Yarn4 RPC框架详解

RPC框架背景

> 网络通信模块是分布式系统中最底层的模块，支撑了上层分布式环境下复杂的进程间通信（Inter-Process Communication， IPC）逻辑，是所有分布式系统的基础。远程过程调用(Remote Produce Call, RPC)是一种常用的分布式网络通信协议。

RPC通信模型介绍

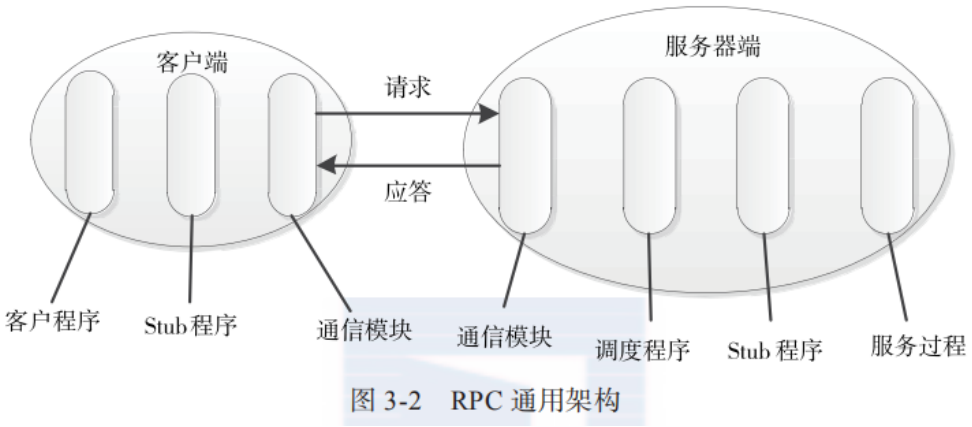


图 3-2　RPC 通用架构

- 通信模块

  两个相互协作的通信模块实现请求-应答协议。负责传递请求和应答，通常为异步方式

- Stub程序

  代理程序，客户端和服务段均存在。
  在客户端，负责将信息通过网络模块发送给服务器端
  在服务器端，依次进行解码请求消息中的参数，调用相应的服务过程和编码结果的返回值等处理

- 调度程序

  接受来自通信模块的请求信息，并根据其中的标识选择一个`stub`程序进行处理

- 客户程序/服务过程

  请求的发出者和请求的处理者。

RPC请求从发出到获取处理结果

1. 客户程序以本地方式调用系统产生的Stub程序

2. 改Stub程序将函数调用信息按照网络通信模块的要求封装成信息包，并交给通信模块发送到远程服务端
3. 远程服务端接受此消息后，将此消息发送给对应的Stub程序
4. Stub程序拆封消息，形成被调过程要求的形式，并调用对应函数
5. 被调用函数按照所获参数执行，并将结果返回给Stub程序
6. Stub程序将此结果封装成消息，通过网络通信模型逐级传给客户程序

Hadoop RPC 特点

- **透明性**：两台计算机程序互相调用时，感觉像是执行一个本地调用
- **高性能**：各个系统均采用了Master/Slave结构，RPC Server是一个高性能服务器，能够高效地处理来自多个Client地并发RPC请求。
- **可控性**：JDK中有一个RPC框架——RMI，使用Hadoop RPC可以更好地进行参数优化。

