# 为什么要用 Dubbo

## 远程通信背景

技术架构的发展从单体到分布式，是一种顺势而为的架构演进，也是一种被逼无奈的技术变革。架构的复杂度能够体现公司的业务的复杂度，也能从侧面体现公司的产品的发展势头是向上的。

和传统的单体架构相比，分布式多了一个远程服务之间的通信，不管是 soa 还是微服务，他们本质上都是对于业务服务的提炼和复用。那么远程服务之间的调用才是实现分布式的关键因素。

而在远程通信这个领域，其实有很多的技术，比如 Java 的 RMI、WebService、Hessian、Dubbo、Thrift 等 RPC 框架，现在我们接触得比较多的应该就是 RPC 框架 Dubbo 以及应用协议 Http。其实每一个技术都是在某一个阶段产生它的价值，随着架构的变化以及需求的变化，技术的解决方案也在变。所以我们才需要不断的学习

我在前面的几次课，讲到了 RPC 的底层原理，服务与服务之间的调用无非就是跨进程通信而已，我们可以使用 socket 来实现通信，我们也可以使用 nio 来实现高性能通信。我们不用这些开源的 RPC 框架，也可以完成通信的过程。但是为什么要用现成的框架呢?

原因是，如果我们自己去开发一个网络通信，需要考虑到

1. 底层网络通信协议的处理

2. 序列化和反序列化的处理工作

但是这些工作本身应该是通用的，应该是一个中间件服务。为整个公司提供远程通信的服务。而不应该由业务开发人员来自己去实现，所以才有了这样的 rpc 框架，使得我们调用远程方法时就像调用本地方法那么简单，不需要关心底层的通信逻辑。

大规模服务化对于服务治理的要求

我认为到目前为止，还只是满足了通信的基础需求，但是当企业开始大规模的服务化以后，远程通信带来的弊端就越来越明显了。比如说

1. 服务链路变长了，如何实现对服务链路的跟踪和监控呢?
2. 服务的大规模集群使得服务之间需要依赖第三方注册中心来解决服务的发现和服务的感知问题
3. 服务通信之间的异常，需要有一种保护机制防止一个节点故障引发大规模的系统故障，所以要有容错机制
4. 服务大规模集群会是的客户端需要引入负载均衡机制实现请求分发

而这些对于服务治理的要求，传统的 RPC 技术在这样的场景中显得有点力不从心，因此很多企业开始研发自己的 RPC 框架，比如阿里的 HSF、Dubbo；京东的 JSF 框架、当当的 dubbox、新浪的 motan、蚂蚁金服的 sofa 等等又技术输出能力的公司，都会研发适合自己场景的 rpc 框架，要么是从 0 到 1 开发，要么是基于现有的思想结合公司业务特色进行改造。而没有技术输出能力的公司，遇到服务治理的需求时，会优先选择那些比较成熟的开源框

架。而 Dubbo 就是其中一个

dubbo 主要是一个分布式服务治理解决方案，那么什么是服务治理？服务治理主要是针对大规模服务化以后，服务之间的路由、负载均衡、容错机制、服务降级这些问题的解决方案，而 Dubbo 实现的不仅仅是远程服务通信，并且还解决了服务路由、负载、降级、容错等功能。

Dubbo 的发展历史

Dubbo 是阿里巴巴内部使用的一个分布式服务治理框架，2012 年开源，因为 Dubbo 在公司内部经过了很多的验证相对来说比较成熟，所以在很短的的还是件就被很多互联网公司使用，再加上阿里出来的很多技术大牛进入各个创业公司担任技术架构以后，都以 Dubbo 作为主推的 RPC 框架使得 dubbo 很快成为了很多互联网公司的首要选择。并且很多公司在应用 dubbo 时，会基于自身业务特性进行优化和改进，所以也衍生了很多版本，比如京东的 JSF、比如新浪的 Motan、比如当当的 dubbox.
在 2014 年 10 月份，Dubbo 停止了维护。后来在 2017 年的 9 月份，阿里宣布重启 Dubbo，并且对于 Dubbo 做好了长期投入的准备，并且在这段时间 Dubbo 进行了非常多的更新，目前的版本已经到了 2.7.

