9.1 OpenFeign

前面无论是基本调用，还是 Hystrix，我们实际上都是通过手动调用 RestTemplate 来实现远程调用的。使用 RestTemplate 存在一个问题：繁琐，每一个请求，参数不同，请求地址不同，返回数据类型不同，其他都是一样的，所以我们希望能够对请求进行简化。

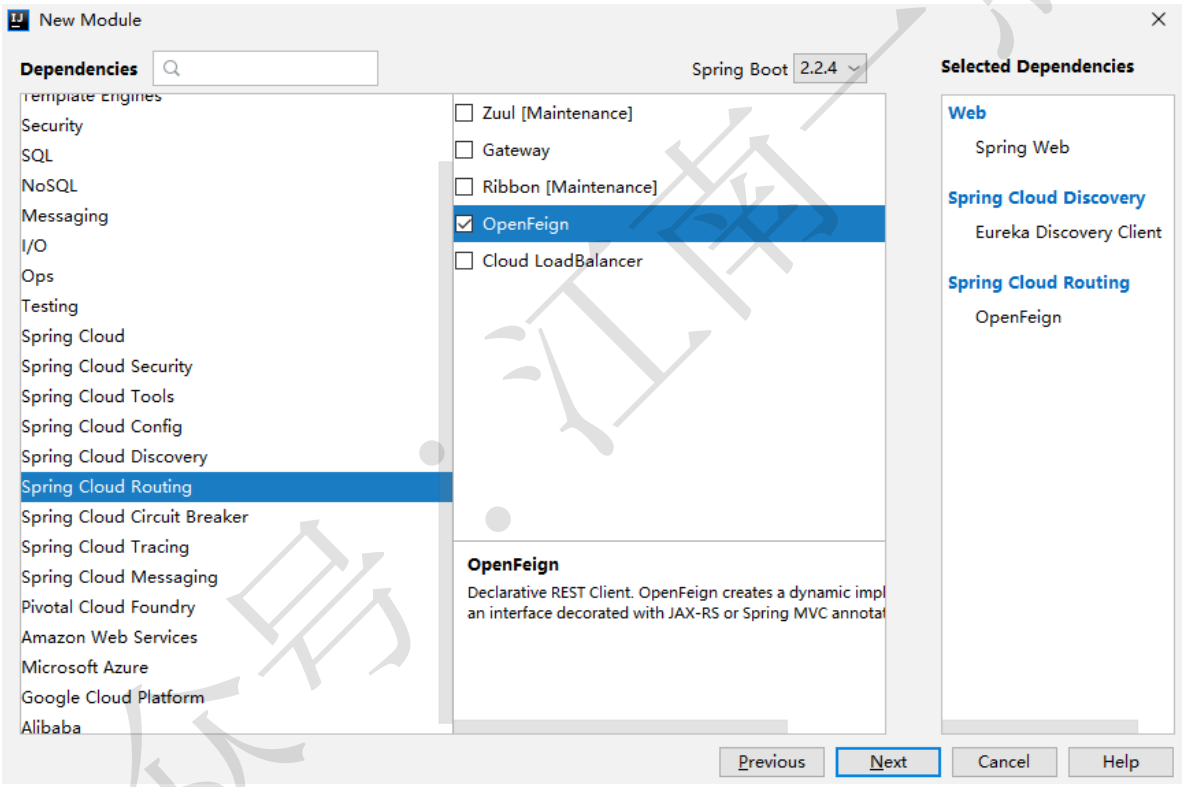
我们希望对请求进行简化，简化方案就是 OpenFeign。

一开始这个组件不叫这个名字，一开始就叫 Feign，Netflix Feign，但是 Netflix 中的组件现在已经停止开源工作，OpenFeign 是 Spring Cloud 团队在 Netflix Feign 的基础上开发出来的声明式服务调用组件。关于 OpenFeign 组件的 Issue：https://github.com/OpenFeign/feign/issues/373

