表格存储服务

开发指南

文档版本 01

发布日期 2025-08-08

华为云计算技术有限公司

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。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 HBase 应用开发指导

1.1 HBase 应用开发流程

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。

图 1-1 应用程序开发流程

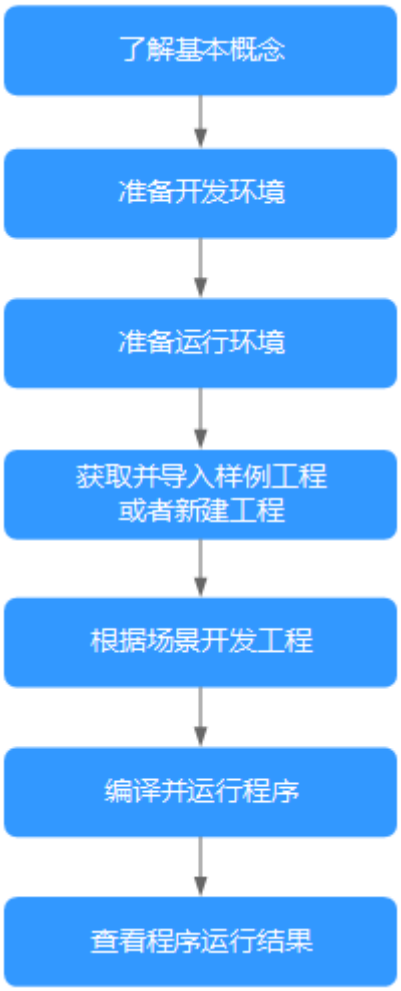


表 1-1 应用开发的流程说明

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。	**准备本地应用开发环境**
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HBase应用运行环境**
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HBase样例工程** **导入并配置HBase样例工程**
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 HBase 应用开发环境

1.2.1 准备本地应用开发环境

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 HBase 应用运行环境

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。

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 HBase 样例工程

前提条件

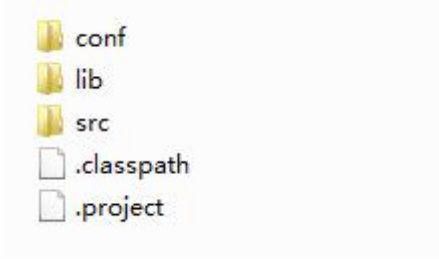
确认表格存储服务已经安装，并正常运行。

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 导入并配置 HBase 样例工程

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。

前提条件

运行环境已经正确配置，请参见**准备HBase应用运行环境**。

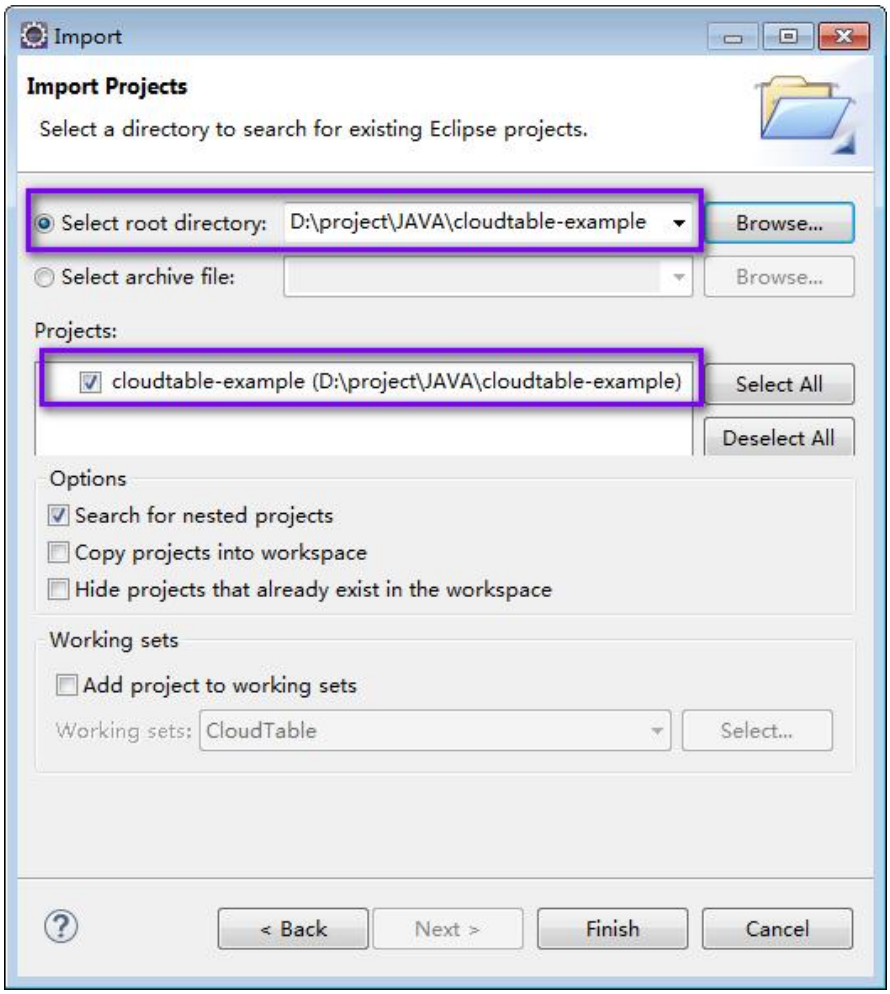
操作步骤

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HBase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”。

