Client Side Load Balancer: Ribbon

Ribbon is a client side load balancer which gives you a lot of control over the behaviour of HTTP and TCP clients.

- ❑ Feign already uses Ribbon

A central concept in Ribbon is that of the named client.

Each load balancer is part of an ensemble of components that work together to contact a remote server on demand, and the ensemble has a name that we give it as an application developer (e.g. using the @FeignClient annotation).

Spring Cloud creates a new ensemble as an ApplicationContext on demand for each named client using RibbonClientConfiguration. This contains an ILoadBalancer, a RestClient, and a ServerListFilter.

Customizing the Ribbon Client

We can configure some bits of a Ribbon client using external properties in <client>.ribbon.*, which is no different than using the Netflix APIs natively, except that we can use Spring Boot configuration files.

Spring Cloud also lets you take full control of the client by declaring additional configuration (on top of the RibbonClientConfiguration) using @RibbonClient.

Example:

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

Package -> com.netflix.loadbalancer

Interface Irule

All Known Implementing Classes:
AbstractLoadBalancerRule,
AvailabilityFilteringRule,
ClientConfigEnabledRoundRobinRule,
RandomRule,
ResponseTimeWeightedRule,
RetryRule,
RoundRobinRule,
WeightedResponseTimeRule,
ZoneAvoidanceRule

public interface Irule
Interface that defines a "Rule" for a LoadBalancer. A Rule can be thought of as a Strategy for loadbalacing. Well known loadbalancing strategies include Round Robin, Response Time based etc.

RoundRobinRule

This rule simply choose servers by round robin. It is often used as the default rule or fallback of more advanced rules.

AvailabilityFilteringRule

This rule will skip servers that are deemed "circuit tripped" or with high concurrent connection count.

Package -> com.netflix.loadbalancer

Interface IPing

All Known Implementing Classes:
AbstractLoadBalancerPing, DummyPing, NoOpPing, PingConstant

public interface IPing
Interface that defines how we "ping" a server to check if its alive

AbstractLoadBalancerPing

Class that provides the basic implementation of determining the "liveness" or suitability of a Server

NoOpPing

This doesn’t actually ping server instances, instead always reporting that they’re stable

Using Ribbon with Eureka

When Eureka is used in conjunction with Ribbon the ribbonServerList is overridden with an extension of DiscoveryEnabledNIWSServerList which populates the list of servers from Eureka.

It also replaces the IPing interface with NIWSDiscoveryPing which delegates to Eureka to determine if a server is up

Example: How to Use Ribbon Without Eureka

We can declare a @RibbonClient for "stores", and Eureka is not in use

application.yml

```
stores:
  ribbon:
    listOfServers: example.com,google.com
```

Example: Disable Eureka use in Ribbon

Setting the property ribbon.eureka.enabled = false will explicitly disable the use of Eureka in Ribbon.

application.yml

```
ribbon:
  eureka:
   enabled: false
```

Declarative REST Client: Feign

- ❑ Feign is a declarative web service client.
- ❑ It makes writing web service clients easier.
- ❑ To use Feign create an interface and annotate it.
- ❑ It has pluggable annotation support including Feign annotations and JAX-RS annotations.
- ❑ Feign also supports pluggable encoders and decoders.
- ❑ Spring Cloud integrates Ribbon and Eureka to provide a load balanced http client when using Feign.

Example spring boot app

```
@Configuration
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
@EnableFeignClients
public class Application {

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

}
```

StoreClient.java

```
@FeignClient("stores")
public interface StoreClient {
   @RequestMapping(method = RequestMethod.GET, value = "/stores")
   List<Store> getStores();

   @RequestMapping(method = RequestMethod.POST, value =
"/stores/{storeId}", consumes = "application/json")
   Store update(@PathVariable("storeId") Long storeId, Store store);
}
```

Note : In the @FeignClient annotation the String value ("stores" above) is an arbitrary client name, which is used to create a Ribbon load balancer

Overriding Feign Defaults

Spring Cloud lets us take full control of the feign client by declaring additional configuration (on top of the FeignClientsConfiguration) using @FeignClient.

Example:

```
@FeignClient(name = "stores", configuration = FooConfiguration.class)
public interface StoreClient {
   //..
}
```

In this case the client is composed from the components already in FeignClientsConfiguration together with any in FooConfiguration

Spring Cloud Netflix provides the following beans by default for feign (BeanType beanName: ClassName):

Decoder feignDecoder: ResponseEntityDecoder (which wraps a SpringDecoder)
Encoder feignEncoder: SpringEncoder
Logger feignLogger: Slf4jLogger
Contract feignContract: SpringMvcContract
Feign.Builder feignBuilder: HystrixFeign.Builder