# Hbase

提纲

1:hbase简介

2:hbase特点

3:hbase数据模型

4:hbase体系结构

5:hbase存储模型

6:hbase应用

## 简介:

1:Hbase是一个分布式的,多版本的,面向列的开源数据库;

2:Hbase利用Hadoop HDFS作为其文件存储系统,提供高可靠性,高性能,列存储,可伸缩,实时读写,适用于非机构化数据存储的数据库系统;

3:Hbase利用Hadoop MapReduce来处理Hbase中的海量数据

4:Hbase利用Zookeeper作为分布式协同服务;

## hbase特点

1:数据量大:一个表可以有上亿行,上百万列(列多时,插入变慢);

2:面向列:面向列(簇)的存储和全向控制,列(簇)独立检索;

3:稀疏:对于为空(null)的列,并不占用存储空间,因此,表可以设计的非常稀疏;

4:多版本:每个cell中的数据可以有多个版本,默认情况下版本号自动分配,是单元格插入时的时间戳;

5:无类型:Hbase中的数据都是字符串,没有类型;

6:强一致性:同一行数据的读写只在同一台Regin Server上进行

7:有限查询方式:仅支持三种查询方式(单个rowkey查询,通过rowkey的range查询,全表扫描);

8:高性能随机读写;

## hbase数据模型

1:行:同一个key对应的所有数据;

2:列簇:相似的列数据通常被划分成一个列簇,建表时确定;

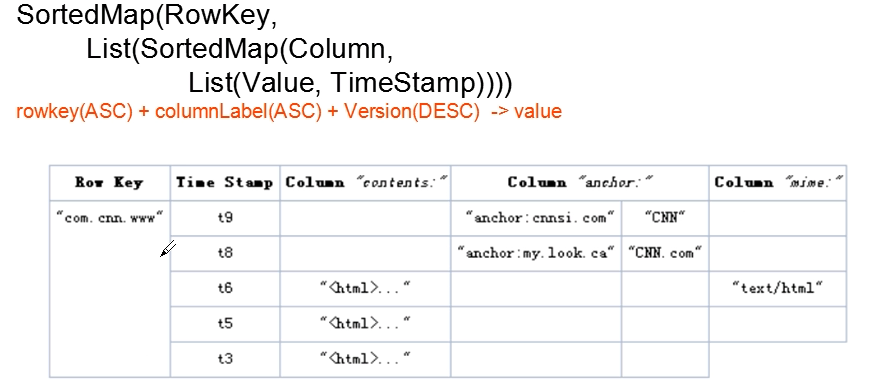
3:列:列名在写入时确定;

4:Cell及时间戳(版本):

每个cell有任意多的版本

建表时设置每个列簇可以保留多个版本;

5:三维有序



## hbase数据模型

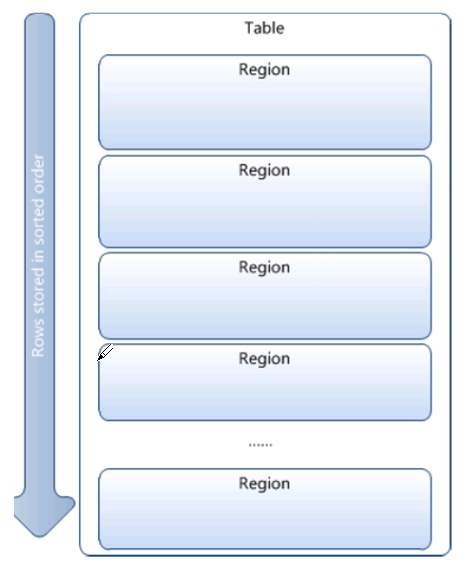
1:面向列的存储

2:一张表可以被分成若干个region

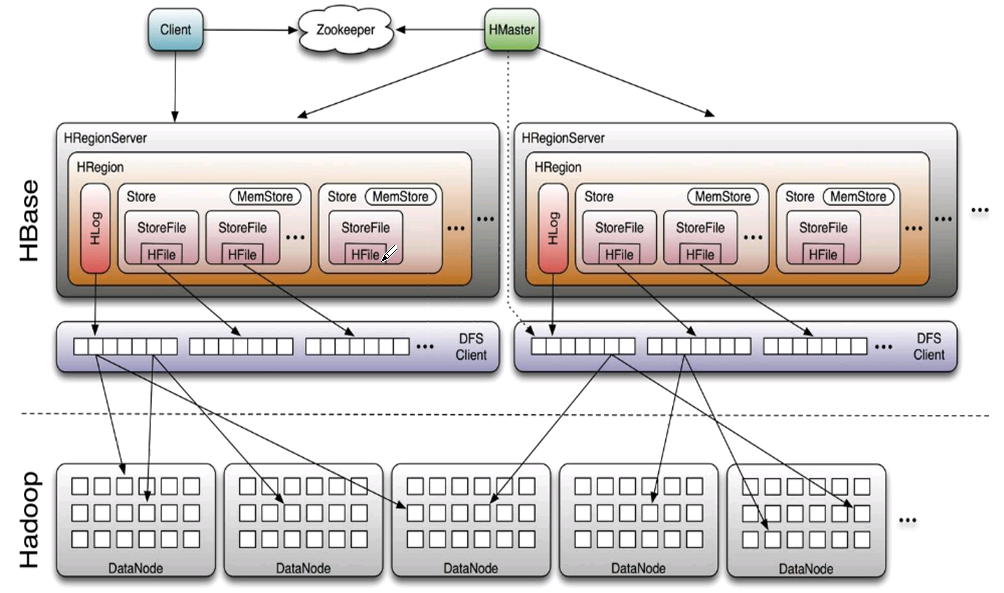
3:行按照rowkey进行字典序排序

4:支持随机读写

5:region是负载均衡调度的最小单位



# hbase体系架构



# hbase体系结构

1:Client

包含访问Hbase的接口并维护cache来加快对Hbase的访问;

