# hadoop.ipc.RPC 简介

hadoop.ipc.RPC 实现了一种远程过程调用框架，应用可以直接定义过程调用的接口和server端实现，就可以直接通过RPC框架获得RPC server 和client端的接口代理。

Hadoop.ipc.RPC作为Hadoop的底层核心组件，在hadoop HDFS，MapReduce以及HBase中都有广泛的使用。

HDFS中NameNode，DataNode等都是通过实现对应协议的接口，然后利用hadoop.ipc.RPC获取服务器实体的。

HBase中的HBaseRPC采用的也是与hadoop.ipc.RPC类似的实现，其中的Region Server, Master Server 都是通过实现对应的协议接口直接获取服务器实体的。

这种ipc抽象的好处是

1. 服务端只需要实现本地接口，序列化以及网络操作的处理得到极大的复用。
2. 客户端只需要调用与服务端相同接口的代理，这样就避免了每个协议都实现一个客户端接口和服务器了，并且服务器和客户端都具有同样的抽象层（相同的API），具有很好的一致性。

# 应用实例: 远程执行shell命令

# 实现

## hadoop.ipc

前面已经提到了hadoop RPC的实现利用了hadoop.ipc，所以在讲解hadoop.RPC之前不得不先分析一下hadoop.ipc相关的实现。

首先hadoop.ipc 主要实现了server和client的接口。hadoop.ipc.server 和hadoop.ipc.client 为上层应用实现了一种基于请求应答消息处理模式，使具体应用协议与TCP网络层的处理隔绝开来，用户无须关注底层网络的实现，而仅仅实现请求和应答就可以完成指定协议的server和client。

### hadoop.ipc协议

### hadoop.ipc 特点

1. request response模式
2. 同一个端口可以处理多种不同的协议(有接口类名确定)
3. 每个连接只处理同一种协议
4. client端单向ping，server不对client的ping做出响应

### 交互过程

1. 连接建立client 向server发送RPCheader，header指定了协议名，用户组信息，权限
2. 请求响应client 向server发送请求(Params)，并接受响应(value)。

ipc协议

header

data

ping

### hadoop IPC client实现

主要接口:

Writable call(Writable param, InetSocketAddress addr,

Class<?> protocol, UserGroupInformation ticket,

int rpcTimeout, Configuration conf) throws InterruptedException, IOException

向指定地址发送请求param，并等待服务响应，将响应返回。其中请求和响应都需要实现Writable接口。还有许多其他的call接口，这里只讲解这个接口。

client.call()的每次调用都会生成一个Call的实体，用于在client上下文中传递请求和响应。

缓存

client发起的连接Connection都有一个连接标识ConnectionID与之关联，ConnectionID中包含了远程服务器地址，连接（Socket选项，超时时间等）参数，用户组信息，采用的协议。ConnectionID用以上信息生成的哈希值，作为Connection缓存的Key，以减少重复创建连接的次数。

消息发送与接收

采用同步阻塞方式发送，独立单线程阻塞收取的模式，以保证数据不会发生交错，并且是有序的。

Connection.sendParam处理消息的发送，直接将Call.id和Call.param进行序列化，然后发送给服务器

Connection.receiveResponse解析id，获取id对应的Call然后解析并设置Call的返回值，通知Call的等待线程。

### hadoop IPC server实现

hadoop IPC server的实现分为以下几层次

Listener 监听客户端连接，然后交给Reader 处理

Reader非阻塞收取网络请求数据

Connection 解析数据:

1. RPC header: 确定连接采用的RPC协议，权限和用户组，
2. data: 每一次过程调用的参数

Handler处理远程过程调用的方法，生成响应

Responder 非阻塞发送响应数据

Handler处理阻塞(也可能非阻塞)的内部调用call的实现。

其中Listener，Reader，Responser处理非阻塞IO的操作，每个实体中都有一个Selector，用于处理IO事件的通知，这里采用的是流水线的结构Listener->Reader->Handler->Responser。

关于网络IO方面处理

将系统层通知机制(select,poll,epoll等)的poll操作(epoll\_wait, select等)也作为一种与计时器事件IO事件等对等的reactor事件，reactor事件可能会阻塞(正如调用epoll\_wait或select会阻塞一样)，当reactor调用返回时，会产生一个局部的就绪事件列表，reactor事件将自身加入到列表的末尾，

Server的实现没有采用多线程并行处理来降低线程切换和同步开销，避免流水线划分不一致的原因是Handler的操作可能导致阻塞，必须要有独立的线程或线程组处理Hander，线程之间的切换必不可少。Listener独立为单独的线程大概是为了Reader之间负载的均衡，新加入的连接按照round robin在Reader之间进行负载均衡（实际上可能并不均衡，每条连接处理的请求以及持续的时间是不确定的），但是产生了同步开销，。Reader和Responser独立的目的暂时还不清楚，

### RPC 调用和返回结果的封装

Invocation 封装调用

ObjectWritable 用于封装返回结果

### RPC 实体获取

# 总结

本文的作者是一个以C或C++为主要语言的程序员，在阅读了hadoop.ipc.RPC的实现后，感觉java反射机制比现有的C++机制更具有优雅性和灵活性。在这种大规模环境，进程之间的交互是非常复杂的，如果没有一种跨进程的统一接口和类的抽象，代码的逻辑复杂性必定会大大增加。如果也许这就是hadoop采用Java而不是C或C++的原因吧。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Leader | Follower |  |
| Nginx | 获取了accept锁的进程 | 其他进程 | LeaderFollower + Half-Sync/Half-Async |
| Cherokee | 正在对listener进行poll的线程 | 其他线程 | LeaderFollower |
| boost.asio | 正在运行reactor的线程 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |