向量数据库

SDK 参考

腾讯云

文档目录

SDK 参考
Python SDK
SDK 准备
Python SDK Demo
写入向量数据并检索
使用 Embedding 写入原始文本并检索
使用 AI 套件上传文件并检索
新建 Client
Database 操作
新建 Database
删除 Database
查询 DataBase
Collection 操作
新建 Collection
删除 Collection
查询 Collection 列表
查询指定 Collection
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
插入向量数据
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
删除 Document
更新 Document

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
Java SDK
SDK 准备
Java SDK Demo
写入原始文本并检索（Embedding）
写入向量数据并检索
使用 AI 套件上传文件
新建 Client
Database 操作
新建 Database
删除 Database
查询 Database
Collection 操作
新建 Collection
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
删除 Document
更新数据
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
Go SDK
SDK 准备
GO Demo
写入原始文本并检索（Embedding）
写入向量数据并检索
使用 AI 套件上传文件并检索
连接数据库实例
管理 Database
管理 Base 类 Database
管理 AI 类 Database
查询集群数据库
管理 Collection
新建 Collection
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
更新 Document 数据
删除 Document 数据
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
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
delete
update
DocumentSet
uploadUrl
get
getChunks
query
search
delete
update
Index
rebuild

错误码

SDK 参考
Python SDK
SDK 准备

最近更新时间：2023-12-08 16:15:43

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	• 执行如下命令，可直接安装最新版本。
		`pip install tcvectordb`
		• 执行如下命令，可获取历史版本。
		`pip install tcvectordb==历史版本`
			https://github.com/Tencent/vectordatabase-sdk-python

Python SDK Demo
写入向量数据并检索

最近更新时间：2023-10-13 11:55:02

腾讯云向量数据库（Tencent Cloud VectorDB）支持直接写入向量数据。本文给出通过 Python SDK 写入或更新向量数据，并进行精确查询与相似度检索的完整示例，便于您更加高效地管理和使用向量数据。

```
# example.py demonstrates the basic operations of tcvectordb, a Python SDK of
tencent cloud vectordb.
# 1. connect to vectordb and create database and collection
# 2. upsert data
# 3. query and search data
# 4. drop collection and database
import json
import time

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

# disable/enable http request log print
tcvectordb.debug.DebugEnable = False


def print_object(obj):
    for elem in obj:
        if hasattr(elem, '__dict__'):
            print(json.dumps(vars(elem), indent=2))
        else:
            print(json.dumps(elem, indent=2))


class TestVDB:

    def __init__(self, url: str, username: str, key: str, timeout: int = 30):
        """
        初始化客户端
        """
        # 创建客户端时可以指定 read_consistency，后续调用 sdk 接口的 read_consistency 将
延用该值
```

```
        self._client = tcvectordb.VectorDBClient(url=url, username=username, key=key,
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=timeout)

    def clear(self):
        db = self._client.database('book')
        db.drop_database('book')

    def delete_and_drop(self):
        db = self._client.database('book')

        # 删除collection，删除collection的同时，其中的数据也将被全部删除
        db.drop_collection('book_segments')

        # 删除db，db下的所有collection都将被删除
        db.drop_database('book')

    def create_db_and_collection(self):
        database = 'book'
        coll_name = 'book_segments'
        coll_alias = 'book_segments_alias'

        # 创建DB--'book'
        db = self._client.create_database(database)

        database_list = self._client.list_databases()
        for db_item in database_list:
            print(db_item.database_name)

        # 创建 Collection

        # 第一步，设计索引（不是设计 Collection 的结构）
        # 1.【重要的事】向量对应的文本字段不要建立索引，会浪费较大的内存，并且没有任何作用。
        # 2.【必须的索引】：主键id、向量字段 vector 这两个字段目前是固定且必须的，参考下面的例子；
        # 3.【其他索引】：检索时需作为条件查询的字段，比如要按书籍的作者进行过滤，这个时候 author 字段就需要建立索引，
        #     否则无法在查询的时候对 author 字段进行过滤，不需要过滤的字段无需加索引，会浪费内存；
        # 4. 向量数据库支持动态 Schema，写入数据时可以写入任何字段，无需提前定义，类似 MongoDB.
        # 5. 例子中创建一个书籍片段的索引，例如书籍片段的信息包括 {id, vector, segment, bookName, author, page},
```

```
        #     id 为主键需要全局唯一，segment 为文本片段, vector 字段需要建立向量索引，假如
我们在查询的时候要查询指定书籍
        #     名称的内容，这个时候需要对 bookName 建立索引，其他字段没有条件查询的需要，
无需建立索引。
        index = Index()
        index.add(VectorIndex('vector', 3, IndexType.HNSW, MetricType.COSINE,
HNSWParams(m=16, efconstruction=200)))
        index.add(FilterIndex('id', FieldType.String, IndexType.PRIMARY_KEY))
        index.add(FilterIndex('bookName', FieldType.String, IndexType.FILTER))
        index.add(FilterIndex('page', FieldType.Uint64, IndexType.FILTER))

        # 第二步：创建 Collection
        db.create_collection(
            name=coll_name,
            shard=3,
            replicas=2,
            description='test collection',
            index=index,
            embedding=None,
            timeout=20
        )

        # 列出所有 Collection
        coll_list = db.list_collections()
        print_object(coll_list)

        # 设置 Collection 的 alias
        db.set_alias(coll_name, coll_alias)

        # 查看 Collection 信息
        coll_res = db.describe_collection(coll_name)
        print(vars(coll_res))

        # 删除 Collection 的 alias
        db.delete_alias(coll_alias)

    def upsert_data(self):
        # 获取 Collection 对象
        db = self._client.database('book')
        coll = db.collection('book_segments')

        # upsert 写入数据，可能会有一定延迟
        # 1. 支持动态 Schema，除了 id、vector 字段必须写入，可以写入其他任意字段；
        # 2. upsert 会执行覆盖写，若文档id已存在，则新数据会直接覆盖原有数据(删除原有数
据，再插入新数据)
```

```
        document_list = [
            Document(id='0001',
                     vector=[0.2123, 0.21, 0.213],
                     bookName='西游记',
                     author='吴承恩',
                     page=21,
                     segment='富贵功名，前缘分定，为人切莫欺心。'),
            Document(id='0002',
                     vector=[0.2123, 0.22, 0.213],
                     bookName='西游记',
                     author='吴承恩',
                     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。'),
            Document(id='0003',
                     vector=[0.2123, 0.23, 0.213],
                     bookName='三国演义',
                     author='罗贯中',
                     page=23,
                     segment='细作探知这个消息，飞报吕布。'),
            Document(id='0004',
                     vector=[0.2123, 0.24, 0.213],
                     bookName='三国演义',
                     author='罗贯中',
                     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。'),
            Document(id='0005',
                     vector=[0.2123, 0.25, 0.213],
                     bookName='三国演义',
                     author='罗贯中',
                     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。'),

        ]
        coll.upsert(documents=document_list)
        time.sleep(1)

    def query_data(self):
        # 获取 Collection 对象
        db = self._client.database('book')
        coll = db.collection('book_segments')

        # 查询
```

```
        # 1. query 用于查询数据
        # 2. 可以通过传入主键 id 列表或 filter 实现过滤数据的目的
        # 3. 如果没有主键 id 列表和 filter 则必须传入 limit 和 offset，类似 scan 的数据扫描功能
        # 4. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回

        document_ids = ["0001", "0002", "0003", "0004", "0005"]
        filter_param = Filter('bookName="三国演义"')
        output_fields_param = ["id", "bookName"]
        res = coll.query(document_ids=document_ids, retrieve_vector=True, limit=2,
offset=1,
                         filter=filter_param, output_fields=output_fields_param)
        print_object(res)

        # searchById
        # 1. searchById 提供按 id 搜索的能力
        # 1. search 提供按照 vector 搜索的能力
        # 2. 支持通过 filter 过滤数据
        # 3. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
        # 4. limit 用于限制每个单元搜索条件的条数，如 vector 传入三组向量，limit 为 3，则
limit 限制的是每组向量返回 top 3 的相似度向量

        # 根据主键 id 查找 Top K 个相似性结果，向量数据库会根据ID 查找对应的向量，再根据向
量进行TOP K 相似性检索
        res = coll.searchById(
            document_ids=['0003'],
            params=SearchParams(ef=200),  # 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
            retrieve_vector=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
            filter=filter_param  # 过滤获取到结果
        )
        print_object(res)

        # search
        # 1. search 提供按照 vector 搜索的能力
        # 其他选项类似 search 接口

        # 批量相似性查询，根据指定的多个向量查找多个 Top K 个相似性结果
        res = coll.search(
            vectors=[[0.3123, 0.43, 0.213], [0.233, 0.12, 0.97]],  # 指定检索向量，最多指定
20个
            params=SearchParams(ef=200),  # 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
```

```
            retrieve_vector=False, # 是否需要返回向量字段，False：不返回，True：返回
            limit=10  # 指定 Top K 的 K 值
        )
        # 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
        print_object(res)

    def update_and_delete(self):
        # 获取 Collection 对象
        db = self._client.database('book')
        coll = db.collection('book_segments')

        # update
        # 1. update 提供基于 [主键查询] 和 [Filter 过滤] 的部分字段更新或者非索引字段新增

        # filter 限制仅会更新 id = "0003"
        document_ids = ["0001", "0003"]
        filter_param = Filter('bookName="三国演义"')
        update_doc = Document(page=24)
        coll.update(data=update_doc, document_ids=document_ids, filter=filter_param)

        # delete
        # 1. delete 提供基于 [主键查询] 和 [Filter 过滤] 的数据删除能力
        # 2. 删除功能会受限于 collection 的索引类型，部分索引类型不支持删除操作

        # filter 限制只会删除 id="0001" 成功
        filter_param = Filter('bookName="西游记"')
        coll.delete(document_ids=document_ids, filter=filter_param)

        # rebuild_index
        # 索引重建，重建期间不支持写入
        coll.rebuild_index()

        # truncate_collection
        # 清空 Collection
        time.sleep(5)
        truncate_res = db.truncate_collection('book_segments')
        print_object(truncate_res)


if __name__ == '__main__':
    test_vdb = TestVDB('vdb http url or ip and post', key='key get from web console', username='vdb username')
    # test_vdb = TestVDB('http://127.0.0.1:8100', key='vdb-key', username='root')
    test_vdb.clear()  # 测试前清理环境
```

```
    test_vdb.create_db_and_collection()
    test_vdb.upsert_data()
    test_vdb.query_data()
    test_vdb.update_and_delete()
    test_vdb.delete_and_drop()
```

使用 Embedding 写入原始文本并检索

最近更新时间：2023-10-13 11:55:02

腾讯云向量数据库（Tencent Cloud VectorDB）目前已支持文本 Embedding 模型，能够覆盖多种主流语言的向量转换。本文给出通过 Python SDK 写入或更新原始文本，并进行精确查询或相似度检索的完整示例，便于您更加高效地管理和使用向量数据。

```
# example.py demonstrates the basic operations of tcvectordb, a Python SDK of
tencent cloud vectordb.
# 1. connect to vectordb and create database and collection
# 2. upsert data
# 3. query and search data
# 4. drop collection and database
import json
import time

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

# disable/enable http request log print
tcvectordb.debug.DebugEnable = False


def print_object(obj):
    for elem in obj:
        if hasattr(elem, '__dict__'):
            print(json.dumps(vars(elem), indent=2))
        else:
            print(json.dumps(elem, indent=2))


class TestVDB:

    def __init__(self, url: str, username: str, key: str, timeout: int = 30):
        """
        初始化客户端
        """
```

```
        # 创建客户端时可以指定 read_consistency，后续调用 sdk 接口的 read_consistency 将
延用该值
        self._client = tcvectordb.VectorDBClient(url=url, username=username, key=key,

read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=timeout)

    def clear(self):
        db = self._client.database('book')
        db.drop_database('book')

    def delete_and_drop(self):
        db = self._client.database('book')

        # 删除collection，删除collection的同时，其中的数据也将被全部删除
        db.drop_collection('book_segments')

        # 删除db，db下的所有collection都将被删除
        db.drop_database('book')

    def create_db_and_collection(self):
        database = 'book'
        coll_embedding_name = 'book_segments'
        coll_alias = 'book_segments_alias'

        # 创建DB--'book'
        db = self._client.create_database(database)

        database_list = self._client.list_databases()
        for db_item in database_list:
            print(db_item.database_name)

        # 新建 Collection
        # 第一步，设计索引（不是设计表格的结构）
        # 1.【重要的事】向量对应的文本字段不要建立索引，会浪费较大的内存，并且没有任何作
用。
        # 2.【必须的索引】：主键 id、向量字段 vector 这两个字段目前是固定且必须的，参考下
面的例子；
        # 3.【其他索引】：检索时需作为条件查询的字段，比如要按书籍的作者进行过滤，这个时
候author字段就需要建立索引，
        #    否则无法在查询的时候对 author 字段进行过滤，不需要过滤的字段无需加索引，会浪
费内存；
        # 4. 向量数据库支持动态 Schema，写入数据时可以写入任何字段，无需提前定义，类似
MongoDB.
        # 5. 例子中创建一个书籍片段的索引，例如书籍片段的信息包括 {id, vector, segment,
bookName, page},
```

```
        #     id 为主键需要全局唯一，segment 为文本片段, vector 为 segment 的向量，vector
字段需要建立向量索引，假如我们在查询的时候要查询指定书籍
        #     名称的内容，这个时候需要对bookName建立索引，其他字段没有条件查询的需要，无
需建立索引。
        # 6. 创建带 Embedding 的 collection 需要保证设置的 vector 索引的维度和
Embedding 所用模型生成向量维度一致，模型及维度关系：
        #     ----------------------------------------------
        #           bge-base-zh              | 768
        #           m3e-base                 | 768
        #           text2vec-large-chinese   | 1024
        #           e5-large-v2              | 1024
        #           multilingual-e5-base     | 768
        #     ----------------------------------------------
        index = Index()
        index.add(VectorIndex('vector', 768, IndexType.HNSW, MetricType.COSINE,
HNSWParams(m=16, efconstruction=200)))
        index.add(FilterIndex('id', FieldType.String, IndexType.PRIMARY_KEY))
        index.add(FilterIndex('bookName', FieldType.String, IndexType.FILTER))
        index.add(FilterIndex('author', FieldType.String, IndexType.FILTER))

        ebd = Embedding(vector_field='vector', field='text',
model=EmbeddingModel.BGE_BASE_ZH)

        # 第二步：创建 Collection
        # 创建支持 Embedding 的 Collection
        db.create_collection(
            name=coll_embedding_name,
            shard=3,
            replicas=2,
            description='test embedding collection',
            index=index,
            embedding=ebd,
            timeout=20
        )

        # 列出所有 Collection
        coll_list = db.list_collections()
        print_object(coll_list)

        # 设置 Collection 的 alias
        db.set_alias(coll_embedding_name, coll_alias)

        # 查看 Collection 信息
        coll_res = db.describe_collection(coll_embedding_name)
        print(vars(coll_res))
```

```
        # 删除 Collection 的 alias
        db.delete_alias(coll_alias)

    def upsert_data(self):
        # 获取 Collection 对象
        db = self._client.database('book')
        coll = db.collection('book_segments')

        # upsert 写入数据，可能会有一定延迟
        # 1. 支持动态 Schema，除了 id、vector 字段必须写入，可以写入其他任意字段；
        # 2. upsert 会执行覆盖写，若文档 id 已存在，则新数据会直接覆盖原有数据(删除原有数据，再插入新数据)

        document_list = [
            Document(id='0001',
                     text='富贵功名，前缘分定，为人切莫欺心。',
                     bookName='西游记',
                     author='吴承恩',
                     page=21,
                     segment='富贵功名，前缘分定，为人切莫欺心。'),
            Document(id='0002',
                     text='正大光明，忠良善果弥深。些些狂妄天加谴，眼前不遇待时临。',
                     bookName='西游记',
                     author='吴承恩',
                     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。'),
            Document(id='0003',
                     text='细作探知这个消息，飞报吕布。',
                     bookName='三国演义',
                     author='罗贯中',
                     page=23,
                     segment='细作探知这个消息，飞报吕布。'),
            Document(id='0004',
                     text='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。',
                     bookName='三国演义',
                     author='罗贯中',
                     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。'),
            Document(id='0005',
                     text='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。',
                     bookName='三国演义',
```

```
                author='罗贯中',
                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。'),

        ]
        coll.upsert(documents=document_list)
        time.sleep(1)

    def query_data(self):
        # 获取 Collection 对象
        db = self._client.database('book')
        coll = db.collection('book_segments')

        # 查询
        # 1. query 用于查询数据
        # 2. 可以通过传入主键 id 列表或 filter 实现过滤数据的目的
        # 3. 如果没有主键 id 列表和 filter 则必须传入 limit 和 offset，类似 scan 的数据扫描功能
        # 4. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回

        document_ids = ["0001", "0002", "0003", "0004", "0005"]
        filter_param = Filter('bookName="三国演义"')
        output_fields_param = ["id", "bookName"]
        res = coll.query(document_ids=document_ids, retrieve_vector=False, limit=2,
offset=1,
                         filter=filter_param, output_fields=output_fields_param)
        print_object(res)

        # searchById
        # 1. searchById 提供按 id 搜索的能力
        # 1. search 提供按照 vector 搜索的能力
        # 2. 支持通过 filter 过滤数据
        # 3. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
        # 4. limit 用于限制每个单元搜索条件的条数，如 vector 传入三组向量，limit 为 3，则
limit 限制的是每组向量返回 top 3 的相似度向量

        # 根据主键 id 查找 Top K 个相似性结果，向量数据库会根据ID 查找对应的向量，再根据向
量进行TOP K 相似性检索
        res = coll.searchById(
            document_ids=['0003'],
            params=SearchParams(ef=200), # 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
            retrieve_vector=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
            filter=filter_param  # 过滤获取到结果
        )
        print_object(res)

        # search
        # 1. search 提供按照 vector 搜索的能力
        # 其他选项类似 search 接口

        # 批量相似性查询，根据指定的多个向量查找多个 Top K 个相似性结果
        query_all = coll.query(document_ids=document_ids, retrieve_vector=True,
limit=2)
        query_document_vector = [x.get("vector") for x in query_all]
        res = coll.search(
            vectors=query_document_vector,  # 指定检索向量，最多指定20个
            params=SearchParams(ef=200),  # 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
            retrieve_vector=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
            filter=filter_param  # 对搜索结果进行过滤
        )
        # 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
        print_object(res)

        # 通过 embedding 文本搜索
        # 1. searchByText 提供基于 embedding 文本的搜索能力，会先将 embedding 内容做
Embedding 然后进行按向量搜索
        # 其他选项类似 search 接口

        # searchByText 返回类型为 Dict，接口查询过程中 embedding 可能会出现截断，如发生
截断将会返回响应 warn 信息，如需确认是否截断可以
        # 使用 "warning" 作为 key 从 Dict 结果中获取警告信息，查询结果可以通过
"documents" 作为 key 从 Dict 结果中获取
        embeddingItems = ['细作探知这个消息，飞报吕布。']
        search_by_text_res = coll.searchByText(embeddingItems=embeddingItems,
                                               params=SearchParams(ef=200))
        print_object(search_by_text_res.get('documents'))

    def update_and_delete(self):
        # 获取 Collection 对象
        db = self._client.database('book')
        coll = db.collection('book_segments')

        # update
```

```
        # 1. update 提供基于 [主键查询] 和 [Filter 过滤] 的部分字段更新或者非索引字段新增

        # filter 限制仅会更新 id = "0003"
        document_ids = ["0001", "0003"]
        filter_param = Filter('bookName="三国演义"')
        update_doc = Document(page=28)
        coll.update(data=update_doc, document_ids=document_ids, filter=filter_param)

        # delete
        # 1. delete 提供基于 [主键查询] 和 [Filter 过滤] 的数据删除能力
        # 2. 删除功能会受限于 collection 的索引类型，部分索引类型不支持删除操作

        # filter 限制只会删除 id="0001" 成功
        filter_param = Filter('bookName="西游记"')
        coll.delete(document_ids=document_ids, filter=filter_param)

        # rebuild_index
        # 索引重建，重建期间不支持写入
        coll.rebuild_index()

        # truncate_collection
        # 清空 Collection
        time.sleep(5)
        truncate_res = db.truncate_collection('book_segments')
        print_object(truncate_res)


if __name__ == '__main__':
    test_vdb = TestVDB('vdb http url or ip and post', key='key get from web console',
username='vdb username')
    # test_vdb = TestVDB('http://127.0.0.1:8100', key='vdb-key', username='root')
    test_vdb.clear()  # 测试前清理环境
    test_vdb.create_db_and_collection()
    test_vdb.upsert_data()
    test_vdb.query_data()
    test_vdb.update_and_delete()
    test_vdb.delete_and_drop()
```

使用 AI 套件上传文件并检索

最近更新时间：2023-12-12 09:51:42

AI 套件是腾讯云向量数据库（Tencent Cloud VectorDB）提供的一站式文档检索解决方案，包含自动化文档解析、信息补充、向量化、内容检索等能力。用户仅需上传原始文档，数分钟内即可快速构建专属知识库，大幅提高知识接入效率。本文介绍通过 Python SDK 操作 AI 类数据库上传文件并进行内容相似度检索的完整示例。

```
import json
import time
from typing import Optional

import tcvectordb
from tcvectordb import exceptions
from tcvectordb.model.ai_database import AIDatabase
from tcvectordb.model.collection_view import Embedding, SplitterProcess, Language,
CollectionView
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.document_set import DocumentSet, Rerank
from tcvectordb.model.enum import FieldType, IndexType, ReadConsistency
from tcvectordb.model.index import Index, FilterIndex

#create vdb client
vdbclient = tcvectordb.VectorDBClient(url="your vdb url",
                        key="your vdb api_key",
                        username="root")

def vdbInit():
    # create database
    db = vdbclient.create_ai_database("test_db")
    # create collectionView
    collView = db.create_collection_view("test_collView")
    # upload and split text
    collView.load_and_split_text(local_file_path="/docs/fileName.md")
    print('upload file sucess')

def knowledgeSearch(query):
    db = vdbclient.database('test_db')
    collView = db.collection_view('test_collView')
    doc_list = collView.search(
            content=query,
            limit=3
        )
    knowledge = "以下是根据输入问题检索到的知识内容：\n"
```

```
    knowledge_id = 1
    for item in doc_list:
        knowledge += ("知识内容%s:\n%s"%(knowledge_id,item.data.text))
        knowledge_id += 1
    return(knowledge)

if __name__ == "__main__":
    vdbInit()
    query = input("请输入查询内容：")
    print("==========================================")
    print(knowledgeSearch(query))
```

新建 Client

最近更新时间：2024-05-11 15:02:12

功能介绍

VectorDBClient() 用于创建一个向量数据库的客户端对象，用于与向量数据库服务器连接并进行数据交互。

SDK 准备

SDK 下载以及安装方式，请参见 SDK 准备 。

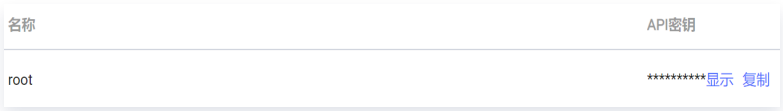
请求接口

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency

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

请求参数

参数名称	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	是	数据库当前仅支持 root 账号。

	向量数据库服务端的账号。		
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。
timeout	请求超时时间。	是	• 单位：秒。 • 默认值：5。 • 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。

Database 操作
新建 Database

最近更新时间：2023-12-08 16:15:44

功能介绍

新建 Database，用于存储向量数据、原始文本或上传文件。

- **create_database()** 用于创建一个 Base 类的向量数据库，用于直接存储向量数据或写入文本 Embedding 向量化存储。
- **create_ai_database()** 用于创建一个 AI 类向量数据库，用于直接上传并存储文件，不直接操作向量数据与索引。

> **说明：**
> 创建数据库之前，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 。

请求示例

基于 VectorDBClient() 创建的客户端对象，创建数据库。

创建 Base 类数据库

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

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

print(db.database_name)
```

创建 AI 类数据库

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

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# create a database
db = client.create_ai_database(database_name='db-test-ai')

print(db.database_name)
```

请求参数

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：2023-12-08 16:15:44

功能介绍

删除 Database，清理过期或无用的数据，重置数据库内容，以提高性能和减少存储空间。

- **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。

接口约束

> **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。

请求示例

删除 Base 类数据库

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

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# drop a database
client.drop_database(database_name='db-test')
```

删除 AI 类数据库

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

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

```
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# drop a database
client.drop_ai_database(database_name='db-test-ai')
```

执行成功，返回如下信息。

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

请求参数

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。

返回参数

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：2023-12-08 16:15:44

功能介绍

查询 Database 列表，方便及时了解当前系统中存在的数据库。**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。

请求示例

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

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********', 
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# 创建数据库
client.create_database('db-test')
client.create_database('db-test-ai')
# list databases
db_list = client.list_databases()

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

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
```

请求参数

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。

返回参数

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。

Collection 操作
新建 Collection

最近更新时间：2024-05-15 10:17:31

功能介绍

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

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

请求示例

创建 Base 类 Collection 存储原始文本 Embedding

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，用于写入原始文本。Embedding 模型自动将原始文本进行向量化。

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
EmbeddingModel
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams
from tcvectordb.model.collection import Embedding

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

```
        FilterIndex(name='author', field_type=FieldType.String,
index_type=IndexType.FILTER),
        FilterIndex(name='tags', field_type=FieldType.String,
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=EmbeddingModel.BGE_BASE_ZH)

# create a collection
coll = db.create_collection(
        name='book-emb',
        shard=1,
        replicas=0,
        description='this is a collection of test embedding',
        embedding=ebd,
        index=index
    )
print(vars(coll))
```

创建 Base 类数据库存储向量数据

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，用于写入 3 维向量数据。

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

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

db = client.database('db-test')
# -- 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),
            VectorIndex(name='vector', dimension=3, index_type=IndexType.HNSW,
                        metric_type=MetricType.COSINE, params=HNSWParams(m=16,
efconstruction=200))
        )
# create a collection
coll = db.create_collection(
            name='book-vector',
            shard=1,
            replicas=2,
            description='this is a collection of test embedding',
            index=index
        )
print(vars(coll))
```

请求参数

> **注意：**
> - 若直接存储向量数据，创建集合时，需指定 Base 类的数据库，配置集合与索引相关参数。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 。
> - 若写入原始文本，需使用 Embedding 功能向量化存储，则在创建 Collection 时，需指定 Base 类的数据库，配置 Embedding 相关参数。相关介绍，请参见 Embedding 介绍 。

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）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 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。
	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，可将该标量字段设置为 **IndexType.FILTER**，在后续检索、更新、删除数据时，可对该字段设置 Filter 条件表达式，综合检索所需的文档。具体信息，请参见 混合检索 。
Vector Index	**name**	是	指定向量索引字段名，固定为 **vector**。
	index_type	是	指定向量索引字段的索引类型。当前所支持的索引类型，及具体索引方式，请参见设计模型中的 Index。 • **FLAT**：暴力检索，召回率100%，但检索效率低。 • **HNSW**：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。具体测试数据，请参见性能白皮书的测试结果。 • **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 upsert() 插入数据时，将参数 build_index 设置为 false，表示在插入数据之后需要通过 rebuild_index() 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]。 • 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：** 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。

	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。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 。该参数仅 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]。

Embedding 参数

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	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_BASE_ZH：适用中文，768维，推荐使用 • 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。

Collection 参数

参数	参数含义	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]。

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 colletion 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** 。
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：2023-12-08 16:15:44

功能介绍

drop_collection() 用于删除已创建的 Collection。

接口约束

> **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。

请求示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va*******************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# 指定 Base 类数据库
db = client.database('db-test')
# 删除 Base 类数据库下的指定集合
db.drop_collection(name='book-vector')
```

请求参数

参数名	是否必选	参数含义	配置方法
name	是	所需删除的 Collection 名称。	• 请使用 list_collections() 查找需删除的集合。
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。

返回消息

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

返回参数

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：2024-05-15 10:17:31

功能介绍

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

请求示例

```
import tcvectordb
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************', timeout=30)
# 指定 Base 类数据库
db = client.database('db-test')
# 查询 Base 类数据库下的集合列表
coll_list = db.list_collections()
for col in coll_list:
    print(vars(col))
```

请求参数

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。

返回参数

```
{
  "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": ""
    }
```

```
}

{
  "database": "db-test",
  "collection": "book-vector",
  "replicaNum": 2,
  "shardNum": 1,
  "description": "this is a collection of test embedding",
  "indexes": [
    {
      "fieldName": "author",
      "fieldType": "string",
      "indexType": "filter"
    },
    {
      "fieldName": "id",
      "fieldType": "string",
      "indexType": "primaryKey"
    },
    {
      "fieldName": "vector",
      "fieldType": "vector",
      "indexType": "HNSW",
      "dimension": 3,
      "metricType": "COSINE",
      "params": {
        "M": 16,
        "efConstruction": 200
      },
      "indexedCount": 3
    },
    {
      "fieldName": "bookName",
      "fieldType": "string",
      "indexType": "filter"
    }
  ],
  "embedding": {
    "status": "disabled"
  },
  "documentCount": 3,
  "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。当前所支持的索引类型及具体索引方式，请参见设计模型中的 Index 。
		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**。具体信息，请参见 混合检索 。
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。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training data**：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 index**：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：2024-05-15 10:17:31

功能介绍

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

请求示例

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

# create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# 指定 Base 类数据库
db = client.database('db-test')
# 查询 Base 类数据库下的集合
res = db.describe_collection('book-emb')
print(vars(res))
```

请求参数

参数名	是否必选	参数含义	配置方法
name	是	指定所需查询的 Collection 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。

返回参数

```
{
  "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。当前所支持的索引类型及具体索引方式，请参见设计模型中的 Index 。
		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**。具体信息，请参见 混合检索 。
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。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training data**：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 index**：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。
		startTim	重建索引开始的时间。

		e	

清空 Collection

最近更新时间：2023-12-08 16:15:45

功能介绍

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。

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。

请求示例

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

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

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

db.truncate_collection(collection_name='book-emb')
```

输出信息，如下所示。

```
{"affectedCount": 1}
```

请求参数

参数	是否必选	参数含义	配置方法及要求
collection_name	是	指定需清空数据的	使用 list_collections() 获取指定数据库名下的 Collection 列表，复制需清空数据的集合名。

		Collection 名。	

返回参数

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：2024-05-13 17:48:31

功能介绍

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

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', 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,
                    ),
                    timeout=100)
print(vars(coll_view))
```

请求参数

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		取值如下所示： • ZH：中文 • EN：英文 • MULTI：多语言
		enable_words_embedding		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。
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` 。

返回消息

输出信息，如下所示。

```
{
 "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"
  }
 ]
}
```

删除 CollectionView

最近更新时间：2023-12-08 17:20:02

功能介绍

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

接口约束

> **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。

请求示例

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

# 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')
```

请求参数

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。

返回消息

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

返回参数

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：2024-01-16 10:50:51

功能介绍

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

请求示例

```
import tcvectordb
from tcvectordb.model.index import Index, VectorIndex, FilterIndex, HNSWParams
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency
#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************', timeout=30)

# 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))
```

请求参数

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。

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
  },
  "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"
  ]
}
```


返回参数

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

e			
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。
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 CollectionView

最近更新时间：2024-01-16 10:42:02

功能介绍

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

请求示例

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

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

# 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))
```

请求参数

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
  },
  "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"
  ]
}
```

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。创建名别，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。
		unInde	未处理的文件数量。

		xedDocumentSets	
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。
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 。

清空 CollectionView 数据

最近更新时间：2023-12-08 17:20:02

功能介绍

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。

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。

请求示例

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

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', 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')
```

输出信息，如下所示。

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

请求参数

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。

返回参数

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：2023-12-08 17:20:02

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。

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 **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。

请求示例

为 Collection 设置别名

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

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

# set alias
db.set_alias(collection_name='book-emb', collection_alias='alias-book-emb')
```

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]。

执行成功，返回消息，如下所示。

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

为 CollectionView 设置别名

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

# Specify the database name
db = client.database('db-test-ai')

# set alias
db.set_alias(collection_view_name='coll-ai-files', alias='alias-sdk-test')
```

参数	是否必选	参数含义	配置方法及要求
collection_view_name	是	指定需创建别名的	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。

		CollectionView 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]。

执行成功，返回信息，如下所示。

```
{'code': 0, 'msg': 'requestId: b86539d4710d9c17d7102a6a*******',
'affectedCount': 1}
```

返回参数

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：2023-12-08 16:15:45

功能介绍

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

请求示例

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

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

# set alias
db.delete_alias(alias='alias-book-emb')
```

输出信息，如下所示。

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

请求参数

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]。

返回参数

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
插入向量数据

最近更新时间：2023-12-08 16:15:45

功能介绍

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

接口约束

> **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。

请求示例

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

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# 指定写入数据的数据库与集合
db = client.database('db-test')
coll = db.collection('book-vector')

# 写入数据。
# 参数 build_index 为 True,指写入数据同时重新创建索引。
```

```
res = coll.upsert(
        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
    )
```

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.collection import Embedding, UpdateQuery
from tcvectordb.model.document import Document, Filter, SearchParams
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
EmbeddingModel
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams, IVFFLATParams
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency
#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# 指定写入原始文本的数据库与集合
db = client.database('db-test')
coll = db.collection('book-emb')

# 写入数据。
# 参数 build_index 为 True,指写入数据同时重新创建索引。
res = coll.upsert(
```

```
        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),
            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),
            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)
        ],
        build_index=True
    )
```

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
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，请务必使用32位浮点数存储向量数据。 **说明：** 如果业务无需使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能做向量化，则配置该参数，写入向量数据，而无需配置 Embedding 参数 **text**（创建 Collection 时，Embedding 参数 **field** 对应指定的文本字段名，示例中为 text）。
		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。
		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。

精确查询

最近更新时间：2024-01-19 11:31:11

功能介绍

基于精确匹配的查询方式，query() 用于精确查找与查询条件完全匹配的向量，具体支持如下功能。

- 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。

请求示例

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 query()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.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

db = client.database('db-test')
coll = db.collection('book-vector')

# Set filter
filter_param=Filter(Filter.In("bookName",["三国演义", "西游记"]) and Filter.Include
("tags",["曹操", "诸葛亮"]))
# query
doc_list = coll.query(document_ids=['0001','0002','0003'], retrieve_vector=True,
filter=filter_param, limit=3, offset=0, output_fields=['bookName','author'])

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

检索结果，如下所示。

```
{'id': '0001', 'vector': [0.21230000257492065, 0.23000000417232513,
0.21299999952316284], 'bookName': '三国演义', 'author': '罗贯中'}
```

请求参数

参数	是否必选	参数含义	参数配置
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（=）、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。 ○ **unit64**：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 Document 数量。	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 **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。
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。

输出参数

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：2024-05-15 10:12:02

功能介绍

基于相似度匹配的查询方式，**search()** 接口用于在 Base 类数据库中查找与给定向量数据相似的 Top K 个数据，并支持搭配自定义的标量字段的 Filter 表达式一并进行相似性检索。

请求示例

如下示例，基于 create_database() 创建数据库db-test，基于 create_collection() 创建集合 **book-vector**，通过 search() 进行向量检索。

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

db = client.database('db-test')
coll = db.collection('book-vector')

# 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 = coll.search(
        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"])),
        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)
```

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。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。

```
{'id': '0001', 'vector': [0.21230000257492065, 0.23000000417232513,
0.21299999952316284], 'score': 0.971423, 'bookName': '三国演义', 'author': '罗贯中'}
{'id': '0002', 'vector': [0.21230000257492065, 0.2199999988079071,
0.21299999952316284], 'score': 0.966884, 'bookName': '西游记', 'author': '吴承恩'}
1
{'id': '0001', 'vector': [0.21230000257492065, 0.23000000417232513,
0.21299999952316284], 'score': 0.978463, 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'vector': [0.21230000257492065, 0.2199999988079071,
0.21299999952316284], 'score': 0.974783, 'author': '吴承恩', 'bookName': '西游记'}
2
{'id': '0001', 'vector': [0.21230000257492065, 0.23000000417232513,
0.21299999952316284], 'score': 0.986069, 'bookName': '三国演义', 'author': '罗贯中'}
{'id': '0002', 'vector': [0.21230000257492065, 0.2199999988079071,
0.21299999952316284], 'score': 0.985201, 'bookName': '西游记', 'author': '吴承恩'}
```

请求参数

参数名	是否必选	参数含义	配置方法

vectors	是	表示要查询的向量列表。	数组元素数量最大为20。
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（=）、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。 ○ unit64：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')。
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。
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。
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。

输出参数

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 Doc ID 相似度检索

最近更新时间：2024-01-18 17:43:01

功能介绍

基于相似度匹配的查询方式，**searchById()** 用于在 Base 类数据中根据指定的 Document id 进行相似度查询，并支持搭配自定义的标量字段的 Filter 表达式一并进行相似度检索，返回指定的 Top K 个最相似的数据。

请求示例

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

db = client.database('db-test')
coll = db.collection('book-vector')
# 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 = coll.searchById(
            document_ids=['0001','0002'],
            filter=Filter(Filter.In("bookName",["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)
```

请求参数

参数	是否必选	参数含义	配置方法
document_ids	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。

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（=）、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。
			  ○ **unit64**：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')`。

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 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。
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。

输出参数

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', 'vector': [0.21230000257492065, 0.23000000417232513,
0.21299999952316284], 'score': 1.0000001192092896, 'author': '罗贯中', 'bookName':
'三国演义'}
{'id': '0002', 'vector': [0.21230000257492065, 0.2199999988079071,
0.21299999952316284], 'score': 0.9997729659080505, 'author': '吴承恩', 'bookName':
'西游记'}
1
```

{'id': '0002', 'vector': [0.21230000257492065, 0.2199999988079071, 0.21299999952316284], 'score': 0.9997580051422119, 'author': '吴承恩', 'bookName': '西游记'}
{'id': '0001', 'vector': [0.21230000257492065, 0.23000000417232513, 0.21299999952316284], 'score': 0.9990617632865906, 'bookName': '三国演义', 'author': '罗贯中'}

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：2024-01-18 17:43:01

功能介绍

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()** 接口输入原始文本进行相似度查询，并支持搭配自定义的标量字段的 Filter 表达式一并进行相似度检索，返回指定的 Top K 个最相似的文本信息。

请求示例

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-emb**，通过 searchByText() 检索与 **embeddingItems** 参数的文本信息相度读最高，且满足 Filter 表达式的文档。

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
EmbeddingModel, 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.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

db = client.database('db-test')
coll = db.collection('book-emb')

# 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 = coll.searchByText(
            embeddingItems=['天下大势，分久必合，合久必分'],
            filter=Filter(Filter.In("bookName",["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.get("documents")):
            print(i)
            for doc in docs:
                print(doc)
```

请求参数

参数	是否必选	参数含义	配置方法
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 。

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。
			  ○ unit64：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')`。

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。
output_fields	否	配置需返回的字段。	指定需要输出的字段。若不设置，将返回所有字段。 **说明：** **retrieve_vector** 和 **output_fields** 只要有其中一个配置输出向量字段即可输出 vector。

输出参数

说明：

- 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': 0.9792741537094116, 'author': '罗贯中', 'bookName': '三国演义'}
{'id': '0002', 'score': 0.7909858226776123, 'bookName': '西游记', 'author': '吴承恩'}
```

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。

删除 Document

最近更新时间：2024-01-18 17:43:01

功能介绍

delete() 接口用于删除指定 id（Document ID）的文档。

接口约束

索引类型为 FLAT，不支持删除。

请求示例

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，已使用 upsert() 插入数据，通过 delete() 删除指定的 Document。

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

db = client.database('db-test')
coll = db.collection('book-vector')

# delete
# 删除指定 id（Document ID），并满足 Filter 表达式的文档。
coll.delete(document_ids=['0001', '0002', '0003'], filter=Filter(Filter.In("bookName",
["三国演义", "西游记"])))
# 删除之后，确认book-vector 的文档
doc_list = coll.query(document_ids=['0001','0002', '0003'], retrieve_vector=True,
limit=3)
# 输出确认结果，仅输出 0003 的数据。
for doc in doc_list:
        print(doc)
```

仅输出 id 为 0003 的数据，如下所示。

```
{'id': '0003', 'vector': [0.21230000257492065, 0.20999999344348907,
0.21299999952316284], 'bookName': '红楼梦', 'page': 23, 'author': '曹雪芹'}
```

请求参数

参数名	是否必选	参数含义	配置方法
document_ids	是	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（=）、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。
			  ○ **unit64**：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')`。

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。

更新 Document

最近更新时间：2024-01-18 17:43:01

功能介绍

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。

> **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。

接口约束

不能变更 Document ID 字段，不要求事务完整性。

请求示例

写入向量数据

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
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# 指定需更新文档所属的数据库
db = client.database('db-test')
# 指定集合
coll = db.collection('book-vector')
#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
coll.update(data=update_doc, document_ids=['0001','0002','0003'],
filter=Filter(Filter.In("bookName",["三国演义", "西游记"])))
# 更新之后，确认字段已更新
doc_list = coll.query(document_ids=['0001','0002'], retrieve_vector=True,
limit=3)
# 输出确认结果
for doc in doc_list:
        print(doc)
```

输出如下信息，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。

```
{'id': '0001', 'vector': [0.21230000257492065, 0.23000000417232513,
0.21299999952316284], 'test_new_field': 'new field value', 'author': '罗贯中',
'page': 30, 'bookName': '三国演义'}
{'id': '0002', 'vector': [0.21230000257492065, 0.2199999988079071,
0.21299999952316284], 'author': '吴承恩', 'test_new_field': 'new field value',
'bookName': '西游记', 'page': 30}
```

写入原始文本

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.enum import FieldType, IndexType, MetricType,
EmbeddingModel, 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.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)

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

```
coll = db.collection('book-emb')

# 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
coll.update(data=update_doc, document_ids=['0001', '0002','0003'],
filter=Filter(Filter.In("bookName",["三国演义", "西游记"])))
# 更新之后，确认字段已更新，注意 limit 指每页返回的 Document 数量，默认为 1。
doc_list = coll.query(document_ids=['0001','0002'], retrieve_vector=False,
limit=3)
# 输出确认结果
for doc in doc_list:
        print(doc)
```

输出如下信息，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。

```
{'id': '0002', 'page': 30, 'bookName': '西游记', 'test_new_field': 'new field value',
'text': '分久必合，合久必分', 'author': '吴承恩'}
{'id': '0001', 'author': '罗贯中', 'text': '分久必合，合久必分', 'page': 30,
'test_new_field': 'new field value', 'bookName': '三国演义'}
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法
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（=）、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。 ○ **unit64**：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，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。

DocumentSet 操作
上传文件

最近更新时间：2024-05-15 15:38:01

功能介绍

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

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。

请求示例

```
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

#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', 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.md",
        document_set_name="腾讯云向量数据库.md",
        metadata={
            'author': 'Tencent',
            'tags': ['向量', 'Embedding', 'AI']
        },
```

```
            splitter_process=SplitterProcess(
                append_keywords_to_chunk=True,
                append_title_to_chunk=False,
            ),
        )
print(vars(res))
```

输出信息，如下所示。

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

请求参数

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。

返回参数

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：2024-05-15 15:59:41

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。

请求示例

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

#create a database client object
client = tcvectordb.VectorDBClient(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.get_document_set(document_set_name="腾讯云向量数据库.md")
print(vars(res))
```

请求参数

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	-

返回消息

```
{
 'documentSetId': '11901307631454126O8',
 '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',
'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
  },
  '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。
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。
author、tags	-	自定义扩展的文件 Metadata 信息的标量字段。

获取 Chunks

最近更新时间：2024-05-15 15:59:42

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。

- 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。

请求示例

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

#create a database client object
client = tcvectordb.VectorDBClient(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')

chunk_list = coll_view.get_chunks(document_set_name='腾讯云向量数据库.md',
limit=3)
for chunks in chunk_list:
        print(vars(chunks))
```

请求参数

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：[1,16384]。
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。

返回信息

{'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。
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。

精确查询文件信息

最近更新时间：2024-05-15 15:59:42

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。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，过滤需查找的文件。

请求示例

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

#create a database client object
client = tcvectordb.VectorDBClient(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.query(
        document_set_name="腾讯云向量数据库.md",
        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.VectorDBClient(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': '11901307631454126 08',
  '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
  },
  '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.VectorDBClient(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'
}
```

请求参数

子参数	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_s	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]。

et_id		
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。

返回消息


```
{
  'documentSetId': '11901307631454126 08',
  '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
  },
  '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。
author	-	自定义的文件 Meta 信息字段。

文件内容检索

最近更新时间：2024-05-15 15:59:42

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。**

请求示例

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.md` 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.VectorDBClient(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',
        document_set_name = ['腾讯云向量数据库.md'],
        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.VectorDBClient(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))
```

请求参数

参数名称	是否必选	参数含义及配置方法
content	否	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	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]。

_set_name		
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。

返回参数

如下为检索到的相似数据，返回最相似的 Top 3 条数据。


```
{
 'score': 0.8924504518508911,
 'data': {
   '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,
   'pre': [
     '### 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'
   ],
   'next': [

   ]
 },
```

```
  'documentSet': {
    'documentSetId': '1180107147771117568',
    'documentSetName': '腾讯云向量数据库.md',
    'author': 'Tencent',
    'tags': [
      '向量',
      'Embedding',
      'AI'
    ]
  }
}{
  'score': 0.8905023932456 97,
  'data': {
    '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,
    'pre': [
      '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'
    ],
    'next': [

    ]
  },
  'documentSet': {
    'documentSetId': '1180107147771117568',
    'documentSetName': '腾讯云向量数据库.md',
    'author': 'Tencent',
    'tags': [
      '向量',
      'Embedding',
      'AI'
    ]
  }
}{
  'score': 0.8756946921348572,
  'data': {
    '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,
    'pre': [

    ],
    'next': [

    ]
  },
  'documentSet': {
    'documentSetId': '1180107147771117568',
    'documentSetName': '腾讯云向量数据库.md',
    '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。
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：2024-05-15 16:01:01

功能介绍

delete() 接口用于删除存储于 CollectionView 文件。

- **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。**

请求示例

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.md` ，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.VectorDBClient(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_name = ['腾讯云向量数据库.md'],
        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.VectorDBClient(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.VectorDBClient(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)
```

请求参数

参数名称	参数含义	是否	配置方法及要求

		必选	
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并集。
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并集。
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')`

更新文件

最近更新时间：2024-05-15 15:59:42

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。

- 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。

> **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。

请求示例

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。

```
import tcvectordb
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.VectorDBClient(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.update(
        data=Document(author ='tencent', tag = '向量'),
```

```
        document_set_name = ['腾讯云向量数据库.md'],
        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。

```
import tcvectordb
from tcvectordb.model.document import Filter, Document
from tcvectordb.model.enum import ReadConsistency
#create a database client object
client = tcvectordb.VectorDBClient(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.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.VectorDBClient(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.update(
        data=Document(author ='tencent', tag = '向量'),
        filter=Filter("author=\"Tencent\"")
    )
print(res)
```

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
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并集。
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并集。
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]。

返回消息

```
{'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：2024-05-17 15:54:51

功能介绍

rebuild_index() 接口用于重建 Base 类 Collection 的所有索引。

接口约束

> ⚠ **注意：**
> - **索引重建过程中 Collection 禁止写入、读取。**
> - **重建索引需要新的内存来构建索引。**

请求示例

基于 create_collection() 创建的集合，为集合 book 指定别名。

```
import tcvectordb
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency
#create a database client object
client = tcvectordb.VectorDBClient(url='http://10.0.X.X', username='root',
key='eC4bLRy2va**********************************',
read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY, timeout=30)
# Specify the database name and Collection name
db = client.database('db-test')
coll = db.collection('book-vector')
# rebuild
coll.rebuild_index(drop_before_rebuild=False,throttle=1,timeout=30)
```

请求参数

参数	是否必选	参数含义	配置方法及要求
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。	取值如下所示： • True：重建之前，先删除旧索引在重

		说明： 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。	建索引。 **说明：** 内存资源不足时，可先删除旧索引，在新索引还没有创建完成之前，该表无法正常读写。 • 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。 **说明：** 内存资源足够的情况下，可不删除旧索引。在新索引还没有创建完成之前，该表可读，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：CPU 核数为 1。

返回参数

输出信息，如下所示。

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

参数名	参数含义
affectedCount	影响行数，即为重建索引的集合数量。

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。

- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training data**：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 index**：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。

Java SDK
SDK 准备

最近更新时间：2024-04-07 15:59:51

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。

> **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 。

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	https://github.com/Tencent/vectordatabase-sdk-java

接入方式

> **注意：**
> 如下为 Gradle 与 Maven 项目添加 SDK 版本1.3.2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:1.3.2
```

