Consumer

. RPC和HTTP

常见远程调用方式：

RPC:(Remote Produce Call)远程过程调用

1.基于Socket

2.自定义数据格式

3.速度快，效率高

4.典型应用代表：Dubbo，WebService，ElasticSearch集群间互相调用

HTTP：网络传输协议

1.基于TCP/IP

2.规定数据传输格式

3.缺点是消息封装比较臃肿、传输速度比较慢

4.优点是对服务提供和调用方式没有任何技术限定，自由灵活，更符合微服务理念

RestTemplate是Rest的HTTP客户端模板工具类

对基于Http的客户端进行封装

实现对象与JSON的序列化与反序列化

不限定客户端类型，目前常用的3种客户端都支持：HttpClient、OKHttp、JDK原生URLConnection(默认方式)

**RPC和HTTP的区别：RPC是根据语言API来定义，而不是根据基于网络的应用来定义。**

1 pom增加nacos服务，需要可以加入feign

*<!-- nacos客户端依赖 -->*

<dependency>  
 <groupId>com.alibaba.cloud</groupId>  
 <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>  
</dependency>  
*<!-- feign客户端依赖 -->*<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
</dependency>

2

增加服务发现注解

ConsumerDemoApplication

增加@EnableDiscoveryClient

3

1. **loadBalancer方式：**

@Autowired//

private LoadBalancerClient loadBalancerClient;

@Autowired**//增加RestTemplate，用于rpc调用**

private RestTemplate restTemplate;

@Value("${spring.application.name}")//

private String appName; **//增加本地项目名字**

@RequestMapping(value = "/rpcProvider", method = RequestMethod.GET)

public String rpcProvider() {

ServiceInstance serviceInstance = loadBalancerClient.choose("nacos-provider");**//通过nacos取得了提供者实例**

String url = String.format("http://%s:%s/echo/%s", serviceInstance.getHost(), serviceInstance.getPort(), appName);

return "nacos-consume调用," + restTemplate.getForObject(url, String.class); **//通过url用restTemplate完成rpc**

}

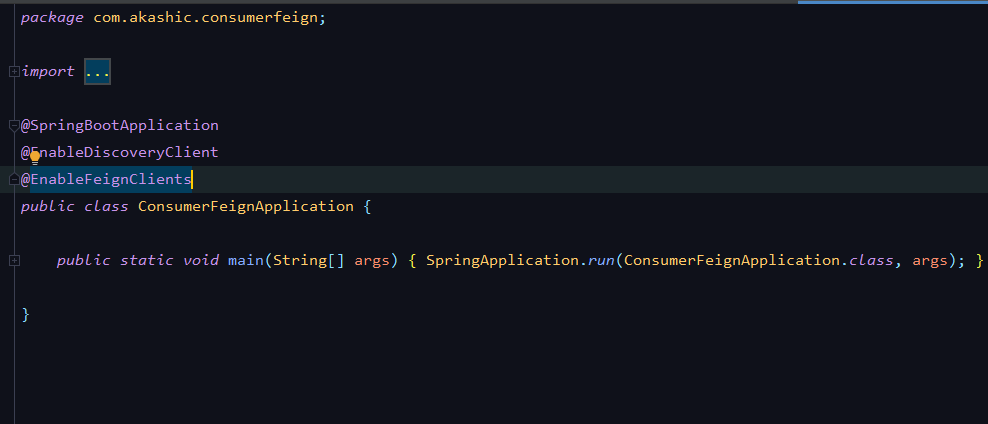
1. **feingn方式：**

@FeignClient(value = "nacos-provider")**//feign通过nacos里面application-name获取到对应的provider**

public interface ConsumerFeignProviderService{

@GetMapping(value = "/echo/{message}")

String echo(@PathVariable("message") String message);**//方法名字与nacos-provider的方法一样**



整体pom，不是feign调用

<?*xml version*="1.0" *encoding*="UTF-8"?>  
<project *xmlns*="http://maven.apache.org/POM/4.0.0" *xmlns:xsi*="http://www.w3.org/2001/XMLSchema-instance"  
 *xsi:schemaLocation*="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.4.RELEASE</version>  
 <relativePath/> *<!-- lookup parent from repository -->* </parent>  
 <groupId>com.akashic</groupId>  
 <artifactId>sentinel-demo</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>sentinel-demo</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>