# Spring Cloud Feign：声明式服务调用

- 整合了 Spring Cloud Ribbon 与 Spring Cloud Hystrix, 除了提供这两者的强大功能之外， 它还提供了一 种声明式的 Web 服务客户端定义方式
- 只需创建一个接口并用注解的方式来配置它，即可完成对服务提供方的接口绑定，简化了在使用Spring CloudRibbon时自行封装服务调用客户端的开发量

## 入门

- 通过 Spring Cloud Feign 提供的声明式服务绑定功能来实现对该服务接口的调用
- web、eureka、feign
- 主类：@EnableFeignClients 注解开启 Spring Cloud Feign 的支待功能、@EnableDiscoveryClien
- 定义 HelloService 接口， 通过@FeignClient 注解指定服务名来绑定服务， 然后再使用 Spring MVC 的注解来绑定具体该服务提供的 REST 接口

```
// 指定服务名，不区分大小写
@FeignClient("hello-service")
public interface HelloService {
  // Spring MVC 的注解来绑定具体该服务提供的 REST 接口
  @RequestMapping("/hello")
  String hello();

  //参数绑定
  @RequestMapping(value= "/hellol", method= RequestMethod.GET)
  String hello(@RequestParam("name") String name);

  @RequestMapping(value = "/hello3", method= Reques七Method.POST)
  String hello(@RequestBody User user);
}
```

- 接着， 创建一个 ConsumerController 来实现对 Feign 客户端的调用。 使用@Autowired 直接注入上面定义的 HelloService 实例， 并在 helloConsumer函数中调用这个绑定了 hello-service 服务接口的客户端来向该 服 务发起/hello 接口的调用

```
@RestController
public class ConsumerController {
  @Autowired
  HelloService helloService;

  @RequestMapping(value = "/feign-consumer", method = RequestMethod.GET)
  public String helloConsumer() {
    return helloService.hello();
  }
```

```
  //参数绑定 GET 请求到 http://localhost:9001/feign-consumer2,
  @RequestMapping(value = "/feign-consumer2", method = RequestMethod.GET}
  public String helloConsumer2() {
    StringBuilder sb = new StringBuilder();
    sb.append(helloService.hello().append("\n"};
    sb.append(helloService.hello("DIDI")).append("\n"};
    sb.append(helloService.hello(new User("DIDI", 30))).append("\n");
    return sb.toString();
}
```

- application.properties指定服务名称（feign-consumer）、端口（9001）、注册中心
- 访问`GET 请求到 http://localhost:9001/feign-consumer`
- 参数绑定

```
  //hello-service服务中，添加
  //参数绑定
  @RequestMapping(value= "/hellol", method= RequestMethod.GET)
  public String hello(@RequestParam String name) {
    return "Hello " + name;
  }
  @RequestMapping(value= "/hello3", method = RequestMethod.POST)
  public String hello(@RequestBody User user) {
    return "Hello "+ user.getNarne() + ", " + user.getAge(); // User要有默认
构造函数
  }
```

# 继承特性

- 通过Spring Cloud Feign 的继承特性来实现 REST 接口定义的复用
- hello-service-api，web依赖、User.java
- 新增HelloService接口

```
@RequestMapping("/refactor")
public interface HelloService {
  @RequestMapping(value = "/hello4", method= RequestMethod.GET)
  String hello(@RequestParam("name") String name) ;

  @RequestMapping(value= "/hello6", method= RequestMethod.POST)
  String hello(@RequestBody User user);
}
```

- 将hello-service-api打成依赖jar包（不是可执行jar包）

- 服务提供者的重构，hello-service 中引入依赖， 并创建 RefactorHelloController 类（@RestController）继承 hello-service-api 中定义的HelloService 接口，不需要RequestMapping标注，方法参数上的注解要带上，并实现所有方法

```
@RestController
public class RefactorHelloController implements HelloService {
  @Override
  public String hello(@RequestParam("name") String name) {
    return "Hello"+ name;
  }

  @Override
  public String hello(@RequestBody User user) {
    return "Hello "+ user.getName() + ", " + user.getAge();
  }
}
```

- 服务消费者重构，feign-consumer中引入依赖，并创建 RefactorHelloService 接口， 并继承 hello-service-api 包中的HelloService 接口， 然后添加@FeignClient注解来绑定服务。

```
//服务名
@FeignClient(value = "HELLO-SERVICE")
public interface RefactorHelloService extends
com.didispace.service.HelloService {
}
```

- 在 ConsumeController 中， 注入 RefactorHelloService 的实例，并新增一个请求/feign-consumer3 来触发对 RefactorHelloService 的实例的调用

```
@Autowired
RefactorHelloService refactorHelloService;

@RequestMapping(value = "/feign-consumer3", method = RequestMethod.GET}
  public String helloConsumer3() {
    StringBuilder sb = new StringBuilder();
    sb.append(refactorHelloService.hello("MIMI")).append("\n");
    sb.append(refactorHelloService.hello(new
com.didispace.dto.User("MIMI", 20)}}.append("\n");
    return sb.toString();
}
```

- 访问`http://localhost:9001/feign-consumer3,`

# Ribbon配置

- 由于SpringCloudFeign的客户端负载均衡是通过SpringCloudRibbon实现的，所以可以直接通过配置Ribbon客户端的方式来自定义各个服务客户端调用的参数

