对 Java 请求的测试实施

概要

本篇文章主要介绍以下三个方面的内容。

1. 准备工作
2. 用Jmeter测试JAVA工程
3. 总结

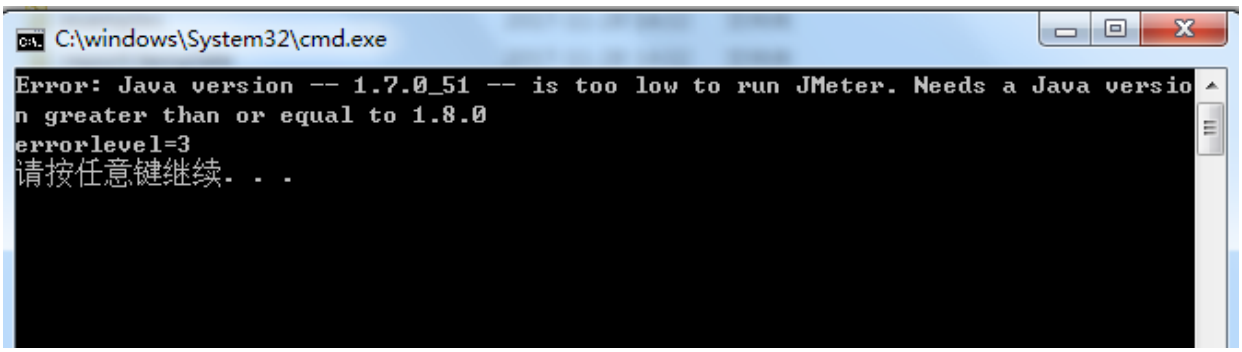
准备工作

主要包括以下方面：

1. JDK的注意事项
2. Jmeter的配置
3. Eclipse工具的注意事项

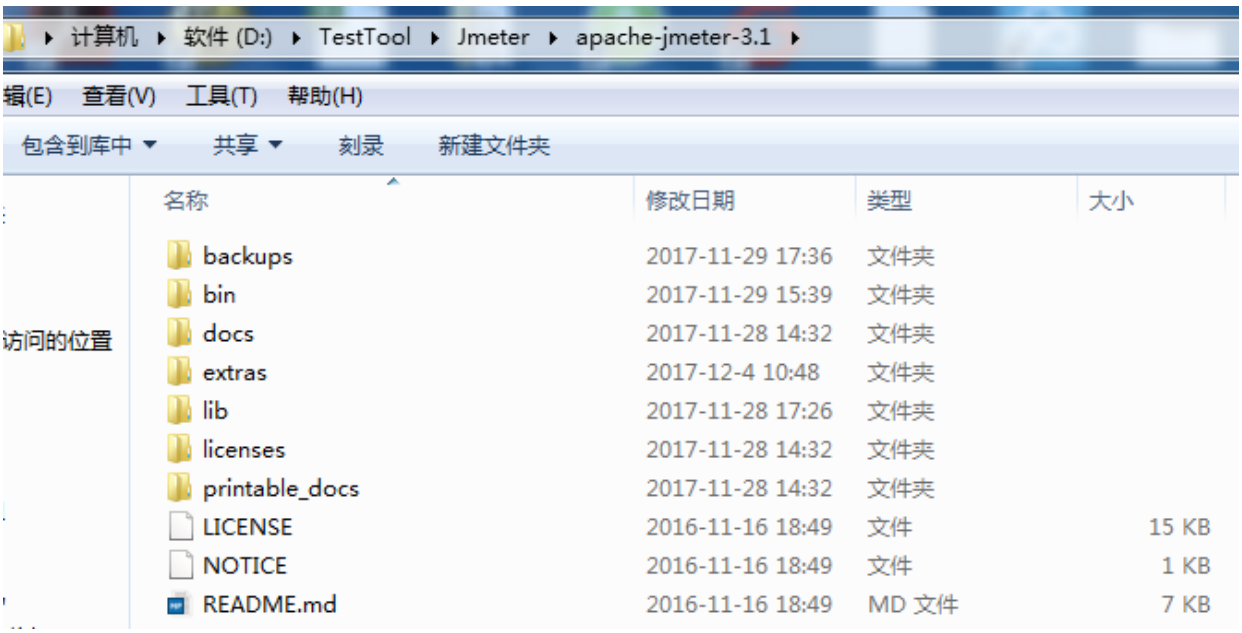
JDK的注意事项

Jmeter是依赖JDK的，目前从官网下载下来的Jmeter支持JDK1.7和1.8；这里推荐大家自行安装JDK1.8的版本。如果JDK版本过低，启动Jmeter时会报错，报错信息如下：



Jmeter的配置

1）自行下载Jmeter（从官网上下载最新版本即可），下载到本地进行解压缩。解压后的目录结构如下图所示：



2）配置环境变量

新建系统变量为：JMETER_HOME;

