1. HBase高级编程

在使用 HBase 时，如果你的数据量达到了数十亿行或数百万列，此时能否在查询中返回大量数据将受制于网络的带宽，即便网络状况允许，但是客户端的计算处理也未必能够满足要求。在这种情况下，协处理器（Coprocessors）应运而生。它允许你将业务计算代码放入在 RegionServer 的协处理器中，将处理好的数据再返回给客户端，这可以极大地降低需要传输的数据量，从而获得性能上的提升。同时协处理器也允许用户扩展实现 HBase 目前所不具备的功能，如权限校验、二级索引、完整性约束等。

1.1. 协处理器—Coprocessor

1、 起源

HBase作为列族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。

2、协处理器有两种：observer 和 endpoint

1.2. Observer

Observer 协处理器类似于关系型数据库中的触发器，当发生某些事件的时候这类协处理器会被 Server 端调用。通常可以用来实现下面功能：

```
1、权限校验：在执行 Get 或 Put 操作之前，您可以使用 preGet 或 prePut 方法检查权限；
2、完整性约束： HBase 不支持关系型数据库中的外键功能，可以通过触发器在插入或者删除数据的时候，对关联的数据进行检查；
3、二级索引： 可以使用协处理器来维护二级索引。
```

类似于传统数据库中的**触发器**，当发生某些事件的时候这类协处理器会被 Server 端调用。Observer Coprocessor 就是一些散布在 HBase Server端代码中的 hook 钩子，在固定的事件发生时被调用。比如：put 操作之前有钩子函数 prePut，该函数在put操作执行前会被 RegionServer 调用；在 put 操作之后则有 postPut 钩子函数

以HBase0.92版本为例，它提供了三种观察者接口：

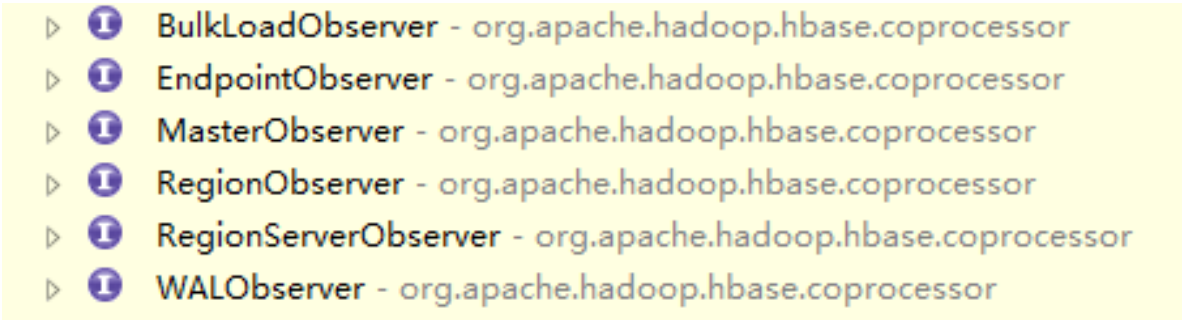
● RegionObserver：提供客户端的数据操纵事件钩子：Get、Put、Delete、Scan等。

