向量数据库

核心概念

腾讯云

文档目录

核心概念
数据类型

最近更新时间：2025-07-01 10:19:42

说明：

数据类型半精度浮点类型 float16_vector 与 bfloat16_vector 内存压缩特性说明：

- **理论压缩与实际限制：** 使用 float16_vector 或 bfloat16_vector 半精度浮点数理论上可将原始 float32 数据的内存占用降低 50%。然而，由于 HNSW 索引结构中的“边”（edges）需要额外的存储空间，实际内存压缩率通常无法达到理论上的50%。
- **维度对压缩率的影响：** 内存压缩效果会随着向量维度的增加而提升，并逐渐接近 50% 的理论值。维度越高，向量数据本身所占用的内存比例就越大。对高维数据进行半精度压缩带来的节省（接近50%）会显著超过存储索引“边”所需的固定或相对增长较慢的额外开销，使得“边”存储对整体压缩率的影响相对变小。
- **实测参考：** 基于真实数据集的验证结果，在1024维和2304维的数据上应用半精度量化后，其内存占用相对于原始 float32 格式通常能实现约 45% 左右的压缩。
- **检索质量影响（召回率）：** 根据多组公开数据集和业务数据集的测试验证，将数据转换为半精度（float16_vector / bfloat16_vector ）进行检索，其召回率（Recall）下降低于 0.5%。

数据类型	适用字段	存储格式	使用场景
vector	向量字段	单精度浮点数向量，采用8位指数位 + 23位尾数位 + 1位符号位（共32位）	适用于需最高精度的计算任务或对误差敏感的向量检索。
float16_vector		半精度浮点数向量，采用5位指数位 + 10位尾数位 + 1位符号位 的格式	适用于尾数精度较高（小数点后保留更多位数）的场景，如 bge-large 等模型生成的嵌入向量。
bfloat16_vector		bfloat16 浮点数向量，采用8位指数位 + 7位尾数位 + 1位符号位的格式	适用于向量数值范围较大（更多bit位表达整数）的场景。
binary_vector		每个维度仅用 1 个比特（bit）表示 0 或 1，无指数位、尾数位和符号位，直接存储二进制数据	适用于稀疏向量。
string	• 主键 ID • 标量字段	字符串	适用于文本类标签、分类名称、唯一标识符 ID、短描

			述、状态码、关键词等离散的、非数值型的分类信息。
uint64	标量字段	整型数值，包含正整数和零。	适用于需要大范围整数值的标量属性信息。
double		双精度浮点型数值	适用于需要高精度的连续数值或小数的属性信息，例如：价格、权重、分数等。
array		数组类型，存储一组相同类型的数据元素。当前，数组元素仅支持 string 类型。	适用于一个标量字段包含多个同类型的元素集合。
json		基于文本的格式，由键值对组成的数据对象。用花括号 {} 包裹，例如： "a": {"b": "test", "c": 12}，a 中的字段 b 用 a.b 表示。	存储结构灵活、嵌套复杂或动态变化的属性集合。

使用限制

最近更新时间：2025-05-16 17:21:32

资源命名长度限制

> **说明：**
> 计费实例不支持在同一个实例的不同 DB 下，创建相同的集合或集合视图名称。

参数	命名要求	长度要求
Database	只能使用英文字母，数字，下划线_、中划线-，并以英文字母开头。	[1,128]。
Collection/CollectionView		
Field	由用户自定义，不限制字符。	不限制字段长度。

资源数量限制

类别	数量
Instance	一个主 UIN 在一个地域最多购买实例数量为[1,10]。
Field	每个集合的标量字段数量最大为1024。

Collection 数量

> **说明：**
> 实例的 Collection 数量上限仅和单节点内存规格相关，增加节点数不会增加 Collection 上限。

单个节点内存规格（GB）	Collection 数量上限（包括 CollectionView）
1GB <= 内存规格 < 8GB	600
8GB <= 内存规格 < 32GB	1500
32GB <= 内存规格 <= 64GB	3000

Collection 资源

类别	最大限制数量
副本（replicas）	• 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]。 **说明：** 副本数+1，对应占用内存多一倍，默认推荐配置为不同实例规格的最低副本数。
分片（shard）	取值范围：[1,100] **说明：** HNSW 索引推荐单分片数据量在200-500w之内，且确保 (shard * (1 + replica)) % 实例节点数 = 0。例如实例节点数为6，是高可用三可用区版，集合存放数据量约为1kw，集合副本数为2，则 shard 数可以为2或者4。

Document 资源

类别	支持功能	长度限制最大值
ID	支持自定义、自动生成。	128

向量维度

类别	最大维度
Dimension	4,096

操作限制

类别	最大操作上限
Search	相似性检索按 ID 、Vectors 或者原始文本信息检索，每次最大可查询20条数据。
Query/Update/Delete	按 ID 检索每次最大可查询1000条数据。

索引类型

最近更新时间：2025-02-06 16:16:43

腾讯云向量数据库（Tencent Cloud VectorDB）在建表时，需指定不同字段的索引类型，数据存储时将会按照指定的索引类型进行索引；在检索时，便会依据索引并使用已选择的相似性计算方法进行匹配，快速高效地获取目标数据。腾讯云向量数据库（Tencent Cloud VectorDB）支持主键索引、向量索引和 Filter 索引，各自适用于不同的查询场景。

向量索引

向量索引（Vector Index）是基于某种数学模型，对向量数据构建一种时间和空间上更高效的数据结构，以便高效地查询与目标向量相似的若干个向量。当前向量化数据库支持 FLAT、HNSW、 IVF 系列向量索引方式。

> **说明：**
> Base 类数据库在创建 Collection 时便需要指定向量数据具体的索引方式，而 AI 类数据库在创建 CollectionVIew 时无需指定向量索引，系统自动构建。

向量索引类型

向量索引类型	索引说明	细分类别	细分类别说明
FLAT	FLAT 索引，向量数据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	HNSW（Hierarchical Navigable Small World Graphs）算法通过构建一个分层的图结构来组织和索引向量数据。图结构从顶层到底层，每一层的节点和边逐层密集。每个节点代表一个向量，而边则表示向量之间的相似性或接近度。 • 新插入节点，根据其特征和已有节点的分布被分配到不同的层，与一定数量的最相似的邻居建立连接。 • 在搜索时，查询向量从顶层开始，通过在顶层找到最接近的节点，快速跳过大量不相关的数据点，再以这个接近的节点作为新的起点，沿着路径向下移动到更密集的层级，	-	-

	每一层的检索都是在上一层筛选后的结果集上进行，直到在底层找到最接近的向量。		
IVF 系列	全称为 Inverted File，IVF 系列索引，是基于倒排索引和向量量化的高效近似最近邻搜索算法。其核心思想是将整个高维向量空间通过算法（如 k-means、DBSCAN 等）划分为多个预定义的聚类中心，并为每个聚类中心构建一个倒排文件，倒排索引表与这些聚类中心一一对应，每个表维护着一个向量列表。 • 在构建索引时，将数据集中的每个数据点分配到距离其最近的聚类中心所在的倒排索引表中。 • 当进行向量检索时，输入向量会在最相似的聚类中心进行检索，从而减少搜索空间，提高检索效率。	IVF_FLAT	基于倒排文件（inverted file）的索引结构，该索引在检索时，对每个聚类中的向量进行扁平扫描，计算它们与目标向量的距离，并选择最近的向量作为搜索结果。
		IVF_PQ	基于乘积量化（Product Quantization）的向量索引算法，通过将高维向量分割成多个子向量，并对每个子向量进行聚类，生成码本，然后将原始向量映射为一组离散的码本索引。
		IVF_SQ4, IVF_SQ8, IVF_SQ16	基于标量量化（Scalar Quantization）的索引方式，将原始的高维向量分割成多个较小的向量块，每个小向量块通过量化过程被转换为一个较低精度的表示形式，通常是4比特（即0.5字节）的整数。例如：4、8、16比特。随着量化精度的提高（从 SQ4 到 SQ16），索引的存储需求和搜索精度也随之增加，搜索速度可能会有所降低。
BIN_FLAT	用于处理二进制向量数据的索引类型。二进制向量通常表示数据点，每个向量由一系列二进制值（0或1）组成，BIN_FLAT 索引并不对二进制向量进行压缩，适用于图像二值数据或者其他二进制特征数据的处理场景。	-	-

配置索引参数

索引类型	参数	参数含义	参数设置
FLAT	-	-	-
HNSW	M	每个节点在检索构图中可以连接多少个邻居节	• 取值范围：[4,64]。

		点。	• M 值过大会影响写入效率。维度低于768，M 建议为16；维度高于768，M 建议为24~32。
	efConstruction	写入数据时，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。 • 取值范围：[8,512]， • 写入时，默认为200。若无特殊要求，可直接使用默认参数。
	ef	检索数据时，控制动态候选元素集合的大小。	• 取值范围[1,32768]，默认为10。 • ef 参数越大，遍历的元素越多，召回率越高，但是检索性能会下降。维度低于768，ef 值建议为200；维度高于768，ef 值建议为500。
IVF	nlist	聚类中心个数。	建议取值范围为：[sqrt(单分片数据量) * 4, sqrt(单分片数据量) * 16]。
	M	IVF_PQ 索引，构建数据表时，指定乘积量化中原始数据被拆分的子向量的数量。	每个子向量的维度为 D/M，其中 D 是原始向量的维度。M 必须能被 D （原始向量的维度）整除。
	nprobe	检索数据时，指定所需查询的单位数量。	取值范围[1,nlist]。

主键索引

为了确保数据库表中的每一行数据都可以被唯一地识别和访问，数据库中的每一条数据（通常称为记录或行）都需定义一个唯一的标识符，这个标识符被称为主键（Primary Key）。在整个数据表中，主键具有唯一性、不变性与非空性。主键索引（Primary Key Index）是基于主键来快速查找特定行数据的检索方式。

> **说明：**
> Base 类向量数据库默认将 Document id 作为主键来构建主键索引，而 AI 类数据库对应将 DocumentSet ID 作为主键构建索引。

Filter 索引

Filter 索引（Filter Index）是建立在标量字段的索引。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来过滤查询和范围查询来匹配相似向量。当前所支持的数据类型与对应的条件表达式，请参见 Filter 条件表达式 。

> **说明：**

> 在创建 Collection 或 CollectionView 时，Filter 索引指定标量字段支持 string、uint64、array 类型。插入数据时动态增加的标量字段暂不支持创建索引。

检索方式

最近更新时间：2024-12-17 11:54:52

检索方法	解释	功能明细
相似性检索	基于多维向量进行 相似性计算 的检索方式，通过计算查询向量与数据库存储的向量之间的相似度，找到与查询的多维向量最相似的文档。	• 支持指定多维向量数值，检索与指定的多维向量数值最相似的 Top K 条文档。 • 支持指定 id（Document ID），检索与该 id 的向量值最相似的 Top K 条文档。 • 支持输入原始文本，检索与该文本信息最相似的 Top K 条文档。 • 支持指定 id、多维向量或原始文本，搭配标量字段的 Filter 表达式一并检索与 id、多维向量、原始文本相似的文档。 • 支持批量进行相似度检索，即支持输入多条向量数据值、多个 id 分别检索与每一个向量数值、每一个 id 相似的 Top K 条数据。
精确查询	基于数据的标量字段（指一个单独的数值，例如文本、数值或日期等字段），精确检索具体数据的查询方式。例如，根据 id（Document ID）查询对应向量数据，或者根据文件名查询。适用于根据单个属性进行数据查询的场景。	• 支持根据指定的文件 id（DocumentSet ID）或文件名，检索对应的文件信息。 • 支持指定多个文件 id（DocumentSet ID）或文件名，单次批量查询对应的多个文件，最大数量为20个。 • 支持指定标量字段的 Filter 条件表达式 过滤文件信息。 • 支持指定查询起始位置和返回数量，查询该范围的文件。 • 支持指定多个文件 id（DocumentSet ID）或文件名，并搭配标量字段的 Filter 表达式一起过滤数据。
过滤检索	基于标量字段的 Filter 条件表达式 过滤需检索的信息，并对已过滤的数据进行向量相似性计算，获取与输入文本或向量数据最相似的数据。	• 在精确查询时，支持指定 Filter 字段的条件表达式过滤需查询的文档。 • 在相似度检索时，支持指定 Filter 字段的条件表达式过滤需检索的文档。 • 在删除数据时，支持指定 Filter 字段的条件表达式过滤需删除的文档。 • 在更新数据时，支持指定 Filter 字段的条件表达式过滤需更新的文档。

混合检索	支持稠密向量 Dense vector + 稀疏向量 Sparse vector 混合检索双路召回。Sparse vector 通过 BM25算法完成分数计算达到类关键字检索效果，进一步提升业务召回效果，弥补在语义检索中对具体数字、编码、数学公式等不敏感以及语义过度泛化的问题。	• 支持文本内容转化稀疏向量表示。 • 支持 RFF、Weight 两路权重合并排序。

Filter 表达式

最近更新时间：2025-07-01 10:19:42

条件表达式是一种用于筛选文档的查询条件。它可以根据文档中的字段值来进行筛选。通常，条件表达式由一个或多个条件组成，每个条件包括一个字段名、一个比较运算符和一个值。

> **说明：**
> 新增 IS NULL 与 IS NOT NULL 语法支持，覆盖所有主要标量索引类型（uint64, string, array, json）。对于 JSON 字段，使用 a.b 的形式进行单层属性路径的空值判断，满足常见的数据筛选需求，允许用户精确筛选出字段值为空 (NULL) 或非空（有具体值）的数据记录。

逻辑运算表达式

腾讯云向量数据库（Tencent Cloud VectorDB）所支持的逻辑运算符包括 and、or、not，如下表所示。

运算符	描述	示例
and	与	game_tag = "Robert" and (video_tag = "dance" or video_tag = "music")
or	或	game_tag = "1000" or video_tag = "dance"
not	非	game_tag = "1000" and not(video_tag = "dance")

