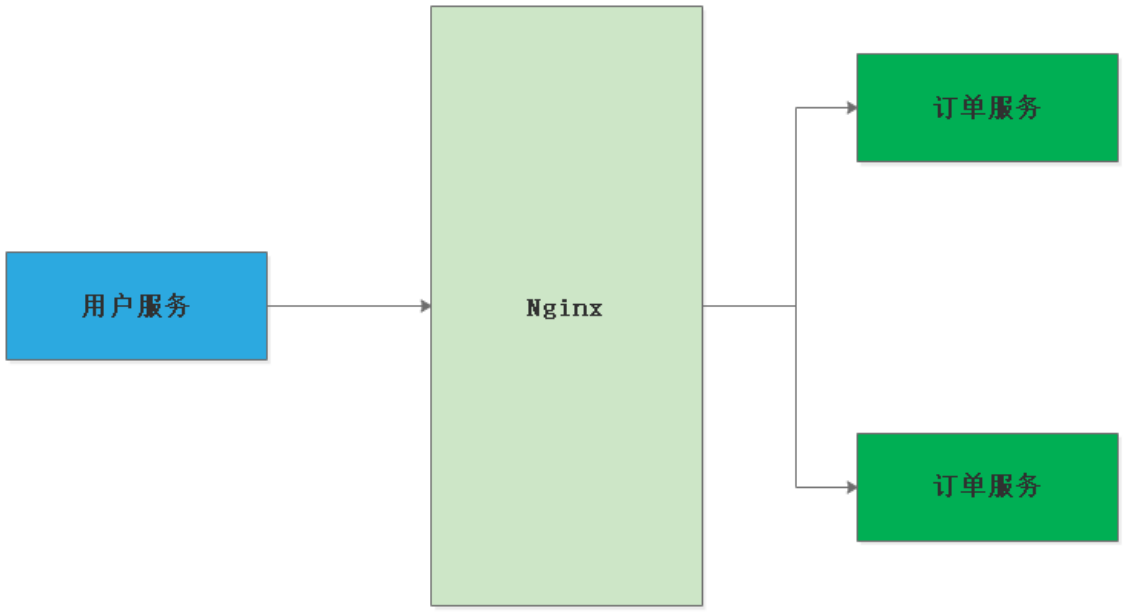
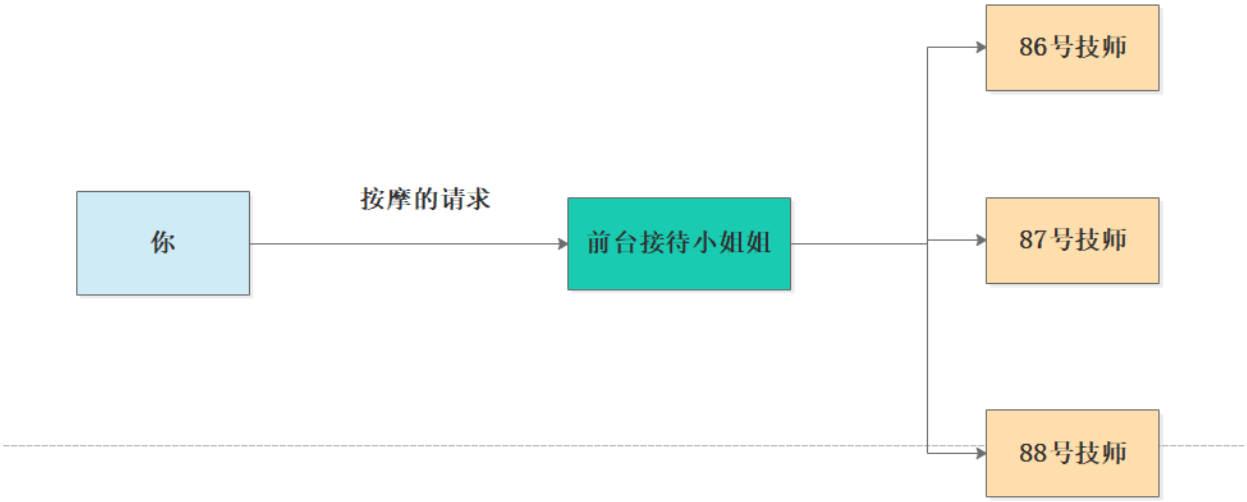
一:负载均衡的二种实现

1.1)服务端的负载均衡(Nginx)



①：我们用户服务发送请求首先打到Ng上，然后Ng根据负载均衡算法进行选择一个服务调用,而我们的Ng部署在服务器上的，所以Ng又称为服务端的负载均衡(具体调用哪个服务，由Ng所了算)

