecretKey": "8ZkQpR1KTgAwM7R65+5RVd8pPOv0zFnkq8svRsWEU/I=",
    "SessionToken": "*********************************",
    "Expiration": "2024-12-04T11:44:31Z",
    "ExpiredTime": 1733312671,
    "MaxSupportContentLength": 10485760
  }
}
```

参数名	参数含义
CosEndpoint	腾讯云对象存储（COS）的服务端点（Endpoint），即 COS 服务的访问地址。
cosBucket	COS 服务端存储桶名称。
cosRegion	COS 服务端存储桶所属地域。
uploadPath	依据数据库名、集合名、文件名拼接生成的 COS 端存放路径。
TmpSecretId	密钥 ID。
TmpSecretKey	密钥信息。
Token	Token 信息。
maxSupportContentLength	限制上传文件的最大字节数。
DocumentSetName	文件存储于数据库中的名称。
documentSetId	COS 给文件分配的 ID 信息。

获取文件内容

最近更新时间：2025-05-12 18:23:12

接口定义

- `GetDocumentSetByName()`：根据文件名查询文件内容。
- `GetDocumentSetById()`：根据文件 ID 查询文件内容。

GetDocumentSetByName()

```
GetDocumentSetByName(ctx context.Context, documentSetName string)
(*tcvectordb.GetAIDocumentSetResult, error)
```

GetDocumentSetById()

```
GetDocumentSetById(ctx context.Context, documentSetId string)
(*tcvectordb.GetAIDocumentSetResult, error)
```

使用示例

> **注意：**
> 使用该接口获取单个文档的 chunk 数量上限为1000，若超过此限制，接口将返回错误提示信息，并提示可使用 获取 Chunks 接口，结合 offset（起始位置）和 limit（每页数量）参数分批获取文档内容。

使用文件名获取文件内容

如下示例，获取文件名为 `腾讯云向量数据库.pdf` 的文件内容。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName  = "go-sdk-test-ai-coll"
```

```
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, _ := col.GetDocumentSetByName(ctx, "腾讯云向量数据库.pdf")
log.Printf("GetDocumentSetByName success: %+v", result)
```

使用文件 ID 获取文件内容

如下示例，指定文件 ID ，查询文件 ID 对应的文件内容。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
res, _ := col.GetDocumentSetById(ctx, "11938357******")
log.Printf("GetDocumentSetById success: %+v", res)
```

入参描述

参数名	是否必选	参数含义	获取方式
DocumentSetId	否	文件上传在数据库之后，系统分配的文件 ID	使用文件名获取文件 ID 之后，可使用文件 ID 查询文件
DocumentSetName	否	文件名	-

出参描述

```
2024/01/08 16:37:41 GetDocumentSetByName success: &{AIDocumentSet:
{AIDocumentSetInterface:0xc0003ce450 DatabaseName:db-test-ai
CollectionViewName:coll-ai-files DocumentSetId:1193835783695106048
```

DocumentSetName:腾讯云向量数据库.pdf Text:本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。

腾讯云向量数据库是什么?

腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。

关键概念

如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。

什么是向量?

向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。

什么是非结构化数据?

非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。

什么是 AI 中的向量表示?

当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。

什么是向量检索?

向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。

为什么是腾讯云向量数据库?

腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。

腾讯云向量数据库应用示例有哪些?

腾讯云向量数据库可进行高性能向量存储和检索，主要适用于以下应用场景。

- [大规模知识库]：企业的私域数据存储在向量数据库中可构建外部知识库，帮助企业更好地管理和利用自己的数据资源。
- [推荐系统]：向量数据库会基于用户特征进行向量存储与检索，最终筛选用户可能感兴趣的物品推荐给用户。
- [问答系统]：向量数据库会基于问题信息进行向量存储与检索，并返回最相关的问题与对应的答案。
- [文本/图像检索]：向量数据库对输入的图像和文本信息进行向量存储与检索，会找到最匹配输入信息的文本或图像结果。

腾讯云向量数据库支持哪些索引类型?

索引是数据的组织单位。您必须先声明索引类型和相似性度量，然后才能搜索或查询向量数据。目前，腾讯云向量数据库支持如下类型。具体信息，请参见 [Index]。

- FLAT 索引：向量会以浮点型的方式进行存储，不做任何压缩处理。搜索向量会遍历所有向量与目标向量进行比较。

- HNSW 索引：全称为 Hierarchical Navigable Small World， 是基于图的索引，适合对搜索效率要求较高的场景。
- IVF 系列：全称为 Inverted File，IVF 系列索引的核心思想是将高维空间划分为多个聚类，并为每个聚类构建一个倒排文件。适用于高维向量数据的快速检索。

腾讯云向量数据库支持哪些相似度计算方法?

在 VectorDB 中，相似度度量用于衡量向量之间的相似度。选择良好的距离度量有助于显著提高分类和聚类性能。根据输入数据形式，选择特定的相似性度量方法，获得数据库最佳性能。

相似性计算方法 | **方法说明**

:-: | :-:

内积（IP） | 全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 |

欧式距离（L2） | 全称为 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 |

余弦相似度（COSINE） | 余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度。所得值越大越与搜索值相似。 |

腾讯云向量数据库是如何设计的?

- **部署架构**：腾讯云向量数据库采用分布式部署架构，每个节点相互通信和协调，实现数据存储与检索。客户端请求通过 **Load Balancer** 分发到各节点上。。
- **逻辑架构**：实例是腾讯云中独立运行的数据库环境，是用户购买向量数据库服务的基本单位。腾讯云向量数据库数据存储的一个实例集群中包括 [Database]、[Collection]、[Document] 三个逻辑层级。其中，一个实例可以包含很多个 Database，一个 Database 可以包含多个 Collection，一个 Collection 可以包含多个 Document。
- **数据安全**：腾讯云向量数据库的多副本设计、多可用区分布节点、API 密钥认证，并运行于私有网络环境，通过安全组控制访问来源，CAM 账户授权等多方面保护向量数据的完整性和隐私。
- **鉴权方式**：腾讯云向量数据库使用账号（account）和 API 密钥（api_key）的组合进行鉴权，以验证用户身份并授权其访问。
- **连接方式**：腾讯云向量数据库支持通过 HTTP 协议进行数据写入和查询等操作。
- **检索方法**：腾讯云向量数据库支持通过精确检索、相似度检索、混合检索的方法。
- 精确查询：基于标量（指一个单独的数值，例如文本字段、数值字段或日期字段，区别于向量等多维数据结构）字段精确查找数据的方式。
- 相似度检索：基于向量相似度计算的检索方式，通过计算向量之间的相似度来找到与查询向量最相似的文档。
- 混合检索：基于标量字段和向量字段，搭配自定义的标量字段的 Filter 表达式进行检索的方式。

如何快速体验向量数据库?

腾讯云向量数据库目前是公测阶段。免费测试版实例每个账号仅限申领1个，高可用版与单机版实例免费试用时长1个月，到期后可 [提交工单](https://console.cloud.tencent.com/workorder/category) 进行续期；若一个月内未使用实例，平台将自动回收。

序号 | **步骤描述** | **具体操作**

:-: | :-: | :-:
1 | 申请腾讯云账号并认证。 | - 如需注册腾讯云账号：请单击［注册腾讯云账号］(https://cloud.tencent.com/register?s_url=https%3A%2F%2Fcloud.tencent.com%2F)。 | - 如需完成实名认证：请单击［实名认证］(https://console.cloud.tencent.com/developer)。 |
2 | 了解向量数据库所支持的规格与类型。 | 预估数据规模，选择合适的类型与规格。 |
3 | 确定向量数据库所部署的地域。 | 当前支持的地域信息，请参见［发布地域］|
4 | 规划数据库实例的私有网络与安全组。 | 具体操作，请参见［创建私有网络］(https://cloud.tencent.com/document/product/215/36515)与［创建安全组］，并同时设置安全组入站规则。 |
5 | 购买实例。 | 具体操作，请参见［新建数据库实例］。购买实例中，直接选择上一步已准备的私有网络与安全组。 |
6 | 申请与腾讯云向量数据库在同一地域同一个 VPC 内的 Linux 云服务器 CVM。 | 具体操作，请参见［快速配置 Linux 云服务器］(https://cloud.tencent.com/document/product/213/2936)。 |
7 | 连接并操作向量数据库。 | ［连接并写入数据库］，本文使用［API 接口］从创建 DataBase 到 插入数据、检索数据到最终删除数据，均给出了具体的使用示例。您可以简单并快速体验向量数据库。 |
8 | 管理向量数据库实例 | 您可以体验通过控制台直接管理实例，查看实例状态或销毁实例。 |
9 | 智能运维 | 您可以在控制台查看监控数据库实例的各项指标。目前仅支持对节点信息的监控，后续还会支持更丰富的监控项目。 |

开发者工具

开发者工具	**API**
HTTP API	［API 接口］(https://cloud.tencent.com/document/product/1709/98666)
Python SDK	［Python SDK Demo］(https://cloud.tencent.com/document/product/1709/96724)
Java SDK	［Java SDK Demo］(https://cloud.tencent.com/document/product/1709/97768)

TextPrefix:本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。

腾讯云向量数据库是什么？

腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似 DocumentSetInfo:0xc0003b40c0 ScalarFields:map[author:Tencent] SplitterPreprocess:0xc0000125b8 ParsingProcess:0xc000382980} Count:1}

参数名	子参数	参数含义
databaseName	-	数据库名。

collectionViewName	-	集合视图名。
documnetSetId	-	文件 ID。
documnetSetId	-	文件名。
textPrefix	-	文件内容前200个字符。
textPrefix	-	文件完整内容。
documentSetInfo	textLength	文件的字符数。
	byteLength	文件的字节数。
	indexedProgress	文件被预处理、Embedding 向量化的进度。
	indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。
	createTime	文件创建时间。
	lastUpdateTime	文件最后更新时间。
	keywords	文件关键字。
ScalarFields	-	自定义的文件 Metadata 信息字段。
SplitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • false：不追加。 • true：将段落 Title 追加到切分后的段落。
	appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • false：不追加。 • true：将全文的 keywords 追加到切分后的段落。

ParsingProcess	ParsingType	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

获取 Chunks

最近更新时间：2025-05-26 11:14:52

接口定义

`GetChunks()` 接口用于获取文件切分后的语块。

> **说明：**
> Chunk 指语块，较长文本在处理时会切分为多个语块，以便于向量化和更高效地检索，多个 Chunk 组成一个 DocumentSet。

```
GetChunks(ctx context.Context, param
tcvectordb.GetAIDocumentSetChunksParams)
(*tcvectordb.GetAIDocumentSetChunksResult, error)
```

使用示例

> **说明：**
> - 支持指定具体的文件名获取文件切分后的语块。
> - 支持指定具体的 DocumentSet ID 获取文件切分后的语块。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col := client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, err := col.GetChunks(ctx,
tcvectordb.GetAIDocumentSetChunksParams{
        DocumentSetName: "腾讯云向量数据库.pdf",
    })
log.Printf("GetChunks, count: %v", result.Count)
for _, chunk := range result.Chunks {
        log.Printf("chunk: %+v", chunk)
    }
```

入参描述

参数名	是否必选	参数含义
DocumentSetId	否	文件上传在数据库之后，系统分配的文件 ID。 **说明：** DocumentSetId 与 documentSetName 二者必须配置其中之一。
DocumentSetName	否	文件名。
Limit	否	每页返回的 Chunks 数量。 • 数据类型：uint 64。 • 默认值：10。 • 取值范围：[1,1000]。
Offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 Limit 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。

出参描述

```
2024/01/08 16:27:29 GetChunks, count: 10
2024/01/08 16:27:29 chunk: {Text:本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。
 StartPos:0 EndPos:122}
2024/01/08 16:27:29 chunk: {Text:## 腾讯云向量数据库是什么?
腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。
 StartPos:122 EndPos:313}
2024/01/08 16:27:29 chunk: {Text:## 关键概念
```

如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。
StartPos:313 EndPos:441}
2024/01/08 16:27:29 chunk: {Text:### 什么是向量?
向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。
StartPos:441 EndPos:508}
2024/01/08 16:27:29 chunk: {Text:### 什么是非结构化数据?
非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。
StartPos:508 EndPos:585}
2024/01/08 16:27:29 chunk: {Text:### 什么是 AI 中的向量表示?
当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。
StartPos:585 EndPos:784}
2024/01/08 16:27:29 chunk: {Text:### 什么是向量检索?
向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。
StartPos:784 EndPos:876}
2024/01/08 16:27:29 chunk: {Text:## 为什么是腾讯云向量数据库?
腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。
StartPos:876 EndPos:1037}
2024/01/08 16:27:29 chunk: {Text:## 腾讯云向量数据库应用示例有哪些?
腾讯云向量数据库可进行高性能向量存储和检索，主要适用于以下应用场景。
- [大规模知识库]：企业的私域数据存储在向量数据库中可构建外部知识库，帮助企业更好地管理和利用自己的数据资源。
- [推荐系统]：向量数据库会基于用户特征进行向量存储与检索，最终筛选用户可能感兴趣的物品推荐给用户。 StartPos:1037 EndPos:1302}
2024/01/08 16:27:29 chunk: {Text:
- [问答系统]：向量数据库会基于问题信息进行向量存储与检索，并返回最相关的问题与对应的答案。
- [文本/图像检索]：向量数据库对输入的图像和文本信息进行向量存储与检索，会找到最匹配输入信息的文本或图像结果。
StartPos:1302 EndPos:1511}

参数名	子参数	参数含义
count	-	获取的 Chunks 数量。
chunk	text	获取的 Chunks 内容。

	startPos	每个 Chunks 在文件中偏移的起始位置。
	endPos	每个 Chunks 在文件中偏移的结束位置。

版权所有：腾讯云计算（北京）有限责任公司

查询文件信息

最近更新时间：2025-07-01 10:19:43

接口定义

Query() 用于精确查找与查询条件完全匹配的文件，可获取文件长度、向量化的进度与状态等，不包括文件内容。具体支持如下方式查找文件。

- **支持指定具体的文件名查找文件，或搭配文件 Metadata 信息对应字段的 Filter 表达式查询文件信息。**
- **支持指定查询起始位置 offset 和返回数量 limit，查找指定范围的文件信息。**
- **支持根据文件 Meta 信息对应字段 Filter 表达式，过滤需查找的文件。**

```
Query(ctx context.Context, params tcvectordb.QueryAIDocumentSetParams)
(*tcvectordb.QueryAIDocumentSetResult, error)
```

使用示例

使用文件名搭配 Filter 查询文件

根据存储于向量数据库的文件名，搭配标量字段 author 与 tags 的 Filter 表达式一并过滤文件。

```
import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)

var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
// 指定查询参数
param := tcvectordb.QueryAIDocumentSetParams{
  DocumentSetName: []string{"腾讯云向量数据库.pdf"},
```

```
  Filter:           tcvectordb.NewFilter(`author="sam"`),
  Limit:            3,
  Offset:           0,
  //OutputFields: []string{"indexedStatus", "textPrefix"},
}
result, _ := col.Query(ctx, param)
// 输出查询结果
log.Printf("Query success: %+v", result.Count)
for _, doc := range result.Documents {
    b, err := json.Marshal(doc)
    if err != nil {
        return
    }
    fmt.Println(fmt.Sprintf("res %v", string(b)))
}
res {"databaseName":"go-sdk-test-ai-db","collectionViewName":"go-
sdk-test-ai-
coll","documentSetId":"1313843877669244928","documentSetName":"腾讯云
向量数据库.pdf","text":"","textPrefix":"![vdb-image](22ec6930-1629-
4b18-94d9-cd5a3d27faf6.png)\n\n![vdb-image](67cb122b-5322-4931-905a-
7c0ee2e7ce98.png)\n\n## 版权所有：腾讯云计算（北京）有限责任公司\n\n![vdb-
image](50933877-8262-439f-96e6-a7e7aee2d841.png)\n\n\n\n!
[vdb","documentSetInfo":
{"textLength":11945,"byteLength":1925502,"indexedProgress":21,"index
edStatus":"Loading","createTime":"2024-12-04
20:26:30","lastUpdateTime":"2024-12-04 20:26:59"},"ScalarFields":
{"author_name":{"val":"sam"},"fileKey":
{"val":1024}},"splitterPreprocess":
{"appendTitleToChunk":false,"appendKeywordsToChunk":false},"parsingP
rocess":{"parsingType":"VisionModelParsing"}}
```

查询指定范围的文件

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
```

```
param := tcvectordb.QueryAIDocumentSetParams{
  Limit:          3,
  Offset:         0,
  // 使用OutputFields一定会输出documentSetId、documentSetName便于后续操作
  OutputFields: []string{"indexedStatus", "textPrefix"},
}
result, _ := col.Query(ctx, param)
log.Printf("GetDocumentSetByName success: %+v", result.Count)
```

入参描述

参数	是否必选	配置方法及要求
`DocumentSetName`	否	表示要查询的文档的名称，支持批量查询，数组元素范围[1,20]。
`DocumentSetId`	否	表示要查询的文档的所有 ID，支持批量查询，数组元素范围[1,20]。

`Filter`	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中：
		• <field_name>：表示要过滤的字段名。
		• <operator>：表示要使用的运算符。
		  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
		  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。
		  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
		• <value>：表示要匹配的值。
		示例：`Filter('author="jerry"').And('page>20')`。

`Limit`	是	每页返回的 DocumentSet 数量。 • 数据类型：uint 64。 • 默认值：10。 • 取值范围：[1,16384]。

		注意： • 若不配置任何查询条件，即 `doc_list = coll_view.query()`，则默认返回 10 个 DocumentSet。 • 若查询条件仅配置 Filter 表达式，不配置 limit，则默认返回 10 条 DocumentSet。 • 若查询条件仅设置 document_set_name 或 document_set_id，则可不配置 limit 参数，默认返回 10 条数据。
`Offset`	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 limit 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。
`OutputFields`	否	以数组形式配置需返回的字段。

出参描述

```
res {"databaseName":"go-sdk-test-ai-db","collectionViewName":"go-sdk-test-ai-coll","documentSetId":"1313843877669244928","documentSetName":"腾讯云向量数据库.pdf","text":"","textPrefix":"![vdb-image](22ec6930-1629-4b18-94d9-cd5a3d27faf6.png)\n\n![vdb-image](67cb122b-5322-4931-905a-7c0ee2e7ce98.png)\n\n## 版权所有：腾讯云计算（北京）有限责任公司\n\n![vdb-image](50933877-8262-439f-96e6-a7e7aee2d841.png)\n\n\n\n![vdb","documentSetInfo":{"textLength":11945,"byteLength":1925502,"indexedProgress":21,"indexedStatus":"Loading","createTime":"2024-12-04 20:26:30","lastUpdateTime":"2024-12-04 20:26:59"},"ScalarFields":{"author":{"val":"sam"},"fileKey":{"val":1024}},"splitterPreprocess":{"appendTitleToChunk":false,"appendKeywordsToChunk":false},"parsingProcess":{"parsingType":"VisionModelParsing"}}
```

参数名	子参数	参数含义
Count	-	查询到的文件数量。

databaseName	-	数据库名。
collectionViewName	-	集合视图名。
documnetSetId	-	文件 ID。
documnetSetName	-	文件名。
text	-	该参数为空。
textPrefix	-	文件内容前 200个字符。
documentSetInfo	textLength	文件的字符数。
	byteLength	文件的字节数。
	indexedProgress	文件被预处理、Embedding 向量化的进度。
	indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。
	indexedErrorMsg	文件解析、写入错误描述信息。 **说明：** 当 IndexedStatus 为 Failure 时，返回 indexedErrorMsg 信息。
	createTime	文件创建时间。
	lastUpdateTime	文件最后更新时间。
	keywords	文件关键字。
ScalarFields	-	自定义的文件 Metadata 信息字段。
splitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • false：不追加。

		• **true**：将段落 Title 追加到切分后的段落。
	appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 **keywords** 追加到切分后的段落。
parsingProcess	**parsingType**	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

相似性检索文件内容

最近更新时间：2025-07-01 10:19:43

功能介绍

Search() 接口用于在指定的文件内，查找与给定文本信息相似的 Top K 条文本信息。

```
Search(ctx context.Context, param tcvectordb.SearchAIDocumentSetsParams)
(*tcvectordb.SearchAIDocumentSetResult, error)
```

请求示例

说明：

- 支持指定文件名称检索最相似的文本信息。
- 支持文件名搭配文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。
- 支持仅使用文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。

根据文件名搭配 Filter 检索相似数据

如下示例，在文件 腾讯云向量数据库.pdf 中，相似性检索与 `什么是向量数据库` 相似的文本信息，并使用标量字段 author_name 的 Filter 表达式的文本信息。

```
var (
    ctx                = context.Background()
    aiDatabase         = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
searchRes, _ := col.Search(ctx,
tcvectordb.SearchAIDocumentSetsParams{
    Content: "什么是向量数据",
    DocumentSetName: []string{"腾讯云向量数据库.pdf"},
    //Filter:          tcvectordb.NewFilter(`author="sam"`),
    Limit:  3,    // 指定 Top K 的 K 值
    ExpandChunk: []int{1,1},
})
```

```
for _, doc := range searchRes.Documents {
    log.Printf("document: %+v", doc)
}
```

根据 Filter 表达式检索相似数据

如下示例，通过文件 meta 信息的标量字段 author 的 Filter 表达式，检索与 `什么是向量数据库` 相似的文本信息。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
searchRes, _ := col.Search(ctx,
tcvectordb.SearchAIDocumentSetsParams{
    Content: "什么是向量数据库",
    Filter:           tcvectordb.NewFilter(`author="sam"`),
    Limit:  3,    // 指定 Top K 的 K 值
    ExpandChunk: []int{1,1},
})
for _, doc := range searchRes.Documents {
    log.Printf("document: %+v", doc)
}
```

参数名称	是否必选	参数含义及配置方法
`Content`	是	以 String 类型输入检索的文本信息。
`ExpandChunk`	否	• 以数组形式配置检索的目标信息所需向前扩展的段落数量以及向后扩展的段落数。例如，输入[2,3]，指所检索到的 Chunk 返回时，同时返回其之前的 2 个段落与之后的3个段落。 • 段落指文件在上传存储时，自动向量化拆分的段落。

		• 默认值：[1,1]。
DocumentSetName	否	表示要查询的文档的名称，支持批量查询，数组元素范围[1,10]。
Filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`。
Limit	是	指定返回最相似的 Top K 的 K 的值。

出参描述

```
2025/06/27 11:47:48 ---------------------------- Search ----------------
------------
2025/06/27 11:47:49 SearchDocument: {DatabaseName:go-sdk-demo-ai-db
CollectionViewName:go-sdk-demo-ai-col DocumentSetId:1388002727443693568
DocumentSetName:腾讯云向量数据库.pdf Score:0.8344933390617371 SearchData:
{Text:## 什么是向量?

向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向
量在每个坐标轴上的分量。

 StartPos:1418 EndPos:1486 Pre:[## 关键概念

如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一
个初步的了解。更多名词解释，请阅读关键概念。

] Next:[## 什么是非结构化数据?
```

非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。

] ParagraphTitle:什么是向量？ AllParentParagraphTitles:[什么是向量？]}
ScalarFields:map[author:sam fileKey:1024 test_str:v1]}
2025/06/27 11:47:49 SearchDocument: {DatabaseName:go-sdk-demo-ai-db
CollectionViewName:go-sdk-demo-ai-col DocumentSetId:1388002727443693568
DocumentSetName:腾讯云向量数据库.pdf Score:0.7606131434440613 SearchData:
{Text:## 向量（Vector）

向量可以理解为一组数值的有序集合，通常用于表示某个对象或事物的属性或者特征。这些数值可以有不同的维度，每个维度都表示一个属性或特征。在机器学习和人工智能领域，向量常用于表示图像、文本、音频等数据，通过计算向量之间的距离或相似度来实现分类、聚类、检索等任务。

StartPos:7309 EndPos:7454 Pre:[## 向量数据库

关键概念

最近更新时间：2023-07-2419:36:42

] Next:[## OLAMA

OLAMA是腾讯自研的向量引擎，具有高性能、高可用、简单易用等特点。它支持单索引10亿级向量规模，适用于AI运算、检索场景，已稳定服务于近40个线上业务。

] ParagraphTitle:向量（Vector） AllParentParagraphTitles:[向量（Vector）]}
ScalarFields:map[author:sam fileKey:1024 test_str:v1]}
2025/06/27 11:47:49 SearchDocument: {DatabaseName:go-sdk-demo-ai-db
CollectionViewName:go-sdk-demo-ai-col DocumentSetId:1388002727443693568
DocumentSetName:腾讯云向量数据库.pdf Score:0.7552775144577026 SearchData:
{Text:## 什么是AI中的向量表示?

当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。具体上线时间，请关注产品动态。

StartPos:1564 EndPos:1708 Pre:[## 什么是非结构化数据?

非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。

] Next:[## 什么是向量检索?

