开源性能测试工具Apache JMeter快速入门

目录

[目录 1](#_Toc11205)

[1 Apache JMeter概述 3](#_Toc4590)

[1.1 Apache JMeter优点 3](#_Toc17816)

[1.2 Apache JMeter缺点 3](#_Toc12826)

[1.3 Apache JMeter所支持的协议 4](#_Toc26694)

[2 JMeter的安装与使用 5](#_Toc23577)

[2.1 安装 5](#_Toc24912)

[2.2 主要功能模块（组件） 5](#_Toc1161)

[2.2.1 测试计划（Test Plan）---相当于场景设计（单、多） 5](#_Toc17266)

[2.2.2 线程组（threads group） 7](#_Toc2409)

[2.2.3 测试片段（Test Fragment） 8](#_Toc31260)

[2.2.4 取样器（Sampler）： 8](#_Toc16430)

[2.2.5 逻辑控制器（Logic Controller）：少用 12](#_Toc17832)

[2.2.6 监听器（Listener）---analysis 12](#_Toc3481)

[2.2.7 配置元件（Config Element）--维护请求 13](#_Toc20064)

[2.2.8 定时器（Timer）(固定、同步定时器) 13](#_Toc7469)

[2.2.9 断言（Assertions） 13](#_Toc11407)

[2.2.10 前置处理器（Per Processors）--修改请求 15](#_Toc15739)

[3 JMeter快速入门之HTTP脚本实战 16](#_Toc30254)

[3.1 测试计划 16](#_Toc20380)

[3.2 Vuser及场景设计 16](#_Toc9969)

[3.3 JMeter之脚本录制 17](#_Toc27525)

[3.3.1 利用BadBoy进行脚本录制 17](#_Toc7613)

[3.3.2 利用JMeter代理服务器功能进行录制 19](#_Toc10284)

[3.3.3 两种录制方式的区别 21](#_Toc717)

[3.4 JMeter参数化 22](#_Toc11661)

[3.4.1 使用用户自定义变量-- Test Plan(测试计划) 22](#_Toc17430)

[3.4.2 用户定义的变量”测试组件，名称：default variables，变量名与值分别如下(可以用绝对路径) 23](#_Toc10252)

[3.4.3 使用配置元件中的CSV Data Set Config进行参数化设置 24](#_Toc423)

[3.4.4 使用函数助手里的随机函数进行参数化 25](#_Toc12241)

[3.5 JMeter关联 30](#_Toc21009)

[3.6 JMeter检查点—响应断言 33](#_Toc3082)

[3.7 思考时间和集合点—定时器 35](#_Toc18211)

[3.7.1 . 思考时间—两次请求间停留的间隔 35](#_Toc14650)

[3.7.2 集合点—同步定时器 36](#_Toc7552)

[3.8 运行结果分析 37](#_Toc4860)

[3.8.1 聚合报告（Aggregate Report/ Aggregate Graph） 37](#_Toc5586)

[3.8.2 查看结果树 38](#_Toc17634)

[3.8.3 图形结果 38](#_Toc11434)

[3.8.4 用表格查看结果树 39](#_Toc862)

[3.8.5 断言结果 40](#_Toc20044)

[3.9 JMeter之分布式测试—多台负载机测试 40](#_Toc21715)

[4 JMeter JDBC脚本实战 42](#_Toc19963)

[4.1 创建JDBC连接 42](#_Toc13372)

[4.1.1 导入jar包 42](#_Toc25727)

[4.1.2 添加一个“JDBC Connection Configuration”配置元件 42](#_Toc25106)

[4.2 创建JDBC测试用例 44](#_Toc22521)

[4.3 添加监听器 46](#_Toc7507)

[5 JMeter SOAP脚本实战Web Service 47](#_Toc16198)

[6 基于jmeter java协议的接口测试 50](#_Toc3548)

[6.1 通用方法 50](#_Toc9594)

[6.1.1 核心步骤 50](#_Toc12915)

[6.1.2 实例2: extends AbstractJavaSamplerClient 50](#_Toc13886)

[6.1.3 实例：implements JavaSamplerClient 55](#_Toc30767)

[6.1.4 JMeter Java Sampler介绍 59](#_Toc8724)

[6.1.5 自带Java Request Sampler 60](#_Toc3623)

[6.2 jmeter java实现http协议 PerfmonTest代码 61](#_Toc8661)

[6.2.1 源码：类HttpRequest(Get/Post) 61](#_Toc24460)

[6.2.2 解析：发送get/post请求类（HttpRequest） 65](#_Toc31446)

[6.3 socket协议 PerfmonTest代码 70](#_Toc12184)

[6.3.1 源码：发送socket请求的类、方法 70](#_Toc12707)

[6.4 71](#_Toc28156)

[7 JMeter使用注意事项 72](#_Toc18229)

[8 附录A 正则表达式全集 74](#_Toc20399)

