Atitit rpc之道 attilax著 艾龙 著

[1. 远程过程调用协议](#_Toc32446) [1](#_Toc32446)

[2. 历史 2](#_Toc28988)

[2.1. RPC的早期发展 3](#_Toc27214)

[3. RPC这种编程范式存在的三大问题以及这些问题 5](#_Toc23079)

[3.1. 过程调用通常是一种命令式操作，而命令式操作通常是一种来自底层抽象的非常快速的上下文切换操作。 5](#_Toc21346)

[3.2. 本地调用与远程调用的不同之处在于：远程调用可能会产生延迟，甚至在产生故障时可能永远也不会返回. 5](#_Toc16267)

[3.3. 异步的消息传递，或是发送某个消息并等待响应是一种更理想的模型，因为这会使消息的传递变得更加明确。 5](#_Toc7406)

[4. RPC 的实现方式有很多， 5](#_Toc25309)

[4.1. 可以基于常见的 HTTP 协议， 6](#_Toc27576)

[4.2. 也可以在TCP上层封装自定义协议，常见的 Web Service 就是基于 HTTP 协议的 RPC 6](#_Toc8803)

[4.3. Rest也是rpc的实现 6](#_Toc10012)

[5. RPC 调用分类 6](#_Toc2416)

[5.1. 从通信协议层面可以分为：基于 HTTP 协议的 RPC；基于二进制协议的 RPC；基于 TCP 协议的 RPC。 7](#_Toc4328)

[5.2. 从是否跨平台可分为：单语言 RPC，如 RMI, Remoting；跨平台 RPC，如 google protobuffer, restful json，http XML。 7](#_Toc12121)

[5.3. 从调用过程来看，可以分为同步通信RPC和异步通信RPC： 7](#_Toc13545)

[6. RPC 模块详解 7](#_Toc17212)

[7. 自定义 RPC 协议 9](#_Toc13868)

[7.1. 协议头 9](#_Toc21199)

[8. 对RPC范式的批评 10](#_Toc28416)

[9. RPC 与 微服务(MicroService) 15](#_Toc3277)

[10. REST是一种设计风格，它的很多思维方式与RPC是完全冲突的。 16](#_Toc16948)

# 远程过程调用协议

[编辑](https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/javascript:;)

[同义词](https://baike.baidu.com/subview/71844/10028254.htm" \o "同义词" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank) RPC一般指远程过程调用协议

RPC（Remote Procedure Call）—[远程过程调用](https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)，它是一种通过[网络](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。[RPC协议](https://baike.baidu.com/item/RPC%E5%8D%8F%E8%AE%AE" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)假定某些[传输协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)的存在，如TCP或UDP，为通信程序之间携带信息数据。在OSI[网络通信](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)模型中，RPC跨越了[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)和[应用层](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E5%B1%82" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)。RPC使得开发包括网络[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)多程序在内的应用程序更加容易。

RPC采用客户机/服务器模式。请求程序就是一个客户机，而服务提供程序就是一个服务器。首先，客户机调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复[信息](https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)，然后等待下一个调用信息，最后，客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。

有多种 RPC模式和执行。最初由 Sun 公司提出。IETF ONC 宪章重新修订了 Sun 版本，使得 ONC RPC 协议成为 IETF 标准协议。现在使用最普遍的模式和执行是开放式软件基础的分布式计算[环境](https://baike.baidu.com/item/%E7%8E%AF%E5%A2%83" \t "https://baike.baidu.com/item/%E8%BF%9C%E7%A8%8B%E8%BF%87%E7%A8%8B%E8%B0%83%E7%94%A8%E5%8D%8F%E8%AE%AE/_blank)（DCE）。

传统的开发模式中，我们通常将系统的各个服务部署在单台机器，随着服务的扩展，这种方式已经完全无法满足系统大规模的扩展需要，分布式系统由此诞生，在分布式系统中，最重要就是各个服务之间的 RPC 调用。