向量检索是将向量与数据库进行比较以查找与查询向量最相似的向量的过程。相似的向量通常具有相近的原始数据，通过向量检索可以挖掘出原始非结构化数据之间的联系。

] ParagraphTitle:什么是AI中的向量表示? AllParentParagraphTitles:[什么是AI中的向量表示?]} ScalarFields:map[author:sam fileKey:1024 test_str:v1]}

参数名	子参数	参数含义
Score	-	表示查询向量与检索结果向量之间的相似性计算分数。
SearchData	Text	检索的结果。
	EndPos	检索结果在文件中偏移的结束位置。
	StartPos	检索结果在文件中偏移的起始位置。
	Next	根据检索时，设置的参数 chunkExpand，返回检索结果向后扩展的段落。
	Pre	根据检索时，设置的参数 chunkExpand，返回检索结果向前扩展的段落。
	ParagraphTitle	检索到的 chunk 所属的段落标题。
	AllParentParagraphTitles	检索到的 chunk 所属的所有层级的段落标题。
DocumentSet	DocumentSetId	文件 ID。
	DocumentSetName	文件名。
	ScalarFields	自定义的文件 Meta 信息的标量字段。例如：author、bookName、page 等。 **说明：** 显示创建 CollectionView 时设置为 Filter 索引的字段，同时显示上传文件时或使用 update 新增的字段，但新增的字段不会构建索引。

更新文件信息

最近更新时间：2025-07-01 10:19:43

接口定义

`update()` 接口用于对通过主键（DocumentSet ID）与 Filter 表达式过滤检索 DocumentSet，对 DocumentSet 的部分字段进行更新。同时，支持新增字段。

```
Update(ctx context.Context, updateFields map[string]interface{}, param
tcvectordb.UpdateAIDocumentSetParams)
(*tcvectordb.UpdateAIDocumentSetResult, error)
```

使用示例

注意：

- 不能变更系统分配的 DocumentSet ID 字段，不要求事务完整性。
- 不能变更已上传的文件内容。

说明：

- 新增字段，在创建 CollectionView 时没有为这些字段设置索引，那么新增这些字段时，系统不会自动为其创建索引。
- 支持通过主键（DocumentSet ID）或文件名，搭配 Filter 表达式过滤需更新的文件。
- 支持新增字段，支持更改部分字段。

根据文件名过滤需更新的文件

如下示例，更新文件名为 `tcvdb.md` 的数据，修改其字段 **author** 为 **jack**。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, _ := col.Update(ctx, map[string]interface{}{
```

```
    "author": "jack",
}, tcvectordb.UpdateAIDocumentSetParams{
    DocumentSetName: []string{"腾讯云向量数据库.pdf"},
})
log.Printf("update affect count: %d", result.AffectedCount)
```

根据文件 ID 过滤需更新的文件

如下示例，修改指定文件 ID的数据，修改其字段 author 为 jack。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, _ := col.Update(ctx, map[string]interface{}{
    "author": "jack",
}, tcvectordb.UpdateAIDocumentSetParams{
    DocumentSetId: []string{"001"},
})
log.Printf("update affect count: %d", result.AffectedCount)
```

根据 Filter 表达式过滤需更新的文件

如下示例，修改满足 author Filter 表达式的数据，修改其字段 author 为 jack。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, _ := col.Update(ctx, map[string]interface{}{
```

```
    "author": "jack",
}, tcvectordb.UpdateAIDocumentSetParams{
    Filter:          tcvectordb.NewFilter(`author="sam"`),
})
log.Printf("update affect count: %d", result.AffectedCount)
```

入参描述

参数	子参数	参数含义	是否必选	配置方法及要求
UpdateAIDocumentSetParams	DocumentSetName	指定需更新的文件名。	否	支持批量更新，数据元素最大值为20。 **说明：** 同时配置 DocumentSetName 与 Filter 参数，更新数据将会取二者交集。
	DocumentSetId	指定需更新的文件ID。	否	支持批量更新，数据元素最大值为20。 **说明：** 同时配置 DocumentSetId 与 Filter 参数，更新数据将会取二者的交集。

	Filter	配置Filter 表达式过滤需更新的文件	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中：
				• <field_name>：表示要过滤的字段名。
				• <operator>：表示要使用的运算符。
				  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
				  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。

				○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`
`updateFields`	-	设置需更新的字段	否	当前已存在的字段，更新字段对应的数据。 • 类型：string。 • 字符长度要求：[1,256]。
			否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。

删除指定文件

最近更新时间：2025-07-01 10:19:43

接口定义

`delete()` 接口用于删除存储于 CollectionView 文件。

```
Delete(ctx context.Context, param tcvectordb.DeleteAIDocumentSetParams)
(*tcvectordb.DeleteAIDocumentSetResult, error)
```

使用示例

说明：

- 支持批量删除，文件 ID 或文件名数组元素数量最大为20。
- 支持使用 Filter 表达式过滤所需删除的所有文件。

根据文件名过滤需删除的文件

如下示例，删除文件名为 `tcvdb.md` 的文件。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, _ := col.Delete(ctx, tcvectordb.DeleteAIDocumentSetParams{
    DocumentSetNames: []string{"腾讯云向量数据库.pdf"},

})
log.Printf("delete affect count: %d", result.AffectedCount)
```

根据文件 ID 过滤需删除的文件

如下示例，删除指定文件 ID 的文件。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName  = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, _ := col.Delete(ctx, tcvectordb.DeleteAIDocumentSetParams{
    DocumentSetIds: []string{"001"},
})
log.Printf("delete affect count: %d", result.AffectedCount)
```

根据 Filter 表达式过滤需删除的文件

如下示例，删除满足 author 字段 Filter 条件表达式的文件。

```
var (
    ctx                 = context.Background()
    aiDatabase          = "go-sdk-test-ai-db"
    collectionViewName  = "go-sdk-test-ai-coll"
)

col :=
client.AIDatabase(aiDatabase).CollectionView(collectionViewName)
result, _ := col.Delete(ctx, tcvectordb.DeleteAIDocumentSetParams{
    Filter:          tcvectordb.NewFilter(`author="sam"`),
})
log.Printf("delete affect count: %d", result.AffectedCount)
```

入参描述

参数名称	参数含义	是否	配置方法及要求

		必选	
DocumentSetNames	指定需删除的文件名。	否	支持批量删除，数据元素最大值为20。 **说明：** 同时配置 DocumentSetName 与 filter 参数，删除数据将会取二者交集。
DocumentSetIds	指定需删除的文件ID。	否	支持批量删除，数据元素最大值为20。 **说明：** 同时配置 DocumentSetId 与 filter 参数，删除数据将会取二者的交集。
Filter	配置Filter表达式过滤需删除的文件。	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`

管理 Index
重构索引

最近更新时间：2025-07-01 10:19:43

接口定义

`RebuildIndex()` 接口用于重建指定 Collection 的所有索引，清除无用的索引数据，修复损坏的索引数据，优化索引结构，改善性能。

```
RebuildIndex(ctx context.Context, params
...*tcvectordb.RebuildIndexParams) (result
*tcvectordb.RebuildIndexResult, err error)
```

说明：

RebuildIndex 接口用于重建指定集合的向量索引，主要适用于以下两种场景：

- 首次使用 IVF 系列索引时，需要通过 RebuildIndex 接口初始化索引结构。具体信息，请参见 IVF 系列索引应用指南 。
- 在离线批量数据导入场景中，建议先执行数据写入操作（通过 Upsert 接口设置 BuildIndex = false 暂停索引构建），待数据完整导入后，再调用RebuildIndex 接口集中创建索引，以此减少写入过程中的索引维护开销，提升整体导入效率。

警告：

Rebuild 操作需占用额外内存且可能影响服务，执行前请评估业务场景并选择重建方式。

- 保留原索引、同时构建新索引（即设置dropBeforeRebuild=False）：期间集合读取正常，写入被阻止，构建新索引后，替换原索引，构建过程中，旧索引和新索引都会占用内存，如果内存足够可采用该方式，建议根据实际业务高低峰选择停写时段执行。
- 先删除原索引、再构建新索引（即设置dropBeforeRebuild=True）：期间集合读写会受到影响，先删除旧索引，再构建新索引。如果内存严重不足时，则需采用该方案。建议根据业务需求选择可中断服务的时段处理。

使用示例

```
var (
    ctx        = context.Background()
    database   = "go-sdk-test-db"
    collection = "go-sdk-test-emcoll"
```

```
)
_, err = d.client.RebuildIndex(ctx, database, collection,
&tcvectordb.RebuildIndexParams{
    FieldName:          "vector",
    DropBeforeRebuild: true,
    Throttle:           1,
})
if err != nil {
    panic(err)
}
```

入参描述

参数名	参数	是否必选	参数含义	配置方法及要求
databaseName	-	是	配置需重建索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collectionName	-		指定需重建索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
RebuildIndexParams	FieldName	否	指定重建索引的字段。	• vector：重建稠密向量索引。默认值为 vector。 • sparse_vector：重建稀疏向量索引。 **说明：** 同一集合，不支持并发执行 rebuild。如果集合中既有稠密向量又有稀疏向量，则需要分别进行重建。
	DropBeforeRebuild	否	标识在重建索引时，是否需先删除旧索引再重建新索引。 **警告：**	取值如下所示： • true：重建之前，先删除旧索引在重建索引。 **警告：**

			重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在重建索引之前，您需根据实际资源情况选择是否需先删除旧索引再重建，避免引起内存占满而阻塞业务正常运行。	内存资源不足时，可配置为 true，先删除旧索引，在新索引构建过程中，无法正常读写。 • **false**：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为 false。 **警告：** 内存资源足够的情况下，可配置为 false，可不删除旧索引。在新索引构建过程中，可读数据，禁止写入数据。
	Throttle	否	标识是否限制构建索引的单节点 CPU 核数。 **说明：** 重建索引会消耗 CPU 资源，为防止资源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为1。	取值如下所示： • **0**：不限制 CPU 核数。在模型训练期间，会消耗大量的 CPU 资源。重建索引任务将会尽快执行，但可能会对其他集合的读写操作产生影响。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数,可避免构建索引期间对其他集合产生影响，但任务执行较慢。

出参描述

说明：
RebuildIndex() 执行之后，如果抛出异常，说明重建索引失败。具体异常原因，可根据提示信息进行分析。无任何提示信息说明执行成功，可使用 query() 确认删除的 Document 已经不存在。

相关说明

使用 describeCollection() 接口查看 Collection 的索引状态，返回参数 **indexStatus** 中的 **status** 标识当前 Collection 重建索引的状态，**startTime** 显示重建索引开始的时间。

- **initial**：表示首次创建索引，尚未 Rebuild 构建索引。
- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。
- **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。
- **failed**：重建索引失败，可能会影响集合读写操作。

> ⚠ **注意：**
> **training data** 与 **building index** 状态期间不可写入数据。若重建索引之前先删除旧索引，则集合不可进行相似性查询，只能进行精确查询。

新增索引

最近更新时间：2025-04-18 17:58:12

接口定义

AddIndex()接口用于在已有集合上，新增标量字段索引。

```
AddIndex(ctx context.Context, databaseName, collectionName string,
params ...*AddIndexParams) (err error)
```

说明：

- 如果集合开启了动态 schema（即创建集合时配置了 filterIndexConfig 参数），不支持创建标量索引。
- 当前不支持创建 json 类型的标量索引。

使用示例

```
var (
    ctx = context.Background()
    database = "db-test"
    collectionName = "go-sdk-test"
)
buildExistedData := true
addFilterIndexs := []tcvectordb.FilterIndex{
    {FieldName: "bookInfo", FieldType: tcvectordb.String, IndexType:
tcvectordb.FILTER},
    {FieldName: "page", FieldType: tcvectordb.Uint64, IndexType:
tcvectordb.FILTER}}
err := cli.AddIndex(ctx, database, collectionName,
&tcvectordb.AddIndexParams{
    FilterIndexs: addFilterIndexs, BuildExistedData: &buildExistedData})
printErr(err)
```

请求参数

参数	子参数（一）	子参数（二）	是否	参数含义	配置方法及要求

			必选		
databaseName	-	-	是	配置需重建索引的Database名称。	使用 /database/list 获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collectionName	-	-	是	指定需重建索引的Collection名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
AddIndexParams	FilterIndex	FieldName	是	指定新增标量索引的字段。	用户自定义字段名。命名规则为1-128字符，只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。
		FieldType		指定字段数据类型。	指定自定义字段的数据类型。取值如下： • **string**：字符型。 • **uint64**：指无符号整数（unsigned integer）。 • **array**：数组类型，数组元素为string。
		IndexType		指定索引类型。	仅支持设置为 `filter` 。
	BuildExistedData	-	是	指定新增的索引字段是否写入过数据。	**true**：默认为true，新增字段已写入过数据，需扫描历史数据并构建索引。 • **false**：新增索引字段为新增字段，无需扫描历史数据，可以快速创建索引。

删除索引

最近更新时间：2025-04-18 17:58:12

接口定义

DropIndex()接口用于在已有集合上，删除标量字段索引。

```
DropIndex(ctx context.Context, databaseName string, collectionName
string, params tcvectordb.DropIndexParams) (err error)
```

> ⚠ **注意：**
>
> - 仅支持删除标量索引，不支持删除向量、稀疏向量索引。
> - 如果集合开启了动态 schema（即创建集合时配置了 filterIndexConfig 参数），不支持删除标量索引。

使用示例

```
var (
    ctx = context.Background()
    database = "db-test"
    collectionName = "go-sdk-test"
)
err := cli.DropIndex(ctx, database, collectionName,
tcvectordb.DropIndexParams{
        FieldNames: []string{"bookInfo", "page"},
})
printErr(err)
```

求参数

参数	是否必选	参数含义	配置方法及要求
database Name	是	配置需删除索引的 Database 名称。	获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection Name	是	指定需删除索引的 Collection 名称。	获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。

FieldNames	是	用于指定删除的标量索引的字段名。	以数组形式，列出需删除的字段名。

版权所有：腾讯云计算（北京）有限责任公司

修改向量索引

最近更新时间：2025-07-01 10:19:43

接口定义

`ModifyVectorIndex()` 接口用于在不重建整个索引的前提下，调整已创建向量索引的配置参数，包括：数据精度类型、距离度量方式、性能优化参数等。

说明：

- 仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。
- 索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。
- HNSW 索引支持重新指定参数 M 和 efConstruction，IVF 索引支持重新指定 nlist（IVF_PQ 支持重新指定 M 和 nlist）。
- 仅 HNSW 索引，支持修改字段数据类型，例如：从vector（标准单精度浮点数向量）可修改为 float16_vector（半精度浮点数向量）。
- 支持重新指定向量相似度计算方法。

```
ModifyVectorIndex(ctx context.Context, databaseName string,
collectionName string, param tcvectordb.ModifyVectorIndexParam) (err
error)
```

使用示例

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/index"
)
var (
    ctx                 = context.Background()
    database            = "go-sdk-test-db"
    collectionName      = "go-sdk-test-emcoll"
)
setThrottle := int32(1)
param := tcvectordb.ModifyVectorIndexParam{
```

```
        RebuildRules: &index.RebuildRules{
            Throttle: &setThrottle,
        },
    }
    param.VectorIndexes = make([]tcvectordb.ModifyVectorIndex, 0)
    param.VectorIndexes = append(param.VectorIndexes,
    tcvectordb.ModifyVectorIndex{
        FieldName: "vector",
        FieldType:  "bfloat16_vector",
        IndexType:  string(tcvectordb.HNSW),
        MetricType: tcvectordb.COSINE,
        Params: &tcvectordb.HNSWParam{
            M:              8,
            EfConstruction: 200,
    }})
    err := client.ModifyVectorIndex(ctx, database, collectionName, param)
    if err != nil {
        panic(err)
    }
```

入参描述

参数名	子参数一	子参数二	是否必选	配置方法及要求
database	-	-	是	指定 Database 名称，命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
collection	-	-	是	指定需 Collection 的重建索引的 Collection 名称。 • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
	VectorIndexs	FieldName	是	稠密向量数据字段名。固定为 vector。 **说明：**

				仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。
		FieldType	否	仅 HNSW 索引可重新指定向量索引的数据类型。默认值与当前类型一致，具体取值如下所示。 • **vector**：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位），适用于需最高精度的计算任务或对误差敏感的向量检索。 • **float16_vector**：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位 的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 BGE-large 等模型生成的嵌入向量。 • **bfloat16_vector**：bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，指数范围与 float32 一致，适用于向量数值范围较大（更多bit位表达整数）的场景。
		IndexType		指定调整后的目标索引类型。默认与原索引类型一致，如果无需调整索引类型，该参数可不填。取值如下所示。 **说明：** • 索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。 • 切换索引类型时，请同步修改 params 参数，与目标索引类型的参数保持一致。 • FLAT 和 IVF 索引不支持 float16_vector 或 bfloat16_vector，当从 HNSW 索引类型切换到 FLAT 或 IVF 类型时，如果字段类型（fieldType）为 float16_vector 或 bfloat16_vector，则必须同步将字段类型修改为 vector 类型。 • 索引切换时需要注意提前预估内存占用，尤其是在 IVF 系列索引切换成 FLAT/HNSW 时，内存差异可能较大。 • **FLAT**：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。 • **HNSW**：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区

				间数据规模。 • IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。
ModifyVectorIndexParam		MetricType	否	重新指定相似度计算方法。 • L2：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • IP：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 • COSINE：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度，所得值越大越与搜索值相似。
		Params	否	重新指定向量索引参数。 • 索引类型 indexType 为 HNSW ，需配置如下参数。 ○ M：每个节点在检索构图中可以连接多少个邻居节点。 ○ 取值类型：uint64。 ○ 取值范围：[4,64]。一般可设置为16。 ○ efConstruction：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。 ○ 取值类型：uint64。 ○ 取值范围：[8,512]。一般可设置为200。 • 索引类型 indexType 为 IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16需配置如下参数。 ○ nlist：指索引中的聚类中心数量。 取值类型：uint64。取值范围：[1,65536]。 ○ M：指乘积量化中原始数据被拆分的子向量的数量。该参数仅 IVF_PQ 索引类型需配置。更多信息，请参见索引与计算。 ○ 取值要求：原始数据的向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。 ○ 取值范围：[1,向量维度]。
	RebuildRules	DropBeforeRebuild	否	标识在重建索引时，是否需先删除旧索引再重建新索引。取值如下所示：

				警告： 重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在重建索引之前，您需根据实际资源情况选择是否需先删除旧索引再重建，避免引起内存占满而阻塞业务正常运行。 • **true**：重建之前，先删除旧索引再重建索引。 **警告：** 内存资源不足时，可配置为 true，先删除旧索引，在新索引构建过程中，无法正常读写。 • **false**：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为 **false**。 **警告：** 内存资源足够的情况下，可配置为 false，可不删除旧索引。在新索引构建过程中，可读数据，禁止写入数据。
		Throttle	否	标识是否限制构建索引的单节点 CPU 核数。取值如下所示： **说明：** 重建索引会消耗 CPU 资源，为防止资源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为1。 • **0**：不限制 CPU 核数。在模型训练期间，会消耗大量的 CPU 资源。重建索引任务将会尽快执行，但可能会对其他集合的读写操作产生影响。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数,可避免构建索引期间对其他集合产生影响，但任务执行较慢。

说明：
修改向量索引参数之后，自动启动 rebuild 重建索引，可使用 /collection/describe 接口返回的参数 indexStatus 查看集合索引状态。
- **initial**：表示首次创建索引，尚未 Rebuild 构建索引。

- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。
- **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。
- **failed**：重建索引失败，可能会影响集合读写操作。

版权所有：腾讯云计算（北京）有限责任公司

User 管理

新建用户

最近更新时间：2025-01-21 19:52:12

接口定义

CreateUser() 接口用于自定义用户名。

```
CreateUser(ctx context.Context, param CreateUserParams) error
```

使用示例

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
    ctx = context.Background()
)
err := client.CreateUser(ctx, tcvectordb.CreateUserParams{User:
"app_user", Password: "**********"})
if err != nil {
    log.Printf("create user failed, err: %+v", err.Error())
}
```

入参描述

参数名	参数含义	数据类型	是否必填	默认值	参数配置
User	自定义用户名	string	是	-	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。

Password	设置密码	string	是	-	• 密码长度要求：8~128字符。
					• 密码设置支持如下类型的任意组合。
					  ○ 字母：大写 A~Z、小写 a~z。
					  ○ 数字：0~9。
					  ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

为用户授权

最近更新时间：2025-01-21 19:52:12

接口定义

GrantToUser() 接口用于对用户授予指定资源的访问权限。

```
GrantToUser(ctx context.Context, param GrantToUserParams) error
```

使用示例

如下示例，给用户 app_user 授予读取 db-0 下所有集合数据的权限，并具有数据库 db1 下 coll1 集合的读写权限。

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
    ctx            = context.Background()
    database       = "db0"
    database1      = "db1"
    collectionName = "coll1"
)
err := client.GrantToUser(ctx, tcvectordb.GrantToUserParams{
    User: "app_user",
    Privileges: []*user.Privilege{
        {
            Resource: database + ".*",
            Actions:  []string{"read"},
        },
        {
            Resource: database + "." + collectionName,
            Actions:  []string{"readWrite"},
         },
    }})
if err != nil {
    log.Printf("grant to user failed, err: %+v", err.Error())
```

```
}
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
User	需授权的用户名	-	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。 **说明：** root 用户享有最高权限级别，无法被任何账号（包括其自身）删除或修改其权限。
Privileges	对用户授予的权限列表，可以配置多条权限	Resource	string	是	指定用户可操作的资源对象，分为如下三个层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"`。 • DB 层级：对某个 DB 授权，包括该DB下的所有Collection。配置方式：`"resource": "db0.*"`，其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，包含集合下所有文档。配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。 **说明：** resource 只能指向已存在的资源；若指定的资源不存在，则触发错误提示信息。

		Actions	array	是	配置资源对象允许操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

撤销用户授权

最近更新时间：2025-01-21 19:52:12

接口定义

RevokeFromUser() 接口用于撤销用户已拥有的权限。

```
RevokeFromUser(ctx context.Context, param
tcvectordb.RevokeFromUserParams) error
```

说明：

- **root 用户为数据库的默认用户，享有最高权限级别，无法被任何账号（包括其自身）撤销。**
- **撤销权限时，权限需要和授予时完全对应，包括资源对象和允许的操作，都需要保持一致。**
  - **只能针对原本层级回收权限，不能单独回收子对象的权限。例如，用户拥有 *.* 下的权限，那么不能单独回收 db.* 或者 db.coll 的权限。**
  - **只能回收授予时指定的 action。例如，用户拥有某个资源对象的 readWrite 权限，那么回收时不能单独回收其 read 权限。**
- **授权后，如果资源对象被删除，权限不会被撤销。后续如果该资源对象被重新创建，用户依然拥有原本的访问权限。建议用户在删除资源对象后，同时清理对应的访问权限。**

使用示例

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
    ctx = context.Background()
    database        = "db0"
)
err := client.RevokeFromUser(ctx, tcvectordb.RevokeFromUserParams{User:
"app_user",
    Privileges: []*user.Privilege{
        {
            Resource: database + ".*",
```

```
            Actions:   []string{"read"},
        },
    }}
)
if err != nil {
    log.Printf("create user failed, err: %+v", err.Error())
}
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
User	用户名	-	string	是	需撤销操作权限的用户名。 • 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。
Privilege	指定撤销用户的资源对象以及其权限，支持一次撤销多条权限。	resource	string	是	指定用户撤销的资源对象，必须与授权时的指定的层级范围一致。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该DB下的所有Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
		actions	array	是	指定资源对象允许操作的权限，必须与授权时的权限保持一致。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。

					• read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

查询指定用户

最近更新时间：2025-01-21 19:52:12

接口定义

DescribeUser() 接口用于查询指定的用户信息。

```
DescribeUser(ctx context.Context, param tcvectordb.DescribeUserParams)
(result *tcvectordb.DescribeUserResult, err error)
```

使用示例

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
    ctx = context.Background()
)
result, err := client.DescribeUser(ctx,
tcvectordb.DescribeUserParams{User: "app_user"})
if err != nil {
    panic(err)
}
log.Printf("DescribeUser Result: %+v", result)
```

入参描述

参数名	参数含义	子参数	数据类型	是否必填	参数配置
User	需查询的用户名	-	string	是	指定需查询的用户名。 **说明：**

					root 用户享有最高权限级别，无法被任何账号（包括其自身）删除或修改其权限。

出参描述

```
{
  "UserPrivileges": {
    "User": "app_user",
    "CreateTime": "2024-12-31 14:24:46",
    "Privileges": [
      {
        "Resource": "db0.*",
        "Actions": ["read"]
      },
      {
        "resource":"db1.coll1",
        "actions":["readWrite"]
      }
    ]
  }
}
```

参数名	子参数	参数配置
user	-	查询的用户名。
createTime	-	用户创建时间。
privileges	resource	用户可操作的资源对象层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。

		• dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

查看用户列表

最近更新时间：2025-01-21 19:52:12

接口定义

ListUser() 接口用于查询实例下的所有用户。

使用示例

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
    ctx = context.Background()
)
listRes, err := client.ListUser(ctx)
if err != nil {
    log.Printf("ListUser failed, err: %+v", err.Error())
}
log.Printf("ListUser Result: %+v", listRes)
```

输出信息，如下所示：

```
ListUser Result:
{
  "Users": [
    {
      "User": "root",
      "CreateTime": "",
      "Privileges": [
        {
          "Resource": "*.*",
          "Actions": ["fullAccess"]
        }
      ]
    },
    {
```

```
        "User": "app_user",
        "CreateTime": "2024-12-31 15:33:26",
        "Privileges": [
          {
            "Resource": "db0.*",
            "Actions": ["read"]
          },
          {
            "Resource": "db1.coll1",
            "Actions": ["readWrite"]
          }
        ]
      }
    ]
}
```

出参描述

参数名	子参数	参数配置
user	-	查询的用户名。
createTime	-	用户创建时间。 root 用户的创建时间为空。
privileges	resource	用户可操作的资源对象层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB下的所有 Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

删除用户

最近更新时间：2025-01-21 19:52:12

接口定义

DropUser() 接口用于删除自定义用户名。

```
DropUser(ctx context.Context, param tcvectordb.DropUserParams) error
```

使用示例

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
    ctx = context.Background()
)
err := client.DropUser(ctx, tcvectordb.DropUserParams{User: "app_user"})
if err != nil {
    log.Printf("drop user failed, err: %+v", err.Error())
}
```

入参描述

参数名	参数含义	数据类型	是否必填	参数配置
User	需删除的自定义的用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。 **说明：** root 用户无法被任何账号（包括其自身）删除。

重置用户密码

最近更新时间：2025-01-21 19:52:12

接口定义

ChangePassword() 接口用于修改用户名密码。

```
ChangePassword(ctx context.Context, param
tcvectordb.ChangePasswordParams) error
```

使用示例

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
    ctx = context.Background()
)
err := client.ChangePassword(ctx, tcvectordb.ChangePasswordParams{
        User:     "app_user",
        Password: "########",
})
if err != nil {
    log.Printf("change user password err: %+v", err.Error())
}
```

入参描述

参数名	参数含义	数据类型	是否必填	参数配置
User	指定用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母、数字、下划线，并以英文字母开头。

Password	设置新密码	string	是	• 密码长度要求：8~128字符。
				• 密码字符类型要求如下：
				  ○ 字母：大写 A~Z、小写 a~z。
				  ○ 数字：0~9。
				  ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

C++ SDK

最近更新时间：2025-05-13 21:02:02

腾讯云向量数据库（Tencent Cloud VectorDB）的 C++ SDK 是将 HTTP API 封装成易于使用的 C++ 函数或类。开发者可以通过 C++ SDK 更加方便地操作数据库，支持稠密向量所有 API 接口。

SDK 信息

语言	支持版本	开源仓库
C++	C++ 17 或更高版本 **说明：** 兼容 Protobuf 3.11.4或更高版本。	vectordatabase-sdk-cpp

HTTP API
使用前阅读

最近更新时间：2025-07-07 10:52:52

腾讯云向量数据库（Tencent Cloud VectorDB）通过 HTTP 协议进行数据写入和查询等操作。您可以将不同类型的请求消息以 JSON 格式放入 HTTP 请求消息 Body 中，将请求发送到 VectorDB 的 HTTP API 地址即可。VectorDB 将自动解析请求消息 Body 中的 JSON 数据，并将其存储到数据库中。

> **说明：**
> 管理向量数据库，您需优先了解腾讯云向量数据库（Tencent Cloud VectorDB）的 逻辑结构简介 。

准备工作

使用腾讯云向量数据库（Tencent Cloud VectorDB）HTTP API 接口进行数据库操作之前，请您优先做如下准备。

- 已 购买实例 ，且状态为**运行中**。
- 申请与腾讯云向量数据库在同一地域同一个 VPC 内的 Linux 云服务器 CVM 。

  > **说明：**
  > 除了通过腾讯云服务器 CVM 内网访问向量数据库，还可以 开启外网功能 ，在本地通过外网快速体验数据库。本文以腾讯云服务器 CVM 为例。

- 获取向量数据库实例的内网 IP 地址与网络端口。请登录 向量数据库控制台 ，在**实例详情**页面**网络信息**区域直接复制。具体操作，请参见 查看实例信息 。
- 获取向量数据库的 API 访问密钥。请登录 向量数据库控制台 ，在**密钥管理**页面直接复制密钥。具体操作，请参见 密钥管理 。
- 在腾讯云 CVM 安全组中配置**出站规则**，需要把腾讯云向量数据库的 IP 及端口添加到出站规则中。在腾讯云向量数据库安全组中配置**入站规则**，把 CVM 的 IP 及端口添加到入站规则中。 操作详情，可参见 安全组 。

API 列表

腾讯云向量数据库（Tencent Cloud VectorDB）的 HTTP 请求要素包含请求方法（GET、POST）、鉴权方式、数据库服务的 URI 地址以及消息体。其中，鉴权方式通过账号和 API 密钥（HTTP 参数格式为 account=root&api_key=xxx）进行校验。如下列出目前所支持的每个 API 接口，包含的接口含义、请求方式与 URL 拼接地址。

接口层级	接口名	接口含义	请求	URL 拼接地址

			方式	
Database	/database/create	创建 Base类数据库	POST	http://{实例内网 IP 地址}:{实例网络端口}/database/create
	/ai/database/create	创建 AI 类数据库	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/database/create
	/database/drop	删除 Base 类数据库	POST	http://{实例内网 IP 地址}:{实例网络端口}/database/drop
	/ai/database/drop	删除 AI 类数据库所有集合视图数据	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/database/drop
	/database/list	查询所有数据库	GET	http://{实例内网 IP 地址}:{实例网络端口}/database/list
Collection	/collection/create	创建集合	POST	http://{实例内网 IP 地址}:{实例网络端口}/collection/create
	/collection/drop	删除集合	POST	http://{实例内网 IP 地址}:{实例网络端口}/collection/drop
	/collection/list	查询集合	POST	http://{实例内网 IP 地址}:{实例网络端口}/collection/list
	/collection/describe	查询指定集合	POST	http://{实例内网 IP 地址}:{实例网络端口}/collection/describe
	/collection/truncate	清空集合别名	POST	http://{实例内网 IP 地址}:{实例网络端口}/collection/truncate
CollectionView	/ai/collectionView/create	创建集合视图	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/collectionView/create
	/ai/collectionView/drop	删除集合视图	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/collectionView/drop
	/ai/collectionView/list	展示指定 AI Database 下的所有集合视图列表	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/collectionView/list
	/ai/collectionView/describe	返回指定集合的详细信息	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/collectionView/describe

版权所有：腾讯云计算（北京）有限责任公司

	/ai/collectionView/truncate	清空集合视图数据		http://{实例内网 IP 地址}:{实例网络端口}/ai/collectionView/truncate
Alias	/alias/set	给 Collection 创建别名	POST	http://{实例内网 IP 地址}:{实例网络端口}/alias/set
	/ai/alias/set	给 CollectionView 创建别名	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/alias/set
	/alias/delete	删除 Collection 别名	POST	http://{实例内网 IP 地址}:{实例网络端口}/alias/delete
	/ai/alias/delete	删除 CollectionView 别名	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/alias/delete
Document	/document/upsert	插入数据	POST	http://{实例内网 IP 地址}:{实例网络口}/document/upsert
	/document/query	精确查找数据	POST	http://{实例内网IP地址}:{实例网络端口}/document/query
	/document/search	检索相似向量	POST	http://{实例内网 IP 地址}:{实例网络口}/document/search
	/document/delete	删除数据	POST	http://{实例内网 IP 地址}:{实例网络口}/document/delete
	/document/update	更新数据	POST	http://{实例内网 IP 地址}:{实例网络口}/ai/documentSet/update
DocumentSet	/ai/documentSet/uploadUrl	获取文件上传路径与授权签名	POST	http://{实例内网 IP 地址}:{实例网络口}/ai/documentSet/uploadUrl
	/ai/documentSet/get	获取文件信息	POST	http://{实例内网 IP 地址}:{实例网络口}/ai/documentSet/get
	/ai/documentSet/query	精确查询	POST	http://{实例内网 IP 地址}:{实例网络口}/ai/documentSet/query
	/ai/documentSet/search	相似度检索	POST	http://{实例内网 IP 地址}:{实例网络口}/ai/documentSet/search
	/ai/documentSet/delete	删除文档	POST	http://{实例内网 IP 地址}:{实例网络口}/ai/documentSet/delete

版权所有：腾讯云计算（北京）有限责任公司

	/ai/documentSet/update	更新文档	POST	http://{实例内网 IP 地址}:{实例网络端口}/ai/documentSet/update
Index	/index/rebuild	重建索引	POST	http://{实例内网 IP 地址}:{实例网络端口}/index/rebuild

Database create

最近更新时间：2024-09-27 18:00:11

Method 与 URL

- 创建 Base 类向量数据库：`POST https://{实例访问 IP 地址}:{实例网络端口}/database/create`
- 创建 AI 类向量数据库：`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/database/create`

使用示例

> ⚠ **注意：**
> 如下示例 api_key 的值 A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X，需要分别替换为具体实例的 API Key 以及访问地址。

创建 Base 类 Database

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/database/create \
  -d '{
    "database": "db-test"
}'
```

