**Dubbo 调用模块详解**

**课程概要：**

一、Dubbo 调用模块详解（容错、负载均衡、异步调用、过滤器）

二 、Dubbo 其它使用场景详解（泛化调用与引用、隐示传参、令牌验证）

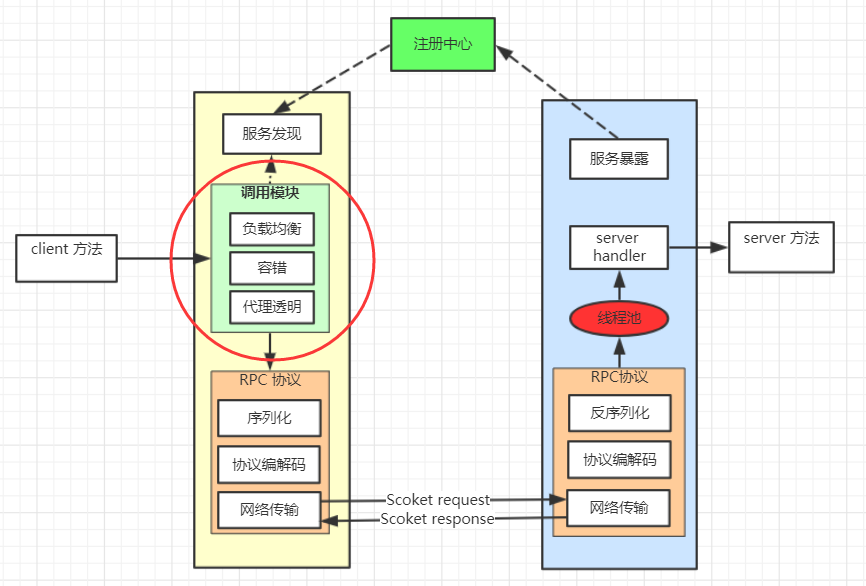
三、Dubbo 路由功能使用

### **一、Dubbo 调用模块详解（容错、负载均衡、异步调用、过滤器）**

**Dubbo调用模块概述：**

dubbo调用模块核心功能是发起一个远程方法的调用并顺利拿到返回结果，其体系组成如下：

1. 透明代理：通过动态代理技术，屏蔽远程调用细节以提高编程友好性。
2. 负载均衡：当有多个提供者是，如何选择哪个进行调用的负载算法。
3. 容错机制：当服务调用失败时采取的策略
4. 调用方式：支持同步调用、异步调用
5. 结果获取：指同步等待结果返回，还是异步通过回调通知获取结果。



**负载均衡**

Dubbo 目前官方支持以下负载均衡策略：

1. **随机**(random)：按权重设置随机概率。此为默认算法.
2. **轮循** (roundrobin):按公约后的权重设置轮循比率。
3. **最少活跃调用数**(leastactive):相同活跃数的随机，活跃数指调用前后计数差。
4. **一致性Hash**(consistenthash ):相同的参数总是发到同一台机器

设置方式支持如下四种方式设置，优先级由低至高

<!-- 服务端级别-->

<dubbo:service interface="..." loadbalance="roundrobin" />

<!-- 客户端级别-->

<dubbo:reference interface="..." loadbalance="roundrobin" />

<!-- 服务端方法级别-->

<dubbo:service interface="...">

<dubbo:method name="..." loadbalance="roundrobin"/>

</dubbo:service>

<!-- 客户端方法级别-->

<dubbo:reference interface="...">

<dubbo:method name="..." loadbalance="roundrobin"/>

</dubbo:reference>

#TODO 一至性hash 演示

配置loadbalance

配置需要hash 的参数与虚拟节点数

发起远程调用

一至性hash 算法详解：

**容错**

Dubbo 官方目前支持以下容错策略：

1. **失败自动切换：**调用失败后基于retries=“2” 属性重试其它服务器
2. **快速失败：**快速失败，只发起一次调用，失败立即报错。
3. **勿略失败：**失败后勿略，不抛出异常给客户端。
4. **失败重试：**失败自动恢复，后台记录失败请求，定时重发。通常用于消息通知操作，只会调用一次
5. **并行调用:** 只要一个成功即返回，并行调用指定数量机器，可通过 forks="2" 来设置最大并行数。
6. **广播调用：**广播调用所有提供者，逐个调用，任意一台报错则报错

设置方式支持如下两种方式设置，优先级由低至高

<!--

Failover 失败自动切换 retries="1" 切换次数

Failfast 快速失败

Failsafe 勿略失败

Failback 失败重试，5秒后仅重试一次

Forking 并行调用 forks="2" 最大并行数

Broadcast 广播调用

-->

<dubbo:service interface="..." cluster="broadcast" />

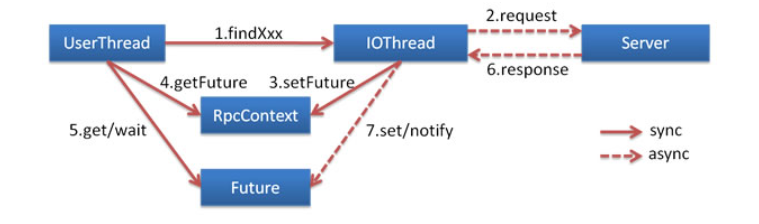
<dubbo:reference interface="..." cluster="broadcast"/ >

