表格存储服务

开发指南

文档版本	14
发布日期	2025-01-21

华为云计算技术有限公司

版权所有 © 华为云计算技术有限公司 2025。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 HBase 应用开发指导......1
1.1 开发流程......1
1.2 准备开发环境......3
1.2.1 开发环境简介......3
1.2.2 准备运行环境......3
1.2.2.1 准备 Windows 运行环境......3
1.2.3 下载样例工程......4
1.2.4 配置并导入工程......5
1.3 开发 HBase 应用......10
1.3.1 典型场景说明......10
1.3.2 开发思路......11
1.3.3 样例代码说明......12
1.3.3.1 配置参数......12
1.3.3.2 创建 Configuration......12
1.3.3.3 创建 Connection......13
1.3.3.4 创建表......14
1.3.3.5 删除表......15
1.3.3.6 修改表......16
1.3.3.7 插入数据......17
1.3.3.8 删除数据......18
1.3.3.9 使用 Get 读取数据......19
1.3.3.10 使用 Scan 读取数据......20
1.3.3.11 使用过滤器 Filter......21
1.4 开发 HBase 冷热分离应用......22
1.4.1 应用背景......22
1.4.2 典型场景说明......22
1.4.3 开发思路......24
1.4.4 样例代码说明......24
1.4.4.1 配置参数......25
1.4.4.2 创建 Configuration......25
1.4.4.3 创建 Connection......25
1.4.4.4 创建表......26
1.4.4.5 删除表......27

1.4.4.6 修改表 28
1.4.4.7 插入数据 30
1.4.4.8 使用 Get 读取数据 34
1.4.4.9 使用 Scan 读取数据 35
1.5 访问 HBase ThriftServer 样例程序 37
1.5.1 访问 ThriftServer 操作表 37
1.6 开发标签索引应用 42
1.6.1 应用背景 42
1.6.2 典型场景说明 42
1.6.3 开发思路 43
1.6.4 样例代码说明 43
1.6.4.1 配置参数 43
1.6.4.2 创建 Configuration 43
1.6.4.3 创建数据表开启标签索引 44
1.6.4.4 写入数据 46
1.6.4.5 普通查询 46
1.6.4.6 抽样查询 47
1.6.4.7 分页查询 49
1.6.4.8 统计查询 50
1.7 调测程序 50
1.7.1 在 Windows 中调测程序 50
1.7.1.1 编译并运行程序 50
1.7.1.2 查看调测结果 51
1.7.2 在 Linux 中调测程序 52
1.7.2.1 安装客户端时编译并运行程序 52
1.7.2.2 未安装客户端时编译并运行程序 56
1.7.2.3 查看调测结果 59
1.8 对外接口 59
1.8.1 HBase Java API 59
2 Doris 应用开发指导 60
2.1 Doris 组件使用规范 60
2.2 建表 62
2.2.1 Doris 数据表和数据模型 62
2.2.1.1 数据表 62
2.2.1.2 数据模型 62
2.2.1.3 最佳实践 77
2.2.2 数据分区和分桶 78
2.2.2.1 分区（Partition） 78
2.2.2.2 分桶 87
2.2.2.3 最佳实践 88
2.2.2.4 常见问题 89
2.2.3 数据分布 90

2.3 数据导入......90
2.3.1 总览......90
2.3.2 批量数据导入......91
2.3.2.1 Broker Load......91
2.3.2.2 Stream Load......99
2.4 数据湖分析......105
2.4.1 多源数据目录......105
2.4.1.1 概述......105
2.4.1.2 Hive......106
2.5 通过 JDBC 方式连接 Doris......109
2.5.1 JDBC 通过非 ssl 方式连接 doris......109
2.5.2 JDBC 通过 ssl 方式连接 doris（验证证书）......110
2.5.3 JDBC 通过 ssl 方式连接 doris（无需验证证书）......111
2.6 开发 Doris 冷热分离应用......112
2.6.1 应用背景......112
2.6.2 典型场景说明......112
2.6.3 开发思路......114
2.6.4 样例代码......114
3 ClickHouse 应用开发指导......117
3.1 ClickHouse 组件使用规范......117
3.2 ClickHouse 表引擎概述......119
3.3 SQL 语法参考......129
3.3.1 数据类型......129
3.3.2 CREATE DATABASE......132
3.3.3 CREATE TABLE......132
3.3.4 DESC 查询表结构......135
3.3.5 CREATE VIEW......135
3.3.6 CREATE MATERIALIZED VIEW......136
3.3.7 INSERT INTO......138
3.3.8 SELETC......138
3.3.9 ALTER TABLE 修改表结构......139
3.3.10 DROP 删除表......139
3.3.11 SHOW 显示数据库和表信息......140
3.4 数据迁移同步......140
3.4.1 数据导入导出......140
3.4.2 ClickHouse 访问 RDS MySQL 服务......143
3.5 开发程序......145
3.5.1 典型场景说明......145
3.5.2 开发思路......145
3.5.3 准备开发和运行环境......145
3.5.4 配置并导入样例工程......146
3.5.5 样例代码说明......148

3.5.5.1 设置属性 148
3.5.5.2 建立连接 148
3.5.5.3 创建库 149
3.5.5.4 创建表 149
3.5.5.5 插入数据 149
3.5.5.6 查询数据 150
3.5.5.7 删除表 150
3.6 调测程序 151
3.7 开发 ClickHouse 冷热分离应用 153
3.7.1 应用背景 153
3.7.2 典型场景说明 153
3.7.3 开发思路 154
3.7.4 样例代码 154

1 HBase 应用开发指导

1.1 开发流程

本文档主要介绍在CloudTable集群模式下如何调用HBase开源接口进行Java应用程序的开发。

开发流程中各阶段的说明如**图1-1**和**表1-1**所示。

版权所有 © 华为云计算技术有限公司

图 1-1 应用程序开发流程

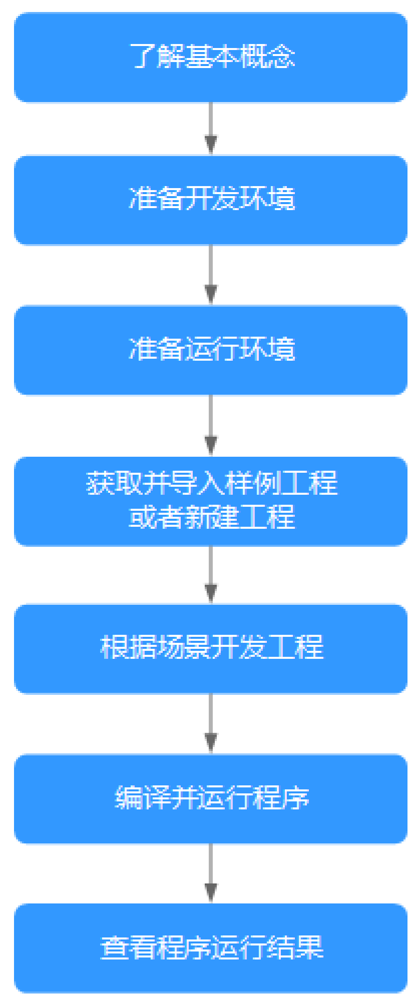


表 1-1 应用开发的流程说明

阶段	说明	参考文档
了解基本概念	在开始开发应用前，需要了解HBase的基本概念，了解场景需求，设计表等。	**HBase**
准备开发环境	HBase应用程序当前推荐使用Java语言进行开发。可使用Eclipse工具。	**开发环境简介**
准备运行环境	应用程序的运行环境即客户端环境，请根据指导完成客户端的安装和配置。	**准备Windows运行环境**
准备工程	CloudTable为用户提供了不同场景下的样例程序，您可以导入样例工程进行程序学习。或者您可以根据指导，新建一个工程。	**下载样例工程** **配置并导入工程**
根据场景开发工程	提供了Java语言的样例工程，包含从建表、写入到删除表全流程的样例工程。	**开发HBase应用**

阶段	说明	参考文档
编译并运行程序	指导用户将开发好的程序编译并提交运行。	**编译并运行程序** **安装客户端时编译并运行程序** 或 **未安装客户端时编译并运行程序**
查看程序运行结果	程序运行结果会写在用户指定的路径下。用户还可以通过UI查看应用运行情况。	● 在Windows环境中：**查看调测结果** ● 在Linux环境中：**查看调测结果**

1.2 准备开发环境

1.2.1 开发环境简介

在进行二次开发时，要准备的开发环境如**表1-2**所示。

表 1-2 开发环境

准备项	说明
操作系统	Windows系统，推荐Windows 7及以上版本。
安装JDK	开发环境的基本配置。版本要求：1.7或者1.8。考虑到后续版本的兼容性，强烈推荐使用1.8。 **说明** 基于安全考虑，CloudTable服务只支持TLS 1.1和TLS 1.2加密协议，IBM JDK默认TLS只支持1.0，如果使用IBM JDK，请配置启动参数“com.ibm.jsse2.overrideDefaultTLS”为“true”，设置后可以同时支持TLS1.0/1.1/1.2。详情请参见IBM官方网站的相关说明。
安装和配置Eclipse	用于开发CloudTable应用程序的工具。
网络	确保开发环境或客户端与表格存储服务主机在网络上互通。

1.2.2 准备运行环境

1.2.2.1 准备 Windows 运行环境

操作场景

CloudTable应用开发的运行环境可以部署在Windows环境下。按照如下操作完成运行环境准备。

版权所有 © 华为云计算技术有限公司

操作步骤

步骤1 确认CloudTable集群已经安装，并正常运行。

步骤2 准备Windows弹性云服务器。

具体操作请参见准备弹性云服务器章节。

步骤3 请在Windows的弹性云服务器上安装JDK1.7及以上版本，强烈推荐使用JDK1.8及以上版本，并且安装Eclipse，Eclipse使用JDK1.7及以上的版本。

说明

- 如果使用IBM JDK，请确保Eclipse中的JDK配置为IBM JDK。
- 如果使用Oracle JDK，请确保Eclipse中的JDK配置为Oracle JDK。
- 不同的Eclipse不要使用相同的workspace和相同路径下的示例工程。

----结束

1.2.3 下载样例工程

前提条件

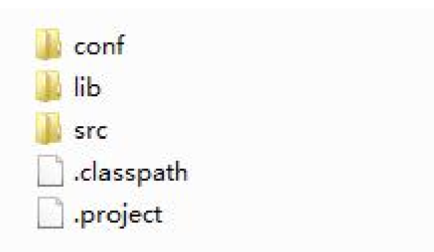
确认表格存储服务已经安装，并正常运行。

下载样例工程

步骤1 下载样例代码工程。

步骤2 下载完成后，将样例代码工程安装包解压到本地，得到一个Eclipse的JAVA工程。如图1-2所示。

图 1-2 样例代码工程目录结构



----结束

Maven 配置

样例工程中已经包含了hbase的客户端jar包，也可以替换成开源的HBase jar包访问表格存储服务，支持1.X.X版本以上的开源HBase API。如果需要在应用中引入表格存储服务的HBase jar包，可以在Maven中配置如下依赖。

```
<dependencies>
  <dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>1.3.1.0305-cloudtable</version>
  </dependency>
```

```
    <dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-common</artifactId>
        <version>1.3.1.0305-cloudtable</version>
    </dependency>
</dependencies>
```

使用如下任意一种配置方法配置镜像仓地址（本文提供了如下两种配置方法）。

- **配置方法一：**

  在setting.xml配置文件的mirrors节点中添加开源镜像仓地址：

```
<mirror>
    <id>repo2</id>
    <mirrorOf>central</mirrorOf>
    <url>https://repo1.maven.org/maven2/</url>
</mirror>
```

  在setting.xml配置文件的profiles节点中添加如下镜像仓地址：

```
<profile>
    <id>xxxcloudsdk</id>
    <repositories>
        <repository>
            <id>xxxcloudsdk</id>
            <url>https://repo.xxxcloud.com/repository/maven/xxxcloudsdk/</url>
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
    </repositories>
</profile>
```

  在setting.xml配置文件的activeProfiles节点中添加如下镜像仓地址：

```
<activeProfile>xxxcloudsdk</activeProfile>
```

  说明

  华为云开源镜像站不提供第三方开源jar包下载，请配置华为云开源镜像后，额外配置第三方Maven镜像仓库地址。

- **配置方法二：**

  在二次开发工程样例工程中的pom.xml文件添加如下镜像仓地址：

```
    <repositories>
        <repository>
            <id>xxxcloudsdk</id>
            <url>https://mirrors.xxxcloud.com/repository/maven/xxxcloudsdk/</url>
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>

        <repository>
            <id>central</id>
            <name>Mavn Centreal</name>
            <url>https://repo1.maven.org/maven2/</url>
        </repository>
    </repositories>
```

1.2.4 配置并导入工程

背景信息

将CloudTable样例代码工程导入到Eclipse，就可以开始CloudTable应用开发样例的学习。

 版权所有 © 华为云计算技术有限公司

前提条件

运行环境已经正确配置，请参见**准备Windows运行环境**。

操作步骤

步骤1 把样例工程上传到Windows开发环境中。样例工程的获取方法请参见**下载样例工程**。

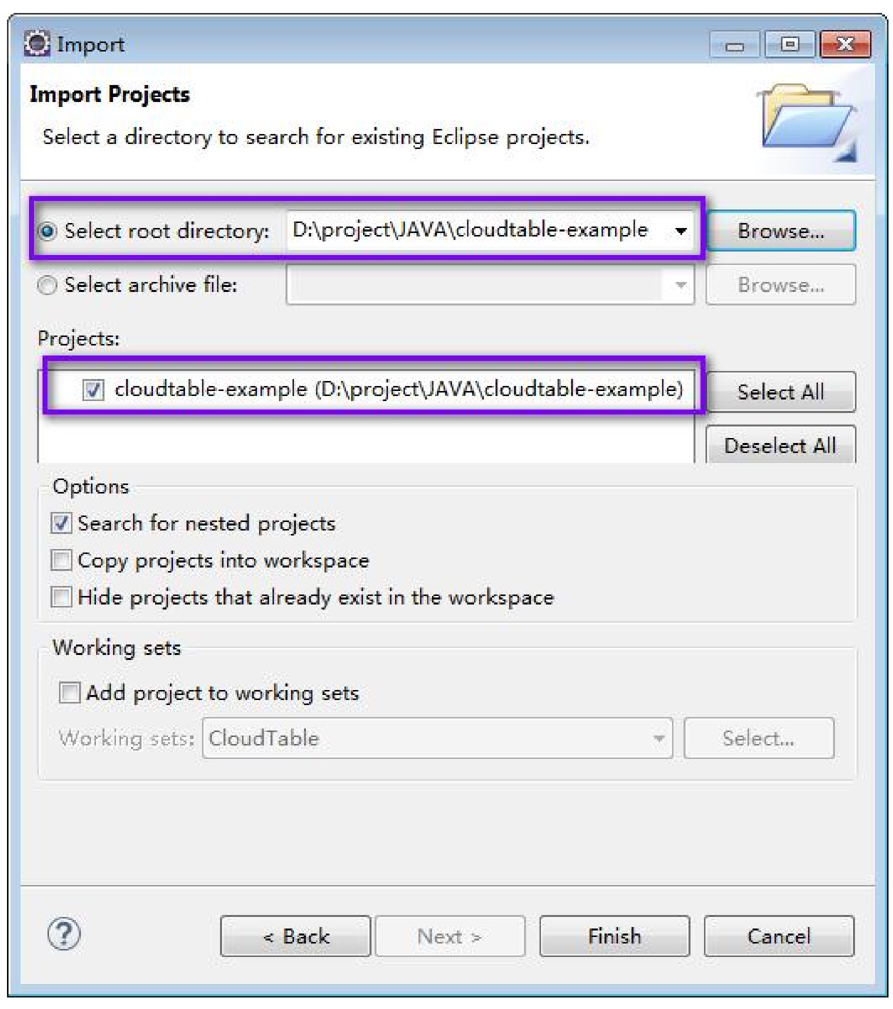
步骤2 在应用开发环境中，导入样例工程到Eclipse开发环境。

1. 选择“File > Import > General > Existing Projects into Workspace > Next > Browse”。

   显示“浏览文件夹”对话框。如**图1-3**所示。

2. 选择样例工程文件夹，单击“Finish”。

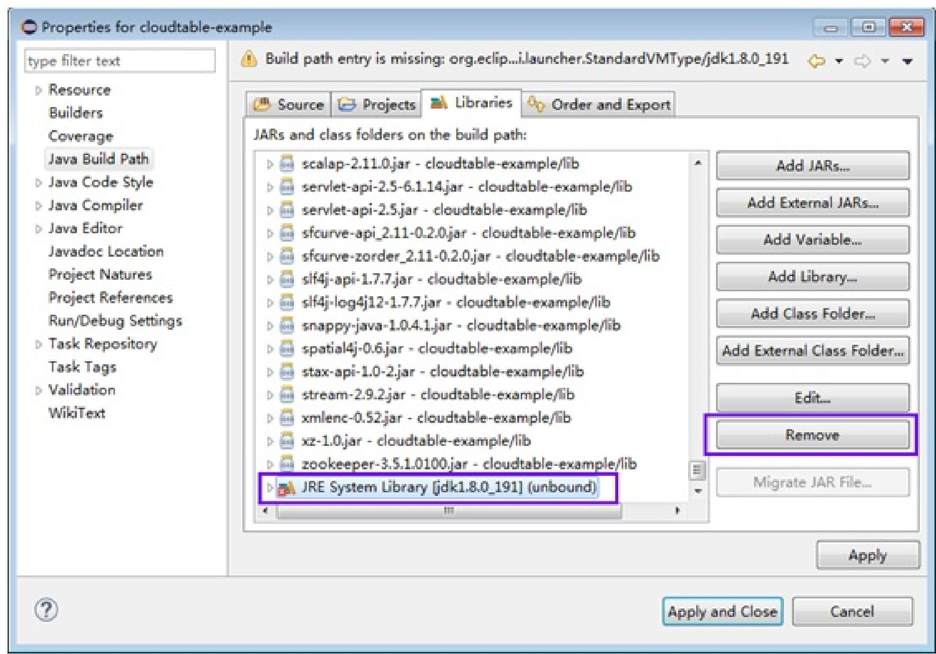
图 1-3 导入样例工程



步骤3 右键单击**cloudtable-example**工程，在弹出的右键菜单中单击“Properties”，弹出“Properties for cloudtable-example”窗口。

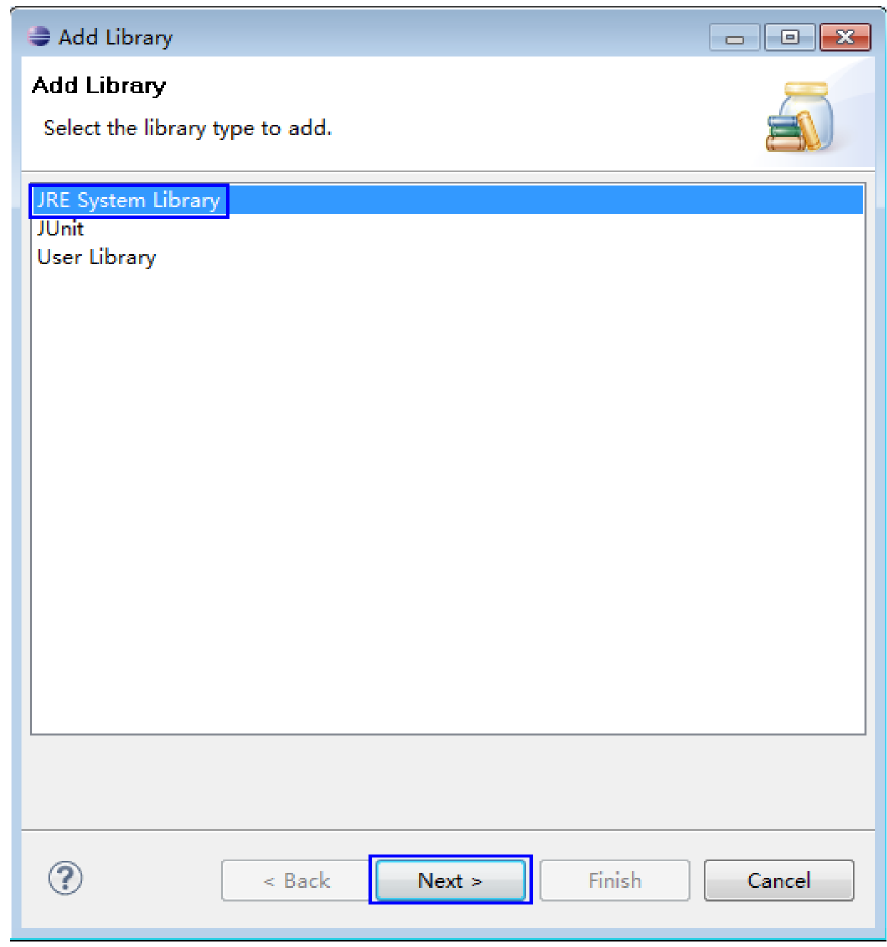
1. 在左边导航上选择“Java Build Path”，单击右侧“Libraries”标签页，按**图1-4**所示将报错的JDK选中后，单击“Remove”删除。

图 1-4 删除报错的 JDK



2. 单击“Add Library...”按钮，如图1-5所示，在弹出的窗口中选择“JRE System Library”。

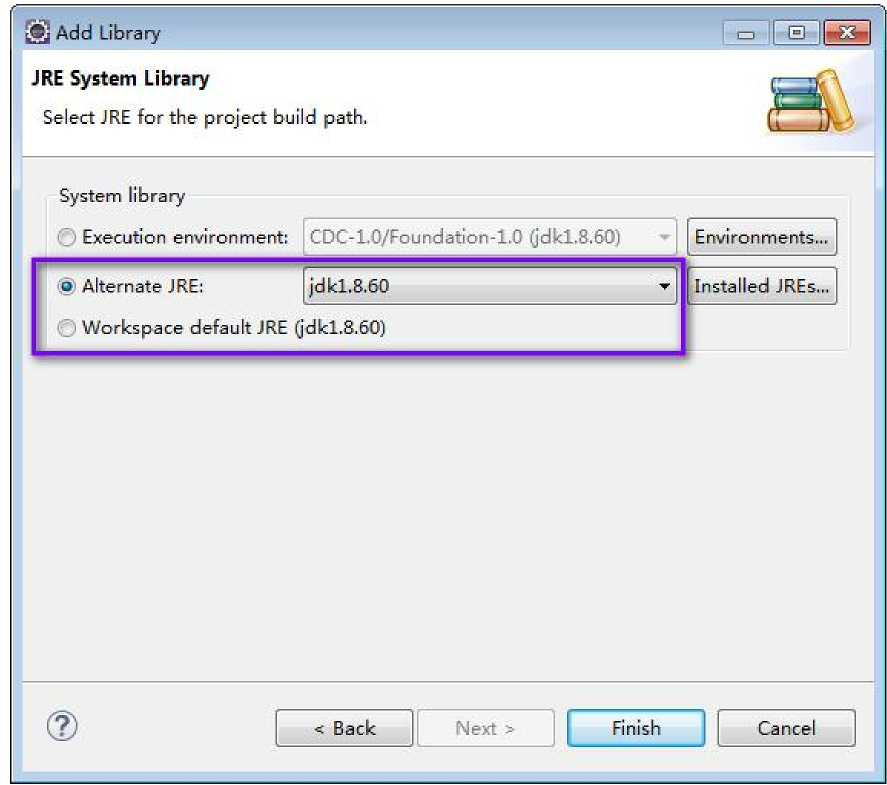
图 1-5 选择增加的 library 类型



3. 在“Add Library”页面中，通过“Alternate JRE”或“Workspace default JRE”选项选择JDK版本。如**图1-6**所示，选中“Alternate JRE”后，选择JDK版本。

版权所有 © 华为云计算技术有限公司

图 **1-6** 选择 JRE



4. 单击“Finish”关闭窗口完成设置。

步骤4 设置Eclipse的文本文件编码格式，解决乱码显示问题。

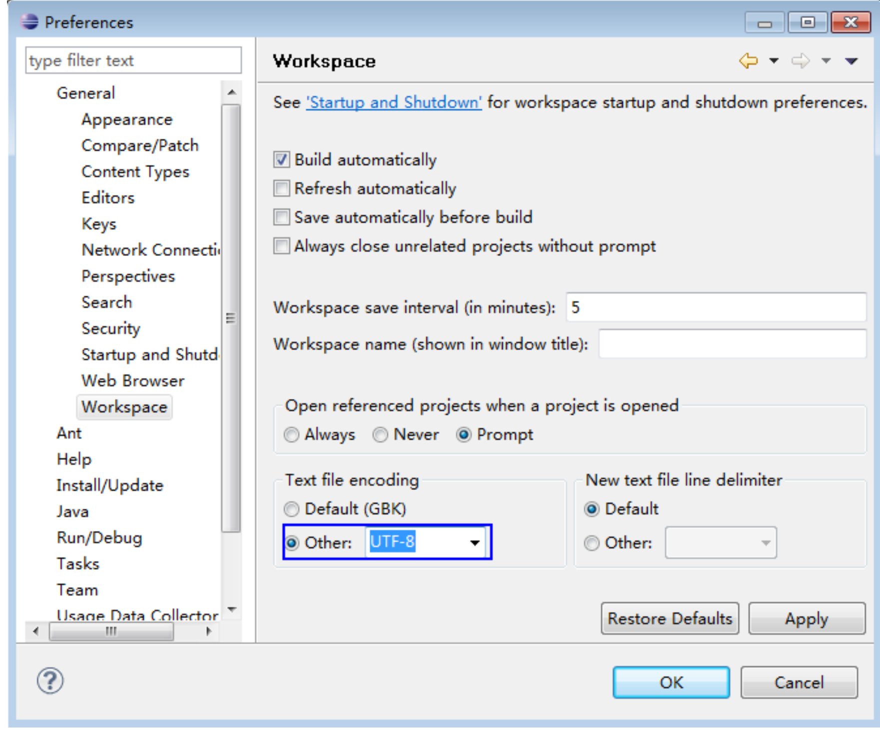
1. 在Eclipse的菜单栏中，选择“Window > Preferences”。

   弹出“Preferences”窗口。

2. 在左边导航上选择“General > Workspace”，在“Text file encoding”区域，选中“Other”，并设置参数值为“UTF-8”，单击“Apply”后，单击“OK”，如**图1-7**所示。

版权所有 © 华为云计算技术有限公司

图 1-7 设置 Eclipse 的编码格式



步骤5 打开样例工程中的“conf/hbase-site.xml”文件，修改“hbase.zookeeper.quorum”的值为正确的Zookeeper地址。

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

其中：*value*中的值为ZooKeeper集群的域名。登录表格存储服务管理控制台，在左侧导航树单击集群管理，然后在集群列表中找到所需要的集群，并获取相应的“ZK链接地址（内网）”。

----结束

1.3 开发 HBase 应用

1.3.1 典型场景说明

通过典型场景，我们可以快速学习和掌握HBase的开发过程，并且对关键的接口函数有所了解。

场景说明

假定用户开发一个应用程序，用于管理企业中的使用A业务的用户信息，如**表1-3**所示，A业务操作流程如下：

- 创建用户信息表。
- 在用户信息中新增用户的学历、职称等信息。
- 根据用户编号查询用户姓名和地址。

版权所有 © 华为云计算技术有限公司

- 根据用户姓名进行查询。
- 查询年龄段在[20–29]之间的用户信息。
- 数据统计，统计用户信息表的人员数、年龄最大值、年龄最小值、平均年龄。
- 用户销户，删除用户信息表中该用户的数据。
- A业务结束后，删除用户信息表。

表 1-3 用户信息

编号	姓名	性别	年龄	地址
12005000201	A	Male	19	Shenzhen, Guangdong
12005000202	B	Female	23	Shijiazhuang, Hebei
12005000203	C	Male	26	Ningbo, Zhejiang
12005000204	D	Male	18	Xiangyang, Hubei
12005000205	E	Female	21	Shangrao, Jiangxi
12005000206	F	Male	32	Zhuzhou, Hunan
12005000207	G	Female	29	Nanyang, Henan
12005000208	H	Female	30	Kaixian, Chongqing
12005000209	I	Male	26	Weinan, Shaanxi
12005000210	J	Male	25	Dalian, Liaoning

数据规划

合理地设计表结构、行键、列名能充分利用HBase的优势。本样例工程以唯一编号作为RowKey，列都存储在info列族中。

1.3.2 开发思路

功能分解

根据上述的业务场景进行功能分解，需要开发的功能点如**表1-4**所示。

版权所有 © 华为云计算技术有限公司

表 1-4 在 HBase 中开发的功能

序号	步骤	代码实现
1	根据典型场景说明中的信息创建表。	请参见创建表。
2	导入用户数据。	请参见插入数据。
3	增加“教育信息”列族，在用户信息中新增用户的学历、职称等信息。	请参见修改表。
4	根据用户编号查询用户姓名和地址。	请参见使用Get读取数据。
5	根据用户姓名进行查询。	请参见使用过滤器Filter。
6	用户销户，删除用户信息表中该用户的数据。	请参见删除数据。
7	A业务结束后，删除用户信息表。	请参见删除表。

关键设计原则

HBase是以RowKey为字典排序的分布式数据库系统，RowKey的设计对性能影响很大，具体的RowKey设计请考虑与业务结合。

1.3.3 样例代码说明

1.3.3.1 配置参数

步骤1 执行样例代码前，必须在hbase-site.xml配置文件中，配置正确的ZooKeeper集群的地址。

配置项如下：

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

其中：*value*中的值为ZooKeeper集群的域名。登录表格存储服务管理控制台，在左侧导航树单击集群管理，然后在集群列表中找到所需要的集群，并获取相应的“ZK链接地址（内网）”。

----结束

1.3.3.2 创建 Configuration

功能介绍

HBase通过加载配置文件来获取配置项。

说明

1. 加载配置文件是一个比较耗时的操作，如非必要，请尽量使用同一个Configuration对象。
2. 样例代码未考虑多线程同步的问题，如有需要，请自行增加。其它样例代码也一样，不再一一进行说明。

代码样例

下面代码片段在com.huaweicloudtable.hbase.examples包中。

```
private static void init() throws IOException {
  // Default load from conf directory
  conf = HBaseConfiguration.create(); // 注[1]
  String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
  Path hbaseSite = new Path(userdir + "hbase-site.xml");
  if (new File(hbaseSite.toString()).exists()) {
    conf.addResource(hbaseSite);
  }
}
```

注意事项

- 注[1] 如果配置文件目录conf已经加入classpath路径中，那么后面的加载指定配置文件的代码可以不执行。

1.3.3.3 创建 Connection

功能介绍

HBase通过ConnectionFactory.createConnection(configuration)方法创建Connection对象。传递的参数为上一步创建的Configuration。

Connection封装了底层与各实际服务器的连接以及与ZooKeeper的连接。Connection通过ConnectionFactory类实例化。创建Connection是重量级操作，而且Connection是线程安全的，因此，多个客户端线程可以共享一个Connection。

典型的用法，一个客户端程序共享一个单独的Connection，每一个线程获取自己的Admin或Table实例，然后调用Admin对象或Table对象提供的操作接口。不建议缓存或者池化Table、Admin。Connection的生命周期由调用者维护，调用者通过调用close()，释放资源。

说明

建议业务代码连接同一个CloudTable集群时，多线程创建并复用同一个Connection，不必每个线程都创建各自Connection。Connection是连接CloudTable集群的连接器，创建过多连接会加重Zookeeper负载，并损耗业务读写性能。

代码样例

以下代码片段是创建Connection对象的示例：

```
private TableName tableName = null;
private Connection conn = null;

public HBaseSample(Configuration conf) throws IOException {
  this.tableName = TableName.valueOf("hbase_sample_table");
  this.conn = ConnectionFactory.createConnection(conf);
}
```

1.3.3.4 创建表

功能简介

HBase通过org.apache.hadoop.hbase.client.Admin对象的createTable方法来创建表，并指定表名、列族名。创建表有两种方式（强烈建议采用预分Region建表方式）：

- 快速建表，即创建表后整张表只有一个Region，随着数据量的增加会自动分裂成多个Region。
- 预分Region建表，即创建表时预先分配多个Region，此种方法建表可以提高写入大量数据初期的数据写入速度。

说明

表名以及列族名不能包含特殊字符，可以由字母、数字以及下划线组成。

代码样例

```
public void testCreateTable() {
  LOG.info("Entering testCreateTable.");

  // Specify the table descriptor.
  HTableDescriptor htd = new HTableDescriptor(tableName); // (1)

  // Set the column family name to info.
  HColumnDescriptor hcd = new HColumnDescriptor("info"); // (2)

  // Set data encoding methods. HBase provides DIFF,FAST_DIFF,PREFIX
  // and PREFIX_TREE
  hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);  // 注[1]

  // Set compression methods, HBase provides two default compression
  // methods:GZ and SNAPPY
  // GZ has the highest compression rate,but low compression and
  // decompression efficiency,fit for cold data
  // SNAPPY has low compression rate, but high compression and
  // decompression efficiency,fit for hot data.
  // it is advised to use SANPPY
  hcd.setCompressionType(Compression.Algorithm.SNAPPY);
  htd.addFamily(hcd); // (3)

  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin(); // (4)
    if (!admin.tableExists(tableName)) {
      LOG.info("Creating table...");
      admin.createTable(htd); // 注[2] (5)
      LOG.info(admin.getClusterStatus());
      LOG.info(admin.listNamespaceDescriptors());
      LOG.info("Table created successfully.");
    } else {
      LOG.warn("table already exists");
    }
  } catch (IOException e) {
    LOG.error("Create table failed.", e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
```

```
  }
  LOG.info("Exiting testCreateTable.");
}
```

解释

（1）创建表描述符。

（2）创建列族描述符。

（3）添加列族描述符到表描述符中。

（4）获取Admin对象，Admin提供了建表、创建列族、检查表是否存在、修改表结构和列族结构以及删除表等功能。

（5）调用Admin的建表方法。

注意事项

- 注[1] 可以设置列族的压缩方式，代码片段如下：

```
//设置编码算法，HBase提供了DIFF，FAST_DIFF，PREFIX和PREFIX_TREE四种编码算法
 hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
 //设置文件压缩方式，HBase默认提供了GZ和SNAPPY两种压缩算法
 //其中GZ的压缩率高，但压缩和解压性能低，适用于冷数据
 //SNAPPY压缩率低，但压缩解压性能高，适用于热数据
 //建议默认开启SNAPPY压缩
 hcd.setCompressionType(Compression.Algorithm.SNAPPY);
```

- 注[2] 可以通过指定起始和结束RowKey，或者通过RowKey数组预分Region两种方式建表，代码片段如下：

```
// 创建一个预划分region的表
 byte[][] splits = new byte[4][];
 splits[0] = Bytes.toBytes("A");
 splits[1] = Bytes.toBytes("H");
 splits[2] = Bytes.toBytes("O");
 splits[3] = Bytes.toBytes("U");
 admin.createTable(htd, splits);
```

1.3.3.5 删除表

功能简介

HBase通过org.apache.hadoop.hbase.client.Admin的deleteTable方法来删除表。

代码样例

```
public void dropTable() {
  LOG.info("Entering dropTable.");
  Admin admin = null;
  try {
    admin = conn.getAdmin();
    if (admin.tableExists(tableName)) {
      // Disable the table before deleting it.
      admin.disableTable(tableName);
      // Delete table.
      admin.deleteTable(tableName);//注[1]
    }
    LOG.info("Drop table successfully.");
  } catch (IOException e) {
    LOG.error("Drop table failed " ,e);
  } finally {
    if (admin != null) {
```

```
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
      }
    }
  }
  LOG.info("Exiting dropTable.");
}
```

注意事项

注[1] 只有在调用disableTable接口后， 再调用deleteTable接口才能将表删除成功。因此，deleteTable常与disableTable，enableTable，tableExists，isTableEnabled，isTableDisabled结合在一起使用。

1.3.3.6 修改表

功能简介

HBase通过org.apache.hadoop.hbase.client.Admin的modifyTable方法修改表信息。

代码样例

```
public void testModifyTable() {
  LOG.info("Entering testModifyTable.");

  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("education");
  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin();
    // Obtain the table descriptor.
    HTableDescriptor htd = admin.getTableDescriptor(tableName);
    // Check whether the column family is specified before modification.
    if (!htd.hasFamily(familyName)) {
      // Create the column descriptor.
      HColumnDescriptor hcd = new HColumnDescriptor(familyName);
      htd.addFamily(hcd);
      // Disable the table to get the table offline before modifying
      // the table.
      admin.disableTable(tableName);
      // Submit a modifyTable request.
      admin.modifyTable(tableName, htd);  //注[1]
      // Enable the table to get the table online after modifying the
      // table.
      admin.enableTable(tableName);
    }
    LOG.info("Modify table successfully.");
  } catch (IOException e) {
    LOG.error("Modify table failed " ,e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
      }
    }
  }
  LOG.info("Exiting testModifyTable.");
}
```

 版权所有 © 华为云计算技术有限公司

注意事项

注[1] 只有在调用disableTable接口后， 再调用modifyTable接口才能将表修改成功。之后，请调用enableTable接口重新启用表。

1.3.3.7 插入数据

功能简介

HBase是一个面向列的数据库，一行数据，可能对应多个列族，而一个列族又可以对应多个列。

通常，写入数据的时候，我们需要指定要写入的列（含列族名称和列名称）。

HBase通过HTable的put方法来Put数据，可以是一行数据也可以是数据集。

代码样例

```
public void testPut() {
  LOG.info("Entering testPut.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifiers = { Bytes.toBytes("name"), Bytes.toBytes("gender"),
      Bytes.toBytes("age"), Bytes.toBytes("address") };
  Table table = null;
  try {
    // Instantiate an HTable object.
    table = conn.getTable(tableName);
    List<Put> puts = new ArrayList<Put>();
    // Instantiate a Put object.
    Put put = new Put(Bytes.toBytes("012005000201"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("A"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("19"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shenzhen, Guangdong"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000202"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("B"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("23"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shijiazhuang, Hebei"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000203"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("C"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Ningbo, Zhejiang"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000204"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("D"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("18"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Xiangyang, Hubei"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000205"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("E"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("21"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shangrao, Jiangxi"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000206"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("F"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("32"));
```

```
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Zhuzhou, Hunan"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000207"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("G"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("29"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Nanyang, Henan"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000208"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("H"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("30"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Kaixian, Chongqing"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000209"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("I"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Weinan, Shaanxi"));
    puts.add(put);
    put = new Put(Bytes.toBytes("012005000210"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("J"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
    put.addColumn(familyName, qualifiers[2], Bytes.toBytes("25"));
    put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Dalian, Liaoning"));
    puts.add(put);
    // Submit a put request.
    table.put(puts);

    LOG.info("Put successfully.");
  } catch (IOException e) {
    LOG.error("Put failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testPut.");
}
```

注意事项

不允许多个线程在同一时间共用同一个HTable实例。HTable是一个非线程安全类，因此，同一个HTable实例，不应该被多个线程同时使用，否则可能会带来并发问题。

1.3.3.8 删除数据

功能简介

HBase通过Table实例的delete方法来Delete数据，可以是一行数据也可以是数据集。

具体删除方法根据用户使用场景选取。

代码样例

```
public void testDelete() {
  LOG.info("Entering testDelete.");

  byte[] rowKey = Bytes.toBytes("012005000201");
```

```
  Table table = null;
  try {
    // Instantiate an HTable object.
    table = conn.getTable(tableName);

    // Instantiate a Delete object.
    Delete delete = new Delete(rowKey);

    // Submit a delete request.
    table.delete(delete);

    LOG.info("Delete table successfully.");
  } catch (IOException e) {
    LOG.error("Delete table failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testDelete.");
}
```

1.3.3.9 使用 Get 读取数据

功能简介

要从表中读取一条数据，首先需要实例化该表对应的Table实例，然后创建一个Get对象。

可以为Get对象设定参数值，如列族的名称和列的名称。

查询到的行数据存储在Result对象中，Result中可以存储多个Cell。

代码样例

```
public void testGet() {
  LOG.info("Entering testGet.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifier = { Bytes.toBytes("name"), Bytes.toBytes("address") };
  // Specify RowKey.
  byte[] rowKey = Bytes.toBytes("012005000201");
  Table table = null;
  try {
    // Create the Table instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Get get = new Get(rowKey);
    // Set the column family name and column name.
    get.addColumn(familyName, qualifier[0]);
    get.addColumn(familyName, qualifier[1]);
    // Submit a get request.
    Result result = table.get(get);
    // Print query results.
    for (Cell cell : result.rawCells()) {
      LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
          + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
          + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
          + Bytes.toString(CellUtil.cloneValue(cell)));
    }
```


