// 2 . / /1 1 12 1 3

- .0 (- .0)

 -

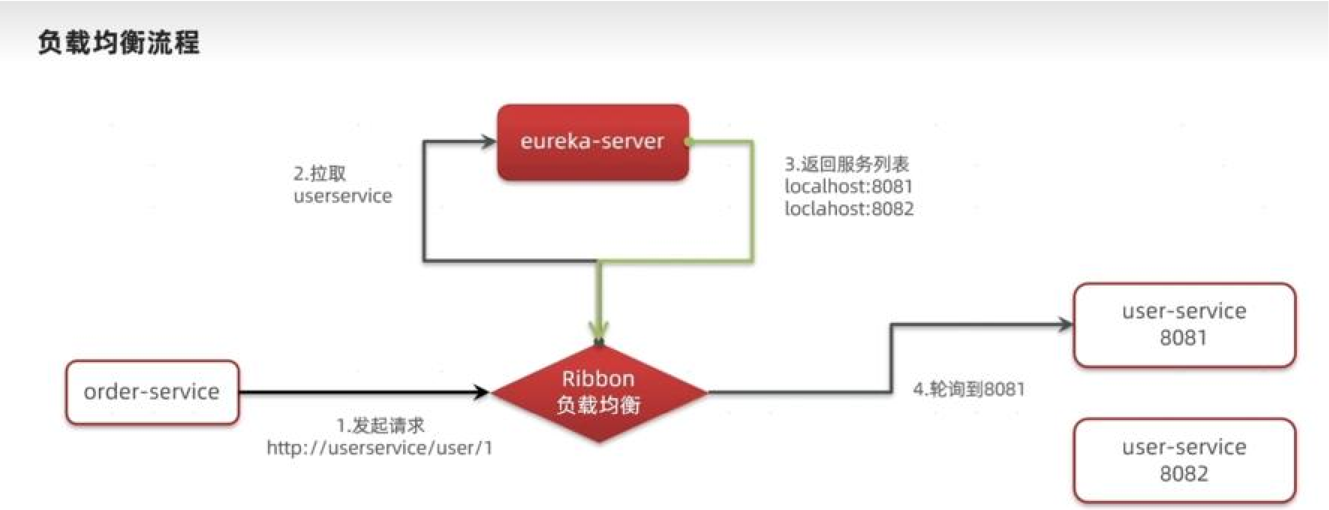
 -

Ribbon负载均衡

_

1.负载均衡原理

1.1原理图



1.2LoadBalancerIntercepor

```
public class LoadBalancerInterceptor implements ClientHttpRequestInterceptor {
    private LoadBalancerClient loadBalancer;
    private LoadBalancerRequestFactory requestFactory;

    public LoadBalancerInterceptor(LoadBalancerClient loadBalancer, LoadBalancerRequestFactory requestFactory) {
        this.loadBalancer = loadBalancer;
        this.requestFactory = requestFactory;
    }

    public LoadBalancerInterceptor(LoadBalancerClient loadBalancer) {
        this(loadBalancer, new LoadBalancerRequestFactory(loadBalancer));
    }

    public ClientHttpResponse intercept(final HttpRequest request, final byte[] body, final ClientHttpRequestExecution execution) throws IOException {
        URI originalUri = request.getURI();
        String serviceName = originalUri.getHost();
        Assert.state( expression: serviceName != null,  message: "Request URI does not contain a valid hostname: " + originalUri);
        return (ClientHttpResponse)this.loadBalancer.execute(serviceName, this.requestFactory.createRequest(request, body, execution));
    }
}
```

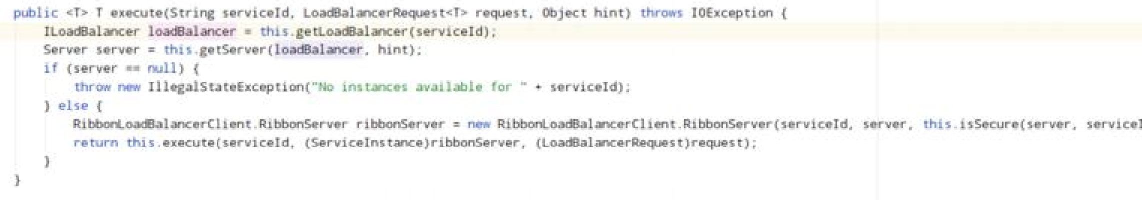
. () // - / /

. () -

. . ()

.

