1. Spark 的 RPC 概述

在Spark中很多地方都涉及网络通信，比如Spark各个组件间的消息互通、用户文件与Jar包的上传、节点间的Shuffle过程、Block数据的复制与备份等。

如果把分布式系统（HDFS, HBASE，SPARK等）比作一个人，那么RPC可以认为是人体的血液循环系统。它将系统中各个不同的组件（如HBase中的master， Regionserver， client）联系了起来。同样，在spark中，不同组件像driver，executor，worker，master（stanalone模式）之间的通信也是基于RPC来实现的。

Spark 是一个 通用的分布式计算系统，既然是分布式的，必然存在很多节点之间的通信，那么 Spark 不同组件之间就会通过 RPC（Remote Procedure Call）进行点对点通信。

spark中网络通信无处不在，例如

```
- driver和master的通信，比如driver会想master发送RegisterApplication消息
- master和worker的通信，比如worker会向master上报worker上运行Executor信息
- executor和driver的的通信，executor运行在worker上，spark的tasks被分发到运行在各个executor中，executor需要通过向driver发送任务运行结果。
- worker和worker的通信，task运行期间需要从其他地方fetch数据，这些数据是由运行在其他worker上的executor上的task产生，因此需要到worker上fetch数据
```

总结起来通信主要存在两个方面：

```
1. 汇集信息，例如task变化信息，executor状态变化信息。
2. 传输数据，spark shuffle（也就是reduce从上游map的输出中汇集输入数据）阶段存在大量的数据传输。
```

2. Spark 的 RPC 通信框架版本更迭

Spark 1.6之前，Spark 的 RPC 是基于 Akaa 来实现的。Akka 是一个基于 scala 语言的异步的消息框架。Spark1.6 后，Spark 借鉴 Akka 的设计自己实现了一个基于 Netty 的 rpc 框架。大概的原因是 1.6 之前，RPC 通过 Akka 来实现，而大文件是基于 netty 来实现的，加之akka 版本兼容性问题，所以 1.6 之后把 Akka 改掉了，具体jira见（https://issues.apache.org/jira/browse/SPARK-5293）。

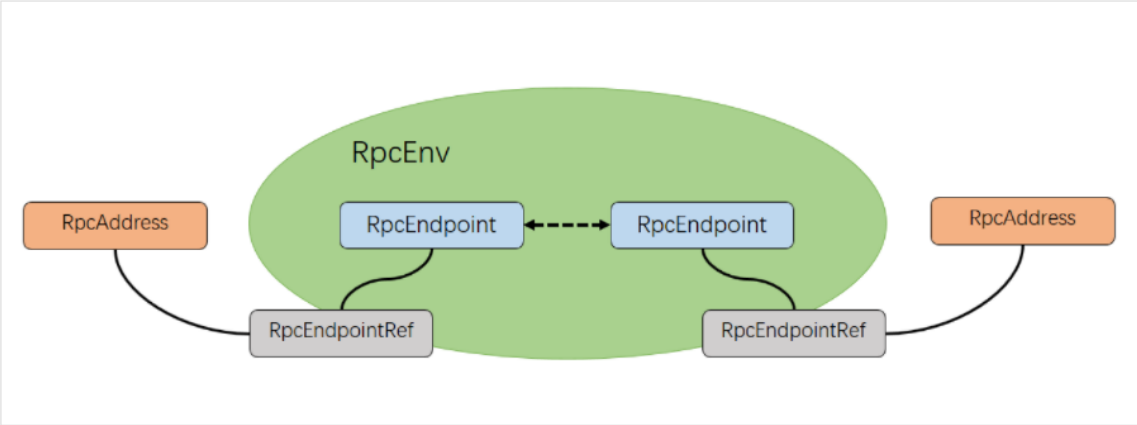
Spark 早期版本中使用 Netty 通信框架做大块数据的传输，使用 Akka 用作 RPC 通信。

