**分布式与单体的关键区别：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。**

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。

关于dubbo的思考：1.因为dubbo嵌入到spring中，并且看到有<dubbo>标签，其一定实现了自定义标签。所以在dubbo的jar包中的spring.handlers发现了DubboNamespaceHandler

相关的类。2.dubbo因为涉及到rpc通信，所以一定要发布服务，而发布服务一定要暴露端口，而暴露端口一般使用nio,socket,netty等通信类或者框架。

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

Spi机制了解：

相比使用提供接口jar包，供第三方服务模块实现接口的方式，SPI的方式使得源框架，不必关心接口的实现类的路径，可以不用通过下面的方式获取接口实现类：

* 代码硬编码import 导入实现类
* 指定类全路径反射获取：例如在JDBC4.0之前，JDBC中获取数据库驱动类需要通过**Class.forName("com.mysql.jdbc.Driver")**，类似语句先动态加载数据库相关的驱动，然后再进行获取连接等的操作
* 第三方服务模块把接口实现类实例注册到指定地方，源框架从该处访问实例

缺点：

1. JDK 标准的 SPI 会一次性加载实例化扩展点的所有实现，什么意思呢？就是如果你在 META-INF/service 下的文件里面加了 N个实现类，那么 JDK 启动的时候都会一次性全部加载。那么如果有的扩展点实现初始化很耗时或者如果有些实现类并没有用到，

那么会很浪费资源

1. 如果扩展点加载失败，会导致调用方报错，而且这个错误很难定位到是这个原因

· 虽然ServiceLoader也算是使用的延迟加载，但是基本只能通过遍历全部获取，也就是接口的实现类全部加载并实例化一遍。如果你并不想用某些实现类，它也被加载并实例化了，这就造成了浪费。获取某个实现类的方式不够灵活，只能通过Iterator形式获取，不能根据某个参数来获取对应的实现类。（dubbo可以实现）

· 多个并发多线程使用ServiceLoader类的实例是不安全的

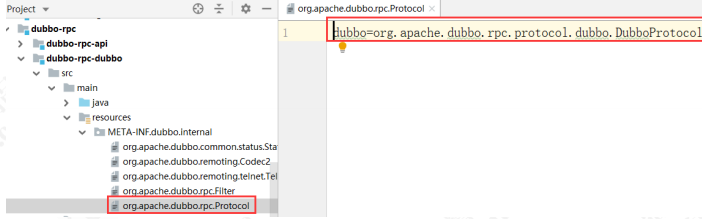
参考网站：<https://www.jianshu.com/p/46b42f7f593c>

Duboo里面充分使用了spi机制：

一．Dubbo 的 SPI 扩展机制，有两个规则

1. 需要在 resource 目录下配置 META-INF/dubbo 或者META-INF/dubbo/internal 或者 META-INF/services，并基于 SPI 接口去 创建一个文件

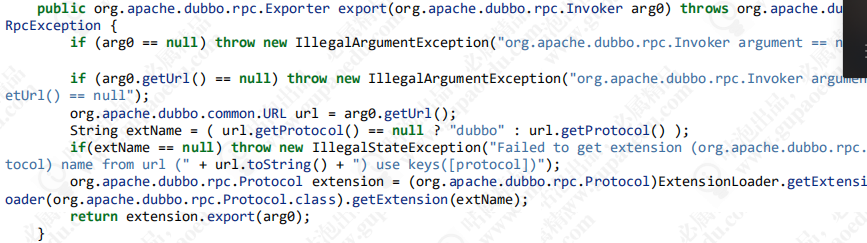
2. 文件名称和接口名称保持一致，文件内容和 SPI 有差异，内容是 KEY 对应 Value



Protocol protocol = ExtensionLoader.getExtensionLoader(Protocol.class). getExtension("myProtocol");

通过这句话就可以得到myProtocol实例对象。但是这里不是还是传入“myProtocol”名称吗，实际上在代码中可以传入一个url地址，其协议头可以解析出来该参数，根据协议头的变化可以获取不同的Protocol类。还可以获取自适应扩展点，跟默认值相似。在类上加了@adaptive注解的为自适应扩展类，这个注解还可以加在方法上，在加载器找不到自适应扩展点，会根据类上的注解创建一个代理类作为自适应扩展类。

以某个类的export方法为例：



在这里，当调用该代理扩展点某个方法时，其会找到被代理类，再调用该类的方法。（注意，代理类的生成使用过拼接字符串的方式生成的）

### 自适应激活扩展点：

自动激活扩展点，有点类似我们讲 springboot 的时候用到的 conditional，根据条件进行自动激活。但是这里设计的初衷是，对于一个类会加载多个扩展点的实现，这个时候可以通过自动激活扩展点进行动态加载， 从而简化配置我们的配置工作。@Activate 提供了一些配置来允许我们配置加载条件，比如 group 过滤，比如 key 过滤。满足某些条件则加载，否则不加载。

Dubbo和spring的结合：细节直接参考pdf文件：

这里只看dubbo服务的入口怎么找。Dubbo将service标签解析后得到ServiceBean类。（第一个spring扩展点）

ServiceBean又继承或者实现了initializingBean，DisposableBean，ApplicationAware，ApplicationListener，ApplicaionEventPublistener。

这几个类都是Spring的扩展点。initializingBean的afterPropertiesSet()方法为发布前进行了一些准备工作。DisposableBean的destroy()方法会在bean销毁时清理资源。

ApplicationContextAware可以将ApplicationContext设置进去。ApplicationListener监听刷新上下文事件，在spring启动完成后执行onApplicationEvent()，dubbo的发布服务就是在这里进行的。

几种不同协议的支持，有不同的使用方法。