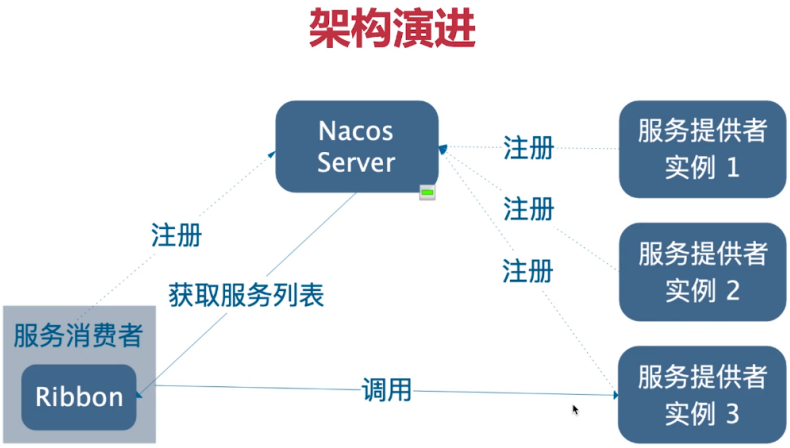
笔记: {chenqian}

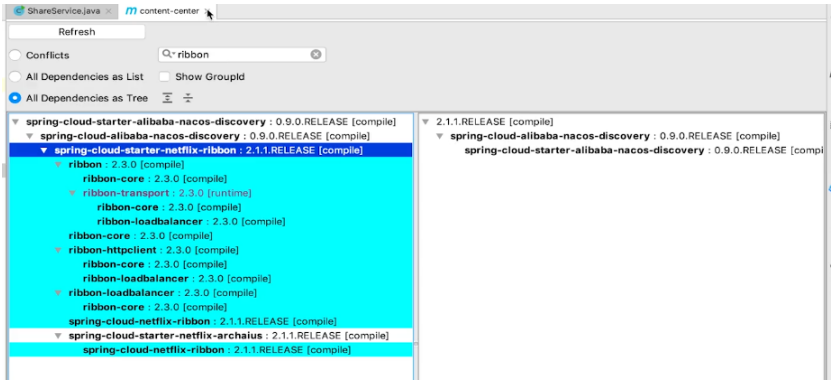
负载均衡的两种方式

使用Ribbon实现负债均衡



Ribbon自动从Nacos获取服务列表,通过负债均衡算法计算出某个实例,交给restTemplate请求.

1. 添加依赖,Nacos已经包含了Ribbon依赖.



2. 添加注解

```
@Bean
@LoadBalanced
public RestTemplate restTemplate() { return n
```

以上简单两步,已经把负载均衡整合进Nacos中了.

Ribbon组成

Ribbon虽然比较简单,功能却非常强大.

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	`DummyPing`
`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`

注意: ServerList<server>,Ribbon的默认值和Spring Cloud Alibaba的默认值不一样.
Ribbon可以把实例列表配置在配置文件中,从配置文件读取列表.而Spring Cloud Alibaba中则是利用NacosClient从NacosServer中获取服务列表.

Ribbon内置的负载均衡规则

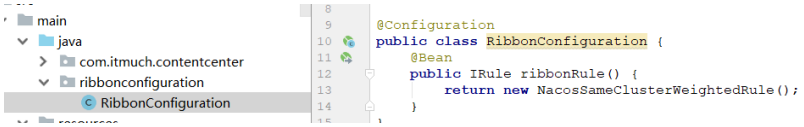
规则名称	特点
AvailabilityFilteringRule	过滤掉一直连接失败的被标记为circuit tripped的后端Server，并过滤掉那些高并发的后端Server或者使用一个AvailabilityPredicate来包含过滤server的逻辑，其实就就是检查status里记录的各个Server的运行状态
BestAvailableRule	选择一个最小的并发请求的Server，逐个考察Server，如果Server被tripped了，则跳过
RandomRule	随机选择一个Server
ResponseTimeWeightedRule	已废弃，作用同WeightedResponseTimeRule
RetryRule	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	轮询选择， 轮询index，选择index对应位置的Server
WeightedResponseTimeRule	根据响应时间加权，响应时间越长，权重越小，被选中的可能性越低
ZoneAvoidanceRule	复合判断Server所Zone的性能和Server的可用性选择Server，**在没有Zone的环境下，类似于轮询(RoundRobinRule)**

默认的规则是ZoneAvoidanceRule,在没有Zone的环境下,默认是轮询.

