微服务负载均衡器Ribbon

1.什么是Ribbon

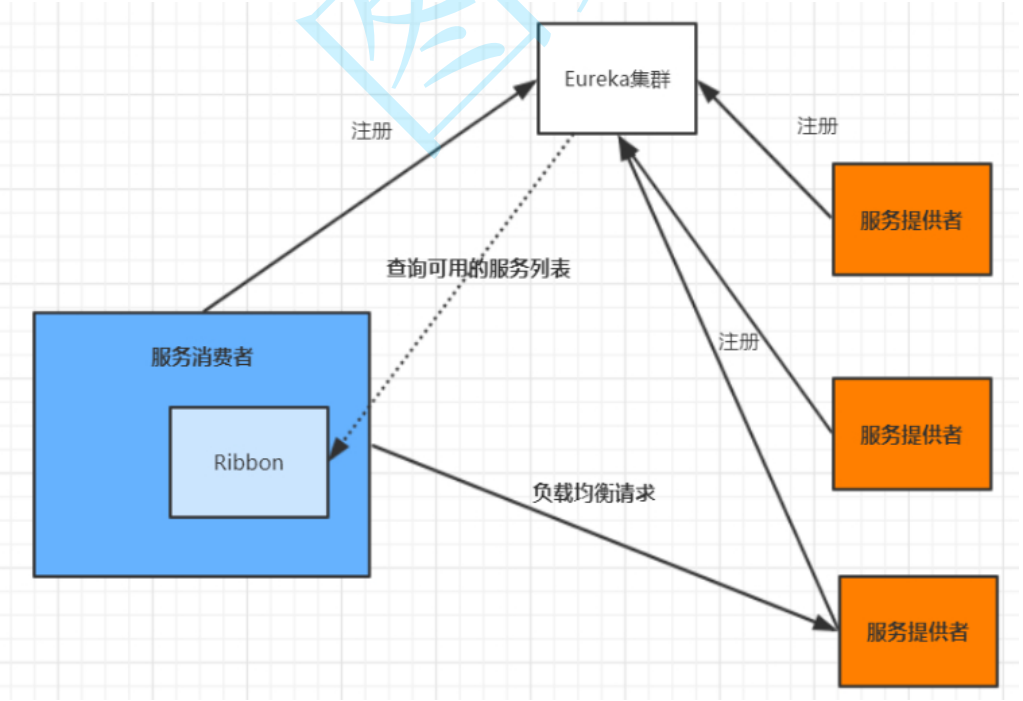
目前主流的负载方案分为以下两种：

- 集中式负载均衡，在消费者和服务提供方中间使用独立的代理方式进行负载，有硬件的（比如 F5），也有软件的（比如 Nginx）。
- 客户端根据自己的请求情况做负载均衡，Ribbon 就属于客户端自己做负载均衡。

Spring Cloud Ribbon是基于Netflix 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。