创建 AI 类 Database

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/ai/database/create \
  -d '{
```

```
    "database": "db-test-ai"
}'
```

请求参数

参数	是否必选	参数含义	配置方法及要求
database	是	设置 Database 名称。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

响应消息

执行成功，输出如下信息。

```
{
    "code":0,
    "msg":"Operation success",
    "affectedCount":1
}
```

参数名	参数含义
affectedCount	影响行数，即为创建数据库的数量。

drop

最近更新时间：2024-09-27 18:00:11

Method 与 URL

- 删除 Base 类 Database：`POST https://{实例访问 IP 地址}:{实例网络端口}/database/drop`
- 删除 AI 类 Database：`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/database/drop`

> **警告：**
> 执行 drop 操作将会彻底删除指定数据库下所有数据。在操作之前，请务必谨慎考虑。

使用示例

删除 Base 类数据库

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/database/drop \
  -d '{
    "database": "db-test"
}'
```

删除 AI 类数据库

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/database/drop \
  -d '{
    "database": "db-test-ai"
```

```
}'
```

请求参数

参数名	是否必选	参数含义	参数取值
database	是	待删除的 Database 库名。	Database 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

响应消息

执行成功，输出如下信息。

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1
}
```

参数名	参数含义
affectedCount	影响行数，即为删除数据库的数量。

list

最近更新时间：2024-09-27 18:00:11

Method 与 URL

查看数据库： `POST https://{实例访问 IP 地址}:{实例网络端口}/database/list`

使用示例

```
curl -i -X GET \
   -H 'Content-Type: application/json' \
   -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
   http://10.0.X.X:80/database/list
```

出参描述

返回参数 databases 中以数组的形式，列出了所有的数据库库名。如下所示：

```
{
  "code": 0,
  "msg": "operation success",
  "databases": [
    "db-test",
    "db-test-ai",
    "go-sdk-test-ai-db"
  ],
  "info": {
    "go-sdk-test-ai-db": {
      "createTime": "2023-11-27 17:09:07",
      "dbType": "AI_DB"
    },
    "db-test": {
      "createTime": "2023-11-28 17:15:15",
      "dbType": "BASE_DB"
    },
    "db-test-ai": {
      "createTime": "2023-11-27 17:08:57",
      "dbType": "AI_DB"
    }
  }
```

```
}
```

参数名	参数含义	子参数	子参数含义
databases	以数组的形式，列出集群所有的数据库库名。	-	-
info	逐一列出每一个数据库的信息：包括创建时间与数据库类型。	createTime	数据库创建时间。
		dbType	数据库类型。 • AI_DB：AI 类存储文件类数据库。 • BASE_DB：Base 类存储向量数据或原始文本的数据库。

Collection create

最近更新时间：2025-07-09 15:17:22

Method 与 URL

在已创建的 Base 类 Database 中创建 Collection：
`POST https://{实例访问 IP 地址}:{实例网络端口}/collection/create` 。

> **说明：**
> - 若使用 Embedding 功能，需要在创建 Collection 时，配置文本向量化相关参数。
> - 当前版本一个数据库实例下，不支持创建同名的 Collection。

使用示例

> **注意：**
> 如下示例 api_key 的值 A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X，需要分别替换为具体实例的 API Key 以及访问地址。

仅向量集合

创建一个名为 **book-vector** 的集合，不配置 Embedding 模型相关参数，用于写入3维向量数据。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "replicaNum": 1,
    "shardNum": 1,
    "description": "this is the collection description",
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
```

```
            "indexType": "primaryKey"
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "dimension": 3,
            "metricType": "IP",
            "params": {
                "M": 16,
                "efConstruction": 200
            }
        },
        {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "expired_at",
            "fieldType": "uint64",
            "indexType": "filter"
        }
    ],
    "filterIndexConfig": {
        "filterAll": true,
        "fieldsWithoutIndex": ["page"],
        "maxStrLen": 32
    },
    "ttlConfig": {
        "enable": true,
        "timeField": "expired_at"
    }
}'
```

Embedding 模型（文本读写）

创建一个名为 book-emb 的集合，配置 Embedding 模型相关参数，用于写入原始文本。Embedding 模型自动将原始文本进行向量化。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
  -d '{
    "database": "db-test",
    "collection": "book-emb",
    "replicaNum": 1,
    "shardNum": 1,
    "description": "this is the collection description",
    "embedding": {
        "field": "text",
        "vectorField": "vector",
        "model": "bge-base-zh"
    },
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "metricType": "COSINE",
            "params": {
                "M": 16,
                "efConstruction": 200
            }
        },
        {
          "fieldName": "sparse_vector",
          "fieldType": "sparseVector",
          "indexType": "inverted",
          "metricType": "IP"
        },
        {
```

```
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
        }
    ]
}'
```

仅稀疏向量集合

> **说明：**
> 创建仅写入稀疏向量的集合，必须配置的字段为 id 与稀疏向量 sparse_vector。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "replicaNum": 1,
    "shardNum": 1,
    "description": "this is the collection description",
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
          "fieldName": "sparse_vector",
          "fieldType": "sparseVector",
          "indexType": "inverted",
          "metricType": "IP"
```

```
        },
        {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        }
    ]
}'
```

向量集合+稀疏向量

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "replicaNum": 1,
    "shardNum": 1,
    "description": "this is the collection description",
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "dimension": 3,
            "metricType": "IP",
            "params": {
                "M": 16,
                "efConstruction": 200
            }
        },
        {
```

```
            "fieldName": "sparse_vector",
            "fieldType": "sparseVector",
            "indexType": "inverted",
            "metricType": "IP"
        },
        {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "expired_at",
            "fieldType": "uint64",
            "indexType": "filter"
        }
    ],
    "filterIndexConfig": {
        "filterAll": true,
        "fieldsWithoutIndex": ["page"],
        "maxStrLen": 32
    },
    "ttlConfig": {
        "enable": true,
        "timeField": "expired_at"
    }
}'
```

二进制集合

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
  -d '{
```

```
    "database": "db-test",
    "collection": "bin-vector",
    "replicaNum": 1,
    "shardNum": 2,
    "description": "this is the collection description",
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
            "fieldName": "vector",
            "fieldType": "binary_vector",
            "indexType": "BIN_FLAT",
            "dimension": 16,
            "metricType": "Hamming"
        }
    ]
}'
```

写入文件集合

创建一个数据库集合来存储文件内容，同时为文件的关键字段（如文件名、文本内容、图片信息等）建立索引以优化查询效率。关键字段，如下表所示。

> **说明：**
> 创建数据库集合，文件的关键字段均可以**自定义**，在文件上传时，可通过 field_mappings 保留的文件参数逐一映射自定义的字段名。具体信息，请参见 uploadUrl 。

字段名	字段含义	索引设置要求或建议
file_name	标记文件名。	该字段**必须定义**为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。
text	存放文件知识点的原始文本内容。	• 由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。 • 在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。

section_num	标记段落（section）在原始文档中的顺序（如1, 2, 3, …）。	该字段可创建 filter 索引，以支持通过 section_num 的条件表达式，查询文件特定段落编号的内容。召回的内容也可以根据该值获取对应召回 chunk 所在段落的全部文本。
chunk_num	标记文本片段（chunk）在文件中的顺序（如1, 2, 3, …）。	该字段可创建 filter 索引，以支持通过 chunk_num 的条件表达式，按语块顺序范围检索（如查询第5-10块的内容）文件。本文示例为该字段创建索引。
image_list	存放 PDF 文件中图片的 Key 列表。 **说明：** 在原始检索出的文本块中，图片采用{key}占位符进行标记。系统会同步生成包含所有图片实际 URL 地址的索引列表，每个{key}对应唯一的图片链接。通过将文本块中的占位符替换为对应的<img>标签及真实 URL，即可完整还原原始文档的图文排版结构，最终生成符合网页标准的 HTML 文件。	**不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
  -d '{
    "database": "db-test",
    "collection": "coll-ai-files",
    "replicaNum": 0,
    "shardNum": 1,
    "description": "this is the collection description",
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
```

```
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "dimension": 768,
            "metricType": "IP",
            "params": {
                "M": 16,
                "efConstruction": 200
            }
        },
        {
            "fieldName": "file_name",
            "fieldType": "string",
            "indexType": "filter"
        }
    ]
}'
```

请求参数

参数	参数含义	子参数	是否必选	参数配置
database	指定 Collection 所在的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需创建集合的数据库名。
collection	指定 Collection 的名称。	-	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

replicaNum	指定 Collection 的副本数。副本数是指每个主分片有多个相同的备份，用来容灾和负载均衡。	-	是	• 取值类型：uint64。
				• 取值范围如下所示。搜索请求量越高的索引，建议设置越多的副本数，避免负载不均衡。
				  ○ 单可用区实例：0。
				  ○ 两可用区实例：[1,节点数-1]。
				  ○ 三可用区实例：[2,节点数-1]。

shardNum	指定 Collection 的分片数。分片是把大数据集切成多个子数据集。	-	是	• 取值类型：uint64。
				• 取值范围：[1,100]。例如：5。
				• 配置建议：在搜索时，全部分片是并发执行的，分片数量越多，平均耗时越低，但是过多的分片会带来额外开销而影响性能。
				  ○ 单分片数据量建议控制在300万以内，例如500万向量，可设置2个分片。
				  ○ 如果数据量小于300万，建议使用1分片。系统对1分片有特定优化，可显著提升性能。

embedding	配置 Embedding 参数 **说明：** 使用 Embedding 功能才需要配置该参数。	field	否	指定文本字段名称。 • 取值类型为：string 字符型。当前仅支持文本到向量的 Embedding 能力。 • 写入（/document/upsert/）、更新（/document/update）或者检索（/document/search）数据时，Embedding 模型会自动将该字段的文本内容转换成向量数据。
		vectorField	否	指定向量字段。通过 Embedding 模型生成的向量会自动存储在该字段中，固定为 `vector`。
		model	否	指定使用的 Embedding 模型的名称。您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，参见 Embedding 介绍。取值如下所示： • **bge-large-zh-v1.5**：适用中文，1024维，推荐使用。 • **bge-base-zh-v1.5**：适用中文，768维。 • **bge-large-zh**：适用中文，1024维。 • **bge-base-zh**：适用中文，768维。 • **m3e-base**：适用中文，768维。

				• **e5-large-v2**：适用英文，1024维。 • **text2vec-large-chinese**：适用中文，1024维。 • **multilingual-e5-base**：适用于多种语言类型，768维。 • **BAAI/bge-m3**：适用于多种语言类型，1024维。
description	指定 Collection 的描述信息	–	否	• 取值类型：string。 • 字符长度要求：[1,256]。 • 示例：This is the collection description。
indexes **说明**：创建Collection需根据不同Index类	主键索引 **说明**：每一个Collection 必须指定主键索引和向量索引。具体信息，请参见 Index 。	**fieldName**	是	指定索引对象为文档 id 。
		fieldType		指定索引对象的数据类型。该参数固定为 `string` 。
		indexType		指定索引对象的索引类型。该参数固定配置为 `primaryKey` ，即默认以 id 为主键构建索引。
		autoId	否	自动生成 ID。指定 "autoId": "uuid"，即可开启自动生成 ID 功能。 **说明**： • 当启用 autoId 功能后，用户在写入或更新数据时，无需手动传入 id 字段。若用户在启用 autoId 的情况下仍传入了 id 字段，则系统将优先采用用户传入的 id 值作为数据的唯一标识。 • 在执行批量写入（batch upsert）时，如果部分数据包含 id 字段，而另一部分数据未包含 id 字段，则系统仅对未传入 id 的数据生成默认 id。
	向量索引 **说明**：	**fieldName**	是	指定索引对象为 vector。该参数固定配置为 `vector` 。

型指定索引字段。	每一个 Collection 必须指定主键索引和向量索引。具体信息，请参见 Index 。	fieldType	是	向量字段的数据类型。支持不同类型、不同精度的向量数据，以适用不同业务场景。更多信息，请参见 数据类型 。 • **vector**：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位），适用于需最高精度的计算任务或对误差敏感的向量检索。 • **float16_vector**：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位 的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 bge-large 等模型生成的嵌入向量。 • **bfloat16_vector**：bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，适用于向量数值范围较大（更多bit位表达整数）的场景。 **说明：** 当前仅索引类型为 HNSW 时，字段类型支持**float16_vector**与**bfloat16_vector**。 • **binary_vector**：每个维度仅用1个比特（bit）表示0或1，无指数位、尾数位和符号位，直接存储二进制数据，适用于存储稀疏向量。
		indexType	是	指定索引类型，取值如下所示。更多信息，请参见 索引类型 。 • **FLAT**：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。 • **HNSW**：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区间数据规模。 • **BIN_FLAT**：二进制索引，暴力检索，召回率100%。 • **IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16**：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。 **注意：**

			如果选择 IVF 系列索引类型，那么，请务必在 /document/upsert 插入数据时，将参数 **buildIndex** 设置为 **false**，在插入数据之后，通过 /Index/rebuild 重建索引。具体操作，请参见 应用 IVF 系列索引 。
		dimension	否	指定向量维度。 • 取值类型：uint64。 • 取值范围：[1,4096]。 **说明：** • 若 **indexType** 为 **BIN_FLAT**，向量维度大于等于 **8**，且必须为**8**的倍数。 • 使用 Embedding 功能，则无需配置该字段。该参数将自动配置为 Embedding 模型对应的向量维度。
		metricType	是	指定向量之间距离度量的算法。取值如下： • **L2**：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • **IP**：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 • **COSINE**：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度，所得值越大越与搜索值相似。

				• **Hamming**：汉明距离（Hamming Distance），计算两个二进制字符串对应位置上不同字符的数量，如果字符不同，两字符串的汉明距离就会加一。汉明距离越小，表示两个字符串之间的相似度越高。

		params	是	指定索引类型参数。
				• 索引类型 **indexType** 为 HNSW，需配置如下参数。
				  ○ **M**：每个节点在检索构图中可以连接多少个邻居节点。
				  ○ 取值类型：uint64。
				  ○ 取值范围：[4,64]。一般可设置为16。
				  ○ **efConstruction**：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。
				  ○ 取值类型：uint64。
				  ○ 取值范围：[8,512]。一般可设置为200。
				• 索引类型 **indexType** 为 IVF_FLAT、IVF_PQ、IVF_SQ4、IVF_SQ8、IVF_SQ16，需配置如下参数。
				  ○ **nlist**：指索引中的聚类中心数量。
				  取值类型：uint64。取值范围：[1,65536]。
				  ○ **M**：指乘积量化中原始数据被拆分的子向量的数量。该参数仅 IVF_PQ 索引类型需配置。更多信息，请参见 索引与计算 。
				  ○ 取值要求：原始数据的向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。
				  ○ 取值范围：[1,向量维度]。

	Filter 索引 **说明：** 创建写入文件的 Collectio	**fieldName**	是	配置可作为 Filter 索引的自定义扩展的标量字段名。 **说明：** • Filter 索引（Filter Index）是建立在标量字段的索引。该标量字段名

			称、类型均由用户自定义，不限制标量字段数量。 • 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询以此来匹配相似向量。
n，务必为存放文件名（例如：file_name）的字段创建 Filter 索引，否则，在获取 COS 授权签名时提示错误信息。	**fieldType**	是	指定自定义字段的数据类型。取值如下： • **string**：字符型。 • **uint64**：指无符号整数（unsigned integer）。 • **array**：数组类型，数组元素为 string。 • **json**：由键值对组成的数据对象。具体规则，请参见 json 类型。为 json 类型的字段创建 Filter 索引，如下所示。 **注意：** 开启 动态标量字段（动态 Schema）时，即使用 filterIndexConfig 开启所有标量字段全索引模式，已支持配置 json 类型的 Filter 索引。 `{ "fieldName": "a", "fieldType": "json", "indexType": "filter" }`
	indexType	是	指定自定义扩展的字段是否需要设置 Filter 索引。若需要设置，请将该参数设置为 `filter`，那么在通过 /document/search 检索相似数据时，可对该字段设置 Filter 条件表达式进行混合检索。 **注意：**

				建立 Filter 索引时，选取需要使用 Filter 表达式高效过滤数据的标量字段。不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。
	稀疏向量索引	fieldName	否	可选择构建稀疏向量索引，字段名固定为 sparse_vector。每个集合仅支持创建1个稀疏向量索引，无需指定稀疏向量维度。
		fieldType		稀疏向量字段类型，固定为 sparseVector。
		indexType		稀疏向量的索引类型，当前定义为 inverted。
		metricType		稀疏向量相似性计算仅支持设置为 IP。
filterIndexConfig	支持开启所有标量字段全索引模式，默认关闭	filterAll	否	控制全索引模式的开启和关闭。 • true：开启。 • false：关闭。默认值为 false。
		fieldsWithoutIndex		当 filterAll 为 true 时，支持通过 fieldsWithoutIndex 参数指定不创建索引的字段。默认值：NULL。
		maxStrLen		当 filterAll 为 true 时，支持通过 maxFieldLength 参数设定单条文档中创建索引的标量字段的最大字符数。如果字段字符数超出此限制，将按照设定的 maxStrLen 值进行截断创建索引。 • 默认值：32。 • 取值范围：[1,65536]。
ttlConfig	配置 TTL 属性	enabled	否	标识数据库是否开启 TTL 配置。 • true：开启。 • false：关闭。

				说明： TTL（生存周期）功能规定了数据自创建或更新后将被自动删除的时间期限。向量数据库默认每小时轮询检查一次数据是否过期，超期即清理，且插入数据时允许存在最多1小时的过期时间误差。具体信息，请参见 TTL 。
		timeField		指定存储数据过期时间戳的字段名。插入数据时，以标准的 Unix 时间戳指定该字段的值。字段名要求如下： • 数据类型：uint64。 • 索引类型：Filter 索引。 • 示例：expired_at，则在 upsert 时，需指定 expired_at 值。

响应消息

执行成功，输出如下信息。

```
{
    "code":0,
    "msg":"Operation success",
    "affectedCount":1
}
```

参数名	参数含义
affectedCount	影响行数，即为创建集合数量。

drop

最近更新时间：2025-02-12 17:58:52

Method 与 URL

删除 Base 类数据库中已存在的 Collection：

`POST https://{实例访问 IP 地址}:{实例网络端口}/collection/drop` 。

使用示例

> **警告：**
> 执行 drop 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

> **说明：**
> 如果您为集合设置了别名，删除集合操作仅支持使用集合的原始名称，无法通过别名来删除对应的集合。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/drop \
 -d '{
    "database": "db-test",
    "collection": "book-vector"
}'
```

请求参数

参数名	是否必选	参数含义	配置方法
database	是	指定 Collection 所在的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需删除集合所属的数据库名。
collection	是	所需删除的 Collection 名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需删除的集合。

响应消息

执行成功，输出如下信息。

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1
}
```

参数名	参数含义
affectedCount	影响行数，即为删除集合数量。

list

最近更新时间：2025-02-12 17:58:52

Method 与 URL

查询指定 Base 类 Database 中所有的 Collection：

`POST https://{实例访问 IP 地址}:{实例网络端口}/collection/list` 。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
http://10.0.X.X:80/collection/list \
 -d '{
    "database": "db-test"
}'
```

请求参数

参数	是否必选	参数含义	配置方法
database	是	设置 Collection 所属的 Database 库名。	使用 /database/list 获取集群中的数据库列表，复制需删除集合所属的数据库名。

响应信息

/collection/list 接口执行成功，将返回数据库中所有的 Collection。

```
{
  "code": 0,
  "msg": "Operation success",
  "collections": [
    {
      "database": "db-test",
      "collection": "book-emb",
      "documentCount": 0,
      "alias": [
```

```
            "alias-book-emb"
        ],
    "replicaNum": 0,
    "shardNum": 1,
    "createTime": "2024-09-05 11:14:43",
    "embedding": {
      "field": "text",
      "vectorField": "vector",
      "model": "bge-base-zh",
      "status": "enabled"
    },
    "description": "this is the collection description",
    "indexes": [
      {
        "fieldName": "vector",
        "fieldType": "vector",
        "indexType": "HNSW",
        "indexedCount": 0,
        "dimension": 768,
        "metricType": "COSINE",
        "params": {
          "M": 16,
          "efConstruction": 200
        }
      },
      {
        "fieldName": "bookName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "sparse_vector",
        "fieldType": "sparseVector",
        "indexType": "inverted",
        "metricType": "IP"
      },
      {
        "fieldName": "id",
        "fieldType": "string",
        "indexType": "primaryKey"
      },
      {
        "fieldName": "author",
```

```
            "fieldType": "string",
            "indexType": "filter"
          }
        ],
        "indexStatus": {
          "status": "ready",
          "startTime": ""
        }
      },
      {
        "database": "db-test",
        "collection": "book-vector",
        "documentCount": 0,
        "replicaNum": 0,
        "shardNum": 1,
        "createTime": "2024-09-05 15:24:07",
        "description": "this is the collection description",
        "indexes": [
          {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
          },
          {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
          },
          {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "indexedCount": 0,
            "dimension": 3,
            "metricType": "COSINE",
            "params": {
              "M": 16,
              "efConstruction": 200
            }
          },
          {
            "fieldName": "sparse_vector",
            "fieldType": "sparseVector",
```

```
            "indexType": "inverted",
            "metricType": "IP"
          },
          {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
          }
        ],
        "indexStatus": {
          "status": "ready",
          "startTime": ""
        }
      }
    ]
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 Collection 所在的 Database 名称。
collection	-	-	显示 Collection 的名称。
documentCount			Collection 中的 Document 数量。
alias	-	-	Collection 的所有别名。
replicaNum	-	-	显示 Collection 的副本数。
shardNum	-	-	显示 Collection 的分片数。
createTime	-	-	显示 Collection 的创建时间。
embedding	Embedding 功能相关参数。	textField	显示 Embedding 模型输入文本的字段名。
		vectorField	显示文本被向量化之后存储向量数据的字段名。
		model	Embedding 模型的名称。
		status	说明该 Collection 是否配置 Embedding 模型。

			• **enabled**：已配置。 • **disabled**：未配置。
description	–	–	显示 Collection 的描述信息。
indexes	主键索引	**fieldName**	标识索引对象为 `id` 。
		filedType	显示该索引对象的数据类型，固定为 string。
		indexType	该参数固定显示为 `primaryKey` 。即该索引对象以 id 为主键构建索引。
	向量索引	**fieldName**	标识索引对象为 **vector**。
		filedType	指索引对象为 vector 的数据类型。即该参数固定为 `vector` 。
		indexType	指定索引对象为 vector 的索引类型。具体信息，请参见**索引类型** 。
		indexedCount	索引对象为 vector 中包含的文档数。
		dimension	显示向量维度。
		metricType	显示向量之间的距离度量的算法。
		params	显示索引类型对应的参数。
	Filter 索引	**fieldName**	自定义扩展字段，例如：author、bookName。
		filedType	显示自定义字段的数据类型。
		indexType	显示自定义字段索引类别为 `filter` 。

	稀疏向量索引	fieldName	稀疏向量的字段名。
		fieldType	稀疏向量字段类型，固定为 sparseVector。
		indexType	稀疏向量的索引类型，当前定义为 inverted。
		metricType	稀疏向量之间距离度量的算法。当前仅支持设置为 IP，欧几里得距离，计算向量之间的直线距离，所得的值越小，越与搜索值相似。
indexStatus	标识 Collection 是否有重建索引	status	标识当前 Collection 是否在重建索引。 • initial：表示首次创建索引，尚未 Rebuild 构建索引。 • ready：表示当前 Collection 已准备就绪，可正常使用。 • training：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • building：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • failed：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

describe

最近更新时间：2025-02-12 17:58:52

Method 与 URL

查询指定 Collection 的信息：

`POST https://{实例访问 IP 地址}:{实例网络端口}/collection/describe` 。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/describe \
 -d '{
  "database": "db-test",
  "collection": "book-emb"
 }'
```

请求参数

参数名	是否必选	参数含义	配置方法
database	是	指定 Collection 所在的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需查询的集合所属的数据库名。
collection	是	指定所需查询的 Collection 名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需查询的集合。

响应消息

/collection/describe 接口执行成功，将返回查询 Collection 的信息。

```
{
    "code": 0,
    "msg": "operation success",
    "collection": {
        "database": "db-test",
```

```
        "collection": "book-emb",
        "documentCount": 4,
        "alias": [
            "alias-book-emb"
        ],
        "replicaNum": 2,
        "shardNum": 1,
        "createTime": "2023-09-14 14:48:17",
        "embedding": {
            "field": "text",
            "vectorField": "vector",
            "model": "bge-base-zh",
            "status": "enabled"
        },
        "description": "this is the collection description",
        "indexes": [
            {
                "fieldName": "id",
                "fieldType": "string",
                "indexType": "primaryKey"
            },
            {
                "fieldName": "author",
                "fieldType": "string",
                "indexType": "filter"
            },
            {
                "fieldName": "vector",
                "fieldType": "vector",
                "indexType": "HNSW",
                "indexedCount": 4,
                "dimension": 768,
                "metricType": "COSINE",
                "params": {
                    "M": 16,
                    "efConstruction": 200
                }
            },
            {
              "fieldName": "sparse_vector",
              "fieldType": "sparseVector",
              "indexType": "inverted",
              "metricType": "IP"
```