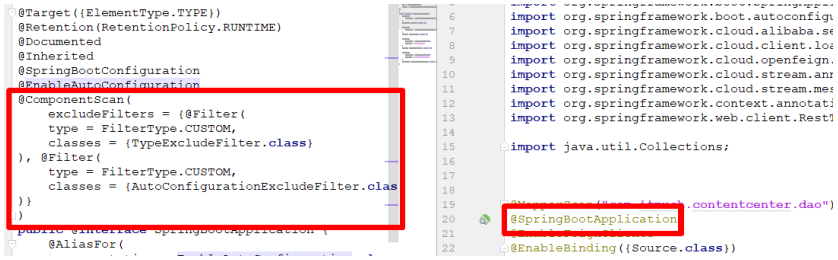
配置方式

1. java代码配置(强烈不推荐)

Ribbon包建在SpringBoot启动类包外.



@SpringBootApplication是一个组合注解,其中@compinentScan会扫描@service@controller@component等注解.
这里称之为父上下文,而自定义的配置类则称之为子上下文.父子上下文扫描一旦重叠,则会发生事务不生效的问题.



除此之外,在官方文档中给出的解释:

> The `CustomConfiguration` clas must be a `@Configuration` class, but take care that it is not in a `@ComponentScan` for the main application context. Otherwise, it is shared by all the `@RibbonClients`. If you use `@ComponentScan` (or `@SpringBootApplication`), you need to take steps to avoid it being included (for instance, you can put it in a separate, non-overlapping package or specify the packages to scan explicitly in the `@ComponentScan`).

必须要有@Configuration但不能被@ComponentScan或@SpringBootApplication重复扫描,否则会被所有RibbonClient共享.

2. 配置文件配置

过于简单,不做介绍.

两种配置方式对比:

配置方式	优点	缺点
代码配置	基于代码，更加灵活	有小坑（父子上下文） 线上修改得重新打包、发布
属性配置	易上手 配置更加直观 线上修改无需重新打包、发布 **优先级更高**	极端场景下没有代码配置方式灵活

Ribbon加载方式

默认懒加载,因此在第一次访问时候就比较慢.

细粒度饥饿加载

```yaml
ribbon:
  eager-load:
    enabled: true
    clients: user-center,xxx,xxx
```

不在名单里依然是懒加载,多个服务使用逗号分割.

扩展Ribbon支持Nacos权重的三种方式(慕课网,大目老师笔记)

原文地址: https://www.imooc.com/article/288660

Nacos权重配置对Spring Cloud Alibaba无效。也就是说，不管在Nacos控制台上如何配置，调用时都不管权重设置的。

Spring Cloud Alibaba通过整合Ribbon的方式，实现了负载均衡。所使用的负载均衡规则是 `ZoneAvoidanceRule` 。

利用Nacos Client的能力[推荐]

思路:

在阅读代码Nacos源码的过程中，发现**Nacos Client本身就提供了负载均衡的能力**，并且负载均衡算法**正是我们想要的根据权重选择实例**!

代码在 `com.alibaba.nacos.api.naming.NamingService#selectOneHealthyInstance` ，只要想办法调用到这行代码，就可以实现我们想要的功能啦!

代码:

```java
@Slf4j
public class NacosWeightRandomV2Rule extends AbstractLoadBalancerRule {
    @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);
            log.info("选中的instance = {}", instance);
            /*
             * instance转server的逻辑参考自:
             * org.springframework.cloud.alibaba.nacos.ribbon.NacosServerList.instancesToServerList
             */
            return new NacosServer(instance);
        } catch (NacosException e) {
            log.error("发生异常", e);
            return null;
        }
    }
    @Override
    public void initWithNiwsConfig(IClientConfig iClientConfig) {
    }
}
```

Ribbon-同集群优先

目标:

1. 优先调用同集群下的实例
2. 实现基于权重配置的负载均衡

```
@Autowired
private NacosDiscoveryProperties nacosDiscoveryProperties;

@Override
public void initWithNiwsConfig(IClientConfig clientConfig) {
}

@Override
public Server choose(Object key) {
    try {
        // 拿到配置文件中的集群名称 BJ
        String clusterName = nacosDiscoveryProperties.getClusterName();
        BaseLoadBalancer loadBalancer = (BaseLoadBalancer) this.getLoadBalancer();
        // 想要请求的微服务的名称
        String name = loadBalancer.getName();
        // 拿到服务发现的相关API
        NamingService namingService = nacosDiscoveryProperties.namingServiceInstance();
        // 1. 找到指定服务的所有实例 A
        List<Instance> instances = namingService.selectInstances(name, true);
        // 2. 过滤出相同集群下的所有实例 B
        List<Instance> sameClusterInstances = instances.stream()
            .filter(instance -> Objects.equals(instance.getClusterName(), clusterName))
            .collect(Collectors.toList());
        // 3. 如果B是空，就用A
        List<Instance> instancesToBeChosen = new ArrayList<>();
        if (CollectionUtils.isEmpty(sameClusterInstances)) {
            instancesToBeChosen = instances;
            log.warn("发生跨集群的调用, name = {}, clusterName = {}, instances = {}",
                name,
                clusterName,
                instances
            );
```

```
        } else {
            instancesToBeChosen = sameClusterInstances;
        }
        // 4. 基于权重的负载均衡算法，返回1个实例
        Instance instance = ExtendBalancer.getHostByRandomWeight2(instancesToBeChosen);
        log.info("选择的实例是 port = {}, instance = {}", instance.getPort(), instance);
        return new NacosServer(instance);
    } catch (NacosException e) {
        log.error("发生异常了", e);
        return null;
    }
}

class ExtendBalancer extends Balancer {
    public static Instance getHostByRandomWeight2(List<Instance> hosts) {
        return getHostByRandomWeight(hosts);
    }
}
```

