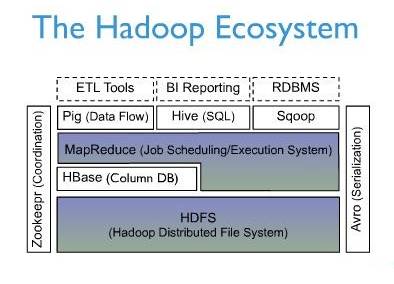
Hadoop深入浅出

课程安排

* HBASE基础知识\*
* HBASE的伪分布的搭建\*
* HBASE Shell\*\*\*
* HBASE的Java客户端\*\*\*

---------------------------加深拓展----------------------

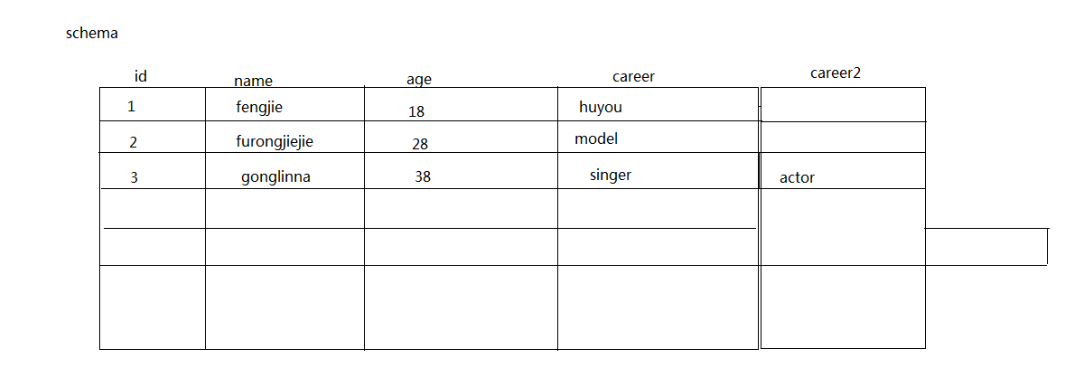
* HBASE的集群的搭建\*
* HBASE的表设计\*
* HBASE的底层存储模型\*



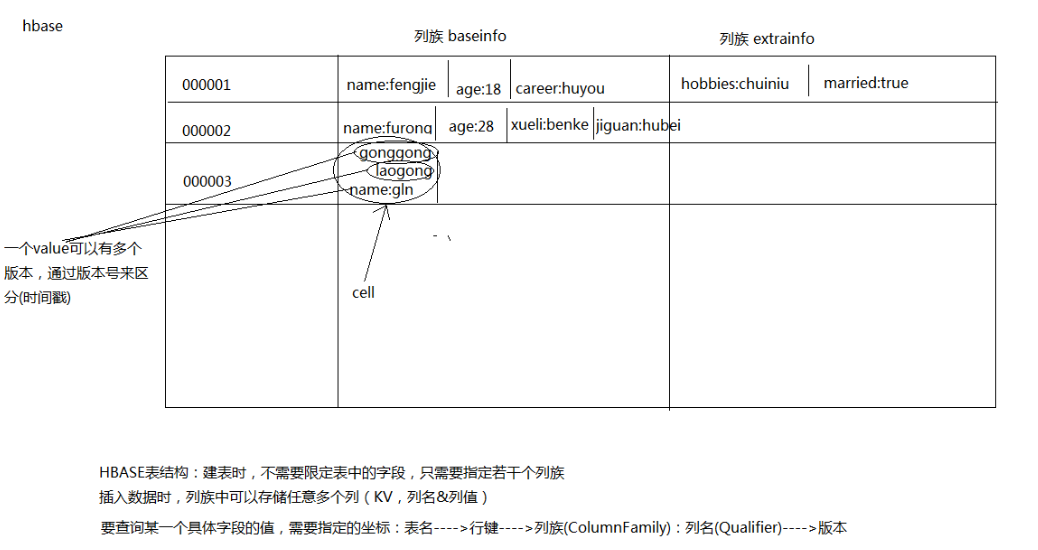
HBase简介

* HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用HBase技术可在廉价PC Server上搭建起大规模结构化存储集群。HBase利用Hadoop HDFS作为其文件存储系统，利用Hadoop MapReduce来处理HBase中的海量数据，利用Zookeeper作为协调工具。