[9 附录B MySql数据库事物隔离级别 78](#_Toc31998)

# Apache JMeter概述

官方介绍：Apache JMeter 是一个100％的纯[Java](http://java.sun.com/)桌面应用，用于压力测试和性能测量。它最初被设计用于Web应用测试但后来扩展到其他测试领域。Apache JMeter 可以用于对静态的和动态的资源（文件、Servlet、Web动态语言[[1]](#footnote-0)、[Java](http://java.sun.com/) 对象，数据库和查询、FTP服务器等）的性能进行测试。它可以用于对服务器、网络或对象模拟繁重的负载来测试它们的强度或分析不同压力类型下的整体性能。用户可以使用它做性能的图形分析或在大并发负载测试你的服务器、脚本、对象等。

目前很多公司都基于JMeter开发出了公司级别的自动化性能测试框架，比如使用Maven+Jenkins+JMeter自动化测试框架、JMeter+ant的自动化测试框架等。

## Apache JMeter优点

（1）能够对HTTP和FTP服务器进行压力和性能测试， 也可以对任何数据库进行同样的测试（通过JDBC）。

（2）完全的可移植性和100％ 纯[Java](http://java.sun.com/)。

（3）完全 Swing和轻量组件支持（预编译的JAR使用 javax.swing.\*)包。

（4）完全多线程：框架允许通过多个线程并发取样和通过单独的线程组对不同的功能同时取样。

（5）精心的GUI设计允许快速操作和更精确的计时。

（6）缓存和离线分析/回放测试结果。

（7）完全开源，开源对JMeter进行定制化的二次开发，扩展自己所需的插件。

## Apache JMeter缺点

对比LoadRunner， Apache JMeter也有一些不足，包括：

（1）无类似LoadRunner的IP欺骗功能。

（2）录制功能要借用第三方工具Badboy或者使用浏览器进行代理录制，但是这个录制都不是那么好用。

（3）报表类型较少，没有LoadRunner报表齐全。

（4）上手较LoadRunner困难一些。

（5）场景设计比LoadRunner复杂。

（6）纯多线程模式，不支持进程模式。

（7）网上详细资料较少，所以学习起来稍微费劲，普及程度也没有LoadRunner广。

（8）场景控制以及干预没有LoadRunner方便，比如无法在压力测试执行过程中人工增加并发用户数。

（9）大并发的时候，结果数据不是很准确。

## Apache JMeter所支持的协议

Apache JMeter所支持的包括以下协议**：**

* Web ——HTTP、 HTTPS
* SOAP
* FTP
* JDBC
* LDAP
* 面向消息的中间件（通过JMS）
* 邮件—— SMTP(S)、 POP3(S) 、IMAP(S)
* MongoDB (NoSQL)
* TCP
* 原生命令或者Shell脚本

# JMeter的安装与使用

JMeter的当时最新版本——2.11版，所以书中所有有关JMeter的使用说明以及截图均基于2.11版。下面先讲解如何安装Apache JMeter。

## 安装

1. 下载：JMeter的下载地址为http://jmeter.apache.org/，下载最新版本到本地电脑，直接解压即可。
2. 安装JDK支持：由于JMeter是基于Java桌面应用程序，所以在使用JMeter之前必须确保本机装有JDK[[2]](#footnote-1)，并配置好JDK的环境变量。

装好JDK1.6以上版本后，在JMeter的bin路径下找到JMeter.bat（适用于Windows）[[3]](#footnote-2)，双击即可启动JMeter[[4]](#footnote-3)，启动后界面如图5-2所示。

## 主要功能模块（组件）

JMeter构成分为以下几个部分：已经添加 的Jmeter的组件可以进行启用和禁用；

### 测试计划（Test Plan）---相当于场景设计（单、多）

测试计划是使用 JMeter 进行测试的起点，它是其它 JMeter 测试元件的容器。



JMeter测试计划的一些参数说明：

（1）名称：你可以为你的测试计划取一个有意义的名字。

（2）注释：对测试计划的注释。

（3）用户定义的变量：用户可以自己定义变量，在用到此变量的时候直接用${ 变量名}引用即可。例如：变量名＝url ，值＝http://www.baidu.com，在需要http://www.baidu.com时直接用${url}即可，这里的变量相当于一个全局变量，作用于整个测试计划。