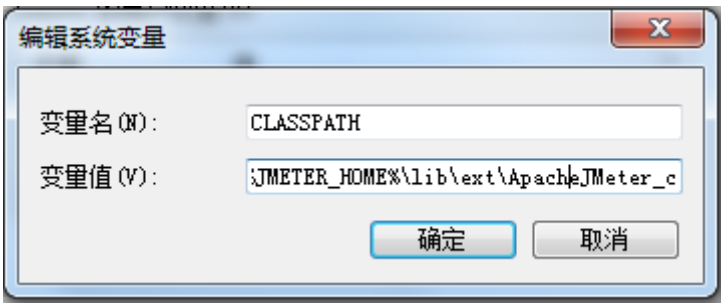
变量值为：D:\TestTool\Jmeter\apache-jmeter-3.1

配置CLASSPATH（没有的话也新建），变量值为：

%JMETER_HOME%\lib\ext\ApacheJMeter_core.jar;

%JMETER_HOME%\lib\jorphan.jar;

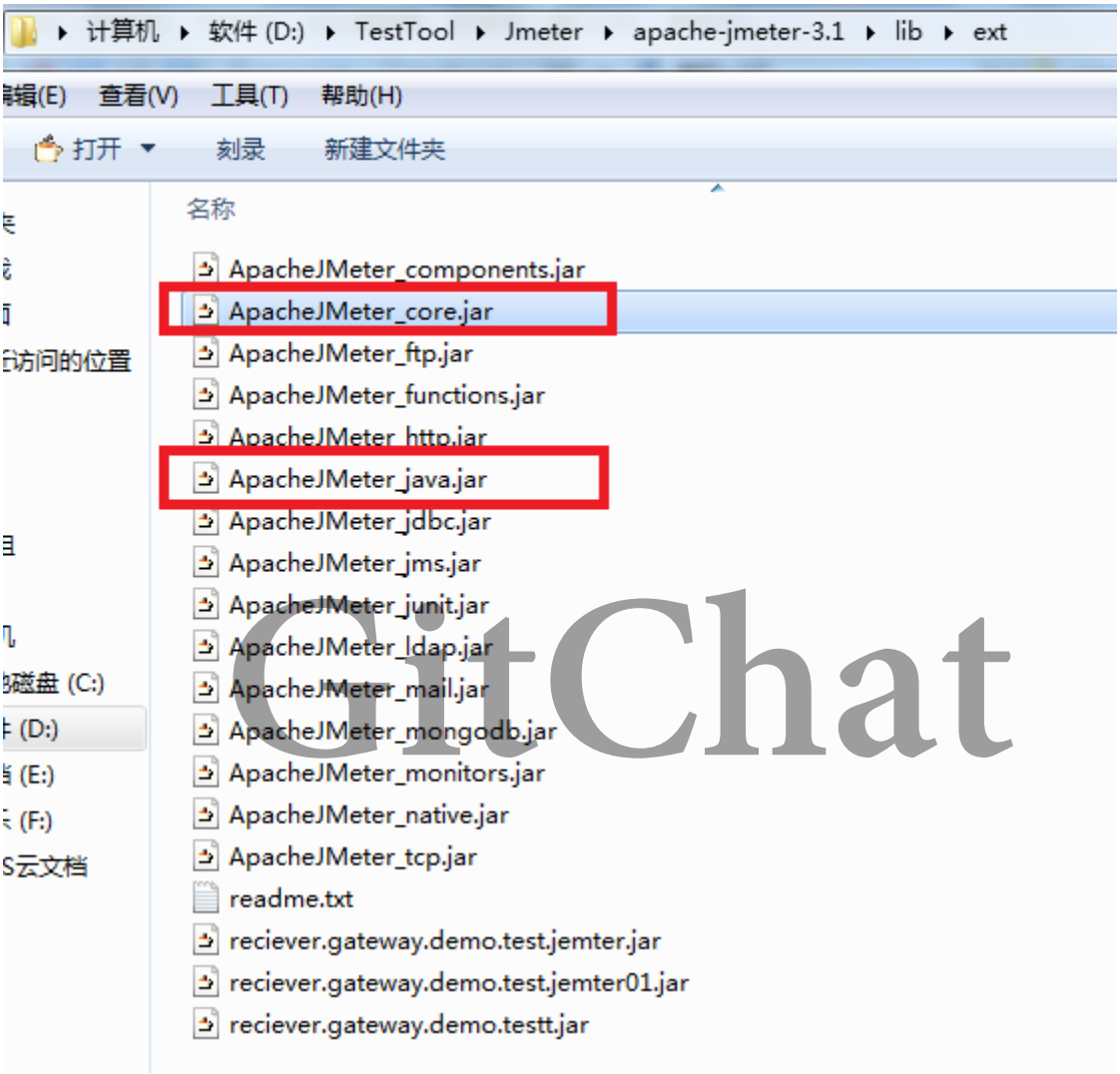
%JMETER_HOME%\lib\logkit-2.0.jar;如果没有其他值，那么前面应该加.;这个三个jar必须配置在CLASSPATH中。



Eclipse工具的注意事项

1）自行下载Eclipse,请注意下载的Eclipse版本要和JDK的版本保持一致，即64位的JDK对应的Eclipse应该是64位，32位JDK对应的Eclipse版本应该是32位。

2）将apache-jmeter-3.1\lib\ext目录下的ApacheJMeter_core.jar和ApacheJMeter_java.jar导入到JAVA工程中；

