Day18_rowKey的设计 & 二级索引 & Phoenix

大数据-张军锋 Day18 rowKey的设计 二级索引 Phoenix

rowkey设计

HBase是一个分布式的、面向列的数据库，它和一般关系型数据库的最大区别是：HBase很适合于存储非结构化的数据，还有就是它基于列的而不是基于行的模式。
既然HBase是采用KeyValue的列存储，那Rowkey就是KeyValue的Key了，表示唯一一行。Rowkey也是一段二进制码流，最大长度为64KB，内容可以由使用的用户自定义。数据加载时，一般也是根据Rowkey的二进制序由小到大进行的。
HBase是根据Rowkey来进行检索的，系统通过找到某个Rowkey (或者某个 Rowkey 范围)所在的Region，然后将查询数据的请求路由到该Region获取数据。HBase的检索支持3种方式：
（1）通过单个Rowkey访问，即按照某个Rowkey键值进行get操作，这样获取唯一一条记录；
（2）通过Rowkey的range进行scan，即通过设置startRowKey和endRowKey，在这个范围内进行扫描。这样可以按指定的条件获取一批记录；
（3）全表扫描，即直接扫描整张表中所有行记录。
HBASE按单个Rowkey检索的效率是很高的，耗时在1毫秒以下，每秒钟可获取1000~2000条记录，不过非key列的查询很慢。

1. 数据的存储
2. region划分
3. rowkey是唯一标识
4. rowkey是表中唯一的索引，不支持其他的索引

rowkey设计原则

- rowkey长度原则

rowkey长度最大64k，使用时不能超过100个字节，rowkey的长度定长，rowkey的长度最好是8的整数倍

- rowkey散列原则

rowkey散列的方法：

1. 随机数
2. uuid
3. MD5,hash等加密算法
4. 业务有序数据反向(对业务有序数据进行 reverse)

rowkey唯一性原则

rowkey作为索引原则
rowkey是hbase里面唯一的索引，对于查询频繁的限定条件需要把内容放在rowkwy里面

```java
import java.io.IOException;
import java.nio.ByteBuffer;

import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellScanner;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.CompareOperator;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.Table;
import org.apache.hadoop.hbase.filter.RegexStringComparator;
import org.apache.hadoop.hbase.filter.RowFilter;
import org.apache.hadoop.hbase.util.Bytes;

/**
* 项目名称：mrhbase
* 类名称：PersonInfo
* 类描述：以个人信息为例，展示rowkey的生成策略
* @author Allen
*/
public class PersonInfo {
    /**
    * getRowKey rowkey生成策略 ,rowkey一般是字符串
    * 格式为:idcard(18byte) + name(30bytes)
    * @param @param idCard 身份证号码
    * @param @param name 姓名
    * @return byte[] 返回类型
    * @Exception 异常对象
    * @author Allen
    */
    public byte[] getRowKey(String idCard,String name){
        byte[] idCardBytes = Bytes.toBytes(idCard);
        if(idCardBytes.length!= 18){
            System.out.println("身份证位数不对!");
            return null;
        }
        byte[] nameBytes = Bytes.toBytes(name);
        ByteBuffer result = ByteBuffer.allocate(48);
        result.put(idCardBytes);
        result.put(nameBytes);
```

```
//        System.out.println(result.position());
          byte blank = 0x1F;
          while(result.position() < 48){
              result.put(blank);
          }
//        System.out.println(result.position());
          return result.array();
      }
      public Put generatePersonInfo(){
          String idCard = "412824199404201234";
          String name = "张三";
          Put put = new Put(getRowKey(idCard, name));
          put.addColumn("i".getBytes(), "gender".getBytes(), "男".getB
ytes());
          put.addColumn("i".getBytes(), "age".getBytes(), "28".getByt
es());
          return put;
      }

      /**
      * getDataByName 通过名称获取数据
      * @param @param table Table对象
      * @param @param name 姓名
      * @param @throws IOException 参数
      * @return void 返回类型
      * @Exception 异常对象
      * @author Allen
      */
      public void getDataByName(Table table,String name) throws IOExc
eption{
          RowFilter filter = new RowFilter(CompareOperator.EQUAL, new
RegexStringComparator(name));
          Scan scan = new Scan();
          scan.setFilter(filter);
          ResultScanner rs = table.getScanner(scan);
          Result result = rs.next();
          while(result != null){
              byte[] row = result.getRow();
              String idCard = Bytes.toString(row, 0, 18);
              String username = Bytes.toString(row, 18, 30);
              String age = Bytes.toString(result.getValue(Bytes.toByt
es("i"), Bytes.toBytes("age")));
              String gender = Bytes.toString(result.getValue(Bytes.to
Bytes("i"), Bytes.toBytes("gender")));
              System.out.println("身份证号:" + idCard + "\t姓名:" + use
rname + "\t年龄:" + age + "\t性别:" + gender);
              result = rs.next();
          }
      }
```

```
    public static void main(String[] args) throws IOException {
        PersonInfo personInfo = new PersonInfo();
//      byte[] result = personInfo.getRowKey("412824199404201111",
"张三");
        Connection connection = ConnectionFactory.createConnectio
n(HBaseConfiguration.create());

        // 创建表: create 'bd14:person','i'
        Table person = connection.getTable(TableName.valueOf("bd1
4:person"));

        person.put(personInfo.generatePersonInfo());

        personInfo.getDataByName(person, "张三");
    }
}
```