- 全局配置
  - ribbon.=
- 指定服务配置
  - . ribbon.key=value
  - 在定义Feign客户端的时候， 使用了@FeignClient注解。 在初始化过程中，SpringCloudFeign会根据该注解的name属性或value属性指定的服务名， 自动创建一个同名的Ribbon客户端。
  - 可以使用@FeignClient注解中的name或value属性值来设置对应的Ribbon参数 `HELLO-SERVICE.ribbon.ConnectTimeout=500`
- 重试机制
  - 在 Spring Cloud Feign 中默认实现了 请求的重试机制
  - Spring Cloud Ribbon重试机制
  - 需要让Hystrix的超时时间大于Ribbon的超时时间， 否则Hystrix命令超时后， 该命令直接熔断， 重试机制就 没有任何意义了

```
# 更换实例访问的次数，尝试更换两次实 例进行重试
HELLO-SERVICE.ribbon.MaxAutoRetriesNextServer=2
# 重试策略先尝试访问首选实 例一次， 失败后才更换实例访问
HELLO-SERVICE.ribbon.MaxAutoRetries= 1
```

# Hystrix配置

- 默认情况下，Spring CloudFeign会为将所有Feign客户端的方法都封装到Hystrix命令中进行服务保护
- 全局配置
  - 对于Hystrix的全局配置同Spring CloudRibbon的全局配置一样，直接使用它的默认配置前缀 hystrix.command.default 就可以进行设置

    `hystrix.command.default.execution.isolation.thread.timeoutinMilliseconds=5000`
  - 对Hystrix进行配置之前，需要确认 feign.hystrix.enabled参数没有被设置为false, 否则该参数设置会关闭Feign客户端的Hystrix支持
- 禁用Hystrix
  - 只想针对某个服务客户端关闭Hystrix支待时， 需要通过使用@Scope("prototype")注解为指定的客户端配置Feign.Builder实例
    - 构建一 个关闭Hystrix的配置类

```
@Confguration
public class DisableHystrixConfiguration {
        @Bean
        @Scope("prototype")
        public Feign.Builder feignBuilder() {
                return Feign.builder();
  }
}
```

- 在HelloService的@FeignClient注解中，通过configuration参数引入上面实现的配置

```
@FeignClient(name="HELLO - SERVICE", configuration =
DisableHystrixConfiguration.class)
  public interface HelloService {...}
```

- 指定命令配置
  - 采用hystrix.command. 作为前缀。 而默认情况下会采用Feign客户端中的方法名作为标识

    `hystrix.command.hello.execution.isolation.thread.timeoutinMilliseconds=5000`
- 服务降级配置
  - 只需要为 Feign 客户端的定义接口编写一个具体的接口实现类，@Component
  - 在服务绑定接口 HelloService 中， 通过@FeignClient 注解的 fallback 属性来指定对应的服务降级实现类

```
@FeignClient{narne="HELLO-SERVICE", fallback=
HelloServiceFallback.class)
 public interface HelloService {
  ...
}
```

- 请求压缩
  - Spring Cloud Feign支持对请求与响应进行 GZIP 压缩，以减少通信过程中的性能损耗

```
# 开启请求与响应的压缩功能
feign.compression.request.enabled=true
feign.compression.response.enabled=true
```

- 日志配置
  - Spring Cloud Feign 在构建被@FeignClient 注解修饰的服务客户端时，会为每一个客户端都创建一个 feign.Logger 实例，我们可以利用该日志对象的 DEBUG 模式来帮助分析 Feign 的请求细节
  - 可以在application.properties 文件中使用logging.level. 的参数配置格式来开启指定 Feign客户端的DEBUG日志，其中<FeignC巨ent>为 Feign 客户端定义接口的完整路径

`logging.level.com.didispace.web.HelloService= DEBUG`

- 只是添加了如上配置， 还无法实现对 DEBUG 日志的输出。 这时由于 Feign 客户端默认的 Logger.Level 对象定义为 NONE 级别， 该级别不会记录任何 Feign 调用过程中的信息， 所以我们需要调整它的级别， 针对全局的日志级别， 可以在应用主类中直接加入 Logger.Level 的 Bean 创建

```
@Bean
Logger.Level feignLoggerLevel() { return Logger.Level.FULL;}
```

- 也可以通过实现配置类， 然后在具体的 Feign 客户端来指定配置类以实现是否要调整不同的日志级别

```
@Configuration
public class FullLogConfiguration {
  @Bean
        Logger.Level feignLoggerLevel() { return Logger.Level.FULL;}
}

@FeignClient(name= "HELLO-SERVICE", configuration =
FullLogConfiguration.class) public interface HelloService {
        ...
}
```

- 访问`http://localhost:9001/feign-consumer`
- Feign 的 Logger级别
  - NONE: 不记录任何信息
  - BASIC: 仅记录请求方法、URL以及响应状态码和执行时间
  - HEADERS: 除了记录BASIC级别的信息之外， 还会记录请求和响应的头信息
  - FULL: 记录所有请求与响应的明细， 包括头信息、 请求体、 元数据等