课时3

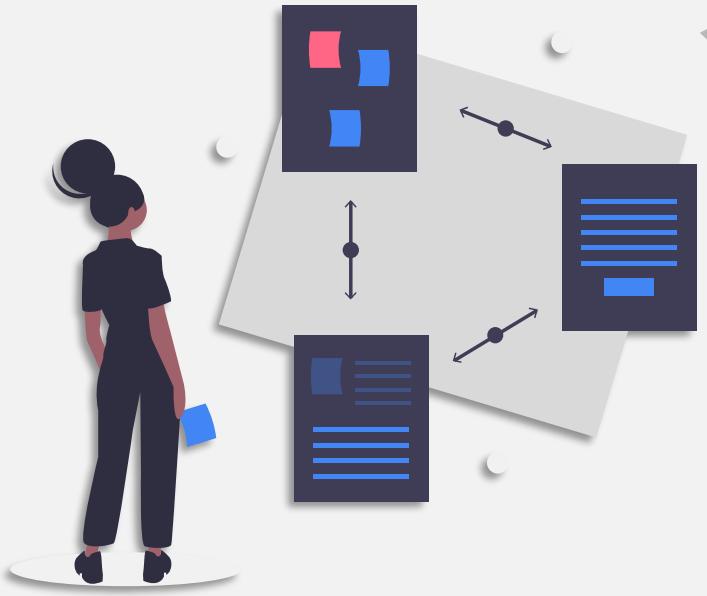
客户端负载均衡-Ribbon

1. 集中式负载均衡
2. 客户端负载均衡
3. Ribbon
4. 负载均衡策略
5. 饥饿加载模式

负载均衡介绍

拉勾
LAGOU.COM

随着用户量的增加，应用访问量也会随之增加，单台服务器已经不能满足高并发的业务需求



这时就需要多台服务器组成集群来应对高并发带来的业务的压力

同时也需要负均衡器来对流量进行合理分配

负载均衡介绍

目前主流的负载方案

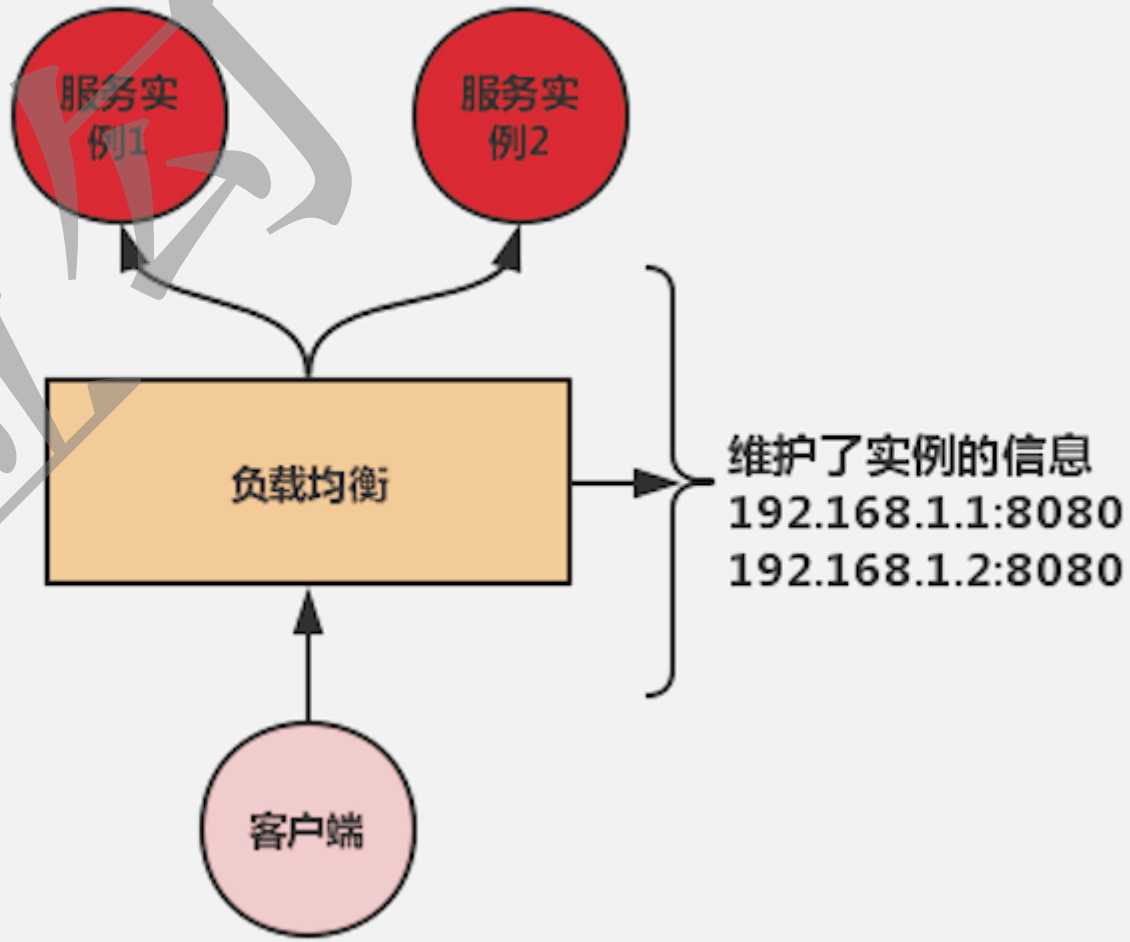
1. 一种是集中式负载均衡，在消费者和服务提供方中间使用独立的代理方式进行负载，有硬件的，比如F5，也有软件的，如Nginx