生活案例: 程序员张三 **去盲人按摩, 前台的小姐姐接待了张三，然后为张三分派技师按摩.**

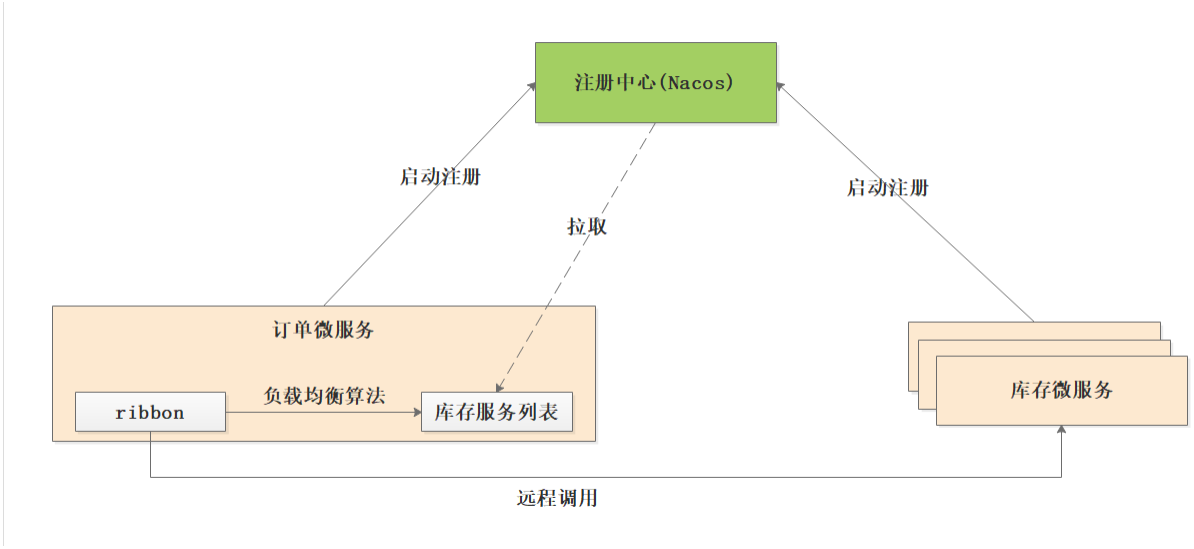


1.2)客户端负载均衡(ribbon)

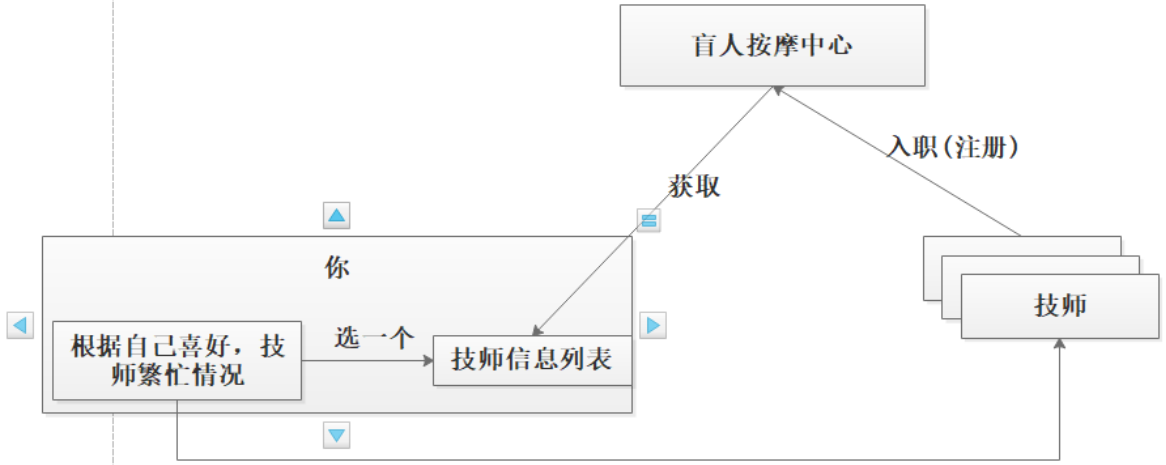
spring cloud ribbon是 基于NetFilix ribbon 实现的一套**客户端的负载均衡工具**,Ribbon客户端组件提供一系列的完善的配置，如超时，重试等。

通过Load Balancer（LB）获取到服务提供的所有机器实例，Ribbon会自动基于某种规则(轮询，随机)去调用这些服务。**Ribbon也可以实现**

我们自己的负载均衡算法。



生活案例:



1.3)自定义的负载均衡算法(随机)

我们在第一节课中得知可以通过DiscoveryClient组件来去我们的Nacos服务端拉取给名称的微服务列表。我们可以通过这个特性来改写我们的RestTemplate组件.

①：经过阅读源码RestTemplate组件得知，不管是post,get请求最终是会调用我们的doExecute()方法，所以我们写一个TulingRestTemplate类继承RestTemplate，从写doExucute()方法。

```
@Slf4j
public class TulingRestTemplate extends RestTemplate {

 private DiscoveryClient discoveryClient;

 public TulingRestTemplate (DiscoveryClient discoveryClient) {
 this.discoveryClient = discoveryClient;
 }

 protected <T> T doExecute(URI url, @Nullable HttpMethod method, @Nullable RequestCallback requestCallback,
 @Nullable ResponseExtractor<T> responseExtractor) throws RestClientException {

 Assert.notNull(url, "URI is required");
 Assert.notNull(method, "HttpMethod is required");
 ClientHttpResponse response = null;
 try {
 //判断url的拦截路径,然后去redis(作为注册中心)获取地址随机选取一个
 log.info("请求的url路径为:{}",url);
 url = replaceUrl(url);
 log.info("替换后的路径:{}",url);
 ClientHttpRequest request = createRequest(url, method);
 if (requestCallback != null) {
 requestCallback.doWithRequest(request);
 }
 response = request.execute();
 handleResponse(url, method, response);
 return (responseExtractor != null ? responseExtractor.extractData(response) : null);
 }
 catch (IOException ex) {
 String resource = url.toString();
 String query = url.getRawQuery();
 resource = (query != null ? resource.substring(0, resource.indexOf('?')) : resource);
```

```
33  throw new ResourceAccessException("I/O error on " + method.name() +
34  " request for \"" + resource + "\": " + ex.getMessage(), ex);
35  } finally {
36  if (response != null) {
37  response.close();
38  }
39  }
40  }
41
42
43  /**
44  * 把服务实例名称替换为ip:端口
45  * @param url
46  * @return
47  */
48  private URI replaceUrl(URI url){
49  //解析我们的微服务的名称
50  String sourceUrl = url.toString();
51  String [] httpUrl = sourceUrl.split("//");
52  int index = httpUrl[1].replaceFirst("/","@").indexOf("@");
53  String serviceName = httpUrl[1].substring(0,index);
54
55  //通过微服务的名称去nacos服务端获取 对应的实例列表
56  List<ServiceInstance> serviceInstanceList = discoveryClient.getInstance
s(serviceName);
57  if(serviceInstanceList.isEmpty()) {
58  throw new RuntimeException("没有可用的微服务实例列表:"+serviceName);
59  }
60
61  //采取随机的获取一个
62  Random random = new Random();
63  Integer randomIndex = random.nextInt(serviceInstanceList.size());
64  log.info("随机下标:{}",randomIndex);
65  String serviceIp = serviceInstanceList.get(randomIndex).getUri().toStri
ng();
66  log.info("随机选举的服务IP:{}",serviceIp);
67  String targetSource = httpUrl[1].replace(serviceName,serviceIp);
68  try {
69  return new URI(targetSource);
70  } catch (URISyntaxException e) {
71  e.printStackTrace();
```

```
 }
 return url;
 }

}
```