版权所有 © 华为云计算技术有限公司

```
      LOG.info("Get data successfully.");
    } catch (IOException e) {
      LOG.error("Get data failed " ,e);
    } finally {
      if (table != null) {
        try {
          // Close the HTable object.
          table.close();
        } catch (IOException e) {
          LOG.error("Close table failed " ,e);
        }
      }
    }
    LOG.info("Exiting testGet.");
}
```

1.3.3.10 使用 Scan 读取数据

功能简介

要从表中读取数据，首先需要实例化该表对应的Table实例，然后创建一个Scan对象，并针对查询条件设置Scan对象的参数值，为了提高查询效率，最好指定StartRow和StopRow。查询结果的多行数据保存在ResultScanner对象中，每行数据以Result对象形式存储，Result中存储了多个Cell。

代码样例

```
public void testScanData() {
  LOG.info("Entering testScanData.");
  Table table = null;
  // Instantiate a ResultScanner object.
  ResultScanner rScanner = null;
  try {
    // Create the Configuration instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
    // Set the cache size.
    scan.setCaching(1000);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Scan data successfully.");
  } catch (IOException e) {
    LOG.error("Scan data failed " ,e);
  } finally {
    if (rScanner != null) {
      // Close the scanner object.
      rScanner.close();
    }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
```

```
    }
  }
  LOG.info("Exiting testScanData.");
}
```

注意事项

1. 建议Scan时指定StartRow和StopRow，一个有确切范围的Scan，性能会更好些。
2. 可以设置Batch和Caching关键参数。
   - Batch
     使用Scan调用next接口每次最大返回的记录数，与一次读取的列数有关。
   - Caching
     RPC请求返回next记录的最大数量，该参数与一次RPC获取的行数有关。

1.3.3.11 使用过滤器 Filter

功能简介

HBase Filter主要在Scan和Get过程中进行数据过滤，通过设置一些过滤条件来实现，如设置RowKey、列名或者列值的过滤条件。

具体过滤条件根据用户使用场景选取。

代码样例

```
public void testSingleColumnValueFilter() {
  LOG.info("Entering testSingleColumnValueFilter.");
  Table table = null;
  ResultScanner rScanner = null;

  try {
    table = conn.getTable(tableName);
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
    // Set the filter criteria.
    SingleColumnValueFilter filter = new SingleColumnValueFilter(
        Bytes.toBytes("info"), Bytes.toBytes("name"), CompareOp.EQUAL,
        Bytes.toBytes("I"));
    scan.setFilter(filter);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Single column value filter successfully.");
  } catch (IOException e) {
    LOG.error("Single column value filter failed " ,e);
  } finally {
      if (rScanner != null) {
          // Close the scanner object.
          rScanner.close();
        }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
```

 版权所有 © 华为云计算技术有限公司

```
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testSingleColumnValueFilter.");
}
```

1.4 开发 HBase 冷热分离应用

1.4.1 应用背景

CloudTable HBase支持冷热数据分离特性。通过该特性，您可以将冷热数据分别存储在不同类型的存储介质中，以降低存储成本。

在海量大数据场景下，表中的部分业务数据随着时间的推移仅作为归档数据或者访问频率很低，同时这部分历史数据体量非常大，比如订单数据或者监控数据，如果降低这部分数据的存储成本将会极大的节省企业的成本。

1.4.2 典型场景说明

通过典型场景，我们可以快速学习和掌握HBase冷热分离的开发过程，并且对关键的接口函数有所了解。

场景说明

假定用户开发一个应用程序，用于实时记录和查询城市的气象信息，记录数据如下表：

表 1-5 原始数据

城市	区域	时间	温度	湿度
Shenzhen	Longgang	2017/7/1 00:00:00	28	54
Shenzhen	Longgang	2017/7/1 01:00:00	27	53
Shenzhen	Longgang	2017/7/1 02:00:00	27	52
Shenzhen	Longgang	2017/7/1 03:00:00	27	51
Shenzhen	Longgang	2017/7/1 04:00:00	27	50
Shenzhen	Longgang	2017/7/1 05:00:00	27	49
Shenzhen	Longgang	2017/7/1 06:00:00	27	48
Shenzhen	Longgang	2017/7/1 07:00:00	27	46
Shenzhen	Longgang	2017/7/1 08:00:00	29	46
Shenzhen	Longgang	2017/7/1 09:00:00	30	48
Shenzhen	Longgang	2017/7/1 10:00:00	32	48
Shenzhen	Longgang	2017/7/1 11:00:00	32	49

城市	区域	时间	温度	湿度
Shenzhen	Longgang	2017/7/1 12:00:00	33	49
Shenzhen	Longgang	2017/7/1 13:00:00	33	50
Shenzhen	Longgang	2017/7/1 14:00:00	32	50
Shenzhen	Longgang	2017/7/1 15:00:00	32	50
Shenzhen	Longgang	2017/7/1 16:00:00	31	51
Shenzhen	Longgang	2017/7/1 17:00:00	30	51
Shenzhen	Longgang	2017/7/1 18:00:00	30	51
Shenzhen	Longgang	2017/7/1 19:00:00	29	51
Shenzhen	Longgang	2017/7/1 20:00:00	29	52
Shenzhen	Longgang	2017/7/1 21:00:00	29	53
Shenzhen	Longgang	2017/7/1 22:00:00	28	54
Shenzhen	Longgang	2017/7/1 23:00:00	28	54
Shenzhen	Longgang	2017/7/2 00:00:00	28	54
Shenzhen	Longgang	2017/7/2 01:00:00	27	53
Shenzhen	Longgang	2017/7/2 02:00:00	27	52
Shenzhen	Longgang	2017/7/2 03:00:00	27	51
Shenzhen	Longgang	2017/7/2 04:00:00	27	50
Shenzhen	Longgang	2017/7/2 05:00:00	27	49
Shenzhen	Longgang	2017/7/2 06:00:00	27	48
Shenzhen	Longgang	2017/7/2 07:00:00	27	46
Shenzhen	Longgang	2017/7/2 08:00:00	29	46
Shenzhen	Longgang	2017/7/2 09:00:00	30	48
Shenzhen	Longgang	2017/7/2 10:00:00	32	48
Shenzhen	Longgang	2017/7/2 11:00:00	32	49
Shenzhen	Longgang	2017/7/2 12:00:00	33	49
Shenzhen	Longgang	2017/7/2 13:00:00	33	50
Shenzhen	Longgang	2017/7/2 14:00:00	32	50
Shenzhen	Longgang	2017/7/2 15:00:00	32	50
Shenzhen	Longgang	2017/7/2 16:00:00	31	51
Shenzhen	Longgang	2017/7/2 17:00:00	30	51

版权所有 © 华为云计算技术有限公司

城市	区域	时间	温度	湿度
Shenzhen	Longgang	2017/7/2 18:00:00	30	51
Shenzhen	Longgang	2017/7/2 19:00:00	29	51
Shenzhen	Longgang	2017/7/2 20:00:00	29	52
Shenzhen	Longgang	2017/7/2 21:00:00	29	53
Shenzhen	Longgang	2017/7/2 22:00:00	28	54
Shenzhen	Longgang	2017/7/2 23:00:00	28	54

数据规划

合理地设计表结构、行键、列名能充分利用HBase的优势。本样例工程以城市+区域+时间作为RowKey，列都存储在info列族中。

当天整点写入数据，同时一天前数据查询频率较低，节省存储空间设置冷热分离，将一天前数据自动归档到冷存储。

1.4.3 开发思路

功能分解

根据上述的业务场景进行功能分解，需要开发的功能点如**表1-6**所示。

表 1-6 在 HBase 中开发冷热分离的功能

序号	步骤	代码实现
1	根据**典型场景说明**中的信息创建表。	请参见**创建表**。
2	写入数据。	请参见**插入数据**。
4	根据城市、区域、时间查询温度和湿度。	请参见**使用Get读取数据**。
5	根据城市、局域、时间范围进行查询。	请参见**使用Scan读取数据**。

关键设计原则

HBase是以RowKey为字典排序的分布式数据库系统，RowKey的设计对性能影响很大，具体的RowKey设计请考虑与业务结合。

1.4.4 样例代码说明

1.4.4.1 配置参数

步骤1 执行样例代码前，必须在hbase-site.xml配置文件中，配置正确的ZooKeeper集群的地址。配置项如下：

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

其中：*value*中的值为ZooKeeper集群的域名。登录表格存储服务管理控制台，在左侧导航树单击集群管理，然后在集群列表中找到所需要的集群，并获取相应的“ZK链接地址”。

----结束

1.4.4.2 创建 Configuration

功能介绍

HBase通过加载配置文件来获取配置项。

说明

- 加载配置文件是一个比较耗时的操作，如非必要，请尽量使用同一个Configuration对象。
- 样例代码未考虑多线程同步的问题，如有需要，请自行增加。其它样例代码也一样，不再进行说明。

代码样例

下面代码片段在com.huawei.cloudtable.hbase.examples.coldhotexample包中。

```
private static void init() throws IOException {
  // Default load from conf directory
  conf = HBaseConfiguration.create(); // 注[1]
  String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
  Path hbaseSite = new Path(userdir + "hbase-site.xml");
  if (new File(hbaseSite.toString()).exists()) {
    conf.addResource(hbaseSite);
  }
}
```

注意事项

- 注[1] 如果配置文件目录conf已经加入classpath路径中，那么后面的加载指定配置文件的代码可以不执行。

  说明

  注[1]指的是代码样例里面的“conf = HBaseConfiguration.create(); // 注[1]”。

1.4.4.3 创建 Connection

功能介绍

HBase通过ConnectionFactory.createConnection(configuration)方法创建Connection对象。传递的参数为上一步创建的Configuration。

版权所有 © 华为云计算技术有限公司

Connection封装了底层与各实际服务器的连接以及与ZooKeeper的连接。Connection通过ConnectionFactory类实例化。创建Connection是重量级操作，而且Connection是线程安全的，因此，多个客户端线程可以共享一个Connection。

典型的用法，一个客户端程序共享一个单独的Connection，每一个线程获取自己的Admin或Table实例，然后调用Admin对象或Table对象提供的操作接口。不建议缓存或者池化Table、Admin。Connection的生命周期由调用者维护，调用者通过调用close（），释放资源。

说明

建议业务代码连接同一个CloudTable集群时，多线程创建并复用同一个Connection，不必每个线程都创建各自Connection。Connection是连接CloudTable集群的连接器，创建过多连接会加重Zookeeper负载，并损耗业务读写性能。

代码样例

以下代码片段是创建Connection对象的示例：

```
private TableName tableName = null;
private Connection conn = null;

public HBaseSample(Configuration conf) throws IOException {
  this.tableName = TableName.valueOf("hbase_sample_table");
  this.conn = ConnectionFactory.createConnection(conf);
}
```

1.4.4.4 创建表

功能介绍

HBase通过org.apache.hadoop.hbase.client.Admin对象的createTable方法来创建表，并指定表名、列族名、冷热时间线。

创建表有两种方式（强烈建议采用预分Region建表方式）：

- 快速建表，即创建表后整张表只有一个Region，随着数据量的增加会自动分裂成多个Region。
- 预分Region建表，即创建表时预先分配多个Region，此种方法建表可以提高写入大量数据初期的数据写入速度。

说明

表名以及列族名不能包含特殊字符，可以由字母、数字以及下划线组成。

代码样例

```
public void testCreateTable() {
  LOG.info("Entering testCreateTable.");

  // Specify the table descriptor.
  HTableDescriptor htd = new HTableDescriptor(tableName); // (1)

  // Set the column family name to info.
  HColumnDescriptor hcd = new HColumnDescriptor("info"); // (2)

  // Set hot and cold data boundary
  hcd.setValue(HColumnDescriptor.COLD_BOUNDARY, "86400");
  htd.addFamily(hcd); // (3)
```


版权所有 © 华为云计算技术有限公司

```
  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin(); // (4)
    if (!admin.tableExists(tableName)) {
      LOG.info("Creating table...");
      admin.createTable(htd); // 注[1] (5)
      LOG.info(admin.getClusterStatus());
      LOG.info(admin.listNamespaceDescriptors());
      LOG.info("Table created successfully.");
    } else {
      LOG.warn("table already exists");
    }
  } catch (IOException e) {
    LOG.error("Create table failed.", e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting testCreateTable.");
}
```

- 代码编号解释
  - （1）创建表描述符。
  - （2）创建列族描述符。
  - （3）添加列族描述符到表描述符中。
  - （4）获取Admin对象，Admin提供了建表、创建列族、检查表是否存在、修改表结构和列族结构以及删除表等功能。
  - （5）调用Admin的建表方法。
- 注意事项
  - 注[1] 表和列族其它属性设置可以参考开发HBase应用。

说明

注[1] 指的是代码样例中的“admin.createTable(htd); // 注[1] (5)”。

1.4.4.5 删除表

功能介绍

HBase通过org.apache.hadoop.hbase.client.Admin的deleteTable方法来删除表。

代码样例

```
public void dropTable() {
  LOG.info("Entering dropTable.");
  Admin admin = null;
  try {
    admin = conn.getAdmin();
    if (admin.tableExists(tableName)) {
      // Disable the table before deleting it.
      admin.disableTable(tableName);
      // Delete table.
      admin.deleteTable(tableName);//注[1]
    }
```


版权所有 © 华为云计算技术有限公司

```
  LOG.info("Drop table successfully.");
 } catch (IOException e) {
  LOG.error("Drop table failed " ,e);
 } finally {
  if (admin != null) {
   try {
    // Close the Admin object.
    admin.close();
   } catch (IOException e) {
    LOG.error("Close admin failed " ,e);
   }
  }
 }
 LOG.info("Exiting dropTable.");
}
```

注意事项

注[1]只有在调用disableTable接口后， 再调用deleteTable接口才能将表删除成功。

因此，deleteTable常与disableTable，enableTable，tableExists，isTableEnabled，isTableDisabled结合在一起使用。

说明

注[1]指的是代码样例中的“admin.deleteTable(tableName);//注[1]”。

1.4.4.6 修改表

功能介绍

HBase通过org.apache.hadoop.hbase.client.Admin的modifyTable方法修改表信息。

代码样例

- 取消冷热时间线。

```
public void testModifyTable() {
 LOG.info("Entering testModifyTable.");

 // Specify the column family name.
 byte[] familyName = Bytes.toBytes("info");
 Admin admin = null;
 try {
  // Instantiate an Admin object.
  admin = conn.getAdmin();
  // Obtain the table descriptor.
  HTableDescriptor htd = admin.getTableDescriptor(tableName);
  // Check whether the column family is specified before modification.
  if (!htd.hasFamily(familyName)) {
   // Create the column descriptor.
   HColumnDescriptor hcd = new HColumnDescriptor(familyName);

   //Disable hot and cold separation.
   hcd .setValue(HColumnDescriptor.COLD_BOUNDARY, null);

   htd.addFamily(hcd);
   // Disable the table to get the table offline before modifying
   // the table.
   admin.disableTable(tableName);
   // Submit a modifyTable request.
   admin.modifyTable(tableName, htd);  //注[1]
   // Enable the table to get the table online after modifying the
   // table.
   admin.enableTable(tableName);
```


版权所有 © 华为云计算技术有限公司

```
    }
    LOG.info("Modify table successfully.");
  } catch (IOException e) {
    LOG.error("Modify table failed " ,e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
      }
    }
  }
  LOG.info("Exiting testModifyTable.");
}
```

– 注意事项。

  注[1] 只有在调用disableTable接口后， 再调用modifyTable接口才能将表修改成功。之后，请调用enableTable接口重新启用表。

  说明

  注[1] 指的是代码样例中的“admin.modifyTable(tableName, htd); //注[1]”。

- 设置已有表的冷热分离功能。

```
public void testModifyTable() {
  LOG.info("Entering testModifyTable.");

  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin();
    // Obtain the table descriptor.
    HTableDescriptor htd = admin.getTableDescriptor(tableName);
    // Check whether the column family is specified before modification.
    if (!htd.hasFamily(familyName)) {
      // Create the column descriptor.
      HColumnDescriptor hcd = new HColumnDescriptor(familyName);

      //Set the hot and cold separation function for an existing table.
      hcd .setValue(HColumnDescriptor.COLD_BOUNDARY, "86400");

      htd.addFamily(hcd);
      // Disable the table to get the table offline before modifying
      // the table.
      admin.disableTable(tableName);
      // Submit a modifyTable request.
      admin.modifyTable(tableName, htd);  //注[1]
      // Enable the table to get the table online after modifying the
      // table.
      admin.enableTable(tableName);
    }
    LOG.info("Modify table successfully.");
  } catch (IOException e) {
    LOG.error("Modify table failed " ,e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Close admin failed " ,e);
      }
    }
  }
```

```
  LOG.info("Exiting testModifyTable.");
}
```

注意事项

注[1] 只有在调用disableTable接口后， 再调用modifyTable接口才能将表修改成功。之后，请调用enableTable接口重新启用表。

说明

注[1]指的是代码样例中的“admin.modifyTable(tableName, htd); //注[1]”。

1.4.4.7 插入数据

功能介绍

HBase是一个面向列的数据库，一行数据，可能对应多个列族，而一个列族又可以对应多个列。通常，写入数据的时候，我们需要指定要写入的列（含列族名称和列名称）。HBase通过HTable的put方法来Put数据，可以是一行数据也可以是数据集。

开启冷热分离特性表的写入逻辑和正常表写入逻辑一致。

代码样例

```
public void testPut() {
  LOG.info("Entering testPut.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifiers = { Bytes.toBytes("temp"), Bytes.toBytes("hum") };
  Table table = null;
  try {
    // Instantiate an HTable object.
    table = conn.getTable(tableName);
    // Instantiate a Put object. Every Hour insert one data.
    Put put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 00:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 01:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
    table.put(put);


    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 02:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 03:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
    puts.add(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 04:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 05:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
```


版权所有 © 华为云计算技术有限公司

```
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 06:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 07:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 08:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 09:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 10:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 11:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 12:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 13:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 14:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 15:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 16:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("31.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 17:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);


put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 18:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 19:00:00"));
```


版权所有 © 华为云计算技术有限公司

```
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 20:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 21:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 22:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/1 23:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 00:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 01:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 02:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 03:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
puts.add(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 04:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 05:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 06:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 07:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("27.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 08:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);
```


版权所有 © 华为云计算技术有限公司

```
put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 09:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("46.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 10:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 11:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("48.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 12:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 13:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("49.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 14:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("33.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 15:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("32.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("50.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 16:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("31.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 17:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 18:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("30.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
puts.clear();
puts.add(put);
table.put(puts);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 19:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("51.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 20:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("52.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 21:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("29.0"));
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("53.0"));
table.put(put);

put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 22:00:00"));
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
```


版权所有 © 华为云计算技术有限公司

```
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
    table.put(put);

    put = new Put(Bytes.toBytes("Shenzhen#Longgang#2017/7/2 23:00:00"));
    put.addColumn(familyName, qualifiers[0], Bytes.toBytes("28.0"));
    put.addColumn(familyName, qualifiers[1], Bytes.toBytes("54.0"));
    table.put(put);

    LOG.info("Put successfully.");
  } catch (IOException e) {
    LOG.error("Put failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testPut.");
}
```

1.4.4.8 使用 Get 读取数据

功能介绍

要从表中读取一条数据，首先需要实例化该表对应的Table实例，然后创建一个Get对象。也可以为Get对象设定参数值，如列族的名称和列的名称。查询到的行数据存储在Result对象中，Result中可以存储多个Cell。

针对开启冷热分离特性的列族，可以从冷热存储中查询数据，也可以只从热存储中查询数据。

代码样例

- 不指定HOT_ONLY参数来查询数据。在这种情况下，将会查询冷存储中的数据。

```
public void testGet() {
  LOG.info("Entering testGet.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifier = { Bytes.toBytes("temp"), Bytes.toBytes("hum") };
  // Specify RowKey.
  byte[] rowKey = Bytes.toBytes("Shenzhen#Longgang#2017/7/1 03:00:00");
  Table table = null;
  try {
    // Create the Table instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Get get = new Get(rowKey);
    // Set the column family name and column name.
    get.addColumn(familyName, qualifier[0]);
    get.addColumn(familyName, qualifier[1]);
    // Submit a get request.
    Result result = table.get(get);
    // Print query results.
    for (Cell cell : result.rawCells()) {
      LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
          + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
          + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
          + Bytes.toString(CellUtil.cloneValue(cell)));
    }
    LOG.info("Get data successfully.");
```

```
  } catch (IOException e) {
    LOG.error("Get data failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testGet.");
}
```

- 通过指定HOT_ONLY参数来查询数据。在这种情况下，只会查询热存储中的数据。

```
public void testGet() {
  LOG.info("Entering testGet.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifier = { Bytes.toBytes("temp"), Bytes.toBytes("hum") };
  // Specify RowKey.
  byte[] rowKey = Bytes.toBytes("Shenzhen#Longgang#2017/7/2 10:00:00");
  Table table = null;
  try {
    // Create the Table instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Get get = new Get(rowKey);
    // Set HOT_ONLY.
    get.setAttribute(HBaseConstants.HOT_ONLY, Bytes.toBytes(true));
    // Set the column family name and column name.
    get.addColumn(familyName, qualifier[0]);
    get.addColumn(familyName, qualifier[1]);
    // Submit a get request.
    Result result = table.get(get);
    // Print query results.
    for (Cell cell : result.rawCells()) {
      LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
          + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
          + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
          + Bytes.toString(CellUtil.cloneValue(cell)));
    }
    LOG.info("Get data successfully.");
  } catch (IOException e) {
    LOG.error("Get data failed " ,e);
  } finally {
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testGet.");
}
```

1.4.4.9 使用 Scan 读取数据

功能介绍

要从表中读取数据，首先需要实例化该表对应的Table实例，然后创建一个Scan对象，并针对查询条件设置Scan对象的参数值，为了提高查询效率，最好指定StartRow和

版权所有 © 华为云计算技术有限公司

StopRow。查询结果的多行数据保存在ResultScanner对象中，每行数据以Result对象形式存储，Result中存储了多个Cell。

代码样例

- 不指定HOT_ONLY参数来查询数据。在这种情况下，将会查询冷存储中的数据。

```
public void testScanData() {
  LOG.info("Entering testScanData.");
  Table table = null;
  // Instantiate a ResultScanner object.
  ResultScanner rScanner = null;
  try {
    // Create the Configuration instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Scan scan = new Scan();
    byte[] startRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/1 00:00:00);
    byte[] stopRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/3 00:00:00);
    scan.setStartRow(startRow);
    scan.setStopRow(stopRow);
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("temp"));
    // Set the cache size.
    scan.setCaching(1000);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Scan data successfully.");
  } catch (IOException e) {
    LOG.error("Scan data failed " ,e);
  } finally {
    if (rScanner != null) {
      // Close the scanner object.
      rScanner.close();
    }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testScanData.");
}
```

- 通过指定HOT_ONLY参数来查询数据。在这种情况下，只会查询热存储中的数据。

```
public void testScanData() {
  LOG.info("Entering testScanData.");
  Table table = null;
  // Instantiate a ResultScanner object.
  ResultScanner rScanner = null;
  try {
    // Create the Configuration instance.
    table = conn.getTable(tableName);
    // Instantiate a Get object.
    Scan scan = new Scan();
    byte[] startRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/1 00:00:00);
    byte[] stopRow = Bytes.toBytes(Shenzhen#Longgang#2017/7/3 00:00:00);
```


版权所有 © 华为云计算技术有限公司

```
    scan.setStartRow(startRow);
    scan.setStopRow(stopRow);
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("temp"));

    // Set HOT_ONLY.
    scan.setAttribute(HBaseConstants.HOT_ONLY, Bytes.toBytes(true));
    // Set the cache size.
    scan.setCaching(1000);
    // Submit a scan request.
    rScanner = table.getScanner(scan);
    // Print query results.
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (Cell cell : r.rawCells()) {
        LOG.info(Bytes.toString(CellUtil.cloneRow(cell)) + ":"
            + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
            + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
            + Bytes.toString(CellUtil.cloneValue(cell)));
      }
    }
    LOG.info("Scan data successfully.");
  } catch (IOException e) {
    LOG.error("Scan data failed " ,e);
  } finally {
    if (rScanner != null) {
      // Close the scanner object.
      rScanner.close();
    }
    if (table != null) {
      try {
        // Close the HTable object.
        table.close();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
  LOG.info("Exiting testScanData.");
}
```

1.5 访问 HBase ThriftServer 样例程序

1.5.1 访问 ThriftServer 操作表

操作场景

用户根据指定的host和port访问对应的ThriftServer实例，进行HBase表的创建，删除等操作。

前提条件

- 集群已启用ThriftServer并从集群详情页面获取到ThriftServer IP。
- 已下载Thrift安装包，安装包下载地址：链接。
- 已下载HBase Thrift定义文件，文件下载地址：地址。

操作步骤

步骤1 登录表格存储服务控制台。

步骤2 在页面左上角选择区域。

版权所有 © 华为云计算技术有限公司

步骤3 单击“集群管理”，进入集群管理界面。

步骤4 单击HBase集群名称，进入集群详情页面查看ThriftServer的状态，如果ThriftServer为开启状态，无需开启操作；如果ThriftServer为关闭状态，则选返回集群管理界面，单击“更多 > 开启ThriftServer”，等待完成后，即可进行使用。

说明

- ThriftServer当前不具备负载均衡的能力，用户需要避免在代码里面同时访问同一个ThriftServer实例，避免单实例过载。
- 用户需要在应用代码里面增加重试机制，保证其中一个ThriftServer实例故障或者重启时，可以重试其他ThriftServer实例。

步骤5 参考**Thrift官方指导**在客户端节点安装Thrift安装包。

步骤6 使用Thrift命令将HBase Thrift定义文件生成对应语言的接口文件，支持的语言有C++，Python等。参考命令如下：

```
thrift --gen <语言> hbase.thrift
```

说明

<语言>为要生成的目标语言，支持cpp(C++)、py(Python)等。

以Python为例，执行命令为：thrift --gen py hbase.thrif

----结束

C++代码样例

```
#include "THBaseService.h"
#include <config.h>
#include <vector>
#include <ostream>
#include <iostream>
#include "transport/TSocket.h"
#include <transport/TBufferTransports.h>
#include <protocol/TBinaryProtocol.h>
using namespace std;
using namespace apache::thrift;
using namespace apache::thrift::protocol;
using namespace apache::thrift::transport;
using namespace apache::hadoop::hbase::thrift2;
using boost::shared_ptr;
int main(int argc, char **argv) {
    // ThriftServer的ip和端口号
    std::string host = "x.x.x.x";
    int port = 9090;
    boost::shared_ptr<TSocket> socket(new TSocket(host, port));
    boost::shared_ptr<TTransport> transport(new TBufferedTransport(socket));
    boost::shared_ptr<TProtocol> protocol(new TBinaryProtocol(transport));
    // 设置表名
    std::string ns("default");
    std::string table("test");
    TTableName tableName;
    tableName.__set_ns(ns);
    tableName.__set_qualifier(table);
     try {
        // 创建连接
        transport->open();
        printf("Opened connection\n");
        // 初始化客户端接口
        THBaseServiceClient  client(protocol);

        // 建表
        TColumnFamilyDescriptor column;
        column.__set_name("f1");
```


版权所有 © 华为云计算技术有限公司

```
column.__set_maxVersions(10);
std::vector<TColumnFamilyDescriptor> columns;
columns.push_back(column);

TTableDescriptor tableDescriptor;
tableDescriptor.__set_tableName(tableName);
tableDescriptor.__set_columns(columns);
std::vector<std::string> splitKeys;
splitKeys.push_back("row2");
splitKeys.push_back("row4");
splitKeys.push_back("row8");
printf("Creating table: %s\n", table.c_str());
try {
    client.createTable(tableDescriptor, splitKeys);
} catch (const TException  &te) {
    std::cerr << "ERROR: " << te.what() << std::endl;
}
// Put写入单条数据
TColumnValue columnValue;
columnValue.__set_family("f1");
columnValue.__set_qualifier("q1");
columnValue.__set_value("val_001");
std::vector<TColumnValue> columnValues;
columnValues.push_back(columnValue);
TPut put;
put.__set_row("row1");
put.__set_columnValues(columnValues);
client.put(table, put);
printf("Put single row success\n");
// Put写入多条数据
TColumnValue columnValue2;
columnValue2.__set_family("f1");
columnValue2.__set_qualifier("q1");
columnValue2.__set_value("val_003");
std::vector<TColumnValue> columnValues2;
columnValues2.push_back(columnValue2);
TPut put2;
put2.__set_row("row3");
put2.__set_columnValues(columnValues2);
TColumnValue columnValue3;
columnValue3.__set_family("f1");
columnValue3.__set_qualifier("q1");
columnValue3.__set_value("val_005");
std::vector<TColumnValue> columnValues3;
columnValues3.push_back(columnValue3);
TPut put3;
put3.__set_row("row5");
put3.__set_columnValues(columnValues3);
std::vector<TPut> puts;
puts.push_back(put2);
puts.push_back(put3);
client.putMultiple(table, puts);
printf("Put multiple rows success\n");
// Get查询单条数据
TResult result;
TGet get;
get.__set_row("row1");
client.get(result, table, get);
std::vector<TColumnValue> list=result.columnValues;
std::vector<TColumnValue>::const_iterator iter;
std::string row = result.row;
for(iter=list.begin();iter!=list.end();iter++) {
    printf("%s=%s, %s,%s\n",row.c_str(),(*iter).family.c_str(),(*iter).qualifier.c_str(),(*iter).value.c_str());
}
printf("Get single row success.\n");
// Get查询多条数据
std::vector<TGet> multiGets;
TGet get1;
get1.__set_row("row1");
```


版权所有 © 华为云计算技术有限公司

```
        multiGets.push_back(get1);
        TGet get2;
        get2.__set_row("row5");
        multiGets.push_back(get2);

        std::vector<TResult> multiRows;
        client.getMultiple(multiRows, table, multiGets);
        for(std::vector<TResult>::const_iterator iter1=multiRows.begin();iter1!=multiRows.end();iter1++) {
            std::vector<TColumnValue> list=(*iter1).columnValues;
            std::vector<TColumnValue>::const_iterator iter2;
            std::string row = (*iter1).row;
            for(iter2=list.begin();iter2!=list.end();iter2++) {
                printf("%s=%s, %s,%s\n",row.c_str(),(*iter2).family.c_str(),(*iter2).qualifier.c_str(),
(*iter2).value.c_str());
            }
        }
        printf("Get multiple rows success.\n");
        // Scan查询数据
        TScan scan;
        scan.__set_startRow("row1");
        scan.__set_stopRow("row7");
        int32_t nbRows = 2;
        std::vector<TResult> results;
        TResult* current = NULL;
        while (true) {
            client.getScannerResults(results, table, scan, nbRows);
            if (results.size() == 0) {
                printf("No more result.\n");
                break;
            }
            std::vector<TResult>::const_iterator itx;
            for(itx=results.begin();itx!=results.end();itx++) {
                current = (TResult*) &(*itx);
                if (current == NULL) {
                    break;
                } else {
                    std::vector<TColumnValue> values=(*current).columnValues;
                    std::vector<TColumnValue>::const_iterator iterator;
                    for(iterator=list.begin();iterator!=list.end();iterator++) {
                        printf("%s=%s, %s,%s\n",(*current).row.c_str(),(*iterator).family.c_str(),
(*iterator).qualifier.c_str(),(*iterator).value.c_str());
                    }
                }
            }
            if (current == NULL) {
                printf("Scan data done.\n");
                break;
            } else {
                scan.__set_startRow((*current).row + (char)0);
            }
        }
        //禁用和删除表
        client.disableTable(tableName);
        printf("Disabled %s\n", table.c_str());
        client.deleteTable(tableName);
        printf("Deleted %s\n", table.c_str());
        transport->close();
        printf("Closed connection\n");
    } catch (const TException &tx) {
        std::cerr << "ERROR(exception): " << tx.what() << std::endl;
    }
    return 0;
}
```

Python 代码样例

```
# -*- coding: utf-8  -*-

# 引入公共模块
```

```
import sys
import os

# 引入Thrift自带模块，如果不存在则需要执行pip install thrift安装
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol
from thrift.transport import THttpClient
from thrift.transport import TSocket

# 引入通过hbase.thrift生成的模块
gen_py_path = os.path.abspath('gen-py')
sys.path.append(gen_py_path)
from hbase import THBaseService
from hbase.ttypes import TColumnValue, TColumn, TTableName, TTableDescriptor,
TColumnFamilyDescriptor, TGet, TPut, TScan
# 配置CloudTable HBase集群的ThriftServer的IP，可以通过集群的详情页面获取
host = "x.x.x.x"

socket = TSocket.TSocket(host, 9090)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = THBaseService.Client(protocol)
transport.open()

# 测试表名
tableNameInBytes = "test".encode("utf8")

tableName = TTableName(ns="default".encode("utf8"), qualifier=tableNameInBytes)
# 预分region的split key
splitKeys=[]
splitKeys.append("row3".encode("utf8"))
splitKeys.append("row5".encode("utf8"))
# 建表操作
client.createTable(TTableDescriptor(tableName=tableName,
columns=[TColumnFamilyDescriptor(name="cf1".encode("utf8"))]), splitKeys)
print("Create table %s success." % tableName)

# Put单条数据
put = TPut(row="row1".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))])
client.put(tableNameInBytes, put)
print("Put single row success.")

# Put多条数据
puts = []
puts.append(TPut(row="row4".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))]))
puts.append(TPut(row="row6".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))]))
puts.append(TPut(row="row8".encode("utf8"), columnValues=[TColumnValue(family="cf1".encode("utf8"),
qualifier="q1".encode("utf8"), value="test_value1".encode("utf8"))]))
client.putMultiple(tableNameInBytes, puts)
print("Put rows success.")

# Get单条数据
get = TGet(row="row1".encode("utf8"))
result = client.get(tableNameInBytes, get)
print("Get Result: ", result)

# Get多条数据
gets = []
gets.append(TGet(row="row4".encode("utf8")))
gets.append(TGet(row="row8".encode("utf8")))
results = client.getMultiple(tableNameInBytes, gets)
print("Get multiple rows: ", results)

# Scan数据
startRow, stopRow = "row4".encode("utf8"), "row9".encode("utf8")
scan = TScan(startRow=startRow, stopRow=stopRow)
```


版权所有 © 华为云计算技术有限公司

```
caching=1
results = []
while True:
    scannerResult = client.getScannerResults(tableNameInBytes, scan, caching)
    lastOne = None
    for result in scannerResult:
        results.append(result)
        print("Scan Result: ", result)
        lastOne = result
    # 没有更多数据，退出
    if lastOne is None:
        break
    else:
        # 重新生成下一次scan的startRow
        newStartRow = bytearray(lastOne.row)
        newStartRow.append(0x00)
        scan = TScan(startRow=newStartRow, stopRow=stopRow)


# 禁用和删除表
client.disableTable(tableName)
print("Disable table %s success." % tableName)
client.deleteTable(tableName)
print("Delete table %s success." % tableName)

# 所有操作都结束后，关闭连接
transport.close()
```

1.6 开发标签索引应用

1.6.1 应用背景

CloudTable作为大数据存储服务，提供高效的kv随机查询能力。在此基础上，CloudTable服务引入自研的**分布式多维标签索引**能力，存储格式与计算基于位图进行。用户可以根据自身业务需求来定义HBase表中的哪些字段需要构建标签索引，用户写入数据时将自动生成标签数据。同时，标签索引基于Lucene的语法，提供高效的多维标签查询接口。可应用于用户画像、推荐系统、人工智能、时空数据等场景。

CloudTable服务支持标签索引能力，您只需要创建CloudTable集群，就可以在弹性云服务器（ECS）上开发客户端应用进行多维标签查询。

1.6.2 典型场景说明

通过典型场景，我们可以快速学习和掌握标签索引的开发过程，并且对关键的接口函数有所了解。

场景说明

某在线付费学习APP给会员打上各种属性标签，以方便后续的资源投放和精准定位营销。例如，需要ms级统计拥有学士和硕士学位的用户数量是多少？以及是哪些用户？

用户信息表字段如下：

表 1-7 用户信息

字段名称	字段描述	是否需要标签索引
name	用户名	否

字段名称	字段描述	是否需要标签索引
education	用户学历	是
otherInfo	用户其他信息	否

1.6.3 开发思路

表 1-8 开发思路

序号	步骤	代码实现
1	创建HBase表时开启标签索引	请参见创建数据表开启标签索引
2	HBase put写入数据	请参见写入数据
3	查询数据	请参见： ● 普通查询 ● 抽样查询 ● 分页查询 ● 统计查询

1.6.4 样例代码说明

1.6.4.1 配置参数

步骤1 执行样例代码前，必须在hbase-site.xml配置文件中，配置正确的ZooKeeper集群的地址。

配置项如下：

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

其中：*value*中的值为ZooKeeper集群的域名。登录表格存储服务管理控制台，在左侧导航树单击“集群管理”，然后在集群列表中找到所需要的集群，并获取相应的“ZK链接地址（内网）”。

----结束

1.6.4.2 创建 Configuration

功能介绍

HBase通过加载配置文件来获取配置项。

版权所有 © 华为云计算技术有限公司

说明

1. 加载配置文件是一个比较耗时的操作，如非必要，请尽量使用同一个Configuration对象。
2. 样例代码未考虑多线程同步的问题，如有需要，请自行增加。其它样例代码也一样，不再一一进行说明。

代码样例

下面代码片段在com.huawei.cloudtable.lemonIndex.examples.LemonIndexTestMain包中。

```
private static void init() throws IOException {
  // Default load from conf directory
  conf = HBaseConfiguration.create(); // 注[1]
  String userdir = System.getProperty("user.dir") + File.separator + "conf" + File.separator;
  Path hbaseSite = new Path(userdir + "hbase-site.xml");
  if (new File(hbaseSite.toString()).exists()) {
    conf.addResource(hbaseSite);
  }
}
```

注意事项

- 注[1] 如果配置文件目录conf已经加入classpath路径中，那么后面的加载指定配置文件的代码可以不执行。

1.6.4.3 创建数据表开启标签索引

功能介绍

建表功能同创建表，在此基础上，表属性配置标签索引schema。

样例代码

```
public void testCreateTable() {
  LOG.info("Entering testCreateTable.");
  HTableDescriptor tableDesc = new HTableDescriptor(tableName);
  HColumnDescriptor cdm = new HColumnDescriptor(FAM_M);
  cdm.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
  tableDesc.addFamily(cdm);
  HColumnDescriptor cdn = new HColumnDescriptor(FAM_N);
  cdn.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
  tableDesc.addFamily(cdn);

  // Add bitmap index definitions.
  List<BitmapIndexDescriptor> bitmaps = new ArrayList<>();//(1)
  bitmaps.add(BitmapIndexDescriptor.builder()
    // Describe which column should be indexed.
    .setColumnName(FamilyOnlyName.valueOf(FAM_M))//(2)
    // Describe how to extract term(s) from KeyValue
    .setTermExtractor(TermExtractor.NAME_VALUE_EXTRACTOR)//(3)
    .build());
  // It will help to add several properties into HTableDescriptor.
  // SHARD_NUM should be less than the region number
  IndexHelper.enableAutoIndex(tableDesc, SHARD_NUM, bitmaps);//(4)

  List<byte[]> splitList = Arrays.stream(SPLIT.split(LemonConstants.COMMA))
    .map(s -> org.lemon.common.Bytes.toBytes(s.trim()))
    .collect(Collectors.toList());
  byte[][] splitArray = splitList.toArray(new byte[splitList.size()][]);

  Admin admin = null;
  try {
```


版权所有 © 华为云计算技术有限公司

```
    // Instantiate an Admin object.
    admin = conn.getAdmin();
    if (!admin.tableExists(tableName)) {
      LOG.info("Creating table...");
      admin.createTable(tableDesc, splitArray);
      LOG.info(admin.getClusterStatus());
      LOG.info(admin.listNamespaceDescriptors());
      LOG.info("Table created successfully.");
    } else {
      LOG.warn("table already exists");
    }
  } catch (IOException e) {
    LOG.error("Create table failed.", e);
  } finally {
    if (admin != null) {
      try {
        // Close the Admin object.
        admin.close();
      } catch (IOException e) {
        LOG.error("Failed to close admin ", e);
      }
    }
  }
  LOG.info("Exiting testCreateTable.");
}
```

注意事项

- (1) BitmapIndexDescriptor描述哪些字段使用什么规则来抽取标签，数据表可以定义一个或多个BitmapIndexDescriptor。
- (2) 定义哪些列需要抽取标签。取值范围：
  - ExplicitColumnName：指定列。
  - FamilyOnlyName：某一ColumnFamily下的所有列。
  - PrefixColumnName：拥有某一前缀的列。
- (3) 定义列的抽取标签的规则，可选值如下：
  - QualifierExtractor：表示按照列名来抽取标签。

