# 1 安装hbase

先确保已经安装hadoop java zookeeper

上传解压hbase

## 配置文件（为分布式）

1 、hbase-env.sh

替换： ****export JAVA\_HOME=/usr/lib/jvm/java-1.7.0****

****export HBASE\_MANAGES\_ZK=false****

****export HBASE\_CLASSPATH=/opt/hadoop-2.7.3/etc/hadoop****

1. ****hbase-site.xml****

****<property>****

****<name>hbase.zookeeper.property.dataDir</name>****

****<value>/home/hadoop/zookeeper</value>****

****</property>****

****<property>****

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

****<value>true</value>****

****</property>****

****<property>****

****<name>>hbase.rootdir</name>****

****<value>hdfs://hadoop:9000/hbase</value>****

****</property>****

****<property>****

****<name>dfs.replication</name>****

****<value>1</value>****

****</property>****

****配置region服务器,修改conf/regionservers文件，其中配置所有hbase主机，每个主机名独占一行，hbase启动或关闭时会按照该配置顺序启动或关闭主机中的hbase。****

****使用已有的zookeeper集群。这种方式下zookeeper脱离了hbase，不会随着hbase的启动关闭而启动关闭。需要自己来启动关闭。****

****hbase默认使用自带的zookeeper，如果需要使用外部zookeeper，需要先关闭。****

****修改conf/hbase-env.sh禁用内部zookeeper****

****export HBASE\_MANAGES\_ZK=false****

拷贝zookeeper的配置文件zoo.cfg到HBase的CLASSPATH（官方推荐的方式）

root@master:/data/soft/hbase# cp /data/soft/zookeeper/conf/zoo.cfg  ./conf/

****在hbase-site.xml中配置Zookeeper的连接地址与端口号****

****<property>****

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

****<value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>****

****</property>****

****--------------------------------------------------****

****~HBASE配置文件说明****

****hbase-env.sh配置HBase启动时需要的相关环境变量****

****hbase-site.xml配置HBase基本配置信息****

****HBASE启动时默认使用hbase-default.xml中的配置，如果需要可以修改hbase-site.xml文件，此文件中的配置将会覆盖hbase-default.xml中的配置****

****修改配置后要重启hbase才会起作用****

### **设置类路径**

****export CLASSPATH=$CLASSPATH://home/hadoop/hbase/lib/\*****

****3 启动****

****$cd /usr/local/HBase****

****$bin/start-hbase.sh****

### **启动HBase主服务器：**

****$./bin/local-master-backup.sh start 2 (number signifies specific****

****server.)****

### **启动区域服务：**

****$./bin/./local-regionservers.sh start 3****

### **启动HBase Shell：**

****$cd bin****

****$./hbase shell****

## **启动和停止主服务器**

使用“local-master-backup.sh”就可以启动多达10台服务器。打开HBase的master主文件夹，并执行以下命令来启动它。

****$ ./bin/local-master-backup.sh 2 4****

要中止备份主服务，需要它的进程ID，它被存储在一个文件名为“/tmp/hbase-USER-X-master.pid”中，可以使用下面的命令中止备份主服务。

****$ cat /tmp/hbase-user-1-master.pid |xargs kill -9****

## **启动和停止区域服务器**

可以使用下面的命令来运行在单一系统中的多个区域的服务器。

****$ .bin/local-regionservers.sh start 2 3****

要停止区域服务器，可以使用下面的命令。

****$ .bin/local-regionservers.sh stop 3****

## **HBase的Web界面**

****http://localhost:60010****

# 2 hbase原理

## HBase的工作方式：

hbase中的表在行的方向上分隔为多个HRegion，分散在不同的RegionServer中

这样做的目的是在查询时可以将工作量分布到多个RegionServer中以提高速度

region由[startkey,endkey)表示

HRegion是hbase分布式存储和负载均衡的最小单元

要注意HRegion不在hdfs中，而是在RegionServer的内存中，在内存（其实也有数据在本地文件系统中，因为内存空间毕竟是有限的）中存储数据以提高查询性能，对于修改会将数据最终同步到hdfs中，以持久化数据。