9.1.1 HelloWorld

继续使用之前的 Provider。

新建一个 Spring Boot 模块，创建时，选择 OpenFeign 依赖，如下：



项目创建成功后，在 application.properties 中进行配置，使项目注册到 Eureka 上：

```
spring.application.name=openfeign
server.port=4000
eureka.client.service-url.defaultZone=http://localhost:1111/eureka
```

接下来在启动类上添加注解，开启 Feign 的支持：

```
@SpringBootApplication
@EnableFeignClients
public class OpenfeignApplication {

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

}
```

接下来，定义 HelloService 接口，去使用 OpenFeign：

```
@FeignClient("provider")
public interface HelloService {

    @GetMapping("/hello")
    String hello();//这里的方法名无所谓，随意取
}
```

最后调用 HelloController 中，调用 HelloService 进行测试：

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

    @GetMapping("/hello")
    public String hello() {
        return helloService.hello();
    }
}
```

接下来，启动 OpenFeign 项目，进行测试。

9.2 参数传递

和普通参数传递的区别：

1. 参数一定要绑定参数名。
2. 如果通过 header 来传递参数，一定记得中文要转码。

测试的服务端接口，继续使用 provider 提供的接口。

这里，我们主要在 openfeign 中添加调用接口即可：

```
@FeignClient("provider")
public interface HelloService {

    @GetMapping("/hello")
    String hello();//这里的方法名无所谓，随意取

    @GetMapping("/hello2")
    String hello2(@RequestParam("name") String name);

    @PostMapping("/user2")
    User addUser(@RequestBody User user);
```

```
    @DeleteMapping("/user2/{id}")
    void deleteUserById(@PathVariable("id") Integer id);

    @GetMapping("/user3")
    void getUserByName(@RequestHeader("name") String name);
}
```

注意，凡是 key/value 形式的参数，一定要标记参数的名称。

HelloController 中调用 HelloService：

```
@GetMapping("/hello")
public String hello() throws UnsupportedEncodingException {
    String s = helloService.hello2("江南一点雨");
    System.out.println(s);
    User user = new User();
    user.setId(1);
    user.setUsername("javaboy");
    user.setPassword("123");
    User u = helloService.addUser(user);
    System.out.println(u);
    helloService.deleteUserById(1);
    helloService.getUserByName(URLEncoder.encode("江南一点雨", "UTF-8"));
    return helloService.hello();
}
```

注意：

放在 header 中的中文参数，一定要编码之后传递。

9.3 继承特性

将 provider 和 openfeign 中公共的部分提取出来，一起使用。

我们新建一个 Module，叫做 hello-api，注意，由于这个模块要被其他模块所依赖，所以这个模块是一个 Maven 项目，但是由于这个模块要用到 SpringMVC 的东西，因此在创建成功后，给这个模块添加一个 web 依赖，导入 SpringMVC 需要的一套东西。

项目创建成功后，首先添加依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.2.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.javaboy</groupId>
    <artifactId>commons</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

然后定义公共接口，就是provider 和 openfeign 中公共的部分：

```
public interface IUserService {
    @GetMapping("/hello")
    String hello();//这里的方法名无所谓，随意取
```

```
    @GetMapping("/hello2")
    String hello2(@RequestParam("name") String name);

    @PostMapping("/user2")
    User addUser(@RequestBody User user);

    @DeleteMapping("/user2/{id}")
    void deleteUserById(@PathVariable("id") Integer id);

    @GetMapping("/user3")
    void getUserByName(@RequestHeader("name") String name);
}
```

定义完成后，接下来，在 provider 和 openfeign 中，分别引用该模块：

```
<dependency>
    <groupId>org.javaboy</groupId>
    <artifactId>hello-api</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