    例如，qualifier是Male，value是1，那么抽取的标签是Male。
  - QualifierValueExtractor：表示按照列名和value来抽取标签。

    例如，qualifier是education，value是master，那么抽取的标签是education:master。
  - QualifierArrayValueExtractor：可以抽取多个标签，value是json array格式。

    例如，qualifier是hobby，value 是 ["basketball","football","volleyball"]，抽取的标签如下：
    ```
    hobby:basketball
    hobby:football
    hobby:volleyball
    ```
  - QualifierMapValueExtractor：可以抽取多个标签，value是json map格式。

    例如，qualifier是hobby，value是{"basketball":"9","football":"8","volleyball":"7"}，抽取的标签如下：
    ```
    hobby:basketball
    hobby:football
    hobby:volleyball
    hobby:basketball_9
    hobby:football_8
    hobby:volleyball_7
    ```
- (4) 索引表的分区数量SHARD_NUM必须要小于或等于数据表。

 版权所有 © 华为云计算技术有限公司

1.6.4.4 写入数据

写入数据接口和HBase原生API一致。

可以参考以下样例代码。

样例代码

```
public void testPut() {
  LOG.info("Entering testPut.");
  try(Table table = conn.getTable(tableName)) {
    List<Put> puts = new ArrayList<>();
    Arrays.stream(SPLIT.split(LemonConstants.COMMA))
      .forEach(startkey -> {
        // Instantiate a Put object.
        Put put = new Put(Bytes.toBytes(startkey + "-rowkey001"));
        put.addColumn(FAM_M, QUA_M, Bytes.toBytes("bachelor"));
        put.addColumn(FAM_N, QUA_N, Bytes.toBytes("xiaowang"));
        puts.add(put);

        Put put1 = new Put(Bytes.toBytes(startkey + "-rowkey0012"));
        put1.addColumn(FAM_M, QUA_M, Bytes.toBytes("master"));
        put1.addColumn(FAM_N, QUA_N, Bytes.toBytes("xiaoming"));
        puts.add(put1);

        // Submit a put request.
        try {
          table.put(puts);
        } catch (IOException e) {
          LOG.info("put exception", e);
        }
      });
    LOG.info("Put successfully.");
  } catch (IOException e) {
    LOG.error("Put failed ", e);
  }
  LOG.info("Exiting testPut.");
}
```

1.6.4.5 普通查询

功能介绍

CloudTable标签索引基于Lucene语法，提供了自研的查询接口LemonTable.query(LemonQuery query)。

样例代码

```
public void testNormalQuery() {
  LOG.info("Entering testNormalQuery.");

  try (Table table = conn.getTable(tableName)) {
    // Using Table instance to create LemonTable.
    LemonTable lemonTable = new LemonTable(table);
    // Build LemonQuery.
    LemonQuery query = LemonQuery.builder()
      // Set ad-hoc query condition.
      .setQuery("education:bachelor OR education:master") //(1)
      // Set how many rows should be cached on client for the initial request.
      .setCaching(10) //(2)
      // Set return column family/columns.
      .addFamily(FAM_M)
      .addColumn(FAM_N, QUA_N) //(3)
      // Set return return result just contains rowkeys, no any qualifier
      // the CF of LemonConstants.EMPTY_COLUMN_RETURN can be a random existing CF
```


版权所有 © 华为云计算技术有限公司

```
    //.addColumn(FAM_M, LemonConstants.EMPTY_COLUMN_RETURN)
    .build();
  ResultSet resultSet = lemonTable.query(query);
  // Read result rows.

  int count = resultSet.getCount();
  LOG.info("the entity count of query is " + count);

  List<EntityEntry> entries = resultSet.listRows();
  for (EntityEntry entry : entries) {
    Map<String, Map<String, String>> fams = entry.getColumns();
    for (Map.Entry<String, Map<String, String>> familyEntry : fams.entrySet()) {
      String family = familyEntry.getKey();
      Map<String, String> qualifiers = familyEntry.getValue();
      for (Map.Entry<String, String> qualifier : qualifiers.entrySet()) {
        String Qua = qualifier.getKey();
        String value = qualifier.getValue();
        LOG.info("rowkey is " + Bytes.toString(entry.getRow()) + ", qualifier is "
          + family + ":" + Qua + ", value is " + value);
      }
    }
  }
 } catch (IOException e) {
  LOG.error("testNormalQuery failed ", e);
 }

 LOG.info("Exiting testNormalQuery.");
}
```

注意事项

(1) 查询条件，遵循Lucene语法/BNF范式，例如：

```
"Male ANDMarried AND AGE:25-30 AND BLOOD_TYPE:A"
"Male ANDMarried AND (AGE:25-30 OR AGE:30-35)AND BLOOD_TYPE:A"
```

(2) 查询返回多少行数据。

(3) 每行数据返回哪些qualifier，如果只设置列族，则表示返回列族下的所有列。

1.6.4.6 抽样查询

功能介绍

在普通查询的基础上设置setSampling()，查询时从索引表中随机选择一个分片执行查询任务。

可以参考以下样例代码。

样例代码

```
public void testSamplingQuery() {
 LOG.info("Entering testSamplingQuery.");

 try (Table table = conn.getTable(tableName)) {
  // Using Table instance to create LemonTable.
  LemonTable lemonTable = new LemonTable(table);
  // Build LemonQuery.
  LemonQuery query = LemonQuery.builder()
    // Set ad-hoc query condition.
    .setQuery("education:bachelor OR education:master")
    // Set how many rows should be cached on client for the initial request.
    .setCaching(10)
    // sampling query will be select one random shard/region to query
    .setSampling()
    // Set return column family/columns.
```

```
      .addFamily(FAM_M)
      .addColumn(FAM_N, QUA_N)
      // Set return return result just contains rowkeys, no any qualifier
      // the CF of LemonConstants.EMPTY_COLUMN_RETURN can be a random existing CF
      //.addColumn(FAM_M, LemonConstants.EMPTY_COLUMN_RETURN)
      .build();
    ResultSet resultSet = lemonTable.query(query);
    // Read result rows.
    int count = resultSet.getCount();
    LOG.info("the entity count of query is " + count);

    List<EntityEntry> entries = resultSet.listRows();
    for (EntityEntry entry : entries) {
      Map<String, Map<String, String>> fams = entry.getColumns();
      for (Map.Entry<String, Map<String, String>> familyEntry : fams.entrySet()) {
        String family = familyEntry.getKey();
        Map<String, String> qualifiers = familyEntry.getValue();
        for (Map.Entry<String, String> qualifier : qualifiers.entrySet()) {
          String Qua = qualifier.getKey();
          String value = qualifier.getValue();
          LOG.info("rowkey is " + Bytes.toString(entry.getRow()) + ", qualifier is "
            + family + ":" + Qua + ", value is " + value);
        }
      }
    }
  } catch (IOException e) {
    LOG.error("testSamplingQuery failed ", e);
  }

  LOG.info("Exiting testSamplingQuery.");
  LOG.info("");
}
public void testSamplingQuery() {
  LOG.info("Entering testSamplingQuery.");

  try (Table table = conn.getTable(tableName)) {
    // Using Table instance to create LemonTable.
    LemonTable lemonTable = new LemonTable(table);
    // Build LemonQuery.
    LemonQuery query = LemonQuery.builder()
      // Set ad-hoc query condition.
      .setQuery("education:bachelor OR education:master")
      // Set how many rows should be cached on client for the initial request.
      .setCaching(10)
      // sampling query will be select one random shard/region to query
      .setSampling()
      // Set return column family/columns.
      .addFamily(FAM_M)
      .addColumn(FAM_N, QUA_N)
      // Set return return result just contains rowkeys, no any qualifier
      // the CF of LemonConstants.EMPTY_COLUMN_RETURN can be a random existing CF
      //.addColumn(FAM_M, LemonConstants.EMPTY_COLUMN_RETURN)
      .build();
    ResultSet resultSet = lemonTable.query(query);
    // Read result rows.
    int count = resultSet.getCount();
    LOG.info("the entity count of query is " + count);

    List<EntityEntry> entries = resultSet.listRows();
    for (EntityEntry entry : entries) {
      Map<String, Map<String, String>> fams = entry.getColumns();
      for (Map.Entry<String, Map<String, String>> familyEntry : fams.entrySet()) {
        String family = familyEntry.getKey();
        Map<String, String> qualifiers = familyEntry.getValue();
        for (Map.Entry<String, String> qualifier : qualifiers.entrySet()) {
          String Qua = qualifier.getKey();
          String value = qualifier.getValue();
          LOG.info("rowkey is " + Bytes.toString(entry.getRow()) + ", qualifier is "
            + family + ":" + Qua + ", value is " + value);
```


版权所有 © 华为云计算技术有限公司

```
      }
    }
  }
} catch (IOException e) {
  LOG.error("testSamplingQuery failed ", e);
}

LOG.info("Exiting testSamplingQuery.");
LOG.info("");
}
```

1.6.4.7 分页查询

功能介绍

先执行query接口返回简要数据信息，而后调用listRows接口翻页。

可以参考以下样例代码。

样例代码

```
public void testPagingQuery() {
  LOG.info("Entering testPagingQuery.");

  try (Table table = conn.getTable(tableName)) {
    // Using Table instance to create LemonTable.
    LemonTable lemonTable = new LemonTable(table);
    // Build LemonQuery.
    LemonQuery query = LemonQuery.builder()
      // Set ad-hoc query condition.
      .setQuery("education:bachelor OR education:master")
      // Set how many rows should be cached on client for the initial request.
      .setCaching(10)
      // Set return column family/columns.
      .addFamily(FAM_M)
      .addColumn(FAM_N, QUA_N)
      // Set return return result just contains rowkeys, no any qualifier
      // the CF of LemonConstants.EMPTY_COLUMN_RETURN can be a random existing CF
      //.addColumn(FAM_M, LemonConstants.EMPTY_COLUMN_RETURN)
      .build();
    ResultSet resultSet = lemonTable.query(query);
    // Read result rows.
    int count = resultSet.getCount();
    LOG.info("the entity count of query is " + count);

    // Read result page by page, every page show 10 lines data
    int maxPage = 100;
    final int lineNumPerPage = 5;
    for (int i = 0; i < maxPage; i++) {
      int start = lineNumPerPage * i;
      List<EntityEntry> entries = resultSet.listRows(start, lineNumPerPage);
      if (entries == null || entries.size() == 0)
      {
        break;
      }

      LOG.info("page " + (i + 1) + " count is " + entries.size() + ", result is following:");
      for (EntityEntry entry : entries) {
        Map<String, Map<String, String>> fams = entry.getColumns();
        for (Map.Entry<String, Map<String, String>> familyEntry : fams.entrySet()) {
          String family = familyEntry.getKey();
          Map<String, String> qualifiers = familyEntry.getValue();
          for (Map.Entry<String, String> qualifier : qualifiers.entrySet()) {
            String Qua = qualifier.getKey();
            String value = qualifier.getValue();
            LOG.info("rowkey is " + Bytes.toString(entry.getRow()) + ", qualifier is "
              + family + ":" + Qua + ", value is " + value);
```


版权所有 © 华为云计算技术有限公司

```
        }
      }
    }
  }
} catch (IOException e) {
  LOG.error("testPagingQuery failed ", e);
}

LOG.info("Exiting testPagingQuery.");
}
```

1.6.4.8 统计查询

功能介绍

返回满足查询条件的实体总量，不返回数据的具体信息，代码中设置setCountOnly()。

可以参考以下样例代码。

样例代码

```
public void testCountOnlyQuery() {
  LOG.info("Entering testCountOnlyQuery.");

  try (Table table = conn.getTable(tableName)) {
    // Using Table instance to create LemonTable.
    LemonTable lemonTable = new LemonTable(table);
    // Build LemonQuery.
    LemonQuery query = LemonQuery.builder()
      // Set ad-hoc query condition.
      .setQuery("education:bachelor OR education:master")
      // just return how many entities meet the query condition, without any rowkey/column
      .setCountOnly()
      .build();
    ResultSet resultSet = lemonTable.query(query);
    // Read result rows.
    int count = resultSet.getCount();
    LOG.info("the entity count of query is " + count);
  } catch (IOException e) {
    LOG.error("testCountOnlyQuery failed ", e);
  }

  LOG.info("Exiting testCountOnlyQuery.");
}
```

1.7 调测程序

1.7.1 在 Windows 中调测程序

1.7.1.1 编译并运行程序

操作场景

在程序代码完成开发后，您可以在Windows开发环境中运行应用。

版权所有 © 华为云计算技术有限公司

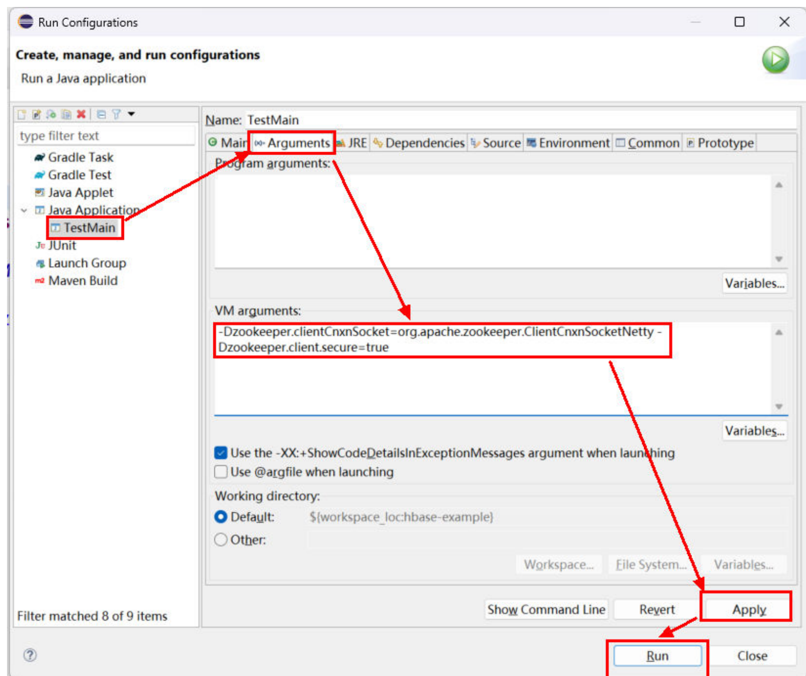
操作步骤

- **未开启加密通道的HBase集群**

  在开发环境中（例如Eclipse中），右击“TestMain.java”，单击“Run as > Java Application”运行对应的应用程序工程。

- **开启加密通道的HBase集群**

  在开发环境中（例如Eclipse中），右击“TestMain.java”，单击“Run as > Run Configurations”，参考如下截图添加环境变量"-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true"后，再运行对应的应用程序工程。



1.7.1.2 查看调测结果

运行结果中没有异常或失败信息即表明运行成功。

图 1-8 运行成功

```
2016-07-13 14:36:12,736 INFO  [main] basic.CreateTableSample: Create table sampleNameSpace:sampleTable successful!
2016-07-13 14:36:15,426 INFO  [main] basic.ModifyTableSample: Modify table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:16,708 INFO  [main] basic.MultiSplitSample: Mmulti split table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:17,299 INFO  [main] basic.PutDataSample: Successfully put 9 items data into sampleNameSpace:sampleTable.
2016-07-13 14:36:18,992 INFO  [main] basic.ScanSample: Scan data successfully.
2016-07-13 14:36:20,532 INFO  [main] basic.DeletaDataSample: Successfully delete data from table sampleNameSpace:sampleTable.
2016-07-13 14:36:21,006 INFO  [main] acl.AclSample: Grant ACL for table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:27,836 INFO  [main] index.CreateIndexSample: Successfully add index for table sampleNameSpace:sampleTable.
```

说明

在Windows环境运行样例代码时会出现下面的异常，但是不影响业务：

java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.

日志说明：

版权所有 © 华为云计算技术有限公司

日志级别默认为INFO，可以通过调整日志打印级别（DEBUG，INFO，WARN，ERROR，FATAL）来显示更详细的信息。可以通过修改log4j.properties文件来实现，如：

```
hbase.root.logger=INFO,console
log4j.logger.org.apache.zookeeper=INFO
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG
log4j.logger.org.apache.hadoop.hbase=INFO
# Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO
```

1.7.2 在 Linux 中调测程序

1.7.2.1 安装客户端时编译并运行程序

操作场景

HBase应用程序支持在安装HBase客户端的Linux环境中运行。在程序代码完成开发后，您可以上传Jar包至Linux环境中运行应用。

前提条件

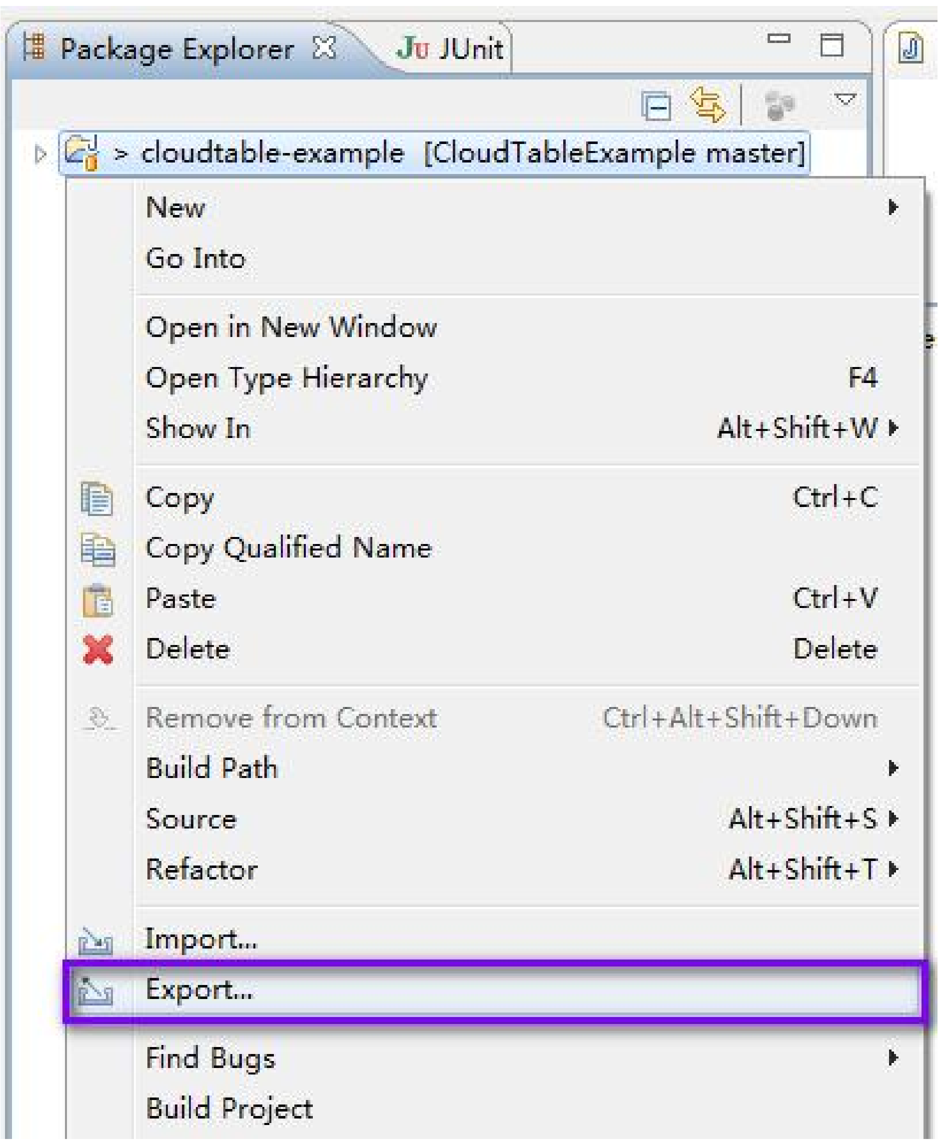
- 已安装HBase客户端。
- Linux环境已安装JDK，版本号需要和Eclipse导出Jar包使用的JDK版本一致。

操作步骤

步骤1 导出Jar包。

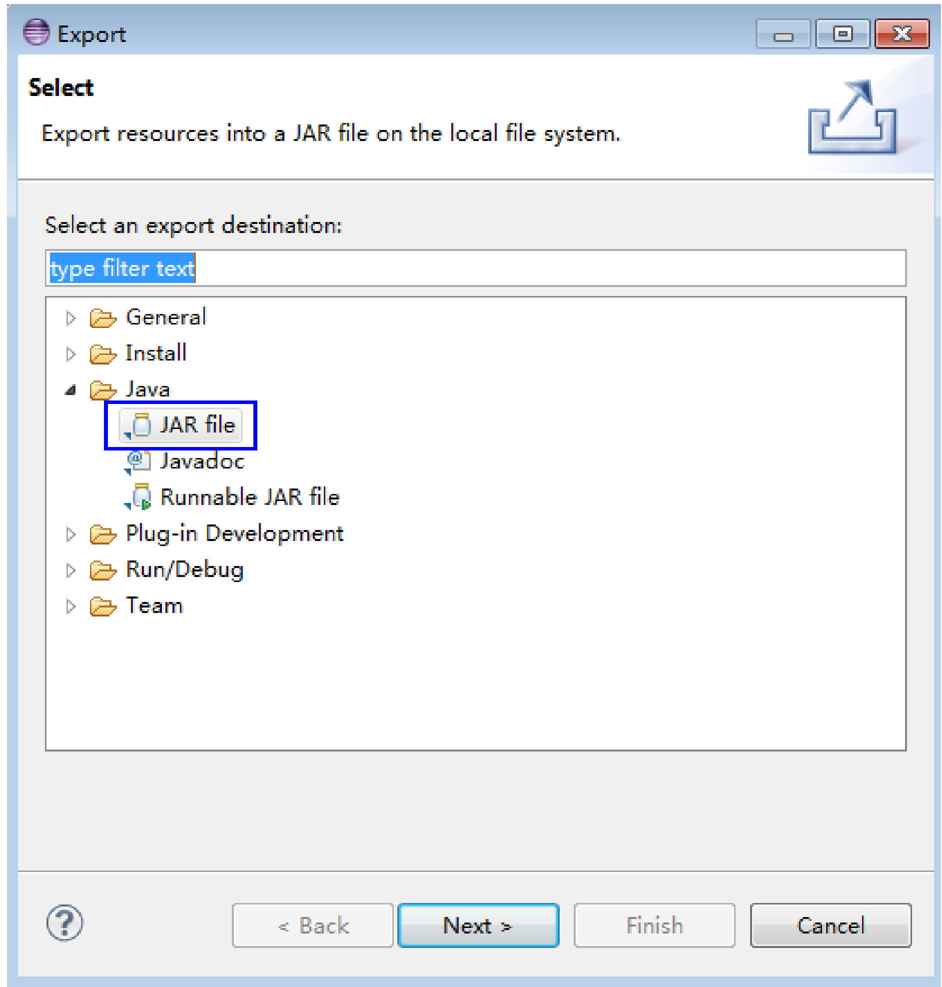
1. 右击样例工程，选择导出。

图 1-9 导出 Jar 包



2. 选择JAR file，单击“Next”。

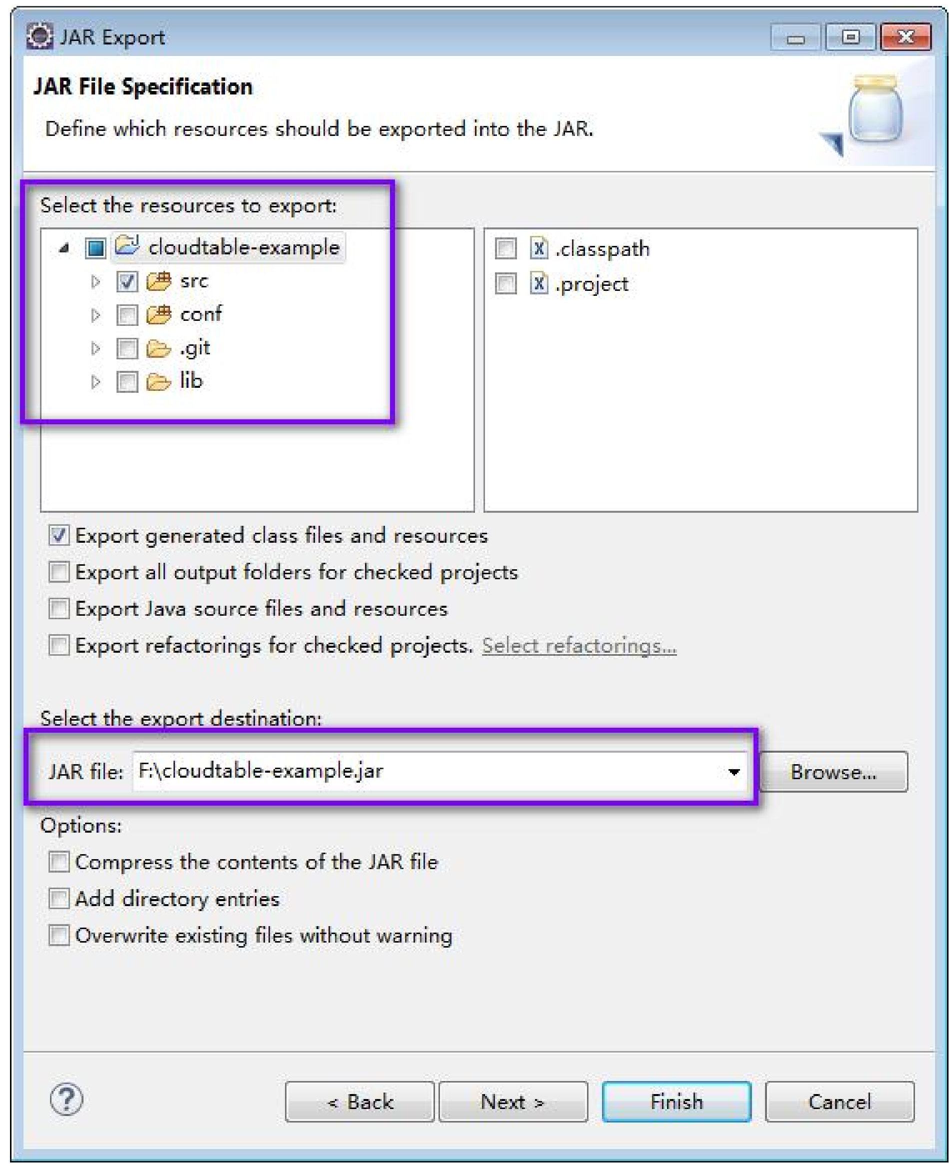
图 1-10 选择 JAR file



3. 勾选“src”和“conf”目录，导出Jar包到指定位置。单击两次“Next”。

版权所有 © 华为云计算技术有限公司

图 1-11 选择导出路径



4. 单击“Finish”，完成导出Jar包。

步骤2 执行Jar包。

1. 在Linux客户端下执行Jar包的时候，先将应用开发环境中生成的Jar包拷贝上传至客户端安装目录的“lib”目录中，并确保Jar包的文件权限与其它文件相同。
2. 用安装用户切换到客户端目录的“bin”目录下，然后运行如下命令使Jar包执行：

```
[Ruby@cloudtable-08261700-hmaster-1-1 bin]# ./hbase
com.huawei.cloudtable.hbase.examples.TestMain
```

   其中，*com.huawei.cloudtable.hbase.examples.TestMain*为举例，具体以实际样例代码为准。

----结束

版权所有 © 华为云计算技术有限公司

1.7.2.2 未安装客户端时编译并运行程序

操作场景

HBase应用程序支持在未安装HBase客户端的Linux环境中运行。在程序代码完成开发后，您可以上传Jar包至Linux环境中运行应用。

前提条件

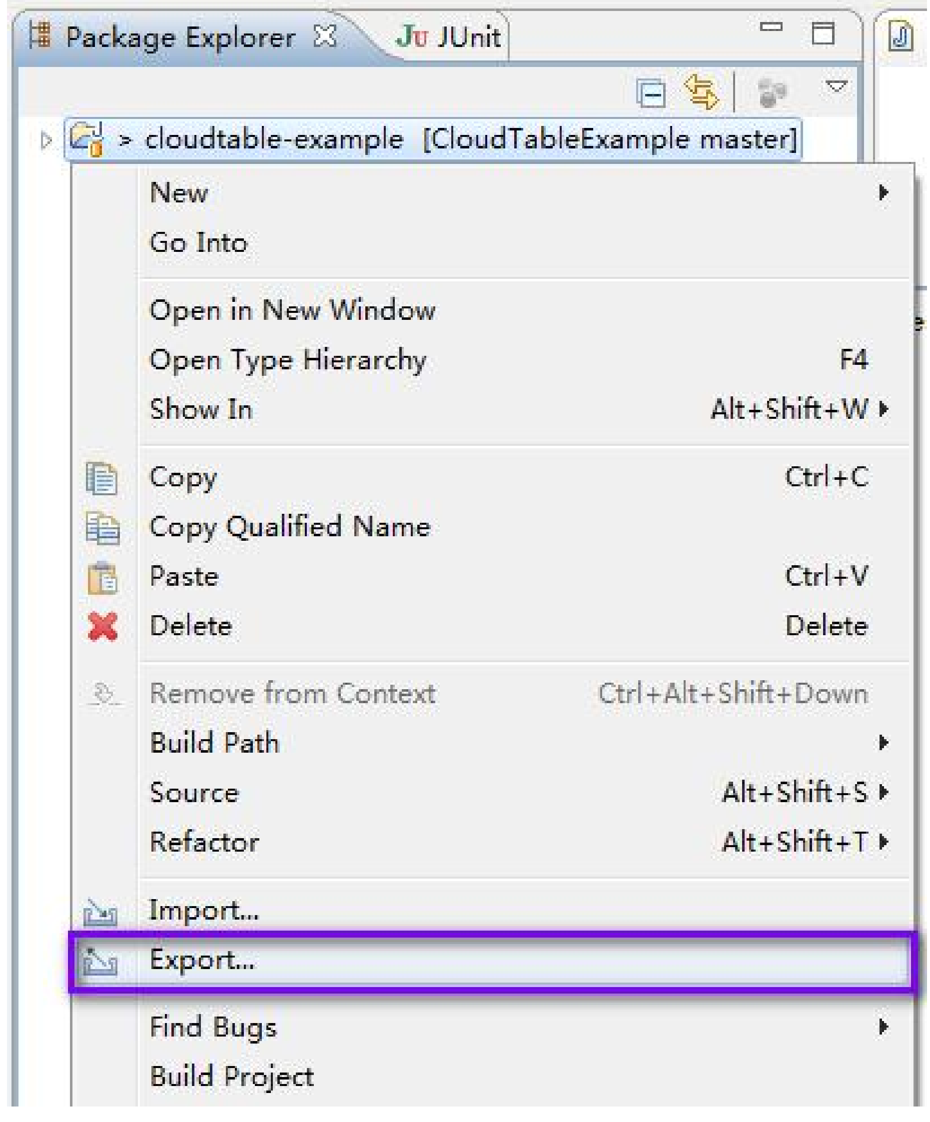
Linux环境已安装JDK，版本号需要和Eclipse导出Jar包使用的JDK版本一致。

操作步骤

步骤1 导出Jar包。

1. 右击样例工程，选择导出。

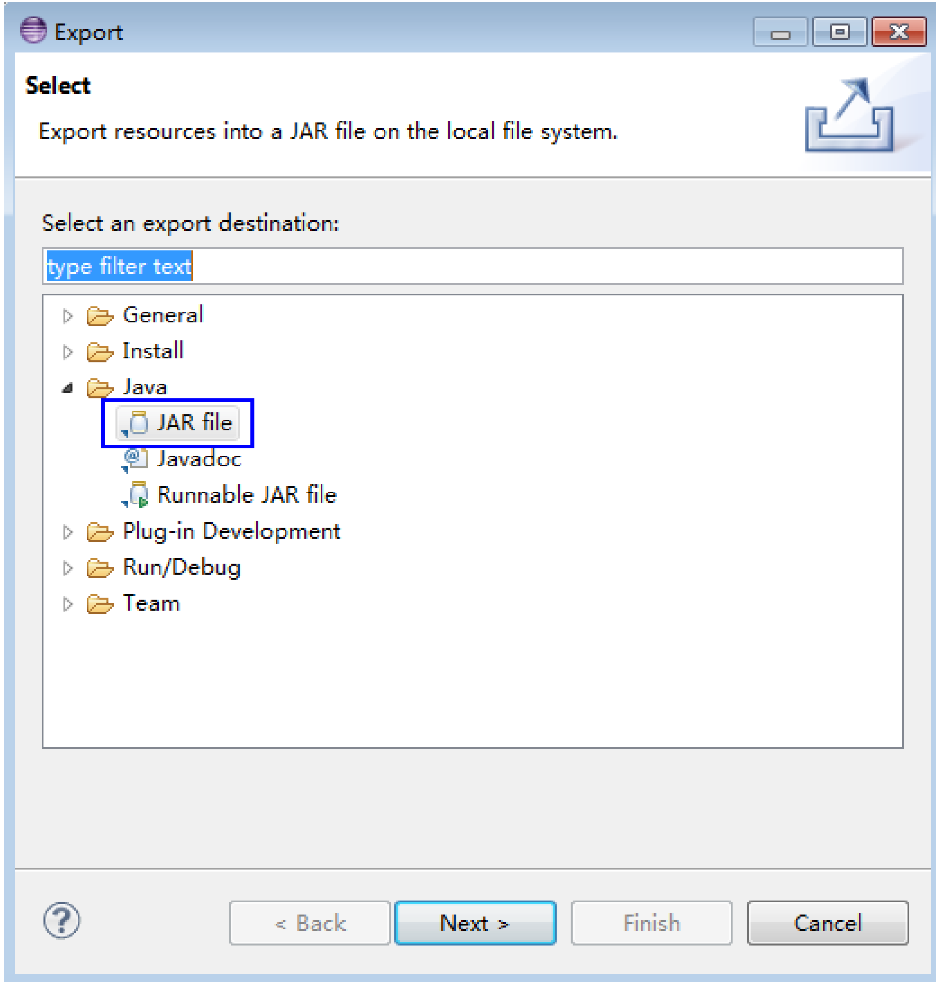
图 1-12 导出 Jar 包



2. 选择JAR file，单击“Next”。

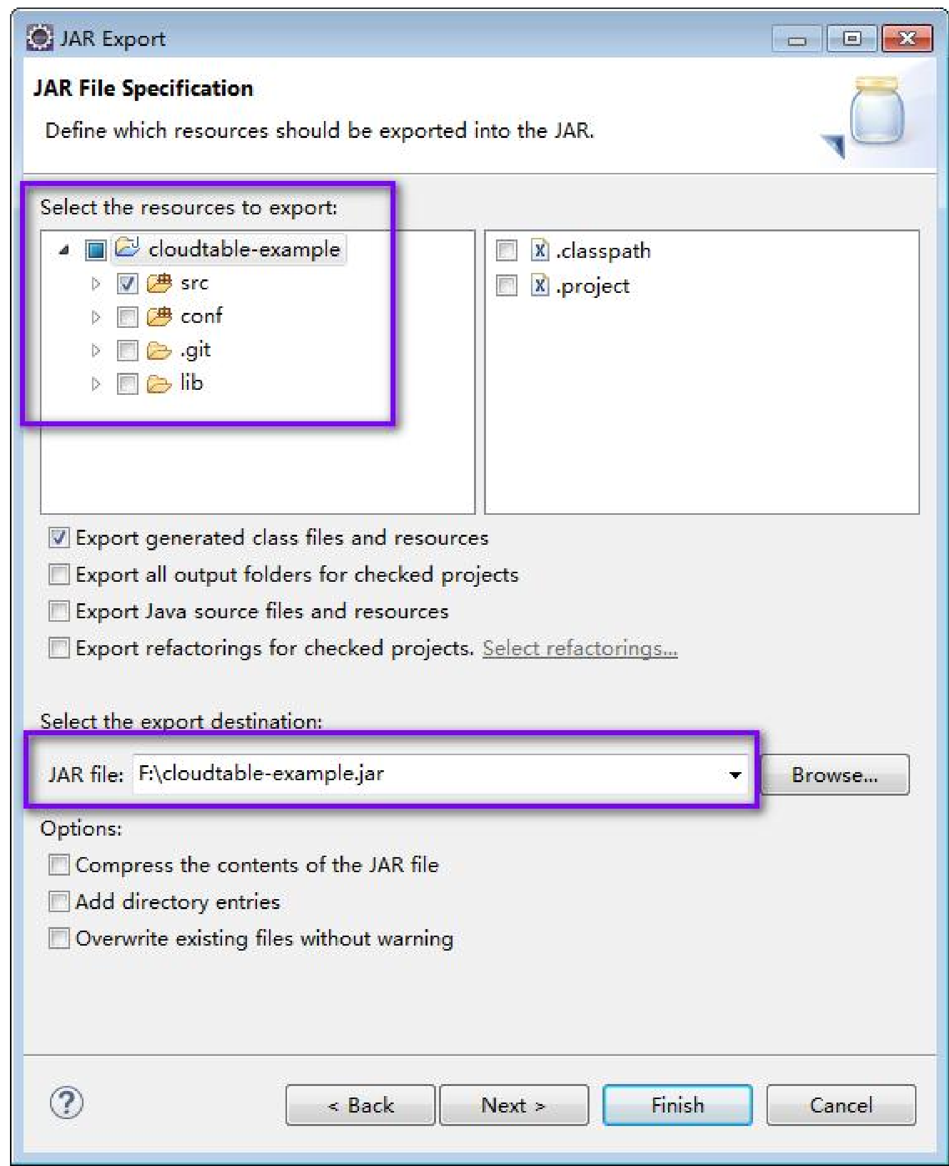
版权所有 © 华为云计算技术有限公司

图 **1-13** 选择 JAR file



3. 勾选“src”目录，导出Jar包到指定位置。单击两次“Next”。

图 1-14 选择导出路径



4. 单击“Finish”，完成导出Jar包。

步骤2 准备依赖的Jar包和配置文件。

1. 在Linux环境新建目录，例如“/opt/test”，并创建子目录“lib”和“conf”。将样例工程中“lib”的Jar包，以及步骤1导出的Jar包，上传到Linux的“lib”目录。将样例工程中“conf”的配置文件上传到Linux中“conf”目录。

2. 在“/opt/test”根目录新建脚本“run.sh”，修改内容如下并保存：

```
#!/bin/sh
BASEDIR=`pwd`
SECURE=""
if [ $# -eq 1 ]; then
  SECURE="-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -
Dzookeeper.client.secure=true"
fi
cd ${BASEDIR}
for file in ${BASEDIR}/lib/*.jar
do
i_cp=$i_cp:$file
echo "$file"
```


版权所有 © 华为云计算技术有限公司

```
done
for file in ${BASEDIR}/conf/*
do
i_cp=$i_cp:$file
done
java -cp .${i_cp} ${SECURE} com.huawei.cloudtable.hbase.examples.TestMain
```

步骤3 切换到“/opt/test”，执行以下命令，运行Jar包。

- **未开启加密通道的HBase集群**

  sh run.sh

- **开启加密通道的HBase集群**

  sh run.sh secure

说明

如果使用其他方式运行应用访问开启了加密通道的HBase集群，需要自行添加JVM参数："-Dzookeeper.clientCnxnSocket=org.apache.zookeeper.ClientCnxnSocketNetty -Dzookeeper.client.secure=true

----结束

1.7.2.3 查看调测结果

运行结果中没有异常或失败信息即表明运行成功。

图 1-15 运行成功

```
2016-07-13 14:36:12,736 INFO  [main] basic.CreateTableSample: Create table sampleNameSpace:sampleTable successful!
2016-07-13 14:36:15,426 INFO  [main] basic.ModifyTableSample: Modify table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:16,708 INFO  [main] basic.MultiSplitSample: Mmulti split table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:17,299 INFO  [main] basic.PutDataSample: Successfully put 9 items data into sampleNameSpace:sampleTable.
2016-07-13 14:36:18,992 INFO  [main] basic.ScanSample: Scan data successfully.
2016-07-13 14:36:20,532 INFO  [main] basic.DeletaDataSample: Successfully delete data from table sampleNameSpace:sampleTable.
2016-07-13 14:36:21,006 INFO  [main] acl.AclSample: Grant ACL for table sampleNameSpace:sampleTable successfully.
2016-07-13 14:36:27,836 INFO  [main] index.CreateIndexSample: Successfully add index for table sampleNameSpace:sampleTable.
```

日志说明：日志级别默认为INFO，可以通过调整日志打印级别（DEBUG，INFO，WARN，ERROR，FATAL）来显示更详细的信息。可以通过修改log4j.properties文件来实现，如：

```
hbase.root.logger=INFO,console
log4j.logger.org.apache.zookeeper=INFO
#log4j.logger.org.apache.hadoop.fs.FSNamesystem=DEBUG
log4j.logger.org.apache.hadoop.hbase=INFO
# Make these two classes DEBUG-level. Make them DEBUG to see more zk debug.
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZKUtil=INFO
log4j.logger.org.apache.hadoop.hbase.zookeeper.ZooKeeperWatcher=INFO
```