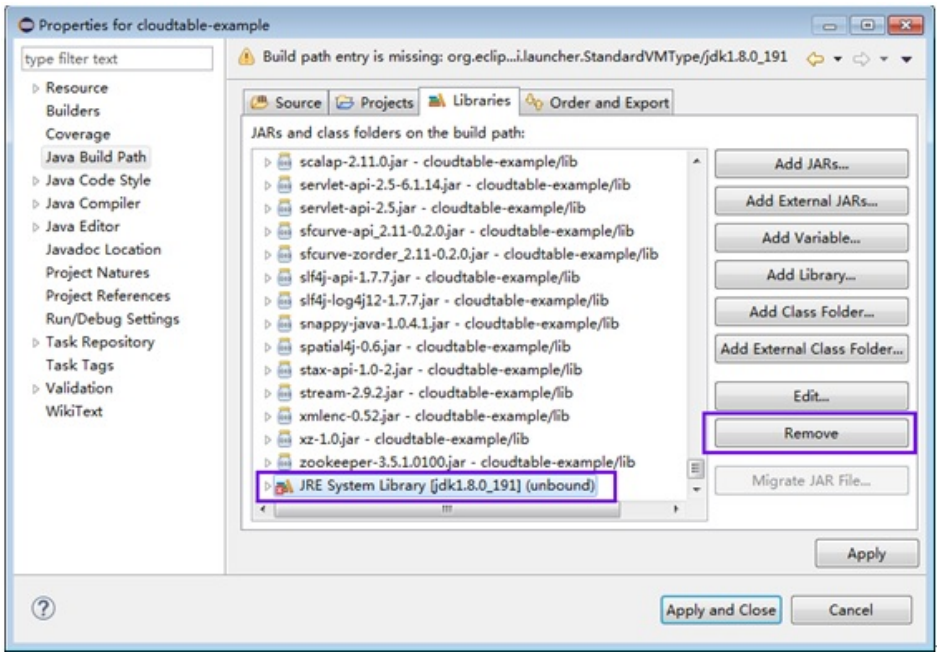
图 1-3 导入样例工程



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。

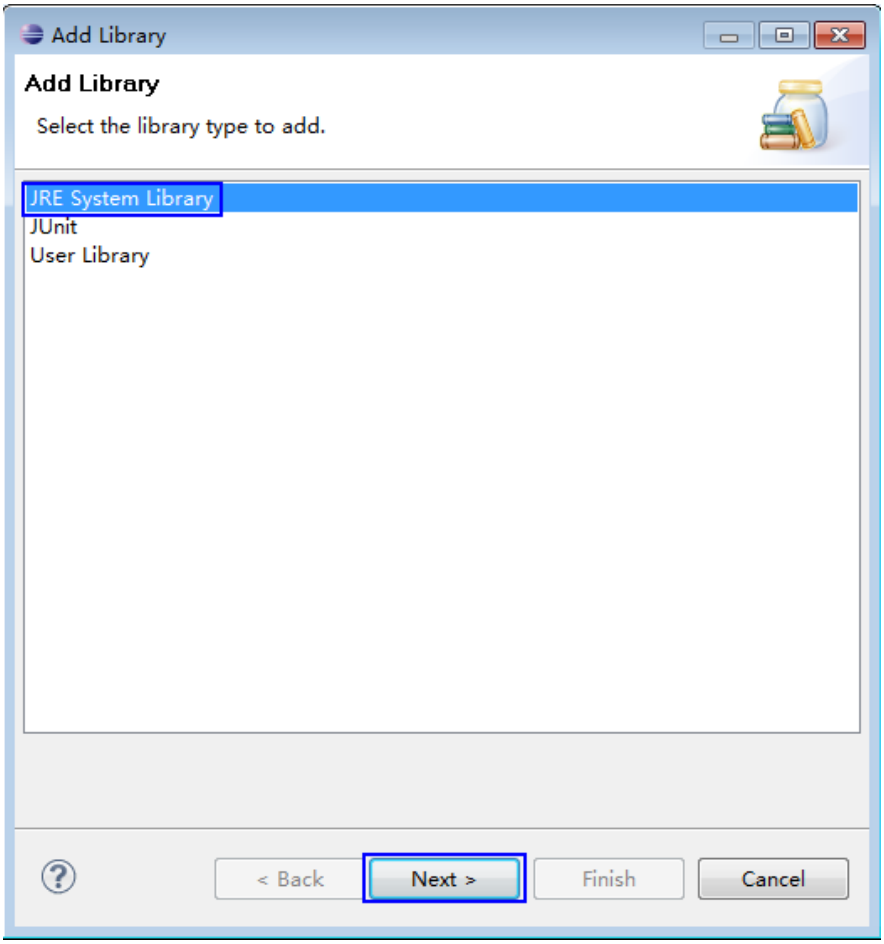
1. 在左边导航上选择“Java Build Path”，单击右侧“Libraries”标签页，按**图1-4**所示将报错的JDK选中后，单击“Remove”删除。

图 1-4 删除报错的 JDK



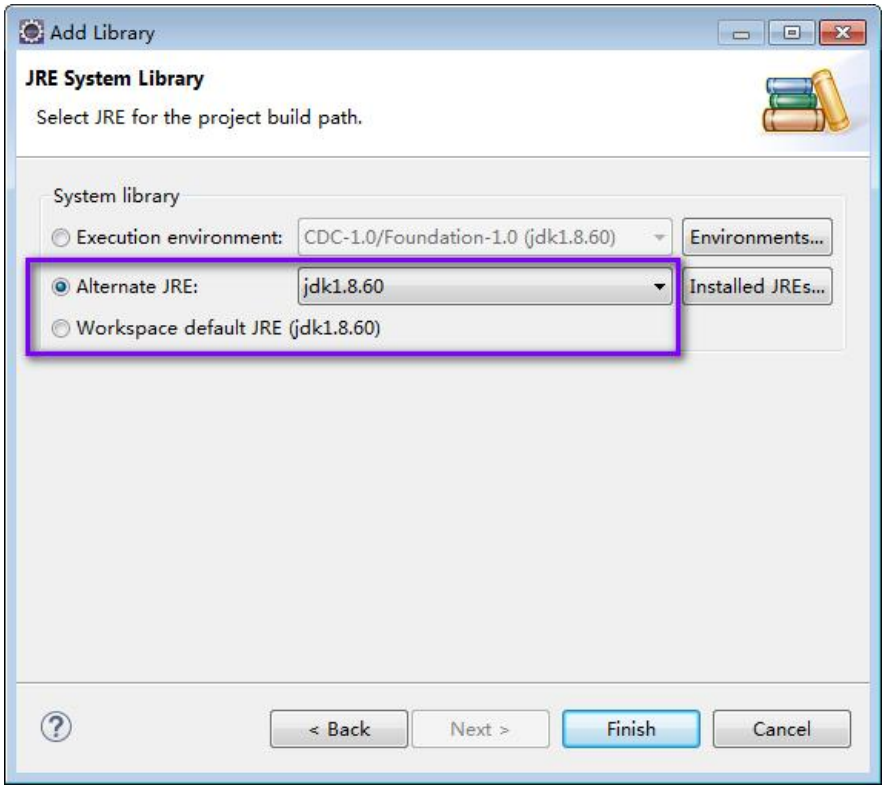
2. 单击“Add Library...”按钮，如图1-5所示，在弹出的窗口中选择“JRE System Library”。

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。

图 1-6 选择 JRE

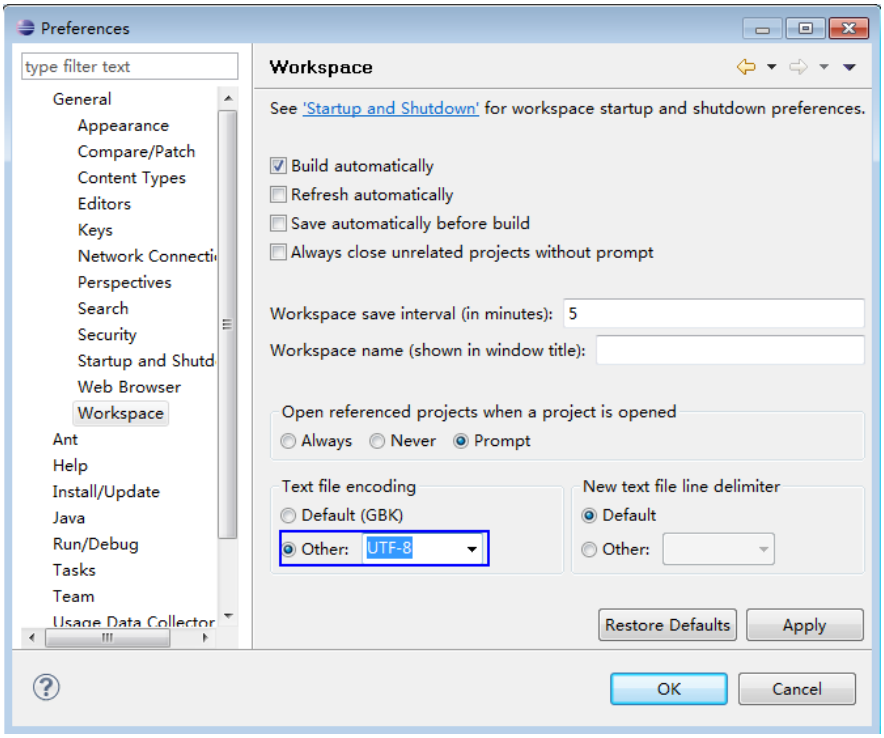


4. 单击“Finish”关闭窗口完成设置。

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。

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 HBase 数据读写样例工程

1.3.1.1 HBase 数据读写样例程序典型场景

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。
- 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	IPA, IPB
12005000202	B	Female	23	IPC, IPD
12005000203	C	Male	26	IPE, IPF
12005000204	D	Male	18	IPG, IPH
12005000205	E	Female	21	IPI, IPJ
12005000206	F	Male	32	IPK, IPL
12005000207	G	Female	29	IPM, IPN
12005000208	H	Female	30	IPO, IPP
12005000209	I	Male	26	IPQ, IPR
12005000210	J	Male	25	IPS, IPT

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。

1.3.1.2 HBase 数据读写样例程序开发思路

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。

表 1-4 在 HBase 中开发的功能

序号	步骤	代码实现
1	根据**HBase数据读写样例程序典型场景**中的信息创建表。	请参见**创建HBase表**。
2	导入用户数据。	请参见**插入HBase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HBase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HBase数据**。
5	根据用户姓名进行查询。	请参见**使用HBase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HBase数据**。
7	A业务结束后，删除用户信息表。	请参见**删除HBase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。

1.3.1.3 配置 HBase ZooKeeper 地址

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）”。

1.3.1.4 初始化 HBase 配置

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.huawei.cloudtable.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[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。

1.3.1.5 创建 HBase 客户端连接

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);
}
```

1.3.1.6 创建 HBase 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);
```

1.3.1.7 删除 HBase 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。

1.3.1.8 修改 HBase 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.");
}
```

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。

1.3.1.9 插入 HBase 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("IPA, IPB"));
    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("IPC, IPD"));
    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("IPE, IPF"));
    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("IPG, IPH"));
    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("IPI, IPJ"));
    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("IPK, IPL"));
    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("IPM, IPN"));
```

```
    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("IPO, IPP"));
    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("IPQ, IPR"));
    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("IPS, IPT"));
    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。

1.3.1.10 删除 HBase 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.");
}
```

1.3.1.11 使用 Get 读取 HBase 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)));
  }
  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.");
}
```

1.3.1.12 使用 Scan 读取 HBase 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。

1.3.1.13 使用 HBase 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();
      } catch (IOException e) {
        LOG.error("Close table failed " ,e);
      }
    }
  }
```

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

1.3.2 HBase 冷热分离样例工程

1.3.2.1 HBase 冷热分离数据样例程序典型场景

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。

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
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

城市	区域	时间	温度	湿度
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
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

城市	区域	时间	温度	湿度
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。

1.3.2.2 HBase 冷热分离数据样例程序开发思路

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**HBase冷热分离数据样例程序典型场景**中的信息创建表。	请参见**创建HBase冷热分离数据表**。
2	写入数据。	请参见**插入HBase冷热分离数据**。
4	根据城市、区域、时间查询温度和湿度。	请参见**使用Get读取HBase冷热分离数据**。
5	根据城市、局域、时间范围进行查询。	请参见**使用Scan读取HBase冷热分离数据**。

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。

1.3.2.3 配置 HBase ZooKeeper 地址

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”。

1.3.2.4 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]”。

1.3.2.5 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);
}
```

1.3.2.6 创建 HBase 冷热分离数据表

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)

  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)”。

1.3.2.7 删除 HBase 冷热分离数据表

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]
    }
    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]”。

1.3.2.8 修改 HBase 冷热分离数据表

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);
    }
    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[1] 只有在调用disableTable接口后， 再调用modifyTable接口才能将表修改成功。再请调用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[1] 只有在调用disableTable接口后， 再调用modifyTable接口才能将表修改成功。再请调用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]”。

1.3.2.9 插入 HBase 冷热分离数据

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"));
    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"));
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);

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"));
  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.");
}
```

1.3.2.10 使用 Get 读取 HBase 冷热分离数据

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.");
}
```

1.3.2.11 使用 Scan 读取 HBase 冷热分离数据

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);
    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.");
}
```

1.3.3 配置 HBase 多语言访问

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。

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.thrift

----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");
        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");
    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)
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()
```

1.4 调测 HBase 应用

1.4.1 在 Windows 中调测程序

1.4.1.1 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。

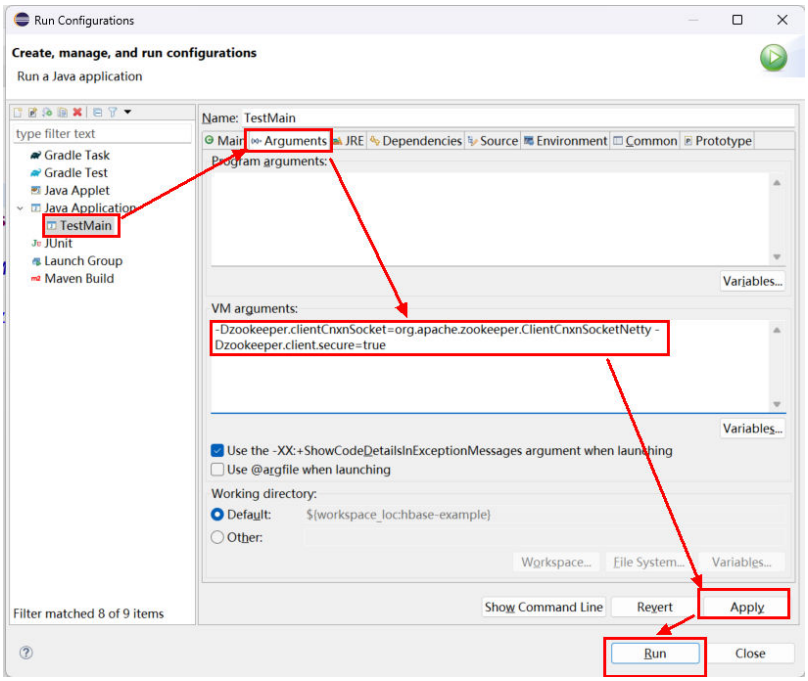
操作步骤

- **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。



1.4.1.2 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：

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
```