hbase中的数据按照rowkey的字典顺序（字典顺序！！！先比较第一位 如果相同再比较第二位。。。）按序存储，所以当以rowkey查询数据时，可以提高速度。

hregion的分裂，当hregion变的很大时会分裂成两个，存放到不同的RegionServer中，防止一个Region过大，导致查询其中数据时速度很慢

hbase的系统结构：主要有client hmaster regionServer zookeeper

## 为什么hbase可以很快：

从逻辑结构上来说：

表按照行键进行了排序，所以查询时可以很快定位

数据按照行键切分为多个HRegion，分布在多个RegionServer中，查询大量数据时，多个RegionServer可以一起工作，从而提高速度

从物理结构上来说：

HRegion是存活在RegionServer的内存中的，读写会非常的高效

还有HFile的支持保证大量的数据可以持久化的保存

数据最终落地到HDFS中，分布式的存储，保证数据段可靠性和可扩展性

## 为什么hbase可以存储很多数据：

基于hdfs，所以支持可扩展性，可以通过增加大量的廉价的硬件提高存储容量

按列存储，空的数据不占用空间，当存储稀疏数据时，不会浪费空间

按例存储，同一列的数据存放在一起，而同一列的数据一般都是同样的类型的内容相似的数据，可以实现非常高效的压缩，节省空间

## 为什么hbase的数据是可靠的：

基于hdfs，由hdfs的可靠性保证了hbase的可靠性--即数据可以有多个备份

利用zookeeper实现了HA，即使某一台机器挂掉另外的机器也可以很快的替换它

## hbase和hive和传统的关系型数据库的比较：

比起传统的关系型数据库，可以存储半结构化非结构化的数据，可以存储和处理更大级别的数据，提供高效的查询，对于稀疏数据的处理更好，具有更好的横向扩展性，免费开源性价比很高。但是不能支持非常好的事务特性，只支持行级的事务。只能通过行键来查询，表设计时难度更高。而mysql用来存储结构化的数据提供更好的事务控制。

比起hive，hive只是在mapreduce上包了一层壳，本质上还是离线数据的处理的工具，实时查询性能有限，本质上是一个基于hadoop的数据仓库工具，不能支持行级别的新增修改和删除。hbase可以提供实时的数据的处理能力，适用于在线数据查询处理，本质上是一种数据库工具。

# 3 JAVA API和命令

hbase命令行下不能使用删除：

可以使用 ctrl+删除键 来进行删除

或

修改xshell配置：

文件->属性->终端->键盘

->delete键序列[VT220Del]

->backspace键序列[ASCII127]

## 创建表：

create ‘table name’,’column family’

#建表时可以指定VERSIONS，配置的是当前列族在持久化到文件系统中时，要保留几个最新的版本数据，这并不影响内存中的历史数据版本

hbase>create 'testtable',{NAME=>'colfam1',VERSIONS=>3},{NAME=>'colfam2',VERSIONS=>1}

**//关闭api：admin.shutdown();**

|  |
| --- |
| **public void** createTable() **throws** IOException {  Configuration conf= HBaseConfiguration.*create*();  conf.set(**"hbase.zookeeper.quorum"**,**"hadoop:2181"**);  Connection conn=**null**;  conn= ConnectionFactory.*createConnection*(conf);  HBaseAdmin admin= (HBaseAdmin) conn.getAdmin();  //HBaseAdmin admin=**new** HBaseAdmin(conf);旧版api  *//表名对象* TableName name=TableName.*valueOf*(**"javaTab"**);  HTableDescriptor hTable=**new** HTableDescriptor(name);  *//列族描述,* HColumnDescriptor cf2=**new** HColumnDescriptor(**"cf2"**.getBytes());  hTable.addFamily(cf2);  HColumnDescriptor cf3=**new** HColumnDescriptor(**"cf3"**.getBytes());  hTable.addFamily(cf3);  admin.createTable(hTable);  admin.close(); } |

## 列出表和scan：

列出表命令：list

查看表#可以在查询时加上RAW=>true来开启对历史版本数据的查询，VERSIONS=>3指定查询最新的几个版本的数据

hbase>scan 'testtable',{RAW=>true,VERSIONS=>3}

JavaAPI：

HTableDescriptor[] tableDescriptor =admin.listTables();

for (int i=0; i<tableDescriptor.length;i++ ){

System.out.println(tableDescriptor[i].getNameAsString());

