

07 | 架构设计：设计一个灵活的RPC框架

2020-03-04 何小锋

RPC实战与核心原理

[进入课程 >](#)



讲述：张浩

时长 10:08 大小 9.29M



你好，我是何小锋。到今天为止，基础篇的知识我们就全部学习完了，接下来我们进入进阶篇。

在基础篇里面，我们讲了 RPC 的通信原理以及 RPC 里各个功能组件的作用，不妨用一段话再次回顾下：“其实 RPC 就是把拦截到的方法参数，转成可以在网络中传输的二进制，并保证在服务提供方能正确地还原出语义，最终实现像调用本地一样地调用远程的目的。” **你记住了吗？**



那学到这儿，距离实现一个灵活的 RPC 框架其实还是有距离的。知道了各个功能组件只是迈出了第一步，接下来你必须要清楚各个组件之间是怎么完成数据交互的，这也是今天这讲的重点，我们一起搞清楚 RPC 的架构设计。

RPC 架构

说起架构设计，我相信你一定不陌生。我理解的架构设计呢，就是从顶层角度出发，厘清各模块组件之间数据交互的流程，让我们对系统有一个整体的宏观认识。我们先看看 RPC 里面都有哪些功能模块。

我们讲过，RPC 本质上就是一个远程调用，那肯定就需要通过网络来传输数据。虽然传输协议可以有多种选择，但考虑到可靠性的话，我们一般默认采用 TCP 协议。为了屏蔽网络传输的复杂性，我们需要封装一个单独的数据传输模块用来收发二进制数据，这个单独模块我们可以叫做传输模块。

用户请求的时候是基于方法调用，方法出入参数都是对象数据，对象是肯定没法直接在网络中传输的，我们需要提前把它转成可传输的二进制，这就是我们说的序列化过程。但只是把方法调用参数的二进制数据传输到服务提供方是不够的，我们需要在方法调用参数的二进制数据后面增加“断句”符号来分隔出不同的请求，在两个“断句”符号中间放的内容就是我们请求的二进制数据，这个过程我们叫做协议封装。

虽然这是两个不同的过程，但其目的都是一样的，都是为了保证数据在网络中可以正确传输。这里我说的正确，可不仅指数据能够传输，还需要保证传输后能正确还原出传输前的语义。所以我们可以把这两个处理过程放在架构中的同一个模块，统称为协议模块。

除此之外，我们还可以在协议模块中加入压缩功能，这是因为压缩过程也是对传输的二进制数据进行操作。在实际的网络传输过程中，我们的请求数据包在数据链路层可能会因为太大而被拆分成多个数据包进行传输，为了减少被拆分的次数，从而导致整个传输过程时间太长的的问题，我们可以在 RPC 调用的时候这样操作：在方法调用参数或者返回值的二进制数据大于某个阈值的情况下，我们可以通过压缩框架进行无损压缩，然后在另外一端也用同样的压缩算法进行解压，保证数据可还原。

传输和协议这两个模块是 RPC 里面最基础的功能，它们使对象可以正确地传输到服务提供方。但距离 RPC 的目标——实现像调用本地一样地调用远程，还缺少点东西。因为这两个模块所提供的都是一些基础能力，要让这两个模块同时工作的话，我们需要手写一些黏合的代码，但这些代码对我们使用 RPC 的研发人员来说是没有意义的，而且属于一个重复的工作，会导致使用过程的体验非常不友好。

这就需要我们在 RPC 里面把这些细节对研发人员进行屏蔽，让他们感觉不到本地调用和远程调用的区别。假设有用到 Spring 的话，我们希望 RPC 能让我们把一个 RPC 接口定义成一个 Spring Bean，并且这个 Bean 也会统一被 Spring Bean Factory 管理，可以在项目中通过 Spring 依赖注入到方式引用。这是 RPC 调用的入口，我们一般叫做 Bootstrap 模块。

学到这儿，一个点对点 (Point to Point) 版本的 RPC 框架就完成了。我一般称这种模式的 RPC 框架为单机版本，因为它没有集群能力。所谓集群能力，就是针对同一个接口有着多个服务提供者，但这多个服务提供者对于我们的调用方来说是透明的，所以在 RPC 里面我们还需要给调用方找到所有的服务提供方，并需要在 RPC 里面维护好接口跟服务提供者地址的关系，这样调用方在发起请求的时候才能快速找到对应的接收地址，这就是我们常说的“服务发现”。