1.4.2 在 Linux 中调测程序

1.4.2.1 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

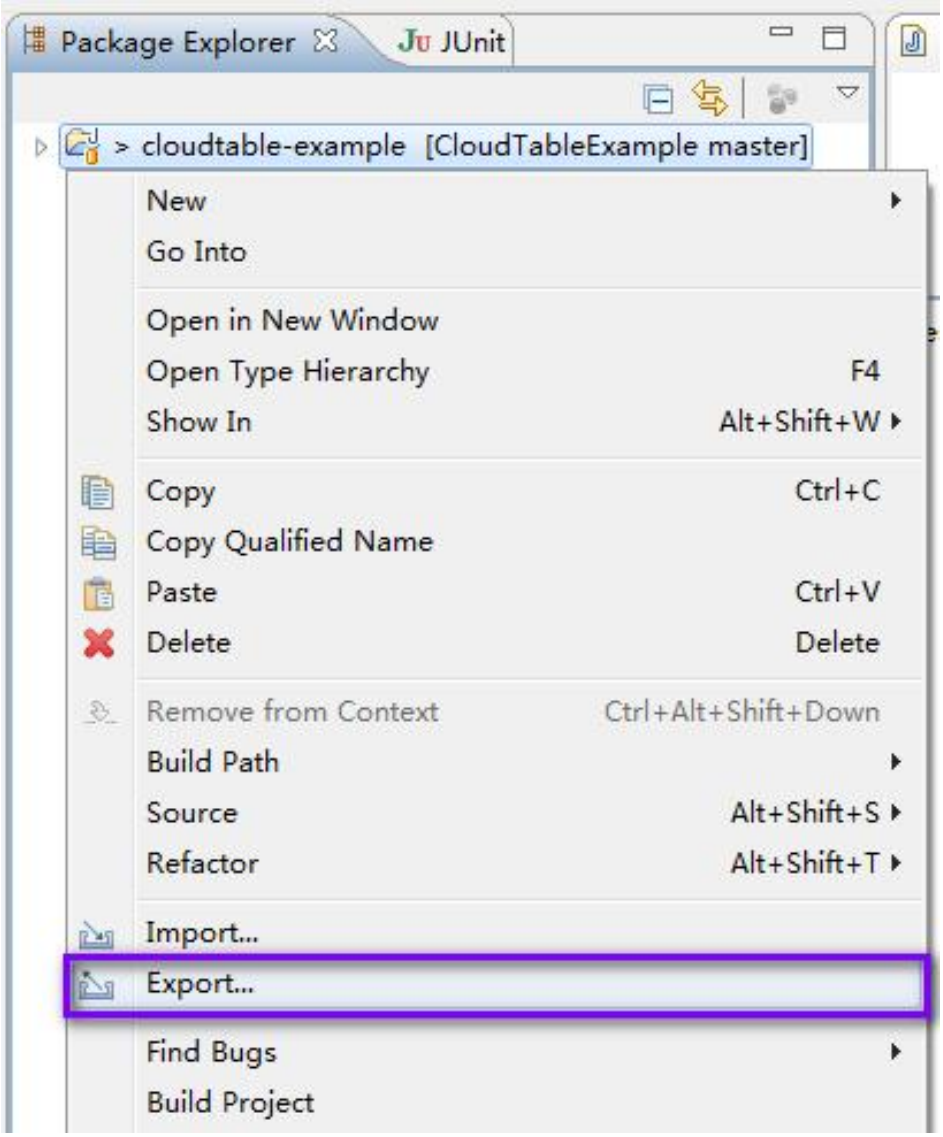
- 已安装HBase客户端。
- Linux环境已安装JDK，版本号需要和Eclipse导出Jar包使用的JDK版本一致。

