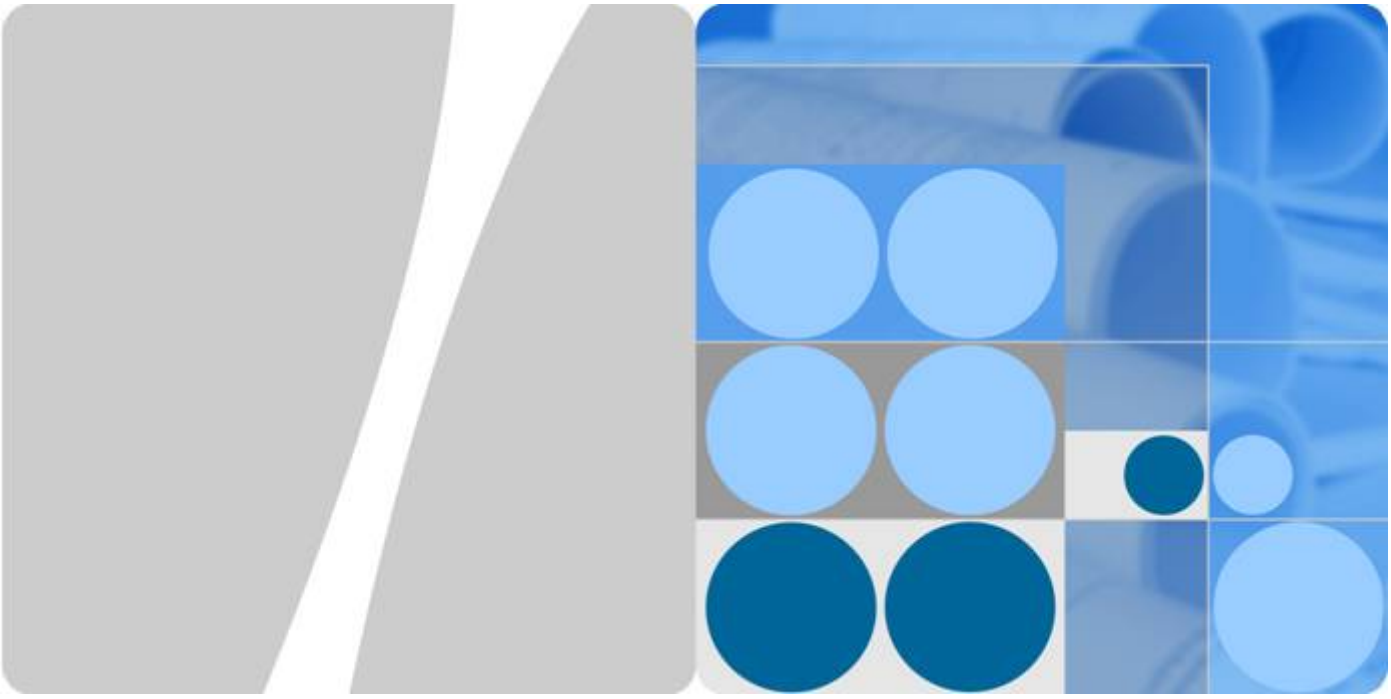

FusionInsight Hadoop

V100R002C02

HBase 开发指南

华为技术有限公司

HUAWEI

前 言

概述

本文档简要介绍如何对FusionInsight Hadoop产品进行HBase应用开发。

读者对象

本指南主要适用于具备Java开发经验的开发人员。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	以本标志开始的文本表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
 警告	以本标志开始的文本表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 注意	以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
 窍门	以本标志开始的文本能帮助您解决某个问题或节省您的时间。
 说明	以本标志开始的文本是正文的附加信息，是对正文的强调和补充。

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 01 (2013-03-30)

第一次发布。

目 录

1 概述

简介

HBase是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，其设计目标是用来解决关系型数据库在处理海量数据时的局限性。

HBase使用场景有如下几个特点：

- 海量数据（TB或PB级别以上）。
- 需要很高的吞吐量。
- 需要在海量数据中实现高效的随机读取。
- 需要很好的伸缩能力。
- 能够同时处理结构化和非结构化的数据。
- 不需要完全拥有传统关系型数据库所具备的ACID特性。

