# 1.Dubbo是什么

Dubbo是一个分布式、高性能、透明化的RPC服务调用，提供服务自身自动注册、自动发现等高效服务治理方案，可以与Spring框架无缝集成。

RPC指的是远程调用协议，也就是说两个服务器交互数据。

# 2.Dubbo的由来

互联网的快速发展，Web应用程序规模不断扩大，一般会经历如下四个发展阶段

1. 单一应用架构：当网站流量很小时，只需要一个应用，将所有功能都部署在一起即可。
2. 垂直应用架构：当访问量逐渐增大，单一应用按照业务拆分成多个应用，以提升效率。此时，用来加速前端页面开发的**Web框架(MVC)**是关键。
3. 分布式服务架构：当垂直应用越来越多，应用之间交互不可避免，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，使前端应用能更快的响应多变的市场需求。此时，用于提高业务复用及整合的**分布式服务框架(RPC)**是关键。
4. 流动计算架构：当服务越来越多，容量的评估，小服务器资源的浪费等问题逐渐呈现，**此时需要增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率**。此时，用于提高机器利用率的**资源调度和治理中心(SOA)**是关键。

# 3.Dubbo的主要应用场景

透明化的远程方法调用，就像调用本地方法一样调用远程方法，只需简单配置，没有任何API侵入。

软负载均衡及容错机制，可在内网代替F5等硬件负载均衡器，降低成本，减少单点。

服务自动注册和发现，不需要写死服务提供方地址，注册中心基于接口查询服务提供者的IP地址，并且能够平滑的添加和删除服务提供者。

# 4.Dubbo的核心功能

主要有如下三个核心功能：

Remoting：网络通信框架，提供多种NIO框架抽象封装，包括’同步转异步’和’请求-响应’模式的信息交换方式。

Cluster：服务框架，提供基于接口方法的透明远程调用，包括多协议支持，以及软负载均衡，失败容错，地址路由，动态配置等集群支持。

Registery：服务注册，基于注册中心目录服务，使服务消费方能动态的查找服务提供方，使地址透明，使服务提供方可以平滑的增加或减少机器。

# Dubbo的核心组件

Provider：暴露服务的服务提供方

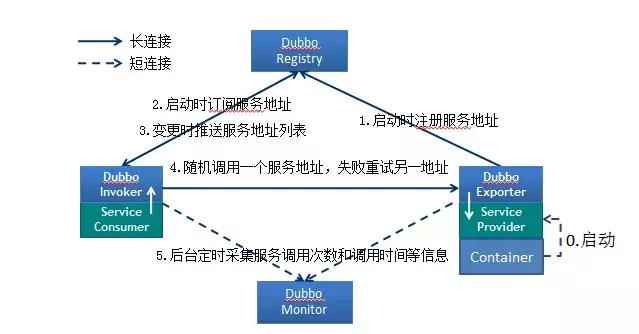
Consumer：调用远程服务的服务消费方

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

# 6.Dubbo服务注册和发现的流程



流程说明：

1.Provider(提供者)：绑定指定端口并启动服务

2.提供者连接注册中心：并发本机IP、端口、应用信息和提供服务信息发送至注册中心存储

3.Consumer(消费者)：连接注册中心，并发送应用信息、所求服务信息至注册中心

4.注册中心根据消费者所求服务信息匹配对应的提供者列表发送至Consumer应用缓存

5.Consumer在发起远程调用时基于缓存的消费者列表选择其一发起调用

6.Proider状态变更会实时通知注册中心、在由注册中心实时推送到Consumer

设计的原因：

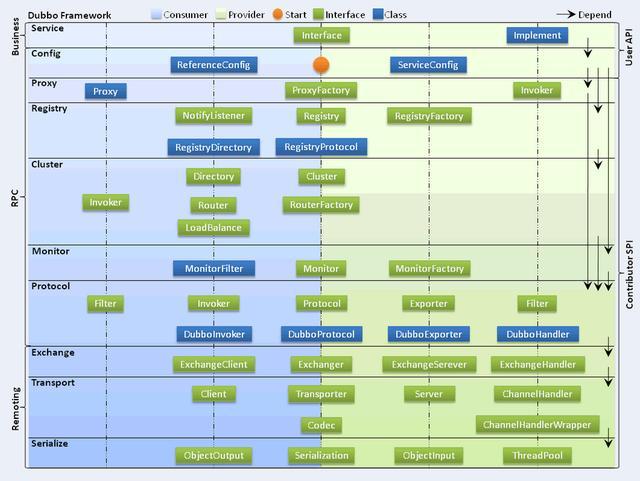
1.Consumer与Provider解耦，双方都可以横向增减节点数。

2.注册中心对本身可做对等集群，可动态的增减节点，并且任意一台宕掉后，将自动切换到另一台。

3.去中心化，双方不直接依赖注册中心，即使注册中心全部宕机短时间内也不影响服务的调用。

4.服务提供者无状态，任意一台宕机后，不影响使用。

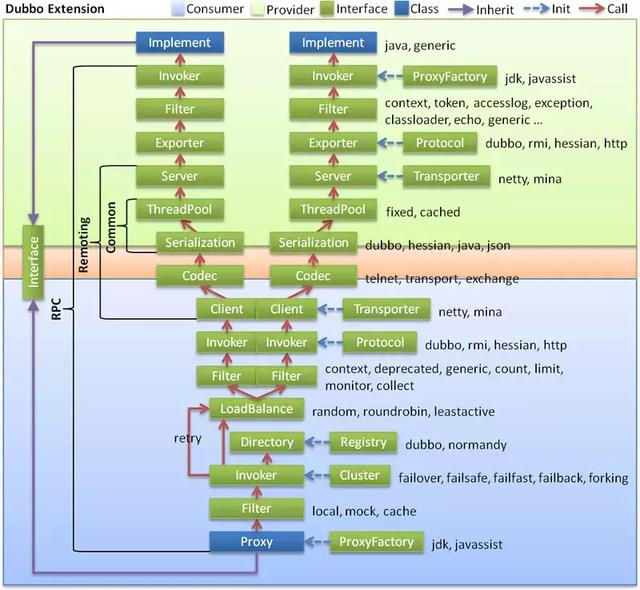
# 7.Dubbo的架构设计



Dubbo框架设计一共划分了10个层：

1. 服务接口层(Service)：该层是和实际业务逻辑相关的，根据服务提供方和服务消费方的业务设计对应的接口和实现。
2. 配置层(Config)：对外配置接口，以ServiceConfig和RefenceConfig为中心。
3. 服务代理层(Proxy)：服务接口透明代理，生成服务的客户端Stub和服务端Skeleton。
4. 服务注册层(Registry)：封装服务地址的注册和发现，以服务URL为中心。
5. 集群层(Cluster)：封装多个提供者的路由及负载均衡，并桥接注册中心，以Invoker为中心。
6. 监控层(Monitor)：RPC调用次数和调用时间监控。
