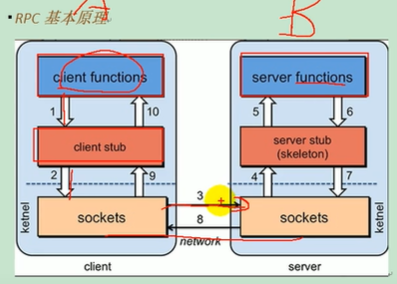
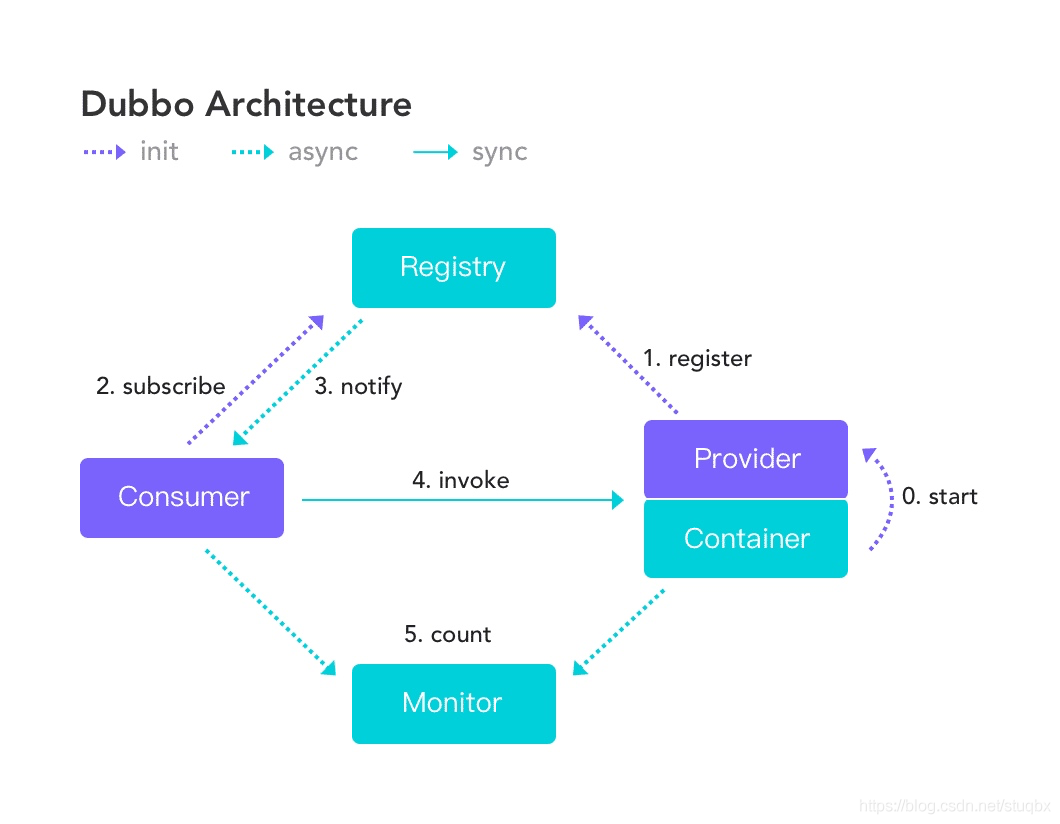
应用程序间的通讯，服务化 、微服务 和分布式系统架构 的基础场景。即RPC框架就是实现以上结构的有力方式。



RPC两个核心模块，通讯，序列化

使用Dubbo可以将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，可用于提高业务复用灵活扩展，使前端应用能更快速的响应多变的市场需求。

架构图



DUBBO分为四个模块，分别为：注册中心（Registry）、提供者（Provider）、消费者（Consumer）和监控（Monitor）。

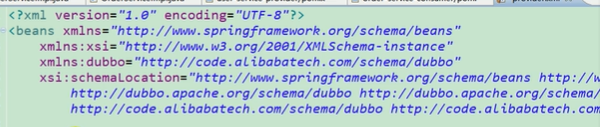
* 注册中心（Registry）：可以是zookeeper、redis、multicast、simple（官方推荐使用Zookeeper）；
* 提供者（Provider）：服务启动时，Provider引用容器中的服务，并向Registry注册服务，同时暴露服务（Consumer是直接和Provider通讯实现服务调用的）。
* 消费者（Consumer）：服务启动时，Consumer向Registry订阅服务，如果没有订阅到自己想获得的服务，它会不断的尝试订阅。新的服务注册到注册中心以后，注册中心会将这些服务通过notify到消费者。Consumer直接调用Provider提供的服务。
* 监控（Monitor）：Consumer和Provider会通过异步的方式定时向Monitor发送消息，报告服务的状态。Monitor在整个架构中是可选的，Monitor功能需要单独配置，不配置或者配置后挂掉并不会影响服务的调用。

工程建立：

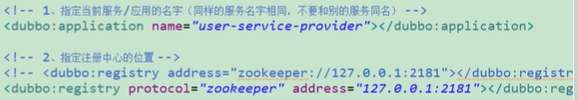
1. 引入dubbo依赖



1. Dubbo命名空间



1. 注册中心



1. 通信规则和端口



1. 暴露服务 需要服务实现





Dubbo和spring依赖信息



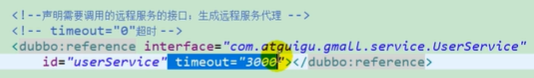
@service注解可以暴露服务

@Reference 远程引用服务，从注册中心发现

**参数覆盖策略**



**超时属性**



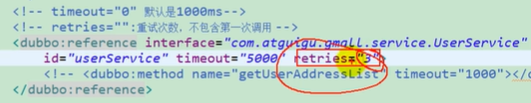


使用注解进行配置的话，属性就在@service中或者@reference

配置覆盖



重试次数，并不包含第一次，例如设置3，失败后还会再设置3次



幂等例如查询删除修改操作可以设置重试次数，非幂等例如新增则不能设置重试次数

**单个提供方如果想要提供多个相同服务，需要修改端口然后再运行**。

**负载均衡：提升某个服务的权重**

**服务降级：当服务器压力剧增时，根据业务情况和流量，对一些服务和页面有策略的不处理或换种简单的方式处理，从而释放服务器资源以保证核心交易正常运作或高效运作**

**两种降级方式**