```
            },
            {
                "fieldName": "bookName",
                "fieldType": "string",
                "indexType": "filter"
            }
        ],
        "indexStatus": {
            "status": "ready",
            "startTime": ""
        }
    }
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 Collection 所在的 Database 名称。
collection	-	-	显示 Collection 的名称。
documentCount			Collection 中的 Document 数量。
alias	-	-	Collection 的所有别名。
replicaNum	-	-	显示 Collection 的副本数。
shardNum	-	-	显示 Collection 的分片数。
createTime	-	-	显示 Collection 的创建时间。
embedding	Embedding 功能相关参数。	textField	显示 Embedding 模型输入文本的字段名。
		vectorField	显示文本被向量化之后存储向量数据的字段名。
		model	Embedding 模型的名称。
		status	说明该 Collection 是否配置 Embedding 模型。 • **enabled**：已配置。 • **disabled**：未配置。

description	–	–	显示 Collection 的描述信息。
indexes	主键索引	fieldName	标识索引对象为 `id` 。
		filedType	显示该索引对象的数据类型，固定为 string。
		indexType	该参数固定显示为 `primaryKey` 。即该索引对象以 id 为主键构建索引。
	向量索引	fieldName	标识索引对象为 **vector**。
		filedType	指索引对象为 vector 的数据类型。即该参数固定为 `vector` 。
		indexType	指定索引对象为 vector 的索引类型。具体信息，请参见**索引类型** 。
		indexedCount	索引对象为 vector 中包含的文档数。
		dimension	显示向量维度。
		metricType	显示向量之间的距离度量的算法。
		params	显示索引类型对应的参数。
	Filter 索引	fieldName	自定义扩展字段，例如：author、bookName。
		filedType	显示自定义字段的数据类型。
		indexType	显示自定义字段索引类别为 `filter` 。
	稀疏向量索引	fieldName	稀疏向量的字段名。

		fieldType	稀疏向量字段类型，固定为 sparseVector。
		indexType	稀疏向量的索引类型，当前定义为 inverted。
		metricType	稀疏向量之间距离度量的算法。当前仅支持设置为 IP，欧几里得距离，计算向量之间的直线距离，所得的值越小，越与搜索值相似。
indexStatus	标识 Collection 是否有重建索引	status	标识当前 Collection 是否在重建索引。 • initial：表示首次创建索引，尚未 Rebuild 构建索引。 • ready：表示当前 Collection 已准备就绪，可正常使用。 • training：表示当前 Collection 正在进行数据训练，即训练模型以生成向量数据。 • building：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • failed：重建索引失败，可能会影响集合读写操作。
		startTime	重建索引开始的时间。

truncate

最近更新时间：2024-09-27 18:00:12

Method 与 URL

清空 Collection 中所有的数据与索引：

`POST https://{实例访问 IP 地址}:{实例网络端口}/collection/truncate` 。

请求示例

> **警告：**
> 执行 truncate 操作将会永久删除指定 Collection 下的所有数据。在操作之前，务必谨慎考虑。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/truncate \
 -d '{
    "database": "db-test",
    "collection": "book-vector"
}'
```

请求参数

参数名	是否必选	参数含义	配置方法
database	是	指定 Collection 所在的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需删除数据的集合所属的数据库名。
collection	是	所需清空数据的 Collection 名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需清空数据的集合。

响应消息

执行成功，输出如下信息。

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1
}
```

参数名	参数含义
affectedCount	清空的集合数量。

CollectionView create

最近更新时间：2025-01-06 15:35:22

Method 与 URL

在已创建的 AI 类 Database 中创建 CollectionView：
`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/collectionView/create` 。

使用示例

> **注意：**
> 如下示例 api_key 的值 A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X，需要分别替换为具体实例的 API Key 以及访问地址。

如下示例，为 AI 类数据库 **db-test-ai**，创建一个集合视图 **coll-ai-files**，用于直接存储文件。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/ai/collectionView/create \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "description": "this is a collection view description",
    "embedding": {
        "language": "zh",
        "enableWordsEmbedding": true
    },
    "splitterPreprocess": {
        "appendTitleToChunk": false,
        "appendKeywordsToChunk": true,
        "chunkSplitter": "\n{2,}"
    },
    "parsingProcess": {
        "parsingType": "VisionModelParsing"
    },
    "indexes": [
        {
```

```
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "tags",
            "fieldType": "array",
            "indexType": "filter"
        }
    ]
}'
```

请求参数

参数	参数含义	子参数	是否必选	配置方法
database	指定 CollectionView 所在的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制数据库名。
collectionView	指定 CollectionView 的名称。	-	是	CollectionView 命名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
description	指定 CollectionView 的描述信息	-	否	• 取值类型：string。 • 字符长度要求：[1,256]。 • 示例：this is the collection view description。
embedding	Embedding 相关配置	language	否	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • multi：多语言。
		enableWordsEmb	否	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。

		edding		• **false**：不开启，默认为 **false**。
splitterPreprocess	文件预处理方式配置	appendTitleToChunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。默认值为 false。 • **true**：将段落 Title 追加到切分后的段落。
		appendKeywordsToChunk	否	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 **keywords** 追加到切分后的段落。默认值为 true。
		chunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
indexes	配置需使用 Filter 索引的字段，以便检索时使用该字段的 Filter 条件表达式过滤查找文档。	fieldName	是	自定义配置可作为 Filter 索引的文件 meta 信息的标量字段名。 **说明：** • Filter 索引（Filter Index）是建立在标量字段的索引。该标量字段名称、类型均由用户自定义，不限制标量字段数量。 • 标量字段被建立 Filter 索引之后，向量检索时，将依据 Filter 指定的标量字段的条件表达式进行过滤查询和范围查询来匹配相似向量。 • 建立 Filter 索引时，选取需要使用 Filter 表达

				式高效过滤数据的标量字段。不做过滤查询、检索的标量字段不必建立 Filter 索引。切勿将所有标量字段建立索引，导致内存资源的浪费。
		fieldType	是	指定自定义字段的数据类型。取值如下： • **string**：字符型。 • **uint64**：指无符号整数（unsigned integer）。 • **array**：数组类型，数组元素为 string。
		indexType	否	该参数固定设置为 `filter`。
parsingProcess	PDF 文件解析方式相关参数	parsingType	否	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

响应消息

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1
}
```

参数名	参数含义

affectedCount	影响行数，即为创建集合数量。

drop

最近更新时间：2024-09-27 18:00:12

Method 与 URL

删除 AI 类数据库中存在的 CollectionView：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/collectionView/drop` 。

> **警告：**
> 执行 drop 操作将会永久删除指定 collectionView 下的所有数据。在操作之前，务必谨慎考虑。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/collectionView/drop \
 -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files"
}'
```

请求参数

参数名	是否必选	参数含义	配置方法
database	是	指定 CollectionView 所在的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需删除集合所属的数据库名。
collectionView	是	所需删除的 CollectionView 名称。	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制需删除的 CollectionView 。

响应消息

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1
}
```

参数名	参数含义
affectedCount	影响行数，即为删除集合数量。

list

最近更新时间：2024-12-05 20:37:03

Method 与 URL

查询指定 AI 类 Database 中所有的 CollectionView：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/collectionView/list` 。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/collectionView/list \
 -d '{
    "database": "db-test-ai"
}'
```

请求参数

参数	是否必选	参数含义	配置方法
database	是	设置 CollectionView 所属的 Database 库名。	使用 /database/list 获取集群中的数据库列表，复制需删除集合所属的数据库名。

响应信息

```
{
  "code": 0,
  "msg": "Operation success, requestId:
73c1bbef76709b17c6b9ef**********",
  "collectionViews": [
    {
      "database": "db-test-ai",
      "collectionView": "coll-ai-files",
      "description": "this is a collection description",
      "embedding": {
```

```
        "language": "zh",
        "enableWordsEmbedding": false
      },
      "splitterPreprocess": {
        "appendTitleToChunk": true,
        "appendKeywordsToChunk": true
      },
      "parsingProcess":{
          "parsingType":"VisionModelParsing"
      },
      "indexes": [
        {
          "fieldName": "tags",
          "fieldType": "array",
          "indexType": "filter"
        },
        {
          "fieldName": "documentSetName",
          "fieldType": "string",
          "indexType": "filter"
        },
        {
          "fieldName": "documentSetId",
          "fieldType": "string",
          "indexType": "primaryKey"
        },
        {
          "fieldName": "author",
          "fieldType": "string",
          "indexType": "filter"
        }
      ],
      "createTime": "2023-11-27 17:16:54",
      "alias": [
        "alias-coll-ai-files"
      ],
      "stats": {
        "indexedDocumentSets": 0,
        "totalDocumentSets": 0,
        "unIndexedDocumentSets": 0
      }
    }
  ]
```

```
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 CollectionView 所在的 AI 类 Database 名称。
collectionView	-	-	显示 CollectionView 的名称。
embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • mutil：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
alias	-	-	CollectionView 的所有别名。创建别名，请参见 /ai/alias/set 。
createTime	-	-	显示 CollectionView 的创建时间。
description	-	-	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。
		unIndexedDocumentSets	未处理的文件数量。

splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将段落 Title 追加到切分后的段落。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 keywords 追加到切分后的段落。
parsingProcess	文件解析方式相关参数	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 documentSetId 文件ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 documentSetName 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式进行检索。
	其他自定义需建立 Filter 索引的	fieldName	自定义扩展字段，例如：author、

	字段		tag。
		filedType	显示自定义字段的数据类型，支持：uinit、string、array。
		indexType	显示自定义字段索引类别为 `filter` 。

describe

最近更新时间：2024-12-05 20:37:03

Method 与 URL

查询指定的 CollectionView：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/collectionView/describe` 。

请求示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/collectionView/describe \
 -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files"
}'
```

请求参数

参数	是否必选	参数含义	配置方法
database	是	设置 CollectionView 所属的 Database 库名。	使用 /database/list 获取集群中的数据库列表，复制需删除集合所属的数据库名。

响应信息

```
{
  "code": 0,
  "msg": "Operation success, requestId:
73c1bbef76709b17c6b9ef**********",
  "collectionViews": [
    {
      "database": "db-test-ai",
      "collectionView": "coll-ai-files",
      "description": "this is a collection description",
```

```
    "embedding": {
      "language": "zh",
      "enableWordsEmbedding": false
    },
    "splitterPreprocess": {
      "appendTitleToChunk": true,
      "appendKeywordsToChunk": true
    },
    "parsingProcess":{
        "parsingType":"VisionModelParsing"
    },
    "indexes": [
      {
        "fieldName": "tags",
        "fieldType": "array",
        "indexType": "filter"
      },
      {
        "fieldName": "documentSetName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "documentSetId",
        "fieldType": "string",
        "indexType": "primaryKey"
      },
      {
        "fieldName": "author",
        "fieldType": "string",
        "indexType": "filter"
      }
    ],
    "createTime": "2023-11-27 17:16:54",
    "alias": [
      "alias-coll-ai-files"
    ],
    "stats": {
      "indexedDocumentSets": 0,
      "totalDocumentSets": 0,
      "unIndexedDocumentSets": 0
    }
  }
```

```
    ]
}
```

参数	子参数	子参数	参数含义
database	-	-	显示 CollectionView 所在的 AI 类 Database 名称。
collectionView	-	-	显示 CollectionView 的名称。
embedding		language	指定文件的语言类型，取值如下所示： • zh：中文。 • en：英文。 • mutil：多语言。
		enableWordsEmbedding	配置在检索时，是否开启词（Words）向量精排，并进行词向量化。 • true：开启。 • false：不开启，默认为 false。
alias	-	-	CollectionView 的所有别名。创建别名，请参见 /ai/alias/set 。
createTime	-	-	显示 CollectionView 的创建时间。
description	-	-	显示 CollectionView 的描述信息。
stats	文件处理的状态	indexedDocumentSets	已处理完成的文件的数量。
		totalDocumentSets	所有的文件的数量。

		unIndexedDocumentSets	未处理的文件数量。
splitterPreprocess	文件预处理策略	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将段落 Title 追加到切分后的段落。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 **keywords** 追加到切分后的段落。
parsingProcess	文件解析方式相关参数	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
Indexes	默认以 **documentSetId** 文件 ID 创建主键索引	fieldName	标识索引对象为 `documentSetId` 。
		filedType	显示该索引对象的数据类型，固定为 `string` 。
		indexType	该参数固定显示为 `primaryKey` 。
	默认以 **documentSetName** 文件名创建 Filter 索引	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。
		filedType	显示索引对象为文件名的数据类型，固定为 `string` 。
		indexType	显示索引对象为文件名的索引类型，固定为 `filter` 。在后续检索数据时，才能对该字段设置 Filter 条件表达式进行检

版权所有：腾讯云计算（北京）有限责任公司

			索。
	其他自定义需建立 Filter 索引的字段	fieldName	自定义扩展字段，例如：author、tag。
		filedType	显示自定义字段的数据类型，支持：uinit、string、array。
		indexType	显示自定义字段索引类别为 `filter` 。

truncate

最近更新时间：2024-09-27 18:00:12

Method 与 URL

清空 CollectionView 中所有的数据与索引：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/collectionView/truncate` 。

> **警告：**
> 执行 truncate 操作将会永久删除指定 CollectionView 下的所有数据。在操作之前，务必谨慎考虑。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/collectionView/truncate \
 -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files"
}'
```

请求参数

参数名	是否必选	参数含义	配置方法
database	是	指定 CollectionView 所在的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需删除数据的集合所属的数据库名。
collectionView	是	所需清空数据的 CollectionView 名称。	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制需清空集合视图。

响应消息

```
{
  "code": 0,
  "msg": "operation success",
  "affectedCount": 1
}
```

参数名	参数含义
affectedCount	清空的集合数量

版权所有：腾讯云计算（北京）有限责任公司

Alias set

最近更新时间：2024-09-27 18:00:12

Method 与 URL

别名可以是一个简短的字符串，方便标识和访问对应的集合。通常，别名会替换 Collection 或 CollectionView 的名称用于业务切换等场景。

- 为 Base 数据库的 Collection 指定别名：
  `POST https://{实例访问 IP 地址}:{实例网络端口}/alias/set`
- 为 AI 类数据库的 CollectionView 指定别名：
  `POST https://{实例访问 IP 地址}:{实例网络端口}/ai/alias/set`

接口约束

- DB 和 Collection 级别（包含 AI 类数据库的 CollectionView ）的 drop 操作会同时删除库表下的所有别名。
- Document 与 DocumentSet 层级的访问优先访问别名，其余级别仅支持原 Collection 或 CollectionView 名操作。
- 集合或集合视图的别名可以和名称重复，一个集合或集合视图的多个别名之间不能重复。

> **说明：**
> 通过集合的别名做业务迁移时，仅需通过 **/alias/set** 或 **/ai/alias/set** 接口将同一别名指向新的集合，别名与集合的映射关系将自动更新为新集合，可直接通过别名访问新集合。

使用示例

> **注意：**
> 如下示例 api_key 的值 A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X，需要分别替换为具体实例的 API Key 以及访问地址。

为 Collection 设置别名

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
```

```
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/alias/set \
 -d '{
    "database": "db-test",
    "collection": "book-emb",
    "alias": "alias-book-emb"
}'
```

执行成功，返回消息，如下所示。

```
{"code":0,"msg":"operation success","affectedCount":1}
```

为 CollectionView 设置别名

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/alias/set \
 -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "alias": "alias-coll-ai-files"
}'
```

执行成功，返回消息，如下所示。

```
{"code":0,"msg":"operation success","requestId:
46ef397c7e059c1769bcef**********","affectedCount":1}
```

请求参数

参数	是否必选	参数含义	配置方法及要求

database	是	指定需创建别名的 Collection 或 CollectionView 所对应的 Database 名。	使用 /database/list 获取集群中的数据库列表，复制需创建别名的集合所属的数据库名。
collection	是	为 Collection 设置别名，指定需创建别名的 Collection 名。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需创建别名的集合名。
collectionView	是	为 CollectionView 设置别名，指定需创建别名的 CollectionView 名。	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制需创建别名的集合视图。
alias	是	设置别名。	别名要求如下： • 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。

响应消息

```
{
  "code": 0,
  "msg": "operation success",
  "affectedCount": 1
}
```

参数名	参数含义
affectedCount	创建别名的集合数量。

delete

最近更新时间：2024-09-27 18:00:12

Method 与 URL

删除别名，用于清理 Collection 或 CollectionView 中已经过期或废弃的别名。

- 删除 Base 类数据库指定 Collection 的别名：
  `POST https://{实例访问 IP 地址}:{实例网络端口}/alias/delete` 。
- 删除 AI 类数据库指定 CollectionView 的别名：
  `POST https://{实例访问 IP 地址}:{实例网络端口}/ai/alias/delete` 。

使用示例

删除 Collection 别名

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/alias/delete\
 -d '{
    "database": "db-test",
    "alias": "alias-book-emb"
}'
```

执行成功，返回信息，如下所示。

```
{"code":0,"msg":"operation success","affectedCount":1}
```

删除 CollectionView 别名

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
```

```
  http://10.0.X.X:80/ai/alias/delete\
 -d '{
    "database": "db-test-ai",
    "alias": "alias-coll-ai-files"
}'
```

执行成功，返回信息，如下所示。

```
{"code":0,"msg":"operation success","requestId:
459fa606c5059c176bbcef5792800f7c","affectedCount":1}
```

请求参数

参数	是否必选	参数含义
database	是	指定需删除的 Collection 或 CollectionView 所属的 Database 名。
alias	是	指定需删除的别名。

返回参数

```
{
    "code":0,
    "msg":"operation success",
    "affectedCount":1
}
```

参数名	参数含义
affectedCount	删除别名的集合数量。

Document upsert

最近更新时间：2025-07-01 10:19:43

本接口（ `/document/upsert` ）用于给 Collection 中插入向量或文本信息。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/upsert` 。

> **说明：**
> 在插入数据时， Collection 中已经存在相同 ID 的 Document，则会删除源 Document，插入新的 Document 数据。

使用示例

> **注意：**
> 如下示例 api_key 的值 A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X，需要分别替换为具体实例的 API Key 以及访问地址。

插入稠密向量+稀疏向量

如果您无需使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能做向量化，则可以直接写入向量数据。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/document/upsert \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "documents": [
        {
            "id": "0001",
            "vector": [
                0.2123,
```

```
                0.23,
                0.213
            ],
            "sparse_vector": [[2,0.96],[5,0.53],[100,0.443]],
            "author": "罗贯中",
            "bookName": "三国演义",
            "page": 21
        },
        {
            "id": "0002",
            "vector": [
                0.2123,
                0.22,
                0.213
            ],
            "sparse_vector": [[2,0.97],[5,0.54],[100,0.445]],
            "author": "吴承恩",
            "bookName": "西游记",
            "page": 22
        },
        {
            "id": "0003",
            "vector": [
                0.2123,
                0.21,
                0.213
            ],
            "sparse_vector": [[2,0.97],[5,0.54],[100,0.445]],
            "author": "曹雪芹",
            "bookName": "红楼梦",
            "page": 23
        }
    ]
}'
```

插入文本数据+稀疏向量

如果您使用 Embedding 功能，在 /collection/create 建表时，配置 Embedding 模型相关参数之后，便可以通过 **/document/upsert** 接口可直接传入原始文本。Embedding 模型会将原始文本转换为向量数据，并将转换后的向量数据以及原始文本一并存入数据库。具体信息，请参见 Embedding 介绍 。如下示例，基于 /collection/create 创建的集合 **book-emb**，写入原始文本。

> **说明：**
> 创建 Collection 时，已指定 Embedding 模型，则只能插入文本信息，不能插入向量数据。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/upsert \
  -d '{
    "database": "db-test",
    "collection": "book-emb",
    "buildIndex": true,
    "documents": [
        {
            "id": "0001",
            "text": "话说天下大势，分久必合，合久必分。",
            "sparse_vector": [[2,0.97],[5,0.54],[100,0.445]],
            "author": "罗贯中",
            "bookName": "三国演义",
            "page": 21
        },
        {
            "id": "0002",
            "text": "混沌未分天地乱，茫茫渺渺无人间。",
            "author": "吴承恩",
            "bookName": "西游记",
            "page": 22
        },
        {
            "id": "0003",
            "text": "甄士隐梦幻识通灵，贾雨村风尘怀闺秀。",
            "author": "曹雪芹",
            "bookName": "红楼梦",
            "page": 23
        }
    ]
}'
```

写入二进制数据

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/upsert \
  -d '{
    "database": "db-test",
    "collection": "bin-vector",
    "documents": [
        {
            "id": "0001",
            "vector": [21, 13],
            "page": 21
        }
    ]
}'
```

写入 json 数据

以下示例展示了在开启 autoId 功能的情况下，写入 json 类型字段 bookInfo 的操作。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/upsert \
  -d '{
    "database": "db-test",
    "collection": "coll-json-test",
    "documents": [
        {
            "vector": [
                0.2123,
                0.23,
                0.213
            ],
            "bookInfo": {
                "bookName": "三国演义",
                "author": "罗贯中"
```

```
                }
            }
        ]
}'
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要插入的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需写入数据的集合所属的数据库名。
collection	指定要插入数据的 Collection 名称。	-	是	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需写入数据的集合名。
buildIndex	指定是否需要更新索引。	-	否	取值如下所示： • **true**：需更新索引。默认值是 **true**。 • **false**：不更新索引。 **注意：** 如果创建 Collection 选择的索引类型为 IVF 系列： • 当第一次写入时，当前集合还没有向量索引，此时 **buildIndex** 必须为 **false**。插入原始数据之后，需通过 rebuild 训练数据并重建索引。 • 当集合已经调用过 rebuild 创建索引后，集合已经存在向量索引，此时： ○ 如果 **buildIndex** = **true**，表示新写入的数据会加入到已有的 IVF 索引中，但不会更新索引结构，此时新写入的数据可以被检索到。 ○ 如果 **buildIndex** = **false**，表示新写入的数据不会加入到已有的 IVF 索引中，此时新写入的数据无法被检索到。

				具体应用实践，请参见 应用 IVF 系列索引 。
documents	指定要插入的 Document 数据，是一个数组，支持单次插入一条或者多条 Document，最大可插入1000条。	id	是	Document 主键，长度限制为[1,128]。
		vector	否	表示文档的向量值， • 集合的向量索引为 FLAT/HNSW/IVF 时，直接传入向量数据，以32位浮点数存储，如[0.2123, 0.9281, ..., 0.6712]。 • 集合的向量索引为二进制索引时，写入向量时需要以十进制写入，并且维度是创建集合时指定维度的1/8。如创建集合时指定索引为 BIN_FLAT，维度为8维，则 upsert 时可以直接写入[150]，其代表的二进制数据为“10010110”。 **说明：** 如果使用 Embedding 功能做向量化，则无需写入向量数据。
		text	否	该参数为 创建集合 时，Embedding 参数 **field** 对应指定的文本字段名。写入原始文本。 **说明：** • 写入原始文本数据，系统会自动从该字段中提取原始文本信息，并将其转换为向量数据，并将原始文本以及转化后的向量数据一起存储在数据库中。 • 输入的文本长度有限制，超过512个 token，约400个字符之后会自动截断。
		other_scalar_field	否	其他标量字段，用于存储文档的其他信息。例如：请求示例中的 bookName、author、page、segment。 **说明：** 若字段为存储数据过期时间戳的标量字段，需以标准的 Unix 时间戳格式为该字段赋值，且数据类型为 uint64。例如：`"expired_at": 1728372822` 。

				• 若时间戳字段为空未指定具体的过期时间，插入的该条数据将不会执行过期删除操作，数据将永久保留。 • 若时间戳字段传入 Value 的数据类型有误或者不符合 Unix 时间戳规范时，将会报出错误信息，请根据提示信息修改。 • 若时间戳字段指定的过期时间小于当前插入数据时的时间，则在下一个定时删除操作时，直接删除该条数据。
			否	Json 类型是一种由键值对组成的数据对象，使用花括号{}进行包裹。在 Json 对象中，键（Key）和值（Value）通过冒号（:）连接。例如，Json 类型的对象 a 的写入格式为 `"a": {"b": "test", "c": 12}` 。更多信息，请参见 Json 类型 。 **说明：** Json 字段的键（Key）命名规则：仅允许使用字母、数字及下划线（_）组成键名，且键名必须以字母或下划线开头，不得以数字开头。

响应消息

```
{"code":0,"msg":"Operation
success","affectedCount":3,"embeddingExtraInfo":{"tokenUsed":51}}
```

参数名	参数含义
affectedCount	影响行数，即插入的文档数量。
embeddingExtraInfo	消耗的 Token 的数量。

query

最近更新时间：2025-07-01 10:19:43

本接口（ `/document/query` ）用于精确查找与查询条件完全匹配的文档。

- 支持根据主键 id（Document ID），搭配自定义的标量字段的 Filter 表达式一并检索。
- 支持指定查询起始位置 offset 和返回数量 limit，实现数据 SCAN 能力。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/query`

使用示例

使用 **/document/query** 接口，查询集合 **book-vector** 中， Document Id 为0001、0002、0003中，满足 Filter 表达式 `bookName in (\"三国演义\",\"西游记\")` 的文档。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/query \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "readConsistency": "eventualConsistency",
    "query": {
        "documentIds": [
            "0001",
            "0002",
            "0003"
        ],
        "retrieveVector": true,
        "filter": "bookName in (\"三国演义\",\"西游记\")",
        "limit": 3,
        "offset": 0,
        "outputFields": [
            "id",
            "author",
            "bookName",
            "page"
        ]
    }
```

```
}'
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要查询的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需查询数据的集合所属的数据库名。
collection	指定要查询的 Collection 名称。	-	是	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需查询数据的集合名。
readConsistency	配置读一致性要求。	-	否	取值如下所示，默认为 eventualConsistency。 • **strongConsistency**：强一致性。 • **eventualConsistency**：最终一致性。
query	设置查询条件。	documentIds	否	表示要查询的文档的所有 ID，支持批量查询，数组元素范围[1,20]。
		retrieveVector	否	标识是否需要返回检索结果的向量值。 • **true**：需要。 • **false**：不需要。默认为 false。
		filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。

			○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如，exipred_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `"filter": "bookInfo.bookName in (\"三国演义\",\"西游记\")"` • <value>：表示要匹配的值。
	limit	否	每页返回的 Document 数量。默认为1。 • 数据类型：uint 64。 • 取值范围：[1,16384] ⚠ **注意：** 若使用 **query** 检索数据时，不配置 **documentIds** 和 **filter** 参数，则必须配置 **offeset** 和 **limit** 参数，返回从 offset 开始的 limit 条数据，避免遍历所有数据而浪费不必要的资源。
	offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 **limit** 整数倍。 • 计算公式：offset = limit * (page−1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 − 1) = 10，表示从查询结果的第11条记录开始返回数据。

		sort	否	指定查询结果按照特定字段升序（或降序）排序，数据类型为数组。 `"sort": [{ "fieldName": "scalar1", "direction": "desc" }]` • **fieldName**：指定特定字段，查询结果将依据该字段确定输出顺序。当前排序字段仅支持 uint64 的类型，且该字段为标量字段索引，可通过接口 describe 确认该字段是否为索引字段。 **说明：** • 当满足条件的数据量小于或等于1000时，orderby 操作将正常生效，否则会引发错误。 • 目前仅支持对单个字段进行排序，暂时不支持多字段排序功能。 • **direction**：指定排序的方向。 ○ desc：降序排序。 ○ asc：升序排序，默认为 asc。
		outputFields	否	以数组形式配置需返回的字段。 **说明：** • outputFields 与 retrieveVector 参数均可以配置是否输出向量值，二者任意一个配置需输出向量字段，则将输出向量字段。 • 输出 Json 字段时，outputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，outputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

响应消息

查询结果，如下所示，返回了 id（Document ID）为2的文档数据。


```
{
  "code": 0,
  "msg": "Operation success",
  "count": 2,
  "documents": [
    {
      "id": "0002",
      "vector": [
        0.21230000257492066,
        0.2199999988079071,
        0.21299999952316285
      ],
      "sparse_vector": [
        [2, 0.9700000286102295],
        [5, 0.5400000214576721],
        [100, 0.4449999928474426]
      ],
      "bookName": "西游记",
      "page": 22,
      "author": "吴承恩"
    },
    {
      "id": "0001",
      "vector": [
        0.21230000257492066,
        0.23000000417232514,
        0.21299999952316285
      ],
      "sparse_vector": [
        [2, 0.9599999785423279],
        [5, 0.5299999713897705],
        [100, 0.4429999887943268]
      ],
      "page": 21,
      "author": "罗贯中",
      "bookName": "三国演义"
    }
  ]
}
```

参数名	子参数	参数含义
count	-	统计返回的 Document 数量。
documents	id	Document 的 ID 信息。
	vector	Document 的向量值。
	other_scalar_field	Document 其他自定义的标量字段。例如：author、page、bookName。 **说明：** • 自定义输出字段为在请求参数 **outputFields** 中配置的字段。 • 如果创建 Collection 时配置 Embedding 参数，写入原始文本，则可通过 **outputFields** 参数配置输出 Embedding 的文本字段。

search

最近更新时间：2025-07-09 15:17:22

本接口（ `/document/search` ）用于检索与给定的查询向量最相似的文档，适用于仅写入稠密向量的集合。

- 支持根据指定 id 或向量数值进行相似度检索，返回指定的 Top K 个最相似的 Document。
- 支持根据主键 id（Document ID）或向量数值，搭配自定义的标量字段的 Filter 表达式一并进行相似度检索。
- 配置 Embedding 参数，支持输入原始文本检索与输入文本相似的文档，同时，支持搭配标量字段的 Filter 表达式一并检索。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/search` 。

> **说明：**
> search 接口支持根据主键 id（Document ID）、向量 vector 以及文本进行检索。id、vector、文本仅需指定其中一个即可。

使用示例

根据文本进行向量检索

如果使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能，基于 /document/upsert 写入的原始文本，在集合 **book-emb** 中，使用 /document/search 接口查询与输入文本 `'天下大势，分久必合，合久必分'` 最相似，且满足过滤条件的 Top3 数据。因写入文本数据为高维数据，不便于显示，则设置 **retrieveVector** 参数为 **false**，检索信息不显示向量数据。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/search \
  -d '{
    "database": "db-test",
    "collection": "book-emb",
    "search": {
        "embeddingItems": [
            "天下大势，分久必合，合久必分"
        ],
```

```
        "limit": 3,
        "params": {
            "ef": 200
        },
        "retrieveVector": false,
        "filter": "bookName in (\"三国演义\",\"西游记\")",
        "outputFields": [
            "id",
            "author",
            "text",
            "bookName"
        ]
    }
}'
```

