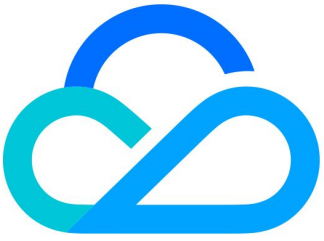
向量数据库

快速入门



腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

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

【商标声明】

腾讯云

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

【服务声明】

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

【联系我们】

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

文档目录

快速入门
- 购买默认规格实例
- 连接前先决条件
- 第一次相似性检索
  - 选择相似性检索场景
  - 写入向量数据并检索
  - 应用 Embedding 相似性检索
- 可视化访问数据库

版权所有：腾讯云计算（北京）有限责任公司

快速入门
购买默认规格实例

最近更新时间：2025-06-09 14:32:22

操作场景

您可根据本文的介绍，快速配置您的第一台向量数据库（Tencent Cloud VectorDB）实例。

地域

当前支持北京、上海、广州、成都、上海自动驾驶云、深圳金融、中国香港、新加坡、硅谷等，其他地域在规划准备中。详情请参见 向量数据库 VectorDB 购买页。

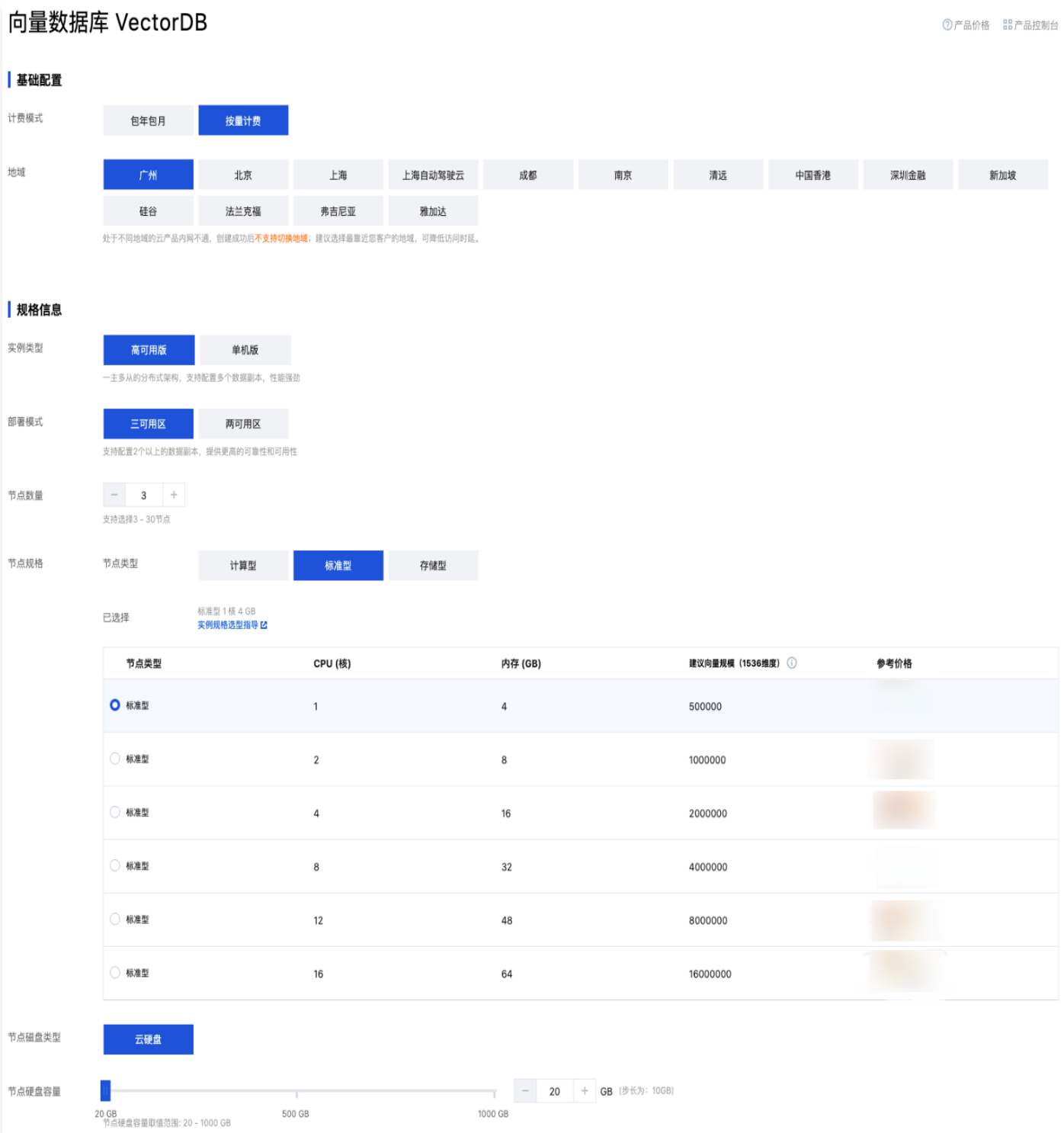
前提条件

- 已注册腾讯云账号并完成实名认证。
  - 如需注册腾讯云账号：请单击 注册腾讯云账号 。
  - 如需完成实名认证：请单击 实名认证 。
- 已创建数据库实例的私有网络与安全组，请参见 私有网络 与 安全组 。

操作步骤

1. 使用腾讯云账号登录 向量数据库 VectorDB 购买页。
2. 请参见下表，配置如下相关参数，如下图所示。

版权所有：腾讯云计算（北京）有限责任公司



分类	界面参数	配置说明
基础配置	计费模式	选择**按量计费**实例。
	地域	处于不同地域的云产品内网不通，新建后不能切换地域，请选择最靠近业务的地域。
规格信息	-	实例类型、部署模式、节点规格、节点数量、磁盘类型及容量等均保持默认配置。

网络和安全组	**网络**	• 在下拉列表分别选择已配置的私有网络及子网。 • 若现有网络不满足需求，请单击**新建私有网络**或**新建子网**，创建所需的网络环境。具体操作，请参见 创建私有网络 。
	安全组	您可以在**选择已有安全组**下拉框中选择已有的安全组，也可以单击**自定义安全组**，设置新的安全组入站规则。具体操作，请参见 安全组 。
实例设置	**实例名**	请在输入框输入实例名称。仅支持长度不超过 60 的中文/英文/数字/-/_。
	服务条款	勾选**我已阅读并同意云数据库服务条款**。

3. 单击右下角的**立即购买**，自动返回实例列表页面，当前实例**状态为创建中**，等待实例状态更新为**运行中**即可使用已购买的实例。

下一步

待购买的实例运行正常，便可以进行 连接前准备事项：配置网络环境、安装 SDK 等。

连接前先决条件

最近更新时间：2025-04-11 17:50:13

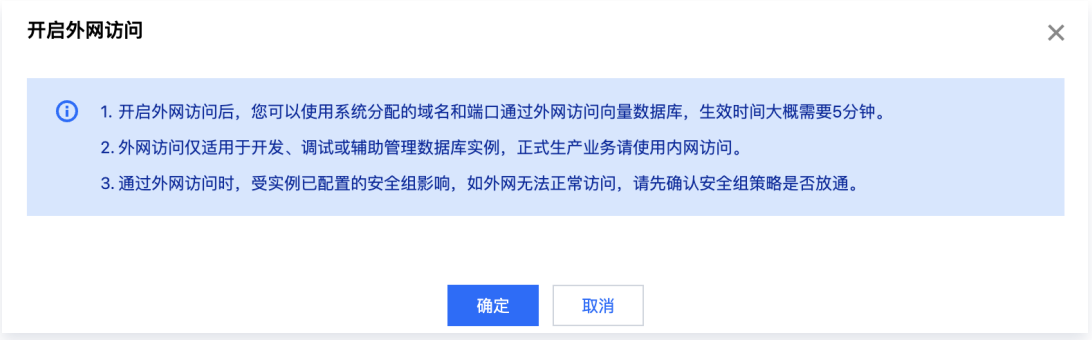
开始连接操作向量数据库之前，您需要确保已准备好必要的网络环境，并已获取连接账户与访问密钥。此外，客户端还需要安装适用的 SDK。

