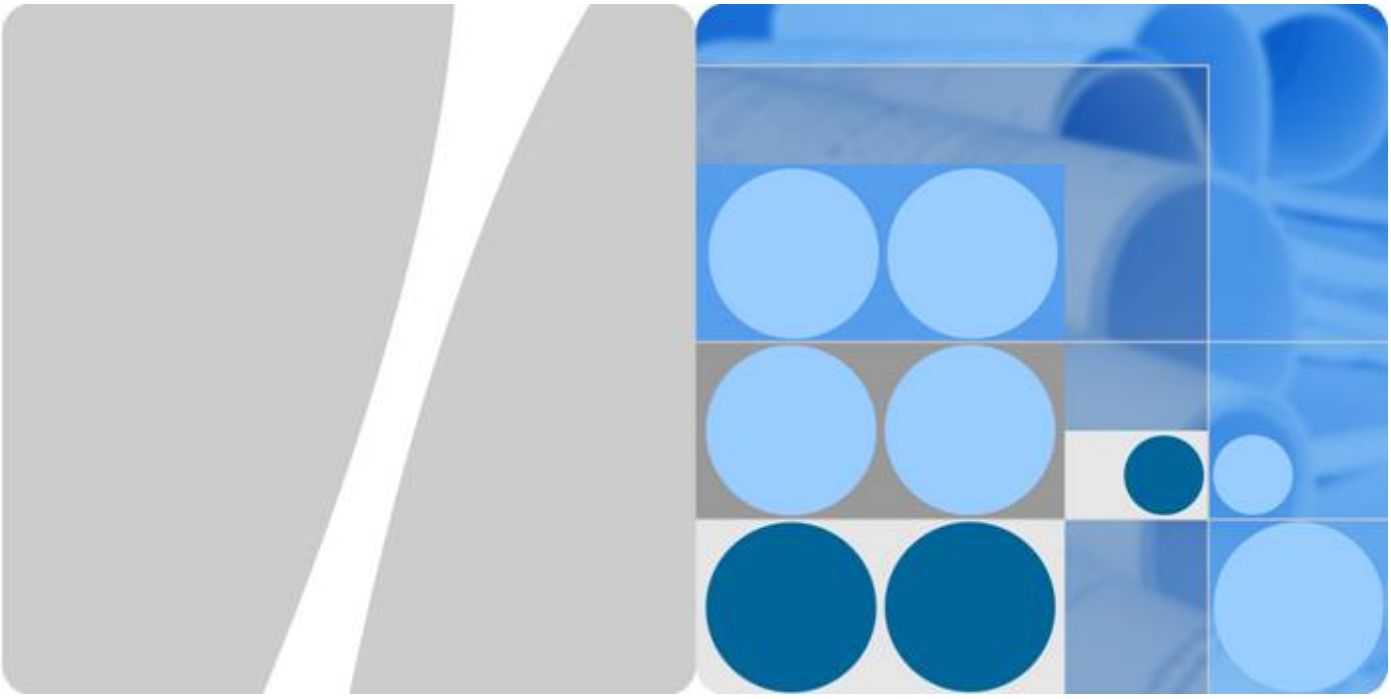


表格存储服务

开发指南

文档版本　06

发布日期　2019-04-04

华为技术有限公司

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，华为公司对本文档内容不做任何明示或默示的声明或保证。

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。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 开发流程

本文档主要介绍在CloudTable集群模式下如何调用HBase、OpenTSDB或GeoMesa的开源接口进行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

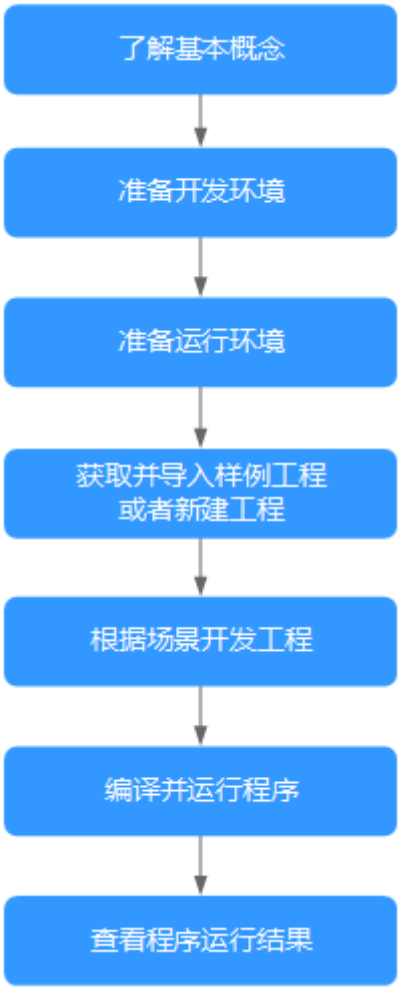


表 1-1 应用开发的流程说明

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

2 准备开发环境

2.1 开发环境简介

在进行二次开发时，要准备的开发环境如表2-1所示。

表 2-1 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，若使用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。

2.2 准备运行环境

2.2.1 准备 Windows 运行环境

操作场景

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

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

- 若使用IBM JDK，请确保Eclipse中的JDK配置为IBM JDK。
- 若使用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

2.3 下载样例工程

前提条件

确认表格存储服务已经安装，并正常运行。

下载样例工程（集群模式）

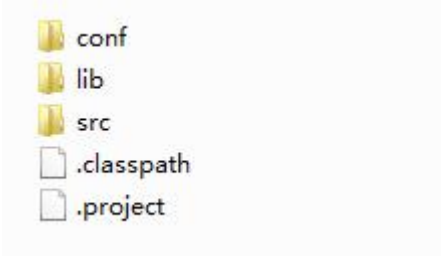
步骤1 下载样例代码工程。

登录表格存储服务管理控制台，单击“帮助”，在帮助页面右侧的“常用链接”区域下方单击“样例代码下载”，下载样例代码工程安装包。如**图2-1**所示。

图 2-1 样例代码下载链接

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2-2**所示。

图 2-2 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>huaweicloudsdk</id>
     <repositories>
        <repository>
           <id>huaweicloudsdk</id>
           <url>https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/</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>huaweicloudsdk</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>huaweicloudsdk</id>
        <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/</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>
```

2.4 配置并导入工程

背景信息

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

前提条件

运行环境已经正确配置，请参见**准备Windows运行环境**。

操作步骤

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

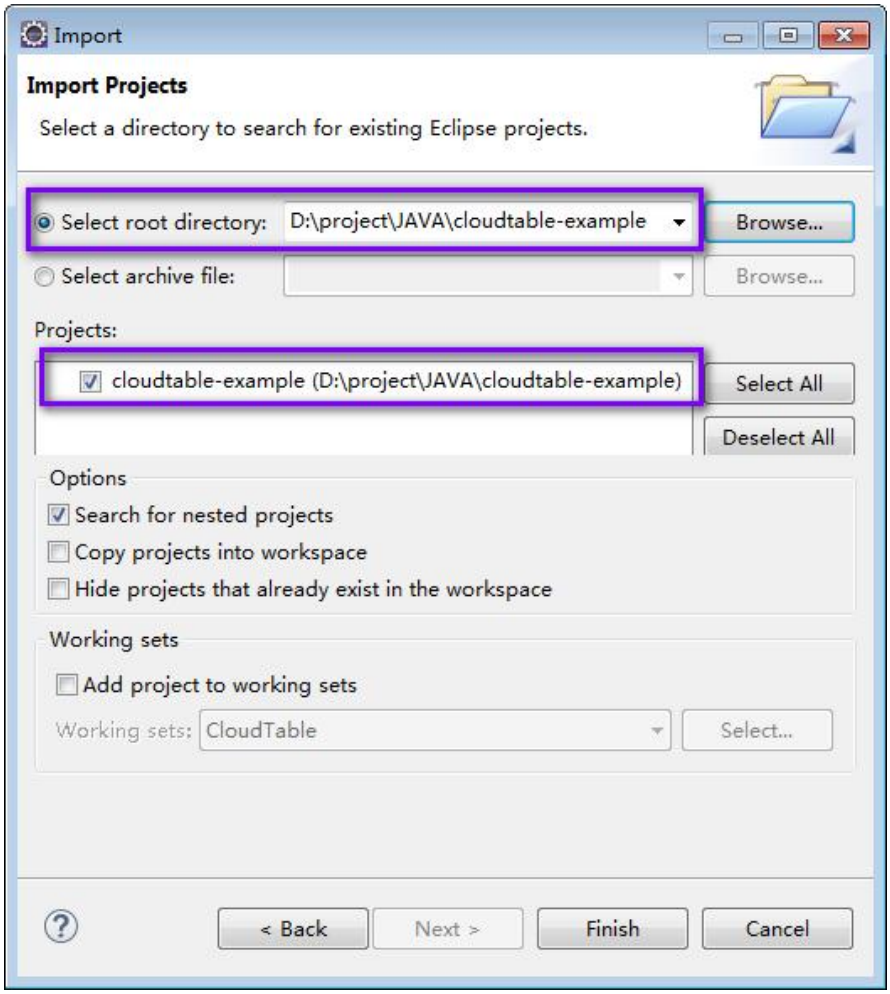
步骤2 在应用开发环境中，导入样例工程到Eclipse开发环境。

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

   显示“浏览文件夹”对话框。如**图2-3**所示。

2. 选择样例工程文件夹，单击“Finish”。

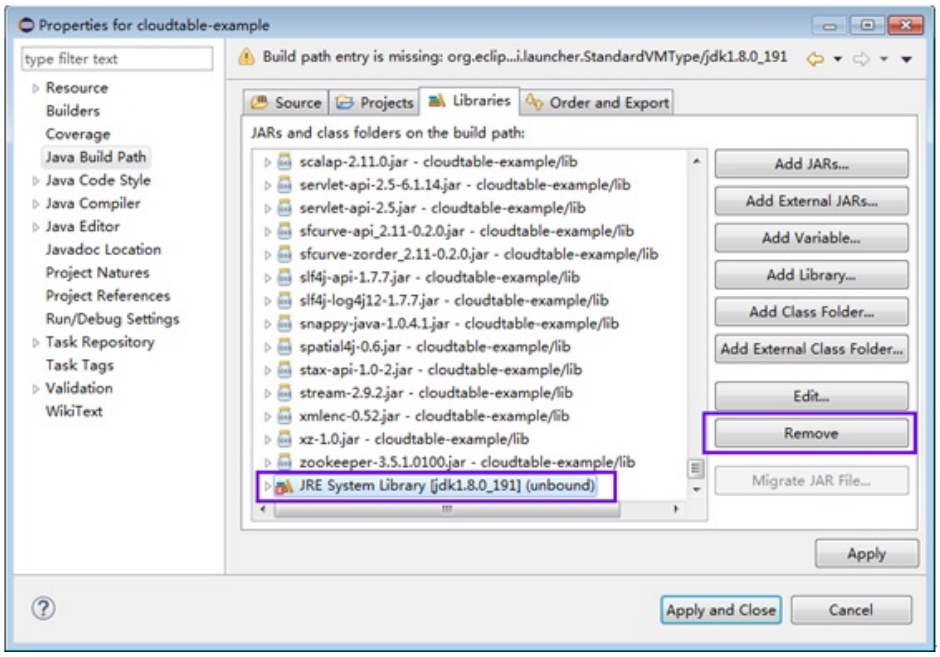
图 2-3 导入样例工程



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。

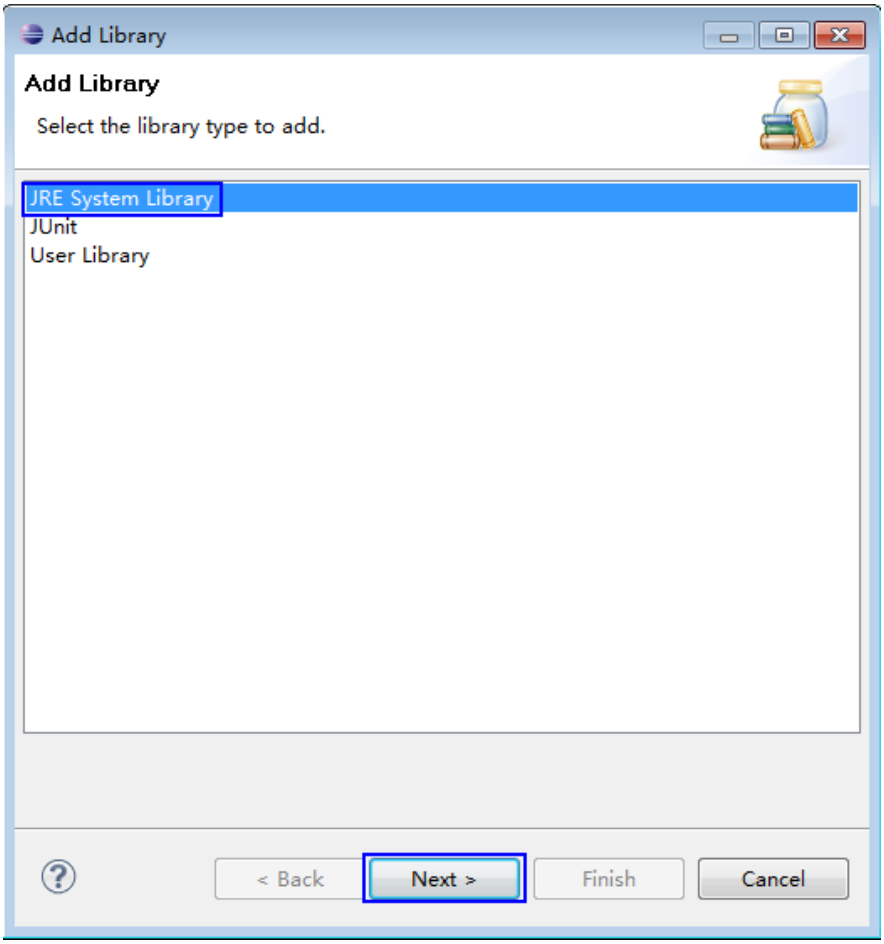
1. 在左边导航上选择“Java Build Path”，单击右侧“Libraries”标签页，按**图2-4**所示将报错的JDK选中后，单击“Remove”删除。

图 2-4 删除报错的 JDK



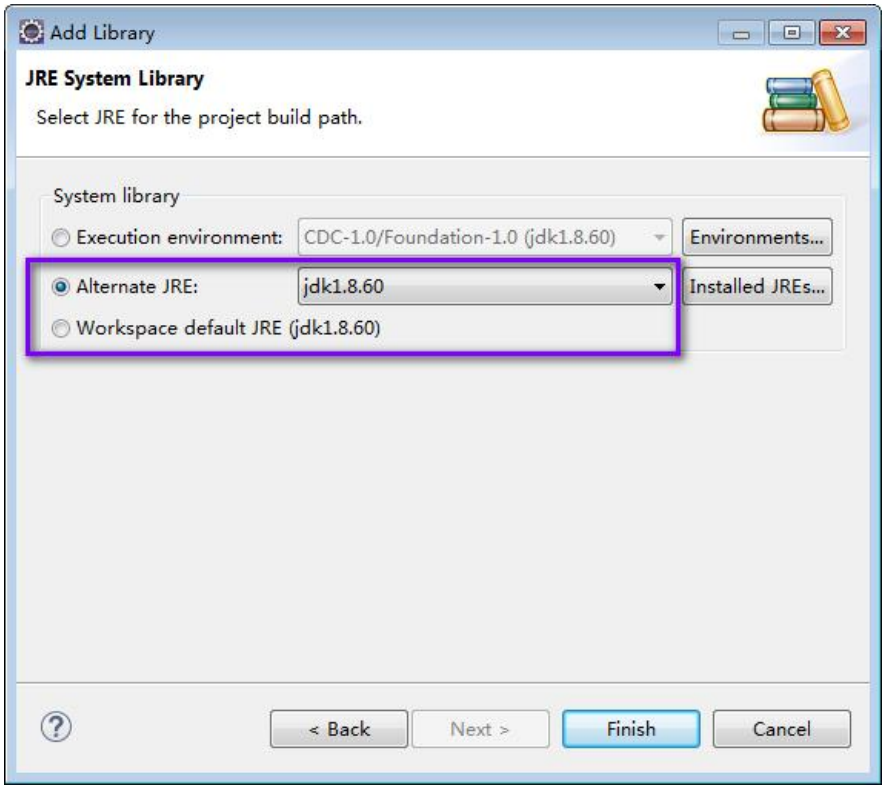
2. 单击“Add Library...”按钮，如图2-5所示，在弹出的窗口中选择“JRE System Library”。

图 2-5 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2-6所示，选中“Alternate JRE”后，选择JDK版本。

图 2-6 选择 JRE

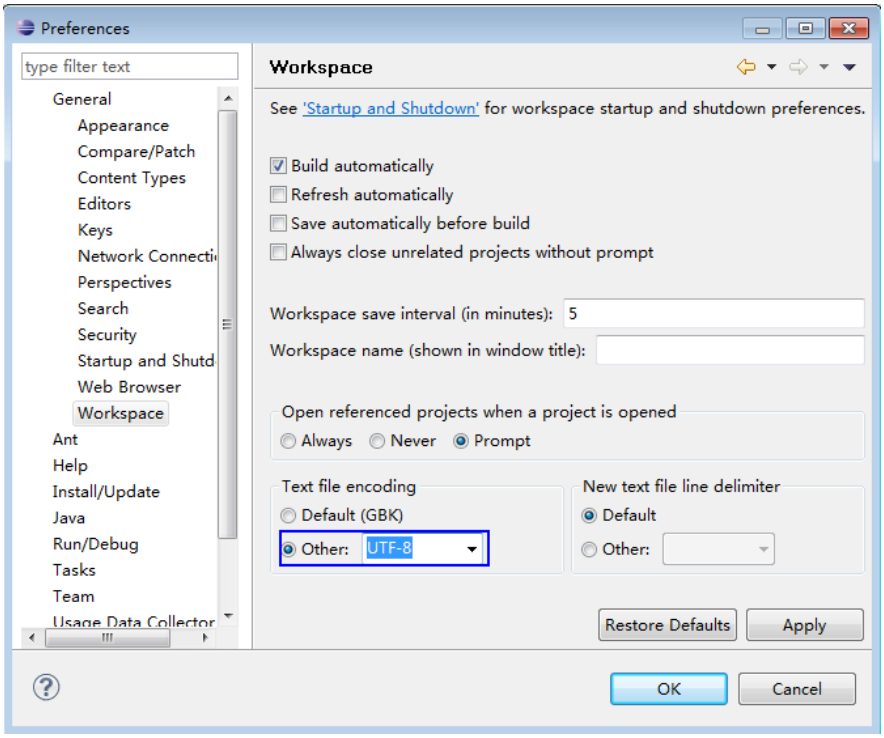


4. 单击“Finish”关闭窗口完成设置。

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2-7所示。

图 2-7 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“集群模式”，然后在集群列表中找到所需要的集群，并获取相应的“ZK链接地址”。

----结束

3 开发 HBase 应用

3.1 典型场景说明

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

场景说明

假定用户开发一个应用程序，用于管理企业中的使用A业务的用户信息，如表3-1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 3-1 用户信息

编号	姓名	性别	年龄	地址
12005000201	A	Male	19	Shenzhen, Guangdong
12005000202	B	Female	23	Shijiazhuang, Hebei
12005000203	C	Male	26	Ningbo, Zhejiang
12005000204	D	Male	18	Xiangyang, Hubei

编号	姓名	性别	年龄	地址
12005000205	E	Female	21	Shangrao, Jiangxi
12005000206	F	Male	32	Zhuzhou, Hunan
12005000207	G	Female	29	Nanyang, Henan
12005000208	H	Female	30	Kaixian, Chongqing
12005000209	I	Male	26	Weinan, Shaanxi
12005000210	J	Male	25	Dalian, Liaoning

数据规划

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

3.2 开发思路

功能分解

根据上述的业务场景进行功能分解，需要开发的功能点如**表3-2**所示。

表 3-2 在 HBase 中开发的功能

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

关键设计原则

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

3.3 样例代码说明

3.3.1 配置参数

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

配置项如下：

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

其中：*value*中的值为ZooKeeper集群的域名。登录表格存储服务管理控制台，在左侧导航树单击“集群模式”，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”。

步骤2 在样例代码工程中修改IAM认证相关的参数。

- 如果CloudTable集群开启了IAM认证的功能

  在样例代码工程中修改com.huawei.cloudtable.hbase.examples.TestMain类中“IAM_AUTH_MODE”必须为“true”，同时配置user、ak和sk参数。

  代码如下：

