答疑课堂 | 基础篇与进阶篇思考题答案合集

2020-03-27 何小锋

RPC实战与核心原理 进入课程 >

讲述：张浩

时长 08:43 大小 8.00M

你好，我是何小锋。到今天为止，基础篇和进阶篇我们就都学习完了，在这个过程中我一直在看大家的留言，知道你可能还有很多地方存在着疑问，今天这一讲我整理了一些关注度比较高的课后思考题答案，希望能给你带来帮助。

第二讲

思考题：在 RPC 里面，我们是怎么实现请求跟响应关联的？

首先我们要弄清楚为什么要把请求与响应关联。这是因为在 RPC 调用过程中，调用端会向服务端发送请求消息，之后它还会收到服务端发送回来的响应消息，但这两个操作并不是同步进行的。在高并发的情况下，调用端可能会在某一时刻向服务端连续发送很多条消息之

后，才会陆续收到服务端发送回来的各个响应消息，这时调用端需要一种手段来区分这些响应消息分别对应的是之前的哪条请求消息，所以我们说 RPC 在发送消息时要请求跟响应关联。

解决这个问题不难，只要调用端在收到响应消息之后，从响应消息中读取到一个标识，告诉调用端，这是哪条请求消息的响应消息就可以了。在这一讲中，你会发现我们设计的私有协议都会有消息 ID，这个消息 ID 的作用就是起到请求跟响应关联的作用。调用端为每一个消息生成一个唯一的消息 ID，它收到服务端发送回来的响应消息如果是同一消息 ID，那么调用端就可以认为，这条响应消息是之前那条请求消息的响应消息。

第五讲

思考题：如果没有动态代理帮我们完成方法调用拦截，用户该怎么完成 RPC 调用？

这个问题我们可以参考下 gRPC 框架。gRPC 框架中就没有使用动态代理，它是通过代码生成的方式生成 Service 存根，当然这个 Service 存根起到的作用和 RPC 框架中的动态代理是一样的。

gRPC 框架用代码生成的 Service 存根来代替动态代理主要是为了实现多语言的客户端，因为有些语言是不支持动态代理的，比如 C++、go 等，但缺点也是显而易见的。如果你使用过 gRPC，你会发现这种代码生成 Service 存根的方式与动态代理相比还是很麻烦的，并不如动态代理的方式使用起来方便、透明。

第六讲

思考题：在 gRPC 调用的时候，我们有一个关键步骤就是把对象转成可传输的二进制，但是在 gRPC 里面，我们并没有直接转成二进制数组，而是返回一个 InputStream，你知道这样做的好处是什么吗？

RPC 调用在底层传输过程中也是需要使用 Stream 的，直接返回一个 InputStream 而不是二进制数组，可以避免数据的拷贝。

第八讲

思考题：目前服务提供者上线后会自动注册到注册中心，服务调用方会自动感知到新增的实例，并且流量会很快打到该新增的实例。如果我想把某些服务提供者实例的流量切走，除了

下线实例，你有没有想到其它更便捷的办法呢？

解决这个问题的方法还是有很多的，比如留言中提到的改变服务提供者实例的权重，将权重调整为 0，或者通过路由的方式也可以。

但解决这个问题最便捷的方式还是使用动态分组，在[第 16 讲]中我讲解了业务分组的概念，通过业务分组来实现流量隔离。如果业务分组是动态的，我们就可以在管理平台动态地自由调整，那是不是就可以实现动态地流量切换了呢？这个问题我们还会在高级篇中详解，期待一下。

第十二讲

思考题：在整个 RPC 调用的流程中，异常重试发生在哪个环节？

在回答这个问题之前，我们先回想下这一讲中讲过的内容。我在讲 RPC 为什么需要异常重试时我说过，如果在发出请求时恰好网络出现问题了，导致我们的请求失败，我们可能需要进行异常重试。从这一点我们可以看出，异常重试的操作是要在调用端进行的。因为如果在调用端发出请求时恰好网络出现问题导致请求失败，那么这个请求很可能还没到达服务端，服务端当然就没办法去处理重试了。

另外，我还讲过，我们需要在所有发起重试、负载均衡选择节点的时候，去掉重试之前出现过问题的那个节点，以保证重试的成功率。由此可见异常重试的操作应该发生在负载均衡之前，在发起重试的时候，会调用负载均衡插件来选择一个服务节点，在调用负载均衡插件时我们要告诉负载均衡需要刨除哪些有问题的服务节点。

在整个 RPC 调用的过程中，从动态代理到负载均衡之间还有一系列的操作，如果你研究过开源的 RPC 框架，你会发现在调用端发送请求消息之前还会经过过滤链，对请求消息进行层层的过滤处理，之后才会通过负载均衡选择服务节点，发送请求消息，而异常重试操作就发生在过滤链处理之后，调用负载均衡选择服务节点之前，这样的重试是可以减少很多重复操作的。

第十四讲

思考题：在启动预热那部分，我们特意提到过一个问题，就是“当大批量重启服务提供方的时候，会导致请求大概率发到没有重启的机器上，这时服务提供方有可能扛不住”，不知道

你是怎么看待这个问题的，是否有好的解决方案呢？

我们可以考虑在非流量高峰的时候重启服务，将影响降到最低；也可以考虑分批次重启，控制好每批重启的服务节点的数量，当一批服务节点的权重与访问量都到正常水平时，再去重启下一批服务节点。

第十五讲

思考题：在使用 RPC 的过程中业务要实现自我保护，针对这个问题你是否还有其他的解决方案？

通过这一讲我们知道，在 RPC 调用中无论服务端还是调用端都需要自我保护，服务端自我保护的最简单有效的方式是“限流”，调用端则可以通过“熔断”机制来进行自我保护。

除了“熔断”和“限流”外，相信你一定听过“降级”这个词。简单来说就是当一个服务处理大量的请求达到一定压力的时候，我们可以让这个服务在处理请求时减少些非必要的功能，从而降低这个服务的压力。

还有就是我们可以通过服务治理，降低一个服务节点的权重来减轻某一方服务节点的请求压力，达到保护这个服务节点的目的。

第十六讲

思考题：在我们的实际工作中，测试人员和开发人员的工作一般都是并行的，这就导致一个问题经常出现：开发人员在开发过程中可能需要启动自身的应用，而测试人员为了能验证功能，会在测试环境中部署同样的应用。如果开发人员和测试人员用的接口分组名刚好一样，在这种情况下，就可能会干扰其它正在联调的调用方进行功能验证，进而影响整体的工作效率。不知道面对这种情况，你有什么好办法吗？

我们可以考虑配置不同的注册中心，开发人员将自己的服务注册到注册中心 A 上，而测试人员可以将自己的服务注册到测试专属的注册中心 B 上，这样测试人员在验证功能的时候，调用端会从注册中心 B 上拉取服务节点，开发人员重启自己的服务是影响不到测试人员的。

如果你使用过或者了解 k8s 的话，你一定知道“命名空间”的概念，RPC 框架如果支持“命名空间”，也是可以解决这一问题的。

今天的答疑就到这里，如果你有更多问题，欢迎继续在留言区中告知，我们共同讨论。下节课再见！

上一篇　16 | 业务分组：如何隔离流量?

下一篇　17 | 异步RPC：压榨单机吞吐量

精选留言 (2)

写留言

安排

2020-03-27

限流一般是在服务端做还是在调用端做呢?

展开

作者回复: 都能做，但最好在服务端，因为限流是对服务端的保护，如果在调用端做，不排除调用端使用不当的情况。

3 1

山顶的洞
2020-03-28

读老师的文章是一种享受

展开