字符串类型表达式

字符串类型的值必须要用英文双引号括起来。字符串类型可以用单值或多值匹配。腾讯云向量数据库当前所支持的运算符如下表所示。

运算符	描述	示例
in	匹配任意一个字符串值	game_tag in("Detective","Action Roguelike","Party-Based RPG","1980s")
not in	排除所有字符串值	game_tag not in ("Detective")
=	匹配单个字符串值	game_tag = "Detective"
!=	排除单个字符串值	game_tag != "Detective"

数值类型表达式

腾讯云向量数据库标量索引字段的数值类型当前仅支持 uint64，暂不支持负数和浮点型。如有需求，可以把负数和浮点型转换成整型。所支持的运算符，如下表所示。

运算符	描述	示例
>	大于	exipred_time > 1623388524
>=	大于等于	exipred_time >= 1623388524
=	等于	exipred_time = 1623388524
<	小于	exipred_time < 1623388524
<=	小于等于	exipred_time <= 1623388524
!=	不等于	exipred_time != 1623388524
in	匹配任意一个数值	exipred_time in (1, 2)
not in	排除所有字符串值	exipred_time not in (1, 2)

数组（Array）类型表达式

数据类型目前仅只是数组元素为 string 类型。具体运算符，请参见下表。

运算符	描述	示例
include	包含数组元素之一	name include (\"Bob\", \"Jack\")
exclude	排除数组元素之一	name exclude (\"Bob\", \"Jack\")
include all	全包含数组元素	name include all (\"Bob\", \"Jack\")

JSON 类型表达式

json 类型是一种由键值对组成的数据对象，使用花括号{}进行包裹。在 json 对象中，键（Key）和值（Value）通过冒号（:）连接。例如，json 类型的对象 a 的写入格式为 `"a": {"b": "test", "c": 12}` 。若访问 json 对象中的键，使用点（.）符号连接。例如，对象 a 中的字段 b 可以通过 a.b 表示。

> **说明：**
> json 字段的键（Key）命名规则：仅允许使用字母、数字及下划线（_）组成键名，且键名必须以字母或下划线开头，不得以数字开头。

json 类型字段在进行 Filter 检索时，支持的条件表达式语法和 json 字段的键值类型保持一致，支持的数据类型包括 string、uint64 和 array。若键值为 string 类型，Filter 表达式仅支持 string 类型的运算符。例如，写入格式为"a": {"b": "test", "c": 12}的文档，字段 a.b 为 string 类型，Filter 表达式仅支持 string 类型所允许的 in、

not in、=、!= 操作符，也只能与另一个 string 类型的字符串进行比较。类型 uint64，则仅支持数值类型表达式，也只能与另一个 uint64 类型的整数进行比较。

> **注意：**
> 向量数据库写入 json 类型时，不支持在 json 字段值中嵌套字典结构，也无法对嵌套结构内部的字段进行单独检索。例如，
> `“a': {b”: {c1": "test1", "c2": "test2"}}` 。其中，字段 b 嵌套了一层 Json 结构 {c1": "test1", "c2": "test2"}，在写入时将提示错误信息，写入失败。

数据类型	json 示例	Filter 表达式示例

string	{
	"a": {
	"b": "test",
	"c": 12
	}
	}
		• a.b != "demo"
		• a.b = "test"
		• a.b in ("test", "example", "hello")
		• a.b not in ("error", "invalid", "null")

uint64	{
	"data": {
	"id": 184467,
	"count": 100
	}
	}
		• data.id = 184467
		• data.count > 50
		• data.count in (100, 200, 300)

array	{
	"user": {
	"name": "Bob",
	"hobbies": ["gaming", "reading"],
	}
	}
		• user.name include (\"Bob\", \"Jack\")
		• user.hobbies include all (\"gaming\", \"reading\")

应用示例

本文给出过滤检索的请求示例，帮助您直观地了解过滤检索的方式。如下示例，使用 /document/search 接口，综合检索满足 `bookName` 字段条件表达式（ `"bookName in (\"三国演义\",\"西游记\")"` ），与指定向量 `"vectors": [[0.3123,0.43,0.213]]` 最相似的 Top3 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/search \
  -d '{
    "database": "db-test",
    "collection": "book-vector",
    "search": {
        "vectors": [
            [
                0.3123,
                0.43,
                0.213
            ]
        ],
        "params": {
            "ef": 200
        },
        "retrieveVector": true,
        "filter": "bookName in (\"三国演义\",\"西游记\")",
        "limit": 3
    }
}'
```

最终返回满足过滤条件最相近的文档数据，结果如下所示：

> **说明：**
> 每一个查询结果按照相似程度由高到低逐级排列，且均返回 Top K 条相似度计算的结果。其中，K 为 limit 设置的数值。如果检索的数据不足 K 条，则返回实际的 Document 数量。

```
{
  "code": 0,
  "msg": "Operation success",
  "documents": [
    [
      {
        "id": "0001",
        "vector": [
          0.21230000257492066,
          0.23000000417232514,
          0.21299999952316285
        ],
        "score": 0.97142,
        "bookName": "三国演义",
        "page": 21,
```

```
        "author": "罗贯中"
      },
      {
        "id": "0002",
        "vector": [
          0.21230000257492066,
          0.2199999988079071,
          0.21299999952316285
        ],
        "score": 0.96688,
        "bookName": "西游记",
        "author": "吴承恩",
        "page": 22
      }
    ]
  ]
}
```

稀疏向量
基本介绍

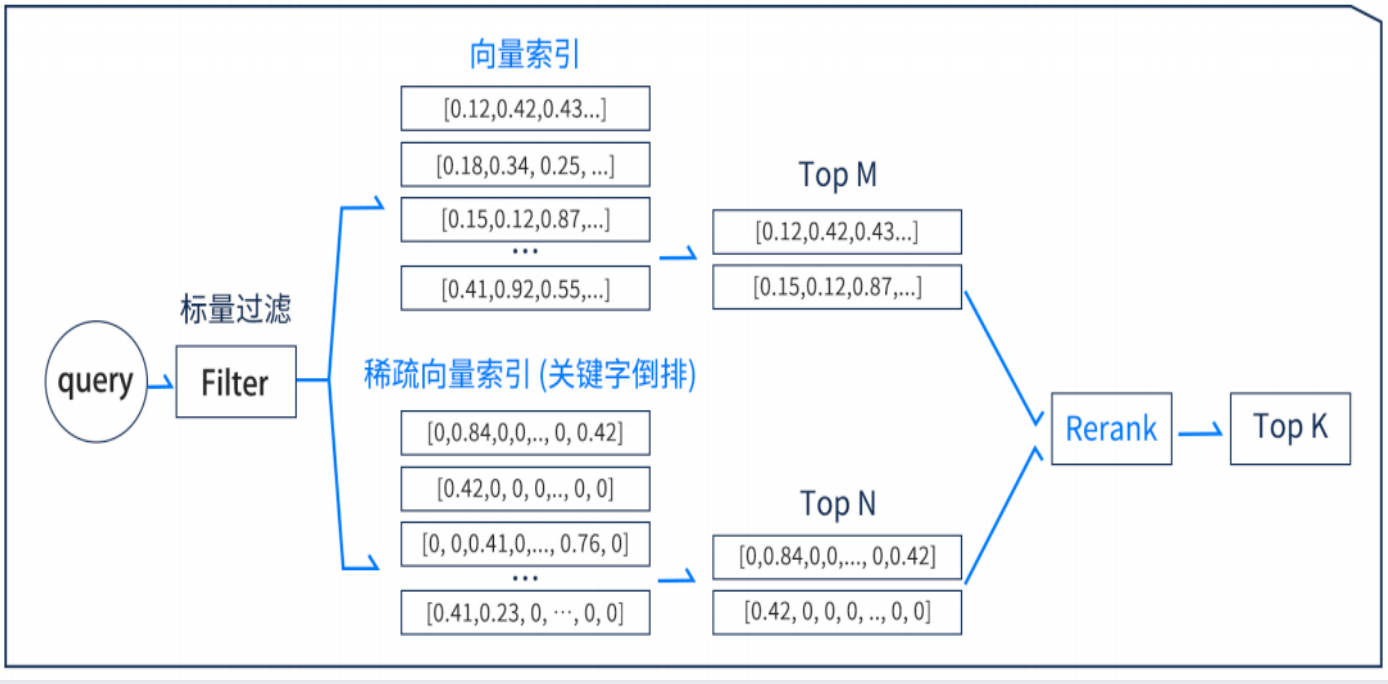
最近更新时间：2024-12-17 16:26:02

基本概念

稀疏向量是一种特殊的向量，它仅包含少数几个非零元素，其余元素为0。这种数据结构特性使得稀疏向量非常适合表示在大量数据点中只含有少量活跃值的场景，如文本数据中的词频分布。
关键字检索是通过将查询词与文档中的文本内容进行匹配，这通常是基于文本的直接查询，而不包含深层的语义分析。在关键字检索中，可以用稀疏向量来高效地表示和处理文档集合中的关键词信息，稀疏向量中的每个非空值，都可以对应一个词语的权重分数。在实际的 AI 应用及搜索推荐等业务场景中，尤其是在文本检索的场景中，混合使用语义检索（稠密向量）和关键字检索（稀疏向量）可以进一步改善检索效果，弥补在语义检索中对具体数字、编码、数学公式等不敏感以及语义过度泛化的问题。

实现机制

在文本写入时，每个文档被分解成单独的词汇，通过哈希函数将这些词汇映射到特定的索引值。基于索引值，通过 BM25 算法计算每个词的词频以及与原文档的相关性，形成稀疏向量存储于数据库。在文本检索时，检索的文本信息同样被分解成单独的词汇，通过哈希函数将这些词汇映射到特定索引，得到查询文本的词向列表，通过 BM25算法计算每一个词汇对应的稀疏向量，再与原始数据库的相似性计算得分比较，按得分排序，返回检索结果。
在进行稠密向量和稀疏向量的混合检索时，同一段文本内容，可以通过 Embedding 模型生成稠密向量，也可以通过 tcvdb-text 等工具生成对应的稀疏向量。这两个向量分别表示了语义级别和词级别的信息，共同存储在同一个集合内。在进行混合检索时，两路向量会分别检索 Top M 和 Top N 条相似数据，随后通过加权、RRF 等 Rerank 方法，得到综合两路信息的排序结果，最终返回排序最靠前的Top K条数据。



功能与限制

功能	功能描述	使用限制
混合检索	1. 创建集合时，指定稀疏向量字段与索引方式。 2. 通过 tcvdb-text 传入文本生成稀疏向量，并写入。 3. 使用稠密向量 + 稀疏向量混合检索方法，支持加权（Weighted）、RRF 等排序，对两路召回的结果进行排序。	• 每个集合仅支持定义1个稀疏向量字段，每条稀疏向量最多支持1024个非空值。 • 稀疏向量仅支持 IP 相似度计算方法。
稀疏向量生成工具	**tcvdb-text** 是由腾讯云向量数据库团队提供的一款稀疏向量工具包。它旨在帮助用户将文本内容高效生成稀疏向量。具体信息，请参见 tcvdb-text 稀疏向量生成工具 。	当前支持**中文**、**英文**两种文本语言，默认为中文。

快速开始

最近更新时间：2025-04-01 11:55:01

本章节以 Python 语言的使用方式为例，主要介绍如何生成稀疏向量，并写入向量数据库进行混合检索。

> **说明：**
> 如需使用稀疏向量，至少需要升级 Python SDK 至 1.4.5 版本，使用 Embedding 文本与稀疏向量混合检索则需升级至 1.5.0或更高版本。

步骤1：安装 SDK

```
## 安装稀疏向量工具包
pip3 install tcvdb-text
## 安装腾讯云向量数据库 Python SDK
pip3 install tcvectordb==1.4.4
```

步骤2：导入依赖

```
import tcvectordb
from tcvectordb.model.document import AnnSearch, WeightedRerank,
RRFRerank, KeywordSearch, SearchParams
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams, SparseVector, SparseIndex
from tcvdb_text.encoder import BM25Encoder
from typing import List
```

步骤3：创建客户端

创建一个客户端对象，并创建 SparseVectorEncoder。

```
vdb_url = 'YOUR CONNECTION URL'
vdb_key = 'YOUR CONNECTION KEY'

client = tcvectordb.RPCVectorDBClient(
    url=vdb_url,
    key=vdb_key,
    username='root',
```

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

步骤4：创建 Database

```
db_name ='db-test-sparse-vec'
db = client.create_database(db_name)
```

步骤5：创建包含稀疏向量的 Collection

1. 定义索引结构。

> **说明：**
> - 每个集合仅支持定义1个稀疏向量字段，每条稀疏向量最多支持1024个非空值。
> - 目前稀疏向量的字段名固定为“sparse_vector”，暂不支持自定义。
> - 稀疏向量字段无需指定数据维度，会根据写入的数据自动判断。

```
# 定义集合的索引结构，包括稠密向量索引、稀疏向量索引
index = Index()
index.add(FilterIndex('id', FieldType.String, IndexType.PRIMARY_KEY))
index.add(VectorIndex(name='vector',
                      dimension=3,
                      index_type=IndexType.HNSW,
                      metric_type=MetricType.IP,
                      params=HNSWParams(m=16, efconstruction=200)))
index.add(SparseIndex(name='sparse_vector',
                      field_type=FieldType.SparseVector,
                      index_type=IndexType.SPARSE_INVERTED,
                      metric_type=MetricType.IP))
```

2. 创建集合。

```
# 创建 Collection
coll_name = 'coll-sparse-vec'
res = db.create_collection(
    name=coll_name,
    shard=1,
    replicas=1,
    description='test collection',
    index=index
```