2:Zookeeper

保证任何时候,集群中只有一个master;

存储所有Region的寻址入口;

实时监控Region server的上线和下线信息,并实时通知给Master;

存储Hbase的schema和table元数据;

3:Master

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;

管理用户对table的增删改查操作;

4:Region server

负责维护region,处理对这些region的IO请求;

负责切分在运行过程中变得过大的region;

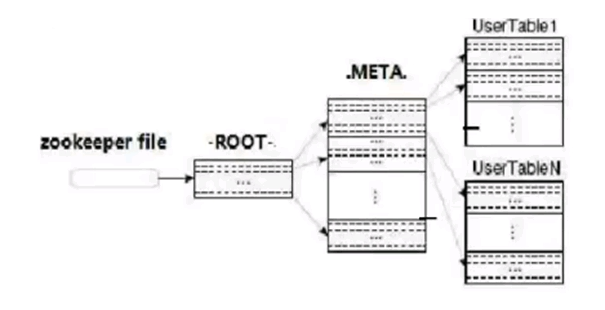
5:root表

记录META表中每个region的位置,root表最多只有一个region

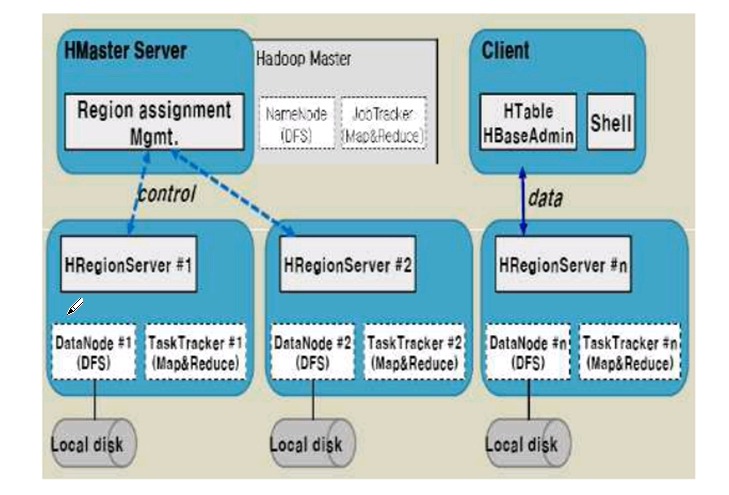
zookeeper中记录了root表的location;

6:META表

记录各个表中每个region所在的region server,META表可能包含多个region;



## Hbase部署模型



Hbase操作

1:flush

内存容量有限,需要定期将内存中的数据flush到磁盘;

每次flush,每个region的每个column family都会产生一个Hfile

读取操作,region server会把多个Hfile数据归并到一起;

2:compaction

flush操作产生的Hfile会越来越低,需要归并来减少Hfile的数量,

旧数据会被清理;

3:split

Hfile大小增长到某个阈值就会split,同时把Region split成两个region,这两个region被分发到其他不同的region server上;

4:scan

hbase原生提供的方法,顺序扫库;当然可以使用mapreduce并发库的方法;

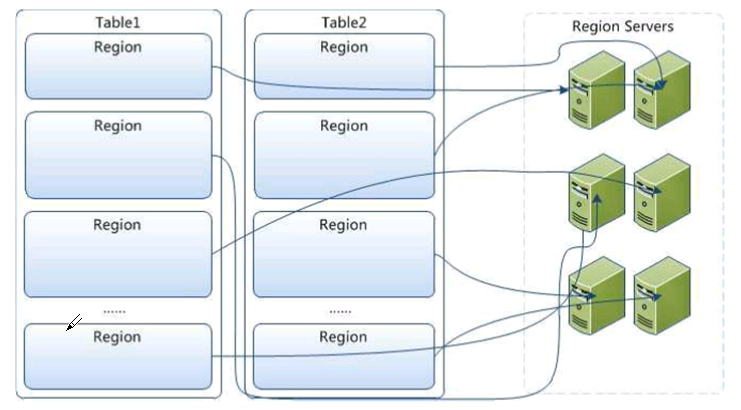
5:Bulk Load

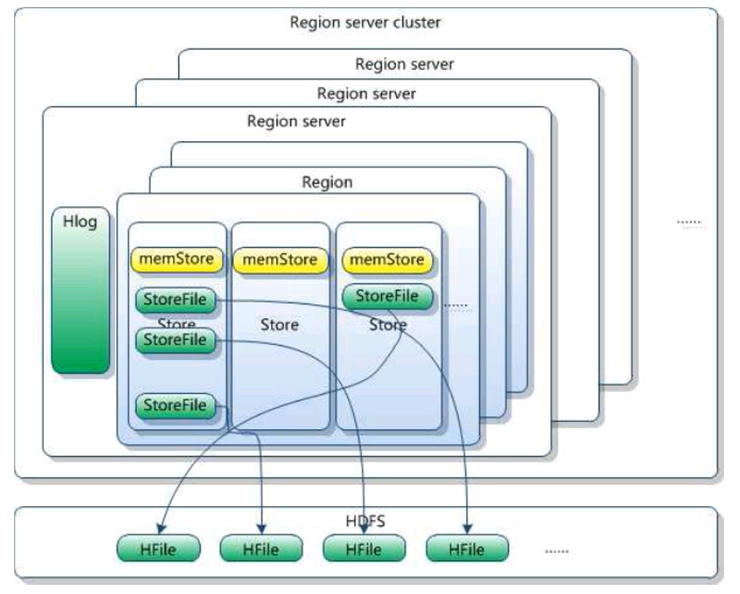
快速导入大批量数据的方法;