Maven 引3

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>1.3.2</version>
</dependency>
```

Java SDK Demo
写入原始文本并检索（Embedding）

最近更新时间：2023-11-15 15:16:21

腾讯云向量数据库（Tencent Cloud VectorDB）目前已支持文本 Embedding 模型，能够覆盖多种主流语言的向量转换。本文给出通过 Java SDK 写入或更新原始文本，并进行精确查询或相似度检索的完整示例，便于您更加高效地管理和使用向量数据。完整项目，请参见 SDK 准备 。

```
package com.tencent.tcvectordb;

import com.tencent.tcvectordb.client.VectorDBClient;
import com.tencent.tcvectordb.exception.VectorDBException;
import com.tencent.tcvectordb.model.Collection;
import com.tencent.tcvectordb.model.Database;
import com.tencent.tcvectordb.model.DocField;
import com.tencent.tcvectordb.model.Document;
import com.tencent.tcvectordb.model.param.collection.*;
import com.tencent.tcvectordb.model.param.database.ConnectParam;
import com.tencent.tcvectordb.model.param.dml.*;
import com.tencent.tcvectordb.model.param.entity.AffectRes;
import com.tencent.tcvectordb.model.param.entity.SearchRes;
import com.tencent.tcvectordb.model.param.enums.ReadConsistencyEnum;


import java.util.*;
import java.util.stream.Collectors;


import static
com.tencent.tcvectordb.model.param.enums.EmbeddingModelEnum.BGE_BASE_ZH;


/**
 * VectorDB Java SDK usage example
 */
public class VectorDBExampleWithEmbedding {


    private static final String DBNAME = "book";
    private static final String COLL_NAME = "book_segments";
    private static final String COLL_NAME_ALIAS = "collection_alias";
```

```
    public static void example() throws InterruptedException {
        // 创建VectorDB Client
        ConnectParam connectParam = initConnectParam();
        VectorDBClient client = new VectorDBClient(connectParam,
ReadConsistencyEnum.EVENTUAL_CONSISTENCY);


        // 测试前清理环境
        System.out.println("---------------------- clear before test ----------------------");
        anySafe(() -> clear(client));
        createDatabaseAndCollection(client);
        upsertData(client);
        queryData(client);
        updateAndDelete(client);
        deleteAndDrop(client);
        testFilter();
    }




    /**
     * init connect parameter
     *
     * @return {@link ConnectParam}
     */
    private static ConnectParam initConnectParam() {
        System.out.println("\tvdb_url: " + System.getProperty("vdb_url"));
        System.out.println("\tvdb_key: " + System.getProperty("vdb_key"));
        return ConnectParam.newBuilder()
                .withUrl(System.getProperty("vdb_url"))
                .withUsername("root")
                .withKey(System.getProperty("vdb_key"))
                .withTimeout(30)
                .build();
    }


    /**
     * 执行 {@link Runnable} 捕获所有异常
     *
     * @param runnable {@link Runnable}
     */
```

```
    private static void anySafe(Runnable runnable) {
        try {
            runnable.run();
        } catch (VectorDBException e) {
            System.err.println(e);
            e.printStackTrace();
        }
    }


    private static void createDatabaseAndCollection(VectorDBClient client) {
        // 1. 创建数据库
        System.out.println("---------- createDatabase ----------");
        Database db = client.createDatabase(DBNAME);


        // 2. 列出所有数据库
        System.out.println("---------- listDatabase ----------");
        List<String> database = client.listDatabase();
        for (String s : database) {
            System.out.println("\tres: " + s);
        }


        // 3. 创建 collection
        System.out.println("---------- createCollection ----------");
        CreateCollectionParam collectionParam =
initCreateEmbeddingCollectionParam(COLL_NAME);
        db.createCollection(collectionParam);


        // 4. 列出所有 collection
        System.out.println("---------- listCollections ----------");
        List<Collection> cols = db.listCollections();
        for (Collection col : cols) {
            System.out.println("\tres: " + col.toString());
        }


        // 5. 设置 collection 别名
        System.out.println("---------- setAlias ----------");
        AffectRes affectRes = db.setAlias(COLL_NAME, COLL_NAME_ALIAS);
        System.out.println("\tres: " + affectRes.toString());
```

```
        // 6. describe collection
        System.out.println("---------------------- describeCollection ----------------------");
        Collection descCollRes = db.describeCollection(COLL_NAME);
        System.out.println("\tres: " + descCollRes.toString());


        // 7. delete alias
        System.out.println("---------------------- deleteAlias ----------------------");
        AffectRes affectRes1 = db.deleteAlias(COLL_NAME_ALIAS);
        System.out.println("\tres: " + affectRes1);


        // 8. describe collection
        System.out.println("---------------------- describeCollection ----------------------");
        Collection descCollRes1 = db.describeCollection(COLL_NAME);
        System.out.println("\tres: " + descCollRes1.toString());


    }



    private static void upsertData(VectorDBClient client) throws InterruptedException {
        Database database = client.database(DBNAME);
        Collection collection = database.describeCollection(COLL_NAME);
        List<Document> documentList = new ArrayList<>(Arrays.asList(
                Document.newBuilder()
                        .withId("0001")
                        .addDocField(new DocField("bookName", "西游记"))
                        .addDocField(new DocField("author", "吴承恩"))
                        .addDocField(new DocField("page", 21))
                        .addDocField(new DocField("segment", "富贵功名，前缘分定，为人切莫欺
心。"))
                        .addDocField(new DocField("text", "富贵功名，前缘分定，为人切莫欺
心。"))
                        .build(),
                Document.newBuilder()
                        .withId("0002")
                        .addDocField(new DocField("bookName", "西游记"))
                        .addDocField(new DocField("author", "吴承恩"))
                        .addDocField(new DocField("page", 22))
                        .addDocField(new DocField("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。"))
                .addDocField(new DocField("text",
                        "正大光明，忠良善果弥深。些些狂妄天加谴，眼前不遇待时临。"))
                .build(),
        Document.newBuilder()
                .withId("0003")
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .addDocField(new DocField("page", 23))
                .addDocField(new DocField("segment", "细作探知这个消息，飞报吕布。"))
                .addDocField(new DocField("text", "细作探知这个消息，飞报吕布。"))
                .build(),
        Document.newBuilder()
                .withId("0004")
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .addDocField(new DocField("page", 24))
                .addDocField(new DocField("segment", "富贵功名，前缘分定，为人切莫欺
心。"))
                .addDocField(new DocField("text", "富贵功名，前缘分定，为人切莫欺
心。"))
                .build(),
        Document.newBuilder()
                .withId("0005")
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .addDocField(new DocField("page", 25))
                .addDocField(new DocField("segment",
                        "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，可往投之。"))
                .addDocField(new DocField("text",
                        "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，可往投之。"))
                .build()));
    System.out.println("---------------------- upsert ----------------------");
    InsertParam insertParam = InsertParam.newBuilder()
            .addAllDocument(documentList)
            .withBuildIndex(true)
            .build();
    collection.upsert(insertParam);


    // notice：upsert操作可用会有延迟
    Thread.sleep(1000 * 5);
}
```

```
    private static void queryData(VectorDBClient client) {
        Database database = client.database(DBNAME);
        Collection collection = database.describeCollection(COLL_NAME);


        // query  查询
        // 1. query 用于查询数据
        // 2. 可以通过传入主键 id 列表或 filter 实现过滤数据的目的
        // 3. 如果没有主键 id 列表和 filter 则必须传入 limit 和 offset，类似 scan 的数据扫描功能
        // 4. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回


        System.out.println("---------------------- query ----------------------");
        List<String> documentIds = Arrays.asList("0001", "0002", "0003", "0004",
"0005");
        Filter filterParam = new Filter("bookName=\"三国演义\"");
        List<String> outputFields = Arrays.asList("id", "bookName");
        QueryParam queryParam = QueryParam.newBuilder()
                .withDocumentIds(documentIds)
                // 使用 filter 过滤数据
                .withFilter(filterParam)
                // limit 限制返回行数，1 到 16384 之间
                .withLimit(2)
                // 偏移
                .withOffset(1)
                // 指定返回的 fields
                .withOutputFields(outputFields)
                // 是否返回 vector 数据
                .withRetrieveVector(false)
                .build();
        List<Document> qdos = collection.query(queryParam);
        for (Document doc : qdos) {
            System.out.println("\tres: " + doc.toString());
        }


        // searchById
        // 1. searchById 提供按 id 搜索的能力
        // 2. 支持通过 filter 过滤数据
        // 3. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
        // 4. limit 用于限制每个单元搜索条件的条数，如 vector 传入三组向量，limit 为 3，则
limit 限制的是每组向量返回 top 3 的相似度向量
```

```
        System.out.println("---------------------- searchById ----------------------");
        SearchByIdParam searchByIdParam = SearchByIdParam.newBuilder()
                .withDocumentIds(documentIds)
                // 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))
                // 指定 Top K 的 K 值
                .withLimit(2)
                // 过滤获取到结果
                .withFilter(filterParam)
                .build();
        List<List<Document>> siDocs = collection.searchById(searchByIdParam);
        int i = 0;
        for (List<Document> docs : siDocs) {
            System.out.println("\tres: " + i++);
            for (Document doc : docs) {
                System.out.println("\tres: " + doc.toString());
            }
        }


        // search
        // 1. search 提供按照 vector 搜索的能力
        // 其他选项类似 search 接口


        System.out.println("---------------------- search ----------------------");
        queryParam = QueryParam.newBuilder()
                .withDocumentIds(documentIds)
                // limit 限制返回行数，0 到 16384 之间
                .withLimit(2)
                // 偏移
                .withOffset(1)
                // 指定返回的 fields
                .withOutputFields(outputFields)
                // 是否返回 vector 数据
                .withRetrieveVector(true)
                .build();
        List<Document> allRes = collection.query(queryParam);
        SearchByVectorParam searchByVectorParam =
SearchByVectorParam.newBuilder()

.withVectors(allRes.stream().map(Document::getVector).collect(Collectors.toList()))
```

```
            // 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))
            // 指定 Top K 的 K 值
            .withLimit(2)
            // 过滤获取到结果
            .withFilter(filterParam)
            .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 = collection.search(searchByVectorParam);
        i = 0;
        for (List<Document> docs : svDocs) {
            System.out.println("\tres: " + i++);
            for (Document doc : docs) {
                System.out.println("\tres: " + doc.toString());
            }
        }


        // searchByEmbeddingItems 返回类型为 SearchRes，接口查询过程中 embedding 可
能会出现截断
        // 如发生截断将会返回响应 warn 信息，如需确认是否截断可以使用
SearchRes#getWarning" 获取警告信息，
        // 查询结果可以通过 SearchRes#getDocuments
        System.out.println("---------------------- searchByEmbeddingItems ----------------------");
        SearchByEmbeddingItemsParam searchByEmbeddingItemsParam =
SearchByEmbeddingItemsParam.newBuilder()
            .withEmbeddingItems(Arrays.asList("闻刘玄德新领徐州", "细作探知这个消息"))
            .withParams(new HNSWSearchParams(100))
            .withLimit(5)
            .build();
        SearchRes searchRes =
collection.searchByEmbeddingItems(searchByEmbeddingItemsParam);
        i = 0;
        for (List<Document> docs : searchRes.getDocuments()) {
            System.out.println("\tres: " + i++);
            for (Document doc : docs) {
                System.out.println("\tres: " + doc.toString());
            }
        }
    }
```

```
    private static void updateAndDelete(VectorDBClient client) throws
InterruptedException {
        Database database = client.database(DBNAME);
        Collection collection = database.describeCollection(COLL_NAME);



        // update
        // 1. update 提供基于 [主键查询] 和 [Filter 过滤] 的部分字段更新或者非索引字段新增


        // filter 限制仅会更新 id = "0003"
        System.out.println("---------------------- update ----------------------");
        Filter filterParam = new Filter("bookName=\"三国演义\"");
        List<String> documentIds = Arrays.asList("0001", "0003");
        UpdateParam updateParam = UpdateParam
                .newBuilder()
                .addAllDocumentId(documentIds)
                .withFilter(filterParam)
                .build();
        Document updateDoc = Document
                .newBuilder()
                .addDocField(new DocField("page", 100))
                // 支持添加新的内容
                .addDocField(new DocField("extend", "extendContent"))
                .build();
        collection.update(updateParam, updateDoc);


        // delete
        // 1. delete 提供基于[ 主键查询]和[Filter 过滤]的数据删除能力
        // 2. 删除功能会受限于 collection 的索引类型，部分索引类型不支持删除操作


        // filter 限制只会删除 id = "00001" 成功
        System.out.println("---------------------- delete ----------------------");
        filterParam = new Filter("bookName=\"西游记\"");
        DeleteParam build = DeleteParam
                .newBuilder()
                .addAllDocumentId(documentIds)
                .withFilter(filterParam)
                .build();
        collection.delete(build);
```

```
    // notice：delete操作可用会有延迟
    Thread.sleep(1000 * 5);


    // rebuild index
    System.out.println("---------------------- rebuild index ----------------------");
    RebuildIndexParam rebuildIndexParam = RebuildIndexParam
            .newBuilder()
            .withDropBeforeRebuild(false)
            .withThrottle(1)
            .build();
    collection.rebuildIndex(rebuildIndexParam);
    Thread.sleep(5 * 1000);



    // truncate 会清除整个 Collection 的数据，包括索引
    System.out.println("---------------------- truncate collection ----------------------");
    AffectRes affectRes = database.truncateCollections(COLL_NAME);
    System.out.println("\tres: " + affectRes.toString());


    Thread.sleep(5 * 1000);
}


private static void deleteAndDrop(VectorDBClient client) {
    Database database = client.database(DBNAME);


    // 删除 collection
    System.out.println("---------------------- dropCollection ----------------------");
    database.dropCollection(COLL_NAME);


    // 删除 database
    System.out.println("---------------------- dropDatabase ----------------------");
    client.dropDatabase(DBNAME);
}
```

```
    private static void clear(VectorDBClient client) {
        List<String> databases = client.listDatabase();
        for (String database : databases) {
            client.dropDatabase(database);
        }
    }



    /**
     * 初始化创建 Collection 参数
     * 通过调用 addField 方法设计索引（不是设计 Collection 的结构）
     * <ol>
     *     <li>【重要的事】向量对应的文本字段不要建立索引，会浪费较大的内存，并且没有任何
作用。</li>
     *     <li>【必须的索引】：主键id、向量字段 vector 这两个字段目前是固定且必须的，参考
下面的例子；</li>
     *     <li>【其他索引】：检索时需作为条件查询的字段，比如要按书籍的作者进行过滤，这个
时候author字段就需要建立索引，
     *     否则无法在查询的时候对 author 字段进行过滤，不需要过滤的字段无需加索引，会浪费
内存；</li>
     *     <li>向量数据库支持动态 Schema，写入数据时可以写入任何字段，无需提前定义，类似
MongoDB.</li>
     *     <li>例子中创建一个书籍片段的索引，例如书籍片段的信息包括 {id, vector, segment,
bookName, author, page},
     *     id 为主键需要全局唯一，segment 为文本片段, vector 字段需要建立向量索引，假如我
们在查询的时候要查询指定书籍
     *     名称的内容，这个时候需要对 bookName 建立索引，其他字段没有条件查询的需要，无
需建立索引。/li>
     *     <li>创建带 Embedding 的 collection 需要保证设置的 vector 索引的维度和
Embedding 所用模型生成向量维度一致，模型及维度关系
     *     见下方表格
     *     </li>
     * </ol>
     * <table border>
     * <caption>模型列表</caption>
     *     <tr>
     *         <th>model</th>
     *         <th>dimension</th>
     *     </tr>
     *     <tr>
     *         <td>bge-base-zh</td>
```

```
     *       <td>768</td>
     *     </tr>
     *     <tr>
     *       <td>m3e-base</td>
     *       <td>768</td>
     *     </tr>
     *     <tr>
     *       <td>text2vec-large-chinese</td>
     *       <td>1024</td>
     *     </tr>
     *     <tr>
     *       <td>e5-large-v2</td>
     *       <td>1024</td>
     *     </tr>
     *     <tr>
     *       <td>multilingual-e5-base</td>
     *       <td>768</td>
     *     </tr>
     * </table>
     *
     * @param collName
     * @return
     */
    private static CreateCollectionParam initCreateEmbeddingCollectionParam(String
collName) {
        return CreateCollectionParam.newBuilder()
                .withName(collName)
                .withShardNum(3)
                .withReplicaNum(2)
                .withDescription("test embedding collection0")
                .addField(new FilterIndex("id", FieldType.String, IndexType.PRIMARY_KEY))
                .addField(new VectorIndex("vector", BGE_BASE_ZH.getDimension(),
IndexType.HNSW,
                        MetricType.COSINE, new HNSWParams(16, 200)))
                .addField(new FilterIndex("bookName", FieldType.String, IndexType.FILTER))
                .addField(new FilterIndex("author", FieldType.String, IndexType.FILTER))
                .withEmbedding(
                        Embedding
                                .newBuilder()
                                .withModel(BGE_BASE_ZH)
                                .withField("text")
                                .withVectorField("vector")
                                .build())
                .build();
    }
```

```
    /**
     * 测试 Filter
     */
    public static void testFilter() {
        System.out.println("\tres: " + new Filter("author=\"jerry\"")
                .and("a=1")
                .or("r=\"or\"")
                .orNot("rn=2")
                .andNot("an=\"andNot\"")
                .getCond());
        System.out.println("\tres: " + Filter.in("key", Arrays.asList("v1", "v2", "v3")));
        System.out.println("\tres: " + Filter.in("key", Arrays.asList(1, 2, 3)));
    }



}
```

写入向量数据并检索

最近更新时间：2023-11-15 15:16:21

腾讯云向量数据库（Tencent Cloud VectorDB）支持直接写入向量数据。本文给出通过 Java SDK 写入或更新向量数据，并进行精确查询与相似度检索的完整示例，便于您更加高效地管理和使用向量数据。完整项目，请参见 SDK 准备 。

```
package com.tencent.tcvectordb;


import com.tencent.tcvectordb.client.VectorDBClient;
import com.tencent.tcvectordb.exception.VectorDBException;
import com.tencent.tcvectordb.model.Collection;
import com.tencent.tcvectordb.model.Database;
import com.tencent.tcvectordb.model.DocField;
import com.tencent.tcvectordb.model.Document;
import com.tencent.tcvectordb.model.param.collection.*;
import com.tencent.tcvectordb.model.param.database.ConnectParam;
import com.tencent.tcvectordb.model.param.dml.*;
import com.tencent.tcvectordb.model.param.entity.AffectRes;
import com.tencent.tcvectordb.model.param.enums.ReadConsistencyEnum;


import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;


/**
 * VectorDB Java SDK usage example
 */
public class VectorDBExample {


    private static final String DBNAME = "book";
    private static final String COLL_NAME = "book_segments";
    private static final String COLL_NAME_ALIAS = "collection_alias";


    public static void example() throws InterruptedException {
        // 创建 VectorDB Client
        ConnectParam connectParam = initConnectParam();
```

```
        VectorDBClient client = new VectorDBClient(connectParam,
ReadConsistencyEnum.EVENTUAL_CONSISTENCY);


        // 测试前清理环境
        System.out.println("---------------------- clear before test ----------------------");
        anySafe(() -> clear(client));
        createDatabaseAndCollection(client);
        upsertData(client);
        queryData(client);
        updateAndDelete(client);
        deleteAndDrop(client);
        testFilter();
    }



    /**
     * init connect parameter
     *
     * @return {@link ConnectParam}
     */
    private static ConnectParam initConnectParam() {
        System.out.println("\tvdb_url: " + System.getProperty("vdb_url"));
        System.out.println("\tvdb_key: " + System.getProperty("vdb_key"));
        return ConnectParam.newBuilder()
                .withUrl(System.getProperty("vdb_url"))
                .withUsername("root")
                .withKey(System.getProperty("vdb_key"))
                .withTimeout(30)
                .build();
    }


    /**
     * 执行 {@link Runnable} 捕获所有异常
     *
     * @param runnable {@link Runnable}
     */
    private static void anySafe(Runnable runnable) {
        try {
            runnable.run();
        } catch (VectorDBException e) {
            System.err.println(e);
```

```
            e.printStackTrace();
        }
    }


    private static void createDatabaseAndCollection(VectorDBClient client) {
        // 1. 创建数据库
        System.out.println("---------- createDatabase ----------");
        Database db = client.createDatabase(DBNAME);


        // 2. 列出所有数据库
        System.out.println("---------- listCollections ----------");
        List<String> database = client.listDatabase();
        for (String s : database) {
            System.out.println("\tres: " + s);
        }


        // 3. 创建 collection
        System.out.println("---------- createCollection ----------");
        CreateCollectionParam collectionParam =
initCreateCollectionParam(COLL_NAME);
        db.createCollection(collectionParam);


        // 4. 列出所有 collection
        System.out.println("---------- listCollections ----------");
        List<Collection> cols = db.listCollections();
        for (Collection col : cols) {
            System.out.println("\tres: " + col.toString());
        }


        // 5. 设置 collection 别名
        System.out.println("---------- setAlias ----------");
        AffectRes affectRes = db.setAlias(COLL_NAME, COLL_NAME_ALIAS);
        System.out.println("\tres: " + affectRes.toString());




        // 6. describe collection
        System.out.println("---------- describeCollection ----------");
        Collection descCollRes = db.describeCollection(COLL_NAME);
```

```
        System.out.println("\tres: " + descCollRes.toString());


        // 7. delete alias
        System.out.println("---------------------- deleteAlias ----------------------");
        AffectRes affectRes1 = db.deleteAlias(COLL_NAME_ALIAS);
        System.out.println("\tres: " + affectRes1);


        // 8. describe collection
        System.out.println("---------------------- describeCollection ----------------------");
        Collection descCollRes1 = db.describeCollection(COLL_NAME);
        System.out.println("\tres: " + descCollRes1.toString());
    }

    private static void upsertData(VectorDBClient client) throws InterruptedException {
        Database database = client.database(DBNAME);
        Collection collection = database.describeCollection(COLL_NAME);
        List<Document> documentList = new ArrayList<>(Arrays.asList(
                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("segment", "富贵功名，前缘分定，为人切莫欺
心。"))
                        .build(),
                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("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。"))
                        .build(),
                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("segment", "细作探知这个消息，飞报吕布。"))
                        .build(),
```

```
            Document.newBuilder()
                    .withId("0004")
                    .withVector(Arrays.asList(0.2123, 0.24, 0.213))
                    .addDocField(new DocField("bookName", "三国演义"))
                    .addDocField(new DocField("author", "罗贯中"))
                    .addDocField(new DocField("page", 24))
                    .addDocField(new DocField("segment", "富贵功名，前缘分定，为人切莫欺
心。"))
                    .build(),
            Document.newBuilder()
                    .withId("0005")
                    .withVector(Arrays.asList(0.2123, 0.25, 0.213))
                    .addDocField(new DocField("bookName", "三国演义"))
                    .addDocField(new DocField("author", "罗贯中"))
                    .addDocField(new DocField("page", 25))
                    .addDocField(new DocField("segment",
                            "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，可往投之。"))
                    .build()));
        System.out.println("---------------------- upsert ----------------------");
        InsertParam insertParam =
InsertParam.newBuilder().addAllDocument(documentList).withBuildIndex(true).build();
        collection.upsert(insertParam);


        // notice：upsert 操作可用会有延迟
        Thread.sleep(1000 * 5);
    }


    private static void queryData(VectorDBClient client) {
        Database database = client.database(DBNAME);
        Collection collection = database.describeCollection(COLL_NAME);


        System.out.println("---------------------- query ----------------------");
        List<String> documentIds = Arrays.asList("0001", "0002", "0003", "0004",
"0005");
        Filter filterParam = new Filter("bookName=\"三国演义\"");
        List<String> outputFields = Arrays.asList("id", "bookName");
        QueryParam queryParam = QueryParam.newBuilder()
                .withDocumentIds(documentIds)
                // 使用 filter 过滤数据
                .withFilter(filterParam)
                // limit 限制返回行数，1 到 16384 之间
                .withLimit(2)
```

```
            // 偏移
            .withOffset(1)
            // 指定返回的 fields
            .withOutputFields(outputFields)
            // 是否返回 vector 数据
            .withRetrieveVector(false)
            .build();
        List<Document> qdos = collection.query(queryParam);
        for (Document doc : qdos) {
            System.out.println("\tres: " + doc.toString());
        }



        // searchById
        // 1. searchById 提供按 id 搜索的能力
        // 2. 支持通过 filter 过滤数据
        // 3. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
        // 4. limit 用于限制每个单元搜索条件的条数，如 vector 传入三组向量，limit 为 3，则
limit 限制的是每组向量返回 top 3 的相似度向量


