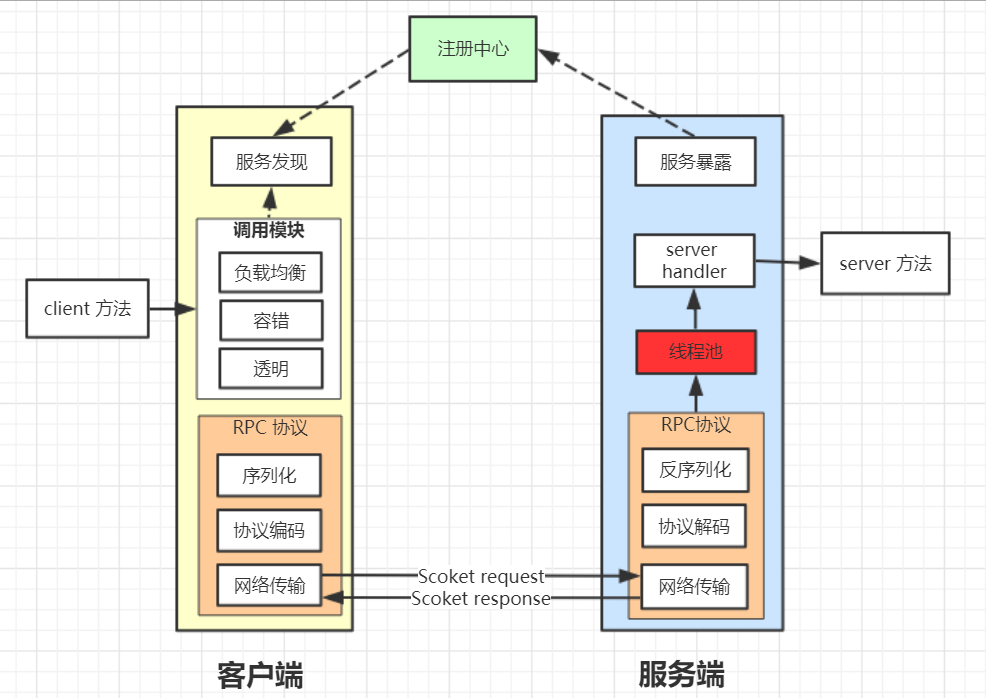
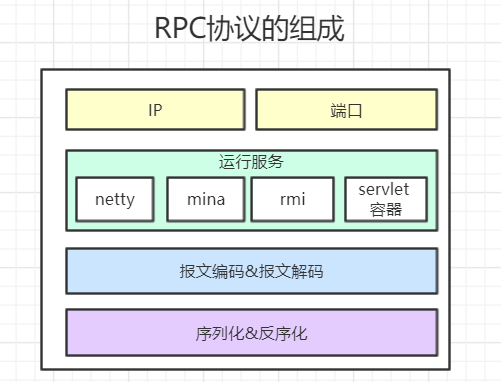
**RPC协议基本组成**

**RPC 协议名词解释**

在一个典型RPC的使用场景中，包含了服务发现、负载、容错、网络传输、序列化等组件，其中RPC协议就指明了程序如何进行网络传输和序列化 。也就是说一个RPC协议的实现就等于一个非透明的远程调用实现，如何做到的的呢？



**协议基本组成：**



1. 地址：服务提供者地址
2. 端口：协议指定开放的端口
3. 报文编码：协议报文编码 ，分为请求头和请求体两部分。
4. 序列化方式：将请求体序列化成对象
   1. Hessian2Serialization、
   2. DubboSerialization、
   3. JavaSerialization
   4. JsonSerialization
5. 运行服务: 网络传输实现
   1. netty
   2. mina
   3. RMI 服务
   4. servlet 容器（jetty、Tomcat、Jboss）