配置网络环境

开启外网功能，系统自动分配域名与端口，在本地 Linux 系统的客户端便可通过外网快速连接数据库。

外网方式（推荐）

1. 登录 向量数据库控制台 。
2. 在实例列表中，找到需开启外网访问的实例。
3. 单击**实例 ID** 进入**实例详情**页面，在**网络信息**区域，单击**外网地址**后面的**开启**。
4. 在**开启外网访问**的小窗口，阅读开启外网的使用说明，如下图所示。



5. 单击**确定**，等待任务执行完成，在**网络信息**区域的**外网地址**，可查看系统分配的外网地址。单击 ，可复制外网地址。

内网方式

- 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 ，通过内网快速连接向量数据库。
- 在腾讯云 CVM 安全组中需配置**出站规则**，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 地址及向量数据库的端口添加到入站规则中，才能连接成功。 操作详情，可参见 安全组 。

获取登录密钥

1. 登录 向量数据库控制台 。
2. 在实例列表中，找到目标实例。
3. 单击目标**实例 ID**，或在其**操作**列，单击**管理**，进入**实例详情**页面。

4. 在**实例详情**页面，选择**密钥管理**。
5. 在**密钥管理**页面，可查看到访问数据库的账户及其对应的 **API 密钥**。单击**复制**，可直接复制 **root** 账户默认的密钥。

安装 SDK

腾讯云向量数据库（Tencent Cloud VectorDB）提供了 Python、Java、Go 语言的 SDK， 辅助您快速访问数据库。

- Python 3.6 及以上版本。
- Java 8 或更高版本。
- Go 1.15 或更高版本。

Python

```
pip install tcvectordb
```

> **说明：**
> 使用 pip 安装 Python SDK 之前，请确保 pip 指向3.6及以上的 Python 版本，否则选用相应的 pip3 安装。

Java

Gradle 项目，请在 build.gradle 文件中添加如下依赖。

```
com.tencent.tcvectordb:vectordatabase-sdk-java:1.1.2
```

Maven 项目，请在 pom.xml 文件中添加如下依赖。

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

```
        <version>1.1.2</version>
    </dependency>
```

Go

```
go get -u github.com/tencent/vectordatabase-sdk-go/tcvectordb
```

下一步

准备好以上事项，便可以开始您的 第一次相似性检索 体验。

第一次相似性检索
选择相似性检索场景

最近更新时间：2024-10-14 18:01:21

腾讯云向量数据库（Tencent Cloud VectorDB）提供了三种常见场景管理数据库进行相似性检索的教程，辅助您快速体验产品能力。

相似性检索	场景描述
直接写入向量数据并检索	适用于已有自身的向量数据，无需使用腾讯云向量数据库进行数据向量化的场景。
基于文本信息写入向量数据	适用于无向量数据，需要应用腾讯云向量数据库的 Embedding 功能输入原始文本进行数据向量化的场景。
基于文件写入向量数据	适用于无向量数据，需要应用 AI 套件上传 Markdown、PPT、Pdf、Word 等类型文件进行数据向量化的场景。

写入向量数据并检索

最近更新时间：2025-04-28 19:11:32

本章节介绍如何连接向量数据库写入数据，并进行第一次相似性检索。以 Linux 操作系统为例，使用 Python、Java、Go SDK 与 HTTP API 示例代码分别进行演示。运行本章节所提供的示例代码，您将初步了解向量数据库的检索能力。

导入 SDK 依赖模块

Python

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

Java

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

Go

```
package main

import (
    "context"
    "log"
    "time"
```

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

创建 Client

导入 SDK 所需的模块之后，需先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。

> **说明：**
> 如下示例 **url** 与 **key** 需要分别替换为已购买的默认版实例的 外网访问地址 与 API Key。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**外网地址**，在**密钥管理**页面直接复制密钥。

Python

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

Java

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

Go

```
func main() {
    var defaultOption = &tcvectordb.ClientOption{
        Timeout:             time.Second * 5,
        MaxIdldConnPerHost: 2,
        IdleConnTimeout:    time.Minute,
        ReadConsistency:    tcvectordb.EventualConsistency,
    }
    client, err := tcvectordb.NewRpcClient("http://10.0.X.X:80",
"root", "eC4bLRy2va*****************************", defaultOption)
    if err != nil {
        panic(err)
    }
}
```

创建数据库

基于创建的客户端对象，创建数据库。

Python

Python SDK 通过 create_database() 接口创建数据库 **db-test**。

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

Java

Java SDK 通过 createDatabase() 创建数据库 **db-test**。

```
Database db = client.createDatabase("db-test");
```

Go

Go SDK 通过 CreateDatabase() 创建数据库 **db-test**。

```
var (
    ctx                 = context.Background()
    database            = "db-test"
)
db, _ := client.CreateDatabase(ctx, database)
```

Curl

HTTP 使用 /database/create 接口创建数据库 **db_test**。如下示例，**url** 地址与 **api_key** 需要分别替换为已购买实例的外网地址与 API Key。

```
curl -i -X POST \
 -H 'Content-Type: application/json' \
 -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
http://10.0.X.X:80/database/create \
 -d '{
 "database": "db-test"
}'
```

创建集合

1. 创建集合之前，需先设计索引结构，指定索引字段。如下示例，创建一个可写入3维向量数据存储书籍的集合 **book-vector**，其书籍信息字段包括： **id**、**vector**、**bookName**，分别对各字段构建索引。有关索引的具体信息，请参见 Index 。
   - 主键索引（Primary Key Index）：固定且必须，对应字段 **id**，每条数据的唯一标识。并对主键 **id** 构建 Filter 索引，以便可通过 id 的条件表达式进行特定行的检索。
   - 向量索引（Vector Index）：固定且必须，对应字段 **vector**，对向量数据构建索引，指定向量数据的维度、数据存储的索引类型、相似性计算方法及相关索引参数。
   - Filter 索引（Filter Index）：需根据检索需求选取可作为条件查询过滤数据的字段。通常，向量数据对应的文本字段，不具有过滤属性，则无需对该字段建立索引，否则，将浪费较大的内存也无实际意义。如下示例，预按书籍的名称过滤数据，对 **bookName** 字段建立 Filter 索引。
2. 创建集合，高可用默认规格实例，其分片 shard 可为1，副本 replicas 可为1。

Python

Python SDK 通过接口 create_collection() 创建集合 book-vector。

```
index = Index(
            FilterIndex(name='id', field_type=FieldType.String,
index_type=IndexType.PRIMARY_KEY),
            VectorIndex(name='vector', dimension=3,
index_type=IndexType.HNSW,
                        metric_type=MetricType.COSINE,
params=HNSWParams(m=16, efconstruction=200)),
            FilterIndex(name='bookName',
field_type=FieldType.String, index_type=IndexType.FILTER)
        )
# create a collection

coll = db.create_collection(
            name='book-vector',
            shard=1,
            replicas=1,
            description='this is a collection book vector',
            index=index
        )
print(vars(coll))
```

Java

Java SDK 通过接口 createCollection 创建集合 book-vector。

```
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("book-vector")
        .withShardNum(1)
        .withReplicaNum(1)
        .withDescription("this is a collection book vector")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
        .addField(new VectorIndex("vector", 3, IndexType.HNSW,
```

版权所有：腾讯云计算（北京）有限责任公司

```
                MetricType.COSINE, new HNSWParams(16, 200)))
        .addField(new FilterIndex("bookName", FieldType.String,
IndexType.FILTER))
        .build();
Collection collection = db.createCollection(collectionParam);
```