        System.out.println("---------------------- searchById ----------------------");
        SearchByIdParam searchByIdParam = SearchByIdParam.newBuilder()
                .withDocumentIds(documentIds)
                // 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))
                // 指定 Top K 的 K 值
                .withLimit(2)
                // 过滤获取到结果
                .withFilter(filterParam)
                .build();
        List<List<Document>> siDocs = collection.searchById(searchByIdParam);
        int i = 0;
        for (List<Document> docs : siDocs) {
            System.out.println("\tres: " + i++);
            for (Document doc : docs) {
                System.out.println("\tres: " + doc.toString());
            }
        }
```

```
        // search
        // 1. search 提供按照 vector 搜索的能力
        // 其他选项类似 search 接口


        System.out.println("---------------------- search ----------------------");
        SearchByVectorParam searchByVectorParam =
SearchByVectorParam.newBuilder()
                .addVector(Arrays.asList(0.2123, 0.23, 0.213))
                // 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))
                // 指定 Top K 的 K 值
                .withLimit(2)
                // 过滤获取到结果
                .withFilter(filterParam)
                .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 = collection.search(searchByVectorParam);
        i = 0;
        for (List<Document> docs : svDocs) {
            System.out.println("\tres: " + i);
            i++;
            for (Document doc : docs) {
                System.out.println("\tres: " + doc.toString());
            }
        }
    }


    private static void updateAndDelete(VectorDBClient client) throws
InterruptedException {
        Database database = client.database(DBNAME);
        Collection collection = database.describeCollection(COLL_NAME);


        System.out.println("---------------------- update ----------------------");
        // update
        // 1. update 提供基于 [主键查询] 和 [Filter 过滤] 的部分字段更新或者非索引字段新增


        // filter 限制仅会更新 id = "0003"
```

```
        Filter filterParam = new Filter("bookName=\"三国演义\"");
        List<String> documentIds = Arrays.asList("0001", "0003");
        UpdateParam updateParam = UpdateParam
                .newBuilder()
                .addAllDocumentId(documentIds)
                .withFilter(filterParam)
                .build();
        Document updateDoc = Document
                .newBuilder()
                .addDocField(new DocField("page", 100))
                // 支持添加新的内容
                .addDocField(new DocField("extend", "extendContent"))
                .build();
        collection.update(updateParam, updateDoc);


        System.out.println("---------------------- delete ----------------------");
        // delete
        // 1. delete 提供基于[ 主键查询]和[Filter 过滤]的数据删除能力
        // 2. 删除功能会受限于 collection 的索引类型，部分索引类型不支持删除操作


        // filter 限制只会删除 id = "00001" 成功
        filterParam = new Filter("bookName=\"西游记\"");
        DeleteParam build = DeleteParam
                .newBuilder()
                .addAllDocumentId(documentIds)
                .withFilter(filterParam)
                .build();
        collection.delete(build);



        // rebuild index
        System.out.println("---------------------- rebuildIndex ----------------------");


        RebuildIndexParam rebuildIndexParam = RebuildIndexParam
                .newBuilder()
                .withDropBeforeRebuild(false)
                .withThrottle(1)
                .build();
        collection.rebuildIndex(rebuildIndexParam);
```

```
        Thread.sleep(1000 * 5);


        // query
        System.out.println("----------------------  query ----------------------");
        documentIds = Arrays.asList("0001", "0002", "0003", "0004", "0005");
        List<String> outputFields = Arrays.asList("id", "bookName", "page", "extend");
        QueryParam queryParam = QueryParam.newBuilder()
                .withDocumentIds(documentIds)
                // 使用 filter 过滤数据
                .withOutputFields(outputFields)
                // 是否返回 vector 数据
                .withRetrieveVector(false)
                .build();
        List<Document> qdos = collection.query(queryParam);
        for (Document doc : qdos) {
            System.out.println("\tres: " + doc.toString());
        }



        // truncate 会清除整个 Collection 的数据，包括索引
        System.out.println("---------------------- truncate collection ----------------------");
        AffectRes affectRes = database.truncateCollections(COLL_NAME);
        System.out.println("\tres: " + affectRes.toString());


        // notice：delete操作可用会有延迟
        Thread.sleep(1000 * 5);
    }


    private static void deleteAndDrop(VectorDBClient client) {
        Database database = client.database(DBNAME);


        // 删除 collection
        System.out.println("---------------------- truncate collection ----------------------");
        database.dropCollection(COLL_NAME);


        // 删除 database
```

```
        System.out.println("---------------------- truncate collection ----------------------");
        client.dropDatabase(DBNAME);
    }




    private static void clear(VectorDBClient client) {
        List<String> databases = client.listDatabase();
        for (String database : databases) {
            client.dropDatabase(database);
        }
    }




    /**
     * 初始化创建 Collection 参数
     * 通过调用 addField 方法设计索引（不是设计 Collection 的结构）
     * <ol>
     *     <li>【重要的事】向量对应的文本字段不要建立索引，会浪费较大的内存，并且没有任何
作用。</li>
     *     <li>【必须的索引】：主键id、向量字段 vector 这两个字段目前是固定且必须的，参考
下面的例子；</li>
     *     <li>【其他索引】：检索时需作为条件查询的字段，比如要按书籍的作者进行过滤，这个
时候author字段就需要建立索引，
     *     否则无法在查询的时候对 author 字段进行过滤，不需要过滤的字段无需加索引，会浪费
内存；</li>
     *     <li>向量数据库支持动态 Schema，写入数据时可以写入任何字段，无需提前定义，类似
MongoDB.</li>
     *     <li><例子中创建一个书籍片段的索引，例如书籍片段的信息包括 {id, vector,
segment, bookName, author, page},
     *     id 为主键需要全局唯一，segment 为文本片段, vector 字段需要建立向量索引，假如我
们在查询的时候要查询指定书籍
     *     名称的内容，这个时候需要对 bookName 建立索引，其他字段没有条件查询的需要，无
需建立索引。/li>
     * </ol>
     *
     * @param collName
     * @return
     */
    private static CreateCollectionParam initCreateCollectionParam(String collName) {
        return CreateCollectionParam.newBuilder()
                .withName(collName)
```

```
            .withShardNum(3)
            .withReplicaNum(2)
            .withDescription("test collection0")
            .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))
            .build();
    }


    /**
     * 测试 Filter
     */
    public static void testFilter() {
        System.out.println("---------------------- testFilter ----------------------");
        System.out.println("\tres: " + new Filter("author=\"jerry\"")
                .and("a=1")
                .or("r=\"or\"")
                .orNot("rn=2")
                .andNot("an=\"andNot\"")
                .getCond());
        System.out.println("\tres: " + Filter.in("key", Arrays.asList("v1", "v2", "v3")));
        System.out.println("\tres: " + Filter.in("key", Arrays.asList(1, 2, 3)));
    }



}
```

使用 AI 套件上传文件

最近更新时间：2023-12-12 15:36:11

AI 套件是腾讯云向量数据库（Tencent Cloud VectorDB）提供的一站式文档检索解决方案，包含自动化文档解析、信息补充、向量化、内容检索等能力。用户仅需上传原始文档，数分钟内即可快速构建专属知识库，大幅提高知识接入效率。本文介绍通过 Java SDK 操作 AI 类数据库上传文件并进行内容相似度检索的完整示例。

```
/*
 * Copyright (C) 2023 Tencent Cloud.
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the vectordb-sdk-java), to
 * deal in the Software without restriction, including without limitation the
 * rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is furnished
 * to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package com.tencent.tcvectordb;

import com.tencent.tcvectordb.client.VectorDBClient;
import com.tencent.tcvectordb.exception.VectorDBException;
import com.tencent.tcvectordb.model.*;
import com.tencent.tcvectordb.model.param.collection.*;
import com.tencent.tcvectordb.model.param.collectionView.*;
import com.tencent.tcvectordb.model.param.database.ConnectParam;
import com.tencent.tcvectordb.model.param.dml.*;
import com.tencent.tcvectordb.model.param.entity.AffectRes;
import com.tencent.tcvectordb.model.param.entity.SearchContentInfo;
```

```
import com.tencent.tcvectordb.model.param.enums.ReadConsistencyEnum;
import com.tencent.tcvectordb.utils.JsonUtils;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * VectorDB Java SDK usage example
 */
public class VectorDBExampleWithAI_doc {

    private static final String DBNAME = "db_test-ai";
    private static final String COLL_NAME = "coll-ai-files";
    private static final String COLL_NAME_ALIAS = "alias-coll-ai-files";

    public static void example() throws Exception {
        // 创建VectorDB Client
        ConnectParam connectParam = initConnectParam();
        VectorDBClient client = new VectorDBClient(connectParam,
ReadConsistencyEnum.EVENTUAL_CONSISTENCY);

        // 测试前清理环境
        System.out.println("---------------------- clear before test ----------------------");
        anySafe(() -> clear(client));
        createDatabaseAndCollection(client);
        Map<String, Object> metaDataMap = new HashMap<>();
        metaDataMap.put("author", "Tencent");
        metaDataMap.put("tags", Arrays.asList("Embedding","向量","AI"));
        loadAndSplitText(client, "/Users/test/Downloads/jar/腾讯云向量数据库.md", "腾讯云
向量数据库.md", metaDataMap);
        // 解析加载文件需要等待时间
        Thread.sleep(1000 * 10);

        queryData(client);
        GetFile(client, "腾讯云向量数据库.md");
        updateAndDelete(client);
        deleteAndDrop(client);
    }


    /**
     * init connect parameter
     *
```

```
     * @return {@link ConnectParam}
     */
    private static ConnectParam initConnectParam() {
        System.out.println("\tvdb_url: " + System.getProperty("vdb_url"));
        System.out.println("\tvdb_key: " + System.getProperty("vdb_key"));
        return ConnectParam.newBuilder()
                .withUrl(System.getProperty("vdb_url"))
                .withUsername("root")
                .withKey(System.getProperty("vdb_key"))
                .withTimeout(100)
                .build();
    }

    /**
     * 执行 {@link Runnable} 捕获所有异常
     *
     * @param runnable {@link Runnable}
     */
    private static void anySafe(Runnable runnable) {
        try {
            runnable.run();
        } catch (VectorDBException e) {
            System.err.println(e);
            e.printStackTrace();
        }
    }

    private static void createDatabaseAndCollection(VectorDBClient client) throws
InterruptedException {
        // 1. 创建数据库
        System.out.println("---------------------- create AI Database ----------------------");
        AIDatabase db = client.createAIDatabase(DBNAME);

        // 2. 列出所有数据库
        System.out.println("---------------------- listDatabase ----------------------");
        List<String> database = client.listDatabase();
        for (String s : database) {
            System.out.println("\tres: " + s);
        }

//        AIDatabase db = client.aiDatabase(DBNAME);

        // 3. 创建 collection
        System.out.println("---------------------- createCollectionView ----------------------");
```

```
        CreateCollectionViewParam collectionParam =
initCreateCollectionViewParam(COLL_NAME);
        db.createCollectionView(collectionParam);

        // 4. 列出所有 collection
        System.out.println("---------------------- listCollectionView ----------------------");
        List<CollectionView> cols = db.listCollectionView();
        for (CollectionView col : cols) {
            System.out.println("\tres: " + col.toString());
        }

        // 5. 设置 collection 别名
        System.out.println("---------------------- setAIAlias ----------------------");
        AffectRes affectRes = db.setAIAlias(COLL_NAME, COLL_NAME_ALIAS);
        System.out.println("\tres: " + affectRes.toString());
        Thread.sleep(5*1000);

        // 6. describe collection
        System.out.println("---------------------- describeCollectionView ----------------------");
        CollectionView descCollRes = db.describeCollectionView(COLL_NAME);
        System.out.println("\tres: " + descCollRes.toString());


        // 7. delete alias
        System.out.println("---------------------- deleteAIAlias ----------------------");
        AffectRes affectRes1 = db.deleteAIAlias(COLL_NAME_ALIAS);
        System.out.println("\tres: " + affectRes1);

        // 8. describe collection
        System.out.println("---------------------- describeCollectionView ----------------------");
        CollectionView descCollRes1 = db.describeCollectionView(COLL_NAME);
        System.out.println("\tres: " + descCollRes1.toString());

    }

    private static void loadAndSplitText(VectorDBClient client, String filePath, String
documentSetName, Map<String, Object> metaDataMap) throws Exception {
        AIDatabase database = client.aiDatabase(DBNAME);
        CollectionView collection = database.describeCollectionView(COLL_NAME);
        LoadAndSplitTextParam param = LoadAndSplitTextParam.newBuilder()

.withLocalFilePath(filePath).withDocumentSetName(documentSetName).Build();
        collection.loadAndSplitText(param, metaDataMap);
    }
```

```
    private static void GetFile(VectorDBClient client, String fileName) {
        AIDatabase database = client.aiDatabase(DBNAME);
        CollectionView collection = database.describeCollectionView(COLL_NAME);
        System.out.println(collection.getFile(fileName, "").toString());
    }


    private static void queryData(VectorDBClient client) {
        AIDatabase database = client.aiDatabase(DBNAME);
        CollectionView collection = database.describeCollectionView(COLL_NAME);

        // query  查询
        // 1. query 用于查询数据
        // 2. 可以通过传入主键 id 列表或 filter 实现过滤数据的目的
        // 3. 如果没有主键 id 列表和 filter 则必须传入 limit 和 offset，类似 scan 的数据扫描功能
        // 4. 如果需要指定部分documentSetName,可以传入documentSetNames
        // 5. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
        System.out.println("---------------------- query ----------------------");
        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.md")).
//                withOutputFields(Arrays.asList("textPrefix", "author", "tags")).
                build();
        List<DocumentSet> qdos = collection.query(queryParam);
        for (DocumentSet doc : qdos) {
            System.out.println("\tres: " + doc.toString());
        }

        // search
        // 1. search 用于检索数据
        // 2. content 为必填参数，使用该参数值检索数据与之符合的数据
        // 3. SearchOption配置搜索参数，开启rerank参数只有collectionView开启词向量精排才
能使用
        // 4. 如果需要指定部分documentSetName,可以传入documentSetNames,检索数据会在
传入的documentSetName对应的文件中搜索
        // 5. 可以通过传入filter实现指定在符合条件的文件中检索
        System.out.println("---------------------- search ----------------------");

        SearchOption option =
SearchOption.newBuilder().withChunkExpand(Arrays.asList(1,1))
                .withRerank(new RerankOption(true, 3))
```

```
            .build();
        SearchByContentsParam searchByContentsParam =
SearchByContentsParam.newBuilder()
            .withContent("什么是向量")
            .withSearchContentOption(option)
            .withFilter(new Filter(Filter.in("author", Arrays.asList("Tencent","tencent"))).
                    and(Filter.include("tags", Arrays.asList("AI","Embedding"))).getCond())
            .withDocumentSetName(Arrays.asList("腾讯云向量数据库.md"))
            .build();
//        System.out.println(qdos.get(0).search(searchByContentsParam).toString());
        List<SearchContentInfo> searchRes =
collection.search(searchByContentsParam);
        int i = 0;
        for (SearchContentInfo doc : searchRes) {
            System.out.println("\tres" +(i++)+": "+ doc.toString());
        }
    }

    private static void updateAndDelete(VectorDBClient client) throws
InterruptedException {
        AIDatabase database = client.aiDatabase(DBNAME);
        CollectionView collection = database.describeCollectionView(COLL_NAME);
        // update
        // 1. update 提供基于 [主键查询] 和 [Filter 过滤] 的部分字段更新或者非索引字段新增
        // 2. 如果需要指定部分documentSetName,可以传入documentSetNames,指定更新的数
据范围
        // 3. filter 限制仅会更新 条件符合的记录
        System.out.println("---------------------- update ----------------------");
        Filter filterParam = new Filter("author=\"Tencent\"");
        CollectionViewConditionParam updateParam = CollectionViewConditionParam
                .newBuilder()
                .withDocumentSetNames(Arrays.asList("腾讯云向量数据库.md"))
                .withFilter(filterParam)
                .build();
        Map<String, Object> updateFieldValues = new HashMap<>();
        updateFieldValues.put("page", 100);
        updateFieldValues.put("author", "tencent");
        updateFieldValues.put("array_test", Arrays.asList("1", "2","5"));
        collection.update(updateParam, updateFieldValues);

        System.out.println(collection.query(10).get(0).toString());

        // delete
        // 1. delete 提供基于[ 主键查询]和[Filter 过滤]的数据删除能力
```

```
        // 2. 如果需要指定部分documentSetName,可以传入documentSetNames,指定删除的数
据范围
        // 3. filter 限制只会删除命中的记录
        System.out.println("---------------------- delete ----------------------");
        Filter filterParam1 = new Filter("author=\"tencent\"");
        CollectionViewConditionParam build = CollectionViewConditionParam
                .newBuilder()
                .withDocumentSetNames(Arrays.asList("腾讯云向量数据库.md"))
                .withFilter(filterParam1)
                .build();
        AffectRes affectRes = collection.deleteDocumentSets(build);
        System.out.println("\tres: " + affectRes.toString());
        System.out.println(collection.query().size());

        // truncate collection
        System.out.println("---------------------- truncate ----------------------");
        database.truncateCollectionView(COLL_NAME);
    }

    private static void deleteAndDrop(VectorDBClient client) {
        AIDatabase database = client.aiDatabase(DBNAME);

        // 删除 collection
        System.out.println("---------------------- dropCollection ----------------------");
        database.dropCollectionView(COLL_NAME);

        // 删除 database
        System.out.println("---------------------- dropDatabase ----------------------");
        client.dropAIDatabase(DBNAME);
    }


    private static void clear(VectorDBClient client) {
        client.dropAIDatabase(DBNAME);
    }


    private static CreateCollectionViewParam initCreateCollectionViewParam(String
collName) {
        return CreateCollectionViewParam.newBuilder()
                .withName(collName)
                .withDescription("test create ai collection")

.withEmbedding(EmbeddingParams.newBuilder().withEnableWordEmbedding(true).wit
hLanguage(LanuageType.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).Build())
            .build();
    }
}
```

新建 Client

最近更新时间：2023-12-18 16:44:52

功能介绍

VectorDBClient()用于创建一个向量数据库的客户端对象，用于与向量数据库服务器连接并进行数据交互。

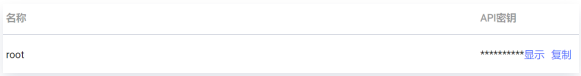
请求示例

```
import com.tencentcloudapi.client.VectorDBClient;
import com.tencentcloudapi.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
VectorDBClient(connectParam,ReadConsistencyEnum.EVENTUAL_CONSISTENCY);
    }
}
```

请求参数

参数名称	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 。 
Timeout	请求超时时间。	是	• 单位：秒。 • 默认值：5。 • 取值范围：大于等于0。
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。

返回信息

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：2023-12-12 09:51:42

功能介绍

createDatabase() 用于创建一个 Base 类的向量数据库。
createAIDatabase() 用于创建一个 AI 类的向量数据库。

请求接口

创建 Base 类接口

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

创建 AI 类接口

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

请求参数

参数	是否必选	参数含义	配置方法及要求
`databaseName`	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]。

请求示例

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

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

返回结果

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。

删除 Database

最近更新时间：2023-12-13 17:18:21

功能介绍

- **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。

请求接口

删除 Base 类数据库

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

删除 AI 类数据库

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

请求参数

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

请求示例

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");
```

返回结果

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：2023-12-12 09:51:42

功能介绍

listDatabase()用于查询集群中所有的向量数据库。

请求接口

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

请求参数

无

请求示例

```
// 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);
}
```

输出数据库名，如下所示。

```
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：2024-04-19 20:01:51

功能介绍

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

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

请求示例

创建 Collection 配置 Embedding 参数

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", BGE_BASE_ZH.getDimension(),
IndexType.HNSW,
                MetricType.COSINE, new HNSWParams(16, 200)))
        .addField(new FilterIndex("bookName", FieldType.String, IndexType.FILTER))
        .addField(new FilterIndex("author", FieldType.String, IndexType.FILTER))
        .withEmbedding(
                Embedding
                        .newBuilder()
                        .withModel(BGE_BASE_ZH)
                        .withField("text")
                        .withVectorField("vector")
                        .build())
        .build();
```

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

创建 Collection 不配置 Embedding 参数

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。

```
// 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 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))

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

请求参数

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	-	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。

	个主分片有多个相同的备份，用来容灾和负载均衡。			○ 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	指定 Filter 索引标量字段名。 **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 为 id。 • 配置其他自定义扩展的可作为 Filter 索引标量字段，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。 • Uinit64：指无符号整数，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。
		indexType	是	指定 Filter 字段的索引类型。 • 若 fieldName 为 id，该参数固定配置为 IndexType.PRIMARY_KEY，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 IndexType.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 。
	VectorIndex	fieldName	是	指定向量索引的索引对象为 vector。
		indexType	是	指定向量索引字段的索引类型。当前所支持的索引类型，及具体索引方式，请参见设计模型中的 Index。 • FLAT：暴力检索，召回率100%，但检索效率低。 • HNSW：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。具体测试数据，请参见性能白皮书的测试结果。 • 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	指定索引对象为 vector 的向量维度。 • 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。
		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。
		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。 • 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	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]。
Embedding	设置 Embedding 相关参数	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_BASE_ZH：适用中文，768 维，推荐使用。 • 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。
		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。

删除 Collection

最近更新时间：2023-12-13 16:16:31

功能介绍

dropCollection()用于删除已创建的 Collection。

请求示例

```
// link database， client 为 VectorDBClient() 创建的客户端对象
Database db = client.database("db-test");
// drop
AffectRes affectRes = db.dropCollection("book-vector");
System.out.println("\tres: " + affectRes);
```

请求参数

参数名	是否必选	参数含义	配置方法
collectionName	是	所需删除的 Collection 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]。

返回信息

```
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：2024-01-17 16:14:01

功能介绍

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

请求示例

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

输出参数

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。当前所支持的索引类型及具体索引方式，请参见设计模型中的 Index 。
		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。具体信息，请参见 混合检索 。
embed	Embedd	status	说明该 Collection 是否配置 Embedding 模型。

ding	ing 相关参数		• 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。 • ready：表示当前 Collection 已准备就绪，可正常使用。 • training data：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 index：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：2023-12-12 09:51:43

功能介绍

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

请求示例

```
// link database
Database db = client.database("db-test");
// describe collection
Collection coll = db.describeCollection("book-emb");
System.out.println(coll.toString());
```

请求参数

参数名	是否必选	参数含义	配置方法
collectionName	是	所需查询的 Collection 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]。

输出参数

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。当前所支持的索引类型及具体索引方式，请参见设计模型中的 Index 。
		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**。具体信息，请参见 混合检索 。
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。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training data**：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 index**：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：2023-12-12 09:51:43

功能介绍

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。

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。

请求示例

```
// link database
Database db = client.database("db-test");
// truncate
AffectRes affectRes = db.truncateCollections("book-vector");
System.out.println("\tres: " + affectRes);
```

请求参数

参数名	是否必选	参数含义	配置方法
collectionName	是	所需删除的 Collection 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]。

返回信息

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

返回参数

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：2024-03-07 19:38:21

功能介绍

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

```
// 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).wit
hLanguage(LanuageType.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).Build())
            .build();
// Create CollectionView
CollectionView db.createCollectionView(collectionParam);
```

请求参数

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		取值如下所示： • ZH：中文。 • EN：英文。 • MULTI：多语言。
		EnableWordEmbedding		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。
		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。
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	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

	为文件 ID 构建主键索引，为文件名构建 Filter 索引。			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。
		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 。

返回说明

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：2023-12-13 16:16:31

功能介绍

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

请求示例

```
// 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);
```

请求参数

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]。

返回信息

```
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：2024-02-27 11:11:51

功能介绍

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

请求示例

```
// 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());
}
```

返回信息

```
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
  },
  "embedding": {
    "language": "zh",
    "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"
    }
  ]
}
```

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 /ai/alias/set 。
createTime	-	-	显示 CollectionView 的创建时间。
descripti	-	-	显示 CollectionView 的描述信息。

on			
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。
Indexes	默认以 documentSetId 文件 ID 创建	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` 。

	主键索引	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：2024-01-17 17:19:01

功能介绍

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

请求示例

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

请求参数

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]。

返回信息

```
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
  },
  "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 setAIAlias()。
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。
		appendKeyw	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。

		ordsToChunk	• true：将全文的 keywords 追加到切分后的段落。
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：2023-12-13 16:16:31