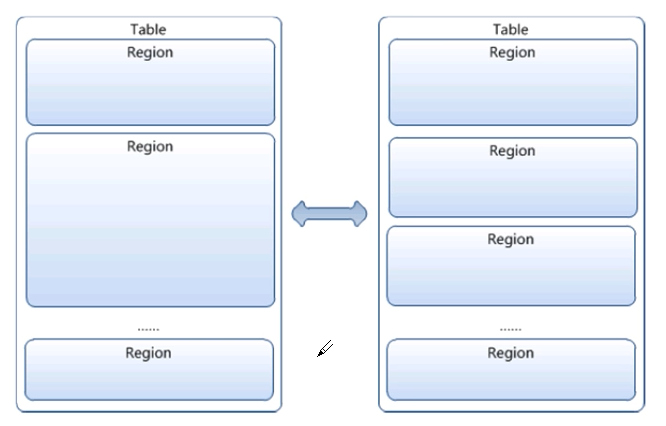
## 存储模型

1:一个table中的一个region会被随机分配给一个region server;

2:region是分布式存储和负载均衡的最小单元;

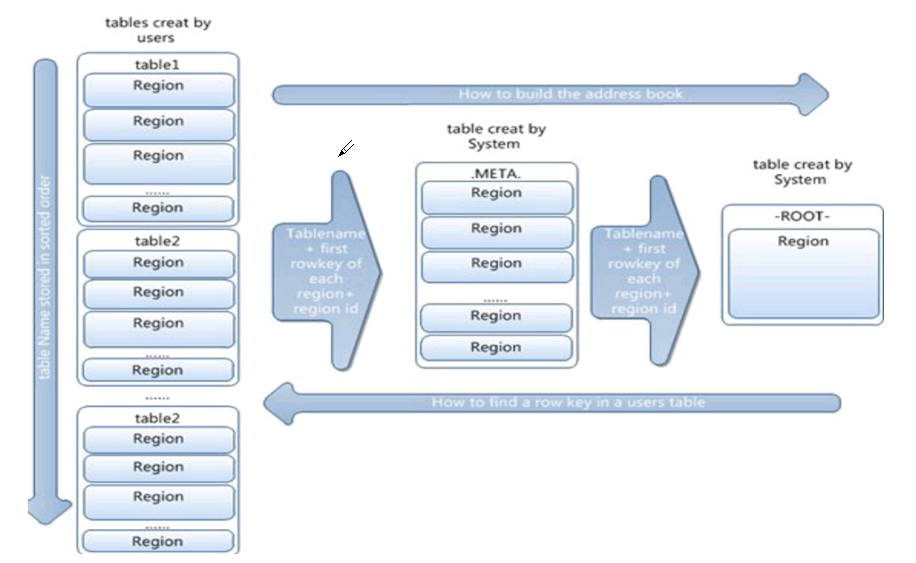






最开始table只有一个region,但是随着数据的put,到达某个阈值的时候,一个大的region会被split成几个小region,被分散到其他region server;

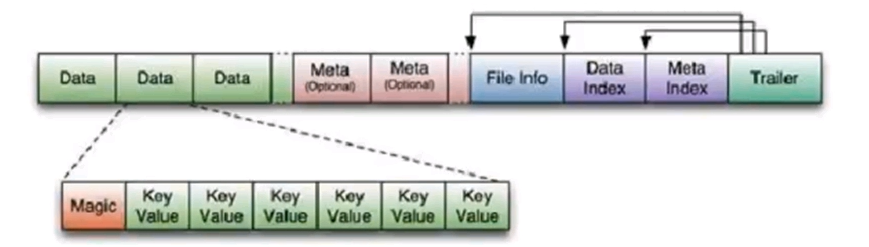
## Region定位



## Hfile结构

1:基于Block的存储结构;

2:Block的索引驻留内存;



## Hbase应用

1:Hbase环境搭建

配置hbase-env.sh

hbase-site.xml

regionservers

运行start-hbase.sh

启动后执行hbase shell进入交互式命令行

2:Hbase api使用举例

Put/Get

scan

......

## Hbase集群部署

1:开源社区版本下载

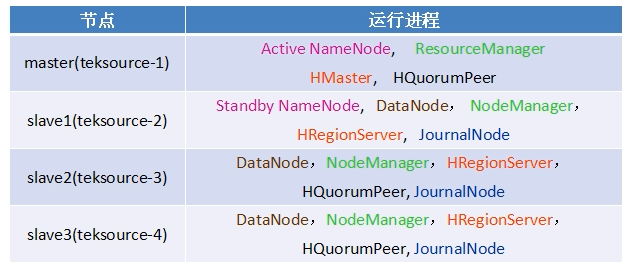


2:Cloudera发行版本下载



如果选择社区版本,注意jar包的替换;

集群规模,一台master,三台slave;



## Hadoop2.0部署

1:配置hosts,确保设计的主机名均可以解析

2:编辑hadoop-env.sh, mapred-env.sh, yarn-env.sh

3:编辑core-site.xml, hdfs-site.xml, yarn-site.xml

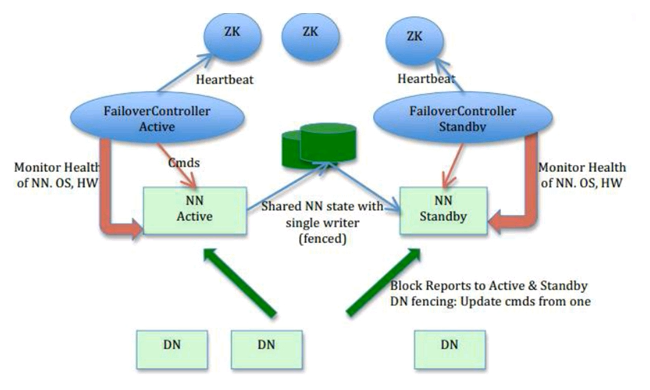
4:编辑slaves文件

5:把hadoop复制到其他节点

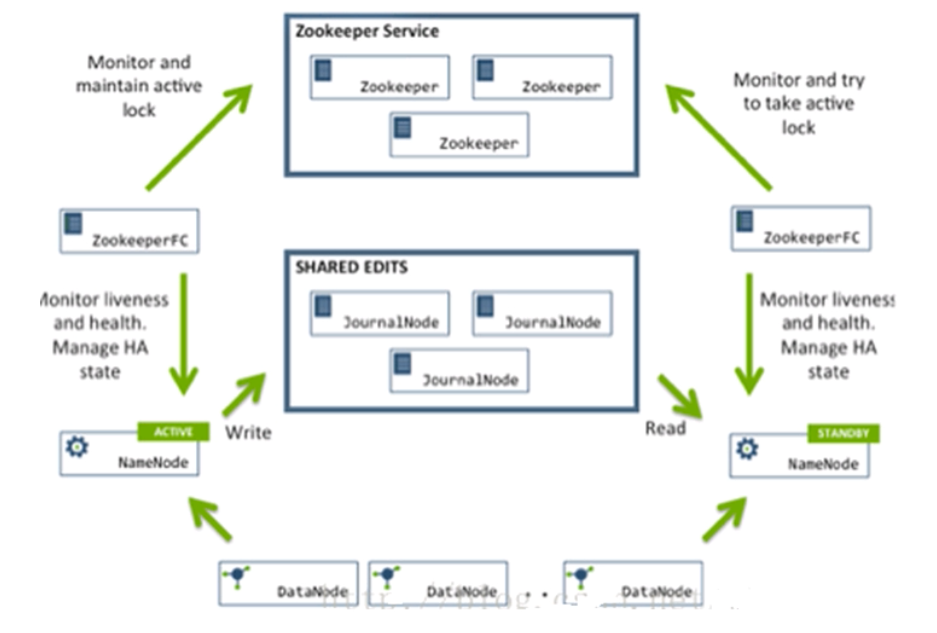
6:启动hadoop

7:验证启动;

## Hadoop2.0HA部署



## QJM方案



## Hadoop HA特点

共享存储

-解决单点故障问题

-利用NFS或QJM来保存Editlog

-保证数据的一致性

FailvoerController

-独立短小的watchdog

-避免NN GC时的心跳暂停

-ZKFC控制

Fencing

-防止闹裂

## Hadoop2.0 HA部署

采用QJM方案

1:启动JN

./sbin/hadoop-daemon.sh start journalnode

./sbin/hadoop-daemons.sh start journalnode

hdfs namenode –initializeSharedEdits

2:启动Active NN

hdfs namenode –format

bin/hadoop-daemon.sh start namenode

3:启动Standby NN

hdfs namenode –boostrapStandby

./sbin/hadoop-daemin.sh start namenode

4:启动Automotic Failover

hdfs zkfc –formatZK

./sbin/hadoop-daemon.sh start zkfc

## Hbase部署

1:配置hosts,确保设计的主机名均可以解析;

2:编辑hbase-env.sh

3:编辑hbase-site.xml(关联hadoop)

4:覆盖hadoop核心jar包

5:编辑regionservers文件

6:把Hbase复制到其他节点

7:启动Hbase

8:验证启动

## Hbase Shell操作

Shell操作

1:建立一个表scores,有两个列簇grad和course;

hbase(main):001:0>create ’scores’,’grade’,’course’

2:查看Hbase中的所有表

hbase(main):001:0>list

3:查看表结构

hbase(main):001:0>describe ’scores’

4:按设计的表结构插入值:

put ’scores’,’Tom’,’grade:’,’5’

put ’scores’,’Tom’,’course:math’,’97’

put ’scores’,’Tom’,’course:art’,’87’

put ’scores’,’Tom’,’grade:’,’4’

put ’scores’,’Tom’,’course:’,’89’

put ’scores’,’Tom’,’course:’,’80’

put命令比较简单,只有一种方法:

hbase>put ’t1’,’r1’, ’c1’,’value’,ts1

t1指表名,r1指行键名,c1指列名,value指单元格值,ts1指时间戳,一般都省略掉了;

