**-----动态生成类查验**

**在dubbo的源码包里，把**ExtensionLoader的createAdaptiveExtensionClassCode方法改为public

然后直接新建一个类（或者直接在ProtocolAdativeDemo里使用），使用下面的main方法（在源码工程中做）

**public static void** main(String[] args) {  
 ExtensionLoader loader = ExtensionLoader.*getExtensionLoader*(Cluster.**class**);  
 String code = loader.createAdaptiveExtensionClassCode();  
 System.***out***.println(code);  
}

上堂课的作业回顾：

利用消费端filter向生产端隐式传参：

-------加入参数：RpcContext.getContext().setAttachments(rpc\_ip);

-------取出参数：RpcContext.getContext().getAttachment("rpc\_ip")

前面课程回顾

1、rpc的初始化逻辑：

server端：interface+Impl+url =====》创建中转对象，暴露服务---- protocol.export

reference端：interface+url =======》创建代理对象，引入服务---- protocol.refer

2、dubbo初始化过程

dubbonamespace扫描标签---》dubbobeanDefinitionParser解析标签配置

---》ServiceBean + ReferenceBean ===》ptotocol.export + refprotocol.refer

3、dubbo的spi目标：

META-INF/dubbo.internal ======> key1=impl1, key2=impl2

xml文件里配置 =============》property = key1/key2

4、dubbo的spi机制

ExtensionLoader + 接口 interface ======》接口代理对象 proxy

**proxy + URL =================》extName ======》key1 = impl**

5、spi扩展实战

interface ======》impl =========》 key + impl

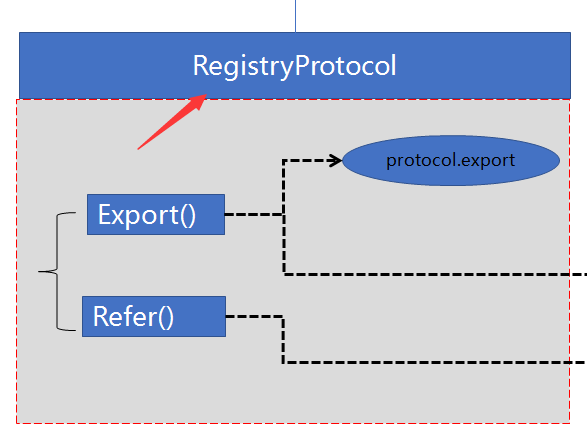
META-INF/dubbo.internal ======> key1=impl1, key2=impl2

xml配置 ====》property = key1

本堂课内容

1、dubbo的发布订阅：

---------使用RegistryProtocol，将发布订阅模块作为一种协议，作为protocol的扩展



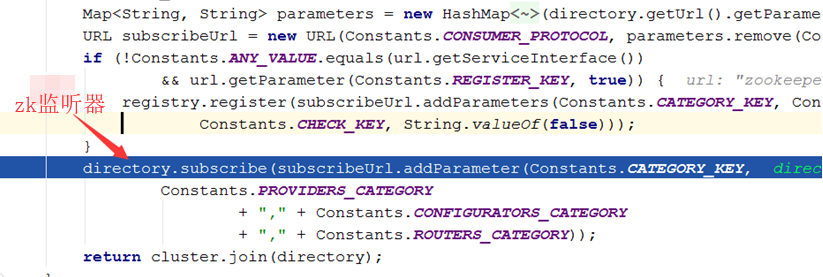
2、protocol工作的关键点是：根据URL得到extName，进而对应到实际的协议类



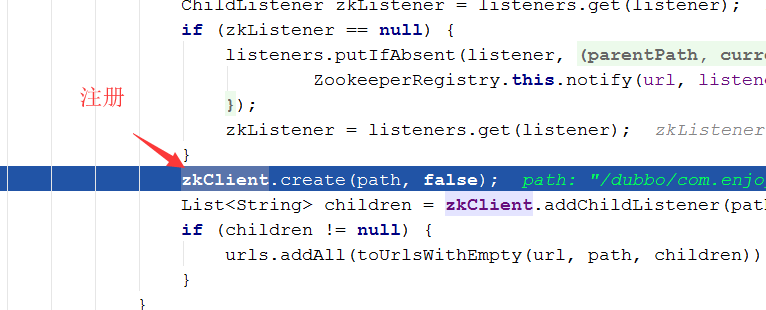
2.1、dubbo消费端启动过程中，第一次在serviceconfig中执行refprotocol.refer,此时对应的url是registryUrl