**异步调用**

异步调用是指发起远程调用之后获取结果的方式。

1. 同步等待结果返回
2. 异步等待结果返回
3. 不需要返回结果

Dubbo 中关于异步等待结果返回的实现流程如下图：



异步调用配置:

<dubbo:reference id="asyncDemoService"

interface="com.tuling.teach.service.async.AsyncDemoService">

<!-- 异步调async：true 异步调用 false 同步调用-->

<dubbo:method name="sayHello1" async="true"/>

<dubbo:method name="sayHello2" async="false"/>

<!-- return="false" 不需要返回结果,直接返回-->

<dubbo:method name="notReturn" return="false"/>

</dubbo:reference>

**异步获取结果演示：**

编写异步调用代码

编写同步调用代码

分别演示同步调用与异步调用耗时

*异步调用结果获取Demo*

*demoService*.sayHello1("han");

Future<Object> future1 = RpcContext.*getContext*().getFuture();

*demoService*.sayHello2("han2");

Future<Object> future2 = RpcContext.*getContext*().getFuture();

Object r1 = null, r2 = null;

// wait 直到拿到结果 获超时

r1 = future1.get();

// wait 直到拿到结果 获超时

r2 = future2.get();

**过滤器**

类似于 WEB 中的Filter ，Dubbo本身提供了Filter 功能用于拦截远程方法的调用。其支持自定义过滤器与官方的过滤器使用：

#TODO 演示添加日志访问过滤:

<dubbo:provider filter="accesslog" accesslog="E:\workspace\tuling-teach-dubbo\target\logs\dubbo.log"/>

以上配置 就是 为 服务提供者 添加 日志记录过滤器， 所有访问日志将会集中打印至 accesslog 当中

### **二、Dubbo 其它使用场景详解**

**泛化提供&引用**

**泛化提供**

是指不通过接口的方式直接将服务暴露出去。通常用于Mock框架或服务降级框架实现。

#TODO 示例演示

**public static void** doExportGenericService() {

ApplicationConfig applicationConfig = **new** ApplicationConfig();

applicationConfig.setName(**"demo-provider"**);

*// 注册中心*

RegistryConfig registryConfig = **new** RegistryConfig();

registryConfig.setProtocol(**"zookeeper"**);

registryConfig.setAddress(**"192.168.0.147:2181"**);

ProtocolConfig protocol=**new** ProtocolConfig();

protocol.setPort(-1);

protocol.setName(**"dubbo"**);

GenericService demoService = **new** MyGenericService();

ServiceConfig<GenericService> service = **new** ServiceConfig<GenericService>();

*// 弱类型接口名*

service.setInterface(**"com.tuling.teach.service.DemoService"**);

*// 指向一个通用服务实现*

service.setRef(demoService);

service.setApplication(applicationConfig);

service.setRegistry(registryConfig);

service.setProtocol(protocol);

*// 暴露及注册服务*

service.export();

}

**泛化引用**

是指不通过常规接口的方式去引用服务，通常用于测试框架。

ApplicationConfig applicationConfig = new ApplicationConfig();

applicationConfig.setName("demo-provider");

*// 注册中心*

RegistryConfig registryConfig = new RegistryConfig();

registryConfig.setProtocol("zookeeper");

registryConfig.setAddress("192.168.0.147:2181");

*// 引用远程服务*

ReferenceConfig<GenericService> reference = new ReferenceConfig<GenericService>();

*// 弱类型接口名*

reference.setInterface("com.tuling.teach.service.DemoService");

*// 声明为泛化接口*

reference.setGeneric(true);

reference.setApplication(applicationConfig);

reference.setRegistry(registryConfig);

*// 用com.alibaba.dubbo.rpc.service.GenericService可以替代所有接口引用*

GenericService genericService = reference.get();

Object result = genericService.$invoke("sayHello", new String[]{"java.lang.String"}, new Object[]{"world"});

**隐示传参**

是指通过非常方法参数传递参数，类似于http 调用当中添加cookie值。通常用于分布式追踪框架的实现。使用方式如下 ：

//客户端隐示设置值

RpcContext.getContext().setAttachment("index", "1"); // 隐式传参，后面的远程调用都会隐

//服务端隐示获取值

String index = RpcContext.getContext().getAttachment("index");

**令牌验证**

通过令牌验证在注册中心控制权限，以决定要不要下发令牌给消费者，可以防止消费者绕过注册中心访问提供者，另外通过注册中心可灵活改变授权方式，而不需修改或升级提供者



使用：

<!--随机token令牌，使用UUID生成--><dubbo:provider interface="com.foo.BarService" token="true" />

### **三、Dubbo 路由功能使用**