目标读者

本文档专供需要对FusionInsight Hadoop产品进行HBase应用开发的用户使用。本指南主要适用于具备Java开发经验的开发人员。

基本概念

Client

HBase Client主要包括三种方式：JAVA API、Shell、WEB UI三种方式。

- JAVA API

  提供HBase数据库客户端开发接口，本开发指南主要介绍如何使用Java API进行HBase数据库客户端应用开发。

- Shell

  提供shell命令完成HBase数据库的基本操作。

- WEB UI

  提供Web可视化组件管理界面。

keytab文件

存放用户信息的密钥文件。应用程序采用此密钥文件在FusionInsight Hadoop产品中进行API方式认证。

2 开发环境准备

操作场景

本开发指南提供了华为FusionInsight Hadoop产品HBase组件的样例代码和常用接口，便于开发者快速熟悉HBase系统。为了运行华为FusionInsight Hadoop产品HBase组件的样例代码，你需要完成下面的操作。

操作步骤

步骤1 确认华为FusionInsight Hadoop产品HBase组件已经安装，并正常运行。

步骤2 客户端机器安装Eclipse和JDK程序，Eclipse使用3.0及以上版本，JDK使用1.6及以上版本。

步骤3 配置客户端网络连接。

1. 确认客户端机器与服务端各个主机网络上互通。
2. 确认当前系统中是否存在DNS服务器。
   - 是，执行**步骤4**。
   - 否，执行下一步。
3. 打开客户端机器上的hosts文件，在hosts文件中按照如下格式增加集群中各主机的主机名和IP地址。例如：

   IP地址 主机名
   160.172.0.130 dc1_rack1_host13

   说明

   hosts文件存放路径，例如：
   - windows: “C:\WINDOWS\system32\drivers\etc\hosts”
   - linux: “/etc/hosts”

步骤4 下载客户端程序。

1. 登录FushionInsight Hadoop Manager系统。

   在浏览器地址栏中输入访问地址，地址格式为**http://FushionInsight Hadoop Manager系统的IP地址:8080/web**。

   例如，在IE浏览器地址栏中，输入“http://160.172.0.172:8080/web”。
2. 单击“Services > Download Client Configurations”。
   保存文件包到本地。

3. 解压文件包，放到机器的指定目录下。

步骤5 解压随本文档一起提供的examples.tar.gz压缩包，得到样例工程。

该样例工程为Java工程，可以直接导入到Eclipse中。

步骤6 准备配置文件。

1. 将客户端解压目录下的“KrbClient\krb5.conf”文件拷贝到如下目录。若文件存在，请直接覆盖。
   - windows：“C:\WINDOWS\”

     需要将“krb5.conf”文件重命名为“krb5.ini”。
   - linux：“/etc/”
2. 从管理员处获取一个Kerberos机机帐号以及keytab文件。

   该帐号及keytab文件用于登录FusionInsight Hadoop产品并通过认证。
3. 将keytab文件放置在样例工程的“conf”目录下。
4. 在客户端解压目录下，获取子目录“HBase”的“core-site.xml”、“hbase-site.xml”、“jaas.conf”配置文件，并放置到样例工程的“conf”目录下。配置文件存放在客户端文件包的“HBase”中。
5. 修改样例工程文件“conf\jaas.conf”中如下内容。

   ```
   keyTab="keytab文件存放路径"
   principal="Kerberos机机帐号"
   ```

   例如：

   ```
   keyTab="D:\\workspace\\examples\\conf"
   principal="hbase/hadoop@HADOOP.COM"
   ```
6. 在客户端解压目录下，获取如下子目录中的jar文件到样例工程的“lib”目录下。
   - “HBase\hbase-0.94.0-security”目录下的“hbase-0.94.0-security.jar”、“hbase-0.94.0-security-tests.jar”文件。
   - “HBase\hbase-0.94.0-security\lib”目录下的所有文件。