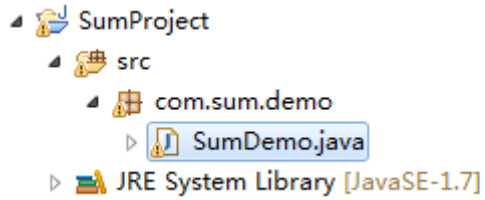


这一步是关键，反编译ApacheJMeter_java.jar，可以看到JavaSampler类，这即是Jmeter可以测试JAVA请求的关键所在。

用Jmeter测试JAVA工程

新建工程

在Eclipse里新建一个JAVA工程。

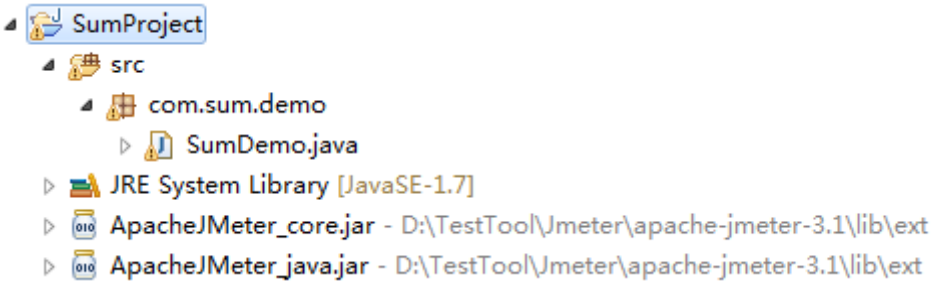


定义一个类为SumDemo：实现两个数的求和，代码如下：

```
package com.sum.demo;
class Math{
    private int a;
    private int b;
    //定义一个方法实现a+b的和
    public int sumTest(int a,int b){
        return a+b;
    }
}
public class SumDemo {

    public static void main(String[] args) {
        Math m = new Math();
        int c = m.sumTest(10, 20);
        System.out.println("两数相加之和:"+c);
    }

}
```


导入jar

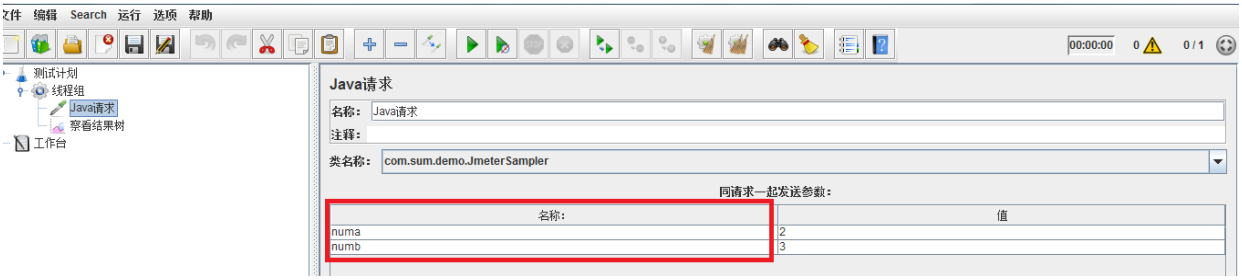
向工程里导入ApacheJMeter_core.jar和ApacheJMeter_java.jar。



编写辅助函数

要想将Java的jar导入到Jmeter的Java请求中，需要编写Jmeter的辅助方法，常用的方法如下：

1）public Arguments getDefaultParameters()，用于获取Jmeter界面上的参数。如下所示：



2）public void setupTest(JavaSamplerContext context),每个线程只执行一次，编写一些初始化数据，类似于LoadRunner中的JAVA Vuser中的Init()函数和Junit的setUp();

3）public SampleResult runTest(JavaSamplerContext context)，JmeterSampler的主方法，类似于LoadRunner中的JAVA Vuser中的Action()函数；

4）public void teardownTest(JavaSamplerContext context)，每个线程执行一次，类似于LoadRunner中的JAVA Vuser中的end()函数和Junit的tearDown();测试结束时调用，常用来编写关闭流资源，关闭数据库连接，没有的话，就什么都不要写。

上述方法在执行的时候是有先后顺序的，类似于Testng，执行的先后顺序为：

getDefaultParameters()>setupTest(JavaSamplerContext context)>runTest(JavaSamplerContext context)>teardownTest(JavaSamplerContext context)
以上的四个方法组成了JavaSampler的基本结构，缺一不可。

除了上述四个主要方法外，还有如下方法：

1. sampleStart()用来定义一个事务的开始;
2. sampleEnd()用来定义一个事务的结束;
3. addArgument(“参数名称”,”参数值”)，定义参数；
4. setSuccessful（”true/false”），设置运行结果的成功和失败，用来帮助Jmeter统计成功、失败的次数，并在聚合报告中给与体现；

查看结果树的显示方法

必须在程序中编写如下代码，才能将运行结果输出在”查看结果树”中。代码如下：

```
if (resultData != null && resultData.length() > 0) {
                        sr.setResponseData(resultData, null);
    //请求头中会多个类型输出 Data type ("text"|"bin"|""): text
                        sr.setDataType(SampleResult.TEXT);
                    }
```