1.4）通过Ribbon组件来实习负载均衡(默认的负载均衡算法是 轮询)

①：创建整合Ribbon的工程:

服务消费者(tulingvip02-ms-alibaba-ribbon-order)

服务提供者(tulingvip02-ms-alibaba-ribbon-product)**服务提供者不需要Ribbon的依赖**

第一步:加入依赖（加入nocas-client和ribbon的依赖）

```
<!--加入nocas-client-->
<dependency>
 <groupId>com.alibaba.cloud</groupId>
 <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
</dependency>

<!--加入ribbon-->
<dependency>
 <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
 </dependency>
```

第二步:写注解: 在RestTemplate上加入@LoadBalanced注解

```
@Configuration
public class WebConfig {

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

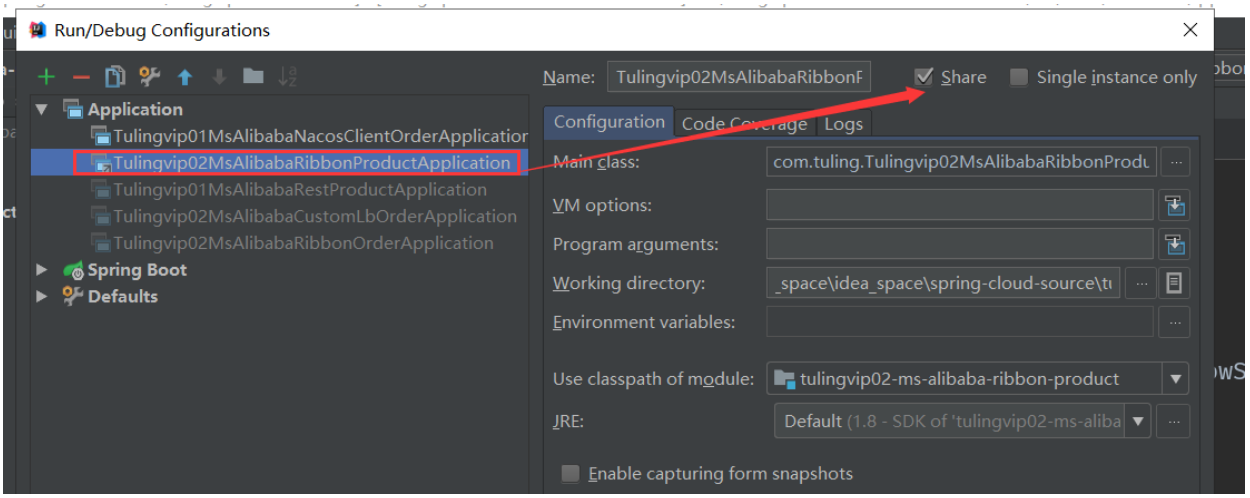
第三步:写配置文件(这里是写Nacos 的配置文件,暂时没有配置Ribbon的配置)

```
spring:
 cloud:
 nacos:
 discovery:
```

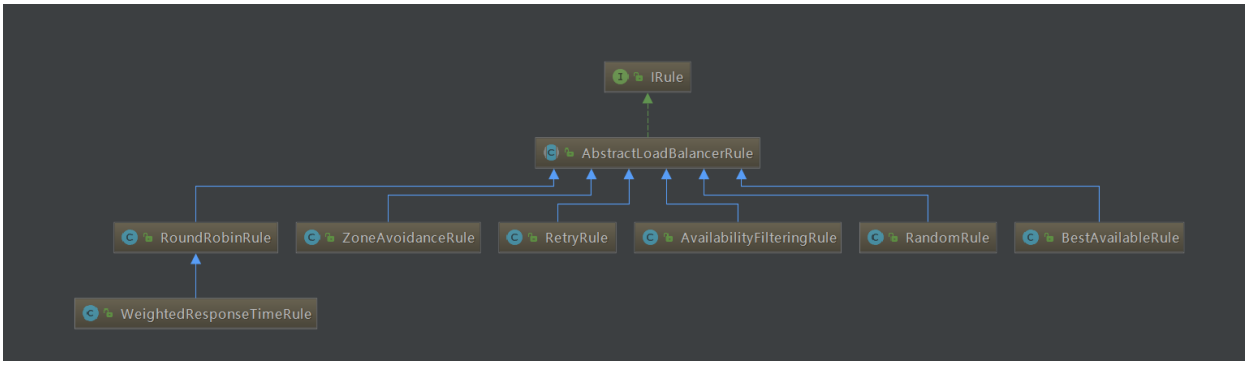
```
5   server-addr: localhost:8848
6   application:
7   name: order-center
```

启动技巧: 我们启动服务提供者的技巧(并行启动)

1）我们的tulingvip02-ms-alibaba-ribbon-product的端口是8081,我们用8081启动一个服务后，然后修改端口为8082,然后修改下图，那么久可以同一个工程启动二个实例.



1.5)Ribbon的内置的负载均衡算法



①:RandomRule（随机选择一个Server）

②: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.

③:RoundRobinRule

轮询选择， 轮询index，选择index对应位置的Server

④: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

⑤: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。

⑥: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；

⑦:ZoneAvoidanceRule（默认是这个）

复合判断Server所在Zone的性能和Server的可用性选择Server，在没有Zone的情况下类是轮询。

1.6)Ribbon的细粒度自定义配置

场景:我订单中心需要采用随机算法调用 库存中心

而采用轮询算法调用其他中心微服务。

基于java代码细粒度配置

注意点

我们针对调用具体微服务的具体配置类

ProductCenterRibbonConfig,OtherCenterRibbonConfig不能被放在我们主启动类所在包以及子包下，不然就起不到细粒度配置.