检索信息如下所示。具体返回参数含义，请参见 输出参数 。

说明：

- 返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值，如果检索的数据不足 K 条，则返回实际检索的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。


```
{
    "code": 0,
    "msg": "operation success",
    "embeddingExtraInfo":{"tokenUsed":14},
    "documents": [
        [
            {
                "id": "0001",
                "score": 0.9792741537094116,
                "bookName": "三国演义",
                "author": "罗贯中",
                "text": "话说天下大势，分久必合，合久必分。"
            },
            {
                "id": "0002",
                "score": 0.7909858226776123,
                "bookName": "西游记",
                "author": "吴承恩",
```

```
                "text": "混沌未分天地乱，茫茫渺渺无人间。"
            }
        ]
    ]
}
```

根据 id（Document ID）进行向量检索

基于 /document/upsert 写入的向量数据，使用 /document/search 接口在集合 book-vector 中，查询与 id（Document ID）为0001、 0002、0003最相似，且满足过滤条件的 Top3 数据。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/search \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "search": {
        "documentIds": [
            "0001",
            "0002",
            "0003"
        ],
        "params": {
            "ef": 200
        },
        "retrieveVector": true,
        "filter": "bookName in (\"三国演义\",\"西游记\")",
        "limit": 3
    }
}'
```

检索信息，如下所示。

> **说明：**
> - 每一个查询结果都返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值，如果检索的数据不足 K 条，则返回实际的 Document 数量。

- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。相似程度则通过 L2（欧几里得距离）、IP（内积）或 COSINE（余弦相似度）计算得出的分数来衡量，输出参数 score 表示相似性计算分数。其中，欧式距离（L2）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。


```
{
    "code": 0,
    "msg": "operation success",
    "documents": [
        [
            {
                "id": "0001",
                "vector": [
                    0.21230000257492066,
                    0.23000000417232514,
                    0.21299999952316285
                ],
                "score": 1.0000001192092896,
                "author": "罗贯中",
                "bookName": "三国演义",
                "page": 21
            },
            {
                "id": "0002",
                "vector": [
                    0.21230000257492066,
                    0.2199999988079071,
                    0.21299999952316285
                ],
                "score": 0.9997729659080505,
                "bookName": "西游记",
                "author": "吴承恩",
                "page": 22
            }
        ],
        [
            {
                "id": "0002",
                "vector": [
                    0.21230000257492066,
                    0.2199999988079071,
```

```
                0.21299999952316285
            ],
            "score": 1.0000002384185791,
            "author": "吴承恩",
            "page": 22,
            "bookName": "西游记"
        },
        {
            "id": "0001",
            "vector": [
                0.21230000257492066,
                0.23000000417232514,
                0.21299999952316285
            ],
            "score": 0.9997729659080505,
            "bookName": "三国演义",
            "author": "罗贯中",
            "page": 21
        }
    ],
    [
        {
            "id": "0002",
            "vector": [
                0.21230000257492066,
                0.2199999988079071,
                0.21299999952316285
            ],
            "score": 0.9997580051422119,
            "author": "吴承恩",
            "bookName": "西游记",
            "page": 22
        },
        {
            "id": "0001",
            "vector": [
                0.21230000257492066,
                0.23000000417232514,
                0.21299999952316285
            ],
            "score": 0.9990617632865906,
            "page": 21,
            "bookName": "三国演义",
```

```
                "author": "罗贯中"
            }
        ]
    ]
}
```

根据 Vector 进行向量检索

基于 /document/upsert 写入的向量数据，则可使用 /document/search 接口，在集合 book-vector 中，根据指定向量以及过滤条件，查找 Top 3 个相似性结果。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/search \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "search": {
        "vectors": [
            [
                0.3123,
                0.43,
                0.213
            ]
        ],
        "params": {
            "ef": 200
        },
        "filter": "bookName in (\"三国演义\",\"西游记\")",
        "retrieveVector": true,
        "limit": 3
    }
}'
```

检索信息如下所示。具体返回参数含义，请参见 输出参数 。

> **说明：**

- 返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值，如果检索的数据不足 K 条，则返回实际检索的 Document 数量。
- 检索结果会按照与查询向量的相似程度进行排列，相似度最高的结果会排在最前面，相似度最低的结果则排在最后面。相似程度则通过 L2（欧几里得距离）、IP（内积）或 COSINE（余弦相似度）计算得出的分数来衡量，输出参数 score 表示相似性计算分数。其中，欧式距离（L2）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。


```
{
    "code": 0,
    "msg": "operation success",
    "documents": [
        [
            {
                "id": "0001",
                "vector": [
                    0.21230000257492066,
                    0.23000000417232514,
                    0.21299999952316285
                ],
                "score": 0.9714228510856628,
                "page": 21,
                "author": "罗贯中",
                "bookName": "三国演义"
            },
            {
                "id": "0002",
                "vector": [
                    0.21230000257492066,
                    0.2199999988079071,
                    0.21299999952316285
                ],
                "score": 0.9668837785720825,
                "bookName": "西游记",
                "author": "吴承恩",
                "page": 22
            }
        ]
    ]
```

```
}
```

二进制向量检索

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/search \
  -d '{
    "database": "db-test",
    "collection": "bin-vector",
    "search": {
        "vectors": [
            [
                20,
                18
            ]
        ],
        "retrieveVector": true,
        "limit": 2
    }
}'
```

响应信息，如下所示。

```
{"code":0,"msg":"Operation success","documents":
[[{"id":"0001","vector":[21.0,13.0],"score":6.0,"page":21}]]}
```

json 类型检索

如下示例，使用 json 类型的字段 bookInfo 的 Filter 表达式检索数据。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
```

```
  -H 'Authorization: Bearer
account=root&api_key=9M6AucQGql5FlMyuBNnUuzrd10W7MTmZ0L2ZFDx7' \
  http://gz-vdb-jggokqoy.sql.tencentcdb.com:8100/document/search \
  -d '{
    "database": "db-test-1",
    "collection": "coll-json-test",
    "search": {
        "vectors": [
            [
                0.3123,
                0.43,
                0.213
            ]
        ],
        "params": {
            "ef": 200
        },
        "filter": "bookInfo.bookName in (\"三国演义\",\"西游记\")",
        "retrieveVector": true,
        "limit": 3
    }
}'
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要查询的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需查询数据的集合所属的数据库名。
collection	指定要查询的 Collection 名称。	-	是	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需查询数据的集合名。
readConsi	设置检索数据的一	-	否	取值如下所示： • strongConsistency：强一致性。

stency	致性要求。			• eventualConsistency：最终一致性。
search	表示查询条件。	vectors	否	指定要查询的向量列表。 • 数组元素数量最大为20。 • vectors 与 documentIds 两个字段，只需配置其中一个即可。 • 集合为二进制索引时，支持使用十进制写入。原始二进制每8bit（即1字节）被视为向量的一个维度的值。以十进制写入数据的长度=创建集合时指定的维度/8。例如：创建集合时，指定的维度为128，那么写入数据时元素的长度只能为16。
		documentIds	否	指定待查询的文档 ID 列表。数组元素数量最大为20。 **注意：** vectors、documentIds、embeddingItems 三个字段，只需配置其中一个即可。这三个字段任何一个可以结合 filter 字段进行混合检索。
		embeddingItems	否	输入文本信息，用于检索与该文本信息相似的数据。 • 类型：字符串数组。 • 范围：数组元素最大批量为20。
		params	否	索引类型不同，检索时，所需配置的参数不同。 • FLAT：无需指定参数。 • HNSW 类型：需配置参数 ef，指定需要访问向量的数目。取值范围[1,32768]，默认为10。 • IVF 系列：需设置参数 nprobe，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置，可通过 /collection/list 查看。
		filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。

		○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `"filter" : "bookInfo.bookName in (\"三国演义\",\"西游记\")"` • <value>：表示要匹配的值。
retrieveVector	否	标识是否需要返回检索结果的向量值。 • **true**：需要。 • **false**：不需要。默认为 false。
limit	是	指定返回最相似的 Top K 条数据的 K 的值。K 为大于0的正整数。
radius	否	• IP：score >= radius 时返回，取值范围 (-∞, +∞)。 • COSINE：score >= radius 时返回，取值范围 [-1, 1]。 • L2：score <= radius 时返回，取值范围 [0, +∞)。
outputFields	否	指定需要输出的字段。若不设置，将返回所有字段。

				说明： • retrieveVector 和 outputFields 只要有其中一个配置输出向量字段即可输出 vector。 • 输出 Json 字段时，outputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，outputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

输出参数

参数名	参数名	参数含义
embeddingExtraInfo	tokenUsed	使用 Embedding 功能已经消耗的 Token 数量。
documents	id	Document 的 ID 信息。
	vector	Document 的向量值。
	score	表示查询向量与检索结果向量之间的相似性计算分数。欧式距离（L2）与汉明距离（Hamming Distance）计算所得的分数越小与搜索值越相似；而余弦相似度（COSINE）与内积（IP）计算所得的分数越大与搜索值越相似。
	text	如果该 Collection 开启 Embedding 功能，在建表时，指定文本信息的字段名，则显示该字段。
	other_scalar_field	Document 其他自定义的标量字段。例如：author、bookName、page 等。

版权所有：腾讯云计算（北京）有限责任公司

hybridSearch

最近更新时间：2025-07-01 10:19:43

本接口（ `/document/hybridSearch` ）支持稠密向量与稀疏向量两路混合检索，用于检索与指定的稠密向量与稀疏向量综合相似性最高的文档。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/hybridSearch` 。

使用示例

稠密向量 + 稀疏向量

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/hybridSearch \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "search": {
        "ann": [
          {
              "fieldName": "vector",
              "data": [[0.3123, 0.43, 0.213]],
              "params": {
                  "ef": 200
              },
                "limit": 10
          }
        ],
       "match": [{
                "fieldName": "sparse_vector",
                "data": [[[2, 0.9611]]],
                "limit": 10,
                "terminateAfter": 4000,
                "cutoffFrequency": 0.1
        }],
```

```
        "rerank": {
                "method": "weighted",
                "fieldList": ["vector", "sparse_vector"],
                "weight": [0.9, 0.1]
        },
        "limit": 3,
        "retrieveVector": false,
        "filter": "bookName in (\"三国演义\",\"西游记\")",
        "outputFields": [
            "id",
            "author",
            "bookName"
        ]
    }
}'
```

文本 + 稀疏向量检索

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/hybridSearch \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "search": {
        "ann": [
          {
              "fieldName": "text",
              "data": ["什么是向量数据库"],
              "params": {
                  "ef": 200
              },
                "limit": 10
          }
        ],
       "match": [{
                "fieldName": "sparse_vector",
                "data": [[[2, 0.9611]]],
```

```
                "limit": 10,
                "terminateAfter": 4000,
                "cutoffFrequency": 0.1
            }],
            "rerank": {
                    "method": "weighted",
                    "fieldList": ["vector", "sparse_vector"],
                    "weight": [0.9, 0.1]
            },
            "limit": 3,
            "retrieveVector": false,
            "filter": "bookName in (\"三国演义\",\"西游记\")",
            "outputFields": [
                "id",
                "author",
                "bookName"
            ]
        }
    }'
```

入参描述

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要查询的 Database 名称。	-	是	获取集群中的数据库列表，复制需查询数据的集合所属的数据库名。
collection	指定要查询的 Collection 名称。	-	是	获取集群中的集合列表，复制需查询数据的集合名。
readConsistency	设置检索数据的一致性要求。	-	否	取值如下所示： • strongConsistency：强一致性。 • eventualConsistency：最终一致性。

search	配置检索条件	ann	是	配置检索信息。 • **fieldName**：检索的字段名，支持设置为：vector 或 text。 **说明：** 其中，vector 为向向量数据的字段名，不可更改。text 为创建 Collection 时，开启 Embedding 功能自定义的原始文本字段名，请以实际定义为准，二者只能选择其中之一。 • **data**：检索的数据列表，当前仅支持输入一条向量或原始文本。 • **params**：检索参数，与索引类型相关。 ○ **HNSW 类型**：需配置参数 **ef**，指定需要访问向量的数目。取值范围[1,32768]，默认为10。 ○ **IVF 系列**：需设置参数 **nprobe** ，指定所需查询的单位数量。取值范围[1,nlist]，其中 nlist 在创建 Collection 时已设置。 • **limit**：指定稠密向量返回的数量。
		match	否	稀疏向量检索配置。 • **fieldName**：检索的字段名，例如：sparse_vector。 • **data**：检索的稀疏向量，仅支持输入一个。 • **limit**：指定返回的数量。 • **terminateAfter**：设置检索数量上限，即检索数量超过 terminateAfter 值之后，提前终止检索行为，可以有效的控制速率，在数据量较大的情况下，推荐经验值：4000。 • **cutoffFrequency**：设置高频词出现的频次或频率上限，即词频大于 cutoffFrequency 的值之后，该关键词被忽略。 ○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。 ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。 **说明：** cutoff_frequency 可以排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：

			0.1。
	rerank	否	检索中重排序（rerank）的配置。参数 method 指定 Rerank 的方法，枚举如下值： • **weighted**：基于不同字段的加权组合来进行排序。 ○ fieldList：列出用于加权计算的字段列表。例如，`"fieldList": ['vector', 'sparse_vector']` 表示密集型与稀疏型加权计算。 ○ weight：定义了 "fieldList" 中每个字段的权重，例如，`"weight": [0.9, 0.1]` 表示"vector" 字段的权重是 0.9，"sparse_vector" 字段的权重是0.1。 • **rrf**：一种融合多个搜索方法的排名结果，通过为每个文档分配倒数排名分数并合并这些分数来生成一个新的综合排名，从而提高搜索结果的准确性和相关度。k 是一个用于计算倒数排名分数的常数，其作用是调整分数计算公式，以控制排名分数的分布。默认值为60。
	filter	否	设置标量字段的 Filter 表达式。格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的

			字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `"filter" : "bookInfo.bookName in (\"三国演义\",\"西游记\")"` • <value>：表示要匹配的值。
		retrieveVector	否	标识是否需要返回检索结果的向量值。 • **true**：需要。 • **false**：不需要。默认为 false。
		radius	否	指定相似性检索的半径范围。每一种相似性计算方法的检索半径范围如下所示。 • IP：检索结果返回满足 score >= radius 的数据，此时，radius 取值范围为 (−∞, +∞)。 • COSINE：检索结果返回满足 score >= radius 的数据。此时，radius 取值范围为 [−1, 1] • L2：检索结果返回满足 score <= radius 的数据。此时，radius 取值范围为 [0, +∞) **说明：** • 支持的索引类型为：FLAT、IVF 系列。 • 检索结果将限定在指定的检索半径内，并根据您设置的 'limit' 参数值来确定返回结果的数量。
		limit	是	指定返回最相似的 Top K 条数据的 K 的值。K 为大于0的正整数。
		outputFields	否	指定需要输出的字段。若不设置，将返回所有字段。 **说明：** • retrieveVector 和 outputFields 只要有其中一个配置输出向量字段即可输出 vector。 • 输出 Json 字段时，outputFields 仅支持指定 Json 字段的名称，而不支持直接指定

				Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，outputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述

```
{
  "code": 0,
  "msg": "Operation success",
  "documents": [
    [
      {
        "id": "0001",
        "score": 0.97028,
        "author": "罗贯中",
        "bookName": "三国演义"
      },
      {
        "id": "0002",
        "score": 0.9672,
        "author": "吴承恩",
        "bookName": "西游记"
      }
    ]
  ]
}
```

参数名	子参数	参数含义
documents	id	Document 的 ID 信息。
	vector	Document 的向量值。
	score	表示查询向量与检索结果向量之间的相似性计算分数。
	other_scalar_field	Document 其他自定义的标量字段。例如：author、bookName、page 等。

fullTextSearch

最近更新时间：2025-07-01 10:19:42

本接口（ `/document/fullTextSearch` ）基于稀疏向量实现全文检索，对查询词与文档内容进行匹配，并按相关度排序结果。

> **说明：**
> fullTextSearch 接口适用于定义了稀疏向量字段的集合。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/fullTextSearch`

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/fullTextSearch \
  -d '{
    "database": "db-test",
    "collection": "coll-sparse",
    "readConsistency": "eventualConsistency",
    "search": {
      "match": {
                "fieldName": "sparse_vector",
                "data": [[[2, 0.9611]]],
                "terminateAfter": 40,
                "cutoffFrequency": 5
        },
        "limit": 3,
        "retrieveVector": false,
        "outputFields": [
            "id",
            "text"
        ]
    }
}'
```

入参描述

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要查询的Database名称。	-	是	获取集群中的数据库列表，复制需查询数据的集合所属的数据库名。
collection	指定要查询的Collection名称。	-	是	获取集群中的集合列表，复制需查询数据的集合名。
readConsistency	设置检索数据的一致性要求。	-	否	取值如下所示： • **strongConsistency**：强一致性。 • **eventualConsistency**：最终一致性。

search	配置检索参数	match	否	稀疏向量检索配置。
				• **fieldName**：检索的字段名，例如：sparse_vector。
				• **data**：检索的稀疏向量，仅支持输入一个。
				• **limit**：指定返回的数量。
				• **terminateAfter**：设置检索数量上限，即检索数量超过 terminateAfter 值之后，提前终止检索行为，可以有效的控制速率，在数据量较大的情况下，推荐经验值：4000。
				• **cutoffFrequency**：设置高频词出现的频次或频率上限，即词频大于 cutoffFrequency 的值之后，该关键词被忽略。
				  ○ 频数：可指定关键词出现的频数，取值为正整数，范围为[1,+∞]。
				  ○ 频率：可指定为关键词出现的频率，取值范围为[0,1]的小数。例如：0.8。
				说明：

			cutoff_frequency 可以排序阶段去掉高频词，在数据量较大的情况下（>=50w）推荐值：0.1。
		filter	否	设置标量字段的 Filter 表达式。格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 表达式 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `"filter" : "bookInfo.bookName in (\"三国演义\",\"西游记\")"` • <value>：表示要匹配的值。
		retrieveVector	否	标识是否需要返回检索结果的向量值。 • **true**：需要。 • **false**：不需要。默认为 false。
		limit	是	指定返回最相似的 Top K 条数据的 K 的值。K 为大于0

				的正整数。
		outputFields	否	指定需要输出的字段。若不设置，将返回所有字段。 **说明：** • retrieveVector 和 outputFields 只要有其中一个配置输出向量字段即可输出 vector。 • 输出 Json 字段时，outputFields 仅支持指定 Json 字段的名称，而不支持直接指定 Json 字段内部的键（key）。例如，写入 `"a": {"b": "test", "c": 12}`，outputFields 只能指定返回整个 "a" 字段，而无法单独指定返回 "a.b" 。

出参描述


```
{
  "code": 0,
  "msg": "Operation success",
  "documents": [
    [
      {
        "id": "0003",
        "sparse_vector": [
          [2, 0.9700000286102295],
          [5, 0.5400000214576721],
          [100, 0.4449999928474426]
        ],
        "score": 0.93227
      },
      {
        "id": "0002",
        "sparse_vector": [
          [2, 0.9700000286102295],
          [5, 0.5400000214576721],
          [100, 0.4449999928474426]
        ],
        "score": 0.93227
      },
      {
```

```
        "id": "0001",
        "sparse_vector": [
          [2, 0.9599999785423279],
          [5, 0.5299999713897705],
          [100, 0.4429999887943268]
        ],
        "score": 0.92266
      }
    ]
  ]
}
```

参数名	子参数	参数含义
documents	id	Document 的 ID 信息。
	sparse_vector	Document 的稀疏向量。
	score	表示查询向量与检索结果向量之间的相似性计算分数。
	other_scalar_field	Document 其他自定义的标量字段。例如：author、bookName、page 等。

delete

最近更新时间：2025-07-01 10:19:43

本接口（ `/document/delete` ）用于删除指定 id（Document ID）的文档。

> **说明：**
> 索引类型为 FLAT，在1.6及之后版本，支持删除操作，而1.6之前版本，则不支持删除操作。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/delete`

使用示例

如下示例，删除 **book-emb** 集合中， id（Document ID）为0001、0002、0003，且满足 `bookName in (\"三国演义\",\"西游记\")` 的数据。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/delete \
  -d '{
    "database": "db-test",
    "collection": "book-emb",
    "query": {
        "documentIds": [
            "0001",
            "0002",
            "0003"
        ],
        "filter": "bookName in (\"三国演义\",\"西游记\")"
        "limit": 2
    }
}'
```

请求参数

参数名称	参数含义	子参数	是否	配置方法及要求

			必选	
database	指定需删除文档的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需删除数据的集合所属的数据库名。
collection	指定需删除文档的 Collection 名称。	-	是	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需删除数据的集合名。
query	设置查询条件，满足查询条件的文档将被删除。	documentIds	否	指定需删除的 Document 的 ID，数据元素最大值为 20。 **说明：** 至少配置 documentIds 或 filter 参数中的一个，同时配置 documentIds 与 filter 参数，删除数据将会取二者交集。
		filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中

				的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo ，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `"filter" : "bookInfo.bookName in (\"三国演义\",\"西游记\")"` • <value>：表示要匹配的值。
		limit	否	控制删除文档的数量。若不设置该参数，最大删除16384条数据。 • 数据类型：整型。 • 取值范围：[1,16384]

响应消息

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 2
}
```

参数名	参数含义
affectedCount	删除文档数量。

update

最近更新时间：2025-07-01 10:19:43

本接口（ `/document/update` ）用于更新文档。

> **说明：**
> - 若在创建 Collection 时未显式定义某个字段的索引方式，则后续通过 Update 操作新增该字段时，系统不会自动为其创建索引。
> - 不能变更 Document ID 字段，不要求事务完整性。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/update`

请求示例

更新向量数据

集合未配置 Embedding 参数，则直接更新向量数据。如下示例，在集合 **book-vector** 中，基于 /document/upsert 插入的向量数据，通过 documentIds 与 filter 表达式，过滤出 Document ID 为0001的数据，更新其 vectors 字段的向量数据，并更新 page 字段值为30，新增字段 test_new_field。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/update\
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "query": {
        "documentIds": [
            "0001",
            "0003"
        ],
        "filter": "bookName in (\"三国演义\",\"西游记\")"
    },
    "update": {
        "vector": [
            0.2123,
```

```
            0.28,
            0.213
        ],
        "page": 30,
        "test_new_field": "new field value"
    }
}'
```

执行成功之后，返回如下信息。

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1
}
```

通过 /document/query 查询 Document ID 为0001的数据，请求消息如下所示，确认更新的字段是否生效。返回如下信息，可看到 vectors 字段与 page 字段值已更新，新增字段 test_new_field 也已生效。

```
{
    "code": 0,
    "msg": "operation success",
    "count": 1,
    "documents": [
        {
            "id": "0001",
            "vector": [
                0.21230000257492066,
                0.2800000011920929,
                0.21299999952316285
            ],
            "bookName": "三国演义",
            "author": "罗贯中",
            "test_new_field": "new field value",
            "page": 30
        }
    ]
}
```

写入文本更新向量数据

实例在创建 Collection 时，已配置 Embedding 模型，通过 /document/upsert 写入原始文本，则可输入文本信息，通过 Embedding 将数据向量化更新向量数据。如下示例，基于 /document/upsert 插入的原始文本，使用 **/document/update** 接口，通过 documentIds 与 filter 表达式过滤 Document ID 为0001的数据，更新其 text 字段的文本信息，更新 page 字段值为30，并新增字段 test_new_field。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/update \
  -d '{
    "database": "db-test",
    "collection": "book-emb",
    "query": {
        "documentIds": [
            "0001",
            "0003"
        ],
        "filter": "bookName in (\"三国演义\",\"西游记\")"
    },
    "update": {
        "text": "合久必分，分久必合",
        "page": 30,
        "test_new_field": "new field value"
    }
}'
```

执行成功之后，返回如下信息。

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1,
    "embeddingExtralnfo": {"tokenUsed":9}
}
```

通过 /document/query 查询 Document ID 为0001的数据，确认更新的字段是否生效。返回如下信息，可看到 text 字段与 page 字段值已更新，新增字段 test_new_field 也已生效。

```
{
    "code": 0,
    "msg": "operation success",
    "count": 1,
    "documents": [
        {
            "id": "0001",
            "author": "罗贯中",
            "test_new_field": "new field value",
            "bookName": "三国演义",
            "page": 30,
            "text": "合久必分，分久必合"
        }
    ]
}
```


请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要更文档的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需更新数据的集合所属的数据库名。
collection	指定要更新文档的 Collection 名称。	-	是	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需更新数据的集合名。
query	设置查询条件检索需更新的文档	documentIds	是	表示要更新的文档的所有 ID，支持批量查询，数组元素范围 [1,20]。
		filter	否	使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 的表达式格式为 '<field_name><operator><value>'，多个表达式之间支持 and

				（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式。 `"filter" : "bookInfo.bookName in (\"三国演义\",\"西游记\")"` • <value>：表示要匹配的值。
update	设置需更新的字段	**vector**	否	更新 vector 字段的向量数据。 **说明：** 如果 Collection 在创建时，未配置 Embedding 参数，或者该实例并未开通 Embedding 功能，则只能配置该参数，输入向量数据，更新数据。
		text	否	Embedding 模型输入文本的字段名。该字段在创建 Collection 时定义。本示例为 text 。 • **string**：字符型。 • **float**：浮点型数据。 **说明：**

				如果 Collection 在创建时，已配置 Embedding 参数，则只能配置该参数，依据输入的文本信息更新向量数据，并将文本字段与向量数据更新存于数据库。
		old_field	否	当前已存在的字段，更新字段对应的数据。 • 类型：string。 • 字符长度要求：[1,256]。
		new_field	否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。

响应消息

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 1
}
```

参数名	参数含义
affectedCount	更新的文档数量。如果该参数返回的值为0，说明更新无效。

count

最近更新时间：2025-07-01 10:19:43

本接口（ `/document/count` ）用于查询集合中文档的数量。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/document/count` 。

请求示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/document/count \
  -d '{
    "database": "db-test",
    "collection": "book-emb",
    "query": {
        "filter": "bookName in (\"三国演义\",\"西游记\")"
    }
}'
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要更新文档的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需更新数据的集合所属的数据库名。
collection	指定要更新文档的 Collection 名称。	-	是	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需更新数据的集合名。

query	设置查询条件检索需更新的文档	filter	是	表示要更新的文档的所有 ID，支持批量查询，数组元素范围[1,20]。使用创建 Collection 指定的 Filter 索引的字段设置查询过滤表达式。Filter 的表达式格式为'<field_name><operator><value>'，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 Filter 条件表达式 。其中 • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）、不等于（!=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 ○ **json**：json 类型的 Filter 表达式语法和 json 字段的键值类型保持一致。若访问 Json 对象中的键，使用点（.）符号连接。例如：Json 类型的字段 bookInfo，其键 bookName 的 Filter 表达式如下所示。更多信息，请参见 Json 类型表达式 。 `"filter" : "bookInfo.bookName in (\"三国演义\",\"西游记\")"` • <value>：表示要匹配的值。

响应消息

