## 一站式微服务解决方案Spring Cloud Alibaba

主讲：Cat 老师

北京动力节点教育科技有限公司  
2020 • 北京

动力节点•版权所有•禁止传播

## 快速回顾

1、Spring家族开源项目梳理

2、Spring Cloud下的开源项目梳理

3、Spring Cloud Alibaba下的开源及商业项目梳理

4、微服务的基础模型：服务消费者-注册中心-服务提供者

5、What is Nacos？

6、Nacos的运行环境部署（Java写的，springboot项目）

7、Nacos的后台web管控台

8、Nacos作为注册中心注册服务

9、Nacos作为注册中心发现/订阅服务

10、服务消费者负载均衡调用服务提供者（ribbon）restTemple、feign

11、Nacos宕机时服务消费者缓存注册中心信息

12、Nacos作为**配置中心**存储项目各种配置

13、Nacos作为配置中心支持自动配置刷新（不需要重启应用）

14、Nacos配置中心DataId+Group+Properties/yaml+配置内容（比较灵活）

15、Nacos配置中心多环境配置（profile）

即${spring.application.name}-${profile}.${file-extension:properties}

16、Nacos服务配置数据模型（命名空间、Group、Data Id）

17、Nacos数据持久化（mysql）

18、Ncaos集群部署（结合nginx）

## RestTemplete请求模板类解读

当我们服务消费者去调用服务提供者提供的服务的时候，使用了一个极其方便的对象叫RestTemplate，我们通常使用RestTemplate中最简单的一个功能getForObject 发起了一个get请求去调用服务端的数据，同时，我们还通过配置@LoadBalanced注解开启客户端负载均衡，RestTemplate的功能非常强大， 接下来我们就来详细的看一下RestTemplate中几种常见请求方法的使用，在日常操作中，基于Rest的方式通常是四种情况，它们分表是：

GET 请求 --查询数据

POST 请求 –添加数据

PUT 请求 – 修改数据

DELETE 请求 –删除数据

下面我们逐一解读；

**RestTemplate 的 GET 请求**

Get 请求可以有两种方式：

第一种：getForEntity

该方法返回一个 ResponseEntity<T>对象，ResponseEntity<T>是 Spring 对 HTTP 请求响应的封装，包括了几个重要的元素，比如响应码、contentType、 contentLength、响应消息体等；

ResponseEntity<String> responseEntity =restTemplate.getForEntity(

"http://29-SPRINGCLOUD-SERVICE-PROVIDER/service/hello", String.class);

String body =responseEntity.getBody();

HttpStatus statusCode =responseEntity.getStatusCode();

int statusCodeValue =responseEntity.getStatusCodeValue();

HttpHeaders headers =responseEntity.getHeaders();

System.*out*.println(body);

System.*out*.println(statusCode);

System.*out*.println(statusCodeValue);

System.*out*.println(headers);

以上代码：

getForEntity 方法第一个参数为要调用的服务的地址，即服务提供者提供的

**http://29-SPRINGCLOUD-SERVICE-PROVIDER/service/hello** 接口地址，注意这里是通过服务名调用而不是服务地址，如果改为服务地址就无法使用 Ribbon 实现客户端负载均衡了；

getForEntity方法第二个参数String.class表示希望返回的body类型是 String类型，如果希望返回一个对象，也是可以的，比如User对象；

另外两个重载方法：

**public** <T> ResponseEntity<T> getForEntity(String url, Class<T> responseType, Object... uriVariables) **throws** RestClientException

比如：

**restTemplate**.getForEntity(**"http://01-SPRINGCLOUD-SERVICE-PROVIDER/service/hello?id= {1}&name={2}"**, String.**class**, **"{1, '张无忌'}"**).getBody();

**public** <T> ResponseEntity<T> getForEntity(String url, Class<T> responseType, Map<String, ?> uriVariables) **throws** RestClientException

比如：

Map<String, Object> paramMap =**new** ConcurrentHashMap<>();

paramMap.put(**"id"**, 1);

paramMap.

put(**"name"**, **"张无忌"**);

**restTemplate**.getForEntity(**"http://01-SPRINGCLOUD-SERVICE-PROVIDER/service/hello?id= {id}&name={name}"**, String.**class**, paramMap).getBody();

第二种：getForObject()与 getForEntity 使用类似，只不过getForObject是在getForEntity基础上进行了再次封装，可以将http的响应体body信息转化成指定的对象，方便我们的代码开发；

当你不需要返回响应中的其他信息，只需要body体信息的时候，可以使用这个更方便；

它也有两个重载的方法，和 getForEntity 相似；

<T> T getForObject(URI url, Class<T> responseType) **throws** RestClientException;

<T> T getForObject(String url, Class<T> responseType, Object... uriVariables) **throws**

RestClientException;

<T> T getForObject(String url, Class<T> responseType, Map<String, ?> uriVariables)

**throws** RestClientException;

**RestTemplate 的 POST 请求：**

Post 与 Get 请求非常类似：

restTemplate.postForObject()

restTemplate.postForEntity()

restTemplate.postForLocation()

RestTemplate 的 PUT 请求：

restTemplate.put();

