Dubbo教程

1. **dubbo**

**概念：**

dubbo就是一个rpc实现框架。Dubbo是阿里巴巴SOA服务化治理方案的核心框架，每天为2,000+个服务提供3,000,000,000+次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点。Dubbo[]是一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案，以及SOA服务治理方案。

**其核心部分包含:**

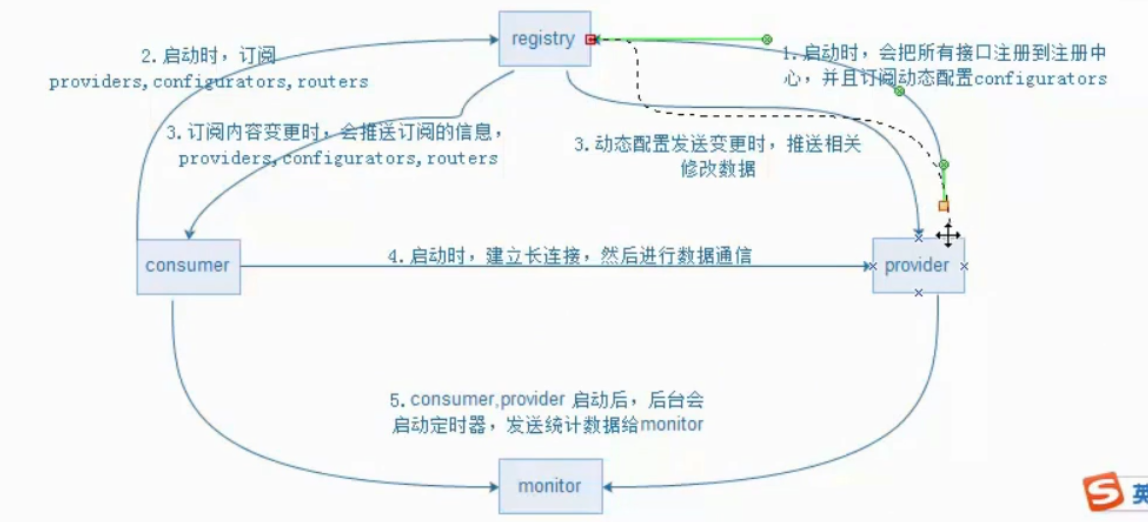
远程通讯: 提供对多种基于长连接的NIO框架抽象封装，包括多种线程模型，序列化，以及“请求-响应”模式的信息交换方式。

集群容错: 提供基于接口方法的透明远程过程调用，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。

自动发现: 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增加或减少机器。

Dubbo支持多种协议，比如dubbo、rmi、http、redis等协议，如果不配置协议的时候，会采用默认协议（单一长连接和NIO异步通讯），默认协议适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况。

**架构初探：**



**原理：源码实现**

**初始化过程细节：**第一步，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。生产者或者消费者初始化的时候，会将Bean对象转会为url格式，将所有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、 直接暴露服务端口

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(](https://link.jianshu.com?t=registry:/registry-host/com.alibaba.dubbo.registry.RegistryService?export=URL.encode()“[dubbo://service-host/com.xxx.TxxService?version=1.0.0](https://link.jianshu.com?t=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(](https://link.jianshu.com?t=registry:/registry-host/com.alibaba.dubbo.registry.RegistryService?refer=URL.encode()“[consumer://consumer-host/com.foo.FooService?version=1.0.0](https://link.jianshu.com?t=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。

