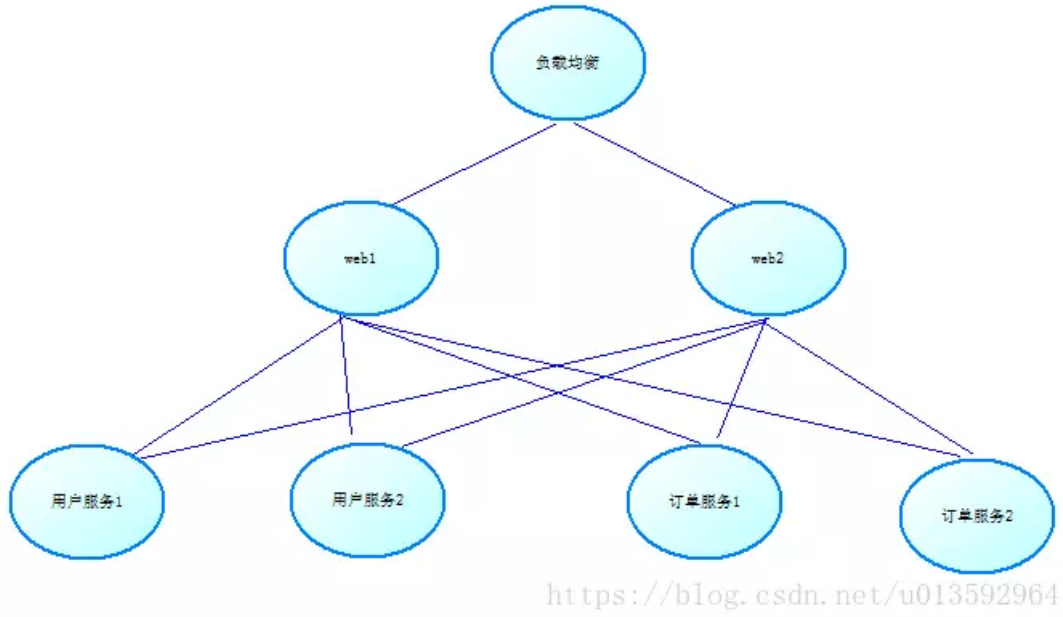
牛逼哄哄的 RPC 框架，底层到底什么原理?

_Yasin Java技术栈 11月2日

1. RPC框架的概念

RPC（Remote Procedure Call）–远程过程调用，通过网络通信调用不同的服务，共同支撑一个软件系统，微服务实现的基石技术。

使用RPC可以解耦系统，方便维护，同时增加系统处理请求的能力。



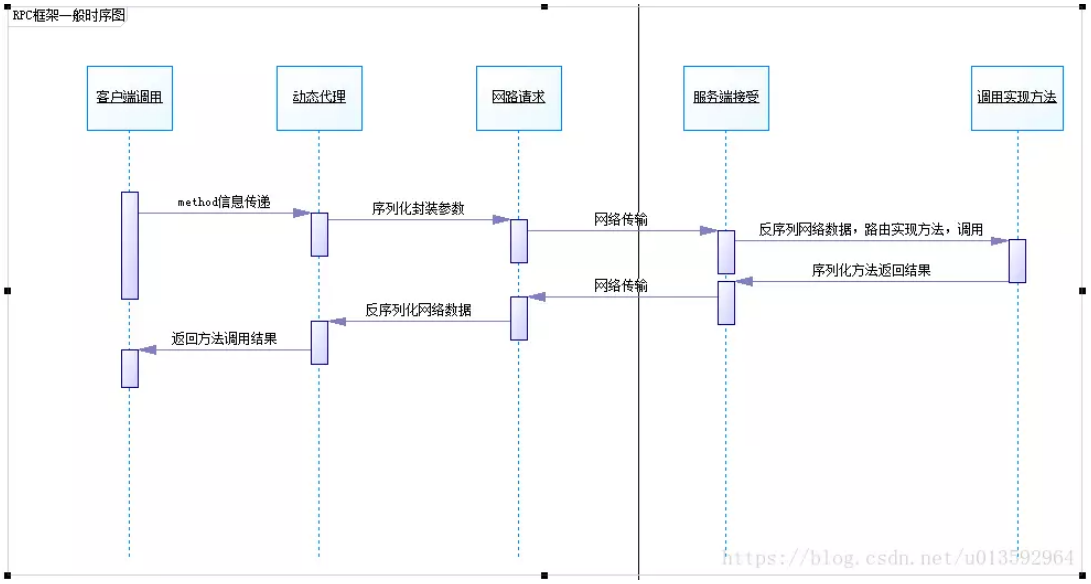
 上面是一个简单的软件系统结构，我们拆分出来用户系统和订单系统做为服务存在，让不同的站点去调用。

只需要引入各个服务的接口包，在代码中调用RPC服务就跟调用本地方法一样，我刚接触到这种调用方式的时候颇为惊奇，我明明调用的就是java语言方法啊（已java为例，现在RPC框架一般都支持多语言），怎么就调用了远程的服务了呢？？