二级索引

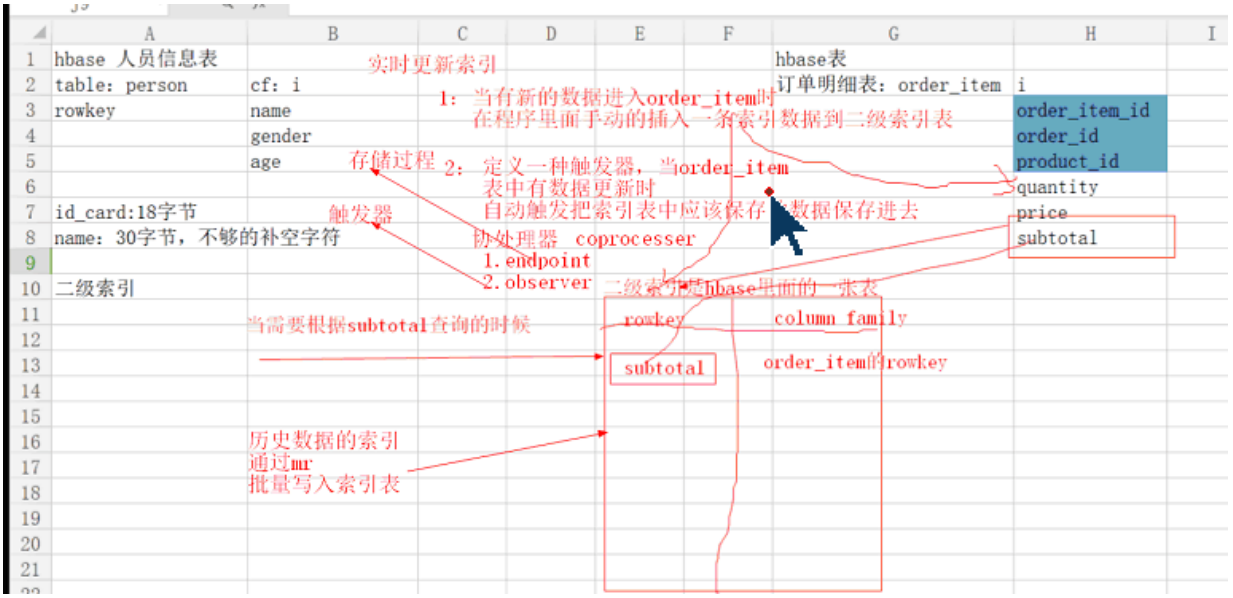
使用索引的目的就是为了增加查询的速度，在hbase中默认支持rowkey索引，但在实际的项目开发过程中，我们往往需要使用其他的索引进行查询我们需要的信息，所以就引入了二级索引的概念