1.3LoadBalancerClient

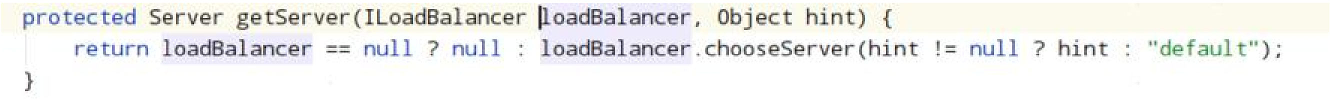


```java
public <T> T execute(String serviceId, LoadBalancerRequest<T> request, Object hint) throws IOException {
    ILoadBalancer loadBalancer = this.getLoadBalancer(serviceId);
    Server server = this.getServer(loadBalancer, hint);
    if (server == null) {
        throw new IllegalStateException("No instances available for " + serviceId);
    } else {
        RibbonLoadBalancerClient.RibbonServer ribbonServer = new RibbonLoadBalancerClient.RibbonServer(serviceId, server, this.isSecure(server, serviceI
        return this.execute(serviceId, (ServiceInstance)ribbonServer, (LoadBalancerRequest)request);
    }
}
```

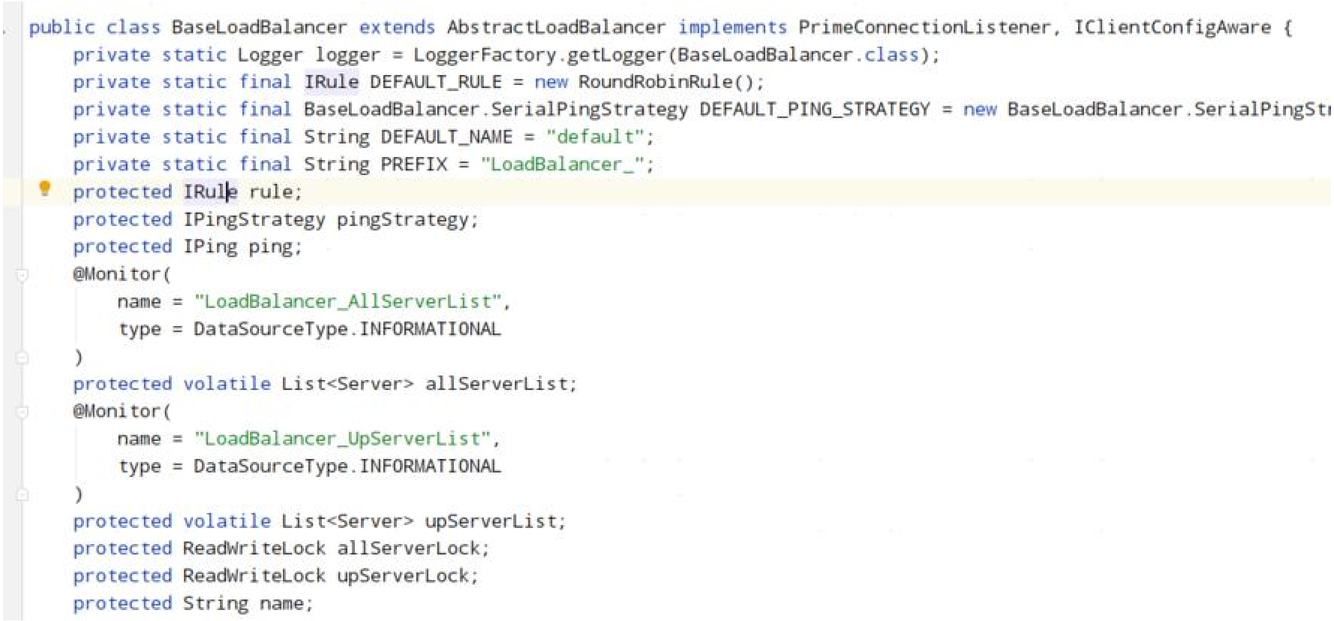
( )

( )