步骤7 样例工程导入到Eclipse开发环境。

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

   显示“浏览文件夹”对话框。
2. 选择样例工程的examples文件夹。
3. 单击“Finish”。

   工程成功导入。

----结束

3 开发指引

应用程序实例

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
012005000201	张三	男	19	广东省深圳市
012005000202	李婉婷	女	23	河北省石家庄市
012005000203	王明	男	26	浙江省宁波市
012005000204	李刚	男	18	湖北省襄阳市
012005000205	赵恩如	女	21	江西省上饶市
012005000206	陈龙	男	32	湖南市株洲市
012005000207	周微	女	29	河南省南阳市
012005000208	杨艺文	女	30	重庆市开县
012005000209	徐兵	男	26	陕西省渭南市

编号	姓名	性别	年龄	地址
012005000210	肖凯	男	25	辽宁省大连市

开发过程与实现

开发人员使用HBase开发A业务程序的过程如下：

表 3-2 开发过程及代码实现

序号	步骤	代码实现
1	根据表3-1中的信息创建表。	请参见**创建用户信息表**。
2	导入用户数据。	请参见**Put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**修改表信息**。
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SingleColumnValueFilter过滤器**。
6	查询年龄段在[20－29]之间的用户信息。	请参见**使用FilterList过滤器**。
7	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。	请参见**聚合函数Aggregate**。
8	用户销户，删除用户信息表中该用户的数据。	请参见**Delete数据**。
9	A业务结束后，删除用户信息表。	请参见**删除表**。

4 代码样例

关于本章

4.1 创建表

功能介绍

HBase通过HBaseAdmin的createTable方法来创建表，并指定表名、列族名，创建表有两种方式：

- 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。

代码样例

下面代码片段在com.huawei.bigdata.hbase.examples.TestSample类中：

创建用户信息表

```
public void testCreateTable() {

  boolean result = false;
  // 指定表名
  String tableName = "user";
  // 创建Configuration实例，注[1]
  Configuration conf = getConfiguration();
  // 指定表描述信息对象
  HTableDescriptor htd = new HTableDescriptor(tableName);
  // 指定列族名称为info，注[2]
  HColumnDescriptor hcd = new HColumnDescriptor("info");
  htd.addFamily(hcd);

  HBaseAdmin admin = null;
  try {
    // 实例化一个HBaseAdmin对象
    admin = new HBaseAdmin(conf);
    // 如果表存在，删除表
    removeTable(conf, tableName);
    // 创建一个没有划分region个数的的表，注[3]
    admin.createTable(htd);
    // 判断表是否存在
    result = admin.tableExists(tableName);
    Assert.assertTrue(result);

  } catch (IOException e) {
    LOG.error(e.getMessage());
    Assert.fail(e.getMessage());
  } finally {
    try {
      // 关闭HBaseAdmin对象
      if (admin != null) {
        admin.close();
      }
    } catch (IOException e) {
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    }
  }
}
```

扩展应用

- 注[1] 共享Configuration实例

  HBase客户端代码通过创建一个连接实例，来获取与一个HBase集群进行交互的权限。如果频繁的创建Configuration实例，会导致创建很多不必要的连接实例，很容易达到连接数上限。建议在整个客户端代码范围内，都共用同一个Configuration对象实例。

- 注[2] 可以设置列族的压缩方式，代码片段如下：

```
columnDesc = new HColumnDescriptor(families[i]);
//设置前缀压缩，HBase提供了PREFIX、DIF、FAST_DIFF三种前缀压缩方法
columnDesc.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);
//设置文件压缩方式
columnDesc.setCompressionType(Algorithm.SNAPPY);
tableDesc.addFamily(columnDesc);
```

- 注[3] 可以指定起始和结束RowKey或者RowKey数组预分Region建表。

4.2 删除表

功能介绍

HBase通过HBaseAdmin的deleteTable方法来删除表。

代码样例

下面代码片段在com.huawei.bigdata.hbase.examples.TestSample类中：

样例：删除表

```
public void testDeleteTable() {

  boolean result = false;
  // 指定表名
  String tableName = "user";
  // 创建Configuration实例
  Configuration conf = getConfiguration();

  HBaseAdmin admin = null;
  try {
    // 实例化一个HBaseAdmin对象
    admin = new HBaseAdmin(conf);

    // 判断表是否处于上线状态
    result = admin.isTableEnabled(tableName);
    Assert.assertTrue(result);

    // 修改表的Enabled属性为false，此时表处于下线状态，可以执行删除表的操作
    admin.disableTable(tableName);
    // 判断表是否处于下线状态
    result = admin.isTableDisabled(tableName);
    Assert.assertTrue(result);

    // 提交一次deleteTable请求，注[1]
    admin.deleteTable(tableName);
    // 判断表是否存在
    result = admin.tableExists(tableName);
    Assert.assertFalse(result);

  } catch (IOException e) {
    LOG.error(e.getMessage());
    Assert.fail(e.getMessage());
  } finally {
```

```
    try {
      // 关闭HBaseAdmin对象
      if (admin != null) {
        admin.close();
      }
    } catch (IOException e) {
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    }
  }
}
```

扩展应用

注[1] 只有表的Enabled属性为false时，才能被删除掉，所以deleteTable常与disableTable，enableTable，tableExists，isTableEnabled，isTableDisabled结合在一起使用。

4.3 修改表

功能介绍

HBase通过HBaseAdmin的modifyTable和modifyColumn方法修改表信息和表中已存在列族的信息。

代码样例

下面代码片段在com.huawei.bigdata.hbase.examples.TestSample类中：

样例：修改表信息

```
public void testModifyTable() {

  // 指定表名
  String tableName = "user";
  // 指定列族名
  byte[] familyName = Bytes.toBytes("education");
  // 创建Configuration实例
  Configuration conf = getConfiguration();

  HBaseAdmin admin = null;
  try {
    // 实例化一个HBaseAdmin对象
    admin = new HBaseAdmin(conf);

    // 获取表描述信息对象
    HTableDescriptor htd = admin.getTableDescriptor(Bytes.toBytes(tableName));
    // 修改前，判断表是否有指定列族
    Assert.assertFalse(htd.hasFamily(familyName));
    // 创建列描述对象
    HColumnDescriptor hcd = new HColumnDescriptor(familyName);
    htd.addFamily(hcd);

    // 修改表前，你需要disable表，使表处于下线状态
    admin.disableTable(tableName);
    // 提交modifyTable请求，注[1]
    admin.modifyTable(Bytes.toBytes(tableName), htd);
    // 修改表后，你需要enable表，使表处于上线状态
    admin.enableTable(tableName);

    // 修改后，判断表是否有指定列族
    Assert.assertTrue(htd.hasFamily(familyName));

  } catch (IOException e) {
```

```
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    } finally {
      try {
        // 关闭HBaseAdmin对象
        if (admin != null) {
          admin.close();
        }
      } catch (IOException e) {
        LOG.error(e.getMessage());
        Assert.fail(e.getMessage());
      }
    }
  }
```

扩展应用

注[1] modifyTable和modifyColumn只有表的Enabled属性为false时，才能生效。

4.4 Put 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。

代码样例

下面代码在com.huawei.bigdata.hbase.examples.TestSample类中：

Put数据

```
public void testPut() {

  // 指定表名
  String tableName = "user";
  // 指定列族名
  byte[] FAMILIES = Bytes.toBytes("info");
  // 指定列名
  byte[][] qualifiers = { Bytes.toBytes("name"), Bytes.toBytes("gender"),
      Bytes.toBytes("age"), Bytes.toBytes("address") };
  // 创建Configuration实例
  Configuration conf = ConfigureUtil.getConfiguration();

  HTable table = null;
  try {
    // 实例化一个HTable对象，注[1]
    table = new HTable(conf, tableName);
    List<Put> puts = new ArrayList<Put>();
    // 实例化一个Put对象
    Put put = new Put(Bytes.toBytes("012005000201"));
    put.add(FAMILIES, qualifiers[0], Bytes.toBytes("张三"));
    put.add(FAMILIES, qualifiers[1], Bytes.toBytes("男"));
    put.add(FAMILIES, qualifiers[2], Bytes.toBytes(new Long(19)));
    put.add(FAMILIES, qualifiers[3], Bytes.toBytes("广东省深圳市"));
    puts.add(put);

    // 提交put数据请求
    table.put(puts);

  } catch (IOException e) {
    Assert.fail(e.getMessage());
```

```
    } finally {
      try {
        // 关闭HTable对象
        if (table != null) {
          table.close();
        }
      } catch (IOException e) {
        Assert.fail(e.getMessage());
      }
    }
  }
```