```
@Configuration
@RibbonClients(value = {
 @RibbonClient(name = "product-center",configuration = ProductCenterRibbonConfig.class),
 @RibbonClient(name = "pay-center",configuration = PayCenterRibbonConfig.class)
})
public class CustomRibbonConfig {

}

@Configuration
public class ProductCenterRibbonConfig {

 @Bean
 public IRule randomRule() {
 return new RandomRule();
 }
}

@Configuration
public class PayCenterRibbonConfig {
```

```
21
22   public IRule roundRobinRule() {
23   return new RoundRobinRule();
24   }
25  }
```

基于yml配置:(我们可以在order-center的yml中进行配置)

配置格式的语法如下

serviceName:

ribbon:

NFLoadBalancerRuleClassName: 负载均衡的对应class的全类名

配置案例: 我们的order-center调用我们的product-center

```
1  #自定义Ribbon的细粒度配置
2  product-center:
3   ribbon:
4   NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule
5  pay-center:
6   ribbon:
7   NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RoundRobinRule
```

推荐:推荐同学们使用yml的配置方法配置(没有坑)

极力不推荐同学们 使用二种配置混合使用

1.7)解决Ribbon 第一次调用耗时高的配置

开启饥饿加载

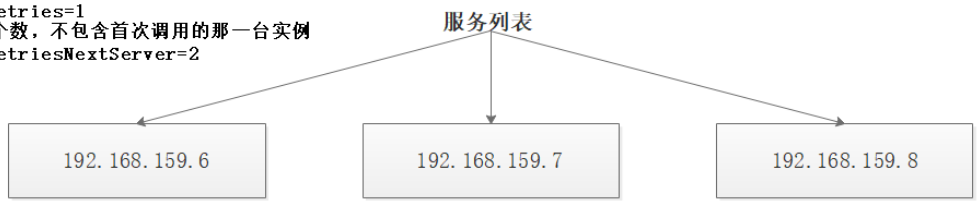
```
1  ribbon:
2   eager-load:
3   enabled: true
4   clients: product-center #可以指定多个微服务用逗号分隔
```

常用参数讲解:

```
1  每一台服务器重试的次数，不包含首次调用的那一次
2  ribbon.MaxAutoRetries=1
3
4  # 重试的服务器的个数，不包含首次调用的那一台实例
5  ribbon.MaxAutoRetriesNextServer=2
6
```

```
# 是否对所以的操作进行重试(True 的话 会对post put操作进行重试，存在服务幂等问题)
ribbon.OkToRetryOnAllOperations=true

# 建立连接超时
ribbon.ConnectTimeout=3000

# 读取数据超时
ribbon.ReadTimeout=3000

举列子：上面会进行几次重试
MaxAutoRetries
+
MaxAutoRetriesNextServer
+
(MaxAutoRetries *MaxAutoRetriesNextServer)
```