2. 另一种则是客户端自己做负载均衡，根据自己的请求情况做负载，Ribbon就是属于客户端自己做负载的框架

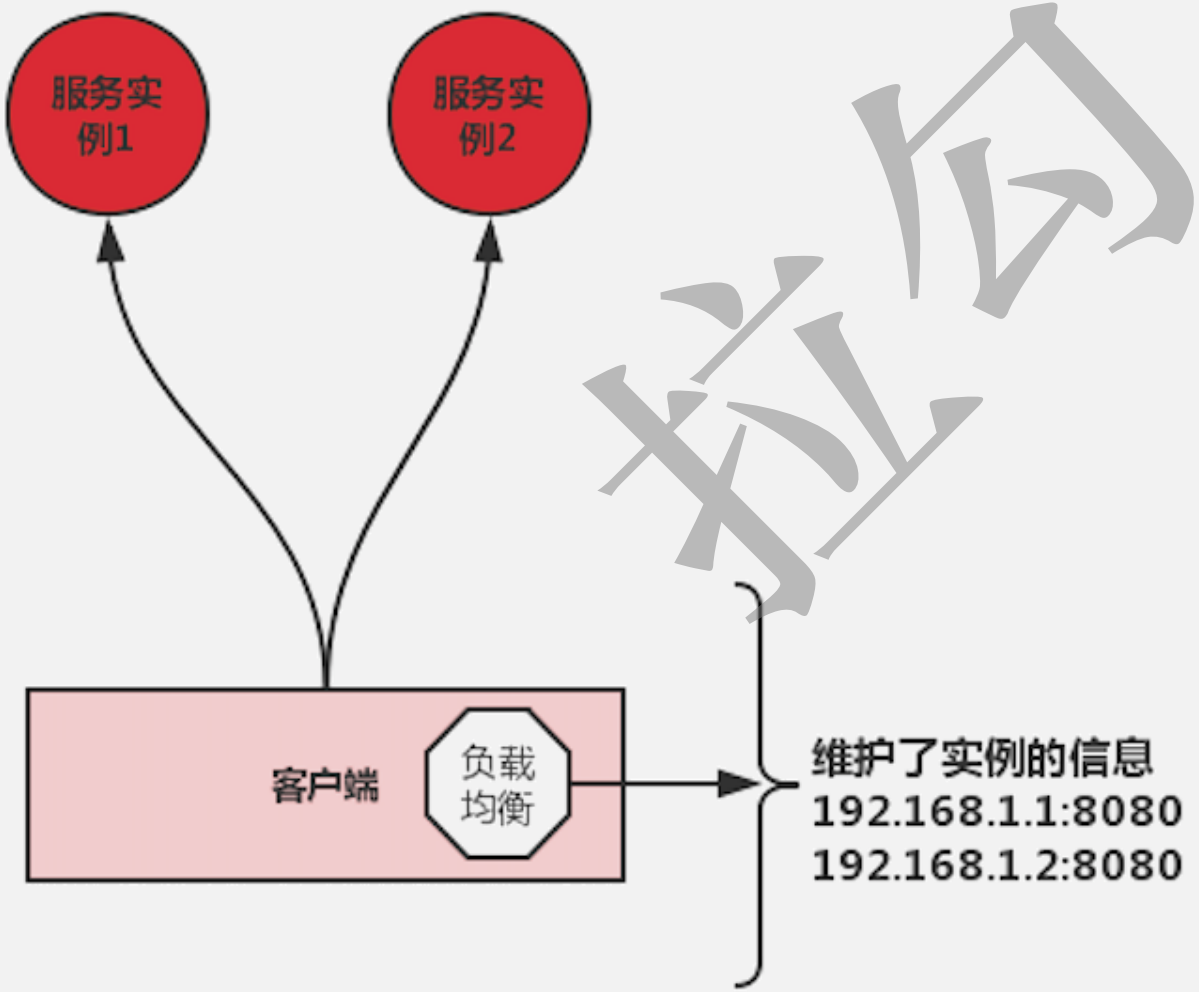
负载均衡详细介绍

拉勾
LAGOU.COM

集中式负载均衡



负载均衡详细介绍



客户端负载均衡

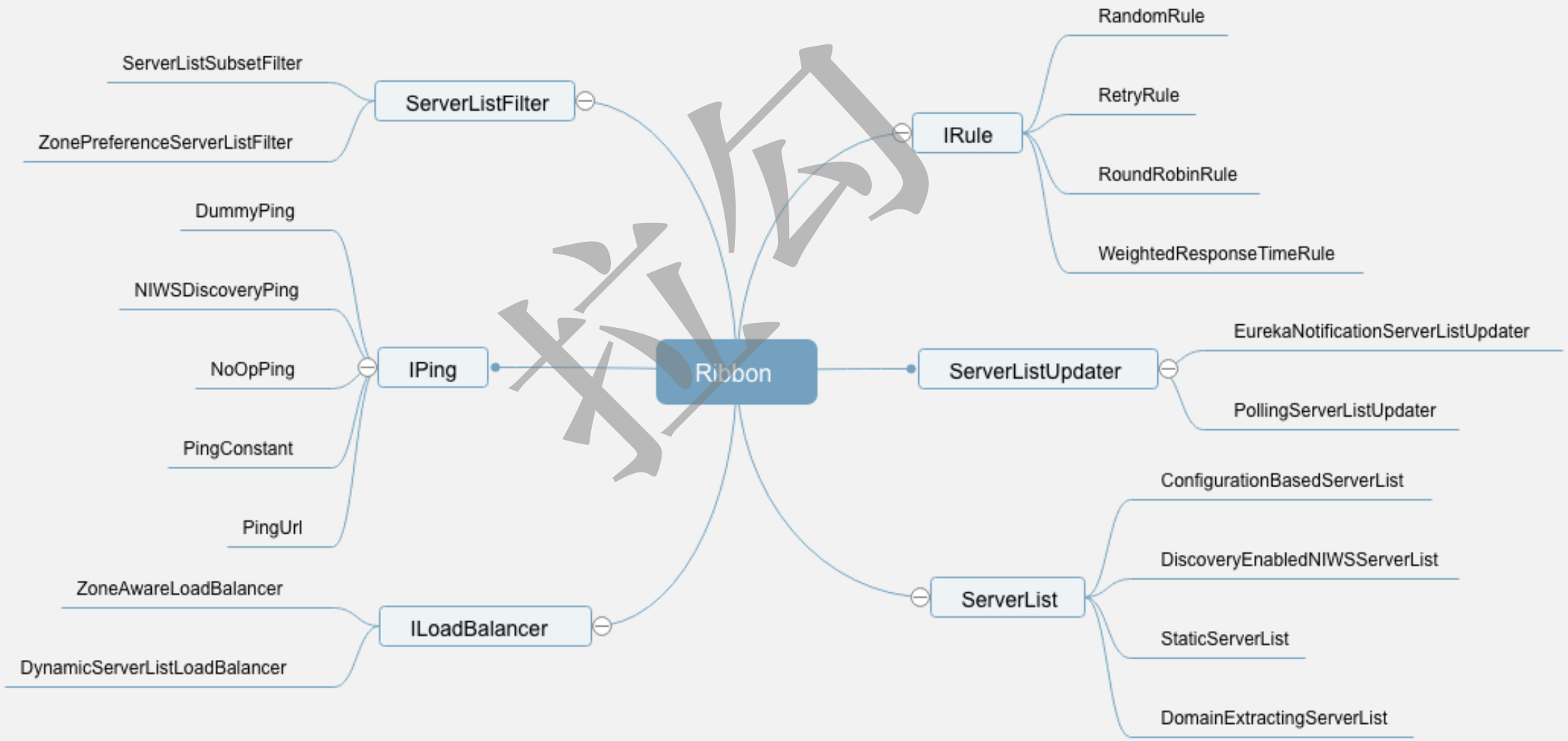
Ribbon简介

拉勾 LAGOU.COM

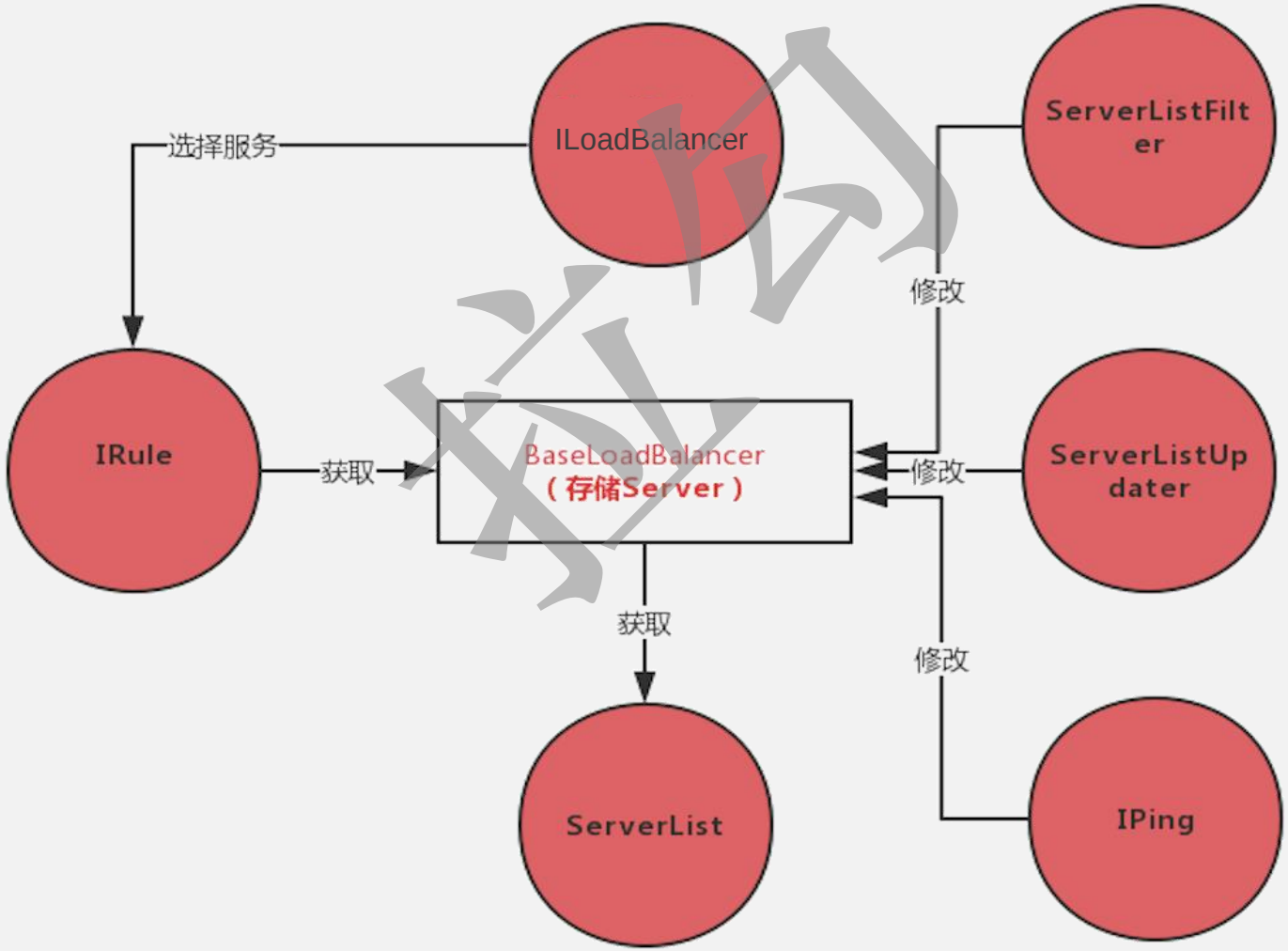
Ribbon

- 是Netflix发布的负载均衡器，它有助于控制HTTP和TCP的客户端的行为
- 为Ribbon配置服务提供者地址后，Ribbon就可基于某种负载均衡算法，自动地帮助服务消费者去请求
- Ribbon默认为我们提供了很多负载均衡算法，例如轮询、随机等

Ribbon的主要组件



服务实例调用流程图



各组件职责

拉勾 LAGOU.COM

组件	作用
LoadBalancer	定义了一系列的操作接口，比如选择服务实例
IRule	算法策略，内置了很多的算法策略来为服务实例的选择提供服务
ServerList	负责服务实例信息的获取，可以获取配置文件中的，也可以从注册中心获取
ServerListFilter	可以过滤掉某些不想要的服务实例信息
ServerListUpdater	负责更新本地缓存的服务实例信息