RPC 全称 Remote Procedure Call——远程过程调用，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的方式。简单一点就是：**通过一定协议和方法使得调用远程计算机上的服务，就像调用本地服务一样。**

# 历史

模型 服务器 编程 分布式 分布式系统

RPC机制的出现可以追溯到40年之前。

只是近些年来，人们对于RPC技术的质疑与批评声逐渐多了起来。尽管面临着这些尖锐的批评，但RPC的历史地位是不容置疑的，而它在现代化的应用中仍能够占据一席之地，成为分布式计算中一种重要的编程模型。Christopher Meiklejohn近来开设了一系列博客文章以讲解分布式计算中的各种编程模型与语言，在其中一篇文章中对RPC进行了详尽的回顾与展望。

远程式调用（RPC）范式的出现可以追溯到40年之前。

时至今日，它仍是在编写分布式应用时使用率较高的一种编程模型。只是近些年来，人们对于RPC技术的质疑与批评声逐渐多了起来。Steve Vinoski在2008年曾尖锐地指出，之所以RPC仍然能够得到诸多开发者的支持，其原因只有一个：舒适感！Vinoski完全不认可这种思想，他表示：

简单来说，一台机器上的程序对另一台机器上的子程序的调用就是一次RPC调用。在调用过程中，主程序不需要操心与远程执行相关的任何代码，与本地调用相比，其区别就在于需要提供远程节点的标识。最早为人所知并接受的RPC实现是由Sun提供的SunRPC机制，使用在其网络文件系统（NFS）中。

除此之外，常见的RPC机制还包括[Java](http://www.dataguru.cn/article-9246-1.html?union_site=innerlink" \t "http://it.dataguru.cn/_blank)的RMI、DCOM、XML-RPC、SOAP、CORBA，以及Google的gRPC等等。

## RPC的早期发展

RPC思想最早的原型可追溯至1974年所发布的RFC 674草案 —— “过程调用协议文档第2版”，该草案当时的目标是为因特网上的全部70个节点定义一种共享资源的通用方式，在该草案中引入了过程调用范围第2版（PCP）的概念。而在第二年发布的RFC 684草案 —— “对以过程调用作为网络协议的评论”中，首次分析了RPC这种编程范式存在的三大问题以及这些问题与[分布式系统](http://www.dataguru.cn/article-6873-1.html?union_site=innerlink" \t "http://it.dataguru.cn/_blank)的本质问题之间的关联。这三大问题可以简要地概述如下：

过程调用通常是一种命令式操作，而命令式操作通常是一种来自底层抽象的非常快速的上下文切换操作。

本地调用与远程调用的不同之处在于：远程调用可能会产生延迟，甚至在产生故障时可能永远也不会返回.

异步的消息传递，或是发送某个消息并等待响应是一种更理想的模型，因为这会使消息的传递变得更加明确。

伴随着这三大问题的是使用这种编程范式时的一系列麻烦，这些麻烦在RPC的40多年发展历史中始终阴魂不散，包括：如何从故障或错误中恢复；如何始终保证操作的正确顺序；RPC范式强制使用者进行同步编程方式；RPC的调用－响应模型使得因系统过载而导致消息无法正常处理时，对优先级的排列变得相当困难。

随后发布的RFC707草案继承了RFC 684的思想，并提出了一个新问题，即各种服务，例如TELNET与FTP之间的资源共享问题。因为这些服务各自具有不同的接口，因此使用者必须了解他们的操作端口与命令。该草案的作者提出了一个建议：为远程过程的执行定义一个通用的接口，该接口接受一个参数列表，并依然遵循RPC的调用－响应模型。虽然这一提议并未解决RPC 684中所提出的问题，但这一模型在之后依然得到了许多系统的采纳

# RPC这种编程范式存在的三大问题以及这些问题

与[分布式系统](http://www.dataguru.cn/article-6873-1.html?union_site=innerlink" \t "http://it.dataguru.cn/_blank)的本质问题之间的关联。这三大问题可以简要地概述如下：