基于元数据的版本管理(慕课网,大目老师笔记)

原文地址:https://www.imooc.com/article/288674

一个微服务在线上可能多版本共存，例如：

- 服务提供者有两个版本：v1、v2
- 服务消费者也有两个版本：v1、v2

v1/v2是不兼容的。服务消费者v1只能调用服务提供者v1；消费者v2只能调用提供者v2。如何实现呢?
下面围绕该场景，实现微服务之间的版本控制。
元数据就是一堆的描述信息，以map存储。举个例子：

```
spring:
  cloud:
    nacos:
        metadata:
          # 自己这个实例的版本
          version: v1
          # 允许调用的提供者版本
          target-version: v1
```

需求分析

- 优先选择同集群下，符合metadata的实例
- 如果同集群加没有符合metadata的实例，就选择所有集群下，符合metadata的实例

```
@Slf4j
public class NacosFinalRule extends AbstractLoadBalancerRule {
    @Autowired
    private NacosDiscoveryProperties nacosDiscoveryProperties;

    @Override
    public Server choose(Object key) {
        // 负载均衡规则：优先选择同集群下，符合metadata的实例
        // 如果没有，就选择所有集群下，符合metadata的实例

         // 1. 查询所有实例 A
         // 2. 筛选元数据匹配的实例 B
         // 3. 筛选出同cluster下元数据匹配的实例 C
```

```
        // 4. 如果C为空，就用B
        // 5. 随机选择实例
        try {
            String clusterName = this.nacosDiscoveryProperties.getClusterName();
            String targetVersion = this.nacosDiscoveryProperties.getMetadata().get("target-version"
            DynamicServerListLoadBalancer loadBalancer = (DynamicServerListLoadBalancer) getLoadBal
            String name = loadBalancer.getName();
            NamingService namingService = this.nacosDiscoveryProperties.namingServiceInstance();
            // 所有实例
            List<Instance> instances = namingService.selectInstances(name, true);
            List<Instance> metadataMatchInstances = instances;
            // 如果配置了版本映射，那么只调用元数据匹配的实例
            if (StringUtils.isNotBlank(targetVersion)) {
                metadataMatchInstances = instances.stream()
                        .filter(instance -> Objects.equals(targetVersion, instance.getMetadata().ge
                        .collect(Collectors.toList());
                if (CollectionUtils.isEmpty(metadataMatchInstances)) {
                    log.warn("未找到元数据匹配的目标实例！请检查配置。targetVersion = {}, instance = {}",
                    return null;
                }
            }
            List<Instance> clusterMetadataMatchInstances = metadataMatchInstances;
            // 如果配置了集群名称，需筛选同集群下元数据匹配的实例
            if (StringUtils.isNotBlank(clusterName)) {
                clusterMetadataMatchInstances = metadataMatchInstances.stream()
                        .filter(instance -> Objects.equals(clusterName, instance.getClusterName()))
                        .collect(Collectors.toList());
                if (CollectionUtils.isEmpty(clusterMetadataMatchInstances)) {
                    clusterMetadataMatchInstances = metadataMatchInstances;
                    log.warn("发生跨集群调用。clusterName = {}, targetVersion = {}, clusterMetadataMa
                }
            }
            Instance instance = ExtendBalancer.getHostByRandomWeight2(clusterMetadataMatchInstances
            return new NacosServer(instance);
        } catch (Exception e) {
            log.warn("发生异常", e);
            return null;
        }
    }

    @Override
    public void initWithNiwsConfig(IClientConfig iClientConfig) {
    }
}
```

负载均衡算法：

```
public class ExtendBalancer extends Balancer {
    /**
     * 根据权重，随机选择实例
     * @param instances 实例列表
     * @return 选择的实例
     */
    public static Instance getHostByRandomWeight2(List<Instance> instances) {
        return getHostByRandomWeight(instances);
```

```
9       }
10  }
```