在Hbase中实现二级索引的目的是：

1. 实现高性能的范围查询
2. 数据的低冗余(存储所占的数据量)
3. 数据的一致性

创建二级索引分为两种，一种是在插入数据的同时，直接创建索引，另一种形式是已经存在历史数据，我们创建索引

当历史数据存在时，如果现在我们的需求是，创建subtotal的二级索引；
按照rowkey的生成策略，我们将order_item_id,order_id,product_id作为rowkey，将其他的数据读取到columnFamily中，因为hbase天生不支持二级索引，但是hbase支持rowkey索引，因此，我们只能在rowkey上下点功夫了。我们可以这样做，将suntotal作为rowkey，生成一张与oreder_item对应的表，这样就能索引了。我们将subtotal作为索引，必须通过这个索引找到我们需要的信息才可以，我们将刚才说的那张表的rowkey作为这张表的quality保存到这张表中，这样这两张表之间就有关系了，通过查询就能得到我们需要的数据了。


hbase 人员信息表
table：person
rowkey
cf：i
name
gender
age
id_card:18字节
name：30字节，不够的补空字符
二级索引
实时更新索引
1：当有新的数据进入order_item时
在程序里面手动的插入一条索引数据到二级索引表
存储过程
触发器
2：定义一种触发器，当order_item
表中有数据更新时
自动触发把索引表中应该保存
数据保存进去
协处理器 coprocesser
1.endpoint
2.observer
二级索引是hbase里面的一张表
当需要根据subtotal查询的时候
历史数据的索引
通过mr
批量写入索引表
hbase表
订单明细表：order_item
i
order_item_id
order_id
product_id
quantity
price
subtotal
rowkey
column family
subtotal
order_item的rowkey

- 创建表一

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Mutation;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

/**
* 项目名称：mrhbase
* 类名称：OrderToHBase
* 类描述：将订单数据导入导入到 bd14:order
* @author Allen
*/
public class OrderToHBase {

    public static class OrderToHBaseMapper extends Mapper<LongWrita
ble, Text, Text, Text> {
        private Text outKey = new Text();
        private Text outValue = new Text();
        private String[] infos;

        @Override
        protected void map(LongWritable key, Text value, Mapper<Lon
gWritable, Text, Text, Text>.Context context)
                throws IOException, InterruptedException {
            infos = value.toString().split("\\|");
            outKey.set(infos[0]);
            outValue.set(infos[1] + "--" + infos[2] + "--" + info
s[3]);
            context.write(outKey, outValue);
        }
    }

    public static class OrderToHBaseReducer extends TableReducer<Te
xt, Text, NullWritable> {
        private NullWritable outputKey = NullWritable.get();
        private Put outputValue;
        private String[] infos;
```

```
        @Override
        protected void reduce(Text key, Iterable<Text> values,
                Reducer<Text, Text, NullWritable, Mutation>.Context
context) throws IOException, InterruptedException {
            if (!key.toString().equals("")) {
                for (Text value : values) {
                    infos = value.toString().split("--");
                    System.out.println("数据: "+infos[0] + "--" + in
fos[1] + "--" + infos[2]);
                    outputValue = new Put(key.toString().getByte
s());
                    outputValue.addColumn("i".getBytes(), "produc
t_id".getBytes(), infos[0].getBytes());
                    outputValue.addColumn("i".getBytes(), "quantit
y".getBytes(), infos[1].getBytes());
                    outputValue.addColumn("i".getBytes(), "subtota
l".getBytes(), infos[2].getBytes());
                    outputValue.addColumn("i".getBytes(), "produc
t_price".getBytes(), infos[3].getBytes());
                    context.write(outputKey, outputValue);
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        Job job = Job.getInstance(conf);
        job.setJarByClass(OrderToHBase.class);
        job.setJobName("将order的数据写入到hbase");

        job.setMapperClass(OrderToHBaseMapper.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        TableMapReduceUtil.initTableReducerJob("bd14:orders", Order
ToHBaseReducer.class, job);
        FileInputFormat.addInputPath(job, new Path("/orderdata/orde
r_items"));

