# 概述

# 知识点

## Spring Schema

参照Demo学习

## Dubbo SPI

参考官网样例

<http://dubbo.apache.org/zh-cn/blog/introduction-to-dubbo-spi.html>

博客

<https://cxis.me/2017/02/18/Dubbo%E4%B8%ADSPI%E6%89%A9%E5%B1%95%E6%9C%BA%E5%88%B6%E8%AF%A6%E8%A7%A3/>

### 概念

|  |  |
| --- | --- |
| 概念 | 描述 |
| 扩展点 | 被注解SPI标注的接口 |
| 扩展 | 扩展点的具体实现 |
| 扩展实例 | 某个扩展的具体实例（已经在内存被创建） |
| 扩展自适应实例 | 一种特殊的扩展实例，其扩展的具体实现是通过自动生成的。 |
| 扩展别名 | 一个扩展的别名，位于配置文件中 |
| ExtensionLoader | dubbo扩展机制的“调度中心”，类似于JDK SPI的ServiceLoader，负责扩展的加载和生命周期 |
| @SPI | 标注注解，用于指定扩展点，没有标注该注解的接口被ExtensionLoader加载会报异常 |
| @Adaptive |  |

### 原理

原理参照Demo逐步调试代码。

## Proxy

### 生产者代理

### 消费者代理

重点代码在ReferenceConfig的Init方法，调用createProxy方法。

生成proxy0类是调用Proxy类中的getProxy方法

# 服务发布过程

1. spring 加载ServiceBean时，会调用对应的export方法将服务暴露。

# 服务订阅过程

1. Spring加载<dubbo:reference>标签

Spring加载<dubbo:reference>标签时会创建一个代理对象。代理对象的创建过程在ReferenceBean的afterPropertiesSet方法。

# 服务提供者

## 线程相关

1. NettyServerWorker线程监听到请求数据，将数据提交给DubboServerHandler线程池

## 总结

dubbo服务提供者是通过netty监听网络请求，当服务调用请求到达时，netty会将请求向上传递，指导调用dubbo封装的nettyserver。

1. netty监听网络请求
2. netty会调用dubbo实现的decode
3. netty将请求传给dubbo，调用com.alibaba.dubbo.remoting.transport.netty4. NettyServerHandler#channelRead方法。
4. 经过层层上传，最终会将请求放到dubbo业务调度线程池。com.alibaba.dubbo.remoting.transport.dispatcher.all. AllChannelHandler#received方法，线程池的任务是ChannelEventRunnable。
5. ChannelEventRunnable的run方法会根据任务状态执行不同操作。在此处有个代码性能优化，因为run方法执行最多的状态是RECEIVED（也就是有90%（瞎说的比例）的run方法调用会执行这个状态），所以把这个状态放到if代码，其他状态通过switch来处理，这样可以提供性能。
6. 调用DecodeHandler的再次判断是否需要解码操作

## 服务提供者Proxy创建时机

Dubbo的代理是通过spi来扩展的，扩展点事ProxyFactory，默认是采用javassit来实现动态代理，对应的扩展实现是JavassistProxyFactory，获取方法是getInvoker，因此打个断点便可知道服务提供者proxy创建时机。

为啥代理方式没有cglib？

因为cglib是通过类继承来实现动态代理，通过dubbo实现的远程调用中一般都是通过接口来实现服务的，因此没有cglib。

### javassit代理实现

这种代理实现中，服务提供者proxy涉及三类对象

* 第一个是通过服务接口的实现对象realObject，通过spring配置文件配置或者自动扫描生成；
* Wrapper类对象wrapper，该类的invokeMethod方法会获取到realObject方法，根据服务方法名字调用对应的方法。wrapper对象存放在Wrapper的static map中缓存起来。
* AbstractProxyInvoker子类对象，invoker对象包含一个wrapper对象wrapper。

因此当发起服务调用时，dubbo首先根据接口名获取对应的invoker对象，调用invoke方法，invoke方法会触发wrapper调用invokemethod方法，然后调用realobject中的服务方法。

### jdk代理实现

通过在标签中加入<dubbo:service proxy="jdk" />来修改代理方式。

这种方式的代理主要通过AbstractProxyInvoker子类对象来完成，该子类对象的doInvoke方法通过反射调用realobject的服务方法。

# 服务消费全过程

参考官网<https://dubbo.apache.org/zh-cn/docs/source_code_guide/service-invoking-process.html>

自我总结技术点

1. Spring加载dubbo bean（理解Spring schema）
2. 通过spring获取对应的consumer代理对象（代理对象何时生成，代理如何实现）？

代理类是在ReferenceConfig的init方法中创建的，getBean的时候创建对应的代理对象。

1. 代理对象调用接口方法；
2. 调用抽象的Invoker，通过LoadBalance筛选provider（前提是多个提供者），此处掌握负载均衡算法
3. 通过具体的invoker（负载均衡选出来的）调用
4. u

# 集群Cluster

Dubbo提供多种Cluster，可以查看spi配置文件com.alibaba.dubbo.rpc.cluster.Cluster中的别名。

## Mock

这种Cluster是为Consumer设置的，主要是用于服务降级。如果远程调用Provider失败，并且Consumer设置了mock属性，则会降级调用Consumer本地的Mock服务。

Mock枚举值：

* false：不使用mock服务，如果远程调用失败则反馈上游系统失败；
* true：如果远程调用失败，则调用本地mock服务；
* default，通true含义；
* force：强制使用mock服务，不调用远程服务。

注意事项：

* Mock服务的包名必须和Provider的接口所在包名一样；
* Mock服务的类名必须是接口名+Mock

举例，如果接口包名和类名是：com.lc.api.provider.Provider，则mock服务的全限定名是com.lc.api.provider.ProvideMock.

获取Mock服务实例对象是在MockInvoker#getMockObject方法，其代码如下：



通过代码可以看出，Mock服务必须包含一个缺省构造方法（default constructor）。

## Failover

故障转移（failover，字面意思就是失败则跳过）。Dubbo默认的集群模式。该模式的核心代码是FailoverClusterInvoker#doInvoke方法。

结合代码可以知道doInvoke的逻辑如下：

1. 获取重试次数retry，默认重试2次，则总的是len=retry+1；
2. for i=0->len
   1. 负载均衡选择最合适的invoker（从注册中心订阅所有的provider）
   2. 调用invoker的invoke方法
      1. 成功，返回结果
      2. 不成功，判断是否超过次数，否则继续重试

上面的逻辑比较简单，容易理解，需要掌握的两个点

* 注册中心的Provider是如何订阅到的？
* 被负载均衡选中的invoker是如何生成的

# 负载均衡

Dubbo提供四种负载均衡算法。分别是：

* 加权随机负载均衡
* 加权最小活跃数负载均衡
* 一致性hash负载均衡
* 加权轮训负载均衡

## RandomLoadBalance

算法核心思想是累加各个接口的权重，然后随机生成一个值，判断该值所处区间。算法伪代码：

for i=0:N

sum+=weight[i];

random = random(sum);//生成0到sum区间的随机数

for i=0:N

if(random-weight[i]<=0)

break;

return node[i];

## LeastActiveLoadBalance

Dubbo会为每个服务提供者统计当前服务调用数，当一个请求到达，该服务提供者的活跃数+1，请求结束后活跃数-1，LeastActiveLoadBalance就是依照活跃判断该使用哪个服务提供者。加权是为了解决如果有多个相同的最小活跃数，则在这部分提供者中执行加权RandomLoadBalance。

## RoundRobinLoadBalance

没有加权的轮询是按照所有候选者逐个选中并使用，如果候选者存在差异，可以为性能高的候选者赋更高的权重，性能较差的候选者赋较小的权重。Dubbo中的加权RoundRobinLoadBalance是在轮询的基础上增加权重来筛选候选者，算法的具体执行过程可以参照如下例子：

假设三个候选者A,B,C的权重分别为3,5,2，权重构成一个序列[3,5,2]，同时总权重为10：

Round 1：选择A，同时A的剩余权重减1，此时序列变为了[2,5,2];

Round 2：选择B，同时B的剩余权重减1，此时序列变为了[2,4,2];

Round 3：选择C，同时C的剩余权重减1，此时序列变为了[2,4,1];

Round 4：选择A，同时A的剩余权重减1，此时序列变为了[1,4,1];

Round 5：选择B，同时B的剩余权重减1，此时序列变为了[1,3,1];

Round 6：选择C，同时C的剩余权重减1，此时序列变为了[1,3,0];

Round 7：选择A，同时A的剩余权重减1，此时序列变为了[0,3,0];

Round 8：选择B，同时B的剩余权重减1，此时序列变为了[0,2,0];

Round 9：选择B，同时B的剩余权重减1，此时序列变为了[0,1,0];

Round 10：选择B，同时B的剩余权重减1，此时序列变为了[0,0,0]

经过10轮，A被选中了3次，B被选中5次，C被选中2次。如果按照一般的轮询，10轮会选择A4次，B3次，C3次。

# 序列化和反序列化

Dubbo中处理序列化的地方时DubboCodec和ExchangeCodec两个类中。序列化的具体实现在package com.alibaba.dubbo.common.serialize中。

序列化是将请求/响应的header和body序列成可持久化存储或者网络传输的数据；

反序列就是和序列化相反的过程。

序列化的具体实现类采用的spi（类似于工厂模式，通过传入的命令选择具体的实现），默认是hessian2，可以在配置文件中修改序列化方式

<dubbo:protocol name="dubbo" port="20881" serialization="fastjson"/>，比如这个配置就采用的fastjson进行序列化。

现有的序列化方式中，hessian2是性能最好和最可靠的序列化方式，dubbo官方也指出后续会增加更高效更可靠的序列方式fst，kryo等。

参考文档：

<https://dubbo.apache.org/zh-cn/docs/user/serialization.html>

# 模块

## 注册

参考：<https://juejin.im/post/5d5cc544e51d4561df7805a9>

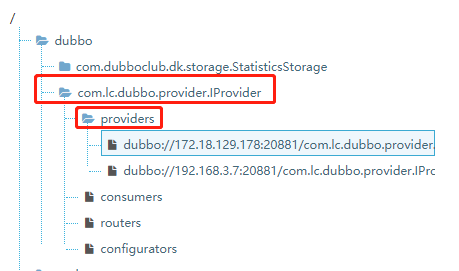
关键字：

AbstractRegistry—内存缓存和文件缓存，文件名格式dubbo-registry-${application.name}-${register\_address}.cache

FailbackRegistry—失败重试，重试机制就是一个定时线程池不断遍历配置的注册中心地址进行注册

### Zookeeper

在学习以Zookeeper作为注册中心之前，首先以一个例子开头，以此对zookeeper有个直观感受。同一个服务提供者，部署在两台服务器上并以同一个zookeeper作为注册中心时，zookeeper的树形结构如下所示：



可以看出dubbo提供中在注册服务时，提供的内网ip，不会提供公网ip。

# 参考

官网https://dubbo.incubator.apache.org/

API官网<https://dubbo.incubator.apache.org/books/dubbo-user-book-en/preface/>

Dubbo介绍<http://www.baeldung.com/dubbo>

Zookeeper简介<https://www.w3cschool.cn/zookeeper/zookeeper_installation.html>

博客<https://juejin.im/post/5bb18c8e5188255c6140de76>

<https://segmentfault.com/a/1190000016741532>（推荐）