功能介绍

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。

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。

请求示例

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

请求参数

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]。

返回信息

```
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：2023-12-12 16:06:33

功能介绍

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

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，仅需通过该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。

请求示例

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
	是	指定需创建别	使用 listCollections() 获取指定数据库名下的

collectionName		名的 Collection 名。	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 affectRes = 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 Collection View 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]。

返回信息

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：2023-12-12 09:51:43

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。

请求示例

删除 Collection 别名

```
// link database
Database db = client.database("db-test");
// delete alias
AffectRes affectRes2 = 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 affectRes1 = db.deleteAIAlias("alias-sdk-test");
System.out.println("\tres: " + affectRes);
```

输出信息，如下所示。

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

请求参数

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]。

返回参数

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：2023-12-13 16:16:32

功能介绍

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。

请求示例

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。

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-emb");
// upsert
Document doc1 = Document.newBuilder()
        .withId("0001")
        .addDocField(new DocField("text", "富贵功名，前缘分定，为人切莫欺心。"))
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 21))
        .addDocField(new DocField("segment", "富贵功名，前缘分定，为人切莫欺心。"))
        .build();
Document doc2 = Document.newBuilder()
        .withId("0002")
        .addDocField(new DocField("text",
                "正大光明，忠良善果弥深。些些狂妄天加谴，眼前不遇待时临。"))
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 22))
        .addDocField(new DocField("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。"))
        .build();
```

```
Document doc3 = Document.newBuilder()
        .withId("0003")
        .addDocField(new DocField("text", "细作探知这个消息，飞报吕布。"))
        .addDocField(new DocField("bookName", "三国演义"))
        .addDocField(new DocField("author", "罗贯中"))
        .addDocField(new DocField("page", 23))
        .addDocField(new DocField("segment", "细作探知这个消息，飞报吕布。"))
        .build();
Document doc4 = Document.newBuilder()
                .withId("0004")
                .addDocField(new DocField("text", "富贵功名，前缘分定，为人切莫欺
心。"))
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .addDocField(new DocField("page", 24))
                .addDocField(new DocField("segment", "富贵功名，前缘分定，为人切莫欺
心。"))
                .build(),
 Document doc5 = Document.newBuilder()
                .withId("0005")
                .addDocField(new DocField("text",
                        "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，可往投之。"))
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .addDocField(new DocField("page", 25))
                .addDocField(new DocField("segment",
                        "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，可往投之。"))
                .build()));

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

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// upsert
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("segment", "富贵功名，前缘分定，为人切莫欺心。"))
        .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("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。"))
        .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("segment", "细作探知这个消息，飞报吕布。"))
        .build();
Document doc4 = Document.newBuilder()
                .withId("0004")
                .withVector(Arrays.asList(0.2123, 0.24, 0.213))
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .addDocField(new DocField("page", 24))
                .addDocField(new DocField("segment", "富贵功名，前缘分定，为人切莫欺
心。"))
                .build(),
 Document doc5 = Document.newBuilder()
                .withId("0005")
                .withVector(Arrays.asList(0.2123, 0.25, 0.213))
```

```
            .addDocField(new DocField("bookName", "三国演义"))
            .addDocField(new DocField("author", "罗贯中"))
            .addDocField(new DocField("page", 25))
            .addDocField(new DocField("segment",
                    "布大惊，与陈宫商议。宫曰：“闻刘玄德新领徐州，可往投之。"))
            .build()));

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

请求参数

参数名称	参数含义	子参数	是否必选	配置方法
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	表示文档的向量值，请务必使用32位浮点数存储向量数据。 **说明：** 如果业务无需使用腾讯云向量数据库（Tencent Cloud Vector DB）的 Embedding 功能做向量化，则配置该参数，写入向量数据，而无需配置 Embedding 参数 **text**（创建 Collection 时，Embedding 参数 **field** 对应

				指定的文本字段名，示例中为 text）。
		DocField	否	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。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。
				其他标量字段，用于存储文档的其他信息。例如：bookName、author、page。
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 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。

返回信息

```
res: AffectRes{affectedCount=1, 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：2023-12-13 17:18:21

功能介绍

基于精确匹配的查询方式，query() 用于精确查找与查询条件完全匹配的向量，具体支持如下功能。

- 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。

请求示例

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// query
QueryParam queryParam = QueryParam.newBuilder()
        .withDocumentIds(Arrays.asList("0001", "0002", "0003", "0004", "0005"))
        // 使用 filter 过滤数据
        .withFilter(new Filter("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(2)
        // 偏移
        .withOffset(1)
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName"))
        // 是否返回 vector 数据
        .withRetrieveVector(true)
        .build();
List<Document> qdos = collection.query(queryParam);
for (Document doc : qdos) {
    System.out.println("\tres: " + doc.toString());
}
```

请求参数

参数	是否必选	参数含义	参数配置
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 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 Document 数量。	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 **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。
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。

输出参数

```
res: {"id":"0003","vector":
[0.21230000257492065,0.23000000417232513,0.21299999952316284],"bookName"
:"三国演义"}
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"bookName"
:"三国演义"}
```

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

向量数据相似性检索

最近更新时间：2023-12-13 10:58:21

功能介绍

基于相似度匹配的查询方式，**search()** 接口用于查找与给定查询向量相似的文档，返回指定的 Top K 个最相似的文档，并支持搭配自定义的标量字段的 Filter 表达式一并进行相似度检索。

请求示例

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// search by filter
System.out.println("-searchByFilter----------");
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(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)
        // 设置标量字段的 Filter 表达式，过滤所需查询的文档
        .withFilter(new Filter("bookName=\"三国演义\""))
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName"))
        .build();
// 输出相似度检索结果
List<List<Document>> svDocs = collection.search(searchByVectorParam);
int i = 0;
for (List<Document> docs : svDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

请求参数

参数名	是否必选	参数含义	配置方法
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。	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')
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。

输出参数

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 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":"0005","vector":
[0.21230000257492065,0.25,0.21299999952316284],"score":0.9788496494293213,"
bookName":"三国演义"}
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"score":0.97
53975868225098,"bookName":"三国演义"}
res: 1
res: {"id":"0005","vector":
[0.21230000257492065,0.25,0.21299999952316284],"score":0.9942749738693237,"
bookName":"三国演义"}
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"score":0.99
2622435092926,"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。

Doc ID 相似性检索

最近更新时间：2023-12-13 10:58:21

功能介绍

基于相似度匹配的查询方式，**searchById()**根据指定的 Document id 进行相似度查询，并返回指定的 Top K 个最相似的 Document。

请求示例

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// search by id
SearchByIdParam searchByIdParam = SearchByIdParam.newBuilder()
        .withDocumentIds(Arrays.asList("0001", "0002", "0003", "0004", "0005"))
        // 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)
        // 使用 filter 过滤数据
        .withFilter(new Filter("bookName=\"三国演义\""))
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName"))
        .build();
List<List<Document>> siDocs = collection.searchById(searchByIdParam);
int i = 0;
for (List<Document> docs : siDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

请求参数

参数	是否必选	参数含义	配置方法
	是	待查询的文档	每个 ID 长度限制为[1,128]。数组元素数量最大为20。

DocumentIds		ID。	
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。	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')`

	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。

OutputFields			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 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":"0003","vector":
[0.21230000257492065,0.23000000417232513,0.21299999952316284],"score":0.99
90617632865906,"bookName":"三国演义"}
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"score":0.99
79545474052429,"bookName":"三国演义"}
res: 1
res: {"id":"0003","vector":
[0.21230000257492065,0.23000000417232513,0.21299999952316284],"score":0.99
97729659080505,"bookName":"三国演义"}
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"score":0.99
91196990013123,"bookName":"三国演义"}
res: 2
res: {"id":"0003","vector":
[0.21230000257492065,0.23000000417232513,0.21299999952316284],"score":1.00
00000119209289 6,"bookName":"三国演义"}
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"score":0.99
97869729995728,"bookName":"三国演义"}
```

```
res: 3
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"score":1.0,"
bookName":"三国演义"}
res: {"id":"0005","vector":
[0.21230000257492065,0.25,0.21299999952316284],"score":0.9998002052307129,"
bookName":"三国演义"}
res: 4
res: {"id":"0005","vector":
[0.21230000257492065,0.25,0.21299999952316284],"score":0.9999998807907104,"
bookName":"三国演义"}
res: {"id":"0004","vector":
[0.21230000257492065,0.23999999463558197,0.21299999952316284],"score":0.99
98002052307129,"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。

文本信息相似性检索

最近更新时间：2023-12-13 10:58:21

功能介绍

若在创建 Collection 时，已配置 Embedding 模型，则可使用 **searchByEmbeddingItems()** 接口输入原始文本进行相似度查询，并支持搭配自定义的标量字段的 Filter 表达式一并进行相似度检索，返回指定的 Top K 个最相似的 Document。

请求示例

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。

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-emb");
// search by id

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(5)
        // 使用 filter 过滤数据
        .withFilter(new Filter("bookName=\"三国演义\""))
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("id", "bookName"))
        .build();
List<List<Document>> siDocs =
collection.searchByEmbeddingItems(searchByEmbeddingItemsParam);
int i = 0;
for (List<Document> docs : siDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println(doc.toString());
    }
```

```
}
```

请求参数

参数	是否必选	参数含义	配置方法
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。	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')`
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。

输出参数

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":"0004","score":0.7809984683990479,"bookName":"三国演义"}
res: {"id":"0005","score":0.7298625707626343,"bookName":"三国演义"}
res: {"id":"0003","score":0.7266470193862915,"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。

删除 Document

最近更新时间：2023-12-13 16:16:32

功能介绍

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。

接口约束

索引类型为 FLAT，不支持删除。

请求示例

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// delete
DeleteParam deleteParam = DeleteParam.newBuilder()
        .withDocumentIds(Arrays.asList("0001", "0003"))
        .withFilter(new Filter("bookName=\"三国演义\""))
        .build();
AffectRes affectRes = collection.delete(deleteParam);
System.out.println("\tres: " + affectRes);
```

请求参数

参数名	是否必选	参数含义	配置方法
DocumentIds	是	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 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')`

返回信息

```
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。

更新数据

最近更新时间：2023-12-13 16:16:32

功能介绍

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。

请求示例

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。

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-emb");
//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 = collection.update(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。

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
//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
List<Document> qdos = collection.update(updateParam, updateDoc);
for (Document doc : qdos) {
    System.out.println(doc.toString());
}
```

请求参数

参数	参数含义	子参数	是否必选	子参数含义
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 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')`
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]。

返回信息

```
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。

DocumentSet 操作
上传文件

最近更新时间：2024-01-17 17:19:01

功能介绍

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

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。

请求示例

```
// 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.md")
            .withDocumentSetName("腾讯云向量数据库.md")
            .withSplitterProcess(SplitterPreprocessParams.newBuilder()
                .withAppendKeywordsToChunkEnum(true)
                .withAppendTitleToChunkEnum(false)
                .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);
```

请求参数

参数名	子参数	是否必选	参数含义
	–	是	本地上传文件路径。

LocalFilePath			
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。
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。
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。

返回消息

> **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：2024-04-09 16:16:11

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。

- **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。**

请求示例

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.md";
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());
```

请求参数

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	-

输出信息


```
{
  "documentSetName": "腾讯云向量数据库.md",
  "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
},
"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]"
   }
  ]
}
```

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。
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。
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 pdate 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：2024-04-09 16:16:11

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。

- 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。

请求示例

```
// 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.md")));
```

请求参数

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：[1,16384]。
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。

返回信息


```
{
  "code":0,
  "msg":"Operation success",
  "count":10,
  "requestId":"ade87e83c8d0a61790139728d0db4007",
  "documentSetId":"1192057501114437632",
  "documentSetName":"腾讯云向量数据库.md",
  "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。

精确查询文件

最近更新时间：2024-01-17 17:19:01

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。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 Metadata 信息对应字段 Filter 表达式，过滤需查找的文件。

请求示例

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。

```
// 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.md"))
        .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.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': '2023-12-01 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());
```

请求参数

子参数	是否必	配置方法及要求

	选	
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。

返回消息

res: {"documentSetId":"11920675605150474 24","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\":\"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}","author":"Tencent","tags":"[Embedding, 向量, 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。
	indexedStatu	文件预处理、Embedding 向量化的状态。 • New：等待解析。

	s	• 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。
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 pdate 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：2024-04-09 16:16:11

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。**

请求示例

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.md` 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(Arrays.asList(1,1)))
            .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.md"))
            .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(Arrays.a
sList(1,1)))
            .withFilter(new Filter(Filter.in("author",
Arrays.asList("Tencent","tencent"))).and(Filter.include("tags",
Arrays.asList("AI","Embedding"))))
            .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());
}
```

请求参数

参数名称	是否必选	参数含义及配置方法
Content	否	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[1,20]。
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	指定返回最相似的 Top K 的 K 的值。

返回参数

如下为检索到的相似数据，返回最相似的 Top 3 条数据。


```
res0: {
  "score": 0.8938464522361755,
  "data": {
    "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,
    "next": [
      "### 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"
    ],
```

```
    "pre": [
     "## 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"
    ]
   },
   "documentSet": {
    "documentSetName": "\"腾讯云向量数据库.md\"",
    "documentSetId": "\"1182525034158882816\"",
    "docFields": [
     {
      "name": "author",
      "value": "Tencent"
     },
     {
      "name": "tags",
      "value": [
       "Embedding",
       "向量",
       "AI"
      ]
     }
    ]
   }
}res1: {
   "score": 0.8378515839576721,
   "data": {
    "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,
    "next": [
     "### 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"
    ],
    "pre": [
     "### 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"
    ]
   },
   "documentSet": {
    "documentSetName": "\"腾讯云向量数据库.md\"",
    "documentSetId": "\"1182525034158882816\"",
```

```
    "docFields": [
     {
      "name": "author",
      "value": "Tencent"
     },
     {
      "name": "tags",
      "value": [
       "Embedding",
       "向量",
       "AI"
      ]
     }
    ]
   }
}res2: {
  "score": 0.8152828216552734,
  "data": {
    "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,
    "next": [
     "## 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"
    ],
    "pre": [
     "### 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"
    ]
  },
  "documentSet": {
    "documentSetName": "\"腾讯云向量数据库.md\"",
    "documentSetId": "\"1182525034158882816\"",
    "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，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。
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。
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：2023-12-13 16:16:32

功能介绍

deleteDocumentSets() 接口用于删除存储于 CollectionView 的文件。

- 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。

请求示例

根据文件名过滤需删除的文件

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// 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.md"))
        .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
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// 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
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// 配置删除条件，以便检索需删除的文件
CollectionViewConditionParam build = CollectionViewConditionParam
        .newBuilder()
        .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());
```

请求参数

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]。
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。

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')`

返回信息

```
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：2023-12-13 16:16:32

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。

- 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。

> **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。

请求示例

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
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// 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.md"))
        .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
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// 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.md"))
        .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
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
```

```
// 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.md"))
        .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());
```

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
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并集。
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并集。
Filter	配置 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>

	表达式过滤需更新的文件。			<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]。

返回信息

```
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：2023-12-12 14:58:11

功能介绍

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。

接口约束

> ⚠ **注意：**
> - 索引重建过程中 collection 禁止写入、读取。
> - 重建索引需要新的内存来构建索引。

请求示例

```
// link database
Database db = client.database("db-test");
// link collection
Collection collection = db.collection("book-vector");
// rebuild
RebuildIndexParam rebuildIndexParam = RebuildIndexParam.newBuilder()
        .withDropBeforeRebuild(false)
        .withThrottle(1)
        .build();
collection.rebuildIndex(rebuildIndexParam);
```

请求参数

参数	是否必选	参数含义	配置方法及要求
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。 > ⓘ **说明：** 重建索引需要占用额外的内存空间，数据量越	取值如下所示： • **true**：重建之前，先删除旧索引在重建索引。 > ⓘ **说明：**

		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，可先删除旧索引，在新索引还没有创建完成之前，该表无法正常读写。 • **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。 **说明：** 内存资源足够的情况下，可不删除旧索引。在新索引还没有创建完成之前，该表可读数据，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**：CPU 核数为 1，即仅使用 CPU 1 核进行模型训练，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。

返回参数

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。

- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training data**：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 index**：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。

Go SDK
SDK 准备

最近更新时间：2023-12-26 15:12:01

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	SDK 安装
GO	Go 1.15 或更高版本	https://github.com/Tencent/vectordatabase-sdk-go	`go get -u github.com/tencent/vectordatabase-sdk-go/tcvectordb`

GO Demo
写入原始文本并检索（Embedding）

最近更新时间：2023-12-26 15:12:02

腾讯云向量数据库（Tencent Cloud VectorDB）目前已支持文本 Embedding 模型，能够覆盖多种主流语言的向量转换。本文给出通过 Go SDK 写入或更新原始文本，并进行精确查询或相似度检索的完整示例，便于您更加高效地管理和使用向量数据。

```
package main

import (
    "context"
    "log"
    "time"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)

type EmbeddingDemo struct {
    client *tcvectordb.Client
}

func NewEmbeddingDemo(url, username, key string) (*EmbeddingDemo, error) {
    cli, err := tcvectordb.NewClient(url, username, key,
&tcvectordb.ClientOption{ReadConsistency: tcvectordb.EventualConsistency})
    if err != nil {
        return nil, err
    }
    // disable/enable http request log print
    // cli.Debug(false)
    return &EmbeddingDemo{client: cli}, nil
}

func (d *EmbeddingDemo) Clear(ctx context.Context, database string) error {
    log.Println("------------------------- DropDatabase -------------------------")
    result, err := d.client.DropDatabase(ctx, database)
    if err != nil {
        return err
    }
    log.Printf("drop database result: %+v", result)
    return nil
}
```

```
func (d *EmbeddingDemo) DeleteAndDrop(ctx context.Context, database, collection
string) error {
    // 删除collection，删除collection的同时，其中的数据也将被全部删除
    log.Println("------------------------- DropCollection -------------------------")
    colDropResult, err := d.client.Database(database).DropCollection(ctx, collection)
    if err != nil {
        return err
    }
    log.Printf("drop collection result: %+v", colDropResult)


    log.Println("------------------------- DropDatabase -------------------------")
    // 删除db，db下的所有collection都将被删除
    dbDropResult, err := d.client.DropDatabase(ctx, database)
    if err != nil {
        return err
    }
    log.Printf("drop database result: %+v", dbDropResult)
    return nil
}

func (d *EmbeddingDemo) CreateDBAndCollection(ctx context.Context, database,
collection, alias string) error {
    // 创建DB--'book'
    log.Println("------------------------- CreateDatabase -------------------------")
    db, err := d.client.CreateDatabase(ctx, database)
    if err != nil {
        return err
    }

    log.Println("------------------------- ListDatabase -------------------------")
    dbList, err := d.client.ListDatabase(ctx)
    if err != nil {
        return err
    }
    for _, db := range dbList.Databases {
        log.Printf("database: %s", db.DatabaseName)
    }

    log.Println("----------------------- CreateCollection -----------------------")
    // 新建 Collection
    // 第一步，设计索引（不是设计表格的结构）
    // 1.【重要的事】向量对应的文本字段不要建立索引，会浪费较大的内存，并且没有任何作
用。
```

```
    // 2.【必须的索引】：主键 id、向量字段 vector 这两个字段目前是固定且必须的，参考下面的例子；
    // 3.【其他索引】：检索时需作为条件查询的字段，比如要按书籍的作者进行过滤，这个时候author字段就需要建立索引，
    //    否则无法在查询的时候对 author 字段进行过滤，不需要过滤的字段无需加索引，会浪费内存；
    // 4. 向量数据库支持动态 Schema，写入数据时可以写入任何字段，无需提前定义，类似 MongoDB.
    // 5. 例子中创建一个书籍片段的索引，例如书籍片段的信息包括 {id, vector, segment, bookName, page},
    //    id 为主键需要全局唯一，segment 为文本片段, vector 为 segment 的向量，vector 字段需要建立向量索引，假如我们在查询的时候要查询指定书籍
    //    名称的内容，这个时候需要对bookName建立索引，其他字段没有条件查询的需要，无需建立索引。
    // 6. 创建带 Embedding 的 collection 需要保证设置的 vector 索引的维度和 Embedding 所用模型生成向量维度一致，模型及维度关系：
    //    ------------------------------------------------
    //          bge-base-zh              | 768
    //          bge-large-zh             | 1024
    //          m3e-base                 | 768
    //          text2vec-large-chinese   | 1024
    //          e5-large-v2              | 1024
    //          multilingual-e5-base     | 768
    //    ------------------------------------------------
    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.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})
```

```
    ebd := &tcvectordb.Embedding{VectorField: "vector", Field: "text", Model:
tcvectordb.BGE_BASE_ZH}
    // 第二步：创建 Collection
    // 创建支持 Embedding 的 Collection
    db.WithTimeout(time.Second * 30)
    _, err = db.CreateCollection(ctx, collection, 3, 0, "test collection", index,
&tcvectordb.CreateCollectionParams{
        Embedding: ebd,
    })
    if err != nil {
        return err
    }

    log.Println("------------------------ ListCollection ------------------------")
    // 列出所有 Collection
    collListRes, err := db.ListCollection(ctx)
    if err != nil {
        return err
    }
    for _, col := range collListRes.Collections {
        log.Printf("ListCollection: %+v", col)
    }

    log.Println("--------------------------- SetAlias ---------------------------")
    // 设置 Collection 的 alias
    _, err = db.SetAlias(ctx, collection, alias)
    if err != nil {
        return err
    }

    log.Println("---------------------- DescribeCollection ----------------------")
    // 查看 Collection 信息
    colRes, err := db.DescribeCollection(ctx, collection)
    if err != nil {
        return err
    }
    log.Printf("DescribeCollection: %+v", colRes)

    log.Println("-------------------------- DeleteAlias --------------------------")
    // 删除 Collection 的 alias
    delAliasRes, err := db.DeleteAlias(ctx, alias)
    if err != nil {
        return err
    }
    log.Printf("DeleteAliasResult: %v", delAliasRes)
```

```
    return nil
}

func (d *EmbeddingDemo) UpsertData(ctx context.Context, database, collection
string) error {
    // 获取 Collection 对象
    coll := d.client.Database(database).Collection(collection)

    log.Println("---------------------------- Upsert ----------------------------")
    // upsert 写入数据，可能会有一定延迟
    // 1. 支持动态 Schema，除了 id、vector 字段必须写入，可以写入其他任意字段；
    // 2. upsert 会执行覆盖写，若文档id已存在，则新数据会直接覆盖原有数据(删除原有数据，再
插入新数据)

    documentList := []tcvectordb.Document{
        {
            Id: "0001",
            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。"},
                "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},
                "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。"},
                "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},
                "segment":  {Val: "细作探知这个消息，飞报吕布。"},
                "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},
                "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。"},
                "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},
                "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。"},
                "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。"},
            },
        },
    }
    result, err := coll.Upsert(ctx, documentList)
    if err != nil {
        return err
    }
    log.Printf("UpsertResult: %+v", result)
    return nil
}

func (d *EmbeddingDemo) QueryData(ctx context.Context, database, collection
string) error {
    // 获取 Collection 对象
    coll := d.client.Database(database).Collection(collection)


    log.Println("---------------------------- Query ----------------------------")
    // 查询
    // 1. query 用于查询数据
    // 2. 可以通过传入主键 id 列表或 filter 实现过滤数据的目的
    // 3. 如果没有主键 id 列表和 filter 则必须传入 limit 和 offset，类似 scan 的数据扫描功能
```

```
    // 4. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
    documentIds := []string{"0001", "0002", "0003", "0004", "0005"}
    filter := tcvectordb.NewFilter(`bookName="三国演义"`)
    outputField := []string{"id", "bookName"}

    result, err := coll.Query(ctx, documentIds, &tcvectordb.QueryDocumentParams{
        Filter:         filter,
        RetrieveVector: true,
        OutputFields:   outputField,
        Limit:          2,
        Offset:         1,
    })
    if err != nil {
        return err
    }
    log.Printf("QueryResult: total: %v, affect: %v", result.Total, result.AffectedCount)
    for _, doc := range result.Documents {
        log.Printf("QueryDocument: %+v", doc)
    }

    log.Println("-------------------------- SearchById --------------------------")
    // searchById
    // 1. searchById 提供按 id 搜索的能力
    // 1. search 提供按照 vector 搜索的能力
    // 2. 支持通过 filter 过滤数据
    // 3. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
    // 4. limit 用于限制每个单元搜索条件的条数，如 vector 传入三组向量，limit 为 3，则 limit
限制的是每组向量返回 top 3 的相似度向量

    // 根据主键 id 查找 Top K 个相似性结果，向量数据库会根据ID 查找对应的向量，再根据向量进
行TOP K 相似性检索
    searchResult, err := coll.SearchById(ctx, []string{"0003"},
&tcvectordb.SearchDocumentParams{
        Filter: filter,
        Params: &tcvectordb.SearchDocParams{Ef: 200},
        Limit:  2,
    })
    if err != nil {
        return err
    }
    for i, item := range searchResult.Documents {
        log.Printf("SearchDocumentResult, index: %d ====================", i)
        for _, doc := range item {
```

```
            log.Printf("SearchDocument: %+v", doc)
        }
    }

    log.Println("------------------------- SearchByText -------------------------")
    // 通过 embedding 文本搜索
    // 1. searchByText 提供基于 embedding 文本的搜索能力，会先将 embedding 内容做
Embedding 然后进行按向量搜索
    // 其他选项类似 search 接口

    // searchByText 返回类型为 Dict，接口查询过程中 embedding 可能会出现截断，如发生截
断将会返回响应 warn 信息，如需确认是否截断可以
    // 使用 "warning" 作为 key 从 Dict 结果中获取警告信息，查询结果可以通过 "documents"
作为 key 从 Dict 结果中获取
    searchResult, err = coll.SearchByText(ctx, map[string][]string{"text": {"细作探知这个
消息，飞报吕布。"}}, &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
        Limit:  2,                          // 指定 Top K 的 K 值
    })
    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 searchResult.Documents {
        log.Printf("SearchDocumentResult, index: %d ====================", i)
        for _, doc := range item {
            log.Printf("SearchDocument: %+v", doc)
        }
    }
    return nil
}

func (d *EmbeddingDemo) UpdateAndDelete(ctx context.Context, database, collection
string) error {
    // 获取 Collection 对象
    db := d.client.Database(database)
    coll := db.Collection(collection)

    log.Println("---------------------------- Update ----------------------------")
    // update
    // 1. update 提供基于 [主键查询] 和 [Filter 过滤] 的部分字段更新或者非索引字段新增

    // filter 限制仅会更新 id = "0003"
```

```
documentId := []string{"0001", "0003"}
filter := tcvectordb.NewFilter(`bookName="三国演义"`)
updateField := map[string]tcvectordb.Field{
    "page": {Val: 24},
}
result, err := coll.Update(ctx, tcvectordb.UpdateDocumentParams{
    QueryIds:     documentId,
    QueryFilter:  filter,
    UpdateFields: updateField,
})
if err != nil {
    return err
}
log.Printf("UpdateResult: %+v", result)

log.Println("---------------------------- Delete ----------------------------")
// delete
// 1. delete 提供基于 [主键查询] 和 [Filter 过滤] 的数据删除能力
// 2. 删除功能会受限于 collection 的索引类型，部分索引类型不支持删除操作

// filter 限制只会删除 id="0001" 成功
filter = tcvectordb.NewFilter(`bookName="西游记"`)
delResult, err := coll.Delete(ctx, tcvectordb.DeleteDocumentParams{
    Filter:      filter,
    DocumentIds: documentId,
})
if err != nil {
    return err
}
log.Printf("DeleteResult: %+v", delResult)

log.Println("------------------------- RebuildIndex -------------------------")
// rebuild_index
// 索引重建，重建期间不支持写入
indexRebuildRes, err := coll.RebuildIndex(ctx)
if err != nil {
    return err
}
log.Printf("%+v", indexRebuildRes)

log.Println("---------------------- TruncateCollection ----------------------")
// truncate_collection
// 清空 Collection
time.Sleep(time.Second * 5)
truncateRes, err := db.TruncateCollection(ctx, collection)
```

```
    if err != nil {
        return err
    }
    log.Printf("TruncateResult: %+v", truncateRes)
    return nil
}

func printErr(err error) {
    if err != nil {
        log.Fatal(err)
    }
}

func main() {
    database := "go-sdk-demo-db"
    collectionName := "go-sdk-demo-em-col"
    collectionAlias := "go-sdk-demo-em-alias"

    ctx := context.Background()
    testVdb, err := NewEmbeddingDemo("vdb http url or ip and post", "vdb username",
"key get from web console")
    printErr(err)
    err = testVdb.Clear(ctx, database)
    printErr(err)
    err = testVdb.CreateDBAndCollection(ctx, database, collectionName,
collectionAlias)
    printErr(err)
    err = testVdb.UpsertData(ctx, database, collectionName)
    printErr(err)
    err = testVdb.QueryData(ctx, database, collectionName)
    printErr(err)
    err = testVdb.UpdateAndDelete(ctx, database, collectionName)
    printErr(err)
    err = testVdb.DeleteAndDrop(ctx, database, collectionName)
    printErr(err)
}
```