        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }
}
```

- 创建表二

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.CellScanner;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Mutation;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.io.ImmutableBytesWritable;
import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;
import org.apache.hadoop.hbase.mapreduce.TableMapper;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

public class MrFromHBaseToHBase {

    public static class MrFromHBaseToHBaseMapper extends TableMappe
r<Text, Text>{
        private Text outputKey = new Text();
        private Text outputValue = new Text();
        private Cell cell;
        private String rowkey;
        private String columnFamily;
        private String columnQualifier;
        private String columnValue;
        @Override
        protected void map(ImmutableBytesWritable key, Result valu
e,
                Mapper<ImmutableBytesWritable, Result, Text, Tex
t>.Context context)
                throws IOException, InterruptedException {
            CellScanner cellScanner = value.cellScanner();
            while(cellScanner.advance()){
                cell = cellScanner.current();
                rowkey = Bytes.toString(CellUtil.cloneRow(cell));
                columnFamily = Bytes.toString(CellUtil.cloneFamil
y(cell));
                columnQualifier = Bytes.toString(CellUtil.cloneQual
ifier(cell));
                columnValue = Bytes.toString(CellUtil.cloneValue(ce
```

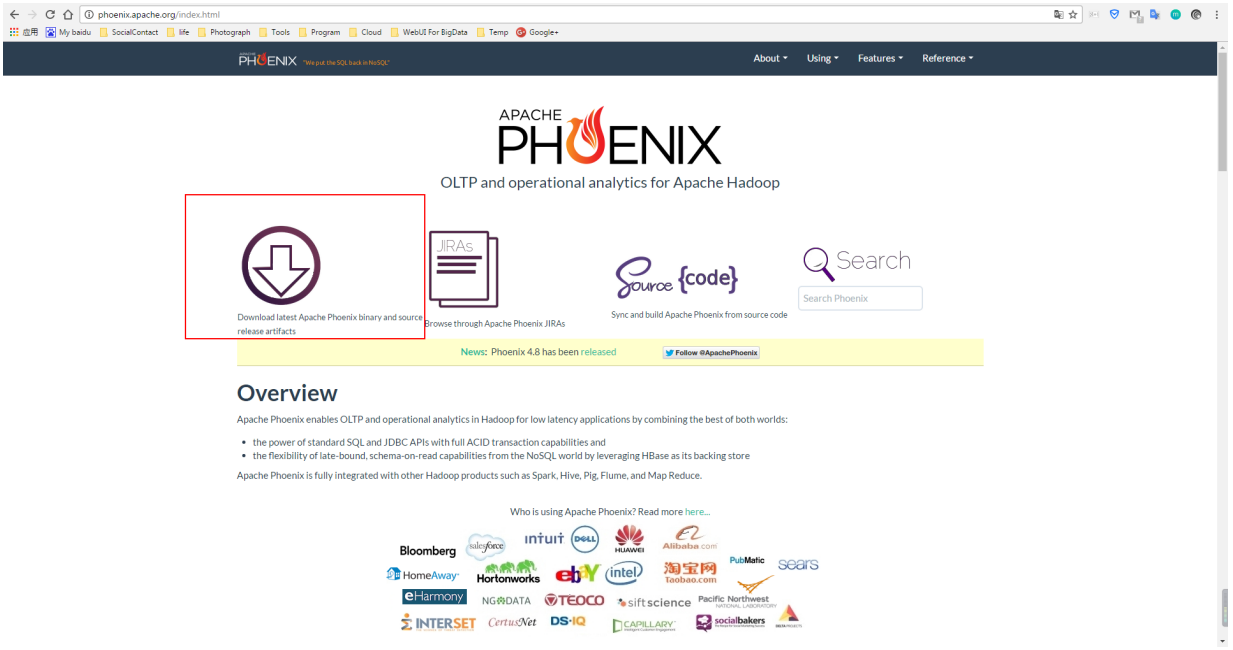
```
ll));
                outputKey.set(rowkey);
                outputValue.set(columnFamily + "--" + columnQualifi
er + "--" + columnValue);
                context.write(outputKey, outputValue);
            }
        }