（4）独立运行每个线程组：勾选以后所有的线程组都是顺序执行的了。一般不勾选，让所有的线程组并发启动。

（5）Run tearDown Thread Group after shutdown of main threads：当结束线程运行后停止主线程。

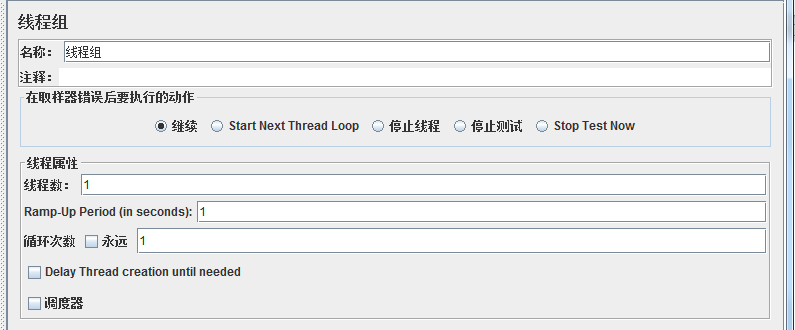
（6）函数测试模式：勾选后会有详细的请求记录，消耗资源，严重影响客户端性能。一般不勾选。

（7）Add directory or jar to classpath：向类路径（即%JMeterHOME%\lib）中添加目录及JAR包。(添加额外的全局jar包)比如，在使用JDBC请求前需要加载的JDBC驱动包，就可以在此添加，测试Java请求时候需要引用的JAR包也可以在此处直接添加引用。

启动JMeter之后，就默认打开了一个测试计划，测试计划用来描述一个性能测试，包含与本次性能测试所有相关所有执行步骤以及相关报表。一个完整的测试计划包含一个或者多个线程组、逻辑控制、取样发生控制、监听器、定时器、断言和配置元件。

### 线程组（threads group）

相当于LoadRunner中的虚拟用户，每个JMeter测试计划的第一步就是添加线程组。在线程组里进行设计要模拟的并发用户量，并发时间或并发次数、线程的启动频率。



JMeter的线程组分为三类：

* setup thread group：相当于Junit的setup，进行测试前的初始化工作，也就是相当于LoadRunner脚本中init部分。
* teardown thread group：相当于Junit的teardown，进行测试结束后的垃圾回收等清理工作，也就是相当于LoadRunner脚本中的end部分。
* thread group（线程组）：我们最常用的就是线程组，相当于LoadRunner里的Action。一个线程组就是相当于一个LoadRunner中的虚拟用户，这些线程在脚本运行后无法变动，不像LoadRunner一样，可以在执行过程中进行人为干预。

下面为JMeter线程组元素的使用说明：

（1）名称：线程组的名称。

（2）注释：对于这个线程组的解释。

（3）在取样器错误后要执行的动作：继续、启动下个线程、停止线程、停止测试。

（4）线程属性：包含以下几个选项。

* 线程数：相当于LoadRunner的并发用户数，代表启动多少个线程对应用进行并发，图中是对应用进行200并发。（如果是多台负载机时，这里的数是每台的线程数）
* Ramp\_up Period(in seconds)：线程间的时间间隔（几秒内初始化完成），图中代表200个线程在1秒钟内启动，类似LoadRunner中Vuser的初始化设置。（初始化、加载时间）
* 循环次数：每个请求的重复次数。如果选择后面的永远（默认），那么请求将一直继续；如果不选择永远，而在输入框中输入数字，那么请求将重复指定的次数。比如输入1，就表示请求将执行一次。但如果输入0，请求就不会被执行。 图5-3中表示200个线程对服务器发送请求，每个线程请求1次，一共对服务器产生200个请求。这比LoadRunner中的场景设计要灵活一些，可以直接按照“多少并发×运行次数”，来控制对服务器发送多次请求。
* Delay Thread creation until needed：延迟创建线程直到需要创建时进行创建。

（5）调度器：包含以下几个选项。

* 启动时间：测试计划什么时候启动，启动延迟会覆盖它；
* 结束时间：测试计划什么时候结束，持续时间会覆盖它；
* 持续时间（秒）：测试计划持续多长时间，会覆盖结束时间；
* 启动延迟（秒）：测试计划延迟多长时间启动，会覆盖启动时间；--优先级高
* 备注：调度器的组合设置，可以控制Jmeeter测试计划什么时间执行、持续执行的时间。

### 测试片段（Test Fragment）

测试片段元素是控制器上的一个种特殊的线程组，它在测试树上与线程组处于一个层级。它与线程组有所不同，因为它不被执行，除非它是一个模块控制器或者是被控制器所引用时才会被执行。在2.11版本里，也新增了保存测试片段的功能。（调用才会执行）