● WALObserver：提供WAL相关操作钩子，允许您观察与预写日志（WAL）相关的事件。

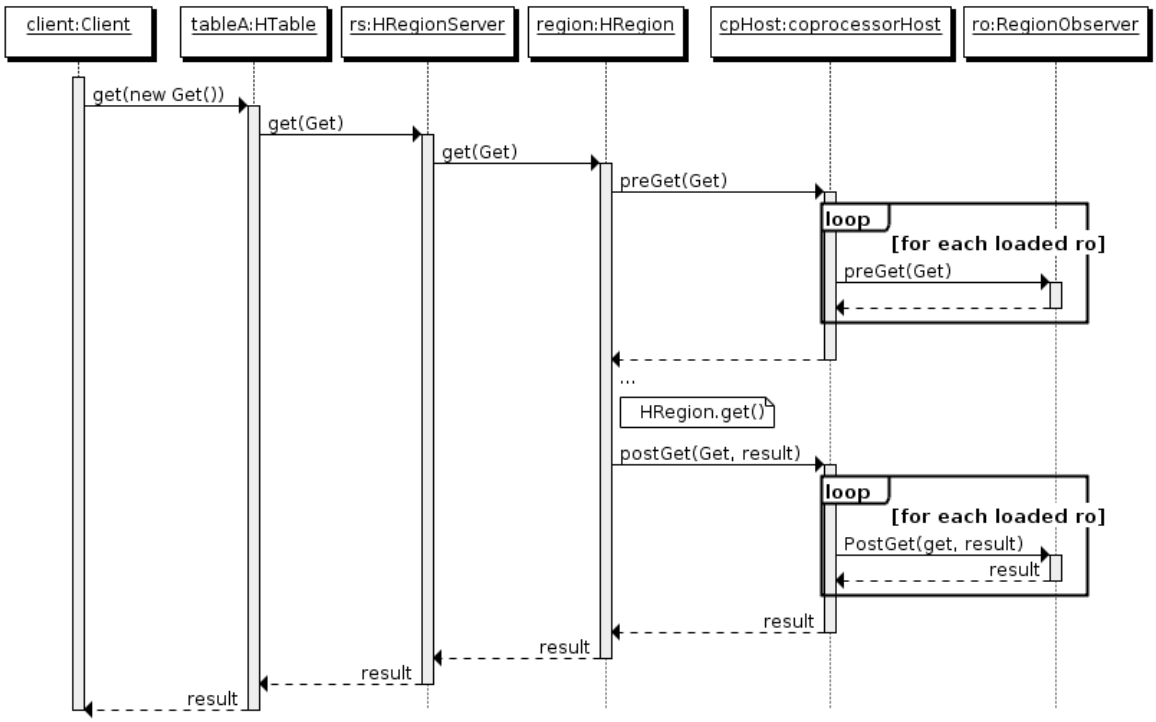
● MasterObserver：提供DDL-类型的操作钩子。如创建、删除、修改数据表等。

到0.96版本又新增一个RegionServerObserver，允许您观察与 RegionServer 操作相关的事件，例如启动，停止或执行合并，提交或回滚。

其实到现在的hbase-1.2.12版本中：Observer的种类已经很多了：



下图是以RegionObserver为例子讲解Observer这种协处理器的原理：



详细步骤：

```
1、客户端发出put请求
2、该请求被分派给合适的RegionServer和region
3、coprocessorHost拦截该请求，然后在该表上登记的每个RegionObserver上调用prePut()
4、如果没有被prePut()拦截，该请求继续送到region，然后进行处理
5、region产生的结果再次被CoprocessorHost拦截，调用postPut()
6、假如没有postPut()拦截该响应，最终结果被返回给客户端
```

如果大家了解 Spring，可以将这种执行方式类比于其 AOP 的执行原理即可，官方文档当中也是这样类比的：

```
If you are familiar with Aspect Oriented Programming (AOP), you can think of a coprocessor as applying advice by intercepting a request and then running some custom code,before passing the request on to its final destination (or even changing the destination).
```