扩展应用

注[1] 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。

4.5 Delete 数据

功能介绍

HBase通过HTable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

下面代码在com.huawei.bigdata.hbase.examples.TestSample类中：

样例：Delete数据

```
  public void testDelete() {

    // 指定表名
    String tableName = "user";
    // 指定rowKey值,即编号为012005000201
    byte[] rowKey = Bytes.toBytes("012005000201");
    // 创建Configuration实例
    Configuration conf = getConfiguration();

    HTable table = null;
    try {
      // 实例化一个HTable对象
      table = new HTable(conf, tableName);

      // 实例化一个Get对象
      Get get = new Get(rowKey);
      // 提交一次get数据请求
      Result result = table.get(get);
      // 检查rowkey是否存在
      Assert.assertTrue(result.raw().length > 0);

      // 实例化一个Delete对象
      Delete delete = new Delete(rowKey);
      // 提交一次delete数据请求
      table.delete(delete);
      result = table.get(get);
      // 检查rowkey是否被删除
      Assert.assertTrue(result.raw().length == 0);

    } catch (IOException e) {
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    } finally {
```

```
    try {
      // 关闭HTable对象
      if (table != null) {
        table.close();
      }
    } catch (IOException e) {
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    }
  }
}
```

扩展应用

无。

4.6 使用 Get 读取数据

功能介绍

要从表中读取一条数据，首先需要实例化该表对应的HTable对象，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。查询结果的该行数据存储Result对象中，Result中存储了多个KeyValue对。

代码样例

下面代码在com.huawei.bigdata.hbase.examples.TestSample类中：

样例：使用Get读取数据

```
public void testGet() {

  // 指定表名
  String tableName = "user";
  // 指定列族名
  byte[] FAMILY = Bytes.toBytes("info");
  // 指定列名
  byte[][] qualifier = { Bytes.toBytes("name"), Bytes.toBytes("address") };
  // 指定rowKey值
  byte[] rowKey = Bytes.toBytes("012005000201");
  // 创建Configuration实例
  Configuration conf = getConfiguration();

  HTable table = null;
  try {
    // 实例化一个HTable对象
    table = new HTable(conf, tableName);
    // 实例化一个Get对象
    Get get = new Get(rowKey);
    // 设置列族和列名
    get.addColumn(FAMILY, qualifier[0]);
    get.addColumn(FAMILY, qualifier[1]);
    // 提交一次get数据请求
    Result result = table.get(get);
    // 打印查询返回的数据
    for (KeyValue kv : result.raw()) {
      LOG.info(Bytes.toString(kv.getRow()) + ","
          + Bytes.toString(kv.getFamily()) + ","
          + Bytes.toString(kv.getQualifier()) + ","
          + Bytes.toString(kv.getValue()));
    }
  } catch (IOException e) {
    LOG.error(e.getMessage());
```

```
      Assert.fail(e.getMessage());
    } finally {
      try {
        // 关闭HTable对象
        if (table != null) {
          table.close();
        }
      } catch (IOException e) {
        LOG.error(e.getMessage());
        Assert.fail(e.getMessage());
      }
    }
  }
```

扩展应用

无。

4.7 使用 Scan 读取数据

功能介绍

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

代码样例

下面代码在com.huawei.bigdata.hbase.examples.TestSample类中：

样例：使用scan读取数据

```
public void testScanData() {

  // 指定表名
  String tableName = "user";
  // 创建Configuration实例
  Configuration conf = getConfiguration();

  HTable table = null;
  try {
    // 实例化一个HTable对象
    table = new HTable(conf, tableName);
    // 实例化一个Scan对象，注[1]
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));

    // 实例化一个ResultScanner对象
    ResultScanner rScanner = null;
    // 提交一次scan数据请求，注[2]
    rScanner = table.getScanner(scan);
    // 打印查询返回的数据
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
      for (KeyValue kv : r.raw()) {
        LOG.info(Bytes.toString(kv.getRow()) + ","
            + Bytes.toString(kv.getFamily()) + ","
            + Bytes.toString(kv.getQualifier()) + ","
            + Bytes.toString(kv.getValue()));
      }
    }
  } catch (IOException e) {
    LOG.error(e.getMessage());
```

```
      Assert.fail(e.getMessage());
    } finally {
      try {
        // 关闭HTable对象
        if (table != null) {
          table.close();
        }
      } catch (IOException e) {
        LOG.error(e.getMessage());
        Assert.fail(e.getMessage());
      }
    }
  }
```