分析源代码，基本原理如下：

1. client一个线程调用远程接口，生成一个唯一的ID（比如一段随机字符串，UUID等），Dubbo是使用AtomicLong从0开始累计数字的
2. 将打包的方法调用信息（如调用的接口名称，方法名称，参数值列表等），和处理结果的回调对象callback，全部封装在一起，组成一个对象object
3. 向专门存放调用信息的全局ConcurrentHashMap里面put(ID, object)
4. 将ID和打包的方法调用信息封装成一对象connRequest，使用IoSession.write(connRequest)异步发送出去
5. 当前线程再使用callback的get()方法试图获取远程返回的结果，在get()内部，则使用synchronized获取回调对象callback的锁， 再先检测是否已经获取到结果，如果没有，然后调用callback的wait()方法，释放callback上的锁，让当前线程处于等待状态。
6. 服务端接收到请求并处理后，将结果（此结果中包含了前面的ID，即回传）发送给客户端，客户端socket连接上专门监听消息的线程收到消息，分析结果，取到ID，再从前面的ConcurrentHashMap里面get(ID)，从而找到callback，将方法调用结果设置到callback对象里。
7. 监听线程接着使用synchronized获取回调对象callback的锁（因为前面调用过wait()，那个线程已释放callback的锁了），再notifyAll()，唤醒前面处于等待状态的线程继续执行（callback的get()方法继续执行就能拿到调用结果了），至此，整个过程结束。

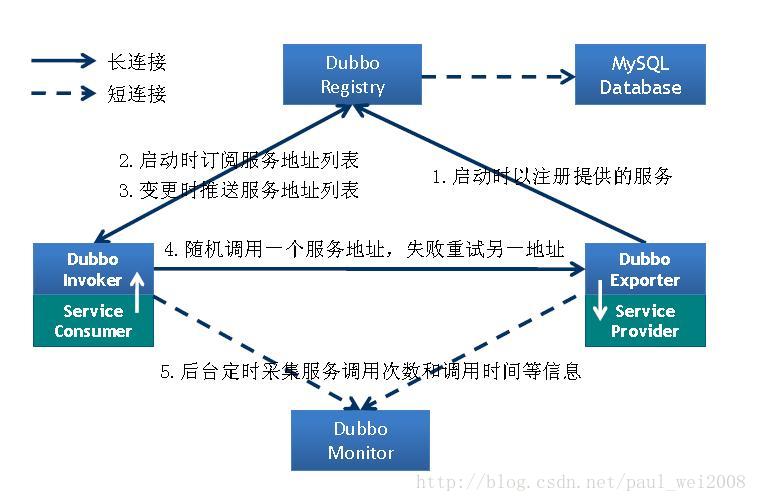
**问题：**

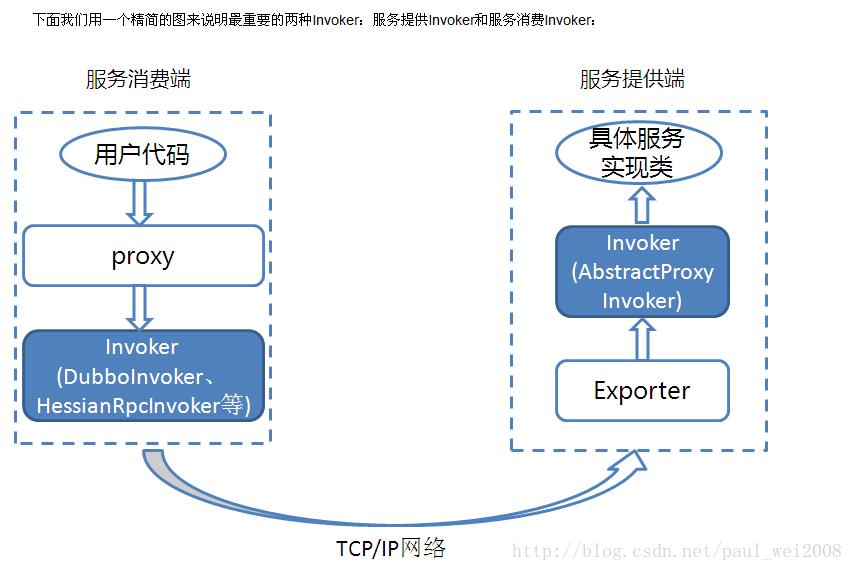
当前线程怎么让它“暂停”，等结果回来后，再向后执行？

答：先生成一个对象obj，在一个全局map里put(ID,obj)存放起来，再用synchronized获取obj锁，再调用obj.wait()让当前线程处于等待状态，释放锁，然后另一消息监听线程等到服务端结果来了后，再map.get(ID)找到obj，再用synchronized获取obj锁，再调用obj.notifyAll()唤醒前面处于等待状态的线程。