}

## 禁用表：

禁用单表： disable ‘emp’

来验证是否禁用：用scan 'emp'

查看是否被禁用：is\_disabled 'table name'

禁用所有匹配给定正则表达式的表：disable\_all 'r.\*'

javaAPI:

Boolean bool = admin.isTableDisabled("emp");

if(!bool){

admin.disableTable("emp");

System.out.println("Table disabled");

}

## 启用表是否存在：

启用一个表：enable 'emp'

查看是否启用：is\_enabled 'table name'

启用所有匹配给定正则表达式的表：disable\_all 'r.\*'

看表是否存在：exists 'emp'

javaAPI：

//查看是否被启用

Boolean bool=admin.isTableEnabled("emp");

//位禁用则禁用它

if(!bool){

admin.enableTable("emp");

System.out.println("Table enabled");}

//表时候存在

boolean bool = admin.tableExists("emp");

## \*描述和修改表：

表描述：describe 'table name' 或者 desc ‘tbalename’

alter用于更改现有表的命令。使用此命令可以更改列族的单元，设定最大数量和删除表范围运算符，并从表中删除列家族。

改变列族单元的最大数目：alter 't1', NAME => 'f1', VERSIONS => 5

使用alter，可以设置和删除表范围，运算符，如MAX\_FILESIZE，READONLY，MEMSTORE\_FLUSHSIZE，DEFERRED\_LOG\_FLUSH等。

设置表为只读：alter 'emp', READONLY

删除表范围运算，从emp表中删除MAX\_FILESIZE:

alter 't1', METHOD => 'table\_att\_unset', NAME => 'MAX\_FILESIZE'

删除列族：alter ‘ table name ’, ‘delete’ => ‘ column family ’

JavaAPI:

TableName name=TableName.valueOf("javaTab");  
HTableDescriptor hTable=new HTableDescriptor(name);  
//列族描述,  
HColumnDescriptor cf2=new HColumnDescriptor("cf2".getBytes());  
hTable.addFamily(cf2);

## 删除表：

删除表之前要先禁用表：disable 'emp' 然后：drop 'emp'

删除所有正则表达式匹配的表：drop\_all ‘t.\*’

JavaAPI:

admin.disableTable("emp1");

admin.deleteTable("emp12");

## 插入数据：

put 'emp','1','personal data:name','raju'

|  |
| --- |
| **public void** insertData() **throws** IOException {  Configuration conf=HBaseConfiguration.*create*();  conf.set(**"hbase.zookeeper.quorum"**,**"hadoop:2181"**);  HTable table=**new** HTable(conf,**"javaTab"**.getBytes());   *//指定行键* Put put=**new** Put(Bytes.*toBytes*(**"rk3"**));  put.add(**"cf3"**.getBytes(),**"c1"**.getBytes(),**"v3"**.getBytes());  put.add(**"cf3"**.getBytes(),**"c2"**.getBytes(),**"v4"**.getBytes());  table.put(put);  table.close(); } |

## 获取数据：

get ’table name’,’row1’

读取指定列：get 'table name', ‘rowid’, {COLUMN => ‘column family:column name ’}

|  |
| --- |
| **public void** get() **throws** IOException {  Configuration conf=HBaseConfiguration.*create*();  conf.set(**"hbase.zookeeper.quorum"**,**"hadoop:2181"**);  HTable table=**new** HTable(conf,**"javaTab"**.getBytes());  //get.addFamily(personal)  Get get=**new** Get(**"rk1"**.getBytes());  get.addColumn(**"cf2"**.getBytes(),**"c1"**.getBytes());  Result result=table.get(get);  **byte**[] bytes=result.getValue(**"cf2"**.getBytes(),**"c1"**.getBytes());  System.***out***.println(**new** String(bytes));  table.close(); } |
| 新api查询单元格：  conn = ConnectionFactory.createConnection(configuration);  table = (HTable) conn.getTable(TableName.valueOf("test"));    scann = table.getScanner(new Scan());  for(Result rs : scann){  System.out.println("该表RowKey为："+new String(rs.getRow()));  for(Cell cell : rs.rawCells()){  System.out.println("列簇为："+new String(CellUtil.cloneFamily(cell)));  System.out.println("列修饰符为："+new String(CellUtil.cloneQualifier(cell)));  System.out.println("值为："+new String(CellUtil.cloneValue(cell)));  } |