1.4负载均衡策略IRule



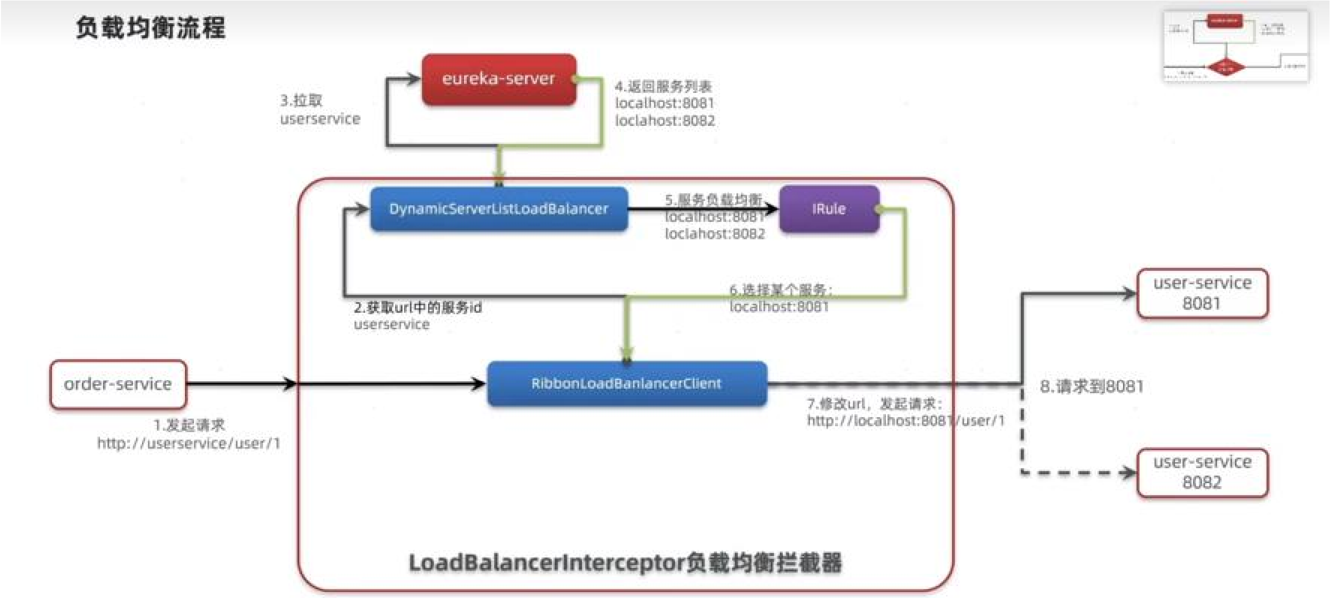
```java
protected Server getServer(ILoadBalancer loadBalancer, Object hint) {
    return loadBalancer == null ? null : loadBalancer.chooseServer(hint != null ? hint : "default");
}
```



```java
public class BaseLoadBalancer extends AbstractLoadBalancer implements PrimeConnectionListener, IClientConfigAware {
    private static Logger logger = LoggerFactory.getLogger(BaseLoadBalancer.class);
    private static final IRule DEFAULT_RULE = new RoundRobinRule();
    private static final BaseLoadBalancer.SerialPingStrategy DEFAULT_PING_STRATEGY = new BaseLoadBalancer.SerialPingStr
    private static final String DEFAULT_NAME = "default";
    private static final String PREFIX = "LoadBalancer_";
    protected IRule rule;
    protected IPingStrategy pingStrategy;
    protected IPing ping;
    @Monitor(
        name = "LoadBalancer_AllServerList",
        type = DataSourceType.INFORMATIONAL
    )
    protected volatile List<Server> allServerList;
    @Monitor(
        name = "LoadBalancer_UpServerList",
        type = DataSourceType.INFORMATIONAL
    )
    protected volatile List<Server> upServerList;
    protected ReadWriteLock allServerLock;
    protected ReadWriteLock upServerLock;
    protected String name;
```

```
/.../

package com.netflix.loadbalancer;

public interface IRule {
    Server choose(Object var1);

    void setLoadBalancer(ILoadBalancer var1);

    ILoadBalancer getLoadBalancer();
}
```

1.5总结



// / /1

-

-

0 1 0 2

0 1

0 1 //
0 1/ /1

ヽ(°▽°)ノ

2.负载均衡策略

2.1基本规则

负载均衡策略

内置负载均衡规则类	规则描述
RoundRobinRule	简单轮询服务列表来选择服务器。它是Ribbon默认的负载均衡规则。
AvailabilityFilteringRule	对以下两种服务器进行忽略： （1）在默认情况下，这台服务器如果3次连接失败，这台服务器就会被设置为“短路”状态。短路状态将持续30秒，如果再次连接失败，短路的持续时间就会几何级地增加。 （2）并发数过高的服务器。如果一个服务器的并发连接数过高，配置了AvailabilityFilteringRule规则的客户端也会将其忽略。并发连接数的上限，可以由客户端的<clientName>.<clientConfigNameSpace>.ActiveConnectionsLimit属性进行配置。
WeightedResponseTimeRule	为每一个服务器赋予一个权重值。服务器响应时间越长，这个服务器的权重就越小。这个规则会随机选择服务器，这个权重值会影响服务器的选择。
ZoneAvoidanceRule	以区域可用的服务器为基础进行服务器的选择。使用Zone对服务器进行分类，这个Zone可以理解为一个机房、一个机架等。而后再对Zone内的多个服务做轮询。
BestAvailableRule	忽略哪些短路的服务器，并选择并发数较低的服务器。
RandomRule	随机选择一个可用的服务器。
RetryRule	重试机制的选择逻辑

内置负载均衡规则类 **规则描述**

1 3

30 2

..

ZoneAvoidanceRule

2.2自定义负载均衡策略

1. -

()
()

2. - .

#

. . . #

注意

3.饥饿加载

-