Spark 的 RPC 主要在 spark-network-common 中，该模块是 java 语言写的，最新版本是基于 Netty 开发的。

Akka 在 Spark 2.0.0 版本中被移除了，Spark 官网文档对此的描述为：“Akka 的依赖被移除了，因此用户可以使用任何版本的 Akka 来编程了。”Spark 团队的决策者或许认为对于 Akka 具体版本的依赖，限制了用户对于 Akka 不同版本的使用。尽管如此，笔者依然认为 Akka 是一款非常优秀的开源分布式系统，我参与的一些 Java Application 或者 Java Web 就利用 Akka 的丰富特性实现了分布式一致性、最终一致性以及分布式事务等分布式环境面对的问题。在 Spark 1.x.x 版本中，用户文件与 Jar 包的上传采用了由 Jetty[2]实现的HttpFileServer，但在 Spark 2.0.0 版本中也被废弃了，现在使用的是基于 Spark 内置 RPC 框架的NettyStreamManager。节点间的Shuffle 过程和 Block 数据的复制与备份这两个部分在 Spark 2.0.0 版本中依然沿用了 Netty[3]，通过对接口和程序进行重新设计将各个组件间的消息互通、用户文件与 Jar 包的上传等内容统一纳入到 Spark 的 RPC 框架体系中。

3. Spark 的 RPC 组成

spark 基于netty新的rpc框架借鉴了Akka的中的设计，它是基于Actor模型，各个组件可以认为是一个个独立的实体，各个实体之间通过消息来进行通信。



1、RpcEndPonit 和 RpcCallContext

`RpcEndPoint` 是一个可以相应请求的服务，类似于 `Akka` 中的 `Actor`。其中有 `receive` 方法用来接收客户端发送过来的信息，也有 `receiveAndReply` 方法用来接收并应答，应答通过 `RpcContext` 回调。

表示一个个需要通信的个体（如`master`，`worker`，`driver`），主要根据接收的消息来进行对应的处理。一个`RpcEndpoint`经历的过程依次是：构建 `->` `onStart` `->` `receive` `->` `onStop`。其中`onStart`在接收任务消息前调用，`receive`和`receiveAndReply`分别用来接收另一个`RpcEndpoint`（也可以是本身）`send`和`ask`过来的消息。

2、RpcEndpointRef

类似于 `Akka` 中的 `ActorRef`，是 `RpcEndPoint` 的引用，持有远程 `RpcEndPoint` 的地址名称等，提供了 `send` 方法和 `ask` 方法用于发送请求。

`RpcEndpointRef` 是对远程 `RpcEndpoint` 的一个引用。当我们需要向一个具体的 `RpcEndpoint` 发送消息时，一般我们需要获取到该 `RpcEndpoint` 的引用，然后通过该引用发送消息。

3、RpcEnv 和 NettyRpcEnv

RpcEnv 类似于 ActorSystem，服务端和客户端都可以使用它来做通信。

对于 server 端来说，RpcEnv 是 RpcEndpoint 的运行环境，负责 RpcEndPoint 的生命周期管理，解析 Tcp 层的数据包以及反序列化数据封装成 RpcMessage，然后根据路由传送到对应的 Endpoint；

对于 client 端来说，可以通过 RpcEnv 获取 RpcEndpoint 的引用，也就是 RpcEndpointRef，然后通过 RpcEndpointRef 与对应的 Endpoint 通信。

RpcEnv 为 RpcEndpoint 提供处理消息的环境。RpcEnv 负责 RpcEndpoint 整个生命周期的管理，包括：注册endpoint，endpoint 之间消息的路由，以及停止 endpoint。

4、Dispacher 与 Inbox 与 Outbox

NettyRpcEnv 中包含 Dispatcher，主要针对服务端，帮助路由到指定的 RpcEndPoint，并调用起业务逻辑。

RpcEndpoint：RPC端点 ，Spark针对于每个节点（Client/Master/Worker）都称之一个Rpc端点 ,且都实现RpcEndpoint接口，内部根据不同端点的需求，设计不同的消息和不同的业务处理，如果需要发送（询问）则调用Dispatcher

Dispatcher：消息分发器，针对于RPC端点需要发送消息或者从远程RPC接收到的消息，分发至对应的指令收件箱/发件箱。如果指令接收方是自己存入收件箱，如果指令接收方为非自身端点，则放入发件箱

Inbox：指令消息收件箱，一个本地端点对应一个收件箱，Dispatcher在每次向Inbox存入消息时，都将对应EndpointData加入内部待Receiver Queue中，另外Dispatcher创建时会启动一个单独线程进行轮询Receiver Queue，进行收件箱消息消费

