Hbase 学习笔记

01课时 Hbase介绍

1. Hbase是面向列的数据库

什么是面向列的数据库？面向列族（column family），对于一列数据进行拆分，比如地址信息（addresss）可以拆分成为省份（Provience）、市（City），这些属于同一个列族。

2. Hbase是一个开源的非关系型分布式数据库（NoSQL），他参考了Google的BigTable建模，实现语言是JAVA，是Apache软件基金会的Hadoop项目的一部分，运行在HDFS文件系统之上，为Hadoop提供类似BigTable规模的服务，因此HBase可以容错的存储海量稀疏数据。

HBase在列上实现了BigTable提高的压缩算法、内存操作。HBase的表能够作为MapReduce任务的输入和输出，可以通过Java API方式访问数据，也可以通过REST、Avro或者Thrift的API进行访问。

虽然HBase性能很好，但是不能够直接替代SQL数据库。如今它已经应用于多个数据驱动型网站，包括Facebook的消息平台。

3. HBase Regions

自动进行水平分区：将一张大表拆成多个表格分不到不同的节点上进行处理

Row的子集：

第一行(include ) 最后一行（exclude）

每张表至少一个region

增长到阈值的时候，切割成两个相同的region

Row update是原子性操作

4. HBase架构

底层基于HDFS文件系统，将文件块放在内存中（之后再讲解）

HBase master：只有一个

负责初始安装

指定region到regionserver

恢复故障的regionserver

轻负载

HBase RegionServer(N 个)

携带0-N个region

负责客户端RW的请求

管理region split

通知Master新的子region

管理offline的父代region以及对其进行替换

HBase构成

HBase依赖于ZooKeeper

默认HBase管理ZooKeeper（start/stop）

HMaster和HregionServer在ZooKeeper中进行注册

02课时 HBase安装过程

1.下载之后解压到/usr/cloud/下面，穿件hbase连接到指定的文件夹

2.修改配置文件hbase-site.xml，修改Hbase的目录

<property>

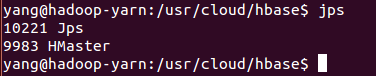
<name>hbase.rootdir</name>

<value>file:///tmp/hbase</value>

</property>

3. 启动本地文件系统的hbase

./bin/start-hbase.sh hbase已经启动起来，多一个HMaster进程



4. 基本的hbase shell模式下

./hbase shell

create ‘test’, ‘cf’ #创建test表 cf列

put ‘test’, ‘row1’, ‘cf:a’, ‘value1’

list ‘test’

put ‘test’,’row2’,’cf:b’, ‘value2’

put ‘test’,’row3’,’cf:c’, ‘value3’

scan ‘test’

get ‘test’, ‘row1’

disable ‘test’

drop ‘test’

exit

./stop-hbase.sh

03课时 完全分布式配置、启动与Shell命令查看

1.hbase-site.xml （配置hdfs文件系统、分布式开关、Zookeeper节点）

hbase.rootdir=hdfs://hostname:8020/hbase

hbase.cluster.distributed=true

hbase.zookeeper.quorum=hostnames(hadoop-yarn1,hadoop-yarn2,hadoop-yarn3)

hbase.zookeeper.property.dataDir=/tmp/zookeeper

2.regionservers(配置区域服务器列表)

hostnames

3.backup-masters

Hostname

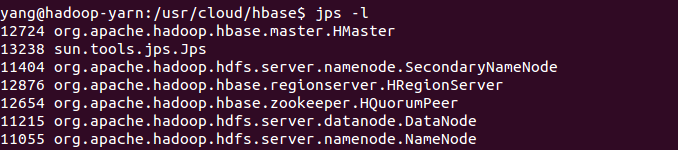
4.hbase-env.sh

设置JAVA\_HOME 环境变量

5.启动HDFS文件系统

start-dfs.sh

6.启动Hbase

./bin/start-hbase.sh

04课时 使用客户端API创建Hbase数据表并且在hbase下导出执行

1. Hbase数据库名字空间[namespace],默认是在default， hbase的自己内置的数据表，数据库名字空间在逻辑上对表进行分组。我们创建的数据库默认在default下面

2.使用Eclipse操作hbase数据库

|  |
| --- |
| Configuration conf = HBaseConfiguration.*create*();  HBaseAdmin admin = **new** ~~HBaseAdmin~~(conf);  conf.set("hbase.zookeeper.quorum", "hadoop-yarn");  conf.set("hbase.zookeeper.property.clientPort", "2181");  TableName tName = TableName.*valueOf*("teng");    HTableDescriptor tDesc = **new** HTableDescriptor(tName);    HColumnDescriptor cDesc = **new** HColumnDescriptor("data");    tDesc.addFamily(cDesc);    admin.createTable(tDesc);    System.***out***.println("OK"); |

