**负载均衡**

# 介绍

Feign+Ribbon

推荐使用Feign+Ribbon+Nacos

ribbon是对服务之间调用做负载，是服务之间的负载均衡，zuul是可以对外部请求做负载均衡。

建议使用Feign，声明式编程，入侵少。默认集成了ribbon实现负载均衡。通过接口调用比Ribbon+restTemplate更简单。

1. 引入依赖,feign的依赖即可。Feign默认集成了ribbon。

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId>  </dependency> |

1. Ribbon自定义配置
2. 自己通过继承AbstractLoadBalancerRule类或者实现IRule接口来自定义负载均衡逻辑。

|  |
| --- |
| public class NacosWeightRandomV2Rule extends AbstractLoadBalancerRule {  private static final Logger LOGGER = LoggerFactory.getLogger(NacosWeightRandomV2Rule.class);  @Autowired  private NacosDiscoveryProperties discoveryProperties;  @Override  public Server choose(Object key) {  DynamicServerListLoadBalancer loadBalancer = (DynamicServerListLoadBalancer) getLoadBalancer();  String name = loadBalancer.getName();  try {  Instance instance = discoveryProperties.namingServiceInstance()  .selectOneHealthyInstance(name);  return new NacosServer(instance);  } catch (NacosException e) {  LOGGER.error("发生异常", e);  return null;  }  }  @Override  public void initWithNiwsConfig(IClientConfig iClientConfig) {  }  } |

1. 在application.yml中添加配置

|  |
| --- |
| #ribbon相关配置  #nacos-provider为客户端调用的服务名  nacos-provider:  ribbon:  # 代表Ribbon使用的负载均衡策略，可以是自己定义的规则。  NFLoadBalancerRuleClassName: com.zoe.rule.NacosWeightRandomV2Rule  # 每台服务器最多重试次数，但是首次调用不包括在内， Max number of retries on the same server (excluding the first try)  MaxAutoRetries: 1  # 最多重试多少台服务器，Max number of next servers to retry (excluding the first server)  MaxAutoRetriesNextServer: 1  # 无论是请求超时或者socket read timeout都进行重试，Whether all operations can be retried for this client  OkToRetryOnAllOperations: true  # Interval to refresh the server list from the source  ServerListRefreshInterval: 2000  # Connect timeout used by Apache HttpClient  ConnectTimeout: 3000  # Read timeout used by Apache HttpClient  ReadTimeout: 3000 |

|  |
| --- |
| public class NacosWeightRandomV2Rule extends AbstractLoadBalancerRule {  private static final Logger LOGGER = LoggerFactory.getLogger(NacosWeightRandomV2Rule.class);  @Autowired  private NacosDiscoveryProperties discoveryProperties;  @Override  public Server choose(Object key) {  DynamicServerListLoadBalancer loadBalancer = (DynamicServerListLoadBalancer) getLoadBalancer();  String name = loadBalancer.getName();  try {  Instance instance = discoveryProperties.namingServiceInstance()  .selectOneHealthyInstance(name);  return new NacosServer(instance);  } catch (NacosException e) {  LOGGER.error("发生异常", e);  return null;  }  }  @Override  public void initWithNiwsConfig(IClientConfig iClientConfig) {  }  } |

|  |  |  |  |
| --- | --- | --- | --- |
| 策略名 | 策略声明 | 策略描述 | 实现说明 |
| BestAvailableRule | public class BestAvailableRule extends ClientConfigEnabledRoundRobinRule | 选择一个最小的并发请求的server | 逐个考察Server，如果Server被tripped了，则忽略，在选择其中ActiveRequestsCount最小的server |
| AvailabilityFilteringRule | public class AvailabilityFilteringRule extends PredicateBasedRule | 过滤掉那些因为一直连接失败的被标记为circuit tripped的后端server，并过滤掉那些高并发的的后端server（active connections 超过配置的阈值） | 使用一个AvailabilityPredicate来包含过滤server的逻辑，其实就就是检查status里记录的各个server的运行状态 |
| WeightedResponseTimeRule | public class WeightedResponseTimeRule extends RoundRobinRule | 根据相应时间分配一个weight，相应时间越长，weight越小，被选中的可能性越低。 | 一个后台线程定期的从status里面读取评价响应时间，为每个server计算一个weight。Weight的计算也比较简单responsetime 减去每个server自己平均的responsetime是server的权重。当刚开始运行，没有形成statas时，使用roubine策略选择server。 |
| RetryRule | public class RetryRule extends AbstractLoadBalancerRule | 对选定的负载均衡策略机上重试机制。 | 在一个配置时间段内当选择server不成功，则一直尝试使用subRule的方式选择一个可用的server |
| RoundRobinRule | public class RoundRobinRule extends AbstractLoadBalancerRule | roundRobin方式轮询选择server | 轮询index，选择index对应位置的server |
| RandomRule | public class RandomRule extends AbstractLoadBalancerRule | 随机选择一个server | 在index上随机，选择index对应位置的server |
| ZoneAvoidanceRule | public class ZoneAvoidanceRule extends PredicateBasedRule | 复合判断server所在区域的性能和server的可用性选择server | 使用ZoneAvoidancePredicate和AvailabilityPredicate来判断是否选择某个server，前一个判断判定一个zone的运行性能是否可用，剔除不可用的zone（的所有server），AvailabilityPredicate用于过滤掉连接数过多的Server。 |