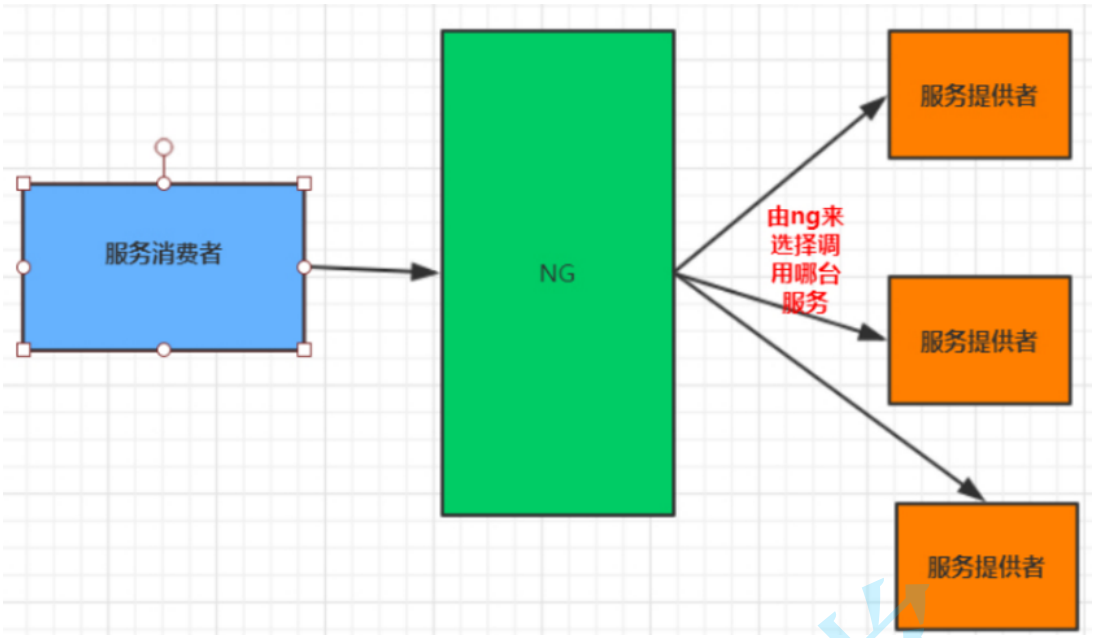
1.1 客户端的负载均衡

例如spring cloud中的ribbon，客户端会有一个服务器地址列表，在发送请求前通过负载均衡算法选择一个服务器，然后进行访问，这是客户端负载均衡；即在客户端就进行负载均衡算法分配。



1.2 服务端的负载均衡

例如Nginx，通过Nginx进行负载均衡，先发送请求，然后通过负载均衡算法，在多个服务器之间选择一个进行访问；即在服务器端再进行负载均衡算法分配。



1.3 常见负载均衡算法

- 随机，通过随机选择服务进行执行，一般这种方式使用较少;
- 轮训，负载均衡默认实现方式，请求来之后排队处理;
- 加权轮训，通过对服务器性能的分型，给高配置，低负载的服务器分配更高的权重，均衡各个服务器的压力;
- 地址Hash，通过客户端请求的地址的HASH值取模映射进行服务器调度。 ip --->hash
- 最小链接数，即使请求均衡了，压力不一定会均衡，最小连接数法就是根据服务器的情况，比如请求积压数等参数，将请求分配到当前压力最小的服务器上。 最小活跃数

2. Nacos使用Ribbon

nacos-discovery依赖了ribbon，可以不用再引入ribbon依赖

- com.alibaba.cloud:spring-cloud-starter-alibaba-nacos-discovery:2.2.1.RELEASE
  - com.alibaba.nacos:nacos-client:1.2.1
  - com.alibaba.spring:spring-context-support:1.0.6
  - org.springframework.cloud:spring-cloud-commons:2.2.2.RELEASE
  - org.springframework.cloud:spring-cloud-context:2.2.2.RELEASE
  - org.springframework.cloud:spring-cloud-starter-netflix-ribbon:2.2.2.RELEASE