  ```
  private static boolean IAM_AUTH_MODE = true;
  private static String user = "XXXXXX";
  private static String ak = "XXXXXX";
  private static String sk = "XXXXXX";
  ```

  - **user**：为用户名。如果集群是由用户的子用户创建的，子用户访问集群时user必须配置为“*子用户.最终用户*”。最终用户访问集群时user配置为用户名即可。
  - **ak和sk**：AK（Access Key ID）为访问密钥ID，SK（Secret Access Key）为私有访问密钥，分别设置为AK明文和SK明文。将鼠标移到管理控制台右上角的用户名，单击“我的凭证”，再单击“管理访问密钥”，可以查看已有的访问密钥，也可以单击“新增访问密钥”进行创建。

    说明

    IAM认证方式的安全性高于普通模式，建议CloudTable集群开启IAM认证功能，并在客户端或应用程序代码中采用IAM认证方式连接集群。

- 如果CloudTable集群没有开启IAM认证的功能

  com.huawei.cloudtable.hbase.examples.TestMain类中“IAM_AUTH_MODE”必须为“false”。

----结束

3.3.2 创建 Configuration

功能介绍

HBase通过加载配置文件来获取配置项。

说明

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

代码样例

下面代码片段在com.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。

3.3.3 IAM 集群认证

功能介绍

如果CloudTable集群开启了IAM认证功能，那么就需要先使用最终租户的AK和SK进行认证，才有权限可以进行后续的操作。

代码样例

```
private static boolean IAM_AUTH_MODE = true;
private static String user = "";
private static String ak = "";
private static String sk = "";
public static void login(Configuration conf) throws IOException {
  if (IAM_AUTH_MODE) {
    UserProviderExtend.loginWithAKSK(conf, user, ak, sk);
  }
}
```

说明

在进程的生命周期内，UserProviderExtend.loginWithAKSK函数只需要调用一次即可。

3.3.4 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);
}
```

3.3.5 创建表

功能简介

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

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

说明

表名以及列族名不能包含特殊字符，可以由字母、数字以及下划线组成。

代码样例

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

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

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

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

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

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

解释

（1）创建表描述符

（2）创建列族描述符

（3）添加列族描述符到表描述符中

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

（5）调用Admin的建表方法。

注意事项

- 注[1] 可以设置列族的压缩方式，代码片段如下：

```
//设置编码算法，HBase提供了DIFF，FAST_DIFF，PREFIX和PREFIX_TREE四种编码算法
 hcd.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);

 //设置文件压缩方式，HBase默认提供了GZ和SNAPPY两种压缩算法
 //其中GZ的压缩率高，但压缩和解压性能低，适用于冷数据
```



```
//SNAPPY压缩率低，但压缩解压性能高，适用于热数据
//建议默认开启SNAPPY压缩
hcd.setCompressionType(Compression.Algorithm.SNAPPY);
```

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

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

3.3.6 删除表

功能简介

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

代码样例

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

注意事项

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

3.3.7 修改表

功能简介

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

代码样例

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

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

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。

3.3.8 插入数据

功能简介

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

代码样例

```
public void testPut() {
  LOG.info("Entering testPut.");
  // Specify the column family name.
  byte[] familyName = Bytes.toBytes("info");
  // Specify the column name.
  byte[][] qualifiers = { Bytes.toBytes("name"), Bytes.toBytes("gender"),
      Bytes.toBytes("age"), Bytes.toBytes("address") };
```



```
Table table = null;
try {
  // Instantiate an HTable object.
  table = conn.getTable(tableName);
  List<Put> puts = new ArrayList<Put>();
  // Instantiate a Put object.
  Put put = new Put(Bytes.toBytes("012005000201"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("A"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("19"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shenzhen, Guangdong"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000202"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("B"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("23"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shijiazhuang, Hebei"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000203"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("C"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Ningbo, Zhejiang"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000204"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("D"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("18"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Xiangyang, Hubei"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000205"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("E"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("21"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Shangrao, Jiangxi"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000206"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("F"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("32"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Zhuzhou, Hunan"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000207"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("G"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("29"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Nanyang, Henan"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000208"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("H"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Female"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("30"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Kaixian, Chongqing"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000209"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("I"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("26"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Weinan, Shaanxi"));
  puts.add(put);
  put = new Put(Bytes.toBytes("012005000210"));
  put.addColumn(familyName, qualifiers[0], Bytes.toBytes("J"));
  put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Male"));
  put.addColumn(familyName, qualifiers[2], Bytes.toBytes("25"));
  put.addColumn(familyName, qualifiers[3], Bytes.toBytes("Dalian, Liaoning"));
  puts.add(put);
  // Submit a put request.
  table.put(puts);
```



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

注意事项

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

3.3.9 删除数据

功能简介

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

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.");
}
```


3.3.10 使用 Get 读取数据

功能简介

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

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.");
}
```

3.3.11 使用 Scan 读取数据

功能简介

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

代码样例

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

注意事项

1. 建议Scan时指定StartRow和StopRow，一个有确切范围的Scan，性能会更好些。
2. 可以设置Batch和Caching关键参数。
   - Batch

     使用Scan调用next接口每次最大返回的记录数，与一次读取的列数有关。
   - Caching

     RPC请求返回next记录的最大数量，该参数与一次RPC获取的行数有关。

3.3.12 使用过滤器 Filter

功能简介

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

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.");
}
```

4 开发 OpenTSDB 应用

4.1 典型场景说明

通过典型场景，我们可以快速学习和掌握OpenTSDB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，用于记录和查询城市的气象信息，记录数据如下表

表 4-1 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

城市	区域	时间	温度	湿度
Shenzhen	Longgang	2017/7/1 13:00:00	33	50
Shenzhen	Longgang	2017/7/1 14:00:00	32	50
Shenzhen	Longgang	2017/7/1 15:00:00	32	50
Shenzhen	Longgang	2017/7/1 16:00:00	31	51
Shenzhen	Longgang	2017/7/1 17:00:00	30	51
Shenzhen	Longgang	2017/7/1 18:00:00	30	51
Shenzhen	Longgang	2017/7/1 19:00:00	29	51
Shenzhen	Longgang	2017/7/1 20:00:00	29	52
Shenzhen	Longgang	2017/7/1 21:00:00	29	53
Shenzhen	Longgang	2017/7/1 22:00:00	28	54
Shenzhen	Longgang	2017/7/1 23:00:00	28	54

该场景里我们记录了深圳市龙岗区在2017年7月1日零时的温度和湿度数据，这里通过OpenTSDB的方式建模实质上是两组数据点。

表 4-2 指标数据点 1

指标项(metric)	城市(city)	区域(region)	Unix timestamp	指标数值(value)
city.temp	Shenzhen	Longgang	1498838400	28
city.temp	Shenzhen	Longgang	1498842000	27
city.temp	Shenzhen	Longgang	1498845600	27
city.temp	Shenzhen	Longgang	1498849200	27
city.temp	Shenzhen	Longgang	1498852800	27
city.temp	Shenzhen	Longgang	1498856400	27
city.temp	Shenzhen	Longgang	1498860000	27
city.temp	Shenzhen	Longgang	1498863600	27
city.temp	Shenzhen	Longgang	1498867200	29
city.temp	Shenzhen	Longgang	1498870800	30
city.temp	Shenzhen	Longgang	1498874400	32
city.temp	Shenzhen	Longgang	1498878000	32
city.temp	Shenzhen	Longgang	1498881600	33

指标项 (metric)	城市(city)	区域 (region)	Unix timestamp	指标数值 (value)
city.temp	Shenzhen	Longgang	1498885200	33
city.temp	Shenzhen	Longgang	1498888800	32
city.temp	Shenzhen	Longgang	1498892400	32
city.temp	Shenzhen	Longgang	1498896000	31
city.temp	Shenzhen	Longgang	1498899600	30
city.temp	Shenzhen	Longgang	1498903200	30
city.temp	Shenzhen	Longgang	1498906800	29
city.temp	Shenzhen	Longgang	1498910400	29
city.temp	Shenzhen	Longgang	1498914000	29
city.temp	Shenzhen	Longgang	1498917600	28
city.temp	Shenzhen	Longgang	1498921200	28

表 4-3 指标数据点 2

指标项 (metric)	城市(city)	区域 (region)	Unix timestamp	指标数值 (value)
city.hum	Shenzhen	Longgang	1498838400	54
city.hum	Shenzhen	Longgang	1498842000	53
city.hum	Shenzhen	Longgang	1498845600	52
city.hum	Shenzhen	Longgang	1498849200	51
city.hum	Shenzhen	Longgang	1498852800	50
city.hum	Shenzhen	Longgang	1498856400	49
city.hum	Shenzhen	Longgang	1498860000	48
city.hum	Shenzhen	Longgang	1498863600	46
city.hum	Shenzhen	Longgang	1498867200	46
city.hum	Shenzhen	Longgang	1498870800	48
city.hum	Shenzhen	Longgang	1498874400	48
city.hum	Shenzhen	Longgang	1498878000	49
city.hum	Shenzhen	Longgang	1498881600	49
city.hum	Shenzhen	Longgang	1498885200	50

指标项(metric)	城市(city)	区域(region)	Unix timestamp	指标数值(value)
city.hum	Shenzhen	Longgang	1498888800	50
city.hum	Shenzhen	Longgang	1498892400	50
city.hum	Shenzhen	Longgang	1498896000	51
city.hum	Shenzhen	Longgang	1498899600	51
city.hum	Shenzhen	Longgang	1498903200	51
city.hum	Shenzhen	Longgang	1498906800	51
city.hum	Shenzhen	Longgang	1498910400	52
city.hum	Shenzhen	Longgang	1498914000	53
city.hum	Shenzhen	Longgang	1498917600	54
city.hum	Shenzhen	Longgang	1498921200	54

其中这两组指标数据点都有2个标签：

- 标签(tag)：城市city、区域region
- 标签值(tag value)：ShenZhen、Longgang

用户可以执行以下数据操作：

- 获取每天的监控数据，通过OpenTSDB的put接口将两个组数据点写入数据库中。
- 对已有的数据，使用OpenTSDB的query接口进行数据查询和分析。

4.2 开发思路

功能分解

根据上述的业务场景进行功能分解，需要开发的功能点如表4-4。

表 4-4 在 OpenTSDB 中开发的功能

序号	步骤	代码实现
1	根据典型场景说明建立了数据模型	请参见**写入数据**
2	写入指标数据	请参见**写入数据**
3	根据指标项进行数据查询	请参见**查询数据**
4	删除指定范围的数据	请参见**删除数据**

4.3 样例代码说明

4.3.1 配置参数

步骤1 执行样例代码前，必须在样例代码工程中修改“com.huawei.cloudtable.opentsdb.examples.OpenTsdbSample”类中的如下参数：