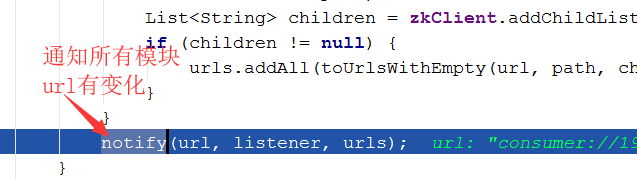
2.2、进入registryProtocol执行监听器的zk注册



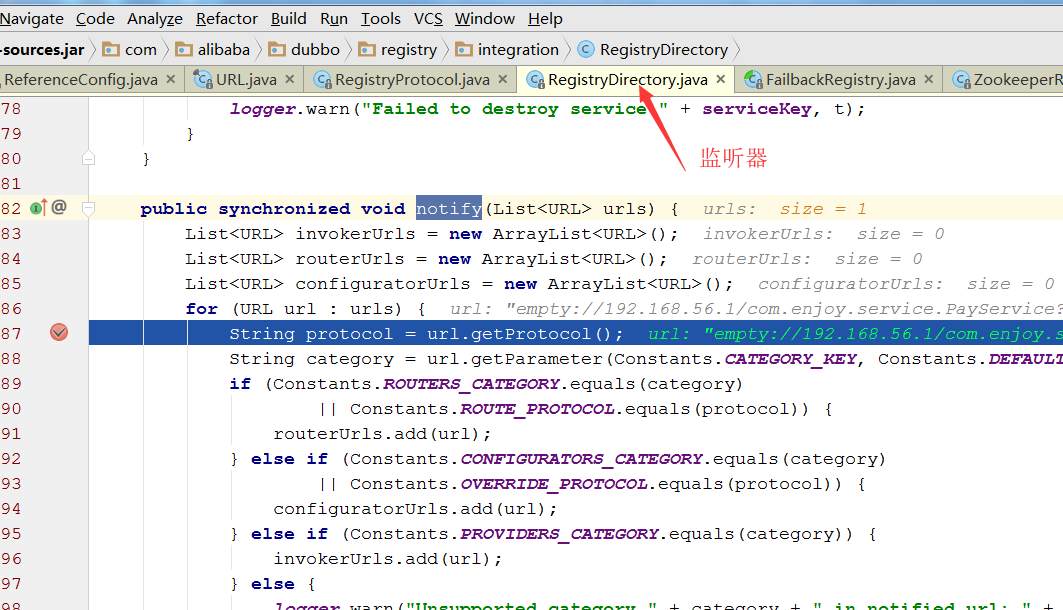
2.3、进入ZookeeperRegistry中，进行zk节点创建（注册）



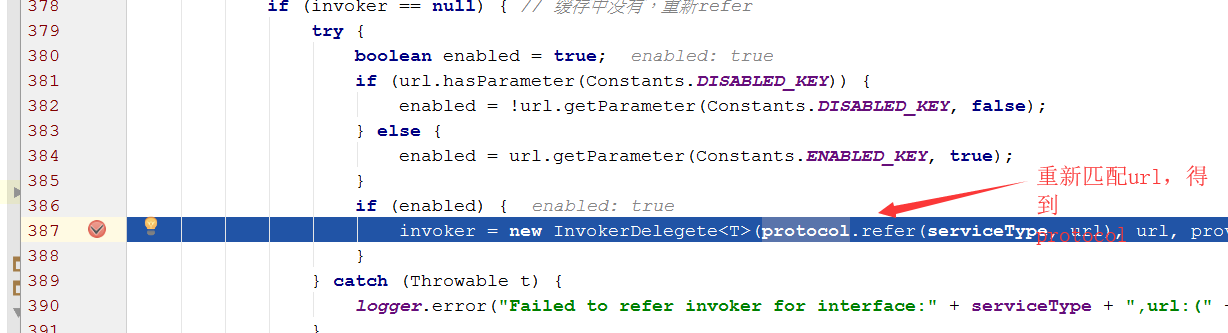
2.4、注册完毕之后，进行消息通知



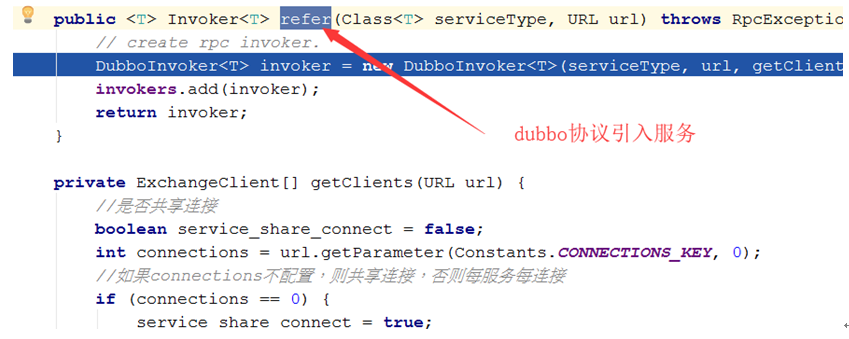
2.5、消息通知，是调用监听器来刷新缓存



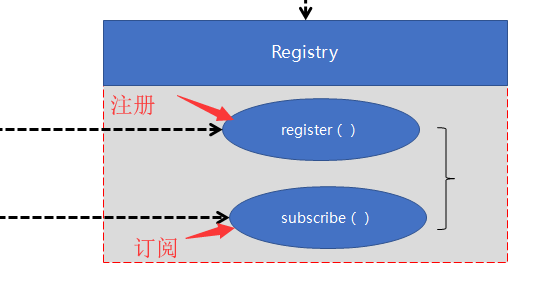
2.6、监听器最后，调用toInvokers方法，将新注册的url转为invokers对象，此处url转为invokers的过程，交由dubboprotocol来完成，即：**protocol**.refer的url在此时是dubbo协议（SPI机制）