```
{
  "code": 0,
  "msg": "Operation success",
  "count": 2
}
```

参数名	参数含义

count	集合中的文档数量。

uploadUrl

最近更新时间：2025-05-26 19:31:32

该接口用于调用 AI 套件服务，获取上传文件所需的凭证。文件上传后，其内容将被直接拆分并写入指定的 Collection 中，而非 CollectionView。

> **说明：**
> 当前支持导入 Collection 的文件类型包含： Markdown、PDF、Word、PPT。其中，仅 PDF 类型支持解析图片信息。

Method 与 URL

获取 COS 文件上传路径和授权签名凭证：
`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/document/uploadUrl` 。

> **说明：**
> - 获取授权签名后，您需使用 COS（对象存储）SDK 或 Restful API 将文档上传至 COS 服务器。文件上传完成状态通知之后，向量数据库便会开始拉取文件并解析文件内容，进行拆分、向量化，创建索引并进行存储。
> - HTTP 上传文件的方式需要借助 COS 上传文件的接口，操作较繁琐，推荐使用向量数据库 Python SDK 的方式直接上传文件。

约束限制

- 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
- 该接口当前不支持使用别名获取文件的上传路径与授权。

> **注意：**
> 在创建存储文件内容的 Collection 时，索引字段的设计需遵循以下限制与建议，否则可能导致上传文件失败。
> - 文件名字段（file_name）：该字段**必须**定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。字段名可以自定义，在上传文件时，需通过参数 field_mappings 的 filename 映射自定义的字段名。
> - 文件内容字段（text）：该字段用于存储知识点的原始文本内容。由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。字段名可以自定义，但需通过参数 field_mappings 的 text 映射自定义的字段名。

版权所有：腾讯云计算（北京）有限责任公司

- 文件图片信息字段（image_list）：该字段存放 PDF 文件中图片的 Key 列表。在原始检索出的文本块中，图片位置将以 {key} 的形式进行占位。通过调用接口 getImageUrl，可以获取图片 URL 地址列表，这些地址与 Key 一一对应，从而实现将原始 PDF 文档恢复并展示为 HTML 格式。
  - 不建议为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。
  - 字段名可以自定义，但需要在上传文件时通过参数 field_mappings 的 imageList 映射到自定义的字段名。

请求参数

说明：
如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/document/uploadUrl \
  -d '{
    "database":"db-test",
    "collection":"coll-ai-files",
    "fileName": "腾讯云向量数据库.pdf",
    "splitterPreprocess": {
        "appendTitleToChunk": false,
        "appendKeywordsToChunk": true,
        "chunkSplitter": "\n{2,}"
    },
    "parsingProcess": {
        "parsingType": "VisionModelParsing"
    },
    "embeddingModel": "bge-base-zh",
    "byteLength": 1024,
    "fieldMappings": {
        "filename": "file_name",
        "text": "text",
        "imageList": "image_list",
        "chunkNum": "chunk_num",
        "sectionNum": "section_num"
    }
  }'
```

参数	参数含义	子参数	是否必选	配置方法
database	指定 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制数据库名。
collection	指定 Collection 的名称。	-	是	Collection 命名要求如下： • 只能使用英文字母，数字，下划线 _、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
fileName	上传文件名称。	-	是	-
splitterPreprocess	文件预处理方式配置	appendTitleToChunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。默认值为 false。 • **true**：将段落 Title 追加到切分后的段落。
		appendKeywordsToChunk	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 keywords 追加到切分后的段落。默认值为 true。
		chunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsingProcess	PDF 文件解析方式相关参数	parsingType	否	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

embeddingModel	指定使用的 Embedding 模型的名称。	–	是	您需根据业务的语言类型、数据维度要求等综合选择合适的模型。具体信息，参见 Embedding 介绍 。取值如下所示： • **bge-large-zh-v1.5**：适用中文，1024维，推荐使用。 • **bge-base-zh-v1.5**：适用中文，768维。 • **bge-large-zh**：适用中文，1024维。 • **bge-base-zh**：适用中文，768维。 • **m3e-base**：适用中文，768维。 • **e5-large-v2**：适用英文，1024维。 • **text2vec-large-chinese**：适用中文，1024维。 • **multilingual-e5-base**：适用于多种语言类型，768维。 • **BAAI/bge-m3**：适用于多种语言类型，1024维。
byteLength	指定上传文件的大小。	–	否	指定文件大小，以便在上传文件之前校验文件大小，避免超限上传。
fieldMappings	指定文件字段在数据库中映射关系。	filename	是	指定文件名映射字段。例如："filename": "file_name"，其中，file_name 为用户自定义的字段，以在数据库中存储对应的文件名数据。 **注意：** 必须定义为 string 类型的 filter 索引，以支持文件的过滤检索和处理同名文件覆盖的情况。
		text	是	指定文件内容文本映射的字段。例如："text": "text"，其中，text 为用户自定义字段，以在数据库中存储文件的原始文本内容。 **说明：**

				由于文本内容可能较大，**不建议**为该字段创建索引，以避免占用过多内存空间。数据库会自动写入该字段的内容。在查询时，只需要通过 output_fields 参数指定该字段即可返回原始文本。
		imageList	是	指定图片信息存放的字段。例如："imageList": "image_list"，其中，image_list 为用户自定义的字段，以在数据库中存储图片 Key 列表。 **说明：** **不建议**为该字段创建索引。如果需要创建索引，必须确保其为数组（array）类型，否则接口将报错，导致文件无法上传。
		chunkNum	否	标记文件切分语块序号的字段。例如："chunkNum": "chunk_num"，其中，chunk_num 为用户自定义的字段，其值为唯一标识语块片段的整数。 **说明：** 在创建 Collection 时若为该字段（如 chunk_num）创建索引，则查询文件内容时可通过 Filter 表达式指定语块范围，快速定位目标文件内容。
		sectionNum	否	标记文件段落序号的字段。例如："sectionNum": "section_num"，其中，section_num 为用户自定义字段，其值为唯一标识段落的整数。 **说明：** 在创建 Collection 时若为该字段（如 section_num）创建

				索引，则查询文件内容时可通过 Filter 表达式指定段落范围内的语块，快速定位目标文件内容。

响应信息

返回信息，如下所示。


```
{
  "code": 0,
  "msg": "Operation success",
  "requestId": "4b1bd009b5ee241839b18210070fdc1b",
  "cosEndpoint": "https://vecterdb-backup-gz-131891****.cos.ap-
guangzhou.myqcloud.com",
  "cosRegion": "ap-guangzhou",
  "cosBucket": "vecterdb-backup-gz-131891****",
  "uploadPath": "embedding_file/vdb-jggo****/db-test/book-vector-1/腾讯云
向量数据库.pdf",
  "credentials": {
    "TmpSecretId":
"AKID************************************************************",
    "TmpSecretKey": "****************************************",
    "Token":
"************************************************************************
********************",
    "Expiration": "2025-02-17T07:46:57Z",
    "ExpiredTime": 1739778417
  },
  "uploadCondition": {
    "maxSupportContentLength": 10485760
  }
}
```


接口 **/ai/document/uploadurl** 获取授权签名凭证之后，需使用 COS 接口上传文件，其与接口对应的参数解释如下所示。具体应用，请参见 使用 COS 接口上传文件 。

VDB 参数名	参数含义	子参数	子参数含义	对应 COS SDK 参数名	COS 子参数
cosEndpoint	腾讯云对象存储（COS）的服务端点	-	-	`cos_endpoint`	-

	（Endpoint），即 COS 服务的访问地址。				
cosBucket	COS 服务端存储桶名称。	–	–	–	–
cosRegion	COS 服务端存储桶所属地域。	–	–	`cosRegion`	–
uploadPath	依据数据库名、集合名、文件名拼接生成的 COS 端存放路径。	–	–	`upload_path`	–
credentials	COS 给该文件上传分配的临时身份凭证，以便在上传文件时进行身份验证和授权访问。	TmpSecretId	密钥 ID	`credentials`	`SecretId`
		TmpSecretKey	密钥信息		`SecretKey`
		Token	Token 信息		`Token`
uploadCondition	上传约束条件。	maxSupportContentLength	限制上传文件的最大字节数	`maxSupportContentLength`	–

使用 COS Python SDK 上传文件示例

如下示例为 COS Python SDK 上传文件的参考代码。您需要根据代码注释替换相关授权凭证，以及如下表所列参数，便可以通过 upload_file() 接口上传文件。更多信息，请参见 COS Python SDK 。

参数	是否必须	参数含义
local_file_path	是	本地上传路径。
cos_metadata	是	**文件元数据库配置，配置方法，请参见如下代码示例注释。**文件的 Metadata 元数据信息，可自定义扩展字段。例如：author、tags 等。

		• 上传文件时，可为创建 CollectionView 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。
appendTitleToChunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • 0：不追加。默认值为0。 • 1：将段落 title 追加到切分后的段落。
appendKeywordsToChunk	否	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • 0：不追加。 • 1：将全文的 **keywords** 追加到切分后的段落。默认值为1。
chunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsingProcess	否	通过参数 parsingType 指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

```
import requests

import json
import os
import sys
import urllib
import base64
from qcloud_cos import CosConfig, CosS3Client

local_file_path = "<your_local_file_path>"

# 1. json解析 /ai/documentSet/uploadUrl 获得的签名结果：
result = json.loads("<results from /ai/documentSet/uploadUrl>")
```

```
# 2. 文件Size校验(支持1MB Markdown文件)
limit_size = result["uploadCondition"]["maxSupportContentLength"]
file_stat = os.stat(local_file_path)
if limit_size < file_stat.st_size:
    print("The file exceeds its maximum permitted size")
    sys.exit(0)

# 3. 设置COS上传签名
credentials = result["credentials"]
print(credentials)

config = CosConfig(
    Region=result["cosRegion"],
    SecretId=credentials["TmpSecretId"],
    SecretKey=credentials["TmpSecretKey"],
    Token=credentials["Token"]
)

# 4. 您可以自定义文件 Metadata 信息
# 说明1：x-cos-meta-data，自定义字段与赋值；{类型}支持string、uint64、array；
{字段名}为元数据字段名，例如：author，page，city等。
metadata = {"fieldStr": "v1", "fieldInt": 1024, "fieldList": ["a", "b",
"c"]}
# notice：必填字段，没有传：metadata = {}
# 说明2：x-cos-meta-config，解析与embedding增强参数
config = {"appendTitleToChunk": 1, "appendKeywordsToChunk": 1,
"parsingProcess": {"parsingType": "VisionModelParsing"}} #上传文件时单独配
置预处理逻辑

cos_metadata = {
    'x-cos-meta-config':
urllib.parse.quote(base64.b64encode(json.dumps(config).encode('utf-
8'))),
    'x-cos-meta-data':
urllib.parse.quote(base64.b64encode(json.dumps(metadata).encode('utf-
8')))
}

# 5. 上传文件:
CosS3Client(config).upload_file(
    Bucket=result.get('cosBucket'),
    Key=result.get('uploadPath'),
    LocalFilePath=local_file_path,  # 本地文件路径
```

```
    Metadata=cos_metadata
)
```

getImageUrl

最近更新时间：2025-05-15 18:10:42

该接口用于在使用 AI 套件上传 PDF 类型的文件后，获取图片 URL 地址列表。

> **说明：**
> 当在数据库的 Collection 中上传 PDF 文件时，系统会提取 PDF 中的图片信息，并将这些信息以 Key 列表的形式存储。在检索出的原始文本块中，图片的位置会以 {key} 的形式进行占位。通过调用接口 getImageUrl，可以获取到与这些 Key 一一对应的图片 URL 地址列表。利用这种映射关系，可以将原始 PDF 文档的内容恢复并以 HTML 格式进行展示。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/document/getImageUrl` 。

请求参数

> **说明：**
> 如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/document/getImageUrl \
  -d '{
    "database":"db-test",
    "collection":"coll-ai-files",
    "fileName": "腾讯云向量数据库.pdf",
    "documentIds": ["1340988975862403135"]
  }'
```

参数	参数含义	是否必选	配置方法
database	指定 Database 名称。	是	使用 /database/list 获取集群中的数据库列表，复制数据库名。
collection	指定 Collection 的名称。	是	Collection 命名要求如下：

			• 只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。 • 长度要求：[1,128]。
fileName	指定查询的文件名称。	是	-
documentIds	表示要查询的文档的所有 ID。	是	• 每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。 • 文件上传成功之后，可使用 query 查询文件的 ID 信息。

响应消息


```
{
  "code": 0,
  "msg": "Operation success",
  "requestId": "24ce1fbeb1f8241842b182108096d669",
  "images": [
  [
    {
      "documentId": "1340988975862403135",
      "imageName": "ea97227d-1b12-4d35-8b67-************.png",
      "imageUrl": "https://vecterdb************.cos.ap-
guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-
test-6/****************************************"
    },
    {
      "documentId": "1340988975862403135",
      "imageName": "a5daca6f-a27e-4b35-89f5-************.png",
      "imageUrl": "https://vecterdb************.cos.ap-
guangzhou.myqcloud.com/embedding_file/vdb-********/db-test/coll-file-
test-6/****************************************"
    }
  ]
 ]
}
```


参数名	参数含义
documentId	Document 的 ID 信息。

imageName	图片名。
imageUrl	图片 URL。

版权所有：腾讯云计算（北京）有限责任公司

DocumentSet uploadUrl

最近更新时间：2025-05-26 19:31:32

Method 与 URL

获取文件上传路径和授权签名：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/documentSet/uploadUrl` 。

> **说明：**
> - 获取授权签名后，您需使用 COS（对象存储）SDK 或 Restful API 将文档上传至 COS 服务器。文件上传完成的状态通知之后，向量数据库便会开始拉取文件并解析文件内容，进行拆分、向量化，创建索引并进行存储。
> - HTTP 上传文件的方式需要借助 COS 上传文件的接口，操作较繁琐，推荐使用向量数据库 Python SDK 的方式直接上传文件。

约束限制

- 每次仅能上传一个文件，上传之后，将自动进行拆分、向量化等。
- 该接口当前不支持使用别名获取文件的上传路径与授权。

请求参数

> **注意：**
> 如下示例 api_key 的值 A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X，需要分别替换为具体实例的 API Key 以及访问地址。

> **说明：**
> 如果需要进行腾讯云向量数据库的测试，请获取 腾讯云向量数据库.pdf 文档。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/ai/documentSet/uploadUrl \
  -d '{
    "database":"db-test-ai",
```

```
    "collectionView":"coll-ai-files",
    "documentSetName":"腾讯云向量数据库.pdf",
    "byteLength": 1024
  }'
```

参数名	是否必选	参数含义
database	是	文件预上传的目标数据库名。
collectionView	是	文件预上传的目标数据库的集合视图名。 **说明：** 该参数目前不支持配置为集合视图的别名，即该接口当前不支持使用别名获取文件的上传路径与授权。
documentSetName	是	文件名称。
byteLength	否	指定文件大小，以便在上传文件之前校验文件大小，避免超限上传。

响应信息

```
{
  "code": 0,
  "msg": "operation success, requestId:
eea021e910f89417****************",
  "cosEndpoint": "https://gz-vector-test-cos-1257943044.cos.ap-
guangzhou.myqcloud.com",
  "cosRegion":"ap-guangzhou",
  "cosBucket":"gz-vector-test-cos-1257943044",
  "uploadPath": "embedding_file/vdb-jcrlbp46/db-test-ai/coll-ai-files/腾
讯云向量数据库.md",
  "credentials": {
    "TmpSecretId":
"*****************************************************************",
    "TmpSecretKey": "pnIGo********+Jete***************************",
    "Token":
"4EGei*****************************************************************
*****************************************************************
****************"
  },
```

```
  "uploadCondition": {
    "maxSupportContentLength":1048576
  },
  "documentSetId": "MTE2ODgzNDAxNDU0MzYxMzk1Mg=="
}
```

接口 /ai/document/uploadurl 获取授权签名凭证之后，需使用 COS 接口上传文件，其与接口对应的参数解释如下所示。具体应用，请参见 使用 COS 接口上传文件 。

VDB 参数名	参数含义	子参数	子参数含义	对应 COS SDK 参数名	COS 子参数
cosEndpoint	腾讯云对象存储（COS）的服务端点（Endpoint），即 COS 服务的访问地址。	-	-	cos_endpoint	-
cosBucket	COS 服务端存储桶名称。	-	-	-	-
cosRegion	COS 服务端存储桶所属地域。	-	-	cosRegion	-
uploadPath	依据数据库名、集合名、文件名拼接生成的 COS 端存放路径。	-	-	upload_path	-
credentials	COS 给该文件上传分配的临时身份凭证，以便在上传文件时进行身份验证和授权访问。	TmpSecretId	密钥 ID	credentials	SecretId
		TmpSecretKey	密钥信息		SecretKey
		Token	Token 信息		Token

uploadCondition	上传约束条件。	maxSupportContentLength	限制上传文件的最大字节数	`maxSupportContentLength`	-
documentSetId	COS 给文件分配的 ID 信息。	-		`fileId`	-

使用 COS Python SDK 上传文件示例

如下示例为 COS Python SDK 上传文件的参考代码。您需要根据代码注释替换相关授权凭证，以及如下表所列参数，便可以通过 upload_file() 接口上传文件。更多信息，请参见 COS Python SDK 。

参数	是否必须	参数含义
local_file_path	是	本地上传路径
cos_metadata	是	文件元数据库配置，配置方法，请参见如下代码示例注释。文件的 Metadata 元数据信息，可自定义扩展字段。例如：author、tags 等。 • 上传文件时，可为创建 CollectionView 设置的 Filter 索引的字段赋值，以便在检索时，使用该字段的 Filter 表达式检索文件。 • 上传文件时，可以新增标量字段，但新增字段不会构建 Filter 索引。
appendTitleToChunk	否	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • 0：不追加。默认值为 0。 • 1：将段落 title 追加到切分后的段落。
appendKeywordsToChunk	否	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • 0：不追加。 • 1：将全文的 keywords 追加到切分后的段落。默认值为 1。
chunkSplitter	否	以正则表达式的方式配置文档拆分方式，例如：`\n{2,}` 代表以两个及以上的换行进行拆分，常用在 QA 对文件拆分中。
parsingProcess	否	通过参数 parsingType 指定 PDF 类型文件的解析方式，取值如下所示：

		• **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。

```
import requests

import json
import os
import sys
import urllib
import base64
from qcloud_cos import CosConfig, CosS3Client

local_file_path = "<your_local_file_path>"

# 1. json解析 /ai/documentSet/uploadUrl 获得的签名结果：
result = json.loads("<results from /ai/documentSet/uploadUrl>")

# 2. 文件Size校验（支持1MB Markdown文件）
limit_size = result["uploadCondition"]["maxSupportContentLength"]
file_stat = os.stat(local_file_path)
if limit_size < file_stat.st_size:
    print("The file exceeds its maximum permitted size")
    sys.exit(0)

# 3. 设置COS上传签名
credentials = result["credentials"]
print(credentials)

config = CosConfig(
    Region=result["cosRegion"],
    SecretId=credentials["TmpSecretId"],
    SecretKey=credentials["TmpSecretKey"],
    Token=credentials["Token"]
)

# 4. 您可以自定义文件 Metadata 信息
# 说明1：x-cos-meta-data，自定义属性及赋值，必填字段；{类型}支持string、uint64、
array；{字段名}为元数据字段名，例如：author，page，city等。
metadata = {"fieldStr": "v1", "fieldInt": 1024, "fieldList": ["a", "b",
"c"]}
```

```
# notice：必填字段，没有传：metadata = {}
# 说明2：chunk embedding增强参数
config = {"appendTitleToChunk": 1, "appendKeywordsToChunk": 1,
"parsingProcess": {"parsingType": "VisionModelParsing"}} #上传文件时单独配
置预处理逻辑

cos_metadata = {
    'x-cos-meta-id': result.get('documentSetId'),  # 文件唯一标识符，必填字
段
    'x-cos-meta-config':
urllib.parse.quote(base64.b64encode(json.dumps(config).encode('utf-
8'))),
    'x-cos-meta-data':
urllib.parse.quote(base64.b64encode(json.dumps(metadata).encode('utf-
8')))
}

# 5. 上传文件：
CosS3Client(config).upload_file(
    Bucket=result.get('cosBucket'),
    Key=result.get('uploadPath'),
    LocalFilePath=local_file_path,  # 本地文件路径
    Metadata=cos_metadata
)
```

get

最近更新时间：2025-05-12 18:23:12

Method 与 URL

获取文件数据： `POST https://{实例访问 IP 地址}:{实例网络端口}/ai/documentSet/get` 。

使用示例

> **注意：**
> 使用 get 获取单个文档的 chunk 数量上限为1000，若超过此限制，接口将返回错误提示信息，并提示可使用 documentSet/getChunks 接口，结合 offset（起始位置）和 limit（每页数量）参数分批获取文档内容。

使用文件名查询文件内容

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/get \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "documentSetName":"腾讯云向量数据库.pdf"
  }'
```

使用文件 ID 查询文件内容

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
   http://10.0.X.X:80/ai/documentSet/get \
  -d '{
```

```
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "documentSetId": "11790179945********"
  }'
```

请求参数

参数名	是否必选	参数含义	获取方式
database	是	文件所存储的数据库名	使用 /database/list 获取集群中的数据库列表，复制数据库名。
collectionView	是	文件所存储的集合名	使用 /ai/collectionView/list 获取指定数据库名下的 Collection 列表，复制集合名。
documentSetId	否	文件上传在数据库之后，系统分配的文件 ID。	使用文件名查找文件内容，可获取文件 ID。
documentSetName	否	文件名	-

响应消息


```
{
  "code": 0,
  "msg": "Operation success",
  "requestId": "d348ae93653ba5179bbec46d94b8e49d",
  "count": 1,
  "documentSet": {
    "documentSetId": "11901307631454126 08",
    "documentSetName": "腾讯云向量数据库.pdf",
    "text": "本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。\n## 关键概念\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。\n### 什么是向量？\n向量是指在数学和物理中用
```

来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。\n### 什么是非结构化数据？\n非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。\n### 什么是 AI 中的向量表示？\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。\n### 什么是向量检索？\n向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。\n## 为什么是腾讯云向量数据库？\n腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户，在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。\n## 腾讯云向量数据库应用示例有哪些？\n腾讯云向量数据库可进行高性能向量存储和检索，主要适用于以下应用场景。\n- [大规模知识库]：企业的私域数据存储在向量数据库中可构建外部知识库，帮助企业更好地管理和利用自己的数据资源。\n- [推荐系统]：向量数据库会基于用户特征进行向量存储与检索，最终筛选用户可能感兴趣的物品推荐给用户。\n- [问答系统]：向量数据库会基于问题信息进行向量存储与检索，并返回最相关的问题与对应的答案。\n- [文本/图像检索]向量数据库对输入的图像和文本信息进行向量存储与检索，会找到最匹配输入信息的文本或图像结果。\n## 腾讯云向量数据库支持哪些索引类型？\n索引是数据的组织单位。您必须先声明索引类型和相似性度量，然后才能搜索或查询向量数据。目前，腾讯云向量数据库支持如下类型。具体信息，请参见 [Index]\n- FLAT 索引：向量会以浮点型的方式进行存储，不做任何压缩处理。搜索向量会遍历所有向量与目标向量进行比较。\n- HNSW 索引：全称为 Hierarchical Navigable Small World， 是基于图的索引，适合对搜索效率要求较高的场景。\n- IVF 系列：全称为 Inverted File，IVF 系列索引的核心思想是将高维空间划分为多个聚类，并为每个聚类构建一个倒排文件。适用于高维向量数据的快速检索。\n## 腾讯云向量数据库支持哪些相似度计算方法？\n在 VectorDB 中，相似度度量用于衡量向量之间的相似度。选择良好的距离度量有助于显著提高分类和聚类性能。根据输入数据形式，选择特定的相似性度量方法，获得数据库最佳性能。\n**相似性计算方法** | **方法说明** \n:-: | :-: \n内积（IP） | 全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 | \n欧式距离（L2） | 全称为 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 | \n余弦相似度（COSINE） | 余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度。所得值越大越与搜索值相似。 | \n## 腾讯云向量数据库是如何设计的？\n- **部署架构**：腾讯云向量数据库采用分布式部署架构，每个节点相互通信和协调，实现数据存储与检索。客户端请求通过 **Load Balancer** 分发到各节点上。\n- **逻辑架构**：实例是腾讯云中独立运行的数据库环境，是用户购买向量数据库服务的基本单位。腾讯云向量数据库数据存储的一个实例集群中包括 [Database]、[Collection]、[Document] 三个逻辑层级。其中，一个实例可以包含很多个 Database，一个 Database 可以包含多个 Collection，一个 Collection 可以包含多个 Document。\n- **数据安全**：腾讯云向量数据库的多副本设计、多可用区分布节点、API 密钥认证，并运行于私有网络环境，通过安全组控制访问来源，CAM 账户授权等多方面保护向量数据的完整性和隐私。\n- **鉴权方式**：腾讯云向量数据库使用账号（account）和 API 密钥（api_key）的组合进行鉴权，以验证用户身份并授权其访问。\n- **连接方式**：腾讯云向量数据库支持通过 HTTP 协议进行数据写入和查询等操作。\n- **检索方法**：腾讯云向量数据库

支持通过精确检索、相似度检索、混合检索的方法。\n- 精确查询：基于标量（指一个单独的数值，例如文本字段、数值字段或日期字段，区别于向量等多维数据结构）字段精确查找数据的方式。\n- 相似度检索：基于向量相似度计算的检索方式，通过计算向量之间的相似度来找到与查询向量最相似的文档。\n- 混合检索：基于标量字段和向量字段，搭配自定义的标量字段的 Filter 表达式进行检索的方式。\n## 如何快速体验向量数据库？\n腾讯云向量数据库目前是公测阶段。免费测试版实例每个账号仅限申领1个，高可用版与单机版实例免费试用时长1个月，到期后可 [提交工单](https://console.cloud.tencent.com/workorder/category) 进行续期；若一个月内未使用实例，平台将自动回收。\n**序号** | **步骤描述** | **具体操作** \n:-: | :-: | :-: \n1 | 申请腾讯云账号并认证。 | - 如需注册腾讯云账号：请单击 [注册腾讯云账号](https://cloud.tencent.com/register?s_url=https%3A%2F%2Fcloud.tencent.com%2F)。 | - 如需完成实名认证：请单击 [实名认证](https://console.cloud.tencent.com/developer)。 | \n2 | 了解向量数据库所支持的规格与类型。 | 预估数据规模，选择合适的类型与规格。 | \n3 | 确定向量数据库所部署的地域。 | 当前支持的地域信息，请参见 [发布地域]| \n4 | 规划数据库实例的私有网络与安全组。 | 具体操作，请参见 [创建私有网络](https://cloud.tencent.com/document/product/215/36515)与 [创建安全组]，并同时设置安全组入站规则。 | \n5 | 购买实例。 | 具体操作，请参见 [新建数据库实例]。购买实例中，直接选择上一步已准备的私有网络与安全组。 | \n6 | 申请与腾讯云向量数据库在同一地域同一个 VPC 内的 Linux 云服务器 CVM。 | 具体操作，请参见 [快速配置 Linux 云服务器]。 | \n7 | 连接并操作向量数据库。 | [连接并写入数据库] ，本文使用 [API 接口] 从创建 DataBase 到 插入数据、检索数据到最终删除数据，均给出了具体的使用示例。您可以简单并快速体验向量数据库。 | \n8 | 管理向量数据库实例 | 您可以体验通过控制台直接管理实例，查看实例状态或销毁实例。 | \n9 | 智能运维 | 您可以在控制台查看监控数据库实例的各项指标。目前仅支持对节点信息的监控，后续还会支持更丰富的监控项目。 | \n## 开发者工具 \n**开发者工具** | **API** \n:-: | :-: \nHTTP API | [API 接口](https://cloud.tencent.com/document/product/1709/98666) | \nPython SDK | [Python SDK Demo](https://cloud.tencent.com/document/product/1709/96724) | \nJava SDK | [Java SDK Demo](https://cloud.tencent.com/document/product/1709/97768) | \n",
    "textPrefix": "本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似",
    "documentSetInfo": {
      "textLength": 5526,
      "byteLength": 12886,
      "indexedProgress": 100,
      "indexedStatus": "Ready",
      "createTime": "2023-12-29 11:14:45",
      "lastUpdateTime": "2023-12-29 11:14:47",
      "keywords": "向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似"
    },

```
    "splitterPreprocess": {
      "appendTitleToChunk": true,
      "appendKeywordsToChunk": true
    },
    "parsingProcess":{
      "parsingType":"VisionModelParsing"
    },
    "author": "Tencent",
    "tags": [
      "向量",
      "Embedding",
      "AI"
    ]
  }
}
```