## 删除数据：

删除单元格：delete ‘table name’, ‘row’, ‘column name ’, ‘time stamp’

删除表所有单元格：deleteall ‘table name’, ‘row’

|  |
| --- |
| **public void** delete() **throws** IOException {  Configuration conf=HBaseConfiguration.*create*();  conf.set(**"hbase.zookeeper.quorum"**,**"hadoop:2181"**);  HTable table=**new** HTable(conf,**"javaTab"**.getBytes());  *//指定行键* Delete delete=**new** Delete(**"rk1"**.getBytes());  //可选删除列或者列族  delete.deleteColumn(Bytes.toBytes("personal"),Bytes.toBytes("name"));  delete.deleteFamily(Bytes.toBytes("professional"));  table.delete(delete);  table.close(); } |

## 条件查询，批量查询：

|  |
| --- |
| **public void** select() **throws** IOException {  Configuration conf=HBaseConfiguration.*create*();  conf.set(**"hbase.zookeeper.quorum"**,**"hadoop:2181"**);  HTable table=**new** HTable(conf,**"javaTab"**.getBytes());  *//类似于查询条件* Scan scan=**new** Scan();  ResultScanner rs=table.getScanner(scan);  Result r=**null**;  **while**((r=rs.next())!=**null**){   **byte**[] bytes=r.getValue(**"cf2"**.getBytes(),**"c1"**.getBytes());  System.***out***.println(**new** String(bytes));  } } |

## 计数和权限：

计算表的行数：count ‘table name’

禁用删除并重新创建表：truncate 'table name'

grant命令授予特定的权限：grant <user> <permissions> [<table> [<column family> [<column; qualifier>]]

R - 代表读取权限

W - 代表写权限

X - 代表执行权限

C - 代表创建权限

A - 代表管理权限

授予权限：grant 'Tutorialspoint', 'RWXCA'

撤销用户访问表的权限：revoke 'tablename'

列出表的所有权限：user\_permission 'tablename'

# 4 内置过滤器

## \*RowFilter：

筛选出匹配的所有的行，对于这个过滤器的应用场景，是非常直观的：使用BinaryComparator可以筛选出具有某个行键的行，或者通过改变比较运算符（下面的例子中是CompareFilter.CompareOp.EQUAL）来筛选出符合某一条件的多条数据，以下就是筛选出行键为row1的一行数据：

1. Filter rf = new RowFilter(CompareFilter.CompareOp.EQUAL, new BinaryComparator(Bytes.toBytes("row1"))); // OK 筛选出匹配的所有的行

Filter rf = new RowFilter(CompareFilter.CompareOp.EQUAL, new BinaryComparator(Bytes.toBytes("row1"))); // OK 筛选出匹配的所有的行

## \*PrefixFilter：

筛选出具有特定前缀的行键的数据。这个过滤器所实现的功能其实也可以由RowFilter结合RegexComparator来实现，不过这里提供了一种简便的使用方法，以下过滤器就是筛选出行键以row为前缀的所有的行：

Filter pf = new PrefixFilter(Bytes.toBytes("row")); // OK  筛选匹配行键的前缀成功的行

## KeyOnlyFilter：

这个过滤器唯一的功能就是只返回每行的行键，值全部为空，这对于只关注于行键的应用场景来说非常合适，这样忽略掉其值就可以减少传递到客户端的数据量，能起到一定的优化作用：

1. Filter kof = new KeyOnlyFilter(); // OK 返回所有的行，但值全是空

## RandomRowFilter：

从名字上就可以看出其大概的用法，本过滤器的作用就是按照一定的几率（<=0会过滤掉所有的行，>=1会包含所有的行）来返回随机的结果集，对于同样的数据集，多次使用同一个RandomRowFilter会返回不通的结果集，对于需要随机抽取一部分数据的应用场景，可以使用此过滤器：