```
每一台服务器重试的次数，不包含首次调用的那一次
ribbon.MaxAutoRetries=1
# 重试的服务器的个数，不包含首次调用的那一台实例
ribbon.MaxAutoRetriesNextServer=2
```

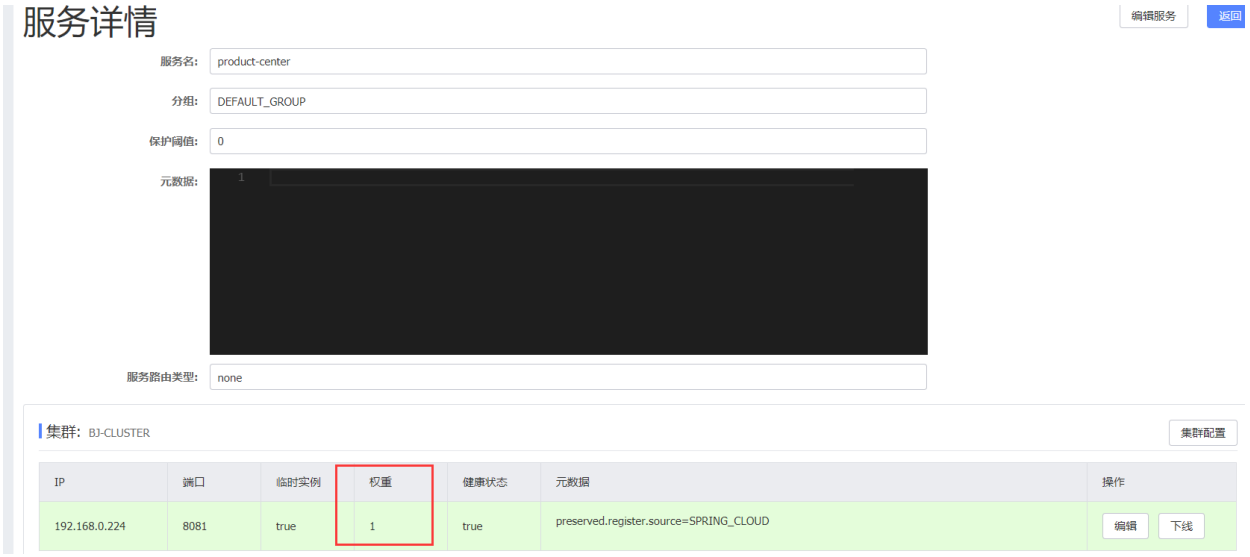


Ribbon详细配置:http://c.biancheng.net/view/5356.html

1.8)Ribbon 自定义负载均衡策略

我们发现，nacos server上的页面发现 注册的微服务有一个权重的概念。

取值为0-1之间



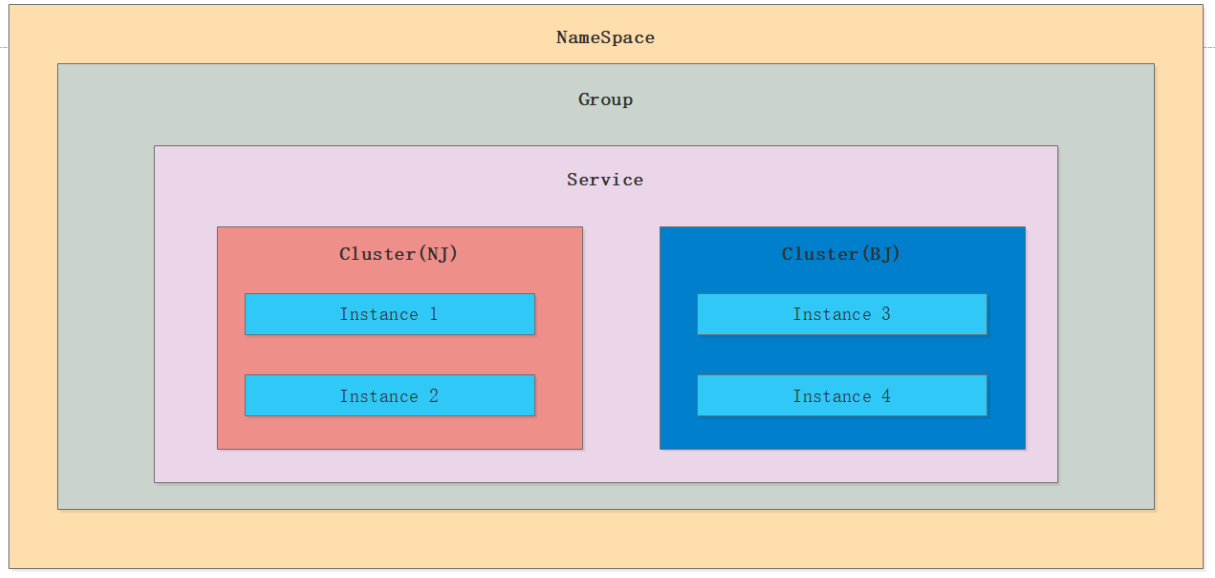
权重选择的概念: 假设我们一个微服务部署了三台服务器A,B,C 其中A,B,C三台服务的性能不一，A的性能最牛逼，B次之，C最差. 那么我们设置权重比例 为5：3:2 那就说明 10次请求到A上理论是5次,B服务上理论是3次,B服务理论是2次.

①:但是Ribbon 所提供的负载均衡算法中没有基于权重的负载均衡算法。我们自己实现一个.

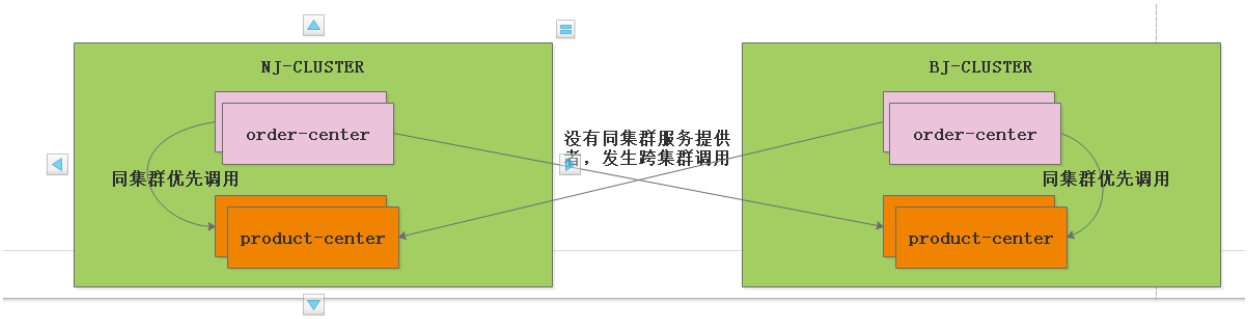
```
@Slf4j
public class TulingWeightedRule extends AbstractLoadBalancerRule {

 @Autowired
 private NacosDiscoveryProperties discoveryProperties;

 @Override
 public void initWithNiwsConfig(IClientConfig clientConfig) {
 //读取配置文件并且初始化,ribbon内部的 几乎用不上
  }


 @Override
 public Server choose(Object key) {
 try {
 log.info("key:{}",key);
 BaseLoadBalancer baseLoadBalancer = (BaseLoadBalancer) this.getLoadBalancer();
 log.info("baseLoadBalancer--->:{}",baseLoadBalancer);
```

```
19
20    //获取微服务的名称
21    String serviceName = baseLoadBalancer.getName();
22
23    //获取Nocas服务发现的相关组件API
24    NamingService namingService =
discoveryProperties.namingServiceInstance();
25
26    //获取 一个基于nacos client 实现权重的负载均衡算法
27    Instance instance =
namingService.selectOneHealthyInstance(serviceName);
28    //返回一个server
29    return new NacosServer(instance);
30    } catch (NacosException e) {
31    log.error("自定义负载均衡算法错误");
32    }
33    return null;
34    }
35  }
```

进阶版本1:我们第一节课上发现Nacos领域模型中有一个集群的概念同集群优先权重负载均衡算法



业务场景：现在我们有二个微服务order-center，product-center二个微服务。我们在南京机房部署一套order-center,product-center。为了容灾处理，我们在北京同样部署一套order-center,product-center



代码实现:

```
@Slf4j
public class TheSameClusterPriorityRule extends AbstractLoadBalancerRule
{

 @Autowired
 private NacosDiscoveryProperties discoveryProperties;

 @Override
 public void initWithNiwsConfig(IClientConfig clientConfig) {

 }

 @Override
 public Server choose(Object key) {
 try {
 //第一步:获取当前服务所在的集群
 String currentClusterName = discoveryProperties.getClusterName();

 //第二步:获取一个负载均衡对象
 BaseLoadBalancer baseLoadBalancer = (BaseLoadBalancer)
getLoadBalancer();

 //第三步:获取当前调用的微服务的名称
 String invokedSerivceName = baseLoadBalancer.getName();

 //第四步:获取nacos clinet的服务注册发现组件的api
 NamingService namingService =
discoveryProperties.namingServiceInstance();

 //第五步:获取所有的服务实例
 List<Instance> allInstance = namingService.getAllInstances(invokedSeriv
ceName);

```

```
30   List<Instance> theSameClusterNameInstList = new ArrayList<>();
31
32   //第六步:过滤筛选同集群下的所有实例
33   for(Instance instance : allInstance) {
34   if(StringUtils.endsWithIgnoreCase(instance.getClusterName(),currentClus
terName)) {
35   theSameClusterNameInstList.add(instance);
36   }
37   }
38
39   Instance toBeChooseInstance ;
40
41   //第七步:选择合适的一个实例调用
42   if(theSameClusterNameInstList.isEmpty()) {
43
44   toBeChooseInstance = TulingWeightedBalancer.chooseInstanceByRandomWeigh
t(allInstance);
45
46   log.info("发生跨集群调用--->当前微服务所在集群:{},被调用微服务所在集群:{},Ho
st:{},Port:{}",
47   currentClusterName,toBeChooseInstance.getClusterName(),toBeChooseInstan
ce.getIp(),toBeChooseInstance.getPort());
48   }else {
49   toBeChooseInstance = TulingWeightedBalancer.chooseInstanceByRandomWeigh
t(theSameClusterNameInstList);
50
51   log.info("同集群调用--->当前微服务所在集群:{},被调用微服务所在集群:{},Host:
{},Port:{}",
52   currentClusterName,toBeChooseInstance.getClusterName(),toBeChooseInstan
ce.getIp(),toBeChooseInstance.getPort());
53   }
54
55   return new NacosServer(toBeChooseInstance);
56
57   } catch (NacosException e) {
58   log.error("同集群优先权重负载均衡算法选择异常:{}",e);
59   }
60   return null;
61   }
62  }
63
64  /**
65   * 根据权重选择随机选择一个
```

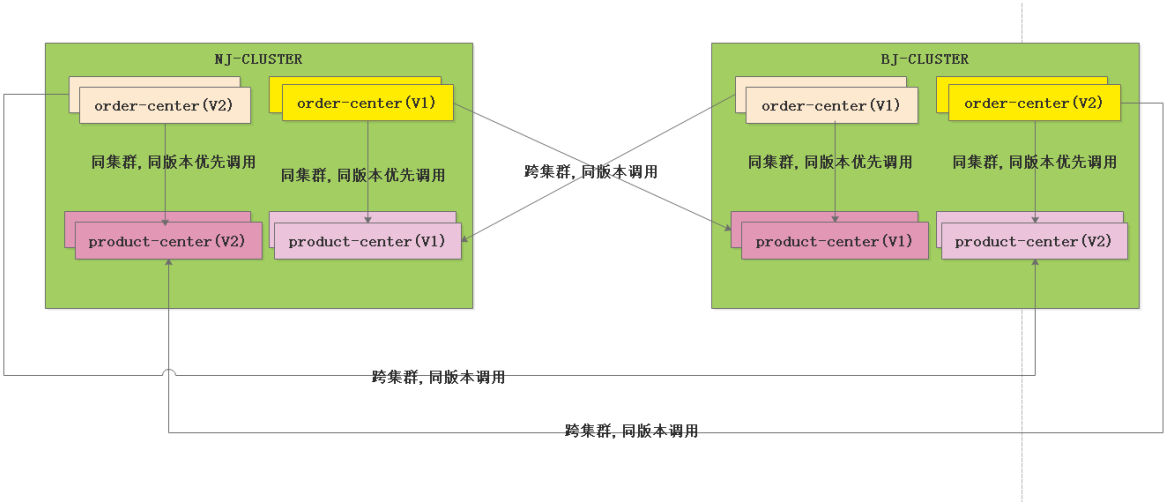
```
 * Created by smlz on 2019/11/21.
 */
public class TulingWeightedBalancer extends Balancer {

 public static Instance chooseInstanceByRandomWeight(List<Instance> hosts) {
 return getHostByRandomWeight(hosts);
 }
}
```

配置文件

```
spring
 cloud:
 nacos:
 discovery:
 server-addr: localhost:8848
 #配置集群名称
 cluster-name: NJ-CLUSTER
 #namespace: bc7613d2-2e22-4292-a748-48b78170f14c #指定namespace的id
 application:
 name: order-center
```

进阶版本2: 根据进阶版本1,我们现在需要解决生产环境金丝雀发布问题

比如 order-center 存在二个版本 V1(老版本) V2（新版本）,product-center也存在二个版本V1(老版本) V2新版本 现在需要做到的是 order-center(V1)---->product-center（v1），order-center(V2)---->product-center（v2）。记住v2版本是小面积部署的，用来测试用户对新版本功能的。若用户完全接受了v2。我们就可以把V1版本卸载完全部署V2版本。



代码实现思路：通过我们实例的元数据控制