## 过程调用通常是一种命令式操作，而命令式操作通常是一种来自底层抽象的非常快速的上下文切换操作。

## 本地调用与远程调用的不同之处在于：远程调用可能会产生延迟，甚至在产生故障时可能永远也不会返回.

## 异步的消息传递，或是发送某个消息并等待响应是一种更理想的模型，因为这会使消息的传递变得更加明确。

# RPC 的实现方式有很多，

## 可以基于常见的 HTTP 协议，

## 也可以在TCP上层封装自定义协议，常见的 Web Service 就是基于 HTTP 协议的 RPC

，HTTP 协议的优点是具有良好的跨平台性，特别适合异构系统较多的公司，但是由于 HTTP 报头较为冗长，性能较差，基于 TCP 协议的 RPC 可以建立长连接，速度和效率明显，但是难度和复杂程度很高。

RPC 的诞生让构建分布式应用更容易，极大的扩大系统的可扩展性，容错性。为复杂业务逻辑的系统进行服务化改造和高可用性升级提供了可能。

## Rest也是rpc的实现

# RPC 调用分类

RPC 调用的分类方式有很多种。

## 从通信协议层面可以分为：基于 HTTP 协议的 RPC；基于二进制协议的 RPC；基于 TCP 协议的 RPC。

## 从是否跨平台可分为：单语言 RPC，如 RMI, Remoting；跨平台 RPC，如 google protobuffer, restful json，http XML。

## 从调用过程来看，可以分为同步通信RPC和异步通信RPC：

**同步 RPC：**指的是客户端发起调用后，必须等待调用执行完成并返回结果；

**异步 RPC**：指客户方调用后不关心执行结果返回，如果客户端需要结果，可用通过提供异步 callback 回调获取返回信息。大部分 RPC 框架都同时支持这两种方式的调用

# RPC 模块详解

下面我们根据上面的RPC的架构图，对图中的各个模块进行拆解，并解释每个模块的作用。

服务端（Server)：RPC 服务的提供者，负责将 RPC 服务导出；

客户端 （Client）：RPC 服务的消费者，负责调用 RPC 服务；

代理（Proxy）：通过动态代理，提供对远程接口的代理实现；

执行器（Invoker）：对于客户端：主要负责服务调用的编码，调用请求发送和等待结果返回；对于服务方：负责处理调用逻辑并返回调用结果；

协议管理（Protocol）：协议管理组件，负责整个 RPC 通信协议的编/解码；

连接端口（Connector）：负责维持客户方和服务方的长连接通道；

后台处理（Processor）：负责整个调用服务中的管理调度，包括线程池，分发，异常处理等；

连接通道（Channel）：客户端和服务器端的数据传输通道。

具体到 JAVA 平台来说，其中的3,4通常使用动态代理实现，5,6,7,8使用 NIO 或者一些高性能 NIO 框架，如 mina,netty 实现。

最简单的 RPC JAVA 实现

在进一步拆解了组件并划分了职责之后，这里

# 自定义 RPC 协议

## **协议头**

在上面的示例程序当中，我们仅仅是完成了一个基本的远程调用，并没有实现 RPC 框架中的很多组件功能，从最简单的代码版本中我们可以发现，发起一个 RPC 调用，需要传输的最基本数据如下：

**接口方法**：包括接口的名字和相应的方法名字；

**方法参数**：包括参数的类型和取值；

**附件参数**，包括调用接口版本，接口超时时间等等。

因此，如果要自定义协议实现 RPC，我们必须再协议的消息体中包含这部分数据，另外，我们需要定义一些协议元数据，这些元数据通常放在协议头中，和包含必要参数的协议体一期组成了自定义消息。

元数据通常会包含以下字段，大部分字段只需要1-2位：

**magic**: 魔数，方便协议解码

**header\_size**: 协议头大小，便于解码，同时可用用于处理TCP粘包问题

**id :**消息 id，用来标示这次调用

**version**: 接口版本

**type:**消息类型，可用包括普通调用消息，心跳，控制消息

**status:**消息状态,是否首次处理或者已经处理

**body\_size**: 消息体长度

**serialize\_type:**消息体序列化类型

**body**：具体消息

# 对RPC范式的批评

Tanenbaum与van Renesse对RPC范式提出了尖锐的批评，他们认为将远程调用与本地调用一视同仁的思想在本质上就是错的，RPC试图打造的透明性也是根本不可能实现的。他们认为为远程访问专门设计一种协议是更好的做法。

Tanenbaum与van Renesse的批评意见涵盖了RFC 684草案中已经提到的几点内容：延迟、缺乏并行性、异常处理以及故障检测等等。此外，他们还提出了一些批评意见：

单线程服务器

如果服务器无法立即向客户端发送响应，比如它正在等待来自另一台服务器的输入。在这种情况下，不仅服务器端产生了阻塞，客户端也无法继续执行本地计算过程。

两军问题

怎样才能让两台服务器对于某个RPC的成功执行以及收到响应的结果达成一致呢？虽然某一方可以向对方发送确认信息，但对方还得向这个确认信息发送另一个确认信息以再次确认。因此无论发送几次确认都无法实现100%的一致性。这一主题其实也是一致性问题的核心，许多与分布式系统相关的文献对其进行了更深入的探讨。

两军问题

怎样才能让两台服务器对于某个RPC的成功执行以及收到响应的结果达成一致呢？虽然某一方可以向对方发送确认信息，但对方还得向这个确认信息发送另一个确认信息以再次确认。因此无论发送几次确认都无法实现100%的一致性。这一主题其实也是一致性问题的核心，许多与分布式系统相关的文献对其进行了更深入的探讨。

参数

Tanenbaum与van Renesse也叙述了参数传递与参数封送的问题，这一问题在CORBA等有可能包含引用的对象系统中显得更为严重。在这种情况下，为了保证引用的有效性，必须使用某种特定的分布式引用。

幂等性

最后一个问题是如何跨网络表达只执行一次的语义，作者在此处强调了幂等性（idempotence）的重要性。简单来说，具有幂等性的操作即使经过多次执行，其结果与只执行一次也没有区别。举例来说，HTTP中的PUT就具有幂等性的语义，而POST则不具有这一语义。作者提到了一个可能发生的场景：假设服务器在完成某个操作之后突然崩溃而来不及发送确认信息，客户端就有可能在超时之后再次发送这个实际上已经完成的请求，如果此时服务器完成了重启，就有可能再次执行这一操作。而如果该操作不满足幂等性，就可能产生一些意外的副作用。

分布式计算备忘录

Jim Waldo和Sam Kendall等人共同撰写了一篇非常有名的论文“分布式计算备忘录”，这篇论文在Reddit上被人推荐为“每个程序员都应当至少读上两篇”的论文。在这篇论文中，作者表示“忽略本地计算与分布式计算之间的区别是一种危险的思想”，特别指出了Emerald、Argus、DCOM以及CORBA的设计问题。作者将这些设计问题归纳为“三个错误的原则”：

“对于某个应用来说，无论它的部署环境如何，总有一种单一的、自然的面向对象设计可以符合其需求。”

“故障与性能问题与某个应用的组件实现直接相关，在最初的设计中无需考虑这些问题。”

“对象的接口与使用对象的上下文无关”

十年一轮回的错误

Waldo表示，每过10年，人们就会再次尝试将本地计算与远程计算的设计揉合在一起，再一次犯下相同的错误。他再次强调：本地计算与远程计算的本质是完全不同的。

延迟

最明显的区别就在于延迟问题：如果忽略了延迟问题，软件的性能就会受到直接影响。Waldo表示，“依赖于底层硬件速度的逐步提高”是错误的，一些实际的问题是很难通过测试找出的。性能分析是一个复杂的问题，在某一时刻表现良好的设计未必永远是合适的。

内存访问

Waldo对内存访问的批评是特定于CORBA与它的继任者的：对象可能会引用在同一地址空间内的指针，但一旦对象产生了移动，这些指针就会变得无效化。他认为处理这一问题的一种途径是使用分布式共享内存，但在实践上更常见的做法是使用封送或CORBA引用替换技术。

局部故障

作者在最后谈到了一个最本质的问题：局部故障。在本地计算中，故障都是可检测的。而在分布式计算中，相互独立的组件可能会产生故障，并且故障可能是局部的。

舒适感胜于正确性

在文章的开头部分曾经提到了Vinoski对于RPC的批评，他认为选择RPC的原因在于开发者的舒适感。在提出这一说法几年之后，他提出了几个非常重要的论点：

IDL的阻抗失调：对基础类型进行映射可能比较简单，但复杂的类型是非常难以映射的。

可伸缩性：RPC范式本身并没有对缓冲提供任何支持，或是提出任何缓解高延迟的机制，它仍然以一种偏命令式的操作构建分布式应用。

REST：REST本身是一种很好的思想，它为管理分布式资源的问题提出了特别的应对方式。但大多数基于REST打造的框架都改变了这一抽象思想，仍然重复了这一问题。

分布式编程语言

当我们在谈到分布式编程语言时，多数开发者所想到的其实只是如何用一般性的编程语言去构建分布式系统。实际上，只要某种语言支持并发元素，并且能够打开一个网络套接字，那么就能够构建一个分布式系统。而真正的分布式编程语言为分布式特性提供了第一等的支持。像Go这样的语言更像是一种并发语言，它为并发提供了第一等的支持。虽然并发是分布式中的一个重要部分，但他们毕竟还是不同的主题。

而Erlang则为分布式提供了第一等的支持，它虽然同样使用了RPC机制，但更倾向于在进程之间使用异步消息传递方式。受到这一设计优秀表达能力的激励，Distributed Process与Akka等框架也随之出现，以提供Erlang风格的语义能力。

# RPC 与 微服务(MicroService)

微服务是一种架构思想，一个微服务一般只完成某个特定的功能，比如下单服务，订单查询服务，是将应用分解为小的，相互连接的服务。

微服务在系统层面有多种多样的表现形式，例如暴露restful api，SOA服务或者http接口。RPC可以作为实现微服务系统的一种实现方式将各个应用都暴露出RPC的服务接口，从而实现微服务的架构。

# REST是一种设计风格，它的很多思维方式与RPC是完全冲突的。

RPC的思想是把本地函数映射到API，也就是说一个API对应的是一个function，我本地有一个getAllUsers，远程也能通过某种约定的协议来调用这个getAllUsers。至于这个协议是Socket、是HTTP还是别的什么并不重要；

RPC中的主体都是动作，是个动词，表示我要做什么。

而REST则不然，它的URL主体是资源，是个名词。而且也仅支持HTTP协议，规定了使用HTTP Method表达本次要做的动作，类型一般也不超过那四五种。这些动作表达了对资源仅有的几种转化方式。

这种设计思路是反程序员直觉的，因为在本地业务代码中仍然是一个个的函数，是动作，但表现在接口形式上则完全是资源的形式。

# RPC协议，包括PHPRPC、HPRose、JsonRPC以及Yar

## PHPRPC支持

PHPRPC 是一个轻型的、安全的、跨网际的、跨语言的、跨平台的、跨环境的、跨域的、支持复杂对象传输的、支持引用参数传递的、支持内容输出重定向的、支持分级错误处理的、支持会话的、面向服务的高性能远程过程调用协议。目前该协议的最新版本为 3.0。详细的资料可以参考phprpc官网（ <http://www.phprpc.org/zh_CN/>）

ThinkPHP提供了对PHPRpc的服务端和客户端调用的支持（客户端是跨平台跨语言的，可以支持任何语言的调用）。

## Hprose支持

**Hprose** (High Performance Remote Object Service Engine) 是一个MIT开源许可的新型轻量级跨语言跨平台的面向对象的高性能远程动态通讯中间件。它支持众多语言，例如nodeJs, C++, .NET, Java, Delphi, Objective-C, ActionScript, JavaScript, ASP, PHP, Python, Ruby, Perl 等语言，通过 Hprose 可以在这些语言之间实现方便且高效的互通。

