如何自己实现一个 RPC 框架?

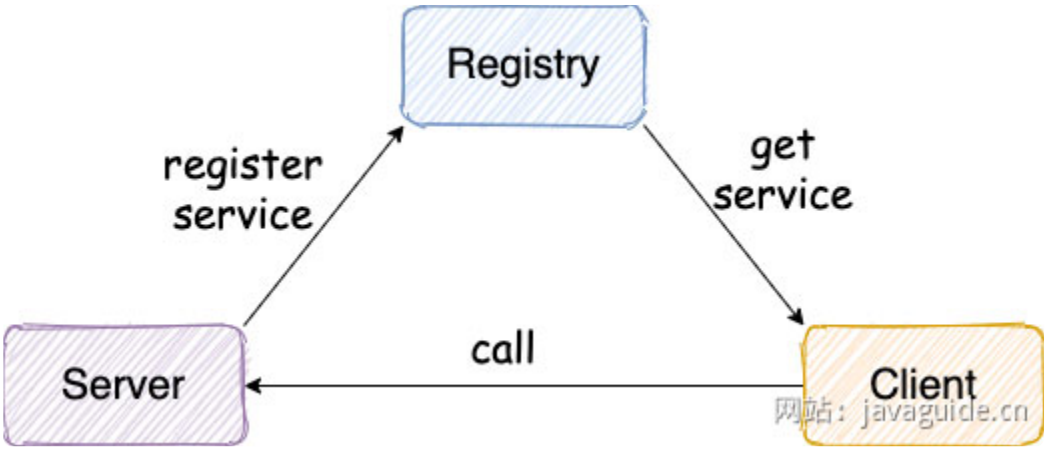
像设计一个 RPC 框架/消息队列这类问题在面试中还是非常常见的。这是一道你花点精力稍微准备一下就能回答上来的一个问题。如果你回答的比较好的话，那面试官肯定会对你印象非常不错!

消息队列的设计实际上和 RPC 框架/非常类似，我这里就先拿 RPC 框架开涮。

如果让你自己设计 RPC 框架你会如何设计?

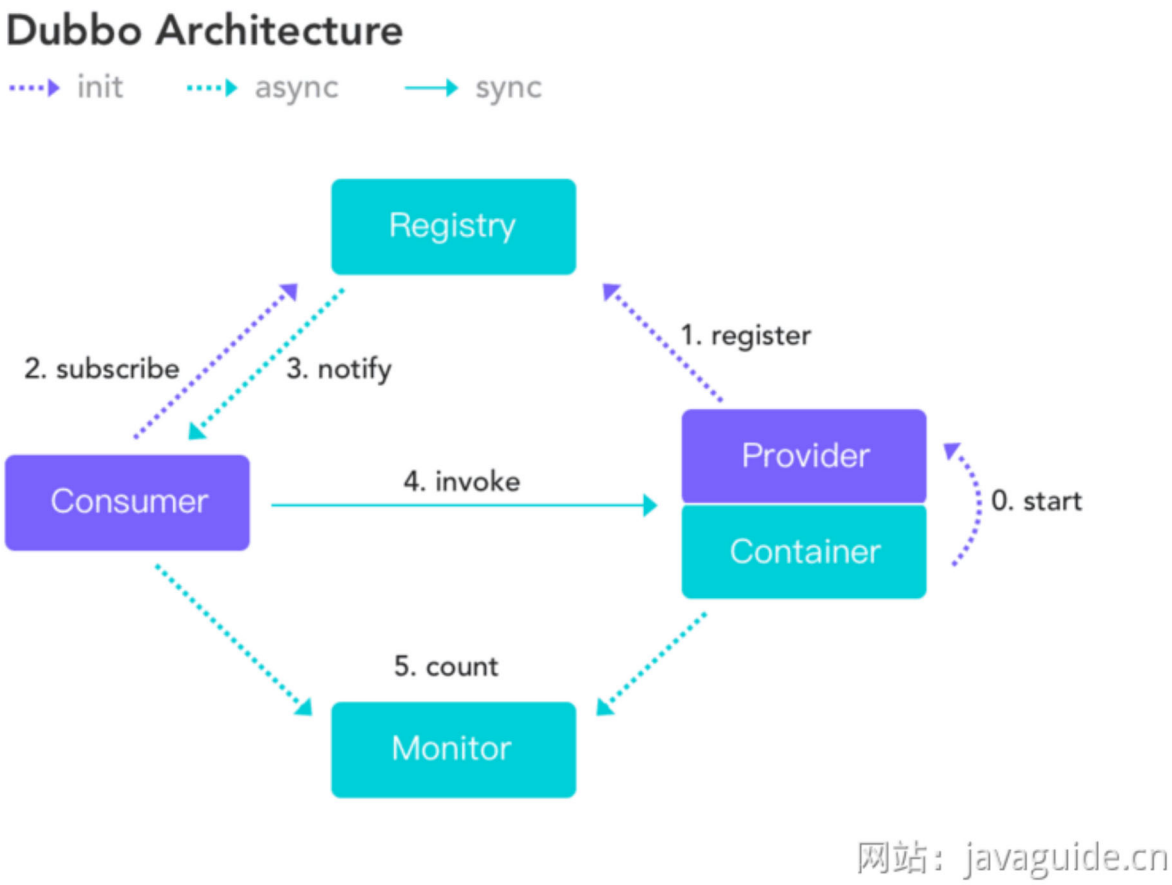
一般情况下， RPC 框架不仅要提供服务发现功能，还要提供负载均衡、容错等功能，这样的 RPC 框架才算真正合格的。

为了便于小伙伴们理解，我们先从一个最简单的 RPC 框架使用示意图开始。这也是 guide-rpc-framework <https://github.com/Snailclimb/guide-rpc-framework> 目前的架构 。

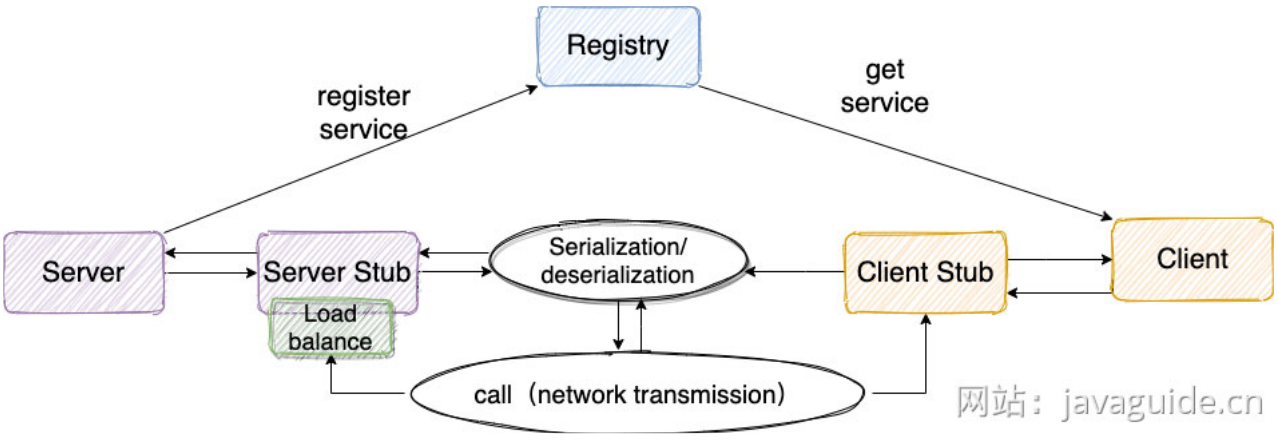


从上图我们可以看出：**服务提供端 Server 向注册中心注册服务，服务消费者 Client 通过注册中心拿到服务相关信息，然后再通过网络请求服务提供端 Server。**

作为 RPC 框架领域的佼佼者Dubbo <https://github.com/apache/dubbo> 的架构如下图所示,和我们上面画的大体也是差不多的。



下面我们再来看一个比较完整的 RPC 框架使用示意图如下：



参考上面这张图，我们简单说一下设计一个最基本的 RPC 框架的思路或者说实现一个最基本的 RPC 框架需要哪些东西：

注册中心

注册中心首先是要有的。比较推荐使用 Zookeeper 作为注册中心。

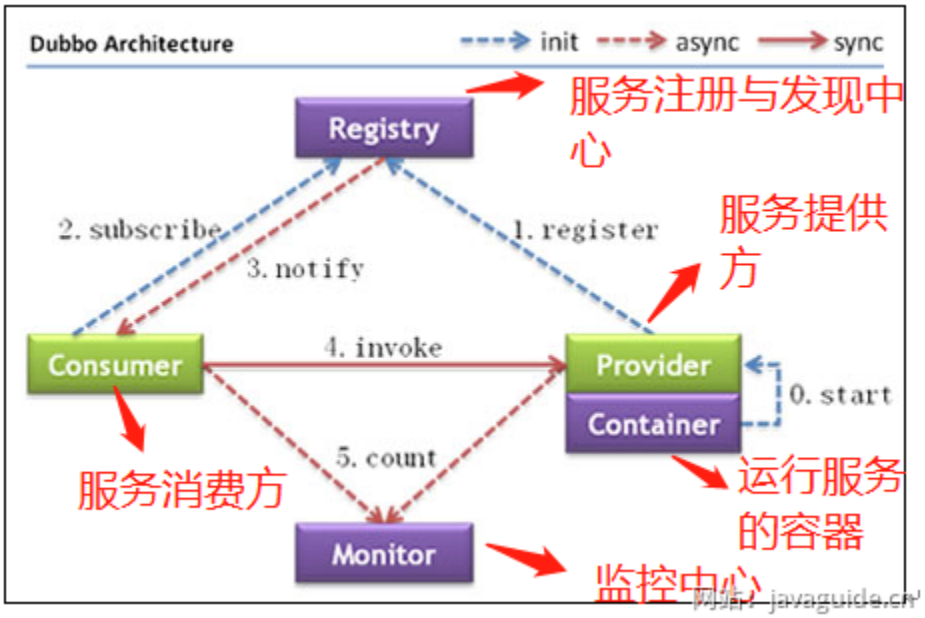
ZooKeeper 为我们提供了高可用、高性能、稳定的分布式数据一致性解决方案，通常被用于实现诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master 选举、分布式锁和分布式队列等功能。并且，ZooKeeper 将数据保存在内存中，性能是非常棒的。 在“读”多于“写”的应用程序中尤其地高性能，因为“写”会导致所有的服务器间同步状态。（“读”多于“写”是协调服务的典型场景）。

关于 ZooKeeper 的更多介绍可以看我总结的这篇文章：《ZooKeeper 相关概念总结》<https://javaguide.cn/distributed-system/distributed-process-coordination/zookeeper/zookeeper-intro.html>

当然了，如果你想通过文件来存储服务地址的话也是没问题的，不过性能会比较差。

注册中心负责服务地址的注册与查找，相当于目录服务。 服务端启动的时候将服务名称及其对应的地址(ip+port)注册到注册中心，服务消费端根据服务名称找到对应的服务地址。有了服务地址之后，服务消费端就可以通过网络请求服务端了。

我们再来结合 Dubbo 的架构图来理解一下！



上述节点简单说明：

- **Provider：** 暴露服务的服务提供方
- **Consumer：** 调用远程服务的服务消费方
- **Registry：** 服务注册与发现的注册中心
- **Monitor：** 统计服务的调用次数和调用时间的监控中心
- **Container：** 服务运行容器

调用关系说明：

1. 服务容器负责启动，加载，运行服务提供者。
2. 服务提供者在启动时，向注册中心注册自己提供的服务。
3. 服务消费者在启动时，向注册中心订阅自己所需的服务。
4. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
5. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
6. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

网络传输

既然我们要调用远程的方法，就要发送网络请求来传递目标类和方法的信息以及方法的参数等数据到服务提供端。

网络传输具体实现你可以使用 **Socket** （Java 中最原始、最基础的网络通信方式。但是，Socket 是阻塞 IO、性能低并且功能单一）。

你也可以使用同步非阻塞的 I/O 模型 **NIO**，但是用它来进行网络编程真的太麻烦了。不过没关系，你可以使用基于 NIO 的网络编程框架 Netty，它将是你最好的选择！

我先简单介绍一下 Netty，后面的文章中我会详细介绍到。

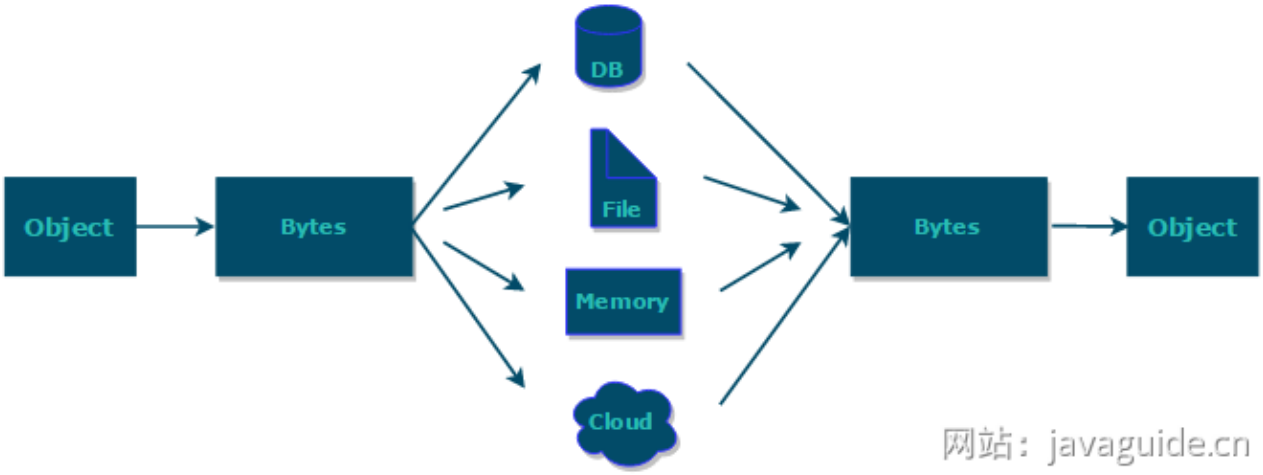
1. **Netty 是一个基于 NIO 的 client-server(客户端服务器)框架，使用它可以快速简单地开发网络应用程序。**
2. 它极大地简化并简化了 TCP 和 UDP 套接字服务器等网络编程,并且性能以及安全性等很多方面甚至都要更好。
3. 支持多种协议如 FTP，SMTP，HTTP 以及各种二进制和基于文本的传统协议。

序列化和反序列化

要在网络传输数据就要涉及到**序列化。为什么需要序列化和反序列化呢?**

因为网络传输的数据必须是二进制的。因此，我们的 Java 对象没办法直接在网络中传输。为了能够让 Java 对象在网络中传输我们需要将其**序列化**为二进制的数据。我们最终需要的还是目标 Java 对象，因此我们还要将二进制的数据“解析”为目标 Java 对象，也就是对二进制数据再进行一次**反序列化**。

另外，不仅网络传输的时候需要用到序列化和反序列化，将对象存储到文件、数据库等场景都需要用到序列化和反序列化。



JDK 自带的序列化，只需实现 `java.io.Serializable` 接口即可，不过这种方式不推荐，因为不支持跨语言调用并且性能比较差。

现在比较常用序列化的有 **hessian**、**kyro**、**protostuff** ……。我会在下一篇文章中简单对比一下这些序列化方式。

动态代理

动态代理也是需要的。很多人可能不清楚为啥需要动态代理？我来简单解释一下吧！

我们知道代理模式就是： 我们给某一个对象提供一个代理对象，并由代理对象来代替真实对象做一些事情。你可以把代理对象理解为一个幕后的工具人。 举个例子：我们真实对象调用方法的时候，我们可以通过代理对象去做一些事情比如安全校验、日志打印等等。但是，这个过程是完全对真实对象屏蔽的。

讲完了代理模式，再来说动态代理在 RPC 框架中的作用。

前面第一节的时候，我们就已经提到： **RPC 的主要目的就是让我们调用远程方法像调用本地方法一样简单，我们不需要关心远程方法调用的细节比如网络传输。**

怎样才能屏蔽程方法调用的底层细节呢?

答案就是**动态代理**。简单来说，当你调用远程方法的时候，实际会通过代理对象来传输网络请求，不然的话，怎么可能直接就调用到远程方法。

相关文章： 代理模式详解：静态代理+JDK/CGLIB 动态代理实战 <https://javaguide.cn/java/basis/proxy.html>

负载均衡

负载均衡也是需要的。为啥?

举个例子：我们的系统中的某个服务的访问量特别大，我们将这个服务部署在了多台服务器上，当客户端发起请求的时候，多台服务器都可以处理这个请求。那么，如何正确选择处理该请求的服务器就很关键。假如，你就要一台服务器来处理该服务的请求，那该服务部署在多台服务器的意义就不复存在了。负载均衡就是为了避免单个服务器响应同一请求，容易造成服务器宕机、崩溃等问题，我们从负载均衡的这四个字就能明显感受到它的意义。

传输协议

我们还需要设计一个私有的 RPC 协议，这个协议是客户端（服务消费方）和服务端（服务提供方）交流的基础。

简单来说：**通过设计协议，我们定义需要传输哪些类型的数据， 并且还会规定每一种类型的数据应该占多少字节。这样我们在接收到二进制数据之后，就可以正确的解析出我们需要的数据。**这有一点像密文传输的感觉。

通常一些标准的 RPC 协议包含下面这些内容：

- **魔数**： 通常是 4 个字节。这个魔数主要是为了筛选来到服务端的数据包，有了这个魔数之后，服务端首先取出前面四个字节进行比对，能够在第一时间识别出这个数据

包并非是遵循自定义协议的，也就是无效数据包，为了安全考虑可以直接关闭连接以节省资源。

- **序列化器编号**： 标识序列化的方式，比如是使用 Java 自带的序列化，还是 json，kyro 等序列化方式。
- **消息体长度**： 运行时计算出来。
- ……

如果你想看 guide-rpc-framework <https://github.com/Snailclimb/guide-rpc-framework> 的 RPC 协议设计的话，可以在 Netty 编解码器相关的类中找到。

实现一个最基本的 RPC 框架需要哪些技术?

刚刚我们已经聊了如何实现一个 RPC 框架，下面我们就来看看实现一个最基本的 RPC 框架需要哪些技术吧!

按照我实现的这一款基于 Netty+Kyro+Zookeeper 实现的 RPC 框架来说的话，你需要下面这些技术支撑:

Java

1. 动态代理机制;
2. 序列化机制以及各种序列化框架的对比，比如 hession2、kyro、protostuff;
3. 线程池的使用;
4. `CompletableFuture` 的使用;
5. ……

Netty

1. 使用 Netty 进行网络传输;
2. `ByteBuf` 介绍;
3. Netty 粘包拆包;
4. Netty 长连接和心跳机制;
5. ……

Zookeeper

1. 基本概念;
2. 数据结构;
3. 如何使用 Netflix 公司开源的 zookeeper 客户端框架 Curator 进行增删改查;
4. ……

总结

实现一个最基本的 RPC 框架应该至少包括下面几部分:

1. **注册中心**： 注册中心负责服务地址的注册与查找，相当于目录服务。
2. **网络传输**： 既然我们要调用远程的方法，就要发送网络请求来传递目标类和方法的信息以及方法的参数等数据到服务提供端。
3. **序列化和反序列化**： 要在网络传输数据就要涉及到**序列化**。
4. **动态代理**： 屏蔽程方法调用的底层细节。
5. **负载均衡**： 避免单个服务器响应同一请求，容易造成服务器宕机、崩溃等问题。
6. **传输协议**： 这个协议是客户端（服务消费方）和服务端（服务提供方）交流的基础。

更完善的一点的 RPC 框架可能还有监控模块（拓展：你可以研究一下 Dubbo 的监控模块的设计）。

db1c94b9fd04.png&title=%E5%A6%82%E4%BD%95%E8%87%AA%E5%B7%B1%E5%AE%9E%E7