5:根据键值查询数据

hbase> get 'scores','Jim'

hbase> get 'scores','Jim','grade'

hbase> get 't1','r1'

hbase> get 't1','r1',{TIMERANGE => [ts1,ts2]}

hbase> get 't1','r1',{COLUMN => 'c1'}

hbase> get 't1','r1',{COLUMN => ['c1','c2','c3']}

hbase> get 't1','r1',{COLUMN => 'c1',TIMESTAMP => ts1}

hbase> get 't1','r1',{COLUMN => 'c1',TIMESTAMP => [ts1,ts2],VERSIONS => 4}

hbase> get 't1','r1',{COLUMN => 'c1',TIMESTAMP => ts1,VERSIONS => 4}

hbase> get 't1','r1','c1'

hbase> get 't1','r1','c1','c2'

hbase> get 't1','r1',['c1','c2']

6:扫描所有数据

scan 'scores'

也可以指定一些修饰词:TIMERANGE,FILTER,LIMIT,STARTROW,STOPROW,TIMESRAMP,MAXLENGTH,or COLUMNS.

没任何修饰词,就是上边例句,就会显示所有数据行;

hbase> scan '.META.'

hbase> scan '.META.',{COLUMNS => 'info:regioninfo'}

hbase> scan 't1',{COLUMNS => ['c1','c2'],LIMIT => 10,STRATROW => 'xyz'}

hbase> scan 't1',{COLUMNS => 'c1',TIMERANGE => [1303668804,1303668904]}

hbase> scan 't1',{FILTER => "(PrefixFilter('row2') AND (QualifierFilter(>=,'binary:xyz'))) AND (TimestampsFilter(123,456))"}

hbase> scan 't1',{FILTER => org.apache.hadoop.hadoop.hbase.filter.ColumnPaginationFilter.new(1,0)}

7:删除指定数据

delete 'scores','Jim','grade'

delete 'scores','Jim'

删除数据命令耶没太多变化,只有一个:

hbase> delete 't1','r1','c1',ts1

另外有一个deleteall命令,可以进行整行的范围的删除操作,慎用!

如果需要进行全表删除操作,就使用truncate命令,其实没有直接的全表删除命令,这个命令也是disable,drop,create三个命令组合出来的.

8:修改表结构

disable 'scores'

alter 'scores',NAME=> 'info'

enable 'scores'

alter命令使用如下(如果无法完成的版本,需要先通用表disable):

a:改变或添加一个列族:

hbase> alter 't1',NAME => 'f1',VERSIONS => 5

b:删除一个列族:

hbase> alter 't1',NAME => 'f1',METHOD => 'delete'

hbase> alter 't1','delete'=> 'f1'

9:统计行数:

hbase> count 't1'

hbase> count 't1',INTERVAL => 100000

hbase> count 't1',CACHE => 1000

hbase> count 't1',INTERVAL => 10,CACHE => 1000

count一般会比较耗时,使用mapreduce进行统计,统计结果会缓存,默认是10行,统计间隔默认的是1000行(INTERVAL)

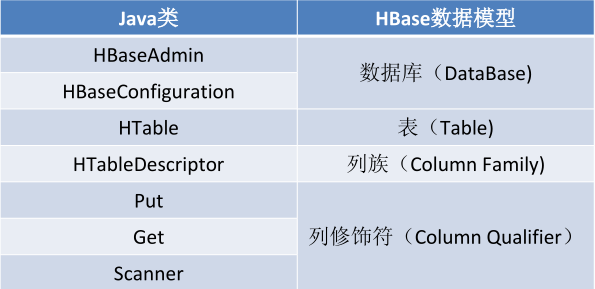
10:表的删除

先禁用表disable ‘table’

删除表 drop ‘table’

## Hbase客户端api操作

java类与数据模型



## HbaseConfiguration

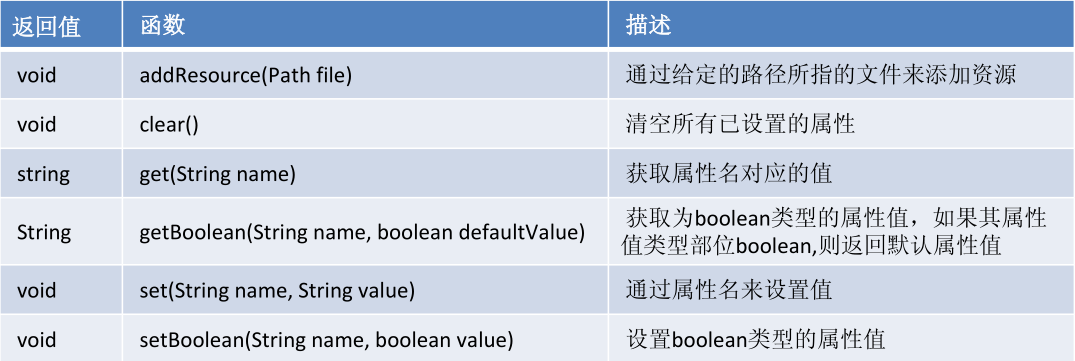
包名:org.apache.hadoop.hbase.HbaseConfiguration

作用:对hbase进行配置

用法示例:

HbaseConfiguration hconfig = new HbaseConfiguration();

hcongig.set("hbase.zookeeper.property.clientPort", "2181");



## HBaseAdmin

包名：org.apache.hadoop.hbase.client.HBaseAdmin

作用：提供了一个接口来管理HBase数据库的表信息。它提供的方法包括：创

建表，删除表，列出表项，使表有效或无效，以及添加或删除表列族成员等。

用法示例：

HBaseAdmin admin = new HBaseAdmin(config);

admin.disableTable("tablename")