1.8 对外接口

1.8.1 HBase Java API

HBase采用的接口与Apache HBase保持一致。

详细内容请参见 **https://hbase.apache.org/1.2/apidocs/index.html**

版权所有 © 华为云计算技术有限公司

2 Doris 应用开发指导

2.1 Doris 组件使用规范

本章节介绍Doris组件使用规范。

建表规范

- 【强制】创建表指定分桶buckets时，每个桶的数据大小为应保持在100M-3G之间，单分区中最大分桶数据不超过5000。
- 【强制】表数据超过5亿条以上必须设置分区分桶策略。
- 【强制】分桶的列不要设置太多，一般情况下1或2个列，同时需要兼顾数据分布均匀和查询吞吐之间的均衡，考虑数据均匀是为了避免某些桶的数据存在倾斜影响数据均衡和查询效率，考虑查询吞吐是为了利用查询SQL的分桶剪裁优化避免全桶扫描提升查询性能，所以优先考虑哪些数据较为均匀且常用于查询条件的列适合做分桶列。
- 【强制】2000kw 以内数据禁止使用动态分区（动态分区会自动创建分区，而小表用户客户关注不到，会创建出大量不使用分区分桶）。
- 【强制】创建表时的副本数必须至少为2，默认是3，禁止使用单副本。
- 【建议】单表物化视图不能超过6个。
- 【建议】对于有大量历史分区数据，但是历史数据比较少，或者不均衡，或者查询概率的情况，使用如下方式将数据放在特殊分区：
  - 对于历史数据，如果数据量比较小我们可以创建历史分区（比如年分区，月分区），将所有历史数据放到对应分区里。
  - 创建历史分区方式：FROM ("2000-01-01") TO ("2022-01-01") INTERVAL 1 YEAR。
- 【建议】1000w-2亿以内数据为了方便可以不设置分区，直接用分桶策略（不设置其实Doris内部会有个默认分区）。
- 【建议】如果分桶字段存在30%以上的数据倾斜，则禁止使用Hash分桶策略，改使用random分桶策略：Create table ... DISTRIBUTED BY RANDOM BUCKETS 10 ...
- 【建议】建表时第一个字段一定是最常查询使用的列，默认有前缀索引快速查询能力，选取分区分桶外最长查询且高基数的列，前缀索引36位，如果列超长也不能使用前缀索引能力。

 版权所有 © 华为云计算技术有限公司

- 【建议】亿级别以上数据，如果有模糊匹配或者等值/in条件，可以使用倒排索引或者是 Bloomfilter。如果是低基数列的正交查询适合使用bitmap索引。
- 【强制】Doris 建表不要指定Merge-On-Write属性，当前有很多开源问题，不推荐。如使用了该属性，CloudTable服务不承诺SLA。

数据变更类

- 【强制】应用程序不可以直接使用delete后者update语句变更数据，可以使用CDC的upsert方式来实现。
  – 低频操作上使用，比如Update几分钟更新一次。
  – 如果使用Delete一定带上分区条件。
- 【强制】禁止使用INSERT INTO tbl1 VALUES (“1”), (“a”);这种方式做数据导入，少量少次写可以，多量多频次时要使用Doris提供的StreamLoad、BrokerLoad、SparkLoad或者Flink Connector方式。
- 【建议】执行特殊的长SQL操作时，可以使用SELECT /*+ SET_VAR(query_timeout = xxx*/ from table 类似这样通过Hint方式去设置Session 会话变量，不要设置全局的系统变量。

命名规范

- 【强制】数据库字符集指定utf-8，并且只支持utf-8。
- 【建议】库名统一使用小写方式，中间用下划线（_）分隔，长度62字节内。
- 【建议】表名称大小写敏感，统一使用小写方式，中间用下划线（_）分隔，长度64字节内。

数据查询规范

- 【强制】鉴于外表存在不稳定性，目前doris暂不支持外表查询。
- 【强制】in中条件超过2000后，必须修改为子查询。
- 【强制】禁止使用REST API（Statement Execution Action）执行大量SQL查询，该接口仅仅用于集群维护。
- 【建议】一次insert into select数据超过1亿条后，建议拆分为多个insert into select语句执行，分成多个批次来执行。如果非要这样执行不可，必须在集群资源相对空闲的时候可以通过调整并发度来加快的数据导入速度 。

  例如：set parallel_fragment_exec_instance_num = 8 建议数值是单BE节点上CPU内核的一半。
- 【强制】query查询条件返回结果在5w条以上，使用JDBC Catalog或者OUTFILE方式导出。不然大量FE上数据传输将占用FE资源，影响集群稳定性。
  – 如果是交互式查询，建议使用分页方式（offset limit），分页要加Order by。
  – 如果是数据导出提供给第三方使用，建议使用 outfile或者export 方式。
- 【强制】2个以上大于3亿的表JOIN使用Colocation Join。
- 【强制】亿级别大表禁止使用select * 查询，查询时需要明确要查询的字段。
  – 使用SQL Block方式禁止这种操作。
  – 如果是高并发点查，建议开启行存（2.x版本）。
  – 使用PreparedStatement查询。

- 【强制】亿级以上表数据查询必须带分区分桶条件。
- 【建议】尽量不要使用OR作为JOIN条件。
- 【建议】大量数据排序（5亿以上）后返回部分数据，建议先减少数据范围再执行排序，否则大量排序会影响性能。

  例如：将from table order by datatime desc limit 10优化为from table where datatime='2023-10-20' order by datatime desc limit 10。

数据导入

- 【建议】在Flink实时写入数据到Doris的场景下，CheckPoint设置的时间需要考虑每批次数据量，如果每批次数据太小会造成大量小文件，推荐值为60s。
- 【建议】建议低频攒批导入数据，平均单表导入批次间隔需大于30s，推荐间隔60s，一次导入10000~100000行数据。

2.2 建表

2.2.1 Doris 数据表和数据模型

2.2.1.1 数据表

在Doris中，数据以表的形式进行逻辑上的描述。表是具有相同模式的同质数据的集合。 一张表包括行（Row）和列（Column）。Row即用户的一行数据。Column用于描述一行数据中不同的字段，可以根据实际情况采用不同的数据类型（如整型、字符串、布尔型等）。

从OLAP场景看，Column可以分为两大类：Key和Value。Key表示维度列，Value表示指标列。

2.2.1.2 数据模型

Doris的数据模型主要分为3类：

- **Aggregate模型**。
- **Unique模型**。
- **Duplicate**。

Aggregate 模型

以实际的例子来说明什么是聚合模型，以及如何正确的使用聚合模型。

- 示例1：导入数据聚合

  假设业务有以下模式：

表 2-1 参数说明

ColumnName	Type	AggregationType	Comment
user_id	LARGEINT	-	用户 ID
date	DATE	-	数据导入日期
city	VARCHAR(20)	-	用户所在城市
age	SMALLINT	-	用户年龄
sex	TINYINT	-	用户性别
last_visit_date	DATETIME	REPLACE	用户最后一次访问时间
cost	BIGINT	SUM	用户总消费
max_dwell_time	INT	MAX	用户最大停留时间
min_dwell_time	INT	MIN	用户最小停留时间

转换成建表语句，如下所示。

```
CREATE TABLE IF NOT EXISTS demo.example_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "用户id",
    `date` DATE NOT NULL COMMENT "数据灌入日期时间",
    `city` VARCHAR(20) COMMENT "用户所在城市",
    `age` SMALLINT COMMENT "用户年龄",
    `sex` TINYINT COMMENT "用户性别",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "用户最后一次访问时间",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "用户总消费",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "用户最大停留时间",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "用户最小停留时间"
)
AGGREGATE KEY(`user_id`, `date`, `city`, `age`, `sex`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

可以看到，这是一个典型的用户信息和访问行为的事实表。在一般星型模型中，用户信息和访问行为一般分别存放在维度表和事实表中。这里我们为了更加方便的解释Doris的数据模型，将两部分信息统一存放在一张表中。

表中的列按照是否设置了AggregationType，分为Key（维度列）和Value（指标列）。没有设置AggregationType的，如user_id、date、age、sex称为Key，而设置了AggregationType的称为Value。

当导入数据时，对于Key列相同的行会聚合成一行，而Value列会按照设置的AggregationType进行聚合。AggregationType目前有以下四种聚合方式：

- SUM：求和，多行的Value进行累加。
- REPLACE：替代，下一批数据中的Value会替换之前导入过的行中的Value。
- MAX：保留最大值。

版权所有 © 华为云计算技术有限公司

– MIN：保留最小值。

表 2-2 原始数据

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	A	20	0	2017-10-01 06:00:00	20	10	10
10000	2017-10-01	A	20	0	2017-10-01 07:00:00	15	2	2
10001	2017-10-01	A	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	B	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	C	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	D	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	D	35	0	2017-10-03 10:20:22	11	6	6

我们假设这是一张记录用户访问某商品页面行为的表。我们以第一行数据为例，解释如下：

表 2-3 参数说明

数据	说明
10000	用户id，每个用户唯一识别id
2017-10-01	数据入库时间，精确到日期
A	用户所在城市

版权所有 © 华为云计算技术有限公司

数据	说明
20	用户年龄
0	性别男（1 代表女性）
2017-10-01 06:00:00	用户本次访问该页面的时间，精确到秒
20	用户本次访问产生的消费
10	用户本次访问，驻留该页面的时间
10	用户本次访问，驻留该页面的时间（冗余）

那么当这批数据正确导入到Doris中后，Doris中最终存储如下：

表 2-4 插入数据

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	A	20	0	2017-10-01 07:00:00	35	10	2
10001	2017-10-01	A	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	B	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	C	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	D	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	D	35	0	2017-10-03 10:20:22	11	6	6

可以看到，用户10000只剩下了一行聚合后的数据。而其余用户的数据和原始数据保持一致。这里先解释下用户10000 聚合后的数据：

前5列没有变化，从第6列 last_visit_date 开始：

- 2017-10-01 07:00:00：因为last_visit_date列的聚合方式为REPLACE，所以2017-10-01 07:00:00替换了2017-10-01 06:00:00保存了下来。

  说明

  在同一个导入批次中的数据，对于REPLACE这种聚合方式，替换顺序不做保证。如在这个例子中，最终保存下来的，也有可能是2017-10-01 06:00:00。而对于不同导入批次中的数据，可以保证，后一批次的数据会替换前一批次。
- 35：因为cost列的聚合类型为SUM，所以由20+15累加获得35。
- 10：因为max_dwell_time列的聚合类型为MAX，所以10和2取最大值，获得10。
- 2：因为min_dwell_time列的聚合类型为MIN，所以10和2取最小值，获得2。

  经过聚合，Doris中最终只会存储聚合后的数据。换句话说，即明细数据会丢失，用户不能够再查询到聚合前的明细数据了。

- 示例2：保留，明细数据。

  接示例1，将表结构修改如下：

表 2-5 参数说明

ColumnName	Type	AggregationType	Comment
user_id	LARGEINT	-	用户 ID
date	DATE	-	数据导入日期
timestamp	DATETIME	-	数据导入时间，精确到秒
city	VARCHAR(20)	-	用户所在城市
age	SMALLINT	-	用户年龄
sex	TINYINT	-	用户性别
last_visit_date	DATETIME	REPLACE	用户最后一次访问时间
cost	BIGINT	SUM	用户总消费
max_dwell_time	INT	MAX	用户最大停留时间
min_dwell_time	INT	MIN	用户最小停留时间

即增加了一列timestamp，记录精确到秒的数据导入时间。

同时，将AGGREGATE KEY设置为AGGREGATE KEY(user_id, date, timestamp, city, age, sex)。

版权所有 © 华为云计算技术有限公司

导入数据如下：

表 2-6 原始数据

user_id	date	timestamp	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	2017-10-01 08:00:05	A	20	0	2017-10-01 06:00:00	20	10	10
10000	2017-10-01	2017-10-01 09:00:05	A	20	0	2017-10-01 07:00:00	15	2	2
10001	2017-10-01	2017-10-01 18:12:10	A	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	2017-10-02 13:10:00	B	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	2017-10-02 13:15:00	C	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	2017-10-01 12:12:48	D	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	2017-10-03 12:38:20	D	35	0	2017-10-03 10:20:22	11	6	6

那么当这批数据正确导入到Doris中后，Doris中最终存储如下：

版权所有 © 华为云计算技术有限公司

表 2-7 数据结果

user_id	date	timestamp	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	2017-10-01 08:00:05	A	20	0	2017-10-01 06:00:00	20	10	10
10000	2017-10-01	2017-10-01 09:00:05	A	20	0	2017-10-01 07:00:00	15	2	2
10001	2017-10-01	2017-10-01 18:12:10	A	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	2017-10-02 13:10:00	B	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	2017-10-02 13:15:00	C	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	2017-10-01 12:12:48	D	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	2017-10-03 12:38:20	D	35	0	2017-10-03 10:20:22	11	6	6

- 示例3：导入数据与已有数据聚合。

  接示例1中的参数列表，插入以下表中数据。

版权所有 © 华为云计算技术有限公司

表 2-8 原始数据

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	A	20	0	2017-10-01 07:00:00	35	10	2
10001	2017-10-01	A	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	B	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	C	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	D	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	D	35	0	2017-10-03 10:20:22	11	6	6

在导入一批新的数据：

表 2-9 新数据

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10004	2017-10-03	D	35	0	2017-10-03 11:22:00	44	19	19
10005	2017-10-03	E	29	1	2017-10-03 18:11:02	3	1	1

版权所有 © 华为云计算技术有限公司

那么当这批数据正确导入到Doris中后，Doris中最终存储如下:

表 2-10

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017-10-01	A	20	0	2017-10-01 07:00:00	35	10	2
10001	2017-10-01	A	30	1	2017-10-01 17:05:45	2	22	22
10002	2017-10-02	B	20	1	2017-10-02 12:59:12	200	5	5
10003	2017-10-02	C	32	0	2017-10-02 11:20:00	30	11	11
10004	2017-10-01	D	35	0	2017-10-01 10:00:15	100	3	3
10004	2017-10-03	D	35	0	2017-10-03 11:22:00	55	19	6
10005	2017-10-03	E	29	1	2017-10-03 18:11:02	3	1	1

可以看到，用户10004的已有数据和新导入的数据发生了聚合。同时新增了10005用户的数据。

数据的聚合，在Doris中有如下三个阶段发生：

- 每一批次数据导入的ETL阶段。该阶段会在每一批次导入的数据内部进行聚合。
- 底层BE进行数据Compaction的阶段。该阶段，BE会对已导入的不同批次的数据进行进一步的聚合。
- 数据查询阶段。在数据查询时，对于查询涉及到的数据，会进行对应的聚合。

版权所有 © 华为云计算技术有限公司

数据在不同时间，可能聚合的程度不一致。例如一批数据刚导入时，可能还未与之前已存在的数据进行聚合。但是对于用户而言，用户只能查询到聚合后的数据。即不同的聚合程度对于用户查询而言是透明的。用户需始终认为数据以最终的完成的聚合程度存在，而不应假设某些聚合还未发生。

Unique 模型

在某些多维分析场景下，用户更关注的是如何保证Key的唯一性，即如何获得Primary Key唯一性约束。因此，我们引入了Unique的数据模型。该模型本质上是聚合模型的一个特例，也是一种简化的表结构表示方式。举例说明：

说明

Unique模型表，不推荐开启merge-on-write属性，默认使用merge-on-read。

表 2-11 参数说明

ColumnName	Type	IsKey	Comment
user_id	BIGINT	Yes	用户 ID
username	VARCHAR(50)	Yes	用户昵称
city	VARCHAR(20)	No	用户所在城市
age	SMALLINT	No	用户年龄
sex	TINYINT	No	用户性别
phone	LARGEINT	No	用户电话
address	VARCHAR(500)	No	用户住址
register_time	DATETIME	No	用户注册时间

这是一个典型的用户基础信息表。这类数据没有聚合需求，只需保证主键唯一性。（这里的主键为user_id+username）。那么我们的建表语句如下：

```
CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "用户id",
    `username` VARCHAR(50) NOT NULL COMMENT "用户昵称",
    `city` VARCHAR(20) COMMENT "用户所在城市",
    `age` SMALLINT COMMENT "用户年龄",
    `sex` TINYINT COMMENT "用户性别",
    `phone` LARGEINT COMMENT "用户电话",
    `address` VARCHAR(500) COMMENT "用户地址",
    `register_time` DATETIME COMMENT "用户注册时间"
)
UNIQUE KEY(`user_id`, `username`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

而这个表结构，完全同等于以下使用聚合模型描述的表结构：

 版权所有 © 华为云计算技术有限公司

表 2-12 参数说明

ColumnName	Type	AggregationType	Comment
user_id	BIGINT	-	用户 ID
username	VARCHAR(50)	-	用户昵称
city	VARCHAR(20)	REPLACE	用户所在城市
age	SMALLINT	REPLACE	用户年龄
sex	TINYINT	REPLACE	用户性别
phone	LARGEINT	REPLACE	用户电话
address	VARCHAR(500)	REPLACE	用户住址
register_time	DATETIME	REPLACE	用户注册时间

建表语句。

```
CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "用户id",
    `username` VARCHAR(50) NOT NULL COMMENT "用户昵称",
    `city` VARCHAR(20) REPLACE COMMENT "用户所在城市",
    `age` SMALLINT REPLACE COMMENT "用户年龄",
    `sex` TINYINT REPLACE COMMENT "用户性别",
    `phone` LARGEINT REPLACE COMMENT "用户电话",
    `address` VARCHAR(500) REPLACE COMMENT "用户地址",
    `register_time` DATETIME REPLACE COMMENT "用户注册时间"
)
AGGREGATE KEY(`user_id`, `username`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

即Unique模型完全可以用聚合模型中的REPLACE方式替代。其内部的实现方式和数据存储方式也完全一样。

Duplicate 模型

在某些多维分析场景下，数据既没有主键，也没有聚合需求。因此，我们引入Duplicate数据模型来满足这类需求。

表 2-13 数据

ColumnName	Type	SortKey	Comment
timestamp	DATETIME	Yes	日志时间
type	INT	Yes	日志类型
error_code	INT	Yes	错误码
error_msg	VARCHAR(1024)	No	错误详细信息

版权所有 © 华为云计算技术有限公司

ColumnName	Type	SortKey	Comment
op_id	BIGINT	No	负责人 ID
op_time	DATETIME	No	处理时间

建表语句。

```
CREATE TABLE IF NOT EXISTS example_db.expamle_tbl
(
    `timestamp` DATETIME NOT NULL COMMENT "日志时间",
    `type` INT NOT NULL COMMENT "日志类型",
    `error_code` INT COMMENT "错误码",
    `error_msg` VARCHAR(1024) COMMENT "错误详细信息",
    `op_id` BIGINT COMMENT "负责人id",
    `op_time` DATETIME COMMENT "处理时间"
)
DUPLICATE KEY(`timestamp`, `type`)
DISTRIBUTED BY HASH(`type`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

这种数据模型区别于Aggregate和Unique模型。数据完全按照导入文件中的数据进行存储，不会有任何聚合。即使两行数据完全相同，也都会保留。 而在建表语句中指定的DUPLICATE KEY，只是用来指明底层数据按照哪些列进行排序。（更贴切的名称应该为“Sorted Column”，这里取名“DUPLICATE KEY”只是用以明确表示所用的数据模型。在DUPLICATE KEY的选择上，我们建议适当地选择前2-4列就可以。

这种数据模型适用于既没有聚合需求，又没有主键唯一性约束的原始数据的存储。更多使用场景，可参见聚合模型的局限性。

聚合模型的局限

- Aggregate模型&Unique模型。

  这里我们针对Aggregate模型（包括Unique模型），来介绍下聚合模型的局限性。

  在聚合模型中，模型对外展现的，是最终聚合后的数据。也就是说，对于任何还未聚合的数据（例如说两个不同导入批次的数据），必须通过某种方式保证对外展示的一致性。

  假设表结构如下：

  表 2-14 数据

ColumnName	Type	AggregationType	Comment
user_id	LARGEINT	-	用户ID
date	DATE	-	数据导入日期
cost	BIGINT	SUM	用户总消费

  假设存储引擎中有如下两个已经导入完成的批次的数据。

batch1

表 2-15 数据

user_id	date	cost
10001	2017-11-20	50
10002	2017-11-21	39

batch2

表 2-16 数据

user_id	date	cost
10001	2017-11-20	1
10001	2017-11-21	5
10003	2017-11-22	22

可以看到，用户10001分属在两个导入批次中的数据还没有聚合。但是为了保证用户只能查询到如下最终聚合后的数据。

表 2-17 数据

user_id	date	cost
10001	2017-11-20	51
10001	2017-11-21	5
10002	2017-11-21	39
10003	2017-11-22	22

建数据表。

```
CREATE TABLE IF NOT EXISTS example_db.expamle_tb2
(
    `user_id` LARGEINT NOT NULL COMMENT "用户id",
    `date` DATE NOT NULL COMMENT "数据导入日期时间",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "用户总消费"
)
AGGREGATE KEY(`user_id`, `date`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

插入**表15**、**表16**的数据。

版权所有 © 华为云计算技术有限公司

```
INSERT INTO example_db.expamle_tb2 (user_id,date,cost) VALUES('10001','2017-11-20','50'),
('10002','2017-11-21','39'),('10001','2017-11-20','1'),('10001','2017-11-21','5'),
('10003','2017-11-22','22');
```

我们在查询引擎中加入了聚合算子，来保证数据对外的一致性。

另外，在聚合列（Value）上，执行与聚合类型不一致的聚合类查询时，要注意语意。例如我们在如上示例中执行如下查询：

```
mysql> SELECT MIN(cost) FROM example_db.expamle_tb2;
+-------------+
| min(`cost`) |
+-------------+
|           5 |
+-------------+
1 row in set (0.02 sec)
```

得到的结果是5，而不是1。

同时，这种一致性保证，在某些查询中，会极大降低查询效率。

我们以最基本的count(*)查询为例：

```
SELECT COUNT(*) FROM table;
```

在其他数据库中，这类查询都会很快返回结果。因为在实现上，我们可以通过如导入时对行进行计数，保存count的统计信息，或者在查询时仅扫描某一列数据，获得count值的方式，只需很小的开销，即可获得查询结果。但是在Doris的聚合模型中，这种查询的开销非常大。以刚才的数据为例。

batch1

表 2-18 数据

user_id	date	cost
10001	2017-11-20	50
10002	2017-11-21	39

batch2

表 2-19 数据

user_id	date	cost
10001	2017-11-20	1
10001	2017-11-21	5
10003	2017-11-22	22

最终聚合结果如下表所示。

表 2-20 数据

user_id	date	cost
10001	2017-11-20	51

版权所有 © 华为云计算技术有限公司

user_id	date	cost
10001	2017-11-21	5
10002	2017-11-21	39
10003	2017-11-22	22

查询结果。

```
mysql> SELECT COUNT(date) FROM example_db.expamle_tb2;
+---------------+
| count(`date`) |
+---------------+
|             4 |
+---------------+
1 row in set (0.01 sec)
```

所以，select count(*) from table; 的正确结果应该为4。但如果我们只扫描user_i 这一列，如果加上查询时聚合，最终得到的结果是3（10001, 10002, 10003）。而如果不加查询时聚合，则得到的结果是5（两批次一共5行数据）。可见这两个结果都是不对的。

为了得到正确的结果，我们必须同时读取user_id和date这两列的数据，再加上查询时聚合，才能返回4这个正确的结果。也就是说，在count(*) 查询中，Doris必须扫描所有的AGGREGATE KEY列（这里就是user_id 和 date），并且聚合后，才能得到语意正确的结果。当聚合列非常多时，count(*)查询需要扫描大量的数据。

因此，当业务上有频繁的count(*)查询时，我们建议用户通过增加一个值恒为1的，聚合类型为SUM的列来模拟count(*)。如刚才的例子中的表结构，我们修改如下：

表 2-21 参数说明

ColumnName	Type	AggregateType	Comment
user_id	BIGINT	-	用户 ID
date	DATE	-	数据导入日期
cost	BIGINT	SUM	用户总消费
count	BIGINT	SUM	用于计算 count

增加一个count列，并且导入数据中，该列值恒为1。则select count(*) from table; 的结果等价于select sum(count) from table;。而后者的查询效率将远高于前者。不过这种方式也有使用限制，就是用户需要自行保证，不会重复导入AGGREGATE KEY列都相同的行。否则，select sum(count) from table; 只能表述原始导入的行数，而不是select count(*) from table; 的语义，前者值会错误的增大。

另一种方式，就是 将如上的count列的聚合类型改为REPLACE，且依然值恒为1。那么select sum(count) from table; 和select count(*) from table; 的结果将是一致的。并且这种方式，没有导入重复行的限制。

版权所有 © 华为云计算技术有限公司

- Duplicate

  Duplicate模型没有聚合模型的这个局限性。因为该模型不涉及聚合语意，在做count(*)查询时，任意选择一列查询，即可得到语意正确的结果。

2.2.1.3 最佳实践

因为数据模型在建表时就已经确定，且无法修改。所以，选择一个合适的数据模型非常重要。

数据模型选择

Doris数据模型上目前分为三类：AGGREGATE KEY，UNIQUE KEY，DUPLICATE KEY。三种模型中数据都是按KEY进行排序。

- Aggregate模型。

  Aggregate模型可以通过预聚合，极大地降低聚合查询时所需扫描的数据量和查询的计算量，非常适合有固定模式的报表类查询场景。但是该模型对count(*) 查询很不友好。同时因为固定了Value列上的聚合方式，在进行其他类型的聚合查询时，需要考虑语意正确性。

  Aggregate Key相同时，新旧记录进行聚合，目前支持的聚合函数有SUM，MIN，MAX，REPLACE。

```
CREATE TABLE site_visit
(
    siteid      INT,
    city        SMALLINT,
    username    VARCHAR(32),
    pv BIGINT   SUM DEFAULT '0'
)
AGGREGATE KEY(siteid, city, username)
DISTRIBUTED BY HASH(siteid) BUCKETS 10;
```

- Unique模型。

  Unique模型针对需要唯一主键约束的场景，Unique key相同时，新记录覆盖旧记录，可以保证主键唯一性约束。适用于有更新需求的分析业务。目前Unique key实现上和Aggregate key的 REPLACE聚合方法一样，二者本质上相同。但是无法利用ROLLUP等预聚合带来的查询优势（因为本质是REPLACE，没有SUM这种聚合方式）。

```
CREATE TABLE sales_order
(
    orderid     BIGINT,
    status      TINYINT,
    username    VARCHAR(32),
    amount      BIGINT DEFAULT '0'
)
UNIQUE KEY(orderid)
DISTRIBUTED BY HASH(orderid) BUCKETS 10;
```

- Duplicate模型。

  Duplicate模型相同的行不会合并，适合任意维度的Ad-hoc查询。虽然无法利用预聚合的特性，但是不受聚合模型的约束，可以发挥列存模型的优势（列裁剪、向量执行等）。

```
CREATE TABLE session_data
(
    visitorid   SMALLINT,
    sessionid   BIGINT,
    visittime   DATETIME,
    city        CHAR(20),
    province    CHAR(20),
```

```
    ip          varchar(32),
    brower      CHAR(20),
    url         VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid)
DISTRIBUTED BY HASH(sessionid, visitorid) BUCKETS 10;
```

大宽表与 Star Schema

业务方建表时, 为了和前端业务适配, 往往不对维度信息和指标信息加以区分, 而将Schema定义成大宽表，这种操作对于数据库其实不是那么友好，我们更建议用户采用星型模型。

- Schema中字段数比较多, 聚合模型中可能key列比较多, 导入过程中需要排序的列会增加。
- 维度信息更新会反应到整张表中，而更新的频率直接影响查询的效率。

使用过程中，建议用户尽量使用Star Schema区分维度表和指标表。频繁更新的维度表也可以放在MySQL外部表中。而如果只有少量更新, 可以直接放在Doris中。在Doris中存储维度表时，可对维度表设置更多的副本，提升Join的性能。

2.2.2 数据分区和分桶

Doris支持两层的数据划分。第一层是Partition（分区），支持Range（按范围）和List（按枚举值）的划分方式。第二层是Bucket（分桶），仅支持Hash的划分方式。分区和分桶都是对数据进行横向分隔。

也可以仅使用一层分区。使用一层分区时，只支持Bucket划分。下面我们来分别介绍下分区以及分桶。

2.2.2.1 分区（Partition）

分区用于将数据划分成不同区间，逻辑上可以理解为将原始表划分成了多个子表。可以方便的按分区对数据进行管理。

- Partition列可以指定一列或多列，分区列必须为KEY列。多列分区的使用方式在后面多列分区小结介绍。
- 不论分区列是什么类型，在写分区值时，都需要加双引号。
- 分区数量理论上没有上限。
- 当不使用Partition建表时，系统会自动生成一个和表名同名的，全值范围的Partition。该Partition对用户不可见，并且不可删改。
- 创建分区时不可添加范围重叠的分区。

Range 分区

分区列通常为时间列，以方便的管理新旧数据。

Partition支持通过VALUES LESS THAN (...) 仅指定上界，系统会将前一个分区的上界作为该分区的下界，生成一个左闭右开的区间。

- 通过VALUES[...) 同时指定上下界比较容易理解。这里举例说明，当使用VALUES LESS THAN (...) 语句进行分区的增删操作时，分区范围的变化情况。

```
CREATE TABLE IF NOT EXISTS example_db.expamle_range_tbl
    (
        `user_id` LARGEINT NOT NULL COMMENT "用户id",
```


版权所有 © 华为云计算技术有限公司

```
    `date` DATE NOT NULL COMMENT "数据灌入日期时间",
    `timestamp` DATETIME NOT NULL COMMENT "数据灌入的时间戳",
    `city` VARCHAR(20) COMMENT "用户所在城市",
    `age` SMALLINT COMMENT "用户年龄",
    `sex` TINYINT COMMENT "用户性别",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "用户最后一次访问时
间",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "用户总消费",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "用户最大停留时间",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "用户最小停留时间"
  )
  ENGINE=OLAP
  AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
  PARTITION BY RANGE(`date`)
  (
    PARTITION `p201701` VALUES LESS THAN ("2017-02-01"),
    PARTITION `p201702` VALUES LESS THAN ("2017-03-01"),
    PARTITION `p201703` VALUES LESS THAN ("2017-04-01")
  )
  DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
  PROPERTIES
  (
    "replication_num" = "3"
  );
```

- 查看分区。

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+---------------+----------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                      | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+---------------+----------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
| 16040       | p201701       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false      |
tag.location.default: 3 |
| 16041       | p201702       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false      |
tag.location.default: 3 |
| 16042       | p201703       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-03-01]; ..types: [DATE]; keys: [2017-04-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false      |
tag.location.default: 3 |
+-------------+---------------+----------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
3 rows in set (0.01 sec)
```

- 增加一个分区。

```
mysql> alter table example_db.expamle_range_tbl add partition p201705 VALUES LESS THAN
("2017-06-01");
Query OK, 0 rows affected (0.02 sec)
```

查看分区。

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+---------------+----------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+-----------------+---------
+----------------+---------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                      | DistributionKey | Buckets | ReplicationNum |
```


版权所有 © 华为云计算技术有限公司

```
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+-------------------+------------------------+----------
+------------+------------------------+
| 16040       | p201701       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16041       | p201702       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16042       | p201703       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-03-01]; ..types: [DATE]; keys: [2017-04-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16237       | p201705       | 1              | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+-------------------+------------------------+----------
+------------+------------------------+
4 rows in set (0.00 sec)
```

- 删除分区。

```
mysql> alter table example_db.expamle_range_tbl drop partition p201703;
Query OK, 0 rows affected (0.01 sec)
```

查看分区。

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+-------------------+------------------------+----------
+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                  | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+-------------------+------------------------+----------
+------------+------------------------+
| 16040       | p201701       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16041       | p201702       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16237       | p201705       | 1              | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------+----------------+---------
+---------------+--------------+-------------------+-------------------+------------------------+----------
+------------+------------------------+
3 rows in set (0.00 sec)
```

说明

p201702和p201705的分区范围并没有发生变化，而这两个分区之间，出现了一个空洞：[2017-03-01, 2017-04-01)。即如果导入的数据范围在这个空洞范围内，是无法导入的。

版权所有 © 华为云计算技术有限公司

- 继续删除分区。

```
mysql> alter table example_db.expamle_range_tbl drop partition p201702;
Query OK, 0 rows affected (0.00 sec)
```

查看分区。

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+--------------+---------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+----------------+---------
+----------------+--------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                     | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+----------------+---------
+----------------+--------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
| 16040       | p201701       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16237       | p201705       | 1              | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
+-------------+--------------+---------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+----------------+---------
+----------------+--------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
2 rows in set (0.00 sec)
```

说明

空洞范围变为：[2017-02-01, 2017-04-01)。

- 增加新分区。

```
mysql> alter table example_db.expamle_range_tbl add partition p201702new VALUES LESS THAN
("2017-03-01");
Query OK, 0 rows affected (0.01 sec)
```

查看分区。

```
mysql> show partitions from example_db.expamle_range_tbl;
+-------------+--------------+---------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+----------------+---------
+----------------+--------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                     | DistributionKey | Buckets | ReplicationNum |
StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize |
IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+----------------+---------
+----------------+--------------+--------------------+--------------------+-------------------------+----------
+------------+------------------------+
| 16040       | p201701       | 1              | 2023-04-11 07:35:02 | NORMAL | date         | [types: [DATE];
keys: [0000-01-01]; ..types: [DATE]; keys: [2017-02-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16302       | p201702new    | 1              | 2023-04-11 08:14:25 | NORMAL | date         | [types:
[DATE]; keys: [2017-02-01]; ..types: [DATE]; keys: [2017-03-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
| 16237       | p201705       | 1              | 2023-04-11 07:45:18 | NORMAL | date         | [types: [DATE];
keys: [2017-04-01]; ..types: [DATE]; keys: [2017-06-01]; ) | user_id         | 16      | 3              |
HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      |
tag.location.default: 3 |
+-------------+--------------+---------------+---------------------+--------+--------------
+--------------------------------------------------------------------------+----------------+---------
```

 版权所有 © 华为云计算技术有限公司

```
+----------------+---------------+---------------------+---------------------+-------------------------+----------
+------------+------------------------+
3 rows in set (0.00 sec)
```

说明

综上，分区的删除不会改变已存在分区的范围。删除分区可能出现空洞。通过VALUES LESS THAN语句增加分区时，分区的下界紧接上一个分区的上界。

- 多列分区。

  Range分区除了上述我们看到的单列分区，也支持多列分区。

```
CREATE TABLE IF NOT EXISTS example_db.expamle_range_multi_partiton_key_tbl
(
`user_id` LARGEINT NOT NULL COMMENT "用户id",
`date` DATE NOT NULL COMMENT "数据灌入日期时间",
`timestamp` DATETIME NOT NULL COMMENT "数据灌入的时间戳",
`city` VARCHAR(20) COMMENT "用户所在城市",
`age` SMALLINT COMMENT "用户年龄",
`sex` TINYINT COMMENT "用户性别",
`last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "用户最后一次访问时间",
`cost` BIGINT SUM DEFAULT "0" COMMENT "用户总消费",
`max_dwell_time` INT MAX DEFAULT "0" COMMENT "用户最大停留时间",
`min_dwell_time` INT MIN DEFAULT "99999" COMMENT "用户最小停留时间"
)
ENGINE=OLAP
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY RANGE(`date`, `user_id`)
(
PARTITION `p201701_1000` VALUES LESS THAN ("2017-02-01", "1000"),
PARTITION `p201702_2000` VALUES LESS THAN ("2017-03-01", "2000"),
PARTITION `p201703_all`  VALUES LESS THAN ("2017-04-01")
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
"replication_num" = "3"
);
```

  在以上示例中，我们指定date（DATE 类型）和user_id（INT 类型）作为分区列。以上示例最终得到的分区如下：

```
mysql> show partitions from example_db.expamle_range_multi_partiton_key_tbl;
+-------------+---------------+----------------+---------------------+--------+---------------
+------------------------------------------------------------------------------------------------------------------
--------------------------+-----------------+---------+----------------+---------------+---------------------
+---------------------+------------------------+----------+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey  |
Range                                                                                                             |
DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        |
RemoteStoragePolicy | LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+---------------+----------------+---------------------+--------+---------------
+------------------------------------------------------------------------------------------------------------------
--------------------------+-----------------+---------+----------------+---------------+---------------------
+---------------------+------------------------+----------+------------+-------------------------+
| 16367       | p201701_1000  | 1              | 2023-04-11 08:28:12 | NORMAL | date, user_id | [types:
[DATE, LARGEINT]; keys: [0000-01-01, -170141183460469231731687303715884105728]; ..types:
[DATE, LARGEINT]; keys: [2017-02-01, 1000]; ) | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false      | tag.location.default: 3 |
| 16368       | p201702_2000  | 1              | 2023-04-11 08:28:12 | NORMAL | date, user_id | [types:
[DATE, LARGEINT]; keys: [2017-02-01, 1000]; ..types: [DATE, LARGEINT]; keys: [2017-03-01,
2000]; )                                | user_id         | 16      | 3              | HDD           | 9999-12-31
15:59:59 |                     | NULL                     | 0.000    | false      | tag.location.default: 3 |
| 16369       | p201703_all   | 1              | 2023-04-11 08:28:12 | NORMAL | date, user_id | [types:
[DATE, LARGEINT]; keys: [2017-03-01, 2000]; ..types: [DATE, LARGEINT]; keys: [2017-04-01,
-170141183460469231731687303715884105728]; ) | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false      | tag.location.default: 3 |
+-------------+---------------+----------------+---------------------+--------+---------------
+------------------------------------------------------------------------------------------------------------------
```


版权所有 © 华为云计算技术有限公司

```
---------------------------+-----------------+---------+----------------+---------------+---------------------
+---------------------+-------------------------+----------+------------+-------------------------+
3 rows in set (0.00 sec)
```

说明

最后一个分区用户缺省只指定了date列的分区值，所以user_id列的分区值会默认填充MIN_VALUE。当用户插入数据时，分区列值会按照顺序依次比较，最终得到对应的分区。

示例：

```
* 数据  -->  分区
* 2017-01-01, 200     --> p201701_1000
* 2017-01-01, 2000    --> p201701_1000
* 2017-02-01, 100     --> p201701_1000
* 2017-02-01, 2000    --> p201702_2000
* 2017-02-15, 5000    --> p201702_2000
* 2017-03-01, 2000    --> p201703_all
* 2017-03-10, 1       --> p201703_all
* 2017-04-01, 1000    --> 无法导入
* 2017-05-01, 1000    --> 无法导入
```

验证方法：

插入一条数据并检查存入到哪个分区。分区字段VisibleVersionTime、VisibleVersion刚刚有更新的分区即为刚插入数据所在的分区。

```
insert into example_db.expamle_range_multi_partiton_key_tbl values (200, '2017-01-01', '2017-01-01 12:00:05', 'A', 25, 1, '2017-01-01 12:00:05', 100, 30, 10);
insert into example_db.expamle_range_multi_partiton_key_tbl values (2000, '2017-01-01', '2017-01-01 16:10:05', 'B', 33, 1, '2017-01-01 16:10:05', 800, 50, 1);
insert into example_db.expamle_range_multi_partiton_key_tbl values (200, '2017-02-01', '2017-01-01 16:10:05', 'C', 22, 0, '2017-02-01 16:10:05', 80, 200, 1);
show partitions from example_db.expamle_range_multi_partiton_key_tbl\G
```

List 分区

- 分区列支持BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DATE, DATETIME, CHAR, VARCHAR数据类型，分区值为枚举值。只有当数据为目标分区枚举值其中之一时，才可以命中分区。
- Partition支持通过VALUES IN (...) 来指定每个分区包含的枚举值。
- 下面通过示例说明，进行分区的增删操作时，分区的变化。

```
CREATE TABLE IF NOT EXISTS example_db.expamle_list_tbl
(
`user_id` LARGEINT NOT NULL COMMENT "用户id",
`date` DATE NOT NULL COMMENT "数据灌入日期时间",
`timestamp` DATETIME NOT NULL COMMENT "数据灌入的时间戳",
`city` VARCHAR(20) NOT NULL COMMENT "用户所在城市",
`age` SMALLINT COMMENT "用户年龄",
`sex` TINYINT COMMENT "用户性别",
`last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "用户最后一次访问时间",
`cost` BIGINT SUM DEFAULT "0" COMMENT "用户总消费",
`max_dwell_time` INT MAX DEFAULT "0" COMMENT "用户最大停留时间",
`min_dwell_time` INT MIN DEFAULT "99999" COMMENT "用户最小停留时间"
)
ENGINE=olap
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY LIST(`city`)
(
PARTITION `p_cn` VALUES IN ("A", "B", "F"),
PARTITION `p_usa` VALUES IN ("G", "H"),
PARTITION `p_jp` VALUES IN ("I")
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
```


版权所有 © 华为云计算技术有限公司