```
private final static String OPENTSDB_IP = "opentsdb-vdswm7gj45awlom.cloudtable.com";
private final static int OPENTSDB_PORT = 4242;
```

- OPENTSDB_IP：修改为CloudTable集群信息页面上的OpenTSDB的URL地址。
- OPENTSDB_PORT：端口修改为4242。

步骤2 如果CloudTable集群开启了IAM认证的功能，在样例代码工程中，修改“com.huawei.cloudtable.opentsdb.examples.OpenTsdbSample”类中的如下参数：

```
private final static boolean securityMode = true;
private final static String PROJECT_ID = "XXXXXX";
private final static String USER = "XXXXXX";
private final static String AK = "XXXXXX";
private final static String TOKEN = "XXXXXX";
```

- securityMode：设置为true。如果CloudTable集群未开启IAM认证功能，必须将该参数设置为false。

  说明

  IAM认证方式的安全性高于普通模式，建议CloudTable集群开启IAM认证功能，并在客户端或应用程序代码中采用IAM认证方式连接集群。

- PROJECT_ID：设置项目ID。将鼠标移到管理控制台右上角的用户名，单击“我的凭证”。在“项目列表”页面可以查看项目ID。
- USER：设置为IAM用户名。如果集群是由用户的子用户创建的，子用户访问集群时user必须配置为“*子用户.最终用户*”。最终用户访问集群时user配置为用户名即可。
- AK：Access Key ID，即访问密钥ID。将鼠标移到管理控制台右上角的用户名，单击“我的凭证”，再单击“管理访问密钥”，可以查看已有的访问密钥，也可以单击“新增访问密钥”进行创建。
- TOKEN：可以通过org.apache.hadoop.hbase.security.token.web.AKSKWebTokenCommonUtil#createPassword方法生成。

步骤3 （可选）如果不使用样例工程，样例代码只需要依赖如下第三方jar：

1. gson

```
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.2.4</version>
</dependency>
```

2. httpcore

```
<!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpcore -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpcore</artifactId>
    <version>4.4.4</version>
</dependency>
```


3. httpclient

```
<!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.2</version>
</dependency>
```

步骤4 每个HTTP请求都应该设置超时间，设置超时时间的方法如下：

```
public static void addTimeout(HttpRequestBase req) {
  RequestConfig requestConfig = RequestConfig.custom().setConnectTimeout(5000)
      .setConnectionRequestTimeout(10000).setSocketTimeout(60000).build();
  req.setConfig(requestConfig);
}
```

----结束

4.3.2 IAM 集群认证

功能简介

如果CloudTable启用了IAM认证功能，那么OpenTSDB必须使用HTTPS进行连接，并且在HTTP请求的HEADER中携带必需的参数，如下表所示：

表 4-5 HTTP Header 携带的参数

HTTP Header	Value
X-TSD-IamAuth	true
X-Auth-ProjectId	集群所在的ProjectID
X-Auth-User	租户名
X-Auth-AK	租户的AccessKey
X-Auth-Token	使用租户AccessKey和SecretKey生成的Token信息

说明

Token的生成方法如下：

在客户端主机的操作系统的shell界面中执行。在客户端机器上进入HBase目录后执行Token工具，Token工具的命令格式如下：

./bin/hbase com.huawei.cloudtable.tool.RestTokenUtil <AccessKey> <SecretKey> <UserName>

“AccessKey”：用户的AccessKey。

“SecretKey”：用户的SecretKey。

“UserName”：用户名。

例如：

```
./bin/hbase com.huawei.cloudtable.tool.RestTokenUtil YourAccessKey YourSecretKey YourUserName
```

样例代码

使用HTTPS连接的时候，应用侧需要跳过校验证书。使用如下方法创建的HTTP Client可以跳过校验证书：

```
private static CloseableHttpClient createSSLClientDefault() {
  try {
    X509TrustManager x509mgr = new X509TrustManager() {
      public void checkClientTrusted(X509Certificate[] xcs, String string) {
      }

      public void checkServerTrusted(X509Certificate[] xcs, String string) {
      }

      public X509Certificate[] getAcceptedIssuers() {
        return null;
      }
    };
    SSLContext sslContext = SSLContext.getInstance("TLS");
    sslContext.init(null, new TrustManager[] { x509mgr }, null);
    @SuppressWarnings("deprecation")
    SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslContext,
        SSLConnectionSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);

    return HttpClients.custom().setSSLSocketFactory(sslsf).build();
  } catch (KeyManagementException e) {
    throw new RuntimeException(e);
  } catch (NoSuchAlgorithmException e) {
    throw new RuntimeException(e);
  } catch (Exception e) {
    throw new RuntimeException(e);
  }
}
```

在构造HTTP请求的时候，需要在HTTP请求中增加必需的HEADER，方法如下：

```
HttpPost httpPost = new HttpPost(PUT_URL);
httpPost .addHeader("X-TSD-IamAuth", "true");
httpPost .addHeader("X-Auth-ProjectId", PROJECT_ID);
httpPost .addHeader("X-Auth-User", USER);
httpPost .addHeader("X-Auth-AK", AK);
httpPost .addHeader("X-Auth-Token", TOKEN);
```

4.3.3 写入数据

功能简介

使用OpenTSDB的接口写入数据。

函数genWeatherData()模拟生成的气象数据，函数put()发送气象数据到OpenTSDB服务端。

样例代码

```
private static String PUT_URL = (securityMode ? "https://" : "http://") + OPENTSDB_IP + ":"
    + OPENTSDB_PORT + "/api/put/?sync&sync_timeout=60000";

static class DataPoint {
  public String metric;
  public Long timestamp;
  public Double value;
  public Map<String, String> tags;
  public DataPoint(String metric, Long timestamp, Double value, Map<String, String> tags) {
```

```
    this.metric = metric;
    this.timestamp = timestamp;
    this.value = value;
    this.tags = tags;
  }
}

private String genWeatherData() {
  List<DataPoint> dataPoints = new ArrayList<DataPoint>();
  Map<String, String> tags = ImmutableMap.of("city", "Shenzhen", "region", "Longgang");

  // Data of air temperature
  dataPoints.add(new DataPoint("city.temp", 1498838400L, 28.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498842000L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498845600L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498849200L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498852800L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498856400L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498860000L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498863600L, 27.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498867200L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498870800L, 30.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498874400L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498878000L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498881600L, 33.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498885200L, 33.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498888800L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498892400L, 32.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498896000L, 31.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498899600L, 30.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498903200L, 30.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498906800L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498910400L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498914000L, 29.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498917600L, 28.0, tags));
  dataPoints.add(new DataPoint("city.temp", 1498921200L, 28.0, tags));

  // Data of humidity
  dataPoints.add(new DataPoint("city.hum", 1498838400L, 54.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498842000L, 53.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498845600L, 52.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498849200L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498852800L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498856400L, 49.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498860000L, 48.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498863600L, 46.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498867200L, 46.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498870800L, 48.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498874400L, 48.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498878000L, 49.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498881600L, 49.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498885200L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498888800L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498892400L, 50.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498896000L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498899600L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498903200L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498906800L, 51.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498910400L, 52.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498914000L, 53.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498917600L, 54.0, tags));
  dataPoints.add(new DataPoint("city.hum", 1498921200L, 54.0, tags));

  Gson gson = new Gson();
  return gson.toJson(dataPoints);
}

public void put() throws ClientProtocolException, IOException {
  try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
```

```
    HttpPost httpPost = new HttpPost(PUT_URL);
    // 请求需要设置超时时间
    addTimeout(httpPost);
    String weatherData = genWeatherData();
    StringEntity entity = new StringEntity(weatherData, "ISO-8859-1");
    entity.setContentType("application/json");
    httpPost.setEntity(entity);
    HttpResponse response = httpClient.execute(httpPost);

    int statusCode = response.getStatusLine().getStatusCode();
    System.out.println("Status Code : " + statusCode);
    if (statusCode != HttpStatus.SC_NO_CONTENT) {
      System.out.println("Request failed! " + response.getStatusLine());
    }
  }
}
```

说明

PUT_URL中加入了sync参数，表示必须等到数据写入HBase后才可以返回，强烈建议使用此参数；如果不使用sync，表示采用异步写入HBase的方式，可能存在丢失数据的风险。具体信息请参考**OpenTSDB API简介**。

4.3.4 查询数据

功能简介

使用OpenTSDB的查询接口读取数据。

函数genQueryReq()生成查询请求，函数query()把查询请求发送到OpenTSDB服务端。

样例代码

```
private static String QUERY_URL = (securityMode ? "https://" : "http://") + OPENTSDB_IP + ":"
    + OPENTSDB_PORT + "/api/query";

static class Query {
  public Long start;
  public Long end;
  public boolean delete = false;
  public List<SubQuery> queries;
}

static class SubQuery {
  public String metric;
  public String aggregator;
  public SubQuery(String metric, String aggregator) {
    this.metric = metric;
    this.aggregator = aggregator;
  }
}

String genQueryReq() {
  Query query = new Query();
  query.start = 1498838400L;
  query.end = 1498921200L;
  query.queries = ImmutableList.of(new SubQuery("city.temp", "sum"), new SubQuery("city.hum", "sum"));
  Gson gson = new Gson();
  return gson.toJson(query);
}

public void query() throws ClientProtocolException, IOException {
  try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
    HttpPost httpPost = new HttpPost(QUERY_URL);
```

```
    // 请求需要设置超时时间
    addTimeout(httpPost);
    String queryRequest = genQueryReq();
    //System.out.println("Request=" + queryRequest);
    StringEntity entity = new StringEntity(queryRequest, "ISO-8859-1");
    entity.setContentType("application/json");
    httpPost.setEntity(entity);
    HttpResponse response = httpClient.execute(httpPost);

    int statusCode = response.getStatusLine().getStatusCode();
    System.out.println("Status Code : " + statusCode);
    if (statusCode != HttpStatus.SC_OK) {
      System.out.println("Request failed! " + response.getStatusLine());
    }

    String body = EntityUtils.toString(response.getEntity(), "ISO-8859-1");
    System.out.println("Response content : " + body);
  }
}
```

4.3.5 删除数据

功能简介

在OpenTSDB的查询接口中增加delete参数，并且设置delete参数为true。函数genQueryReq()生成删除请求，函数delete()把删除请求发送到OpenTSDB服务端。

样例代码

```
private static String QUERY_URL = (securityMode ? "https://" : "http://") + OPENTSDB_IP + ":"
      + OPENTSDB_PORT + "/api/query";

static class Query {
  public Long start;
  public Long end;
  public boolean delete = false;
  public List<SubQuery> queries;
}

static class SubQuery {
  public String metric;
  public String aggregator;
  public SubQuery(String metric, String aggregator) {
    this.metric = metric;
    this.aggregator = aggregator;
  }
}

String genQueryReq() {
  Query query = new Query();
  query.start = 1498838400L;
  query.end = 1498921200L;
  query.queries = ImmutableList.of(new SubQuery("city.temp", "sum"), new SubQuery("city.hum", "sum"));
  Gson gson = new Gson();
  return gson.toJson(query);
}

String genDeleteReq() {
  Query query = new Query();
  query.start = 1498838400L;
  query.end = 1498921200L;
  query.queries = ImmutableList.of(new SubQuery("city.temp", "sum"), new SubQuery("city.hum", "sum"));
  query.delete = true;

  Gson gson = new Gson();
  return gson.toJson(query);
}
```

```
public void delete() throws ClientProtocolException, IOException {
  try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
    HttpPost httpPost = new HttpPost(QUERY_URL);
    // 请求需要设置超时时间
    addTimeout(httpPost);
    String deleteRequest = genDeleteReq();
    StringEntity entity = new StringEntity(deleteRequest, "ISO-8859-1");
    entity.setContentType("application/json");
    httpPost.setEntity(entity);
    HttpResponse response = httpClient.execute(httpPost);

    int statusCode = response.getStatusLine().getStatusCode();
    System.out.println("Status Code : " + statusCode);
    if (statusCode != HttpStatus.SC_OK) {
      System.out.println("Request failed! " + response.getStatusLine());
    }
  }
}
```

说明

query.delete的参数设置为true后，表示会把查询到的数据都执行删除。具体请参考**OpenTSDB API简介**。

4.4 性能调优

OpenTSDB服务的吞吐量可以通过扩容tsd节点数量来横向扩展，当前采用Round Robin DNS的机制来把应用端的请求均衡负载到不同的tsd节点。为了使应用端的http请求更加细粒度地均衡分发，可以优化应用端DNS域名解析缓存。

JVM 优化

- 代码中关闭JVM的DNS域名缓存，同时使用http连接池发送请求。

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "0")
  private static CloseableHttpClient httpClient;
  static {
    PoolingHttpClientConnectionManager cm = new PoolingHttpClientConnectionManager();
    cm.setMaxTotal(200);
    cm.setDefaultMaxPerRoute(20);
    cm.setDefaultMaxPerRoute(50);
    httpClient = HttpClients.custom().setConnectionManager(cm).build();
  }
```

- 代码中设置JVM的DNS域名缓存时间为1秒。

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
```

- 如果需要考虑扩容后不重启应用能够识别出新的IP地址，可以考虑设置连接的TTL，但是对性能会有一点影响。