OutBox：指令消息发件箱，一个远程端点对应一个发件箱，当消息放入Outbox后，紧接着将消息通过TransportClient发送出去。消息放入发件箱以及发送过程是在同一个线程中进行，这样做的主要原因是远程消息分为RpcOutboxMessage，OneWayOutboxMessage两种消息，而针对于需要应答的消息直接发送且需要得到结果进行处理

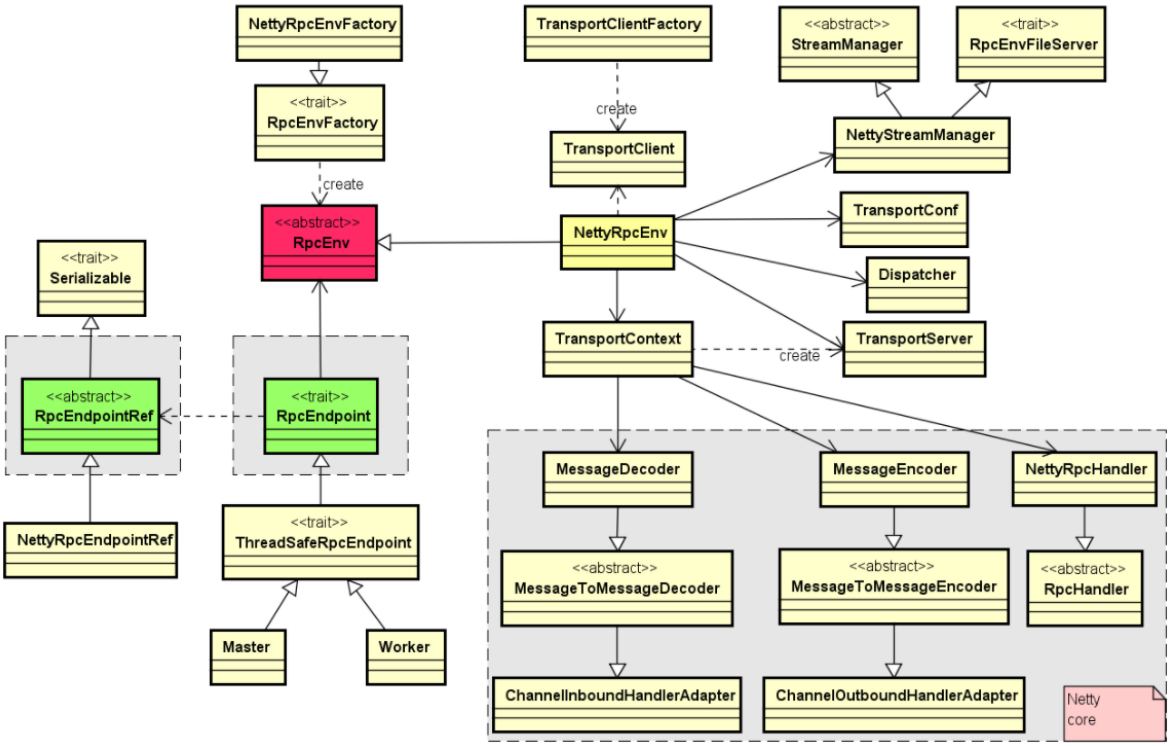
TransportClient：Netty通信客户端，根据OutBox消息的receiver信息，请求对应远程TransportServer

TransportServer：Netty通信服务端，一个RPC端点一个TransportServer,接受远程消息后调用Dispatcher分发消息至对应收发件箱

5、RpcAddress

表示远程的 RpcEndpointRef 的地址，Host + Port。

4. Spark 的 RPC 类图关系



核心要点如下：

1，核心的 RpcEnv 是一个特质（trait），它主要提供了停止，注册，获取 endpoint 等方法的定义，而 NettyRpcEnv 提供了该特质的一个具体实现。

2，通过工厂 RpcEnvFactory 来产生一个 RpcEnv，而 NettyRpcEnvFactory 用来生成 NettyRpcEnv 的一个对象。

3，当我们调用 RpcEnv 中的 setupEndpoint 来注册一个 endpoint 到 rpcEnv 的时候，在 NettyRpcEnv 内部，会将该 endpoint 的名称与其本省的映射关系，rpcEndpoint 与 rpcEndpointRef 之间映射关系保存在 dispatcher 对应的成员变量中。

5. 简单示例

见代码！