2) 添加@LoadBalanced注解

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

3) 修改controller

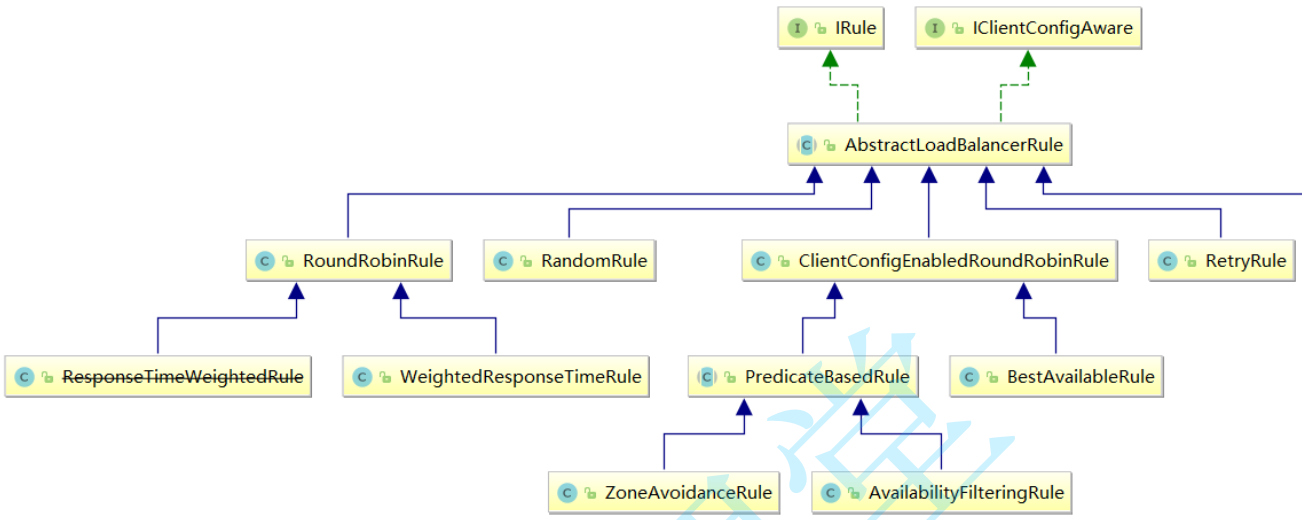
```
@Autowired
private RestTemplate restTemplate;

@RequestMapping(value = "/findOrderByUserId/{id}")
public R findOrderByUserId(@PathVariable("id") Integer id) {
 // RestTemplate调用
 //String url = "http://localhost:8020/order/findOrderByUserId/"+id;
 //模拟ribbon实现
 //String url = getUri("mall-order")+"/order/findOrderByUserId/"+id;
```

```
// 添加@LoadBalanced
String url = "http://mall-order/order/findOrderByUserId/"+id;
R result = restTemplate.getForObject(url,R.class);

return result;
}
```

3 Ribbon负载均衡策略



IRule

这是所有负载均衡策略的父接口，里边的核心方法就是choose方法，用来选择一个服务实例。

AbstractLoadBalancerRule

AbstractLoadBalancerRule是一个抽象类，里边主要定义了一个ILoadBalancer，这里定义它的目的主要是辅助负责均衡策略选取合适的服务端实例。

- **RandomRule**

看名字就知道，这种负载均衡策略就是**随机选择一个服务实例**，看源码我们知道，在RandomRule的无参构造方法中初始化了一个Random对象，然后在它重写的choose方法又调用了choose(ILoadBalancer lb, Object key)这个重载的choose方法，在这个重载的choose方法中，每次利用random对象生成一个不大于服务实例总数的随机数，并将该数作为下标所以获取一个服务实例。

- **RoundRobinRule**

RoundRobinRule这种负载均衡策略叫做线性**轮询负载均衡策略**。这个类的choose(ILoadBalancer lb, Object key)函数整体逻辑是这样的：开启一个计数器count，在while循环中遍历服务清单，获取清单之前先通过incrementAndGetModulo方法获取一个下标，这个下标是一个不断自增长的数先加1然后和服务清单总数取模之后获取到的（所以这个下标从来不会越界），拿着下标再去服务清单列表中取服务，每次循环计数器都会加1，如果连续10次都没有取到服务，则会报一个警告No available alive servers after 10 tries from load balancer: XXXX。

- **RetryRule**（在轮询的基础上进行重试）

看名字就知道这种负载均衡策略带有**重试**功能。首先RetryRule中又定义了一个subRule，它的实现类是RoundRobinRule，然后在RetryRule的choose(ILoadBalancer lb, Object key)方法中，每次还是采用RoundRobinRule中的choose规则来选择一个服务实例，如果选到的实例正常就返回，如果选择的服务实例为null或者已经失效，则在失效时间deadline之前不断的进行重试（重试时获取服务的策略还是RoundRobinRule中定义的策略），如果超过了deadline还是没取到则会返回一个null。

- **WeightedResponseTimeRule**（权重 —nacos的NacosRule，Nacos还扩展了一个自己的基于配置的权重扩展）

WeightedResponseTimeRule是RoundRobinRule的一个子类，在WeightedResponseTimeRule中对RoundRobinRule的功能进行了扩展，WeightedResponseTimeRule中会根据每一个实例的运行情况来给计算出该实例的一个**权重**，然后在挑选实例的时候则根据权重进行挑选，这样能够实现更优的实例调用。WeightedResponseTimeRule中有一个名叫DynamicServerWeightTask的定时任务，默认情况下每隔30秒会计算一次各个服务实例的权重，权重的计算规则也很简单，**如果一个服务的平均响应时间越短则权重越大，那么该服务实例被选中执行任务的概率也就越大**。