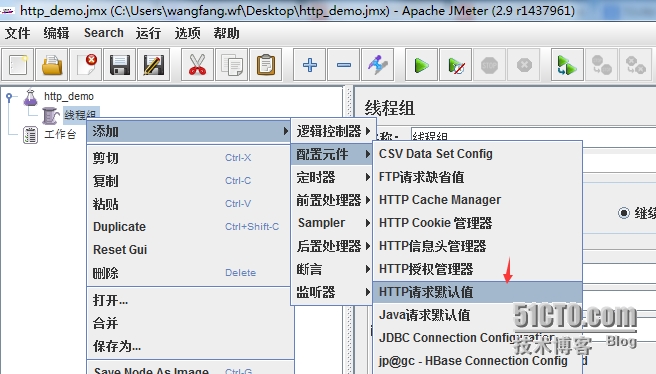
### 取样器（Sampler）：

是性能测试中向服务器发送请求并等待响应的最小单元，每个采样器有不同的属性可以进行设置。目前Jmerter的取样包括如下：

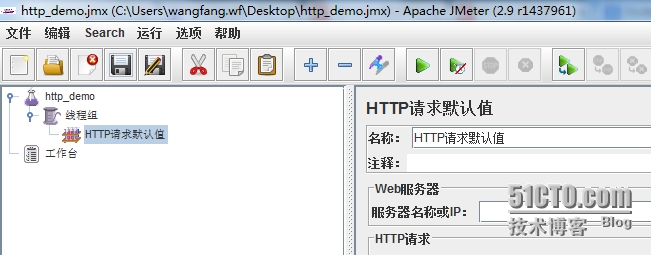
#### HTTP请求

添加http默认请求：（用来配置公共参数,不是http请求）

右键线程组，选择“添加”—〉 “配置元件”—〉“HTTP请求默认值”，点击“HTTP请求默认值”后

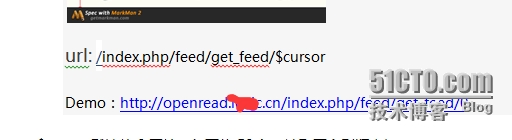
[](http://s3.51cto.com/wyfs02/M01/5A/63/wKiom1T8MfSR4XkXAAKA8e8xa_k858.jpg)

添加成功后，线程组”节点下多了“HTTP请求默认值”节点

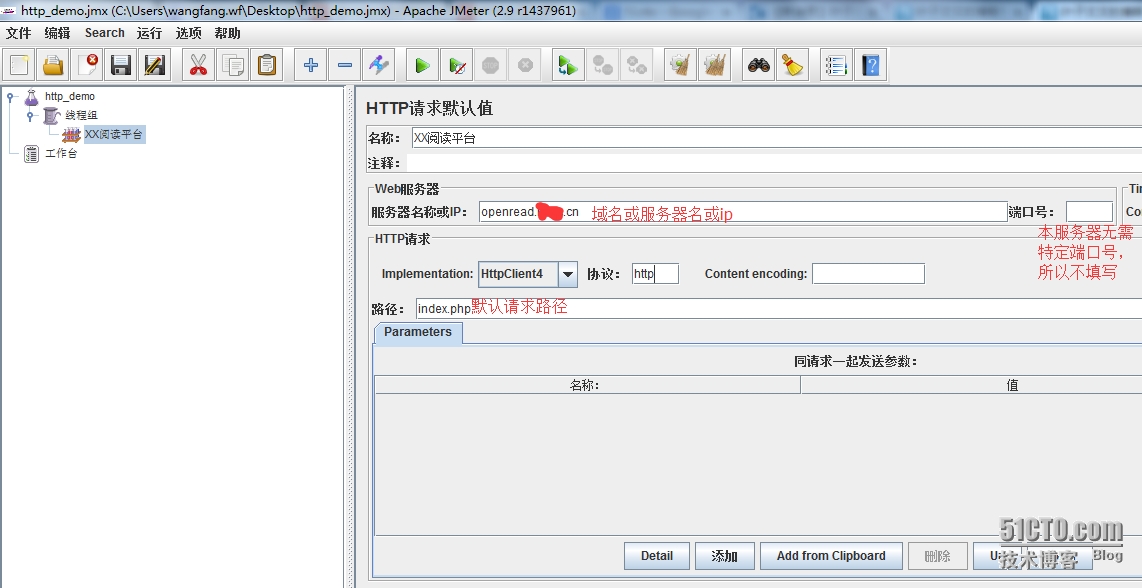
[](http://s3.51cto.com/wyfs02/M02/5A/5F/wKioL1T8M3SAGf_qAAF2WxtwRww553.jpg)

6. 设置“HTTP请求默认值”：

针对本项目填写如下，服务端的系分说明

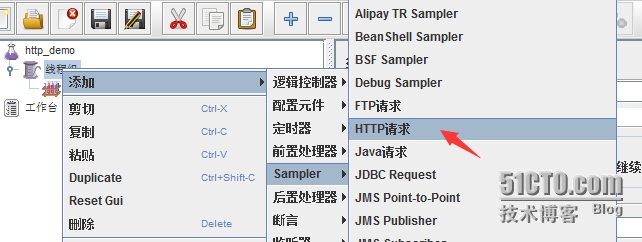
[](http://s3.51cto.com/wyfs02/M00/5A/5F/wKioL1T8NUeBVGExAACbOHV11NA997.jpg)