Hadoop RPC 总体架构

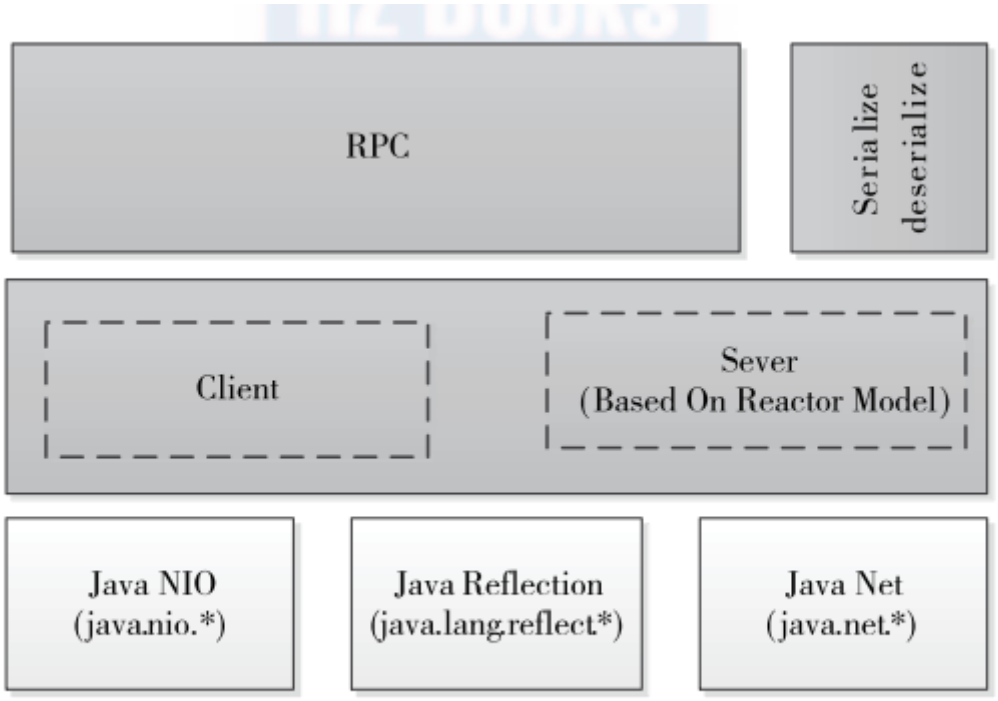


图 3-3　Hadoop RPC 总体架构

- 序列化层

  序列化主要作用是将结构化对象转为字节流以便于通过网络进行传输或写入持久存储。Protocol buffer和Apache Avro均可用在序列化层，Hadoop还有一个本身地Writeable序列化框架

- 函数调用层

  定位要调用地函数并执行该函数。Hadoop RPC采用了Java反射机制与动态代理实现了函数调用

- 网络传输层

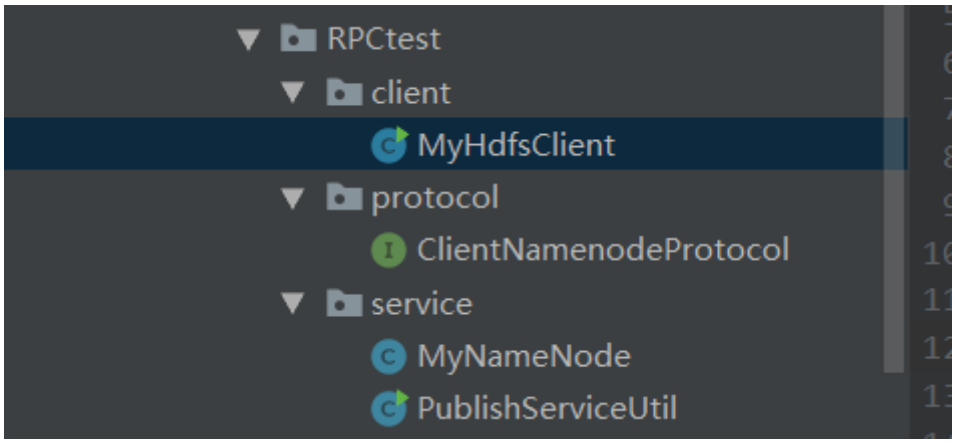
  描述了Client与Server之间消息传输地方式，Hadoop RPC采用了基于 TCP/IP的Socket机制

- 服务器端处理框架

> 可被抽象为网络I/O模型，描述客户端与服务端间信息交互方式，常见有网络I/O模型有阻塞式I/O，非阻塞式I/O，事件驱动I/O

Hadoop RPC框架简单实例

目录



实现RPC协议

```
#ClientNamenodeProtocol.java
package cn.neu.connection.RPCtest.protocol;


public interface ClientNamenodeProtocol {
    //协议版本
    public static final long versionID =1L;
    public String getMetaData (String path);
}
```

实现业务实现类

```
# ClientNamenodeProtocol.java
package cn.neu.connection.RPCtest.service;

import cn.neu.connection.RPCtest.protocol.ClientNamenodeProtocol;

public class MyNameNode implements ClientNamenodeProtocol {
    // 模拟namenode的业务方法之一：查询元数据
    @Override
    public String getMetaData(String path) {
        return path+": 3 - {BLK_1,BLK_2 } ....";
    }
}
```

服务端代码

使用下面的代码可以把业务类发布为一个服务

```
#PublishServiceUtil.java
package cn.neu.connection.RPCtest.service;


import cn.neu.connection.RPCtest.protocol.ClientNamenodeProtocol;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.ipc.RPC.Builder;
import org.apache.hadoop.ipc.Server;

public class PublishServiceUtil {

    public static void main(String[] args) throws Exception {

        Builder builder = new Builder(new Configuration());
        builder.setBindAddress("master")
                .setPort(8888)
                .setProtocol(ClientNamenodeProtocol.class)
                .setInstance(new MyNameNode());

        Server service = builder.build();
        service.start();

    }
}
```