```
HttpClients.custom().setConnectionTimeToLive(600, TimeUnit.SECONDS);
```

OS 优化

- Linux操作系统关闭本地DNS域名解析缓存。检查/etc/init.d/目录下是否有nscd 或者 named 或者 dnsmasq，如果存在，分别执行如下命令关闭缓存。

```
/etc/init.d/nscd stop
/etc/init.d/named stop
/etc/init.d/dnsmasq stop
```

- Windows操作系统关闭本地DNS域名解析缓存。在cmd中执行如下命令。

```
net stop dnscache
```

5 开发 GeoMesa 应用

5.1 典型场景说明

通过典型场景，我们可以快速学习和掌握GeoMesa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，用于记录和查询其与朋友见面约会信息：与谁在什么时间什么地点做了什么，记录数据格式如表：

字段名称	字段类型
Who	String
What	Long
When	Date
Where	Point
Why	String

5.2 开发思路

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：

序号	步骤	代码实现
1	连接GeoMesa集群	请参见**创建DataStore**。
2	创建数据表	请参见**创建数据表**。
3	插入数据	请参见**插入数据**。
4	查询数据	请参见**查询数据**。

5.3 样例代码说明

5.3.1 配置参数

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

配置项如下：

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

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

----结束

5.3.2 创建 DataStore

功能简介

使用GeoMesa的接口连接GeoMesa集群创建DataStore，DataStore实例提供可对GeoMesa指定目录操作的接口。

样例代码

```
// find out where -- in HBase -- the user wants to store data
CommandLineParser parser = new BasicParser();
Options options = getCommonRequiredOptions();
CommandLine cmd = parser.parse(options, new String[]{"--bigtable_table_name", "geomesa"});
// verify that we can see this HBase destination in a GeoTools manner
Map<String, Serializable> dsConf = getHBaseDataStoreConf(cmd);
DataStore dataStore = DataStoreFinder.getDataStore(dsConf);
```

5.3.3 创建数据表

功能简介

在指定目录的DataStore中创建数据表。

样例代码

```
// establish specifics concerning the SimpleFeatureType to store
String simpleFeatureTypeName = "QuickStart";
// list the attributes that constitute the feature type
String attributes = "Who:String,What:java.lang.Long,When:Date,*Where:Point:srid=4326,Why:String";
// create the bare simple-feature type
SimpleFeatureType simpleFeatureType = SimpleFeatureTypes.createType(simpleFeatureTypeName,
attributes);
dataStore.createSchema(simpleFeatureType);
```


5.3.4 插入数据

功能介绍

构造数据并将数据插入到指定表中。

样例代码

```
DefaultFeatureCollection featureCollection = new DefaultFeatureCollection();
String id;
Object[] NO_VALUES = {};
String[] PEOPLE_NAMES = {"Addams", "Bierce", "Clemens"};
Long SECONDS_PER_YEAR = 365L * 24L * 60L * 60L;
Random random = new Random(5771);
DateTime MIN_DATE = new DateTime(2014, 1, 1, 0, 0, 0, DateTimeZone.forID("UTC"));
Double MIN_X = -79.5;
Double MIN_Y = 37.0;
Double DX = 2.0;
Double DY = 2.0;
for (int i = 0; i < numNewFeatures; i++) {
  // create the new (unique) identifier and empty feature shell
  id = "Observation." + Integer.toString(i);
  SimpleFeature simpleFeature = SimpleFeatureBuilder.build(simpleFeatureType, NO_VALUES, id);
  // be sure to tell GeoTools explicitly that you want to use the ID you provided
  simpleFeature.getUserData().put(Hints.USE_PROVIDED_FID, java.lang.Boolean.TRUE);
  // populate the new feature's attributes
  // Who: string value
  simpleFeature.setAttribute("Who", PEOPLE_NAMES[i % PEOPLE_NAMES.length]);
  // What: long value
  simpleFeature.setAttribute("What", i);
  // Where: location: construct a random point within a 2-degree-per-side square
  double x = MIN_X + random.nextDouble() * DX;
  double y = MIN_Y + random.nextDouble() * DY;
  Geometry geometry = WKTUtils$.MODULE$.read("POINT(" + x + " " + y + ")");
  simpleFeature.setAttribute("Where", geometry);
  // When: date-time:construct a random instant within a year
  DateTime dateTime = MIN_DATE.plusSeconds((int) Math.round(random.nextDouble() *
SECONDS_PER_YEAR));
  simpleFeature.setAttribute("When", dateTime.toDate());
  // Why: another string value
  // left empty, showing that not all attributes need values
  // accumulate this new feature in the collection
  featureCollection.add(simpleFeature);
  FeatureStore featureStore = (FeatureStore) dataStore.getFeatureSource(simpleFeatureTypeName);
  featureStore.addFeatures(featureCollection);
}
```

5.3.5 查询数据

功能简介

根据指定时空条件查询数据。

样例代码

```
// construct a (E)CQL filter from the search parameters,
// and use that as the basis for the query
Filter cqlFilter = createFilter(geomField, x0, y0, x1, y1, dateField, t0, t1, attributesQuery);
Query query = new Query(simpleFeatureTypeName, cqlFilter);
List<String> properties = new LinkedList<>();
// submit the query, and get back an iterator over matching features
FeatureSource featureSource = dataStore.getFeatureSource(simpleFeatureTypeName);
FeatureCollection featureCollection = featureSource.getFeatures(query);
FeatureIterator featureItr = featureCollection.features();
```



```
// loop through all results
int n = 0;
while (featureItr.hasNext()) {
  Feature feature = featureItr.next();
  feature.getIdentifier().getID();
  System.out.println((++n) + "." +
    feature.getProperty("Who").getValue() + "|" +
    feature.getProperty("What").getValue() + "|" +
    feature.getProperty("When").getValue() + "|" +
    feature.getProperty("Where").getValue() + "|" +
    feature.getProperty("Why").getValue());
}
featureItr.close();
```

6 开发 HBase Elasticsearch 全文检索应用

6.1 应用背景

HBase-Elasticsearch的全文检索能力，是以HBase为基础存储用户源数据，在KV（key value）查询能力的基础上使用云搜索服务（简称CSS）中的Elasticsearch搜索引擎来补充全文检索能力。用户可以根据自身业务需求来定义HBase中的哪些字段需要全文检索，在创建HBase表时会自动连接用户指定的云搜索服务集群并在Elasticsearch中创建索引，索引数据存放在Elasticsearch。同时，HBase的原生API（put和scan接口）支持索引数据的写入和查询。

6.2 使用前提

用户创建的表格存储服务集群（HBase）、弹性云服务器（ECS）实例（作为HBase客户端）、云搜索服务集群（Elasticsearch引擎）需要保持VPC、子网和安全组一致，来确保网络通畅。

Elasticsearch的基础知识，请参考**https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html**。

6.3 典型场景说明

某专业文献公司使用HBase收录了文献内容以及字数统计、评分、等级等相关信息。表中contentCh（文献中文内容）、contentEng（文献英文内容）需要提供关键词检索的功能。

文献内容信息表字段如下：

表 6-1 字段说明

字段名称	在Elasticsearch中映射的字段类型	是否需要全文索引	字段描述
contentCh	text	是	文献中文内容
contentEng	text	是	文献英文内容

字段名称	在Elasticsearch中映射的字段类型	是否需要全文索引	字段描述
id	long	否	文献刊发id
charNum	int	否	文献字数总量
pageNum	short	否	文献页数
level	byte	否	文献等级
researchCost	double	否	研究费用
score	float	否	文献评分
male	boolean	否	作者性别
whatever	-	否	HBase表中的其他字段，但是不在Elasticsearch中做字段映射。

6.4 HBase Elasticsearch schema 说明

HBase通过表的METADATA来存储Elasticsearch schema的定义：

表 6-2 schema 定义

字段名称	value说明	是否必填
hbase.index.es.enabled	该HBase表在Elasticsearch中是否创建全文索引，true表示创建，默认为false。	是
hbase.index.es.endpoint	云搜索服务集群（Elasticsearch引擎）的访问地址，例如'ip1:port,ip2:port'。	是
hbase.index.es.indexname	HBase表对应在Elasticsearch中的索引名称，必须小写。	是
hbase.index.es.shards	Elasticsearch中索引的分片数量，默认5。取值为大于等于1的整数。	否
hbase.index.es.replicas	Elasticsearch中的索引的副本数量，默认1。取值为大于等于0的整数。	否

字段名称	value说明	是否必填
hbase.index.es.schema	HBase和Elasticsearch的字段映射，json数组格式的字符，每个元素包含以下字段： • **name**：Elasticsearch中的字段名称。 • **type**：Elasticsearch中的字段类型。 • **hbaseQualifier**：数据源HBase qualifier。 • **analyzer**：text类型的字段通过“analyzer”可以指定分词器。中文分词器一般使用“ik_smart”。默认是“Standard”分词器，支持英文。 例如： '[{"name":"contentCh","type":"text","hbaseQualifier":"cf1:contentCh","analyzer":"ik_smart"}, {"name":"contentEng","type":"text","hbaseQualifier":"cf2:contentEng"},{"name":"id","type":"long","hbaseQualifier":"cf1:id"}]'	是

HBase-Elasticsearch全文检索当前支持的数据类型有{"text", "long", "integer", "short", "byte", "double", "float","boolean"}，也就是schema中**type**的取值类型。text是Elasticsearch中的文本类型。全文检索一般是指对text类型数据的检索，同时也支持基本数据类型的准确检索。

6.5 开发思路

表 6-3 开发思路

序号	步骤	代码实现
1	创建HBase表时开启Elasticsearch中的索引	请参见**创建数据表开启Elasticsearch索引**
2	HBase put写入数据	请参见**写入数据**
3	HBase scan查询数据	请参见**查询数据**

6.6 样例代码说明

6.6.1 配置参数

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

配置项如下：

```
<property>
<name>hbase.zookeeper.quorum</name>
```



```
<value>xxx-zk1.cloudtable.com,xxx-zk2.cloudtable.com,xxx-zk3.cloudtable.com</value>
</property>
```

其中：*value*中的值为ZooKeeper集群的域名。登录表格存储服务管理控制台，在左侧导航树单击“集群模式”，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”。

步骤2 在样例代码工程中修改IAM认证相关的参数。

- 如果CloudTable集群开启了IAM认证的功能

  在样例代码工程中修改com.huawei.cloudtable.hbase.examples.ES.HBaseESTestMain类中“IAM_AUTH_MODE”必须为“true”，同时配置user、ak和sk参数。

  代码如下：

  ```
  private static boolean IAM_AUTH_MODE = true;
  private static String user = "XXXXXX";
  private static String ak = "XXXXXX";
  private static String sk = "XXXXXX";
  ```

  - **user**：为用户名。如果集群是由用户的子用户创建的，子用户访问集群时user必须配置为“*子用户.最终用户*”。最终用户访问集群时user配置为用户名即可。
  - **ak和sk**：AK（Access Key ID）为访问密钥ID，SK（Secret Access Key）为私有访问密钥，分别设置为AK明文和SK明文。将鼠标移到管理控制台右上角的用户名，单击“我的凭证”，再单击“管理访问密钥”，可以查看已有的访问密钥，也可以单击“新增访问密钥”进行创建。

  说明

  IAM认证方式的安全性高于普通模式，建议CloudTable集群开启IAM认证功能，并在客户端或应用程序代码中采用IAM认证方式连接集群。

- 如果CloudTable集群没有开启IAM认证的功能

  com.huawei.cloudtable.hbase.examples.ES.HBaseESTestMain类中“IAM_AUTH_MODE”必须为“false”。

----结束

6.6.2 创建 Configuration

功能介绍

HBase通过加载配置文件来获取配置项。

说明

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

代码样例

下面代码片段在com.huawei.cloudtable.hbase.examples.ES.HBaseESTestMain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。

6.6.3 创建数据表开启 Elasticsearch 索引

功能简介

建表功能同**创建表**，在此基础上，表属性配置Elasticsearch中的索引开启的字段，请参见**HBase Elasticsearch schema说明**。

样例代码

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

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

  // Set the column family name to info.
  HColumnDescriptor hcd = new HColumnDescriptor(CF1);

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

  // 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);

  HColumnDescriptor hcd2 = new HColumnDescriptor(CF2);
  hcd2.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
  hcd2.setCompressionType(Compression.Algorithm.SNAPPY);

  htd.addFamily(hcd);
  htd.addFamily(hcd2);

  //add HBase ES schema
  String ESJsonSchema = "[" +
    "{\"name\":\"contentCh\",\"type\":\"text\",\"hbaseQualifier\":\"cf1:contentCh\",\"analyzer\":\"ik_smart
\"}," +
    "{\"name\":\"contentEng\",\"type\":\"text\",\"hbaseQualifier\":\"cf2:contentEng\"}," +
    "{\"name\":\"id\",\"type\":\"long\",\"hbaseQualifier\":\"cf1:id\"}," +
    "{\"name\":\"charNum\",\"type\":\"integer\",\"hbaseQualifier\":\"cf1:charNum\"}," +
    "{\"name\":\"pageNum\",\"type\":\"short\",\"hbaseQualifier\":\"cf1:pageNum\"}," +
    "{\"name\":\"level\",\"type\":\"byte\",\"hbaseQualifier\":\"cf1:level\"}," +
    "{\"name\":\"researchCost\",\"type\":\"double\",\"hbaseQualifier\":\"cf1:researchCost\"}," +
    "{\"name\":\"score\",\"type\":\"float\",\"hbaseQualifier\":\"cf1:score\"}," +
    "{\"name\":\"male\",\"type\":\"boolean\",\"hbaseQualifier\":\"cf1:male\"}" +
    "]";
  htd.setValue(HBaseESConst.HBASE_INDEX_ES_ENABLED, "true");
  htd.setValue(HBaseESConst.HBASE_INDEX_ES_ENDPOINT, ESClusterHosts);//(1)
  htd.setValue(HBaseESConst.HBASE_INDEX_ES_INDEXNAME, ES_INDEX_NAME);//(2)
```

```
  htd.setValue(HBaseESConst.HBASE_INDEX_ES_SCHEMA, ESJsonSchema);//(3)

  Admin admin = null;
  try {
    // Instantiate an Admin object.
    admin = conn.getAdmin();
    if (!admin.tableExists(tableName)) {
      LOG.info("Creating table...");
      admin.createTable(htd);
      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.");
}
```

说明：

（1）从云搜索服务console界面，查看的云搜索服务（Elasticsearch搜索引擎）集群访问地址。

（2）用户自定义索引名称。

（3）HBase字段和Elasticsearch字段映射信息。

6.6.4 写入数据

功能简介

写入数据同**插入数据**，需要注意的是，写入的数据必须包含Elasticsearch中索引的所有字段 。字段不全，Elasticsearch索引数据异常，用户客户端需要根据自己的业务场景处理ESDocIndexException异常，可以重试，可以忽略或其他操作。