Go

Go SDK 通过接口 CreateCollection() 创建集合 book-vector。

```
var (
    ctx                 = context.Background()
    database            = "db-test"
    collectionName      = "book-vector"
)
index := tcvectordb.Indexes{
    VectorIndex: []tcvectordb.VectorIndex{
        {
            Dimension: 3,
            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,
```

```
        },
    },
}
coll, _ := db.CreateCollection(ctx, collectionName, 1, 1, "this is a
collection book vector", index)
log.Printf("CreateCollection success: %v: %v", coll.DatabaseName,
coll.CollectionName
```

Curl

HTTP 通过 /collection/create 创建集合 book-vector。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "replicaNum": 1,
    "shardNum": 1,
    "description": "this is a collection book vector",
    "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"
        }
    ]
}'
```

插入数据

如下示例，为书籍集合 **book-vector** 写入3条数据。

> **说明：**
> - 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。如下示例，page 与 author 为新定义的书籍信息字段。
> - 创建集合时，并未对 page 与 author 构建 Filter 索引，因此，二者不具有过滤属性，仅 bookName 具有过滤属性。

Python

Python SDK 通过接口 upsert() 为集合 **book-vector** 写入向量数据。

```
res = client.upsert(
            database_name='db-test',
            collection_name='book-vector',
            documents=[
                Document(id='0001', vector=[
                        0.2123, 0.23, 0.213], author='罗贯中',
bookName='三国演义', page=21),
                Document(id='0002', vector=[
                        0.2123, 0.22, 0.213], author='吴承恩',
bookName='西游记', page=22),
                Document(id='0003', vector=[
                        0.2123, 0.21, 0.213], author='曹雪芹',
bookName='红楼梦', page=23)
            ]
        )
```

Java

Java SDK 通过 upsert() 接口为集合 book-vector 批量插入数据。

```
Document doc1 = Document.newBuilder()
        .withId("0001")
        .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", 21))
        .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))
        .build();
Document doc3 = Document.newBuilder()
        .withId("0003")
        .withVector(Arrays.asList(0.2123, 0.21, 0.213))
        .addDocField(new DocField("bookName", "红楼梦"))
        .addDocField(new DocField("author", "曹雪芹"))
        .addDocField(new DocField("page", 23))
        .build();
InsertParam insertParam = InsertParam.newBuilder()
        .addDocument(doc1)
        .addDocument(doc2)
        .addDocument(doc3)
        .build();
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-vector";
AffectRes affectRes = client.upsert(DBNAME, COLL_NAME, insertParam);
System.out.println("\tres: " + affectRes);
```

Go

Go SDK 通过 Upsert() 接口为集合 book-vector 批量插入数据。

```
result, err := client.Upsert(ctx, database, collectionName,
[]tcvectordb.Document{
    {
        Id:     "0001",
        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: 21},
        },
    },
    {
        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},
        },
    },
    {
        Id:     "0003",
        Vector: []float32{0.2123, 0.21, 0.213},
        Fields: map[string]tcvectordb.Field{
            "bookName": {Val: "红楼梦"},
            "author":   {Val: "曹雪芹"},
            "page":     {Val: 23},
        },
    },
}, &tcvectordb.UpsertDocumentParams{})
log.Printf("upsert result: %+v", result)
```

Curl

如下示例，通过 /document/upsert 给集合 book-vector 批量插入数据。插入数据以 Document 为最小单元，插入3个 Document。

```
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
        }
    ]
}'
```

执行成功，返回如下信息：

```
{
    "code": 0,
    "msg": "operation success",
    "affectedCount": 3
}
```

相似性检索

相似度检索 是基于向量数据之间的相似度计算方法来检索与查询向量最相似的数据。

Python

检索与输入向量数据相似的数据

Python SDK 提供了 search() 按照 Vector 搜索的能力，可根据指定的多个向量查找 TopK 个相似性结果。如下示例，检索与 vectors 字段指定的三组向量数据分别相似，且满足 bookName 条件表达式的 Top3 数据。

```
# 1. vectors 指定了需检索的向量数据。
# 2. filter 指定了 bookName 字段的条件表达式，过滤数据。
# 3. limit 限制每个单元返回的相似性数据的条数，如 vector 写入三组向量数据，
limit 为3，则每组向量返回 top3 的相似数据。
# 4. 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。
doc_lists = client.search(
            database_name='db-test',
            collection_name='book-vector',
            vectors=[[0.3123, 0.43, 0.213],[0.315, 0.4, 0.216],
[0.40, 0.38, 0.26]],
            filter='bookName in ("三国演义","西游记")',
            params=SearchParams(ef=200),
            limit=3
        )
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，则返回实际检索到的 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': 0.971423, 'bookName': '三国演义', 'page': 21,
'author': '罗贯中'}
{'id': '0002', 'score': 0.966884, 'author': '吴承恩', 'page': 22,
'bookName': '西游记'}
1
{'id': '0001', 'score': 0.978463, 'page': 21, 'bookName': '三国演义',
'author': '罗贯中'}
{'id': '0002', 'score': 0.974783, 'author': '吴承恩', 'bookName': '西
游记', 'page': 22}
2
{'id': '0001', 'score': 0.986069, 'bookName': '三国演义', 'page': 21,
'author': '罗贯中'}
{'id': '0002', 'score': 0.985201, 'author': '吴承恩', 'bookName': '西
游记', 'page': 22}
```

检索与指定 Document ID 相似的数据

Python SDK 还提供了 search_by_id() 按 id 检索的能力。如下示例，检索与 id（Document ID）为 0001、0002 分别相似，且满足 bookName 条件表达式相似度最高的 Top3 数据。

```
# 1. document_ids 指定了需检索文档的 id。
# 2. filter 指定了 bookName 字段的条件表达式，过滤数据。
# 3. limit 限制每个单元返回的相似性数据的条数，如 document_ids 传入2个数据
id，limit 为 3，则返回每个 id 相似的 top3 向量。
```

```
# 4. 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。
doc_lists = client.search_by_id(
                database_name='db-test',
                collection_name='book-vector',
                document_ids=['0001','0002'],
                filter='bookName in ("三国演义","西游记")',
                params=SearchParams(ef=200),
                limit=3
            )
for i, docs in enumerate(doc_lists):
                print(i)
                for doc in docs:
                        print(doc)
```

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。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。

```
0
{'id': '0001', 'score': 1.0, 'author': '罗贯中', 'bookName': '三国演义', 'page': 21}
{'id': '0002', 'score': 0.999773, 'page': 22, 'author': '吴承恩', 'bookName': '西游记'}
1
{'id': '0002', 'score': 1.0, 'author': '吴承恩', 'bookName': '西游记', 'page': 22}
{'id': '0001', 'score': 0.999773, 'bookName': '三国演义', 'page': 21, 'author': '罗贯中'}
```

Java

检索与输入向量相似的数据

Java SDK 提供了 search() 接口按照 Vector 搜索的能力。如下示例，检索与 **Vectors** 字段指定的三组向量数据分别相似，且满足 **bookName** 条件表达式的 Top3 条数据。

```
SearchByVectorParam searchByVectorParam =
SearchByVectorParam.newBuilder()
        // Vectors 指定了需检索的向量数据
        .withVectors(Arrays.asList(Arrays.asList(0.3123,
0.43, 0.213), Arrays.asList(0.315, 0.4, 0.216),
Arrays.asList(0.40, 0.38, 0.26)))
        // 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))
         // 指定返回的最相似的 Top K 的 K 值
        .withLimit(3)
        // 设置标量字段的 Filter 表达式，过滤所需查询的文档
        .withFilter("bookName in (\"三国演义\",\"西游记\")")
        .build();
// 输出相似度检索结果
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-vector";
List<List<Document>> svDocs = client.search(DBNAME, COLL_NAME,
searchByVectorParam);
int i = 0;
for (List<Document> docs : svDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