2018 年 1 月 8 日，Dubbo 创始人之一梁飞在 Dubbo 交流群里透露了 Dubbo 3.0 正在动工的消息。Dubbo 3.0 内核与 Dubbo2.0 完全不同，但兼容 Dubbo 2.0。Dubbo 3.0 将支持可选 Service Mesh

2018 年 2 月份，Dubbo 捐给了 Apache。另外，阿里巴巴对于 Spring Cloud Alibaba 生态的完善，以及 Spring Cloud 团队对于 alibaba 整个服务治理生态的支持，所以 Dubbo 未来依然是国内绝大部分公司的首要选择。

## Dubbo 的基本使用

先声明一下， 我们这节课会基于 dubbo 最新版本 2.7.2 的版本来讲解

按照国际惯例，我们还是基于一个 demo，使用 dubbo 完成基本的远程通信

**创建两个项目**

创建一个 server 项目和和 client 项目

**添加 jar 包依赖**

```
<dependency>
    <groupId>org.apache.dubbo</groupId>
    <artifactId>dubbo</artifactId>
    <version>2.7.2</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
```

```
    <artifactId>slf4j-api</artifactId>
    <version>1.7.26</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
```

**定义接口及实现**

```
public interface LoginService {
    String login(String username, String password);
}

public class LoginServiceImpl implements LoginService{
    @Override
    public String login(String username, String password) {
        if(username.equals("admin")&&password.equals("admin")){
            return "SUCCESS";
        }
        return "FAILED";
```

```
    }
}
```

**创建配置文件发布服务**

*在 resources/META-INF/spring 下创建 application.xml 文件*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <!-- 提供方应用信息，用于计算依赖关系 -->
    <dubbo:application name="practice-server"  />
    <!-- 使用 multicast 广播注册中心暴露服务地址 -->
    <dubbo:registry address="N/A" />
    <!-- 用 dubbo 协议在 20880 端口暴露服务 -->
    <dubbo:protocol name="dubbo" port="20880" />
    <!-- 声明需要暴露的服务接口 -->
    <dubbo:service
```

```
interface="com.gupaoedu.practice.LoginService"
ref="loginService" />
    <!-- 和本地 bean 一样实现服务 -->
    <bean id="loginService"
class="com.gupaoedu.practice.LoginServiceImpl" />

</beans>
```

**启动服务**

```
Main.main(args);
```

**添加日志的支持**

*添加 logback.xml 文件*

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT"
class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>%date{ISO8601} %-5level
[%thread] %logger{32} - %message%n</pattern>
        </layout>
    </appender>
```

```
    <root>
        <level value="DEBUG"/>
        <appender-ref ref="STDOUT"/>
    </root>
</configuration>
```

客户端配置

**添加配置文件**

在 resources/下创建 application.xml 文件

```
<!-- 提供方应用信息，用于计算依赖关系 -->
<dubbo:application name="practice-client"  />

<!-- 使用 multicast 广播注册中心暴露服务地址 -->
<dubbo:registry address="N/A" />

<dubbo:reference id="loginService"
interface="com.gupaoedu.practice.LoginService"
url="dubbo://192.168.13.1:20880/com.gupaoedu.practice.LoginService"/>
```

**访问远程服务**

```
public static void main( String[] args ){
    ClassPathXmlApplicationContext
classPathXmlApplicationContext=
            new ClassPathXmlApplicationContext(new
String[]{"application.xml"});
    LoginService
loginService=(LoginService)classPathXmlApplicationContext.getBean
("loginService");
    System.out.println(loginService.login("admin","admin"));
}
```

## 关于 DubboMain 启动的真相

我们刚刚使用 Main.main(args); 来启动 dubbo 服务，到底是如何实现的呢?