写入向量数据并检索

最近更新时间：2023-12-26 15:12:02

本文给出通过 Go SDK 写入或更新向量数据，并进行精确查询与相似度检索的完整示例，便于您更加高效地管理和使用向量数据。

```
package main

import (
    "context"
    "log"
    "time"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)

type Demo struct {
    client *tcvectordb.Client
}

func NewDemo(url, username, key string) (*Demo, error) {
    cli, err := tcvectordb.NewClient(url, username, key,
&tcvectordb.ClientOption{ReadConsistency: tcvectordb.EventualConsistency})
    if err != nil {
        return nil, err
    }
    // disable/enable http request log print
    // cli.Debug(false)
    return &Demo{client: cli}, nil
}

func (d *Demo) Clear(ctx context.Context, database string) error {
    log.Println("------------------------- DropDatabase -------------------------")
    result, err := d.client.DropDatabase(ctx, database)
    if err != nil {
        return err
    }
    log.Printf("drop database result: %+v", result)
    return nil
}

func (d *Demo) DeleteAndDrop(ctx context.Context, database, collection string) error
{
```

```
    // 删除collection，删除collection的同时，其中的数据也将被全部删除
    log.Println("------------------------- DropCollection -------------------------")
    colDropResult, err := d.client.Database(database).DropCollection(ctx, collection)
    if err != nil {
        return err
    }
    log.Printf("drop collection result: %+v", colDropResult)

    log.Println("------------------------- DropDatabase -------------------------")
    // 删除db，db下的所有collection都将被删除
    dbDropResult, err := d.client.DropDatabase(ctx, database)
    if err != nil {
        return err
    }
    log.Printf("drop database result: %+v", dbDropResult)
    return nil
}

func (d *Demo) CreateDBAndCollection(ctx context.Context, database, collection,
alias string) error {
    // 创建DB--'book'
    log.Println("------------------------- CreateDatabase -------------------------")
    db, err := d.client.CreateDatabase(ctx, database)
    if err != nil {
        return err
    }

    log.Println("------------------------- ListDatabase -------------------------")
    dbList, err := d.client.ListDatabase(ctx)
    if err != nil {
        return err
    }
    for _, db := range dbList.Databases {
        log.Printf("database: %s", db.DatabaseName)
    }

    log.Println("------------------------ CreateCollection ------------------------")
    // 创建 Collection

    // 第一步，设计索引（不是设计 Collection 的结构）
    // 1.【重要的事】向量对应的文本字段不要建立索引，会浪费较大的内存，并且没有任何作
用。
    // 2.【必须的索引】：主键id、向量字段 vector 这两个字段目前是固定且必须的，参考下面的
例子；
```

```
    // 3.【其他索引】：检索时需作为条件查询的字段，比如要按书籍的作者进行过滤，这个时候 author 字段就需要建立索引，
    //    否则无法在查询的时候对 author 字段进行过滤，不需要过滤的字段无需加索引，会浪费内存；
    // 4. 向量数据库支持动态 Schema，写入数据时可以写入任何字段，无需提前定义，类似 MongoDB.
    // 5. 例子中创建一个书籍片段的索引，例如书籍片段的信息包括 {id, vector, segment, bookName, author, page},
    //    id 为主键需要全局唯一，segment 为文本片段, vector 字段需要建立向量索引，假如我们在查询的时候要查询指定书籍
    //    名称的内容，这个时候需要对 bookName 建立索引，其他字段没有条件查询的需要，无需建立索引。
    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})

    // 第二步：创建 Collection
    // 创建collection耗时较长，需要调整客户端的timeout
    // 这里以三可用区实例作为参考，具体实例不同的规格所支持的shard和replicas区间不同，需要参考官方文档
    db.WithTimeout(time.Second * 30)
    _, err = db.CreateCollection(ctx, collection, 3, 0, "test collection", index)
    if err != nil {
        return err
    }

    log.Println("------------------------ ListCollection -------------------------")
    // 列出所有 Collection
```

```
    collListRes, err := db.ListCollection(ctx)
    if err != nil {
        return err
    }
    for _, col := range collListRes.Collections {
        log.Printf("ListCollection: %+v", col)
    }

    log.Println("---------------------------- SetAlias ----------------------------")
    // 设置 Collection 的 alias
    _, err = db.SetAlias(ctx, collection, alias)
    if err != nil {
        return err
    }

    log.Println("----------------------- DescribeCollection -----------------------")
    // 查看 Collection 信息
    colRes, err := db.DescribeCollection(ctx, collection)
    if err != nil {
        return err
    }
    log.Printf("DescribeCollection: %+v", colRes)

    log.Println("--------------------------- DeleteAlias ---------------------------")
    // 删除 Collection 的 alias
    delAliasRes, err := db.DeleteAlias(ctx, alias)
    if err != nil {
        return err
    }
    log.Printf("DeleteAliasResult: %v", delAliasRes)
    return nil
}

func (d *Demo) UpsertData(ctx context.Context, database, collection string) error {
    // 获取 Collection 对象
    coll := d.client.Database(database).Collection(collection)

    log.Println("----------------------------- Upsert -----------------------------")
    // upsert 写入数据，可能会有一定延迟
    // 1. 支持动态 Schema，除了 id、vector 字段必须写入，可以写入其他任意字段；
    // 2. upsert 会执行覆盖写，若文档id已存在，则新数据会直接覆盖原有数据(删除原有数据，再
插入新数据)
    documentList := []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。"},
        },
      },
      {
        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。"},
        },
      },
      {
        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。"},
        },
      },
      {
        Id:     "0004",
        Vector: []float32{0.2123, 0.24, 0.213},
        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: []float32{0.2123, 0.25, 0.213},
        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, err := coll.Upsert(ctx, documentList)
if err != nil {
    return err
}
log.Printf("UpsertResult: %+v", result)
return nil
}

func (d *Demo) QueryData(ctx context.Context, database, collection string) error {
    // 获取 Collection 对象
    coll := d.client.Database(database).Collection(collection)

    log.Println("---------------------------- Query ----------------------------")
    // 查询
    // 1. query 用于查询数据
    // 2. 可以通过传入主键 id 列表或 filter 实现过滤数据的目的
    // 3. 如果没有主键 id 列表和 filter 则必须传入 limit 和 offset，类似 scan 的数据扫描功能
    // 4. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些 field，不指定默认全部返回
    documentIds := []string{"0001", "0002", "0003", "0004", "0005"}
    filter := tcvectordb.NewFilter(`bookName="三国演义"`)
    outputField := []string{"id", "bookName"}

    result, err := coll.Query(ctx, documentIds, &tcvectordb.QueryDocumentParams{
        Filter:         filter,
        RetrieveVector: true,
        OutputFields:   outputField,
        Limit:          2,
        Offset:         1,
    })
    if err != nil {
        return err
    }
    log.Printf("QueryResult: total: %v, affect: %v", result.Total, result.AffectedCount)
    for _, doc := range result.Documents {
        log.Printf("QueryDocument: %+v", doc)
    }
```

```
    log.Println("--------------------------- SearchById ---------------------------")
    // searchById
    // 1. searchById 提供按 id 搜索的能力
    // 1. search 提供按照 vector 搜索的能力
    // 2. 支持通过 filter 过滤数据
    // 3. 如果仅需要部分 field 的数据，可以指定 output_fields 用于指定返回数据包含哪些
field，不指定默认全部返回
    // 4. limit 用于限制每个单元搜索条件的条数，如 vector 传入三组向量，limit 为 3，则 limit
限制的是每组向量返回 top 3 的相似度向量

    // 根据主键 id 查找 Top K 个相似性结果，向量数据库会根据ID 查找对应的向量，再根据向量进
行TOP K 相似性检索
    searchResult, err := coll.SearchById(ctx, []string{"0003"},
&tcvectordb.SearchDocumentParams{
        Filter: filter,
        Params: &tcvectordb.SearchDocParams{Ef: 200},
        Limit:  2,
    })
    if err != nil {
        return err
    }
    for i, item := range searchResult.Documents {
        log.Printf("SearchDocumentResult, index: %d ====================", i)
        for _, doc := range item {
            log.Printf("SearchDocument: %+v", doc)
        }
    }
    log.Println("--------------------------- Search ---------------------------")
    // search
    // 1. search 提供按照 vector 搜索的能力
    // 其他选项类似 search 接口

    // 批量相似性查询，根据指定的多个向量查找多个 Top K 个相似性结果
    searchResult, err = coll.Search(ctx,
        [][]float32{{0.3123, 0.43, 0.213}, {0.233, 0.12, 0.97}}, //指定检索向量，最多指定
20个
        &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:          10,                             // 指定 Top K 的 K 值
        })
    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 searchResult.Documents {
        log.Printf("SearchDocumentResult, index: %d ==================", i)
        for _, doc := range item {
            log.Printf("SearchDocument: %+v", doc)
        }
    }
    return nil
}

func (d *Demo) UpdateAndDelete(ctx context.Context, database, collection string)
error {
    // 获取 Collection 对象
    db := d.client.Database(database)
    coll := db.Collection(collection)

    log.Println("---------------------------- Update ----------------------------")
    // update
    // 1. update 提供基于 [主键查询] 和 [Filter 过滤] 的部分字段更新或者非索引字段新增

    // filter 限制仅会更新 id = "0003"
    documentId := []string{"0001", "0003"}
    filter := tcvectordb.NewFilter(`bookName="三国演义"`)
    updateField := map[string]tcvectordb.Field{
        "page": {Val: 24},
    }
    result, err := coll.Update(ctx, tcvectordb.UpdateDocumentParams{
        QueryIds:     documentId,
        QueryFilter:  filter,
        UpdateFields: updateField,
    })
    if err != nil {
        return err
    }
    log.Printf("UpdateResult: %+v", result)

    log.Println("---------------------------- Delete ----------------------------")
    // delete
    // 1. delete 提供基于 [主键查询] 和 [Filter 过滤] 的数据删除能力
    // 2. 删除功能会受限于 collection 的索引类型，部分索引类型不支持删除操作

    // filter 限制只会删除 id="0001" 成功
    filter = tcvectordb.NewFilter(`bookName="西游记"`)
```

```
    delResult, err := coll.Delete(ctx, tcvectordb.DeleteDocumentParams{
        Filter:      filter,
        DocumentIds: documentId,
    })
    if err != nil {
        return err
    }
    log.Printf("DeleteResult: %+v", delResult)

    log.Println("------------------------- RebuildIndex -------------------------")
    // rebuild_index
    // 索引重建，重建期间不支持写入
    indexRebuildRes, err := coll.RebuildIndex(ctx)
    if err != nil {
        return err
    }
    log.Printf("%+v", indexRebuildRes)

    log.Println("----------------------- TruncateCollection -----------------------")
    // truncate_collection
    // 清空 Collection
    time.Sleep(time.Second * 5)
    truncateRes, err := db.TruncateCollection(ctx, collection)
    if err != nil {
        return err
    }
    log.Printf("TruncateResult: %+v", truncateRes)
    return nil
}

func printErr(err error) {
    if err != nil {
        log.Fatal(err)
    }
}

func main() {
    database := "go-sdk-demo-db"
    collectionName := "go-sdk-demo-col"
    collectionAlias := "go-sdk-demo-alias"

    ctx := context.Background()
    testVdb, err := NewDemo("vdb http url or ip and post", "vdb username", "key get
from web console")
    // testVdb := NewDemo("http://127.0.0.1:80", "root","vdb-key")
```

```
    printErr(err)
    err = testVdb.Clear(ctx, database)
    printErr(err)
    err = testVdb.CreateDBAndCollection(ctx, database, collectionName,
collectionAlias)
    printErr(err)
    err = testVdb.UpsertData(ctx, database, collectionName)
    printErr(err)
    err = testVdb.QueryData(ctx, database, collectionName)
    printErr(err)
    err = testVdb.UpdateAndDelete(ctx, database, collectionName)
    printErr(err)
    err = testVdb.DeleteAndDrop(ctx, database, collectionName)
    printErr(err)
}
```

使用 AI 套件上传文件并检索

最近更新时间：2023-12-26 15:12:02

AI 套件是腾讯云向量数据库（Tencent Cloud VectorDB）提供的一站式文档检索解决方案，包含自动化文档解析、信息补充、向量化、内容检索等能力。用户仅需上传原始文档，数分钟内即可快速构建专属知识库，大幅提高知识接入效率。本文介绍通过 GO SDK 操作 AI 类数据库上传文件并进行内容相似性检索的完整示例。

```
package main

import (
    "context"
    "log"
    "time"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/ai_document_set"
    collection_view "github.com/tencent/vectordatabase-sdk-
go/tcvectordb/api/collection_view"
)

type AIDemo struct {
    client *tcvectordb.Client
}

func NewAIDemo(url, username, key string) (*AIDemo, error) {
    cli, err := tcvectordb.NewClient(url, username, key, &tcvectordb.ClientOption{
        Timeout:         10 * time.Second,
        ReadConsistency: tcvectordb.EventualConsistency})
    if err != nil {
        return nil, err
    }
    // disable/enable http request log print
    // cli.Debug(false)
    return &AIDemo{client: cli}, nil
}

func (d *AIDemo) Clear(ctx context.Context, database string) error {
    log.Println("------------------------- DropDatabase -------------------------")
    result, err := d.client.DropAIDatabase(ctx, database)
    if err != nil {
        return err
    }
    log.Printf("drop database result: %+v", result)
```

```
    return nil
}

func (d *AIDemo) DeleteAndDrop(ctx context.Context, database, collectionView,
documentSetName string) error {
    // 删除Document
    log.Println("------------------------ Delete Document ------------------------")
    cdocDelResult, err :=
d.client.AIDatabase(database).CollectionView(collectionView).Delete(ctx,
tcvectordb.DeleteAIDocumentSetParams{
        DocumentSetNames: []string{documentSetName},
    })
    if err != nil {
        return err
    }
    log.Printf("delete document result: %+v", cdocDelResult)


    // 删除collectionView，删除collectionView的同时，其中的数据也将被全部删除
    log.Println("------------------------ DropCollectionView ------------------------")
    colDropResult, err := d.client.AIDatabase(database).DropCollectionView(ctx,
collectionView)
    if err != nil {
        return err
    }
    log.Printf("drop collection result: %+v", colDropResult)

    log.Println("------------------------ DropDatabase ------------------------")
    // 删除db，db下的所有collectionView都将被删除
    dbDropResult, err := d.client.DropAIDatabase(ctx, database)
    if err != nil {
        return err
    }
    log.Printf("drop database result: %+v", dbDropResult)
    return nil
}

func (d *AIDemo) CreateAIDatabase(ctx context.Context, database string) error {
    log.Println("------------------------ CreateDatabase ------------------------")
    _, err := d.client.CreateAIDatabase(ctx, database)
    if err != nil {
        return err
    }
    log.Println("------------------------ ListDatabase ------------------------")
    dbList, err := d.client.ListDatabase(ctx)
```

```
    if err != nil {
        return err
    }

    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)
    }
    return nil
}

func (d *AIDemo) CreateCollectionView(ctx context.Context, database, collectionView
string) error {
    db := d.client.AIDatabase(database)

    log.Println("------------------------ CreateCollectionView ------------------------")
    index := tcvectordb.Indexes{}
    index.FilterIndex = append(index.FilterIndex, tcvectordb.FilterIndex{FieldName:
"test_str", FieldType: tcvectordb.String, IndexType: tcvectordb.FILTER})

    db.WithTimeout(time.Second * 30)

    enableWordsEmbedding := true
    appendTitleToChunk := true
    appendKeywordsToChunk := false

    _, err := db.CreateCollectionView(ctx, collectionView,
tcvectordb.CreateCollectionViewParams{
        Description: "desc",
        Indexes:     index,
        Embedding: &collection_view.DocumentEmbedding{
            Language:            string(tcvectordb.LanguageChinese),
            EnableWordsEmbedding: &enableWordsEmbedding,
        },
        SplitterPreprocess: &collection_view.SplitterPreprocess{
            AppendTitleToChunk:    &appendTitleToChunk,
            AppendKeywordsToChunk: &appendKeywordsToChunk,
        },
```

```
    })

    if err != nil {
        return err
    }

    log.Println("------------------------- ListCollectionViews -------------------------")
    // 列出所有 CollectionView
    collListRes, err := db.ListCollectionViews(ctx)
    if err != nil {
        return err
    }
    for _, col := range collListRes.CollectionViews {
        log.Printf("ListCollectionViews: %+v", col)
    }
    return nil
}

func (d *AIDemo) LoadAndSplitText(ctx context.Context, database, collection, filePath
string) (*tcvectordb.LoadAndSplitTextResult, error) {
    log.Println("------------------------- UploadFile -------------------------")
    coll := d.client.AIDatabase(database).CollectionView(collection)
    res, err := coll.LoadAndSplitText(ctx, tcvectordb.LoadAndSplitTextParams{
        LocalFilePath: filePath,
        MetaData: map[string]interface{}{
            "test_str": "v1",
            "fileKey":  1024,
            "author":   "sam",
        },
    })
    if err != nil {
        return nil, err
    }
    log.Printf("UploadFileResult: %+v", res)
    return res, nil
}

func (d *AIDemo) GetFile(ctx context.Context, database, collection, fileName string)
error {
    coll := d.client.AIDatabase(database).CollectionView(collection)
    log.Println("------------------------- GetFile by Name -------------------------")
    result, err := coll.Query(ctx, tcvectordb.QueryAIDocumentSetParams{
        DocumentSetName: []string{fileName},
    })
    if err != nil {
```

```
        return err
    }
    log.Printf("QueryResult: count: %v", result.Count)
    for _, doc := range result.Documents {
        log.Printf("QueryDocument: %+v", doc)
    }
    return nil
}

func (d *AIDemo) QueryAndSearch(ctx context.Context, database, collectionView
string) error {
    db := d.client.AIDatabase(database)
    coll := db.CollectionView(collectionView)

    log.Println("--------------------------- Search ---------------------------")
    // 查找与给定查询向量相似的向量。支持输入文本信息检索与输入文本相似的内容，同时，支持
搭配标量字段的 Filter 表达式一并检索。
    enableRerank := true
    res, err := coll.Search(ctx, tcvectordb.SearchAIDocumentSetsParams{
        Content:      "什么是向量数据库",
        ExpandChunk: []int{1, 0},
        Filter:       tcvectordb.NewFilter(`test_str="v1"`),
        Limit:        2,
        RerankOption: &ai_document_set.RerankOption{
            Enable:               &enableRerank,
            ExpectRecallMultiples: 2.5,
        },
    })
    if err != nil {
        return err
    }
    for _, doc := range res.Documents {
        log.Printf("SearchDocument: %+v", doc)
    }

    log.Println("--------------------------- Update ---------------------------")
    updateRes, err := coll.Update(ctx, map[string]interface{}{
        "test_str": "v2",
    }, tcvectordb.UpdateAIDocumentSetParams{
        Filter: tcvectordb.NewFilter(`test_str="v1"`),
    })
    if err != nil {
        return err
    }
    log.Printf("updateResult: %+v", updateRes)
```

```
    log.Println("------------------------- Query -------------------------")
    queryRes, err := coll.Query(ctx, tcvectordb.QueryAIDocumentSetParams{
        Filter: tcvectordb.NewFilter(`test_str="v2"`),
        Limit:  1,
    })
    if err != nil {
        return err
    }
    for _, doc := range queryRes.Documents {
        log.Printf("QueryDocument: %+v", doc)
    }
    return nil
}

func (d *AIDemo) Alias(ctx context.Context, database, collectionView, alias string)
error {
    db := d.client.AIDatabase(database)
    log.Println("------------------------- SetAlias -------------------------")
    setRes, err := db.SetAlias(ctx, collectionView, alias)
    if err != nil {
        return err
    }
    log.Printf("SetAlias result: %+v", setRes)

    log.Println("---------------------- DescribeCollectionView ----------------------")
    collRes, err := db.DescribeCollectionView(ctx, collectionView)
    if err != nil {
        return err
    }
    log.Printf("Collection: %+v", collRes)

    log.Println("------------------------- DeleteAlias -------------------------")
    delRes, err := db.DeleteAlias(ctx, alias)
    if err != nil {
        return err
    }
    log.Printf("SetAlias result: %+v", delRes)
    return nil
}

func printErr(err error) {
    if err != nil {
        log.Fatal(err)
    }
```

```
}

func main() {
    database := "go-sdk-demo-ai-db"
    collectionView := "go-sdk-demo-ai-col"
    collectionViewAlias := "go-sdk-demo-ai-alias"

    ctx := context.Background()
    testVdb, err := NewAIDemo("vdb http url or ip and post", "vdb username", "key get
from web console")
    printErr(err)
    err = testVdb.Clear(ctx, database)
    printErr(err)
    err = testVdb.CreateAIDatabase(ctx, database)
    printErr(err)
    err = testVdb.CreateCollectionView(ctx, database, collectionView)
    printErr(err)
    loadFileRes, err := testVdb.LoadAndSplitText(ctx, database, collectionView,
"../tcvdb.md")
    printErr(err)
    time.Sleep(time.Second * 30) // 等待后台解析文件完成
    err = testVdb.GetFile(ctx, database, collectionView,
loadFileRes.DocumentSetName)
    printErr(err)
    err = testVdb.QueryAndSearch(ctx, database, collectionView)
    printErr(err)
    err = testVdb.Alias(ctx, database, collectionView, collectionViewAlias)
    printErr(err)
    err = testVdb.DeleteAndDrop(ctx, database, collectionView,
loadFileRes.DocumentSetName)
    printErr(err)
}
```