检索结果，如下所示。

说明：

- 输出数组的 Document 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，如果检索的数据不足 K 条，则返回实际检索到的 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。

```
 res: 0
 res: {"id":"0001","score":0.971423,"page":21,"bookName":"三国演
义","author":"罗贯中"}
 res: {"id":"0002","score":0.966884,"bookName":"西游
记","page":22,"author":"吴承恩"}
 res: 1
 res: {"id":"0001","score":0.978463,"page":21,"author":"罗贯
中","bookName":"三国演义"}
 res: {"id":"0002","score":0.974783,"bookName":"西游
记","page":22,"author":"吴承恩"}
 res: 2
 res: {"id":"0001","score":0.986069,"bookName":"三国演义","author":"罗
贯中","page":21}
 res: {"id":"0002","score":0.985201,"author":"吴承恩","bookName":"西游
记","page":22}
```

根据 ID 进行相似度检索

Java SDK 还提供了 searchById() 接口按 id 检索的能力。如下示例，检索与 id（Document ID）为 0001、0002分别相似，且满足 **bookName** 条件表达式的 Top3 条数据。

```
SearchByIdParam searchByIdParam = SearchByIdParam.newBuilder()
        .withDocumentIds(Arrays.asList("0001", "0002"))
        // 若使用 HNSW 索引，则需要指定参数 ef，ef 越大，召回率越高，但也会影
响检索速度
        .withParams(new HNSWSearchParams(200))
         // 指定 Top K 的 K 值
        .withLimit(3)
        // 使用 filter 过滤数据
        .withFilter("bookName in (\"三国演义\",\"西游记\")")
        .build();
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-vector";
List<List<Document>> siDocs = client.searchById(DBNAME, COLL_NAME,
searchByIdParam);
int i = 0;
for (List<Document> docs : siDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

检索结果，如下所示。

> **说明：**
> 输出数组的 Document 顺序与查询时配置的参数 **DocumentIds** 输入的顺序一致。如下 documents 中第一个中括号为 id 为0001的相似数据，第二个中括号为 id 为0002的相似数据。

```
res: 0
res: {"id":"0001","score":1.0,"bookName":"三国演义","page":21,"author":"罗贯中"}
res: {"id":"0002","score":0.999773,"author":"吴承恩","bookName":"西游记","page":22}
res: 1
res: {"id":"0002","score":1.0,"author":"吴承恩","bookName":"西游记","page":22}
res: {"id":"0001","score":0.999773,"bookName":"三国演义","author":"罗贯中","page":21}
```

Go

检索与输入向量相似的数据

Go SDK 提供了 Search() 接口按照 Vector 搜索的能力。如下示例，检索与指定的三组向量数据分别相似，且满足 **bookName** 条件表达式的 Top3 条数据。

```
filter := tcvectordb.NewFilter(`bookName in ("三国演义","西游记")`)
searchRes, _ := client.Search(ctx, database, collectionName, []
[]float32{
    {0.3123, 0.43, 0.213},
    {0.315, 0.4, 0.216},
    {0.40, 0.38, 0.26},
}, &tcvectordb.SearchDocumentParams{
    Filter:          filter,
    Params:          &tcvectordb.SearchDocParams{Ef: 200},
    Limit:           3,
})
for i, docs := range searchRes.Documents {
    log.Printf("doc %d result: ", i)
    for _, doc := range docs {
        log.Printf("document: %+v", doc)
```

```
    }
}
```

检索结果，如下所示。

> **说明：**
> - 输出数组的 Document 顺序与查询时配置的向量数据的顺序一致。
> - 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，如果检索的数据不足 K 条，则返回实际检索到的 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。

```
2024/01/03 17:19:11 doc 0 result:
2024/01/03 17:21:29 document: {Id:0001 Vector:[] Score:0.971423
Fields:map[author:罗贯中 bookName:三国演义 page:21]}
2024/01/03 17:21:29 document: {Id:0002 Vector:[] Score:0.966884
Fields:map[author:吴承恩 bookName:西游记 page:22]}
2024/01/03 17:21:29 doc 1 result:
2024/01/03 17:21:29 document: {Id:0001 Vector:[] Score:0.978463
Fields:map[author:罗贯中 bookName:三国演义 page:21]}
2024/01/03 17:21:29 document: {Id:0002 Vector:[] Score:0.974783
Fields:map[author:吴承恩 bookName:西游记 page:22]}
2024/01/03 17:21:29 doc 2 result:
2024/01/03 17:21:29 document: {Id:0001 Vector:[] Score:0.986069
Fields:map[author:罗贯中 bookName:三国演义 page:21]}
2024/01/03 17:21:29 document: {Id:0002 Vector:[] Score:0.985201
Fields:map[author:吴承恩 bookName:西游记 page:22]}
```

检索与指定 id 相似的数据

Go SDK 还提供了 SearchById() 接口按 id 检索的能力。如下示例，检索与 id（Document ID）为 0001、0002分别相似，且满足 **bookName** 条件表达式的 Top3 条数据。

```
documentId := []string{"0001", "0002"}
filter := tcvectordb.NewFilter(`bookName in ("三国演义","西游记")`)
searchRes, err := client.SearchById(ctx, database, collectionName,
documentId, &tcvectordb.SearchDocumentParams{
```

```
    Filter:          filter,
    Params:          &tcvectordb.SearchDocParams{Ef: 200},
    Limit:           3,
})
for i, docs := range searchRes.Documents {
    log.Printf("doc %d result: ", i)
    for _, doc := range docs {
        log.Printf("document: %+v", doc)
    }
}
```

检索结果，如下所示。

```
2024/01/03 17:24:27 doc 0 result:
2024/01/03 17:24:27 document: {Id:0001 Vector:[] Score:1
Fields:map[author:罗贯中 bookName:三国演义 page:21]}
2024/01/03 17:24:27 document: {Id:0002 Vector:[] Score:0.999773
Fields:map[author:吴承恩 bookName:西游记 page:22]}
2024/01/03 17:24:27 doc 1 result:
2024/01/03 17:24:27 document: {Id:0002 Vector:[] Score:1
Fields:map[author:吴承恩 bookName:西游记 page:22]}
2024/01/03 17:24:27 document: {Id:0001 Vector:[] Score:0.999773
Fields:map[author:罗贯中 bookName:三国演义 page:21]}
```

Curl

检索与输入向量相似的数据

HTTP 支持 /document/search 接口按照 Vector 检索的能力，查询与指定向量 vectors 字段相似的数据。如下示例，检索与指定的三组向量数据分别相似，且满足 **bookName** 条件表达式的 Top3 条数据。

```
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" :  {
        "vectors" :  [
            [
                0.3123 ,
                0.43 ,
                0.213
            ] ,
            [
                0.315 ,
                0.4 ,
                0.216
            ] ,
            [
                0.40 ,
                0.38 ,
                0.26
            ]
        ] ,
        "params" :  {
            "ef" :  200
        } ,
        "filter" :  "bookName in (\"三国演义\",\"西游记\")" ,
        "limit" :  3
    }
} '
```

执行成功，返回如下信息：

说明：

- 输出数组的 Document 顺序与查询时配置的参数 `vectors` 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。

版权所有：腾讯云计算（北京）有限责任公司

```
{
  "code": 0,
  "msg": "operation success",
  "documents": [
    [
      {
        "id": "0001",
        "score": 0.971423,
        "page": 21,
        "author": "罗贯中",
        "bookName": "三国演义"
      },
      {
        "id": "0002",
        "score": 0.966884,
        "author": "吴承恩",
        "page": 22,
        "bookName": "西游记"
      }
    ],
    [
      {
        "id": "0001",
        "score": 0.978463,
        "author": "罗贯中",
        "bookName": "三国演义",
        "page": 21
      },
      {
        "id": "0002",
        "score": 0.974783,
        "page": 22,
        "bookName": "西游记",
        "author": "吴承恩"
      }
    ],
    [
      {
        "id": "0001",
        "score": 0.986069,
        "bookName": "三国演义",
        "author": "罗贯中",
        "page": 21
```