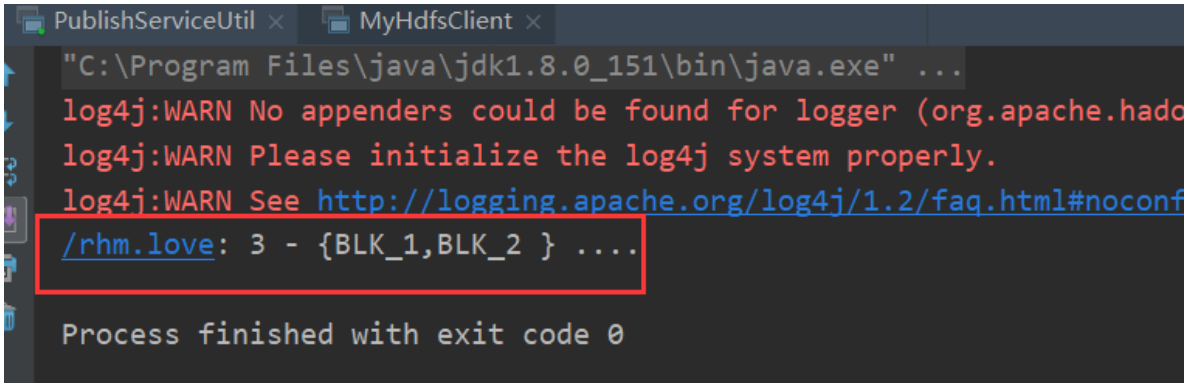
客户端

> 需要注意的是客户端使用的接口的完整签名必须跟服务端使用的一致。
>
> 这里我为了方便，直接在一个项目中写客户端了。如果服务端跟客户端项目不同，一定要记得这一点。

```
# MyHdfsClient.java
package cn.neu.connection.RPCtest.client;

import cn.neu.connection.RPCtest.protocol.ClientNamenodeProtocol;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.ipc.RPC;
import java.net.InetSocketAddress;

public class MyHdfsClient {

    public static void main(String[] args) throws  Exception{

        ClientNamenodeProtocol namenode =
RPC.getProxy(ClientNamenodeProtocol.class,1L,new
InetSocketAddress("master",8888),new Configuration());

        String metaData = namenode.getMetaData("/rhm.love");
```

```
            System.out.println(metaData);
        }
    }
```