2.7、dubboprotocol执行url服务的引入动作

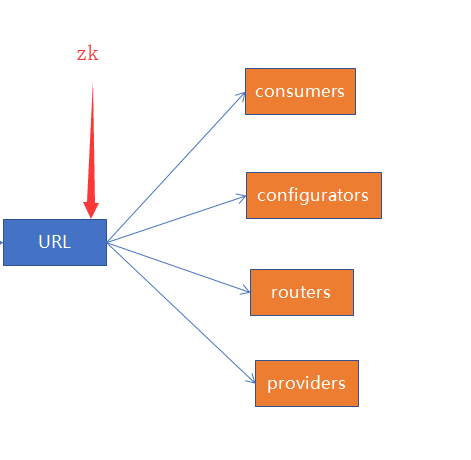


3.1、dubbo注册模块的逻辑集中类

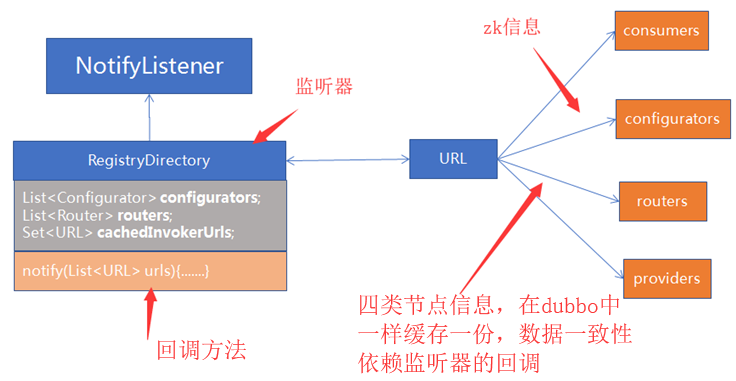


3.2、当有新的消费节点或者服务节点进入时，触发notify----》刷新所有的url信息

（图为zk中存储的信息）

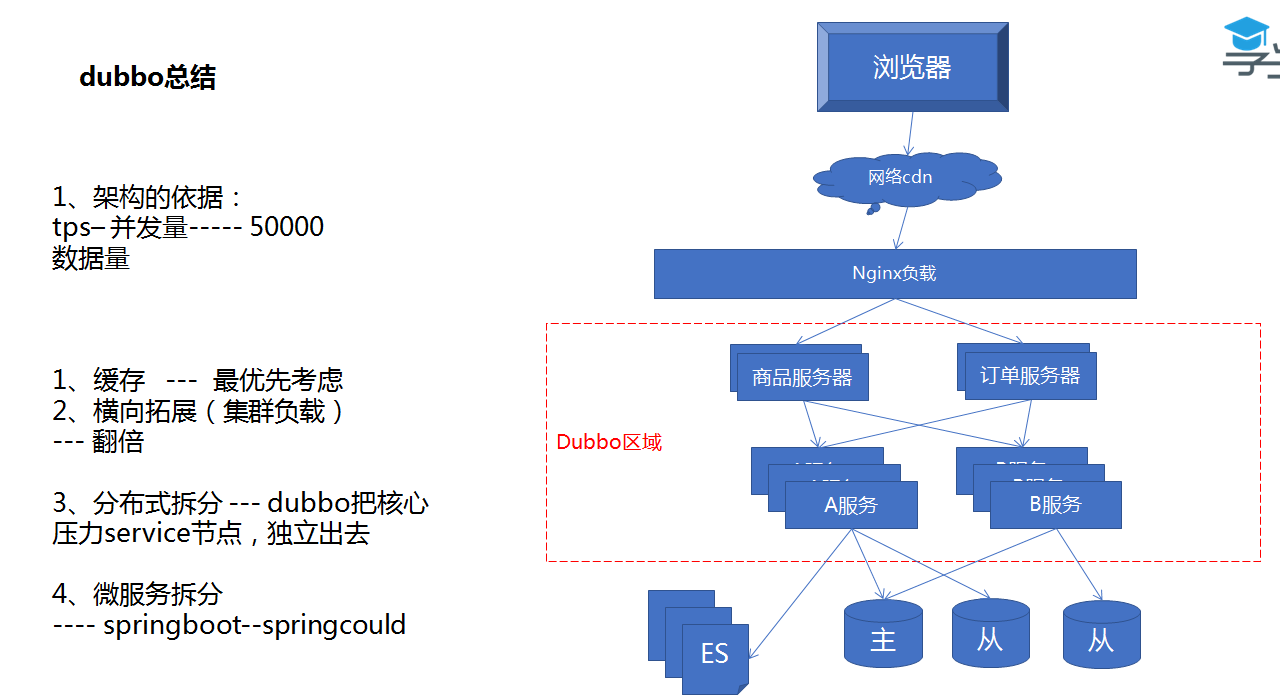


3.3、zk中的信息，在dubbo代码中，一样要缓存一份（查询会消耗时间），用于容错/负载等动作，此缓存信息的如何与zk中信息保持一致性，依赖发布订阅机制（监听器的及时回调）

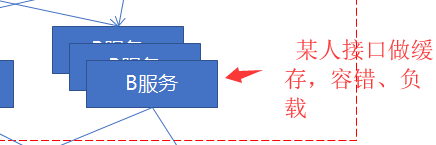


3.4、dubbo注册中心选择很多（数据库功能+发布订阅功能），之所以推荐使用zk，是因为zk在分布式环境中，搭建集群（数据冗余）方便，抗灾性强

4.1、架构顶层策略



4.2、dubbo在局部使用场地



4.3、dubbo用法梳理

4.3.1、使用标签：

reference标签：

service标签：服务端的代码配置---- 原则一定要是一样的

属性---- 继承默认属性 --- 漫游覆盖

服务端：A服务 ----- 三台五台 -----------------代码是一样的

消费端：B服务/C/D/E ---------------------------引用配置不一样

只在消费端使用的------loadbalance/cluster/cache/async/onreturn/generic

4.3.2、参考官方文档：

http://dubbo.apache.org/zh-cn/docs/user/references/xml/dubbo-service.html

4.4、dubbo初始化流程走一遍

4.4.1、入口

spring标签DubboNameSpaceHandler--------》referencebean + servicebean

-----------------》继承了spring initalizingBean接口 ------》afterPropertiesSet

----------------》protocol.refer（url） + protocol.export（url） +（registryURL）

4.4.2、消费端

----------------》RegistryProtocol.refer ----------------》zkRegistry.subscribe