**Dubbo中所支持RPC协议使用**

**dubbo 支持的RPC协议列表**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **实现描述** | **连接描述** | **适用场景** |
| **dubbo** | 传输服务: mina, netty(默认), grizzy  序列化: hessian2(默认), java, fastjson  自定义报文 | 单个长连接  NIO异步传输 | 1、常规RPC调用  2、传输数据量小  3、提供者少于消费者 |
| **rmi** | 传输：java rmi 服务  序列化：java原生二进制序列化(固定) | 多个短连接  BIO同步传输 | 1、常规RPC调用  2、与原RMI客户端集成  3、可传少量文件  4、不支持防火墙穿透 |
| **hessian** | 传输服务：servlet容器  序列化：hessian二进制序列化（固定） | 基于Http 协议传输，  依懒servlet容器配置 | 1、提供者多于消费者  2、可传大字段和文件  3、跨语言调用 |
| **http** | 传输服务：servlet容器  序列化：java原生二进制序列化（固定） | 依懒servlet容器配置 | 1、数据包大小混合 |
| **thrift** | 与thrift RPC 实现集成，并在其基础上修改了报文头 | 长连接、NIO异步传输 |  |

***关于RMI不支持防火墙穿透的补充说明：***

原因在于RMI 底层实现中会有两个端口，一个是固定的用于服务发现的注册端口，另外会生成一个***随机***端口用于网络传输。因为这个随机端口就不能在防火墙中提前设置开放开。所以存在*防火墙穿透问题*

**协议的使用与配置:**

Dubbo框架配置协议非常方便，用户只需要在  provider 应用中 配置*<dubbo:protocol>* 元素即可。

 <!--

   name: 协议名称 dubbo|rmi|hessian|http|

   host:本机IP可不填，则系统自动获取

   port：端口、填-1表示系统自动选择

   server：运行服务  mina|netty|grizzy|servlet|jetty

   serialization：序列化方式 hessian2|java|compactedjava|fastjson

   详细配置参见dubbo 官网 dubbo.io

 -->

 <dubbo:protocol name="dubbo" host="192.168.0.11" port="20880" server="netty"

  serialization=“hessian2” charset=“UTF-8” />

#TODO 演示采用其它协议来配置Dubbo

* dubbo 协议采用 json 进行序列化  (源码参见：com.alibaba.dubbo.rpc.protocol.dubbo.DubboProtocol*)*
* 采用RMI协议 (源码参见：*com.alibaba.dubbo.rpc.protocol.rmi.RmiProtocol)*
* 采用Http协议 (源码参见：*com.alibaba.dubbo.rpc.protocol.http.HttpProtocol.InternalHandler)*
* 采用Heason协议 (源码参见:com.alibaba.dubbo.rpc.protocol.hessian.HessianProtocol.HessianHandler)

netstat -aon|findstr "17732"

序列化：

|  |  |
| --- | --- |
|  | 特点 |
| fastjson | 文本型：体积较大，性能慢、跨语言、可读性高 |
| fst | 二进制型：体积小、兼容 JDK 原生的序列化。要求 JDK 1.7 支持。 |
| hessian2 | 二进制型：跨语言、容错性高、体积小 |
| java | 二进制型：在JAVA原生的基础上 可以写入Null |
| compactedjava | 二进制型：与java 类似，内容做了压缩 |
| nativejava | 二进制型：原生的JAVA 序列化 |
| kryo | 二进制型：体积比hessian2 还要小，但容错性 没有hessian2 好 |

Hessian 序列化：

* 参数及返回值需实现 Serializable 接口
* 参数及返回值不能自定义实现 List, Map, Number, Date, Calendar 等接口，只能用 JDK 自带的实现，因为 hessian 会做特殊处理，自定义实现类中的属性值都会丢失。
* Hessian 序列化，只传成员属性值和值的类型，不传方法或静态变量，兼容情况 [[1]](http://dubbo.apache.org/zh-cn/docs/user/references/protocol/dubbo.html#fn1)[[2]](http://dubbo.apache.org/zh-cn/docs/user/references/protocol/dubbo.html#fn2)：

|  |  |  |
| --- | --- | --- |
| **数据通讯** | **情况** | **结果** |
| A->B | 类B少一种 属性 | 不抛异常，A多的那 个属性的值，B没有， 其他正常 |
| A->B | 枚举A多一种 枚举（或者说B少一种 枚举），A使用多 出来的枚举进行传输 | 抛异常 |
| A->B | 枚举A多一种 枚举（或者说B少一种 枚举），A不使用 多出来的枚举进行传输 | 不抛异常，B正常接 收数据 |
| A->B | A和B的属性 名相同，但类型不相同 | 抛异常 |
| A->B | serialId 不相同 | 正常传输 |