结果如下图所示：



Hadoop RPC 类详解

主要由以下三类组成：
RPC：对外编程接口
Client：客户端实现
Server：服务器实现

ipc.RPC类分析

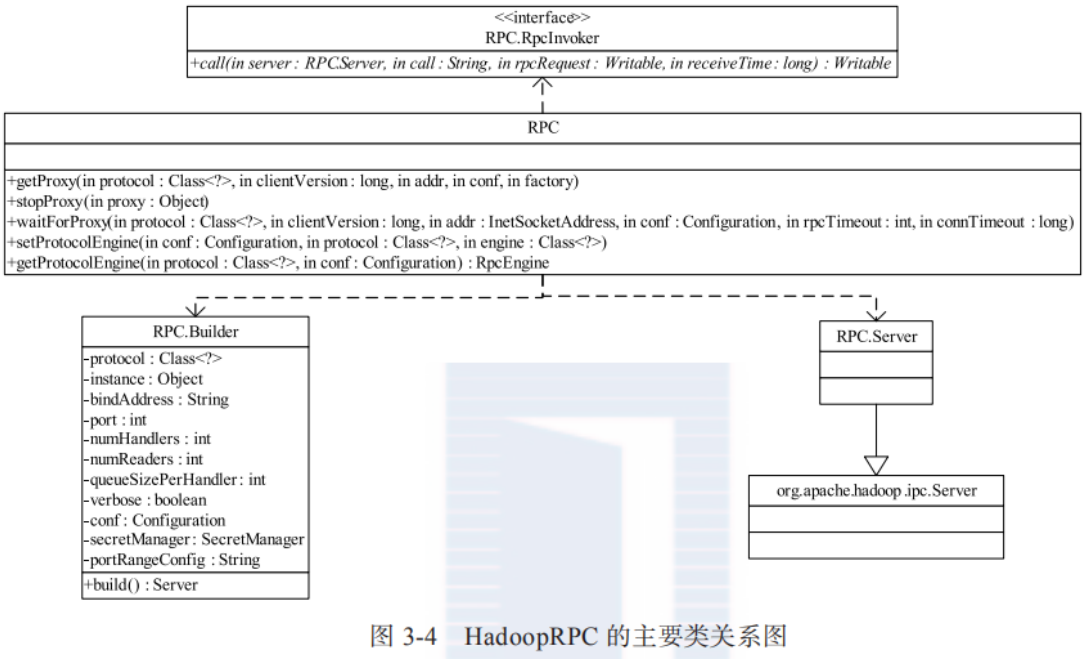


图 3-4　HadoopRPC 的主要类关系图

RPC类实际上是对底层客户机-服务器网络模型的封装，以便为程序员提供一套更方便简洁的编程接口。

- RPC客户端构建方法

```
getProxy()

waitForProxy()
```

- RPC客户端销毁方法

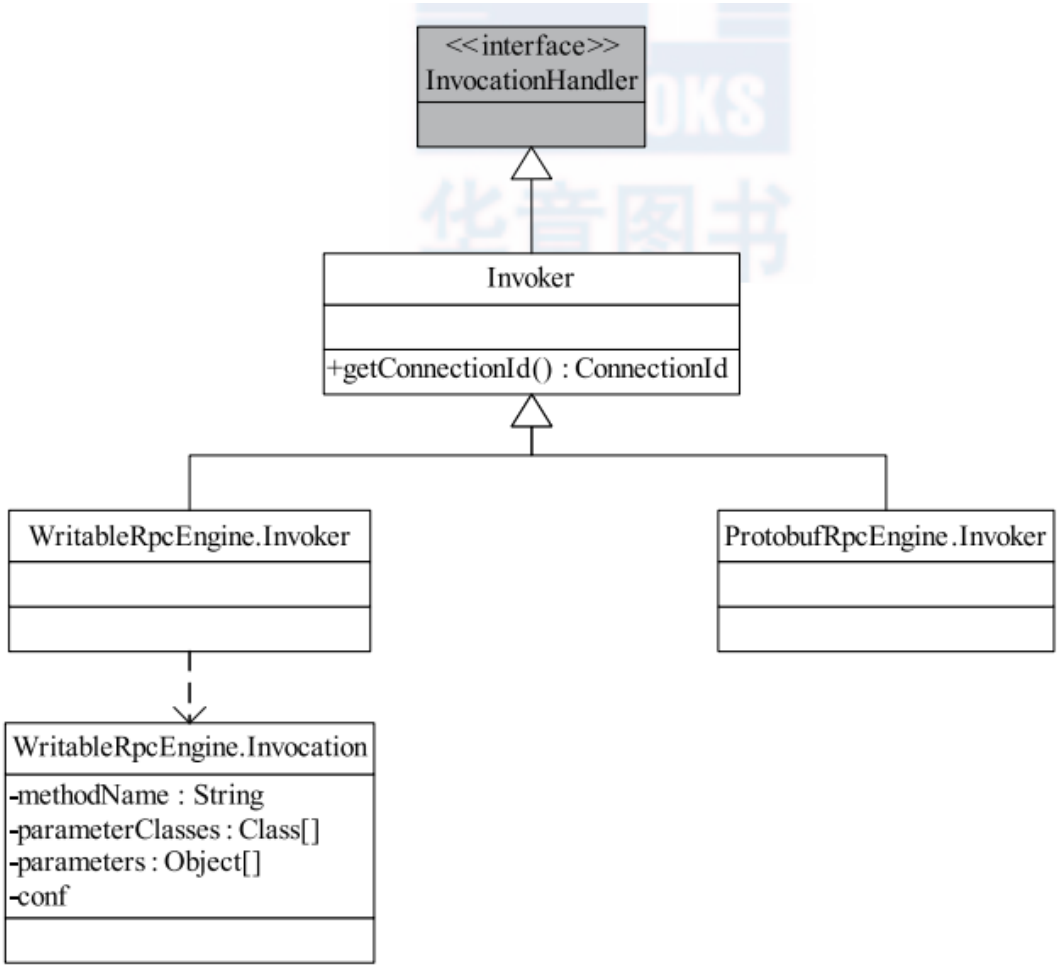
```
stopProxy()
```

- RPC服务器的构建

有静态内部类RPC.Builder，该类提供了一系列setXxx(某个参数名)供用户设置一些基本的参数，如：RPC协议，RPC协议实现对象，服务器绑定地址。完成参数设置后，可以通过调用RPC.Builder.build()完成一个服务器对象的构建。
之后直接调用Server.start()方法启动该服务器。

Hadoop RPC的序列化框架：

目前提供了Writable(WritableRpcEngine)和Protocol Buffers(ProtobufRpcEngine)两种，默认实现是Writable，用户可以调用RPC.setProtocolEngine()修改采用的方法



Hadoop RPC使用了Java动态代理完成对远程方法的调用：
1. 实现实现java.lang.reflect.InvocationHandler接口
2. 按照自己需求实现invoke(方法)

ps:在invoke方法中，将函数调用信息（函数名，函数参数列表等）打包成可序列化的WritableRpcEngine.Invocation对象，并通过网络发送给服务端，服务端收到该调用信息后，解析出和函数名，函数参数列表等信息

