登录 | 注册

瞬息之间
写下，记下，留下

个人资料

-mindwind-

访问： 788597次
积分： 8208
等级： BLOG 6
排名： 第1957名

原创： 147篇 转载： 0篇
译文： 8篇 评论： 529条

深入浅出 RPC - 深入篇

标签： RPC 分布式计算

2014-09-22 11:25 132569人阅读 评论(18) 收藏 举报

分类：
踏莎行·术（63）

版权声明：本文为博主原创文章，未经博主允许不得转载。

《深入篇》我们主要围绕 RPC 的功能目标和实现考量去展开，一个基本的 RPC 框架应该提供什么功能，满足什么要求以及如何去实现它？

RPC 功能目标

RPC 的主要功能目标是让构建分布式计算（应用）更容易，在提供强大的远程调用能力时不损失本地调用的语义简洁性。为实现该目标，RPC 框架需提供一种透明调用机制让使用者不必显式的区分本地调用和远程调用，在前文《浅出篇》中给出了一种实现结构，基于 stub 的结构来实现。下面我们将具体细化 stub 结构的实现。

RPC 调用分类

RPC 调用分以下两种：

```
[plain]
1. 同步调用
   客户方等待调用执行完成并返回结果。
2. 异步调用
   客户方调用后不用等待执行结果返回，但依然可以通过回调通知等方式获取返回结果。
   若客户方不关心调用返回结果，则变成单向异步调用，单向调用不用返回结果。
```

异步和同步的区分在于是否等待服务端执行完成并返回结果。

RPC 结构拆解

程序员，你为什么值这么 (14823)
RPC 的概念模型与实现 (12902)
IM设计思考：XMPP多用 (12862)
tomcat 性能之谜 (12543)

评论排行

程序员，别了校园入了江 (49)
程序员，你为什么值这么 (24)
2016 工作、生活与得失 (22)
深入浅出 RPC - 深入篇 (18)
技术宅找女朋友的技术分 (18)
程序员的沟通之痛 (17)
京东咚咚架构演进 (17)
程序员的口头禅：技术上 (15)
Java 征途：行者的地图 (14)
成长路上，刀剑如梦 (14)

最新评论

技术晋升的评定与博弈
jiqiao1892: 有人的地方就是江湖，没有绝对公平

论年度计划的可行性
且听风雨999: @mindfloating:首席你好，哈哈。 我通过一个朋友问到你了。我在一家创业公司，做算法

RPC 使用中的一些注意点
二师兄1986: 拜读，我收入了我的知识图谱

论年度计划的可行性
-mindwind-: 你在哪家公司，做什么的呢？

技术晋升的评定与博弈
-mindwind-: 对，处在两端的人很容易分辨，但中间的那部分就有点看运气了

技术晋升的评定与博弈
l-M: 要比别人优秀1.3~1.5而不是一点点，这样自己的成绩就不会有争议了吧

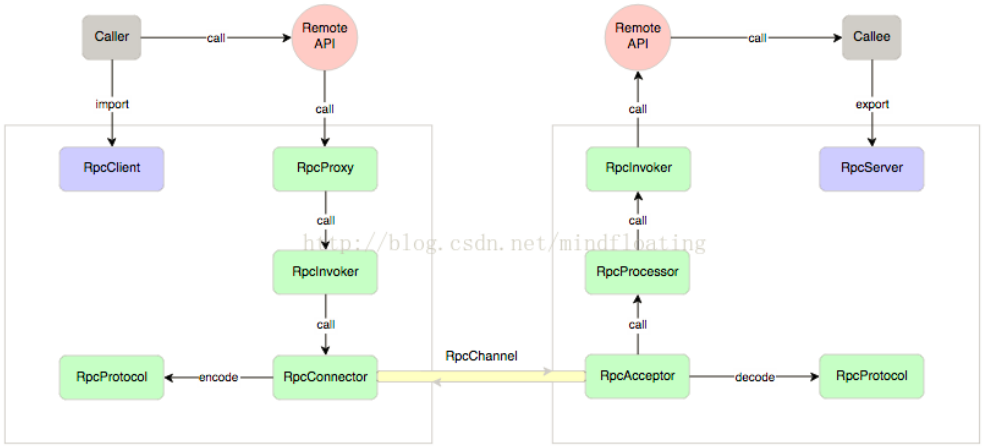
程序员的沟通之痛
冯尧: 站的角度不同，想法也不一样。

技术晋升的评定与博弈
韩梦甜: 博主，很不错的分享！

技术晋升的评定与博弈
mimixi666: @gzy11:搞it这一行的话，这种情况比较少吧

技术晋升的评定与博弈
蜗牛水里爬: 说的在理，实际中，还是暗箱操作的居多。不提名，你连机会都没有。看似公平，其实没有公平。呵呵，其实加薪...