```
      },
      {
        "id": "0002",
        "score": 0.985201,
        "author": "吴承恩",
        "page": 22,
        "bookName": "西游记"
      }
    ]
  ]
}
```

检索与指定 ID 相似的数据

HTTP 还提供了 /document/search 按 id 检索的能力。如下示例，检索与 id（Document ID）为 0001、0002分别相似，且满足 bookName 条件表达式的 Top3 条数据。

```
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"
        ],
        "params": {
            "ef": 200
        },
        "retrieveVector": true,
        "filter": "bookName in (\"三国演义\",\"西游记\")",
        "limit": 3
    }
}'
```

执行成功，返回如下信息：

> **说明：**

> 输出数组的 Document 顺序与查询时配置的参数 documentIds 输入 id 的顺序一致。如下 documents 中第一个中括号为 id 为 0001的相似数据，第二个中括号为 id 为 0002 的相似数据。


```
{
  "code": 0,
  "msg": "operation success",
  "documents": [
    [
      {
        "id": "0001",
        "score": 1.0,
        "bookName": "三国演义",
        "author": "罗贯中",
        "page": 21
      },
      {
        "id": "0002",
        "score": 0.999773,
        "page": 22,
        "bookName": "西游记",
        "author": "吴承恩"
      }
    ],
    [
      {
        "id": "0002",
        "score": 1.0,
        "page": 22,
        "author": "吴承恩",
        "bookName": "西游记"
      },
      {
        "id": "0001",
        "score": 0.999773,
        "page": 21,
        "author": "罗贯中",
        "bookName": "三国演义"
      }
    ]
  ]
```

```
}
```

删除数据库

Python

Python SDK 通过 drop_database() 接口删除数据库 db-test。

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

Java

Java SDK 通过 dropDatabase() 删除数据库 db-test。

```
client.dropDatabase("db-test");
```

Go

Go SDK 通过 DropDatabase() 删除数据库 db-test。

```
result, _ := client.DropDatabase(context.Background(), database)
```

Curl

HTTP 使用 /database/drop 接口删除数据库 db_test。如下示例，url 地址与 api_key 需要分别替换为已购买的实例的外网地址与 API Key。

```
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"
}'
```

应用 Embedding 相似性检索

最近更新时间：2025-05-23 17:17:53

腾讯云向量数据库（Tencent Cloud VectorDB）默认开通 Embedding 功能。本章节介绍如何应用 Embedding 功能写入原始文本，并基于输入的文本信息进行相似性检索。以 Linux 操作系统为例，使用 Python、Java、Go SDK 与 HTTP API 示例代码分别演示。运行本章节所提供的示例代码，您将初步了解基于文本信息进行数据写入检索的能力。

导入 SDK 依赖模块

Python

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

Java

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

Go

```
package main

import (
    "context"
```

```
    "log"
    "time"

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

创建 Client

导入 SDK 所需的模块之后，需先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。

> **说明：**
> 如下示例 **url** 与 **key** 需要分别替换为已购买的默认规格实例的 外网访问地址 与 API Key。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**外网地址**，在**密钥管理**页面直接复制密钥。

Python

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

Java

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

```
}
```

Go

```
func main() {
    var defaultOption = &tcvectordb.ClientOption{
        Timeout:             time.Second * 5,
        MaxIdldConnPerHost: 2,
        IdleConnTimeout:     time.Minute,
        ReadConsistency:     tcvectordb.EventualConsistency,
    }
    client, err := tcvectordb.NewRpcClient("http://10.0.X.X:80",
"root", "eC4bLRy2va*****************************", defaultOption)
    if err != nil {
        panic(err)
    }
}
```

创建数据库

Python

Python SDK 通过 create_database() 接口创建数据库 **db-test**。

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

Java

Java SDK 通过 createDatabase() 创建数据库 **db-test**。

```
 Database db = client.createDatabase("db-test");
```

Go

Go SDK 通过 CreateDatabase() 创建数据库 **db-test**。

```
var (
    ctx                 = context.Background()
    database            = "db-test"
)
db, _ := client.CreateDatabase(ctx, database)
```

Curl

HTTP 使用 /database/create 接口创建名为 **db_test** 的数据库。其中， **url** 地址与 **api_key** 需要分别替换为已购买实例的外网地址与 API Key。

```
curl -i -X POST \
 -H 'Content-Type: application/json' \
 -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
http://10.0.X.X:80/database/create \
 -d '{
 "database": "db-test"
 }'
```

创建集合

1. 创建集合之前，需先设计索引结构，指定索引字段。如下示例，创建一个可写入3维向量数据存储书籍的集合 book-emb，其书籍信息字段包括： id、vector、bookName，分别对各字段构建索引。有关索引的具体信息，请参见 Index 。
   - 主键索引（Primary Key Index）：固定且必须，对应字段 **id**，每条数据的唯一标识。并对主键 **id** 构建 Filter 索引，以便可通过 id 的条件表达式进行特定行的检索。
   - 向量索引（Vector Index）：固定且必须，对应字段 **vector**，对向量数据构建索引，指定向量数据存储的索引类型、相似性计算方法及相关索引参数。
   - Filter 索引（Filter Index）：需根据检索需求选取可作为条件查询过滤数据的字段。通常，向量数据对应的文本字段，不具有过滤属性，则无需对该字段建立索引，否则，将浪费较大的内存也无实际意义。如下示例，预按书籍的名称过滤数据，对 **bookName** 字段建立 Filter 索引。

版权所有：腾讯云计算（北京）有限责任公司

2. 应用 Embedding 功能，则需配置 Embedding 参数。如下示例，自定义文本信息的字段名为 **text**、选择 Embedding 模型 **bge-base-zh**，向量数据字段固定为 **vector**。
3. 创建集合，高可用默认规格例，其分片 shard 可为1，副本 replicas 可为1。

Python

Python SDK 通过接口 create_collection() 创建集合 **book-vector**。

```
# 第一步：设计索引字段
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='bookName',
field_type=FieldType.String, index_type=IndexType.FILTER)
        )

# 第二步：配置 Embedding 参数
# 1. 指定文本字段与向量字段，向量字段固定为vector
# 2. 指定 Embedding 模型，推荐使用 BGE_BASE_ZH。
ebd = Embedding(vector_field='vector', field='text',
model_name='bge-base-zh')

# 第三步，创建 Collection
coll = db.create_collection(
            name='book-emb',
            shard=1,
            replicas=1,
            description='this is an embedding collection',
            embedding=ebd,
            index=index
        )
print(vars(coll))
```

Java

Java SDK 通过接口 createCollection 创建集合 book-emb。

```
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("book-emb")
        .withShardNum(1)
        .withReplicaNum(1)
        .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))

.withEmbedding(Embedding.newBuilder().withVectorField("vector")
                .withField("text").withModelName("bge-base-
zh").build())
        .build();
Collection collection = db.createCollection(collectionParam);
```

Go

Go SDK 通过接口 CreateCollection() 创建集合 book-emb。

```
var (
    embeddingCollection = "book-emb"
)
// 第一步：设计索引字段
index := tcvectordb.Indexes{
    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,
        },
    },
}
// 第二步：配置 Embedding 参数
param := &tcvectordb.CreateCollectionParams{
        Embedding: &tcvectordb.Embedding{
            Field:       "text",
            VectorField: "vector",
            ModelName:   "bge-base-zh",
        },
    }
// 第三步：创建集合
coll, _ := db.CreateCollection(ctx, embeddingCollection, 1, 1, "this
is an embedding collection", index, param)
log.Printf("CreateCollection success: %v: %v", coll.DatabaseName,
coll.CollectionName)
```