接口增加方法，对客户端无影响，如果该方法不是客户端需要的，客户端不需要重新部署。输入参数和结果集中增加属性，对客户端无影响，如果客户端并不需要新属性，不用重新部署。

输入参数和结果集属性名变化，对客户端序列化无影响，但是如果客户端不重新部署，不管输入还是输出，属性名变化的属性值是获取不到的。

总结：服务器端和客户端对领域对象并不需要完全一致，而是按照最大匹配原则。

**三 、RPC协议报文编码与实现详解**

**RPC 传输实现：**

RPC的协议的传输是基于 TCP/IP 做为基础使用Socket 或Netty、mina等网络编程组件实现。但有个问题是TCP是面向字节流的无边边界协议，其只管负责数据传输并不会区分每次请求所对应的消息，这样就会出现TCP协义传输当中的拆包与粘包问题

**拆包与粘包产生的原因：**

我们知道tcp是以流动的方式传输数据，传输的最小单位为一个报文段（segment）。tcp Header中有个Options标识位，常见的标识为mss(Maximum Segment Size)指的是，连接层每次传输的数据有个最大限制MTU(Maximum Transmission Unit)，一般是1500比特，超过这个量要分成多个报文段，mss则是这个最大限制减去TCP的header，光是要传输的数据的大小，一般为1460比特。换算成字节，也就是180多字节。

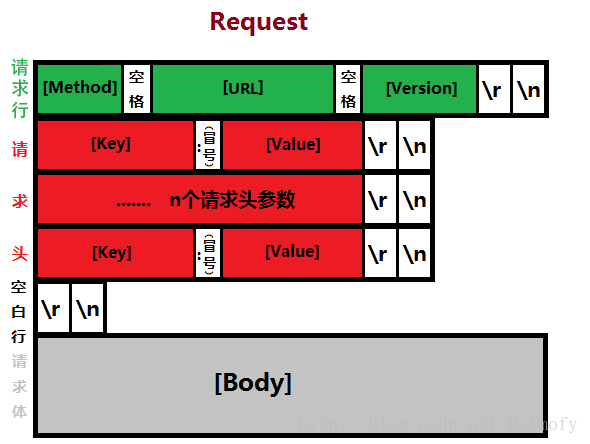
tcp为提高性能，发送端会将需要发送的数据发送到缓冲区，等待缓冲区满了之后，再将缓冲中的数据发送到接收方。同理，接收方也有缓冲区这样的机制，来接收数据。这时就会出现以下情况：

1. 应用程序写入的数据大于MSS大小，这将会发生拆包。
2. 应用程序写入数据小于MSS大小，这将会发生粘包。
3. 接收方法不及时读取套接字缓冲区数据，这将发生粘包。

**拆包与粘包解决办法：**

1. 设置定长消息，服务端每次读取既定长度的内容作为一条完整消息。
2. {"type":"message","content":"hello"}\n
3. 使用带消息头的协议、消息头存储消息开始标识及消息长度信息，服务端获取消息头的时候解析出消息长度，然后向后读取该长度的内容。

**比如：**Http协议 heade 中的 Content-Length 就表示消息体的大小。

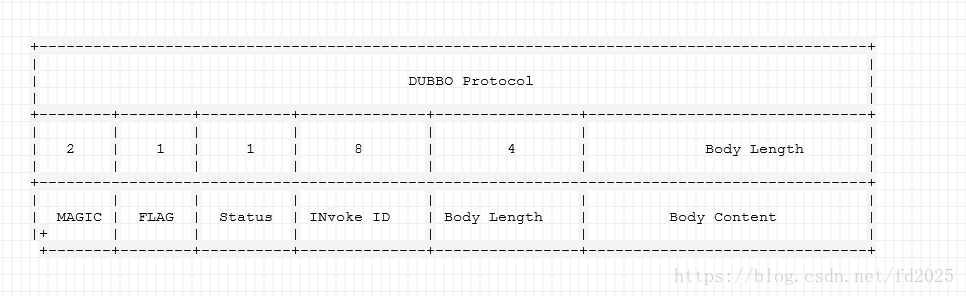


(注①：http 报文编码)