《浅出篇》给出了一个比较粗粒度的 RPC 实现概念结构，这里我们进一步细化它应该由哪些组件构成，如下图所示。



RPC 服务方通过 RpcServer 去导出（export）远程接口方法，而客户方通过 RpcClient 去引入（import）远程接口方法。客户方像调用本地方法一样去调用远程接口方法，RPC 框架提供接口的代理实现，实际的调用将委托给代理RpcProxy 。代理封装调用信息并将调用转交给RpcInvoker 去实际执行。在客户端的RpcInvoker 通过连接器RpcConnector 去维持与服务端的通道RpcChannel，并使用RpcProtocol 执行协议编码（encode）并将编码后的请求消息通过通道发送给服务方。

RPC 服务端接收器 RpcAcceptor 接收客户端的调用请求，同样使用RpcProtocol 执行协议解码（decode）。解码后的调用信息传递给RpcProcessor 去控制处理调用过程，最后再委托调用给RpcInvoker 去实际执行并返回调用结果。

RPC 组件职责

上面我们进一步拆解了 RPC 实现结构的各个组件组成部分，下面我们详细说明下每个组件的职责划分。

```plain
1. RpcServer
   负责导出（export）远程接口
2. RpcClient
   负责导入（import）远程接口的代理实现
3. RpcProxy
   远程接口的代理实现
4. RpcInvoker
   客户方实现：负责编码调用信息和发送调用请求到服务方并等待调用结果返回
   服务方实现：负责调用服务端接口的具体实现并返回调用结果
5. RpcProtocol
   负责协议编/解码
6. RpcConnector
   负责维持客户方和服务方的连接通道和发送数据到服务方
7. RpcAcceptor
   负责接收客户方请求并返回请求结果
8. RpcProcessor
   负责在服务方控制调用过程，包括管理调用线程池、超时时间等
9. RpcChannel
   数据传输通道
```

RPC 实现分析

在进一步拆解了组件并划分了职责之后，这里以在 Java 平台实现该 RPC 框架概念模型为例，详细分析下实现中需要考虑的因素。

导出远程接口

导出远程接口的意思是指只有导出的接口可以供远程调用，而未导出的接口则不能。在 java 中导出接口的代码片段可能如下：

```java
DemoService demo   = new ...;
RpcServer   server = new ...;
server.export(DemoService.class, demo, options);
```

我们可以导出整个接口，也可以更细粒度一点只导出接口中的某些方法，如：

```java
// 只导出 DemoService 中签名为 hi(String s) 的方法
server.export(DemoService.class, demo, "hi", new Class<?>[] { String.class }, options);
```

java 中还有一种比较特殊的调用就是多态，也就是一个接口可能有多个实现，那么远程调用时到底调用哪个？这个本地调用的语义是通过 jvm 提供的引用多态性隐式实现的，那么对于 RPC 来说跨进程的调用就没法隐式实现了。如果前面DemoService 接口有 2 个实现，那么在导出接口时就需要特殊标记不同的实现，如：

```java
DemoService demo   = new ...;
DemoService demo2  = new ...;
RpcServer   server = new ...;
server.export(DemoService.class, demo, options);
server.export("demo2", DemoService.class, demo2, options);
```

上面 demo2 是另一个实现，我们标记为 "demo2" 来导出，那么远程调用时也需要传递该标记才能调用到正确的实现类，这样就解决了多态调用的语义。

导入远程接口与客户端代理

导入相对于导出远程接口，客户端代码为了能够发起调用必须要获得远程接口的方法或过程定义。目前，大部分跨语言平台 RPC 框架采用根据 IDL 定义通过 code generator 去生成 stub 代码，这种方式下实际导入的过程就是通过代码生成器在编译期完成的。我所使用过的一些跨语言平台 RPC 框架如 CORBAR、WebService、ICE、Thrift 均是此类方式。