Curl

HTTP 通过 /collection/create 创建集合 book-emb。

```
curl -i -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer
account=root&api_key=A5VOgsMpGWJhUI0WmUbY********************' \
  http://10.0.X.X:80/collection/create \
```

```
  -d '{
    "database": "db-test",
    "collection": "book-emb",
    "replicaNum": 1,
    "shardNum": 1,
    "description": "this is an embedding collection",
    "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"
        }
    ]
}'
```

基于文本信息写入数据

如下示例，为书籍集合 **book-emb** 写入3条数据，其中 **text** 字段为写入的文本信息。Embedding 将文本信息自动向量化后写入数据库，并保存原始文本。

> **说明：**

- 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。如下示例，page 与 author 为新定义的书籍信息字段。
- 创建集合时，并未对 page 与 author 构建 Filter 索引，因此，二者不具有过滤属性，仅 bookName 具有过滤属性。
- 写入数据，可能存在一定延迟。

Python

Python SDK 通过 upsert() 接口为集合 **book-emb**，写入原始文本。

```
res = client.upsert(
            database_name='db-test',
            collection_name='book-emb',
            documents=[
                Document(id='0001', text="话说天下大势，分久必合，合久必分。", author='罗贯中', bookName='三国演义', page=21),
                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)
            ],
        )
```

Java

Java SDK 通过 upsert() 接口可直接传入原始文本。

```
Document doc1 = Document.newBuilder()
        .withId("0001")
        .addDocField(new DocField("text", "话说天下大势，分久必合，合久必分。"))
        .addDocField(new DocField("bookName", "三国演义"))
        .addDocField(new DocField("author", "罗贯中"))
        .addDocField(new DocField("page", 21))
        .build();
Document doc2 = Document.newBuilder()
        .withId("0002")
```

```
        .addDocField(new DocField("text", "混沌未分天地乱，茫茫渺渺无人
间。"))
        .addDocField(new DocField("bookName", "西游记"))
        .addDocField(new DocField("author", "吴承恩"))
        .addDocField(new DocField("page", 22))
        .build();
Document doc3 = Document.newBuilder()
        .withId("0003")
        .addDocField(new DocField("text", "甄士隐梦幻识通灵，贾雨村风尘怀
闺秀。"))
        .addDocField(new DocField("bookName", "红楼梦"))
        .addDocField(new DocField("author", "曹雪芹"))
        .addDocField(new DocField("page", 23))
        .build();

InsertParam insertParam = InsertParam.newBuilder()
        .addDocument(doc1)
        .addDocument(doc2)
        .addDocument(doc3)
        .build();
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-emb";
AffectRes affectRes = client.upsert(DBNAME, COLL_NAME, insertParam);
```

Go

Go SDK 通过 Upsert() 接口为集合 book-vector 批量插入数据。

```
result, err := client.Upsert(ctx, database, embeddingCollection,
[]tcvectordb.Document{
    {
        Id:     "0001",
        Fields: map[string]tcvectordb.Field{
            "text":  {Val: "话说天下大势，分久必合，合久必分。"},
            "bookName": {Val: "三国演义"},
            "author":   {Val: "罗贯中"},
            "page":     {Val: 21},
        },
    },
    {
        Id:     "0002",
```

```
        Fields: map[string]tcvectordb.Field{
            "text":  {Val: "混沌未分天地乱，茫茫渺渺无人间。"},
            "bookName": {Val: "西游记"},
            "author":   {Val: "吴承恩"},
            "page":     {Val: 22},
        },
    },
    {
        Id:     "0003",
        Fields: map[string]tcvectordb.Field{
            "text":  {Val: "甄士隐梦幻识通灵，贾雨村风尘怀闺秀。"},
            "bookName": {Val: "红楼梦"},
            "author":   {Val: "曹雪芹"},
            "page":     {Val: 23},
        },
    },
}, &tcvectordb.UpsertDocumentParams{})
log.Printf("upsert result: %+v", result)
```

Curl

如下示例，通过 /document/upsert 为集合 book-emb，写入原始文本。

```
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",

    "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
        }
    ]
}'
```

相似性检索

基于 Embedding 功能的相似性检索，支持检索与输入的文本信息相似的文本。如下示例，检索与 **embeddingItems** 参数输入的文本信息相似度最高，且满足 **bookName** 条件表达式的文本。

说明：

- **params**：指定索引类型对应的查询参数。其中，**ef** 为 HNSW 索引类型对应的检索参数，指定寻找节点邻居遍历的范围，默认为200。**ef** 越大，召回率越高。
- **filter**：指定了 bookName 字段的条件表达式，过滤数据。
- **limit**：限制每个单元返回的相似性数据的条数，如 limit 为3，则返回 top3 的相似数据。
- **retrieve_vector**：指定是否输出向量字段。示例中，文本信息被 Embedding 向量化为768维数据，数据量大，不便展示，设置为 False。
- **output_fields**：可自定义需要输出的字段。若不自定，则返回所有字段。

Python

Python SDK 提供了 searchByText() 接口按照输入的文本批量进行相似性查询的能力。如下示例，在集合 **book-emb** 中，检索与 **embeddingItems** 参数配置的文本信息相似，且满足 **bookName** 条件表达式的文本。

```
doc_lists = client.searchByText(
                database_name='db-test',
```

```
                collection_name='book-emb',
                embeddingItems=['天下大势，分久必合，合久必分'],
                filter='bookName in ("三国演义","西游记")',
                params=SearchParams(ef=200),
                limit=3,
                retrieve_vector=False,
                output_fields=['bookName','author','text']
            )
for i, docs in enumerate(doc_lists.get("documents")):
                print(i)
                for doc in docs:
                        print(doc)
```

检索结果，如下所示。

> **说明：**
> 检索结果将按照相似程度的高低排列。相似度最高的结果会排在最前面，最低的结果则排在后面。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', 'score': 0.979274, 'text': '话说天下大势，分久必合，合久必分。', 'bookName': '三国演义', 'author': '罗贯中'}
{'id': '0002', 'score': 0.790986, 'bookName': '西游记', 'text': '混沌未分天地乱，茫茫渺渺无人间。', 'author': '吴承恩'}
```

Java

Java SDK 通过 searchByEmbeddingItems() 接口，在集合 **book-emb** 中，检索与 **EmbeddingItems** 参数的文本信息最相似，且满足 **bookName** 条件表达式的文本。

```
 SearchByEmbeddingItemsParam searchByEmbeddingItemsParam = SearchByEmbeddingItemsParam.newBuilder()
        .withEmbeddingItems(Arrays.asList("天下大势，分久必合，合久必分"))
        // 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("bookName in (\"三国演义\",\"西游记\")")
        // 指定返回的 fields
        .withOutputFields(Arrays.asList("author", "bookName",
"text"))
        .build();
private static final String DBNAME = "db-test";
private static final String COLL_NAME = "book-emb";
List<List<Document>> siDocs = client.searchByEmbeddingItems(DBNAME,
COLL_NAME, searchByEmbeddingItemsParam).getDocuments();
int i = 0;
for (List<Document> docs : siDocs) {
    System.out.println("\tres: " + i++);
    for (Document doc : docs) {
        System.out.println(doc.toString());
    }
}
```

检索结果，如下所示。

> **说明：**
> 检索结果将按照相似程度的高低排列。相似度最高的结果会排在最前面，最低的结果则排在后面。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。

```
res: 0
res: {"id":"0001","score":0.979274,"bookName":"三国演义","author":"罗
贯中","text":"话说天下大势，分久必合，合久必分。"}
res: {"id":"0002","score":0.790986,"bookName":"西游记","author":"吴承
恩", "text":"沌未分天地乱，茫茫渺渺无人间。"}
```

Go

Go SDK 通过 SearchByText() 接口，在集合 **book-emb** 中，检索与 text 字段的文本信息最相似，且满足 **bookName** 条件表达式的文本。