- **ClientConfigEnabledRoundRobinRule**

ClientConfigEnabledRoundRobinRule选择策略的实现很简单，内部定义了RoundRobinRule，choose方法还是采用了RoundRobinRule的choose方法，所以它的选择策略**和RoundRobinRule的选择策略一致**，不赘述。

- **BestAvailableRule**

BestAvailableRule继承自ClientConfigEnabledRoundRobinRule，它在ClientConfigEnabledRoundRobinRule的基础上主要增加了根据loadBalancerStats中保存的服务实例的状态信息来**过滤掉失效的服务实例的功能，然后顺便找出并发请求最小的服务实例来使用。**然而loadBalancerStats有可能为null，如果loadBalancerStats为null，则BestAvailableRule将采用它的父类即ClientConfigEnabledRoundRobinRule的服务选取策略（线性轮询）。

- **ZoneAvoidanceRule**（**默认规则**，复合判断server所在区域的性能和server的可用性选择服务器。）

ZoneAvoidanceRule是PredicateBasedRule的一个实现类，只不过这里多一个过滤条件，ZoneAvoidanceRule中的过滤条件是以ZoneAvoidancePredicate为主过滤条件和以AvailabilityPredicate为次过滤条件组成的一个叫做CompositePredicate的组合过滤条件，过滤成功之后，继续采用线性轮询（**RoundRobinRule**）的方式从过滤结果中选择一个出来。

- **AvailabilityFilteringRule**（先过滤掉故障实例，再选择并发较小的实例）

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，其实就是检查status里记录的各个Server的运行状态。

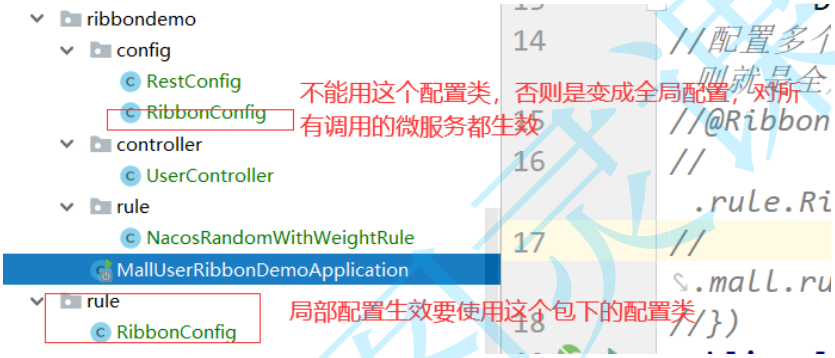
3.2.1 修改默认负载均衡策略

1.配置类:

```
@Configuration
public class RibbonConfig {

 /**
 * 全局配置
 * 指定负载均衡策略
 * @return
 */
 @Bean
  public IRule iRule() {
  // 指定使用Nacos提供的负载均衡策略（优先调用同一集群的实例，基于随机权重）
  return new NacosRule();
  }
}
```

注意：此处有坑。不能写在@SpringbootApplication注解的@CompentScan扫描得到的地方，否则自定义的配置类就会被所有的RibbonClients共享。 不建议这么使用，推荐yml方式



Spring Cloud also lets you take full control of the client by declaring additional configuration (on top of the `RibbonClientConfiguration`) using `@RibbonClient`, as shown in the following example:

```
@Configuration
@RibbonClient(name = "custom", configuration = CustomConfiguration.class)
public class TestConfiguration {
}
```

In this case, the client is composed from the components already in `RibbonClientConfiguration`, together with any in `CustomConfiguration` (where the latter generally overrides the former).

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`).