触发监听器回调 --------》registryDictory ----------》 protocol.refer

------------》dubboProtocol.refer + interface-----------》消费方interface代理对象

4.4.3、服务端

----------------》RegistryProtocol.export ------------------> protocol.export -------> 创建中转对象

-----------------> zkRegistry.subscribe 注册

5、答疑

服务方的中转对象，一旦建立，就可以被调用------------ 如：泛化调用（没有代理对象）

5.1、spi框架，为什么要有那个自编码代理对象？

答：spi里不使用代理对象，也可以 ----- 参见loadbalance,就没有使用

使用代理对象，方便在编码时，通过改变url值，来切换实现对象

5.2、生成动态代码：

参见文档开头的demo

5.3、对于spi的扩展加载器，可以直接使用它的extName取目标类的方式

Protocol dubboProtocol = (Protocol)loader.getExtension(extName);

5.4、dubbo的动态源码生成，使用工具**Javassit**工具

5.5、面试问spi ------ 思想表达清楚（参照jdk的SPI）

------ jdk的spi：把实现类，装入一个list中

------dubbo的spi：把实现类，装入一个map中（配置文件指定一key），key指代实现类

5.6、RegistryProtocol做发布订阅

dubbo偷懒，借用protocol流程模块，把注册模块，伪装成protocol协议------ 注册协议

注册协议跟其它不一样，任何协议dubbo/rmi/rest。。。。要生效，都必须注册，

因此RegistryProtocol是最优先被调用，然后再转给真实的协议

5.7、mq是面向业务的，发的消息都是业务消息。Mq的消息，没有数据存储功能。

5.8、dubbo只是服务治理，springcloud包含所有全套微服务功能

dubbo把rpc做了成透明化的调度------ 减少出错机率

springcloud的远程调用，没能完全透明化。

透明化调度，意思你使用它，像本地服务一样用。

5.9、dubbo控制台，可以关闭某个服务，不对外开放（开发环境常用）

5.10、关于dubbo里协议的默认，很多同学容易混淆，看下图说明