可以直接在Eclipse中运行。

05课时 使用客户端完成数据库的CRUD操作

|  |
| --- |
| public class HBaseApp {  private Configuration conf = null;  private HBaseAdmin admin = null;  private void initConf() throws MasterNotRunningException, ZooKeeperConnectionException, IOException {  System.out.println("===============init start==========================");  conf = HBaseConfiguration.create();  conf.set("hbase.zookeeper.quorum", "hadoop-yarn");  conf.set("hbase.zookeeper.property.clientPort", "2181");  admin = new HBaseAdmin(conf);  System.out.println("===============init end==========================");  }  private void createTable(String tablename, String family)  throws MasterNotRunningException, ZooKeeperConnectionException, IOException {  System.out.println("===============create table start==========================");  TableName tName = TableName.valueOf(tablename);  HTableDescriptor tDesc = new HTableDescriptor(tName);  HColumnDescriptor cDesc = new HColumnDescriptor(family);  tDesc.addFamily(cDesc);  admin.createTable(tDesc);  System.out.println("Create Table" + tablename);  System.out.println("===============create table finished==========================");  }  private void putData(HTable table, String family) throws IOException {  System.out.println("===============put data start==========================");  Put put = new Put(Bytes.toBytes("row1"));  put.addColumn(Bytes.toBytes(family), Bytes.toBytes("1"), Bytes.toBytes("zhang"));  put.addColumn(Bytes.toBytes(family), Bytes.toBytes("2"), Bytes.toBytes("zhangsan"));  put.addColumn(Bytes.toBytes(family), Bytes.toBytes("3"), Bytes.toBytes("zhangsansan"));  table.put(put);  System.out.println("===============put data finished==========================");  }  private void getData(HTable table, String family, String[] qualifiers) throws IOException {  System.out.println("===============get data start=============================");  Get get = new Get(Bytes.toBytes("row1"));  for (String qualifier : qualifiers) {  get.addColumn(Bytes.toBytes(family), Bytes.toBytes(qualifier));  }  Result result = table.get(get);  for (KeyValue kv : result.list()) {  System.out.println(" family :" + Bytes.toString(kv.getFamily()) + " qualifiter :"  + Bytes.toString(kv.getQualifier()) + " value :" + Bytes.toString(kv.getValue())  + " timestamp :" + kv.getTimestamp());  // (Bytes.toBytes(family), Bytes.toBytes(qualifier));  // System.out.println(family+":"+ qualifier+" key="+ Bytes.toString(  // kv.getKey()));  // System.out.println(family+":"+ qualifier+" value="+  // Bytes.toString( kv.getKey()));  }  System.out.println("===============get data finished===========================");  }  private void scanData(HTable table, String family, String[] qualifiers) throws IOException {  System.out.println("===============scan data start=============================");  Scan scan = new Scan();  ResultScanner rs = table.getScanner(scan);  for (Result result : rs) {  for (String qualifier : qualifiers) {  System.out  .println(  Bytes.toString(result.getRow()) + ":" + family + ":" + qualifier + "="  + Bytes.toString(result  .getColumnLatestCell(Bytes.toBytes(family), Bytes.toBytes(qualifier))  .getValue()));  }  }  System.out.println("===============scan data finished===========================");  }  private void deleteData(HTable table, String family, String qualifier) throws IOException {  System.out.println("===============delete data start=============================");  Delete delete = new Delete(Bytes.toBytes("row1"));  delete.addColumn(Bytes.toBytes(family), Bytes.toBytes(qualifier));  table.delete(delete);  System.out.println("===============delete data finished===========================");  }  private void dropTable(String tableName) throws IOException {  System.out.println("===============drop table start=============================");  this.admin.disableTable(tableName);  this.admin.deleteTable(tableName);  System.out.println("===============drop table finished===========================");  }  public static void main(String[] args) throws MasterNotRunningException, ZooKeeperConnectionException, IOException {  // TODO Auto-generated method stub  String tablename = "teng";  String family = "data";  String[] qualifiers = { "1", "2", "3" };  String[] qualifiers\_new = { "2", "3" };  HBaseApp app = new HBaseApp();  app.initConf();  app.createTable(tablename, family);  HTable table = new HTable(app.conf, tablename);  app.putData(table, family);  app.getData(table, family, qualifiers);  app.scanData(table, family, qualifiers);  app.deleteData(table, family, "1");  app.scanData(table, family, qualifiers\_new);  app.dropTable(tablename);  table.close();  app.admin.close();  }  } |

06课时 HBase 使用新型的API访问数据库

|  |
| --- |
| public class HBaseApp {  private Configuration conf = null;  private Admin admin = null;  private Connection conn = null;  private void initConf() throws MasterNotRunningException, ZooKeeperConnectionException, IOException {  System.out.println("===============init start==========================");  conf = HBaseConfiguration.create();  conf.set("hbase.zookeeper.quorum", "hadoop-yarn");  conf.set("hbase.zookeeper.property.clientPort", "2181");    conn = ConnectionFactory.createConnection(conf);  admin = conn.getAdmin();    System.out.println("===============init end==========================");  }  private void createTable(String tablename, String family)  throws MasterNotRunningException, ZooKeeperConnectionException, IOException {  System.out.println("===============create table start==========================");  TableName tName = TableName.valueOf(tablename);  HTableDescriptor tDesc = new HTableDescriptor(tName);  HColumnDescriptor cDesc = new HColumnDescriptor(family);  tDesc.addFamily(cDesc);  admin.createTable(tDesc);  System.out.println("Create Table" + tablename);  System.out.println("===============create table finished==========================");  }  private void putData(Table table, String family) throws IOException {  System.out.println("===============put data start==========================");  Put put = new Put(Bytes.toBytes("row1"));  put.addColumn(Bytes.toBytes(family), Bytes.toBytes("1"), Bytes.toBytes("zhang"));  put.addColumn(Bytes.toBytes(family), Bytes.toBytes("2"), Bytes.toBytes("zhangsan"));  put.addColumn(Bytes.toBytes(family), Bytes.toBytes("3"), Bytes.toBytes("zhangsansan"));  table.put(put);  System.out.println("===============put data finished==========================");  }  private void getData(Table table, String family, String[] qualifiers) throws IOException {  System.out.println("===============get data start=============================");  Get get = new Get(Bytes.toBytes("row1"));  for (String qualifier : qualifiers) {  get.addColumn(Bytes.toBytes(family), Bytes.toBytes(qualifier));  }  Result result = table.get(get);  for (KeyValue kv : result.list()) {  System.out.println(" family :" + Bytes.toString(kv.getFamily()) + " qualifiter :"  + Bytes.toString(kv.getQualifier()) + " value :" + Bytes.toString(kv.getValue())  + " timestamp :" + kv.getTimestamp());  // (Bytes.toBytes(family), Bytes.toBytes(qualifier));  // System.out.println(family+":"+ qualifier+" key="+ Bytes.toString(  // kv.getKey()));  // System.out.println(family+":"+ qualifier+" value="+  // Bytes.toString( kv.getKey()));  }  System.out.println("===============get data finished===========================");  }  private void scanData(Table table, String family, String[] qualifiers) throws IOException {  System.out.println("===============scan data start=============================");  Scan scan = new Scan();  ResultScanner rs = table.getScanner(scan);  for (Result result : rs) {  for (String qualifier : qualifiers) {  System.out  .println(  Bytes.toString(result.getRow()) + ":" + family + ":" + qualifier + "="  + Bytes.toString(result  .getColumnLatestCell(Bytes.toBytes(family), Bytes.toBytes(qualifier))  .getValue()));  }  }  System.out.println("===============scan data finished===========================");  }  private void deleteData(Table table, String family, String qualifier) throws IOException {  System.out.println("===============delete data start=============================");  Delete delete = new Delete(Bytes.toBytes("row1"));  delete.addColumn(Bytes.toBytes(family), Bytes.toBytes(qualifier));  table.delete(delete);  System.out.println("===============delete data finished===========================");  }  private void dropTable(String tableName) throws IOException {  System.out.println("===============drop table start=============================");  this.admin.disableTable(TableName.valueOf(tableName));;  this.admin.deleteTable(TableName.valueOf(tableName));  System.out.println("===============drop table finished===========================");  }  public static void main(String[] args) throws MasterNotRunningException, ZooKeeperConnectionException, IOException {  // TODO Auto-generated method stub  String tablename = "teng";  String family = "data";  String[] qualifiers = { "1", "2", "3" };  String[] qualifiers\_new = { "2", "3" };  HBaseApp app = new HBaseApp();  app.initConf();  app.createTable(tablename, family);  // HTable table = new HTable(app.conf, tablename);  Table table = app.conn.getTable(TableName.valueOf(tablename));  app.putData(table, family);  app.getData(table, family, qualifiers);  app.scanData(table, family, qualifiers);  app.deleteData(table, family, "1");  app.scanData(table, family, qualifiers\_new);  app.dropTable(tablename);  table.close();  app.admin.close();  }  } |