ipc.Client

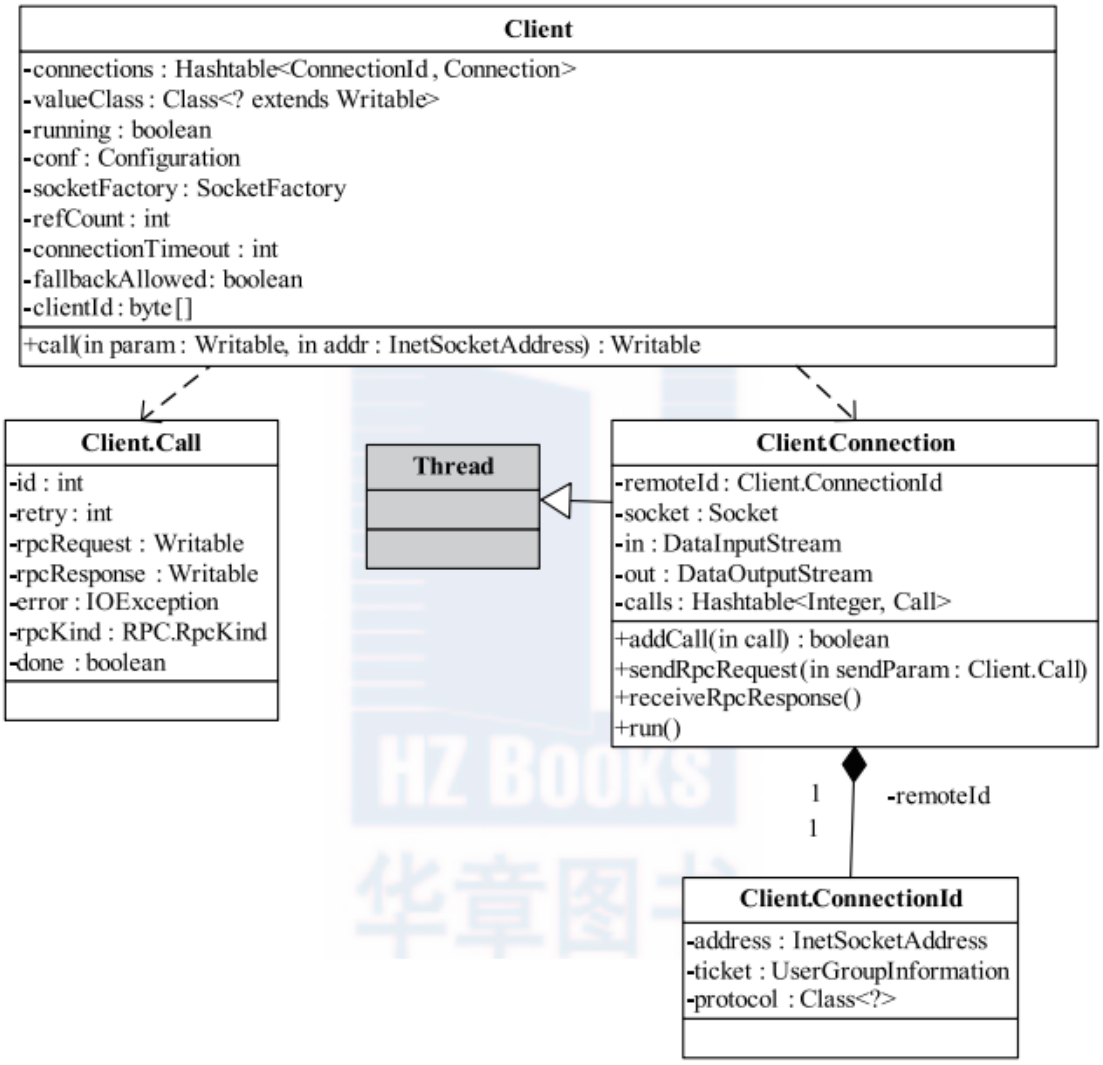


图 3-6　Client 类图

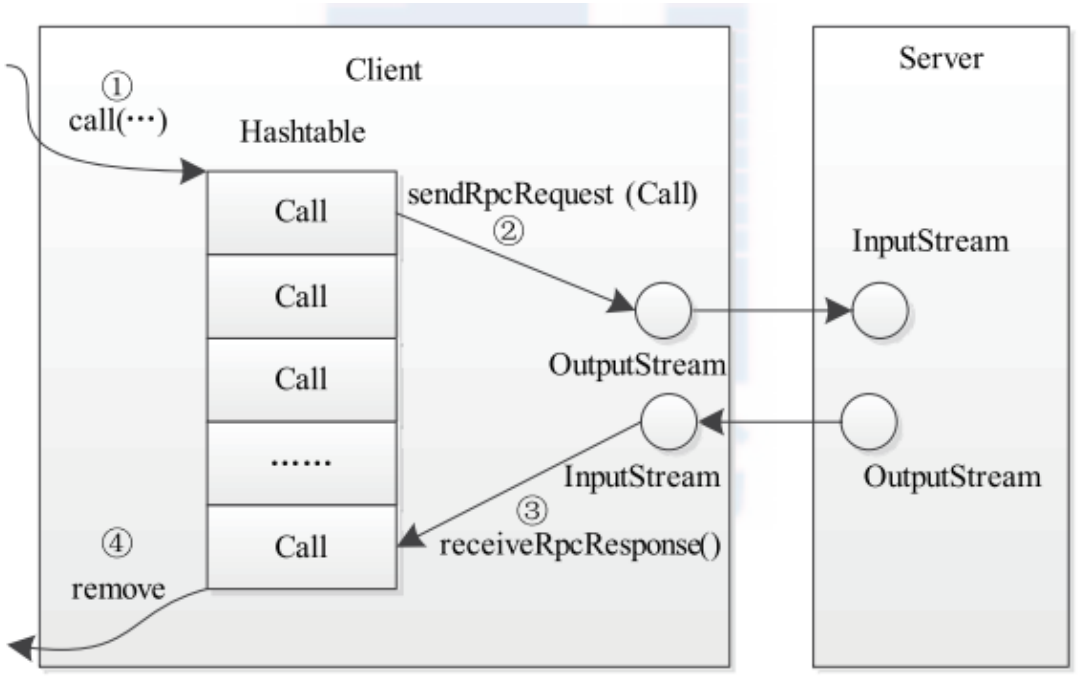
Client主要完成的功能是发送远程过程调用信息并接受执行结果。Client类对外提供了一类执行进程调用的接口，这些接口的名称一样，仅仅是参数列表不同。

Client内部有两个重要的内部类，分别是Call和Connection:

Call类：　封装一个RPC请求，包含以下五个成员变量：
1. 唯一标识 id
2. 函数调用信息 param
3. 函数执行返回值 value
4. 出错或者异常信息 error

5. 执行完成标识符 done
当客户端向服务器端发出请求时，只需要填充id和param两个变量，其他有服务器端自己填充

Connect类：封装Client与每个Server的通信连接的基本信息及操作
基本信息包括：
1. 通信连接唯一标识（remoteId)
2. 与Server端通信的Socket(socket)
3. 网络输入数据流（in）
4. 网络输出数据流（out）
5. 保存RPC请求的哈希表（calls）
操作包括：
1. addCall:将一个Call对象添加到哈希表中
2. sendParam: 向服务器端发送RPC请求
3. receiveResponse:从服务器端接受已经处理完的RPC请求
4. run:Connection是一个线程类，run调用了receiveResponse方法



当调用cal函数执行某个远程方法时，Client端需要进行下面步骤：
1. 创建一个Connection对象，并将远程方法调用信息封装成Call对象，放到Connnection对象中的哈希表；
2. 调用Connection类中的sendRPpcRequest()方法将当前的Call对象发送给Server端
3. Server端处理完RPC请求后，将结果通过网络返回给Client端，Client端通过receiveRpcRespondse()函数获取结果；
4. Client检查结果处理状态，并将对应的Call对象从哈希表中删除

ipc.Server类分析

背景

> Hadoop采用了Master/Slave结构，而Master通过ipc.Server接收并处理所有的Slave发送的请求。因此ipc.Server采用了很多提高并发处理能力的技术，其中主要包括线程池，事件驱动和Reactor设计模式。