关系型数据库存储结构图：



Hbase存储结构图：



思考：怎样存储物流信息

运单号：**728311858891**

2014-01-07 10:09:25 上海 已发货

2014-01-07 18:27:32 上海航空部 已收件

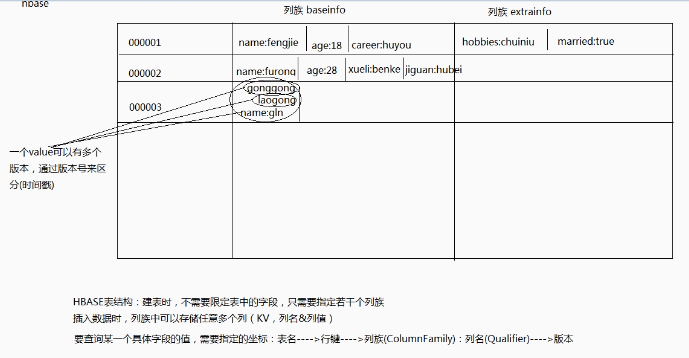
2014-01-07 20:25:38 快件离开上海航空部 已发往北京

2014-01-09 08:27:14 北京回龙观 派件中

2014-01-09 12:37:37 北京回龙观 已签收

主键：**Row Key**

* 主键是用来检索记录的主键，访问hbase table中的行，只有三种方式
* 通过单个row key访问
* 通过row key的range，访问多行
* 全表扫描，类似scan，逐行扫描



列族：Column Family

* 列族在创建表的时候声明，一个列族可以包含多个列，列中的数据都是以二进制形式存在，没有数据类型。

时间戳：timestamp

* HBase中通过row和columns确定的为一个存贮单元称为cell。每个 cell都保存着同一份数据的多个版本。版本默认通过时间戳来索引

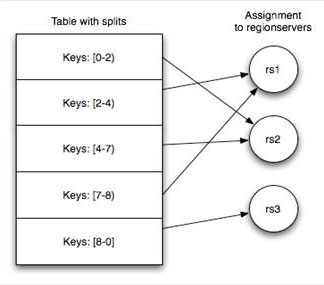
HBASE基础知识

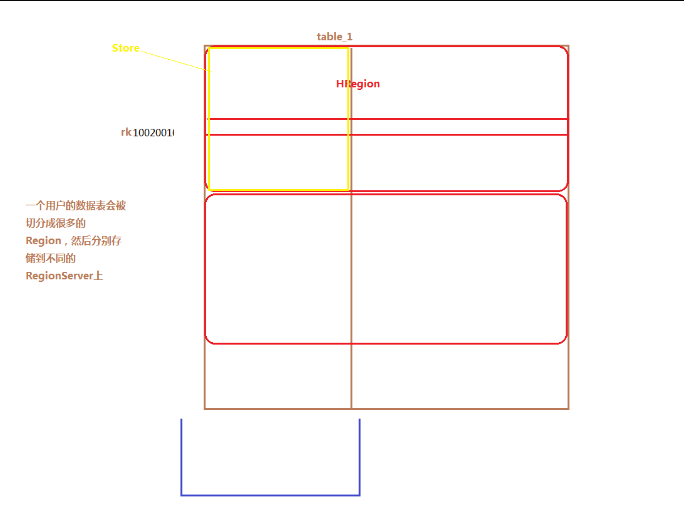
* **物理存储**

Table 在行的方向上分割为多个HRegion，一个region由[startkey,endkey)表示

参数hbase.hregion.max.filesize

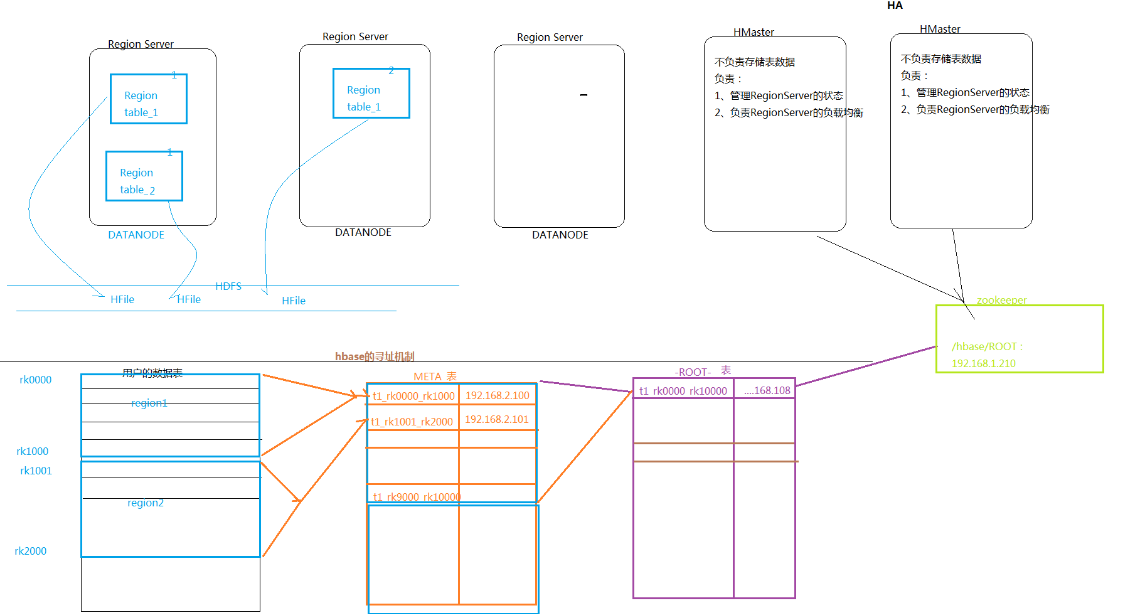






HBASE基础知识

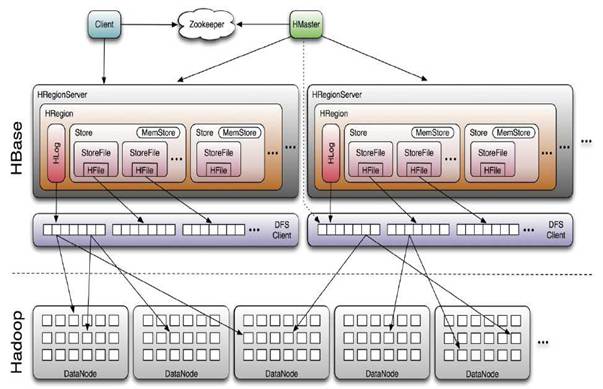
* **架构体系**
  + Client 包含访问hbase 的接口，client 维护着一些cache 来加快对hbase 的访问，比如region 的位置信息
  + Zookeeper
    - 保证任何时候，集群中只有一个running master
    - 存贮所有Region的寻址入口
    - 实时监控Region Server的状态，将Region server的上线和下线信息，实时通知给Master
    - 存储Hbase 的schema,包括有哪些table，每个table有哪些column family
  + Master 可以启动多个HMaster，HMaster不存储region，通过Zookeeper的Master Election机制保证总有一个Master运行
    - 为Region server 分配region
    - 负责region server 的负载均衡
    - 发现失效的region server 并重新分配其上的region



Region Server

* 维护Master 分配给它的region，处理对这些region的IO 请求
* 负责合并较小的region，切分在运行过程中变得过大的region

可以看出，client 访问hbase 上数据的过程并不需要master 参与，寻址访问先zookeeper再regionserver，数据读写访问regioneserver。HRegionServer主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBase中最核心的模块。 、



HBASE与相关软件

* hadoop.1.1.2
* hbase-0.94.2-security.tar.gz
* JDK6
* RHEL6.3

**前提条件：本机或集群环境下hadoop.1.1.2已经安装成功**

本地伪分布模式搭建（一）

* 把hbase-0.94.2-security.tar.gz复制到/usr/local
* 解压hbase-0.94.2-security.tar.gz与重命名