你可以认为它是 PHPRPC 的商业版本，但是它跟 PHPRPC 完全不同，hprose 协议是全新设计的，比 PHPRPC 更加高效，实现也完全是全部从头开始的，比 PHPRPC 更加易用。更多信息可以参考（<http://www.hprose.com/>）

ThinkPHP同样也提供了对Hprose的服务端和客户端调用的支持。

服务器端的使用和PHPRPC的区别只是把控制器继承Think\Controller\HproseController类即可，其他用法基本一致，例如

## JsonRPC支持

json-rpc是基于json的跨语言远程调用协议，比xml-rpc、webservice等基于文本的协议传输数据格小；相对hessian、java-rpc等二进制协议便于调试、实现、扩展，是非常优秀的一种远程调用协议。

ThinkPHP3.2提供了对JsonRPC的服务器端和客户端调用支持，服务器端实现示例

Yar支持

Yar (yet another RPC framework) 是一个PHP扩展的RPC框架, 和现有的RPC框架(xml-rpc, soap)不同, 这是一个轻量级的框架, 支持多种打包协议(msgpack, json, php), 并且最重要的一个特点是, 它是可并行化的。

要使用Yar支持首先需要安装Yar扩展，扩展下载地址： <http://pecl.php.net/package/yar>

# 流行的RPC框架

目前流行的开源RPC框架还是比较多的。下面重点介绍三种：

1. gRPC是Google最近公布的开源软件，基于最新的HTTP2.0协议，并支持常见的众多编程语言。 我们知道HTTP2.0是基于二进制的HTTP协议升级版本，目前各大浏览器都在快马加鞭的加以支持。 这个RPC框架是基于HTTP协议实现的，底层使用到了Netty框架的支持。
2. Thrift是Facebook的一个开源项目，主要是一个跨语言的服务开发框架。它有一个代码生成器来对它所定义的IDL定义文件自动生成服务代码框架。用户只要在其之前进行二次开发就行，对于底层的RPC通讯等都是透明的。不过这个对于用户来说的话需要学习特定领域语言这个特性，还是有一定成本的。
3. Dubbo是阿里集团开源的一个极为出名的RPC框架，在很多互联网公司和企业应用中广泛使用。协议和序列化框架都可以插拔是及其鲜明的特色。同样 的远程接口是基于Java Interface，并且依托于spring框架方便开发。可以方便的打包成单一文件，独立进程运行，和现在的微服务概念一致。

偷偷告诉你集团内部已经不怎么使用dubbo啦，现在用的比较多的叫HSF，又名“好舒服”。后面有可能会开源，大家拭目以待。

# Rpc相对于rest http服务的好处

## **HTTP服务**

其实在很久以前，我对于企业开发的模式一直定性为HTTP接口开发，也就是我们常说的RESTful风格的服务接口。的确，对于在接口不多、系统与系统交互较少的情况下，解决信息孤岛初期常使用的一种通信手段；优点就是简单、直接、开发方便。利用现成的http协议进行传输。我们记得之前本科实习在公司做后台开发的时候，主要就是进行接口的开发，还要写一大份接口文档，严格地标明输入输出是什么？说清楚每一个接口的请求方法，以及请求参数需要注意的事项等。比如下面这个例子：  
POST http://www.httpexample.com/restful/buyer/info/share  
接口可能返回一个JSON字符串或者是XML文档。然后客户端再去处理这个返回的信息，从而可以比较快速地进行开发。但是对于大型企业来说，内部子系统较多、接口非常多的情况下，RPC框架的好处就显示出来了，首先就是长链接，不必每次通信都要像http一样去3次握手什么的，减少了网络开销；其次就是RPC框架一般都有注册中心，有丰富的监控管理；发布、下线接口、动态扩展等，对调用方来说是无感知、统一化的操作

# 参考资料

RPC是什么 - BruceFeng.mhtml

RPC服务和HTTP服务对比 - CSDN博客.mhtml