2. RPC框架的原理解析

最近自己写了一个简单的RPC框架KRPC，本文原理分析结合中代码，均为该框架源码，RPC与RMI的区别看这篇文章《Java RMI 和 RPC 的区别》。

2.1 流程纵览



如上图所示，我将一个RPC调用流程概括为上图中5个流程，左边3个为客户端流程，右边两个为服务端流程。

下面就各流程进行解析

2.2 客户端调用

服务调用方在调用服务时，一般进行相关初始化，通过配置文件/配置中心 获取服务端地址用户调用。

```
// 用户服务接口
public interface UserService {
  public User genericUser(Integer id,String name,Long phone);
}

//调用方
//服务初始化
KRPC.init("D:\\krpc\\service\\demo\\conf\\client.xml");
UserService service = ProxyFactory.create(UserService.class, "demo","demoService");
User user = service.genericUser(1, "yasin", 1888888888L);
```

一开始接触RPC调用方法肯定就有疑惑，它不是一个接口吗，直接调用应该没啥效果啊，我也没有引入实现包。

带着这个疑惑，我们就进入下一个知识点，动态代理。

2.3 动态代理

动态代理这东西意如其名，它代理你帮你做事情，动态代理看这篇文章《详解 Java 中的三种代理模式》。

上面我们不说道直接调用一个接口中的方法，并且没有用该接口的实现类调用，那么方法是怎么生效的呢?

可以看到这个用户服务这个service是由ProxyFactory代理工程创造的，在该ProxyFactory#create()方法中就跟一个代理处理器绑定在一起了。

```
@Override
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

    //构造请求request
    Request request = new Request();
    ....

    return RequestHandler.request(serviceName, request,returnClass);
}
```

这个类实现了InvocationHandler接口（JDK提供的动态代理技术），每次去调用接口方法，最终都交由该handler进行处理。

这个环节一般会获取方法的一些信息，例如方法名，方法参数类型，方法参数值，返回对象类型。

同时这个环节会提供序列化功能，一般的RPC网络传输使用TCP（哪怕使用HTTP）传输，这里也要将这些参数进行封装成我们定义的数据接口进行传输。

2.4 网络传输

我们通过将方法参数进行处理后，就要使用发起网络请求，使用tcp传输的就利用socket通信进行传输，这一块我开源项目中使用的同步堵塞的方案进行请求，也可以使用一些非堵塞方案进行请求，效率会更高一些。

2.5 服务端数据接受

这一块使用netty，可以快速一个高性能、高可靠的一个服务端。

```
public class ServerHandler extends ChannelInboundHandlerAdapter {

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        ByteBuf buf = (ByteBuf)msg;
        byte[] bytes = new byte[buf.readableBytes()];
        buf.readBytes(bytes);

        byte[] responseBytes = RequestHandler.handler(bytes);
        ByteBuf resbuf = ctx.alloc().buffer(responseBytes.length);
        resbuf.writeBytes(responseBytes);
        ctx.writeAndFlush(resbuf);

    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {
        cause.printStackTrace();
        ctx.close();
    }
}
```


上面代码是我项目中使用的服务端代码。关于netty网上学习的资料很多，这里也只是宏观的讲解RPC原理，就不展开。

2.6 真实调用

服务端获取客户端请求的数据后， 调用请求中的方法，方法参数值，通过反射调用真实的方法，获取其返回值，将其序列化封装，通过netty进行数据返回，客户端在接受数据并解析，这就完成了一次rpc请求调用的全过程。

```
method = clazz.getMethod(request.getMethodName(),requestParamTypes);
method.setAccessible(true);
result = method.invoke(service, requestParmsValues)
```

上面代码片段为通过反射调用真实方法

2.7 服务端的动态加载

通过2.2到2.6的说明，一次RPC请求过程大致如此，但是一个RPC框架会有很多细节需要处理。

其实在一次请求调用前，服务端肯定要先启动。

服务端作为一个容器，跟我们熟知的tomcat一样，它可以动态的加载任何项目。所以在服务端启动的时候，必须要进行一个动态加载的过程。

在KRPC中，我使用了URLClassLoader动态加载一个指定路径的jar包，任何业务服务的实现所依赖的jar包都可以放入该路径中。

3. 总结

一个RPC框架大致需要动态代理、序列化、网络请求、网络请求接受(netty实现)、动态加载、反射这些知识点。

现在开源及各公司自己造的RPC框架层出不穷，唯有掌握原理是一劳永逸的。

掌握原理最好的方法莫不是阅读源码，自己动手写是最快的。

> *来源：blog.csdn.net/u013592964/article/details/80441205*
>
> *整编：Java技术栈（公众号id: javastack）*

推荐阅读

[刚写完这段代码，就被开除了……](#)

[1 分钟教会你用 Spring Boot 发邮件](#)

[都几套房了，还写个毛的代码！](#)

[你必须掌握的 21 个 Java 核心技术！](#)

阅读原文