```
)
```

步骤6：插入带有 Sparse Vector 的数据

> **说明：**
> 腾讯云向量数据库提供了的 tcvdb-text 工具包，可以基于原始文本快速生成稀疏向量。若业务需要自己生成稀疏向量，也可以直接写入生成好的稀疏向量数值。

直接写入稀疏向量

```
client.upsert(
    database_name=db_name,
    collection_name=coll_name,
    documents=[{
        "id": "0001",
        "vector": [0.21241, -0.9182, 0.8769],
        "sparse_vector": [[2502995674, 0.7129621405718145],
[1574737055, 0.7129621405718145], [1169440797, 0.8324318718845897]]
    }]
)
```

使用 tcvdb-text 工具包写入稀疏向量

调用 encode_texts 接口将原始文本转化为稀疏向量，并通过 upsert 接口写入数据库。

```
# 初始化稀疏向量 Encoder
bm25 = BM25Encoder.default('zh')

# 根据文本内容，生成对应的稀疏向量
texts = ['腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级
分布式数据库服务',
         '腾讯云向量数据库可以和大语言模型 LLM 配合使用']
sparse_vectors: List[SparseVector] = bm25.encode_texts(texts)

# 写入包含稀疏向量的 Document 数据
client.upsert(
    database_name=db_name,
```

```
    collection_name=coll_name,
    documents=[
        {
            "id": "0000",
            "vector": [0.1273, 0.0871, -0.6573],
            "sparse_vector": sparse_vectors[0]
        },
        {
            "id": "0001",
            "vector": [0.9172, 0.7612, 0.5523],
            "sparse_vector": sparse_vectors[1]
        }
    ]
)
```

步骤7：稠密向量 + 稀疏向量混合检索

调用 Python SDK 接口 hybrid_search 进行混合检索。

- **ann 参数**：指定稠密向量相似性检索的相关规则。
- **match 参数**：指定稀疏向量关键字检索的相关规则。其中，用于检索的稀疏向量可以调用腾讯云向量数据库提供的 tcvdb-text 工具生成，也可以直接传入用户自己生成的稀疏向量数值。
- **rerank 参数**：指定重排序的规则及参数。

> **说明：**
> hybrid_search 接口暂不支持批量检索操作。

直接传入稀疏向量数值进行混合检索

```
# 执行混合检索，并使用指定权重（Weighted）的Rerank方法
doc_lists = client.hybrid_search(
    database_name=db_name,
    collection_name=coll_name,
    ann=[
        AnnSearch(
            field_name="vector",
            data=[0.3123, 0.43, 0.213],
        ),
    ],
```

```
    match=[
        KeywordSearch(
            field_name="sparse_vector",
            data=bm25.encode_queries('向量数据库'),
            terminate_after=4000,
            cutoff_frequency=0.1,
        ),
    ],
    rerank=WeightedRerank(
        field_list=['vector', 'sparse_vector'],
        weight=[0.9, 0.1],
    ),
    retrieve_vector=False,
    limit=1,
)
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)

# 执行混合检索，并使用RRF的Rerank方法
doc_lists = client.hybrid_search(
    database_name=db_name,
    collection_name=coll_name,
    ann=[
        AnnSearch(
            field_name="vector",
            data=[0.3123, 0.43, 0.213],
        ),
    ],
    match=[
        KeywordSearch(
            field_name="sparse_vector",
            data=bm25.encode_queries('向量数据库'),
        ),
    ],
    rerank=RRFRerank(k=60),
    retrieve_vector=False,
    limit=1,
)
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
```

```
        print(doc)
```

使用 tcvdb-text 工具包生成稀疏向量并进行混合检索

请注意在进行混合检索时，需要使用 encode_quires 函数生成文本对应的稀疏向量，这与写入数据时使用的 encode_texts 函数不同。

```
# 执行混合检索，并使用指定权重（Weighted）的Rerank方法
doc_lists = client.hybrid_search(
    database_name=db_name,
    collection_name=coll_name,
    ann=[
        AnnSearch(
            field_name="vector",
            data=[0.3123, 0.43, 0.213],
        ),
    ],
    match=[
        KeywordSearch(
            field_name="sparse_vector",
            data=bm25.encode_quires('向量数据库'),
            terminate_after=4000,
            cutoff_frequency=0.1,
        ),
    ],
    rerank=WeightedRerank(
        field_list=['vector', 'sparse_vector'],
        weight=[0.9, 0.1],
    ),
    retrieve_vector=False,
    limit=1,
)
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)

# 执行混合检索，并使用RRF的Rerank方法
doc_lists = client.hybrid_search(
    database_name=db_name,
```

```
    collection_name=coll_name,
    ann=[
        AnnSearch(
            field_name="vector",
            data=[0.3123, 0.43, 0.213],
        ),
    ],
    match=[
        KeywordSearch(
            field_name="sparse_vector",
            data=bm25.encode_quires('向量数据库'),
        ),
    ],
    rerank=RRFRerank(k=60),
    retrieve_vector=False,
    limit=1,
)
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)
```

执行上述混合检索后，以 RRF Rerank 为例，返回结果如下：

```
{'id': '0001', 'score': 0.032786883413791656}
```

稀疏向量工具
工具介绍

最近更新时间：2025-02-21 11:07:12

腾讯云向量数据库团队推出了一款稀疏向量工具包，旨在帮助用户高效生成稀疏向量。该工具包集成“jieba”分词库，并提供了一系列高级且灵活的功能，能够满足用户在不同领域和任务中的个性化需求。

- **快速生成稀疏向量**：具有优化的算法和数据结构，并区分写入和检索场景，使用不同的词表拆分计算方法，将文本内容快速转换为稀疏向量表示。当前支持**中文**、**英文**两种语言，默认为中文。
- **训练自定义语料**：支持针对特定领域的数据集进行模型训练，生成适配特定领域的词频计算参数，并可下载与上传参数，以灵活调整优化参数，持续提高模型稀疏向量生成的准确性。
- **SDK**：为了帮助用户快速生成稀疏向量，腾讯云向量数据库提供了 Python、Java、Go 三种语言的稀疏向量生成工具。

> **说明：**
> 在使用不同语言的工具生成稀疏向量时，由于底层使用的分词工具存在差异，因此在支持的拆分参数以及拆分结果上可能会存在细微差别，如 Python SDK 支持指定使用 PaddlePaddle 的分词库，其他语言暂不支持。

语言	语言版本	SDK 下载	SDK 源码

Python	推荐使用3.8及以上版本	• 安装最新版本 tcvectordb SDK，已包含 tcvdb-text 工具，无需单独下载。
		`pip3 install tcvectordb`
		• 执行如下命令，可单独安装 tcvdb_text 最新版本。
		`pip3 install tcvdb-text`
			vectordatabase-sdk-python

Java	Java 8或更高版本	安装最新版本的 vectordatabase-sdk-java，已依赖tcvdb_text 工具，无需单独下载。如需单独依赖，请参见tcvdb-text 。	vectordatabase-sdk-java
GO	Go 1.17或更高版本	安装最新版本（V1.4.7及以上版本支持稀疏向量）的 vectordatabase-sdk-go，已包含 tcvdbtext 工具。 > **说明：**	vectordatabase-sdk-go

		使用 Go SDK 通过 Bm25 tcvdbtext 将文本转化成稀疏向量的过程中，英文文本统一会转换成小写字母处理。	

使用工具混合检索 SDK Demo

最近更新时间：2024-10-21 11:24:01

本章节提供在腾讯云向量数据库中，使用稀疏向量生成工具进行混合检索的 Demo。

Python

```
import tcvectordb
from tcvectordb.model.document import AnnSearch, WeightedRerank,
RRFRerank, KeywordSearch
from tcvectordb.model.enum import FieldType, IndexType, MetricType,
ReadConsistency
from tcvectordb.model.index import Index, VectorIndex, FilterIndex,
HNSWParams, SparseVector, SparseIndex
from tcvdb_text.encoder import BM25Encoder
from typing import List


vdb_url = 'YOUR CONNECTION URL'
vdb_key = 'YOUR CONNECTION KEY'

client = tcvectordb.RPCVectorDBClient(
    url=vdb_url,
    key=vdb_key,
    username='root',
    read_consistency=ReadConsistency.EVENTUAL_CONSISTENCY,
    timeout=30)

db_name = 'db-test-sparse-vec'
client.drop_database(db_name)
db = client.create_database(db_name)

# 定义集合的索引结构，包括稠密向量索引、稀疏向量索引
index = Index()
index.add(FilterIndex('id', FieldType.String,
IndexType.PRIMARY_KEY))
index.add(VectorIndex(name='vector',
                      dimension=3,
                      index_type=IndexType.HNSW,
                      metric_type=MetricType.IP,
```

```
                        params=HNSWParams(m=16, efconstruction=200)))
index.add(SparseIndex(name='sparse_vector',
                        field_type=FieldType.SparseVector,
                        index_type=IndexType.SPARSE_INVERTED,
                        metric_type=MetricType.IP))

# 创建 Collection
coll_name = 'coll-sparse-vec'
res = db.create_collection(
    name=coll_name,
    shard=1,
    replicas=1,
    description='test collection',
    index=index
)

# 初始化稀疏向量 Encoder
bm25 = BM25Encoder.default('zh')

# 写入数据将文本转为稀疏向量：bm25.encode_texts(texts)
texts = ['腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级
分布式数据库服务',
         '腾讯云向量数据库可以和大语言模型 LLM 配合使用']
sparse_vectors: List[SparseVector] = bm25.encode_texts(texts)

# 写入包含稀疏向量的 Document 数据
client.upsert(
    database_name=db_name,
    collection_name=coll_name,
    documents=[
        {
            "id": "0000",
            "vector": [0.1273, 0.0871, -0.6573],
            "sparse_vector": sparse_vectors[0]
        },
        {
            "id": "0001",
            "vector": [0.9172, 0.7612, 0.5523],
            "sparse_vector": sparse_vectors[1]
        }
    ]
)
```

```
# 执行混合检索，并使用指定权重（Weighted）的Rerank方法
doc_lists = client.hybrid_search(
    database_name=db_name,
    collection_name=coll_name,
    ann=[
        AnnSearch(
            field_name="vector",
            data=[0.3123, 0.43, 0.213],
        ),
    ],
    match=[
        KeywordSearch(
            field_name="sparse_vector",
            # search前将查询文本转稀疏向量：bm25.encode_queries(texts)
            data=bm25.encode_queries('向量数据库'),
        ),
    ],
    rerank=WeightedRerank(
        field_list=['vector', 'sparse_vector'],
        weight=[0.9, 0.1],
    ),
    retrieve_vector=False,
    limit=1,
)
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)

# 执行混合检索，并使用RRF的Rerank方法
doc_lists = client.hybrid_search(
    database_name=db_name,
    collection_name=coll_name,
    ann=[
        AnnSearch(
            field_name="vector",
            data=[0.3123, 0.43, 0.213],
        ),
    ],
    match=[
        KeywordSearch(
            field_name="sparse_vector",
            data=bm25.encode_queries('向量数据库'),
```

```
        ),
    ],
    rerank=RRFRerank(k=60),
    retrieve_vector=False,
    limit=1,
)
for i, docs in enumerate(doc_lists):
    print(i)
    for doc in docs:
        print(doc)
```

Java

```
/*
 * 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.examples;

import com.tencent.tcvectordb.client.VectorDBClient;
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.dml.*;
import com.tencent.tcvectordb.model.param.entity.AffectRes;
import com.tencent.tcvdbtext.encoder.SparseVectorBm25Encoder;
import org.apache.commons.lang3.tuple.Pair;

import java.util.*;

import static
com.tencent.tcvectordb.model.param.enums.EmbeddingModelEnum.BGE_BASE
_ZH;

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

    private static final String DBNAME = "book_5";
    private static final String COLL_NAME =
"book_segments_sparse_4";
    private static final String COLL_NAME_ALIAS =
"collection_alias_sparse_4";

    public static void main(String[] args) throws
InterruptedException {
        // 创建VectorDB Client
        VectorDBClient client = CommonService.initClient();

        // 清理环境
        CommonService.anySafe(() -> client.dropDatabase(DBNAME));
```

```
        // 测试
        createDatabaseAndCollection(client);
        upsertData(client);
        queryData(client);
        updateAndDelete(client);
        deleteAndDrop(client);
        testFilter();
    }


    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);
        }
//         Database db = client.database(DBNAME);

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

        // 4. 列出所有 collection
//         Database db = client.database(DBNAME);
        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 List<Double> generateRandomVector(int dim){
        Random random = new Random();
        List<Double> vectors = new ArrayList<>();

        for (int i = 0; i < dim; i++) {
            double randomDouble = 0 + random.nextDouble() * (1.0 -
0.0);
            vectors.add(randomDouble);
        }
        return vectors;
    }

    private static void upsertData(VectorDBClient client) throws
InterruptedException {
        Database database = client.database(DBNAME);
```

```
        Collection collection =
database.describeCollection(COLL_NAME);
        SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
        List<String> texts = Arrays.asList(
                "富贵功名，前缘分定，为人切莫欺心。",
                "正大光明，忠良善果弥深。些些狂妄天加谴，眼前不遇待时临。",
                "细作探知这个消息，飞报吕布。",
                "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。",
                "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。"
        );

        List<List<Pair<Long, Float>>> sparseVectors =
encoder.encodeTexts(texts);
        List<Document> documentList = new ArrayList<>(Arrays.asList(
                Document.newBuilder()
                        .withId("0001")
                        .withVector(generateRandomVector(768))
                        .withSparseVector(sparseVectors.get(0))
                        .addDocField(new DocField("bookName", "三国演
义"))
                        .addDocField(new DocField("author", "罗贯
中"))
                        .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")
                        .withVector(generateRandomVector(768))
                        .withSparseVector(sparseVectors.get(1))
                        .addDocField(new DocField("bookName", "三国演
义"))
                        .addDocField(new DocField("author", "罗贯
中"))
                        .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")
                        .withVector(generateRandomVector(768))
                        .withSparseVector(sparseVectors.get(2))
                        .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")
                        .withVector(generateRandomVector(768))
                        .withSparseVector(sparseVectors.get(3))
                        .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")
                        .withVector(generateRandomVector(768))
                        .withSparseVector(sparseVectors.get(4))
                        .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 * 3);
    }