代码生成的方式对跨语言平台 RPC 框架而言是必然的选择，而对于同一语言平台的 RPC 则可以通过共享接口定义来实现。在 java 中导入接口的代码片段可能如下：

```java
RpcClient client = new ...;
DemoService demo = client.refer(DemoService.class);
demo.hi("how are you?");
```

在 java 中 'import' 是关键字，所以代码片段中我们用 refer 来表达导入接口的意思。这里的导入方[illegible]种代码生成技术，只不过是在运行时生成，比静态编译期的代码生成看起来更简洁些。java 里至少提供了两种技术来提供动态代码生成，一种是 jdk 动态代理，另外一种是字节码生成。动态代理相比字节码生成使用起来更方便，但动态代理方式在性能上是要逊色于直接的字节码生成的，而字节码生成在代码可读性上要差很多。两者权衡起来，个人认为牺牲一些性能来获得代码可读性和可维护性显得更重要。

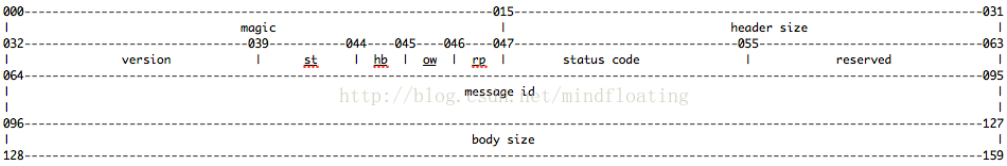
协议编解码

客户端代理在发起调用前需要对调用信息进行编码，这就要考虑需要编码些什么信息并以什么格式传输到服务端才能让服务端完成调用。出于效率考虑，编码的信息越少越好（传输数据少），编码的规则越简单越好（执行效率高）。我们先看下需要编码些什么信息：

```plain
-- 调用编码 --
1. 接口方法
   包括接口名、方法名
2. 方法参数
   包括参数类型、参数值
3. 调用属性
   包括调用属性信息，例如调用附件隐式参数、调用超时时间等

-- 返回编码 --
1. 返回结果
   接口方法中定义的返回值
2. 返回码
   异常返回码
3. 返回异常信息
   调用异常信息
```

除了以上这些必须的调用信息，我们可能还需要一些元信息以方便程序编解码以及未来可能的扩展。这样我们的编码消息里面就分成了两部分，一部分是元信息、另一部分是调用的必要信息。如果设计一种 RPC 协议消息的话，元信息我们把它放在协议消息头中，而必要信息放在协议消息体中。下面给出一种概念上的 RPC 协议消息设计格式：




```plain
-- 消息头 --
magic      : 协议魔数，为解码设计
header size: 协议头长度，为扩展设计
version    : 协议版本，为兼容设计
st         : 消息体序列化类型
hb         : 心跳消息标记，为长连接传输层心跳设计
ow         : 单向消息标记，
rp         : 响应消息标记，不置位默认是请求消息
status code: 响应消息状态码
reserved   : 为字节对齐保留
message id : 消息 id
body size  : 消息体长度

-- 消息体 --
采用序列化编码，常见有以下格式
xml    : 如 webservie soap
json   : 如 JSON-RPC
binary: 如 thrift; hession; kryo 等
```

格式确定后编解码就简单了，由于头长度一定所以我们比较关心的就是消息体的序列化方式。序列化我们关心三个方面：

1. 序列化和反序列化的效率，越快越好。
2. 序列化后的字节长度，越小越好。
3. 序列化和反序列化的兼容性，接口参数对象若增加了字段，是否兼容。

上面这三点有时是鱼与熊掌不可兼得，这里面涉及到具体的序列化库实现细节，就不在本文进一步

传输服务

协议编码之后，自然就是需要将编码后的 RPC 请求消息传输到服务方，服务方执行后返回结果消息或确认消息给客户方。RPC 的应用场景实质是一种可靠的请求应答消息流，和 HTTP 类似。因此选择长连接方式的 TCP 协议会更高效，与 HTTP 不同的是在协议层面我们定义了每个消息的唯一 id，因此可以更容易的复用连接。