操作步骤

步骤1 导出Jar包。

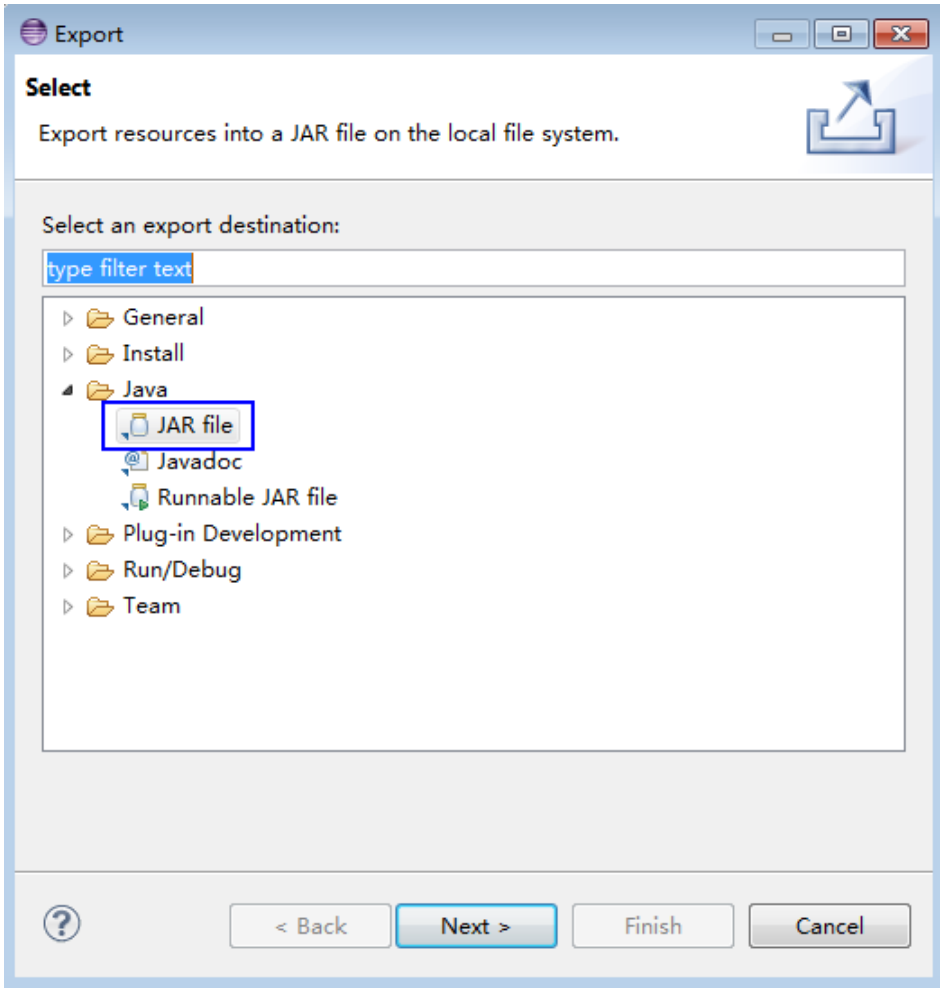
1. 右击样例工程，选择导出。

图 1-9 导出 Jar 包



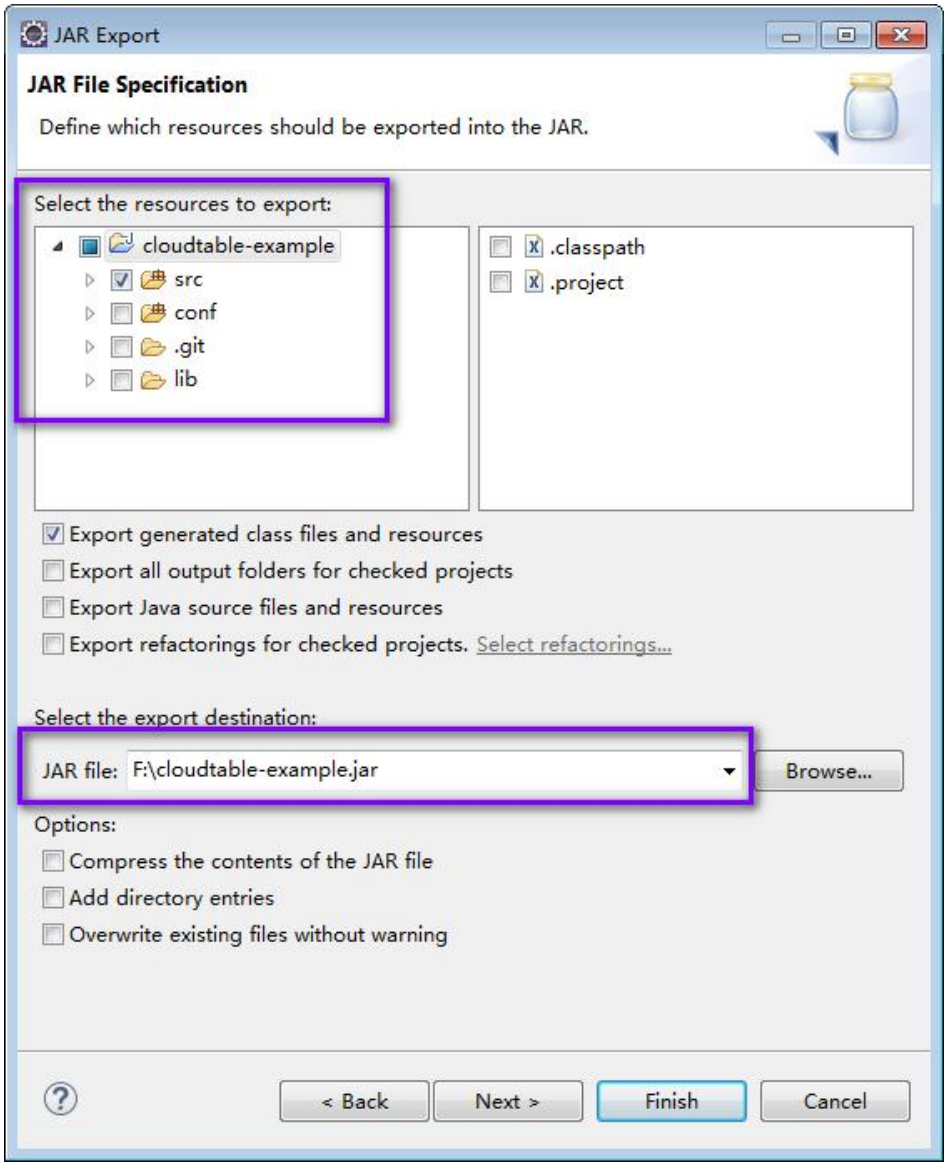
2. 选择JAR file，单击“Next”。

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”。

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

1.4.2.2 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

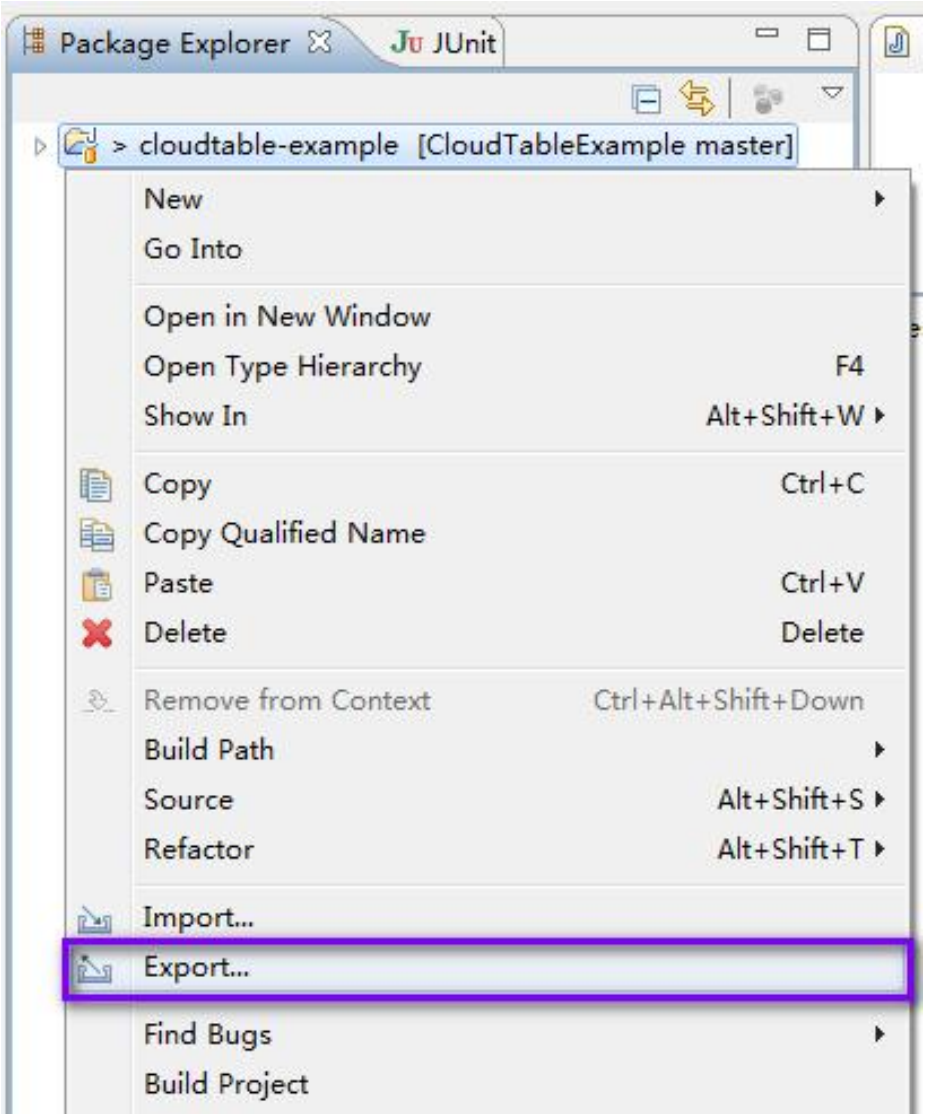
Linux环境已安装JDK，版本号需要和Eclipse导出Jar包使用的JDK版本一致。

操作步骤

步骤1 导出Jar包。

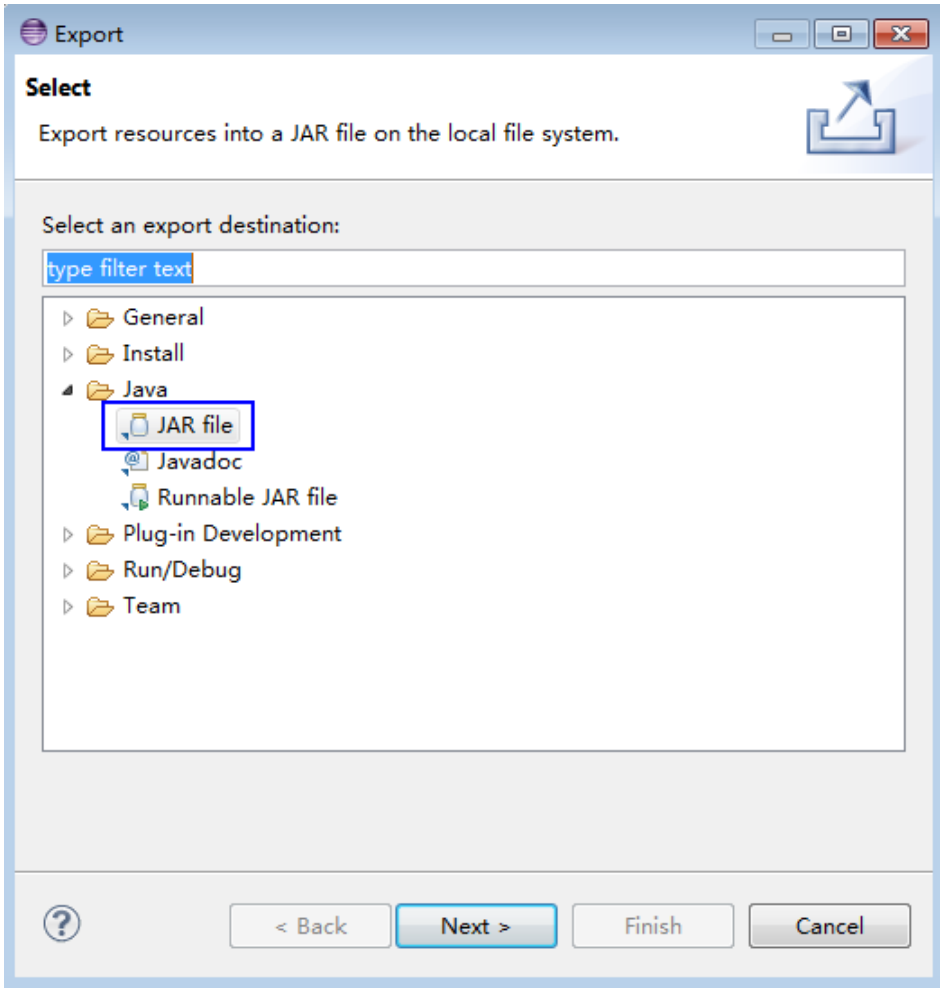
1. 右击样例工程，选择导出。

图 1-12 导出 Jar 包



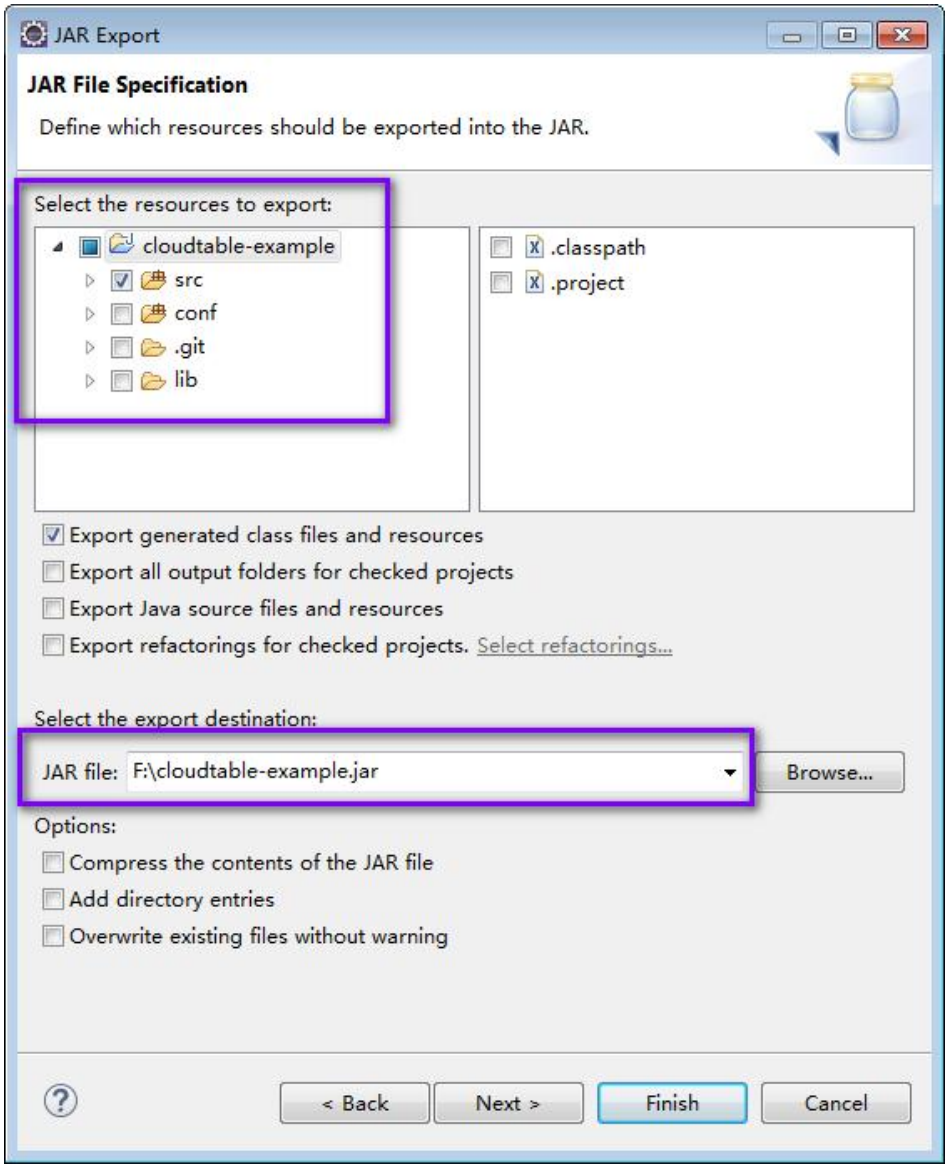
2. 选择JAR file，单击“Next”。

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"
```



```
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

1.4.2.3 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
```

2 Doris 应用开发指导

2.1 创建 Doris 连接

2.1.1 JDBC 通过非 ssl 方式连接 Doris 集群

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://" +
        "[FE1_host]:[FE1_port],[FE2_host]:[FE2_port],[FE3_host]:[FE3_port]/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();
            }
        }
    }
}
```

2.1.2 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：

   图 2-1 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();
        }
    }
}
```

2.1.3 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。

- 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();
```

```
    }
  }
}
```

2.2 Doris 组件使用规范

2.2.1 Doris 建表规范

该章节主要介绍Doris建表时需遵循的规范和建议。

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。
- 【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。
- 【建议】根据业务手动分桶，不使用AUTO自动分桶策略。
- 【建议】2.1及以上版本如果存在高频大量数据入库，建议使用MOR。如果优先考虑查询性能且可接受较长入库耗时，建议使用MOW。2.0及以下版本不建议使用MOW。

2.2.2 Doris 数据变更规范

该章节主要介绍Doris数据变更时需遵循的规范和建议。

数据变更类

- 【强制】应用程序不可以直接使用delete或者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。

2.2.3 Doris 命名规范

该章节主要介绍创建Doris数据库或表时，数据库名或表名需遵循的规则和建议。

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。

2.2.4 Doris 数据查询规范

该章节主要介绍Doris数据查询时需遵循的规范和建议。

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。

2.2.5 Doris 数据导入规范

该章节主要介绍Doris数据导入规范。

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。

2.2.6 Doris 分区规范

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节介绍。
- 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",
      `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 |
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。

- 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 |
+-------------+---------------+----------------+---------------------+--------+--------------
+-------------------------------------------------------------------------------+-----------------+---------
+---------------+--------------+--------------------+--------------------+-------------------------+----------
+------------+-------------------------+
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 |
+-------------+---------------+----------------+---------------------+--------+---------------
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------+-----------------+---------+----------------+---------------+---------------------
+---------------------+--------------------------+----------+------------+-------------------------+
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
(
"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 |
+-------------+---------------+----------------+--------------------+--------+--------------
+-------------------------------------------------------------------------------------------------+-----------------
+---------+----------------+---------------+--------------------+--------------------+--------------------------
+----------+------------+-----------------------+
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。

```
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)
```

2.2.7 Doris 分桶规范

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。

2.2.8 分区分桶数量和数据量规范

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。

3 ClickHouse 应用开发指导

3.1 准备 ClickHouse 应用开发环境

3.1.1 准备 ClickHouse 应用开发和运行环境

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。

表 3-1 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。

准备项	说明
7-zip	用于解压“*.zip”和“*.rar”文件，支持7-Zip 16.04版本。

3.1.2 配置并导入 ClickHouse 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。

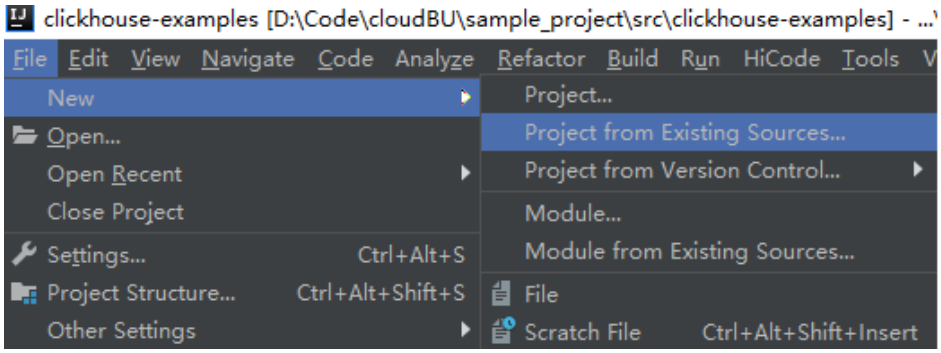
操作场景

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”。



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 3-2 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，获取集群访问地址。 多个地址使用逗号分隔，例如配置为“cloudtable-wlr-cli-server-1-1-***.com,cloudtable-wlr-cli-server-2-1-***.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。
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。
native_dataSource_ip_list	-	clickhouse节点的ip和tcp端口集合。

----结束

3.2 开发 ClickHouse 应用

3.2.1 ClickHouse 应用典型场景说明

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。

3.2.2 ClickHouse 应用开发思路

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。

3.2.3 设置 ClickHouse 连接属性

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");
```

3.2.4 建立 ClickHouse 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。

3.2.5 创建 ClickHouse 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);
}
```

3.2.6 创建 ClickHouse 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);
}
```

3.2.7 插入 ClickHouse 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);
```

3.2.8 查询 ClickHouse 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());
        }
    }
}
```

3.2.9 删除 ClickHouse 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);
}
```

3.3 调测 ClickHouse 应用

3.3.1 在 Linux 环境中调测 ClickHouse 应用

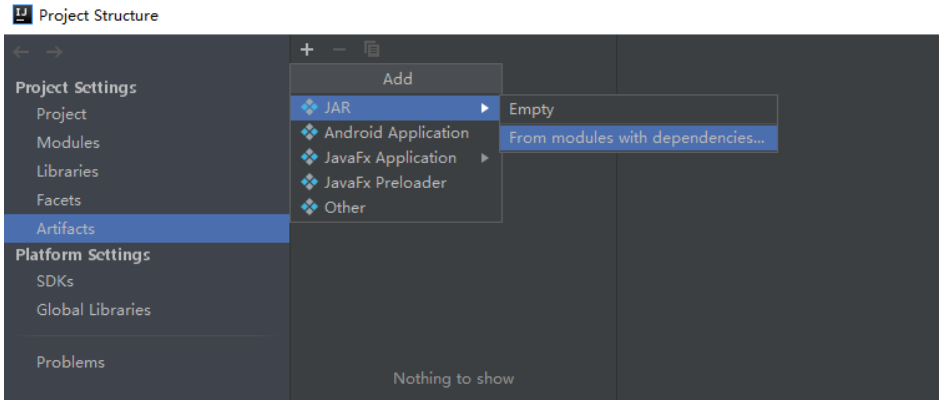
ClickHouse应用程序支持在Linux环境中运行。在程序代码完成开发后，您可以上传Jar包至准备好的Linux运行环境中运行。该环境需要和ClickHouse集群处于同一vpc和安全组，以保证网络连通。