    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",
"segment");
        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());
        }

        // search稀疏向量搜索和向量搜索混合
//         List<Double> vector =
Arrays.asList(0.011228000745177269,-0.01778145506978035,-0.000842009
5546171069,0.058591078966856,0.025626985356211662,-0.009485375136137
009,0.0044272118248045444,0.03963795304298401,-0.07349739968776703,0
.033373408019542694,-0.017562853172421455,-0.06693584471940994,-0.00
08283713832497597,-0.09282511472702026,0.019856665283441544,0.002936
9608964771032,0.025461247190833092,0.043309904634952545,-0.001094068
5169771314,-0.02834116853773594,0.024333607405424118,-0.063586719334
12552,0.0004886172828264534,-0.003996695391833782,-0.005141071975231
171,0.031137600541114807,-0.03720816969871521,0.03373042494058609,-0
.005284181796014309,0.06811655312776566,0.024133838713169098,-0.0082
62764662504196,-0.02401343360543251,0.01113040093332529,-0.019650375
470519066,0.024059455841779 71,0.008977336809039116,-0.02221423573791
9807,0.010524755343794823,-0.06512012332677841,-0.007044251542538404
5,0.039866313338279724,0.00365936080925 1666,-0.014790602959692478,0.
03323712199926376,0.030882341787219048,0.012858539819717407,0.036260
66818833351,-0.02777714841067791,-0.0006111871916800737,0.0119515396
65460587,0.21934707462787628,-0.009810359217226505,-0.006874813232570
887,0.07120531797409058,0.028546100482344627,0.01646684668958187,0.
0647200047969818 1,-0.024948634207248688,0.02457079477608204,0.015692
222863435745,0.015110153704881668,0.04548252373933792,-0.03442720696
3300705,0.0035766353830695152,-0.006357731763273 4776,-0.036407884210
34813,-0.03309265896677971,0.016592761501669884,-0.0184243842959404,
0.025951147079467773,0.0009172717691399157,-0.020834090188145638,-0.
029650429263710976,-0.00929233431816101,-0.02037796750664711,0.00540
0816909968853,-0.003652009414508939,0.0004404079227242619,0.00437577
0688056946,0.04158478602766991,0.002212094841524 9586,0.023043958470 2
25334,0.02082330361008644,-0.0348212383687 4962,-0.03934218361973762 5
,-0.022602153941988945,0.006933159194886684,0.020710067823529243,0.0
```

2493712119758129,0.008838232606649399,-0.01082011591643095,-0.047118
29870939255,0.015358409844338894,-0.05259833112359047,0.035649448633
19397,-0.02094186656177044,-0.014302685856819153,-0.0009043896570801
735,0.060996394604444504,0.012411488220095634,-0.014214642345905304,
-0.026238352060317993,-0.033097539097070694,0.019276197999715805,0.0
1992779212594032,-0.0897335335612297,0.04069192707538605,0.01696757
2271823883,0.012736618518829346,0.007248705718666315,-0.042650915682
315826,-0.012213833630084991,0.029378674924373627,-0.011317588388919
83,-0.017445610836148262,0.01528218761086464,-0.0010133420582860708,
0.008936449885368347,-0.04863755777478218,0.006830527447164059,-0.00
06641799700446427,-0.03017701581120491,0.0045319232158362865,-0.0256
10102340579033,-0.024471720680594444,-0.028333375230431557,0.0400088
3921980858,0.007346018683165312,0.013694453984498978,-0.010897736065
089703,-0.009626672603189945,-0.05798979103565216,0.014471616595983505
05,0.005314654670655727,-0.027487266808748245,-0.020013470202684402,
-0.023798091337084 77,-0.019267942756414413,-7.192990597104654e-
05,0.03751828148961067,0.051989246159791946,0.022071769461035 73,0.01
5709249302744865,0.04722846299409866,-0.009285308420658112,0.0481977
53727436066,0.0047728074714541435,0.03828258439898491,0.011275202967
226505,0.03798242658376694,0.052456334233284,-0.02123964950442314,0.
10464958846569061,-0.00095760339172557,0.005099969916045666,-0.01903
274841606617,0.033146798610687256,-0.001944964868016541,0.0354474745
6908226,0.05200682580471039,0.004140638280659914,-0.0012725018896162
51,-0.029995812103152275,-0.036847393959760666,0.07203970104455948,-
0.03457489609718323,0.013243802823126316,-0.0024343973491340876,0.04
353635758161545,0.011767414398491383,-0.03771014139056206,-0.0338407
7921509743,-0.06499378383159637,-0.08102220296859741,0.0684260800480
8426,-0.007303804624825716,0.007441019639372826,0.00384475733153522,
-0.015521292574703693,0.001064897864125669,-0.017786353826522827,0.0
228169746696949,-0.001740469015203 4163,-0.08029986172914505,-0.02340
6831547617912,-0.020283214747905 73,-0.03325213864445686,0.0116588398
81420135,-0.02427353709936142,0.02202378585934639,0.0129101071506738
66,-0.028136733919382095,0.04223033040761947 6,0.04168698564171791,0.
015558804385364056,-0.06827197968959808,0.03491898998618126,-0.04686
718061566353,0.027400435879826546,0.009755358099937439,0.02574536204
3380737,-0.009593848139047623,0.08101218193769455,-0.023704754188656
807,0.02434670366346836,0.021201400086283684,-0.02352142333984375,0.
0294501408934 5932,-0.006461021490395069,0.002558466047048 5687,-0.045
635148882865906,0.011546051129698753,-0.03846436366438866,0.00266063
1900653243,0.011385934427380562,-0.05482086911797 5235,-0.02753876522
1834183,-0.012130278162658215,0.02718573808670044,0.0575387924909591
7,-0.024369744583964348,0.013582611456513405,-0.08181896805763245,-0
.04349633306264877,0.03299983590841293,-0.004695754963904619,-0.0169

2030020058155,-0.011063349433243275,-0.0001466975954826921 2,0.014689693227410316,-0.038261156529188156,-0.015512768179178238,-0.004468604456633329,-0.048229288309812546,0.006024762522429228,-0.049573648720979 69,0.00813916977494955,-0.026627950370311737,-0.009431248530745506,-0.002486441750079393 4,0.017048964276909828,-0.02847607247531414,-0.020705807954072952,0.00494916038587689 4,-0.015850678086280823,0.06021637469530105 6,-0.015156073495745659,-0.007588365115225315,-0.03735107928514480 6,0.042187843471765 52,0.020519305020570755,-0.001068396260961890 2,-0.017848571762442 59,-0.001070767524652183,-0.068963371217250 82,-0.013779735192656517,0.021526616066694 26,-0.042298782616853714,-0.09644242376089096,-0.043055132031440735,-0.013538303785026073,0.04751390591263771,0.057626064866781235,-0.009500116109848022,0.00708649680018425,-0.024209508672356606,-0.08286508917808533,-0.007599340286105871,0.032298244535923004,0.06640364229679108,0.024041226133704185,0.004293238278478384,0.020563913509249687,-0.022327011451125145,0.053777556866407394,-0.018423566594719887,-0.035526152700185776,-0.05626888573169708,0.016877761110663414,0.006738790310919285,0.06473775207996368,0.004535271320492029,0.04972827062010765,-0.07362700998783112,-0.003007926512509584 4,0.059974104166030884,-0.013675615191459656,-0.034515395760536194,0.014622291550040245,-0.03385680168867111,-0.025142354890704155,0.0254383347928524,0.001493200776167214,0.334794282913208,0.02830587700009346,-0.07777972519397736,0.043088871985673904,-0.01293167844414711,-0.005174708086997271,-0.005819339305116243,-0.07398725301027298,0.017567740753293037,0.022265968844294548,0.11928874999284744,-0.055033378303050995,0.02395421266555786,-0.011595740914344788,0.02233700640499592,-0.018687181174755096,-0.06958524882793427,0.0009067300125025213,-0.004662649706006 05,-0.039599407466426582,0.025952531024813652,0.019410174340009 69,-0.03903195634484291,-0.002233545063063502 3,0.067644387483596 8,0.029737481847405434,-0.015014370903372765,0.007381070405244827,-0.048463366925716 4,0.023242061957716942,-0.034604370594024 66,-0.013248912990093231,-0.012566791847348213,-0.023838806897401 81,0.014195072464644909,0.009522831067442894,0.05593474581837654,-0.048447150737047195,0.024550078436732292,-0.019980145618319 51,0.004192877560853958,-0.000720351177733391 5,-0.06000881642103195,0.007732178084552288,0.003466855501756072,-0.03827047348022461,0.024848824366927147,0.013989027589559555,0.017047369852662086,0.024627817794680595,0.019813280552625656,0.010560467839241028,-0.007128795608878136,0.021267788484692574,-0.050384741276502 61,0.011191616176974773 4,0.015836659818887 71,-0.071007378399372 1,0.068022951483726 5,0.015228376723825932,0.012478475458920002,-0.000791455560829490 4,0.045038223266601 56,-0.022816233336925507,-0.029615147039294243,-0.041380070149898 53,0.015486898832023144,-0.033105045557022095,-0.027106864377856255,-0.018030496314167976,0.026013921946287155,0.005

876969546079636,0.0209262426942586 9,0.05155663192272186,0.0006067254
580557346,-0.060211531817913055,0.059832267463207245,0.0092186639085
41203,0.0034593825694173574,0.00601182272657752,0.0525486096739769,-
0.017541436478495598,-0.016692979261279106,-0.012630617246031761,0.0
1190961059182 8823,0.027539147064089775,0.030525386333465576,-0.05234
1122180223465,-0.04240258410573006,-0.015112871304154396,-0.01146265
3055787086,0.016128763556480408,0.01829593814909458,-0.0090163089334
96475,-0.04230230674147606,0.007138686720281839,0.011453477665781975
,-0.025860127061605453,0.023999856784939766,0.020570704713463783,-0.
028624074533581734,0.01875831000506878,0.0018658454064279795,0.02916
049025952816,-0.057269223034381866,0.0036049848422408104,0.010065961
629152298,0.007160463370382786,-0.056595977395772934,-0.033417794853
44887,-0.053864892572164536,-0.006826115772128105,-0.032975882291793
82,0.01128164678812027,0.022906454280018806,-0.0031081661581993103,0
.0009896974079310894,-0.042085934430360794,-0.05885877087712288,0.05
95456063747406,0.010770737193524837,-0.021534759551286697,-0.0332218
0360555649,-0.04332531988620758,-0.03466140478849411,0.0221787840127
94495,-0.01332541648298502,-0.05094131454825401,0.04912945628166199,
0.024204950779676437,-0.01659112051129341,-0.042198680341243744,0.05
102888867259 0256,0.02925165370106697,-0.017100905999541283,0.0088296
96103930473,0.0001529446890344 8433,-0.010915480554103851,-0.01535091
8285548687,-0.06739851087331772,0.049365222454071045,0.0093643739819
52667,-0.021441806107759476,0.005624465644359589,0.04153790697455406
,0.031615592539310455,0.009158196859061718,0.011399844661355019,-0.0
36585573107004166,0.008067300543189049,0.04017193615436554,-0.004444
2773796617985,-0.030282024294137955,0.011847498826682568,0.005726383
533328772,-0.014980319887399673,0.01719740219414234,-0.0555508770048
6183,0.009031839668750763,-0.03642238676548004,-0.04240124672651291,
-0.023465050384402275,-0.0525900200009346,-0.019889121875166893,0.00
7329991087317467,-0.0348493717610836,-0.012565468437969685,0.0048930
8126270771,-0.01772800274193287,-0.03134306147694588,0.0438482910394
6686,0.03994167596101761,0.027157746255397797,0.05475940182805061,0.
06232653930783272,0.020926667377352715,0.023273082450032234,0.008956
98368549347,-0.02129791304469 1086,-0.007659547030925751,-0.001779131
2420740724,0.01271106954663992,0.000761198869440 7046,-0.019898725673
556328,0.01639355905354023,-0.029277022927999496,0.024333303794264793
,-0.016217123717069626,-0.05249570310115814,0.06855613738298416,0.0
19668709486722946,-0.013475027866661549,0.040491845458745956,-0.0128
74310836195946,-0.03878059238195419,-0.007048196159303188,-0.0104899
73239600658,-0.018570009618997574,0.008968203328549862,-0.0336739346
38500214,-0.018318500369787216,-0.009717299602925777,0.0226728022098
54126,-0.03186922520399 0936,0.025763994082808495,-0.0474987663328647
6,-0.048569515347480774,0.015351547859609127,-0.006761960685253143,0

.02838185615837574,-0.0014055072097107768,-0.05071890354156494,-0.00
23035514168441296,0.016775229945778847,0.038840923458337784,-0.00248
5994016751647,0.05317004397511482,-0.012521522119641304,0.0264921542
25707054,0.041588228195905685,0.016285225749015808,-0.03644036501646
042,-0.018341371789574623,-0.001415338832885027,-0.01759654283523559
6,-0.0425528883934021,0.006830585654824972,-0.05454989895224571,0.03
156057000160217,-0.004265787079930305 5,-0.02366132847964763 6,-0.0173
19653183221817,-0.02879355102777481,0.006474115885794163,-0.05250633
880496025,0.025808880105614662,-0.02048797346651554,-0.0507780537009
2392,-0.014409005641937256,0.012029886245727539,0.012740569189190865
,0.022116873413324356,0.03960123285651207,-0.03199855983257294,-0.00
8848310448229313,-0.07307054847478867,-0.03872452303767204,-0.029967
492446303368,0.03927792236208916,-0.035551927983760834,0.02775380201
637745,-0.0018905715551227331,0.013925244100391865,0.011243425309658
05,0.012179143726825714,-0.022628935053944588,0.022103911265730858,0
.049286894500255585,-0.07614579051733017,0.03758956119418144,0.00659
7153376787901,-0.009310507215559483,-0.05107691138982773,-0.00772778
0379354954,-0.009930762462317944,0.01052931509912014,-0.022319145500
659943,0.017863847315311432,0.010758845135569572,0.013254022225737572
2,-0.00085820798994973 3,-0.025006303563714027,-0.02406756952404976,-
0.008208317682147026,-0.010748173110187054,-0.00698865344747901,-0.0
3852702677249908 4,-0.003004539292305708,-0.02299804426729679,-0.0098
05059060454369,0.028496552258729935,0.01125089917331934,0.0317572019
9942589,-0.03186783194541931,-0.023211995139718056,-0.012901285663247
7108,-0.011883283965289593,-0.035922542214393616,-0.0054933540523052
216,-0.03287219628691673,0.0075798616744577885,-0.017400970682501793
,-0.06543558835983276,-0.07086843997240067,-0.02236025407910347,0.01
1312036775052547,0.013881144113838673,0.011187789030373096,-0.021352
853626012802,0.012095217593014 24,0.010986448265612125,0.028046626597
6429,0.0028384842444211245,-0.012631536461412907,-0.0135343223810195
92,-0.015131715685129166,0.040526192635297775,0.037994548678398 13,0.
017581030726432 8,0.0023343598004430532,0.004337592050433159,0.011728
398501873016,0.009851860813796 52,-0.04791787639260292,0.000323022453
8128823,0.07999759912490845,-0.05766419321298599,0.04435455054044723
5,0.030577993020415306,0.03863826021552086,0.012316863052546978,-0.0
1816105842590332,0.010841290466487408,0.004113945178687 5725,0.036628
97273898125,0.07028222084045 41,0.053291574120521545,-0.0105751883238
554,-0.011610777117311954,0.006535766180604696,-0.04369916766881943,
-0.050802748650312424,-0.019643759354948997,0.014163920655846596,0.0
5906696245074272,0.05733434855937958,0.015502110123634338,0.04357575
252652168,0.052199672907590866,0.02231508120894432,0.013078796677291
393,-0.007436834741383791,-0.003492080606520176,0.01568499580025673,
-0.09041984379291534,-0.05082314461469 65,0.03946860134601593,0.00012

```
49655761057511,0.016918376088142395,0.0001259226264664903 3,-0.074632
86072015762,-0.003312620334327221,-0.0365043468773365,-0.02361810766
160488,-0.026499824598431587,0.07702545076608658,0.00300560705363750
46,-0.021168861538171768,0.008991563692688942,-0.04691014811396599,0
.031971074640750885,-0.0174985621124506,0.05094505473971367,-0.03515
065833926201,0.04063650965690613,-0.0017413242021575 57,0.02026153355
8368683,-0.058702364563941956,-0.05843470990657 8064,-0.0521428696811
1992,0.058829329907894135,-0.034132201224565506,0.018815433606505394
,0.015444993041455746,-0.019551441073417664,-0.007262536324560642,0.
08271834999322891,0.019005021080374718,-0.031974907964468,0.02266382
0534944534,0.007609522435814142,-0.09004736691713333,-0.049411952495
57495,-0.006096153520047665,-0.03533686697483063,0.00563322287052869
8,-0.041101887822151184,0.02555353380739689,-0.023150743916630745,-0
.03864051774144173,-0.011613970622420311,-0.05419612675905228,0.0334
1307491064072,-0.03822252154350281,-0.0159537885338068,0.05595235154
032707,-0.001516361953690648,-0.008176641538739204,0.048195272684097
29,-0.012294626794755459,0.034582968801259995,0.03888314217329025,-0
.03052809089422226,0.017518021166324615,0.003148387419059 8726,0.0113
57742361724377,0.045062460005283356,-0.03600004315376282,-0.00239939
8246780038,0.004231774713844061,0.04196177423000336,0.05473398789763
4506,-0.032597851008176804,-0.022816237062215805,-0.0009701708913780
749,0.011314227245748043,-0.048491064459085464,-0.009732344187796116
,-0.035001225769519806,-0.009082093834877014,0.0709235668182373,-0.0
1534789241850376 1,-0.0015579608734697104,0.017158204689621925,-0.059
91761386394501,-0.018858136609196663,-0.001102324342355132,-0.016811
04116141796,0.03128630295395851,0.026118021458387375,-0.031302850693
46428,0.0374942384660244,-0.0007803713670000434,-0.03938727825880051
,0.028128813952207565,-0.0022652731277048588,-0.013466103002429008,-
0.0370059572160244,-0.022594835609197617,0.003917117603123188,0.0242
0860342681408,0.04335761442780495,0.04469241946935654,0.051333390176
296234,-0.023145707324147224,0.021898699924349785,-0.043951995670795
44,0.010561014525592327,0.04481233283877373,-0.012639965862035751,0.
0635089352726936 3,0.01094555202871561,0.03683890774846077,0.01902900
2636671066,0.020309962332248688,0.02548789419233799,-0.0155893396586
17973,0.013123778626322746,0.06714979559183121,-0.005883317440748215
,-0.02384098246693611,0.034502606838941574,0.04979805275797844,0.026
102885603904724,-0.010899506509304047,0.03565442934632301,0.01066827
7740478516,-0.04402590170502663,0.05710256099700928,0.05378012731671
333,-0.003941510338336229,-0.013945785351097584,0.008478756062686443
,-0.048995040357112885);
        System.out.println("---------------------- hybridSearch ----
------------------");
```

```
        SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
        HybridSearchParam hybridSearchParam =
HybridSearchParam.newBuilder()

.withAnn(AnnOption.newBuilder().withFieldName("vector")
                        .withData(generateRandomVector(768))
                        .build())

.withMatch(MatchOption.newBuilder().withFieldName("sparse_vector")

.withData(encoder.encodeQueries(Arrays.asList("正大光明，忠良善果弥
深")))
                        .build())
                // 指定 Top K 的 K 值
                .withRerank(new
WeightRerankParam(Arrays.asList("vector","sparse_vector"),
Arrays.asList(1, 1)))
                .withLimit(10)
                // 过滤获取到结果
                .withFilter(filterParam)
//                .withRetrieveVector(true)
//                .withOutputFields(Arrays.asList("segment"))
                .build();
        List<List<Document>> siDocs =
collection.hybridSearch(hybridSearchParam).getDocumentsList();
        int i = 0;
        for (Object docs : siDocs) {
//            System.out.println("\tres: " + (i++) +
docs.toString());
            for (Document doc : (List<Document>)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");
        SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
        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"))

.withSparseVector(encoder.encodeQueries(Arrays.asList("正大光明，忠良善
果弥深")).get(0))
                .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);
//         }
        client.dropDatabase(DBNAME);
    }



    /**
     * 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、稀疏向量 sparse_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>
     * </ol>
     *
     * @param collName
     * @return
     */
    private static CreateCollectionParam initCreateCollectionParam(String collName) {
        return CreateCollectionParam.newBuilder()
                .withName(collName)
                .withShardNum(1)
                .withReplicaNum(1)
                .withDescription("test sparse collection0")
                .addField(new FilterIndex("id", FieldType.String, IndexType.PRIMARY_KEY))
```

```
                .addField(new VectorIndex("vector",
BGE_BASE_ZH.getDimension(), IndexType.HNSW,
                        MetricType.IP, new HNSWParams(16, 200)))
                .addField(new SparseVectorIndex("sparse_vector",
IndexType.INVERTED, MetricType.IP))
                .addField(new FilterIndex("bookName",
FieldType.String, IndexType.FILTER))
                .addField(new FilterIndex("author",
FieldType.String, IndexType.FILTER))
                .build();
    }
}
```

Go

```
package main

import (
    "context"
    "fmt"
    "log"
    "math/rand"
    "strconv"
    "time"

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

type Demo struct {
    client *tcvectordb.Client
}

var (
    vectors = generateRandomVecs(768, 5)
)

func NewDemo(url, username, key string) (*Demo, error) {
    // cli, err := tcvectordb.NewRpcClient(url, username, key,
&tcvectordb.ClientOption{
```

```
    //  ReadConsistency: tcvectordb.EventualConsistency})
    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；如果使用稀疏向量，需要创建稀疏向量对应的索引
    // 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:  768,
        MetricType: tcvectordb.IP,
        Params: &tcvectordb.HNSWParam{
            M:              16,
            EfConstruction: 200,
        },
    })
    index.SparseVectorIndex = append(index.SparseVectorIndex,
tcvectordb.SparseVectorIndex{
        FieldName:  "sparse_vector",
        FieldType:  tcvectordb.SparseVector,
        IndexType:  tcvectordb.SPARSE_INVERTED,
        MetricType: tcvectordb.IP,
    })

    index.FilterIndex = append(index.FilterIndex,
tcvectordb.FilterIndex{FieldName: "id", FieldType:
tcvectordb.String, IndexType: tcvectordb.PRIMARY})

    // 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, 1, "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）

    bm25, err := encoder.NewBM25Encoder(&encoder.BM25EncoderParams{Bm25Language: "zh"})
    if err != nil {
        log.Fatalf(err.Error())
    }

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

    // 如需了解分词的情况，可参考下一行代码获取
    tokens := bm25.GetTokenizer().Tokenize(segments[0])
    fmt.Println("tokens: ", tokens)

    sparse_vectors, err := bm25.EncodeTexts(segments)
    if err != nil {
        log.Fatalf(err.Error())
    }

    documentList := make([]tcvectordb.Document, 0)
    for i := 0; i < 5; i++ {
        id := "000" + strconv.Itoa(i)
        documentList = append(documentList, tcvectordb.Document{
            Id:           id,
```

```
            Vector:       vectors[i],
            SparseVector: sparse_vectors[i],
        })
    }

    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{"0000", "0001", "0002", "0003", "0004"}
    outputField := []string{"id", "sparse_vector"}

    result, err := coll.Query(ctx, documentIds,
&tcvectordb.QueryDocumentParams{
        RetrieveVector: false,
        OutputFields:   outputField,
        Limit:          5,
        Offset:         0,
    })
    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("------------------------------ hybridSearch -------
------------------------")
    // 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
    bm25, err :=
encoder.NewBM25Encoder(&encoder.BM25EncoderParams{Bm25Language:
"zh"})
    if err != nil {
        log.Fatalf(err.Error())
    }
    sparseVec, err := bm25.EncodeQuery("腾讯云向量数据库")
    if err != nil {
        log.Fatalf(err.Error())
    }

    annSearch := &tcvectordb.AnnParam{
        FieldName: "vector",
        Data:      vectors[0],
    }

    keywordSearch := &tcvectordb.MatchOption{
        FieldName: "sparse_vector",
        Data:      sparseVec,
    }

    limit := 2
    searchRes, err := coll.HybridSearch(ctx,
tcvectordb.HybridSearchDocumentParams{
        AnnParams: []*tcvectordb.AnnParam{annSearch},
        Match:     []*tcvectordb.MatchOption{keywordSearch},
        // rerank也支持rrf，使用方式见下
        // Rerank: &tcvectordb.RerankOption{
        //  Method:    tcvectordb.RerankRrf,
        //  RrfK: 1,
        // },
        Rerank: &tcvectordb.RerankOption{
            Method:    tcvectordb.RerankWeighted,
```

```
            FieldList: []string{"vector", "sparse_vector"},
            Weight:    []float32{0.1, 0.9},
        },
        Limit:         &limit,
        OutputFields: []string{"id", "sparse_vector"},
    })
    if err != nil {
        return err
    }

    // 输出相似性检索结果，检索结果为二维数组，每一位为一组返回结果，分别对应search时指定的多个向量
    for i, item := range searchRes.Documents {
        log.Printf("HybridSearchDocumentResult, index: %d ==================", i)
        for _, doc := range item {
            log.Printf("HybridSearchDocument: %+v", doc)
        }
    }
    return nil
}

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

    log.Println("----------------------------- Update -----------------------------")

    documentId := []string{"0002"}
    bm25, err := encoder.NewBM25Encoder(&encoder.BM25EncoderParams{Bm25Language: "zh"})
    if err != nil {
        log.Fatalf(err.Error())
    }

    segments := []string{
        "腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。",
```

```
    }

    sparse_vectors, err := bm25.EncodeTexts(segments)
    if err != nil {
        log.Fatalf(err.Error())
    }

    result, err := coll.Update(ctx, tcvectordb.UpdateDocumentParams{
        QueryIds:         documentId,
        UpdateSparseVec: sparse_vectors[0],
    })
    if err != nil {
        return err
    }
    log.Printf("UpdateResult: %+v", result)

    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 generateRandomVecs(dim int, vecNum int) [][]float32 {
    var randGen = rand.New(rand.NewSource(time.Now().UnixNano()))
    arr := make([][]float32, vecNum)
    for i := range arr {
        vector := make([]float32, dim)
        for j := 0; j < dim; j++ {
            vector[j] = randGen.Float32()
        }
```