连接数据库实例

最近更新时间：2023-12-26 16:00:02

功能介绍

操作数据库之前，您需要通过 `NewClient()` 创建一个向量数据库的客户端对象，与向量数据库服务器连接才能进行交互。

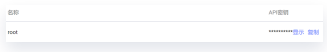
请求示例

```
package main
import (
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
)

func main() {
    // 初始化客户端
    var defaultOption = &tcvectordb.ClientOption{
        Timeout:          time.Second * 5,
        MaxIdldConnPerHost: 2,
        IdleConnTimeout:    time.Minute,
        ReadConsistency:    tcvectordb.EventualConsistency,
    }
    client, err := tcvectordb.NewClient("http://10.0.X.X:80", "root",
"eC4bLRy2va*****************************", defaultOption)
    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	最大的空闲连接数。	是	• 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。		• 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。

管理 Database
管理 Base 类 Database

最近更新时间：2023-12-26 15:12:02

连接向量数据库实例之后，便可以在集群中创建向量数据库。本文介绍创建、删除 Base 类 Database 的 SDK 接口。

> ⚠ **注意：**
> 操作 Database 之前，您需要先 连接数据库 。

创建 Base 类 Database

```
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
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]。

删除 Base 类 Database

```
result, _ := client.DropDatabase(context.Background(), database)
log.Printf("DropDatabase result: %+v", result)
```

输出信息，如下所示。

```
DropDatabase result: &{AffectedCount:1}
```

参数名	参数含义
AffectedCount	影响行数，即为删除的数据库数量。

管理 AI 类 Database

最近更新时间：2023-12-26 15:12:02

本文介绍创建、删除 AI 类 Database 的 SDK 接口。

> **注意：**
> 操作 Database 之前，您需要先 连接数据库 。

创建 AI 类 Database

```
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

```
result, _ := client.DropAIDatabase(context.Background(), aiDatabase)
log.Printf("DropAIDatabase result: %+v", result)
```

输出信息，如下所示。

```
DropAIDatabase result: &{AffectedCount:1}
```

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

查询集群数据库

最近更新时间：2023-12-26 15:12:02

功能介绍

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。

请求示例

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)
}
```

返回信息

```
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：2024-04-19 20:01:51

`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。

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。

```
index := tcvectordb.Indexes{
    // 指定embedding时，vector的维度可以不传，系统会使用embedding model的维度
    VectorIndex: []tcvectordb.VectorIndex{
        {
            FilterIndex: tcvectordb.FilterIndex{
                FieldName: "vector",
                FieldType: tcvectordb.Vector,
                IndexType: tcvectordb.HNSW,
            },
            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: "page",
            FieldType: tcvectordb.Uint64,
            IndexType: tcvectordb.FILTER,
        },
    },
}
```

参数	子参数	是否必选	参数含义
FilterIndex	FieldName	是	指定 Filter 索引标量字段名。 **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。 • 配置 **name** 为 **id**，构建唯一的 Document id 为主键的 Filter 索引。 • 配置其他自定义扩展的可作为 Filter 索引标量字段，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。
	Index Type	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 。
Vector Index	Field Name	是	指定向量数据的索引字段为 vector。
	Filed Type	是	指定向量索引的数据类型，固定为 tcvectordb.Vector。
	Index Type	是	指定向量索引字段的索引类型。当前所支持的索引类型，及具体索引方式，请参见设计模型中的 Index 。 • FLAT：暴力检索，召回率100%，但检索效率低。 • HNSW：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。具体测试数据，请参见性能白皮书的测试结果。 • 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	指定向量数据维度。 • 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。
	MetricType	是	指定目标向量与查询向量之间距离度量的相似性算法。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	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。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]。

（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 segment，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:       "segment",
        VectorField: "vector",
        Model:       tcvectordb.BGE_BASE_ZH,
    },
}
```

参数名	是否必选	参数含义
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，请参见 Embedding 介绍 。取值如下所示： • BGE_BASE_ZH：适用中文，768维，推荐使用。 • 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。
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。
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 。

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。

创建 Collection 配置 Embedding 参数

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, 0, "desription doc",
index, param)
log.Printf("CreateCollection success: %v: %v", coll.DatabaseName,
coll.CollectionName)
```

创建 Collection 不配置 Embedding 参数

```
var (
    ctx             = context.Background()
    database        = "go-sdk-test-db"
    collectionName     = "go-sdk-test-coll"
)

db := client.Database(database)
coll, _ := db.CreateCollection(ctx, collectionName, 1, 0, "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。

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。

查询 Collection 配置

最近更新时间：2023-12-26 15:12:03

功能介绍

`ListCollection()` 接口可查询指定 Base 类 Database 中所有的 Collection。

请求示例

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"
    collectionName      = "go-sdk-test-coll"
)
db := client.Database(database)
result, _ := db.ListCollection(ctx)
for _, col := range result.Collections {
    data, _ := json.Marshal(col)
    log.Printf("%+v", string(data))
}
```

返回信息

```
{
  "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
      }
     }
    ],
    "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。
		filedTyp	Filter 字段的数据类型。

		e	
		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**。
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。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training data**：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 index**：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：2023-12-26 15:12:03

功能介绍

`DescribeCollection()` 用于查找指定 Collection 的配置信息。

请求示例

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。

返回信息

```
{
    "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
            }
        }],
        "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。

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。
		indexTy	• fieldName 为 id，应 indexType 为 primaryKey。

		pe	• 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**。
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。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training data**：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 index**：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：2023-12-26 16:00:01

功能介绍

`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。

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。

请求示例

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)
```

请求参数

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。

返回信息

```
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。

管理 Collection 别名

最近更新时间：2023-12-26 16:00:01

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

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

如下示例，删除别名 go-sdk-test-alias。

```
colRes, _ := db.DeleteAlias(ctx, aliasName)
log.Printf("DeleteAlias: %v", colRes)
```

返回参数

```
&{AffectedCount: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。

删除指定 Collection

最近更新时间：2024-01-17 16:14:01

`DropCollection()` 用于删除已创建的 Base 类 Collection。

请求示例

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)
```

请求参数

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。

返回信息

```
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：2024-01-05 16:52:21

`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。

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。

```
index := tcvectordb.Indexes{
    FilterIndex: []tcvectordb.FilterIndex{
        {
            FieldName: "author_name",
            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	是	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。
		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。

SplitterPreprocess	appendTitleToChunk		• 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。

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"
)

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,
    },
})
log.Printf("CreateCollectionView success: %v: %v", coll.DatabaseName,
coll.CollectionViewName)
```

请求参数

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]。
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。

返回消息

输出信息，如下所示。

```
{
 "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": "tag",
   "fieldType": "array",
   "indexType": "filter"
  },
  {
   "fieldName": "author",
   "fieldType": "string",
   "indexType": "filter"
  }
 ]
}
```

查询 CollectionView 配置

最近更新时间：2024-01-17 17:51:41

功能介绍

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

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。

```
var (
    ctx              = context.Background()
    aiDatabase       = "go-sdk-test-ai-db"
    collectionViewName = "go-sdk-test-ai-coll"
)
db := client.AIDatabase(aiDatabase)
coll, _ := db.ListCollectionViews(ctx)
for _, col := range coll.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
    },
    "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。
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 document	fieldName	标识索引对象为文件名，固定为 `documentSetName` 。

	SetName 文件名创建 Filter 索引	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 CollectionView

最近更新时间：2024-03-18 10:48:21

功能介绍

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

请求示例

如下示例，查询集合视图 **go-sdk-test-ai-coll** 的配置信息。

```
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))
```

请求参数

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。

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
    },
    "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。
		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。
Indexes	默认以 documen	fieldName	标识索引对象为 `documentSetId` 。

	tSetId 文件ID 创建主键索引	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 。

清空 CollectionView 数据

最近更新时间：2023-12-26 15:12:03

功能介绍

`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。

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。

请求示例

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)
```

请求参数

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。

返回信息

```
TruncateCollectionView success result: &{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。

管理 CollectionView 别名

最近更新时间：2023-12-26 15:12:03

功能介绍

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。

为 CollectionView 创建别名

```
var (
    ctx            = context.Background()
    aiDatabase     = "go-sdk-test-ai-db"
    collectionViewName  = "go-sdk-test-ai-coll"
    aliasName      = "go-sdk-test-alias"
)

db := cli.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

```
db := cli.AIDatabase(aiDatabase)
colRes, _:= db.DeleteAlias(ctx, aliasName)
log.Printf("Delete CollectionView Alias : %v", colRes)
```

返回信息

```
Set a CollectionView alias: &{AffectedCount:1}
Delete CollectionView Alias : &{AffectedCount:1}
```

参数名	参数含义
AffectedCount	影响的集合视图数量。

接口约束

- DB 和 CollectionView 级别的 drop 操作会同时删除库表下的所有别名。
- 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。

删除 CollectionView

最近更新时间：2023-12-26 15:12:04

功能介绍

`DropCollectionView()` 用于删除已创建的 CollectionView。

接口约束

> **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。

请求示例

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。

返回信息

```
DropCollectionView success result: &{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。

管理 Document 数据

写入数据

最近更新时间：2023-12-26 15:12:04

功能介绍

`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。

请求示例

基于 Embedding 写入原始文本

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"
)
col := client.Database(database).Collection(embeddingCollection)

result, err := col.Upsert(ctx, []tcvectordb.Document{
    {
        Id: "0001",
        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",
        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",
        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",
        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",
        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。"},
        },
    },
})

log.Printf("upsert result: %+v", result)
```

写入向量数据

如下示例，给集合 **go-sdk-test-coll** 直接写入向量数据，对应字段为 Vector。

```
var (
    cli              *tcvectordb.Client
    ctx               = context.Background()
```

```
    database          = "go-sdk-test-db"
    collectionName    = "go-sdk-test-coll"
)
col := cli.Database(database).Collection(collectionName)

buildIndex := true
result, err := col.Upsert(ctx, []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"}},
        },
    },
    {
        Id:     "0004",
        Vector: []float32{0.2123, 0.24, 0.213},
        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。"},
            "tag":      {Val: []string{"曹操", "诸葛亮", "刘备"}},
        },
    },
    {
        Id:     "0005",
        Vector: []float32{0.2123, 0.25, 0.213},
        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。"},
            "tag":      {Val: []string{"曹操", "诸葛亮", "刘备"}},
        },
    },
}, &tcvectordb.UpsertDocumentParams{BuildIndex: &buildIndex})

log.Printf("upsert result: %+v", result)
```

请求参数

参数名称	参数含义	子参数	是否必选	配置方法
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	表示文档的向量值，请务必使用32位浮点数存储向量数据。 **说明：** 如果业务无需使用腾讯云向量数据库（Tencent Cloud Vector DB）的 Embedding 功能做向量化，则配置该参数，写入向量数据，而无需配置 Embedding 参数 text（创建 Collection 时，

				Embedding 参数 **field** 对应指定的文本字段名，示例中为 text）。
		Fields	否	**segment** 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。

精确查询

最近更新时间：2023-12-26 15:12:04

功能介绍

基于精确匹配的查询方式， `query()` 用于精确查找与查询条件完全匹配的数据，具体支持如下功能。

- 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。

接口约束

> **说明：**
> 该接口目前仅支持主键 id（Document ID）结合 Filter 表达式查询，即只能根据文档的唯一标识符来进行查询。

请求示例

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"
)
col := client.Database(database).Collection(collectionName)
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 := col.Query(ctx, 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
documentIds	-	否	表示要查询的	每个 ID 长度限制为[1,128]。支持批量查询，数组元素范围[1,20]。

			文档的所有 ID。	
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 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	否	每页返回的 Document 数量。	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。
	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。

返回信息

```
2023/12/20 19:10:00 document: {Id:0003 Fields:map[bookName:三国演义]}
2023/12/20 19:10:00 document: {Id:0004 Fields:map[bookName:三国演义]}
```

参数名	参数含义
id	Document 的 ID 信息。

Vector	Document 的向量值。该示例未打印此数据。
Fields	Document 自定义扩展的标量字段。例如：bookName、author 等。

基于 ID 相似性检索

最近更新时间：2023-12-26 15:12:04

功能介绍

基于相似度匹配的查询方式， **searchById()** 根据指定的 Document id 进行相似度查询，并返回指定的 Top K 个最相似的 Document。

请求示例

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"
)
col := client.Database(database).Collection(collectionName)
filter := tcvectordb.NewFilter(`bookName="三国演义"`)
documentId := []string{"0003"}
searchRes, err := col.SearchById(ctx, 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)
    }
}
```

请求参数

参数	子参数	是否必选	参数含义	配置方法
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 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')`
	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 list_collections() 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	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。

			K 的 K 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。

返回信息

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。

基于向量数据相似性检索

最近更新时间：2023-12-26 15:12:04

功能介绍

基于相似度匹配的查询方式， **search()** 接口用于查找与给定查询向量相似的文档，返回指定的 Top K 个最相似的文档，并支持搭配自定义的标量字段的 Filter 表达式一并进行相似度检索。

请求示例

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。

```
var (
    ctx            = context.Background()
    database       = "go-sdk-test-db"
    collectionName = "go-sdk-test-coll"
)
col := client.Database(database).Collection(collectionName)
searchRes, _ := col.Search(ctx, [][]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)
    }
}
```

参数	子参数	是否必选	参数含义	配置方法
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 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')。
	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 list_collections() 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	如果插入的数据不足 K 条，则返回实际插入的 Document 数量。

			K 的 K 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。

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 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。

```
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。
Fields	Document 其他自定义的标量字段。例如：author、bookName。

基于原始文本相似性检索

最近更新时间：2023-12-26 15:12:04

功能介绍

若在创建 Collection 时，已配置 Embedding 模型，则可使用 **SearchByText()** 接口输入原始文本进行相似度查询，并支持搭配自定义的标量字段的 Filter 表达式一并进行相似度检索，返回指定的 Top K 个最相似的 Document。

请求示例

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"
)
col := client.Database(database).Collection(embeddingCollection)
searchRes, err := col.SearchByText(ctx, 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)
    }
}
```

请求参数

参数	子参数	是否必选	参数含义	配置方法
	–	是	待检索的文	输入文本信息，用于检索与该文本信息相似的数

text			本信息。	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 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')`
	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 list_collections() 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。
	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。

返回信息

```
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。

更新 Document 数据

最近更新时间：2023-12-26 16:00:02

功能介绍

`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。

请求示例

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"
)
col := client.Database(database).Collection(collectionName)
result, _ := col.Update(ctx, 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)
```

请求参数

参数	参数	是否必选	参数含义
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 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	**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]。

返回信息

```
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：2023-12-26 16:00:02

功能介绍

`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。

接口约束

索引类型为 FLAT，不支持删除。

请求示例

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"
)
col := client.Database(database).Collection(collectionName)
result, err := col.Delete(ctx, tcvectordb.DeleteDocumentParams{
    DocumentIds: []string{"0001", "0003"},
    Filter:      tcvectordb.NewFilter(`bookName="西游记"`),
})
log.Printf("delete affect count: %d", result.AffectedCount)
```

请求参数

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
`DocumentIds`	是	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 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')`。

返回信息

```
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。

管理 DocumentSet 数据
基于文件写入数据

最近更新时间：2024-01-17 17:19:01

功能介绍

`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。

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。

请求示例

```
import (
    "context"
    "log"
    "time"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/ai_document_set"
)

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
        "author_name": "sam",
        "fileKey":     1024,
    }

appendTitleToChunk := false
appendKeywordsToChunk := true

result, _ := col.LoadAndSplitText(ctx, tcvectordb.LoadAndSplitTextParams{
    DocumentSetName: "tcvdb.md",
    // Reader:         fd,
```

```
    LocalFilePath: "../example/tcvdb.md",
    MetaData:      metaData,
    SplitterPreprocess: ai_document_set.DocumentSplitterPreprocess{
        AppendTitleToChunk:    &appendTitleToChunk,
        AppendKeywordsToChunk: &appendKeywordsToChunk,
    },
})
log.Printf("LoadAndSplitText success: %+v", result)
```

请求参数

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。
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。
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。

返回信息

```
{
  DocumentSetId: 1187010173194******
  DocumentSetName: tcvdb.md
  CosEndpoint: https: //vectordb-pre-gz-131891****.cos.ap-guangzhou.myqcloud.com
  CosRegion: ap-guangzhou
  CosBucket: vectordb-pre-gz-131891****
  UploadPath: embedding_file/vdb-nfcrhc2s/go-sdk-test-ai-db/go-sdk-test-ai-coll-
1703065687448/tcvdb.md
  TmpSecretID:
AKIDIHuF4d7Ms1bYt4r9qnywSr4Vd40kwrCNpAxUuClgHRbSzRLO**********
  TmpSecretKey: AKT2/ZG1VtkWttjDCq7/NpJ/xmCvT7akuY0bwZUrvVE=
  SessionToken: *********
  MaxSupportContentLength: 1048576
}
```

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：2024-04-09 16:16:11

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。

- **GetDocumentSetByName()**：根据文件名查询文件内容。
- **GetDocumentSetById()**：根据文件 ID 查询文件内容。

请求示例

使用文件名获取文件内容

如下示例，获取文件名为 tcvdb.md 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, "tcvdb.md")
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, res.DocumentSetId)
log.Printf("GetDocumentSetById success: %+v", res)
```

请求参数

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	-

返回信息

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.md 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} 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。
SplitterPrepr	appendTitleT	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

ocess	oChunk	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。

获取 Chunks

最近更新时间：2024-04-09 16:16:11

`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。

- 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。

请求示例

```
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: "tcvdb.md",
    })
log.Printf("GetChunks, count: %v", result.Count)
for _, chunk := range result.Chunks {
        log.Printf("chunk: %+v", chunk)
    }
```

请求参数

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：[1,16384]。
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。

返回信息

```
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。

查询文件信息

最近更新时间：2024-01-17 16:14:01

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。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。

请求示例

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{"tcvdb.md"},
  Filter:          tcvectordb.NewFilter(`author_name="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)))
}
```

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)
```

请求参数

子参数	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]。
	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]。

DocumentSetId		

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。

返回信息

res {"databaseName":"db-test-ai","collectionViewName":"coll-ai-files","documentSetId":"1193839887725101056","documentSetName":"腾讯云向量数据库.md","text":"","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":"2024-01-08 16:53:29","lastUpdateTime":"2024-01-08 16:53:30","keywords":"向量 数据库 数据 腾讯 检索 索引 支持 结构化 进行 相似"},"ScalarFields":{"author":{"val":"Tencent"}},"splitterPreprocess":{"appendTitleToChunk":false,"appendKeywordsToChunk":true}}

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。
	indexedProgr	文件被预处理、Embedding 向量化的进度。

	ess	
	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。

相似性检索文件内容

最近更新时间：2024-01-08 12:27:51

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。

请求示例

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.md` 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: "什么是向量数据库",
    DocumentSetName: []string{"腾讯云向量数据库.md"},
    Filter:          tcvectordb.NewFilter(`author_name="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_name="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	否	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[1,20]。
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	指定返回最相似的 Top K 的 K 的值。

返回信息

{DatabaseName:go-sdk-test-ai-db CollectionViewName:go-sdk-test-ai-coll DocumentSetId:1186989568189****** DocumentSetName:tcvdb.md Score:0.8791403770446777 SearchData:{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:865 EndPos:1061 Pre:[## 腾讯云向量数据库
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。

] Next:[### 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。

]} ScalarFields:map[author_name:sam fileKey:1024]}
/Users/liuqingsong/go/src/vectordb/vectordatabase-sdk-go/test/aidb_test.go:213:
document: {DatabaseName:go-sdk-test-ai-db CollectionViewName:go-sdk-test-ai-coll DocumentSetId:1186989568189****** DocumentSetName:tcvdb.md Score:0.8646935820579529 SearchData:{Text:#### 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。

StartPos:1439 EndPos:1532 Pre:[#### 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。

] Next:[### 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。 具体信息，请参见 产品优势 。

]} ScalarFields:map[author_name:sam fileKey:1024]}
/Users/liuqingsong/go/src/vectordb/vectordatabase-sdk-go/test/aidb_test.go:213:
document: {DatabaseName:go-sdk-test-ai-db CollectionViewName:go-sdk-test-ai-coll
DocumentSetId:1186989568189****** DocumentSetName:tcvdb.md
Score:0.8587197065353394 SearchData:{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。 具体信息，请参见 产品优势 。

StartPos:1532 EndPos:1644 Pre:[#### 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。

] Next:[### 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，并返回与用户可能感兴趣的物品作为推荐结果。
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。

]} ScalarFields:map[author_name:sam fileKey:1024]}

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。
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：2023-12-26 15:12:05

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。

- 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。

> **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。

请求示例

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。

```
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_name": "jack",
}, tcvectordb.UpdateAIDocumentSetParams{
    DocumentSetName: []string{"tcvdb.md"},
})
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，并满足 Filter 表达式的文件的标量字段 **author** 为 tencent。

```
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_name": "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** 的 Filter 表达式的文件的字段 author 为 tencent，并新增字段 tag。

```
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_name": "jack",
}, tcvectordb.UpdateAIDocumentSetParams{
    Filter:       tcvectordb.NewFilter(`author_name="sam"`),
})
log.Printf("update affect count: %d", result.AffectedCount)
```

请求参数

参数	子参数	参数含义	是否必选	配置方法及要求
UpdateADocumentSetParams	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的并集。
	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并集。

	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：2023-12-26 15:12:05

功能介绍

`delete()` 接口用于删除存储于 CollectionView 文件。

- 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。

请求示例

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{
    DocumentSetName: []string{"tcvdb.md"},

})
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{
    DocumentSetId: []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_name 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_name="sam"`),
})
log.Printf("delete affect count: %d", result.AffectedCount)
```

请求参数

参数名称	参数含义	是否必选	配置方法及要求
DocumentSetName	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的并集。
	指定需	否	支持批量删除，数据元素最大值为20。

DocumentSetId	删除的文件ID。		**说明：** 同时配置 DocumentSetId 与 filter 参数，删除数据将会取二者的并集。

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：2023-12-26 15:12:05

功能介绍

`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。

接口约束

> **注意：**
> - 索引重建过程中 Colletion 禁止写入、读取。
> - 重建索引需要新的内存来构建索引。

请求示例

```
var (
    ctx             = context.Background()
    database        = "go-sdk-test-db"
    embeddingCollection = "go-sdk-test-emcoll"
)
col := client.Database(database).Collection(embeddingCollection)
col.RebuildIndex(ctx, &tcvectordb.RebuildIndexParams{
    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
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。 **说明：**	取值如下所示： • **true**：重建之前，先删除旧索引在重建索引。

		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。	**说明：** 内存资源不足时，可先删除旧索引，在新索引还没有创建完成之前，该表无法正常读写。 • **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。 **说明：** 内存资源足够的情况下，可不删除旧索引。在新索引还没有创建完成之前，该表可读数据，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**：CPU 核数为 1，即仅使用 CPU 1 核进行模型训练，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。

返回参数

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。

- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training data**：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 index**：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。