```
"replication_num" = "3"
);
```

- 如上表所示，建表完成后，会自动生成3个分区。

```
mysql> show partitions from
example_db.expamle_list_tbl;
                                                    +-------------+--------------+---------------+--------------------+--------
+-------------+-----------------------------------------------------------------------------------------------+
+---------------+---------+---------------+--------------+------------------+--------------------
+-----------------------+----------+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                                  | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+------------------+--------------------+------------------------
+----------+------------+------------------------+
| 16764       | p_cn          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [A]; , types: [VARCHAR]; keys: [B]; , types: [VARCHAR]; keys: [F]; ] | user_id         | 16      |
3             | HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false
| tag.location.default: 3 |
| 16765       | p_usa         | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types:
[VARCHAR]; keys: [G]; , types: [VARCHAR]; keys: [H]; ]                           | user_id         | 16      |
3             | HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false
| tag.location.default: 3 |
| 16766       | p_jp          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [I]; ]                                                   | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                    | NULL                     | 0.000    | false      | tag.location.default: 3 |
+-------------+--------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+------------------+--------------------+------------------------
+----------+------------+------------------------+
3 rows in set (0.00 sec)
```

- 增加一个分区。

```
mysql> alter table example_db.expamle_list_tbl add partition p_uk VALUES IN ("L");
Query OK, 0 rows affected (0.01 sec)
```

查询分区。

```
mysql> show partitions from example_db.expamle_list_tbl;
+-------------+--------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+------------------+--------------------+------------------------
+----------+------------+------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                                  | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+------------------+--------------------+------------------------
+----------+------------+------------------------+
| 16764       | p_cn          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [A]; , types: [VARCHAR]; keys: [B]; , types: [VARCHAR]; keys: [F]; ] | user_id         | 16      |
3             | HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false
| tag.location.default: 3 |
| 16765       | p_usa         | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types:
[VARCHAR]; keys: [G]; , types: [VARCHAR]; keys: [H]; ]                           | user_id         | 16      |
3             | HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    | false
| tag.location.default: 3 |
| 16766       | p_jp          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [I]; ]                                                   | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                    | NULL                     | 0.000    | false      | tag.location.default: 3 |
| 16961       | p_uk          | 1              | 2023-04-11 09:24:39 | NORMAL | city         | [types: [VARCHAR];
keys: [L]; ]                                                   | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                    | NULL                     | 0.000    | false      | tag.location.default: 3 |
+-------------+--------------+---------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+------------------+--------------------+------------------------
```


版权所有 © 华为云计算技术有限公司

```
+----------+------------+-------------------------+
4 rows in set (0.00 sec)
```

- 删除分区。

```
mysql> alter table example_db.expamle_list_tbl drop partition p_jp;
Query OK, 0 rows affected (0.01 sec)
```

查看分区。

```
mysql> show partitions from example_db.expamle_list_tbl;
+-------------+--------------+---------------+---------------------+--------+--------------
+-----------------------------------------------------------------------------------------+-----------------
+---------+---------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+-------------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey |
Range                                                                                     | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+---------------------+--------+--------------
+-----------------------------------------------------------------------------------------+-----------------
+---------+---------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+-------------------------+
| 16764       | p_cn          | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types: [VARCHAR];
keys: [A]; , types: [VARCHAR]; keys: [B]; , types: [VARCHAR]; keys: [F]; ] | user_id         | 16      |
3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false
| tag.location.default: 3 |
| 16765       | p_usa         | 1              | 2023-04-11 09:21:34 | NORMAL | city         | [types:
[VARCHAR]; keys: [G]; , types: [VARCHAR]; keys: [H]; ]                              | user_id         | 16      |
3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false
| tag.location.default: 3 |
| 16961       | p_uk          | 1              | 2023-04-11 09:24:39 | NORMAL | city         | [types: [VARCHAR];
keys: [L]; ]                                                        | user_id         | 16      | 3              | HDD           |
9999-12-31 15:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 3 |
+-------------+--------------+---------------+---------------------+--------+--------------
+-----------------------------------------------------------------------------------------+-----------------
+---------+---------------+--------------+--------------------+--------------------+-------------------------
+----------+------------+-------------------------+
3 rows in set (0.00 sec)
```

- 多列分区（List支持多列分区），如下示例。

```
CREATE TABLE IF NOT EXISTS example_db.expamle_list_multi_partiton_key_tbl
(
`user_id` LARGEINT NOT NULL COMMENT "用户id",
`date` DATE NOT NULL COMMENT "数据灌入日期时间",
`timestamp` DATETIME NOT NULL COMMENT "数据灌入的时间戳",
`city` VARCHAR(20) NOT NULL COMMENT "用户所在城市",
`age` SMALLINT COMMENT "用户年龄",
`sex` TINYINT COMMENT "用户性别",
`last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "用户最后一次访问时
间",
`cost` BIGINT SUM DEFAULT "0" COMMENT "用户总消费",
`max_dwell_time` INT MAX DEFAULT "0" COMMENT "用户最大停留时间",
`min_dwell_time` INT MIN DEFAULT "99999" COMMENT "用户最小停留时间"
)
ENGINE=olap
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY LIST(`user_id`, `city`)
(
PARTITION `p1_city` VALUES IN (("1", "A"), ("1", "B")),
PARTITION `p2_city` VALUES IN (("2", "A"), ("2", "B")),
PARTITION `p3_city` VALUES IN (("3", "A"), ("3", "B"))
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
"replication_num" = "3"
);
```

在以上示例中，我们指定user_id（INT类型）和city（VARCHAR类型）作为分区列，最终分区如下。

版权所有 © 华为云计算技术有限公司

```
mysql> show partitions from example_db.expamle_list_multi_partiton_key_tbl;
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+-------------------+--------------------+------------------------
+----------+-----------+-----------------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey  |
Range                                                                             | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       |
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+-------------------+--------------------+------------------------
+----------+-----------+-----------------------+
| 17026       | p1_city       | 1              | 2023-04-11 09:31:33 | NORMAL | user_id, city | [types:
[LARGEINT, VARCHAR]; keys: [1, A]; , types: [LARGEINT, VARCHAR]; keys: [1, B]; ] | user_id         |
16      | 3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    |
false      | tag.location.default: 3 |
| 17027       | p2_city       | 1              | 2023-04-11 09:31:33 | NORMAL | user_id, city | [types:
[LARGEINT, VARCHAR]; keys: [2, A]; , types: [LARGEINT, VARCHAR]; keys: [2, B]; ] | user_id         |
16      | 3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    |
false      | tag.location.default: 3 |
| 17028       | p3_city       | 1              | 2023-04-11 09:31:33 | NORMAL | user_id, city | [types:
[LARGEINT, VARCHAR]; keys: [3, A]; , types: [LARGEINT, VARCHAR]; keys: [3, B]; ] | user_id         |
16      | 3              | HDD           | 9999-12-31 15:59:59 |                     | NULL                     | 0.000    |
false      | tag.location.default: 3 |
+-------------+--------------+---------------+-------------------+--------+--------------
+-----------------------------------------------------------------------------------+----------------
+---------+---------------+--------------+-------------------+--------------------+------------------------
+----------+-----------+-----------------------+
3 rows in set (0.00 sec)
```

- 当用户插入数据时，分区列值会按照顺序依次比较，最终得到对应的分区。如下所示。

```
* 数据  --->  分区
* 1, A     ---> p1_city
* 1, B    ---> p1_city
* 2, B    ---> p2_city
* 3, A    ---> p3_city
* 1, M    ---> 无法导入
* 4, A    ---> 无法导入
```

验证方法：

插入一条数据并检查存入到哪个分区。分区字段VisibleVersionTime、VisibleVersion刚刚有更新的分区即为刚插入数据所在的分区。

```
INSERT INTO example_db.expamle_list_multi_partiton_key_tbl values (1, '2017-01-01', '2017-01-01
12:00:05', 'A', 25, 1, '2017-01-01 12:00:05', 100, 30, 10);
```

查数据插入分区。

```
mysql> SHOW partitions from example_db.expamle_list_multi_partiton_key_tbl\G
*************************** 1. row ***************************
             PartitionId: 17026
           PartitionName: p1_city
          VisibleVersion: 3
      VisibleVersionTime: 2023-04-11 09:42:34
                   State: NORMAL
            PartitionKey: user_id, city
                   Range: [types: [LARGEINT, VARCHAR]; keys: [1, A]; , types: [LARGEINT, VARCHAR]; keys:
[1, B]; ]
         DistributionKey: user_id
                 Buckets: 16
          ReplicationNum: 3
           StorageMedium: HDD
            CooldownTime: 9999-12-31 15:59:59
     RemoteStoragePolicy:
LastConsistencyCheckTime: NULL
                DataSize: 9.340 KB
              IsInMemory: false
       ReplicaAllocation: tag.location.default: 3
*************************** 2. row ***************************
```

```
            PartitionId: 17027
          PartitionName: p2_city
         VisibleVersion: 1
     VisibleVersionTime: 2023-04-11 09:31:33
                  State: NORMAL
           PartitionKey: user_id, city
                  Range: [types: [LARGEINT, VARCHAR]; keys: [2, A]; , types: [LARGEINT, VARCHAR]; keys: 
[2, B]; ]
        DistributionKey: user_id
                Buckets: 16
         ReplicationNum: 3
          StorageMedium: HDD
           CooldownTime: 9999-12-31 15:59:59
    RemoteStoragePolicy: 
LastConsistencyCheckTime: NULL
               DataSize: 0.000
             IsInMemory: false
      ReplicaAllocation: tag.location.default: 3
*************************** 3. row ***************************
            PartitionId: 17028
          PartitionName: p3_city
         VisibleVersion: 1
     VisibleVersionTime: 2023-04-11 09:31:33
                  State: NORMAL
           PartitionKey: user_id, city
                  Range: [types: [LARGEINT, VARCHAR]; keys: [3, A]; , types: [LARGEINT, VARCHAR]; keys: 
[3, B]; ]
        DistributionKey: user_id
                Buckets: 16
         ReplicationNum: 3
          StorageMedium: HDD
           CooldownTime: 9999-12-31 15:59:59
    RemoteStoragePolicy: 
LastConsistencyCheckTime: NULL
               DataSize: 0.000
             IsInMemory: false
      ReplicaAllocation: tag.location.default: 3
3 rows in set (0.00 sec)
```

2.2.2.2 分桶

根据分桶列的Hash值将数据划分成不同的Bucket。

- 如果使用了Partition，则DISTRIBUTED ... 语句描述的是数据在各个分区内的划分规则。如果不使用Partition，则描述的是对整个表的数据的划分规则。
- 分桶列可以是多列，Aggregate和Unique模型必须为Key列，Duplicate模型可以是Key列和Value列。分桶列可以和Partition列相同或不同。
- 分桶列的选择，是在查询吞吐和查询并发之间的一种权衡：
  - 如果选择多个分桶列，则数据分布更均匀。如果一个查询条件不包含所有分桶列的等值条件，那么该查询会触发所有分桶同时扫描，这样查询的吞吐会增加，单个查询的延迟随之降低。这个方式适合大吞吐低并发的查询场景。
  - 如果仅选择一个或少数分桶列，则对应的点查询可以仅触发一个分桶扫描。此时，当多个点查询并发时，这些查询有较大的概率分别触发不同的分桶扫描，各个查询之间的IO影响较小（尤其当不同桶分布在不同磁盘上时），所以这种方式适合高并发的点查询场景。
- AutoBucket: 根据数据量，计算分桶数。 对于分区表，可以根据历史分区的数据量、机器数、盘数，确定一个分桶。
- 分桶的数量理论上没有上限。

版权所有 © 华为云计算技术有限公司

2.2.2.3 最佳实践

关于 Partition 和 Bucket 的数量和数据量的建议

- 一个表的Tablet总数量等于 (Partition num*Bucket num)。
- 一个表的Tablet数量，在不考虑扩容的情况下，推荐略多于整个集群的磁盘数量。
- 单个Tablet的数据量理论上没有上下界，但建议在1G-10G的范围内。如果单个Tablet数据量过小，则数据的聚合效果不佳，且元数据管理压力大。如果数据量过大，则不利于副本的迁移、补齐，且会增加Schema Change或者Rollup操作失败重试的代价（这些操作失败重试的粒度是Tablet）。
- 当Tablet的数据量原则和数量原则冲突时，建议优先考虑数据量原则。
- 在建表时，每个分区的Bucket数量统一指定。但是在动态增加分区时（ADD PARTITION），可以单独指定新分区的Bucket数量。可以利用这个功能方便的应对数据缩小或膨胀。
- 一个Partition的Bucket数量一旦指定，不可更改。所以在确定Bucket数量时，需要预先考虑集群扩容的情况。比如当前只有3台host，每台host有1块盘。如果Bucket的数量只设置为3或更小，那么后期即使再增加机器，也不能提高并发度。
- 举一些例子：假设在有10台BE，每台BE一块磁盘的情况下。如果一个表总大小为500MB，则可以考虑4-8个分片。5GB：8-16个分片。50GB：32个分片。500GB：建议分区，每个分区大小在50GB左右，每个分区16-32个分片。5TB：建议分区，每个分区大小在50GB 左右，每个分区16-32个分片。

关于 Random Distribution 的设置以及使用场景

- 如果OLAP表没有更新类型的字段，将表的数据分桶模式设置为RANDOM，则可以避免严重的数据倾斜（数据在导入表对应的分区的时候，单次导入作业每个batch的数据将随机选择一个tablet进行写入）。
- 当表的分桶模式被设置为RANDOM时，因为没有分桶列，无法根据分桶列的值仅对几个分桶查询，对表进行查询的时候将对命中分区的全部分桶同时扫描，该设置适合对表数据整体的聚合查询分析而不适合高并发的点查询。
- 如果OLAP表的是Random Distribution的数据分布，那么在数据导入的时候可以设置单分片导入模式（将load_to_single_tablet设置为true），那么在大数据量的导入的时候，一个任务在将数据写入对应的分区时将只写入一个分片，这样将能提高数据导入的并发度和吞吐量，减少数据导入和Compaction导致的写放大问题，保障集群的稳定性。

复合分区与单分区

- 复合分区。
  - 第一级称为Partition，即分区。用户可以指定某一维度列作为分区列（当前只支持整型和时间类型的列），并指定每个分区的取值范围。
  - 第二级称为Distribution，即分桶。用户可以指定一个或多个维度列以及桶数对数据进行HASH分布或者不指定分桶列设置成Random Distribution对数据进行随机分布。

说明

此场景推荐使用复合分区。

- 有时间维度或类似带有有序值的维度，可以以这类维度列作为分区列。分区粒度可以根据导入频次、分区数据量等进行评估。
- 历史数据删除需求：如有删除历史数据的需求（比如仅保留最近N天的数据）。使用复合分区，可以通过删除历史分区来达到目的。也可以通过在指定分区内发送DELET语句进行数据删除。
- 解决数据倾斜问题：每个分区可以单独指定分桶数量。如按天分区，当每天的数据量差异很大时，可以通过指定分区的分桶数，合理划分不同分区的数据,分桶列建议选择区分度大的列。

- 单分区。

  用户也可以不使用复合分区，即使用单分区。则数据只做Hash分布。

2.2.2.4 常见问题

Failed to create partition [xxx] . Timeout

Doris建表是按照Partition粒度依次创建的。当一个Partition创建失败时，可能会报这个错误。即使不使用Partition，当建表出现问题时，也会报Failed to create partition，因为如前文所述，Doris会为没有指定Partition的表创建一个不可更改的默认的Partition。

当遇到这个错误时，通常是BE在创建数据分片时遇到了问题。可以参照以下步骤排查：

- 在fe.log中，查找对应时间点的Failed to create partition日志。在该日志中，会出现一系列类似{10001-10010}字样的数字对。数字对的第一个数字表示Backend ID，第二个数字表示Tablet ID。如上这个数字对，表示ID为10001的Backend上，创建ID为10010的Tablet失败了。
- 前往对应Backend的be.INFO日志，查找对应时间段内，tablet id相关的日志，可以找到错误信息。
- 以下罗列一些常见的tablet创建失败错误，包括但不限于：
  - BE没有收到相关task，此时无法在be.INFO中找到tablet id相关日志或者BE创建成功，但汇报失败。
  - 预分配内存失败。可能是表中一行的字节长度超过了 100KB。
  - Too many open files。打开的文件句柄数超过了Linux系统限制。需修改Linux系统的句柄数限制。

  如果创建数据分片时超时，也可以通过在fe.conf中设置tablet_create_timeout_second=xxx以及max_create_table_timeout_second=xxx来延长超时时间。其中tablet_create_timeout_second默认是1秒, max_create_table_timeout_second默认是60秒，总体的超时时间为min(tablet_create_timeout_second * replication_num, max_create_table_timeout_second)。

建表命令长时间不返回结果

Doris的建表命令是同步命令。该命令的超时时间目前设置的比较简单，即（tablet num*replication num）秒。如果创建较多的数据分片，并且其中有分片创建失败，则可能导致等待较长超时后，才会返回错误。

版权所有 © 华为云计算技术有限公司

正常情况下，建表语句会在几秒或十几秒内返回。如果超过一分钟，建议直接取消掉这个操作，前往FE或BE的日志查看相关错误。

2.2.3 数据分布

数据分片

Doris表按两层结构进行数据划分，分别是分区和分桶。

每个分桶文件就是一个数据分片（Tablet），Tablet是数据划分的最小逻辑单元。每个Tablet包含若干数据行。各个Tablet之间的数据没有交集，并且在物理上是独立存储的。

一个Tablet只属于一个Partition，相应的多个Tablet在逻辑上归属于不同的分区（Partition）。而一个Partition包含若干个Tablet。因为Tablet在物理上是独立存储的，所以可以视为Partition在物理上也是独立。Tablet是数据移动、复制等操作的最小物理存储单元。

2.3 数据导入

2.3.1 总览

导入（Load）功能就是将用户的原始数据导入到Doris中。导入成功后，用户即可通过Mysql客户端查询数据。Doris支持多种导入方式。

支持数据源

Doris提供多种数据导入方案，可以针对不同的数据源选择不同的数据导入方式。

- **Broker Load**
- **Stream Load**

支持的数据格式

不同的导入方式支持的数据格式不同。

表 2-22 导入方式说明

导入方式	支持格式
Broker Load	parquet、orc、obs
Stream Load	csv、json、parquet、orc

导入说明

Doris的数据导入实现有以下共性特征，这里分别介绍，以帮助大家更好的使用数据导入功能。

导入的原子性保证

Doris的每一个导入任务，不论是使用Broker Load进行批量导入，还是使用INSERT语句进行单条导入，都是一个完整的事务操作。导入事务可以保证一批次内的数据原子生效，不会出现部分数据写入的情况。

同时，每个导入任务都会有一个Label。这个Label在数据库（Database）中是唯一的，用于唯一标识一个导入任务。Label可以由用户指定，部分导入功能也会由系统自动生成。

Label是用于保证对应的导入任务，仅能成功导入一次。一个被成功导入的Label，再次使用时，会被拒绝并报错Label already used。通过这个机制，可以在Doris侧做到At-Most-Once语义。如果结合上游系统的At-Least-Once语义，则可以实现导入数据的Exactly-Once语义。

同步和异步

Doris目前的导入方式分为两类，同步和异步。如果是外部程序接入Doris的导入功能，需要判断使用导入方式是哪类再确定接入逻辑。

- 同步

  同步导入方式即用户创建导入任务，Doris同步执行导入，执行完成后返回用户导入结果。用户可直接根据创建导入任务命令返回的结果同步判断导入是否成功。

- 异步

  异步导入方式即用户创建导入任务后，Doris直接返回创建成功。创建成功不代表数据已经导入。导入任务会被异步执行，用户在创建成功后，需要通过轮询的方式发送命令查看导入作业的状态。如果创建失败，则可以根据失败信息，判断是否需要再次创建。

  说明

  无论是异步还是同步的导入类型，都不应该在Doris返回导入失败或导入创建失败后，无休止的重试。外部系统在有限次数重试并失败后，保留失败信息，大部分多次重试均失败问题都是使用方法问题或数据本身问题。

2.3.2 批量数据导入

2.3.2.1 Broker Load

Broker Load是一个异步的导入方式，支持的数据源取决于Broker进程支持的数据源。本文为您介绍Broker Load导入的基本原理、基本操作、系统配置以及最佳实践。

适用场景

- 源数据在Broker可以访问的存储系统中，如HDFS、OBS。
- 数据量在几十到百GB级别。

基本原理

用户在提交导入任务后，FE会生成对应的Plan并根据目前BE的个数和文件的大小，将Plan分给多个BE执行，每个BE执行一部分导入数据。

BE在执行的过程中会从Broker拉取数据，在对数据transform之后将数据导入系统。所有BE均完成导入，由FE最终决定导入是否成功。

```
+
              | 1. user create broker load
              v
         +----+----+
         |         |
         |   FE    |
         |         |
         +----+----+
              |
              | 2. BE etl and load the data
   +--------------------------+
   |          |               |
+---v---+     +--v----+    +---v---+
|       |     |       |    |       |
|  BE   |     |  BE   |    |  BE   |
|       |     |       |    |       |
+---+-^-+     +---+-^-+    +--+-^--+
   | |          | |         | |
   | |          | |         | | 3. pull data from broker
+---v-+-+     +---v-+-+    +--v-+--+
|       |     |       |    |       |
|Broker |     |Broker |    |Broker |
|       |     |       |    |       |
+---+-^-+     +---+-^-+    +---+-^-+
   | |          | |          | |
+---v-+-----------v-+----------v-+-+
|       HDFS/BOS/AFS cluster       |
|                                  |
+----------------------------------+
```

开始导入

下面我们通过几个实际的场景示例来看Broker Load的使用。

数据样例：

'100','101','102','103','104','105',100.00,100.01,100.02,'100',200,100.08,2022-04-01

'101','102','103','104','105','105',100.00,100.01,100.02,'100',200,100.08,2022-04-02

'102','103','104','105','106','105',100.00,100.01,100.02,'100',200,100.08,2022-04-03

准备工作：

在本地创建示例数据文件source_text.txt，并上传至hdfs的/tmp/。

在hive中创建ods_source表。

```
CREATE TABLE `ods_source`(
  `id` string,
  `store_id` string,
  `company_id` string,
  `tower_id` string,
  `commodity_id` string,
  `commodity_name` string,
  `commodity_price` double,
  `member_price` double,
  `cost_price` double,
  `unit` string,
  `quantity` string,
  `actual_price` double,
  `day ` string
)
row format delimited fields terminated by ','
lines terminated by '\n'
stored as textfile;
```

将hdfs创建的txt文件导入到ods_source表。

版权所有 © 华为云计算技术有限公司

```
load data inpath '/tmp/source_text.txt' into table ods_source;
```

- 示例1，Parquet格式表导入。
  - 在hive中创建parquet分区表并写入数据。
    - 创建ods_demo_detail表。

```
CREATE TABLE `ods_demo_detail`(
  `id` string,
  `store_id` string,
  `company_id` string,
  `tower_id` string,
  `commodity_id` string,
  `commodity_name` string,
  `commodity_price` double,
  `member_price` double,
  `cost_price` double,
  `unit` string,
  `quantity` string,
  `actual_price` double
)
PARTITIONED BY (day string)
row format delimited fields terminated by ','
lines terminated by '\n'
stored as textfile;
```

    - 把ods_source表中的数据导入到ods_demo_detail表中。

```
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.dynamic.partition=true;
insert overwrite table ods_demo_detail partition(day) select * from ods_source;
```

  - 查看Hive表ods_demo_detail有没有数据。

```
select * from ods_demo_detail;
```

  - 在Doris创建数据库。

```
create database doris_demo_db;
```

  - 创建Doris表doris_ods_test_detail。

说明

如果创建集群的时候不是超高io，则去掉'storage_medium' = 'SSD'。

```
use doris_demo_db;
CREATE TABLE `doris_ods_test_detail` (
  `rq` date NULL,
  `id` varchar(32) NOT NULL,
  `store_id` varchar(32) NULL,
  `company_id` varchar(32) NULL,
  `tower_id` varchar(32) NULL,
  `commodity_id` varchar(32) NULL,
  `commodity_name` varchar(500) NULL,
  `commodity_price` decimal(10, 2) NULL,
  `member_price` decimal(10, 2) NULL,
  `cost_price` decimal(10, 2) NULL,
  `unit` varchar(50) NULL,
  `quantity` int(11) NULL,
  `actual_price` decimal(10, 2) NULL
) ENGINE=OLAP
UNIQUE KEY(`rq`, `id`, `store_id`)
PARTITION BY RANGE(`rq`)
(
PARTITION P_202204 VALUES [('2022-04-01'),('2022-08-30')))
DISTRIBUTED BY HASH(`store_id`) BUCKETS 1
PROPERTIES (
'replication_allocation' = 'tag.location.default: 3',
'dynamic_partition.enable' = 'true',
'dynamic_partition.time_unit' = 'MONTH',
'dynamic_partition.start' = '-2147483648',
'dynamic_partition.end' = '2',
```


版权所有 © 华为云计算技术有限公司

```
'dynamic_partition.prefix' = 'P_',
'dynamic_partition.buckets' = '1',
'in_memory' = 'false',
'storage_format' = 'V2',
'storage_medium' = 'SSD'
);
```

– 导入数据。

```
LOAD LABEL broker_name_test01
(
    DATA INFILE('hdfs://{hdfs远端ip}:{hdfs远端端口号}/user/hive/warehouse/ods_demo_detail/*/*')
    INTO TABLE doris_ods_test_detail
    COLUMNS TERMINATED BY ','
(id,store_id,company_id,tower_id,commodity_id,commodity_name,commodity_price,member_price,cost_price,unit,quantity,actual_price)
    COLUMNS FROM PATH AS (`day`)
   SET
   (rq = str_to_date(`day`,'%Y-%m-%d'),id=id,store_id=store_id,company_id=company_id,tower_id=tower_id,commodity_id=commodity_id,commodity_name=commodity_name,commodity_price=commodity_price,member_price=member_price,cost_price=cost_price,unit=unit,quantity=quantity,actual_price=actual_price)
    )
WITH BROKER 'broker1'
    (
      'username' = 'hdfs',
      'password' = ''
    )
PROPERTIES
(
    'timeout'='1200',
    'max_filter_ratio'='0.1'
);
```

– 查看导入状态

您可以通过下面的命令查看上面导入任务的状态信息。

```
show load order by createtime desc limit 1\G;
```

> **说明**
>
> 如果状态信息出现"Scan bytes per file scanner exceed limit: 3221225472"，说明导入失败，需要修改参数"max_bytes_per_broker_scanner"，请参见**Doris参数配置**章节的FE节点参数说明表。

图 2-1 查看数据导入状态

```
mysql>  show load order by createtime desc limit 2\G;
*************************** 1. row ***************************
         JobId: 11015
         Label: label_2023070614
         State: CANCELLED
      Progress: ETL:N/A; LOAD:N/A
          Type: BROKER
       EtlInfo: NULL
      TaskInfo: cluster:N/A; timeout(s):120000; max_filter_ratio:0.1
      ErrorMsg: type:ETL_RUN_FAIL; msg:errCode = 2, detailMessage = Scan bytes per file scanner exceed limit: 3221225472
    CreateTime: 2023-07-06 11:34:04
  EtlStartTime: NULL
 EtlFinishTime: NULL
 LoadStartTime: NULL
LoadFinishTime: 2023-07-06 11:34:08
           URL: NULL
    JobDetails: {"Unfinished backends":{},"ScannedRows":0,"TaskNumber":0,"LoadBytes":0,"All backends":{},"FileNumber":1,"FileSize":98638126365}
 TransactionId: 1007
  ErrorTablets: {}
```

- 示例2，ORC格式表导入。

– 在Hive中创建Hive分区表，ORC格式。

```
CREATE TABLE `ods_demo_orc_detail`(
 `id` string,
 `store_id` string,
 `company_id` string,
 `tower_id` string,
 `commodity_id` string,
 `commodity_name` string,
 `commodity_price` double,
 `member_price` double,
 `cost_price` double,
 `unit` string,
```


版权所有 © 华为云计算技术有限公司

```
  `quantity` double,
  `actual_price` double
)
PARTITIONED BY (day string)
row format delimited fields terminated by ','
lines terminated by '\n'
STORED AS ORC;
```

– 查询Source表写入分区表。

```
insert overwrite table ods_demo_orc_detail partition(day) select * from ods_source;
```

– 创建Doris表。

```
CREATE TABLE `doris_ods_orc_detail` (
  `rq` date NULL,
  `id` varchar(32) NOT NULL,
  `store_id` varchar(32) NULL,
  `company_id` varchar(32) NULL,
  `tower_id` varchar(32) NULL,
  `commodity_id` varchar(32) NULL,
  `commodity_name` varchar(500) NULL,
  `commodity_price` decimal(10, 2) NULL,
  `member_price` decimal(10, 2) NULL,
  `cost_price` decimal(10, 2) NULL,
  `unit` varchar(50) NULL,
  `quantity` int(11) NULL,
  `actual_price` decimal(10, 2) NULL
) ENGINE=OLAP
UNIQUE KEY(`rq`, `id`, `store_id`)
PARTITION BY RANGE(`rq`)
(
PARTITION P_202204 VALUES [('2022-04-01'), ('2022-08-30')))
DISTRIBUTED BY HASH(`store_id`) BUCKETS 1
PROPERTIES (
'replication_allocation' = 'tag.location.default: 3',
'dynamic_partition.enable' = 'true',
'dynamic_partition.time_unit' = 'MONTH',
'dynamic_partition.start' = "-2147483648",
'dynamic_partition.end' = '2',
'dynamic_partition.prefix' = 'P_',
'dynamic_partition.buckets' = '1',
'in_memory' = 'false',
'storage_format' = 'V2');
```

– 导入数据。

```
LOAD LABEL orc_2022_02_17
(
    DATA INFILE("hdfs://{hdfs远端ip}:{hdfs远端端口号}/user/hive/warehouse/
ods_demo_orc_detail/*/*")
    INTO TABLE doris_ods_orc_detail
    COLUMNS TERMINATED BY ","
    FORMAT AS 'orc'
(id,store_id,company_id,tower_id,commodity_id,commodity_name,commodity_price,member_pric
e,cost_price,unit,quantity,actual_price)
    COLUMNS FROM PATH AS (`day`)
   SET
   (rq = str_to_date(`day`,'%Y-%m-
%d'),id=id,store_id=store_id,company_id=company_id,tower_id=tower_id,commodity_id=commodi
ty_id,commodity_name=commodity_name,commodity_price=commodity_price,member_price=m
ember_price,cost_price=cost_price,unit=unit,quantity=quantity,actual_price=actual_price)
    )
WITH BROKER 'broker1'
    (
      'username' = 'hdfs',
      'password'= ""
    )
PROPERTIES
(
    'timeout'="1200",
    'max_filter_ratio'="0.1"
);
```

 版权所有 © 华为云计算技术有限公司

- 查询导入数据。

```
show load order by createtime desc limit 1\G;
```

- 示例3，OBS格式数据导入。
  - 创建Doris表。

```
CREATE TABLE `obs_detail_test` (
  `id` varchar(32) NOT NULL,
  `store_id` varchar(32) NULL,
  `company_id` varchar(32) NULL,
  `tower_id` varchar(32) NULL,
  `commodity_id` varchar(32) NULL,
  `commodity_name` varchar(500) NULL,
  `commodity_price` decimal(10, 2) NULL,
  `member_price` decimal(10, 2) NULL,
  `cost_price` decimal(10, 2) NULL,
  `unit` varchar(50) NULL,
  `quantity` int(11) NULL,
  `actual_price` decimal(10, 2) NULL
) ENGINE=OLAP
UNIQUE KEY(`id`, `store_id`)
DISTRIBUTED BY HASH(`store_id`) BUCKETS 1
PROPERTIES (
'replication_allocation' = 'tag.location.default: 3',
'in_memory' = 'false',
'storage_format' = 'V2'
);
```

  - 将OBS数据导入到Doris表。

    构造text数据100条，该数据与Doris表字段对应，将数据上传到OBS桶。

说明

fs.obs.access.key, fs.obs.secret.key,fs.obs.endpoint获取方式如下所示：

- fs.obs.access.key与fs.obs.secret.key的获取方式请参见对象存储服务的**获取访问密钥（AK/SK）**章节。
- fs.obs.endpoint的获取方式见请参见对象存储服务的**获取终端节点**章节。

```
LOAD LABEL label_2023021801
(
DATA INFILE("obs://xxx/source_text2.txt")
INTO TABLE `obs_detail_test`
COLUMNS TERMINATED BY ","
(id,store_id,company_id,tower_id,commodity_id,commodity_name,commodity_price,member_pric
e,cost_price,unit,quantity,actual_price)
)
WITH BROKER 'broker1' (
'fs.obs.access.key' = 'xxx',
'fs.obs.secret.key' = 'xxxxxx',
'fs.obs.endpoint' = 'xxxxxx'
)
PROPERTIES
(
'timeout'="1200",
'max_filter_ratio'='0.1'
);
```

  - 查询数据。

```
show load order by createtime desc limit 1\G;
```

- 示例4，使用With HDFS方式将HDFS的数据导入Doris表。
  - 创建Doris表。

```
CREATE TABLE `ods_dish_detail_test` (
  `id` varchar(32) NOT NULL,
  `store_id` varchar(32) NULL,
  `company_id` varchar(32) NULL,
  `tower_id` varchar(32) NULL,
  `commodity_id` varchar(32) NULL,
```


版权所有 © 华为云计算技术有限公司

```
  `commodity_name` varchar(500) NULL,
  `commodity_price` decimal(10, 2) NULL,
  `member_price` decimal(10, 2) NULL,
  `cost_price` decimal(10, 2) NULL,
  `unit` varchar(50) NULL,
  `quantity` int(11) NULL,
  `actual_price` decimal(10, 2) NULL
) ENGINE=OLAP
UNIQUE KEY(`id`, `store_id`)
DISTRIBUTED BY HASH(`store_id`) BUCKETS 1
PROPERTIES (
'replication_allocation' = 'tag.location.default: 3',
'in_memory' = 'false',
'storage_format' = 'V2'
);
```

– 导入数据。

构造text数据100条，该数据与Doris表字段对应。

```
LOAD LABEL label_2023021703
        (
            DATA INFILE("hdfs://{hdfs远端ip}:{hdfs远端端口号}/tmp/source_text2.txt")
            INTO TABLE `ods_dish_detail_test`
            COLUMNS TERMINATED BY ","
(id,store_id,company_id,tower_id,commodity_id,commodity_name,commodity_price,member_pric
e,cost_price,unit,quantity,actual_price)
        )
        with HDFS (
            'fs.defaultFS'="hdfs://{hdfs远端ip}:{hdfs远端端口号}",
            'hadoop.username'="hdfs",
            'password'=""
        )
        PROPERTIES
        (
            'timeout'="1200",
            'max_filter_ratio'='0.1'
        );
```

– 查询数据。

```
show load order by createtime desc limit 1\G;
```

取消导入

当Broker load作业状态不为CANCELLED或FINISHED时，可以被用户手动取消。取消时需要指定待取消导入任务的Label 。

相关系统配置

- FE配置。

  下面几个配置属于Broker load的系统级别配置，也就是作用于所有Broker load导入任务的配置。主要通过修改**FE配置项**来调整配置值。

  – max_bytes_per_broker_scanner/max_broker_concurrency

    max_bytes_per_broker_scanner配置限制了单个BE处理的数据量的最大值。max_broker_concurrency配置限制了一个作业的最大的导入并发数。最小处理的数据量（默认64M），最大并发数，源文件的大小和当前集群BE的个数共同决定了本次导入的并发数。

```
本次导入并发数=Math.min(源文件大小/最小处理量（默认64M），最大并发数，当前BE节点个数)。
本次导入单个BE的处理量=源文件大小/本次导入的并发数。
```

    通常一个导入作业支持的最大数据量为max_bytes_per_broker_scanner*BE节点数。如果需要导入更大数据量，则需要适当调整max_bytes_per_broker_scanner参数的大小。

版权所有 © 华为云计算技术有限公司

默认配置：

- 参数名：max_broker_concurrency， 默认10。
- 参数名：max_bytes_per_broker_scanner，默认3G，单位bytes。

最佳实践

- 应用场景。

  使用Broker load最适合的场景就是原始数据在文件系统（HDFS，BOS，AFS）中的场景。其次，由于Broker load是单次导入中唯一的一种异步导入的方式，所以如果用户在导入大文件中，需要使用异步接入，也可以考虑使用Broker load。
- 数据量。

  这里仅讨论单个BE的情况，如果用户集群有多个BE则下面标题中的数据量应该乘以BE个数来计算。比如：如果用户有3个BE，则3G以下（包含）则应该乘以3，也就是9G以下（包含）。
  - 3G以下（包含）：用户可以直接提交Broker load创建导入请求。
  - 3G以上：由于单个导入BE最大的处理量为3G，超过3G的待导入文件就需要通过调整Broker load的导入参数来实现大文件的导入。
    - 根据当前BE的个数和原始文件的大小修改单个BE的最大扫描量和最大并发数。

      ```
      修改fe配置项。
      max_broker_concurrency=BE个数。
      当前导入任务单个BE处理的数据量=原始文件大小/max_broker_concurrency。
      max_bytes_per_broker_scanner >=当前导入任务单个BE处理的数据量。
      比如一个100G的文件，集群的BE个数为10个。
      max_broker_concurrency=10。
      max_bytes_per_broker_scanner >=10G=100G/10。
      ```

      修改后，所有的BE会并发的处理导入任务，每个BE处理原始文件的一部分。

      说明

      上述两个FE中的配置均为系统配置，也就是说其修改是作用于所有的Broker load的任务的。
    - 在创建导入的时候自定义当前导入任务的timeout时间。

      这时候不推荐用户将导入最大超时时间直接改大来解决问题。单个导入时间如果超过默认的导入最大超时时间4小时，最好是通过切分待导入文件并且分多次导入来解决问题。主要原因是：单次导入超过4小时的话，导入失败后重试的时间成本很高。

      可以通过如下公式计算出Doris集群期望最大导入文件数据量：

      ```
      期望最大导入文件数据量=14400s*10M/s*BE个数。
      比如：集群的BE个数为10个。
      期望最大导入文件数据量=14400s*10M/s*10 =1440000M≈1440G。
      ```

      说明

      一般用户的环境可能达不到10M/s的速度，所以建议超过500G的文件都进行文件切分，再导入。

作业调度

系统会限制一个集群内正在运行的Broker Load作业数量，以防止同时运行过多的Load作业。

版权所有 © 华为云计算技术有限公司

首先，FE的配置参数：desired_max_waiting_jobs会限制一个集群内未开始或正在运行（作业状态为PENDING或LOADING）的Broker Load作业数量。默认为100。如果超过这个阈值，新提交的作业将会被直接拒绝。

一个Broker Load作业会被分为pending task和loading task阶段。其中pending task负责获取导入文件的信息，而loading task会发送给BE执行具体的导入任务。

FE的配置参数async_pending_load_task_pool_size用于限制同时运行的pending task的任务数量。也相当于控制了实际正在运行的导入任务数量。该参数默认为10。也就是说，假设用户提交了100个Load作业，同时只会有10个作业会进入LOADING状态开始执行，而其他作业处于PENDING等待状态。

FE的配置参数async_loading_load_task_pool_size用于限制同时运行的loading task的任务数量。一个Broker Load作业会有1 pending task和多个loading task（等于LOAD语句中DATA INFILE子句的个数）。所以async_loading_load_task_pool_size应该大于等于async_pending_load_task_pool_size。

性能分析

可以在提交LOAD作业前，先执行set enable_profile=true打开会话变量。然后提交导入作业。待导入作业完成后，可以在FE的web页面的Queris标签中查看到导入作业的Profile。

2.3.2.2 Stream Load

Stream load是一个同步的导入方式，用户通过发送HTTP协议发送请求将本地文件或数据流导入到Doris中。Stream load同步执行导入并返回导入结果。用户可直接通过请求的返回体判断本次导入是否成功。

Stream load主要适用于导入本地文件，或通过程序导入数据流中的数据。

基本原理

下图展示了Stream load的主要流程，省略了一些导入细节。