```
/**
 * 同一个集群,同已版本号 优先调用策略
 * Created by smlz on 2019/11/21.
 */
@Slf4j
public class TheSameClusterPriorityWithVersionRule extends AbstractLoadBalancerRule {

 @Autowired
 private NacosDiscoveryProperties discoveryProperties;

 @Override
 public void initWithNiwsConfig(IClientConfig clientConfig) {

 }

 @Override
 public Server choose(Object key) {

 try {

 String currentClusterName = discoveryProperties.getClusterName();

 List<Instance> theSameClusterNameAndTheSameVersionInstList = getTheSameClusterAndTheSameVersionInstances(discoveryProperties);

 //声明被调用的实例
 Instance toBeChooseInstance;

 //判断同集群同版本号的微服务实例是否为空
```

```
29  if(theSameClusterNameAndTheSameVersionInstList.isEmpty()) {
30  //跨集群调用相同的版本
31  toBeChooseInstance = crossClusterAndTheSameVersionInovke(discoveryProperties);
32  }else {
33  toBeChooseInstance = TulingWeightedBalancer.chooseInstanceByRandomWeight(theSameClusterNameAndTheSameVersionInstList);
34  log.info("同集群同版本调用--->当前微服务所在集群:{},被调用微服务所在集群:{},当前微服务的版本:{},被调用微服务版本:{},Host:{},Port:{}",
35  currentClusterName,toBeChooseInstance.getClusterName(),discoveryProperties.getMetadata().get("current-version"),
36  toBeChooseInstance.getMetadata().get("current-version"),toBeChooseInstance.getIp(),toBeChooseInstance.getPort());
37  }
38
39  return new NacosServer(toBeChooseInstance);
40
41  } catch (NacosException e) {
42  log.error("同集群优先权重负载均衡算法选择异常:{}",e);
43  return null;
44  }
45  }
46
47
48
49  /**
50  * 方法实现说明:获取相同集群下,相同版本的 所有实例
51  * @author:smlz
52  * @param discoveryProperties nacos的配置
53  * @return: List<Instance>
54  * @exception: NacosException
55  * @date:2019/11/21 16:41
56  */
57  private List<Instance> getTheSameClusterAndTheSameVersionInstances(NacosDiscoveryProperties discoveryProperties) throws NacosException {
58
59  //当前的集群的名称
60  String currentClusterName = discoveryProperties.getClusterName();
61
62  String currentVersion = discoveryProperties.getMetadata().get("current-version");
63
```

```
64   //获取所有实例的信息(包括不同集群的)
65   List<Instance> allInstance = getAllInstances(discoveryProperties);
66
67   List<Instance> theSameClusterNameAndTheSameVersionInstList = new ArrayL
ist<>();
68
69   //过滤相同集群的
70   for(Instance instance : allInstance) {
71   if(StringUtils.endsWithIgnoreCase(instance.getClusterName(),currentClus
terName)&&
72   StringUtils.endsWithIgnoreCase(instance.getMetadata().get("current-vers
ion"),currentVersion)) {
73
74   theSameClusterNameAndTheSameVersionInstList.add(instance);
75   }
76   }
77
78   return theSameClusterNameAndTheSameVersionInstList;
79   }
80
81   /**
82   * 方法实现说明:获取被调用服务的所有实例
83   * @author:smlz
84   * @param discoveryProperties nacos的配置
85   * @return: List<Instance>
86   * @exception: NacosException
87   * @date:2019/11/21 16:42
88   */
89   private List<Instance> getAllInstances(NacosDiscoveryProperties discove
ryProperties) throws NacosException {
90
91   //第1步:获取一个负载均衡对象
92   BaseLoadBalancer baseLoadBalancer = (BaseLoadBalancer)
getLoadBalancer();
93
94   //第2步:获取当前调用的微服务的名称
95   String invokedSerivceName = baseLoadBalancer.getName();
96
97   //第3步:获取nacos clinet的服务注册发现组件的api
98   NamingService namingService =
discoveryProperties.namingServiceInstance();
99
```

```
100   //第4步:获取所有的服务实例
101   List<Instance> allInstance = namingService.getAllInstances(invokedSeriv
ceName);
102
103   return allInstance;
104   }
105
106   /**
107   * 方法实现说明:跨集群环境下 相同版本的
108   * @author:smlz
109   * @param discoveryProperties
110   * @return: List<Instance>
111   * @exception: NacosException
112   * @date:2019/11/21 17:11
113   */
114   private List<Instance> getCrossClusterAndTheSameVersionInstList(NacosDi
scoveryProperties discoveryProperties) throws NacosException {
115
116   //版本号
117   String currentVersion = discoveryProperties.getMetadata().get("current-
version");
118
119   //被调用的所有实例
120   List<Instance> allInstance = getAllInstances(discoveryProperties);
121
122   List<Instance> crossClusterAndTheSameVersionInstList = new ArrayList<>
();
123
124   //过滤相同版本
125   for(Instance instance : allInstance) {
126   if(StringUtils.endsWithIgnoreCase(instance.getMetadata().get("current-v
ersion"),currentVersion)) {
127
128   crossClusterAndTheSameVersionInstList.add(instance);
129   }
130   }
131
132   return crossClusterAndTheSameVersionInstList;
133   }
134
135   private Instance crossClusterAndTheSameVersionInovke(NacosDiscoveryProp
erties discoveryProperties) throws NacosException {
```