```
filter := tcvectordb.NewFilter(`bookName in ("三国演义","西游记")`)
searchRes, _ := client.SearchByText(ctx, database,
embeddingCollection, map[string][]string{"text": {"天下大势，分久必合，
合久必分"}}, &tcvectordb.SearchDocumentParams{
    Params:         &tcvectordb.SearchDocParams{Ef: 200},
    RetrieveVector: false,
    Limit:          3,
    Filter:         filter,
    OutputFields:   []string{"author", "bookName", "text"},
})
for i, docs := range searchRes.Documents {
    log.Printf("doc %d result: ", i)
    for _, doc := range docs {
        log.Printf("document: %+v", doc)
    }
}
```

检索结果，如下所示。

> **说明：**
> 检索结果将按照相似程度的高低排列。相似度最高的结果会排在最前面，最低的结果则排在后面。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。

```
2024/01/04 17:42:05 doc 0 result:
2024/01/04 17:42:05 document: {Id:0001 Vector:[] Score:0.979274
Fields:map[author:罗贯中 bookName:三国演义 page:21 text:话说天下大势，分久
必合，合久必分。]}
2024/01/04 17:42:05 document: {Id:0002 Vector:[] Score:0.790986
Fields:map[author:吴承恩 bookName:西游记 page:22 text:混沌未分天地乱，茫茫
渺渺无人间。]}
```

Curl

HTTP 使用 /document/search 接口，在集合 **book-emb** 中，检索与 **embeddingItems** 参数的文本信息相似最高，且满足 **bookName** 条件表达式的文本。

```
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"
        ]
    }
}'
```

检索结果，如下所示。

> **说明：**
> 检索结果将按照相似程度的高低排列。相似度最高的结果会排在最前面，最低的结果则排在后面。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": 0.979274,
        "author": "罗贯中",
        "bookName": "三国演义",
        "text": "话说天下大势，分久必合，合久必分。"
      },
      {
        "id": "0002",
        "score": 0.790986,
        "text": "混沌未分天地乱，茫茫渺渺无人间。",
        "bookName": "西游记",
        "author": "吴承恩"
      }
    ]
  ]
}
```

删除数据库

Python

Python SDK 通过 drop_database() 接口删除数据库 **db-test**。

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

Java

Java SDK 通过 dropDatabase() 删除数据库 **db-test**。

```
client.dropDatabase("db-test");
```

Go

Go SDK 通过 DropDatabase() 删除数据库 **db-test**。

```
result, _ := client.DropDatabase(context.Background(), database)
```

Curl

HTTP 使用 /database/drop 接口删除数据库 **db_test**。如下示例，**url** 地址与 **api_key** 需要分别替换为已购买的实例的外网地址与 API Key。

```
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"
}'
```

可视化访问数据库

最近更新时间：2024-10-24 19:13:12

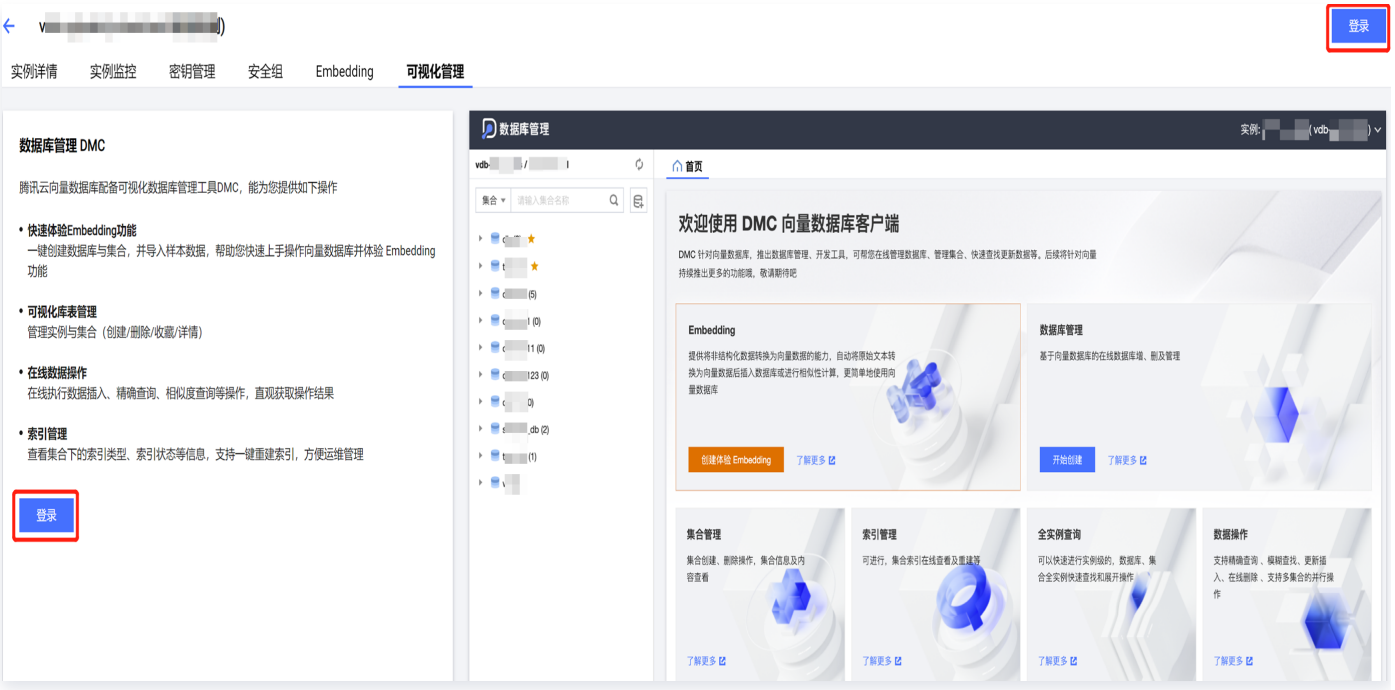
腾讯云向量数据库（Tencent Cloud VectorDB）支持通过 DMC（Database Management Center，DMC）可视化管理数据库。

说明：

- 2023年09月16日 00:00 之前创建的向量数据库实例，需 提交工单 将内核版本升级为最新版本，才可使用 DMC。
- 2023年09月16日 00:00 及之后创建的向量数据库实例，可直接使用 DMC。

登录 DMC

1. 使用腾讯云账号登录 向量数据库控制台 。
2. 在实例列表中，找到需登录 DMC 的实例。
3. 单击**实例 ID** 进入**实例详情**页面，切换到**可视化管理**页签。
4. 单击右上角的**登录**，或左下方的**登录**，如下图所示。