    }
    public static class MrFromHBaseToHBaseReducer extends TableRedu
cer<Text, Text, NullWritable>{
        private NullWritable outputKey = NullWritable.get();
        private Put outputValue;
        private String[] infos;
        @Override
        protected void reduce(Text key, Iterable<Text> values, Redu
cer<Text, Text, NullWritable, Mutation>.Context context)
                throws IOException, InterruptedException {
            for (Text value : values) {
                infos = value.toString().split("--");
                if(infos[1].equals("subtotal")){
                    outputValue = new Put(infos[2].getBytes());
                    outputValue.addColumn("i".getBytes(), key.toStr
ing().getBytes(), "".getBytes());
                    context.write(outputKey, outputValue);
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        Job job = Job.getInstance(conf);
        job.setJarByClass(MrFromHBaseToHBase.class);
        job.setJobName("MrFromHBaseToHBase");

        Scan scan = new Scan();
        TableMapReduceUtil.initTableMapperJob("bd14:sample_order_it
em", scan, MrFromHBaseToHBaseMapper.class, Text.class, Text.class,
                job);
        TableMapReduceUtil.initTableReducerJob("bd14:sindex", MrFro
mHBaseToHBaseReducer.class, job);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

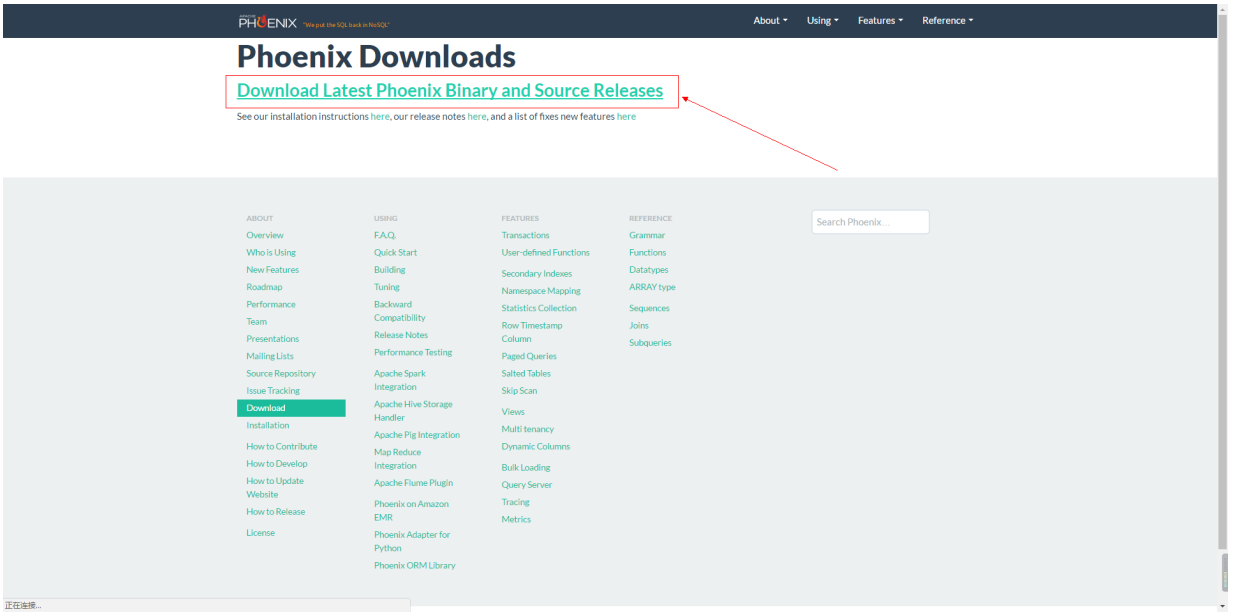
Phoenix

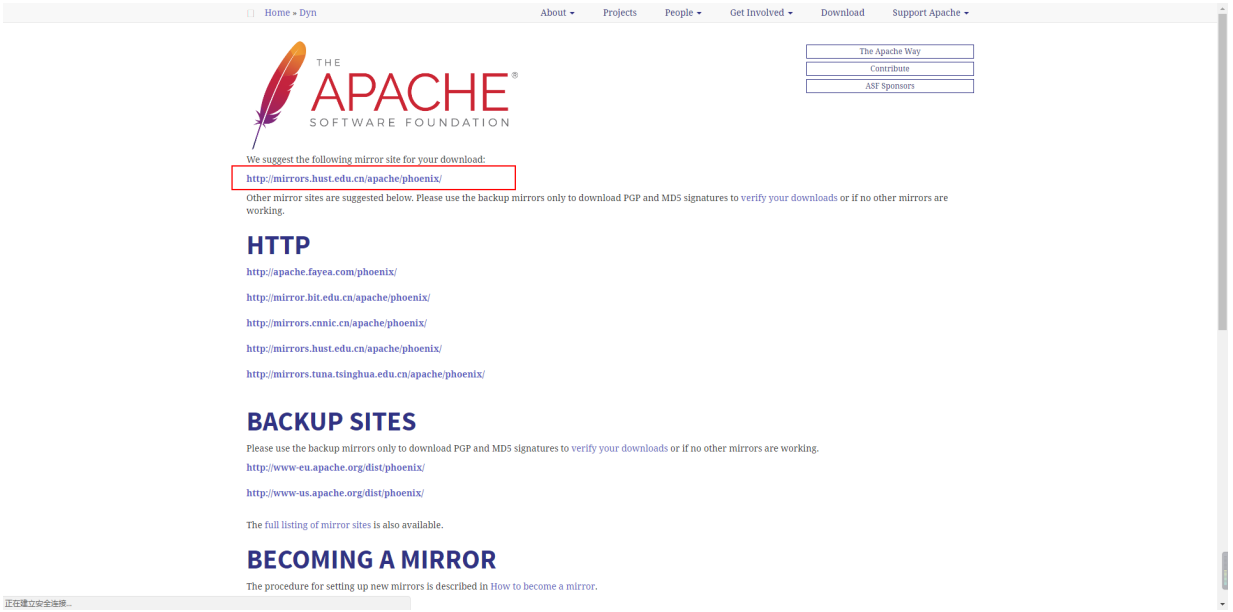
在我们开发的过程中对于数据测查询操作，我们是喜欢使用sql来进行查询操作的，但是hbase并不支持sql的查询操作。因此，我们就引入了Phoenix。Phoenix是hbase的sql引擎，它帮我们将sql解析成hbase能够读懂的查询条件，这样我们的就爽了。

Phoenix安装教程

1. Phoenix需要python进行解析，因此我们需要python的环境设置，使用命令`yum install python-argparse`安装即可。
2. 官网上下载安装包 http://mirror.bit.edu.cn/apache/phoenix/







3. 解压 `tar -zxvf 安装包名称`
4. 配置环境变量 `vim /etc/profile`