IPing	对已有的服务实例进行可用性检查，保证选择到的服务都是可用的

Ribbon使用方式

拉勾 LAGOU.COM

- 原生 API

Ribbon 是 Netflix 开源的，如果你没有使用 Spring Cloud，也可以在项目中单独使用 Ribbon，在这种场景下就需要使用 Ribbon 的原生 API

- Ribbon + RestTemplate

当我们项目整合了 Spring Cloud 时，就可以用 Ribbon 为 RestTemplate 提供负载均衡的功能

- Ribbon + Feign

这也是最常用的一种方式，Feign 的使用会在后面的章节中进行详细的讲解

Ribbon原生API使用

拉勾 LAGOU.COM

```
// 服务列表
List<Server> serverList = Arrays.asList(new Server("localhost", 8081), new Server("localhost", 8083));
// 构建负载实例
BaseLoadBalancer loadBalancer = LoadBalancerBuilder.newBuilder()
                .buildFixedServerListLoadBalancer(serverList);
// 负载策略为随机
loadBalancer.setRule(new RandomRule());
// 调用5次来测试效果
for (int i = 0; i < 5; i++) {
     String result = LoadBalancerCommand.<String>builder().withLoadBalancer(loadBalancer).build()
          .submit(new ServerOperation<String>() {
               public Observable<String> call(Server server) {
                   try {
                      String addr = "http://" + server.getHost() + ":" + server.getPort();
                      System.out.println("调用地址: " + addr);
                      return Observable.just("");
                   } catch (Exception e) {
                      return Observable.error(e);
                   }
                }
          }).toBlocking().first();
}
```

Ribbon结合RestTemplate使用

拉勾 LAGOU.COM

```
1. 配置RestTemplate
@Configuration
public class RestConfiguration {

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

}

2. RestTemplate使用服务名进行调用
@RestController
public class RibbonController {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/get")
    public Object get() {
        User user = restTemplate
          .getForEntity("http://user-service/user/get?id=1", User.class).getBody();
        return user;
    }

}

3. 配置
ribbon.eureka.enabled=false
user-service.ribbon.listOfServers=localhost:8084,localhost:8085
```