代码中的resultData为String的实例对象，sr为SampleResult的实例对象；

JmeterSampler的编写

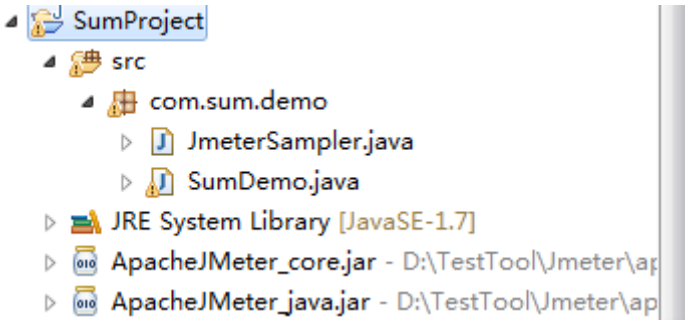
为了实现SumDemo类中的求和功能，编写的JmeterSampler的代码如下所示：

```
package com.sum.demo;

import org.apache.jmeter.config.Arguments;
import
org.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient;
import org.apache.jmeter.protocol.java.sampler.JavaSamplerContext;
import org.apache.jmeter.samplers.SampleResult;

public class JmeterSampler extends AbstractJavaSamplerClient{
    private Math ma=null;
    private String a;
    private String b;
    private String resultData;
    /**通过SumDemo我们知道我们需要传入参数a,b,
     * 所以使用这个方法来定义方法的入参*/
      public Arguments getDefaultParameters() {
            Arguments params = new Arguments();
            params.addArgument("numa", "");
            params.addArgument("numb", "");
            return params;
        }
      //做一些初始化操作，每个线程执行一次
       public void setupTest(JavaSamplerContext context) {
              System.out.println("执行开始");
              ma= new Math();
             a = context.getParameter("numa");
             b = context.getParameter("numb");
        }
    @Override
    public SampleResult runTest(JavaSamplerContext context) {
        SampleResult sr = new SampleResult();
        sr.setSamplerData("请求参数numa：  "+a+"\t"+"请求参数numb：  "+b);
        try{
            sr.sampleStart();//事务开始
            resultData =
String.valueOf(ma.sumTest(Integer.parseInt(a), Integer.parseInt(b)));
        if (resultData != null && resultData.length() > 0) {
            sr.setResponseData("结果是：  "+resultData, null);
            sr.setDataType(SampleResult.TEXT);
            sr.setSuccessful(true);
        }} catch(Throwable e) {
            sr.setSuccessful(false);
            e.printStackTrace();
        } finally {
            sr.sampleEnd();//事务结束
    }
```

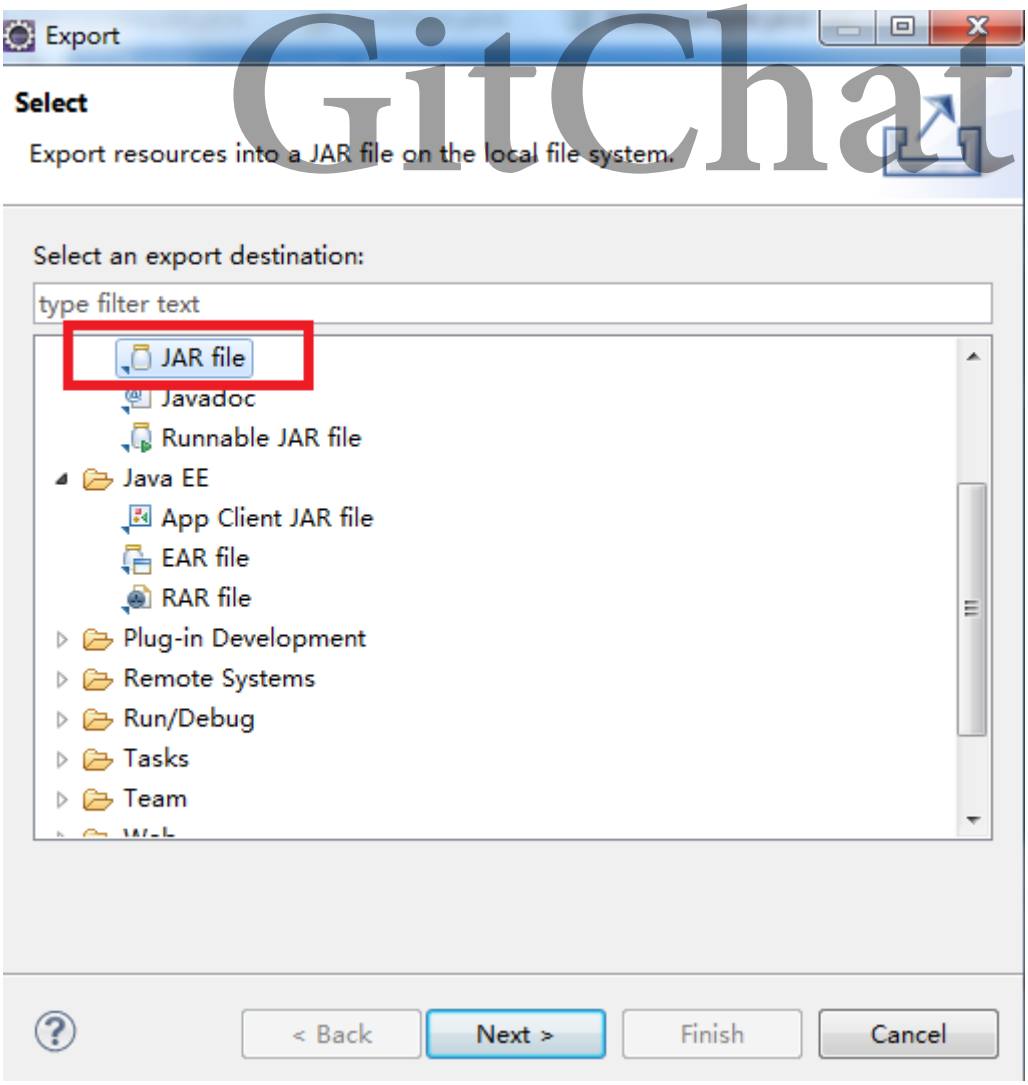
```
            return sr;
        }
        public void teardownTest(JavaSamplerContext context) {
            System.out.println("执行完毕");
        }
    }
```

导出jar

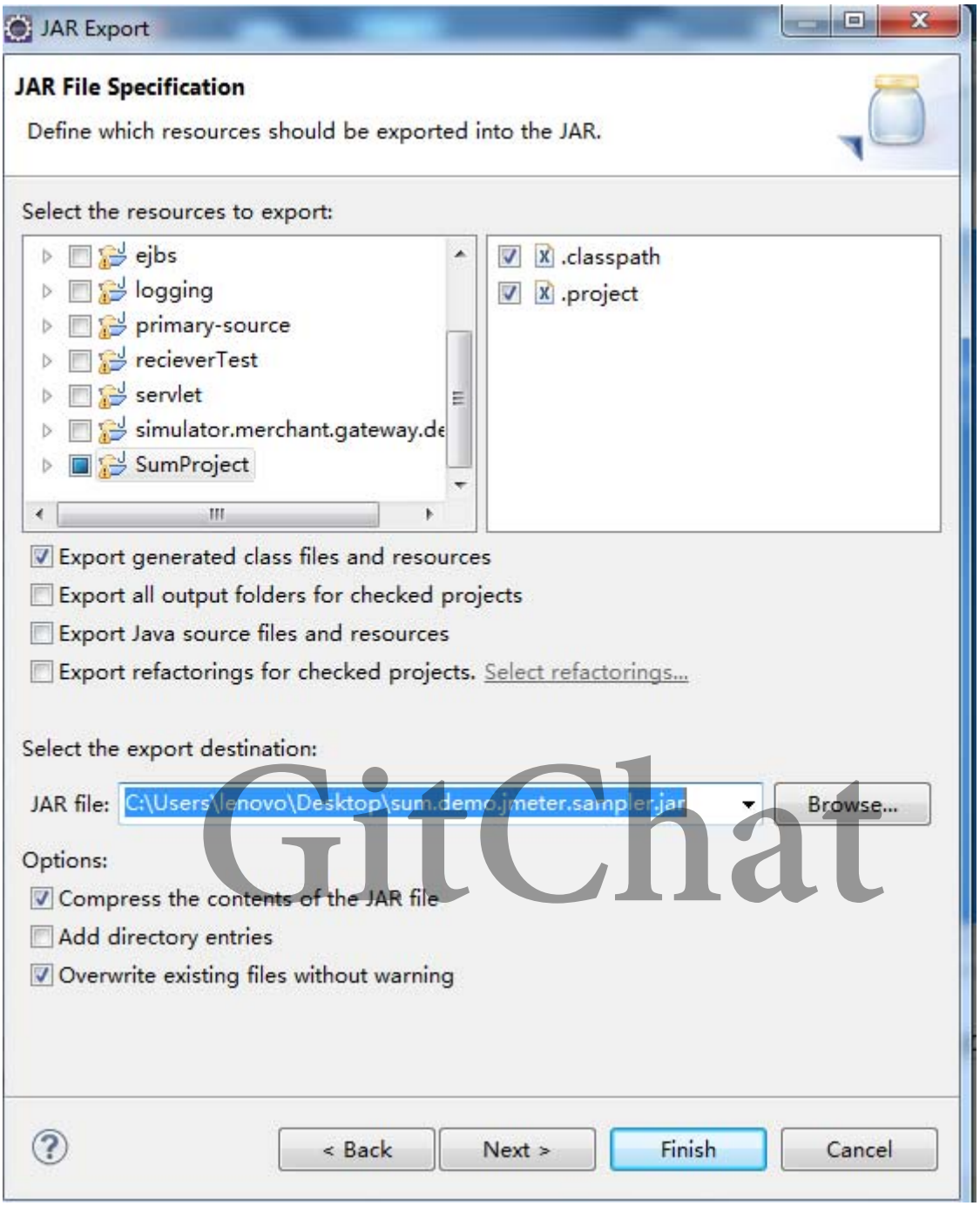
选中此时的Java工程；将jar命名为com.demo.jmeterTest。



右键->Export，选择导出jar;



