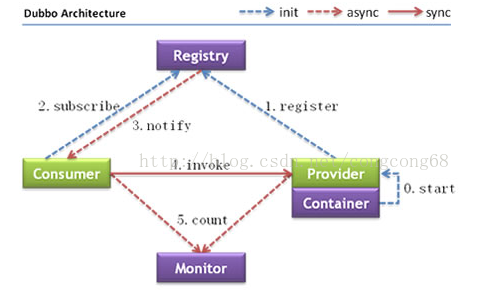
****一、Duboo基本概念解释****

Dubbo是一种分布式服务框架。 Webservice也是一种服务框架，但是webservice并不是分布式的服务框架，他需要结合F5实现负载均衡。因此，dubbo除了可以提供服务之外，还可以实现软负载均衡。它还提供了两个功能Monitor 监控中心和调用中心。这两个是可选的，需要单独配置。

Dubbo的计数架构图如下：



我们解释以下这个架构图：

Consumer服务消费者，Provider服务提供者。Container服务容器。消费当然是invoke提供者了，invoke这条实线按照图上的说明当然同步的意思了，多说一句，在实际调用过程中，Provider的位置对于Consumer来说是透明的，上一次调用服务的位置（IP地址）和下一次调用服务的位置，是不确定的。这个地方就是实现了软负载。

服务提供者先启动start，然后注册register服务。

消费订阅subscribe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。

Monitor这是一个监控，图中虚线表明Consumer 和Provider通过异步的方式发送消息至Monitor，Consumer和Provider会将信息存放在本地磁盘，平均1min会发送一次信息。Monitor在整个架构中是可选的（图中的虚线并不是可选的意思），Monitor功能需要单独配置，不配置或者配置以后，Monitor挂掉并不会影响服务的调用。

****二、dubbo原理****

****I、初始化过程细节：****   
上图中的第一步start，就是将服务装载容器中，然后准备注册服务。和Spring中启动过程类似，spring启动时，将bean装载进容器中的时候，首先要解析bean。所以dubbo也是先读配置文件解析服务。   
****解析服务：****   
1）、基于dubbo.jar内的Meta-inf/spring.handlers配置，spring在遇到dubbo名称空间时，会回调DubboNamespaceHandler类。   
2）、所有的dubbo标签，都统一用DubboBeanDefinitionParser进行解析，基于一对一属性映射，将XML标签解析为Bean对象。   
在ServiceConfig.export 或者ReferenceConfig.get 初始化时，将Bean对象转会为url格式，将所以Bean属性转成url的参数。   
然后将URL传给Protocol扩展点，基于扩展点的Adaptive机制，根据URL的协议头，进行不同协议的服务暴露和引用。   
****暴露服务：****

a、 只暴露服务端口

在没有使用注册中心的情况，这种情况一般适用在开发环境下，服务的调用这和提供在同一个IP上，只需要打开服务的端口即可。   
即，当配置 or   
ServiceConfig解析出的URL的格式为：   
Dubbo：//service-host/com.xxx.TxxService?version=1.0.0   
基于扩展点的Adaptiver机制，通过URL的“dubbo：//”协议头识别，直接调用DubboProtocol的export（）方法，打开服务端口。

b、向注册中心暴露服务：

和上一种的区别：需要将服务的IP和端口一同暴露给注册中心。   
ServiceConfig解析出的url格式为：   
registry://registry-host/com.alibaba.dubbo.registry.RegistryService?export=URL.encode(“dubbo://service-host/com.xxx.TxxService?version=1.0.0”)

基于扩展点的Adaptive机制，通过URL的“registry：//”协议头识别，调用RegistryProtocol的export方法，将export参数中的提供者URL先注册到注册中心，再重新传给Protocol扩展点进行暴露：   
Dubbo：//service-host/com.xxx.TxxService?version=1.0.0

****引用服务：****

a、直接引用服务：

在没有注册中心的，直连提供者情况下，   
ReferenceConfig解析出的URL格式为：   
Dubbo：//service-host/com.xxx.TxxService?version=1.0.0

基于扩展点的Adaptive机制，通过url的“dubbo：//”协议头识别，直接调用DubboProtocol的refer方法，返回提供者引用。

b、从注册中心发现引用服务：

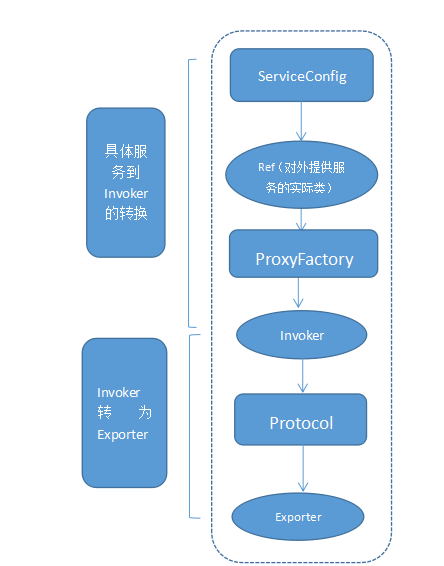
此时，ReferenceConfig解析出的URL的格式为：   
registry://registry-host/com.alibaba.dubbo.registry.RegistryService?refer=URL.encode(“consumer://consumer-host/com.foo.FooService?version=1.0.0”)

基于扩展点的Apaptive机制，通过URL的“registry：//”协议头识别，就会调用RegistryProtocol的refer方法，基于refer参数总的条件，查询提供者URL，如：   
Dubbo：//service-host/com.xxx.TxxService?version=1.0.0

基于扩展点的Adaptive机制，通过提供者URL的“dubbo：//”协议头识别，就会调用DubboProtocol的refer（）方法，得到提供者引用。   
然后RegistryProtocol将多个提供者引用，通过Cluster扩展点，伪装成单个提供这引用返回。

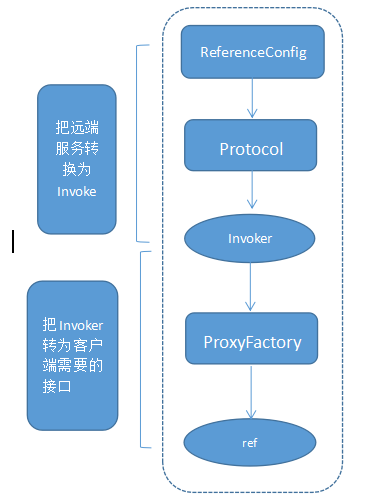
****三、远程调用细节：****

服务提供者暴露一个服务的详细过程：



上图是服务提供者暴露服务的主过程：   
首先ServiceConfig类拿到对外提供服务的实际类ref，然后将ProxyFactory类的getInvoker方法使用ref生成一个AbstractProxyInvoker实例，到这一步就完成具体服务到invoker的转化。接下来就是Invoker转换到Exporter的过程。   
Dubbo处理服务暴露的关键就在Invoker转换到Exporter的过程，下面我们以Dubbo和rmi这两种典型协议的实现来进行说明：   
Dubbo的实现：   
Dubbo协议的Invoker转为Exporter发生在DubboProtocol类的export方法，它主要是打开socket侦听服务，并接收客户端发来的各种请求，通讯细节由dubbo自己实现。   
Rmi的实现：   
RMI协议的Invoker转为Exporter发生在RmiProtocol类的export方法，他通过Spring或Dubbo或JDK来实现服务，通讯细节由JDK底层来实现。

服务消费者消费一个服务的详细过程。



上图是服务消费的主过程：   
首先ReferenceConfig类的init方法调用Protocol的refer方法生成Invoker实例。接下来把Invoker转为客户端需要的接口