样例代码

样例代码如下，请将代码中的字符串“****replace_with_Chinese_characters****”替换为中文字符。

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

   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("rowkey001"));
     put.addColumn(CF1, QUA_ARTICLE_CONTENT_CHINESE,
Bytes.toBytes("***replace_with_Chinese_characters***"));
     put.addColumn(CF2, QUA_ARTICLE_CONTENT_English, Bytes.toBytes("how many apples in the
```



```
market"));
    put.addColumn(CF1, QUA_ARTICLE_ID, Bytes.toBytes(1111l));
    put.addColumn(CF1, QUA_CHARACTER_NUM, Bytes.toBytes(1));
    short shortNum = 1;
    put.addColumn(CF1, QUA_PAGE_NUM, Bytes.toBytes(shortNum));
    char c = 'A';
    put.addColumn(CF1, QUA_ARTICLE_LEVEL, new byte[]{(byte)c});
    put.addColumn(CF1, QUA_RESEARCH_COST, Bytes.toBytes(111.11d));
    put.addColumn(CF1, QUA_ARTICLE_SCORE, Bytes.toBytes(80.5f));
    put.addColumn(CF1, QUA_AUTHOR_MALE, Bytes.toBytes(true));
    put.addColumn(CF1, QUA_WHATEVER, Bytes.toBytes("happy life and sweet love"));
    puts.add(put);

    put = new Put(Bytes.toBytes("rowkey002"));
    put.addColumn(CF1, QUA_ARTICLE_CONTENT_CHINESE,
Bytes.toBytes("***replace_with_Chinese_characters***"));
    put.addColumn(CF2, QUA_ARTICLE_CONTENT_English, Bytes.toBytes("how many people in the
swimming pool"));
    put.addColumn(CF1, QUA_ARTICLE_ID, Bytes.toBytes(2222l));
    put.addColumn(CF1, QUA_CHARACTER_NUM, Bytes.toBytes(2));
    shortNum = 2;
    put.addColumn(CF1, QUA_PAGE_NUM, Bytes.toBytes(shortNum));
    c = 'B';
    put.addColumn(CF1, QUA_ARTICLE_LEVEL, new byte[]{(byte)c});
    put.addColumn(CF1, QUA_RESEARCH_COST, Bytes.toBytes(222.22d));
    put.addColumn(CF1, QUA_ARTICLE_SCORE, Bytes.toBytes(170.5f));
    put.addColumn(CF1, QUA_AUTHOR_MALE, Bytes.toBytes(true));
    put.addColumn(CF1, QUA_WHATEVER, Bytes.toBytes("forever young"));
    puts.add(put);

    put = new Put(Bytes.toBytes("rowkey003"));
    put.addColumn(CF1, QUA_ARTICLE_CONTENT_CHINESE,
Bytes.toBytes("***replace_with_Chinese_characters***"));
    put.addColumn(CF2, QUA_ARTICLE_CONTENT_English, Bytes.toBytes("we play video game in night"));
    put.addColumn(CF1, QUA_ARTICLE_ID, Bytes.toBytes(3333l));
    put.addColumn(CF1, QUA_CHARACTER_NUM, Bytes.toBytes(3));
    shortNum = 3;
    put.addColumn(CF1, QUA_PAGE_NUM, Bytes.toBytes(shortNum));
    c = 'C';
    put.addColumn(CF1, QUA_ARTICLE_LEVEL, new byte[]{(byte)c});
    put.addColumn(CF1, QUA_RESEARCH_COST, Bytes.toBytes(333.33d));
    put.addColumn(CF1, QUA_ARTICLE_SCORE, Bytes.toBytes(180.5f));
    put.addColumn(CF1, QUA_AUTHOR_MALE, Bytes.toBytes(false));
    put.addColumn(CF1, QUA_WHATEVER, Bytes.toBytes("wishes always with you"));
    puts.add(put);

    // Submit a put request.
    try {
      table.put(puts);
    }
    // if your put operation does not contain the all field in your schema, you will get ESDocIndexException.
    // ESDocIndexException means data/document indexing to ES failed, but remember data put in HBase
success.
    // so here you handle this exception depends on your business( you can retry or ignore).
    catch (ESDocIndexException e) {
      //TODO
    }

    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.");
}
```

6.6.5 查询数据

功能简介

HBase通过原生的scan API来支持全文索引能力，同**使用Scan读取数据**类似。在此基础上，需要在configuration中配置HBASE_CLIENT_CONNECTION_IMPL为"org.apache.hadoop.hbase.client.LemonConnectionImplementation"，同时将全文检索条件传入CloudTable定制好的过滤器ESColumnValueFilter。

样例代码

1. configuration定义HBASECLIENTCONNECTION_IMPL为LemonConnectionImplementation实现类。

   样例代码如下：

   ```
   conf = HBaseConfiguration.create();
   conf.set(HConnection.HBASE_CLIENT_CONNECTION_IMPL,
   "org.apache.hadoop.hbase.client.LemonConnectionImplementation");
   Connection conn = ConnectionFactory.createConnection(conf);
   ```

2. 全文检索条件通过ESColumnValueFilter的httpParameters或reqBodyJson传入，参数格式和开源Elasticsearch的官方文档一致，请分别参见**URI Search**和**Request Body Search**。

   如下样例代码通过httpParameters来查询（请将代码中的字符串“***replace_with_Chinese_keywords_01***”替换为中文检索词）：

   ```
   public void testScanDataWithES1() {
     LOG.info("Entering testScanDataWithES1.");

     Table table = null;
     // Instantiate a ResultScanner object.
     ResultScanner rScanner = null;
     try {
       // Create the Configuration instance.
       table = conn.getTable(tableName);
       assert table instanceof LemonWrapperHTable;

       // set specified qualifier.
       Scan scan = new Scan();
       scan.addColumn(CF1, QUA_ARTICLE_CONTENT_CHINESE);
       scan.addColumn(CF2, QUA_ARTICLE_CONTENT_English);
       scan.addColumn(CF1, QUA_ARTICLE_ID);

       // Set the cache size.
       scan.setCaching(1000);

       // with special filter
       Map<String, String> httpParameters = new HashMap<>();
       httpParameters.put("q", "contentCh:***replace_with_Chinese_keywords_01***");
       List<String> indexNames = new ArrayList();
       indexNames.add(ES_INDEX_NAME);
       ESColumnValueFilter filter = new ESColumnValueFilter(indexNames, httpParameters, "");
       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("testScanDataWithES1 by text type condition, scan result:" +
Bytes.toString(CellUtil.cloneRow(cell)) + ":"
          + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
          + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
          + convertCellValue(Bytes.toString(CellUtil.cloneQualifier(cell)), 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 testScanDataWithES1.");
 }
```

如下样例代码通过reqBodyJson来查询（请将代码中的字符串“***replace_with_Chinese_keywords_02******”替换为中文检索词）：

```
 public void testScanDataWithES2() {
   LOG.info("Entering testScanDataWithES2.");

   Table table = null;
   // Instantiate a ResultScanner object.
   ResultScanner rScanner = null;
   try {
     // Create the Configuration instance.
     table = conn.getTable(tableName);
     assert table instanceof LemonWrapperHTable;

     // set specified qualifier.
     Scan scan = new Scan();
     scan.addColumn(CF1, QUA_ARTICLE_CONTENT_CHINESE);
     scan.addColumn(CF2, QUA_ARTICLE_CONTENT_English);
     scan.addColumn(CF1, QUA_ARTICLE_ID);

     // Set the cache size.
     scan.setCaching(1000);

     // with special filter
     Map<String, String> httpParameters = Collections.emptyMap();
     List<String> indexNames = new ArrayList();
     indexNames.add(ES_INDEX_NAME);
     String reqBodyJson =
       "{" +
         "    \"query\" : {" +
         "        \"term\" : { \"contentCh\" : \"***replace_with_Chinese_keywords_02***\" }" +
         "    }" +
         "}";
     ESColumnValueFilter filter = new ESColumnValueFilter(indexNames, httpParameters,
reqBodyJson);
     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("testScanDataWithES2 by text type condition, scan result:" +
Bytes.toString(CellUtil.cloneRow(cell)) + ":"
          + Bytes.toString(CellUtil.cloneFamily(cell)) + ","
          + Bytes.toString(CellUtil.cloneQualifier(cell)) + ","
          + convertCellValue(Bytes.toString(CellUtil.cloneQualifier(cell)), 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 testScanDataWithES2.");
 }
```

7 调测程序

7.1 在 Windows 中调测程序

7.1.1 编译并运行程序

操作场景

在程序代码完成开发后，您可以在Windows开发环境中运行应用。

操作步骤

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。

7.1.2 查看调测结果

运行结果中没有异常或失败信息即表明运行成功。

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
```


7.2 在 Linux 中调测程序

7.2.1 安装客户端时编译并运行程序

操作场景

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

前提条件

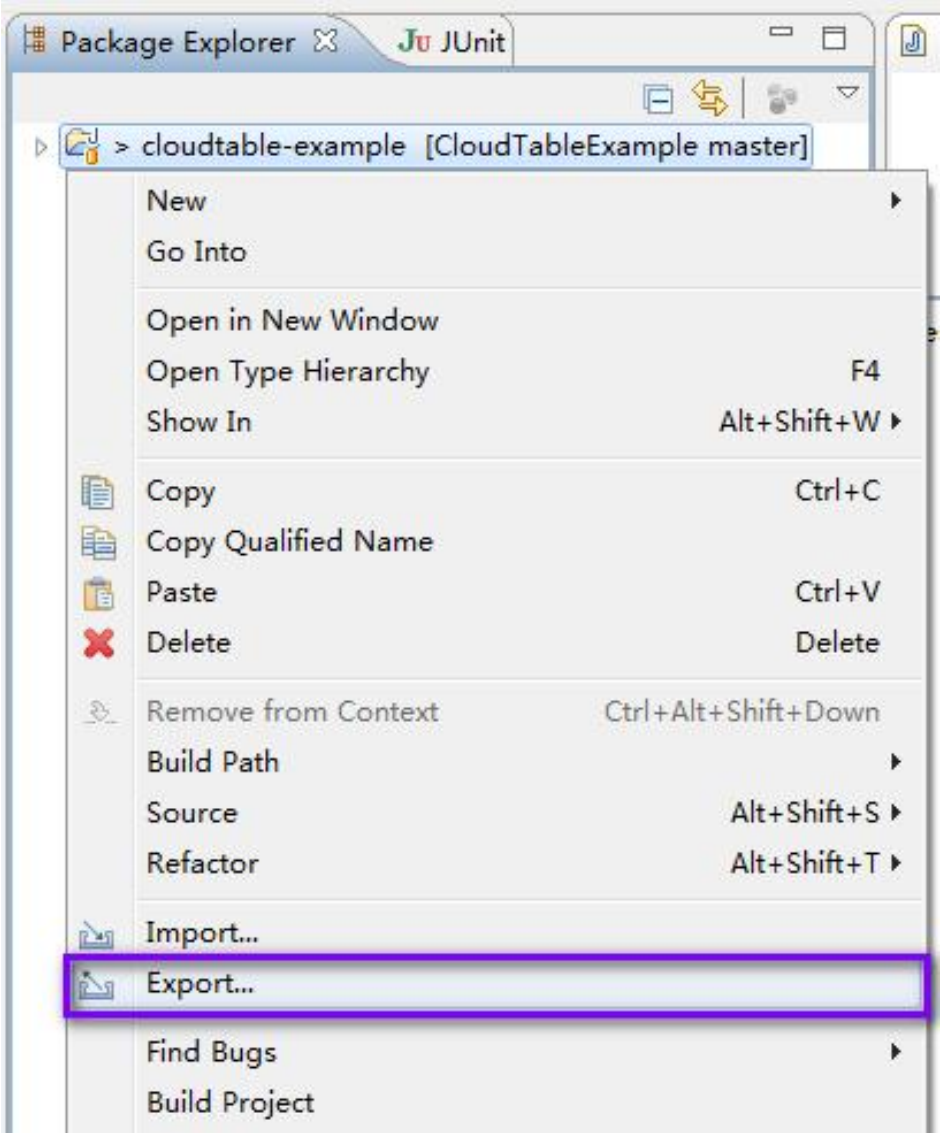
- 已安装HBase客户端。
- Linux环境已安装JDK，版本号需要和Eclipse导出Jar包使用的JDK版本一致。