1. Filter rrf = new RandomRowFilter((float) 0.8); // OK 随机选出一部分的行

## InclusiveStopFilter：

扫描的时候，我们可以设置一个开始行键和一个终止行键，默认情况下，这个行键的返回是前闭后开区间，即包含起始行，单不包含中指行，如果我们想要同时包含起始行和终止行，那么我们可以使用此过滤器：

Filter isf = new InclusiveStopFilter(Bytes.toBytes("row1")); // OK

## FirstKeyOnlyFilter：

如果你只想返回的结果集中只包含第一列的数据，那么这个过滤器能够满足你的要求。它在找到每行的第一列之后会停止扫描，从而使扫描的性能也得到了一定的提升：

Filter fkof = new FirstKeyOnlyFilter(); // OK 筛选出第一个每个第一个单元

## ColumnPrefixFilter：

顾名思义，它是按照列名的前缀来筛选单元格的，如果我们想要对返回的列的前缀加以限制的话，可以使用这个过滤器：

1. Filter cpf = new ColumnPrefixFilter(Bytes.toBytes("qual1")); // OK 筛选出前缀匹配的列

## \*ValueFilter：

1. 按照具体的值来筛选单元格的过滤器，这会把一行中值不能满足的单元格过滤掉，如下面的构造器，对于每一行的一个列，如果其对应的值不包含ROW2\_QUAL1，那么这个列就不会返回给客户端：

Filter vf = new ValueFilter(CompareFilter.CompareOp.EQUAL, new SubstringComparator("ROW2\_QUAL1")); // OK 筛选某个（值的条件满足的）特定的单元格  Filter vf = new ValueFilter(CompareFilter.CompareOp.EQUAL, new SubstringComparator("ROW2\_QUAL1")); // OK 筛选某个（值的条件满足的）特定的单元格

## ColumnCountGetFilter：

这个过滤器来返回每行最多返回多少列，并在遇到一行的列数超过我们所设置的限制值的时候，结束扫描操作：

1 Filter ccf = new ColumnCountGetFilter(2); // OK 如果突然发现一行中的列数超过设定的最大值时，整个扫描操作会停止 Filter ccf = new ColumnCountGetFilter(2); // OK 如果突然发现一行中的列数超过设定的最大值时，整个扫描操作会停止

## SingleColumnValueFilter：

用一列的值决定这一行的数据是否被过滤。在它的具体对象上，可以调用setFilterIfMissing(true)或者setFilterIfMissing(false)，默认的值是false，其作用是，对于咱们要使用作为条件的列，如果这一列本身就不存在，那么如果为true，这样的行将会被过滤掉，如果为false，这样的行会包含在结果集中。

1. SingleColumnValueFilter scvf = new SingleColumnValueFilter(
2. Bytes.toBytes("colfam1"),
3. Bytes.toBytes("qual2"),
4. CompareFilter.CompareOp.NOT\_EQUAL,
5. new SubstringComparator("BOGUS"));
6. scvf.setFilterIfMissing(false);
7. scvf.setLatestVersionOnly(true); // OK

## SingleColumnValueExcludeFilter：

这个与10种的过滤器唯一的区别就是，作为筛选条件的列的不会包含在返回的结

## SkipFilter：

这是一种附加过滤器，其与ValueFilter结合使用，如果发现一行中的某一列不符合条件，那么整行就会被过滤掉：

1. Filter skf = new SkipFilter(vf); // OK 发现某一行中的一列需要过滤时，整个行就会被过滤掉

## WhileMatchFilter

这个过滤器的应用场景也很简单，如果你想要在遇到某种条件数据之前的数据时，就可以使用这个过滤器；当遇到不符合设定条件的数据的时候，整个扫描也就结束了：

1. Filter wmf = new WhileMatchFilter(rf); // OK 类似于Python itertools中的takewhile  Filter wmf = new WhileMatchFilter(rf); // OK 类似于Python itertools中的takewhile

## \*FilterList：

用于综合使用多个过滤器。其有两种关系：FilterList.Operator.MUST\_PASS\_ONE和FilterList.Operator.MUST\_PASS\_ALL，默认的是FilterList.Operator.MUST\_PASS\_ALL，顾名思义，它们分别是AND和OR的关系，并且FilterList可以嵌套使用FilterList，使我们能够表达更多的需求：