```
 ^      +
                         |      |
                         |      | 1A. User submit load to FE
                         |      |
                         |   +--v-----------+
                         |   | FE           |
5. Return result to user |   +--+-----------+
                         |      |
                         |      | 2. Redirect to BE
                         |      |
                         |   +--v-----------+
                         +---+Coordinator BE| 1B. User submit load to BE
                             +-+-----+----+-+
                               |     |    |
                         +-----+     |    +-----+
                         |           |          | 3. Distrbute data
                         |           |          |
                       +-v-+       +-v-+      +-v-+
                       |BE |       |BE |      |BE |
                       +---+       +---+      +---+
```

Stream load中，Doris会选定一个节点作为Coordinator节点。该节点负责接数据并分发数据到其他数据节点。您可以通过HTTP协议提交导入命令。如果提交到FE，则FE会通过HTTP redirect指令将请求转发给某一个BE。用户也可以直接提交导入命令给某一指定BE。导入的最终结果由Coordinator BE返回给用户。

 版权所有 © 华为云计算技术有限公司

基本操作

说明

在执行数据导入Stream Load操作之前，必须确保Doris集群的安全组端口开放，即8030和8040端口，否则Stream Load操作将会连接超时。

- 创建导入

  Stream Load通过HTTP协议提交和传输数据。这里通过curl命令展示如何提交导入。

  用户也可以通过其他HTTP client进行操作。

```
curl --location-trusted -u user:passwd [-H ""...] -T data.file -XPUT http://fe_host:http_port/api/{db}/{table}/_stream_load
```

说明

- Header中支持属性见下面的‘导入任务参数’说明。
- 格式为：-H "key1:value1"。
- port：HTTP的端口。

创建导入任务的详细语法可以通过HELP STREAM LOAD命令查看。Stream Load中所有与导入任务相关的参数均设置在Header中。相关参数描述如下表所示。

表 2-23 参数说明

参数		说明
签名参数	user/passwd	Stream load由于创建导入的协议使用的是HTTP协议，通过Basic access authentication进行签名。Doris系统会根据签名验证用户身份和导入权限。
导入任务参数	label	导入任务的标识。每个导入任务，都有一个在单database内部唯一的label。label是用户在导入命令中自定义的名称。通过这个label，用户可以查看对应导入任务的执行情况。 label的另一个作用，是防止用户重复导入相同的数据。强烈推荐用户同一批次数据使用相同的 label。这样同一批次数据的重复请求只会被接受一次，保证了At-Most-Once。当label对应的导入作业状态为CANCELLED时，该label可以再次被使用。
	column_separator	用于指定导入文件中的列分隔符，默认为\t。如果是不可见字符，则需要加\x作为前缀，使用十六进制来表示分隔符。 如hive文件的分隔符\x01，需要指定为-H "column_separator:\x01"。 可以使用多个字符的组合作为列分隔符。
	line_delimiter	用于指定导入文件中的换行符，默认为\n。 可以使用做多个字符的组合作为换行符。

版权所有 © 华为云计算技术有限公司

参数		说明
	max_filter_ratio	导入任务的最大容忍率，默认为0容忍，取值范围是0~1。当导入的错误率超过该值，则导入失败。 如果用户希望忽略错误的行，可以通过设置这个参数大于 0，来保证导入可以成功。 计算公式为： (dpp.abnorm.ALL / (dpp.abnorm.ALL + dpp.norm.ALL)) > max_filter_ratio dpp.abnorm.ALL 表示数据质量不合格的行数。如类型不匹配，列数不匹配，长度不匹配等等。 dpp.norm.ALL 指的是导入过程中正确数据的条数。可以通过 SHOW LOAD 命令查询导入任务的正确数据量。 原始文件的行数 = dpp.abnorm.ALL + dpp.norm.ALL。
	where	导入任务指定的过滤条件。Stream load支持对原始数据指定where语句进行过滤。被过滤的数据将不会被导入，也不会参与 filter ratio的计算，但会被计入num_rows_unselected。
	Partitions	待导入表的Partition信息，如果待导入数据不属于指定的Partition则不会被导入。这些数据将计入dpp.abnorm.ALL。
	columns	待导入数据的函数变换配置，目前Stream load支持的函数变换方法包含列的顺序变化以及表达式变换，其中表达式变换的方法与查询语句的一致。
	exec_mem_limit	导入内存限制。默认为2GB，单位为字节。
	strict_mode	Stream Load导入可以开启strict mode模式。开启方式为在HEADER中声明strict_mode=true 。默认的strict mode为关闭。 strict mode模式的意思是：对于导入过程中的列类型转换进行严格过滤。严格过滤的策略如下： 对于列类型转换来说，如果strict mode为true，则错误的数据将被filter。这里的错误数据是指：原始数据并不为空值，在参与列类型转换后结果为空值的这一类数据。对于导入的某列由函数变换生成时，strict mode对其不产生影响。对于导入的某列类型包含范围限制的，如果原始数据能正常通过类型转换，但无法通过范围限制的，strict mode对其也不产生影响。例如：如果类型是 decimal(1,0), 原始数据为10，则属于可以通过类型转换但不在列声明的范围内。这种数据strict对其不产生影响。

<table>
<tr><th colspan="2">参数</th><th>说明</th></tr>
<tr><td></td><td>merge_type</td><td>数据的合并类型，一共支持三种类型APPEND、DELETE、MERGE其中，APPEND是默认值，表示这批数据全部需要追加到现有数据中，Delete表示删除与这批数据Key相同的所有行，MERGE语义需要与Delete条件联合使用，表示满足Delete条件的数据按照Delete语义处理其余的按照APPEND语义处理。</td></tr>
<tr><td></td><td>two_phase_commit</td><td>Stream load导入可以开启两阶段事务提交模式：在Stream load过程中，数据写入完成即会返回信息给用户，此时数据不可见，事务状态为PRECOMMITTED，用户手动触发commit操作之后，数据才可见。

示例：

● 发起Stream load预提交操作。
<pre>curl --location-trusted -u user:passwd -H "two_phase_commit:true" -T test.txt http://fe_host:http_port/api/{db}/{table}/_stream_load
{
    "TxnId": 18036,
    "Label": "55c8ffc9-1c40-4d51-b75e-f2265b3602ef",
    "TwoPhaseCommit": "true",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 100,
    "NumberLoadedRows": 100,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 1031,
    "LoadTimeMs": 77,
    "BeginTxnTimeMs": 1,
    "StreamLoadPutTimeMs": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 58,
    "CommitAndPublishTimeMs": 0
}</pre>
● 对事务触发commit操作 注意1) 请求发往fe或be均可注意2) commit的时候可以省略url中的{table}。
<pre>curl -X PUT --location-trusted -u user:passwd -H "txn_id:18036" -H "txn_operation:commit" http://fe_host:http_port/api/{db}/{table}/_stream_load_2pc
{
    "status": "Success",
    "msg": "transaction [18036] commit successfully."
}</pre>
● 对事务触发abort操作注意1) 请求发往FE或BE均可注意2) abort的时候可以省略URL中的 {table}。
<pre>curl -X PUT --location-trusted -u user:passwd -H "txn_id:18037" -H "txn_operation:abort" http://fe_host:http_port/api/{db}/{table}/_stream_load_2pc
{
    "status": "Success",
    "msg": "transaction [18037] abort successfully."
}</pre></td></tr>
</table>

- 示例1，CSV数据格式导入。
  - 创建Doris表

```
CREATE TABLE cloudtable0327.doris_streameload_test01
(
```


版权所有 © 华为云计算技术有限公司

```
user_id bigint,
date date,
group_id bigint,
modify_date date,
keyword VARCHAR(128)
)
UNIQUE KEY(user_id, date, group_id)
DISTRIBUTED BY HASH (user_id) BUCKETS 32
PROPERTIES(
'function_column.sequence_col' = 'modify_date',
'replication_num' = '3',
'in_memory' = 'false'
);
```

– 准备数据表sequencedata01.csv。

表 2-24 sequencedata01.csv

1	2020-02-22	1	2020-02-21	a
1	2020-02-22	1	2020-02-22	b
1	2020-02-22	1	2020-03-05	c
1	2020-02-22	1	2020-02-26	d
1	2020-02-22	1	2020-02-23	e
1	2020-02-22	1	2020-02-24	b

– 执行curl命令load数据。

```
curl -k --location-trusted -u admin:passwd -T sequencedata01.csv -H 'column_separator:,'
https://{fe_host}:{http_port}/api/cloudtable0327/doris_streameload_test01/_stream_load
```

– 返回结果。

由于Stream load是一种同步的导入方式，所以导入的结果会通过创建导入的返回值直接返回给用户。

```
{
    "TxnId": 1003,
    "Label": "b6f3bc78-0d2c-45d9-9e4c-faa0a0149bee",
    "Status": "Success",
    "ExistingJobStatus": "FINISHED", // optional
    "Message": "OK",
    "NumberTotalRows": 1000000,
    "NumberLoadedRows": 1000000,
    "NumberFilteredRows": 1,
    "NumberUnselectedRows": 0,
    "LoadBytes": 40888898,
    "LoadTimeMs": 2144,
    "BeginTxnTimeMs": 1,
    "StreamLoadPutTimeMs": 2,
    "ReadDataTimeMs": 325,
    "WriteDataTimeMs": 1933,
    "CommitAndPublishTimeMs": 106,
    "ErrorURL": "http://fe_host:http_port/api/_load_error_log?file=__shard_0/
error_log_insert_stmt_db18266d4d9b4ee5-
abb00ddd64bdf005_db18266d4d9b4ee5_abb00ddd64bdf005"
}
```

■ Stream load导入结果参数如下表。

表 2-25 参数说明

参数	说明
TxnId	导入的事务ID。用户可不感知。
Label	导入Label。由用户指定或系统自动生成。
Status	导入完成状态。 ● Success：表示导入成功。 ● Publish Timeout：该状态也表示导入已经完成，只是数据可能会延迟可见，无需重试。 ● Label Already Exists：Label重复，需更换Label。 ● Fail：导入失败。
ExistingJobStatus	已存在的Label 对应的导入作业的状态。 这个字段只有在当Status 为 "Label Already Exists" 时才会显示。用户可以通过这个状态，知晓已存在Label对应的导入作业的状态。"RUNNING" 表示作业还在执行，"FINISHED" 表示作业成功。
Message	导入错误信息。
NumberTotalRows	导入总处理的行数。
NumberLoadedRows	成功导入的行数。
NumberFilteredRows	数据质量不合格的行数。
NumberUnselectedRows	被where条件过滤的行数。
LoadBytes	导入的字节数。
LoadTimeMs	导入完成时间。单位毫秒。
BeginTxnTimeMs	向Fe请求开始一个事务所花费的时间，单位毫秒。
StreamLoadPutTimeMs	向Fe请求获取导入数据执行计划所花费的时间，单位毫秒。
ReadDataTimeMs	读取数据所花费的时间，单位毫秒。
WriteDataTimeMs	执行写入数据操作所花费的时间，单位毫秒。
CommitAndPublishTimeMs	向Fe请求提交并且发布事务所花费的时间，单位毫秒。
ErrorURL	如果有数据质量问题，通过访问这个URL查看具体错误行。

版权所有 © 华为云计算技术有限公司

说明

由于Stream load是同步的导入方式，所以并不会在Doris系统中记录导入信息，用户无法异步的通过查看导入命令看到Stream load。使用时需监听创建导入请求的返回值获取导入结果。

- 示例2，json数据格式导入。

  准备json格式数据并保存为testjson.json，并将json数据上传至doris客户端:

  {"id": 100, "city": "beijing", "code" : 1}

  – 创建Doris表。
```
CREATE TABLE `doris_testjson01` (
  `id` varchar(32) NOT NULL,
  `city` ARRAY<int(11)>,
  `code` int(11)
) ENGINE=OLAP
DUPLICATE KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
'replication_allocation' = 'tag.location.default: 3',
'in_memory' = 'false',
'storage_format' = 'V2'
);
```
  – curl 命令进行load数据。
```
curl --location-trusted -u admin:{doris集群密码} -H 'format: json' -T testjson.json  https://
fe_host:http_port/api/{doris数据库}/doris_testjson01/_stream_load -k
```
  – 查询数据。
```
select * from doris_testjson01;
```

取消导入

用户无法手动取消Stream Load，Stream Load在超时或者导入错误后会被系统自动取消。

查看 Stream load

用户可以通过show stream load来查看已经完成的stream load任务。

2.4 数据湖分析

2.4.1 多源数据目录

2.4.1.1 概述

多源数据目录（Multi-Catalog）是Doris 1.2.0版本中推出的功能，旨在能够更方便对接外部数据目录，以增强Doris的数据湖分析和联邦数据查询能力。

在之前的Doris版本中，用户数据只有两个层级：Database和Table。当我们需要连接一个外部数据目录时，我们只能在Database或Table层级进行对接。比如通过create external table的方式创建一个外部数据目录中的表的映射，或通过create external database的方式映射一个外部数据目录中的Database。 如果外部数据目录中的Database或Table非常多，则需要用户手动进行一一映射，使用体验不佳。

版权所有 © 华为云计算技术有限公司

而新的Multi-Catalog功能在原有的元数据层级上，新增一层Catalog，构成Catalog->Database->Table的三层元数据层级。其中，Catalog可以直接对应到外部数据目录。目前支持的外部数据目录包括：

1. Hive
2. JDBC: 对接数据库访问的标准接口（JDBC）来访问各式数据库的数据。

该功能将作为之前外表连接方式（External Table）的补充和增强，帮助用户进行快速的多数据目录联邦查询。

基础概念

1. Internal Catalog

   Doris原有的Database和Table都将归属于Internal Catalog。Internal Catalog是内置的默认Catalog，用户不可修改或删除。

2. External Catalog

   可以通过CREATE CATALOG命令创建一个External Catalog。创建后，可以通过SHOW CATALOGS命令查看已创建的Catalog。

3. 切换Catalog

   用户登录Doris后，默认进入Internal Catalog，因此默认的使用和之前版本并无差别，可以直接使用SHOW DATABASES，USE DB等命令查看和切换数据库。

   用户可以通过SWITCH命令切换Catalog。如：

   ```
   SWITCH internal;
   SWITCH hive_catalog;
   ```

   切换后，可以直接通过SHOW DATABASES，USE DB等命令查看和切换对应Catalog中的Database。Doris会自动通过Catalog中的Database和Table。用户可以像使用Internal Catalog一样，对External Catalog中的数据进行查看和访问。

   当前，Doris只支持对 External Catalog中的数据进行只读访问。

4. 删除Catalog

   External Catalog中的Database和Table都是只读的。但是可以删除Catalog（Internal Catalog无法删除）。可以通过DROP CATALOG命令删除一个External Catalog。

   该操作仅会删除Doris中该Catalog的映射信息，并不会修改或变更任何外部数据目录的内容。

5. Resource

   Resource是一组配置的集合。用户可以通过CREATE RESOURCE命令创建一个Resource。之后可以在创建Catalog时使用这个Resource。

   一个Resource可以被多个Catalog使用，以复用其中的配置。

2.4.1.2 Hive

创建 hive catalog

通过连接Hive Metastore，或者兼容Hive Metastore的元数据服务，Doris可以自动获取Hive的库表信息，并进行数据查询。

除了Hive外，很多其他系统也会使用Hive Metastore存储元数据。所以通过Hive Catalog，我们不仅能访问Hive，也能访问使用Hive Metastore作为元数据存储的系统。

- 创建。
  - Hive On OBS
```
CREATE CATALOG hive_catalog PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://127.x.x.x:port',
    'AWS_ACCESS_KEY' = 'ak',
    'AWS_SECRET_KEY' = 'sk',
    'AWS_ENDPOINT' = 'obs.cn-north-4.myhuaweicloud.com',
    'AWS_REGION' = 'cn-north-4',
    'yarn.resourcemanager.address' = '192.X.X.X:port',
    'yarn.resourcemanager.principal' = 'mapred/hadoop.hadoop.com@HADOOP.COM'
);
```
  - Hive On HDFS
```
CREATE CATALOG hive_catalog PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://127.x.x.x:port',
    'dfs.nameservices'='hacluster',
    'dfs.ha.namenodes.hacluster'='3,4',
    'dfs.namenode.rpc-address.hacluster.3'='192.x.x.x:port',
    'dfs.namenode.rpc-address.hacluster.4'='192.x.x.x:port',

'dfs.client.failover.proxy.provider.hacluster'='org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider',
    'yarn.resourcemanager.address' = '192.X.X.X:port',
    'yarn.resourcemanager.principal' = 'mapred/hadoop.hadoop.com@HADOOP.COM'
);
```
- 创建后查询：
```
select * from hive_catalog.DB.test_table;
```

元数据缓存设置

创建Catalog时可以采用参数file.meta.cache.ttl-second来设置Hive分区文件缓存自动失效时间，也可以将该值设置为0来禁用分区文件缓存，时间单位为：秒。示例如下：

```
CREATE CATALOG hive_catalog PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://127.x.x.x:port',
    'AWS_ACCESS_KEY' = 'ak',
    'AWS_SECRET_KEY' = 'sk',
    'AWS_ENDPOINT' = 'obs.cn-north-4.myhuaweicloud.com',
    'AWS_REGION' = 'cn-north-4',
    'file.meta.cache.ttl-second' = '60',
    'yarn.resourcemanager.address' = '192.X.X.X:port',
    'yarn.resourcemanager.principal' = 'mapred/hadoop.hadoop.com@HADOOP.COM'
);
```

Hive 版本

Doris可以正确访问不同Hive版本中的Hive Metastore。在默认情况下，Doris会以Hive2.3版本的兼容接口访问Hive Metastore。你也可以在创建Catalog时指定hive的版本。如访问Hive1.1.0版本：

```
CREATE CATALOG hive_catalog PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://127.x.x.x:port',
    'AWS_ACCESS_KEY' = 'ak',
    'AWS_SECRET_KEY' = 'sk',
    'AWS_ENDPOINT' = 'obs.cn-north-4.myhuaweicloud.com',
    'AWS_REGION' = 'cn-north-4',
    'hive.version' = '1.1.0',
    'yarn.resourcemanager.address' = '192.X.X.X:port',
    'yarn.resourcemanager.principal' = 'mapred/hadoop.hadoop.com@HADOOP.COM'
);
```


版权所有 © 华为云计算技术有限公司

参数说明

表 2-26 参数说明

参数名	参数说明
type	对接外部数据类型，hms：Hive MetaStore。
hive.metastore.uris	hive元数据uri，可在hive配置页面查看。
AWS_ACCESS_KEY	access key，获取方式请参见对象存储服务的**获取访问密钥（AK/SK）**章节。
AWS_SECRET_KEY	secret key，获取方式请参见对象存储服务的**获取访问密钥（AK/SK）**章节。
AWS_ENDPOINT	obs地址，获取方式见请参见对象存储服务的**获取终端节点**章节。
AWS_REGION	obs region，在obs页面查看。
file.meta.cache.ttl-second	元数据缓存设置。
hive.version	Hive版本。
dfs.nameservices	name service名称，与hdfs-site.xml保持一致。
dfs.ha.namenodes.[nameservice ID]	namenode的ID列表，与hdfs-site.xml保持一致。
dfs.namenode.rpc-address.[nameservice ID].[name node ID]	Name node的rpc地址，数量与namenode数量相同，与hdfs-site.xml保持一致。
dfs.client.failover.proxy.provider.[nameservice ID]	HDFS客户端连接活跃namenode的Java类，通常是org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider。
yarn.resourcemanager.address	● 可在yarn-site.xml配置文件查看。 ● 获取参数步骤： 1. 登录MRS集群的Manager页面。 2. 主页>Yarn>实例>ResoureManager>概览>配置文件>yarn-site.xml。 3. 查看地址，返回实例界面查看。
yarn.resourcemanager.principal	● 可在yarn-site.xml配置文件查看。 ● 获取参数步骤： 1. 登录MRS集群的Manager页面。 2. 主页>Yarn>实例>ResoureManager>概览>配置文件>yarn-site.xml。

版权所有 © 华为云计算技术有限公司

列类型映射

表 2-27 参数说明

HMS Type	Doris Type	Comment
boolean	boolean	-
tinyint	tinyint	-
smallint	smallint	-
int	int	-
bigint	bigint	-
date	date	-
timestamp	datetime	-
float	float	-
double	double	-
char	char	-
varchar	varchar	-
decimal	decimal	-
array<type>	array<type>	支持array嵌套，如array<array<int>>。
map<KeyType, ValueType>	map<KeyType, ValueType>	暂不支持嵌套，KeyType 和 ValueType 需要为基础类型。
struct<col1: Type1, col2: Type2, ...>	struct<col1: Type1, col2: Type2, ...>	暂不支持嵌套，Type1, Type2, ... 需要为基础类型。
other	unsupported	-

2.5 通过 JDBC 方式连接 Doris

2.5.1 JDBC 通过非 ssl 方式连接 doris

在应用层进行代码重试和负载均衡时，代码重试需要应用自己多个配置doris前端节点地址。比如发现一个连接异常退出，就自动在其他连接上进行重试。

JDBC Connector

如果使用mysql jdbc connector来连接Doris，可以使用jdbc的自动重试机制:

```
private static String URL = "jdbc:mysql:loadbalance://" +
        "cloudtable-2e68-ya-frontend-2-1-M07K7np5.mycloudtable.com:9030,cloudtable-2e68-ya-
```


版权所有 © 华为云计算技术有限公司

```
frontend-1-1-y5R8YNiy.mycloudtable.com:9030,cloudtable-2e68-ya-frontend-3-1-
fGg7P4tA.mycloudtable.com:9030/demo?" +
        "loadBalanceConnectionGroup=first&ha.enableJMX=true";
```

样例代码：

```
public class Test {
    private static String URL = "jdbc:mysql:loadbalance://" +
        "FE1:9030,FE2:9030,FE3:9030/demo?" +
        "loadBalanceConnectionGroup=first&ha.enableJMX=true";
    static Connection getNewConnection() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        // 认证用的密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解
密，确保安全；
        // 本示例以密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量
        String password = System.getenv("USER_PASSWORD");
        return DriverManager.getConnection(URL, "admin", password);
    }
    public static void main(String[] args) throws Exception {
        Connection c = getNewConnection();
        while (true) {
            try {
                String query = "your sqlString";
                c.setAutoCommit(false);
                Statement s = c.createStatement();
                ResultSet resultSet = s.executeQuery(query);
                System.out.println("begin print");
                while(resultSet.next()) {
                    int id = resultSet.getInt(1);
                    System.out.println("id is: "+id);
                }
                System.out.println("end print");
                Thread.sleep(Math.round(100 * Math.random()));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

2.5.2 JDBC 通过 ssl 方式连接 doris（验证证书）

在应用层进行代码重试和负载均衡时，代码重试需要应用自己多个配置doris前端节点地址。比如发现一个连接异常退出，就自动在其他连接上进行重试。

说明

前提条件：集群必须开启HTTPS。

下载证书请在集群详情页面下载。

1. 在已安装mysql客户端的ecs服务器上先执行以下命令，导入服务器证书。
   - your_certificate_path：自定义证书存放路径。
   - your_truststore_name：自定义truststore名称。
   - your_truststore_password：自定义 truststore密码。

   ```
   keytool -importcert -alias MySQLCACert -file your_certificate_path -keystore your_truststore_name -
   storepass your_truststore_password
   ```

2. 运行该命令的过程中，需要手动输入yes，如下所示：

版权所有 © 华为云计算技术有限公司

图 2-2 运行图

```
Owner: CN=MySQL_Server_5.7.17_Auto_Generated_CA_Certificate
Issuer: CN=MySQL_Server_5.7.17_Auto_Generated_CA_Certificate
Serial number: 1
Valid from: Thu Feb 16 11:42:43 EST 2017 until: Sun Feb 14 11:42:43 EST 2027
Certificate fingerprints:
     MD5:  18:87:97:37:EA:CB:0B:5A:24:AB:27:76:45:A4:78:C1
     SHA1: 2B:0D:D9:69:2C:99:BF:1E:2A:25:4E:8D:2D:38:B8:70:66:47:FA:ED
     SHA256: C3:29:67:1B:E5:37:06:F7:A9:93:DF:C7:B3:27:5E:09:C7:FD:EE:2D:18:86:F4:9C:40:D8:26:CB:DA:95:A0:24
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 1
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

3. 执行以下代码样例。

   以下java代码中your_truststore_path为truststore文件路径，your_truststore_password为上述命令设置的truststore密码。

```
public class Main {
  private static String URL = "jdbc:mysql:loadbalance://" +
          "[FE1_host]:[FE1_port],[FE2_host]:[FE2_port],[FE3_host]:[FE3_port]/[your_database]?" +
          "loadBalanceConnectionGroup=first&ha.enableJMX=true";
  static Connection getNewConnection() throws SQLException, ClassNotFoundException {
      Class.forName("com.mysql.cj.jdbc.Driver");
      // 认证用的密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
      // 本示例以密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量
      String storePassword = System.getenv("STORE_PASSWORD");
      String userPassword = System.getenv("USER_PASSWORD");
      System.setProperty("javax.net.ssl.trustStore","your_truststore_path");
      System.setProperty("javax.net.ssl.trustStorePassword",storePassword);
      String user = "your_username";
      Properties props = new Properties();
      props.setProperty("user", user);
      props.setProperty("password", userPassword);
      props.setProperty("useSSL", "true");
      props.setProperty("requireSSL", "true");
      props.setProperty("verifyServerCertificate", "true");
      props.setProperty("sslMode", "VERIFY_CA");
      return DriverManager.getConnection(URL, props);
  }
  public static void main(String[] args) throws Exception {
      Connection c = getNewConnection();
      try {
          System.out.println("begin print");
          String query = "your sqlString";
          c.setAutoCommit(false);
          Statement s = c.createStatement();
          ResultSet resultSet = s.executeQuery(query);
          while(resultSet.next()) {
              int id = resultSet.getInt(1);
              System.out.println("id is: "+id);
          }
          System.out.println("end print");
          Thread.sleep(Math.round(100 * Math.random()));
          c.close();
      } catch (Exception e) {
          e.printStackTrace();
      }
  }
}
```

2.5.3 JDBC 通过 ssl 方式连接 doris（无需验证证书）

在应用层进行代码重试和负载均衡时，代码重试需要应用自己多个配置doris前端节点地址。比如发现一个连接异常退出，就自动在其他连接上进行重试。

版权所有 © 华为云计算技术有限公司

说明

前提条件：集群必须开启HTTPS。

下载证书请在集群详情页面下载。

样例代码：

```
public class Main {
  private static String URL = "jdbc:mysql:loadbalance://" +
          "[FE1_host]:[FE1_port],[FE2_host]:[FE2_port],[FE3_host]:[FE3_port]/[your_database]?" +
          "loadBalanceConnectionGroup=first&ha.enableJMX=true";
  static Connection getNewConnection() throws SQLException, ClassNotFoundException {
      Class.forName("com.mysql.cj.jdbc.Driver");
      // 认证用的密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解
密，确保安全；
      // 本示例以密码保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量
      String password = System.getenv("USER_PASSWORD");
      String user = "your_username";
      Properties props = new Properties();
      props.setProperty("user", user);
      props.setProperty("password", password);
      props.setProperty("useSSL", "true");
      props.setProperty("requireSSL", "true");
      return DriverManager.getConnection(URL, props);
  }
  public static void main(String[] args) throws Exception {
      Connection c = getNewConnection();
      try {
          System.out.println("begin print");
          String query = "your sqlString";
          c.setAutoCommit(false);
          Statement s = c.createStatement();
          ResultSet resultSet = s.executeQuery(query);
          while(resultSet.next()) {
              int id = resultSet.getInt(1);
              System.out.println("id is: "+id);
          }
          System.out.println("end print");
          Thread.sleep(Math.round(100 * Math.random()));
          c.close();
      } catch (Exception e) {
          e.printStackTrace();
      }
  }
}
```

2.6 开发 Doris 冷热分离应用

2.6.1 应用背景

CloudTable Doris支持冷热数据分离特性。通过该特性，您可以将冷热数据分别存储在不同类型的存储介质中，以降低存储成本。主要适用以下场景：

- 数据存储周期长：面对历史数据的不断增加，存储成本也随之增加。
- 冷热数据访问频率及性能要求不同：热数据访问频率高且需要快速响应，而冷数据访问频率低且响应速度要求不高。

2.6.2 典型场景说明

通过典型场景，我们可以快速学习和掌握Doris的开发过程，并且对冷热分离的应用场景有所了解。

 版权所有 © 华为云计算技术有限公司

场景说明

假定用户开发一个网站系统，test_tbl用于实时用户访问网站的记录，记录数据如下表：

表 2-28 原始数据

timestamp	type	error_code	error_msg	op_id	op_time
2024-03-26 10:36:00	1	404	Resource Not Found	998756	2024-03-26 11:36:00
2024-03-26 10:35:00	1	404	Resource Not Found	998756	2024-03-26 11:35:00
2024-03-26 10:33:00	1	404	Resource Not Found	998756	2024-03-26 11:33:00
2024-03-27 09:10:00	1	200	ok	998756	2024-03-27 10:10:00
2024-03-25 11:08:00	1	404	Resource Not Found	998756	2024-03-25 12:08:00
2024-03-12 22:35:00	1	404	Resource Not Found	998756	2024-03-12 23:35:00
2024-03-12 20:32:00	1	404	Resource Not Found	998756	2024-03-12 21:32:00
2024-03-21 14:39:00	1	404	Resource Not Found	998756	2024-03-21 15:39:00
2024-03-20 19:35:00	1	404	Resource Not Found	998756	2024-03-20 20:35:00

版权所有 © 华为云计算技术有限公司

数据规划

当天整点写入数据，同时一天前数据查询频率较低，节省存储空间设置冷热分离，将一天前数据自动归档到冷存储。

2.6.3 开发思路

功能分解

根据典型场景说明中的业务进行功能分解，需要开发的功能点如下：

表 2-29 冷热分离功能

步骤	代码实现
步骤1：创建自动归档冷数据策略。	请参见自动存储冷数据。
步骤2：数据表关联数据迁移策略。	请参见数据表关联策略。
步骤3：插入数据。	请参见插入数据。
步骤4：查询插入的数据。	请参见查询插入的数据。

2.6.4 样例代码

此章节主要介绍CloudTable Doris冷热分离的使用命令，以及冷数据如何自动存储到obs桶中。

样例代码

1. 自动存储冷数据。

   创建冷数据自动归档到冷存储的数据迁移策略testPolicy。
   ```
   CREATE STORAGE POLICY testPolicy
   PROPERTIES(
   "storage_resource" = "hot_to_cold",
   "cooldown_ttl" = "1d"
   );
   ```
2. 创建test_tbl1数据表并关联数据迁移策略testPolicy。

   创建数据库。
   ```
   CREATE DATABASE IF NOT EXISTS test_db;
   ```
   创建test_tbl1表并关联数据迁移策略。
   ```
   CREATE TABLE IF NOT EXISTS test_db.test_tbl1
   (
       `timestamp` DATETIME NOT NULL COMMENT "日志时间",
       `type` INT NOT NULL COMMENT "日志类型",
       `error_code` INT COMMENT "错误码",
       `error_msg` VARCHAR(1024) COMMENT "错误详细信息",
       `op_id` BIGINT COMMENT "负责人id",
       `op_time` DATETIME COMMENT "处理时间"
   )
   DISTRIBUTED BY HASH(`type`) BUCKETS 10
   PROPERTIES (
   "storage_policy" = "testPolicy",
   ```


版权所有 © 华为云计算技术有限公司

```
"replication_num" = "3"
);
```

3. 每天整点使用Broker Load导入批量数据到test_tbl1表，请参考**Broker Load**。

   创建数据库test_db。

```
CREATE DATABASE test_db;
```

   使用数据库。

```
use test_db;
```

   批量导入数据。

```
LOAD LABEL brokerload_test_csv_label00001
(
DATA INFILE("obs://xxxxx/doris-data/data1/part-m-00002")
INTO TABLE `test_tbl1`
COLUMNS TERMINATED BY ','
FORMAT AS 'csv'
)
WITH BROKER "broker1"
(
"fs.obs.access.key" = "xxxxx",
"fs.obs.secret.key" = "xxxxx",
"fs.obs.endpoint" = "obs.xxx.xxx.com"
);
```

   查看数据插入进度。

```
mysql> show load order by createtime desc limit 1\G;
*************************** 1. row ***************************
         JobId: 18355
         Label: brokerload_test_csv_label00001
         State: FINISHED
      Progress: 100.00% (1/1)
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=9
      TaskInfo: cluster:broker1; timeout(s):14400; max_filter_ratio:0.0; priority:NORMAL
      ErrorMsg: NULL
    CreateTime: 2024-06-03 09:08:40
  EtlStartTime: 2024-06-03 09:08:44
 EtlFinishTime: 2024-06-03 09:08:44
 LoadStartTime: 2024-06-03 09:08:44
LoadFinishTime: 2024-06-03 09:08:46
           URL: NULL
    JobDetails: {"Unfinished backends":{"67bf44bed68d4d3a-9539b655e550e960":
[]},"ScannedRows":9,"TaskNumber":1,"LoadBytes":506,"All backends":
{"67bf44bed68d4d3a-9539b655e550e960":[10007]},"FileNumber":1,"FileSize":639}
 TransactionId: 143
  ErrorTablets: {}
          User: admin
       Comment: 
1 row in set (0.00 sec)
```

说明

fs.obs.access.key, fs.obs.secret.key,fs.obs.endpoint获取方式如下所示：

- fs.obs.access.key与fs.obs.secret.key的获取方式请参见对象存储服务的**获取访问密钥（AK/SK）**章节。
- fs.obs.endpoint的获取方式见请参见对象存储服务的**获取终端节点**章节。

4. 查询插入的数据。

```
mysql> select * from test_tbl1 limit 10;
+---------------------+------+------------+-------------------+--------+---------------------+
| timestamp           | type | error_code | error_msg         | op_id  | op_time             |
+---------------------+------+------------+-------------------+--------+---------------------+
| 2024-03-12 20:32:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-12 21:32:00 |
| 2024-03-12 22:35:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-12 23:35:00 |
| 2024-03-20 19:35:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-20 20:35:00 |
| 2024-03-21 14:39:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-21 15:39:00 |
```


版权所有 © 华为云计算技术有限公司

```
| 2024-03-25 11:08:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-25 12:08:00 |
| 2024-03-26 10:33:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-26 11:33:00 |
| 2024-03-26 10:35:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-26 11:35:00 |
| 2024-03-26 10:36:00 |    1 |        404 | Resource Not Found | 998756 | 2024-03-26 11:36:00 |
| 2024-03-27 09:10:00 |    1 |        200 | ok                 | 998756 | 2024-03-27 10:10:00 |
+---------------------+------+------------+--------------------+--------+---------------------+
9 rows in set (0.16 sec)
```


版权所有 © 华为云计算技术有限公司

3 ClickHouse 应用开发指导

3.1 ClickHouse 组件使用规范

本章节介绍ClickHouse组件使用规范。

建表规范

- 【规则】不要在system库中创建业务表。system数据库是ClickHouse默认的系统数据库，默认数据库中的系统表记录的是系统的配置、元数据等信息数据。业务在使用ClickHouse的时候，需要指定自己业务的数据库进行连接和使用，业务相关的表创建在自己业务库中，不要将业务表创建在系统数据库中，避免对系统数据库造成不必要的影响。
- 【规则】数据库和表的命名尽量不要使用SQL保留字，请注意大小写敏感。如果必须使用一些保留关键字，请使用双引号或者反引号进行转义。
- 【规则】不允许使用字符类型存放时间或日期类数据，尤其是日期字段进行运算或比较的时候。
- 【规则】不允许使用字符类型存放数值类型的数据，尤其是数值字段进行运算或者比较的时候。
- 【建议】不建议表使用Nullable列，可以考虑使用字符串“NA”。

  Nullable类型的列在做查询条件判断时，会进一步做判空等处理，防止造成额外的计算开销。根据现网的历史经验，Nullable类型的字符串查询性能比String慢20%至30%左右，从性能方面考虑，非必要不使用Nullable类型。

数据写入

- 【规则】外部模块保证数据导入的幂等性。

  ClickHouse不支持数据写入的事务保证。通过外部导入数据模块控制数据的幂等性，比如某个批次的数据导入异常，则drop对应分区数据或清理掉导入的数据后，重新导入该分区或批次数据。
- 【规则】大批量少频次的写入数据。

  ClickHouse每次插入数据时，都会生成一到多个part文件，如果data part过多，merge压力会变大，甚至出现各种异常情况影响数据插入。建议每个批次5k到100k行，根据写入字段数量调整写入行数，降低写入节点内存和CPU的压力，每秒不超过1次插入。

版权所有 © 华为云计算技术有限公司

- 【建议】一次只插入一个分区内的数据。

  如果数据属于不同的分区，则每次插入不同分区的数据会独立生成part文件，导致part总数量膨胀，建议一批插入的数据属于同一个分区。

- 【建议】慎用分布式表批量插入。
  - 写分布式表时，数据会分发到集群的所有本地表，每个本地表插入的数据量是总插入量的1/N，batch size可能比较小，会导致data part过多，merge压力变大，甚至出现异常影响数据插入。
  - 数据的一致性问题：数据先在分布式表写入节点的主机落盘，然后数据被异步地发送到本地表所在主机进行存储，中间没有一致性的校验，如果分布式表写入数据的主机出现异常，会存在数据丢失风险。
  - 对于数据写分布式表和数据写本地表相比，分布式表数据写入性能会变慢，单批次分布式表写入节点的磁盘和网络IO会成为性能的瓶颈点。
  - 分布式表转发给各个shard成功与否，插入数据的客户端是无法感知，转发失败的数据会不断重试转发消耗CPU。
  - 只有在数据去重的场景下，可以使用分布式表插入，通过sharding key将要去重的数据转发到同一个shard，方便后续去重查询。

- 【建议】慎用delete、update操作。

  标准SQL的更新、删除操作是同步的，即客户端等服务端返回执行结果在执行。而Clickhouse的update、delete是通过异步方式实现的，当执行update语句时，服务端立即响应，实际上此时数据还没变，而是排队等待，可能会出现操作覆盖的情况，也无法保证操作的原子性。如果业务场景有update、delete等操作，建议使用ReplacingMergeTree、CollapsingMergeTree、VersionedCollapsingMergeTree引擎。

- 【建议】谨慎执行optimize操作。

  Optimize一般会对表做重写操作，建议在业务压力小时候进行操作，否则对IO/MEM/CPU资源有较大消耗，导致业务查询变慢或不可用。

- 【建议】降低对表的修改频次，多个字段修改时，建议使用单条ALTER TABLE命令修改。

  默认场景下ClickHouse执行alter语句是异步执行，对同一张表频繁执行alter操作可能导致业务失败。ClickHouse每次修改列时都会进行元数据的操作，多次操作会增加集群的负担，尤其是zookeeper的负担，影响集群的稳定。可以使用一条语句进行多列的修改。

数据查询

- 【规则】不要使用select *，只查询需要的字段，减少机器负载，提升查询性能。

  OLAP分析场景，一张大宽表通常能有几百甚至上千列，选择其中少数的几列做维度列、指标列计算。在这种场景下，ClickHouse的数据也是按照列存储。如果使用select *，会加重系统的压力。

- 【规则】通过limit限制查询返回的数据量，节省计算资源、减少网络开销。

  如果返回的数据量过大，客户端有可能出现内存溢出等服务异常。在前端使用ClickHouse的场景下，如果要查询的数据量比较大，建议每次可适当的进行分页查询返回数据，以减少查询数据量对网络带宽和计算资源的占用。

- 【规则】关联查询必须大表join小表。

  对于ClickHouse来说，原则上需要把多表join模型提前加工为宽表模型，多个表以及维度表变化比较频繁情况下，不适合进行宽表加工处理，必须使用Join模型以实时查询到最新数据。两个表做join操作，建议大表join小表，必须使用关联条件。

小表的数据量控制在百万~千万行级别，且需要在join前把小表数据通过条件进行有效过滤。

- 【建议】使用GLOBAL JOIN/IN替换普通的JOIN。

  ClickHouse基于分布式表查询会转换成所有分片的本地表操作，再汇总结果。实际使用中，join和global join的执行逻辑差别很大，建议使用global join做分布式表查询。

- 【规则】合理使用数据表的分区字段和索引字段。
  - MergeTree引擎，数据是以分区目录形式进行组织存储的，在进行数据查询时，使用分区可以有效跳过无用的数据文件，减少数据的读取。
  - MergeTree引擎会根据索引字段进行数据排序，并且根据index_granularity的配置生成稀疏索引。根据索引字段查询，能快速过滤数据，减少数据的读取，大大提升查询性能。

- 【建议】明确数据查询的范围。

  增加条件过滤和查询数据周期过滤，缩小数据查询范围。例如查询指定分区，通过指定分区字段会减少底层数据库扫描的文件数量，提升查询性能。例如：700个分区的千列大表，需要查询一个分区中有7000万数据，其他699个分区中无数据，虽然只有一个分区有数据，其他分区无数据，但是查询指定分区为百毫秒级性能，没有指定分区查询性能为1~2秒左右，性能相差20倍。

- 【建议】慎用final查询。

  在查询语句的最后跟上final，通常是对于ReplacingMergeTree引擎，数据不能完全去重情况下，少数开发人员习惯使用final关键字进行实时合并去重操作（merge-on-read），保证查询数据无重复数据。可以通过argMax函数或其他方式规避此问题。

- 【建议】使用物化视图加速查询。

  对于固定查询方式的场景，建议使用物化视图，提前做好数据聚合，相对于查询明细表，性能有数量级的提升。

- 【建议】物化视图创建时不会进行语法校验，只有发生实际数据插入与查询时才会出错。物化视图上线前，需做好充分验证。

3.2 ClickHouse 表引擎概述

背景介绍

表引擎在ClickHouse中的作用十分关键，不同的表引擎决定了：

- 数据存储和读取的位置。
- 支持哪些查询方式。
- 能否并发式访问数据。
- 能否使用索引。
- 是否可以执行多线程请求。
- 数据复制使用的参数。

其中MergeTree和Distributed是ClickHouse表引擎中最重要，也是最常使用的两个引擎，本文将重点进行介绍。

 版权所有 © 华为云计算技术有限公司

概述

表引擎即表的类型，在云数据库ClickHouse中决定了如何存储和读取数据、是否支持索引、是否支持主备复制等。云数据库ClickHouse支持的表引擎，请参见下表。

说明

MergeTree引擎为单副本，无法保证高可用和数据可靠性，建议只在测试环境中使用。Replicated*MergeTree引擎用于生产环境。

表 3-1 表引擎

系列	描述	表引擎	特点
MergeTree	• MergeTree系列引擎适用于高负载任务，支持大数据量的快速写入并进行后续的数据处理，通用程度高且功能强大。 • 该系列引擎的共同特点是支持数据副本、分区、数据采样等特性。	MergeTree	• 基于分区键（partitioning key）的数据分区分块存储。 • 数据索引排序（基于primary key和order by）。 • 支持数据复制（带Replicated前缀的表引擎）。 • 支持数据抽样。 在写入数据时，该系列引擎表会按照分区键将数据分成不同的文件夹，文件夹内每列数据为不同的独立文件，以及创建数据的序列化索引排序记录文件。该结构使得数据读取时能够减少数据检索时的数据量，极大的提高查询效率。
		RelacingMergeTree	用于解决MergeTree表引擎相同主键无法去重的问题，可以删除主键值相同的重复项。
		CollapsingMergeTree	CollapsingMergeTree它通过定义一个sign标记位字段记录数据行的状态。如果sign标记为1，则表示这是一行有效的数据。如果sign标记为-1，则表示这行数据需要被删除。
		VersionedCollapsingMergeTree	在建表语句中新增Version列，用于解决CollapsingMergeTree表引擎乱序写入导致无法正常折叠（删除）的问题。
		SummigMergeTree	用于对主键列进行预先聚合，将所有相同主键的行合并为一行，从而大幅度降低存储空间占用，提升聚合计算性能。

版权所有 © 华为云计算技术有限公司

系列	描述	表引擎	特点
		AggregatingMergeTree	AggregatingMergeTree是预先聚合引擎的一种，用于提升聚合计算的性能。AggregatingMergeTree引擎能够在合并分区时，按照预先定义的条件聚合数据，同时根据预先定义的聚合函数计算数据并通过二进制的格式存入表内。
		GraphiteMergeTree	用于存储Graphite数据并进行汇总，可以减少存储空间，提高Graphite数据的查询效率。
Replicated*MergeTree	ClickHouse中的所有MergeTree家族引擎前面加上Replicated就成了支持副本的合并树引擎。	Replicated*MergeTree系列	Replicated系列引擎借助ZooKeeper实现数据的同步，创建Replicated复制表时通过注册到ZooKeeper上的信息实现同一个分片的所有副本数据进行同步。
Distributed	-	Distributed	本身不存储数据，可以在多个服务器上进行分布式查询。

MergeTree

- 建表语法。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER ClickHouse集群名]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
    INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1,
    INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2
) ENGINE = MergeTree()
ORDER BY expr
[PARTITION BY expr]
[PRIMARY KEY expr]
[SAMPLE BY expr]
[TTL expr [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'], ...]
[SETTINGS name=value, ...]
```

