Feign 容错机制

- application.yml 添加配置。

```yaml
server:
  port: 8050
spring:
  application:
    name: feign
eureka:
  client:
    service-url:
      defaulZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
feign:
  hystrix:
    enabled: true
```

`feign.hystrix.enabled`：是否开启熔断器。

- 创建 FeignProviderClient 接口的实现类 FeignError，定义容错处理逻辑，通过 `@Component` 注解将 FeignError 实例注入 IoC 容器。

```java
package com.southwind.feign;

import com.southwind.entity.Student;

import java.util.Collection;

@Component
public class FeignError implements FeignProviderClient {
    @Override
    public String index() {
        return "服务器维护中.....";
    }

    @Override
    public Collection<Student> findAll() {
        return null;
    }

    @Override
    public Student findById(long id) {
        return null;
    }
```

```
    @Override
    public void save(Student student) {

    }

    @Override
    public void update(Student student) {

    }

    @Override
    public void deleteById(long id) {

    }
}
```

- 在 FeignProviderClient 类定义处通过 `@FeignClient` 的 fallback 属性设置映射。

```
package com.southwind.feign;

import com.southwind.entity.Student;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@FeignClient(value = "provider",fallback = FeignError.class)
public interface FeignProviderClient {

    @GetMapping("/student/index")
    public String index();

    @GetMapping("/student/findAll")
    public Collection<Student> findAll();

    @GetMapping("/student/findById/{id}")
    public Student findById(@PathVariable("id") long id);

    @PostMapping("/student/save")
    public void save(@RequestBody Student student);

    @PutMapping("/student/update")
    public void update(@RequestBody Student student);

    @DeleteMapping("/student/deleteById/{id}")
    public void deleteById(@PathVariable("id") long id);
}
```

Hystrix 容错机制

- 什么是 Hystrix？

Hystrix 是 Netflix 的一个开源项目，主要作用是通过控制访问远程系统、服务和第三方库的节点，从而对延迟和故障提供更强大的容错能力，可以看作是 Netflix 团队对分布式系统运维的各种理念和实践的总结。

Hystrix 设计原则

- 资源隔离机制。
- 限流机制。
- 熔断机制。
- 降级机制。
- 缓存机制。

如何使用?

- 创建 Maven 工程，pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-openfeign</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-hystrix-
dashboard</artifactId>
    </dependency>
</dependencies>
```

- application.yml 添加 Hystrix 相关配置。

```yaml
server:
  port: 8060
spring:
  application:
    name: hystrix
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
feign:
  hystrix:
    enabled: true
management:
  endpoints:
    web:
      exposure:
        include: 'hystrix.stream'
```

`management.endpoints.web.exposure.include`：用来暴露 endpoints 的相关信息。

- 创建启动类 HystrixApplication。

```java
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import
org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
@EnableCircuitBreaker
@EnableHystrixDashboard
public class HystrixApplication {
    public static void main(String[] args) {
        SpringApplication.run(HystrixApplication.class,args);
    }
}
```

> 注解说明

`@SpringBootApplication`：声明该类是 Sprint Boot 服务的入口。

`@EnableFeignClients`：声明启用 Feign。

`@EnableCircuitBreaker`：声明启动数据监控。

`@EnableHystrixDashboard`：声明启用可视化数据监控。

- HystrixHandler

```
package com.southwind.controller;

import com.southwind.entity.Student;
import com.southwind.feign.FeignProviderClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@RestController
@RequestMapping("/hystrix")
public class HystrixHandler {
    @Autowired
    private FeignProviderClient feignProviderClient;

    @GetMapping("/findAll")
    public Collection<Student> findAll(){
        return feignProviderClient.findAll();
    }

    @GetMapping("/findById/{id}")
    public Student findById(@PathVariable("id") long id){
        return feignProviderClient.findById(id);
    }

    @PostMapping("/save")
    public void save(@RequestBody Student student){
        feignProviderClient.save(student);
    }

    @PutMapping("/update")
    public void update(@RequestBody Student student){
        feignProviderClient.update(student);
    }

    @DeleteMapping("/deleteById/{id}")
    public void deleteById(@PathVariable("id") long id){
        feignProviderClient.deleteById(id);
    }

    @GetMapping("/index")
```

```
    public String index(){
        return feignProviderClient.index();
    }
}
```

直接访问http://localhost:8060/actuator/hystrix.stream 即可看到监控的数据，同时 Hystrix 还提供了可视化的数据监控，直接访问 http://localhost:8060/hystrix。

Spring Cloud Config

配置中心可以对所有微服务的配置文件进行统一管理，便于部署和维护，配置文件可以集中存放在本地，也可以存放在 Git 仓库。

> 本地文件系统

- 新建 Maven，pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
</dependencies>
```

- application.yml 添加 Config Server 相关配置。

```
server:
  port: 8762
spring:
  application:
    name: nativeconfigserver
  profiles:
    active: native
  cloud:
    config:
      server:
        native:
          search-locations: classpath:/shared
```

> 属性说明

`profiles.active`：配置文件的获取方式。

`cloud.config.server.native.search-locations`：本地配置文件的存放路径。

- 在 shared 路径下创建 configclient-dev.yml

```yaml
server:
  port: 8070
foo: foo version 1
```

- 创建启动类

```java
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class NativeConfigServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(NativeConfigServerApplication.class,args);
    }
}
```

`EnableConfigServer`：声明配置中心。

本地配置中心已经创建完成，接下来创建客户端来读取本地配置中的配置文件。

- 创建 Maven 工程，pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
    </dependency>
</dependencies>
```

- bootstrap.yml

```yaml
spring:
  application:
    name: configclient
  profiles:
    active: dev
  cloud:
    config:
      uri: http://localhost:8762
      fail-fast: true
```

`profiles.active`：配置文件名，这里需要与当前微服务在 Eureka Server 注册的名称结合起来使用，两个值用 - 连接，比如当前微服务的名称是 configclient，`profiles.active` 的值是 dev，那么就会在本地 Config Server 中查找名为 configclient-dev 的配置文件。

`cloud.config.uri`：本地 Config Server 的访问路径。

`cloud.config.fail-faset`：设置客户端优先判断 config server 获取是否正常，并快速响应失败内容。

- 创建启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class NativeConfigClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(NativeConfigClientApplication.class,args);
    }
}
```

本地配置中心搭建成功。

远程 Git 仓库

- 将配置文件上传到 GitHub 仓库，创建 configclient.yml

```
server:
  port: 8070
spring:
  application:
    name: configclient
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

- 将 configclient.yml 上传至 GitHub。

创建 Config Server

- 创建 Maven 工程，pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
</dependencies>
```

- 创建 application.yml，添加 Config Server 相关配置。

```yaml
server:
  port: 8888
spring:
  application:
    name: configserver
  cloud:
    config:
      server:
        git:
          uri: # Git仓库地址
          search-paths: # Git仓库中存放配置文件的路径
          username: # Git用户名
          passphrase: # Git密码
      label: master
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

- 创建启动类

```java
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class,args);
    }
}
```

创建 Config Client

- 创建 Maven 工程，pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
    </dependency>
</dependencies>
```

- bootstrap.yml

```
spring:
  cloud:
    config:
      name: configclient
      label: master
      discovery:
        enabled: true
        service-id: configserver
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

- 创建启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ConfigClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigClientApplication.class,args);
    }
}
```

- HelloHandler

```
package com.southwind.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
```

```java
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RequestMapping("/hello")
@RestController
public class HelloHandler {

    @Value("${server.port}")
    private String port;

    @GetMapping("/index")
    public String index(){
        return this.port;
    }
}
```