正常情况下，我们会认为服务的发布，需要 tomcat、或者 jetty 这类的容器支持，但是只用 Dubbo 以后，我们并不需要这样重的服务器去支持，同时也会增加复杂性，和浪费资源。Dubbo 提供了几种容器让我们去启动和发布服务

## 容器类型

Spring Container

自动加载 META-INF/spring 目录下的所有 Spring 配置。

logback Container

自动装配 logback 日志

Log4j Container

自动配置 log4j 的配置

Dubbo 提供了一个 Main.main 快速启动相应的容器，默认情况下，只会启动 spring 容器

## 原理分析

默认情况下，spring 容器，本质上，就是加在 spring ioc 容器，然后启动一个 netty 服务实现服务的发布，所以并没有特别多的黑科技，下面是 spring 容器启动的代码

```
public void start() {
    String configPath =
ConfigUtils.getProperty("dubbo.spring.config");
    if (StringUtils.isEmpty(configPath)) {
        configPath = "classpath*:META-INF/spring/*.xml";
    }
```

```
        context = new
ClassPathXmlApplicationContext(configPath.split("[,\\s]+"),
false);
        context.refresh();
        context.start();
    }
```

## 基于注册中心的 Dubbo 服务

作为主流的服务治理组件，Dubbo 提供了很多丰富的功能，那么最根本的就是要解决大规模集群之后的服务注册和发现的问题。而 dubbo 中对于注册中心这块是使用 zookeeper 来支撑的。当然在目前最新的版本中。Dubbo 能够支持的注册中心有：consul、etcd、nacos、sofa、zookeeper、redis、multicast

使用 zookeeper 作为注册中心

**Jar 包依赖**

```
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>4.0.0</version>
</dependency>
```

```
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>4.0.0</version>
</dependency>
```

**修改 application.xml 文件**

```
<dubbo:registry address="zookeeper://192.168.13.102:2181" />
```

如果是 zookeeper 集群，则配置的方式是

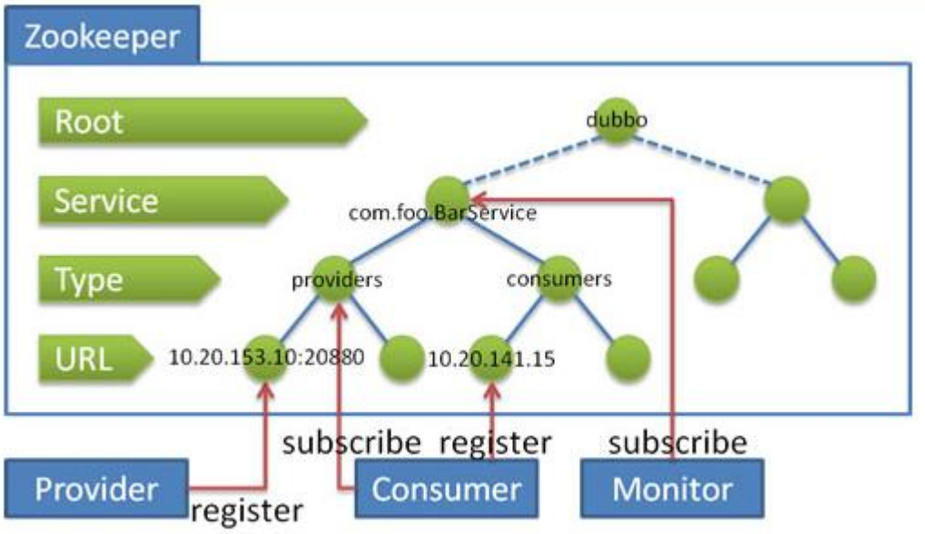
address="zookeeper://ip:host?backup=ip:host,ip:host"

**客户端改造**

```
<!-- 使用 multicast 广播注册中心暴露服务地址 -->
<dubbo:registry address="zookeeper://192.168.13.102:2181" />

<dubbo:reference id="loginService"
interface="com.gupaoedu.practice.LoginService"/>
```

Dubbo 集成 Zookeeper 的实现原理




dubbo 每次都要连 zookeeper?

很多同学会有疑问，是不是每次发起一个请求的时候，都需要访问注册中心呢？作为一个设计者，你会怎么去解决这个问题。

答案很显然是通过缓存实现

在消费端的配置文件中指定如下路径

```
<dubbo:registry id="zookeeper"
address="zookeeper://192.168.13.102:2181" file="d:/dubbo-
server" />
```

其他注册中心的实现，核心本质是一样的，都是为了管理服务地址，后续会讲到 nacos;