1. List<Filter> filters = new ArrayList<Filter>();
2. filters.add(rf);
3. filters.add(vf);
4. FilterList fl = new FilterList(FilterList.Operator.MUST\_PASS\_ALL, filters); // OK 综合使用多个过滤器， AND 和 OR 两种关系

|  |
| --- |
| conn=ConnectionFactory.*createConnection*(**conf**); table= (HTable) conn.getTable(TableName.*valueOf*(**"testtable"**)); Scan scan=**new** Scan(); *//根据正则表达式对行键进行匹配 //Filter filter1=new RowFilter(CompareFilter.CompareOp.EQUAL,new RegexStringComparator("^r.\*?1$"));* SingleColumnValueFilter scvf = **new** SingleColumnValueFilter(  Bytes.*toBytes*(**"colfam1"**),  Bytes.*toBytes*(**"qual2"**),  CompareFilter.CompareOp.***NOT\_EQUAL***,  **new** SubstringComparator(**"BOGUS"**)); scvf.setFilterIfMissing(**false**); scvf.setLatestVersionOnly(**true**); *// OK // OK 如果突然发现一行中的列数超过设定的最大值时，整个扫描操作会停止,不会报错* Filter ccf = **new** ColumnCountGetFilter(2); *// OK 筛选某个（值的条件满足的）特定的单元格* Filter vf = **new** ValueFilter(CompareFilter.CompareOp.***EQUAL***, **new** SubstringComparator(**"ROW2\_QUAL1"**)); *// OK 筛选出前缀匹配的列* Filter cpf = **new** ColumnPrefixFilter(Bytes.*toBytes*(**"qual2"**)); *// OK 筛选出第一个每个第一个单元格* Filter fkof = **new** FirstKeyOnlyFilter(); *// OK 包含了扫描的上限在结果之内* Filter isf = **new** InclusiveStopFilter(Bytes.*toBytes*(**"row1"**)); *// OK 随机选出一部分的行* Filter rrf = **new** RandomRowFilter((**float**) 0.8); *// OK 返回所有的行，但值全是空* Filter kof = **new** KeyOnlyFilter(); *// OK 筛选匹配行键的前缀成功的行* Filter pf = **new** PrefixFilter(Bytes.*toBytes*(**"row"**)); *// OK 筛选出匹配的所有的行* Filter rf = **new** RowFilter(CompareFilter.CompareOp.***NOT\_EQUAL***, **new** BinaryComparator(Bytes.*toBytes*(**"row1"**))); *// OK 类似于Python itertools中的takewhile* Filter wmf = **new** WhileMatchFilter(rf); *// OK 发现某一行中的一列需要过滤时，整个行就会被过滤掉* Filter skf = **new** SkipFilter(vf);   List<Filter> filters = **new** ArrayList<Filter>(); filters.add(rf); filters.add(vf); FilterList fl = **new** FilterList(FilterList.Operator.***MUST\_PASS\_ALL***, filters); *// OK 综合使用多个过滤器， AND 和 OR 两种关系* Filter filter2=**new** ColumnCountGetFilter(1); scan.setFilter(filter2); ResultScanner scanner=table.getScanner(scan); **for**(Result r:scanner){  **for**(Cell cell:r.rawCells()){  System.***out***.println(**"KV"**+cell+**"-----value:"**+Bytes.*toString*(CellUtil.*cloneValue*(cell))  +**"---"**+Bytes.*toString*(CellUtil.*cloneFamily*(cell))+**"---"**+Bytes.*toString*(CellUtil.*cloneQualifier*(cell)));  } } |