操作步骤

步骤1 导出Jar包。

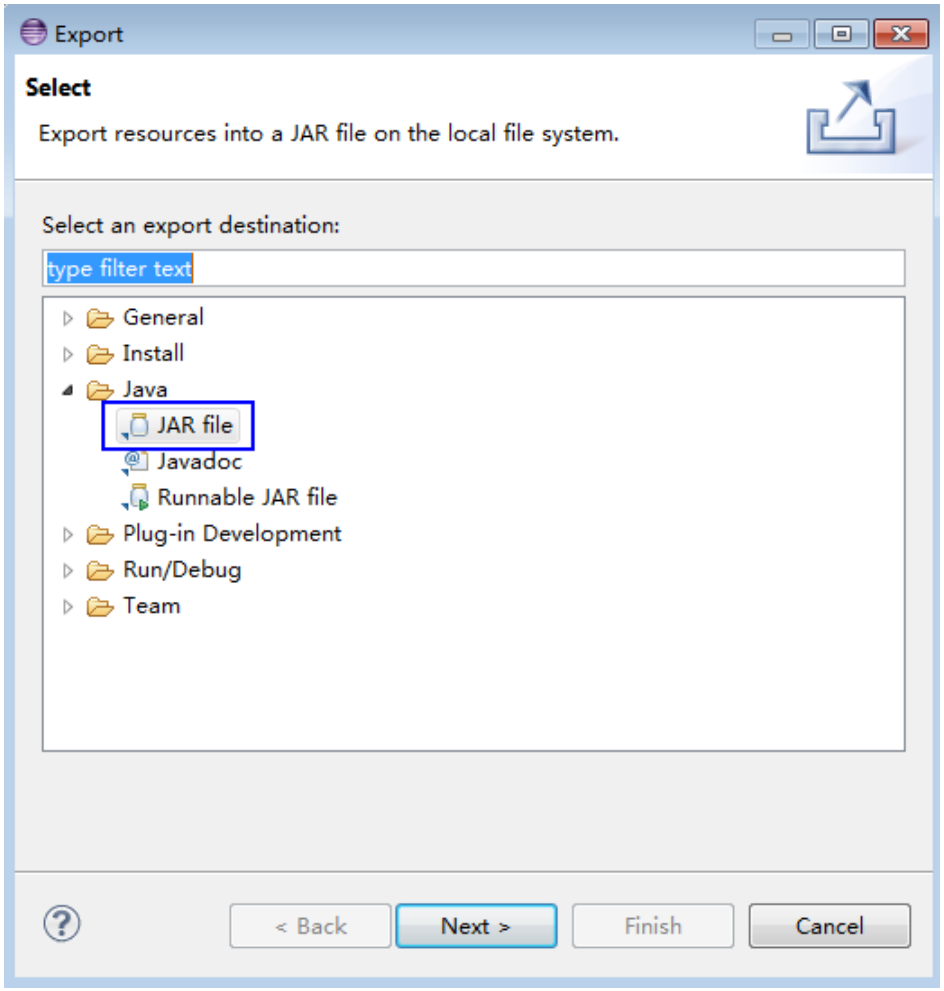
1. 右击样例工程，选择导出。

图 7-1 导出 Jar 包



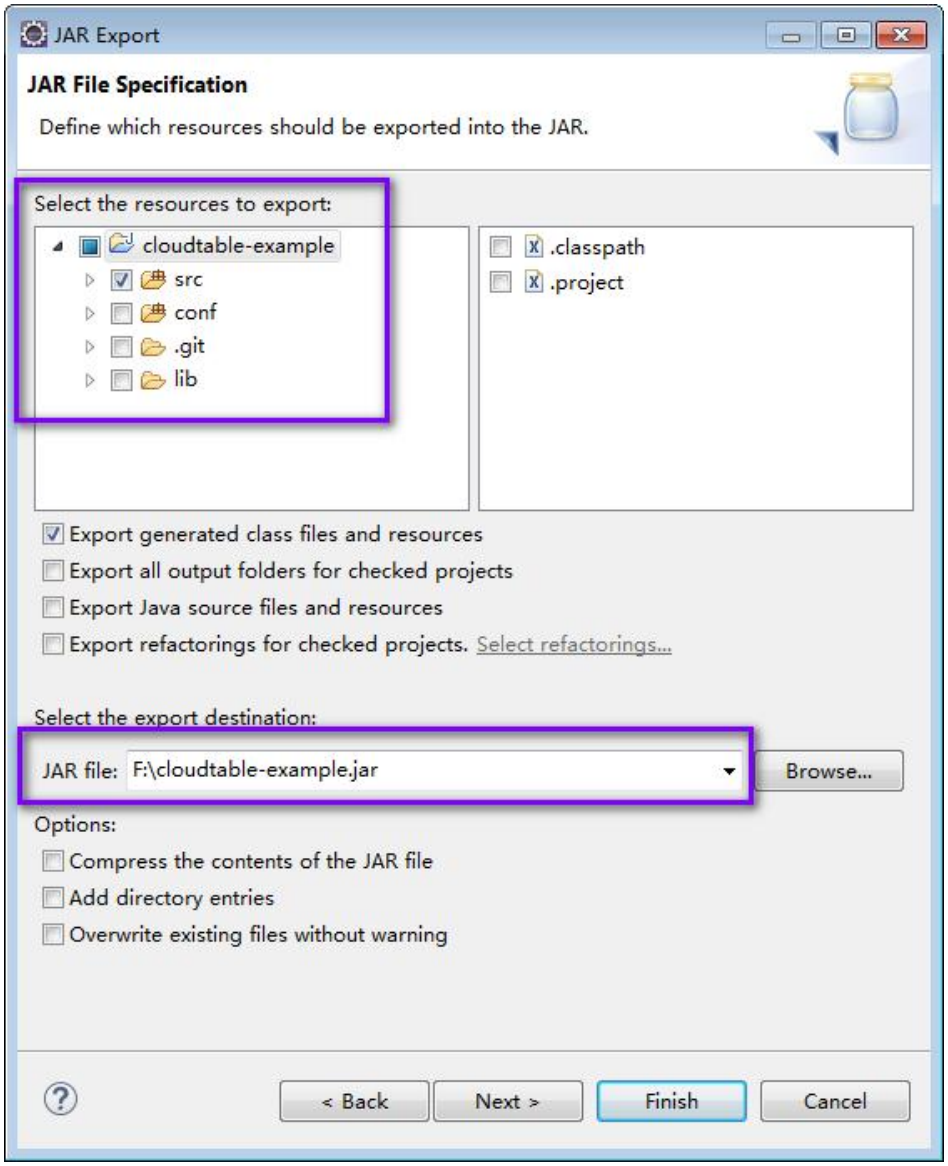
2. 选择JAR file，单击“Next”。

图 7-2 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 7-3 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

7.2.2 未安装客户端时编译并运行程序

操作场景

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

前提条件

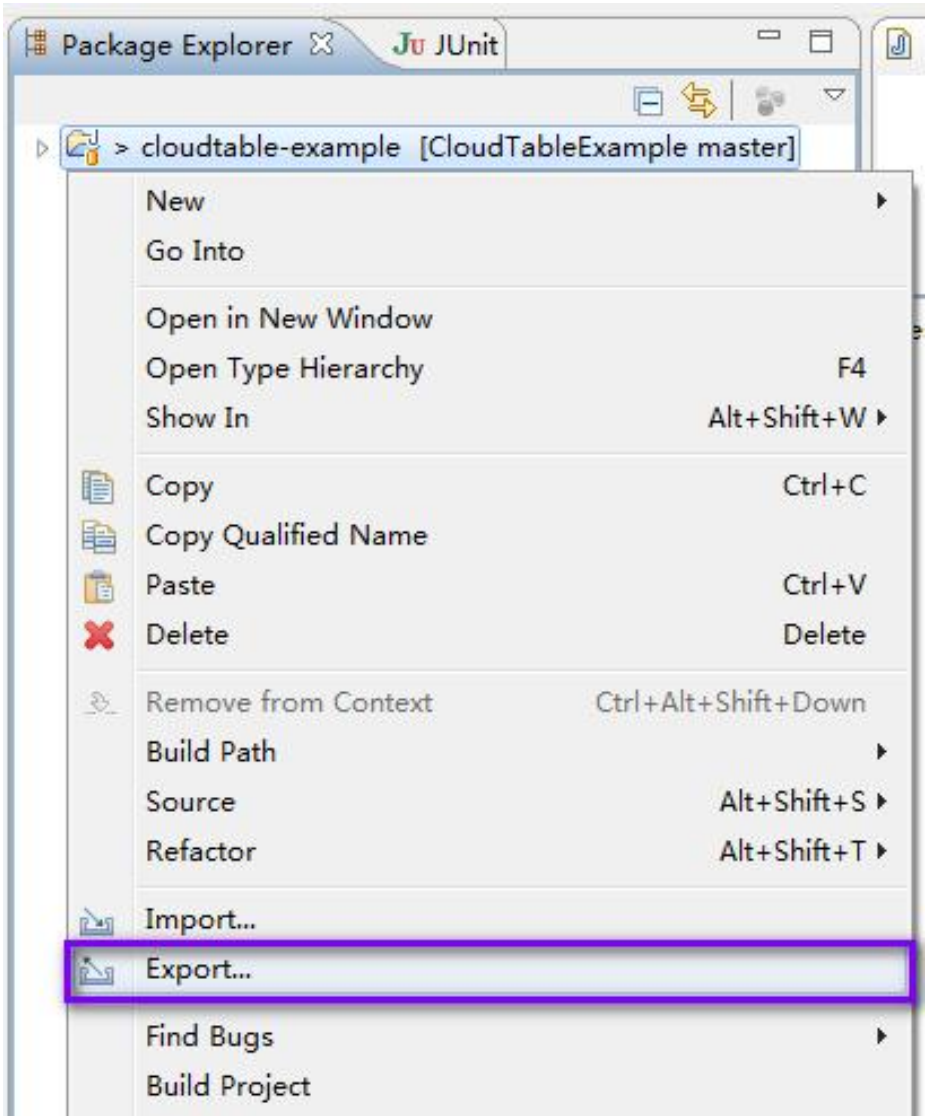
- Linux环境已安装JDK，版本号需要和Eclipse导出Jar包使用的JDK版本一致。

操作步骤

步骤1 导出Jar包。

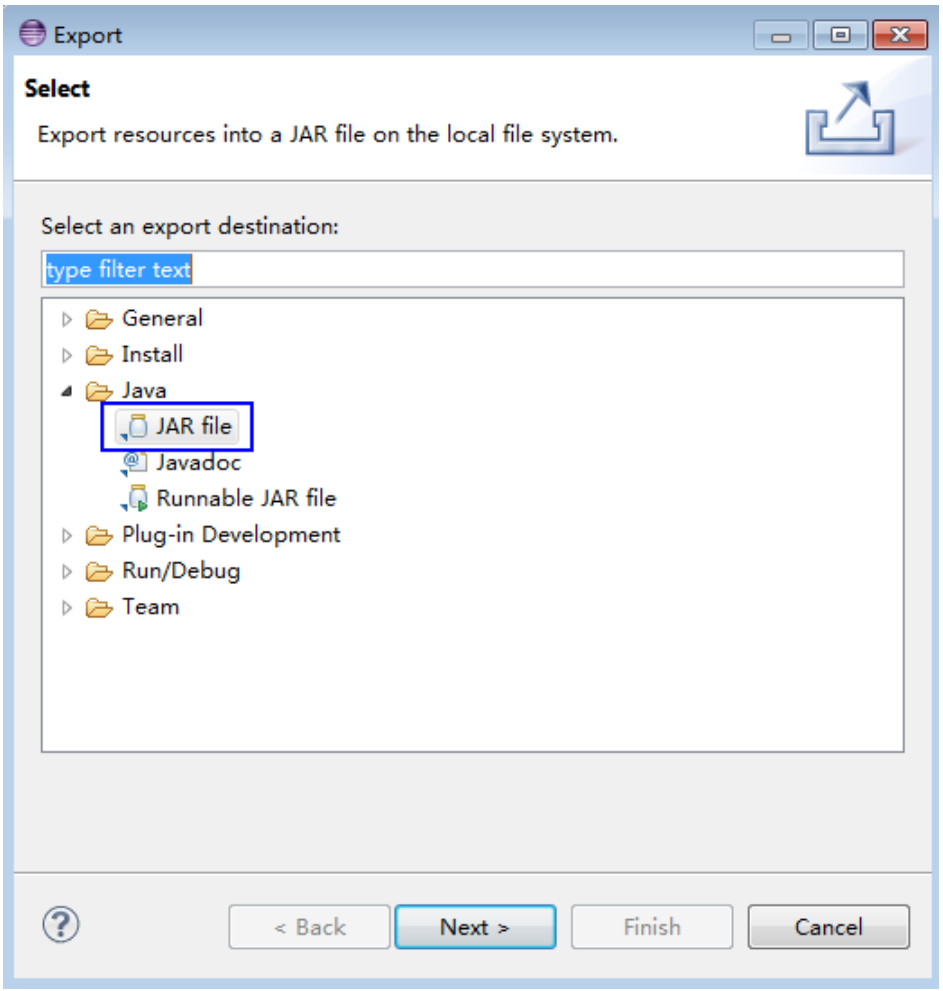
1. 右击样例工程，选择导出。

图 7-4 导出 Jar 包



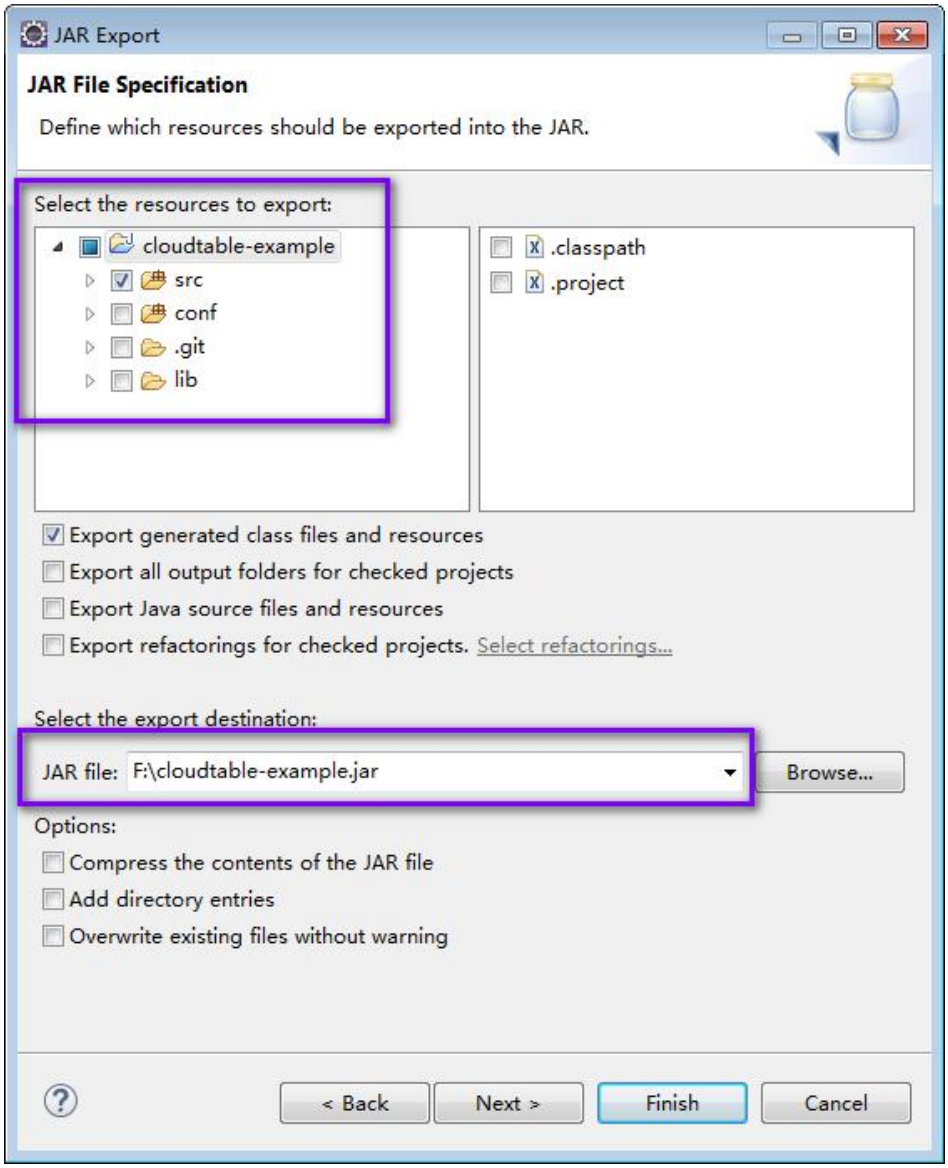
2. 选择JAR file，单击“Next”。

图 7-5 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 7-6 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`
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} com.huawei.cloudtable.hbase.examples.TestMain
```


步骤3 切换到“/opt/test”，执行以下命令，运行Jar包。

sh run.sh

----结束

7.2.3 查看调测结果

运行结果中没有异常或失败信息即表明运行成功。

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
```