多注册中心支持

Dubbo 中可以支持多注册中心，有的时候，客户端需要用调用的远程服务不在同一个注册中心上，那么客户端就需要配置多个注册中心来访问。演示一个简单的案例

**配置多个注册中心**

```
<dubbo:registry address="zookeeper://192.168.13.102:2181"
id="registryCenter1"/>
<dubbo:registry address="zookeeper://192.168.13.102:2181"
id="registryCenter2"/>
```

**将服务注册到不同的注册中心**

通过 registry 设置注册中心的 ID

```
<dubbo:service interface="com.gupaoedu.practice.LoginService"
registry="registryCenter1" ref="loginService" />
```

**消费端配置多个注册中心**

实现的代码和服务端一样

注册中心的其他支持

1. 当设置 <dubbo:registry check="false" /> 时，记录失败注册和订阅请求，后台定时重试

2. 可通过 <dubbo:registry username="admin" password="1234" /> 设置 zookeeper 登录信息
3. 可通过 <dubbo:registry group="dubbo" /> 设置 zookeeper 的根节点，默认使用 dubbo 作为 dubbo 服务注册的 namespace

## Dubbo 仅仅是一个 RPC 框架?

到目前为止，我们了解到了 Dubbo 的核心功能，提供服务注册和服务发现以及基于 Dubbo 协议的远程通信，我想，大家以后不会仅仅只认为 Dubbo 是一个 RPC 框架吧。 Dubbo 从另一个方面来看也可以认为是一个服务治理生态。从目前已经讲过的内容上可以看到。

1. Dubbo 可以支持市面上主流的注册中心
2. Dubbo 提供了 Container 的支持，默认提供了 3 种 container。我们可以自行扩展
3. Dubbo 对于 RPC 通信协议的支持，不仅仅是原生的 Dubbo 协议，它还围绕着 rmi、hessian、http、webservice、thrift、rest

有了多协议的支持，使得其他 rpc 框架的应用程序可以快速的切入到 dubbo 生态中。 同时，对于多协议的支持，使得不同应用场景的服务，可以选择合适的协议来发布服务，并不一定要使用 dubbo 提供的长连接方式。

## 集成 Webservice 协议

webservice 是一个短链接并且是基于 http 协议的方式来实现的 rpc 框架。

### Jar 包依赖

```
    <dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-frontend-simple</artifactId>
    <version>3.3.2</version>
</dependency>
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-transports-http</artifactId>
    <version>3.3.2</version>
</dependency>
<dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-server</artifactId>
    <version>9.4.19.v20190610</version>
</dependency>
<dependency>
    <groupId>org.eclipse.jetty</groupId>
```

```
    <artifactId>jetty-servlet</artifactId>
    <version>9.4.19.v20190610</version>
</dependency>
```

**修改 application.xml**

```
    <!-- 用 dubbo 协议在 20880 端口暴露服务 -->
<dubbo:protocol name="dubbo" port="20880" />
<dubbo:protocol name="webservice" port="8080" server="jetty"/>

<!-- 声明需要暴露的服务接口 -->
<dubbo:service interface="com.gupaoedu.practice.LoginService"
registry="registryCenter1"
                ref="loginService" protocol="dubbo,webservice"/>
```

添加多协议支持，一个服务可以发布多种协议的支持，也可以实现不同服务发布不同的协议

启动服务之后，可以使用：http://localhost:8080/com.gupaoedu.practice.LoginService?wsdl 来获得 Webservice 的 wsdl 描述文档

**客户端使用 webservice 请求服务**

1. 客户端的配置 jar 包依赖

2. 配置多个协议支持，实现方式和服务端一致

Dubbo 对于 REST 协议的支持

我在用 REST 协议发布服务的时候，遇到 Dubbo 的一个坑， Dubbo 启动时后写地方的服务直接被吞掉了，使得服务启动不了的同时，还看不到错误信息。

（可以把 resteasy-client 包暂时不导入，看到效果）

Dubbo 中的 REST（表述性资源转移）支持，是基于 JAX-RS2.0(Java API for RESTful Web Services)来实现的。

REST 是一种架构风格，简单来说就是对于 api 接口的约束，基于 URL 定位资源，使用 http 动词（GET/POST/DELETE）来描述操作

**JAX-RS 协议说明**

REST 很早就提出来了，在早期开发人员为了实现 REST，会使用各种工具来实现，比如 Servlets 就经常用来开发 RESTful 的程序。随着 REST 被越来越多的开发人员采用，所以 JCP(Java community process)提出了 JAX-RS 规范，并且提供了一种新的基于注解的方式来开发 RESTful 服务。有了这样的一个规范，使得开发人员不需要关心通讯层的东西，只需要关注资源以以及数据对象。