07课时 HBase使用新型API批量插入、HBase表结构与Hadoop文件系统的对应关系

|  |
| --- |
| **public** **class** MuchDataInsertApp {  **private** **final** **static** Logger ***LOG*** = Logger.*getLogger*(MuchDataInsertApp.**class**);  **private** **static** Configuration *conf* = **null**;  **private** **static** Connection *conn* = **null**;  // private static Admin admin= null;  **private** **static** **void** init() **throws** IOException {  *conf* = HBaseConfiguration.*create*();  *conf*.set("hbase.zookeeper.quorum", "hadoop-yarn");  *conf*.set("hbase.zookeeper.property.clientPort", "2181");  *conn* = ConnectionFactory.*createConnection*(*conf*);  // admin = conn.getAdmin();  }  **private** **void** createTable(String name, String family) **throws** IOException {  Admin admin = *conn*.getAdmin();  TableName tableName = TableName.*valueOf*(name);  **if** (admin.tableExists(tableName)) {  // drop table  ***LOG***.info("table " + name + " exists drop the table now");  admin.disableTable(tableName);  admin.deleteTable(tableName);  }  HTableDescriptor htd = **new** HTableDescriptor(tableName);  HColumnDescriptor hcd = **new** HColumnDescriptor(family);  htd.addFamily(hcd);  admin.createTable(htd);  }  /\*  \* batch put data  \*/  **private** **void** batchPutData(String tablename, **int** n, String family, String[] qualifiers) **throws** IOException {  **long** start = System.*currentTimeMillis*();  ***LOG***.info("start batch time:" + **new** Date(start).~~toLocaleString~~());  Put put = **null**;  TableName tableName = TableName.*valueOf*(tablename);  Table table = *conn*.getTable(tableName);  **for** (**int** i = 1; i < n; i++) {  put = **new** Put(Bytes.*toBytes*("row" + i));  **for** (String qualifier : qualifiers) {  put.addColumn(Bytes.*toBytes*(family), Bytes.*toBytes*(qualifier),  Bytes.*toBytes*("yang:" + qualifier + ":" + i));  }  table.put(put);  **if** (0 == i % 100) {  ***LOG***.warn("finished/total:" + i + "/" + n);  }  }  table.close();  **long** end = System.*currentTimeMillis*();  ***LOG***.info("finish batch time:" + **new** Date(start).~~toLocaleString~~());  ***LOG***.info("insert " + n + " data, this process spend time:" + (end - start) + " milli seconds");  }  **public** **static** **void** main(String[] args) **throws** IOException {  MuchDataInsertApp.*init*();  MuchDataInsertApp app = **new** MuchDataInsertApp();  String tablename = "batch";  String family = "data";  String[] qualifiers = { "a", "b", "c" };  app.createTable(tablename, family);  app.batchPutData(tablename, 1000, family, qualifiers);  }  } |

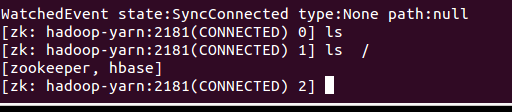
08课时 HBase与Zookeeper组合应用以及在Shell下操纵namespace

status

whoami

Hbase 依赖于ZooKeeper，HQuorumPeer就是ZooKeeper进程

在命令行下 hbase zkcli 进入ZooKeeper的界面，它会连接所有主机的HBase的2181端口，（我们可以使用zkCli.sh -server hostname:port,进入主机的ZooKeeper），



echo “describe ‘batch’“ | hbase shell

echo “scan ‘hbase:meta’” | hbase shell

create\_namespace ‘yangtf’

list\_namespace

list\_namespace\_tables ‘default’