Reactor模式工作原理

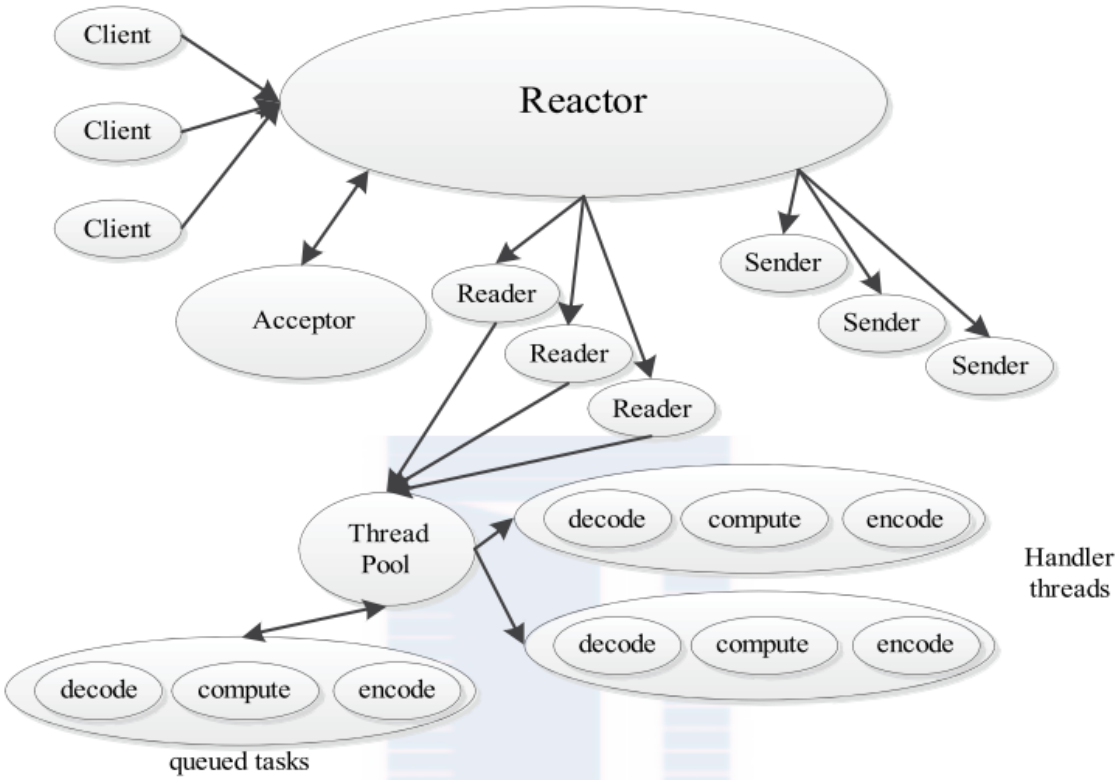
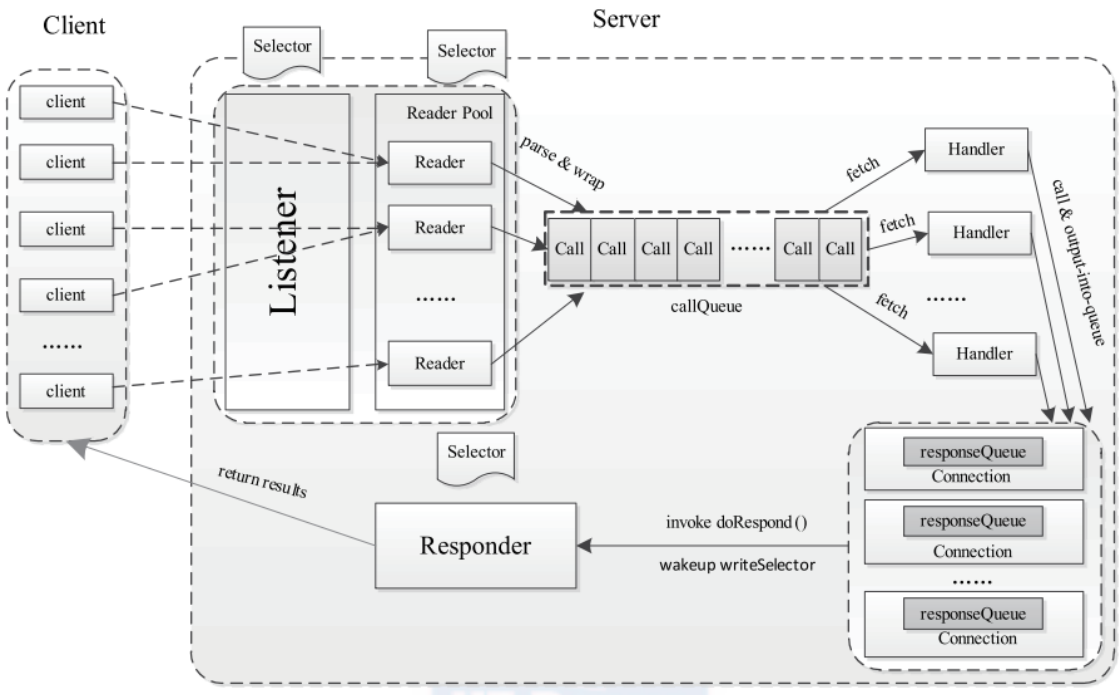


图 3-8　Reactor 模式工作原理

Reactor是并发编程中的一种基于事件驱动的设计模式，具有两个以下的特点：通过派发/分离I/O操作事件提高系统的并发能力；提供了粗粒度的并发控制，使用单线程实现，避免了复杂的同步处理。典型的Reactor模式主要包括以下角色：

- Reactor：I/O事件的派发者
- Acceptor： 接受来自Client的连接，建立与Client对应的Handler，并向Reactor注册Handler
- Handler：与一个Client通信的实体。
- Reader/Sender:为了加速处理速度，Reactor模式一般分离Handler中的读和写过程，分别注册成单独的读事件和写事件，并由对应的Reader和Sender现场处理

ipc.Server实现细细节



> ipc.Server被划分成三个阶段：
>
> 1. 接收请求
> 2. 处理请求
> 3. 返回结果

接收请求

> 工作： 接收来自各个客户端得RPC请求，并将他们封装成固定的格式（Call类）放到一个共享队列(callQueue)中，以便后续处理。

该阶段内部又分为建立连接和接受请求两个子阶段，分别由Listener和Reader两种线程完成。

```
Listener：
1. 整个Server只有一个Listener线程，负责监听来自客户端的连接请求，一旦有一个新的请求到达，就会采用轮询的方式从线程池中选择一个Reader线程进行处理。
2. Listener包含一个Selector对象，用于监听SelectionKey.OP_ACCEPT。对于Listener线程，主循环的实现体是监听是否有新的连接请求到达，并采用用轮询策略选择一个Reader线程处理新连接。

Server：
1. 整个Reader线程可同时存在多个，它们分别负责接收一部分客户端连接的RPC请求，至于每个Reader负责哪些客户端连接，完全由Listener决定。
2. Listener包含一个Selector对象，用于监听SelectionKey.OP_READ。对Reader线程，主循环的实现体是监听（它负责的那部分）客户端连接中是否新的RPC请求到达，并将新的RPC请求封装成Call对象，放到共享队列callQueue.
```