- 使用示例。

```
CREATE TABLE default.test (name1 DateTime,name2 String,name3 String,name4 String,name5 Date)
ENGINE = MergeTree() PARTITION BY toYYYYMM(name5) ORDER BY (name1, name2) SETTINGS
index_granularity = 8192;
```

示例参数说明：

表 3-2 参数说明

参数	说明
ENGINE = MergeTree()	MergeTree表引擎。
PARTITION BY toYYYYMM(name5)	分区，示例数据将以月份为分区，每个月份一个文件夹。

版权所有 © 华为云计算技术有限公司

参数	说明
ORDER BY	排序字段，支持多字段的索引排序，第一个相同的时候按照第二个排序依次类推。
index_granularity = 8192	排序索引的颗粒度，每8192条数据记录一个排序索引值。

说明

如果被查询的数据存在于分区或排序字段中，能极大降低数据查找时间。

ReplacingMergeTree

为了解决MergeTree表引擎相同主键无法去重的问题，云数据库ClickHouse提供了ReplacingMergeTree表引擎，用于删除主键值相同的重复项。

- 建表语句。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER ClickHouse集群名]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = ReplacingMergeTree([ver])
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

CollapsingMergeTree

CollapsingMergeTree表引擎用于消除ReplacingMergeTree表引擎的功能限制。该表引擎要求在建表语句中指定一个标记列Sign，按照Sign的值将行分为两类：Sign=1的行称为状态行，用于新增状态。Sign=-1的行称为取消行，用于删除状态。

- 建表语句。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER ClickHouse集群名]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = CollapsingMergeTree(sign)
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

- 使用示例。
  - 示例数据。

    例如：我们要计算用户在某个网站上访问了多少页面以及他们在那里的时间。 在某个时间点，我们用用户活动的状态写下面的行。

版权所有 © 华为云计算技术有限公司

表 3-3 示例数据

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1
4324182021466249494	5	146	-1
4324182021466249494	6	185	1

- sign— 指定行类型的列名: 1 是一个 “state” 行, -1 是一个“cancel”行。

– 建表Test。

```
CREATE TABLE Test(UserID UInt64,PageViews UInt8,Duration UInt8,Sign Int8)ENGINE = CollapsingMergeTree(Sign) ORDER BY UserID;
```

– 插入数据。

- 第一次插入数据。

```
INSERT INTO Test VALUES (4324182021466249494, 5, 146, 1);
```

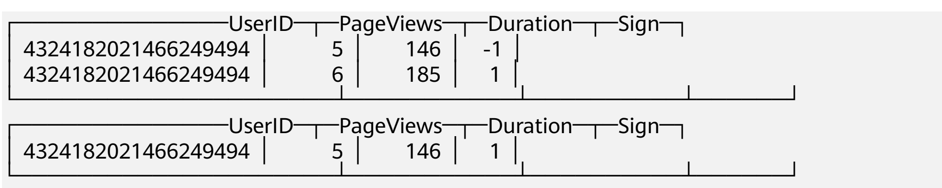
- 第二次插入数据。

```
INSERT INTO Test VALUES (4324182021466249494, 5, 146, -1),(4324182021466249494, 6, 185, 1);
```

– 查看数据。

```
SELECT * FROM Test;
```

查询结果。



```
UserID    PageViews  Duration  Sign
4324182021466249494 │ 5 │ 146 │ -1 │
4324182021466249494 │ 6 │ 185 │ 1 │

UserID    PageViews  Duration  Sign
4324182021466249494 │ 5 │ 146 │ 1 │
```

– 对指定列进行数据聚合。

```
SELECT UserID,sum(PageViews * Sign) AS PageViews,sum(Duration * Sign) AS Duration FROM Test GROUP BY UserID HAVING sum(Sign) > 0;
```

查询结果如下所示。

```
UserID    PageViews  Duration
4324182021466249494 │ 6 │ 185 │
```

– 强制折叠数据，用以下SQL命令。

```
SELECT * FROM Test FINAL;
```

查询结果如下所示。

```
UserID    PageViews  Duration  Sign
4324182021466249494 │ 6 │ 185 │ 1 │
```

VersionedCollapsingMergeTree

为了解决CollapsingMergeTree表引擎乱序写入导致无法正常折叠（删除）问题，云数据库ClickHouse提供了VersionedCollapsingMergeTree表引擎，在建表语句中新增一

版权所有 © 华为云计算技术有限公司

列Version，用于在乱序情况下记录状态行与取消行的对应关系。后台Compaction时会将主键相同、Version相同、Sign相反的行折叠（删除）。

- 建表语句。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER ClickHouse集群名]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = VersionedCollapsingMergeTree(sign, version)
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

- 使用示例。
  - 示例数据。

    例如：我们要计算用户在某个网站上访问了多少页面以及他们在那里的时间。 在某个时间点，我们用用户活动的状态写下面的行。

    表 3-4 示例数据

UserID	PageViews	Duration	Sign	Version
4324182021466249494	5	146	1	1
4324182021466249494	5	146	-1	1
4324182021466249494	6	185	1	2

    - sign— 指定行类型的列名: 1 是一个 “state” 行, -1 是一个“cancel”行。
    - version— 指定对象状态版本的列名。
  - 创建表T。

```
CREATE TABLE T(UserID UInt64,PageViews UInt8,Duration UInt8,Sign Int8,Version UInt8)ENGINE = VersionedCollapsingMergeTree(Sign, Version)ORDER BY UserID;
```

  - 插入两部分不同的数据。

```
INSERT INTO T VALUES (4324182021466249494, 5, 146, 1, 1);
INSERT INTO T VALUES (4324182021466249494, 5, 146, -1, 1),(4324182021466249494, 6, 185, 1, 2);
```

  - 查看数据。

```
SELECT * FROM T;
```

  - 对指定列进行数据聚合。

```
SELECT UserID, sum(PageViews * Sign) AS PageViews,sum(Duration * Sign) AS Duration,Version FROM T GROUP BY UserID, Version HAVING sum(Sign) > 0;
```

    查询显示结果如下。

```
┌──────────────UserID─┬─PageViews─┬─Duration─┬─Version─┐
│ 4324182021466249494 │         6 │      185 │       2 │
└─────────────────────┴───────────┴──────────┴─────────┘
```

  - 强制折叠数据，用以下SQL命令。

```
SELECT * FROM T FINAL;
```


版权所有 © 华为云计算技术有限公司

查询显示结果如下。

```
┌─────────────UserID─┬─PageViews─┬─Duration─┬─Sign─┬─Version─┐
│ 4324182021466249494 │      6 │    185 │   1 │     2 │
└──────────────────────┴───────────┴──────────┴──────┴─────────┘
```

SummingMergeTree

SummingMergeTree表引擎用于对主键列进行预先聚合，将所有相同主键的行合并为一行，从而大幅度降低存储空间占用，提升聚合计算性能。

- 建表语句。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER ClickHouse集群名]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = SummingMergeTree([columns])
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

- 使用示例。
  - 创建一个SummingMergeTree表testTable。

```
CREATE TABLE testTable(id UInt32,value UInt32)ENGINE = SummingMergeTree() ORDER BY id;
```

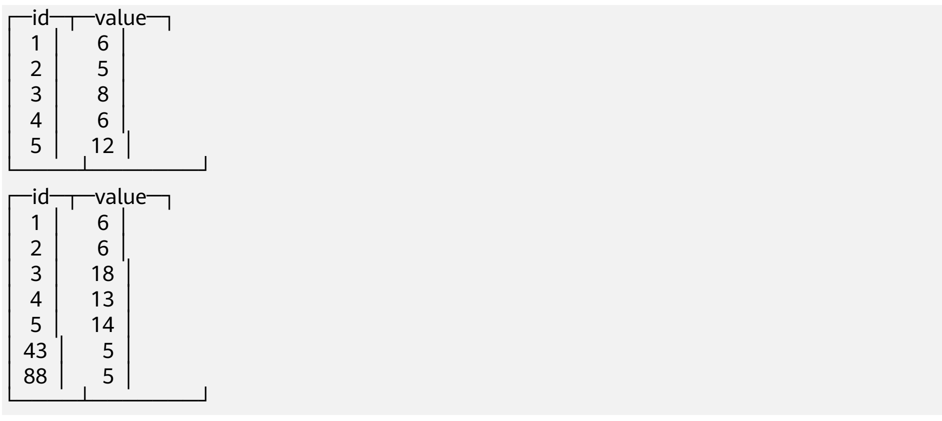
  - testTable表中插入数据。

```
INSERT INTO testTable Values(5,9),(5,3),(4,6),(1,2),(2,5),(1,4),(3,8);
INSERT INTO testTable Values(88,5),(5,5),(3,7),(3,5),(1,6),(2,6),(4,7),(4,6),(43,5),(5,9),(3,6);
```

  - 在未合并parts查询所有数据。

```
SELECT * FROM testTable;
```

  查询结果。

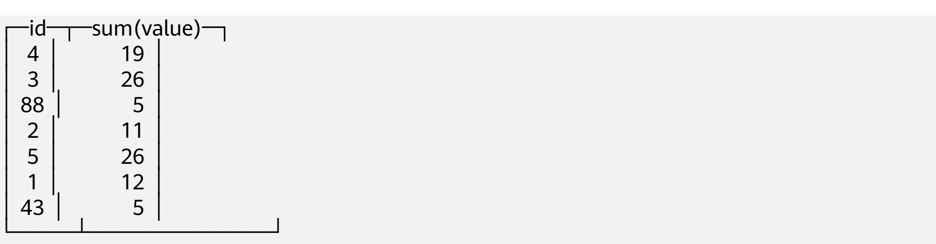


```
┌─id─┬─value─┐
│  1 │     6 │
│  2 │     5 │
│  3 │     8 │
│  4 │     6 │
│  5 │    12 │
└────┴───────┘
┌─id─┬─value─┐
│  1 │     6 │
│  2 │     6 │
│  3 │    18 │
│  4 │    13 │
│  5 │    14 │
│ 43 │     5 │
│ 88 │     5 │
└────┴───────┘
```

  - ClickHouse还没有汇总所有行，需要通过ID进行汇总聚合，需要用到sum和GROUP BY子句。

```
SELECT id, sum(value) FROM testTable GROUP BY id;
```

  查询结果。



```
┌─id─┬─sum(value)─┐
│  4 │         19 │
│  3 │         26 │
│ 88 │          5 │
│  2 │         11 │
│  5 │         26 │
│  1 │         12 │
│ 43 │          5 │
└────┴────────────┘
```


版权所有 © 华为云计算技术有限公司

– 手工执行合并操作。

```
OPTIMIZE TABLE testTable;
```

查询表数据。

```
SELECT * FROM testTable;
```

查询结果。

```
┌─id─┬─value─┐
│  1 │    12 │
│  2 │    11 │
│  3 │    26 │
│  4 │    19 │
│  5 │    26 │
│ 43 │     5 │
│ 88 │     5 │
└────┴───────┘
```

说明

- SummingMergeTree根据ORDER BY排序键作为聚合数据的条件Key。即如果排序key是相同的，则会合并成一条数据，并对指定的合并字段进行聚合。
- 后台执行合并操作时才会进行数据的预先聚合，而合并操作的执行时机无法预测，所以可能存在部分数据已经被预先聚合、部分数据尚未被聚合的情况。因此，在执行聚合计算时，SQL中仍需要使用GROUP BY子句。

AggregatingMergeTree

AggregatingMergeTree表引擎也是预先聚合引擎的一种，用于提升聚合计算的性能。

- 建表语句。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER ClickHouse集群名]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = AggregatingMergeTree()
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[TTL expr]
[SETTINGS name=value, ...]
```

- 使用示例。

  AggregatingMergeTree无单独参数设置，在分区合并时，在每个数据分区内，会按照ORDER BY聚合，使用何种聚合函数，对哪些列字段计算，则是通过定义AggregateFunction函数类型实现。

  – 建表。

```
create table test_table (name1 String,name2 String,name3 AggregateFunction(uniq,String),name4 AggregateFunction(sum,Int),name5 DateTime) ENGINE = AggregatingMergeTree() PARTITION BY toYYYYMM(name5) ORDER BY (name1,name2) PRIMARY KEY name1;
```

  AggregateFunction类型的数据在写入和查询时需要分别调用*state、*merge函数，*表示定义字段类型时使用的聚合函数。如上示例表test_table定义的name3、name4字段分别使用了uniq、sum函数，那么在写入数据时需要调用uniqState、sumState函数，并使用INSERT SELECT语法。

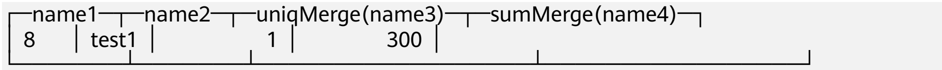
  – 插入数据。

```
insert into test_table select '8','test1',uniqState('name1'),sumState(toInt32(100)),'2021-04-30 17:18:00';
insert into test_table select '8','test1',uniqState('name1'),sumState(toInt32(200)),'2021-04-30 17:18:00';
```

  – 查询数据。

```
select name1,name2,uniqMerge(name3),sumMerge(name4) from test_table group by
name1,name2;
```

查询结果。

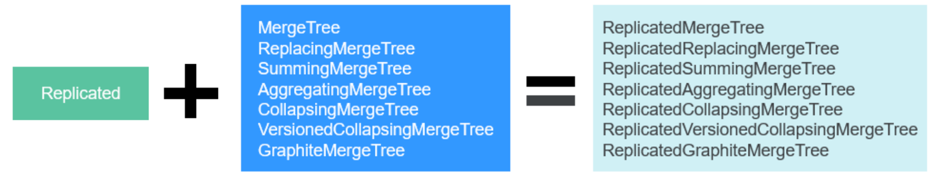


```
┌─name1─┬─name2─┬─uniqMerge(name3)─┬─sumMerge(name4)─┐
│ 8     │ test1 │                1 │             300 │
└───────┴───────┴──────────────────┴─────────────────┘
```

Replicated*MergeTree 引擎

ClickHouse中的所有MergeTree家族引擎前面加上Replicated就成了支持副本的合并树引擎。

图 3-1 合并树引擎图



- Replicated表引擎的创建模板：

```
ENGINE = Replicated*MergeTree('ZooKeeper存储路径','副本名称', ...)
```

表 3-5 参数表

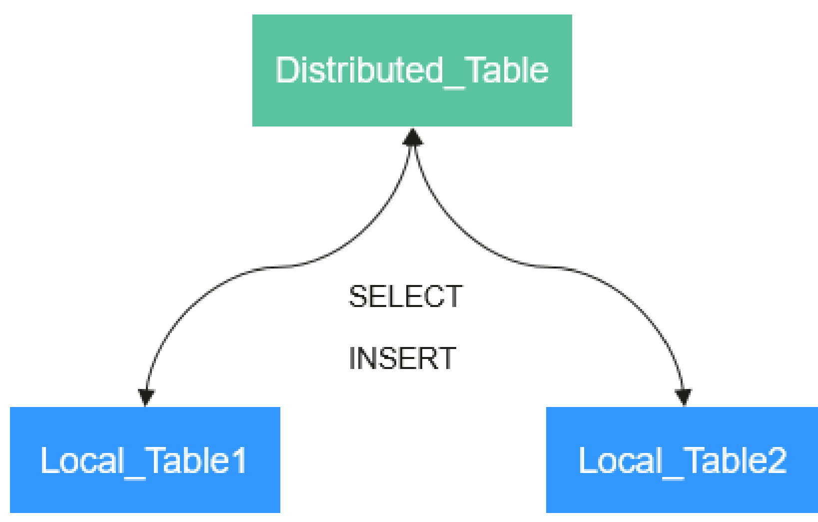
参数	说明
ZooKeeper存储路径	ZooKeeper中该表相关数据的存储路径，建议规范化，如：/clickhouse/tables/{shard}/数据库名/表名。
副本名称	一般用{replica}即可。

Distributed 表引擎

Distributed表引擎本身不存储任何数据，而是作为数据分片的透明代理，能够自动路由数据到集群中的各个节点，分布式表需要和其他本地数据表一起协同工作。分布式表会将接收到的读写任务分发到各个本地表，而实际上数据的存储在各个节点的本地表中。

版权所有 © 华为云计算技术有限公司

图 3-2 Distributed



- Distributed表引擎创建模板：

```
ENGINE = Distributed(cluster_name, database_name, table_name, [sharding_key])
```

表 3-6 Distributed 表参数说明

参数	说明
cluster_name	集群名称，在对分布式表执行读写的过程中，使用集群的配置信息查找对应的ClickHouse实例节点。
database_name	数据库名称。
table_name	数据库下对应的本地表名称，用于将分布式表映射到本地表上。
sharding_key	分片键（可选参数），分布式表会按照这个规则，将数据分发到各个本地表中。

- 使用示例。
  – 先创建一个表名为demo的ReplicatedMergeTree本地表。

```
CREATE TABLE default.demo ON CLUSTER default_cluster( `EventDate` DateTime, `id`
UInt64)ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/default/demo', '{replica}')
PARTITION BY toYYYYMM(EventDate) ORDER BY id;
```

  – 基于本地表demo创建表名为demo_all的Distributed表。

```
CREATE TABLE default.demo_all ON CLUSTER default_cluster( `EventDate` DateTime, `id`
UInt64)ENGINE = Distributed(default_cluster, default, demo, rand());
```

- 分布式表创建规则。
  – 创建Distributed表时需加上on cluster cluster_name，这样建表语句在某一个ClickHouse实例上执行一次即可分发到集群中所有实例上执行。
  – 分布式表通常以本地表加“_all”命名。它与本地表形成一对多的映射关系，之后可以通过分布式表代理操作多张本地表。

 版权所有 © 华为云计算技术有限公司

– 分布式表的表结构尽量和本地表的结构一致。如果不一致，在建表时不会报错，但在查询或者插入时可能会抛出异常。

3.3 SQL 语法参考

3.3.1 数据类型

此章节描述ClickHouse的数据类型。

ClickHouse不支持JSON以及Object('json')数据类型。

数据类型表

表 3-7 数据类型表

分类	关键字	数据类型	描述
整数类型	Int8	Int8	取值范围：【-128，127】
	Int16	Int16	取值范围：【-32768，32767】
	Int32	Int32	取值范围：【-2147483648，2147483647】
	Int64	Int64	取值范围：【-9223372036854775808，9223372036854775807】
浮点类型	Float32	单精度浮点数	同C语言Float类型，单精度浮点数在机内占4个字节，用32位二进制描述。
	Float64	双精度浮点数	同C语言Double类型，双精度浮点数在机内占8个字节，用64位二进制描述。
Decimal类型	Decimal	Decimal	有符号的定点数，可在加、减和乘法运算过程中保持精度。支持几种写法： ● Decimal(P, S) ● Decimal32(S) ● Decimal64(S) ● Decimal128(S) **说明** P：精度，有效范围：[1:38]，决定可以有多少个十进制数字（包括分数）。 S：规模，有效范围：[0：P]，决定数字的小数部分中包含的小数位数。

分类	关键字	数据类型	描述
字符串类型	String	字符串	字符串可以是任意长度的。它可以包含任意的字节集，包含空字节。因此，字符串类型可以代替其他DBMSs中的VARCHAR、BLOB、CLOB等类型。
	FixedString	固定字符串	当数据的长度恰好为N个字节时，FixedString类型是高效的。 在其他情况下，这可能会降低效率。可以有效存储在FixedString类型的列中的值的示例: ● 二进制表示的IP地址（IPv6使用FixedString（16）） ● 语言代码（ru_RU, en_US …） ● 货币代码（USD, RUB …） ● 二进制表示的哈希值（MD5使用FixedString（16），SHA256使用FixedString（32））
时间日期类型	Date	日期	用两个字节存储，表示从1970-01-01（无符号）到当前的日期值。日期中没有存储时区信息。
	DateTime	时间戳	用四个字节（无符号的）存储Unix时间戳。允许存储与日期类型相同的范围内的值。最小值为1970-01-01 00:00:00。时间戳类型值精确到秒（没有闰秒）。时区使用启动客户端或服务器时的系统时区。
	DateTime64	DateTime64	此类型允许以日期（date）加时间（time）的形式来存储一个时刻的时间值。
布尔型	Boolean	Boolean	ClickHouse没有单独的类型来存储布尔值。可以使用UInt8 类型，取值限制为0或1。
数组类型	Array	Array	Array(T)，由T类型元素组成的数组。T可以是任意类型，包含数组类型。但不推荐使用多维数组，ClickHouse对多维数组的支持有限。例如，不能在MergeTree表中存储多维数组。

分类	关键字	数据类型	描述
元组类型	Tuple	Tuple	Tuple(T1, T2, ...)，元组，其中每个元素都有单独的类型，不能在表中存储元组（除了内存表）。它们可以用于临时列分组。在查询中，IN表达式和带特定参数的lambda函数可以来对临时列进行分组。
Domains数据类型	Domains	Domains	Domains类型是特定实现的类型： ● IPv4是与UInt32类型保持二进制兼容的Domains类型，用于存储IPv4地址的值。它提供了更为紧凑的二进制存储的同时支持识别可读性更加友好的输入输出格式。 ● IPv6是与FixedString（16）类型保持二进制兼容的Domain类型，用于存储IPv6地址的值。它提供了更为紧凑的二进制存储的同时支持识别可读性更加友好的输入输出格式。
枚举类型	Enum8	Enum8	取值范围：【-128，127】 Enum保存 'string'= integer的对应关系，例如：Enum8('hello' = 1, 'world' = 2)
	Enum16	Enum16	取值范围：【-32768，32767】
可为空	Nullable	Nullable	除非在ClickHouse服务器配置中另有说明，否则NULL是任何Nullable类型的默认值。Nullable类型字段不能包含在表索引中。 可以与TypeName的正常值存放一起。例如，Nullable(Int8) 类型的列可以存储Int8类型值，而没有值的行将存储NULL。
嵌套类型	nested	nested	嵌套的数据结构就像单元格内的表格。嵌套数据结构的参数（列名和类型）的指定方式与CREATE TABLE查询中的指定方式相同。每个表行都可以对应于嵌套数据结构中的任意数量的行。 示例：Nested(Name1 Type1, Name2 Type2, …)

3.3.2 CREATE DATABASE

本章节介绍创建数据库的基本用法。

CREATE DATABASE

```
CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER ClickHouse集群名];
```

表 3-8 参数说明

参数	说明
db_name	数据库
IF NOT EXISTS	如果CREATE语句中存在IF NOT EXISTS关键字，则当数据库已经存在时，该语句不会创建数据库，且不会返回任何错误。
ON CLUSTER *ClickHouse集群名*	用于指定集群名称。

说明

集群名信息可以使用以下语句的cluster字段获取：

```
select cluster,shard_num,replica_num,host_name from system.clusters;
```

使用示例

- 创建数据库demo。
```
create database demo ON CLUSTER default_cluster;
```
- 查看新建的数据库。
```
host-172-16-30-9 :) show databases;
SHOW DATABASES
Query id: ced1af23-0286-40cc-9c7a-ccbca41178d8
┌─name───────────────┐
│ INFORMATION_SCHEMA │
│ default            │
│ demo               │
│ information_schema │
│ system             │
└────────────────────┘
5 rows in set. Elapsed: 0.002 sec.
```

3.3.3 CREATE TABLE

此章节介绍如何创建表。

创建本地表

```
CREATE TABLE [IF NOT EXISTS] [database_name.]table_name [ON CLUSTER ClickHouse集群名]
(
name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
name2[type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
...
) ENGINE = engine_name()
[PARTITION BY expr_list]
[ORDER BY expr_list]
```


版权所有 © 华为云计算技术有限公司

表 3-9 参数说明

参数	说明
database_name	数据库的名称，默认为当前选择的数据库。
table_name	本地表名。
ON CLUSTER*ClickHouse集群名*	在每一个节点上都创建一个本地表，固定为ON CLUSTER *ClickHouse集群名*。
name1,name2	列名。
ENGINE = engine_name()	表引擎类型。 双副本版集群建表时，需要使用MergeTree系列引擎中支持数据复制的Replicated*引擎，否则副本之间不进行数据复制，导致数据查询结果不一致。使用该引擎建表时，参数填写方式如下。 ● ReplicatedMergeTree('/clickhouse/tables/{database}/{table}/{shard}', '{replica}')，固定配置，无需修改。 ● ReplicatedMergeTree()，等同于ReplicatedMergeTree('/clickhouse/tables/{database}/{table}/{shard}', '{replica}')。
ORDER BY expr_list	排序键，必填项，可以是一组列的元组或任意表达式。
[PARTITION BY expr_list]	分区键。一般按照日期分区，也可以使用其他字段或字段表达式。

示例：

- 创建数据库。请参见**CREATE DATABASE**。
- 使用数据库。
```
use demo;
```
- 创建名为demo.test表。
```
CREATE TABLE demo.test ON CLUSTER default_cluster(`EventDate` DateTime, `id` UInt64)ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/default/test', '{replica}') PARTITION BY toYYYYMM(EventDate) ORDER BY id;
```

复制表结构创建表

可以通过复制表结构创建与源表具有相同结构的表。语法：

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name2 ON CLUSTER ClickHouse集群名
 AS [db.]table_name1 [ENGINE = engine_name];
```

表 3-10 参数说明

参数	说明
db	数据库的名称，默认为当前选择的数据库。
table_name1	被复制表结构的源表。

 版权所有 © 华为云计算技术有限公司

参数	说明
table_name2	新创建的表。
ON CLUSTER *ClickHouse集群名*	在每一个节点上都创建一个表，固定为ON CLUSTER *ClickHouse集群名*。
[ENGINE = engine_name]	表引擎类型。如果没有指定表引擎，默认与被复制表结构的表相同。

示例：

- 创建数据库。
```
create database demo;
```
- 使用数据库。
```
use demo;
```
- 创建数据表。
```
create table demo_t(uid Int32,name String,age UInt32,gender String)engine = TinyLog;
```
- 复制表结构。
```
create table demo_t2 as demo_t;
```
- 查看表结构。

SELECT 语句创建

使用指定的表引擎创建一个与SELECT子句的结果具有相同结构的表，并使用SELECT子句的结果进行填充。

```
CREATE TABLE [IF NOT EXISTS] [database_name.]table_name ENGINE = engine_name AS SELECT ...
```

表 3-11 参数说明

参数	说明
database_name	数据库的名称，默认为当前选择的数据库。
table_name	通过SELECT语句创建的表。
ENGINE = engine_name()	表的引擎类型。
SELECT ...	SELECT子句。

示例：

- 创建表。
```
CREATE TABLE default.demo1 ON CLUSTER default_cluster( `EventDate` DateTime, `id` UInt64)ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/default/demo1', '{replica}') PARTITION BY toYYYYMM(EventDate) ORDER BY id;
```
- 通过SELECT语句创建表。
```
create table t3 ON CLUSTER default_cluster ENGINE =MergeTree() order by EventDate as select * from default.demo1;
```


版权所有 © 华为云计算技术有限公司

- 查询demo1和t3表结构。

```
desc demo1;
```

查询结果显示，两张表结构一样。

```
cloudtable-wlr-click-20230730-06-server-1-1 :) desc demo1;
DESCRIBE TABLE demo1
Query id: 712f6b91-668d-4f70-b160-aac8e52f63a4
┌─name──────┬─type──────┬─default_type─┬─default_expression─┬─comment─┬─
codec_expression─┬─ttl_expression─┐
│ EventDate │ DateTime │             │           │            │         │
│ id        │ UInt64   │             │           │            │         │
└───────────┴──────────┴─────────────┴───────────┴────────────┴─────────┴─
──┴────────────────┘
2 rows in set. Elapsed: 0.001 sec.
cloudtable-wlr-click-20230730-06-server-1-1 :) desc t3;
DESCRIBE TABLE t3
Query id: 11b67532-26f0-49c5-b36d-439d45c279bf
┌─name──────┬─type──────┬─default_type─┬─default_expression─┬─comment─┬─
codec_expression─┬─ttl_expression─┐
│ EventDate │ DateTime │             │           │            │         │
│ id        │ UInt64   │             │           │            │         │
└───────────┴──────────┴─────────────┴───────────┴────────────┴─────────┴─
──┴────────────────┘
2 rows in set. Elapsed: 0.001 sec.
```

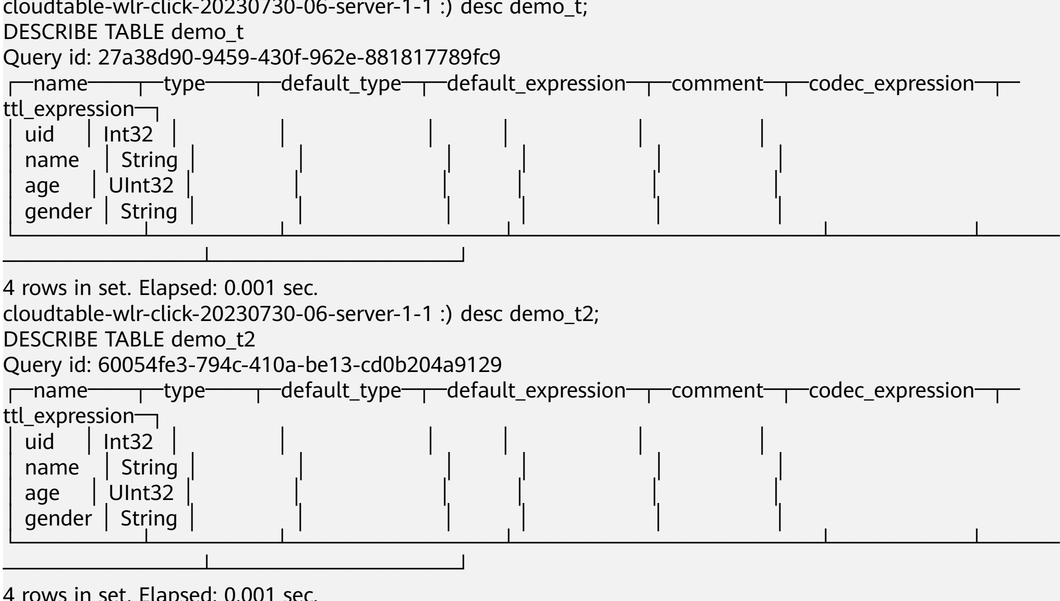
3.3.4 DESC 查询表结构

本章节主要介绍ClickHouse查询表结构的SQL基本语法和使用说明。

基本语法

```
DESC|DESCRIBE TABLE [database_name.]table [INTO OUTFILE filename] [FORMAT format]
```

接复制表结构创建表示例，查询demo_t和demo_2表结构：



```
cloudtable-wlr-click-20230730-06-server-1-1 :) desc demo_t;
DESCRIBE TABLE demo_t
Query id: 27a38d90-9459-430f-962e-881817789fc9
┌─name───┬─type───┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─
ttl_expression─┐
│ uid    │ Int32  │             │                  │        │               │
│ name   │ String │             │                  │        │               │
│ age    │ UInt32 │             │                  │        │               │
│ gender │ String │             │                  │        │               │
└────────┴────────┴─────────────┴──────────────────┴────────┴───────────────┴─
───────────────┘
4 rows in set. Elapsed: 0.001 sec.
cloudtable-wlr-click-20230730-06-server-1-1 :) desc demo_t2;
DESCRIBE TABLE demo_t2
Query id: 60054fe3-794c-410a-be13-cd0b204a9129
┌─name───┬─type───┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─
ttl_expression─┐
│ uid    │ Int32  │             │                  │        │               │
│ name   │ String │             │                  │        │               │
│ age    │ UInt32 │             │                  │        │               │
│ gender │ String │             │                  │        │               │
└────────┴────────┴─────────────┴──────────────────┴────────┴───────────────┴─
───────────────┘
4 rows in set. Elapsed: 0.001 sec.
```

3.3.5 CREATE VIEW

本章节介绍如何在ClickHouse中创建普通视图。

版权所有 © 华为云计算技术有限公司

创建视图

```
CREATE VIEW [IF NOT EXISTS] [db.]view_name [ON CLUSTER ClickHouse集群名] AS SELECT ...
```

表 3-12 参数说明

参数	说明
db	数据库的名称，默认为当前选择的数据库。
view_name	视图名。
[ON CLUSTER ClickHouse集群名]	在每一个节点上都创建一个视图，固定为ON CLUSTER ClickHouse集群名。
SELECT ...	SELECT子句。当数据写入视图中SELECT子句所指定的源表时，插入的数据会通过SELECT子句查询进行转换并将最终结果插入到视图中。

示例：

1. 创建源表。
   ```
   create table DB.table1 ON CLUSTER default_cluster (id Int16,name String) ENGINE = MergeTree()
   ORDER BY (id);
   ```
2. 创建视图。
   ```
   CREATE VIEW test_view ON CLUSTER default_cluster AS SELECT * FROM DB.table1;
   ```
3. 插入数据到源表中。
   ```
   insert into DB.table1 values(1,'X'),(2,'Y'),(3,'Z');
   ```
4. 查询视图。
   ```
   SELECT * FROM test_view;
   ```
5. 删除视图。
   ```
   drop table test_view ON CLUSTER default_cluster;
   ```

说明

- 如果建表语句中包含了“ON CLUSTER ClickHouse集群名”，删除表命令：
  ```
  drop table 表名 ON CLUSTER default_cluster;
  ```
- 如果建表语句不包含“ON CLUSTER ClickHouse集群名”，删除表命令：
  ```
  drop table 表名;
  ```

3.3.6 CREATE MATERIALIZED VIEW

本章节介绍如何在ClickHouse中创建物化视图。

创建物化视图

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [db.]Materialized_name [TO[db.]name] [ON
CLUSTERClickHouse集群名]
ENGINE = engine_name()
ORDER BY expr
[POPULATE]
AS SELECT ...
```

 版权所有 © 华为云计算技术有限公司

表 **3-13** 参数说明

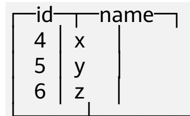
参数	说明
db	数据库的名称，默认为当前选择的数据库。
Materialized_name	物化视图名。
TO[db.]name	将物化视图的数据写入到新表中。
[ON CLUSTER*ClickHouse集群名*]	在每一个节点上都创建一个物化视图，固定为ON CLUSTER *ClickHouse集群名*。
ENGINE = engine_name()	表引擎类型。
[POPULATE]	POPULATE关键字。如果创建物化视图时指定了POPULATE关键字，则在创建时将SELECT子句所指定的源表数据插入到物化视图中。不指定POPULATE关键字时，物化视图只会包含在物化视图创建后新写入源表的数据。 **说明** 一般不推荐使用POPULATE关键字，因为在物化视图创建期间写入源表的数据将不会写入物化视图中。
SELECT ...	SELECT子句。当数据写入物化视图中SELECT子句所指定的源表时，插入的数据会通过SELECT子句查询进行转换并将最终结果插入到物化视图中。 **说明** SELECT查询可以包含DISTINCT、GROUP BY、ORDER BY和LIMIT等，但是相应的转换是在每个插入数据块上独立执行的。

示例：

1. 创建源表。
   ```
   create table DB.table1 ON CLUSTER default_cluster (id Int16,name String) ENGINE = MergeTree()
   ORDER BY (id);
   ```
2. 插入数据。
   ```
   insert into DB.table1 values(1,'X'),(2,'Y'),(3,'Z');
   ```
3. 创建基于源表的物化视图。
   ```
   CREATE MATERIALIZED VIEW demo_view ON CLUSTER default_cluster ENGINE = MergeTree() ORDER
   BY (id) AS SELECT * FROM DB.table1;
   ```
4. 查询物化视图。
   ```
   SELECT * FROM demo_view;
   ```
   说明

   查询数据为空，说明未指定POPULATE关键字时，查询不到物化视图创建前写入源表的数据。
5. DB.table1表中插如数据。
   ```
   insert into demo_view values(4,'x'),(5,'y'),(6,'z');
   ```
6. 查询物化视图。
   ```
   SELECT * FROM demo_view;
   ```
   查询结果。

版权所有 © 华为云计算技术有限公司



3.3.7 INSERT INTO

本章节介绍如何插入数据。

基本语法

- 标准格式插入数据。
```
INSERT INTO [db.]table [(c1, c2, c3)] VALUES (v11, v12, v13), (v21, v22, v23), ...
```

  说明

  对于存在于表结构中但不存在于插入列表中的列，它们将会按照如下方式填充数据：

  - 如果存在DEFAULT表达式，根据DEFAULT表达式计算被填充的值。
  - 如果没有定义DEFAULT表达式，则填充零或空字符串。

  接复制表结构创建表示例，插入数据：
```
insert into demo_t values(1,'Candy','23','M'),(2,'cici','33','F');
```
- 使用SELECT的结果写入。
```
INSERT INTO [db.]table [(c1, c2, c3)] SELECT ...
```

  说明

  写入的列与SELECT的列的对应关系是使用位置来进行对应的，它们在SELECT表达式与INSERT中的名称可以是不同的。需要对它们进行对应的类型转换。

  除了VALUES格式之外，其他格式中的数据都不允许出现诸如now()，1+2等表达式。VALUES格式允许您有限度的使用这些表达式，但是不建议您这么做，因为执行这些表达式很低效。

3.3.8 SELETC

描述如何使用SELECT语句查询数据。

基本语法

```
SELECT [DISTINCT] expr_list
[FROM [database_name.]table | (subquery) | table_function] [FINAL]
[SAMPLE sample_coeff]
[ARRAY JOIN ...]
[GLOBAL] [ANY|ALL|ASOF] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER|SEMI|ANTI] JOIN (subquery)|table
(ON <expr_list>)|(USING <column_list>)
[PREWHERE expr]
[WHERE expr]
[GROUP BY expr_list] [WITH TOTALS]
[HAVING expr]
[ORDER BY expr_list] [WITH FILL] [FROM expr] [TO expr] [STEP expr]
[LIMIT [offset_value, ]n BY columns]
[LIMIT [n, ]m] [WITH TIES]
[UNION ALL ...]
[INTO OUTFILE filename]
[FORMAT format]
```

示例：

- 查看ClickHouse集群信息。
```
select * from system.clusters;
```


版权所有 © 华为云计算技术有限公司

- 显示当前节点设置的宏。
```
select * from system.macros;
```
- 查看数据库容量。
```
select sum(rows) as "总行数", formatReadableSize(sum(data_uncompressed_bytes)) as "原始大小",
formatReadableSize(sum(data_compressed_bytes)) as "压缩大小",
round(sum(data_compressed_bytes) / sum(data_uncompressed_bytes) * 100, 0) "压缩率" from
system.parts;
```
- 查询test表容量。where条件根据实际情况添加修改。
```
select sum(rows) as "总行数", formatReadableSize(sum(data_uncompressed_bytes)) as "原始大小",
formatReadableSize(sum(data_compressed_bytes)) as "压缩大小",
round(sum(data_compressed_bytes) / sum(data_uncompressed_bytes) * 100, 0) "压缩率" from
system.parts where table in ('test') and partition like '2020-11-%' group by table;
```

3.3.9 ALTER TABLE 修改表结构

本章节主要介绍ClickHouse修改表结构的SQL基本语法和使用说明。

基本语法

```
ALTER TABLE [database_name].name [ON CLUSTER ClickHouse集群名] ADD|DROP|CLEAR|COMMENT|
MODIFY COLUMN ...
```

说明

ALTER仅支持 *MergeTree ， Merge以及Distributed等引擎表。

示例：

1. 创建表DB_table1。
```
CREATE TABLE DB_table1 ON CLUSTER default_cluster(Year UInt16,Quarter UInt8,Month
UInt8,DayofMonth UInt8,DayOfWeek UInt8,FlightDate Date,FlightNum String,Div5WheelsOff
String,Div5TailNum String)ENGINE = MergeTree() PARTITION BY toYYYYMM(FlightDate) PRIMARY
KEY (intHash32(FlightDate)) ORDER BY (intHash32(FlightDate),FlightNum) SAMPLE BY
intHash32(FlightDate) SETTINGS index_granularity= 8192;
```
2. 给DB_table1增加列test。
```
ALTER TABLE DB_table1 ADD COLUMN test String DEFAULT 'defaultvalue';
```
查表。
```
desc DB_tables;
```
3. 修改表DB_table1列Year类型为UInt8。
```
ALTER TABLE DB_table1 MODIFY COLUMN Year UInt8;
```
查表结构。
```
desc DB_tables;
```
4. 删除表DB_table1列test。
```
ALTER TABLE DB_table1 DROP COLUMN test;
```
查表。
```
desc DB_tables;
```
5. 修改表DB_table1列Month为Month_test。
```
ALTER TABLE DB_table1 RENAME COLUMN Month to Month_test;
```
查表。
```
desc DB_tables;
```

3.3.10 DROP 删除表

此章节主要介绍ClickHouse删除表的SQL基本语法和使用说明。

版权所有 © 华为云计算技术有限公司

基本语法

```
DROP [TEMPORARY] TABLE [IF EXISTS] [database_name.]name [ON CLUSTER cluster] [SYNC]
```

示例：

- 删除表t1。
  ```
  drop table t1 SYNC;
  ```

  说明

  - 在删除复制表时，因为复制表需要在Zookeeper上建立一个路径，存放相关数据。ClickHouse默认的库引擎是原子数据库引擎，删除Atomic数据库中的表后，它不会立即删除，而是会在24小时后删除。在删除表时，加上SYNC字段，即可解决该问题，例如：drop table t1 SYNC;
  - 删除本地表和分布式表，则不会出现该问题，可不带SYNC字段，例如：drop table t1;
  - 如果建表语句中包含了“ON CLUSTER ClickHouse集群名”，删除表命令：
    ```
    drop table 表名 ON CLUSTER default_cluster;
    ```
  - 如果建表语句不包含“ON CLUSTER ClickHouse集群名”，删除表命令：
    ```
    drop table 表名;
    ```
  - 删除数据表前，需确认此数据表是否应用中，以免引起不必要的麻烦。删除数据表后可在24小时内恢复，超过24小时无法恢复。恢复命令如下：
    ```
    set allow_experimental_undrop_table_query = 1;
    UNDROP TABLE 数据表名;
    ```

3.3.11 SHOW 显示数据库和表信息

此章节主要介绍ClickHouse显示数据库和表信息的SQL基本语法和使用说明。

基本语法

```
show databases;
show tables;
```

示例：

- 查询数据库。
  ```
  show databases;
  ```
- 查询表信息。
  ```
  show tables;
  ```

3.4 数据迁移同步

3.4.1 数据导入导出

本章节主要介绍使用ClickHouse客户端导入导出文件数据的基本语法和使用说明。

CSV 格式数据导入导出

- CSV格式数据导入。
  - 非安全集群
    ```
    cat csv_ssl | ./clickhouse client --host 192.168.x.x --port port --user admin --password password
    --database test010 --query="INSERT INTO test145 FORMAT CSV"
    ```
  - 安全集群
    ```
    cat csv_no_ssl | ./clickhouse client --host 192.168.x.x --port port --user admin --password
    password --config-file ./config.xml --database test010 --query="INSERT INTO test146 FORMAT
    CSV"
    ```

1. host：主机名/ClickHouse实例IP地址。
2. port：端口号（在集群详情页面查看）。
3. user：创建集群时创建的用户名。
4. database：数据库名。
5. password：创建集群时，创建的密码。
6. INSERT INTO：后面跟数据表。
7. cat文件路径：文件存放的路径，路径自定义。
8. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

- CSV格式数据导出。
  - 非安全集群
    ```
    ./clickhouse client --host 192.168.x.x --port port --user admin --password Password --database test010 -m --query="select * from test139 FORMAT CSV" > ./csv_no_ssl
    ```
  - 安全集群
    ```
    ./clickhouse client --host 192.168.x.x --port port --user admin --password password --config-file ./config.xml --database test010 -m --query="select * from test139 FORMAT CSV" > ./csv_no_ssl
    ```

1. host：主机名/ClickHouse实例IP地址。
2. port：端口号（在集群详情页面查看）。
3. user：创建集群时创建的用户名。
4. database：数据库名。
5. password：创建集群时，创建的密码。
6. SELECT * FROM：后面跟数据表。
7. ./csv_no_ssl：指文件存放路径，存放路径自定义。
8. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

parquet 格式数据导入导出

- parquet格式数据导入。
  - 非安全集群
    ```
    cat parquet_no_ssl.parquet | ./clickhouse client --host 192.168.x.x --port port --user admin --password password --database test010 --query="INSERT INTO test145 FORMAT Parquet"
    ```
  - 安全集群
    ```
    cat parquet_no_ssl.parquet | ./clickhouse client --host 192.168.x.x --port port --user admin --password password --config-file ./config.xml --database test010 --query="INSERT INTO test146 FORMAT Parquet"
    ```

1. parquet_no_ssl.parquet：表示格式文件存放路径，路径自定义。
2. host：主机名/ClickHouse实例IP地址。
3. port：端口号（在集群详情页面查看）。
4. user：创建集群时创建的用户名。
5. database：数据库名。
6. password：创建集群时，创建的密码。
7. INSERT INTO：后面跟数据表。
8. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

- parquet格式数据导出。
  - 非安全集群

```
./clickhouse client --host 192.168.x.x --port port --user admin --password password --database test010 -m --query="select * from test139 FORMAT Parquet" > ./parquet_no_ssl.parquet
```

  – 安全集群

```
./clickhouse client --host 192.168.x.x --port port --user admin --password password --config-file ./config.xml --database test010 -m --query="select * from test139 FORMAT Parquet" > ./parquet_ssl.parque
```

1. host：主机名/ClickHouse实例IP地址。
2. port：端口号（在集群详情页面查看）。
3. user：创建集群时创建的用户名。
4. database：数据库名。
5. password：创建集群时，创建的密码。
6. select * from：后面跟数据表。
7. ./parquet_no_ssl.parquet：代表parquet格式文件导出路径，路径自定义。
8. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

ORC 格式数据导入导出

- ORC格式数据导入。
  – 非安全集群

```
cat orc_no_ssl.orc | ./clickhouse client --host 192.168.x.x --port port --user admin --password password --database test010 --query="INSERT INTO test143 FORMAT ORC"
```

  – 安全集群

```
cat orc_no_ssl.orc | ./clickhouse client --host 192.168.x.x --port port --user admin --password password --config-file ./config.xml --database test010 --query="INSERT INTO test144 FORMAT ORC
```

1. cat orc_no_ssl.orc：orc格式文件存放路径，路径自定义。
2. host：主机名/ClickHouse实例IP地址。
3. port：端口号（在集群详情页面查看）。
4. user：创建集群时创建的用户名。
5. database：数据库名。
6. password：创建集群时，创建的密码。
7. INSERT INTO：后面跟数据表。
8. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

- ORC格式数据导出。
  – 安全集群

```
./clickhouse client --host 192.168.x.x --port port --user admin --password password --config-file ./config.xml --database test010 -m --query="select * from test139 FORMAT ORC" > ./orc_ssl.orc
```

  – 非安全集群

```
./clickhouse client --host 192.168.x.x --port port --user admin --password password --database test010 -m --query="select * from test139 FORMAT ORC" > ./orc_no_ssl.orc
```

1. host：主机名/ClickHouse实例IP地址。
2. port：端口号（在集群详情页面查看）。
3. user：创建集群时创建的用户名。
4. database：数据库名。
5. password：创建集群时，创建的密码。
6. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

版权所有 © 华为云计算技术有限公司

7. select * from：后面跟数据表。
8. /opt/student.orc：导出的ORC格式文件路径，路径自定义。

JSON 格式数据导入导出

- JSON格式数据导入。
  - 非安全集群
    ```
    cat ./jsonnossl.json | ./clickhouse client --host 192.168.x.x --port port --user admin --password password --database test010 --query="INSERT INTO test141 FORMAT JSON"
    ```
  - 安全集群
    ```
    cat ./jsonssl.json | ./clickhouse client --host 192.168.x.x --port port --user admin --password password --config-file ./config.xml --database test010 --query="INSERT INTO test142 FORMAT JSON"
    ```

1. cat文件路径：导入文件路径，路径自定义。
2. host：主机名/ClickHouse实例IP地址。
3. port：端口号（在集群详情页面查看）。
4. user：创建集群时创建的用户名。
5. database：数据库名。
6. password：创建集群时，创建的密码。
7. INSERT INTO：后面跟数据表。
8. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

- JSON格式数据导出。
  - 安全集群
    ```
    ./clickhouse client --host 192.168.x.x --port port --user admin --password password --database test010 -m --query="select * from test139 FORMAT JSON" > ./jsonnossl.json
    ```
  - 非安全集群
    ```
    ./clickhouse client --host 192.168.x.x --port port --user admin --password password --config-file ./config.xml --database test010 -m --query="select * from test139 FORMAT JSON" > ./jsonssl.json
    ```

1. host：主机名/ClickHouse实例IP地址。
2. port：端口号（在集群详情页面查看）。
3. user：创建集群时创建的用户名。
4. database：数据库名。
5. password：创建集群时，创建的密码。
6. SELECT * FROM：后面跟数据表。
7. ./jsonssl.json：文件导出路径，路径自定义。
8. config-file ./config.xml：指定配置文件，请参见**ClickHouse安全通道**章节。

3.4.2 ClickHouse 访问 RDS MySQL 服务

ClickHouse面向OLAP场景提供高效的数据分析能力，支持通过MySQL等数据库引擎将远程数据库服务器中的表映射到ClickHouse集群中，后续可以在ClickHouse中进行数据分析。以下操作通过ClickHouse集群和RDS服务下的MySQL数据库实例对接进行举例说明。

前提条件

- 已提前准备好对接的RDS数据库实例及数据库用户名、密码。详细操作可以参考**创建和连接RDS数据库实例**。

- 已成功创建ClickHouse集群且集群和实例状态正常。

约束限制

- RDS数据库实例和ClickHouse集群在相同的VPC和子网内。
- 在进行数据同步操作时需要评估对源数据库和目标数据库性能的影响，同时建议您在业务低峰期执行数据同步。
- 当前ClickHouse支持和RDS服务下的MySQL、PostgreSQL实例进行对接，不支持对接SQL Server实例。

ClickHouse 通过 MySQL 引擎对接 RDS 服务

MySQL引擎用于将远程的MySQL服务器中的表映射到ClickHouse中，并允许您对表进行INSERT和SELECT查询，以方便您在ClickHouse与MySQL之间进行数据交换。

- MySQL引擎使用语法：

```
CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER cluster]
ENGINE = MySQL('host:port', ['database' | database], 'user', 'password')
```

表 3-14 MySQL 数据库引擎参数说明

参数	描述
hostport	RDS服务MySQL数据库实例IP地址和端口。
database	RDS服务MySQL数据库名。
user	RDS服务MySQL数据库用户名。
password	RDS服务MySQL数据库用户密码。

MySQL引擎使用示例：

a. 连接到RDS服务的MySQL数据库。详细操作可以参考**RDS服务MySQL实例连接**。

b. 在MySQL数据库上创建表，并插入数据。

c. 使用客户端命令连接ClickHouse。

非安全集群连接命令

```
./clickhouse client --host 集群内网地址  --port 端口 --user admin --password password
```

安全集群连接命令，详细操作请参见**ClickHouse安全通道**章节。

```
./clickhouse client --host 集群内网地址  --port 端口 --user admin --password password --secure
--config-file /root/config.xml
```

说明

集群内网地址：集群详情页面中集群访问地址，这里替换成您自己购买的集群的访问地址。

d. 在ClickHouse中创建MySQL引擎的数据库，创建成功后自动与MySQL服务器交换数据。

```
CREATE DATABASE mysql_db ENGINE = MySQL('RDS服务MySQL数据库实例IP地址:MySQL数据库实例端口', 'MySQL数据库名', 'MySQL数据库用户名', 'MySQL数据库用户名密码');
```

e. 切换到新建的数据库mysql_db，并查询表数据。

```
USE mysql_db;
```

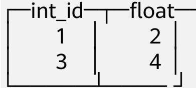
 版权所有 © 华为云计算技术有限公司

在ClickHouse中查询MySQL数据库表数据。

```
SELECT * FROM mysql_table;
┌─int_id─┬─float─┐
│      1 │     2 │
└────────┴───────┘
```

新增插入数据后也可以正常进行查询。

```
INSERT INTO mysql_table VALUES (3,4);
SELECT * FROM mysql_table;
```



3.5 开发程序

3.5.1 典型场景说明

通过典型场景，用户可以快速学习和掌握ClickHouse的开发过程，并且对关键的接口函数有所了解。

场景说明

假定用户需要开发一个应用程序，用于存储或根据一定条件查询人员的姓名、年龄和入职日期。主要操作步骤：

1. 建立数据库的连接。
2. 建立一张人员信息表。
3. 插入数据（样例代码中数据为随机生成）。
4. 根据条件查询数据。

3.5.2 开发思路

ClickHouse作为一款独立的DBMS系统，使用SQL语言就可以进行常见的操作。开发程序示例中，全部通过clickhouse-jdbc API接口来进行描述。

- 设置属性：设置连接ClickHouse服务实例的参数属性。
- 建立连接：建立和ClickHouse服务实例的连接。
- 创建库：创建ClickHouse数据库。
- 创建表：创建ClickHouse数据库下的表。
- 插入数据：插入数据到ClickHouse表中。
- 查询数据：查询ClickHouse表数据。
- 删除表：删除已创建的ClickHouse表。

3.5.3 准备开发和运行环境

准备开发环境

在进行应用开发时，要准备的开发和运行环境如表1所示。

版权所有 © 华为云计算技术有限公司

表 3-15 开发环境

准备项	说明
操作系统	● 开发环境：Windows系统，支持Windows7以上版本。 ● 运行环境：Linux系统。 如需在本地调测程序，运行环境需要和集群业务平面网络互通。
安装JDK	安装JDK，版本为1.8.0_272。
安装和配置IntelliJ IDEA	开发环境的基本配置，建议使用2019.1或其他兼容版本。 **说明** ● 如果使用IBM JDK，请确保IntelliJ IDEA中的JDK配置为IBM JDK。 ● 如果使用Oracle JDK，请确保IntelliJ IDEA中的JDK配置为Oracle JDK。 ● 如果使用Open JDK，请确保IntelliJ IDEA中的JDK配置为Open JDK。 ● 不同的IntelliJ IDEA不要使用相同的workspace和相同路径下的示例工程。
安装Maven	开发环境的基本配置。用于项目管理，贯穿软件开发生命周期。
准备开发用户	准备用于应用开发的ClickHouse集群用户并授予相应权限。
7-zip	用于解压“*.zip”和“*.rar”文件，支持7-Zip 16.04版本。

3.5.4 配置并导入样例工程

背景信息

获取ClickHouse开发样例工程，将工程导入到IntelliJ IDEA开始样例学习。

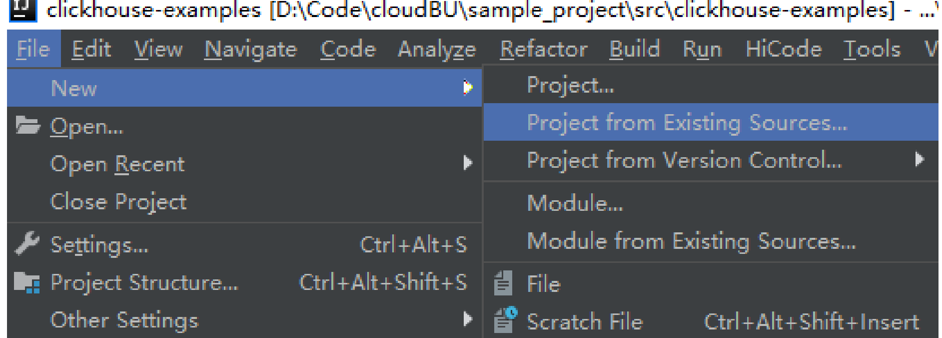
操作场景

ClickHouse针对多个场景提供样例工程，帮助客户快速学习ClickHouse工程。

操作步骤

步骤1 在应用开发环境中，导入代码样例工程到IntelliJ IDEA开发环境。

1. 在IDEA界面选择“File>New>Project from Existing Sources”。



版权所有 © 华为云计算技术有限公司

2. 在显示的“Select File or Directory to Import”对话框中，选择“clickhouse-examples”文件夹中的“pom.xml”文件，单击“OK”。
3. 确认后续配置，单击“Next”，如无特殊需求，使用默认值即可。
4. 选择推荐的JDK版本，单击“Finish”完成导入。

步骤2 工程导入完成后，修改样例工程的“conf”目录下的“clickhouse-example.properties”文件，根据实际环境信息修改相关参数。

```
ipList=
sslUsed=false
httpPort=8123
httpsPort=
CLICKHOUSE_SECURITY_ENABLED=false
user=default
password=
clusterName=default_cluster
databaseName=testdb
tableName=testtb
batchRows=10000
batchNum=10
clickhouse_dataSource_ip_list=ip:8123,ip:8123
native_dataSource_ip_list=ip:9000,ip:9000
```

表 3-16 配置说明表

配置名称	默认值	含义
iPList	-	必填参数，配置为clickhouse节点的集群访问地址列表。 登录cloudtable控制台，单击集群名称，进入集群详情页，拿到集群访问地址。 多个地址使用逗号分隔，例如配置为“cloudtable-wlr-cli-server-1-1-2lIWzDO9.mycloudtable.com,cloudtable-wlr-cli-server-2-1-iqVWp2Mo.mycloudtable.com”。
sslUsed	false	是否启用ssl加密，默认为“false”。
httpPort	8123	连接的HTTP端口，值为8123。
httpsPort	-	连接使用的HTTPS端口，值为8443。
CLICKHOUSE_SECURITY_ENABLED	false	ClickHouse安全模式开关，普通模式集群时该参数填写为false。
user	default	表1中已准备好的开发用户。
password	-	开发用户对应的密码。
clusterName	default_cluster	ClickHouse逻辑集群名称，保持默认值。
databaseName	testdb	样例代码工程中需要创建的数据库名称，可以根据实际情况修改。
tableName	testtb	样例代码工程中需要创建的表名称，可以根据实际情况修改。

 版权所有 © 华为云计算技术有限公司

配置名称	默认值	含义
batchRows	10000	一个批次写入数据的条数。
batchNum	10	写入数据的总批次。
clickhouse_dataSource_ip_list	-	clickhouse节点的ip和http端口集合，例如配置为：cloudtable-wlr-cli-server-1-1-2lIWzDO9.mycloudtable.com:8123,cloudtable-wlr-cli-server-2-1-iqVWp2Mo.mycloudtable.com:8123
native_dataSource_ip_list	-	clickhouse节点的ip和tcp端口集合，格式参考：cloudtable-wlr-cli-server-1-1-2lIWzDO9.mycloudtable.com:9000,cloudtable-wlr-cli-server-2-1-iqVWp2Mo.mycloudtable.com:9000

----结束

3.5.5 样例代码说明

3.5.5.1 设置属性

功能介绍

可以通过Properties设置连接属性。

如下样例代码设置socket超时时间为60s， 设置不使用SSL。

代码样例

```
Properties clickHouseProperties = new Properties();
clickHouseProperties.setProperty(ClickHouseClientOption.CONNECTION_TIMEOUT.getKey(),
Integer.toString(60000));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL.getKey(), Boolean.toString(false));
clickHouseProperties.setProperty(ClickHouseClientOption.SSL_MODE.getKey(), "none");
```

3.5.5.2 建立连接

功能介绍

创建连接时使用ClickHouseDataSource配置连接使用的url和属性。

然后使用clickhouse-example.properties配置的user和password作为认证凭据，ClickHouse会带着用户名和密码在服务端进行安全认证。

样例代码

```
ClickHouseDataSource clickHouseDataSource =new ClickHouseDataSource(JDBC_PREFIX +
serverList.get(tries - 1), clickHouseProperties);
connection = clickHouseDataSource.getConnection(user, password);
```

说明

认证用的密码直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。

3.5.5.3 创建库

功能介绍

如下示例中通过on cluster语句在集群的所有Server节点创建数据库。

其中数据库名定义在clickhouse-example.properties文件的databaseName字段。

样例代码

```
private void createDatabase(String databaseName, String clusterName) throws Exception  {
        String createDbSql = "create database if not exists " + databaseName + " on cluster " + clusterName;
        util.exeSql(createDbSql);
}
```

3.5.5.4 创建表

功能介绍

如下示例中通过on cluster语句在集群的所有Server节点创建分布式表和本地表。

createSql为本地表，createDisSql为基于本地表的分布式表。

样例代码

```
private void createTable(String databaseName, String tableName, String clusterName) throws Exception {
    String createSql = "create table " + databaseName + "." + tableName + " on cluster " + clusterName
            + " (name String, age UInt8, date Date)engine=ReplicatedMergeTree('/clickhouse/tables/{shard}/" +
databaseName
            + "." + tableName + "'," + "'{replica}') partition by toYYYYMM(date) order by age";
    String createDisSql = "create table " + databaseName + "." + tableName + "_all" + " on cluster " +
clusterName + " as "
            + databaseName + "." + tableName + " ENGINE = Distributed(default_cluster," + databaseName +
"," + tableName + ", rand());";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(createSql);
    sqlList.add(createDisSql);
    util.exeSql(sqlList);
}
```

3.5.5.5 插入数据

功能介绍

如下示例代码通过循环batchNum次，构造示例数据并通过PreparedStatement的executeBatch()方法批量插入数据。

其中数据类型为创建的表所指定的三个字段，分别是String、UInt8和Date类型。

样例代码

```
String insertSql = "insert into " + databaseName + "." + tableName + " values (?,?,?)";
PreparedStatement preparedStatement = connection.prepareStatement(insertSql);
long allBatchBegin = System.currentTimeMillis();
```

```
for (int j = 0; j < batchNum; j++) {
    for (int i = 0; i < batchRows; i++) {
        preparedStatement.setString(1, "xxx_" + (i + j * 10));
        preparedStatement.setInt(2, ((int) (Math.random() * 100)));
        preparedStatement.setDate(3, generateRandomDate("2018-01-01", "2021-12-31"));
        preparedStatement.addBatch();
    }
    long begin = System.currentTimeMillis();
    preparedStatement.executeBatch();
    long end = System.currentTimeMillis();
    log.info("Inert batch time is {} ms", end - begin);
}
long allBatchEnd = System.currentTimeMillis();
log.info("Inert all batch time is {} ms", allBatchEnd - allBatchBegin);
```

3.5.5.6 查询数据

功能介绍

查询语句1：querySql1查询创建表创建的tableName表中任意10条数据；

查询语句2：querySql2通过内置函数对创建表创建的tableName表中的日期字段取年月后进行聚合。

样例代码

```
private void queryData(String databaseName, String tableName) throws Exception {
    String querySql1 = "select * from " + databaseName + "." + tableName + "_all" + " order by age limit 
10";
    String querySql2 = "select toYYYYMM(date),count(1) from " + databaseName + "." + tableName + "_all" 
+ " group by toYYYYMM(date) order by count(1) DESC limit 10";
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(querySql1);
    sqlList.add(querySql2);
    ArrayList<ArrayList<ArrayList<String>>> result = util.exeSql(sqlList);
    for (ArrayList<ArrayList<String>> singleResult : result) {
        for (ArrayList<String> strings : singleResult) {
            StringBuilder stringBuilder = new StringBuilder();
            for (String string : strings) {
                stringBuilder.append(string).append("\t");
            }
            log.info(stringBuilder.toString());
        }
    }
 }
```

3.5.5.7 删除表

功能介绍

删除在创建表中创建的副本表和分布式表。

语句1：使用drop table将集群中的本地表删除。

语句2：使用drop table将集群中的分布式表删除。

样例代码

```
private void dropTable(String databaseName, String tableName, String clusterName) throws Exception {
    String dropLocalTableSql = "drop table if exists " + databaseName + "." + tableName + " on cluster " + 
clusterName;
    String dropDisTableSql = "drop table if exists " + databaseName + "." + tableName + "_all" + " on 
```

```
cluster " + clusterName;
    ArrayList<String> sqlList = new ArrayList<String>();
    sqlList.add(dropLocalTableSql);
    sqlList.add(dropDisTableSql);
    util.exeSql(sqlList);
}
```

3.6 调测程序

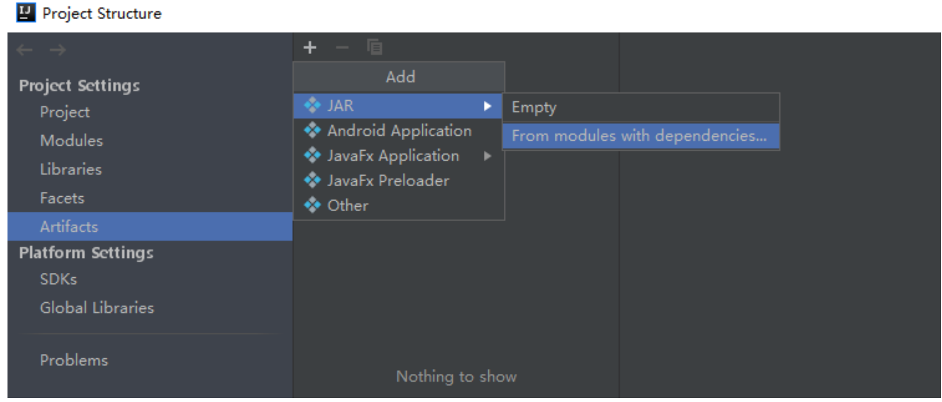
ClickHouse应用程序支持在Linux环境中运行。在程序代码完成开发后，您可以上传Jar包至准备好的Linux运行环境中运行。该环境需要和clickhouse集群处于同一vpc和安全组，以保证网络连通。

前提条件

Linux环境已安装JDK，版本号需要和IntelliJ IDEA导出Jar包使用的JDK版本一致，并设置好Java环境变量。

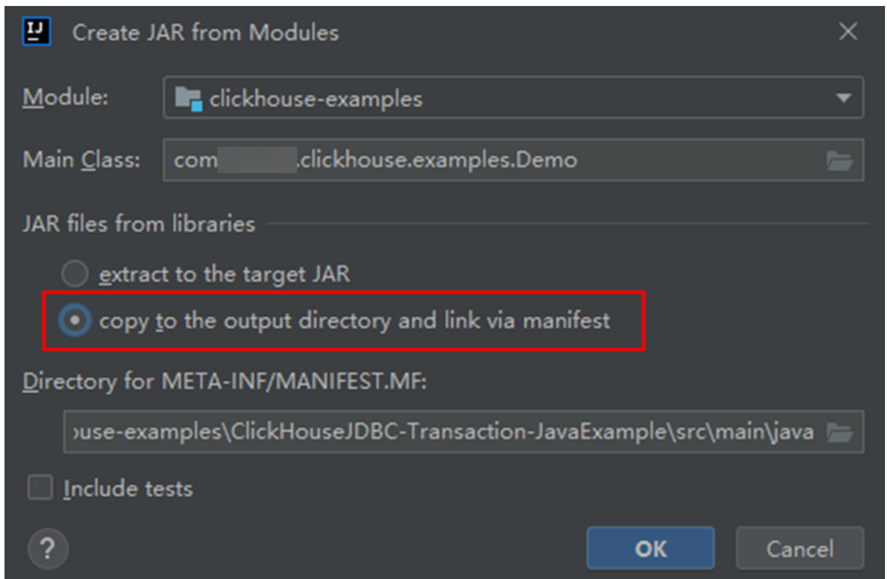
编译并运行程序

1. 导出jar包。
   a. 进入IntelliJ IDEA，选择“File > Project Structure > Artifacts”。
   b. 单击“加号”，选择“JAR > From modules with dependencies”。

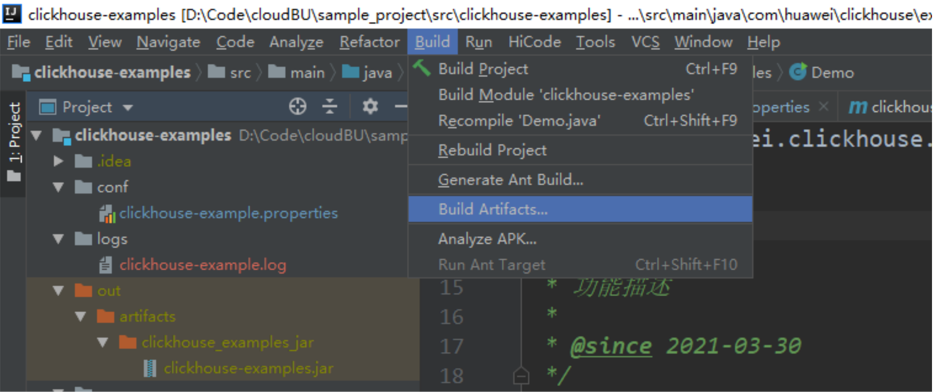


   c. “Main Class”选择“com.xxx.clickhouse.examples.Demo”，单击OK。

版权所有 © 华为云计算技术有限公司



d. 选择“Build> Build Artifacts”。编译成功后在“clickhouse-examples\out\artifacts\clickhouse_examples_jar”目录下查看并获取当前目录的所有jar文件。



2. 将“clickhouse-examples\out\artifacts\clickhouse_examples.jar”目录下的所有jar文件和“clickhouse-examples”目录下的“conf”文件夹拷贝到ECS的同一目录下。

3. 登录客户端节点，进入jar文件上传目录下，修改文件权限为700。

   chmod -R 700 clickhouse-examples.jar

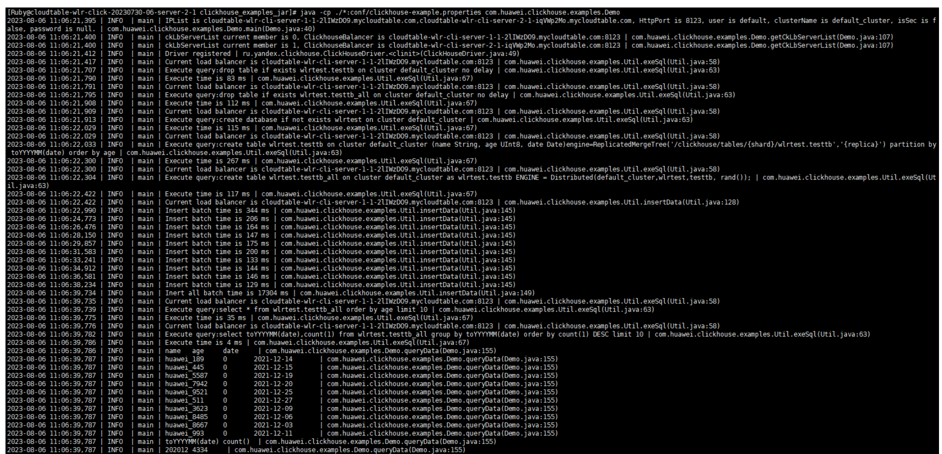
4. 在“clickhouse_examples.jar”所在客户端目录下执行如下命令运行jar包：

   java -cp ./*:conf/clickhouse-example.properties
   com.xxx.clickhouse.examples.Demo

查看调测结果

运行结果中没有异常或失败信息即表明运行成功。

图 3-3 运行日志图



3.7 开发 ClickHouse 冷热分离应用

3.7.1 应用背景

CloudTable ClickHouse支持冷热数据分离特性。通过该特性，您可以将冷热数据分别存储在不同类型的存储介质中，以降低存储成本。

- Hot（热数据）：访问、更新频率较高，未来被调用的概率较高的数据，对访问的响应时间要求很高的数据。
- Cold（冷数据）：不允许更新或更新频率比较低，访问频率比较低，对访问的响应时间要求不高的数据。

3.7.2 典型场景说明

通过典型场景，我们可以快速学习和掌握ClickHouse冷热分离的开发过程，并且对关键的接口函数有所了解。

场景说明

假定用户开发一个网站系统，test_tbl用于实时用户访问网站的记录，记录数据如下表：

表 3-17 原始数据

timestamp	type	error_code	error_msg	op_id	op_time
2024-06-04 10:36:00	1	404	Resource Not Found	998756	2024-06-04 11:36:00

timestamp	type	error_code	error_msg	op_id	op_time
2024-06-04 10:35:00	1	404	Resource Not Found	998756	2024-06-04 11:35:00
2024-06-03 10:33:00	1	404	Resource Not Found	998756	2024-06-03 11:33:00
2024-03-27 09:10:00	1	200	ok	998756	2024-03-27 10:10:00
2024-03-25 11:08:00	1	404	Resource Not Found	998756	2024-03-25 12:08:00

数据规划

当天整点写入数据，同时一天前数据查询频率较低，节省存储空间设置冷热分离，将一天前数据自动归档到冷存储。

3.7.3 开发思路

功能分解

根据**典型场景说明**中的业务进行功能分解，需要开发的功能点如下：

表 3-18 冷热分离功能

步骤	代码实现
步骤1：创建ClickHouse冷热分离表。	请参见**创建ClickHouse冷热分离数据表**。
步骤2：插入数据。	请参见**插入验证数据**。
步骤3：查询插入的数据。	请参见**查询插入数据**。

3.7.4 样例代码

此章节主要介绍CloudTable ClickHouse冷热分离的使用命令，以及冷数据如何自动存储到obs桶中。

版权所有 © 华为云计算技术有限公司

样例代码

- 创建ClickHouse冷热分离表test_table。

```
CREATE TABLE IF NOT EXISTS test_table
(

`timestamp` DATETIME NOT NULL COMMENT '日志时间',
    `type` INT NOT NULL COMMENT '日志类型',
    `error_code` INT COMMENT '错误码',
    `error_msg` VARCHAR(1024) COMMENT '错误详细信息',
    `op_id` BIGINT COMMENT '负责人id',
    `op_time` DATETIME COMMENT '处理时间'

)
ENGINE = MergeTree()
PARTITION BY timestamp
ORDER BY timestamp

TTL timestamp + INTERVAL 1 DAY TO DISK 'cold_disk'
SETTINGS storage_policy = 'hot_to_cold';
```

- 执行以下命令插入验证数据：

```
insert into test_table values('2024-06-04 10:36:00','1','404','Resource Not Found','998756','2024-06-04 11:36:00');  -- hot data
insert into test_table values('2024-06-04 10:35:00','1','404','Resource Not Found','998756','2024-06-04 11:35:00');  -- hot data
insert into test_table values('2024-06-03 10:33:00','1','404','Resource Not Found','998756','2024-06-03 11:33:00');  -- cold data
insert into test_table values('2024-03-27 09:10:00','1','200','ok','998756','2024-03-27 10:10:00');  -- cold data
insert into test_table values('2024-03-25 11:08:00','1','404','Resource Not Found','998756','2024-03-25 12:08:00');  -- cold data
```

- 查询插入的数据。

  查询数据。

```
select * from test_table FORMAT CSV;
```

  查询数据表分区存储的分区字段名、分区和存储路径。

```
SELECT name,partition,active,path FROM system.parts WHERE database = 'default' and table = 'test_table' and active = 1;
```

图 3-4 查询数据

```
cloudtable-ck-linbo-01-server-1-1 :) select * from test_table FORMAT CSV;
                                     SELECT name,partition,active,path FROM system.parts WHERE database = 'default' and table = 'test_table' and active = 1;

SELECT *
FROM test_table
FORMAT CSV

Query id: c5c9e53d-e674-439f-8343-0c95e3ef85ff

"2024-06-04 10:35:00",1,404,"Resource Not Found",998756,"2024-06-04 11:35:00"
"2024-06-04 10:36:00",1,404,"Resource Not Found",998756,"2024-06-04 11:36:00"
"2024-03-25 11:08:00",1,404,"Resource Not Found",998756,"2024-03-25 12:08:00"
"2024-03-27 09:10:00",1,200,"ok",998756,"2024-03-27 10:10:00"
"2024-06-03 10:33:00",1,404,"Resource Not Found",998756,"2024-06-03 11:33:00"

5 rows in set. Elapsed: 0.073 sec.

SELECT
    name,
    partition,
    active,
    path
FROM system.parts
WHERE (database = 'default') AND (table = 'test_table') AND (active = 1)

Query id: 8ca33b74-e6a4-4272-9df4-0a2d1fa49c50

┌─name──────────────┬─partition───────────┬─active─┬─path
│ 1711336080_5_5_0  │ 2024-03-25 11:08:00 │      1 │ /srv/clickhouse/data1/metadata/disks/cold_disk/store/464/46415997-b014-4bca-9130-bf4856f7bab2/1711336080_5_5_0/
│ 1711501800_4_4_0  │ 2024-03-27 09:10:00 │      1 │ /srv/clickhouse/data1/metadata/disks/cold_disk/store/464/46415997-b014-4bca-9130-bf4856f7bab2/1711501800_4_4_0/
│ 1717381980_3_3_0  │ 2024-06-03 10:33:00 │      1 │ /srv/clickhouse/data1/metadata/disks/cold_disk/store/464/46415997-b014-4bca-9130-bf4856f7bab2/1717381980_3_3_0/
│ 1717468500_2_2_0  │ 2024-06-04 10:35:00 │      1 │ /srv/clickhouse/data2/clickhouse/store/464/46415997-b014-4bca-9130-bf4856f7bab2/1717468500_2_2_0/
│ 1717468560_1_1_0  │ 2024-06-04 10:36:00 │      1 │ /srv/clickhouse/data1/clickhouse/store/464/46415997-b014-4bca-9130-bf4856f7bab2/1717468560_1_1_0/

5 rows in set. Elapsed: 0.002 sec.
```

当前系统时间为2024年6月4日22点，test_table表timestamp列超过一天的数据存储到了名为cold_disk的OBS下。