7. 远程调用层(Protocol)：封装RPC调用，以Invocation和Result为中心，扩展接口为Protocol、Invoker和Exporter。
8. 信息交换层(Exchange)：封装请求响应模式，同步转异步，以Request和Response为中心。
9. 网络传输层(Transport)：抽象mina和netty为统一接口，以Message为中心。
10. 序列化层(Serialize)：进行数据序列操作。

# 8.Dubbo的服务调用流程



# 9.Dubbo支持哪些协议

1.**dubbo**(默认)：单一长连接和NIO异步通讯，适合大并发小数据量的服务调用，以及消费者远大于提供者。传输协议TCP、异步、Hessian序列化。

2.**RMI**：采用JDK标准的RMI协议实现，传输参数和返回参数对象需要实现Serializable接口，使用java标准序列化机制，使用阻塞式短连接，传输数据包大小混合，消费者和提供者个数差不多，可传文件，传输协议TCP。

3.**webservice**：基于webservice的远程调用协议，集成CFX实现，提供和原生webservice的互操作。多个短连接，基于传输，同步传输，适用系统集成和跨语言调用。

４.**http**：基于HTTP表单提交的远程调用协议，使用spring的HttpInvoker实现。多个短连接，传输协议HTTP，传入参数大小混合，提供者个数多于消费者，需要给应用程序和浏览器JS调用。

5.**hessian**：集成Hessian服务，基于通讯，采用Servlet暴露服务，Dubbo内嵌Jetty作为服务器时默认实现，提供与Hession服务互操作。多个短连接，同步传输，Hession序列化，传入参数较大，提供者大于消费者，提供者压力较大，可传文件。

6.**memcache**：基于memcache实现的RPC协议。

7.**redis**：基于redis实现的RPC协议。

# 10.Dubbo有哪些注册中心

1.Multicast注册中心：Multicast注册中心不需要任何中心节点，只要广播地址，就能进行服务注册和发现。

2.Zookeeper注册中心：基于分布式协调系统Zookeeper实现，采用Zookeeper的watch机制实现数据变更。

3.Redis注册中心：基于Redis实现，采用key/map存储，key存储服务名和类型，map存储服务URL，value存储服务过期时间。基于Redis的发布/订阅模式通知数据变更。

4.Simple注册中心

# 11.Dubbo的服务治理

1.过多的服务URL配置困难。

2.负载均衡分配节点压力过大的情况下也需要部署集群。

3.服务依赖混乱，启动顺序不清晰。

4.过多服务导致性能指标分析难度较大，需要监控。

# 12.Dubbo的注册中心进群挂掉后，发布者和订阅者还能通信吗？

可以的，启动dubbo时，消费者会从Zookeeper拉取注册的生产者的地址接口等数据，缓存在本地。每次调用时，按照本地存储的地址进行调用。

# 13.Dubbo集群提供了哪些负载均衡策略

1.Random LoadBalance：随机选取提供者策略，有利于动态调整提供者权重。截面碰撞率高，调用次数越多，分布越均匀。

2.RoundRobin LoadBalance：轮询选取提供者策略，平均分布，但是存在请求累积的问题。

3.LeastActive LoadBalance：最少活跃调用策略，解决慢提供者接收更少的请求。

4.ConstantHash LoadBalance：一致性hash策略，使相同参数请求总是发送到同一提供者，一台机器宕机，可以基于虚拟节点，分摊至其他提供者，避免引起提供者的剧烈变动。

# 14.Dubbo集群的容错方案

1.Failover Cluster：失败自动切换，当出现失败，重试其他服务器。通常用于读操作，但重试会带来更长延迟。

2.Failfast Cluster：快速失败，只发起一次调用，失败立即报错。通常用于非幂等性的写操作，比如新增记录。

3.Failsafe Cluster：失败安全，出现异常时，直接忽略。通常用于写入审计日志等操作。

4.Failback Cluster：失败自动恢复，后台记录失败请求，定时重发。通常用于消息通知操作。

5.Forking Cluster：并行调用多个服务器，只要一个成功即返回。通常用于实时性较高的读操作，但需要浪费更多服务资源。可通过forks=2来设置最大并行数。

6.Broadcast Cluster：广播调用所有所有提供者，逐个调用，任意一台报错则报错。通常用于通知所有提供者更新缓存或日志等本地资源。