处理请求

主要都是Handler完成：

```
该阶段主要任务是从共享队列callQueue中获取Call对象，执行对应的函数调用，并将结果返回给客户端，这全部都由Handler线程完成。

如果某些函数调用返回结果很大或者网络速度过慢，可能难以将结果一次性发送到客户端，此使handler尝试将后续发送任务交给Reasponder线程
```

返回结果

```
Server端仅存一个Responder线程，它内部包含一个Selector对象，用于监听SelectionKey.OP_WRITE事件。当Handler没能将结果一次性发送到客户端，会向该Selector对象注册SelectionKey.OP_WRITE事件，进而Responder线程采用异步方式继续发送未完成的结果
```

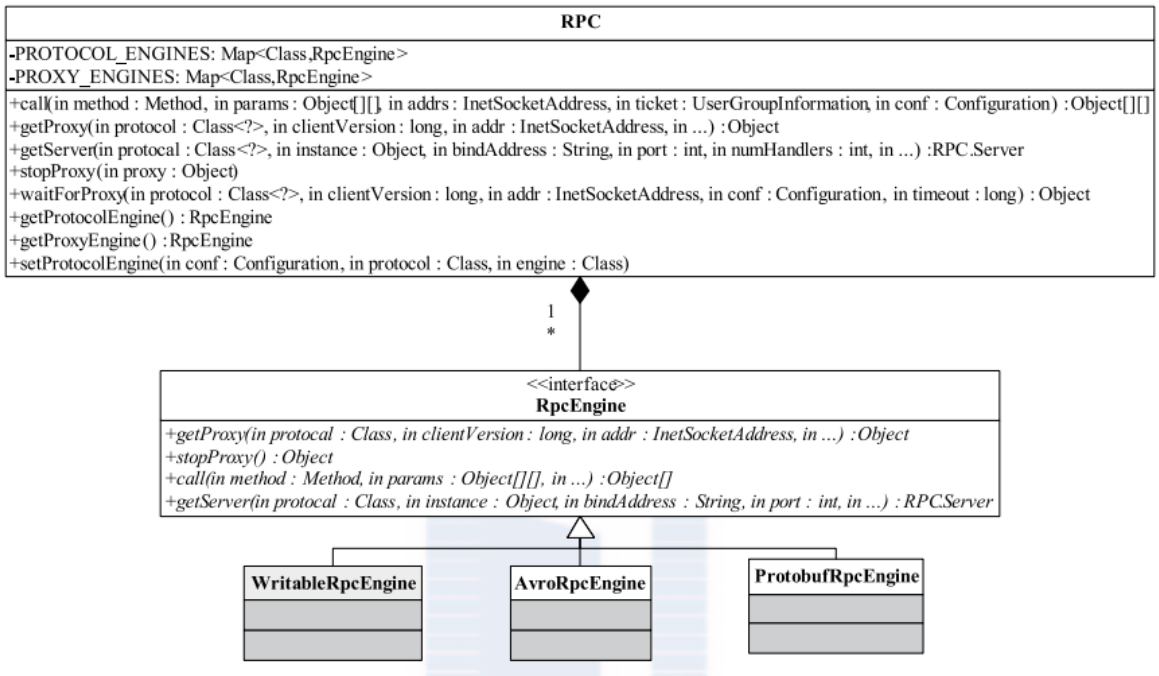
Hadoop RPC 参数调优

主要配置参数如下：

- **Reader线程数目。**由参数ipc.server.read.threadpool.size配置，默认是1，也就是说，默认情况下，一个RPC Server只包含一个Reader线程。
- **每个Handler线程对应的最大Call数目。**由ipc.server.handler.queue.size指定，默认100.。比如：如果Handler数目为10，则整个Call队列最大长度为100 × 10 = 1000

- **客户端最大重试次数**：由ipc.client.connect.max.retries指定，默认值为10，也就是会尝试10次（每次相隔1秒）

YARN RPC实现



> RPC类变成了一个工厂，它将具体的RPC实现授权给了RpcEngine实现类，在该图中，WritableRpcEngine是才用Hadoop自带的序列化框架实现的RPC，而AvroRpcEngine和ProtobufRpcEinge分别是开源的RPC框架框架。
>
> 用户可以通过配置参数rpc.engine.{protocol}以指定协议采用的序列化方式。
>
> YARN提供的对外类是YarnRPC，用户只需要使用该类便可以构建一个基于Hadoop RPC且采用Protocol Buffer序列化框架的通信协议。