JAX-RS 规范的实现有：Apache CXF、Jersey(由 Sun 公司提供的 JAX-RS 的参考实现)、RESTEasy(jboss 实现)等。

而 Dubbo 里面实现的 REST 就是基于 Jboss 提供的 RESTEasy 框架来实现的

SpringMVC 中的 RESTful 实现我们用得比较多，它也是 JAX-RS 规范的一种实现

**Jar 包依赖**

```
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-jaxrs</artifactId>
    <version>3.8.0.Final</version>
</dependency>
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-client</artifactId>
    <version>3.8.0.Final</version>
</dependency>
```

**添加新的协议支持**

```
<dubbo:protocol name="rest" port="8888" server="jetty"/>
```

**提供新的服务**

@Path("/users")：指定访问 UserService 的 URL 相对路径是/users，即 http://localhost:8080/users

@Path("/register")：指定访问 registerUser()方法的 URL 相对路径是/register，

再结合上一个@Path 为 UserService 指定的路径，则调用 UserService.register()的完整路径为 http://localhost:8080/users/register

@POST：指定访问 registerUser()用 HTTP POST 方法

@Consumes({MediaType.APPLICATION_JSON})：指定 registerUser()接收 JSON 格式的数据。REST 框架会自动将 JSON 数据反序列化为 User 对象

（注解可以放在接口上，客户端需要根据接口的注解来进行解析和调用）

```
@Path("/user")
public interface UserService {
    @GET
    @Path("/register/{id}")
    void register(@PathParam("id") int id);
}
```

**客户端配置**

```
<dubbo:protocol name="rest" port="8888" server="jetty"/>
<dubbo:reference id="userService"
interface="com.gupaoedu.practice.UserService" protocol="rest"/>
```

**在服务接口获得上下文的方式**

既然是 http 协议的 REST 接口，那么我们想要获得请求的上下文，怎么做呢?

第一种

HttpServletRequest
request=(HttpServletRequest)RpcContext.getContext().getRequest();
第二种
通过注解的方式

```
@GET
@Path("/register/{id}")
void register(@PathParam("id") int id, @Context
HttpServletRequest request);
```

其他协议我们就不一一演示了，除了协议本身的不同之外，其他所有的东西都是一样，包括负载均衡、服务注册、熔断、监控等。

## Dubbo 监控平台安装

Dubbo 的监控平台也做了更新，不过目前的功能还没有完善，
在这个网站上下载 Dubbo-Admin 的包：https://github.com/apache/dubbo-admin

1. 修改 dubbo-admin-server/src/main/resources/application.properties 中的配置信息
2. mvn clean package 进行构建
3. mvn –projects dubbo-admin-server spring-boot:run

4. 访问 localhost:8080

## Dubbo 的终端操作方法

之前有个同学问我，如果不看 zookeeper、也不看监控平台，如何知道这个服务是否正常呢?

Dubbo 里面提供了一种基于终端操作的方法来实现服务治理

使用 telnet localhost 20880 连接到服务对应的端口

常见命令

ls

ls: 显示服务列表

ls -l: 显示服务详细信息列表

ls XxxService: 显示服务的方法列表

ls -l XxxService: 显示服务的方法详细信息列表

---

ps

ps: 显示服务端口列表

ps -l: 显示服务地址列表

ps 20880: 显示端口上的连接信息

ps -l 20880: 显示端口上的连接详细信息

---

cd

cd XxxService：改变缺省服务，当设置了缺省服务，凡是需要输入服务名作为参数的命令，都可以省略服务参数

cd /：取消缺省服务

---

pwd

pwd：显示当前缺省服务

---

count

count XxxService：统计 1 次服务任意方法的调用情况

count XxxService 10：统计 10 次服务任意方法的调用情况

count XxxService xxxMethod：统计 1 次服务方法的调用情况

count XxxService xxxMethod 10：统计 10 次服务方法的调用情况

---