```
        arr[i] = vector
    }
    return arr
}

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

    ctx := context.Background()
    testVdb, err := NewDemo("vdb http url or ip and port", "root",
"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.UpdateAndDeleteCollection(ctx, database,
collectionName)
    printErr(err)
    err = testVdb.DeleteAndDrop(ctx, database, collectionName)
    printErr(err)
}
```

高级功能：自训练稀疏向量 Encoder

最近更新时间：2024-12-17 16:26:02

考虑到内置的 BM25 Model 是基于通用语料库训练得到，在特定领域下通常不能表现出最佳的效果。因此，在一些特定场景下，用户可以选择上传自己的数据，来调整稀疏向量生成模型的超参数，以确保模型能够准确地反映特定领域的文本特性和用户查询的意图。

步骤1：准备特定领域语料

通常需要提前准备好一定数量 Document 来入库，这些 Document 需要和特定的业务领域直接相关。语料直接决定了 BM25模型的参数。

- 语料来源应尽可能反映对应场景的特性及对应真实场景的词频信息。
- 调节合理的语料切片长度和切片数量，避免出现语料当中只有少量长文本的情况。

步骤2：训练模型，保存参数文件

训练模型是一个“统计”参数的过程。fit_corpus：根据用户传入的文本集调整计算稀疏向量时用到的词频、文档数等参数，使得结果更贴近用户数据集。如下以向量数据库的内容为例。

```
from tcvdb_text.encoder import BM25Encoder

bm25 = BM25Encoder.default('zh')

bm25.fit_corpus([
        '腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务，专用于存储、索引、检索、管理由深度神经网络或其他机器学习模型生成的大量多维嵌入向量。',
        '作为专门为处理输入向量查询而设计的数据库，它支持多种索引类型和相似度计算方法，单索引支持10亿级向量规模，高达百万级 QPS 及毫秒级查询延迟。',
        '不仅能为大模型提供外部知识库，提高大模型回答的准确性，还可广泛应用于推荐系统、NLP 服务、计算机视觉、智能客服等 AI 领域。',
        '腾讯云向量数据库（Tencent Cloud VectorDB）作为一种专门存储和检索向量数据的服务提供给用户， 在高性能、高可用、大规模、低成本、简单易用、稳定可靠等方面体现出显著优势。 ',
        '腾讯云向量数据库可以和大语言模型 LLM 配合使用。企业的私域数据在经过文本分割、向量化后，可以存储在腾讯云向量数据库中，构建起企业专属的外部知识库，从而在后续的检索任务中，为大模型提供提示信息，辅助大模型生成更加准确的答案。',
        '腾讯云数据库托管机房分布在全球多个位置，这些位置节点称为地域（Region），每个地域又由多个可用区（Zone）构成。每个地域（Region）都是一个独立的地理区域。每个地域内都有多个相互隔离的位置，称为可用区（Zone）。每个可用区都是独立的，但同一地域下的可用区通过低时延的内网链路相连。腾讯云支持用户在不同位置分配云资源，建议用户在设计系统时考虑将资源放置在不同可用区以屏蔽单点故障导致的服务不可用状态。'
```

```
    ])
print('download bm25 params')

# 保存训练后的模型参数到指定文件
bm25.download_params('./params.json')
```

步骤3：装载参数文件

在步骤2中，您已经下载了训练后的参数文件。在后续的使用过程中，可以通过 set_params() 接口装载参数文件，从而保证通过 tcvdb-text 工具生成的稀疏向量使用自训练的模型参数。

```
# 装载参数文件里的模型参数
bm25.set_params('./params.json')
```

步骤4：应用训练的参数生成稀疏向量

```
# 根据文本内容，生成对应的稀疏向量
texts = ['腾讯云向量数据库（Tencent Cloud VectorDB）是一款全托管的自研企业级分布式数据库服务',
         '腾讯云向量数据库可以和大语言模型 LLM 配合使用']
sparse_vectors: List[SparseVector] = bm25.encode_texts(texts)
```

步骤5：给数据库写入稀疏向量

```
client.upsert(
    documents=[
        {
            "id": "0000",
            ...
            "sparse_vector": sparse_vectors[0]
        },
        {
            "id": "0001",
            ...
            "sparse_vector": sparse_vectors[1]
        }
    ]
```

步骤6：应用训练的参数将检索的文本信息转为稀疏向量，进行高阶混合检索

调用 bm25.encode_quires() 接口转换检索的文本为稀疏向量，综合稠密向量与稀疏向量权重进行混合检索.

```
# 执行混合检索，并使用指定权重（Weighted）的Rerank方法
doc_lists = client.hybrid_search(
    ......
    ann=[
        AnnSearch(
            field_name="vector",
            data=[0.3123, 0.43, 0.213],
        ),
    ],
    match=[
        KeywordSearch(
            field_name="sparse_vector",
            data=bm25.encode_quires('向量数据库'),
        ),
    ],
    rerank=WeightedRerank(
        field_list=['vector', 'sparse_vector'],
        weight=[0.9, 0.1],
    ),
    ......
)
```

高级功能：分词优化增强文本解析能力

最近更新时间：2025-03-11 10:48:43

jieba 是一个广泛应用于自然语言处理（NLP）领域的流行中文分词库。腾讯云向量数据库的稀疏向量工具集成 jieba 分词库，能够将文本快速拆分为独立的词汇单元。并且，使用稀疏向量工具，支持用户根据实际业务选择合适的分词模式，支持使用自定义分词词表和自定义停用词，便于用户针对特定领域的词表精细化处理分词，进一步完善分词效果，增强文本解析的准确性。

功能明细	说明	具体使用示例
选择分词模式	**精确模式**：将文本精确地切分，确保分词结果的准确性，适用于需要高精度分词的场景，如文本分析和语义理解。	精确模式
	全模式：将文本中所有可能的词语都扫描出来，速度非常快，但可能产生一些冗余数据，适用于需要快速提取所有可能词语的场景。例如，搜索引擎的索引处理和关键词提取。	全模式
使用自定义分词词表	自定义词表，将需要保留、无需拆分的词语整理在一个文本中，每行一个词汇。jieba 在分词时，会优先识别这个文本中的词语，避免将其拆分为更小的单元。例如，在 userdict_example.txt 中，写入一个词汇“腾讯云”。在分词处理时，将不会被处理为 `'腾讯', '云'`，而保留为 `'腾讯云'`。	自定义词表
使用停用词	停用词（Stopwords）指在文本中频繁出现但通常对文本的语义贡献较小的词汇。稀疏向量工具支持灵活指定需过滤掉而不参与分词的词汇，以提高文本处理的效率和准确性。	使用停用词

选择分词模式

当前稀疏向量工具支持两种分词模式，精确模式与全模式。默认为精确模式。

精确模式

稀疏向量工具默认使用**精确模式**分词，应用示例如下所示。

Python

```python
from tcvdb_text.encoder import BM25Encoder

def use_cut_all():
    bm25 = BM25Encoder.default('zh')
```

```
    print("\nJieba分词模式：精确模式，试图将句子最精确地切开")
    print(bm25.tokenizer.toenize('什么是腾讯云向量数据库'))
```

分词结果，如下所示。

```
Jieba分词模式：精确模式，试图将句子最精确地切开
['什么', '是', '腾讯', '云', '向量', '数据库']
```

Java

```
package com.tencent.tcvdbtext;

import com.tencent.tcvdbtext.encoder.SparseVectorBm25Encoder;
import com.tencent.tcvdbtext.tokenizer.JiebaTokenizer;

public class example {
    public static void main(String[] args) {
        SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
        System.out.println("Jieba分词模式：精确模式，试图将句子最精确地切
开");
        System.out.println(encoder.getTokenizer().tokenize("什么是腾讯
云向量数据库"));
    }
}
```

分词结果，如下所示：

```
Jieba分词模式：精确模式，试图将句子最精确地切开
[什么, 是, 腾讯, 云, 向量, 数据库]
```

Go

```
func main() {
    bm25, _:= NewBM25Encoder(&BM25EncoderParams{Bm25Language: "zh"})
```

```
    fmt.Println(bm25.GetTokenizer().Tokenize("什么是腾讯云向量数据库"))
}
```

分词结果，如下所示。

```
[什么 是 腾讯 云 向量 数据库]
```

全模式

参数 cut_all 用来控制是否采用全模式，**bm25.tokenizer.cut_all = True** 即可启用全模式分词。

> **说明：**
> 全模式分词能够召回更多的词汇组合，通常在需要全面覆盖文本信息时效果更佳，但可能会增加存储空间的占用，并对性能产生一定影响。因此，是否采用全模式分词需要根据具体的应用场景和需求来决定。

Python

```
from tcvdb_text.encoder import BM25Encoder

def use_cut_all():
    bm25 = BM25Encoder.default('zh')
    print("\nJieba分词模式：全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义")
    bm25.tokenizer.cut_all = True
    print(bm25.tokenizer.tokenize('什么是腾讯云向量数据库'))
```

```
Jieba分词模式：全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义
['什么', '是', '腾讯', '云', '向量', '数据', '数据库', '据库']
```

Java

```
package com.tencent.tcvdbtext;
```

```
import com.tencent.tcvdbtext.encoder.SparseVectorBm25Encoder;
import com.tencent.tcvdbtext.tokenizer.JiebaTokenizer;
public class example {
    public static void main(String[] args) {
        SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
        encoder.setCutAll(true);
        System.out.println("Jieba分词模式：全模式，把句子中所有的可以成词的
词语都扫描出来，速度非常快，但是不能解决歧义");
        System.out.println(encoder.getTokenizer().tokenize("什么是腾讯
云向量数据库"));
    }
}
```

分词结果，如下所示：

```
Jieba分词模式：全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不
能解决歧义
[什么，是，腾讯，云，向量，数据，据库，数据库]
```

Go

```
func main() {
    bm25, _ := NewBM25Encoder(&BM25EncoderParams{Bm25Language:
"zh"})
    cutAll := true
    jbtParams := tokenizer.TokenizerParams{
        CutAll: &cutAll,
    }
    _ := bm25.GetTokenizer().UpdateParameters(jbtParams)
    fmt.Println(bm25.GetTokenizer().Tokenize("什么是腾讯云向量数据库"))
}
```

分词结果，如下所示。

```
[什么 是 腾讯 云 向量 数据库]
```

自定义分词词表

自定义词表，将需要保留、无需拆分的词语整理在一个文本中，每行一个词汇。jieba 在分词时，会优先识别这个文本中的词语，避免将其拆分为更小的单元。
例如，在 userdict_example.txt 中，写入一个词汇“腾讯云”。在分词处理时，将不会被处理为 `'腾讯', '云'`，而保留为 `'腾讯云'`，如下所示。

```
腾讯云
数据库
```

Python

```python
from tcvdb_text.encoder import BM25Encoder

def set_dict_example():
    bm25 = BM25Encoder.default('zh')
    print(bm25.tokenizer.tokenize('什么是腾讯云向量数据库。'))
    bm25.set_dict('../userdict_example.txt')
    print(bm25.tokenizer.tokenize('什么是腾讯云向量数据库。'))

if __name__ == '__main__':
    set_dict_example()
```

分词结果对比，如下所示。

```
['什么', '是', '腾讯', '云', '向量', '数据库']
['什么', '是', '腾讯云', '向量', '数据库']
```

Java

```java
package com.tencent.tcvdbtext;
```

```
import com.tencent.tcvdbtext.encoder.SparseVectorBm25Encoder;
import com.tencent.tcvdbtext.tokenizer.JiebaTokenizer;

public class example {
    public static void main(String[] args) {
        SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
        // 默认分词词表
        System.out.println(tokenizer.tokenize("什么是腾讯云向量数据
库。"));
        // 自定义用户词表，所在路径，需要根据实际替换
        String path = "user_dict/userdict_example.txt";
        tokenizer.loadDict(path);
        System.out.println(tokenizer.tokenize("什么是腾讯云向量数据
库。"));
    }
}
```

分词结果，如下所示。

```
[什么, 是, 腾讯, 云, 向量, 数据库, 。]
user dict /data/home/emilyjzhang/project/vectordb-sdk-
java/tcvdb_text/src/main/resources/data/user_dict/userdict_example.t
xt load finished, tot words:5, time elapsed:2ms
[什么, 是, 腾讯云, 向量, 数据库, 。]
```

Go

```
func main() {
    bm25, _ := NewBM25Encoder(&BM25EncoderParams{Bm25Language:
"zh"})
    fmt.Println(bm25.GetTokenizer().Tokenize("什么是腾讯云向量数据
库。"))
    _ := bm25.SetDict("../data/userdict_example.txt")
    fmt.Println(bm25.GetTokenizer().Tokenize("什么是腾讯云向量数据
库。"))
 }
```

分词结果，如下所示。

```
[什么 是 腾讯 云 向量 数据库]
[什么 是 腾讯云 向量 数据库]
```

使用停用词

默认开启停用词，使用 默认停用词列表 进行分词，支持关闭停用词，支持指定停用词词表。停用词词表文件内容必须是一行一个词汇，例如，user_stopwords.txt 文件内容如下所示。

```
什么
是
```

Python

```
def set_stopwords():
    bm25 = BM25Encoder.default('zh')
    print("\n使用默认stopwords: ")
    print(bm25.tokenizer.tokenize('什么是腾讯云向量数据库。'))

    print("\n不使用stopwords: ")
    bm25.tokenizer.set_stopwords(False)
    print(bm25.tokenizer.tokenize('什么是腾讯云向量数据库。'))

    print("\n自定义stopwords，从文件加载: ")
    ## 如下示例，在./user_stopwords.txt 文件中，逐行列出了需要过滤的停用词:
什么 是。
    bm25.tokenizer.set_stopwords('./user_stopwords.txt')
    print(bm25.tokenizer.tokenize('什么是腾讯云向量数据库。'))
```

分词结果，如下所示。

```
使用默认stopwords:
['什么', '是', '腾讯', '云', '向量', '数据库']