既然使用长连接，那么第一个问题是到底 client 和 server 之间需要多少根连接？实际上单连接和多连接在使用上没有区别，对于数据传输量较小的应用类型，单连接基本足够。单连接和多连接最大的区别在于，每根连接都有自己私有的发送和接收缓冲区，因此大数据量传输时分散在不同的连接缓冲区会得到更好的吞吐效率。所以，如果你的数据传输量不足以让单连接的缓冲区一直处于饱和状态的话，那么使用多连接并不会产生任何明显的提升，反而会增加连接管理的开销。

连接是由 client 端发起建立并维持。如果 client 和 server 之间是直连的，那么连接一般不会中断（当然物理链路故障除外）。如果 client 和 server 连接经过一些负载中转设备，有可能连接一段时间不活跃时会被这些中间设备中断。为了保持连接有必要定时为每个连接发送心跳数据以维持连接不中断。心跳消息是 RPC 框架库使用的内部消息，在前文协议头结构中也有一个专门的心跳位，就是用来标记心跳消息的，它对业务应用透明。

执行调用

client stub 所做的事情仅仅是编码消息并传输给服务方，而真正调用过程发生在服务方。server stub 从前文的结构拆解中我们细分了 RpcProcessor 和 RpcInvoker 两个组件，一个负责控制调用过程，一个负责真正调用。这里我们还是以 java 中实现这两个组件为例来分析下它们到底需要做什么？

java 中实现代码的动态接口调用目前一般通过反射调用。除了原生的 jdk 自带的反射，一些第三方库也提供了性能更优的反射调用，因此 RpcInvoker 就是封装了反射调用的实现细节。

调用过程的控制需要考虑哪些因素，RpcProcessor 需要提供什么样地调用控制服务呢？下面提出几点以启发思考：

```plain
1. 效率提升
   每个请求应该尽快被执行，因此我们不能每请求来再创建线程去执行，需要提供线程池服务。
2. 资源隔离
   当我们导出多个远程接口时，如何避免单一接口调用占据所有线程资源，而引发其他接口执行阻塞。
3. 超时控制
   当某个接口执行缓慢，而 client 端已经超时放弃等待后，server 端的线程继续执行此时显得毫无意义。
```

RPC 异常处理

无论 RPC 怎样努力把远程调用伪装的像本地调用，但它们依然有很大的不同点，而且有一些异常情况是在本地调用时绝对不会碰到的。在说异常处理之前，我们先比较下本地调用和 RPC 调用的一些差异：

1. 本地调用一定会执行，而远程调用则不一定，调用消息可能因为网络原因并未发送到服务方。
2. 本地调用只会抛出接口声明的异常，而远程调用还会跑出 RPC 框架运行时的其他异常。
3. 本地调用和远程调用的性能可能差距很大，这取决于 RPC 固有消耗所占的比重。

正是这些区别决定了使用 RPC 时需要更多考量。当调用远程接口抛出异常时，异常可能是一个业务异常，也可能是 RPC 框架抛出的运行时异常（如：网络中断等）。业务异常表明服务方已经执行了调用，可能因为某些原因导致未能正常执行，而 RPC 运行时异常则有可能服务方根本没有执行，对调用方而言的异常处理策略自然需要区分。

由于 RPC 固有的消耗相对本地调用高出几个数量级，本地调用的固有消耗是纳秒级，而 RPC 的固有消耗是在毫秒级。那么对于过于轻量的计算任务就并不合适导出远程接口由独立的进程提供服务，只有花在计算任务上时间远远高于 RPC 的固有消耗才值得导出为远程接口提供服务。

总结

至此我们提出了一个 RPC 实现的概念框架，并详细分析了需要考虑的一些实现细节。无论 RPC 的概念是如何优雅，但是“草丛中依然有几条蛇隐藏着”，只有深刻理解了 RPC 的本质，才能更好地应用。

下面是我自己开的一个微信公众号 [瞬息之间]，除了写技术的文章、还有产品的、行业和人生的思考，希望能和更多走在这条路上同行者交流，有兴趣可关注一下，谢谢。

顶 37 踩 2

上一篇 深入浅出 RPC - 浅出篇
下一篇 又是一年校招

我的同类文章

踏莎行·术（63）

- RPC 使用中的一些注意点 2016-12-25 阅读 555
- HA 高可用软件系统保养指南 2016-06-26 阅读 2437
- HTTPS 互联网世界的安全... 2016-11-20 阅读 1289
- 适合程序员的画图技法 2016-06-05 阅读 6041
- LB 负载均衡的层次结构 2016-03-31 阅读 4609