**RestTemplate 的 DELETE 请求：**

restTemplate.delete();

## Ribbon组件解读

Ribbon是什么？Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡器；

我们通常说的负载均衡是指将一个请求均匀地分摊到不同的节点单元上执行，负

载均和分为硬件负载均衡和软件负载均衡：

**硬件负载均衡**：比如 F5、深信服、Array 等；

**软件负载均衡：**比如 Nginx、LVS、HAProxy 等；（是由一个服务器实现的）

Ribbon是Netflix公司发布的开源项目（组件、框架、jar包），主要功能是提供客户端的软件负载均衡算法，它会从nacos中获取一个可用的服务端清单，通过心跳检测来剔除故障的服务端节点以保证清单中都是可以正常访问的服务端节点；

当客户端发送请求，则ribbon负载均衡器按某种算法（比如轮询、权重、随机等）从维护的可用服务端清单中取出一台服务端的地址，然后进行请求；

Ribbon非常简单，可以说就是一个jar包，这个jar包实现了负载均衡算法，Spring Cloud Alibaba底层对Ribbon做了二次封装，可以让我们使用 RestTemplate的服务请求，自动转换成客户端负载均衡的服务调用；

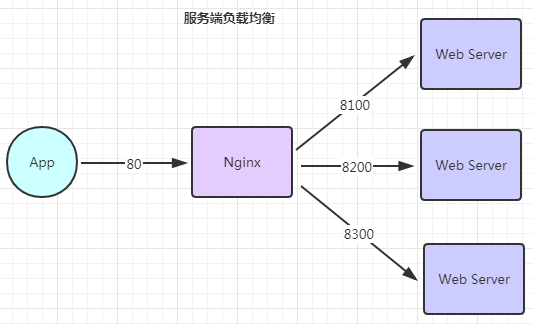
Ribbon支持多种负载均衡算法，还支持自定义的负载均衡算法；

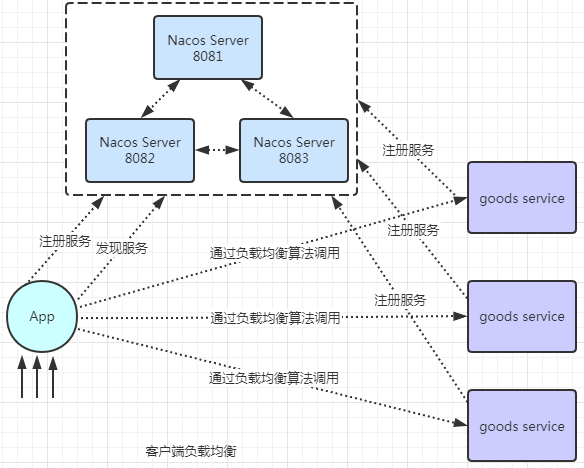
## 客户端负载均衡 vs 服务端负载均衡

从另一个角度来观看负载均衡的话，负载均衡又分为两种方式：

**服务端负载均衡**

**客户端负载均衡**





## 采用Ribbon实现服务调用

1. 首先加入ribbon的依赖；

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

</dependency>

1. 要使用ribbon，只需要一个注解@LoadBalanced：

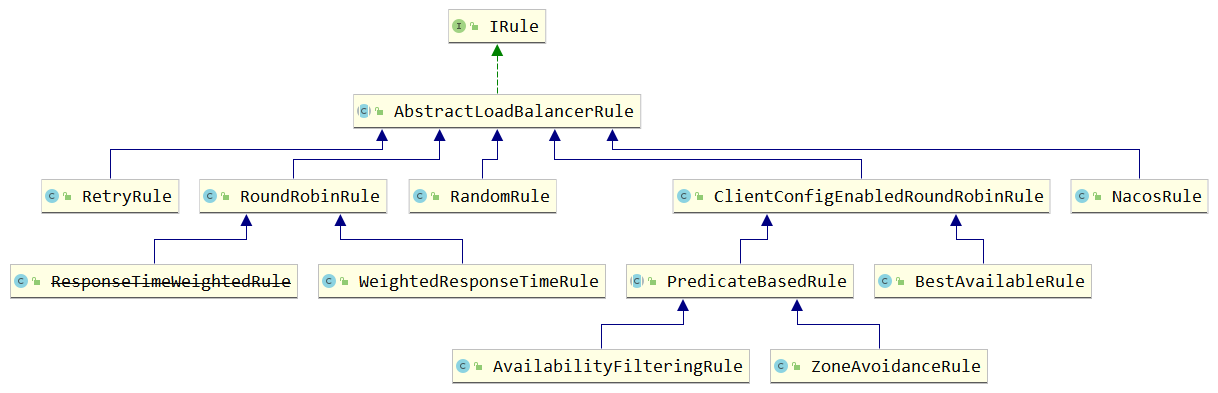
@Bean  
@LoadBalanced  
public RestTemplate restTemplate(){  
 RestTemplate restTemplate = new RestTemplate();  
 return restTemplate;  
}

