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。  
到 0.96 版本又新增一个 RegionServerObserver

Endpoint 协处理器类似传统数据库中的存储过程，客户端可以调用这些 Endpoint 协处 理器执行一段 Server 端代码，并将 Server 端代码的结果返回给客户端进一步处理，最常 见的用法就是进行聚集操作。如果没有协处理器，当用户需要找出一张表中的最大数据，即max 聚合操作，就必须进行全表扫描，在客户端代码内遍历扫描结果，并执行求最大值的 操作。这样的方法无法利用底层集群的并发能力，而将所有计算都集中到 Client 端统一执 行，势必效率低下。利用 Coprocessor，用户可以将求最大值的代码部署到 HBase Server 端，HBase 将利用底层 cluster 的多个节点并发执行求最大值的操作。即在每个 Region 范围内 执行求最大值的代码，将每个 Region 的最大值在 Region Server 端计算出，仅仅将该 max 值返回给客户端。在客户端进一步将多个 Region 的最大值进一步处理而找到其中的最大值。这样整体的执行效率就会提高很多

**Observer 允许集群在正常的客户端操作过程中可以有不同的行为表现  
Endpoint 允许扩展集群的能力，对客户端应用开放新的运算命令**

**observer 类似于 RDBMS 中的触发器，主要在服务端工作  
endpoint 类似于 RDBMS 中的存储过程，主要在 client 端工作**

**observer 可以实现权限管理、优先级设置、监控、 ddl 控制、 二级索引等功能  
endpoint 可以实现 min、 max、 avg、 sum、 distinct、 group by 等功能**

  协处理器的加载方式有两种，我们称之为静态加载方式（ Static Load） 和动态加载方式 （ Dynamic Load）。 静态加载的协处理器称之为 System Coprocessor，动态加载的协处理器称 之为 Table Coprocessor

 1、静态加载

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

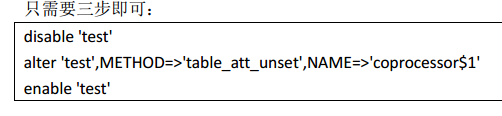
|  |  |
| --- | --- |
| 1  2  3  4 | <property>  <name>hbase.coprocessor.user.region.classes</name>  <value>org.apache.hadoop.hbase.coprocessor.AggregateImplementation</value>  </property> |

　　为所有 table 加载了一个 cp class，可以用” ,”分割加载多个 class

    2、动态加载

启用表 aggregation，只对特定的表生效。通过 HBase Shell 来实现。  
disable 指定表。 hbase> disable 'mytable'  
添加 aggregation  
hbase> alter 'mytable', METHOD => 'table\_att','coprocessor'=>  
'|org.apache.Hadoop.hbase.coprocessor.AggregateImplementation||'  
重启指定表 hbase> enable 'mytable'

 3、协处理器卸载



 row key 在 HBase 中是以 B+ tree 结构化有序存储的，所以 scan 起来会比较效率。单表以 row key 存储索引， column value 存储 id 值或其他数据 ，这就是 Hbase 索引表的结构。

   由于 HBase 本身没有二级索引（ Secondary Index）机制，基于索引检索数据只能单纯地依靠 RowKey，为了能支持多条件查询，开发者需要将所有可能作为查询条件的字段一一拼接到 RowKey 中，这是 HBase 开发中极为常见的做法

**二级索引**

防止检测jar包，报错，自动断开regionserver服务，需要关闭hbase.table.sanity.checks

****设置******hbase.table.****sanity.checks=false**

**hbase.coprocessor.abortonerror=false**

**Observer**

**步骤1：通过如下方法创建表**

hbase(main):001:0> create 'coprocessor\_table','F'

**步骤2：通过alter命令将协处理器加载到表中**

**Disable ‘testOb’**

alter 'testOb' , METHOD =>'table\_att','coprocessor'=>'hdfs://tianxi-ha/test/pagerank/original-test-1.0-SNAPSHOT.jar|hbase.coprocessor.ObserverDemo|1001'

Enable ‘testOb’

其中：'coprocessor'=>'jar文件在hdfs上的绝对路径|协处理器主类|优先级|协处理器参数。上述协处理器并没有参数，所以未给出参数，对于协处理器的优先级不在此做讨论。

alter 't1', METHOD => 'table\_att\_unset', NAME => 'coprocessor$1'

**步骤3：检查协处理器的加载**

Desc 'testOb'

**步骤4:输入数据**

put ''testOb'','row1','F:A',123

delete ''testOb'','row1','F:B'

Endpoint实现:

0.96之前是CoprocessorProtocol,之后是protobuf rpc service

1. 定义.proto文件，用于定义类的一些基本信息

option java\_package = "hbase.coprocessor.endpoint.test";

option java\_outer\_classname = "EndpointTestProtos";

option java\_generic\_services = true;

option java\_generate\_equals\_and\_hash = true;

option optimize\_for = SPEED;

message CountRequest {

}

message CountResponse {

required int64 count = 1 [default = 0];

}

service RowCountService {

rpc getRowCount(CountRequest)

returns (CountResponse);

}

1. 生成java代码 2.x的protobuf版本

$ protoc --java\_out=./java/ ./proto/testend.proto  
protoc 的命令格式为 protoc [OPTION] PROTO\_FILES （最后是待编译的 proto文件）  
--java\_out 为输出java代码的目录，这里指定的是 ./java/ 目录。  
随后我们指定了proto文件的位置 ./proto/helloworld.proto 。  
执行上述命令，我们就 ./java/ 目录下就产生了对应的 java文件。

安装过程：

curl: (35) SSL connect error 执行 yum update nss

解压：tar xvzf protobuf-3.1.0.tar.gz  
安装步骤：  
（0）./autogen.sh   
（1）./configure  --prefix=/usr/local/protobuf  
（2）make   
（3）make check   
（4）make install

1. 定义自己的Endpoint类

Class RowCountEndpoint extends EndpointTestProtos.RowCountService implements Coprocessor,CoprocessorService{

}

1. 部署jar到hdfs

disable 't7'

alter 't7','coprocessor'=**>**'hdfs://tianxi-ha/test/jar/test-1.0-SNAPSHOT.jar|hbase.coprocessor.endpoint.RowCountEndpoint|1001'

enable 't7'

alter 't7' , METHOD =>'table\_att\_unset','coprocessor'=>'coprocessor$1'

**增加thriftServer线程数**

hbase-daemon.sh start thrift --threadpool -m 200 -w 500