但服务发现只是解决了接口和服务提供方地址映射关系的查找问题，这更多是一种“静态数据”。说它是静态数据是因为，对于我们的 RPC 来说，我们每次发送请求的时候都是需要用 TCP 连接的，相对服务提供方 IP 地址，TCP 连接状态是瞬息万变的，所以我们的 RPC 框架里面要有连接管理器去维护 TCP 连接的状态。

有了集群之后，提供方可能就需要管理好这些服务了，那我们的 RPC 就需要内置一些服务治理的功能，比如服务提供方权重的设置、调用授权等一些常规治理手段。而服务调用方需要额外做些什么事情呢？每次调用前，我们都需要根据服务提供方设置的规则，从集群中选择可用的连接用于发送请求。

那到这儿，一个比较完善的 RPC 框架基本就完成了，功能也差不多就是这些了。按照分层设计的原则，我将这些功能模块分为了四层，具体内容见图示：



架构图

可扩展的架构

那 RPC 架构设计出来就完事了吗？当然不，技术迭代谁都躲不过。

不知道你有没有这样的经历，你设计的一个系统它看上去很完善，也能很好地运行，然后你成功地把它交付给了业务方。有一天业务方有了新的需求，要加入很多新的功能，这时候你就会发现当前架构面临的可就是大挑战了，要修改很多地方才能实现。

举个例子，假如你设计了一个商品发布系统，早些年我们只能在网上购买电脑、衣服等实物商品，但现在发展成可以在网上购买电话充值卡、游戏点卡等虚拟商品，实物商品的发布流程是需要选择购买区域的，但虚拟商品并没有这一限制。如果你想要在一套发布系统里面同时完成实物和虚拟商品发布的话，你就只能在代码里面加入很多的 if else 判断逻辑，这样是能行，可整个代码就臃肿、杂乱了，后期也极难维护。

其实，我们设计 RPC 框架也是一样的，我们不可能在开始时就面面俱到。那有没有更好的方式来解决这些问题呢？这就是我们接下来要讲的插件化架构。

在 RPC 框架里面，我们是怎么支持插件化架构的呢？我们可以将每个功能点抽象成一个接口，将这个接口作为插件的契约，然后把这个功能的接口与功能的实现分离，并提供接口的默认实现。在 Java 里面，JDK 有自带的 SPI（Service Provider Interface）服务发现机制，它可以动态地为某个接口寻找服务实现。使用 SPI 机制需要在 Classpath 下的 META-INF/services 目录里创建一个以服务接口命名的文件，这个文件里的内容就是这个接口的具体实现类。

但在实际项目中，我们其实很少使用到 JDK 自带的 SPI 机制，首先它不能按需加载，ServiceLoader 加载某个接口实现类的时候，会遍历全部获取，也就是接口的实现类得全部载入并实例化一遍，会造成不必要的浪费。另外就是扩展如果依赖其它的扩展，那就做不到自动注入和装配，这就很难和其他框架集成，比如扩展里面依赖了一个 Spring Bean，原生的 Java SPI 就不支持。

加上了插件功能之后，我们的 RPC 框架就包含了两大核心体系——核心功能体系与插件体系，如下图所示：



插件化RPC

这时，整个架构就变成了一个微内核架构，我们将每个功能点抽象成一个接口，将这个接口作为插件的契约，然后把这个功能的接口与功能的实现分离并提供接口的默认实现。这样的架构相比之前的架构，有很多优势。首先它的可扩展性很好，实现了开闭原则，用户可以非常方便地通过插件扩展实现自己的功能，而且不需要修改核心功能的本身；其次就是保持了核心包的精简，依赖外部包少，这样可以有效减少开发人员引入 RPC 导致的包版本冲突问题。

总结

我们都知道软件开发的过程很复杂，不仅是因为业务需求经常变化，更难的是在开发过程中要保证团队成员的目标统一。我们需要用一种可沟通的话语、可“触摸”的愿景达成目标，我认为这就是软件架构设计的意义。

但仅从功能角度设计出的软件架构并不够健壮，系统不仅要能正确地运行，还要以最低的成本进行可持续的维护，因此我们十分有必要关注系统的可扩展性。只有这样，才能满足业务变化的需求，让系统的生命力不断延伸。

课后思考

你能分享一下，在日常工作中，你都有哪些地方是用到了插件思想来解决扩展性问题的吗？

欢迎留言和我分享你的思考，也欢迎你把文章分享给你的朋友，邀请他加入学习。我们下节课再见！

更多课程推荐

RPC 实战与核心原理

高效解决分布式系统的通信难题

何小锋

京东技术架构部首席架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 06 | RPC实战：剖析gRPC源码，动手实现一个完整的RPC

下一篇 08 | 服务发现：到底是要CP还是AP？

精选留言 (6)

 写留言



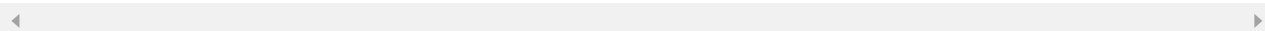
cricket1981

2020-03-04

如何实现按需加载的SPI？

展开 ∨

作者回复：可以参考springboot的Condition实现



 1

 2



凹凸

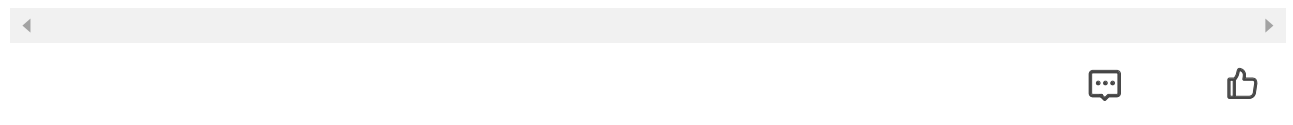
2020-03-08

插件功能目前在我的产品中任务引擎也是通过spi和spring注入两种方式，确实做不到按需

加载，这个可以优化下。

展开 ▾

作者回复: 是的，可以考虑自己实现一个



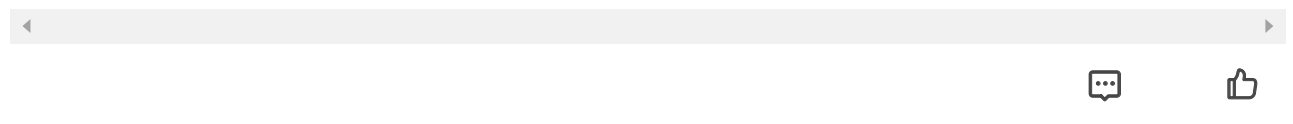
奔奔奔跑

2020-03-06

太好了，这章内容！请问老师我可以对照这张内容和RPC微内核架构图去阅读gRPC的源码吗？我想从源码中学习收获一定非常大！

展开 ▾

编辑回复: 你可以看下第6讲



楼下小黑哥

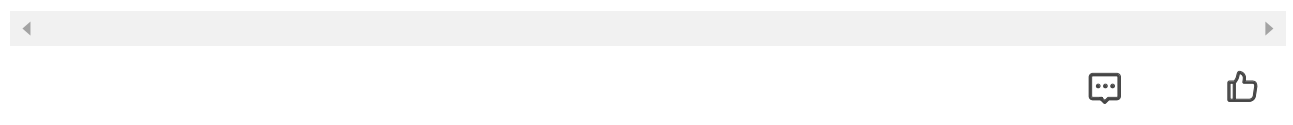
2020-03-06

我们系统中有个路由模块，可以分发交易到指定交易渠道。新接入的模块，只需要接入规定的接口，就能被路由模块识别到。

我觉得这也是类似于插件化开发，每次新接入交易渠道，路由模块都是无需重启。

展开 ▾

作者回复: 很棒！

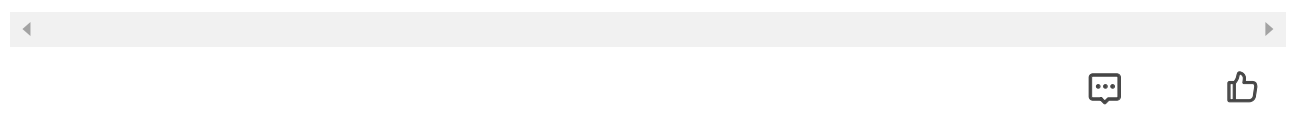


翌

2020-03-05

看到这个问题首先想到的是用过的一些框架工具的中间件结构，这些中间件应该也是插件的一种，比如django的middleware和ror的各种gem包，都可以给框架提供强大的扩展功能。

作者回复: 很棒





艾斯曼

2020-03-04

目前工作是为工业设备联网数据采集，设备种类和型号繁多，产品中通过抽象出一套“驱动”的概念，把每类设备当作一个插件开发，整体产品架构不变，感觉有点这个概念。只是产品还不够大，其他插件体系还不够明确。

展开 ∨

作者回复: 很棒