8 对外接口

8.1 HBase Java API

HBase采用的接口与Apache HBase保持一致，请参见 **https://hbase.apache.org/1.2/apidocs/index.html**

8.2 OpenTSDB API

8.2.1 OpenTSDB API 简介

OpenTSDB提供了基于HTTP或HTTPS的应用程序接口。请求方式是通过向资源对应的路径发送标准的HTTP请求，请求包含GET、POST方法。它的接口与开源OpenTSDB保持一致，请参见**https://opentsdb.net/docs/build/html/api_http/index.html**。

请求以及响应实体的类型为：application/JSON

请求以及响应实体的编码为：ISO-8859-1

说明

HTTP协议本身有安全风险，HTTPS是安全协议，建议使用HTTPS连接方式。

OpenTSDB的常用API主要有：

- **写入数据**
- **查询数据**
- **查询first数据**
- **查询last数据**
- **管理数据生命周期**

获取 OpenTSDB 链接地址

在调用OpenTSDB API之前，请先按照如下步骤获取OpenTSDB链接地址：

登录表格存储服务管理控制台，在左侧导航树单击“集群模式”，选中需要访问的集群并单击集群名称，进入集群基本信息页面，在“集群信息”区域下方找到“OpenTSDB链接地址”，如下图所示：

图 8-1 OpenTSDB 链接地址

8.2.2 写入数据

功能介绍

向OpenTSDB数据库中写入数据，支持在一个请求中将多个数据写入OpenTSDB，多个数据之间可以不相关，每个数据都会被单独处理，其中一条数据出现错误不会影响其他数据的写入。但如果一个请求中有大量数据点，则API可能需要很长时间才能响应。

URI

- URI格式
  a. 写入数据
     POST {OpenTSDB URL}/api/put
  b. 写入数据并返回概要信息
     POST {OpenTSDB URL}/api/put?summary
  c. 写入数据并返回详细信息
     POST {OpenTSDB URL}/api/put?details

     说明

     如果summary和details标志同时存在于查询字符串，该API将响应detailed信息。
  d. 写入数据并等待数据刷入磁盘
     POST {OpenTSDB URL}/api/put?sync

     说明

     强烈建议使用sync参数，否则API将不会等待数据写入成功即返回响应，存在数据丢失的风险。
  e. 写入数据等待数据刷入磁盘，并设置超时时间（毫秒）。当发生超时时，使用details标志将会返回成功和失败的数据点数量。
     POST {OpenTSDB URL}/api/put?sync&sync_timeout=60000

请求

- 请求样例：单数据点写入

```
{
    "metric": "sys.cpu.nice",
    "timestamp": 1346846400,
```



```
  "value": 18,
  "tags": {
    "host": "web01",
    "dc": "lga"
  }
}
```

- 请求样例：多数据点写入

```
[
  {
    "metric": "sys.cpu.nice",
    "timestamp": 1346846400,
    "value": 18,
    "tags": {
      "host": "web01",
      "dc": "lga"
    }
  },
  {
    "metric": "sys.cpu.nice",
    "timestamp": 1346846400,
    "value": 9,
    "tags": {
      "host": "web02",
      "dc": "lga"
    }
  }
]
```

- 参数说明

表 8-1 请求参数说明

属性名	类型	是否必须	描述	限制
metric	String	是	指标项名	• 只能包含大小写英文字母，数字，“-”，“_”，“.”，“/”这些Unicode字符。 • 不允许包含空格及其它字符。 • 区分大小写。
timestamp	Integer	是	时间戳，单位：秒	• 以秒为单位的Unix/POSIX Epoch时间戳，从1970年1月1日00:00:00 UTC时间起经过的秒数。 **说明** 时间戳建议使用4334400秒到4291718400秒之间的时间，即从1970/02/20 12:00:00到2106/01/01 00:00:00。 • 必须是整数。 • 不能超过13位数。

属性名	类型	是否必须	描述	限制
value	Integer,Long,Double,Boolean	是	数据值	整数，单精度浮点数（Float），双精度（Double）浮点数或者布尔值。
tags	Map	是	Tagk和Tagv的键值对	• 只能包含大小写英文字母，数字，“-”，“_”，“.”，“/”这些Unicode字符。 • 不允许包含空格及其它字符。 • 区分大小写。 • 最少1个，最多8个Tagk和TagV的键值对。

响应

- 响应样例： summary

```
{
   "failed": 1,
   "success": 0
}
```

响应样例：details

```
{
   "errors": [
      {
         "datapoint": {
            "metric": "sys.cpu.nice",
            "timestamp": 1365465600,
            "value": "NaN",
            "tags": {
               "host": "web01"
            }
         },
         "error": "Unable to parse value to a number"
      }
   ],
   "failed": 1,
   "success": 0
}
```

- 参数说明

表 8-2 返回信息的属性

名称	类型	描述
success	Integer	写入成功的数据点数量
failed	Integer	写入失败的数据点数量
errors	Array	写入失败的具体数据点的值及原因，仅在details参数下生效

状态码

状态码请参见**响应码**

8.2.3 查询数据

功能介绍

从OpenTSDB数据库中查询数据。

URI

- URI格式
  POST {OpenTSDB URL}/api/query

请求

- 请求样例

```
{
   "start": 1504527820,
   "end": 1504557820,
   "queries": [
      {
         "aggregator": "sum",
         "metric": "cpu.system",
         "rate": "true",
         "filters": [
            {
               "type":"regexp",
               "tagk":"host",
               "filter":"web[0-9]+.lax.mysite.com",
               "groupBy":true
            },
            {
               "type":"literal_or",
               "tagk":"dc",
               "filter":"lax|dal",
               "groupBy":false
            },
         ]
      }
   ]
}
```

- 参数说明

表 8-3 请求参数说明

名称	类型	是否必须	描述
start	Integer	是	起始时间，单位秒。查询结果包含该时间的值。 **说明** 建议使用4334400秒到4291718400秒之间的时间，即从1970/02/20 12:00:00到2106/01/01 00:00:00，也可以为0。否则可能导致查询结果不正确。

名称	类型	是否必须	描述
end	Integer	否	结束时间，单位秒，默认值为OpenTSDB的当前系统时间。查询结果包含该时间的值。 **说明** 建议使用4334400秒到4291718400秒之间的时间，即从1970/02/20 12:00:00到2106/01/01 00:00:00。否则可能导致查询结果不正确。
queries	Array	是	可以有多个查询，请参见表8-4。
delete	Boolean	否	如果为true,符合条件的查询结果数据都会被删除。 **说明** 删除数据是以一小时的数据为单位的。即查询结果关联的所有小时的数据都会被删除。
noAnnotations	Boolean	否	是否返回注释信息。 ● true：不返回。 ● false：返回。 默认为false。
globalAnnotations	Boolean	否	是否返回时间跨度内的全局注释。 ● true：返回。 ● false：不返回。 默认为false。
showTSUIDs	Boolean	否	是否在结果中返回与时间序列关联的TSUID。 ● true：如果多个时间序列被聚合成一个集合，则多个TSUID将以排序方式返回。 ● false：不返回。 默认为false。
showSummary	Boolean	否	是否在结果中返回时间摘要，详细信息请参见https://opentsdb.net/docs/build/html/user_guide/query/stats.html ● true：返回。 ● false：不返回。 默认为false。

名称	类型	是否必须	描述
showStats	Boolean	否	是否在结果中返回详细时间，详细信息请参见https://opentsdb.net/docs/build/html/user_guide/query/stats.html ● true：返回。 ● false：不返回。 默认为false。
showQuery	Boolean	否	是否返回带查询结果的原始子查询。 ● true：返回。 ● false：不返回。 默认为false。
msResolution	Boolean	否	默认情况下，查询结果中的时间戳是以秒为单位的。如果设置为true，则查询结果中的时间戳以毫秒为单位。
returnCount	Boolean	否	是否返回查询结果的数量。 ● true：返回。 ● false：不返回。 默认值为false。
reverse	Boolean	否	是否以时间倒序的方式返回查询结果。 ● true：是。 ● false：否。 默认值为false。
onlyDelete	Boolean	否	● true：符合查询条件的结果数据都会被删除，但不返回查询结果。 ● false：该参数无效。 默认值为false。
returnBoolean	Boolean	否	是否将查询返回的“value”数据值转换为布尔值。 ● true：是，如果“value”数据值大于0，则转换为true，如果“value”数据值等于或小于0，则转换为false。 ● false：否。 默认值为false。

表 8-4 子查询参数说明

名称	类型	是否必须	描述
aggregator	String	是	聚合函数，请参见**•aggregator说明**。
metric	String	是	指标项。
rate	Boolean	否	是否返回倾斜率。 ● true：返回。 ● false：不返回。 默认为false。
downsample	String	否	降时间精度采样，请参见**•downsample说明**。
filters	List	否	过滤器，可以设置多个过滤器，每个过滤器的格式请参见**•filter参数说明**。
explicitTags	Boolean	否	是否只返回filter指定的tag。 ● true：返回。 ● false：不返回。 默认为false。
useMultiGets	Boolean	否	是否使用MultiGet方式查询数据，默认为false。只有当“filters”的“type”为“literal_or”，同时“explicitTags”为“true”时才会起作用。
valueFilter	String	否	指定值过滤的表达式。请参见**•valueFilter说明**。

- **downsample说明**

  当查询的时间跨度很大，例如每秒都将温度作为数据写入OpenTSDB，每个小时会产生3600条数据点，当查询一周的数据时，会返回604800个数据点。展示如此多数据会显得很乱，通常也不需要这样精确的数据。使用降精度的方式将一段时间的数据点聚合后当作一个数据点，比如将每个小时的数据聚合为1个数据点，这样就会只显示168个数据点。

  格式：

  ```
  <Interval><units>-<aggregator>[c][-<fill policy>]
  ```

  - “Interval”：时间数值，“units”为时间单位，s秒，m分，h小时，d天。示例：1h， 30m， 24h
  - “aggregator”：聚合策略，将一段时间点聚合为一个数据点的策略。参见**•aggregator说明**。
  - “fill policy”：补值策略，当使用aggregator计算一段时间内的汇聚值时，遇到中间缺少的数据点时，会使用一定的策略补充数据。补值策略请参见**表 8-5**。

表 8-5 补值策略参数

名称	描述
none	默认，不补值
nan	补NaN
null	补null
zero	补0
pre	补前值（聚合后）

说明

downsample是先对时间线的数据聚合成为一个新的时间线后，再对查询条件的start和end时间进行过滤。

- **aggregator说明**

  aggregator在降精度downsample和多条时间线聚合时使用。通过算子将多个数据点汇聚成一个数据点。汇聚的算子请参见表8-6。

表 8-6 算子

算子	描述	补值方式
avg	平均值	线性插值
count	数据点数	补0
dev	计算标准偏差	线性插值
ep50r3	用R-3方法计算的50%都大于	线性插值
ep50r7	用R-7方法计算的50%都大于	线性插值
ep75r3	用R-3方法计算的75%都大于	线性插值
ep75r7	用R-7方法计算的75%都大于	线性插值
ep90r3	用R-3方法计算的90%都大于	线性插值
ep90r7	用R-7方法计算的90%都大于	线性插值
ep95r3	用R-3方法计算的95%都大于	线性插值
ep95r7	用R-7方法计算的95%都大于	线性插值

算子	描述	补值方式
ep99r3	用R-3方法计算的99%都大于	线性插值
ep99r7	用R-7方法计算的99%都大于	线性插值
ep999r3	用R-3方法计算的99.9%都大于	线性插值
ep999r7	用R-7方法计算的99.9%都大于	线性插值
first	取第一个值	-
last	取最后一个值	-
mimmin	最小值	补最大值
mimmax	最大值	补最小值
min	最小值	线性插值
max	最大值	线性插值
none	不做计算	补0
p50	50%的值都大于	线性插值
p75	75%的值都大于	线性插值
p90	90%的值都大于	线性插值
p95	95%的值都大于	线性插值
p99	99%的值都大于	线性插值
p999	99.9%的值都大于	线性插值
sum	求和	线性插值
zimsum	求和	补0

说明

R-3，R-7 method见https://en.wikipedia.org/wiki/Quantile。

- **filter参数说明**

表 8-7 filter 参数

名称	类型	是否必须	描述	示例
type	String	是	filter类型，参见表8-8	regexp

名称	类型	是否必须	描述	示例
tagk	String	是	要做filter的tag名	host
filter	String	是	filter的表达式值	web[0-9]+.lax.mysite.com
groupBy	Boolean	否	是否对tagv做groupBy，默认false	false

表 8-8 file type 参数

名称	说明	示例
literal_or	tagv等于，区分大小写	{"type":"literal_or","tagk":"host","filter":"web01|web02|web03","groupBy":false}
iliteral_or	tagv等于，不区分大小写	{"type":"iliteral_or","tagk":"host","filter":"web01|web02|web03","groupBy":false}
not_literal_or	tagv不等于，区分大小写	{"type":"not_literal_or","tagk":"host","filter":"web01|web02|web03","groupBy":false}
not_iliteral_or	tagv不等于，不区分大小写	{"type":"not_iliteral_or","tagk":"host","filter":"web01|web02|web03","groupBy":false}
wildcard	tagv需满足通配符表达式，区分大小写	URI中：host=wildcard(*mysite.com) {"type":"wildcard","tagk":"host","filter":"web*.tsdb.net","groupBy":false}
iwildcard	tagv需满足通配符表达式，不区分大小写	{"type":"iwildcard","tagk":"host","filter":"web*.tsdb.net","groupBy":false}
regexp	tagv需满足正则表达式	{"type":"regexp","tagk":"host","filter":".*","groupBy":false}
not_key	tagk不等于	{"type":"not_key","tagk":"host","filter":"","groupBy":false}

- **valueFilter说明**

  根据设置的数值条件表达式，对最终的返回数据点进行过滤。支持 ">", "<", "=", "<=", ">=", "!=","&&","||"等操作符。

  其中 ">", "<", "=", "<=", ">=", "!=",都是单目操作符，用于操作数字；"&&","||"是双目操作符，用于操作表达式。

  表达式例子如下：

a. 只返回value大于或等于10的的结果：>=10

b. 只返回value小于5或者大于10的结果：<5||>10

c. 只返回value大于等于6并且不等于11的结果：>=6&&!=11

说明

当"&&"与"||"在一个表达式中同时存在时，"&&"比"||"的优先级高。即表达式">=6||>11&&!=15"相当于">=6||(>11&&!=15)"。

响应

- 响应样例

```
[
  {
    "metric": "tsd.hbase.puts",
    "tags": {
      "host": "tsdb-1.mysite.com"
    },
    "aggregatedTags": [],
    "dps": {
      "1365966001": 3758788892,
      "1365966061": 3758804070,
...
      "1365974281": 3778141673
    }
  },
  {
    "metric": "tsd.hbase.puts",
    "tags": {
      "host": "tsdb-2.mysite.com"
    },
    "aggregatedTags": [],
    "dps": {
      "1365966001": 3902179270,
      "1365966062": 3902197769,
...
      "1365974281": 3922266478
    }
  }
]
```