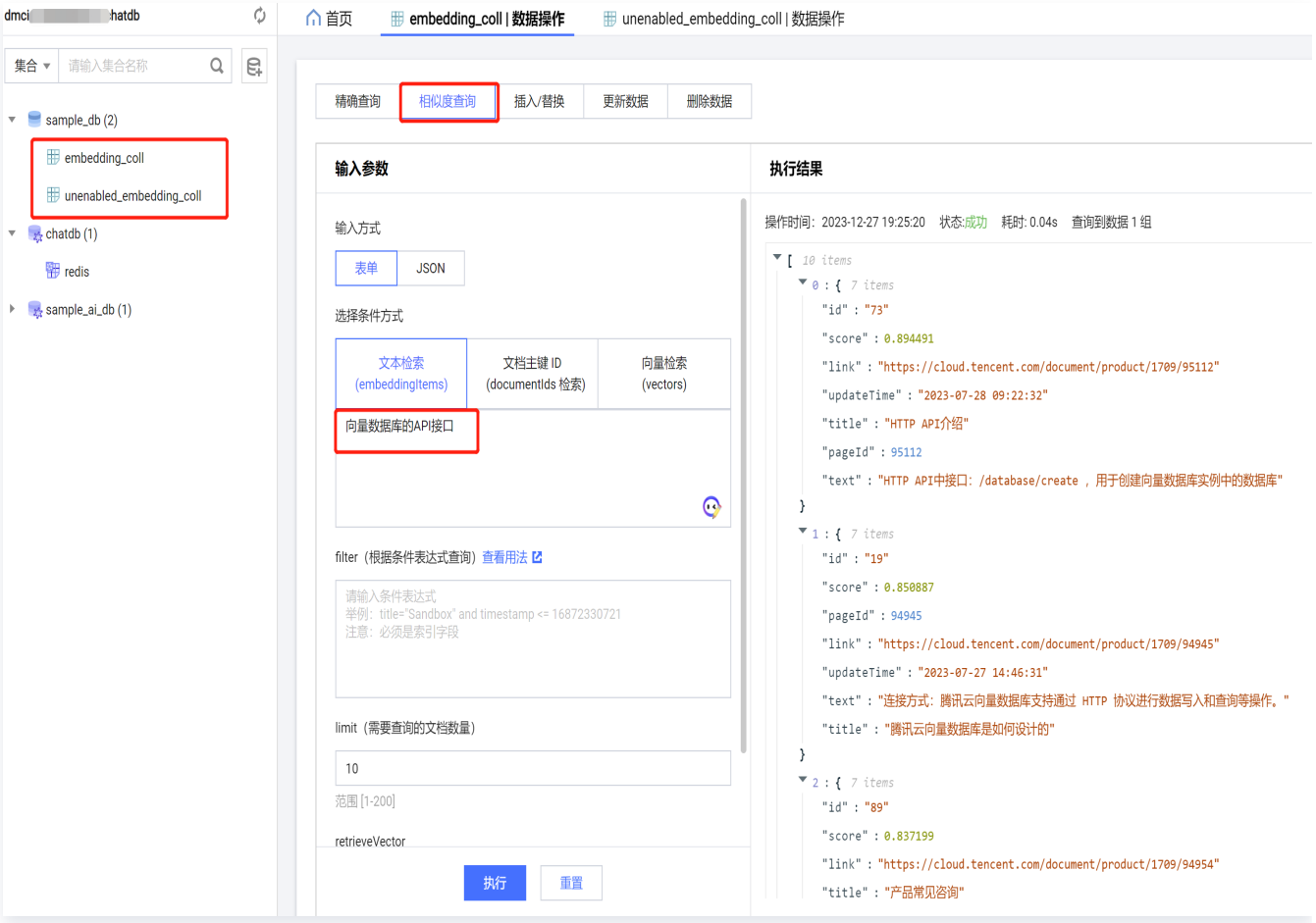
5. 跳转至数据库管理 DMC 登录界面，如下图所示，输入**账户**信息及其对应**密码**，单击**登录**。

   当前账号仅支持 root，对应密码可在**密钥管理**页面复制。具体操作，请参见 密钥管理 。

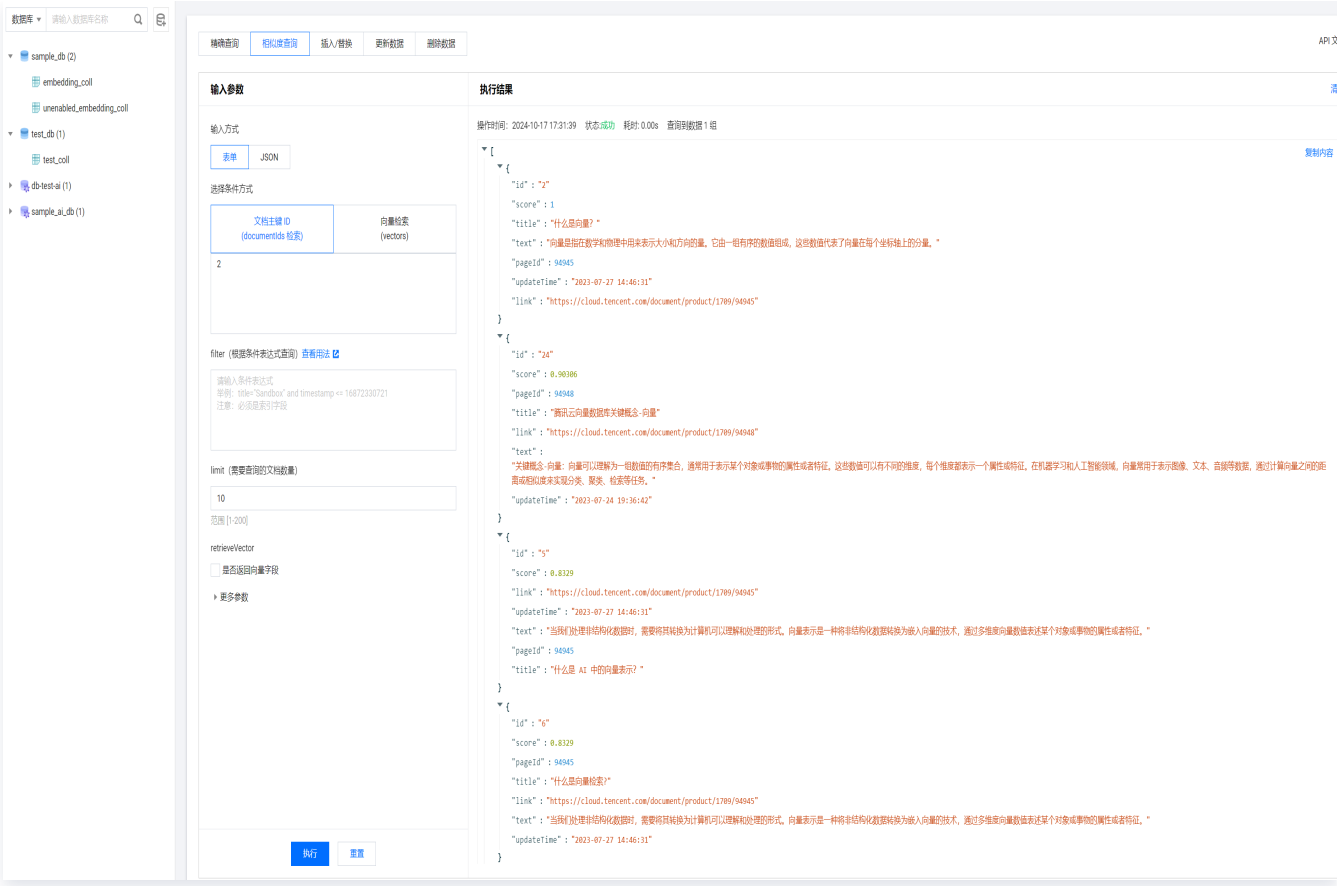
6. 进入向量数据库 DMC **首页**，如下图所示，单击**创建体验**。

7. 在**欢迎创建体验**的窗口，单击**立即创建**。系统会一键创建数据库及对应的集合，并导入样本数据。

8. 应用 DMC 进行相似性检索。如下所示，左侧导航栏为默认创建的数据库 **sample_db**，并创建两个集合 embedding_coll 与 unenabled_embedding_coll，分别选择对应的集合，单击精确查询页签下方的执行，可查看到样本数据。
   - embedding_coll：该集合应用 Embedding 功能可输入文本信息，检索相似性文本信息。单击**精确查询**页签下方的**执行**，可查看到样本数据。如下示例，以 `检索向量数据库的 API 接口` 为例，在**执行结果**区域可直观查看相似性检索得分与结果。



   - unenabled_embedding_coll：该集合为写入向量数据，检索与输入向量数据或输入文档 ID 进行相似性检索。



管理数据库

具体操作及其指引文档，请参见下表。

数据库操作	操作明细	参见文档
Database 级操作	新建、删除数据库	数据库管理
Collection 级操作	新建、删除集合	集合管理
Document 级操作	插入、更新、精确查询、相似度检索文档	数据操作
查询操作	查询数据库、查询集合配置信息	查询数据库与集合
Index 管理	重建索引	索引管理

版权所有：腾讯云计算（北京）有限责任公司