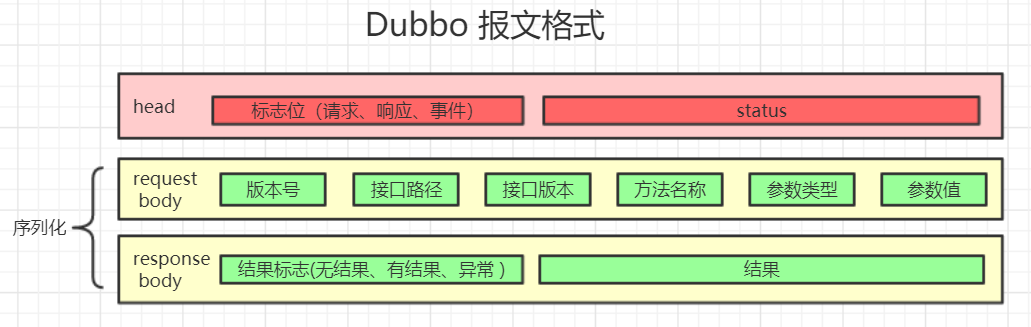
Dubbo 协议报文编码：

**注②Dubbo  协议报文编码：**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0-7 | 8-15 | 16-20 | 21 | 22 | 23 | 24-31 |  |
|  |  | 1 | 1 |  |  |  |  |  |
| 32-95 |  |  |  |  |  |  |  |  |
| 96-127 |  |  |  |  |  |  |  |  |

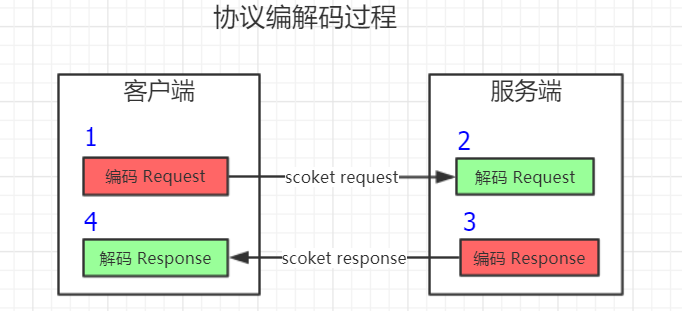


* **magic**：类似java字节码文件里的魔数，用来判断是不是dubbo协议的数据包。魔数是常量0xdabb,用于判断报文的开始。
* **flag**：标志位, 一共8个地址位。低四位用来表示消息体数据用的序列化工具的类型（默认hessian），高四位中，第一位为1表示是request请求，第二位为1表示双向传输（即有返回response），第三位为1表示是心跳ping事件。
* **status**：状态位, 设置请求响应状态，dubbo定义了一些响应的类型。具体类型见 com.alibaba.dubbo.remoting.exchange.Response
* **invoke id：**消息id, long 类型。每一个请求的唯一识别id（由于采用异步通讯的方式，用来把请求request和返回的response对应上）
* **body length：**消息体 body 长度, int 类型，即记录Body Content有多少个字节。



*（注：相关源码参见 com.alibaba.dubbo.rpc.protocol.dubbo.DubboCodec）*

***Dubbo协议的编解码过程：***



**Dubbo 协议编解码实现过程** *(源码来源于dubbo2.5.8  )*

1、DubboCodec.encodeRequestData() 116L // 编码request

2、DecodeableRpcInvocation.decode() 89L // 解码request

3、DubboCodec.encodeResponsseData() 184L // 编码response

4、DecodeableRpcResult.decode() 73L // 解码response