# 详细使用设计

## Htable实例

CRUD（Create, Read, Update, Delete）

这些方法都是由Htable类提供。

ResultScanner scanner=table.getScanner("CF2".getBytes());

限制扫描范围，因为Hbase中的数据是按列族存储的，如果扫描不读取某个列族，那么整个列族文件就不会被读取。

## 限制扫描范围

1. 通过 setTimestamp(long timestamp)设置详细的时间戳
2. 通过 setTimeRange(long minStamp,long maxStamp)设置 时间范围，进一步对结果进行限制。
3. 使用 setStartTow()、setStopRow()以及setFilter() 进一步限定返回的数据

## ResultScanner类

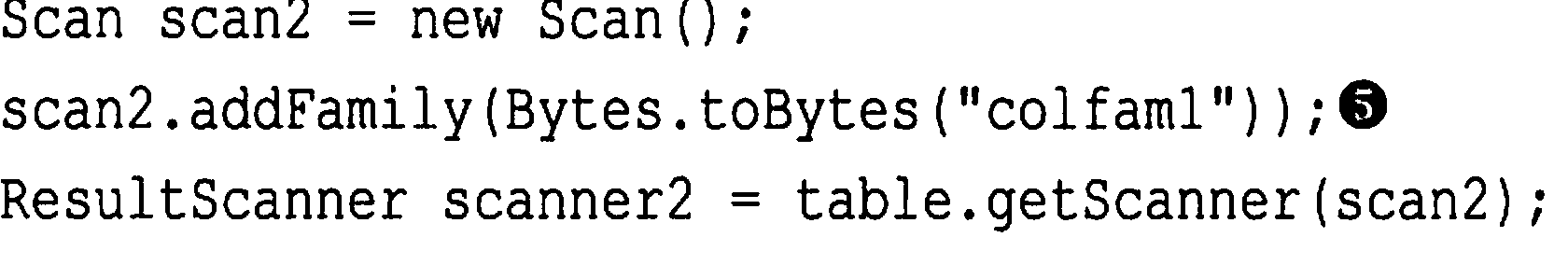
ResulltScanner把扫描操作转换为类似的get操作，它将每一行数据封装成一个Result实例，并将所有的Result实例放入一个迭代器中

### 方法：

Result next() throws IOException

Result[] next(int nbRows) throws IOException 返回Result实例最多可达 nbRows个

Void close() 释放所有扫描控制的资源



## 使用例子

HBaseConfiguration config = Hbase.getHBaseInstance().getHbaseconfig();

HTable table = new HTable(config,"T\_EVENT\_LOG\_RAW" );

ResultScanner scanner=table.getScanner("CF2".getBytes());

使用问一个Htable实例之后，需要调用一次 close()。这个方法会刷写所有客户端缓冲的写操作,close()方法会隐式调用 flushCache()方法。

## 过滤器

get()

读取数据

scan()

它们支持直接访问数据。

Get 和 Scan 两个类都支持过滤器，这类对象的基本API不能对行健、列名或列值进行过滤，但通过过滤器可以到这个目的。

过滤器最底层是Filter 接口和FilterBase抽象类，它们实现了过滤器的空壳和骨架。

注：大部分实体过滤器类一般都直接继承FilterBase

使用流程：用户定义一个所需要的过滤器实例，同时把定义好的多虑器实例传递给Get或 Scan实例； setFilter(filter)

### 三种比较器

BitComparator

RegexStringComparator

三种比较器

SubstringComparator

以上三种方法只能与EQUAL和NOT\_EQUAL运算符搭配使用，匹配时返回0，不匹配返回1，如果与LESS或GREATER运算符搭配使用，会产生错误结果。

### 比较运算符

LESS

LESS\_OR\_EQUAL

EQUAL

CompareFilter中的比较运算符

NOT\_EQUAL

GREATER\_OR\_EQUAL

GREATER

NO\_OP

### RowFilter行过滤器

//rowKey 以"taobao/action.do"结尾的

Filter filter = **new** RowFilter(CompareFilter.CompareOp.*EQUAL*,

**new** SubstringComparator("taobao/action.do"));

scan.setFilter(filter);

Filterr filter= new RowFilter(CompareFilter.CompareOp.LESS\_OR\_EQUAL,new BinaryComparator(Bytes.toBytes(“row-22”)))

scan.setFilter(filter);

ResultScanner scan = table.getScanner(scan)

Filterr filter2=new RowFilter(CompareFilter.CompareOp.EQUAL,new RegexStringComparator(“.\*-.5”);

scan.setFilter(filter2);

ResultScanner scan2= table.getScanner(scan);

前缀过滤器

PrefixFilter prefixFilter = new PrefixFilter("19101148\_".getBytes());

### 单列值过滤器

SingleColumnValueFilter filter = new SingleColumnValueFilter(

"CF1".getBytes(),//Family

"AVGRES".getBytes(), //Column

CompareOp.EQUAL, //Enum

"0.011811953783035279".getBytes());//value

### 列族过滤器 FamilyFilter

Filter filter = new FamilyFilter(CompareFilter.CompareOp.LESS,new BinaryComparator(Bytes.toBytes(“colfam3”)))

scan.setFilter(filter);

Get get= new Get(Bytes.toBytes(“row-5”))

get.setFilter(filter)

Result result=table.get(get) //获取一行数据

### 列名过滤器 QualifierFilter

Filter filter=new QualifierFilter(CompareFilter.CompareOp.LESS\_OR\_EQUAL,new BinaryComparator(Bytes.toBytes(“col-2”)));

scan.setFilter(filter);

### 值过滤器ValueFilter

与RegexStringComparator配合使用，只能与部分运算符搭配，即EQUAL和NOT\_EQUAL运算符

Filter filter= new ValueFilter(CompareFilter.CompareOp.EQUAL,new SubstringComparator(“.4”))

scan.setFilter(filter)

### 分页过滤器 PageFilter

**if**(lastrow!=**null**){

lastrow=Bytes.*add*(lastrow, Bytes.*toBytes*(0));

scan.setStartRow(lastrow);

}

result = table.getScanner(scan);

Filter filter2= new PageFilter(3);

Filter filter= new PageFilter(15);

filterList.addFilter(filter);

filterList.addFilter(filter2);

int totalRows=0;

byte[] lastRow=null;

while(true){

Scan scan = new Scan();

scan.setFilter(filter)

if(lastRow!=null){

byte[] startRow = Bytes.add(lastRow,POSTFIX)

scan.setStartRow(startRow)

}

ResultScanner scanner = table.getScanner(scan);

int localRows = 0;

Result result;

while((result = scanner.next()) != null){

totalRows++;

lastRow = result.getRow();

}

scanner.close();

if(localRows ==0 ) break;

}

### FilterList

FilterList filterList = **new** FilterList();

Filter filter = **new** RowFilter(CompareFilter.CompareOp.*EQUAL*, **new** SubstringComparator("weixin"));

Filter filter2= **new** PageFilter(3);

filterList.addFilter(filter);

filterList.addFilter(filter2);

scan.setFilter(filterList);