09课时 HBase Shell常用的操作

help

namespace：

alter\_namespace <namespace>

create\_namespace <namespace>

list\_namespace

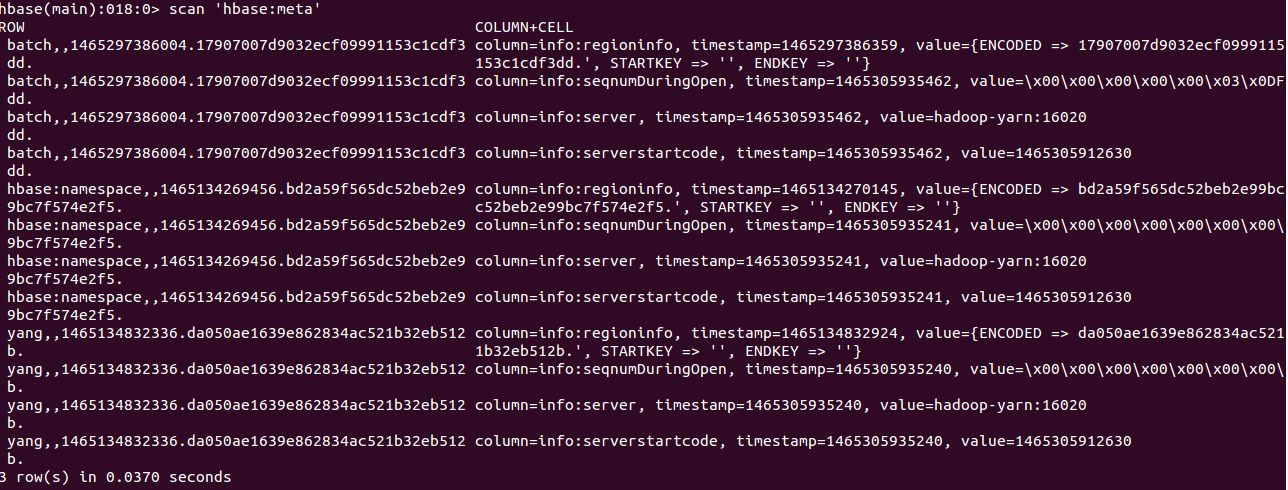
list\_namespace\_table <namespace>

scan

scan <tablename>, LIMIT=>10

count <tablename>, INTERVAL=>1000, CACHE=1000

10课时 HBase内部Meta表分析以及设计原理



RowName

batch,,1465297386004.17907007d9032ecf09991153c1cdf3dd.

info:regioninfo

timestamp

value={

ENCODE=

STARTKEY=

ENDKEY=

}

timestamp

value={}

#不同时间戳下，有不同的值

info:seqnumDuringOpen

timespace

value

info:server

timespace

value=> 在哪一个机器上存储 hostname:16020

info:serverstartcode

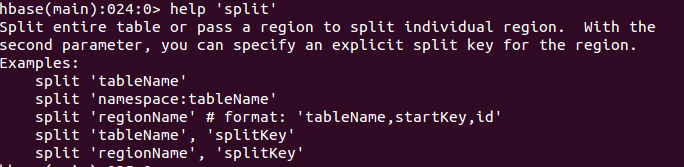
timespace

value

首先找到Key：xxxx，查询Meta表，然后对应列的regionserver中的startkey, endkey , 找到对应的server，然后去对应的server获取数据。

11课时HBase Table的拆分合并原理

help ‘split’



split <tablename>

echo "scan 'hbase:meta'" | hbase shell > meta.txt

查看文件：

scan 'hbase:meta'

ROW COLUMN+CELL

batch,,1465310084147.106b9e90be695577dd2f0037ebb3d1e6.

column=info:regioninfo,

timestamp=1465310084699,

value={

ENCODED => 106b9e90be695577dd2f0037ebb3d1e6,

NAME => 'batch,,1465310084147.106b9e90be695577dd2f0037ebb3d1e6.',

STARTKEY => '',

ENDKEY => 'row55037'

}

batch,,1465310084147.106b9e90be695577dd2f0037ebb3d1e6.

column=info:seqnumDuringOpen,

timestamp=1465310084900, value=\x00\x00\x00\x00\x00\x03\x0DC

batch,,1465310084147.106b9e90be695577dd2f0037ebb3d1e6.

column=info:server, timestamp=1465310084900, value=hadoop-yarn:16020

batch,,1465310084147.106b9e90be695577dd2f0037ebb3d1e6.

column=info:serverstartcode, timestamp=1465310084900, value=1465305912630

batch,row55037,1465310084147.dd4d57307f14d1516d8caae106b26013.

column=info:regioninfo,

timestamp=1465310084699,

value={

ENCODED => dd4d57307f14d1516d8caae106b26013,

NAME => 'batch,row55037,1465310084147.dd4d57307f14d1516d8caae106b26013.',

STARTKEY => 'row55037',

ENDKEY => ''}

batch,row55037,1465310084147.dd4d57307f14d1516d8caae106b26013.

