Spring-Cloud programmer-DD

Spring Cloud Eureka

Spring Cloud Eureka是Spring Cloud Netflix项目下的服务治理模块

Eureka Server的自我保护模式

解决如下警告：

> EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

Spring Cloud Ribbon

负载均衡工具

Spring Cloud Feign

服务调用客户端

Feign文件传输

- 需要添加插件支持

```
<dependency>
   <groupId>io.github.openfeign.form</groupId>
   <artifactId>feign-form</artifactId>
   <version>3.0.3</version>
</dependency>
<dependency>
   <groupId>io.github.openfeign.form</groupId>
   <artifactId>feign-form-spring</artifactId>
   <version>3.0.3</version>
</dependency>
<dependency>
   <groupId>commons-fileupload</groupId>
   <artifactId>commons-fileupload</artifactId>
```

```
    <version>1.3.3</version>
</dependency>
```

- 并且进行代码配置

```
@Configuration
    class MultipartSupportConfig {
        @Bean
        public Encoder feignFormEncoder() {
            return new SpringFormEncoder();
        }
    }
```

- 最后调用文件上传服务

```
@PostMapping(value = "/uploadFile", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
String handleFileUpload(@RequestPart(value = "file") MultipartFile
file);
```

使用中问题及解决

1. 明明是get方法，却总是报错“Request method 'POST' not supported”。

```
@RequestMapping(value="/user/name", method=RequestMethod.GET)
User findByUsername(final String userName, @RequestParam("address")
final String address);
```

> 原因是当userName没有被@RequestParam注解修饰时，会自动被当做request body来处理。只要有body，就会被feign认为是post请求，所以整个服务是被当作带有request parameter和body的post请求发送出去的

Feign的扩展

> 由于Feign是基于Ribbon实现的，所以它自带了客户端负载均衡功能，也可以通过Ribbon的IRule进行策略扩展。另外，Feign还整合的Hystrix来实现服务的容错保护，在Dalston版本中，Feign的Hystrix默认是关闭的。

spring-cloud-config

可以研究一下git的权限管理，考虑spring-cloud-config是否能够实现apollo那样细致的权限管理

加密/解密

- 需要注意：`不限长度的JCE版本`

有效访问地址

- `/encrypt/status`：查看加密功能状态的端点
- `/key`：查看密钥的端点
- `/encrypt`：对请求的body内容进行加密的端点，POST请求
- `/decrypt`：对请求的body内容进行解密的端点，POST请求

对称加密

- 步骤（以`spring-cloud-config`为基础）
  1. 安装不限长度的JCE版本
  2. 在项目配置文件中添加配置：`encrypt.key=password`
     - 也可以通过环境变量设置此配置：`ENCRYPT_KEY`

非对称加密

- 步骤（以`spring-cloud-config`为基础）

  1. `keytool`生成密钥对文件`config-server.keystore`

  2. 配置boot项目

     ```
     encrypt.key-store.location=file://${user.home}/config-
     server.keystore
     encrypt.key-store.alias=config-server
     encrypt.key-store.password=111111
     encrypt.key-store.secret=222222
     ```

     不过这些配置也可以通过环境变量来设置：

     ```
     ENCRYPT_KEY_STORE_LOCATION
     ENCRYPT_KEY_STORE_ALIAS
     ```

```
ENCRYPT_KEY_STORE_PASSWORD
ENCRYPT_KEY_STORE_SECRET
```

加密/解密使用

- 无论是对称还是非对称加密，都是在加密后的字符串前加上`{cipher}`标识，`spring-cloud-config`会自动解密

```
spring.datasource.password=
{cipher}3b6e65af8c10d2766dba099a590496a18cfd816ef9190c983bb56249595a
e3f0
spring.datasource.password=
{cipher}AQCActlsAycDFYRsGHzZ8Jw2S6GO9oeqJSCcm//HenrqiuO7zSo3/vg9BeXL
8xwiyIXtKcp2JN8hnrM4NTyyJDIjxhcCbJMjuGrrFJ2FdO5oJWmksymkP5EOXE6MjgxV
qHh/tc+06TMBQj2xqEcfCO3jBDPxcR88Ci+VXe63xDIVgvAV9IYmCxlfXOCH31bBlK7j
5FXJ8pPLUKgXwaDGzaA5QfqMCGduOfC0AQ+iA0QEW7SdDnwChLNwCHEBfQceWAE7qt6z
asiRFZeZt+waOp8rI1u+4CYcTjnV1iSdXwN5j1lhcsoiIpViNx8kbsxhcmpCzdg3bGrS
1e/Pzq8CjHmV7IRRS9BfgR6K7wuyjue4SO2ZUtMbZAE5V2NHb3XsqeY=
```

  - 注意：`yml`文件是不能使用加密/解密的

Spring Cloud Bus

可实现更改配置后自动应用到应用

Spring Cloud Hystrix

实现了线程隔离、断路器

注解

- `@EnableCircuitBreaker`
- `@HystrixCommand(fallbackMethod = "fallback")`
- 此处可以了解一下注解：`@SpringCloudApplication`

Hystrix-服务降级

Hystrix-依赖隔离

原理

> “舱壁模式”对于熟悉Docker的读者一定不陌生，Docker通过“舱壁模式”实现进程的隔离，使得容器与容器之间不会互相影响。而Hystrix则使用该模式实现线程池的隔离，它会为每一个Hystrix命令创建一个独立的线程池，这样就算某个在Hystrix命令包装下的依赖服务出现延迟过高的情况，也只是对该依赖服务的调用产生影响，而不会拖慢其他的服务。

博客中也对**打消Hystrix线程池隔离技术对性能影响的顾虑**进行了解释；也有关于 `信号量` 的介绍；

- 线程池方式下 `业务请求线程` 和 `执行依赖服务的线程` 不是同一个线程
- 信号量方式下 `业务请求线程` 和 `执行依赖服务的线程` 是同一个线程

> 如果通过信号量来控制系统负载，将不再允许设置超时和异步化，这就表示在服务提供者出现高延迟，其调用线程将会被阻塞，直至服务提供者的网络请求超时，如果对服务提供者有足够的信心，可以通过信号量来控制系统的负载。