## HTableDescriptor

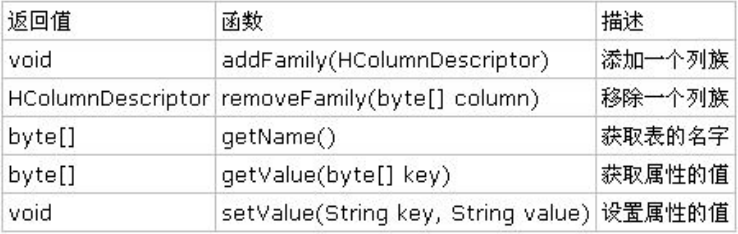
包名：org.apache.hadoop.hbase.HTableDescriptor

作用：包含了表的名字极其对应表的列族

用法示例：

HTableDescriptor htd = new HTableDescriptor(table);

htd.addFamily(new HcolumnDescriptor("family"));



## HColumnDescriptor

包名：org.apache.hadoop.hbase.HColumnDescriptor

作用：维护着关于列族的信息，例如版本号，压缩设置等。它通常在创建表或者

为表添加列族的时候使用。列族被创建后不能直接修改，只能通过删除然后重新

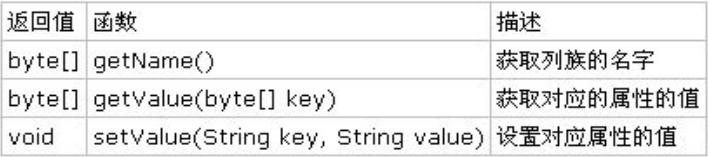
创建的方式。列族被删除的时候，列族里面的数据也会同时被删除。

用法示例：

HTableDescriptor htd = new HTableDescriptor(tablename);

HColumnDescriptor col = new HColumnDescriptor("content:");

htd.addFamily(col);



## HTable

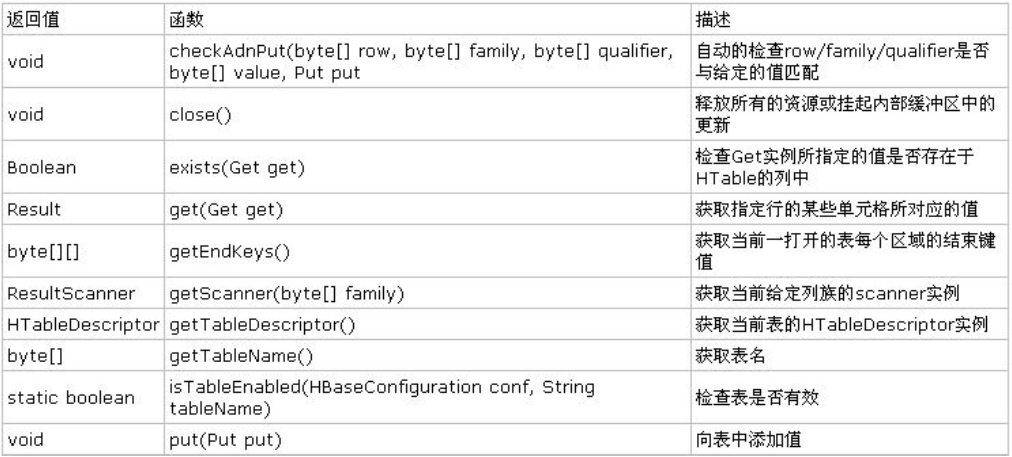
包名：org.apache.hadoop.hbase.client.HTable

作用：可以用来和HBase表直接通信。此方法对于更新操作来说是非线程安全的

用法示例：

HTable table = new HTable(conf, Bytes.toBytes(tablename));

ResultScanner scanner = table.getScanner(family);



## Put

包名：org.apache.hadoop.hbase.client.Put

作用：用来对单个行执行添加操作

用法示例：

HTable table = new HTable(conf,Bytes.toBytes(tablename));

Put p = new Put(brow);//为指定行创建一个Put操作

p.add(family,qualifier,value);

table.put(p);



## Get

包名：org.apache.hadoop.hbase.client.Get

作用：用来获取单个行的相关信息

用法示例：

HTable table = new HTable(conf, Bytes.toBytes(tablename));

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

table.get(g);

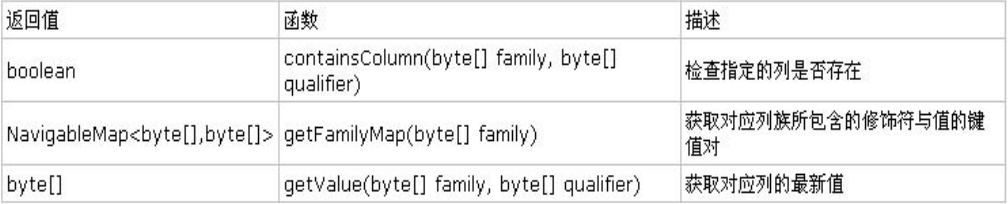


## Result

包名：org.apache.hadoop.hbase.client.Result

作用：存储Get或者Scan操作后获取表的单行值。使用此类提供的方法可以直接

获取值或者各种Map结构（key-value对）



## ResultScanner

包名：org.apache.hadoop.hbase.client.ResultScanner

作用：存储Get或者Scan操作后获取表的单行值。使用此类提供的方法可以直接

获取值或者各种Map结构（key-value对）



## HTablePool

包名：org.apache.hadoop.hbase.client.HTablePool

作用：可以解决HTable存在的线程不安全问题，同时通过维护固定数量的

HTable对象，能够在程序运行期间复用这些HTable资源对象

说明：

1. HTablePool可以自动创建HTable对象，而且对客户端来说使用上是完全透明的，可以避免多线程间数据并发修改问题。

2. HTablePool中的HTable对象之间是公用Configuration连接的，能够可以减少网络开销。

HTablePool的使用很简单：每次进行操作前，通过HTablePool的getTable方法取得一个HTable对象，然后进行put/get/scan/delete等操作，最后通过HTablePool的putTable方法将HTable对象放回到HTablePool中。

