Bipin Kumar

shael.dhn88@gmail.com

Ribbon

Ribbon Setup

Ribbon is used for client-side load balancing in this document we are going to learn how to do client-side load balancing using ribbon.

In this demo we will start one booking -cargo app, one eureka and two instance of routing app.

1. First add ribbon dependency to booking cargo app.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

</dependency>

1. In Step two open ExternalCargoRoutingService.java class and comment DiscoveryClient and its related code.

Here we are going to use below code.

@Autowired

**private** LoadBalancerClient loadBalancerClient;

ServiceInstance serviceInstance = loadBalancerClient.choose("routing-service");

URI uri = serviceInstance.getUri();

1. Now starts services eureka server and open swagger-ui for cargo-booking.

{

"bookingAmount": 100,

"originLocation": "BLR",

"destLocation" : "DEL",

"destArrivalDeadline" : "2020-01-28"

}

Open cargobooking API and use above json to post request. And copy the bookingId

Now open cargorouting API and use above generated bookingID to post request. Click execute method two three times. Now check logs. You can see everytime we make a call it is taking different url as shown in below logs.

URI= http://localhost:8082

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

2021-03-27 12:22:18.128 INFO 11444 --- [erListUpdater-0] c.netflix.config.ChainedDynamicProperty : Flipping property: routing-service.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLimit = 2147483647

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

URI= http://localhost:8082

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

URI= http://localhost:9091

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

URI= http://localhost:8082

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

URI= http://localhost:9091

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

URI= http://localhost:8082

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

URI= http://localhost:9091

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

URI= http://localhost:8082

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

CargoBookingCommandService.assignRouteToCargo()===========================

ExternalCargoRoutingService.fetchRouteForSpecification()======hitting cargorouting service=========

URI= http://localhost:9091

Publishing CargoRoutedEvent with booking id \*\*\*26B5F151-CDAC-4518-A5E1-02FC760B853B

1. Now in this case we have use low level of API call why should we write logic to choose uri. Instead of this we can use restemplate. It is making outbound calls. if we want to pre execute any code we can configure it to interceptor. In code we have autowired the resttemplate. Where we have defined resttemplate bead we need to annotated that with @LoadBalanced

@Bean

@LoadBalanced

**public** RestTemplate restTemplate() {

**return** **new** RestTemplate();

}

Now it automatically configured it. But now we have to follow a some rule to define URI.

**final** String REST\_URI

= "http://routing-service"+"/cargorouting/optimalRoute?origin={origin}&destination={destination}&deadline={deadline}";

1. Now we can restart service and check the logs of two routing-service instance.

Retry-logic:

1. Configure below properties in properties file.

routing-service:

ribbon:

ReadTimeout: 1000

ConnectTimeout: 1000

MaxAutoRetriesNextServer: 4

MaxAutoRetries: 2

Suppose we want to configure different property to other service we can define in same way if we want it to be generic then just remove service name.

**ReadTimeout**: 1000 : cannot try to retry continuously so after sometime it should stop retrying.

**ConnectTimeout**: 1000 : If connection not stabilize within this time.

**MaxAutoRetriesNextServer**: 4 : maximum time it try to connect servers other

**MaxAutoRetries**: 2 : max time it try to reconnect to one instance/ server

Now open RibbonAutoConfiguration class, Now open LoadBalancerAutoConfiguration class

There you can find two inner class which implement interceptor. which is used in case of resttemplate call. If you can not find retry you should add spring-retry jar dependency in pom.xml.

spring:

cloud:

loadbalancer:

retry:

enabled: **true**

<dependency>

<groupId>org.springframework.retry</groupId>

<artifactId>spring-retry</artifactId>

</dependency>

GoodReads :

<http://cloud.spring.io/spring-cloud-static/spring-cloud-netflix/2.1.0.RELEASE/single/spring-cloud-netflix.html#netflix-ribbon-starter>

https://groups.google.com/forum/#!topic/eureka\_netflix/B3uJ0onU\_Bo

http://techblog.netflix.com/2013/01/announcing-ribbon-tying-netflix-mid.html

http://stackoverflow.com/questions/29730310/why-client-side-load-balancers-like-ribbon

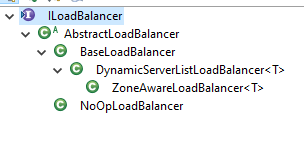
<https://github.com/Oreste-Luci/netflix-oss-example>+

some config:

clientname.ribbon.NFLoadBalancerRuleClassName=com.netflix.loadbalancer.WeightedResponseTimeRule

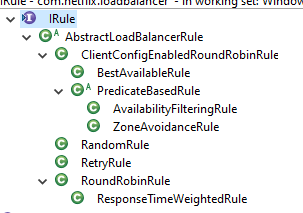
**Internals of Ribbon**:

There is interface of ILoadBalancer and their multiple implementation of it as shown in below screenshot.



This class don’t implement load balancing logic. It is used to fetch all servers or reachable server.

There is another interface call IRule Which implement logic for load balancing



Resttemplate -> interceptor -> Irule.choose() -> Iloadbalancer.getAllserver/reachableserver

<https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-ribbon.html>