如果您熟悉面向切面编程（AOP），您可以将协处理器视为通过拦截请求然后运行一些自定义代 码来使用 Advice，然后将请求传递到其最终目标（或者更改目标）。

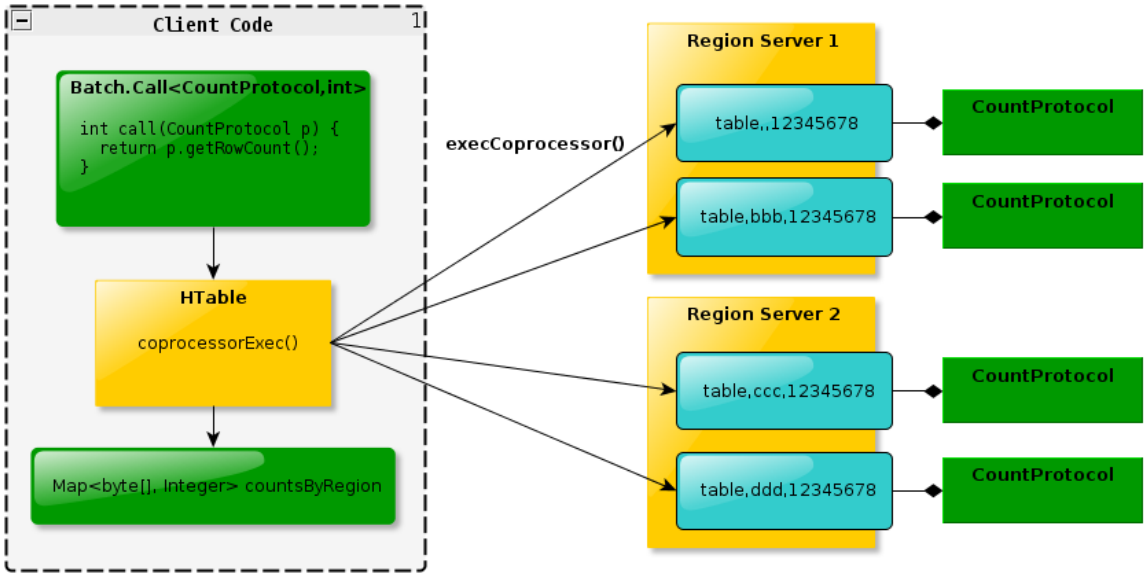
1.3. Endpoint

Endpoint 协处理器类似于关系型数据库中的存储过程。客户端可以调用 Endpoint 协处理器在服务端对数据进行处理，然后再返回。

以聚集操作为例，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，然后在客户端上遍历扫描结果，这必然会加重了客户端处理数据的压力。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 端计算出来，仅仅将该 max 值返回给客户端。之后客户端只需要将每个 Region 的最大值进行比较而找到其中最大的值即可。

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。

下图是EndPoint的工作原理：



总结

```
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，主要在服务端工作

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
```

1.4. 协处理加载方式

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**

1.4.1. 静态加载

第一步：通过修改 hbase-site.xml 这个文件来实现，启动全局 aggregation，能够操纵所有的表上的数据。只需要添加如下代码：

```
<property>
    <name>hbase.coprocessor.region.classes</name>
    <value>com.nx.hbase.coprocessor.endpoint.SumEndPoint</value>
</property>
```

标签的值必须是下面其中之一：

```
RegionObservers 和 Endpoints 协处理器： hbase.coprocessor.region.classes
WALObservers 协处理器： hbase.coprocessor.wal.classes
MasterObservers 协处理器： hbase.coprocessor.master.classes
```

必须是协处理器实现类的全限定类名。如果为加载指定了多个类，则类名必须以逗号分隔。

第二步：将 jar(包含代码和所有依赖项) 放入 HBase 安装目录中的 lib 目录下；

第三步：重启 HBase

1.4.2. 静态卸载

1、从 hbase-site.xml 中删除配置的协处理器的元素及其子元素；

2、从类路径或 HBase 的 lib 目录中删除协处理器的 JAR 文件（可选）；

3、重启 HBase。

1.4.3. 动态加载

使用动态加载协处理器，不需要重新启动 HBase。但动态加载的协处理器是基于每个表加载的，只能用于所指定的表。 此外，在使用动态加载必须使表脱机（disable）以加载协处理器。动态加载通常有两种方式：Shell 和 Java API 。

以下示例基于两个前提：

```
1、coprocessor.jar 包含协处理器实现及其所有依赖项。
2、 JAR 包存放在 HDFS 上的路径为：hdfs://<namenode>:<port>/user/<hadoop-
user>/coprocessor.jar
```

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'=>'hdfs://<namenode>:<port>/user/<hadoop-
user>/coprocessor.jar|org.apache.Hadoop.hbase.coprocessor.AggregateImplementatio
n|1085541823|arg1=1,arg2=2'
```

Coprocessor 包含由管道（|）字符分隔的四个参数，按顺序解释如下：

```
1、JAR 包路径：通常为 JAR 包在 HDFS 上的路径。关于路径以下两点需要注意：
    允许使用通配符，例如： hdfs://<namenode>:<port>/user/<hadoop-user>/*.jar 来添加指
定的 JAR 包；
    可以使指定目录，例如： hdfs://<namenode>:<port>/user/<hadoop-user>/ ，这会添加目录
中的所有 JAR 包，但不会搜索子目录中的 JAR 包。

2、类名：协处理器的完整类名。

3、优先级：协处理器的优先级，遵循数字的自然序，即值越小优先级越高。可以为空，在这种情况下，将分
配默认优先级值。

4、可选参数 ：传递的协处理器的可选参数。
```

◆ 重启指定表

```
hbase> enable 'mytable'
```

◆ 验证协处理器是否已加载

```
hbase > describe 'mytable'
```

协处理器出现在 TABLE_ATTRIBUTES 属性中则代表加载成功。

1.4.4. 协处理器卸载

只需要三步即可：

1、禁用表

```
disable 'mytable'
```

2、移除表协处理器

```
alter 'mytable', METHOD=>'table_att_unset', NAME=>'coprocessor$1'
```

3、启用表

```
enable 'mytable'
```

2. 二级索引（ObserverCoprocessor案例）

RowKey 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 RowKey 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

在社交类应用中，经常需要快速检索各用户的关注列表guanzhu，同时，又需要反向检索各种户的粉丝列表fensi，为了实现这个需求，最佳实践是建立两张互为反向的表：