- RPC 的概念模型与实现解析 2016-05-22 阅读 12889
- 最近碰到的一些 SSL 问题... 2016-02-16 阅读 2560
- Redis 集群的合纵与连横 2016-01-04 阅读 8265
- Raft 为什么是更易理解的... 2016-03-01 阅读 2571

参考知识库

Java SE知识库
23490 关注 | 468 收录

微信开发知识库
19046 关注 | 776 收录

Java EE知识库
15736 关注 | 1265 收录

Hadoop知识库
6271 关注 | 556 收录

Java 知识库
23624 关注 | 1449 收录

Apache Spark知识库
6453 关注 | 401 收录

猜你在找

微信公众平台深度开发Java版 v2.0（第一季）完整版	Java 进阶
Java Swing、JDBC开发桌面级应用	深入浅出 RPC - 浅出篇
微信公众平台深度开发(Java版)	深入浅出 RPC - 草丛中依然有几条蛇隐藏着
微信公众平台企业号开发Java版_2接收消息与响应消.	RPC深入浅出
Servlet入门到精通（备java基础，jsp、javaee、jav	深入浅出RPC框架下

查看评论

15楼 no_more_waiting 2017-02-10 11:47发表

太感谢了,对rpc有了大体的概念:
rpc就是自定义了协议和传输通道,将远程的调用伪装成本地调用,
所以效率的关键就是中间的流程管理

14楼 tianlang_2008 2016-10-27 11:17发表

楼主辛苦了，仔细读了下，对rpc也有了一个大概的概念。

13楼 -mindwind- 2016-06-13 17:20发表

> 引用“fun913510024”的评论：
>
> 楼主你好，我最近在学习dubbo这类rpc框架的时候不禁产生了一些疑问，请问下：
>
> 1、直接使用htt...

1. 相比 http 更有效率，另外 dubbo 提供了额外的增值功能，比如软负载均衡，失败转移，服务发现等。
2. 单独设计一个 API 层暴露出去

Re: HHH独一无二 2016-09-01 17:21发表

回复-mindwind-：恩恩，学习了，多谢

12楼 HHH独一无二 2016-06-13 15:51发表

楼主你好，我最近在学习dubbo这类rpc框架的时候不禁产生了一些疑问，请问下：
1、直接使用httpclient 发送请求，使用json 等形式交互可以实现接口调用，为什么要用dubbo这样的框架呢？ 或者说dubbo这中框架和直接http 有什么优劣？
2、如果使用dubbo这类框架之后，我们代码的那一层暴露出去呢？ service 层，还是dao层，还是有什么原则的去暴露服务？

11楼 lonelyrains 2016-06-03 12:44发表

复杂环境的rpc异步通信的异常处理，可以详细讲讲

10楼 刘佳翰 2015-11-17 10:03发表

学习！

9楼 唐大麦 2015-10-20 16:20发表

这和以前.NET出的WCF框架是一个道理吗？数据传输通道一般是用哪种协议实现呢？

Re: -mindwind- 2015-10-20 17:00发表

回复唐大麦：.NET 不熟，传输走 TCP 自定义协议或一些开源序列化协议都可以

8楼 pandajava 2015-09-02 11:51发表

++++++++++++++++11111111111111

7楼 牛仔very的忙 2015-07-23 21:59发表

好！！

6楼 hua504358405 2015-06-04 14:29发表

你好，请问下画的流程图用的是什么工具?

5楼 Kchiu 2015-02-10 15:42发表

大有收获 谢谢

4楼 xingfeng2510 2015-01-04 12:06发表

图挂了

3楼 qbo4_535 2014-12-19 16:51发表

图挂了，补下图吧

2楼 iter_zc 2014-09-23 18:12发表

和这篇内容一模一样啊 http://blog.csdn.net/weitao1234/article/details/39489175
到底谁是原创的？

Re: -mindwind- 2014-09-25 13:56发表

回复iter_zc：看时间

1楼 iter_zc 2014-09-22 14:16发表

写得很全面，之前看阿里的Dubbo实现，基本就是这个架构，赞。
Thrfit的代码也看了，基本组件一致，上层调用模型比较简单，没有抽象出Invoker, Exporter这些概念

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker
OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC
WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML
LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra
CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App
SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP
HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved