1 背景

随着我们的易云openAPI服务的开发进行中，测试部门对于测试工具的选择也没闲着。
由于Jmeter做性能测试、接口测试，也就选择了对应的Jmeter来对我们写的代码进行测试。那么问题来了！
Jmeter支持常用的HTTP等协议，但是我们产品给出的鉴权方式稍有区别，用这种常规的方式无法达到测试的目的。
我们自己的API鉴权的特定点：
signature测试妹纸每次测试的时候无法自己生成这个值，也就意味着想要用Jmeter就隔着一堵墙。
测试人员无法自己按照我们产品自己定义的生成方式去生成这个signature。
那么，我们来学习一下如何在Jmeter插件中做一个协议插件，嵌入到Jmeter中，实现我们的需求，为测试的妹纸提供这个便利的服务。

2 目标

为测试人员提供Jmeter插件，在录入的HttpRequest请求的RequestBody中自动添加Signature参数，使之可以访问EayunOpenAPI服务，方便测试人员测试。

3 准备材料

1.JMeter源码集成到Eclipse

1.JMeter源码集成到Eclipse

> 在Jmeter官网上下载Jmeter插件源码包，导入到我们的工程，可以为我们开发插件提供一些参考的地方。
> 具体下载地址：http://jmeter.apache.org/download_jmeter.cgi
> 具体的源码导入工程操作请参考：http://www.cnblogs.com/taoSir/p/5144274.html

2.查找资料

2.网上查找资料

> 从网上查到，开发一个插件大致需要两个部分来实现：
>
> 1.TestSampler
>
> 2.TestSamplerGUI

那么我们来看一下大致的代码

TestSampler类代码参考如下：

```
public class TestSampler extends AbstractSampler {
    public final static String FUNCTION = "function";
    @Override
    public SampleResult sample(Entry entry) {
        // TODO Auto-generated method stub
        SampleResult res = new SampleResult();
        res.sampleStart();
        System.out.println(this.getProperty(FUNCTION));//输出GUI界面所输入的函数方法返回结果
        res.sampleEnd();
        res.setSuccessful(true);
        return res;
    }
}
```

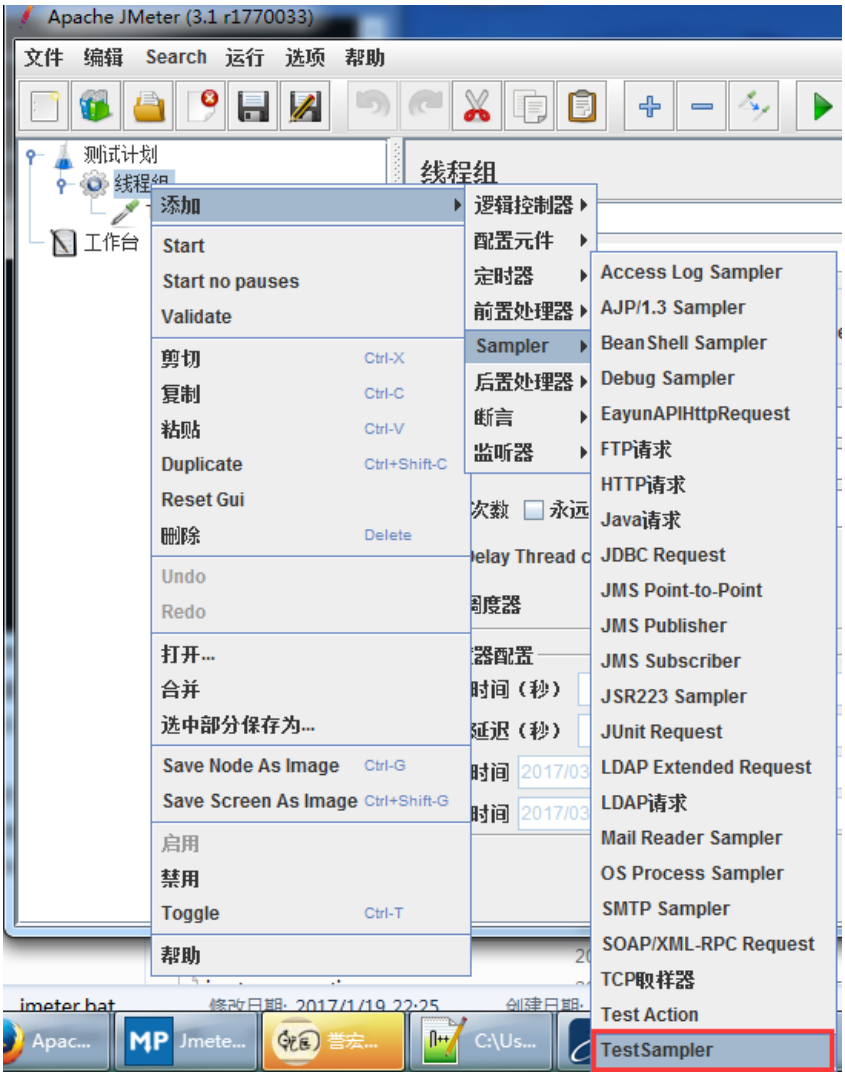
TestSamplerGUI类代码参考如下：

```
public class TestSamplerGUI extends AbstractSamplerGui{
    private JTextField functionTextField = null;
    public TestSamplerGUI(){
        init();
    }
```

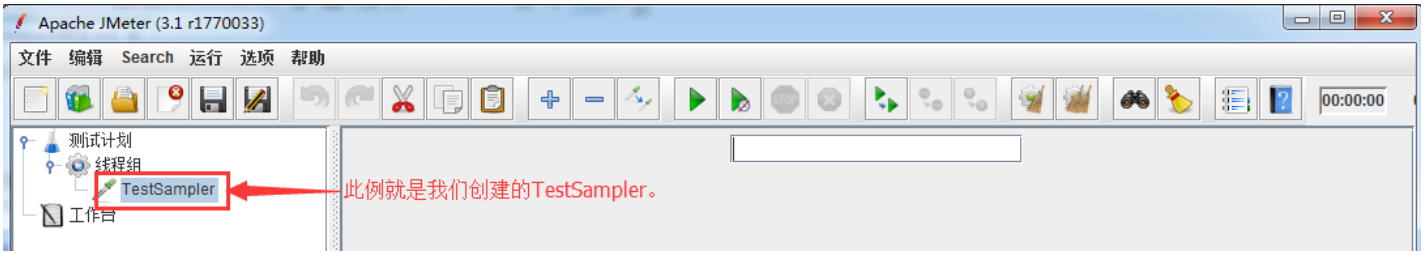
```
    @Override
    public void configure(TestElement element) {

        super.configure(element);
        functionTextField.setText(element.getPropertyAsString(TestSampler.FUNCTION));
    }

    private void init() {

        JPanel mainPanel = new JPanel(new GridBagLayout());
        functionTextField = new JTextField(20);
        mainPanel.add(functionTextField);
        add(mainPanel);
    }

    @Override
    public String getStaticLabel() {//设置显示名称
        // TODO Auto-generated method stub
        return "TestSampler";
    }

    @Override
    public TestElement createTestElement() {//创建所对应的Sampler
        // TODO Auto-generated method stub

        TestElement sampler = new TestSampler();
        modifyTestElement(sampler);
        return sampler;
    }

    @Override
    public String getLabelResource() {
        // TODO Auto-generated method stub
        return this.getClass().getSimpleName();
    }

    @Override
    public void modifyTestElement(TestElement sampler) {
        // TODO Auto-generated method stub
        super.configureTestElement(sampler);
        if (sampler instanceof TestSampler) {
            TestSampler testSmpler = (TestSampler) sampler;
            testSmpler.setProperty(TestSampler.FUNCTION, functionTextField.getText());      }
    }

    private void initFields(){
        functionTextField.setText("");
    }

    @Override
    public void clearGui() {
        super.clearGui();
        initFields();
    }

}
```

> 好了，这样我们就通过上面的简单代码完成了测试用的Sampler。
> export 导出jar包，放到apache-jmeter-3.1\lib\ext目录下。

JMeter每次启动就会加载这个jar包，我们就可以看到该Sampler出现在Sampler列表中：



添加后的拆件结果为：



3.代码分析

需求:我们需要用的是一个sampler组件下的一个类似“HTTP”请求的插件。

> 从上面的例子中可以得出，实现一个类似HTTP的插件其实并不难.
> 只要创建出两个类:
> public class TestSampler extends AbstractSampler
> public class TestSamplerGUI extends AbstractSamplerGui
> 用好这两个类 我们就能实现我们的需求。

3.1 两个关键点

> 由AbstractSamplerGui 来生成页面展示。
> 其中的createTestElement()中new出了一个我们自己定义的Sampler-->TestSampler。
> 那么，我们大概只需要将TestSampler的 public SampleResult sample(Entry entry){}重构一下，应该会实现我们的需求。

> 通过调试，我们可以得到 sample（）方法中可以获取到我们在Jmeter中输入的数据参数的。

3.1.1 HTTPSampleResult

我们再来看一下这个类 HTTPSampleResult extends SampleResult

```
/**
 * This is a specialisation of the SampleResult class for the HTTP protocol.
 *
 */                    这是一个专业化的HTTP协议SampleResult类
public class HTTPSampleResult extends SampleResult {

    private static final long serialVersionUID = 241L;

    /** Set of all HTTP methods, that have no body */
    private static final Set<String> METHODS_WITHOUT_BODY = new HashSet<>(
            Arrays.asList(
                    HTTPConstants.HEAD,
                    HTTPConstants.OPTIONS,
                    HTTPConstants.TRACE));
    private String queryString = ""; // never null   貌似我们看到了查询我们输入的参数的字段了。开心。。。
    private String cookies = ""; // never null

    private String method;
```

通过queryString我们找到了方法getSamplerData(),这个方法记录了我们再Jmeter中输入的数据的参数值

```
@Override
public String getSamplerData() {
    StringBuilder sb = new StringBuilder();
    sb.append(method);
    URL u = super.getURL();
    if (u != null) {
        sb.append(' ');
        sb.append(u.toString());
        sb.append('\n');
        // Include request body if it can have one
        if (!METHODS_WITHOUT_BODY.contains(method)) {
            sb.append("\n").append(method).append(" data:\n");
            sb.append(queryString);
            sb.append('\n');
        }
        if (cookies.length()>0){
            sb.append("\nCookie Data:\n");
            sb.append(cookies);
        } else {
            sb.append("\n[no cookies]");
        }
        sb.append('\n');
    }
    final String sampData = super.getSamplerData();
    if (sampData != null){
        sb.append(sampData);
    }
    return sb.toString();
}
```

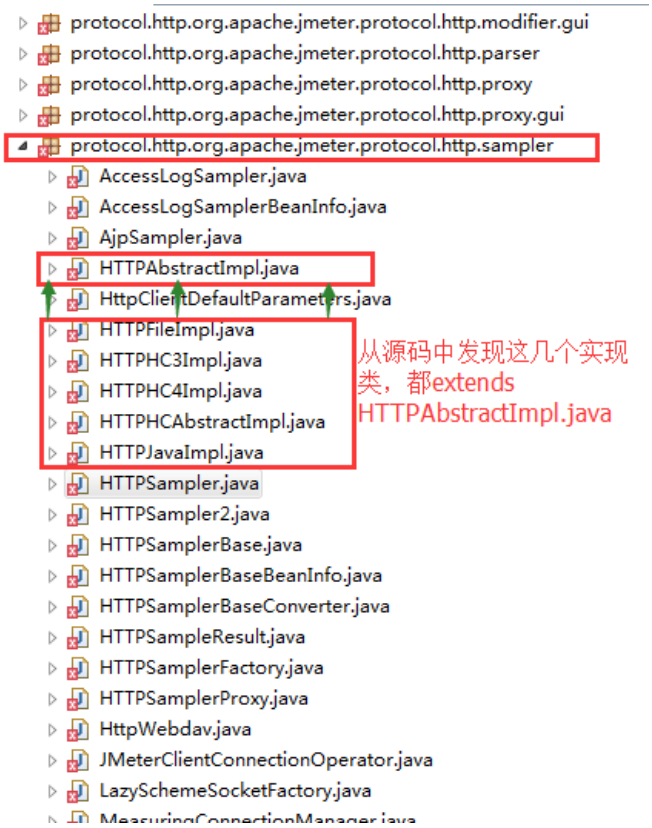
> 那么，我们就可以将之前TestSamplerGUI的createTestElement()方法调整一下。

```
@Override
public TestElement createTestElement() {//创建所对应的Sampler
    // TODO Auto-generated method stub
    TestElement sampler = new TestSampler();
    modifyTestElement(sampler);
```

```
    return sampler;
}

@Override
public TestElement createTestElement() {//创建所对应的Sampler
    HTTPSamplerBase sampler = new EayunHTTPSampler();
    modifyTestElement(sampler);
    return sampler;
}
```

3.1.2 HTTPAbstractImpl

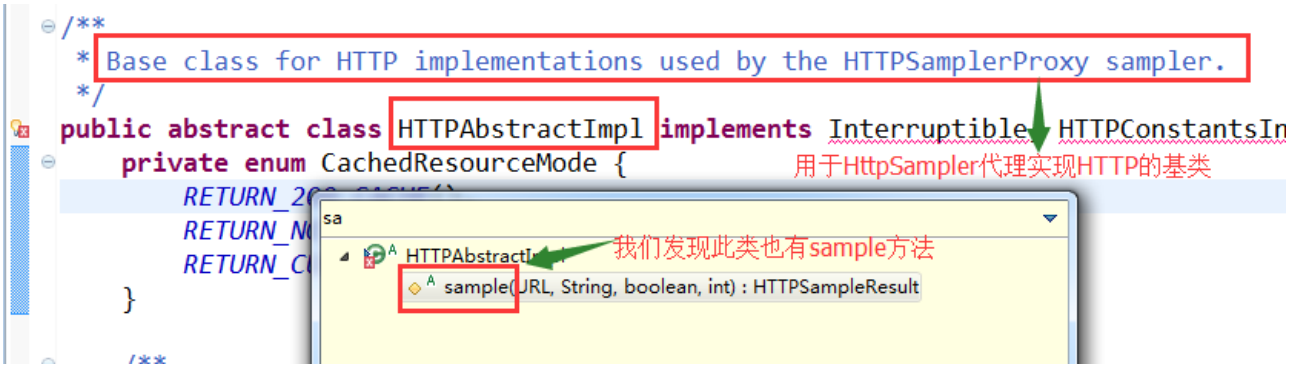
那么，我们如何在得到我们需要数据呢？



通过源码，我们发现：

发现这个类是实现了HTTPSamplerProxy的一个基类

然而这个方法也有我们需要重构的Sampler（）方法。离目标又进了一步。



在HTTPAbstractImpl中，我们发现了getArguments(),通过调试发现可以获取到我们传入的参数。

```
/**
 * Invokes {@link HTTPSamplerBase#getArguments()}
 * @return the arguments of the associated test element
 */
```

```
protected Arguments getArguments() {
    return testElement.getArguments();
}
```

> 这样我们就可以将拿到的值，进行重新拼装，然后再Jmeter返回的时候，把值再添加回去即可完成任务。

如下就是我们再我们的Sampler中实现的业务逻辑代码

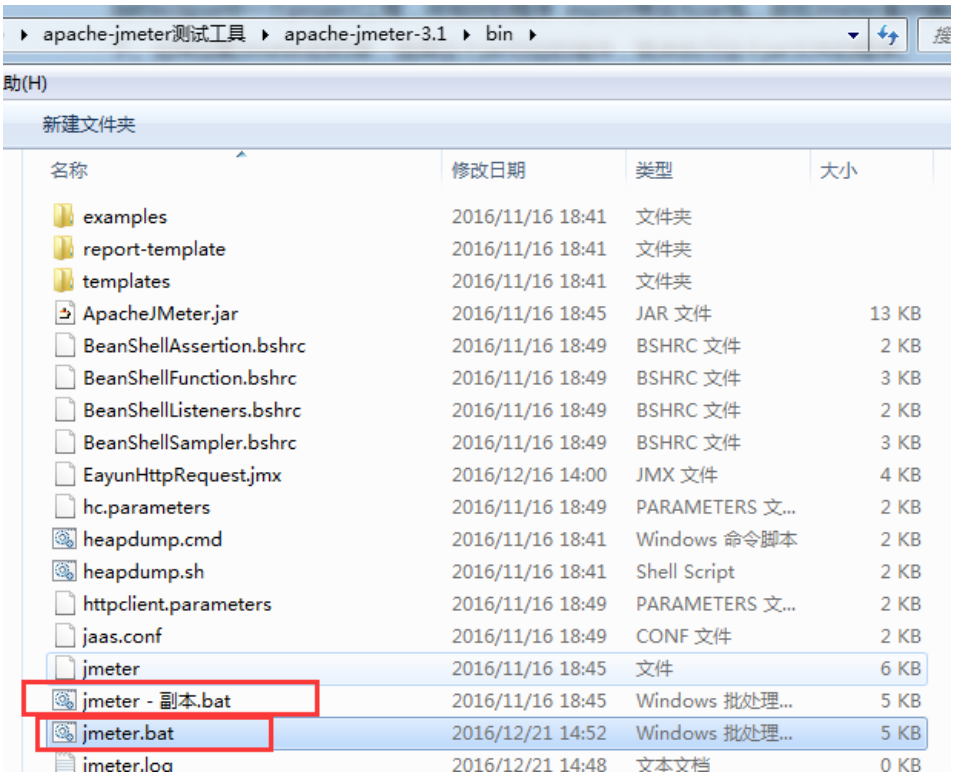
```
    protected HTTPSampleResult sample(URL u, String method,boolean areFollowingRedirect, int depth) {
        if (impl == null) {
            try {
                impl = HTTPSamplerFactory.getImplementation(
                        getImplementation(), this);
            } catch (Exception ex) {
                return errorResult(ex, new HTTPSampleResult());
            }
        }
        Arguments arguments = impl.getArguments();
        FunctionProperty functionProperty = (FunctionProperty) arguments.getArgument(0).getProperty(Argument.VALUE);
        JSONObject jsonObject = null;
        // 获取读取文件的requestBody数据
        jsonObject = (JSONObject) JSONObject.parse(functionProperty.toString());
        // 获取Jmeter RequestBody的输入参数 例："Action": "${Action}",
        CompoundVariable compoundVariable = (CompoundVariable) functionProperty.getObjectValue();
        String rawParameters = compoundVariable.getRawParameters();
        JSONObject json = JSONObject.parseObject(rawParameters);
        // 如果请求的参数中没有Timestamp，系统按当前时间自动加入这个参数
        if (null == jsonObject.get("Timestamp")) {
            Date nowDate = new Date();
            System.out.println("获取系统时间，格式UTC格式化前：" + nowDate);
            System.out.println("系统时间格式UTC格式化后："
                    + SignatureUtil.getUTCDateZ(nowDate));
            jsonObject.put("Timestamp", SignatureUtil.getUTCDateZ(nowDate));// 给用于生成signature的Object添加Timestamp参数；
            json.put("Timestamp", SignatureUtil.getUTCDateZ(nowDate));// 给请求json添加Timestamp
        }
        try {
            json.put("Signature", SignatureUtil.addSignature(jsonObject));
            json.remove("SecretKey");
        } catch (Exception e1) {
            e1.printStackTrace();
        }
        rawParameters = json.toJSONString();
        try {
            compoundVariable.setParameters(rawParameters);
        } catch (InvalidVariableException e) {
            e.printStackTrace();
        }
        functionProperty.setRunningVersion(false);
        functionProperty.setObjectValue(compoundVariable);
        functionProperty.setRunningVersion(true);
        functionProperty.recoverRunningVersion(null);

        if (notifyFirstSampleAfterLoopRestart) {
            impl.notifyFirstSampleAfterLoopRestart();
            notifyFirstSampleAfterLoopRestart = false;
        }
        return impl.sample(u, method, areFollowingRedirect, depth);
    }
```

这里我们并没有关心什么时候用什么方法去将我们拦截到的参数处理后进行返回。 只是在他的Sampler方法中处理完，无需关心其它业务就可以实现我们的需求。

4 Jmeter开启远程调试

> 远程调试如何设置：

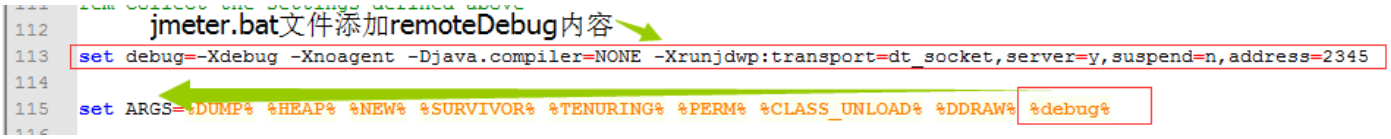
第一步:需要在Jmeter的启动文件（jmeter.bat）添加设置 debug参数



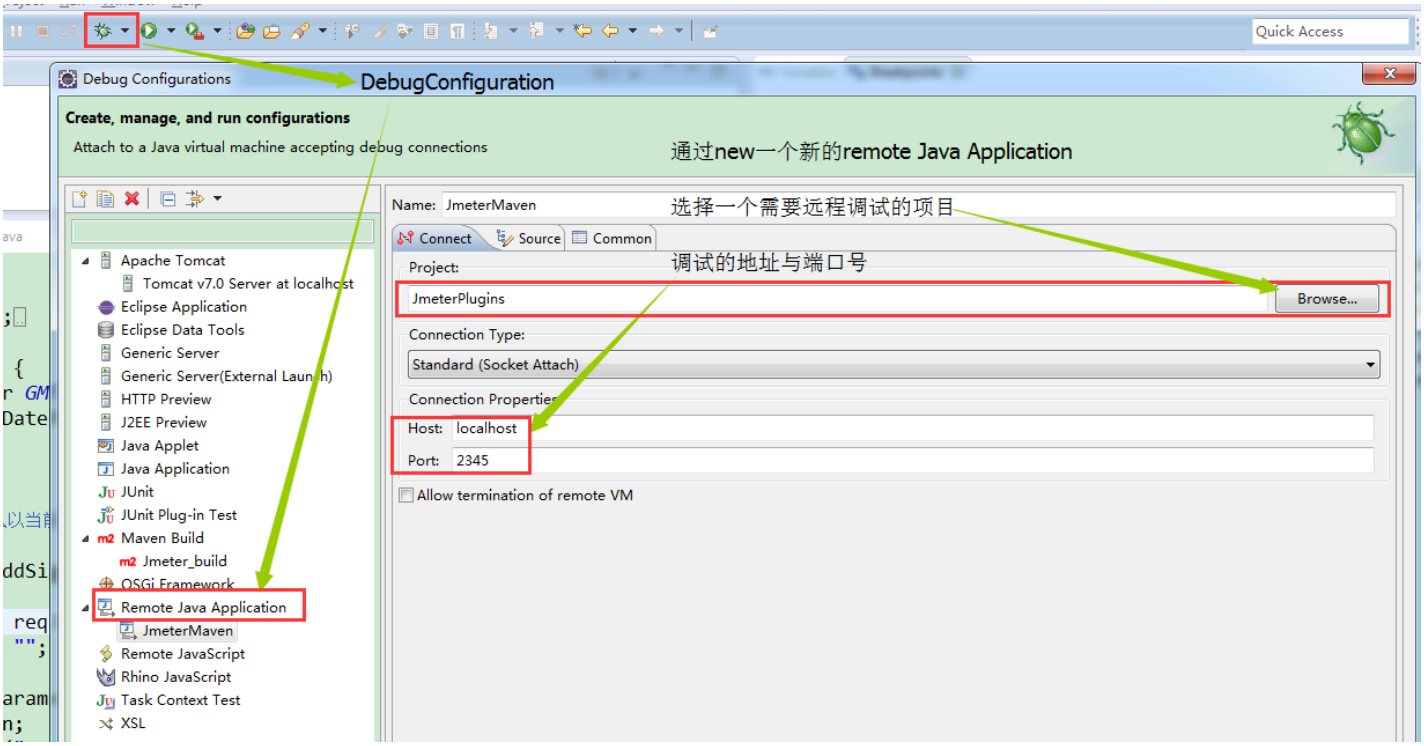
在启动程序时，将以下参数选项添加到自定义的命令行中，程序就会以支持RemoteDeubg的方式启动。

```
-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=2345

set debug=-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=2345
```

其中address=2345 表示 调试端口为2345



第二步：在eclipse中添加这个工程的debug 远程调试参数



第三步：Jmeter发送一个http请求，就会执行我们的Jar包程序，通过远程调试参数配置。
我们的eclipse就会进入debug模式 进入F6 一步步调试代码，看哪里出错了。
等Jar的程序执行完毕后，就会将请求的数据通过HTTP请求发送到请求的服务端进行处理。