#cd /usr/local

#tar -zxvf hbase-0.94.2-security.tar.gz

#mv hbase-0.94.2-security hbase

修改/etc/profile文件。

#vi /etc/profile

增加

export HBASE\_HOME=/home/hbase

修改

export PATH=$JAVA\_HOME/bin:$PATH:$HADOOP\_HOME/bin:$HBASE\_HOME/bin

保存退出

#source /etc/profile

本地伪分布模式搭建（二）

修改$**HBASE\_HOME/conf/hbase-env.sh文件**

**export JAVA\_HOME=/usr/local/jdk**

**export HBASE\_MANAGES\_ZK=true**

**保存后退出**

**修改$HBASE\_HOME/conf/hbase-site.xml**

<property>

<name>hbase.rootdir</name>

<value>hdfs://hadoop0:9000/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

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

<value>hadoop0</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

**注意：$HBASE\_HOME/conf/hbase-site.xml的hbase.rootdir的主机和端口号与$HADOOP\_HOME/conf/core-site.xml的fs.default.name的主机和端口号一致**

本地伪分布模式搭建（三）

在$HBASE\_HOME/conf/regionservers文件增加

localhost

保存退出

先启动Hadoop，后启动HBASE

#cd $HBASE\_HOME/bin

#./start-hbase.sh

停止

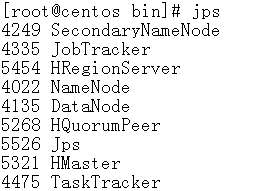
#cd $HBASE\_HOME/bin

#./stop-hbase.sh

本地伪分布模式搭建（三）

本地伪分布模式搭建（四）

Jps:

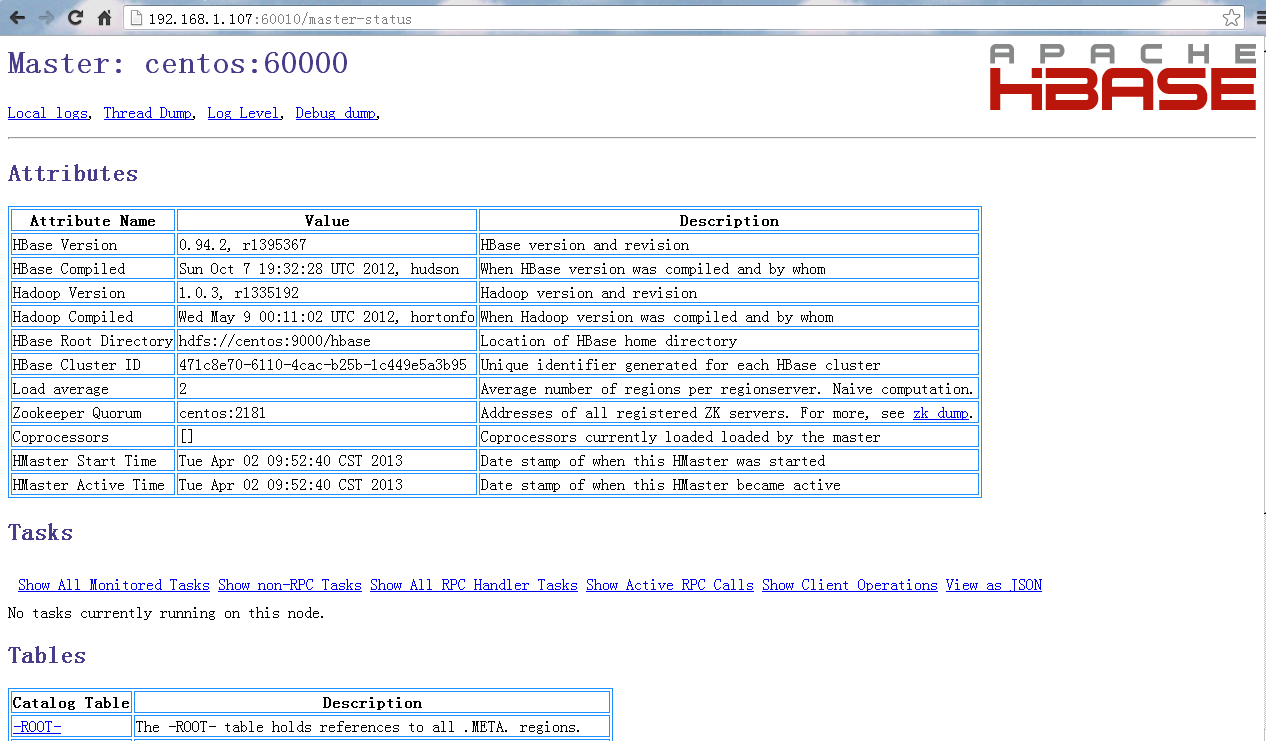


查看hdfs目录，你会发现在根目录下多了一个hbase的目录

#hadoop fs -ls /

Web Console

http://hadoop0:60010/master-status



HBASE Shell

|  |  |
| --- | --- |
| 名称 | 命令表达式 |
| 创建表 | create '表名称', '列族名称1','列族名称2','列族名称N' |
| 添加记录 | put '表名称', '行名称', '列名称:', '值' |
| 查看记录 | get '表名称', '行名称' |
| 查看表中的记录总数 | count '表名称' |
| 删除记录 | delete '表名' ,'行名称' , '列名称' |
| 删除一张表 | 先要屏蔽该表，才能对该表进行删除，第一步 disable '表名称' 第二步 drop '表名称' |
| 查看所有记录 | scan "表名称" |
| 查看某个表某个列中所有数据 | scan "表名称" , {COLUMNS=>'列族名称:列名称'} |
| 更新记录 | 就是重写一遍进行覆盖，值更新到了新的版本 |

HBASE Shell的DDL操作

* 创建表，需要在创建表的时候指明列族，列族一旦确定就不能更改，因为hdfs对修改操作很繁琐

>create 'users','user\_id','address','info'

*表users,有三个列族user\_id,address,info*

* 列出全部表

>list

* 得到表的描述

>describe 'users'

* 创建表

>create 'users\_tmp','user\_id','address','info'

* 删除表

>disable 'users\_tmp'

>drop 'users\_tmp'

#$HBASE\_HOME/bin/hbase shell

……

>quit

查看表是否存在

exists 'users'

>is\_enabled 'users'

>is\_disabled 'users

HBASE Shell的DML操作

* 添加记录
* 获取一条记录

1.取得一个id的所有数据

>get 'users','xiaoming'

2.获取一个id，一个列族的所有数据

>get 'users','xiaoming','info'

3.获取一个id，一个列族中一个列的

所有数据

get 'users','xiaoming','info:age'

put 'users','xiaoming','info:age','24';

put 'users','xiaoming','info:birthday','1987-06-17';

put 'users','xiaoming','info:company','alibaba';

put 'users','xiaoming','address:contry','china';

put 'users','xiaoming','address:province','zhejiang';

put 'users','xiaoming','address:city','hangzhou';

put 'users','zhangyifei','info:birthday','1987-4-17';

put 'users','zhangyifei','info:favorite','movie';

put 'users','zhangyifei','info:company','alibaba';

put 'users','zhangyifei','address:contry','china';

put 'users','zhangyifei','address:province','guangdong';

put 'users','zhangyifei','address:city','jieyang';

put 'users','zhangyifei','address:town','xianqiao';

* 更新记录，不是在原有的基础上修改，而是生成新的版本直接覆盖掉之前的版本

>put 'users','xiaoming','info:age' ,'29'

>get 'users','xiaoming','info:age'

>put 'users','xiaoming','info:age' ,'30'

>get 'users','xiaoming','info:age'

* 获取单元格数据的版本数据

>get 'users','xiaoming',{COLUMN=>'info:age',VERSIONS=>1}

>get 'users','xiaoming',{COLUMN=>'info:age',VERSIONS=>2}

>get 'users','xiaoming',{COLUMN=>'info:age',VERSIONS=>3}

* 获取单元格数据的某个版本数据

〉get 'users','xiaoming',{COLUMN=>'info:age',TIMESTAMP=>1364874937056}

* 全表扫描

>scan 'users'

* 删除xiaoming值的'info:age'字段

>delete 'users','xiaoming','info:age'

>get 'users','xiaoming'

* 删除整行

>deleteall 'users','xiaoming'

* 统计表的行数

>count 'users'

* 清空表

>truncate 'users'