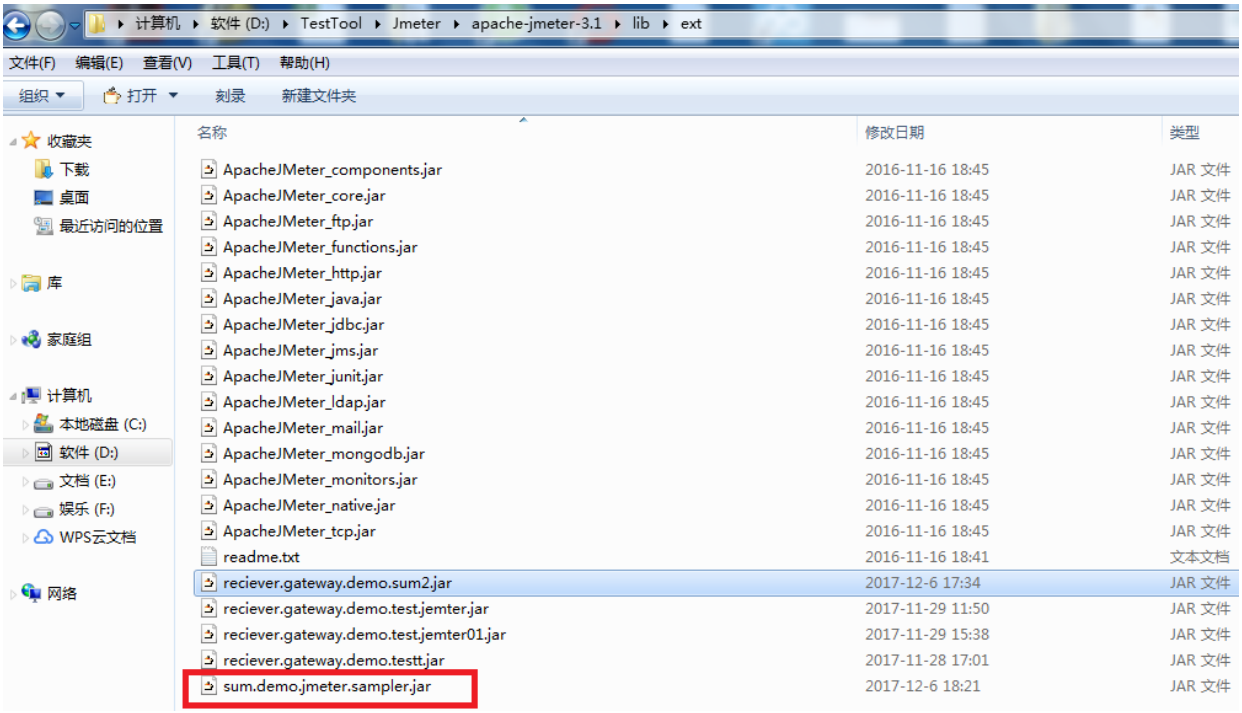
点击next，选择JAR file，并将jar命名为：sum.demo.jmeter.sampler.jar(名字任取哦)



点击finish，此时jar就成功导出了。

将jar导入到Jmeter

将sum.demo.jmeter.sampler.jar复制到\apache-jmeter-3.1\lib\ext目录下，如果JAVA工程还依赖有其他的jar包，我们就将其全部复制到\apache-jmeter-3.1\lib目录下。