参数名	子参数（一级）	子参数（二级）	参数含义
count	-	-	获取的数量。
documentSet	documentSetId	-	文件 ID。
	documentSetName	-	文件名。
	textPrefix	-	文件内容前200个字符。
	text	-	文件完整内容。
	documentSetInfo	textLength	文件的字符数。
		byteLength	文件的字节数。
		indexedProgress	文件被预处理、Embedding 向量化的进度。
		indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。
		createTime	文件创建时间。

		lastUpdateTime	文件最后更新时间。
		keywords	文件关键字。
	splitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将段落 Title 追加到切分后的段落。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 **keywords** 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 **keywords** 追加到切分后的段落。
	parsingProcess	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
	author	-	自定义的文件字段。

getChunks

最近更新时间：2025-05-26 11:14:52

Method 与 URL

获取文件切分后的语块：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/documentSet/getChunks` 。

> **说明：**
> **Chunk 指语块，较长文本在处理时会切分为多个语块，以便于向量化和更高效地检索，多个 Chunk 组成一个 DocumentSet。**
> - **支持指定具体的文件名获取文件切分后的语块。**
> - **支持指定具体的 DocumentSet ID 获取文件切分后的语块。**

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=bxLApUFljBJxoNtsycmg9Kz54KfjpS60wNBObpvr' \
  http://10.0.X.X:80/ai/documentSet/getChunks \
  -d '{
  "database": "db-test-ai",
  "collectionView": "coll-ai-files",
  "documentSetName": "腾讯云向量数据库.pdf",
  "offset": 0,
  "limit": 10
}'
```

参数名称	是否必选	配置方法及要求
database	是	使用 /database/list 获取集群中的数据库列表，复制数据库名。
collectionView	是	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制集合名。
documentSetId	否	文件上传后系统自动分配的 ID。 > **说明：**

		documentSetId 与 documentSetName 必须指定其中一个，以指定查询的文件。
documentSetName	否	文件存储在数据库中的名称。
limit	否	每页返回的 Chunks 数量。 • 数据类型：uint 64。 • 默认值：10。 • 取值范围：[1,1000]。
offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 limit 整数倍。 • 计算公式：offset=limit*(page-1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 - 1) = 10，表示从查询结果的第 11 条记录开始返回数据。

响应信息


```
{
  "code": 0,
  "msg": "Operation success",
  "requestId": "68564c113b2fa517d6bbc46d46f*****",
  "documentSetId": "1190130763145******",
  "documentSetName": "腾讯云向量数据库.pdf",
  "count": 10,
  "chunks": [
    {
      "text": "本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n",
      "startPos": 0,
      "endPos": 122
    },
    {
      "text": "## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级 QPS 及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。\n",
      "startPos": 122,
```

```
      "endPos": 313
    },
    {
      "text": "## 关键概念\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。\n",
      "startPos": 313,
      "endPos": 441
    },
    {
      "text": "### 什么是向量？\n向量是指在数学和物理中用来表示大小和方向的量。它由一组有序的数值组成，这些数值代表了向量在每个坐标轴上的分量。\n",
      "startPos": 441,
      "endPos": 508
    },
    {
      "text": "### 什么是非结构化数据？\n非结构化数据，是指图像、文本、音频等数据。与结构化数据相比，非结构化数据不遵循预定义模型或组织方式，通常更难以处理和分析。\n",
      "startPos": 508,
      "endPos": 585
    },
    {
      "text": "### 什么是 AI 中的向量表示？\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。具体上线时间，请关注［产品动态］(https://write.woa.com/document/117908688477143040)。\n",
      "startPos": 585,
      "endPos": 784
    },
    {
      "text": "### 什么是向量检索？\n向量检索是一种基于向量空间模型的信息检索方法。将非结构化的数据表示为向量存入向量数据库，向量检索通过计算查询向量与数据库中存储的向量的相似度来找到目标向量。\n",
      "startPos": 784,
      "endPos": 876
    },
    {
      "text": "## 为什么是腾讯云向量数据库？\n腾讯云向量数据库作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠、智能运维等方面体现出显著优势。 \n",
      "startPos": 876,
      "endPos": 1037
    },
```

```
    {
      "text": "## 腾讯云向量数据库应用示例有哪些？\n腾讯云向量数据库可进行高性能向量存储和检索，主要适用于以下应用场景。\n- [大规模知识库]：企业的私域数据存储在向量数据库中可构建外部知识库，帮助企业更好地管理和利用自己的数据资源。\n- [推荐系统]：向量数据库会基于用户特征进行向量存储与检索，最终筛选用户可能感兴趣的物品推荐给用户。",
      "startPos": 1037,
      "endPos": 1302
    },
    {
      "text": "\n- [问答系统]：向量数据库会基于问题信息进行向量存储与检索，并返回最相关的问题与对应的答案。\n- [文本/图像检索]：向量数据库对输入的图像和文本信息进行向量存储与检索，会找到最匹配输入信息的文本或图像结果。\n",
      "startPos": 1302,
      "endPos": 1511
    }
  ]
}
```


参数名	子参数	参数含义
documentSetId	-	文件上传后系统自动分配的 ID。
documentSetName	-	文件存储在数据库中的名称。
count	-	获取的 Chunks 数量。
chunks	text	获取的 Chunks 内容。
	startPos	每个 Chunks 在文件中偏移的起始位置。
	endPos	每个 Chunks 在文件中偏移的结束位置。

query

最近更新时间：2024-12-05 20:37:03

Method 与 URL

精确查找与查询条件完全匹配的文件：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/documentSet/query` 。

> **说明：**
> query 可获取文件 ID、向量化的进度与状态等，具体支持如下方式查找文件。
> - 支持指定具体的文件名查找文件，或搭配文件 Meta 信息对应字段的 Filter 表达式查询文件信息。
> - 支持指定具体的 DocumentSet ID 查找文件，或搭配文件 Meta 信息对应字段的 Filter 表达式查询文件信息。
> - 支持指定查询起始位置 offset 和返回数量 limit，查找指定范围的文件信息。
> - 支持根据文件 Meta 信息对应字段 Filter 表达式，直接过滤需查找的文件。

请求示例

使用文件名查询文件信息

如下示例，查找文件名为 `腾讯云向量数据库.md` ，且满足 author 与 tags 的 Filter 表达式的文件信息。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
   http://10.0.X.X:80/ai/documentSet/query \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "query": {
       "documentSetName": ["腾讯云向量数据库.pdf"],
       "filter": "author in (\"Tencent\", \"tencent\") and tags
include (\"AI\", \"Embedding\")",
       "outputFields": ["textPrefix", "tags","documentSetInfo"],
       "limit":10
    }
```

```
}'
```

使用范围查询文件

文件上传于向量数据库之后，可以使用 limit 与 offset 参数，设定查询的范围来查询文件信息。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/query \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "query": {
       "outputFields": ["textPrefix", "tags","documentSetInfo"],
       "offset":0,
       "limit":10
    }
}'
```

使用文件 ID 查询文件信息

文件上传于向量数据库之后，系统会自动分配文件 ID，获取文件的 ID 信息之后可通过文件 ID 批量查询文件信息。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/query \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "query": {
       "documentSetId": ["11801071477********"],
       "filter": "author=\"Tencent\"",
       "outputFields": ["textPrefix", "tags","documentSetInfo"],
```

```
        "limit":10
    }
}'
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要查询的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制数据库名。
collectionView	指定要查询的 CollectionView 名称。	-	是	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制集合名。
query	设置查询条件。	documentSetId	否	表示要查询的文件的所有 ID，支持批量查询，数组元素范围[1,20]。
		documentSetName	否	表示要查询的文档名称，支持批量查询，数组元素范围[1,20]。
		filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。 Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ string：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。

			○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`。
		limit	是	每页返回的 DocumentSet 数量。 • 数据类型：uint 64。 • 默认值：10。 • 取值范围：[1,16384] **⚠ 注意：** • 若不配置任何查询条件，即 `doc_list = coll_view.query()`，则默认返回 10 个 DocumentSet。 • 若查询条件仅配置 **Filter** 表达式，不配置 **limit**，则默认返回 10 条 DocumentSet。 • 若查询条件仅设置 **documentSetName** 或 **documentSetId**，则可不配置 **limit** 参数，默认返回 10 条数据。
		offset	否	设置分页偏移量，用于控制分页查询返回结果的起始位置，方便用户对数据进行分页展示和浏览。 • 取值：为 **limit** 整数倍。 • 计算公式：offset=limit*(page-1)。 • 例如：当 limit = 10，page = 2 时，分页偏移量 offset = 10 * (2 – 1) = 10，表示从查询结果的第 11 条记录开始返回数据。

		outputFields	否	以数组形式配置需返回的字段。

响应消息


```
{
  "code": 0,
  "msg": "Operation success",
  "requestId": "85f094becf3ba517bdbec4**********",
  "count": 1,
  "documentSets": [
    {
      "documentSetId": "1190130763145******",
      "documentSetName": "腾讯云向量数据库.pdf",
      "textPrefix": "本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（Tencent Cloud VectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n## 腾讯云向量数据库是什么？\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似",
      "documentSetInfo": {
        "textLength": 5526,
        "byteLength": 12886,
        "indexedProgress": 100,
        "indexedStatus": "Ready",
        "createTime": "2023-12-29 11:14:45",
        "lastUpdateTime": "2023-12-29 11:14:47",
        "keywords": "向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似"
      },
      "splitterPreprocess": {
        "appendTitleToChunk": true,
        "appendKeywordsToChunk": true
      },
      "parsingProcess":{
        "parsingType":"VisionModelParsing"
      },
      "author": "Tencent",
      "tags": [
        "向量",
        "Embedding",
        "AI"
      ]
    }
```

```
    ]
}
```

参数名	参数名	子参数	参数含义
count	-	-	查找到的文档数量。
documentSets	documnetSetId	-	文件 ID。
	documnetSetName	-	文件名。
	textPrefix	-	文件内容前 200个字符。
	documentSetInfo	textLength	文件的字符数。
		byteLength	文件的字节数。
		indexedProgress	文件被预处理、Embedding 向量化的进度。
		indexedStatus	文件预处理、Embedding 向量化的状态。 • New：等待解析。 • Loading：文件解析中。 • Failure：文件解析、写入出错。 • Ready：文件解析、写入完成。
		indexedErrorMsg	文件解析、写入错误描述信息。 **说明：** 当 IndexedStatus 为 Failure 时，返回 indexedErrorMsg 信息。
		createTime	文件创建时间。
		lastUpdateTime	文件最后更新时间。
		keywords	文件关键字。
	splitterPreprocess	appendTitleToChunk	在对文件拆分时，配置是否将 Title 追加到切分后的段落后面一并

			Embedding。取值如下所示： • **false**：不追加。 • **true**：将段落 Title 追加到切分后的段落。
		appendKeywordsToChunk	在对文件拆分时，配置是否将关键字 keywords 追加到切分后的段落一并 Embedding。取值如下所示： • **false**：不追加。 • **true**：将全文的 keywords 追加到切分后的段落。
	parsingProcess	parsingType	指定 PDF 类型文件的解析方式，取值如下所示： • **VisionModelParsing**：文件依据解析模型解析，推荐使用，可解析 PDF 中双栏、表格等复杂格式。 • **AlgorithmParsing**：文件依据算法解析，系统默认解析方式。Markdown、Word、PPT 类型，无需配置该参数，默认使用 AlgorithmParsing 解析。
	tags	-	自定义的文件 Meta 信息字段。

search

最近更新时间：2025-07-01 10:19:43

Method 与 URL

查找与给定文本信息相似的 Top K 条文本信息：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/documentSet/search` 。

> **说明：**
> - 支持指定文件名称检索最相似的文本信息。
> - 支持文件名搭配文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。
> - 支持仅使用文件元数据的标量字段的 Filter 表达式检索最相似的文本信息。

请求示例

如下示例，在文件名为 `腾讯云向量数据库.md` ，且满足 Filter 表达式的文件中，检索与 `什么是向量数据库` 相似的文本信息。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/search \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "search":{
        "content": "什么是向量数据库",
        "documentSetName": ["腾讯云向量数据库.md"],
        "options": {
          "chunkExpand": [
            1,
            1
          ]
        },
      "filter": "author in (\"Tencent\", \"tencent\") and tags include
(\"AI\", \"Embedding\")",
      "limit": 3
    }
}'
```

参数名称	参数含义	子参数	子参数	是否必选	配置方法及要求
database	指定要查询的 Database 名称。	-	-	是	使用 /database/list 获取集群中的数据库列表，复制数据库名。
collectionView	指定要查询的 CollectionView 名称。	-	-	是	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制集合名。
search	设置检索数据的条件。	content	-	是	以 String 类型输入检索的文本信息。
		documentSetName	-	否	检索的文件名，支持批量查询，数组元素范围 [1,10]。
		options	chunkExpand	否	• 以数组形式配置检索的目标信息所需向前扩展的段落数量以及向后扩展的段落数。例如，输入 [2,3]，指所检索到的 Chunk 返回时，同时返回其之前的 2个段落与之后的3个段落。 • 段落指文件在上传存储时，自动向量化拆分的段落。 • 默认值：[1,1]。
		filter	-	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。 Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其

					对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例： `Filter('author="jerry"').And('page>20')`
		limit	–	是	指定返回最相似的 Top K 的 K 的值。

返回消息


```
{
  "code": 0,
  "msg": "Operation success",
  "requestId": "cb0f2728a7c64c1835211f5f2b3a41b1",
  "documents": [
    {
      "score": 0.855039119720459,
      "data": {
        "text": "## 腾讯云向量数据库是什么？\n\n腾讯云向量数据库是一款全托管的自研企业级分布式数据库服务，专用于存储、检索、分析多维向量数据。该数据库支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，可支持百万级QPS及毫秒级查询延迟。腾讯云向量数据库不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP服务、计算机视觉、智能客服等AI领域。\n\n",
        "startPos": 1154,
        "endPos": 1342,
        "pre": [
          "## 向量数据库\n\n## 产品简介\n\n## 腾讯云向量数据库\n\n最近更新时间：2023-07-2714:46:31\n\n本页面旨在通过回答几个问题来让您大致了解腾讯云向量数据库（TencentCloudVectorDB）。读完本页后，您将了解腾讯云向量数据库是什么、它是如何工作的、关键概念、为什么使用腾讯云向量数据库、支持的索引和指标、架构和相关连接方式。\n\n"
        ],
        "next": [
```

```
            "## 关键概念\n\n如果您不熟悉向量数据库和相似性搜索领域，请优先阅读以下基本概念，便于您对向量数据库有一个初步的了解。更多名词解释，请阅读关键概念。\n\n"
          ],
          "paragraphTitle": "腾讯云向量数据库是什么？",
          "allParentParagraphTitles": [
            "腾讯云向量数据库是什么？"
          ]
        },
        "documentSet": {
          "documentSetId": "1387985304602550272",
          "documentSetName": "腾讯云向量数据库.pdf",
          "author": "Tencent",
          "tags": [
            "向量",
            "Embedding",
            "AI"
          ]
        }
      },
      {
        "score": 0.8471382260322571,
        "data": {
          "text": "## 什么是向量检索？\n\n向量检索是将向量与数据库进行比较以查找与查询向量最相似的向量的过程。相似的向量通常具有相近的原始数据，通过向量检索可以挖掘出原始非结构化数据之间的联系。\n\n",
          "startPos": 1708,
          "endPos": 1799,
          "pre": [
            "## 什么是AI中的向量表示？\n\n当我们处理非结构化数据时，需要将其转换为计算机可以理解和处理的形式。向量表示是一种将非结构化数据转换为嵌入向量的技术，通过多维度向量数值表述某个对象或事物的属性或者特征。腾讯云向量数据库提供的模型能力，目前在开发调试中。具体上线时间，请关注产品动态。\n\n"
          ],
          "next": [
            "## 为什么是腾讯云向量数据库？\n\n## 版权所有：腾讯云计算（北京）有限责任公司\n\n## 第4共17页\n\n\n\n![vdb-image](https://vecterdb-**********\n\n"
          ],
          "paragraphTitle": "什么是向量检索？",
          "allParentParagraphTitles": [
            "什么是向量检索？"
          ]
        },
```

```
      "documentSet": {
        "documentSetId": "1387985304602550272",
        "documentSetName": "腾讯云向量数据库.pdf",
        "author": "Tencent",
        "tags": [
          "向量",
          "Embedding",
          "AI"
        ]
      }
    },
    {
      "score": 0.846755862236023,
      "data": {
        "text": "## 向量数据库\n\n## 产品优势\n\n最近更新时间：2023-08-2115:36：41\n\n腾讯云向量数据库（TencentCloudVectorDB）作为一种专门存储和检索向量数据的服务提供给用户，在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。\n\n",
        "startPos": 4991,
        "endPos": 5130,
        "pre": [
          "## 开发者工具\n\n![vdb-image](https://vecterdb-backup-gz-**********\n\n腾讯云向量数据库支持丰富的API接口，以促进DevOpS。具体操作，请参见API文档。\n\n\n\n![vdb-image](https://vecterdb-backup-gz-*********\n\n"
        ],
        "next": [
          "## 高性能\n\n## 向量数据库单索引支持10亿级向量数据规模，可支持百万级QPS及毫秒级查询延迟。\n\n## 高可用\n\n![vdb-image](https://vecterdb-backup-gz-**********\n\n向量数据库提供多副本高可用特性，其多可用区和三节点的架构可用性可达99.99%，显著提高系统的可靠性和容错性，确保数据库在面临节点故障和负载变化等挑战时仍能正常运行。\n\n"
        ],
        "paragraphTitle": "## 向量数据库\n\n## 产品优势",
        "allParentParagraphTitles": [
          "产品优势"
        ]
      },
      "documentSet": {
        "documentSetId": "1387985304602550272",
        "documentSetName": "腾讯云向量数据库.pdf",
        "author": "Tencent",
        "tags": [
          "向量",
```

```
            "Embedding",
            "AI"
          ]
        }
      }
    ]
}
```

参数名	子参数	参数含义
score	-	表示查询向量与检索结果向量之间的相似性计算分数。
data	text	检索的结果。
	endPos	检索结果在文件中偏移的结束位置。
	startPos	检索结果在文件中偏移的起始位置。
	next	根据检索时，设置的参数 chunkExpand，返回检索结果向后扩展的段落。
	pre	根据检索时，设置的参数 chunkExpand，返回检索结果向前扩展的段落。
	paragraphTitle	检索到的 chunk 所属的段落标题。
	allParentParagraphTitles	检索到的 chunk 所属的所有层级的段落标题。
documentSet	documentSetId	文件 ID。
	documentSetName	文件名。
	other_scalar_field	自定义的文件 Meta 信息的标量字段。例如：author、bookName、page 等。 **说明：** 显示创建 CollectionView 时设置为 Filter 索引的字段，同时显示上传文件时或使用 update 新增的字段，但新增的字段不会构建索引。

delete

最近更新时间：2024-12-05 20:37:03

Method 与 URL

删除文件：`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/documentSet/delete` 。

> **说明：**
> - 支持批量删除，文件 ID 或文件名数组元素数量最大为 20。
> - 支持使用 Filter 表达式过滤所需删除的所有文件。

使用示例

根据文件名查找需删除的文件

如下示例，删除指定文件名，且满足 Filter 表达式的文件。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/delete \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "query": {
        "documentSetName": ["腾讯云向量数据库.pdf"],
        "filter": "author in (\"Tencent\",\"tencent\")"
    }
}'
```

根据文件 ID 查找需删除的文件

如下示例，删除指定文件 ID，且满足 Filter 表达式的文件。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
```

```
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/delete \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "query": {
        "documentSetId": ["1179017994589437952"],
        "filter": "author in (\"Tencent\",\"tencent\")"
    }
}'
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定需删除文档的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需删除数据的集合所属的数据库名。
collectionView	指定需删除文档的 CollectionView 名称。	-	是	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制需删除数据的集合视图。
query	设置查询条件。	documentSetName	否	表示要删除的 DocumentSet 的文件名称，可以批量删除，数据元素最大值为20。 **说明：** 同时配置 documentSetName 与 filter 参数，删除数据将会取二者的并集。
		documentSetId	否	表示要删除的 DocumentSet 的 ID，可以批量删除，数据元素最大值为20。 **说明：**

				同时配置 documentSetId 与 filter 参数，删除数据将会取二者的并集。
		filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。 Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中： • <field_name>：表示要过滤的字段名。 • <operator>：表示要使用的运算符。 ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。 ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。 ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。 • <value>：表示要匹配的值。 示例： `Filter('author="jerry"').And('page>20')`

响应消息

状态码

状态码	含义	响应消息
200	删除 DocumentSet 成功	{ "code": 0, "msg": "operation success", "affectedCount": 2

		<pre>}</pre>

400	删除 DocumentSet 失败	<pre>{
		  "code": 400,
		  "msg": "operation failed, reason...."
		}</pre>

返回参数

参数名	参数含义
affectedCount	删除文档数量。

update

最近更新时间：2024-12-05 20:37:03

Method 与 URL

更新或新增文件标量字段：

`POST https://{实例访问 IP 地址}:{实例网络端口}/ai/documentSet/update` 。

> **说明：**
> - 支持通过主键（DocumentSet ID）或 文件名，搭配 Filter 表达式过滤需更新的文件。
> - 支持新增字段，支持更改部分字段。新增字段，在创建 CollectionView 时没有为这些字段指定索引方式，那么新增这些字段时，系统不会自动为其创建索引。

> **注意：**
> - 不能变更系统分配的 DocumentSet ID 字段，不要求事务完整性。
> - 不能变更已上传的文件内容。

使用示例

根据文件名查找需更新的文件

如下示例，修改指定文件名，且满足 Filter 表达式的文件的 meta 信息字段 author。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/update \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "query":{
        "documentSetName": ["腾讯云向量数据库.pdf"],
        "filter": "author=\"Tencent\""
    },
    "update": {
      "author": "tencent"
    }
```

```
}'
```

根据 ID 查找需更新的文件

如下示例，修改指定文件 ID，且满足 Filter 表达式的文件的 meta 信息字段 author。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/ai/documentSet/update \
  -d '{
    "database": "db-test-ai",
    "collectionView": "coll-ai-files",
    "query":{
        "documentSetId": ["1179017994589437952"],
        "filter": "author=\"Tencent\""
    },
    "update": {
      "author": "tencent"
    }
}'
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法及要求
database	指定要更文档的 Database 名称。	-	是	使用 /database/list 获取集群中的数据库列表，复制需更新数据的集合所属的数据库名。
collectionView	指定要更新文档的 Collecti	-	是	使用 /ai/collectionView/list 获取指定数据库名下的 CollectionView 列表，复制需更新数据的集合名。

	on 名称。			
query	设置查询条件检索需更新的文档	documentSetName	否	表示要更新的 DocumentSet 的文件名称，可以批量更新，数据元素最大值为20。 **说明：** 同时配置 documentSetName 与 filter 参数，更新数据将会取二者的并集。
		documentSetId	否	表示要更新的 DocumentSet 的 ID，可以批量更新，数据元素最大值为20。 **说明：** 同时配置 documentSetId 与 filter 参数，更新数据将会取二者的并集。

		filter	否	使用创建 CollectionView 指定的 Filter 索引的字段设置查询过滤表达式。
				Filter 表达式格式为 <field_name><operator><value>，多个表达式之间支持 and（与）、or（或）、not（非）关系。具体信息，请参见 混合检索 。其中：
				• <field_name>：表示要过滤的字段名。
				• <operator>：表示要使用的运算符。
				  ○ **string**：匹配单个字符串值（=）、排除单个字符串值（!=）、匹配任意一个字符串值（in）、排除所有字符串值（not in）。其对应的 Value 必须使用英文双引号括起来。
				  ○ **uint64**：大于（>）、大于等于（>=）、等于（=）、小于（<）、小于等于（<=）。例如：expired_time > 1623388524。
				  ○ **array**：数组类型，包含数组元素之一（include）、排除数组元素之一（exclude）、全包含数组元素（include all）。例如，name include (\"Bob\", \"Jack\")。
				• <value>：表示要匹配的值。
				示例：
				`Filter('author="jerry"').And('page>20')`

update	设置需更新的字段	old_field	否	当前已存在的字段，更新字段对应的数据。 • 类型：string • 字符长度要求：[1,256]。
		new_field	否	新增字段，并给新字段赋值。 • 类型：string。 • 字符长度要求：[1,256]。

响应消息

```
{
    "code":0,
    "msg":"Operation success, requestId:
0945972e7aca9b1784bbef57c8e7d867",
    "affectedCount":0
}
```

参数名	参数含义
affectedCount	更新的文档数量。如果该参数返回的值为0，说明更新无效。

Index rebuild

最近更新时间：2025-07-01 10:19:43

本接口（/index/rebuild）用于重建指定 Base 类数据库 Collection 的所有索引，清除无用的索引数据，修复损坏的索引数据，优化索引结构，改善性能。

说明：

rebuild 接口用于重建指定集合的向量索引，主要适用于以下两种场景：

- 首次使用 IVF 系列索引时，需要通过 rebuild 接口初始化索引结构。具体信息，请参见 IVF 系列索引应用指南 。
- 在离线批量数据导入场景中，建议先执行数据写入操作（通过 upsert 接口设置 buildIndex=false暂停索引构建），待数据完整导入后，再调用rebuild 接口集中创建索引，以此减少写入过程中的索引维护开销，提升整体导入效率。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/index/rebuild`

警告：

Rebuild 操作需占用额外内存且可能影响服务，执行前请评估业务场景并选择重建方式。

- 保留原索引、同时构建新索引（即设置dropBeforeRebuild=False）：期间集合读取正常，写入被阻止，构建新索引后，替换原索引，构建过程中，旧索引和新索引都会占用内存，如果内存足够可采用该方式，建议根据实际业务高低峰选择停写时段执行。
- 先删除原索引、再构建新索引（即设置dropBeforeRebuild=True）：期间集合读写会受到影响，先删除旧索引，再构建新索引。如果内存严重不足时，则需采用该方案。建议根据业务需求选择可中断服务的时段处理。

使用示例

注意：

如下示例可直接复制，在 CVM 运行之前，您需在文本编辑器将 api_key=A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X 依据实际情况进行替换。

```
curl -i -X POST \
```

```
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/index/rebuild  \
 -d '{
    "database": "db-test",
    "collection": "book-vector",
    "dropBeforeRebuild": true,
    "throttle": 1
}'
```

请求参数

参数	是否必选	参数含义	配置方法及要求
database	是	指定需重建索引的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection	是	指定需重建索引的 Collection 名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
fieldName	是	指定重建索引的字段。	• vector：重建稠密向量索引。 • sparse_vector：重建稀疏向量索引。 **说明：** 同一集合，不支持并发执行 rebuild。如果集合中既有稠密向量又有稀疏向量，则需要分别进行重建。
dropBeforeRebuild	否	标识在重建索引时，是否需先删除旧索引再重建新索引。 **警告：** 重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在重建索引之前，您需根据实际资源情况选择是否需先删除旧索	取值如下所示： • **true**：重建之前，先删除旧索引再重建索引。 **警告：** 内存资源不足时，可配置为 true，先删除旧索引，在新索引构建过程中，无法正常读写。

		引再重建，避免引起内存占满而阻塞业务正常运行。	• false：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为 false。 **警告：** 内存资源足够的情况下，可配置为 false，可不删除旧索引。在新索引构建过程中，可读数据，禁止写入数据。
throttle	否	标识是否限制构建索引的单节点 CPU 核数。 **说明：** 重建索引会消耗 CPU 资源，为防止资源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为1。	取值如下所示： • 0：不限制 CPU 核数。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数。

响应消息

HTTP 响应消息体是服务器返回给客户端的数据，通常包含了请求的结果或者所请求的资源。/index/rebuild 接口返回的状态码以及相关信息，如下表所示。

状态码	含义	响应信息

200	创建成功	{
		"code": 0,
		"msg": "operation success"
		}

400	创建失败	{ "code": 1,

```
    "msg": "operation failed, reason...."
}
```

相关说明

使用 /collection/describe 接口查看 Collection 的索引状态，返回参数 **indexStatus** 中的 **status** 标识当前 Collection 重建索引的状态，**startTime** 显示重建索引开始的时间。

- **initial**：表示首次创建索引，尚未 Rebuild 构建索引。
- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。
- **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。
- **failed**：重建索引失败，可能会影响集合读写操作。