Socket通信是一个全双工的方式，如果有多个线程同时进行远程方法调用，这时建立在client server之间的socket连接上会有很多双方发送的消息传递，前后顺序也可能是乱七八糟的，server处理完结果后，将结果消息发送给client，client收到很多消息，怎么知道哪个消息结果是原先哪个线程调用的？

答：使用一个ID，让其唯一，然后传递给服务端，再服务端又回传回来，这样就知道结果是原先哪个线程的了。

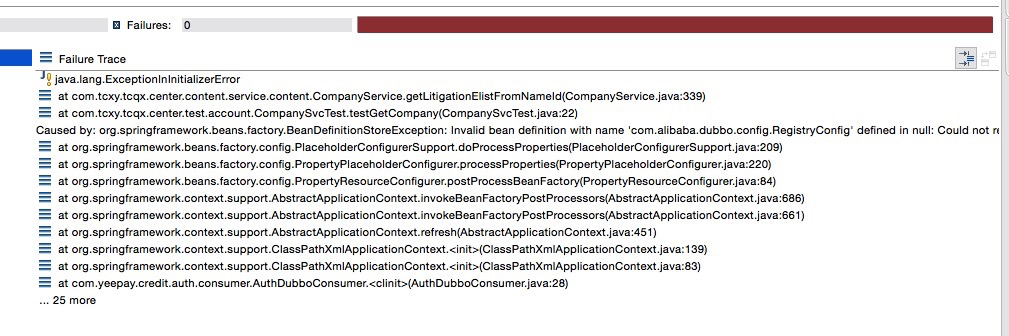




注：dubbo的监控中心是可配置的，不配置也不影响整个应用运行，dubbo的注册中心dubbo自己是没有实现的，目前用的最多的是zookeeper

1. **dubbo跨工程调用**

**问题描述：**如果你的工程会引用到别人的jar包，并且你的工程使用了dubbo框架，你引用的jar包也使用了dubbo框架，这时两者都需要配置dubbo参数。这时你使用引用jar包时无法加载到它的dubbo参数，错误如下：

****

**原因：**因为我们的工程dubbo配置文件叫dubbo.properties,引用的jar包的dubbo配置文件名也叫dubbo.properties,这时系统会默认调用我们的配置文件，而无法加载引用jar的配置文件，所以导致初始化加载异常错误，因为我们的配置文件中不含引用jar所需要的参数。

**解决方法：**

方法1：在我们的dubbo配置文件中加入jar包的配置。此方法不建议使用

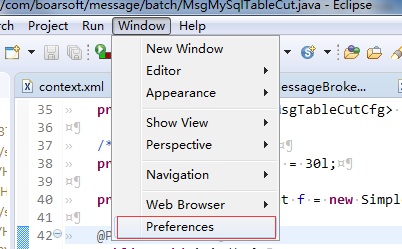
方法2：将我们的配置文件改名，或者引用jar包的配置文件改名（有时改不了引用jar包的配置文件名）。