```
export HADOOP_HOME=/opt/software/hadoop/hadoop-2.7.4
export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_CLASSPATH=$:HADOOP_HOME/lib/toos.jar

# set hive environment
export HIVE_HOME=/opt/software/hive/apache-hive-2.3.0-bin
export PATH=$PATH:$HIVE_HOME/bin

# set zookeeper environment
export ZOOKEEPER=/opt/software/zookepper/zookeeper-3.5.2-alpha
export PATH=$PATH:$ZOOKEEPER/bin

# set hbase environment
export HBASE_HOME=/opt/software/hbase/hbase-2.0.0-alpha3
export PATH=$PATH:$HBASE_HOME/bin

# set scala environment
export SCALA_HOME=/opt/software/scala/scala-2.12.4
export PATH=$PATH:$SCALA_HOME/bin

# set spark environment
export SPARK_HOME=/opt/software/spark/spark-2.2.0-bin-hadoop2.7
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

f set phoenix enviroment
export PHOENIX_HOME=/opt/software/phoenix/apache-phoenix-4.12.0-HBase-1.3-bin
export PHOENIX_CLASSPATH=$PHOENIX_HOME/lib
export PATH=$PATH:$PHOENIX_HOME/bin
"/etc/profile" 110L, 2831C                                    110,1         Bot
```

5. 将/usr/tools/phoenix-4.7.0-HBase-1.1-bin目录下面的phoenix-4.7.0-HBase-1.1-server.jar文件拷贝到每一台HRegionServer的hbase安装目录的lib目录下面去

```
1.cp phoenix-4.7.0-HBase-1.1-server.jar /usr/tools/hbase-1.2.0/lib/
2.scp phoenix-4.7.0-HBase-1.1-server.jar root@jokeros2:/usr/tools/hbase-1.2.0/lib/
3.scp phoenix-4.7.0-HBase-1.1-server.jar root@jokeros3:/usr/tools/hbase-1.2.0/lib/
```

重新启动hbase

stop-hbase.sh

start-hbase.sh

6. 进入phoenix目录的bin目录下面，`sqlline.py jokeros1,jokeros2,jokeros3:2181`