填写默认请求名、服务器、默认请求路径，保存测试计划

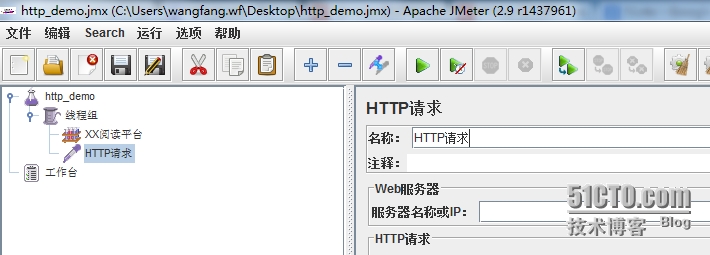
[](http://s3.51cto.com/wyfs02/M01/5A/60/wKioL1T8NfeRNLbaAAN8WuhqJe4459.jpg)

7. 添加http请求

右键“http\_demo”，选择“添加”—〉 “Sampler”—〉“HTTP请求”

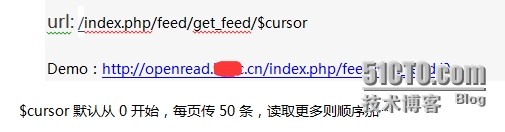
[](http://s3.51cto.com/wyfs02/M02/5A/60/wKioL1T8NuegkhuBAAF7RnjrTvw087.jpg)

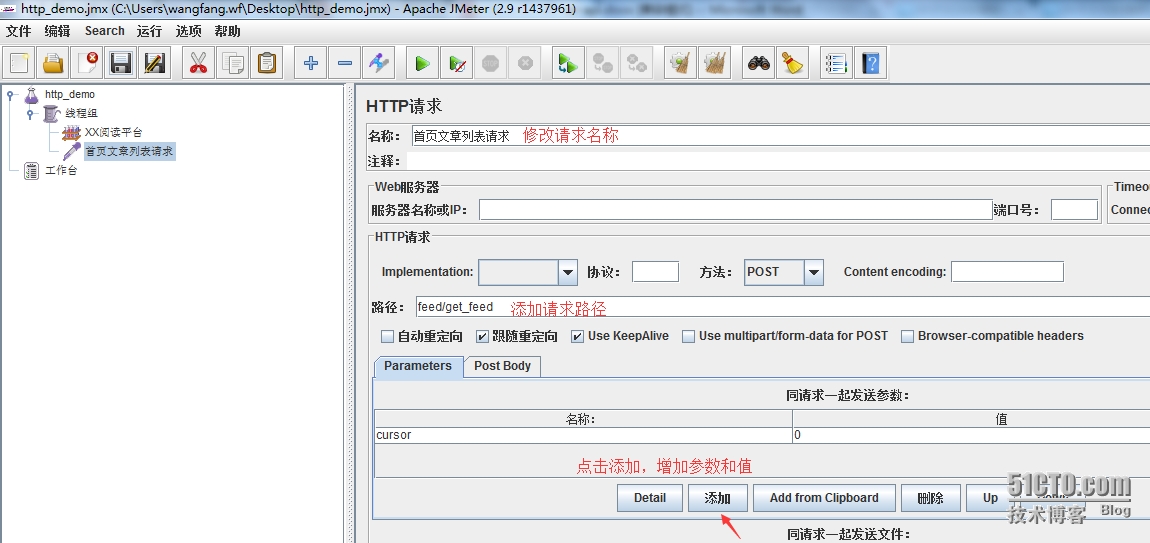
添加成功后，HTTP请求默认值“XX阅读平台”节点下发出现“HTTP请求”

[](http://s3.51cto.com/wyfs02/M02/5A/63/wKiom1T8NjTTDsT0AAGGuRtw63g095.jpg)

8. 设置http请求

填写参考服务端系分：

[](http://s3.51cto.com/wyfs02/M01/5A/60/wKioL1T8OGvQ3Rj3AACxIXhQdPo362.jpg)

[](http://s3.51cto.com/wyfs02/M00/5A/63/wKiom1T8NvuTcT6mAAPhUWwBlOc191.jpg)

填写完成，保存测试计划

* FTP请求
* Java请求
* JDBC请求
* SAOP/XML-RPC请求
* LDAP请求
* Junit请求
* JMS请求
* TCP请求
* STMP请求

.....

### 逻辑控制器（Logic Controller）：少用

逻辑控制器可以自定义JMeter发送请求的行为逻辑，它与Sampler结合使用可以模拟复杂的请求序列。常用的逻辑控制器有if Controller、while Controller、Runtime Controller、事务控制器、随机控制器、交替控制器、吞吐量控制器、模块控制器等，如图5-1所示。



图5-1 JMeter逻辑控制器

### 监听器（Listener）---analysis

对JMeter测试结果进行收集处理并进行可视化展现的一系列元件，包括发送的请求数据、返回的结果数据、响应时间的统计、吞吐量、错误率、返回结果状态等信息。常见的监听器有图形结果、查看结果树、用表格查看结果、聚合报告等。