HBASE基础知识

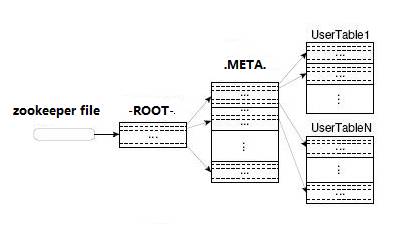
HBase中有两张特殊的Table，**-ROOT-和.META.**

**-ROOT-** ：记录了.META.表的Region信息，-ROOT-只有一个region

**.META**. ：记录了用户创建的表的Region信息，.META.可以有多个regoin

Zookeeper中记录了-ROOT-表的location

Client访问用户数据之前需要首先访问zookeeper，然后访问-ROOT-表，接着访问.META.表，最后才能找到用户数据的位置去访问



HBASE的Java\_API（一）

//hbase操作必备

private static Configuration getConfiguration() {

Configuration conf = HBaseConfiguration.create();

conf.set("hbase.rootdir", "hdfs://hadoop0:9000/hbase");

//使用eclipse时必须添加这个，否则无法定位

conf.set("hbase.zookeeper.quorum", "hadoop0");

return conf;

}

HBASE的Java\_API（二）

//创建一张表

public static void create(String tableName, String columnFamily) throws IOException{

HBaseAdmin admin = new HBaseAdmin(getConfiguration());

if (admin.tableExists(tableName)) {

System.out.println("table exists!");

}else{

HTableDescriptor tableDesc = new HTableDescriptor(tableName);

tableDesc.addFamily(new HColumnDescriptor(columnFamily));

admin.createTable(tableDesc);

System.out.println("create table success!");

}

}

HBASE的Java\_API（三）

//添加一条记录

public static void put(String tableName, String row, String columnFamily, String column, String data) throws IOException{

HTable table = new HTable(getConfiguration(), tableName);

Put p1 = new Put(Bytes.toBytes(row));

p1.add(Bytes.toBytes(columnFamily), Bytes.toBytes(column), Bytes.toBytes(data));

table.put(p1);

System.out.println("put'"+row+"',"+columnFamily+":"+column+"','"+data+"'");

}

HBASE的Java\_API（四）

//读取一条记录

public static void get(String tableName, String row) throws IOException{

HTable table = new HTable(getConfiguration(), tableName);

Get get = new Get(Bytes.toBytes(row));

Result result = table.get(get);

System.out.println("Get: "+result);

}

HBASE的Java\_API（五）

//显示所有数据

public static void scan(String tableName) throws IOException{

HTable table = new HTable(getConfiguration(), tableName);

Scan scan = new Scan();

ResultScanner scanner = table.getScanner(scan);

for (Result result : scanner) {

System.out.println("Scan: "+result);

}

}

HBASE的Java\_API（六）

//删除表

public static void delete(String tableName) throws IOException{

HBaseAdmin admin = new HBaseAdmin(getConfiguration());

if(admin.tableExists(tableName)){

try {

admin.disableTable(tableName);

admin.deleteTable(tableName);

} catch (IOException e) {

e.printStackTrace();

System.out.println("Delete "+tableName+" 失败");

}

}

System.out.println("Delete "+tableName+" 成功");

}

HBASE的Java\_API（七）

public static void main(String[] args) throws IOException {

String tableName="hbase\_tb";

String columnFamily="cf";

HBaseTestCase.create(tableName, columnFamily);

HBaseTestCase.put(tableName, "row1", columnFamily, "cl1", "data");

HBaseTestCase.get(tableName, "row1");

HBaseTestCase.scan(tableName);

HBaseTestCase.delete(tableName);

}

练习——详单入库

* HBASE表定义为：

>create 'wlan\_log', 'cf'



RowKey设计：

msisdn:日期时间串（yyyyMMddHHmmss）

* 源文件数据增加一个字段rowkey

HBASE结合MapReduce批量导入

static class BatchImportMapper extends Mapper<LongWritable, Text, LongWritable, Text>{

SimpleDateFormat dateformat1=new SimpleDateFormat("yyyyMMddHHmmss");

Text v2 = new Text();

protected void map(LongWritable key, Text value, Context context) throws java.io.IOException ,InterruptedException {

final String[] splited = value.toString().split("\t");

try {

final Date date = new Date(Long.parseLong(splited[0].trim()));

final String dateFormat = dateformat1.format(date);