```
136
137   //获取所有集群下相同版本的实例信息
138   List<Instance> crossClusterAndTheSameVersionInstList = getCrossClusterA
ndTheSameVersionInstList(discoveryProperties);
139   //当前微服务的版本号
140   String currentVersion = discoveryProperties.getMetadata().get("current-
version");
141   //当前微服务的集群名称
142   String currentClusterName = discoveryProperties.getClusterName();
143
144   //声明被调用的实例
145   Instance toBeChooseInstance = null ;
146
147   //没有对应相同版本的实例
148   if(crossClusterAndTheSameVersionInstList.isEmpty()) {
149   log.info("跨集群调用找不到对应合适的版本当前版本为:currentVersion:{}",curr
entVersion);
150   throw new RuntimeException("找不到相同版本的微服务实例");
151   }else {
152   toBeChooseInstance = TulingWeightedBalancer.chooseInstanceByRandomWeigh
t(crossClusterAndTheSameVersionInstList);
153
154   log.info("跨集群同版本调用--->当前微服务所在集群:{},被调用微服务所在集群:
{},当前微服务的版本:{},被调用微服务版本:{},Host:{},Port:{}",
155   currentClusterName,toBeChooseInstance.getClusterName(),discoveryPropert
ies.getMetadata().get("current-version"),
156   toBeChooseInstance.getMetadata().get("current-version"),toBeChooseInsta
nce.getIp(),toBeChooseInstance.getPort());
157   }
158
159   return toBeChooseInstance;
160   }
161 }
```

配置文件说明

order-center的yml的配置

```
1 spring:
2   cloud:
3   nacos:
4   discovery:
5   server-addr: localhost:8848
6   #所在集群
7   cluster-name: NJ-CLUSTER
```

```
  metadata:
  #版本号
   current-version: V1
   #namespace: bc7613d2-2e22-4292-a748-48b78170f14c #指定namespace的id
   application:
   name: order-center
```

product-center的yml配置说明:

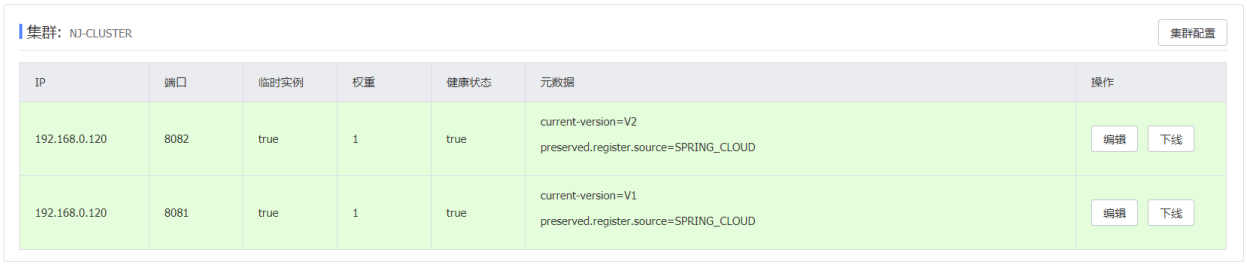
NJ-CLUSTER下的V1版本

```
spring:
 application:
 name: product-center
 cloud:
 nacos:
 discovery:
 server-addr: localhost:8848
 cluster-name: NJ-CLUSTER
 metadata:
  current-version: V1
  #namespace: 20989a73-cdb3-41b8-85c0-e9a3530e28a6
server:
 port: 8084
```

NJ-CLUSTER下的V2版本

```
spring:
 application:
 name: product-center
 cloud:
 nacos:
 discovery:
 server-addr: localhost:8848
 cluster-name: NJ-CLUSTER
 metadata:
  current-version: V2
  #namespace: 20989a73-cdb3-41b8-85c0-e9a3530e28a6
```

```
server:
 port: 8083
```



BJ-CLUSTER下的V1版本

```
spring:
 application:
 name: product-center
 cloud:
 nacos:
 discovery:
 server-addr: localhost:8848
 cluster-name: BJ-CLUSTER
 metadata:
  current-version: V1
  #namespace: 20989a73-cdb3-41b8-85c0-e9a3530e28a6
server:
 port: 8082
```

BJ-CLUSTER下的V2版本

```
spring:
 application:
 name: product-center
 cloud:
 nacos:
 discovery:
 server-addr: localhost:8848
 cluster-name: BJ-CLUSTER
 metadata:
  current-version: V2
  #namespace: 20989a73-cdb3-41b8-85c0-e9a3530e28a6
server:
 port: 8081
```

集群：BJ-CLUSTER　　集群配置

IP	端口	临时实例	权重	健康状态	元数据	操作
192.168.0.120	8084	true	1	true	current-version=V2 preserved.register.source=SPRING_CLOUD	编辑 下线
192.168.0.120	8083	true	1	true	current-version=V1 preserved.register.source=SPRING_CLOUD	编辑 下线

测试说明：**从我们的order-center调用product-center的时候 优先会调用同集群同版本的**

2019-11-21 22:42:42.317 INFO 45980 --- [nio-8080-exec-3] .m.TheSameClusterPriorityWithVersionRule : 同集群同版本调用--->当前微服务所在集群:NJ-CLUSTER,被调用微服务所在集群:NJ-CLUSTER,当前微服务的版本:V1,被调用微服务版本:V1,Host:192.168.0.120,Port:8081

若我们把同集群的 同版本的product-center下线,那们就会发生跨集群调用相同的版本:

集群：NJ-CLUSTER　　集群配置

IP	端口	临时实例	权重	健康状态	元数据	操作
192.168.0.120	8082	true	1	true	current-version=V2 preserved.register.source=SPRING_CLOUD	编辑 下线
192.168.0.120	8081	true	1	true	current-version=V1 preserved.register.source=SPRING_CLOUD	编辑 上线

2019-11-21 22:44:48.723 INFO 45980 --- [nio-8080-exec-6] .m.TheSameClusterPriorityWithVersionRule : 跨集群同版本调用--->当前微服务所在集群:NJ-CLUSTER,被调用微服务所在集群:BJ-CLUSTER,当前微服务的版本:V1,被调用微服务版本:V1,Host:192.168.0.120,Port:8083