## Hbase扫面器

Scanner扫描器

HBase在扫描数据的时候，使用scanner表扫描器。

HTable通过一个Scan实例，调用getScanner(scan)来获取扫描器。可以配置扫描起止位，以及其他的过滤条件。

通过迭代器返回查询结果，使用起来虽然不是很方便，不过并不复杂。但是这里有一点可能被忽略的地方，就是返回的scanner迭代器，每次调用next的获取下一条记录的时候，默认配置下会访问一次RegionServer。这在网络不是很好的情况下，对性能的影响是很大的。

建议配置扫描器缓存。

## 扫描器缓存

hbase.client.scanner.caching配置项可以设置HBase scanner一次从服务端抓取的数据条数，默认情况下一次一条。通过将其设置成一个合理的值，可以减少

scan过程中next()的时间开销，代价是scanner需要通过客户端的内存来维持这些被cache的行记录。

有三个地方可以进行配置：

1）在HBase的conf配置文件中进行配置

2）通过调用HTable.setScannerCaching(int scannerCaching)进行配置

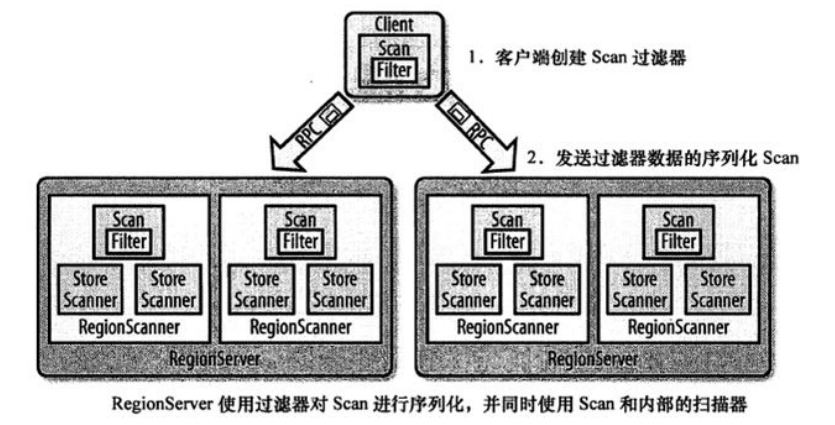
3）通过调用Scan.setCaching(int caching)进行配置。三者的优先级越来越高。

## 过滤器简介

1. 使用过滤器可以提高操作表的效率，HBase中两种数据读取函数get()和scan()都支持过滤器，支持直接访问和通过指定起止行键来访问，但是缺少细粒度的筛选功能，如基于正则表达式对行键或值进行筛选的功能。

2. 可以使用预定义好的过滤器或者是实现自定义过滤器

3. 过滤器在客户端创建，通过RPC传送到服务器端，在服务器端执行过滤操作，把数据返回给客户端



## 过滤器

1. Comparision Filters(比较过滤器)

 RowFilter

 FamilyFilter

 QualifierFilter

 ValueFilter

 DependentColumnFilter

2. Dedicated Filters(专用过滤器)

 SingleColumnValueFilter

 SingleColumnValueExcludeFilter

 PrefixFilter

 PageFilter

 KeyOnlyFilter

 FirstKeyOnlyFilter

 TimestampsFilter

 RandomRowFilter

3、Decorating Filters(附加过滤器)

 SkipFilter

 WhileMatchFilters

## 协处理器

HBase作为列数据库最经常被人诟病的特性包括：

无法轻易建立“二级索引”，难以执行求和、计数、排序等操作。

比如，在旧版本的(<0.92)Hbase中，统计数据表的总行数，需要使用Counter方法，执行一次MapReduce Job才能得到。虽然HBase在数据存储层中集成了MapReduce，能够有效用于数据表的分布式计算。

然而在很多情况下，做一些简单的相加或者聚合计算的时候，如果直接将计算过程放置在server端，能够减少通讯开销，从而获得很好的性能提升。于是，HBase在0.92之后引入了协处理器(coprocessors)，实现一些激动人心的新特性：能够轻易建立二次索引、复杂过滤器(谓词下推)以及访问控制等。

HBase协处理器的灵感来自于Jeff Dean 09年的演讲( P66-67)。它根据该演讲实现了类似于bigtable的协处理器，包括以下特性:

1) 每个表服务器的任意子表都可以运行代码

2) 客户端的高层调用接口(客户端能够直接访问数据表的行地址，多行读写会自动分片成多个并行的RPC调用)

3) 提供一个非常灵活的、可用于建立分布式服务的数据模型

4) 能够自动化扩展、负载均衡、应用请求路由

HBase的协处理器灵感来自bigtable，但是实现细节不尽相同。HBase建立了一个框架，它为用户提供类库和运行时环境，使得他们的代码能够在HBase region server和master上处理。