不使用stopwords:
['什么', '是', '腾讯', '云', '向量', '数据库', '。']
```

```
自定义stopwords，从文件加载：
['腾讯', '云', '向量', '数据库', '。']
```

Java

```
package com.tencent.tcvdbtext;

import com.tencent.tcvdbtext.encoder.SparseVectorBm25Encoder;
import com.tencent.tcvdbtext.tokenizer.JiebaTokenizer;
public class example {
    public static void main(String[] args) {
        SparseVectorBm25Encoder encoder =
SparseVectorBm25Encoder.getBm25Encoder("zh");
        // 使用默认的停用词
        System.out.println(encoder.getTokenizer().tokenize("什么是腾讯
云向量数据库。"));
        //关闭停用词，查看分词效果
        encoder.setEnableStopWords(false);
        System.out.println(encoder.getTokenizer().tokenize("什么是腾讯
云向量数据库。"));
        // 使用自定义停用词词表，指定停用词词表的所在路径，需要根据实际放置路径
替换
        String path = "/data/user_stopwords.txt";
        encoder.setStopWords(path);
        // 再次开启停用词，使用停用词词表分词。本示例中，
在./user_stopwords.txt文件中，逐行列出了需要过滤的停用词：什么 是。
        encoder.setEnableStopWords(true);
        System.out.println(encoder.getTokenizer().tokenize("什么是腾讯
云向量数据库。"));
    }
}
```

分词结果，如下所示。

```
[什么, 是, 腾讯, 云, 向量, 数据库]
[什么, 是, 腾讯, 云, 向量, 数据库, 。]
```

```
[腾讯, 云, 向量, 数据库, 。]
```

Go

分词处理时，自动过滤默认的停用词。

```
func main() {
    bm25, _ := NewBM25Encoder(&BM25EncoderParams{Bm25Language:
"zh"})
    // 默认开启停用词,StopWords 控制是否使用使用停用词。StopWords 为true，使
用停用词；StopWords 为false，则不使用停用词。
    jbtParams := tokenizer.TokenizerParams{
        StopWords: true,
    }
    _ := bm25.GetTokenizer().UpdateParameters(jbtParams)
    fmt.Println(bm25.GetTokenizer().Tokenize("什么是腾讯云向量数据
库。"))
}
```

```
[什么 是 腾讯 云 向量 数据库]
```

不使用停用词， `StopWords: false` **，不识别停用词。**

```
func main() {
    bm25, _ := NewBM25Encoder(&BM25EncoderParams{Bm25Language:
"zh"})
    jbtParams := tokenizer.TokenizerParams{
        StopWords: false,
    }
    _ := bm25.GetTokenizer().UpdateParameters(jbtParams)
    fmt.Println(bm25.GetTokenizer().Tokenize("什么是腾讯云向量数据
库。"))
}
```

```
[什么 是 腾讯 云 向量 数据库 。]
```

使用用户自定义的停用词，将停用词写入在 `user_define_stopwords.txt` 中，分词处理时，自动过滤该文档指定的停用词。如下示例，该文档写入词汇'什么'与'是'，分词效果如下所示。

```
func main() {
    bm25, _ := NewBM25Encoder(&BM25EncoderParams{Bm25Language:
"zh"})
    // 使用用户自定义的停用词，需要加载停用词词表文本所在路径
    // 本示例中，在./user_stopwords.txt文件中，逐行列出了需要过滤的停用词：什
么 是。
    jbtParams := tokenizer.TokenizerParams{
        StopWords: "../data/user_stopwords.txt",
    }
    _ := bm25.GetTokenizer().UpdateParameters(jbtParams)
    fmt.Println(bm25.GetTokenizer().Tokenize("什么是腾讯云向量数据
库。"))
}
```

```
[腾讯 云 向量 数据库]
```

动态标量字段

最近更新时间：2024-12-24 17:53:53

功能介绍

在数据库操作中，创建集合（Collection）的 Schema（结构）时，需要指定各个字段的名称和数据类型。在数据库 Schema 设计之初，由于业务需求的不断演进和数据模型的持续发展，某些字段可能无法完全预见，动态标量字段允许数据库在不修改原有 Schema 的情况下，适应数据结构的变化需求，从而提高数据库的灵活性和适应性。启用动态标量字段功能后，集合中的所有标量字段将自动创建 Filter 索引以提升查询效率。同时，支持灵活选择对特定字段不建立索引，以优化存储空间或减少索引维护开销，实现更加灵活、精细的数据库管理。

开启方式

在创建数据库集合时，可以通过配置控制参数来启用标量字段的全索引功能，并明确指定哪些字段不需要建立索引。Python、Java、Go SDK的开启方式，如下表所示。

SDK	开启标量字段参数	开启方式	创建集合

Python SDK	filter_index_config	`filter_index_config=FilterIndexConfig(`
		`  # filter_all 为 True，则开启标量字段全索引`
		`  filter_all=True,`
		`  # fields_without_index 在开启之后，指定不创建索引的字段`
		`  fields_without_index=['author'],`
		`  # max_str_len 在开启之后，指定创建索引标量字段的最大字符数，超过限制则截断创建索引`
		`  max_str_len=32,`
		`)`
			Python SDK-新建Collection

Java SDK	FilterIndexConfig	`.withFilterIndexConfig(FilterIndexConfig.newBuilder()`
		`  // FilterAll 为 true，则开启标量字段全索引`
		`  .withFilterAll(true)`
			Java SDK-新建Collection

		``// FieldWithoutFilterIndex 在开启之后，指定不创建索引的字段
		.withFieldWithoutFilterIndex(Arrays.asList("test1", "test2"))
		// MaxStrLen 在开启之后，指定创建索引标量字段的最大字符数，超过限制则截断创建索引
		.withMaxStrLen(64)
		.build())```	

Go SDK	FilterIndexConfig	```maxStrLen := uint32(32)
		param := &tcvectordb.CreateCollectionParams{
		FilterIndexConfig: &tcvectordb.FilterIndexConfig{
		// FilterAll 为 true，则开启标量字段全索引
		FilterAll: true,
		// FieldWithoutFilterIndex 在开启之后，指定不创建索引的字段
		FieldsWithoutIndex: []string{"author"},
		// MaxStrLen 在开启之后，指定创建索引标量字段的最大字符数，超过限制则截断创建索引
		MaxStrLen: &maxStrLen,
		},
		}```
			Go SDK-新建 Collection

使用示例

如下以 Java SDK 为例，给出开启动态标量字段全索引，并应用 FIlter 表达式检索的具体方式。获取 Java SDK，请参见 SDK 准备 。

步骤1：创建数据库

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

```
import com.tencent.tcvectordb.client.VectorDBClient;
import com.tencent.tcvectordb.model.*;
import com.tencent.tcvectordb.model.Collection;
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.BaseRes;
import com.tencent.tcvectordb.model.param.enums.ReadConsistencyEnum;
import com.tencent.tcvectordb.utils.JsonUtils;
import java.util.*;

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

步骤2：创建集合，开启动态标量字段

```
// 初始化Collection参数，通过配置 withFilterIndexConfig 开启动态标量字段功能。
// 在以下示例中，集合开启了动态标量字段，同时指定"test1", "test2"两个字段不创建
filter索引，其余字段均默认创建filter索引
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("book-vector")
        .withShardNum(1)
        .withReplicaNum(1)
        .withDescription("this is a java sdk test")
        .addField(new FilterIndex("id", FieldType.String,
IndexType.PRIMARY_KEY))
        .addField(new VectorIndex("vector", 3, IndexType.HNSW,
                MetricType.COSINE, new HNSWParams(16, 200)))
```

```
        .withFilterIndexConfig(FilterIndexConfig.newBuilder()
            .withFilterAll(true)
            .withFieldWithoutFilterIndex(Arrays.asList("test1",
"test2"))
            .withMaxStrLen(64)
            .build())
        .build();
Collection collection = db.createCollection(collectionParam);
```

步骤3：插入数据

写入向量数据，指定标量字段，并查询集合索引结构。

```
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("array_test",
Arrays.asList("1","2","3")))
                .addDocField(new DocField("test1", 28))
                .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("array_test",
Arrays.asList("4","5","6")))
                .addDocField(new DocField("test2", 25))
                .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("array_test",
Arrays.asList("7","8","9")))
                .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("array_test",
Arrays.asList("10","11","12")))
                .addDocField(new DocField("test1", 23)
                .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"))
                .build()));
System.out.println("---------------------- upsert ----------------------
");
InsertParam insertParam =
InsertParam.newBuilder().withDocuments(documentList).build();

AffectRes affectRes = client.upsert("db-test", "book-vector",
insertParam);
System.out.println(JsonUtils.toJsonString(affectRes));
// 查询集合的结构
Database database = client.database("db-test");
Collection collection = database.describeCollection("book-vector");
System.out.println("\tres: " + collection.toString());
```

使用 `describeCollection` 查询集合结构，如下所示，标量字段 bookName、author、array_test 均已自动创建 Filter 索引，而特定的字段 test1、test2并没有创建索引。

```
{
  "database": "db-test",
  "collection": "book-vector",
  "replicaNum": 1,
  "shardNum": 1,
  "description": "this is a java sdk test",
  "indexes": [
    {
      "fieldName": "array_test",
      "fieldType": "array",
      "indexType": "filter",
      "fieldElementType": "string"
    },
    {
      "fieldName": "author",
```

```
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "id",
        "fieldType": "string",
        "indexType": "primaryKey"
      },
      {
        "fieldName": "bookName",
        "fieldType": "string",
        "indexType": "filter"
      },
      {
        "fieldName": "vector",
        "fieldType": "vector",
        "indexType": "HNSW",
        "metricType": "COSINE",
        "params": {
          "efConstruction": 200,
          "M": 16
        },
        "dimension": 3
      }
    ],
    "createTime": "2024-12-19 17:07:53",
    "documentCount": 0,
    "indexStatus": {
      "status": "ready"
    },
    "alias": [],
    "filterIndexConfig": {
      "filterAll": true,
      "fieldsWithoutIndex": [
        "test1",
        "test2"
      ],
      "maxStrLen": 32
    }
}
```


步骤4：应用动态标量字段相似性检索

```
// 使用标量字段设置 Filter 表达式
Filter filterParam = new Filter("bookName=\"三国演义\"")
        .and(Filter.exclude("array_test", Arrays.asList("7")));
System.out.println("---------------------- search ----------------------
");
// 设置检索参数
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(10)
        // 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 = client.search(DBNAME, COLL_NAME,
searchByVectorParam);
int i = 0;
for (List<Document> docs : svDocs) {
    System.out.println("\tres: " + i);
    i++;
    for (Document doc : docs) {
        System.out.println("\tres: " + doc.toString());
    }
}
```

相似性检索结果，如下所示，可以根据动态写入的标量字段进行数据过滤，筛选出满足 filter 条件的检索结果。

```
res: 0
res: {"id":"0004","score":0.9997869729995728,"bookName":"三国演
义","author":"罗贯中","array_test":["10","11","12"]}
res: {"id":"0005","score":0.9991745948791504,"bookName":"三国演
义","author":"罗贯中"}hor":"罗贯中"}
```

TTL

最近更新时间：2024-12-20 10:59:22

功能介绍

TTL（Time to Live，生存周期）功能，指数据项从创建或更新后开始，到自动被系统删除的时间期限。在创建数据库集合时，通过为文档设置TTL，数据库能够自动清理过期的数据，从而优化存储空间的使用，保持数据的时效性，并确保查询结果的准确性，特别是在数据具有时效性要求的场景中，如实时推荐系统等。

适用场景

某些业务场景下，业务数据的增长很快，并且业务数据的热度随着时间推移会有明显的降低，可以使用 TTL 将热度降低的数据在特定的过期时间时自动清理。例如：在电子商务平台中，用户搜索和浏览行为产生的向量数据，如点击流和用户偏好向量，通常具有很高的时效性。这些数据在短期内对于个性化推荐至关重要，但随着时间的推移，其相关性会迅速降低。在向量数据库集合中，设置TTL，便会自动删除过时的用户行为数据，确保推荐系统能够基于最新的用户互动提供实时、相关的推荐，从而提升用户体验和业务效率。

SDK demo

SDK	Demo	SDK 获取方式
Python SDK	ttl.py	Python SDK 准备
Java SDK	VectorDBExample	Java SDK 准备
Go SDK	ttl_demo	Go SDK 准备

使用示例

如下以 Java SDK 为例，给出使用 TTL 的关键步骤。

1. 创建集合时，指定 TTL 字段 expired_at，并将 TTL 字段 expired_at 设置 FIlter 索引。

```
// 初始化 Colleciton 参数
CreateCollectionParam collectionParam =
CreateCollectionParam.newBuilder()
        .withName("book-vector")
        .withShardNum(1)
        .withReplicaNum(1)
        .withDescription("this is a java sdk test")
        .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("expired_at", FieldType.Uint64,
IndexType.FILTER))

.withTtlConfig(TTLConfig.newBuilder().WithEnable(true).WithTimeField("
expired_at").build())
        .build();
Collection collection = db.createCollection(collectionParam);
```

2. 插入向量数据时，指定 TTL 字段 expired_at 过期时间。

> **说明：**
> 数据库插入数据，过期时间允许有1个小时的误差。

```
  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"))
                // 设置数据过期时间为当前时间 24 小时之后
                .addDocField(new DocField("expired_at",
System.currentTimeMillis()/1000 + 24*60*60))
                .build(),
        Document.newBuilder()
                .withId("0002")
                .withVector(Arrays.asList(0.2123, 0.25, 0.213))
                .addDocField(new DocField("bookName", "三国演义"))
                .addDocField(new DocField("author", "罗贯中"))
                .build()));
System.out.println("---------------------- upsert -------------------
--");
InsertParam insertParam =
InsertParam.newBuilder().withDocuments(documentList).build();

AffectRes affectRes = client.upsert("db-test", "book-vector",
insertParam);
System.out.println(JsonUtils.toJsonString(affectRes));
```

账号与权限管理

最近更新时间：2025-04-18 17:58:12

基本介绍

向量数据库通过创建自定义用户，并对用户授权资源对象操作权限，实现细粒度的访问控制与管理。用户（User）、资源对象（Resource）和操作（Action）权限构成了访问控制的基础框架，确保数据的安全性和合理使用。