- 参数说明

表 8-9 响应参数说明

名称	描述
metric	指标名
tags	未做聚合的tagv，如果存在聚合，这个值为空。
aggregateTags	如果存在聚合，显示做聚合操作的tagv。
dps	数据点对，数据点由时间戳和值组成并做了序列化。如果查询请求中设置了“returnCount”为“true”，那么“dps”不会返回。 **说明** 如果值是浮点数，那么是以双精度（Double）方式进行显示，可根据需要转换为单精度（Float）或双精度（Double）浮点数。
annotations	注释信息

名称	描述
globalAnnotations	时间跨度内的全局注释
count	如果查询请求中设置了“returnCount”为“true”，那么“count”才会返回，表示查询结果的数量。

状态码

状态码请参见**响应码**。

8.2.4 查询 first 数据

功能介绍

指定Metric查询时间戳最早的一条DataPoint。

如果该Metric最早的时间戳有多条不同tag的DataPoint，查询只返回一条DataPoint，时间戳单位是毫秒。

URI

- URI格式

  POST {OpenTSDB URL}/api/query/first

请求

- 请求样例

```
{
   "resolveNames":true,
   "backScan":20,
   "queries":[
      {
         "metric":"sys.cpu.nice"
      },
      {
         "metric":"cpu.system"
      }
   ]
}
```

- 参数说明

表 8-10 请求参数说明

名称	类型	是否必须	描述
resolveNames	Boolean	否	是否将返回结果的tsuid转换为对应的metric、Tagk和Tagv名称。 ● true：转换 ● false：不转换

名称	类型	是否必须	描述
backScan	Integer	否	设置需要扫描过去几个小时的数据，单位：小时。 数据存储以小时为单位，假设系统当前时间是2018/07/20 19:30:00，“backScan”设置为2，则查询的时间范围是2018/07/20 17:00:00到2106/01/01 00:00:00，而不是从2018/07/20 17:30:00开始，数据扫描会从2018/07/20 17:00:00开始直到找到第一条数据。 如果“backScan”不设置或者值为0，表示扫描所有时间。
queries	Array	是	需要查询的metric列表，不支持指定Tag和tsuid，可以有多个子查询。

响应

- 响应样例

```
[
  {
    "metric": "sys.cpu.nice",
    "timestamp": 1346846400000,
    "value": "18",
    "tags": {
       "host": "web01",
       "dc": "lga"
    },
    "tsuid": "00000E0000090007E500000A0007E6"
  },
  {
    "metric": "cpu.system",
    "timestamp": 1346846400000,
    "value": "9",
    "tags": {
       "host": "web02",
       "dc": "lga"
    },
    "tsuid": "00000F0000090007E700000A0007E6"
  }
]
```

- 参数说明

表 8-11 响应参数说明

名称	类型	描述
metric	String	指标名称
timestamp	long	时间戳，单位：毫秒
value	String	数据值
tags	Map	Tagk和Tagv的键值对
tsuid	String	metric、Tagk和Tagv对应的tsuid

状态码

状态码请参见响应码

8.2.5 查询 last 数据

功能介绍

指定Metric或tsuid查询时间戳最新的一条DataPoint。

如果该Metric或tsuid最新的时间戳有多条不同tag的DataPoint，查询只返回一条DataPoint，时间戳单位是毫秒。

URI

- URI格式

  POST {OpenTSDB URL}/api/query/last

请求

- 请求样例（指定Metric）

```
{
   "resolveNames":true,
   "backScan":20,
   "queries":[
      {
         "metric":"sys.cpu.nice"
      },
      {
         "metric":"cpu.system"
      }
   ]
}
```

- 请求样例（指定Metric和Tag）

```
{
   "resolveNames":true,
   "backScan":20,
   "queries":[
        {
      "metric":"sys.cpu.nice",
      "tags": {
            "host": "web01",
            "dc": "lga"
        }
       },
       {
      "metric":"cpu.system",
      "tags": {
            "host": "web01",
            "dc": "lga"
        }
       }
   ]
}
```

- 请求样例（指定tsuid）

```
{
   "resolveNames":true,
   "backScan":20,
   "queries":[
```

```
        {
           "tsuids": [
                      "00000E0000090007E500000A0007E6",
                      "00000F0000090007E500000A0007E6"
                    ]
        }
   ]
}
```

- 参数说明

表 8-12 请求参数说明

名称	类型	是否必须	描述
resolveNames	Boolean	否	是否将返回结果的tsuid转换为对应的metric、Tagk和Tagv名称。 ● true：转换 ● false：不转换
backScan	Integer	否	设置需要扫描过去几个小时的数据，单位：小时。 数据存储以小时为单位，假设系统当前时间是2018/07/20 19:30:00，“backScan”设置为2： ● 如果“queries”的子查询中只指定了“metric”，没有指定“tags”或“tsuids”，则该子查询的时间范围是2018/07/20 17:00:00到2106/01/01 00:00:00，数据扫描会从2106/01/01 00:00:00开始往前直到找到最新的一条数据。 ● 如果“queries”的子查询中同时指定了“metric”和“tags”，或指定了“tsuids”，则该子查询的时间范围是2018/07/20 17:00:00到当前时间2018/07/20 19:30:00，数据扫描会从2018/07/20 19:30:00开始往前直到找到最新的一条数据。 ● 如果“backScan”不设置或者值为0，表示扫描所有时间。
queries	Array	是	需要查询的metric、tag、tsuid列表，可以有多个子查询。请参见表8-13。

表 8-13 queries 参数说明

名称	类型	是否必须	描述
metric	String	否	指标名称

名称	类型	是否必须	描述
tags	Map	否	Tagk和Tagv的键值对
tsuids	List	否	metric、Tagk和Tagv对应的tsuid

说明

“queries”的子查询中，“metric”和“tsuids”必须要有一个。

响应

- 响应样例

```
[
  {
    "metric": "sys.cpu.nice",
    "timestamp": 1346846400000,
    "value": "18",
    "tags": {
       "host": "web01",
       "dc": "lga"
    },
    "tsuid": "00000E0000090007E500000A0007E6"
  },
  {
    "metric": "cpu.system",
    "timestamp": 1346846400000,
    "value": "9",
    "tags": {
       "host": "web02",
       "dc": "lga"
    },
    "tsuid": "00000F0000090007E700000A0007E6"
  }
]
```

- 参数说明

表 8-14 响应参数说明

名称	类型	描述
metric	String	指标名称
timestamp	long	时间戳，单位：毫秒
value	String	数据值
tags	Map	Tagk和Tagv的键值对
tsuid	String	metric、Tagk和Tagv对应的tsuid

状态码

状态码请参见**响应码**

8.2.6 管理数据生命周期

功能介绍

针对Metric设置OpenTSDB数据库中数据生命周期TTL（Time To Live）时间，系统会定期清理“DataPoint时间戳 < 系统当前时间 – TTL时间”的数据。

TTL时间单位为小时，数据清理时也是以小时为单位。例如，系统当前时间是19:30，某个Metric的TTL时间为2，则17:00之前的数据才会被清理，而不是17:30之前的数据。

- 如果某个Metric没有设置TTL时间，则使用全局TTL时间，如未设置全局TTL时间，数据不会被清理。如果设置了Metric的TTL，就使用Metric自己的TTL时间。
- 如果某个Metric的TTL时间为0，表示该Metric未设置TTL，数据不会被清理。

URI

- URI格式
  - 设置TTL
    PUT {OpenTSDB URL}/api/ttl
  - 查询TTL
    POST {OpenTSDB URL}/api/ttl
  - 删除TTL
    DELETE {OpenTSDB URL}/api/ttl

请求

- **请求样例：设置TTL**

```
{
   "global":36,
   "ttl":{
      "sys.cpu.nice":24,
      "cpu.system":12
   }
}
```

参数说明如下表所示，“global”和“ttl”参数至少需指定一个。

表 8-15 参数说明

名称	类型	是否必须	描述
global	Integer	否	全局TTL时间，单位为小时。表示未设置TTL的Metric数据有效期统一设置为“global”的取值。如果“global”取值为0，表示全局TTL无效。
ttl	Map	否	设置TTL的Metric列表。采用Key-value格式，key为Metric名称，value为TTL时间，单位为小时。 例如："sys.cpu.nice": 24，表示设置Metric名为sys.cpu.nice的TTL时间为24小时。

- **请求样例：查询TTL**

  查询指定Metric的TTL：

  ```
  {
        "metrics":["sys.cpu.nice", "cpu.system"]
  }
  ```

  查询所有设置TTL的Metric和全局TTL：

  ```
  {
        "all":true
  }
  ```

  参数说明如下表所示，以下三个参数至少指定一个，如果指定了“metrics”至少要指定一个Metric。

  表 8-16 参数说明

名称	类型	是否必须	描述
global	Boolean	否	是否查询全局TTL时间。“true”表示查询当前“global”取值。
all	Boolean	否	是否查询全部Metric的TTL时间。如果“all”为“true”，则返回所有Metric的TTL和全局TTL，“metrics”参数失效。
metrics	String/Array	否	需要查询的Metric名。设置单个Metric时，类型为String，设置多个Metric时，类型为Array。

- **请求样例：删除TTL**

  ```
  {
     "global":true,
        "metrics":["sys.cpu.nice", "cpu.system"]
  }
  ```

  参数说明如下表所示，“metrics”至少要指定一个Metric或者“global”设置为“true”。

  表 8-17 参数说明

名称	类型	是否必须	描述
global	Integer，Boolean	否	是否删除全局TTL。“true”表示删除当前“global”取值。
all	Boolean	否	是否删除全部TTL。“true”表示删除全部TTL，此时“metrics”参数失效。
metrics	String/Array	否	需要删除的Metric名。设置单个Metric时，类型为String，设置多个Metric时，类型为Array。

响应

- **响应样例： 查询TTL**

```
{
    "global": 36,
    "ttl": {
        "sys.cpu.nice": 24,
        "cpu.system": 12
    }
}
```

表 8-18 响应参数说明

名称	类型	描述
global	Integer	全局TTL时间，单位为小时。表示未设置TTL的Metric数据有效期统一为“global”的取值。如果“global”取值为0，表示全局TTL无效。
ttl	Map	TTL的Metric列表。采用Key-value格式，key为Metric名称，value为TTL时间，单位为小时。 例如："sys.cpu.nice": 24，表示Metric名为sys.cpu.nice的TTL时间为24小时。

- **响应样例：设置或删除TTL**

  body体无内容返回。

状态码

状态码请参见**响应码**

8.2.7 响应码

每个请求都将被返回一个标准的HTTP响应码，许多响应还会包含内容，特别是错误代码。

表 8-19 响应成功码

响应码	描述信息
200	请求成功
204	请求成功但没有内容返回，主要用于写数据
301	API请求被转发到其他服务器

表 8-20 响应失败码

响应码	描述信息
400	请求失败，请求参数错误或者缺失，返回内容里包含具体信息
404	请求路径未找到

响应码	描述信息
405	请求方法不允许
406	请求无法以指定格式获得相应
408	请求超时
413	返回结果超过服务器缓存
500	服务器内部错误
501	功能未实现
503	服务器资源不足

表 8-21 返回错误响应

属性名	类型	是否必须	描述	示例
code	Integer	是	HTTP的响应码	400
message	String	是	错误简述	Missing required parameter
details	String	否	错误详细描述	Missing value: type

所有的错误响应都会返回HTTP状态码及响应内容。

示例：

```
{
  "error": {
  "code": 400,
  "message": "Missing parameter <code>type</code>"
  }
}
```

8.2.8 使用限制

使用限制

- 系统中metric、tagk和tagv都有数量限制，每种最多可创建16777215个。
- 查询时延受查询条件和时间范围影响，如果存在聚合且数据量很多，则耗时也越长。
- 谨慎使用返回数据量非常大的查询，会造成查询时较长影响服务使用，从发起查询操作以直至返回结果期间，无法手工终止请求。
- http请求的参数内容最大值为32M。
- 写入数据和查询数据时的时间戳建议使用4334400秒到4291718400秒之间的时间，即从1970/02/20 12:00:00到2106/01/01 00:00:00，否则可能导致查询结果不正确。

- 查询数据时，如果数据点的值为浮点数，查询结果以双精度浮点数（Double）的方式显示，可根据需要转换为单精度或双精度浮点数。

8.3 GeoMesa Java API

如果当前classpath中已经包含GeoMesa的代码，那么可以通过GeoTools接口获取HBase数据存储的实例。 获取HBase数据存储还要求hbase-site.xml位于classpath中，其中包含HBase数据存储的连接参数：hbase.zookeeper.quorum和hbase.zookeeper.property.clientPort。

```
Map<String, Serializable> parameters = new HashMap<>();
parameters.put("bigtable.table.name", "geomesa");
org.geotools.data.DataStore dataStore =
    org.geotools.data.DataStoreFinder.getDataStore(parameters);
```

Data Store仅包含一个参数：

- bigtable.table.name：用于存储feature类型数据的HBase表名称。

更多Java API相关接口信息可以参考GeoTools接口：http://docs.geotools.org/stable/userguide/。

说明

- LockingManager与Listener等相关接口GeoMesa由于有自身的锁管理机制以及其他原因而未实现。
- 下列修改Feature的属性值的接口当前版本不建议使用：

  ```
  FeatureStore.modifyFeatures(Name[], Object[], Filter)
  FeatureStore.modifyFeatures(AttributeDescriptor[], Object[], Filter)
  FeatureStore.modifyFeatures(Name, Object, Filter)
  FeatureStore.modifyFeatures(AttributeDescriptor, Object, Filter)
  SimpleFeatureStore.modifyFeatures(String, Object, Filter)
  SimpleFeatureStore.modifyFeatures(String[], Object[], Filter)
  ```

  可使用查询与插入Features的覆盖方式进行修改Feature的属性值：

  ```
  FeatureSource.getFeatures(Filter)
  FeatureStore.addFeatures(FeatureCollection<T, F>)
  ```


A 修订记录

发布日期	修订说明
2019-04-04	第六次正式发布。 修改如下章节： • **管理数据生命周期**
2019-03-06	第五次正式发布。 修改如下章节： • **开发流程** 新增如下章节： • **开发HBase Elasticsearch全文检索应用**
2018-09-25	第四次正式发布。 新增如下章节： • **配置参数** • **配置参数** • **配置参数** • **OpenTSDB API简介** 修改如下章节： • **查询数据**
2018-09-10	第三次正式发布。 修改如下章节： • **查询数据**

发布日期	修订说明
2018-08-03	第二次正式发布。 新增如下章节： • **IAM集群认证** • **IAM集群认证** • **查询first数据** • **查询last数据** • **管理数据生命周期**
2018-06-30	第一次正式发布。