协处理器分两种类型，系统协处理器可以全局导入region server上的所有数据表，表协处理器即是用户可以指定一张表使用协处理器。

协处理器框架为了更好支持其行为的灵活性，提供了两个不同方面的插件。一个是观察者（observer），类似于关系数据库的触发器。另一个是终端(endpoint)，动态的终端有点像存储过程。

## Observer

观察者的设计意图是允许用户通过插入代码来重载协处理器框架的upcall方法，而具体的事件触发的callback方法由HBase的核心代码来执行。协处理器框架处理所有的callback调用细节，协处理器自身只需要插入添加或者改变的功能。

以HBase0.92版本为例，它提供了三种观察者接口：

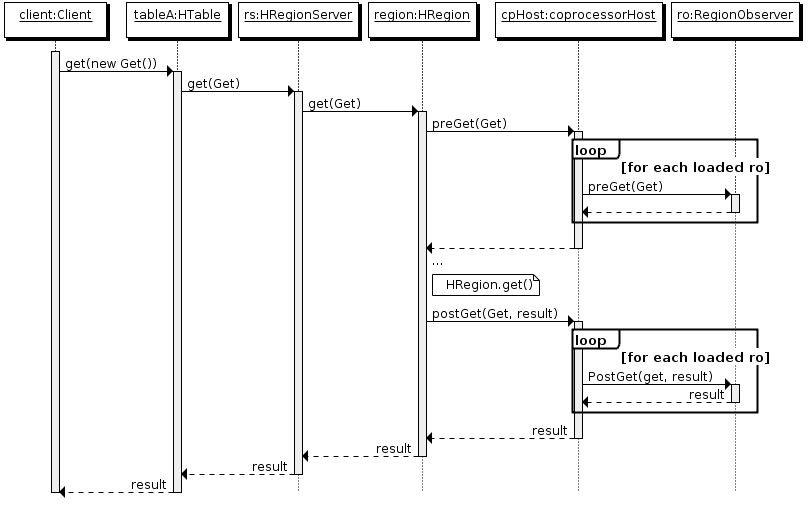
RegionObserver：提供客户端的数据操纵事件钩子：Get、Put、Delete、Scan等。

WALObserver：提供WAL相关操作钩子。

MasterObserver：提供DDL-类型的操作钩子。如创建、删除、修改数据表等。

这些接口可以同时使用在同一个地方，按照不同优先级顺序执行.用户可以任意基于协处理器实现复杂的HBase功能层。HBase有很多种事件可以触发观察者方法，这些事件与方法从HBase0.92版本起，都会集成在HBase API中。不过这些API可能会由于各种原因有所改动，不同版本的接口改动比较大。

## Observer模型



## EndPoint

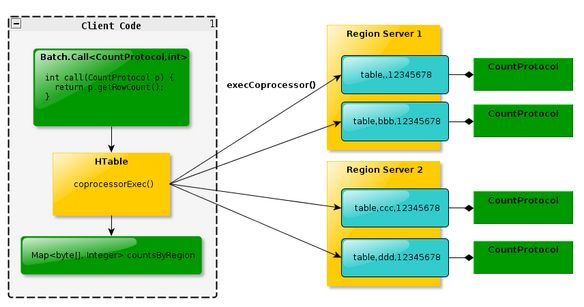
终端是动态RPC插件的接口，它的实现代码被安装在服务器端，从而能够通过HBase RPC唤醒。客户端类库提供了非常方便的方法来调用这些动态接口，它们可以在任意时候调用一个终端，它们的实现代码会被目标region远程执行，结果会返回到终端。用户可以结合使用这些强大的插件接口，为HBase添加全新的特性。终端的使用，如下面流程所示：

定义一个新的protocol接口，必须继承CoprocessorProtocol.

实现终端接口，该实现会被导入region环境执行。

继承抽象类BaseEndpointCoprocessor.

在客户端，终端可以被两个新的HBase Client API调用 。单个region：HTableInterface.coprocessorProxy(Class<T> protocol, byte[] row) 。regions区域：HTableInterface.coprocessorExec(Class<T> protocol, byte[] startKey, byte[] endKey, Batch.Call<T,R> callable)



## EndPoint

有三个方法对Endpoint进行设置：

A. 启动全局aggregation，能过操纵所有的表上的数据。通过修改hbase-site.xml这个文件来实现，只需要添加如下代码：

<property>

<name>hbase.coprocessor.user.region.classes</name>

<value>org.apache.hadoop.hbase.coprocessor.RowCountEndpoint</value>

</property>

B. 启用表aggregation，只对特定的表生效。通过HBase Shell 来实现。

(1)disable指定表。hbase> disable 'mytable'

(2)添加aggregation hbase> alter 'mytable', METHOD => 'table\_att','coprocessor'=>'|org.apache.hadoop.hbase.coprocessor.RowCountEndpoint ||'

(3)重启指定表 hbase> enable 'mytable'

C. API调用

HTableDescriptor htd=new HTableDescriptor("testTable");

htd.setValue("CORPROCESSOR$1",path.toString+"|"+ RowCountEndpoint.class.getCanonicalName()+"|"+Coprocessor.Priority.USER);

## 几点说明

1. 协处理器配置的加载顺序：先加载配置文件中定义的协处理器，后加载表描述符中的协处理器

2. COPROCESSOR$<number>中的number定义了加载的顺序

3. 协处理器配置格式

