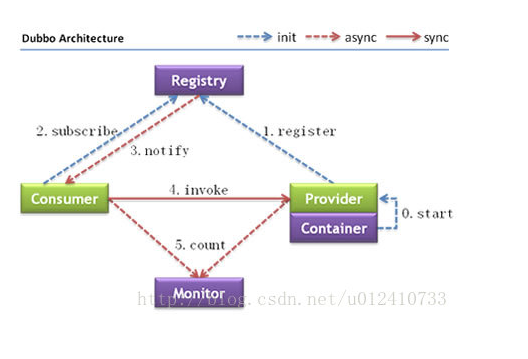
# 基础知识点

### ****Dubbo角色****

从服务模型的角度来看,Dubbo采用的是一种非常简单的模型,要么是提供方提供服务，要么是消费方消费服务，所以基于这一点dubbo里面抽象出了服务提供方(Provider)和服务消费方(Consumer)两个角色。而连接Provider和Consumer的就是注册中心(Registry),Provider把服务暴露在Registry上，而Consumer通过Registry来调用暴露在Registry上面的服务。

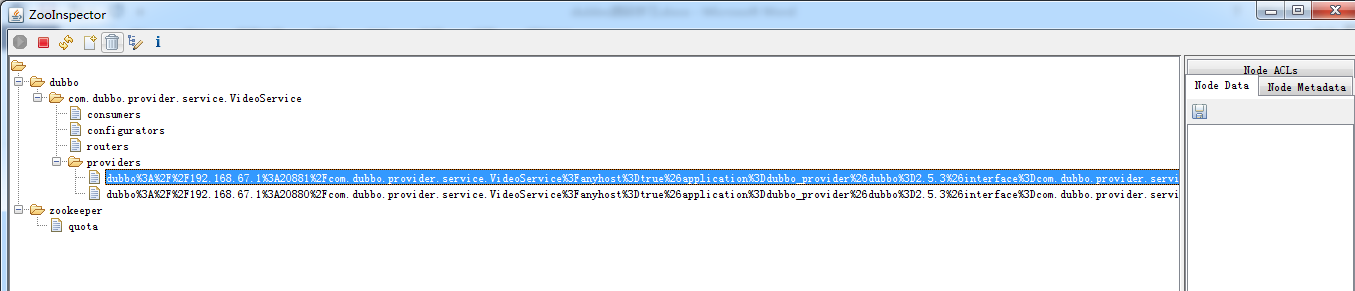
节点角色说明：

* 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: 统计服务的调用次调和调用时间的监控中心。
* Container: 服务运行容器。



所以在provider启动的时候做了以下几件事：

* 连接zookeeper注册中心,用于dubbo的注册中心
* 把服务配置信息以url的形式注册到zookeeper上面
* 创建了一个configuration节点，用于监听订阅admin的对provider的修改



dubbo://192.168.67.1:20881/com.dubbo.provider.service.VideoService?anyhost=true&application=dubbo\_provider&dubbo=2.5.3&interface=com.dubbo.provider.service.VideoService&logger=log4j&methods=query&pid=2288&side=provider&timestamp=1529752754351

通过上面我们对dubbo Provider/Consumer的启动日志以及通过Zookeeper Inspector对zookeeper服务的观察我们可以得到dubbo下面的架构图。

# 二.dubbo源码分析 之 内核SPI实现

Dubbo采用微内核+插件体系，使得设计优雅，扩展性强。那所谓的微内核+插件体系是如何实现的呢！大家是否熟悉spi(service providerinterface)机制，即我们定义了服务接口标准，让厂商去实现（如果不了解spi的请谷歌百度下）, jdk通过ServiceLoader类实现spi机制的服务查找功能 – [Java 规范 SPI](http://blog.csdn.net/u012410733/article/details/52475039)。

### ****1、为什么不使用JDK SPI****

在dubbo中它实现了一套自己的SPI机制。JDK标准的SPI会一次性实例化扩展点所有实现，如果有扩展实现初始化很耗时，但如果没用上也加载，会很浪费资源.

增加了对扩展点IoC和AOP的支持，一个扩展点可以直接setter注入其它扩展点。

### ****2、Dubbo SPI 约定****

SPI文件的存储路径在以下三个文件路径：

* META-INF/dubbo/internal/ **dubbo内部实现的各种扩展都放在了这个目录了**
* META-INF/dubbo/
* META-INF/services/

spi 文件 存储路径在以上三个SPI目录下，并且文件名为接口的全路径名 就是=接口的包名+接口名也就是SPI接口的全路径类名:例如

E:\Project\github\dubbo\dubbo-rpc\dubbo-rpc-default\src\main\resources

\META-INF\dubbo\internal\

com.alibaba.dubbo.rpc.Protocol

* 1
* 2
* 3

每个spi 文件里面的格式定义为： 扩展名=具体的类名，例如

dubbo=com.alibaba.dubbo.rpc.protocol.dubbo.DubboProtoco

* 1
* 2

### ****3、涉及到类与注解简单说明****