HTTP API
使用前阅读

最近更新时间：2024-04-15 10:50:11

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 新建数据库实例 ，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	PO	http://{实例内网 IP 地址}:{实例网络端口}/ai/collectionView/drop

			ST	
	/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
	/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	PO	http://{实例内网 IP 地址}:{实例网络口}/ai/documentSet/update

			ST	
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
	/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：2024-05-17 15:54:51

功能介绍

- **/database/create** 接口用于创建 Base 类向量数据库。
- **/ai/database/create** 接口用于创建 AI 类向量数据库。

请求示例

> **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。

Base 类创建 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"
}'
```

AI 类创建 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

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。/database/create 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	<pre>{
		  "code": 0,
		  "msg": "operation success",
		  "affectedCount": 1
		}</pre>

400	创建失败	<pre>{
		  "code": 1,
		  "msg": "database dbl already exist",
		  "affectedCount": 0
		}</pre>

返回参数

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：2023-12-08 16:15:48

功能介绍

- **/database/drop** 接口用于删除已创建的 Base 类 Database，并删除 Database 中所有 Collection 以及 Document。
- **/ai/database/drop** 接口用于删除已创建的 AI 类 Database，并删除 Database 中所有 CollectionView 以及 DocumentSet。

接口约束

> **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。

请求示例

> **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。

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

状态码	含义	响应消息
200	删除数据库执行成功	`{ "code": 0, "msg": "operation success", "affectedCount": 1 }`

返回参数

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：2023-12-08 16:15:48

功能介绍

/database/list 接口用于查询集群中存在的所有 Base 类或 AI 类的数据库。

请求示例

> **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 GET \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY**********' \
  http://10.0.X.X:80/database/list
```

响应消息

状态码	含义	响应消息
200	执行成功	返回消息的参数 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"
    }
  }
}
```


返回参数

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：2024-05-17 15:54:51

功能介绍

/collection/create 接口用于在已创建的 Base 类 Database 中创建 Collection。

> **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。

请求示例

> **注意：**
> 如下示例可直接复制，在运行之前，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。

创建 Collection 配置 Embedding 参数

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": 2,
    "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": "bookName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "author",
        "fieldType": "string",
        "indexType": "filter"
      }
    ]
}'
```

创建 Collection 不配置 Embedding 参数

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": 2,
    "shardNum": 1,
    "description": "this is the collection description",
    "indexes": [
        {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
        },
        {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "dimension": 3,
            "metricType": "COSINE",
            "params": {
                "M": 16,
                "efConstruction": 200
            }
        },
        {
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        },
        {
            "fieldName": "author",
            "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	-	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。

	在的 Database 名称。			
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-base-zh**：适用中文，768维，推荐使用。 • **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。
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 类型指定索引字	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 。即该参数固定配置为 `id` 。
		fieldType	是	指定索引对象的数据类型。该参数固定为 `string` 。
		indexType	是	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。
	向量索引 **说明**：每一个 Colle	**fieldName**	是	指定索引对象为 vector。该参数固定配置为 `vector` 。
		fieldType	是	指定索引对象为 vector 的数据类型。该参数固定为 `vector` 。

段。	ction 必须指定主键索引和向量索引。具体信息，请参 Index。	indexType	是	指定索引类型。当前所支持的索引类型，及具体索引方式，请参见设计模型中的 Index。取值如下所示。 • FLAT：暴力检索，召回率100%，但检索效率低。 • HNSW：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。具体测试数据，请参见性能白皮书的测试结果。 • 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]。 • 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：** 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。
		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	**fieldN**	是	配置可作为 Filter 索引的自定义扩展的标量字段

		ame		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，数组元素为 string。
		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。

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。/collection/create 接口返回的状态码以及相关信息，如下表所示。

状态码	含义	响应消息
200	集合创建成功	

		{
		"code": 0,
		"msg": "operation success",
		"affectedCount": 1
		}

400	集合创建失败	{
		"code": 1,
		"msg": "Collection already exist: angular_32",
		"affectedCount": 0
		}

返回参数

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：2023-12-08 16:15:48

功能介绍

/collection/drop 接口用于删除 Base 类数据库中已存在的 Collection，包含 Collection 中的所有 Document。

接口约束

> **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。

请求示例

> **注意：**
> 如下示例可直接复制，在运行之前，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/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。
colle	是	所需删除的 Collection 名	使用 /collection/list 获取指定数据库名下的

ction		称。	Collection 列表，复制需删除的集合。

响应消息

状态码	含义	响应消息

200	删除集合执行成功	{
		"code": 0,
		"msg": "operation success",
		"affectedCount": 1
		}

400	删除集合执行失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回参数

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：2023-12-08 16:15:48

功能介绍

/collection/list 接口用于查询指定 Base 类 Database 中所有的 Collection。

接口约束

实例在创建、异常状态下不能执行该操作。

请求示例

> **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/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

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。**/collection/list** 接口返回的状态码以及相关信息，如下所示。

状态码

状态码	含义	响应消息
200	执行成功	返回消息体中，以数组的形式列出了数据库中所有的 Collection 信息。具体参数含义，请参见 返回消息参数 。

400	执行失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回消息

/collection/list 接口执行成功，将返回数据库中所有的 Collection。返回消息体，如下所示。其每一个参数的含义，请参见 返回参数 。


```
{
  "code": 0,
  "msg": "operation success",
  "collections": [
    {
      "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": "bookName",
          "fieldType": "string",
```

```
            "indexType": "filter"
          },
          {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
          },
          {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
          },
          {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "indexedCount": 4,
            "dimension": 768,
            "metricType": "COSINE",
            "params": {
              "M": 16,
              "efConstruction": 200
            }
          }
        ],
        "indexStatus": {
          "status": "ready",
          "startTime": ""
        }
      },
      {
        "database": "db-test",
        "collection": "book-vector",
        "documentCount": 3,
        "replicaNum": 2,
        "shardNum": 1,
        "createTime": "2023-09-14 14:58:04",
        "description": "this is the collection description",
        "indexes": [
          {
            "fieldName": "author",
            "fieldType": "string",
            "indexType": "filter"
          },
          {
```

```
            "fieldName": "bookName",
            "fieldType": "string",
            "indexType": "filter"
          },
          {
            "fieldName": "vector",
            "fieldType": "vector",
            "indexType": "HNSW",
            "indexedCount": 3,
            "dimension": 3,
            "metricType": "COSINE",
            "params": {
              "M": 16,
              "efConstruction": 200
            }
          },
          {
            "fieldName": "id",
            "fieldType": "string",
            "indexType": "primaryKey"
          }
        ],
        "indexStatus": {
          "status": "ready",
          "startTime": ""
        }
      }
  ]
}
```


返回参数

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。创建名别，请参见 /alias/set 。
replicaN	–	–	显示 Collection 的副本数。

um			
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。当前所支持的索引类型及具体索引方式，请参见设计模型中的 Index 。
		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` 。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 。
indexStatus	标识 Collection 是否有重建索引	status	标识当前 Collection 是否在重建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training data**：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 index**：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：2023-12-08 16:15:48

功能介绍

/collection/describe 接口用于查询指定 Base 类数据库的 Collection 的信息。

请求示例

> **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/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

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。/collection/describe 接口返回的状态码以及相关信息，如下所示。

状态码

状态码	含义	响应消息
200	集合查询成功	返回消息体中包含查询 Collection 的信息。具体信息，请参见返回消息 。

400	集合查询失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回消息

/collection/describe 接口执行成功，将返回查询 Collection 的信息。其每一个参数的含义，请参见 返回参数 。


```
{
  "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": "bookName",
            "fieldType": "string",
            "indexType": "filter"
        }
    ],
    "indexStatus": {
        "status": "ready",
        "startTime": ""
    }
  }
}
```


返回参数

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。创建名别，请参见 /alias/set 。
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。当前所支持的索引类型及具体索引方式，请参见设计模型中的 Index 。
		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` 。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 。
indexStatus	标识 Collection 是否有重建索引	status	标识当前 Collection 是否在重建索引。 • **ready**：表示当前 Collection 已准备就绪，可正常使用。 • **training data**：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 index**：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：2023-12-08 16:15:48

功能介绍

/collection/truncate 接口用于清空 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。

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。

请求示例

> **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/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。
colle	是	所需清空数据的	使用/collection/list 获取指定数据库名下的

ction		Collection 名称。	Collection 列表，复制需清空数据的集合。

响应消息

状态码

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。/collection/truncate接口返回的状态码以及相关信息，如下表所示。

状态码	含义	响应消息

200	删除集合执行成功	{
		"code": 0,
		"msg": "operation success",
		"affectedCount": 1
		}

400	删除集合执行失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回参数

参数名	参数含义
affectedCount	清空的集合数量

CollectionView create

最近更新时间：2023-12-22 16:39:21

功能介绍

/ai/collectionView/create 接口用于在已创建的 AI 类 Database 中创建集合视图，用于存储文件数据。

请求示例

> **注意：**
> 如下示例可直接复制，在运行之前，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。

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": true,
        "appendKeywordsToChunk": true
    },
    "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	是	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	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	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，默认值为 **true**。
		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。
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`。

响应消息

状态码	含义	响应消息

200	集合创建成功	<pre>{
		  "code": 0,
		  "msg": "operation success",
		  "affectedCount": 1
		}</pre>

400	集合创建失败	<pre>{
		  "code": 1,
		  "msg": "Collection already exist: angular_32"
		}</pre>

返回参数

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：2023-12-08 16:15:52

功能介绍

/ai/collectionView/drop 接口用于删除 AI 类数据库中存在的 CollectionView，包含 collectionView 中所有的 DocumentSet。

接口约束

> **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。

请求示例

> **注意：**
> 如下示例可直接复制，在运行之前，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/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。
collecti	是	所需删除的	使用 /ai/collectionView/list 获取指定数据库名下

onView		CollectionView 名称。	的 CollectionView 列表，复制需删除的 CollectionView 。

响应消息

状态码	含义	响应消息

200	删除集合执行成功	{
		"code": 0,
		"msg": "operation success",
		"affectedCount": 1
		}

400	删除集合执行失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回参数

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：2024-01-12 17:33:41

功能介绍

/ai/collectionView/list 接口用于查询指定 AI 类 Database 中所有的 CollectionView。

请求示例

> **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/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

状态码

状态码	含义	响应消息

200	执行成功	返回消息体中，以数组的形式列出了数据库中所有的 CollectionView 信息。具体参数含义，请参见 返回消息参数 。

400	执行失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回参数


```
{
 "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
   },
   "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。
descripti	–	–	显示 CollectionView 的描述信息。

on			
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。
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` 。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 。

describe

最近更新时间：2024-01-12 17:33:41

功能介绍

/ai/collectionView/list 接口用于查询指定 AI 类 Database 中所有的 CollectionView。

请求示例

> **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/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

状态码

状态码	含义	响应消息

200	执行成功	返回消息体中，以数组的形式列出了数据库中所有的 CollectionView 信息。具体参数含义，请参见 返回消息参数 。

400	执行失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回参数


```
{
 "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
   },
   "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。
descripti	–	–	显示 CollectionView 的描述信息。

on			
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。
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` 。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 。

truncate

最近更新时间：2023-12-08 16:15:52

功能介绍

/ai/collectionView/truncate 接口用于清空 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。

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。

请求示例

> **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/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

状态码

状态码	含义	响应消息

200	删除集合执行成功	{
		"code": 0,
		"msg": "operation success",
		"affectedCount": 1
		}

400	删除集合执行失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回参数

参数名	参数含义
affectedCount	清空的集合数量

Alias set

最近更新时间：2023-12-08 17:20:03

功能介绍

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。

- **/alias/set** 接口用于为 Base 数据库的 Collection 指定别名。
- **/ai/alias/set** 接口用于为 AI 类数据库的 CollectionView 指定别名。

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。

请求示例

> **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。

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":"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	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

		CollectionView 所对应的 Database 名。	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

状态码

状态码	含义	响应信息

200	创建成功	<pre>{
		"code": 0,
		"msg": "operation success",
		"affectedCount": 1
		}</pre>

400	创建失败	<pre>{
		"code": 1,
		"msg": "operation failed, reason...."
		}</pre>

返回参数

参数名	参数含义

affectedCount	创建别名的集合数量。

delete

最近更新时间：2023-12-08 16:15:49

功能介绍

删除别名，用于清理 Collection 或 CollectionView 中已经过期或废弃的别名。

- **/alias/delete** 接口用于删除数据库指定的集合的别名。
- **/ai/alias/delete** 接口用于删除 AI 类数据库集合视图的别名。

请求示例

> **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。

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":"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

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：2023-12-08 16:15:49

功能介绍

/document/upsert 接口用于给 Collection 中插入向量数据。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。

接口约束

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。

请求示例

> **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。

插入向量数据

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。

```
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
        ],
        "author": "罗贯中",
        "bookName": "三国演义",
        "page": 21
      },
      {
        "id": "0002",
        "vector": [
          0.2123,
          0.22,
          0.213
        ],
        "author": "吴承恩",
        "bookName": "西游记",
        "page": 22
      },
      {
        "id": "0003",
        "vector": [
          0.2123,
          0.21,
          0.213
        ],
        "author": "曹雪芹",
        "bookName": "红楼梦",
        "page": 23
      }
    ]
}'
```

插入文本数据

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。",
          "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
        }
    ]
}'
```

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
datab	指定要插入的	-	是	使用/database/list 获取集群中的数据库列表，

ase	Database 名称。			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，请务必使用32位浮点数存储向量数据。 **说明：** 如果业务无需使用腾讯云向量数据库（Tencent Cloud VectorDB）的 Embedding 功能做向量化，则配置该参

				数，写入向量数据，而无需配置 Embedding 参数 **text**（创建 Collection 时，Embedding 参数 **field** 对应指定的文本字段名，示例中为 text）。
		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。

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。**/document/upsert** 接口返回的状态码以及相关信息，如下所示。

状态码	含义	响应消息
200	插入数据成功	{ "code": 0, "msg": "operation success",

		<pre>{
		  "affectedCount": 3
		}</pre>

400	插入数据失败	<pre>{
		  "code": 1,
		  "msg": "operation failed, reason...."
		}</pre>

返回参数

参数名	参数含义
affectedCount	影响行数，即插入的文档数量。

query

最近更新时间：2024-01-18 17:43:01

功能介绍

/document/query 接口用于精确查找与查询条件完全匹配的向量，具体支持如下功能。

- 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。

请求示例

> **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。

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。 ○ **unit64**：大于（>）、大于等于（>=）、等于（==）、小于（<）、小于等于（<=）。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\")。 • <value>：表示要匹配的值。 示例：`Filter('author="jerry"').And('page>20')`。
		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。
		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。

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。/document/query 接口返回的状态码以及相关信息，如下所示。

状态码

状态码	含义	响应信息
200	查询数据成功	返回所查询的 Document，具体信息，请参见返回消息 。

400	查询数据失败	<pre>{
		"code": 400,
		"msg": "operation failed, reason...."
		}</pre>

返回消息

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": "0001",
      "vector": [
        0.21230000257492066,
        0.23000000417232514,
        0.21299999952316285
      ],
      "page": 21,
      "author": "罗贯中",
      "bookName": "三国演义"
    },
    {
      "id": "0002",
      "vector": [
        0.21230000257492066,
        0.2199999988079071,
```

```
            0.21299999952316285
        ],
        "page": 22,
        "bookName": "西游记",
        "author": "吴承恩"
    }
  ]
}
```

返回参数

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：2023-12-08 16:15:49

功能介绍

基于相似度匹配的查询方式，**/document/search** 接口用于查找与给定查询向量相似的向量。

- 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。

接口约束

该接口支持根据主键 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，若同时指定，或者指定其中两项，将提示如下信息。

`The search condition must be one of the [documentIds, vectors, retrieves]`

请求示例

> **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。

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 返回参数 。

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",
                "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。具体返回参数含义，请参见 返回参数 。

说明：

- 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": 1.000000238418579,
        "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 返回参数 。

> **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
      }
    ]
  ]
}
```


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	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	表示查询条件。	vectors	否	表示要查询的向量列表。 • 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。
		documentIds	否	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（<=）。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')`
		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	否	指定返回最相似的 Top K 的 K 的值。
		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。

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。/document/search 接口返回的状态码以及相关信息，如下所示。

状态码

状态码	含义	响应消息
200	查询数据成功	返回信息，请参见 请求示例 。具体参数含义，请参见 返回参数 。

400	查询数据失败	<pre>{
		"code": 400,
		"msg": "operation failed, reason...."
		}</pre>

返回参数

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。
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

delete

最近更新时间：2024-01-18 17:43:01

功能介绍

/document/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。

接口约束

索引类型为 FLAT，不支持删除。

请求示例

> **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。

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\")"
    }
}'
```

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。
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。	documentIds	否	表示要删除的 Document 的 ID，可以批量删除，数据元素最大值为20。 **说明：** 同时配置 documentIds 与 filter 参数，删除数据将会取二者的并集。

		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。
				  ○ **unit64**：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

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。/document/delete 接口返回的状态码以及相关信息，如下所示。

状态码	含义	响应消息

200	删除 Document 成功	<pre>{
		  "code": 0,
		  "msg": "operation success",
		  "affectedCount": 2
		}</pre>
		说明：
		删除操作执行成功，可使用 /document/query 查询该数据确认结果。示例中删除 book-emb 集合中，id 为 0001与0002 的数据，分别使用 /document/query 查询 id 为0001与0002的数据，显示 documents 为空值。结果如下所示。
		<pre>{
		  "code": 0,
		  "msg": "operation success",
		  "count": 0,
		  "documents": []
		}</pre>

400	删除 Document 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：2024-01-18 17:43:01

功能介绍

/document/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。

> **说明：**
> 新增字段，在创建 Collection 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。

接口约束

> **注意：**
> 不能变更 Document ID 字段，不要求事务完整性。

请求示例

> **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。

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
}
```

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。 ○ **unit64**：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	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

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。**/document/update** 接口返回的状态码以及相关信息，如下所示。

状态码	含义	响应消息

200	更新数据成功	<pre>{
		  "code": 1,
		  "msg": "operation success",
		  "affectedCount": 1
		}</pre>

400	插入数据失败	<pre>{
		 "code": 1,
		 "msg": "operation failed, reason...."
		}</pre>

返回消息参数

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。

DocumentSet uploadUrl

最近更新时间：2024-04-08 11:37:01

功能介绍

/ai/documentSet/uploadUrl 接口用于获取文件上传路径和授权签名。

> **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/ai/documentSet/uploadUrl \
  -d '{
    "database":"db-test-ai",
    "collectionView":"coll-ai-files",
    "documentSetName":"腾讯云向量数据库.md"
  }'
```

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。

返回信息


```
{
  "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": "AKIDhE7OGtx4MtnneVOECDf*********************************************",
    "TmpSecretKey": "pnIGo********+Jete************************",
    "Token": "4EGei*****************************************************************************************************************"
  },
  "uploadCondition": {
    "maxSupportContentLength":1048576
  },
  "documentSetId": "MTE2ODgzNDAxNDU0MzYxMzk1Mg=="
}
```


返回参数

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。
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： • 0：不追加。默认值为 0。 • 1：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： • 0：不追加。 • 1：将全文的 keywords 追加到切分后的段落。默认值为 1。

```
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} #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：2024-04-09 11:51:31

/ai/documentSet/get 接口用于获取文件完整内容，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。

请求示例

> **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。

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.md"
  }'
```

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。
documnetSetName	否	文件名	-

返回消息


```
{
  "code": 0,
  "msg": "Operation success",
  "requestId": "d348ae93653ba5179bbec46d94b8e49d",
  "count": 1,
  "documentSet": {
    "documentSetId": "11901307631454126 08",
    "documentSetName": "腾讯云向量数据库.md",
    "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
  },
  "author": "Tencent",
  "tags": [
    "向量",
    "Embedding",
    "AI"
  ]

```
  }
}
```

返回参数

参数名	子参数（一级）	子参数（二级）	参数含义
count	-	-	获取的数量。
documentSet	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。
author	-	-	自定义的文件 Meta 信息字段。

getChunks

最近更新时间：2024-04-09 16:16:11

/ai/documentSet/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。

- 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。

请求示例

```
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.md",
  "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：[1,16384]。
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。

返回信息


```
{
 "code": 0,
 "msg": "Operation success",
 "requestId": "68564c113b2fa517d6bbc46d46f*****",
 "documentSetId": "1190130763145******",
 "documentSetName": "腾讯云向量数据库.md",
 "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：2024-01-17 16:14:01

功能介绍

/ai/documentSet/query 接口用于精确查找与查询条件完全匹配的文件，可获取文件 ID、向量化的进度与状态等，可控制所需输出的字段。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

> **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。

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.md"],
      "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	否	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

状态码

状态码	含义	响应信息
200	查询数据成功	返回所查询的 DocumentSet。具体参数，请参见 返回消息 。

400	查询数据失败	{
		"code": 400,
		"msg": "operation failed, reason...."
		}

返回消息

{
"code": 0,
"msg": "Operation success",
"requestId": "85f094becf3ba517bdbec4**********",
"count": 1,
"documentSets": [
{
"documentSetId": "1190130763145******",
"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
},
"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。
		lastUpdateTi	文件最后更新时间。

		me	
		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。
tags	-	-	自定义的文件 Meta 信息字段。

search

最近更新时间：2024-04-09 16:16:11

功能介绍

基于相似度匹配的查询方式，**/ai/documentSet/search** 接口用于在指定的文件中，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。

请求示例

> **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。

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	-	否	以 String 类型输入检索的文本信息。
		documentSetName	-	否	检索的文件名。
		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": "requestId: 02555de4b9089c1797c84051c5719ef8",
 "documents": [
  {
   "score": 0.8924504518508911,
   "data": {
    "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,
    "pre": [
     "### 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"
    ],
    "next": [
     "## 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"
```

```
      ]
     },
     "documentSet": {
      "documentSetId": "1179321629693181952",
      "documentSetName": "腾讯云向量数据库.md",
      "author": "Tencent"
     }
    },
    {
     "score": 0.890502393245697,
     "data": {
      "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,
      "pre": [
       "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"
      ],
      "next": [
       "## 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"
      ]
     },
     "documentSet": {
      "documentSetId": "1179321629693181952",
      "documentSetName": "腾讯云向量数据库.md",
      "author": "Tencent"
     }
    },
    {
     "score": 0.8756946921348572,
     "data": {
      "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,
      "pre": [

      ],
```

```
      "next": [
       "## 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"
      ]
     },
     "documentSet": {
      "documentSetId": "1179321629693181952",
      "documentSetName": "腾讯云向量数据库.md",
      "author": "Tencent"
     }
    }
   ]
}
```


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。
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：2023-12-08 17:20:03

功能介绍

/ai/documentSet/delete 接口用于删除存储于 CollectionView 中的文件。

- **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。**

请求示例

> **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。**

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.md"],
        "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",

		<pre>{
		"affectedCount": 2
		}</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：2023-12-08 17:20:04

功能介绍

/ai/documentSet/update 接口用于 AI 类数据库更改或新增文件 Meta 信息的标量字段。

- **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。**

> **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。**

请求示例

> **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。**

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.md"],
```

```
      "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": ["11790179945894379 52"],
       "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Collection 名称。	-	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。
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

状态码	含义	响应消息

200	更新数据成功	{
		"code": 1,
		"msg": "operation success",
		"affectedCount": 1
		}

400	插入数据失败	{
		"code": 1,
		"msg": "operation failed, reason...."
		}

返回消息参数

{"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：2023-12-08 16:15:49

功能介绍

/index/rebuild 接口用于重建指定 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。

接口约束

> **注意：**
> 索引重建过程中 Colletion 禁止写入、读取。rebuild 索引需要新的内存来构建索引。

请求示例

> **说明：**
> 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，使用 **/index/rebuild** 接口之前，您需要做相关准备事项，并了解 **/index/rebuild** 接口的请求方式与 URL 拼接地址。具体信息，请参见 使用前阅读 。

> **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	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	否	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。
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。 **说明：** 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。 **说明：** 内存资源不足时，可先删除旧索引，在新索引还没有创建完成之前，该表无法正常读写。 • **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**。 **说明：** 内存资源足够的情况下，可不删除旧索引。在新索引还没有创建完成之前，该表可读数据，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： • 0：不限制 CPU 核数。 • 1：CPU 核数为 1。

		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。	

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。

- **ready**：表示当前 Collection 准备就绪，可正常使用。
- **training data**：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 index**：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": ""
    }
  }
}
```

错误码

最近更新时间：2024-03-11 15:47:01

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。

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	插入数据时，指定的 Document ID 已经被使用。
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。