前提条件

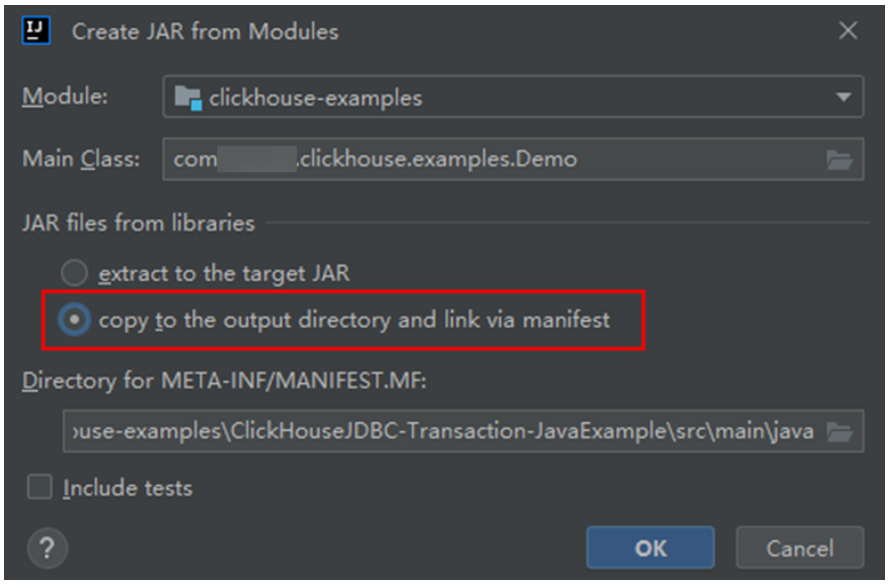
Linux环境已安装JDK，版本号需要和IntelliJ IDEA导出Jar包使用的JDK版本一致，并设置好Java环境变量。

编译并运行程序

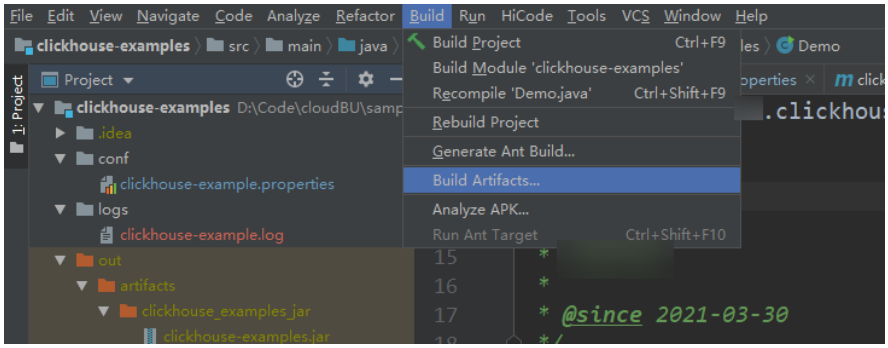
1. 导出jar包。
   a. 进入IntelliJ IDEA，选择“File > Project Structure > Artifacts”。
   b. 单击“加号”，选择“JAR > From modules with dependencies”。



   c. “Main Class”选择“com.xxx.clickhouse.examples.Demo”，单击OK。



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。

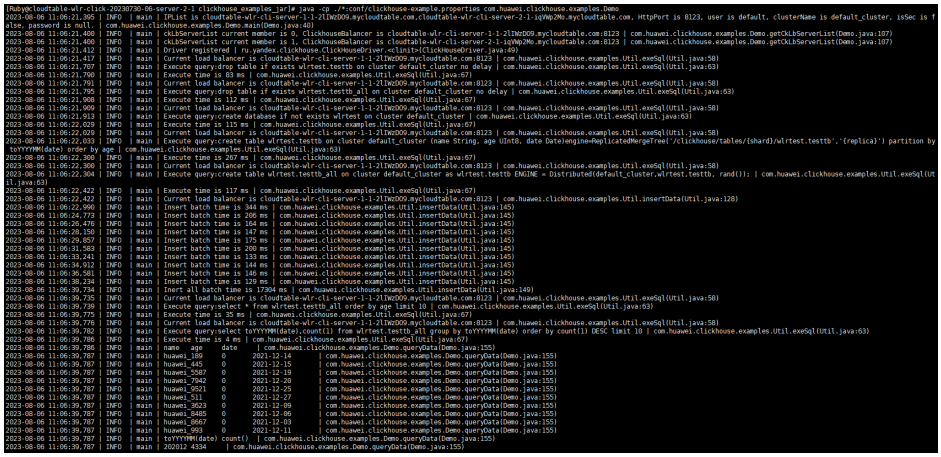
   chmod -R 700 clickhouse-examples.jar
4. 在“clickhouse_examples.jar”所在客户端目录下执行如下命令运行jar包：

   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 3-1 运行日志图



3.4 ClickHouse 组件使用规范

3.4.1 ClickHouse 建表规范

该章节主要介绍ClickHouse建表时需遵循的规范和建议。

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。

3.4.2 ClickHouse 数据写入规范

该章节主要介绍ClickHouse数据写入时需遵循的规范和建议。

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。

- 【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】ALTER TABLE修改表数据。
  - 建议慎用delete、update的mutation操作。

    标准SQL的更新、删除操作是同步的，即客户端要等服务端返回执行结果（通常是int值）；而ClickHouse的update、delete是通过异步方式实现的，当执行update语句时，服务端立即返回执行成功还是失败结果，但是实际上此时数据还没有修改完成，而是在后台排队等着进行真正的修改，可能会出现操作覆盖的情况，也无法保证操作的原子性。

    业务场景要求有update、delete等操作，建议使用ReplacingMergeTree、CollapsingMergeTree、VersionedCollapsingMergeTree引擎，使用方式参见**表引擎**。
  - 建议少或不增删数据列。

    业务提前规划列个数，如果将来有更多列要使用，可以规划预留多列，避免在生产系统跑业务过程中进行大量的alter table modify列操作，导致不可以预知的性能、数据一致性问题。

3.4.3 ClickHouse 数据查询规范

该章节主要介绍ClickHouse数据查询时需遵循的规范和建议。

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，大幅度提升查询性能。

- 【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。

3.4.4 ClickHouse 数据入库规范

该章节主要介绍ClickHouse数据入库时需遵循的规范和建议。

数据入库

【 建议 】不建议建ClickHouse kafka表引擎，进行数据同步到ClickHouse中，当前ClickHouse的kafka引擎有会导致kafka引擎数据入库产生性能等诸多问题，通过用户使用经验，需要应用侧自己写kafka的数据消费，攒批写入ClickHouse，提升ClickHouse的入库性能。