添加成功之后，在 provider 中实现该接口：

```
@RestController
public class HelloController implements IUserService {
    @Value("${server.port}")
    Integer port;

    @Override
    public String hello() {
        return "hello javaboy:" + port;
    }

    @Override
    public String hello2(String name) {
        System.out.println(new Date() + ">>>" + name);
        return "hello " + name;
    }

    @PostMapping("/user1")
    public User addUser1(User user) {
        return user;
    }

    @Override
    public User addUser2(@RequestBody User user) {
        return user;
    }

    @PutMapping("/user1")
    public void updateUser1(User user) {
        System.out.println(user);
    }

    @PutMapping("/user2")
```

```
    public void updateUser2(@RequestBody User user) {
        System.out.println(user);
    }

    @DeleteMapping("/user1")
    public void deleteUser1(Integer id) {
        System.out.println(id);
    }

    @Override
    public void deleteUser2(@PathVariable Integer id) {
        System.out.println(id);
    }

    @Override
    public void getUserByName(@RequestHeader String name) throws
UnsupportedEncodingException {
        System.out.println(URLDecoder.decode(name, "UTF-8"));
    }
}
```

在 openfeign 中，定义接口继承自公共接口：

```
@FeignClient("provider")
public interface HelloService extends IUserService {
}
```

接下来，测试代码不变。

关于继承特性：

1. 使用继承特性，代码简洁明了不易出错。服务端和消费端的代码统一，一改俱改，不易出错。这是优点也是缺点，这样会提高服务端和消费端的耦合度。
2. 9.2 中所讲的参数传递，在使用了继承之后，依然不变，参数该怎么传还是怎么传。

9.4 日志

OpenFeign 中，我们可以通过配置日志，来查看整个请求的调用过程。日志级别一共分为四种：

1. NONE：不开启日志，默认就是这个
2. BASIC：记录请求方法、URL、响应状态码、执行时间
3. HEADERS：在 BASIC 的基础上，加载请求/响应头
4. FULL：在 HEADERS 基础上，再增加 body 以及请求元数据。

四种级别，可以通过 Bean 来配置：

```
@SpringBootApplication
@EnableFeignClients
public class OpenfeignApplication {

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

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

最后，在 application.properties 中开启日志级别：

```
logging.level.org.javaboy.openfeign.HelloService=debug
```

重启 OpenFeign，进行测试。

9.5 数据压缩

```
# 开启请求的数据压缩
feign.compression.request.enabled=true
# 开启响应的数据压缩
feign.compression.response.enabled=true
# 压缩的数据类型
feign.compression.request.mime-types=text/html,application/json
# 压缩的数据下限，2048 表示当要传输的数据大于 2048 时，才会进行数据压缩
feign.compression.request.min-request-size=2048
```

9.6 +Hystrix

Hystrix 中的容错、服务降级等功能，在 OpenFeign 中一样要使用。

首先定义服务降级的方法：

```
@Component
@RequestMapping("/javaboy")//防止请求地址重复
public class HelloServiceFallback implements HelloService {
    @Override
    public String hello() {
        return "error";
    }

    @Override
    public String hello2(String name) {
        return "error2";
    }

    @Override
    public User addUser2(User user) {
        return null;
    }
```

```
    @Override
    public void deleteUser2(Integer id) {

    }

    @Override
    public void getUserByName(String name) throws UnsupportedEncodingException {

    }
}
```

然后，在 HelloService 中配置这个服务降级类：

```
@FeignClient(value = "provider",fallback = HelloServiceFallback.class)
public interface HelloService extends IUserService {
}
```

最后，在 application.properties 中开启 Hystrix。

```
feign.hystrix.enabled=true
```

也可以通过自定义 FallbackFactory 来实现请求降级：

```
@Component
public class HelloServiceFallbackFactory implements
FallbackFactory<HelloService> {
    @Override
    public HelloService create(Throwable throwable) {
        return new HelloService() {
            @Override
            public String hello() {
                return "error---";
            }

            @Override
            public String hello2(String name) {
                return "error2---";
            }

            @Override
            public User addUser2(User user) {
                return null;
            }

            @Override
            public void deleteUser2(Integer id) {

            }

            @Override
            public void getUserByName(String name) throws
UnsupportedEncodingException {

            }
        };
    }
```

```
}
```

HelloService 中进行配置：

```
@FeignClient(value = "provider",fallbackFactory =
HelloServiceFallbackFactory.class)
public interface HelloService extends IUserService {
}
```