> **说明：**
> 当前的账号权限管理功能仅适用于 Base 类数据库和集合（collection），而不涉及 AI 套件类数据库和集合视图（collectionView）。即对于 AI 类数据库和 CollectionView，仅支持使用 root 用户进行访问，并拥有完全读写权限，其他自定义用户均无法访问。

类别		功能	SDK
用户（User）		指数据库系统的使用者，即需要访问和操作数据库中数据的人员或程序。 • root 用户：创建数据库实例时，默认创建的用户，具有最高权限的超级用户，无法被任何账号（包括其自身）删除或修改其权限。 • 自定义用户：通过 API 接口，使用 root 权限创建的新用户，并分配特定资源操作权限，实现精细化访问控制。	• Python-User 管理 • Java-User 管理 • Go-User 管理 • Curl-User 管理
权限（Privilege）	资源对象（Resource）	指数据库中用户权限生效的数据实体，向量数据库可授权的资源对象包括三个层级： • Global：对实例下的所有 DB 和 Collection 对象授权。 • Database 层级：对具体某一个 Database 授权。 • Collection 层级：对具体某一个特定的 Collection 单独授权。	
	操作权限（Action）	定义用户对资源对象可执行的操作权限。目前可分配的操作规则如下（暂不支持授权接口级别的操作）： • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。	

		• read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。	

创建用户

创建自定义用户，需指定用户名及其对应的密码，如下以 Python 为例，说明用户名与密码的配置规则。

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

password	设置密码	• 密码长度要求：8~128字符。
		• 密码设置支持如下类型的任意组合。
		  ○ 字母：大写 A~Z、小写 a~z。
		  ○ 数字：0~9。
		  ○ 特殊符号包括：下划线(_)、加号(+)、减号(-)、逗号(,)、和号(&)、等号(=)、感叹号(!)、@符号(@)、井号(#)、美元符号($)、百分号(%)、脱字符(^)、星号(*)、圆括号(())、点号(.)。

Python

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

Java

```
import com.tencent.tcvectordb.client.VectorDBClient;
import com.tencent.tcvectordb.model.param.database.ConnectParam;
import com.tencent.tcvectordb.model.param.entity.BaseRes;
import com.tencent.tcvectordb.model.param.enums.ReadConsistencyEnum;
import com.tencent.tcvectordb.model.param.user.*;
import com.tencent.tcvectordb.utils.JsonUtils;
// client 为 VectorDBClient() 创建的客户端对象
BaseRes res = client.createUser("app_user", "********");
System.out.println("create user res: code:" + res.getCode() + ",
msg: " + res.getMsg());
```

Go

```
import (
    "context"
    "log"

    "github.com/tencent/vectordatabase-sdk-go/tcvectordb"
    "github.com/tencent/vectordatabase-sdk-go/tcvectordb/api/user"
)
var (
        ctx            = context.Background()
        database       = "db-test"
        database1      = "db-test1"
        collectionName = "book-vector"
)
err := client.CreateUser(ctx, tcvectordb.CreateUserParams{User:
"app_user", Password: "**********"})
if err != nil {
    log.Printf("create user failed, err: %+v", err.Error())
}
```

给用户授权

给用户授权时，需指定用户名及权限，可以给用户批量授予多组权限；每个权限中，需要指定资源对象和允许的操作。如下以 Python 接口的参数为例，说明配置权限的具体规则。

参数名	参数含义	子参数	参数配置
user	自定义用户名	-	-
privileges	对用户授予的权限列表，可以配置多条权限	resource	指定用户可操作的资源对象，分为如下三个层级。 • 全局：对实例下的所有 DB 和 Collection 对象授权。配置方式：`"resource": "*.*"` 。 • DB 层级：对某个 DB 授权，包括该 DB 下的所有 Collection。配置方式：`"resource": "db0.*"`，其中，db0 指授权的数据库名。 • Collection 层级：对某个特定集合授权，配置方式：`"resource": "db0.test_coll"` 其中，db0 指授权的数据库名，test_coll 为授权的集合名。 **说明：** resource 只能指向已存在的资源；若指定的资源不存在，则触发错误提示信息。
		actions	配置资源对象允许操作的权限。 • dbAdmin：允许执行创建（create）、删除（drop）、清空（truncate）等数据定义语言（DDL）操作。 • read：仅限于执行查询（query）和搜索（search）等读取操作，禁止进行任何数据修改。 • readWrite：允许执行查询（query）、搜索（search）等读取操作，以及插入（upsert）、更新（update）等写入操作。

Python

```
res = client.grant_to_user(
        user='app_user',
        privileges=[
        {
            "resource": "db0.*",
            "actions": ["read"]
        },
```

```
        {
            "resource": "db1.book-vector",
            "actions": ["readWrite"]
        }
])
print(res)
```

Java

```
// client 为 VectorDBClient() 创建的客户端对象
BaseRes res = client.grantToUser(UserGrantParam.newBuilder()
        .withUser("app_user")
        .withPrivileges(Arrays.asList(
                PrivilegeParam.newBuilder().withResource("db-
test.*").withActions(Arrays.asList("read")).build(),
                PrivilegeParam.newBuilder().withResource("db-
test1.*").withActions(Arrays.asList("dbAdmin")).build(),
                PrivilegeParam.newBuilder().withResource("db-
test.book-vector").withActions(Arrays.asList("readWrite")).build())
        ).build());
System.out.println("grant user permission res: code:" +
res.getCode() + ", msg: " + res.getMsg());
```

Go

```
err := client.GrantToUser(ctx, tcvectordb.GrantToUserParams{
    User: "app_user",
    Privileges: []*user.Privilege{
        {
            Resource: database + ".*",
            Actions:  []string{"read"},
        },
        {
            Resource: database + "." + collectionName,
            Actions:  []string{"readWrite"},
         },
```

```
            {
                Resource: database1 + ".*",
                Actions:  []string{"dbAdmin"},
            },
        }})
if err != nil {
    log.Printf("grant to user failed, err: %+v", err.Error())
}
```

查询用户权限

Python

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

Java

```
// client 为 VectorDBClient() 创建的客户端对象
UserDescribeRes userDescribeRes = client.describeUser("app_user");
System.out.println("describe user res: " +
userDescribeRes.toString());
```

Go

```
result, err := client.DescribeUser(ctx,
tcvectordb.DescribeUserParams{User: "app_user"})
if err != nil {
    panic(err)
}
```

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

回收用户权限

回收用户权限，需明确指出回收权限的具体用户，列出需要回收的具体权限。

说明：

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

Python

```
res = client.revoke_from_user(
        user='app_user',
        privileges={
            "resource": "db0.*",
            "actions": ["read"]
        })
print(res)
```

Java

```
BaseRes res = client.revokeFromUser(UserRevokeParam.newBuilder()
        .withUser("app_user")
        .withPrivilege(
                PrivilegeParam.newBuilder().withResource("db-
test.*").withActions(Arrays.asList("read")).build()
        ).build());
```

```
System.out.println("revoke user permission res: code:" +
res.getCode() + ", msg: " + res.getMsg());
```

Go

```
err := client.RevokeFromUser(ctx,
tcvectordb.RevokeFromUserParams{User: "app_user",
    Privileges: []*user.Privilege{
        {
            Resource: database + ".*",
            Actions:  []string{"read"},
        },
    }}
)
if err != nil {
    log.Printf("create user failed, err: %+v", err.Error())
}
```

修改用户密码

Python

```
res = client.change_password(user='app_user', password='########')
print(res)
```

Java

```
// client 为 VectorDBClient() 创建的客户端对象
BaseRes res = client.changePassword("app_user", "##########");
```

```
System.out.println("change user password res: " + res.getCode() + "
" + res.getMsg());
```

Go

```
err := client.ChangePassword(ctx, tcvectordb.ChangePasswordParams{
        User:     "app_user",
        Password: "########",
})
if err != nil {
    log.Printf("change user password err: %+v", err.Error())
}
```

API 资源层级与操作权限

向量数据库提供的操作规则（read、readWrite 等）和 API 接口的对应关系，如下表所示。

API 接口	dbAdmin	read	readWrite
/database/create	✓	–	–
/database/drop	✓	–	–
/database/list	✓	✓	✓
/collection/create	✓	–	–
/collection/drop	✓	–	–
/collection/truncate	✓	–	–
/collection/list	✓	✓	✓
/collection/describe	✓	✓	✓
/index/rebuild	–	–	✓
/index/modifyVect	–	–	✓

orIndex			
/index/add	–	–	✓
/index/drop	–	–	✓
/alias/set	–	–	✓
/alias/delete	–	–	✓
/document/query	–	✓	✓
/document/search	–	✓	✓
/document/hybridSearch	–	✓	✓
/document/keywordSearch	–	✓	✓
/document/update	–	–	✓
/document/delete	–	–	✓
/document/upsert	–	–	✓
/document/count	–	✓	✓

集合别名（Alias）

最近更新时间：2025-05-26 11:14:52

定义

别名（Alias）是向量数据库中 Collection（集合）或 CollectionView（集合视图）的逻辑概念，其核心价值在于解耦应用程序与底层数据结构的直接依赖，通过动态映射规则将用户自定义的直观名称与物理存储的集合关联。开发者无需感知集合的真实名称，即可通过别名访问目标数据，从而实现业务逻辑与存储细节的隔离，提升系统的灵活性与可维护性。

适用场景

别名的典型使用场景主要集中在实现业务无感的数据集合切换。例如，在发布新版本（如测试环境上线）时，通过将别名从原生产集合动态指向新集合，实现零停机平滑迁移；若新版本异常，则可通过别名快速回滚至旧集合，保障业务连续性。

使用限制

- **别名数量**：单个 Collection 或 CollectionView 可绑定1个或多个别名，支持多业务视角的灵活映射。
- **唯一性约束**：同一实例内，别名需全局唯一，不可重复。
- **别名级联依赖**：若对数据库（DB）、集合（Collection）或集合视图（CollectionView）执行 DROP 操作，其关联的所有别名将同步被删除。
- **访问层级限制**：
  - Document 与 DocumentSet 层级：支持通过别名直接访问（如 db.alias_name.get_document(id=123)），系统会优先解析别名指向的真实集合。
  - 其他层级操作：对数据库、集合或集合视图的管理操作（如创建、删除、修改 Schema）仅支持使用原始名称，别名无法替代。

命名规则

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。命名规则，如下所示。

> **说明：**
> 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。

- **字符范围**：仅支持英文字母、数字、下划线（_）、中划线（-）。
- **首字符**：必须以英文字母开头（如 data_2023 合法，2023_data 非法）。
- **长度限制**：字符串长度需在 [1, 128] 字符之间。

SDK 接口

SDK	接口及示例
Python	• 设置别名 • 删除别名
Java	• 设置别名 • 删除别名
Go	• 管理 Collection 别名 • 管理 CollectionView 别名

相似性计算

最近更新时间：2024-12-18 15:49:22

相似性计算方法是向量检索的基础，用于衡量高维向量数据之间的相似度。在创建 Collection 时，需要依据数据特征，选择合适的相似性计算方法。下表展示了这些广泛使用的相似性计算方法如何与各种输入数据形式和腾讯云向量数据库（Tencent Cloud VectorDB）索引相匹配。

相似性计算方法	数据格式	向量索引类型
内积（IP）	浮点型	FLAT、HNSW、IVF 系列
欧式距离（L2）		
余弦相似度（COSINE）		
汉明距离（Hamming Distance）	二进制	BIN_FLAT **说明：** 索引类型为二进制索引时，相似性计算方法只能选择 Hamming Distance。

内积（IP）

全称为 Inner Product，内积也称点积，计算结果是一个数。它计算两个向量之间的点积（内积），其计算公式如下所示。其中，a = (a1, a2,..., an) 和 b = (b1, b2,..., bn)，是 n 维空间中的两个点。计算所得值越大，越与搜索值相似。

$$\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

欧式距离（L2）

欧式距离（L2）全称为 Euclidean distance，指欧几里得距离。它计算两个向量点在空间中的直线距离。计算公式如下所示。其中，a = (a1, a2,..., an) 和 b = (b1, b2,..., bn) 是 n 维空间中的两个点。它是最常用的距离度量。计算所得的值越小，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。

$$d(\boldsymbol{a}, \boldsymbol{b}) = d(\boldsymbol{b}, \boldsymbol{a}) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2}$$

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。其计算公式如下所示。其中，a = (a1, a2,..., an) 和 b = (b1, b2,..., bn) 是 n 维空间中的两个点。|a|与|b|分别代表 a 和 b 归一化后的值。cos θ 代表 a 与 b 之间的余弦夹角。计算所得值越大，越与搜索值相似。取值范围为[-1,1]。

> **说明：**
> 在向量归一化之后，内积与余弦相似度等价。余弦相似性只考虑向量夹角大小，而内积不仅考虑向量夹角大小，也考虑了向量的长度差。

$$cos(a,b) = \frac{a \cdot b}{|a||b|}$$

汉明距离（Hamming Distance）

汉明距离，也称为 Hamming Distance，通过计算两个字符串对应位置上不同字符的数量来定义，如果字符不同，那么它们之间的汉明距离就会加一。具体来说，对于长度为 n 的二进制向量，如果两个向量的对应位有 d 个不同，则它们的汉明距离为 d。汉明距离是一种简单而有效的相似度计算方法，尤其适用于处理等长的二进制数据。

- 汉明距离越小，表示两个字符串之间的相似度越高。
- 它是一种对称度量，即字符串 A 与字符串 B 的汉明距离与字符串 B 与字符串 A 的汉明距离相同。

例如，两个十进制数 a=93和 b=73，如果将这两个数用二进制表示的话，即 a=1011101和 b=1001001，这两者的汉明距离为2，因为它们中有两个字符不一致，即在第三和第五个位置上的字符不同。