### 配置元件（Config Element）--维护请求

维护Sampler需要的配置信息，并根据实际的需要修改请求的内容。比如，HTTP、LDAP、Java等默认值，JDBC以及MongoDB配置信息等，最常用的是用CSV Data Set config或者使用随机参数来做参数化。简单起见，建议将配置元件放在请求开始之前。

### 定时器（Timer）(固定、同步定时器)

相当于LoadRunner中的思考时间。设置定时器的目的：一方面是为了更真实的模拟用户并发行为；另一方面是控制每个线程之间的请求间隔时间，以减少服务器压力。

### 断言（Assertions）

相当于LoadRunner中的检查点，通过对返回的响应数据与预期数据进行比对，用来判断请求是否成功，可以确保压力测试是在功能正确的前提下执行的，毕竟压力测试的前提是要保证功能的正确性，这个限制对于有效的测试是非常有用的。常用的断言有响应断言、XML断言、HTM断言、XPath断言等。

**响应断言 ：对服务器的响应进行断言校验**

**（1）应用范围: main sample and sub sample, main sample only , sub-sample only , jmeter variable**

　　关于应用范围，我们大多数勾选“main sample only” 就足够了，因为我们一个请求，实质上只有一个请求。但是当我们发一个请求时，可以触发多个服务器请求，类似于ajax那种，那么就有main sample  和 sub-sample之分了。

　　此外，对于有重定向的请求，并且勾选了“跟随重定向”， 那么这两个请求都是 sub-sample，重定向后的请求（第二个请求）就是main-sample

　　关于main sample and sub sample，有兴趣的可以看看这个帖子

　　http://stackoverflow.com/questions/28214936/jmeter-in-which-scenario-i-can-use-main-sample-or-sub-sample-or-both-for-te

**（2）响应字段： 响应文本，Document(Text)，url样本，响应代码，响应信息，Response Header，ignore status**

　　响应文本: 服务器响应文本，一般普通http响应，都勾选这个。

　　Document(Text)：一切Apache Tika 支持服务器响应，包括文本响应，还支持 PDF, Office, Audio, Video formats。jmeter会用Apache Tika 去解析服务器响应内容，会很耗内存，而且也很容易解析失败。所以一般普通http请求，不要选择这个。

　　url样本：是对sample的url进行断言。如果请求没有重定向（302），那么就是这个就是请求url。 如果有重定向（且跟随重定向），那么url就包含请求url 和 重定向url

　　响应代码：http响应代码，如101,200,302,404,501等。但当我们要验证404,501等http响应代码时，需要勾选“ ignore status”。因为当http 响应代码为400,500时，jmeter默认这个请求时失败的。

　　响应信息：http响应代码对应的响应信息，例如：OK, Found

　　HTTP/1.1 200 Ok

　　HTTP/1.1 302 Found

　　Response Header : 响应头信息，例如

　　Server: Tengine

　　Date: Thu, 12 Mar 2015 09:43:52 GMT

　　Content-Type: text/html

　　Content-Length: 260

　　Connection: close

　　Location: http://www.baidu.com/404.html

**（3）模式匹配规则**

　　包括：返回结果包括你指定的内容，支持正则匹配

　　例如：

　　响应字段为： 响应文本

　　模式匹配规则：匹配

　　断言为：1、invalid    2、[a-z]+

　　当返回值为：{"msg":"channel invalid."} ， 这两个断言都是ok的，返回true

　　匹配：

　　(1) 相当于 equals 。当返回值固定时，可以返回值做断言，效果和equals相同

　　(2) 正则匹配 。 用正则表达式匹配返回结果，但必须全部匹配。 即正则表达式必须能匹配整个返回值，而不是返回值的一部分。

　　例如：

　　响应字段为： 响应文本

　　模式匹配规则：匹配

　　断言为：1、{"msg":"channel invalid."}   2、\{"msg":"[a-z]+ invalid\."\}  ， 3、[a-z]+

　　当返回值为：{"msg":"channel invalid."}   ，断言1 2 是ok的， 断言3是false

　　当返回值为：{"msg":"channel invalid."} ， 断言1 3是false， 断言2才是ok的。

　　原因是，断言1 只能用于equals，而断言3 只匹配了返回值部分，而不是全部匹配。

　　Equals : 返回结果与你指定断言完全一致

　　SubString：与 “包括”差不多，都是指返回结果包括你指定的内容，但是subString不支持正则字符串

　　例如：

　　响应字段为： 响应文本

　　模式匹配规则：匹配

　　断言为：1、invalid    2、[a-z]+

　　当返回值为：{"msg":"channel invalid."} ， 断言1返回true， 但断言2返回false

　　否：就相当于取反。 如果上面断言结果为true，勾选“否”后，最终断言结果为false。如果上面断言结果为false，勾选“否”后，则最终断言结果为 true。

### 前置处理器（Per Processors）--修改请求

常常用来修改请求的设置。例如，HTTP URL重写修复符则可以实现URL重写，当RUL中有sessionID 一类的session信息时，可以通过该处理器填充发出请求的实际的sessionID。