# 5.表设计

HBase是用来存放半结构化 非结构化数据的数据库。

## 1 列族设计

在设计hbase表时候，列族不宜过多，尽量的要少使用列族。

经常要在一起查询的数据最好放在一个列族中，尽量的减少跨列族的数据访问。

## 2行键的设计

hbase表中行键是唯一标识一个表的字段，首先要求唯一，另外最好是一些有意义的值，来帮助我们识别表中的数据。所以hbase中的行键是需要进行设计的。

hbase中的行键的设置至关重要，严重的影响hbase的执行效率和查询的便利性。

行键设计的基本原则：

行键必须唯一

必须唯一才能唯一标识数据

行键最好是字符串类型

因为数值类型在不同的系统中处理的方式可能不同

行键必须有意义

这样才能方便数据的查询

行键最好具有固定的长度

不同长度的数据可能会造成自然排序时排序的结果和预期不一致

行键的最佳实践：

长度原则：

行键最多可以达到64KB,但是最好是在10~100字节之间，最好不要超过16字节，越短越好，最好是8字节的整数倍。

散列原则：

行键的设计将会影响数据在hbase表中的排序方式，这会影响region切分后的结果，要注意，在设计行键时应该让经常要查询的数据分散在不同的region中，防止某一个或某几个regionserver称为热点。

有序原则：

行键的设计将会影响数据在hbase表中的排序方式，所以一种策略是将经常连续查询的条件作为行键最前面的数据，这样一来可以方便批量查询

# 6 phoenix

## 6.1 上传phoenix安装包到linux 并解压

将phoenix-4.8.1-HBase-0.98-server.jar

phoenix-4.8.1-HBase-0.98-server.jar拷贝到hbase的lib下

## 6.2 配置环境变量

后重启hbase

## 6.3 启动

Bin/sqlline.py hadoop01,handoop02,hadoop03

如果进不到命令行，重启

## 6.4 使用

1 在cli客户端下操作

2 在jdbc中使用

3 在图形化客户端使用

### 6.4.1 建表

Create table table1( id integer primary key,name varchar);  
 必须有主键，即行键

注意：

A. 在phenix中创建的表在hbase中有会同时创建表

B. 在phoenix中创建的表在hbase中表明会变成大写

C. 在phoenix表中的主键会程辉hbase表的行键

D. 如果不声明，phoenix中的普通列会程辉hbase表中默认列族（列族名为0）中的一列

创建带列族的表：

Create table table2(id integer primary key not null,”cf1”.”column1” varchar);

### 6.4.1 插入和修改

Upsert into table1 values (1，’hello’);

### 6.4.2 查询数据

Select \* from table1;

### 6.4.3 删除数据

Delete from table where id=1;

### 6.4.5 查看表

！Tables 退出：！Quit

### 6.4.6 删除表

Drop table1; 会删除底层hbase中的表

## 6.5 映射hbase中的表到phoenix中

1创建表与hbase中的表同名，并且创建同名的字段

创建表的时候表名用双引号引起来，否则在hbase中表明会变成大写

查询时候也用双引号吧表名引起来

## 6.6 视图

### 6.6.1 创建视图

Create view table1\_view as select \* from table1;

### 6.6.2 删除视图

Drop view table1\_view;

### 6.6.3 创建视图关联hbase已存在的表

Create view “table2\_view”(id varchar primary key,”cf1”.”col1” varchar,”cf2”.”col2” varchar);

表关联，删除phoenix中的表时，hbase中的表页会被删除

视图关联时，删除phoenix中的视图时，hbase中的表不会被删除

## 6.7 phoenix 在jdbc的使用

### 6.7.1 工程jar

导入链接phoenix的jar包：

Phoenix-4.8.1-HBase-0.98-client.jar

6.7.2 代码

|  |
| --- |
| **public static void** main(String[] args) **throws** ClassNotFoundException, SQLException { *// Class.forName("org.apache.phoenix.jdbc.PhoenixDriver");* Connection conn= DriverManager.*getConnection*(**"jdbc:phoenix:hadoop"**);  Statement stat=conn.createStatement();  ResultSet rs=stat.executeQuery(**"SELECT** *\** **FROM TAB1"**);  **while**(rs.next()){  System.***out***.println(rs.getString(**"col1"**));  }  stat.close();  conn.close();  } |

五、Phoenix使用 - 可以在图形化客户端工具中使用

连接Phoenix的工具Squirrel

a) Squirrel是一个连接数据库的客户端工具，一般支持JDBC的数据库都可以用它来连接，如连接mysql

b) 下载Squirrel SQL Client，java –jar squirrel-sql-3.7.1-standard.jar即可

c) 连接Phoenix在Squirrel的安装目录的lib下添加phoenix-4.8.1-HBase-0.98-client.jar包