```
一个表为正向索引关注表："guanzhu"：
Rowkey:   a
f1:from   b
```

```
另一个表为反向索引粉丝表："fensi"：
Rowkey:   b
f1:from  a
```

建表语句：

```
create 'guanzhu','cf1'
create 'fensi','cf1'
```

实现效果：

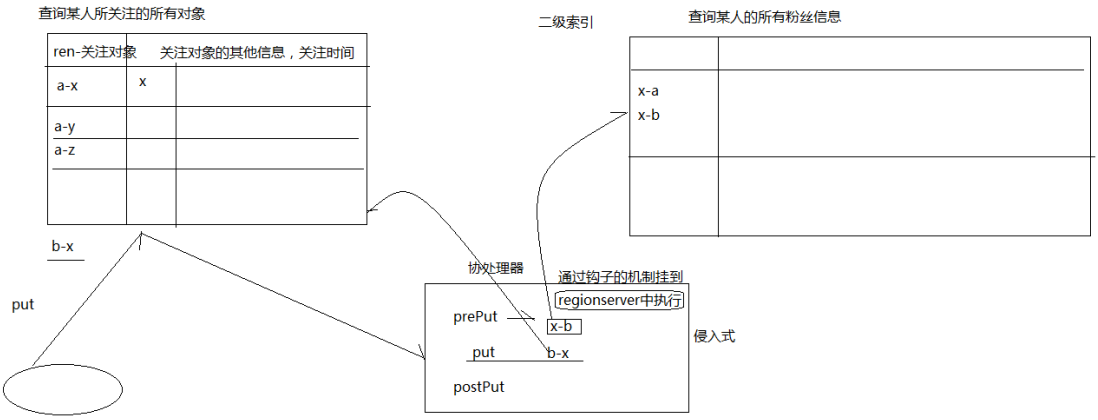
往 guanzhu 表插入一条数据

```
put 'guanzhu','a','cf1:from','b'
```

就会自动往 fensi 表插入一条数据，自动被触发了执行类似下面的这条语句

```
put 'fensi','b','cf1:from','a'
```

插入一条关注信息时，为了减轻应用端维护反向索引表的负担，可用Observer协处理器实现：



1、具体代码实现

```
package com.mazh.hbase.core.nx;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.Cell;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Durability;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.coprocessor.BaseRegionObserver;
import org.apache.hadoop.hbase.coprocessor.ObserverContext;
import org.apache.hadoop.hbase.coprocessor.RegionCoprocessorEnvironment;
import org.apache.hadoop.hbase.regionserver.wal.WALEdit;

import java.io.IOException;

public class FansGaunzhuCoprocessor extends BaseRegionObserver {

    static Configuration config = HBaseConfiguration.create();
    static HTable table = null;

    static {
```

```
        config.set("hbase.zookeeper.quorum",
"bigdata02:2181,bigdata03:2181,bigdata04:2181");
        try {
            table = new HTable(config, "guanzhu");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void prePut(ObserverContext<RegionCoprocessorEnvironment> e, Put put,
WALEdit edit,
                       Durability durability) throws IOException {
        //      super.prePut(e, put, edit, durability);

        byte[] row = put.getRow();
        Cell cell = put.get("f1".getBytes(), "from".getBytes()).get(0);

        Put putIndex = new Put(cell.getValueArray(), cell.getValueOffset(),
cell.getValueLength());

        putIndex.addColumn("f1".getBytes(), "from".getBytes(), row);

        table.put(putIndex);
        table.close();
    }
}
```

2、打成jar包（cppp.jar），上传到hdfs中的hbasecp目录下

```
[hadoop@hadoop02 soft]# hadoop fs -mkdir -p /hbasecp
[hadoop@hadoop02 soft]# hadoop fs -put cppp.jar /hbasecp
```

3、建hbase表，请按以下顺序操作

```
hbase(main):036:0> create 'guanzhu','f1'
hbase(main):036:0> create 'fensi','f1'
hbase(main):036:0> disable 'fensi'
hbase(main):036:0> alter 'fensi',METHOD => 'table_att','coprocessor' =>
'hdfs://hadoop277ha/hbasecp/cppp.jar|com.mazh.hbase.core.coprocessor.FansGaunzhu
Coprocessor|1001|'
hbase(main):036:0> enable 'fensi'
```

参数理解：

```
# 理解coprocessor的四个参数，分别用'|'隔开的
1、你的协处理器jar包所在hdfs上的路径
2、协处理器类全限定名
3、协处理器加载顺序
4、传参
```

4、插入数据验证，命令行和代码都可以

```
testPut("fensi", "a", "f1", "from", "b")
```

或者也可以使用Java代码进行测试。

5、查验结果