扩展应用

- 注[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。
- 注[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

    一个RPC查询请求最大的返回的next数目，与一次RPC获取的行数有关。

4.8 使用过滤器 Filter

功能介绍

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，如设置RowKey、列名或者列值的过滤条件。

代码样例

下面代码片段在com.huawei.bigdata.hbase.examples.TestSample类中：

样例：使用SingleColumnValueFilter过滤器

```
public void testSingleColumnValueFilter() {

  // 指定表名
  String tableName = "user";
  // 创建Configuration实例
  Configuration conf = getConfiguration();

  HTable table = null;
  try {
    // 实例化一个HTable对象
    table = new HTable(conf, tableName);
    // 实例化一个Scan对象
    Scan scan = new Scan();
    scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));

    SingleColumnValueFilter filter = new SingleColumnValueFilter(
        Bytes.toBytes("info"), Bytes.toBytes("name"), CompareOp.EQUAL,
        Bytes.toBytes("徐兵"));

    // 设置过滤条件
    scan.setFilter(filter);

    // 实例化一个ResultScanner对象
```

```
      ResultScanner rScanner = null;
      // 提交一次scan数据请求
      rScanner = table.getScanner(scan);
      // 打印查询返回的数据
      for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
        for (KeyValue kv : r.raw()) {
          LOG.info(Bytes.toString(kv.getRow()) + ","
              + Bytes.toString(kv.getFamily()) + ","
              + Bytes.toString(kv.getQualifier()) + ","
              + Bytes.toString(kv.getValue()));
        }
      }
    } catch (IOException e) {
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    } finally {
      try {
        // 关闭HTable对象
        if (table != null) {
          table.close();
        }
      } catch (IOException e) {
        LOG.error(e.getMessage());
        Assert.fail(e.getMessage());
      }
    }
  }
```

样例：使用FilterList过滤器

```
  public void testFilterList() {

    // 指定表名
    String tableName = "user";
    // 创建Configuration实例
    Configuration conf = getConfiguration();

    HTable table = null;
    try {
      // 实例化一个HTable对象
      table = new HTable(conf, tableName);

      // 实例化一个Scan对象
      Scan scan = new Scan();
      scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));

      // 实例化一个FilterList对象，里面各个filter的是"and"关系
      FilterList list = new FilterList(Operator.MUST_PASS_ALL);
      // 获取age>=20的数据
      list.addFilter(new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes
          .toBytes("age"), CompareOp.GREATER_OR_EQUAL, Bytes.toBytes(new Long(
          20))));
      // 获取age<=29的数据
      list.addFilter(new SingleColumnValueFilter(Bytes.toBytes("info"), Bytes
          .toBytes("age"), CompareOp.LESS_OR_EQUAL, Bytes.toBytes(new Long(29))));
      // 设置过滤条件
      scan.setFilter(list);

      // 实例化一个ResultScanner对象
      ResultScanner rScanner = null;
      // 提交一次scan数据请求
      rScanner = table.getScanner(scan);
      // 打印查询返回的数据
      for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
        for (KeyValue kv : r.raw()) {
          LOG.info(Bytes.toString(kv.getRow()) + ","
              + Bytes.toString(kv.getFamily()) + ","
              + Bytes.toString(kv.getQualifier()) + ","
              + Bytes.toString(kv.getValue()));
        }
```

```
      }
    } catch (IOException e) {
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    } finally {
      try {
        // 关闭HTable对象
        if (table != null) {
          table.close();
        }
      } catch (IOException e) {
        LOG.error(e.getMessage());
        Assert.fail(e.getMessage());
      }
    }
  }
```