column=info:seqnumDuringOpen, timestamp=1465310084877, value=\x00\x00\x00\x00\x00\x03\x0DD

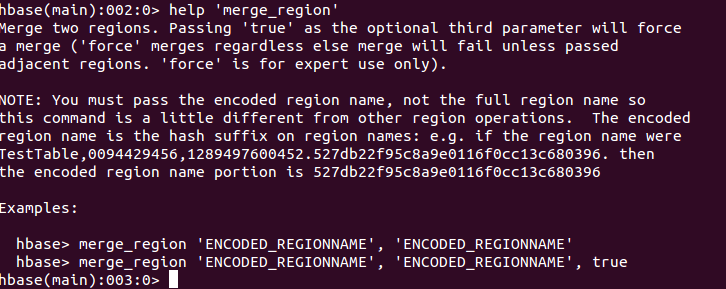
batch,row55037,1465310084147.dd4d57307f14d1516d8caae106b26013.

column=info:server, timestamp=1465310084877, value=hadoop-yarn:16020

batch,row55037,1465310084147.dd4d57307f14d1516d8caae106b26013.

column=info:serverstartcode, timestamp=1465310084877, value=1465305912630

合并表



merge\_region '106b9e90be695577dd2f0037ebb3d1e6','dd4d57307f14d1516d8caae106b26013', true

echo "scan 'hbase:meta'" | hbase shell > meta.txt

scan 'hbase:meta'

ROW COLUMN+CELL

batch,,1465311073095.4237e344bef31a4293368b13b74b5839.

column=info:mergeA,

timestamp=1465311073395, value=PBUF\x08\xB3\xF8\xA5\xDA\xD2\*\x12\x10\x0A\x07default\x12\x05batch\x1A\x00"\x08row55037(\x000\x008\x00

batch,,1465311073095.4237e344bef31a4293368b13b74b5839.

column=info:mergeB,

timestamp=1465311073395, value=PBUF\x08\xB3\xF8\xA5\xDA\xD2\*\x12\x10\x0A\x07default\x12\x05batch\x1A\x08row55037"\x00(\x000\x008\x00

batch,,1465311073095.4237e344bef31a4293368b13b74b5839.

column=info:regioninfo,

timestamp=1465311073395,

value={

ENCODED => 4237e344bef31a4293368b13b74b5839,

NAME => 'batch,,1465311073095.4237e344bef31a4293368b13b74b5839.',

STARTKEY => '',

ENDKEY => ''}

batch,,1465311073095.4237e344bef31a4293368b13b74b5839.

column=info:seqnumDuringOpen,

timestamp=1465311073583, value=\x00\x00\x00\x00\x00\x03\x0DE

batch,,1465311073095.4237e344bef31a4293368b13b74b5839.

column=info:server, timestamp=1465311073583, value=hadoop-yarn:16020

batch,,1465311073095.4237e344bef31a4293368b13b74b5839.

column=info:serverstartcode, timestamp=1465311073583, value=1465305912630

12课时HBase编程实现Table和Region的拆分合并

|  |
| --- |
| public class HBaseSplitMergeApp {  private final static Logger LOG = Logger.getLogger(HBaseSplitMergeApp.class);  private static Configuration conf = null;  private static Connection conn = null;  // private static Admin admin= null;  private static void init() throws IOException {  conf = HBaseConfiguration.create();  conf.set("hbase.zookeeper.quorum", "hadoop-yarn");  conf.set("hbase.zookeeper.property.clientPort", "2181");  conn = ConnectionFactory.createConnection(conf);  }  private static void cleanup() throws IOException {  conn.close();  }  private void findRegions(String tablename) throws IOException {  Admin admin = conn.getAdmin();  List<HRegionInfo> list = admin.getTableRegions(TableName.valueOf(tablename));  LOG.warn("region size:" + list.size());  admin.close();  }  private void splitTable(String tablename) throws IOException {  Admin admin = conn.getAdmin();  admin.split(TableName.valueOf(tablename));  admin.close();  }  private void mergeRegion() {  }  public static void main(String[] args) throws IOException {  // TODO Auto-generated method stub  HBaseSplitMergeApp.init();  HBaseSplitMergeApp app = new HBaseSplitMergeApp();  String tablename = "batch";  app.findRegions(tablename);  app.splitTable(tablename);  app.findRegions(tablename);  HBaseSplitMergeApp.cleanup();  }  } |