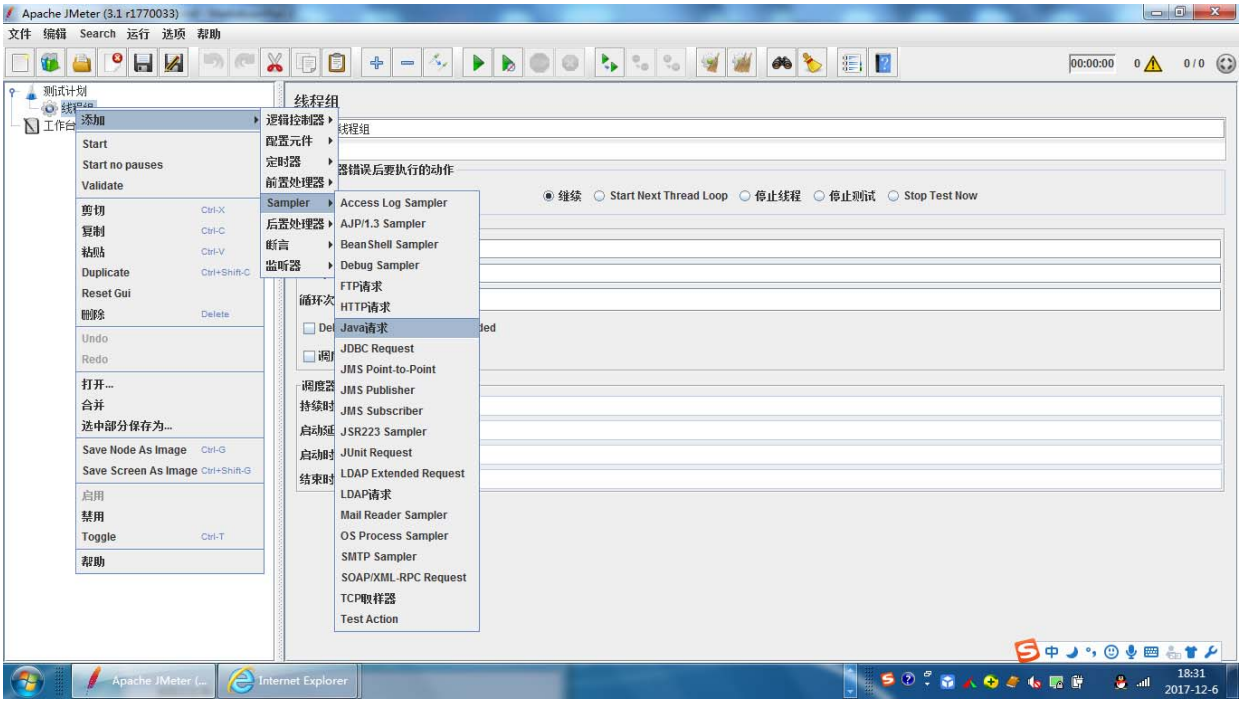


Jmeter运行JAVA工程

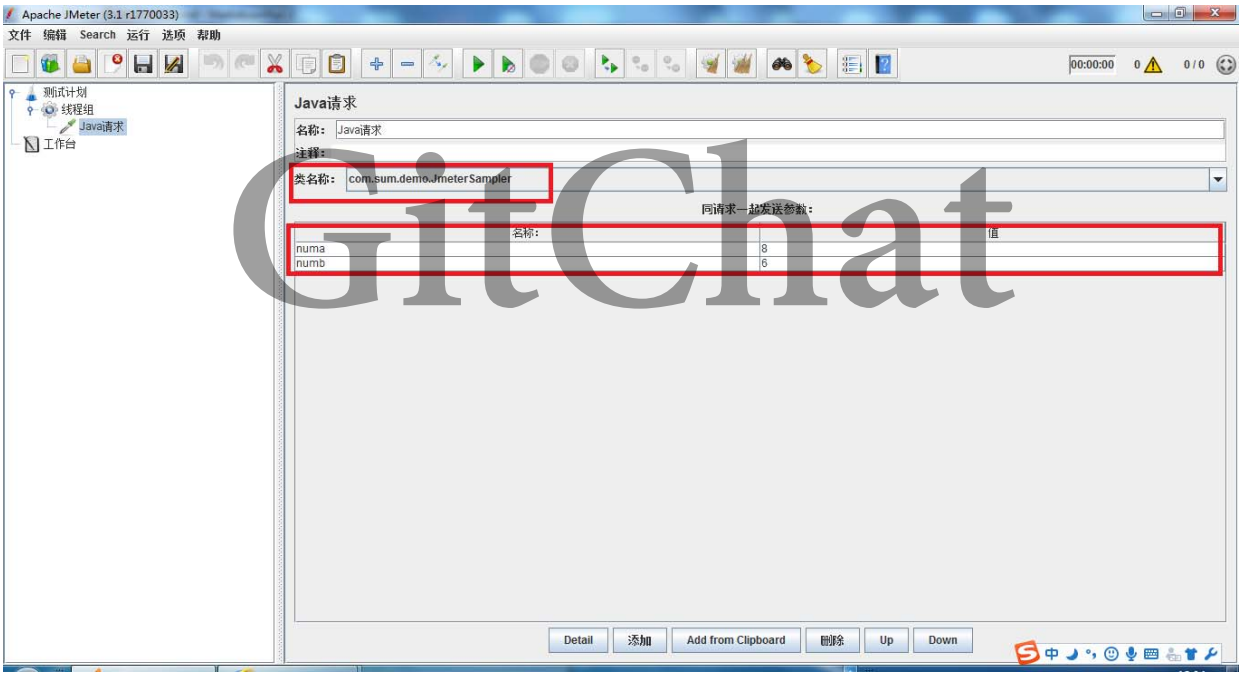
以管理员身份打开Jmeter后，

1）测试计划->添加->Threads(Users)->线程组

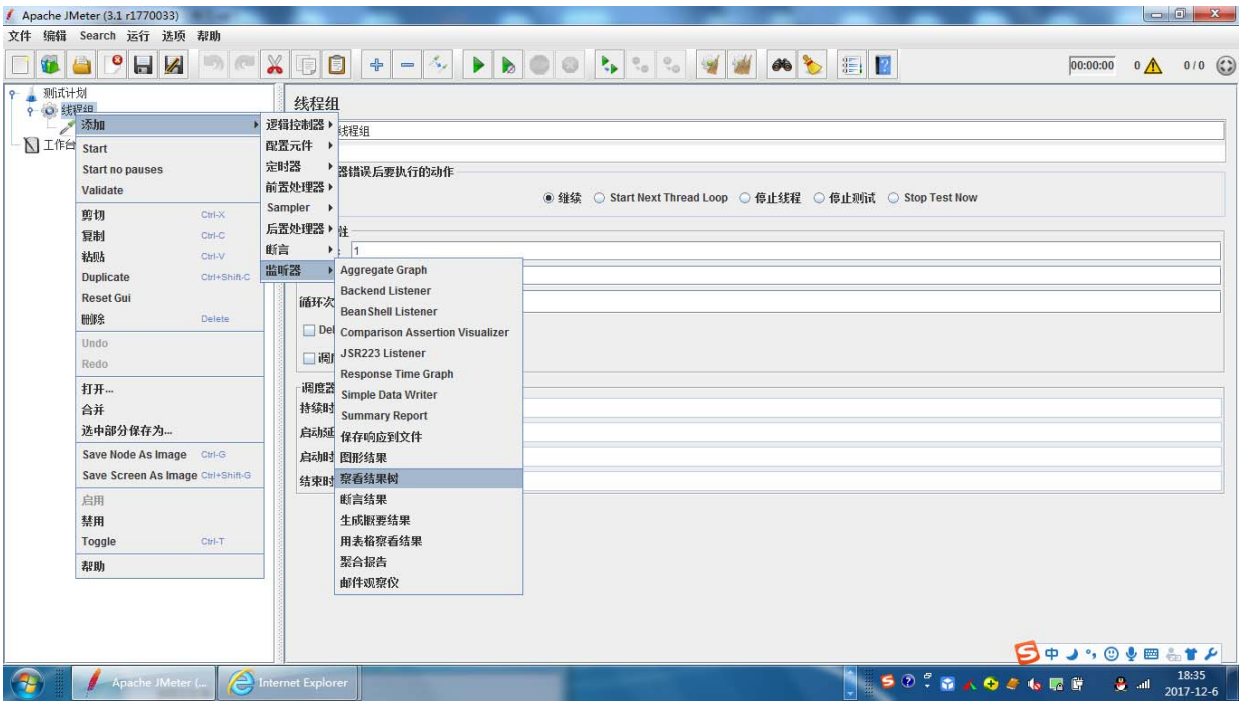
2）线程组->添加->Sampler->Java请求



3）选择我们测试的类名称并且给参数numa和numb分别复制为8和6；

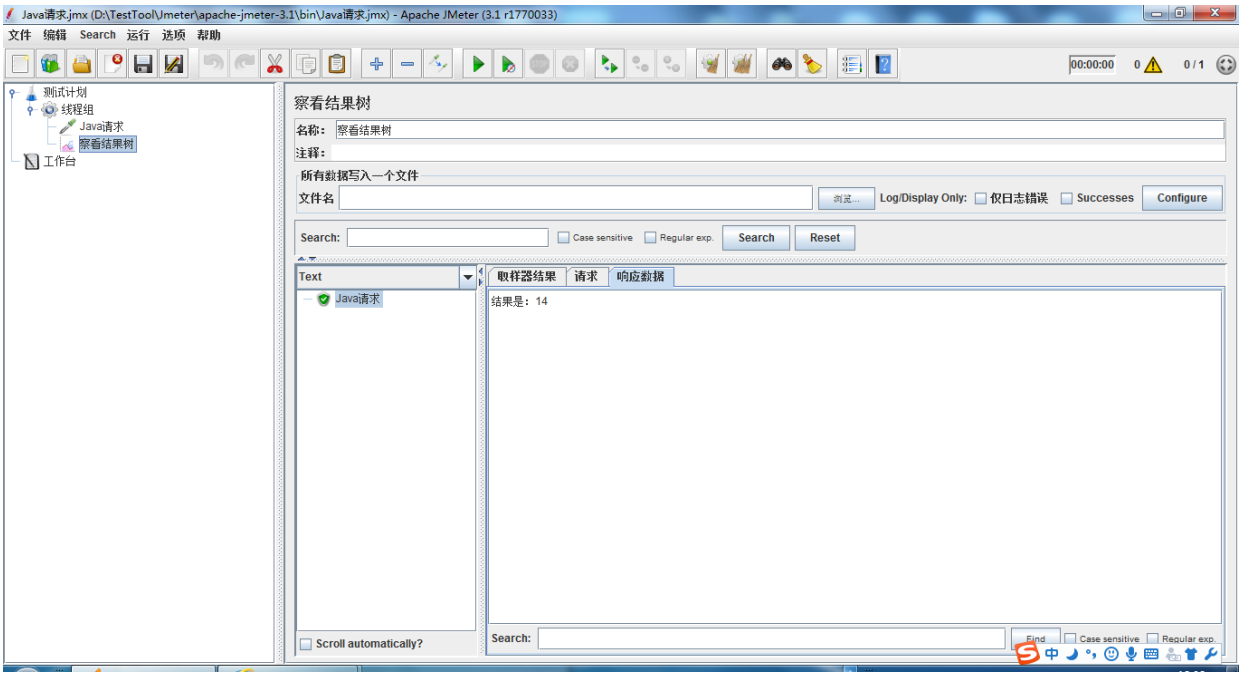


4）线程组->添加->监听器->查看结果树



5）运行

6）查看结果树



此时可以看到JAVA工程已经被成功调用，并且在”查看结果树”中成功打印了请求和响应信息。响应结果也与预期相符。

总结

通过本次实践过程，总体说明如下：

1. 当测试过程中出现性能问题时，可以继续拆分事务，以便更好的定位性能瓶颈。
2. Jmeter要想加载出我们的测试类，必须满足以下条件：
    - Jmeter运行所依赖的JDK版本>JAVA工程编译所使用的JDK版本；
    - 必须在JAVA工程里编写继承自AbstractJavaSamplerClient的测试类；
3. JavaSampler的编写是有先后顺序的，方法也是固定统一的，一定要认真掌握。