扩展应用

无。

4.9 聚合函数 Aggregate

功能介绍

聚合函数Aggregate使用协处理器（coprocessor）对表进行统计分析，比如求最小值、最大值，计算行数，求和，求平均值等。

代码样例

下面代码片段在com.huawei.bigdata.hbase.examples.TestSample类中：

样例：聚合函数Aggregate

```
public void testAggregate() {

  // 指定表名
  byte[] tableName = Bytes.toBytes("user");
  // 指定列族名
  byte[] family = Bytes.toBytes("info");
  // 创建Configuration实例
  Configuration conf = getConfiguration();

  try {
    AggregationClient aggregationClient = new AggregationClient(conf);

    // 实例化一个Scan对象
    Scan scan = new Scan();
    scan.addFamily(family);
    scan.addColumn(family, Bytes.toBytes("age"));

    // 获取行数
    long rowCount = aggregationClient.rowCount(tableName, null, scan);
    LOG.info("row count is " + rowCount);

    // 获取最大值
    long max = aggregationClient.max(tableName, new LongColumnInterpreter(),
        scan);
    LOG.info("max number is " + max);

    // 获取最小值
    long min = aggregationClient.min(tableName, new LongColumnInterpreter(),
        scan);
```

```
      LOG.info("min number is " + min);

    } catch (Throwable e) {
      LOG.error(e.getMessage());
      Assert.fail(e.getMessage());
    }
  }
```

扩展应用

无。

5 常用接口

样例主要使用到下面三个类中的接口方法：

表 5-1 org.apache.hadoop.hbase.client.HBaseAdmin 类

方法	说明
createTable(HTableDescriptor desc)	不预分Region建表。
createTable(HTableDescriptor desc, byte [] startKey, byte [] endKey, int numRegions)	指定起止RowKey和Region个数；此时的起始RowKey为第一个Region的endKey，结束key为最后一个Region的startKey。
createTable(HTableDescriptor desc, byte[][] splitKeys)	指定RowKey数组，不包括第一个Region的startKey和最后一个Region的endKey，Region个数等于数组长度+1。
deleteTable(String tableName)	删除表。
deleteTables(String regex)	删除满足指定样式的表。
enableTable(String tableName)	修改表的Enabled属性为true，此时表处于上线状态，表才能被访问。
disableTable(String tableName)	修改表的Enabled属性为false，此时表处于下线状态，可以执行删除表的操作。
tableExists(String tableName)	判断表是否存在。
isTableEnabled(String tableName)	判断表是否处于上线状态。
isTableDisabled(String tableName)	判断表是否处于下线状态。
modifyTable(byte [] tableName, HTableDescriptor htd)	修改表信息。
modifyColumn(byte [] tableName, HColumnDescriptor descriptor)	修改表中已存在列族的信息。

表 5-2 org.apache.hadoop.hbase.client.HTable 类

方法	说明
delete(Delete delete)	单条数据delete。
delete(List<Delete> deletes)	批量数据delete。
get(Get get)	读取一行数据。
get(List<Get> gets)	读取多行数据。
put(Put put)	单条数据put。
put(List<Put> puts)	多条数据Put。
getScanner(Scan scan)	指定Scan信息，读取表数据。
getScanner(byte[] family)	指定family信息，读取表数据。
getScanner(byte[] family, byte[] qualifier)	指定family和qualifier信息，读取表数据。

表 5-3 org.apache.hadoop.hbase.client.coprocessor.AggregationClient 类

方法	说明
max(byte[] tableName, ColumnInterpreter<R, S> ci, Scan scan)	获取最大值。
min(byte[] tableName, ColumnInterpreter<R, S> ci, Scan scan)	获取最小值。
rowCount(byte[] tableName, ColumnInterpreter<R, S> ci, Scan scan)	获取行数。
sum(byte[] tableName, ColumnInterpreter<R, S> ci, Scan scan)	获取总和。
avg(byte[] tableName, ColumnInterpreter<R, S> ci, Scan scan)	获取平均值。

HBase采用的接口同开源社区版本保持一致，详情请参见**http://hbase.apache.org/apidocs/index.html**