#### 后置处理器（Post Processors）--关联

相当于LoadRunner中的关联。用于对Sampler 发出请求后得到的服务器响应数据进行处理。一般用来提取响应中的特定数据，比如用后置处理器的正则表达式提取器来对服务器返回的特定数据进行关联操作。

# JMeter快速入门之HTTP脚本实战

所有的性能测试工具原理基本是一样的：最里层是发送相应请求的脚本，然后外层的多线程/进程调用我们脚本产生压力对服务器发送压力，一个资源监控器对压力测试过程中请求处理时间进行收集整理分析。（请求—多线程---结果收集）

在前面学习LoadRunner过程中，已经深入理解了Vuser、参数化、关联、检查点、思考时间、场景设计、结果分析，所以在学习JMeter的时候也参考了LoadRunner的一些概念来进行学习，这样大大加快了的理解速度。

## 测试计划

测试计划是使用JMeter进行测试的起点，它是其他JMeter测试元件的容器，每个测试场景/脚本都包含一个测试计划（见图5-2），也可以说每个测试场景/脚本都叫一个测试计划。

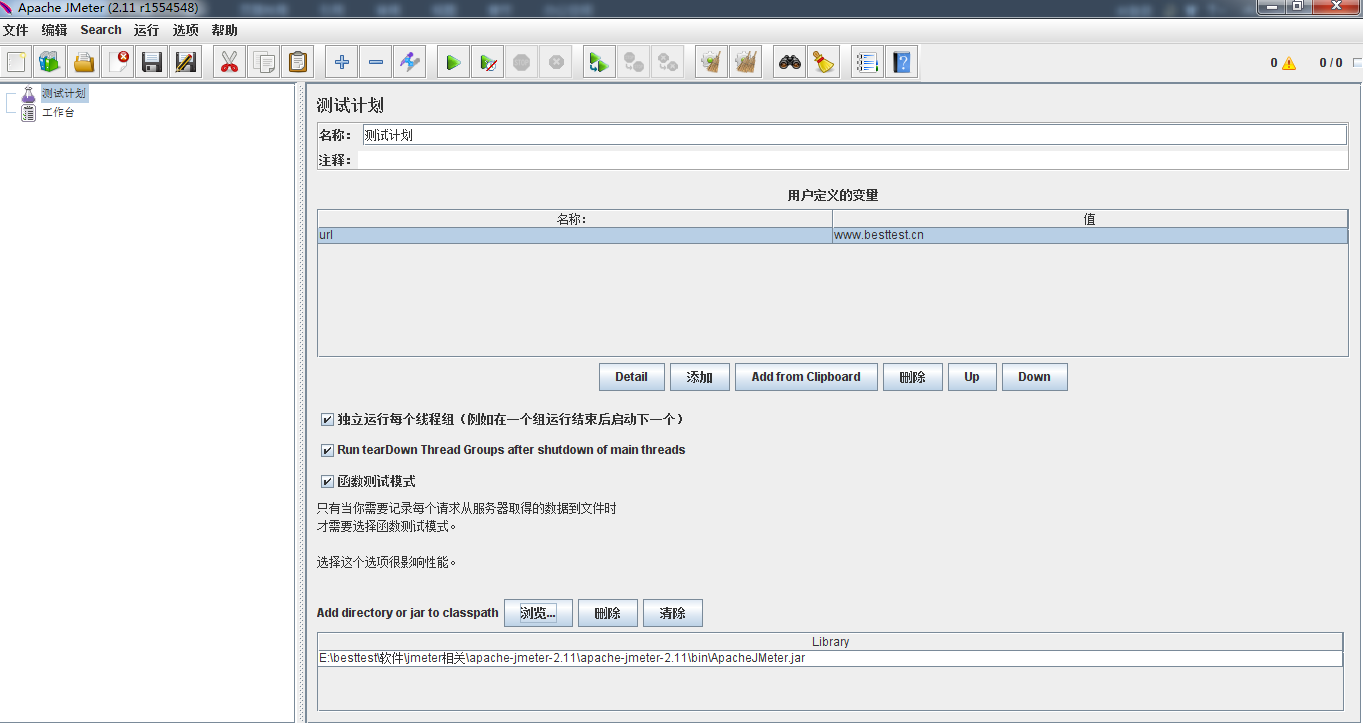


图5-2 JMeter测试计划

## Vuser及场景设计

在测试计划基础上新建一个线程组，线程组的属性如图5-3所示，JMeter的场景设计以及Vuser都是在线程组下进行设计完成的。

线程组里的“线程数”相当于Loadrunner中的Vuser，线程数量就是实际压测过程中的并发量。线程组里的“循环次数”以及“调度器”就相当于Loadrunner中的场景设计模块，它决定了线程组加压的方式以及持续次数或者持续时间。