利用@RibbonClient指定微服务及其负载均衡策略。

```
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class,
 DruidDataSourceAutoConfigure.class})
//@RibbonClient(name = "mall-order",configuration = RibbonConfig.class)
//配置多个 RibbonConfig不能被@SpringbootApplication的@CompentScan扫描到，否则就是全局配置的效果
@RibbonClients(value = {
 // 在SpringBoot主程序扫描的包外定义配置类
 @RibbonClient(name = "mall-order",configuration = RibbonConfig.class),
 @RibbonClient(name = "mall-account",configuration = RibbonConfig.class)
```

```
})
public class MallUserRibbonDemoApplication {

 public static void main(String[] args) {
 SpringApplication.run(MallUserRibbonDemoApplication.class, args);
 }
}
```

配置文件：调用指定微服务提供的服务时，使用对应的负载均衡算法

修改application.yml

```
# 被调用的微服务名
mall-order:
 ribbon:
 # 指定使用Nacos提供的负载均衡策略（优先调用同一集群的实例，基于随机&权重）
 NFLoadBalancerRuleClassName: com.alibaba.cloud.nacos.ribbon.NacosRule
```

3.2.2 自定义负载均衡策略

通过实现 IRule 接口可以自定义负载策略，主要的选择服务逻辑在 choose 方法中。

1）实现基于Nacos权重的负载均衡策略

```
@Slf4j
public class NacosRandomWithWeightRule extends AbstractLoadBalancerRule {

 @Autowired
 private NacosDiscoveryProperties nacosDiscoveryProperties;

 @Override
 public Server choose(Object key) {
 DynamicServerListLoadBalancer loadBalancer = (DynamicServerListLoadBalancer) getLoadBalancer();
 String serviceName = loadBalancer.getName();
 NamingService namingService = nacosDiscoveryProperties.namingServiceInstance();
 try {
 //nacos基于权重的算法
 Instance instance = namingService.selectOneHealthyInstance(serviceName);
 return new NacosServer(instance);
 } catch (NacosException e) {
 log.error("获取服务实例异常: {}", e.getMessage());
 e.printStackTrace();
 }
 return null;
 }
 @Override
 public void initWithNiwsConfig(IClientConfig clientConfig) {

 }
}
```

2) 配置自定义的策略

2.1）配置文件:

修改application.yml

```
# 被调用的微服务名
mall-order:
 ribbon:
 # 自定义的负载均衡策略（基于随机&权重）
 NFLoadBalancerRuleClassName: com.tuling.mall.ribbondemo.rule.NacosRandomWithWeightRule
```

3.3 饥饿加载

在进行服务调用的时候，如果网络情况不好，第一次调用会超时。

Ribbon默认懒加载，意味着只有在发起调用的时候才会创建客户端。

```
c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServl
web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
web.servlet.DispatcherServlet        : Completed initialization in 6 ms
tflix.loadbalancer.BaseLoadBalancer  : Client: mall-order instantiated a LoadBalancer: Dynami
l.DynamicServerListLoadBalancer      : Using serverListUpdater PollingServerListUpdater
l.DynamicServerListLoadBalancer      : DynamicServerListLoadBalancer for client mall-order in
  Total Requests:0;   Successive connection failure:0;    Total blackout seconds:0;   Last co
:0;   Successive connection failure:0;    Total blackout seconds:0;   Last connection made:Th
:0;   Successive connection failure:0;    Total blackout seconds:0;   Last connection made:Th
5723d
ibaba.cloud.nacos.ribbon.NacosRule   : A cross-cluster call occurs, name = mall-order, cluste
ibaba.cloud.nacos.ribbon.NacosRule   : A cross-cluster call occurs, name = mall-order, cluste
```

第一次调用

开启饥饿加载，解决第一次调用慢的问题

```yaml
ribbon:
  eager-load:
    # 开启ribbon饥饿加载
    enabled: true
    # 配置mall-user使用ribbon饥饿加载，多个使用逗号分隔
    clients: mall-order
```

源码对应属性配置类：RibbonEagerLoadProperties

测试：

```
[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherS
rvlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
rvlet.DispatcherServlet        : Completed initialization in 6 ms
cloud.nacos.ribbon.NacosRule   : A cross-cluster call occurs, name = mall-order, cl
```

第一次调用

3. Ribbon内核原理

3.1 Ribbon原理