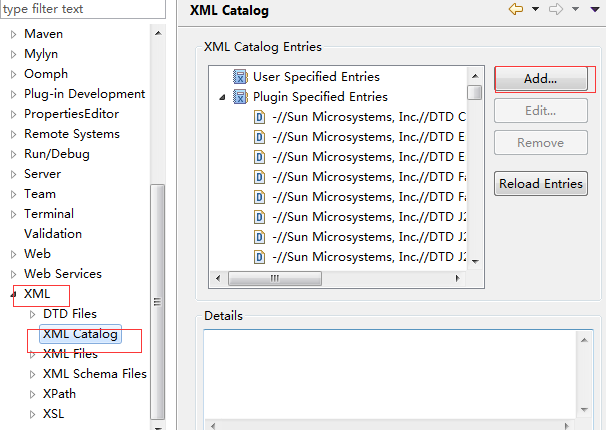
1. **dubbo配置文件报错**

**问题描述：**dubbo-provider.xml文件报错，非配置错误

**解决方法：**

eclipse->偏好设置->xml->xmlcatalog





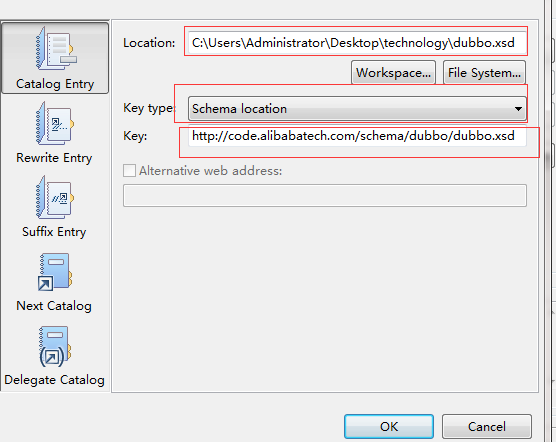
add->catalog entry:

location中用file system 选择刚刚下载的文件路径

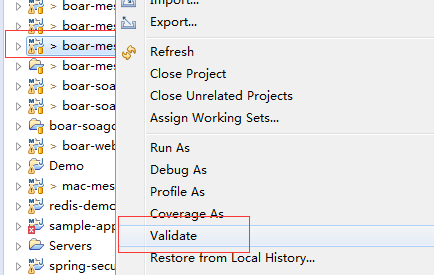
key值http://code.alibabatech.com/schema/dubbo/dubbo.xsd