> ⚠ **注意：**
> **training data** 与 **building index** 状态期间不可写入数据。若重建索引之前先删除旧索引，则集合不可进行相似性查询，只能进行精确查询。

```
{
    "code": 0,
    "msg": "operation success",
    "collection": {
        "database": "db-test",
        "collection": "coll-test",
        "documentCount": 4,
        "indexes": [
            ......
        ],
        "indexStatus": {
            "status": "ready",
            "startTime": ""
        }
    }
}
```

add

最近更新时间：2025-04-18 17:58:12

Method 与 URL

/index/add 接口用于在已有集合上，新增标量字段索引：

`POST https://{实例访问 IP 地址}:{实例网络端口}/index/add` 。

> **说明：**
> - 如果集合开启了动态 schema（即创建集合时配置了 filterIndexConfig 参数），不支持创建标量索引。
> - 当前不支持创建 json 类型的标量索引。

使用示例

> **注意：**
> 如下示例可直接复制，在 CVM 运行之前，您需在文本编辑器将 api_key=A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X 依据实际情况进行替换。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/index/add  \
 -d '{
    "database": "db-test",
    "collection": "book-vector",
    "indexes": [
      {
            "fieldName": "bookInfo",
            "fieldType": "string",
            "indexType": "filter"
      },
      {
            "fieldName": "page",
            "fieldType": "uint64",
            "indexType": "filter"
      }],
```

```
    "buildExistedData": true
}'
```

请求参数

参数	子参数	是否必选	参数含义	配置方法及要求
database	-	是	配置需重建索引的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection	-	是	指定需重建索引的 Collection 名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
indexes	fieldName	是	指定新增标量索引的字段。	用户自定义字段名。命名规则为1-128字符，只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。
	fieldType		指定字段数据类型。	指定自定义字段的数据类型。取值如下： • string：字符型。 • uint64：指无符号整数（unsigned integer）。 • array：数组类型，数组元素为string。
	indexType		指定索引类型。	仅支持设置为 `filter` 。
buildExistedData	-	是	指定新增的索引字段是否写入过数据。	true：默认为true，新增字段已写入过数据，需扫描历史数据并构建索引。 • false：新增索引字段为新增字段，无需扫描历史数据，可以快速创建索引。

drop

最近更新时间：2025-04-18 17:58:12

Method 与 URL

/index/drop 接口用于在已有集合上，删除标量字段索引：

`POST https://{实例访问 IP 地址}:{实例网络端口}/index/drop` 。

> ⚠ **注意：**
>
> - 仅支持删除标量索引，不支持删除向量、稀疏向量索引。
> - 如果集合开启了动态schema（即创建集合时配置了filterIndexConfig），不支持删除标量索引。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=FDmSuVuL8qLPxeGak5FAD8xHBUfVGtMju1XknUIa' \
  http://10.0.X.X:80/index/drop  \
 -d '{
    "database": "db-test",
    "collection": "book-vector",
    "fieldNames": ["bookInfo","page"]
}'
```

请求参数

参数	是否必选	参数含义	配置方法及要求
database	是	配置需删除索引的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection	是	指定需删除索引的 Collection 名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
fieldNames	是	用于指定删除的标量索引的字段名。	以数组形式，列出需删除的字段名。

版权所有：腾讯云计算（北京）有限责任公司

modifyVectorIndex

最近更新时间：2025-07-01 10:19:43

本接口（ `/index/modifyVectorIndex` ）用于在不重建整个索引的前提下，调整已创建向量索引的配置参数，包括：数据精度类型、距离度量方式、性能优化参数等。

说明：

- 仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。
- 索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。
- HNSW 索引支持重新指定参数 M 和 efConstruction，IVF 索引支持重新指定 nlist（IVF_PQ 支持重新指定 M 和 nlist）。
- 仅 HNSW 索引，支持修改字段数据类型，例如：从vector（标准单精度浮点数向量）可修改为 float16_vector（半精度浮点数向量）。
- 支持重新指定向量相似度计算方法。

Method 与 URL

`POST https://{实例访问 IP 地址}:{实例网络端口}/index/modifyVectorIndex`

使用示例

注意：
如下示例可直接复制，在 CVM 运行之前，您需在文本编辑器将 api_key=A5VOgsMpGWJhUI0WmUbY********** 与 10.0.X.X 依据实际情况进行替换。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/index/modifyVectorIndex \
 -d '{
    "database": "db-test",
    "collection": "book-vector",
    "vectorIndexes": [
        {
            "fieldName": "vector",
```

```
            "indexType": "HNSW",
            "fieldType": "bfloat16_vector",
            "metricType": "IP",
            "params": {
                "M": 16,
                "efConstruction": 500
            }
        }
    ],
    "rebuildRules": {
        "dropBeforeRebuild": true,
        "throttle": 1
    }
}'
```

请求参数

参数	子参数	是否必选	参数含义	配置方法及要求
database	-	是	配置需重建索引的 Database 名称。	使用 /database/list 获取集群中的数据库列表，复制需重建索引的集合所属的数据库名。
collection	-	是	指定需 Collection 的重建索引的 Collection 名称。	使用 /collection/list 获取指定数据库名下的 Collection 列表，复制需重建索引的集合名。
vectorIndexes	fieldName	是	稠密向量数据字段名。	固定为 vector。 **说明：** 仅支持对稠密向量的索引参数进行调整，不支持对稀疏向量的索引参数修改。
	fieldType	否	稠密向量数据字段类型。	仅 HNSW 索引可重新指定向量索引的数据类型。默认值与当前类型一致，具体取值如下所示。 • vector：单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32

				位），适用于需最高精度的计算任务或对误差敏感的向量检索。 • **float16_vector**：半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位的格式，适用于尾数精度较高（小数点后保留更多位数）的场景，如 BGE-large 等模型生成的嵌入向量。 • **bfloat16_vector**：bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式，指数范围与 float32 一致，适用于向量数值范围较大（更多bit位表达整数）的场景。
	indexType	否	指定调整后的目标索引类型。如果无需调整索引类型，该参数可不填。	指定索引类型，默认与原索引类型类型，取值如下所示。 **说明：** • 索引类型支持 FLAT、HNSW、IVF 系列索引之间互相调整，二进制索引（BIN_FLAT）不支持调整。 • 切换索引类型时，请同步修改 params 参数，与目标索引类型的参数保持一致。 • FLAT 和 IVF 索引不支持 float16_vector 或 bfloat16_vector，当从 HNSW 索引类型切换到 FLAT 或 IVF 类型时，如果字段类型（fieldType）为 float16_vector 或 bfloat16_vector，则必须同步将字段类型修改为 vector 类型。 • 索引切换时需要注意提前预估内存占用，尤其是在 IVF 系列索引切换成 FLAT/HNSW 时，内存差异可能较大。 • **FLAT**：暴力检索，召回率100%，但检索效率低，适用10万以内数据规模。

				• **HNSW**：召回率95%+，可通过参数调整召回率，检索效率高，但数据量大后写入效率会变低，适用于10万-1亿区间数据规模。 • **IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16**：IVF 系列索引，适用于上亿规模的数据集，检索效率高，内存占用低，写入效率高。
	metricType	否	指定调整后的相似度计算方法。	重新指定向量之间距离度量的算法。取值如下： • **L2**：全称是 Euclidean distance，指欧几里得距离，它计算向量之间的直线距离，所得的值越小，越与搜索值相似。L2在低维空间中表现良好，但是在高维空间中，由于维度灾难的影响，L2的效果会逐渐变差。 • **IP**：全称为 Inner Product，是一种计算向量之间相似度的度量算法，它计算两个向量之间的点积（内积），所得值越大越与搜索值相似。 • **COSINE**：余弦相似度（Cosine Similarity）算法，是一种常用的文本相似度计算方法。它通过计算两个向量在多维空间中的夹角余弦值来衡量它们的相似程度，所得值越大越与搜索值相似。

	params	否	指定目标索引类型的索引参数。	• 索引类型 **indexType** 为 HNSW ，需配置如下参数。
				  ○ **M**：每个节点在检索构图中可以连接多少个邻居节点。
				  ○ 取值类型：uint64。
				  ○ 取值范围：[4,64]。一般可设置为16。
				  ○ **efConstruction**：搜索时，指定寻找节点邻居遍历的范围。数值越大构图效果越好，构图时间越长。
				  ○ 取值类型：uint64。
				  ○ 取值范围：[8,512]。一般可设置为200。
				• 索引类型 **indexType** 为 IVF_FLAT、IVF_PQ、IVF_SQ4, IVF_SQ8, IVF_SQ16需配置如下参数。

				○ **nlist**：指索引中的聚类中心数量。 取值类型：uint64。取值范围：[1,65536]。 ○ **M**：指乘积量化中原始数据被拆分的子向量的数量。该参数仅 IVF_PQ 索引类型需配置。更多信息，请参见 索引与计算 。 ○ 取值要求：原始数据的向量的维度 D（即向量中元素的个数）必须能够被 m 整除，m 必须是一个正整数。 ○ 取值范围：[1,向量维度]。
rebuild Rules	**dropBeforeRebuild**	否	标识在重建索引时，是否需先删除旧索引再重建新索引。 **警告：** 重建索引需要占用额外的内存空间，数据量越大，消耗的内存空间越大。在重建索引之前，您需根据实际资源情况选择是否需先删除旧索引再重建，避免引起内存占满而阻塞业务正常运行。	取值如下所示： • **true**：重建之前，先删除旧索引再重建索引。 **警告：** 内存资源不足时，可配置为 true，先删除旧索引，在新索引构建过程中，无法正常读写。 • **false**：重建之前，不删除旧索引，创建新索引完成之后再删除旧索引。默认为 **false**。 **警告：** 内存资源足够的情况下，可配置为 false，可不删除旧索引。在新索引构建过程中，可读数据，禁止写入数据。
	throttle	否	标识是否限制构建索引的单节点 CPU 核数。 **说明：** 重建索引会消耗 CPU 资源，为防止资	取值如下所示： • 0：不限制 CPU 核数。 • 1~节点 CPU 核数：可用于 rebuild 的 CPU 资源最大为实例单个节点规格的 CPU 核数。

			源打满影响写入或者检索等操作，请根据业务实际配置重建索引的 CPU 核数。默认为限制 CPU 核数为 1。	

响应消息

```
{
  "code": 0,
  "msg": "Start rebuilding. You can use the '/collection/describe' API
to follow the progress of rebuilding."
}
```

参数名	参数含义
msg	提示启动 rebuild，可使用 /collection/describe 接口返回的参数 indexStatus 查看集合索引状态。 • **initial**：表示首次创建索引，尚未 Rebuild 构建索引。 • **ready**：表示当前 Collection 准备就绪，可正常使用。 • **training**：表示当前 Collection 正在进行数据训练，即训练模型已生成向量数据。 • **building**：表示当前 Collection 正在重建索引，即将生成的向量数据存储到新的索引中。 • **failed**：重建索引失败，可能会影响集合读写操作。

User create

最近更新时间：2025-01-21 19:52:13

Method 与 URL

创建用户：`POST https://{实例访问 IP 地址}:{实例网络端口}/user/create` 。

使用须知

说明：

- 单个实例能够支持最多创建100个用户账户。
- 对于新创建的用户，支持自定义用户密码。为了确保与旧版本实例的兼容性，root 用户将继续保留其默认的 API_KEY。同时，用户可以通过 /user/changePassword 接口为 root 用户设置密码，以便使用“password”进行连接认证。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/user/create  \
 -d '{
    "user": "app_user",
    "password": "********"
}'
```

请求参数

参数名	参数含义	数据类型	是否必填	默认值	参数配置
user	自定义用户名	string	是	-	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。

password	设置密码	string	是	-	• 密码长度要求：8~128字符。
					• 密码设置支持如下类型的任意组合。
					  ○ 字母：大写 A~Z、小写 a~z。
					  ○ 数字：0~9。
					  ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

响应消息

```
{
  "code": 0,
  "msg": "Operation success"
}
```

grant

最近更新时间：2025-01-21 19:52:13

Method 与 URL

对用户授予指定资源的访问权限：`POST https://{实例访问 IP 地址}:{实例网络端口}/user/grant`。

使用示例

如下示例，给用户 app_user 授予读取 db-0 下所有集合数据的权限，并具有数据库 db1 下 coll1 集合的读写权限。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/user/grant  \
 -d '{
    "user": "app_user",
    "privileges": [
    {
      "resource": "db0.*",
      "actions": ["read"]
    },
    {
      "resource": "db1.coll1",
      "actions": ["readWrite"]
    }
  ]
}'
```

请求参数

参数名	参数含义	子参数	数据类型	是否必填	参数配置
user	需授权的用户名	-	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。

					说明： root 用户享有最高权限级别，无法被任何账号（包括其自身）删除或修改其权限。
privileges	对用户授予的权限列表，可以配置多条权限	resource	string	是	指定用户可操作的资源对象，分为如下三个层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"`。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有Collection。配置方式：`"resource": "db0.*"`，其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。 **说明：** resource 只能指向已存在的资源；若指定的资源不存在，则触发错误提示信息。
		actions	array	是	配置资源对象允许操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

响应消息

```
{
  "code": 0,
  "msg": "Operation success"
}
```

版权所有：腾讯云计算（北京）有限责任公司

revoke

最近更新时间：2025-01-21 19:52:13

Method 与 URL

撤销用户已拥有的权限：`POST https://{实例访问 IP 地址}:{实例网络端口}/user/revoke`。

说明：

- root 用户为数据库的默认用户，享有最高权限级别，无法被任何账号（包括其自身）撤销。
- 撤销权限时，权限需要和授予时完全对应，包括资源对象和允许的操作，都需要保持一致。
  - 只能针对原本层级回收权限，不能单独回收子对象的权限。例如，用户拥有 *.* 下的权限，那么不能单独回收 db.* 或者 db.coll 的权限。
  - 只能回收授予时指定的 action。例如，用户拥有某个资源对象的 readWrite 权限，那么回收时不能单独回收其 read 权限。
- **授权后，如果资源对象被删除，权限不会被撤销。后续如果该资源对象被重新创建，用户依然拥有原本的访问权限。建议用户在删除资源对象后，同时清理对应的访问权限。**

使用示例

如下示例，撤销用户 app-user 对数据库 db-test1 下集合的 DDL 操作权限。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/user/revoke  \
 -d '{
    "user": "app_user",
    "privileges": [
    {
      "resource": "db0.*",
      "actions": ["read"]
    }
  ]
}'
```

请求参数

参数名	参数含义	子参数	数据类型	是否必填	参数配置
user	用户名	-	string	是	需撤销操作权限的用户名。 • 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。
privileges	指定撤销用户的资源对象以及其权限，支持一次撤销多条权限。	resource	string	是	指定用户撤销的资源对象，必须与授权时的指定的层级范围一致。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
		actions	array	是	指定资源对象允许操作的权限，必须与授权时的权限保持一致。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

响应消息

```
{
```

```
    "code": 0,
    "msg": "Operation success"
}
```

版权所有：腾讯云计算（北京）有限责任公司

describe

最近更新时间：2025-01-21 19:52:13

Method 与 URL

查询指定用户的信息，包括用户的创建时间、拥有的权限列表：

`POST https://{实例访问 IP 地址}:{实例网络端口}/user/describe` 。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/user/describe  \
 -d '{
    "user": "app_user"
}'
```

请求参数

参数名	参数含义	子参数	数据类型	是否必填	参数配置
user	需撤销的用户名	-	string	是	指定需授权的用户名。 **说明：** root 用户享有最高权限级别，无法被任何账号（包括其自身）删除或修改其权限。

响应消息

```
{
  "code": 0,
  "msg": "Operation success",
  "user": "app_user",
  "createTime": "2024-12-26 11:05:42",
  "privileges": [
```

```
    {
      "resource": "db0.*",
      "actions": ["read"]
    },
    {
      "resource": "db1.coll1",
      "actions": ["readWrite"]
    }
  ]
}
```

参数名	子参数	参数配置
user	-	查询的用户名。
createTime	-	用户创建时间。
privileges	resource	用户可操作的资源对象层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

list

最近更新时间：2025-01-21 19:52:13

Method 与 URL

查看实例下所有用户的信息列表：`POST https://{实例访问 IP 地址}:{实例网络端口}/user/list` 。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/user/list  \
```

响应消息

```
{
  "code": 0,
  "msg": "Operation success",
  "users": [
    {
      "user": "root",
      "createTime": "",
      "privileges": [
        {
          "resource": "*.*",
          "actions": ["fullAccess"]
        }
      ]
    },
    {
      "user": "app_user",
      "createTime": "2024-12-26 11:05:42",
      "privileges": [
        {
          "resource": "db0.*",
          "actions": ["read"]
        },
        {
          "resource": "db0.coll1",
```

```
            "actions": ["readWrite"]
          }
        ]
      }
    ]
}
```

参数名	子参数	参数配置
user	-	查询的用户名，包含显示 root 用户以及所有已创建的用户。
createTime	-	用户创建时间，其中 root 用户创建时间为空。
privileges	resource	用户可操作的资源对象层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式：`"resource": "db0.*",` 其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式为为 `"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。
	actions	用户可操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

drop

最近更新时间：2025-01-21 19:52:13

Method 与 URL

删除已创建的用户：`POST https://{实例访问 IP 地址}:{实例网络端口}/user/drop`。

> **说明：**
> - 只能删除自定义用户，root 用户无法删除。
> - 只有 root 能够调用该接口。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/user/drop  \
  -d '{
    "user": "app_user"
}'
```

请求参数

参数名	参数含义	数据类型	是否必填	参数配置
user	需删除的自定义的用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母（大写 A~Z、小写 a~z）、数字（0~9）、下划线(_)，并以英文字母开头。 **说明：** root 用户无法被任何账号（包括其自身）删除。

响应消息

```
{
  "code": 0,
  "msg": "Operation success"
}
```

版权所有：腾讯云计算（北京）有限责任公司

changePassword

最近更新时间：2025-01-21 19:52:13

Method 与 URL

修改自定义用户密码：`POST https://{实例访问 IP 地址}:{实例网络端口}/user/changePassword` 。

使用示例

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/user/changePassword  \
  -d '{
    "user": "app_user",
    "password": "************"
}'
```

请求参数

参数名	参数含义	数据类型	是否必填	参数配置
user	指定用户名	string	是	• 长度要求：1~32字符。 • 字符类型要求：只能使用英文字母、数字、下划线，并以英文字母开头。

password	设置新密码	string	是	• 密码长度要求：8~128字符。
				• 密码字符类型要求如下：
				  ○ 字母：大写 A~Z、小写 a~z。
				  ○ 数字：0~9。
				  ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

响应消息

```
{
```

```
    "code": 0,
    "msg": "Operation success"
}
```

错误码

最近更新时间：2025-03-17 11:52:22

通过 HTTP 协议返回的状态码，您可以知道请求是否成功、是否需要进一步操作、是否有错误发生等情况。本文列举一些常见的状态码，提供了状态码的含义以及相应的解决措施，旨在帮助您自主识别和分析在访问数据库时遇到的异常情况，及时解决问题。

Class 11：Authentication failure

11开头的错误码表明连接数据库认证过程可能是由于凭证不正确或缺乏权限导致失败。

错误码	错误描述	错误含义	解决措施
11100	Datatype privilege failure	使用 AI 套件的接口访问 Base 类数据库，或者使用 Base 类接口访问 AI 套件的数据库，接口使用受限错误。	请使用数据库类型对应的相关接口。

Class 13： Index status exception

13开头的错误码表明构建索引过程存在问题，例如状态无效或者操作不当。

错误码	错误描述	错误含义	解决措施
13000	Collection index state is invalid	Collection 索引状态无效	请等待 Collection 完成索引构建。
13100	Collection index is not ready	Collection 索引未准备就绪	
13101	Index is building	索引正在构建	
13200	Index type does not support delete operations	该索引类型不支持删除操作	索引类型为 FLAT，不支持删除数据。
13201	Index type does not support update operations	该索引类型不支持更新操作	索引类型为 FLAT，不支持更新数据。

Class 14：Data exception

14开头的错误码表明接口输入参数的数据存在问题，例如数据无效、数据类型不对、格式错误、超出取值范围等。

版权所有：腾讯云计算（北京）有限责任公司

错误码	错误描述	错误含义	解决措施
14000	Data exception	数据异常	检查数据的完整性、准确性和一致性。确保数据没有缺失值、重复值或错误值。
14100	Parameters are invalid	参数无效	• 请确认接口所定义的参数类型与取值范围。 • 检查传入的参数或 URL 是否全部正确。
14110	Name is invalid	名称无效	• 请确认名称命名要求。 • 请按照命名规范重新定义名称。
14111	Name is too long	名称字符过长	• 请确认名称的长度要求。 • 按照名称的长度要求重新命名。
14120	The value is out of the allowed range	数值超出允许的范围	请确认参数的取值范围，重新输入数值。
14121	The value does not match the defined type	数值与定义的类型不匹配	请确认的数据类型，重新输入数值。
14122	Negative value error	数值为负数错误	请确认参数的取值范围，重新输入数值。
14130	String value does not match	字符串不匹配	请确认字符串参数要求，重新输入参数。
14131	String length exceeds allowed range	字符串长度超出允许范围	请确认字符串长度要求，重新输入参数。
14132	String length is zero	字符串长度为零的错误	请确认字符串参数要求，重新输入参数。
14200	Empty data exception	空数据异常	请确认输入参数的要求，重新输入参数。
14201	The input parameters are not allowed to be empty.	不允许参数为空数据	请确认接口的参数，按照要求配置接口参数。
14300	Array subscript error	数组下标错误	使用数组时，出现下标越界或者无效的下标值。请确认数组下标的取

版权所有：腾讯云计算（北京）有限责任公司

			值范围，重新配置数组。
14400	Duplicate data encountered while processing batch data	在处理批量数据时出现了重复数据的错误。	请确认可设置批量操作的参数是否配置重复的信息。例如：使用 query 接口参数 documentIds 是否设置了重复的文档 ID。
14500	Object quantity error	对象数量错误	请确认对象数量是否超出限制。
14501	Object quantity error	对象数量超出允许范围	请确认查询过滤的文件数是否超出最大文件数量限制。

Class 15：Syntax Error

15开头的错误码表明输入参数存在配置错误，例如：数据库类型错误、索引类型错误、字段配置不匹配等。

错误码	错误描述	错误含义	解决措施
15000	Syntax error	参数配置错误	输入参数不正确，请确认参数要求，重新输入参数。
15100	Mismatched parameter error	参数不匹配的错误	请确认参数的配置要求，重新输入参数。
15101	Database type mismatch	数据库类型不匹配	• 请确认是否执行 Base 类数据库却使用了 AI 类数据库相关接口。 • 请使用数据类型对应的相关接口。
15111	Index type mismatch	索引类型不匹配	创建集合的参数 indexType 必须是 id、filter 或 vector 类型之一。请确认 indexType 参数设置是否正确。
15112	Index field data type mismatch	索引字段数据类型不匹配	索引有其对应的参数 fieldtype，请根据如下描述确认是否配置正确，重新输入参数。 • 当索引类型是 primary 时，fieldtype 必须是 string。 • 当索引类型是 vector index 时，fieldtype 必须是 vector。

			• 当索引类型为 filter 时，fieldtype 必须是string、uint64 或 array。
15113	Vector index parameters mismatch	向量索引参数不匹配	在向量索引的参数中有错误。例如，HNSW 的参数 M 超出范围。
15114	Mismatched algorithm for vector similarity calculation	向量相似度计算度量算法不匹配	参数 metricType 必须是 IP、L2 或 COSINE 中的一个。
15115	The number of replicas does not match the instance type	副本数数值与实例类型不匹配	每一种实例类型支持的副本数不一样，请根据如下描述确认副本数是否设置正确，重新输入参数。 • 单机版与免费测试版实例：副本仅能为0 ，创建 Collection 设置副本数大于0，则报错。 • 高可用版实例副本数取值范围如下： ○ 两可用区：[1,节点数-1]。 ○ 三可用区：[2,节点数-1]。
15116	Mismatched embedding field	Embedding 字段不匹配	创建 Collection 时配置的 Embedding 的参数不正确。
15117	Mismatch the embedding vector fields	Embedding 向量字段与定义的不匹配	创建 Collection 时定义的存储向量数据的字段与插入数据时，使用的字段名可能不一致。
15131	Alias access denied	别名访问被拒绝	所访问集合的别名使用错误，请确认集合的别名，重新通过别名操作集合。
15141	Search condition mismatch	搜索条件不匹配	请确认接口查询条件相关的参数是否设置正确，数据类型与取值范围是否满足要求。
15142	Read consistency mismatch	读一致性不匹配错误	参数 ReadConsistency 设置的取值不正确，当前仅支持如下一致性要求： • StrongConsistency：强一致性。

			• EventualConsistency：最终一致性。
15171	Dimension mismatch	向量维度不匹配	创建 Collection 时，Embedding 模型对应的向量维度与指定的维度不一致。
15172	Operation type mismatch	操作类型不匹配	请描述异常发生过程，保留异常截图与执行记录，提交工单 咨询腾讯云工程师。
15200	Duplicate object error	重复对象错误	请确认数据库名、集合名、数据 ID 是否存在重复。
15201	Duplicate database name error	数据库名重复错误	创建数据库时，设置的数据库名已经存在。
15202	Duplicate collection error	Collection 重复错误	创建集合时，设置的 Collection 名已经存在。
15203	Duplicate document ID error	Document ID 重复错误	在一次批量更新或插入（upsert）操作中，传入的数据中存在重复的 ID，导致操作发生冲突。
15204	Duplicate Alias error	别名重复错误	创建别名时，别名已存在。
15205	Duplicate embedding model	Embedding 模型重复错误	当前支持的 Embedding 模型，请参见 Embedding 模型信息 。
15300	Undefined object	未定义的对象	请 提交工单 咨询腾讯云工程师。
15301	Undefined database	未定义的数据库	请使用 查看数据库列表 接口查看集群中定义的数据库名，使用正确的数据库名重新访问数据库。
15302	Undefined collection	未定义的集合	请使用 查看集合列表 的接口，查看已定义的集合名，使用正确的集合重新操作集合。
15303	Undefined document I	未定义的文档 ID	请使用 查询文档数据 的接口，确认文档 ID，通过正确的文档 ID 操作数据。
15304	Undefined alias	未定义的别名	请确认所属访问集合的别名，重新使用别名访问集合。

15305	Undefined embedding model	未定义的 Embedding 模型	当前支持的 Embedding 模型，请参见 Embedding 模型信息 。
15306	Undefined index model	未定义的索引	创建集合的参数 indexType 必须是 id、filter 或 vector 类型之一。请确认 indexType 参数设置是否正确。
15400	InValid model	无效的模型	当前支持的 Embedding 模型，请参见 Embedding 模型信息 。

Class 16：External error

16开头的错误码表明数据库应用服务功能异常，例如：Embedding Token 超出限制等。

错误码	错误描述	错误含义	解决措施
16100	Embedding service unavailable	不支持 Embedding 功能或 Embedding 功能未打开	• 当前 Embedding 功能支持地域包含：北京、上海、广州、新加坡。其余地域暂不支持。 • 开通 Embedding 的具体操作，请参见 管理 Embedding 功能 。
16104	Token consumption for Embedding exceeds the range	Embedding 所消耗的 Token 流量超出范围	如果需求更高的 Embedding 速率，请 提交工单 咨询腾讯云工程师评估调整。
16106	Exceeded the free token traffic limit for Embedding	已超过 Embedding 免费限制开通的 Token 流量	免费版实例每月提供500,000个免费Token，请等待下个月1号系统将自动会将免费 Token 数量重置为500,000 个。

Class 17： Internal error

17开头的错误码表明数据库系统内部出现异常，请描述操作过程，保留异常截图与操作记录， 提交工单 咨询腾讯云工程师。

Class 18：Processing exception

18开头的错误码表明数据库处理 DDL/DML/DQL 语句时出现异常，请描述操作过程，保留异常截图与操作记录，提交工单 咨询腾讯云工程师。

Class 19：System overload exception

19开头的错误码表明系统负载过高，内存或磁盘超出限制。

错误码	错误描述	错误含义	解决措施
19000	System overload exception	系统负载过高	• 确认是否有执行消耗资源较大的任务，避免在高峰时期执行，减轻系统负载。 • 清理缓存。
19100	Memory exceeded limit	内存超出限制	• 控制并发连接数。 • 请清理不需要的集合数据减少内存消耗。 • 请 提交工单 升级实例规格。
19200	Disk exceeded limit	磁盘超出限制	请 提交工单 升级实例规格。

Class 20：AI service Internal error

20开头的错误码为 AI 套件内部错误，请描述操作过程，保留异常截图与操作记录，提交工单 咨询腾讯云工程师。

版权所有：腾讯云计算（北京）有限责任公司