@LoadBalanced原理

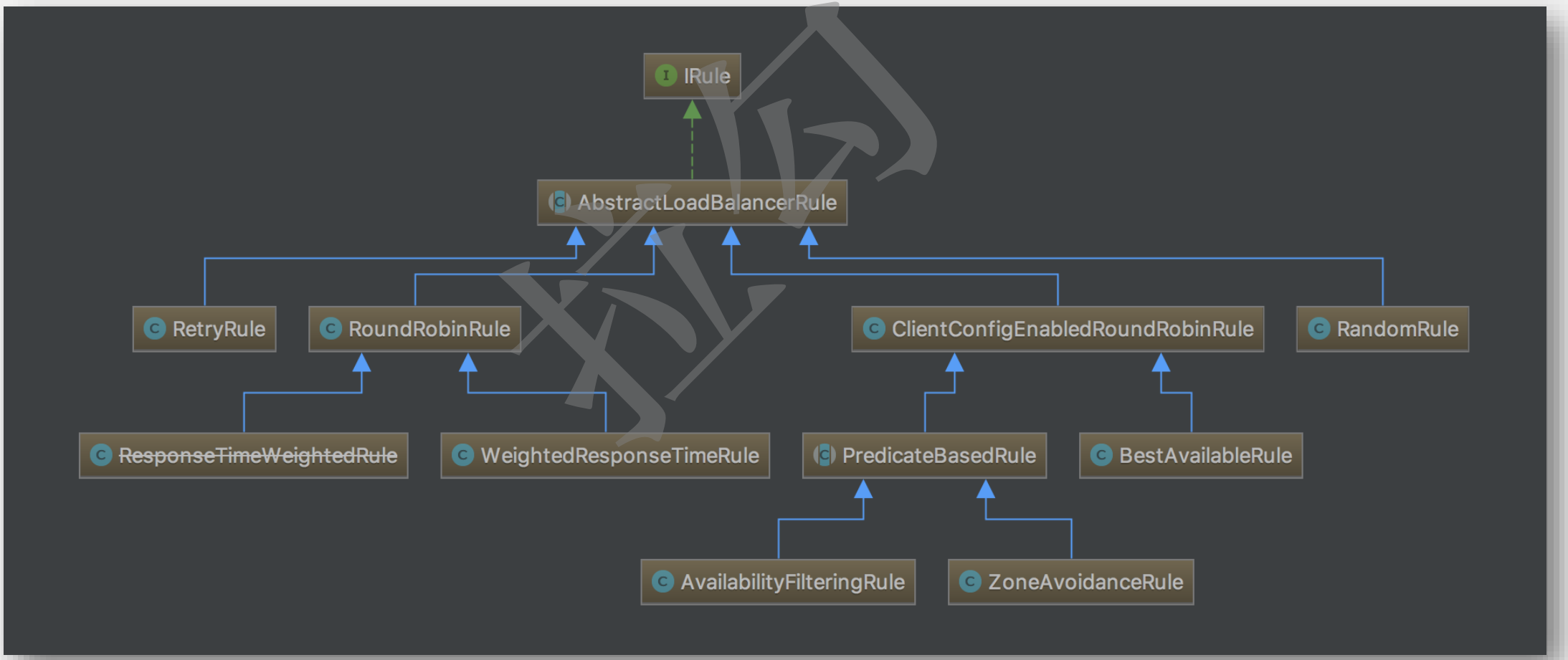
拉勾
LAGOU.COM

通过给加了@LoadBalanced的RestTemplate添加拦截器

拦截器中通过Ribbon选取服务实例

然后将请求地址中的服务名称替换成Ribbon选取服务实例的IP和端口

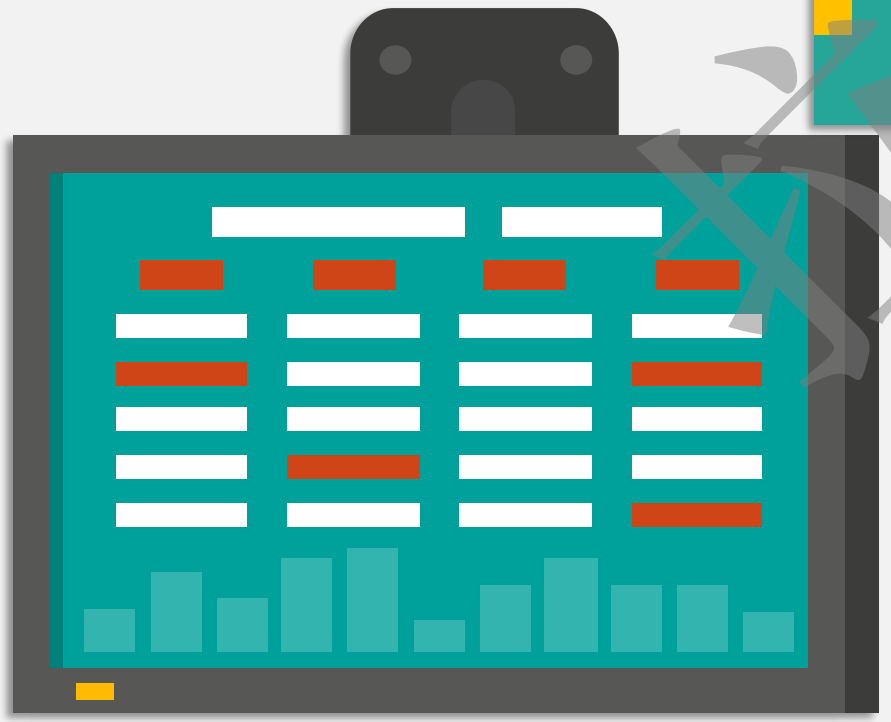
Ribbon内置负载均衡策略



自定义负载均衡算法

拉勾
LAGOU.COM

1. 实现Irule接口或继承AbstractLoadBalancerRule
2. 实现choose方法
3. 指定Ribbon的算法类



自定义负载均衡算法使用场景

拉勾
LAGOU.COM

使用场景

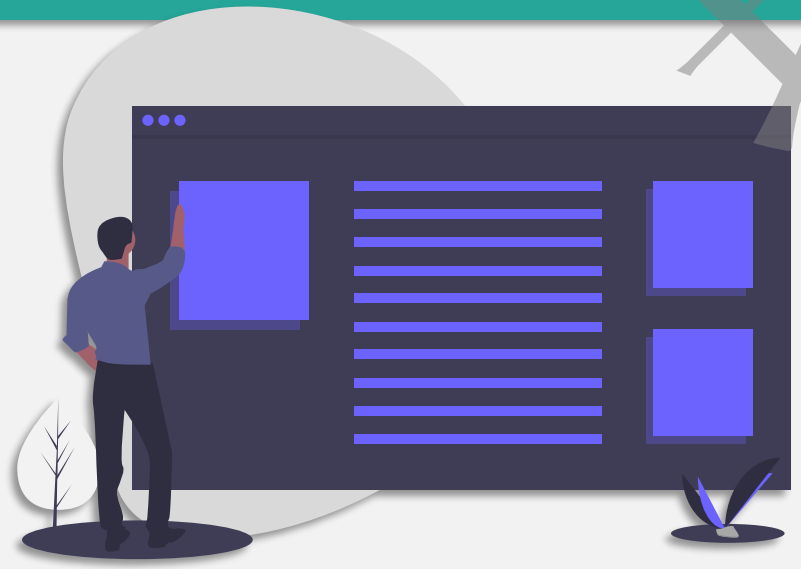
1. 定制跟业务更匹配的策略
2. 灰度发布
3. 多版本隔离
4. 故障隔离

Ribbon饥饿加载模式

Ribbon在进行客户端负载均衡的时候并不是在启动时就加载上下文

而是在第一次请求的时候才去创建，因此第一次调用会比较慢，有可能会引起调用超时

可以通过指定Ribbon客户端的名称，在启动时加载这些子应用程序上下文



```
ribbon.eager-load.enabled=true
ribbon.eager-load.clients=服务A,服务B
```

配置方式自定义Ribbon Client

<clientName>.ribbon.NFLoadBalancerClassName: Should implement ILoadBalancer(负载均衡器操作接口)

<clientName>.ribbon.NFLoadBalancerRuleClassName: Should implement IRule(负载均衡算法)

<clientName>.ribbon.NFLoadBalancerPingClassName: Should implement IPing(服务可用性检查)

<clientName>.ribbon.NIWSServerListClassName: Should implement ServerList（服务列表获取）

<clientName>.ribbon.NIWSServerListFilterClassName: Should implement ServerListFilter（服务列表的过滤）

Next：课时4《服务容错保护-Hystrix》