String rowKey = splited[1]+":"+dateFormat;

v2.set(rowKey+"\t"+value.toString());

context.write(key, v2);

} catch (NumberFormatException e) {

final Counter counter = context.getCounter("BatchImport", "ErrorFormat");

counter.increment(1L);

System.out.println("出错了"+splited[0]+" "+e.getMessage());

}

};

}

HBASE结合MapReduce批量导入

static class BatchImportReducer extends TableReducer<LongWritable, Text, NullWritable>{

protected void reduce(LongWritable key, java.lang.Iterable<Text> values, Context context) throws java.io.IOException ,InterruptedException {

for (Text text : values) {

final String[] splited = text.toString().split("\t");

final Put put = new Put(Bytes.toBytes(splited[0]));

put.add(Bytes.toBytes("cf"), Bytes.toBytes("date"), Bytes.toBytes(splited[1]));

//省略其他字段，调用put.add(....)即可

context.write(NullWritable.get(), put);

}

};

}

HBASE结合MapReduce批量导入

public static void main(String[] args) throws Exception {

final Configuration configuration = new Configuration();

//设置zookeeper

configuration.set("hbase.zookeeper.quorum", "hadoop0");

//设置hbase表名称

configuration.set(TableOutputFormat.OUTPUT\_TABLE, "wlan\_log");

//将该值改大，防止hbase超时退出

configuration.set("dfs.socket.timeout", "180000");

final Job job = new Job(configuration, "HBaseBatchImport");

job.setMapperClass(BatchImportMapper.class);

job.setReducerClass(BatchImportReducer.class);

//设置map的输出，不设置reduce的输出类型

job.setMapOutputKeyClass(LongWritable.class);

job.setMapOutputValueClass(Text.class);

job.setInputFormatClass(TextInputFormat.class);

//不再设置输出路径，而是设置输出格式类型

job.setOutputFormatClass(TableOutputFormat.class);

FileInputFormat.setInputPaths(job, "hdfs://hadoop0:9000/input");

job.waitForCompletion(true);

}

HBASE的Java\_API 练习

* 查询

1. 按RowKey查询
2. 按手机号码查询
3. 按手机号码的区域查询

HBASE的Java\_API练习

**查询手机13450456688的所有上网记录**

public static void scan(String tableName) throws IOException{

HTable table = new HTable(getConfiguration(), tableName);

Scan scan = new Scan();

scan.setStartRow(Bytes.toBytes("13450456688:/"));

scan.setStopRow(Bytes.toBytes("13450456688::"));

ResultScanner scanner = table.getScanner(scan);

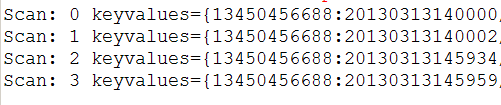
int i=0;

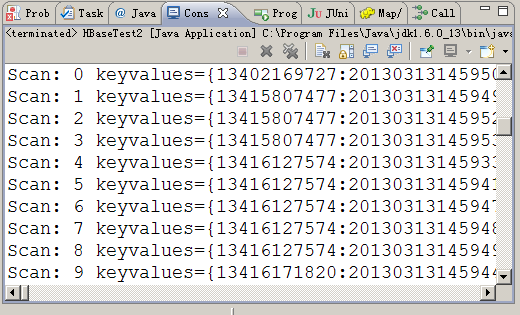
for (Result result : scanner) {

System.out.println("Scan: "+i+++" "+result);

}

}





**查询134号段的所有上网记录**

public static void scanPeriod(String tableName) throws IOException{

HTable table = new HTable(getConfiguration(), tableName);

Scan scan = new Scan();

scan.setStartRow(Bytes.toBytes("134/"));

scan.setStopRow( Bytes.toBytes("134:"));

scan.setMaxVersions(1);

ResultScanner scanner = table.getScanner(scan);

int i=0;

for (Result result : scanner) {

System.out.println("Scan: "+i+++" "+result);

}

}

思考题

* HBASE是什么数据库，与普通RDBMS有什么区别
* HBASE的结构
* HBASE的常用命令