在RestTemplate上面加入@LoadBalanced注解，这样就可以实现RestTemplate在调用时自动负载均衡；

## Ribbon负载均衡策略

Ribbon的负载均衡策略是由 IRule 接口定义, 该接口由如下实现：

在jar包：com.netflix.ribbon#ribbon-loadbalancer中；



要使用ribbon实现负载均衡，在Spring 的配置类里面把对应的负载均衡接口实现类作为一个Bean配置一下就行了，即像下面这样：

/\*\*

\* 更改负载均衡策略，默认是ZoneAvoidanceRule策略

\*

\* @return

\*/

@Bean

public IRule iRule(){

return new NacosRule();

}

负载均衡的入口：ILoadBalancer接口；

@Bean  
public IRule iRule(){  
 return new RoundRobinRule();  
}

|  |  |
| --- | --- |
| **负载均衡实现** | **策略** |
| RandomRule | 随机 |
| RoundRobinRule | 轮询 |
| AvailabilityFilteringRule | 先过滤掉由于多次访问故障的服务，以及并  发连接数超过阈值的服务，然后对剩下的服  务按照轮询策略进行访问； |
| WeightedResponseTimeRule | 根据平均响应时间计算所有服务的权重，响  应时间越快服务权重就越大被选中的概率即  越高，如果服务刚启动时统计信息不足，则  使用RoundRobinRule策略，待统计信息足够会切换到该WeightedResponseTimeRule策  略； |
| RetryRule | 先按照RoundRobinRule策略分发，如果分发  到的服务不能访问，则在指定时间内进行重  试，然后分发其他可用的服务； |
| BestAvailableRule | 先过滤掉由于多次访问故障的服务，然后选  择一个并发量最小的服务； |
| ZoneAvoidanceRule （默认） | 综合判断服务节点所在区域的性能和服务节  点的可用性，来决定选择哪个服务； |

如果我们没有指定负载均衡策略，ribbon默认的负载均衡是ZoneAvoidanceRule，自定义负载均衡策略：

public class MyNacosRule extends AbstractLoadBalancerRule {  
 @Override  
 public void initWithNiwsConfig(IClientConfig clientConfig)

{*//基本上不需要实现*}  
  
 @Override  
 public Server choose(Object key) {*//实现该方法*}

}

配置文件配置负载均衡策略：

**29-nacos-discovery-provider**.ribbon.NFLoadBalancerRuleClassName=com.bjpowernode.ribbon.MyNacosRule

加粗红色部分是远程服务的名称；

## Ribbon组件的核心接口组成

| 接口 | 作用 | 默认值 |
| --- | --- | --- |
| IclientConfig | 读取配置 | DefaultClientConfigImpl |
| IRule | 负载均衡规则，选择实例 | ZoneAvoidanceRule |
| IPing | 筛选掉ping不通的实例 | DumyPing（该类什么不干，认为每个实例都可用，都能ping通） |
| ServerList<Server> | 交给Ribbon的实例列表 | **Ribbon:**ConfigurationBasedServerList **Spring Cloud Alibaba:**NacosServerList |
| ServerListFilter<Server> | 过滤掉不符合条件的实例 | ZonePreferenceServerListFilter |
| ILoadBalancer | Ribbon的入口 | ZoneAwareLoadBalancer |
| ServerListUpdater | 更新交给Ribbon的List的策略 | PollingServerListUpdater |

这里的每一个接口都可以自定义，可以对这些接口进行扩展，和IRule接口的扩展方式一样，配置方式也是一样，比如自定义IPing：

@Bean

public IPing iPing () {  
 return new PingUrl();  
}

通过配置文件配置属性方式

<serviceName>:

ribbon:

NFLoadBalancerClassName: #ILoadBalancer该接口实现类

NFLoadBalancerRuleClassName: #IRule该接口实现类

NFLoadBalancerPingClassName: #Iping该接口实现类

NIWSServerListClassName: #ServerList该接口实现类

NIWSServerListFilterClassName: #ServiceListFilter该接口实现类

比如：

*#iping的策略***29-nacos-discovery-provider**.ribbon.NFLoadBalancerPingClassName=com.netflix.loadbalancer.PingUrl

## Nacos权重负载均衡



Nacos的负载均衡策略NacosRule已经实现了基于权限的负载均衡，直接使用即可：

@Bean  
public IRule iRule(){  
 return new NacosRule();  
}

## Nacos同一集群优先负载均衡

*#指定集群的名称  
spring.cloud.nacos.discovery.cluster-name=beijing*

集群名称相同的优先调用，当具有相同名称的集群宕机了，才会调用另一个名称不相同的集群；

## Nacos不能跨namespace调用

消费者和提供者的命名空间必须相同才能调用，不能跨命名空间进行调用，命名空间可以实现服务的完全隔离；

*#服务消费者的dev命名空间*spring.cloud.nacos.discovery.namespace=b91111e4-8a21-4c12-9a3f-cf40d93a8319

*#服务提供者1的dev命名空间*spring.cloud.nacos.discovery.namespace=b91111e4-8a21-4c12-9a3f-cf40d93a8319

*#服务提供者2的test命名空间*spring.cloud.nacos.discovery.namespace=921723b6-cc51-4f7d-ad2c-ab0be7836774