Dubbo(2.5.4)源码分析

1. 源码分析之Dubbo Extension扩展点

实现插件化的动态扩展。

JDK的SPI实现插拔式扩展，例如JDK驱动。

SPI(Service Provider Interface)->Extension

ExtensionLoader.getExtensionLoader().getAdaptiveExtension();//动态创建适配器扩展点:Protocol$Adaptive

执行流程：

getExtensionLoader

getAdaptiveExtension(获得自适应扩展点)

createAdaptiveExtension

injectExtension(依赖注入扩展点)

getAdaptiveExtensionClass.neInstance(自适应扩展点实例)

getExtensionClasses

@SPI ：value包含默认扩展点

@Adaptive：加载在类和方法上。

方法：动态创建自适应的适配器

类：直接加载当前适配器

loadExtensionClasses

加载配置文件中的扩展点包括：

1. META-INFO/dubbo
2. META-INFO/services
3. META-INFO/dubbo/internal

createAdaptiveExtensionClass（动态创建适配器扩展点）//如：Protocol$Adpative

1. 生成字节码代码

createAdaptiveExtensionClassCode

1. 动态编译字节码

Compiler.compile

injectExtension 作用注入扩展点，如：

|  |
| --- |
| **public class** RegistryProtocol **implements** Protocol {  **private** Cluster **cluster**;    **public void** setCluster(Cluster cluster) {  **this**.**cluster** = cluster;  }    **private** Protocol **protocol**;    **public void** setProtocol(Protocol protocol) { //注入扩展点  **this**.**protocol** = protocol;  }  …  } |

Transporter

RegistryFactory

基于方法的扩展点

1. 源码分析之服务发布及注册流程

Spring 提供扩展标签，扩展点：

NamespaceHandler

BeanDefinitionParse

Dubbo 扩展标签及Bean定义解析

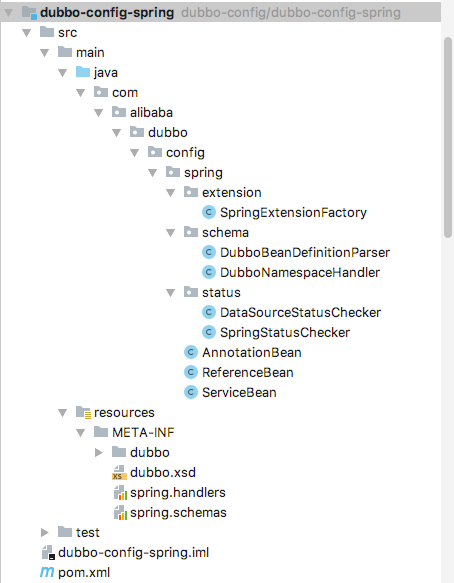
com.alibaba.dubbo.config.spring.schema

DubboNamespaceHandler

DubboBeanDefinitionParser

dubbo-spring-config下：

|  |
| --- |
| /META-INF/spring.handlers  **http\://.alibabatech.com/schema/dubbo**=**com.alibaba.dubbo.config.spring.schema.DubboNamespaceHandlercode**  /META-INF/spring.schemas  **http\://code.alibabatech.com/schema/dubbo/dubbo.xsd**=**META-INF/dubbo.xsd** |



ServiceBean实现服务本地发布及注册。

实现Initial

ReferenceBean 客户端在启动的时候自动订阅发现服务

WrapperClass

ProtocolListenerWrapper

ProtocolFilterWrapper

ProtocolFilterWrapper(ProtocolListenerWrapper(DubboProtocol))

服务启动（发布及注册）流程：

1. 服务本地发布，默认采用netty通信服务
2. 服务注册，默认采用zookeeper

ServiceBean.afterPropertiesSet完成ProtocolConfig、ProviderConfig、ApplicationConfig、ModuleConfig、

RegistryConfig、MonitorConfig、ProtocolConfig等参数初始化。并调用export()完成服务启动。

Invoke-执行远程调用

Protocol-服务地址发布和注册

Exporter-暴露服务和取消服务

dubbo-rpc-api  的META-INF/dubbo/inernal/com.alibaba.dubbo.rpc.Protocol 的包装类

filter=com.alibaba.dubbo.rpc.protocol.ProtocolFilterWrapper  
 listener=com.alibaba.dubbo.rpc.protocol.ProtocolListenerWrapper