key type 选择Schema location





保存。。在xml文件右键validate



1. **内核解剖**

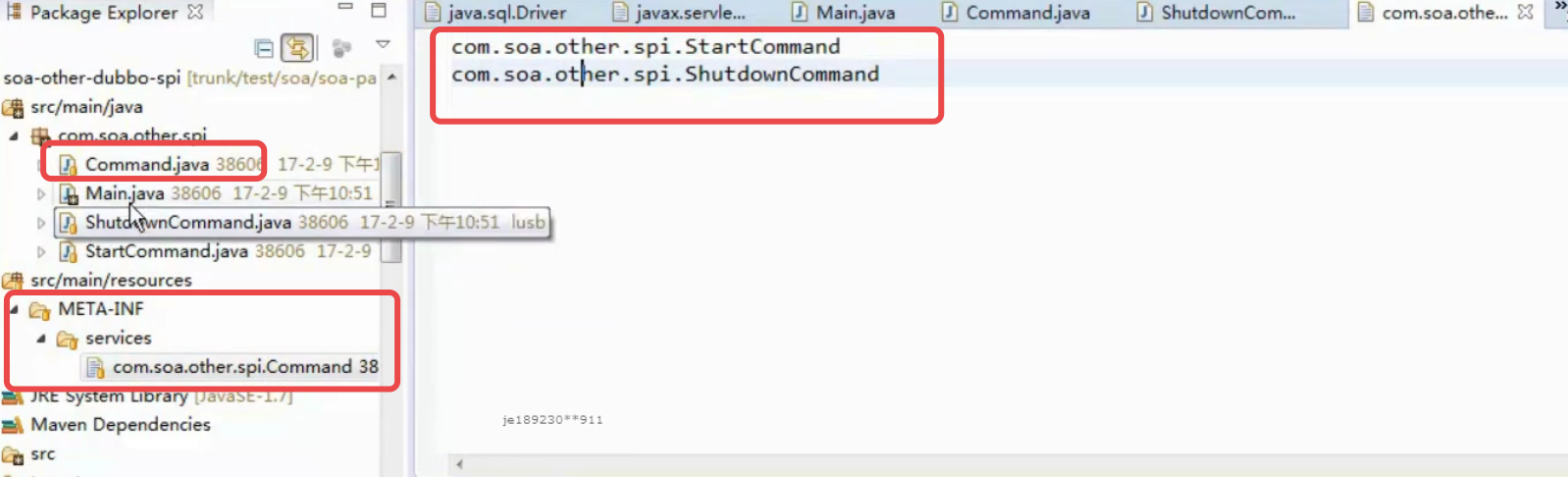
**SPI实现：**

**Java SPI:**

面向对象的设计里，模块之间是基于接口编程，模块之间不对实现类进行硬编码。一旦代码里涉及具体实现类，就违反了可插拔的原则，如果需要替换一种实现，就需要修改代码。为了实现在模块装配的时候，不在模块里面写死代码，这就需要一种服务发现机制，Java spi就是提供这样的一个机制：为某个接口寻找服务实现机制，有点类似IOC的思想，就是将装配的控制权移到代码之外。

SPI的具体约定如下：

当服务的提供者provider提供了一个接口多种实现时，一般会在jar包的META-INF/services/目录下，创建接口的同名文件。该文件里面的内容就是该服务接口的具体实现类的名称。而当外部加载这个模块的时候，就能通过该jar包META-INF/services/里的配置文件得到具体的实现类名，并加载实例化，完成模块的装配。



缺点：

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。

Jdk没有对扩展点IOC和AOP的支持，一个扩展点不可以直接setter注入其它扩展点

**Dubbo SPI：**

Dubbo spi和jdk spi的约定是一样的，在此不加赘述。

Spi文件存储路径在META-INF\dubbo\internal目录下并且文件名为接口的全路径名 就是=接口包名+接口名。

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.DubboProtocol

dubbo解决了jdk上面的两个缺点。

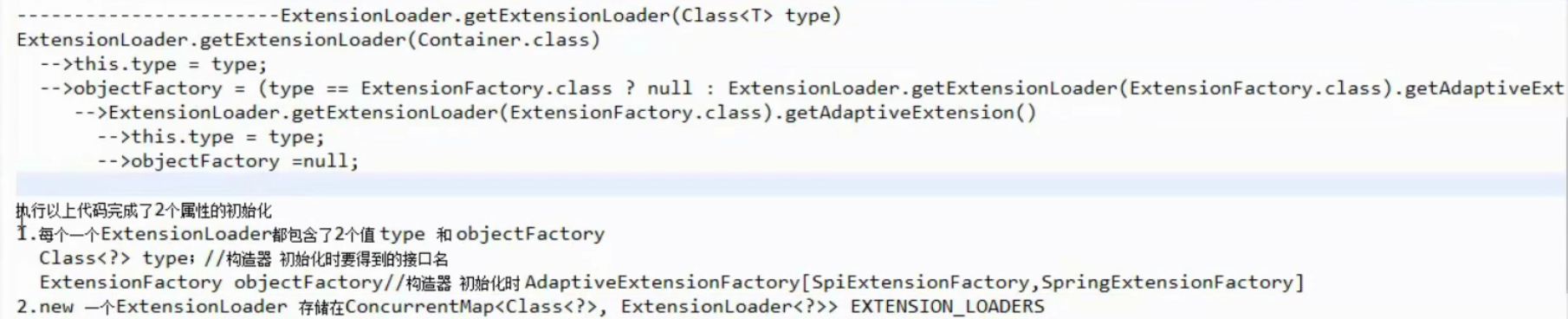
目的：获取一个实现类的对象

途径：通过ExtensionLoader此类的getExtension()方法来获取

实现路径：getExtensionLoader方法就是为该接口new 一个ExtensionLoader，然后缓存起来。

getAdaptiveExtension方法获取一个扩展装饰类的对象，这个类有一个规则，如果它没有一个@Adaptive注解，就动态创建一个装饰类，例如Protocol$ Adaptive对象（动态装饰类：动态编译）。

getExtension方法获取一个对象。









1. **aa**