如无注册中心配置直接走DubboProtocol.export()。

Invoker 执行远程服务

|  |
| --- |
| ***ProxyFactory$Adaptive*** 🡪JavassistProxyFactory(动态字节码代理)  Invoker<?> invoker = ***proxyFactory***.getInvoker(**ref**, (Class) **interfaceClass**, registryURL.addParameterAndEncoded(Constants.***EXPORT\_KEY***, url.toFullString())); |

1. 源码分析之消费端初始化流程
2. 源码分析之服务调用过程
3. 源码分析之什么是Directory

Directory

StaticDirectory：静态目录

RegistryDirectory：动态注册目录Watcher监听远程协议地址变化自动刷新本地协议地址缓存。RegistryDirectory.notify:客户端监听Zookeeper远程服务节点发生变化触发该方法。刷新本地目录(/configurators、/routers、/providers)缓存及invoker列表。

1. 源码分析之什么是Cluster
2. 源码分析之LoadBalance

LoadBalance 客户端请求负载均衡。何时触发：

InvokerInvocationHandler MockClusterInvoker AbstractClusterInvoker

FailoverClusterInvoker

AbstractClusterInvoker 调用invoker方法时生成LoadBalance，默认为RandomLoadBalance(随机

负载均衡)。

LoadBalance类型：

RandomLoadBalance：随机加权重

如：invoker个数4，即：1、2、3、4。总权重为：1+2+3+4=10

ConsistentHashLoadBalance：一致性Hash负载均衡

LeastActiveLoadBalance：最小活跃数负载均衡

RoundRobinLoadBalance：轮询负载均衡

Dubbo设计模式：

适配器、装饰、工厂、模版方法、动态代理(javasist)、责任链等设计模式。

1. Dubbo服务化最佳实践
2. 服务分包

service-api：服务接口、服务类型及异常。考虑在API中放置一份spring的引用配置，配置建议放在模块的目录下如：com/alibaba/china/xxx/dubbo-reference.xml

service-

1. 粒度

dubbo不支持分布式事物，服务接口按功能划分。

1. 版本

每个接口都需要定义版本（version）,为后续不兼容升级提供可能。

<dubbo:service interface=”” version=”1.0.0”/>

1. 兼容性
2. 枚举值
3. 序列化

服务参数及返回值建议使用POJO对象，即通过setter、getter方法表示属性的对象。

1. 异常
2. 推荐用法
3. 在Provider上尽量多配置Consumer端的属性。如：timeout、 retry、loadbalance等。<dubbo:service timeout=”” retry=”” loadbalance=”” />
4. Provider上配置合理的Provider端属性。

Provider属性：

threads：服务线程池大小

executes：并发数。一个服务提供者可以执行的服务请求上限。即当provider对一个服务的并发请求超过了该上限，新调用会wait，此时Consumer端会超时。

1. 配置管理信息

application、provider、consumer的owner属性。

1. 配置Dubbo缓存文件

<dubbo:registry file=”” />

给文件会缓存注册中心和服务提供者列表。

1. 监控配置
2. 使用固定接口暴露服务。
3. 使用Dragoon的http监控项监控注册中心上的服务提供方。
4. 服务提供方，使用Dragoon的telnet或shell监控项。
5. 服务消费方，通过将服务强制转换为EchoService，并调用$echo()，测试该服务提供者是可用的。
6. 建议使用xml配置方式
7. 总结
8. dubbo的发展和使用
9. 基于dubbo的RPC调用
10. dubbo+zookeeper的实现原理
11. dubbo的常用配置（多协议、多注册中心、启动时检查、重试、Loadbalance）
12. dubbo的容器支持
13. SPI和Dubbo的ExtensionLoader
14. ExtensionLoader源码分析
15. Dubbo的服务发布
16. Dubbo消费端初始化
17. 什么是invoker？
18. 消费端的消费调用链
19. 服务端的处理消息的调用链
20. 什么是Directory、Loadbalance、Cluster？