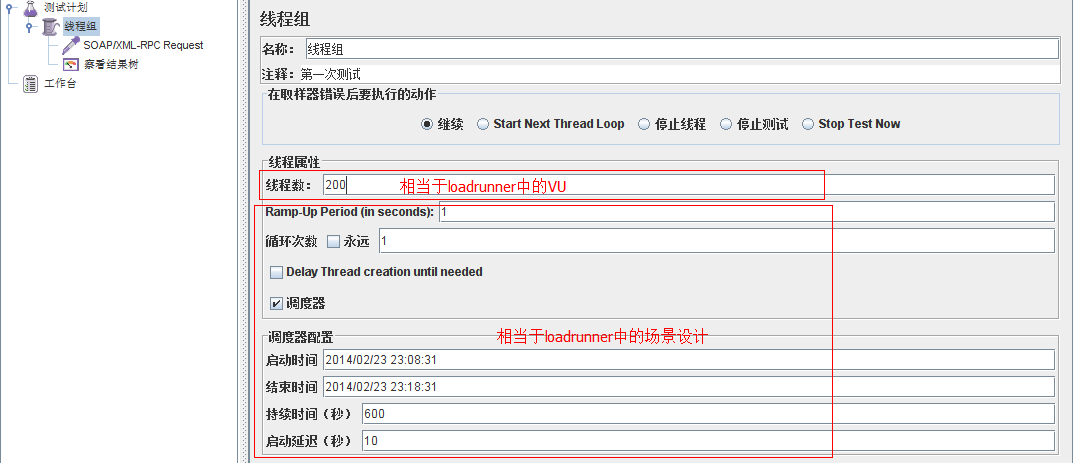


图5-3JMeter线程组属性

## JMeter之脚本录制

JMeter在做HTTP脚本时也是用录制方式进行的，目前JMeter有两种脚本录制方式：利用BadBoy进行录制和使用Jmeter代理进行录制，下面跟一起进行学习。

### 利用BadBoy进行脚本录制

在正式开始讲解之前，我们先来做一些准备工作：

（1）下载：通过Badboy的官方网站（http://www.badboy.com.au）下载Badboy的最新版本；

（2）安装Badboy。安装完成后，在桌面和Windows开始菜单中能看到相应的快捷方式，或Badboy安装目录下的Badboy.exe 文件，直接双击启动Badboy；

（3）启动Badboy。可以看到下面的界面，如图5-4所示。界面：：：

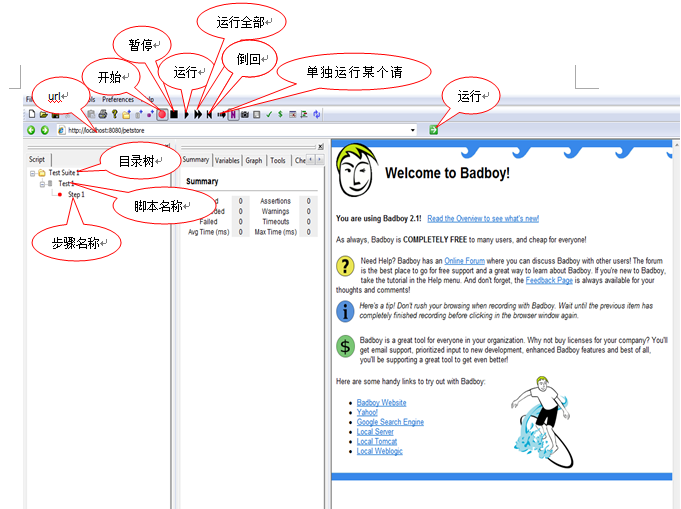


图5-4 Badboy界面

准备工作完成，我们开始正式录制脚本。在地址栏中输入需要录制Web应用的URL，以http://www.baidu.com 为例 。

（1**）点击开始录制** 按钮（地址栏后的绿色箭头）开始录制。

**（2）开始录制后，**可以直接在Badboy内嵌的浏览器（主界面的右侧）中对被测应用进行操作，所有的操作都会被记录在主界面左侧的编辑窗口中。

在baidu的搜索引擎中输入“goodgoodstudy”进行搜索。录制下来的脚本并不是一行行的代码，而是一个个Web对象，有点像LoadRunner的VuGen中的Tree View视图。

（3）**录制完成后，**点击工具栏中的“停止录 制”按钮，完成脚本的录制。

**（4）保存脚本**。选择File→Export to JMeter菜单，填写文件名“goodgoodstudy.jmx”，将录制好脚本导出为JMeter脚本格式。也可以选择File→Save菜单保存为Badboy脚本。

（5）JMeter打开Badboy录制的脚本。启动JMeter，打开刚利用Badboy录制生成的测试脚本（goodgoodstudy.jmx），这样就可以用JMeter进行测试了，如图5-5所示。

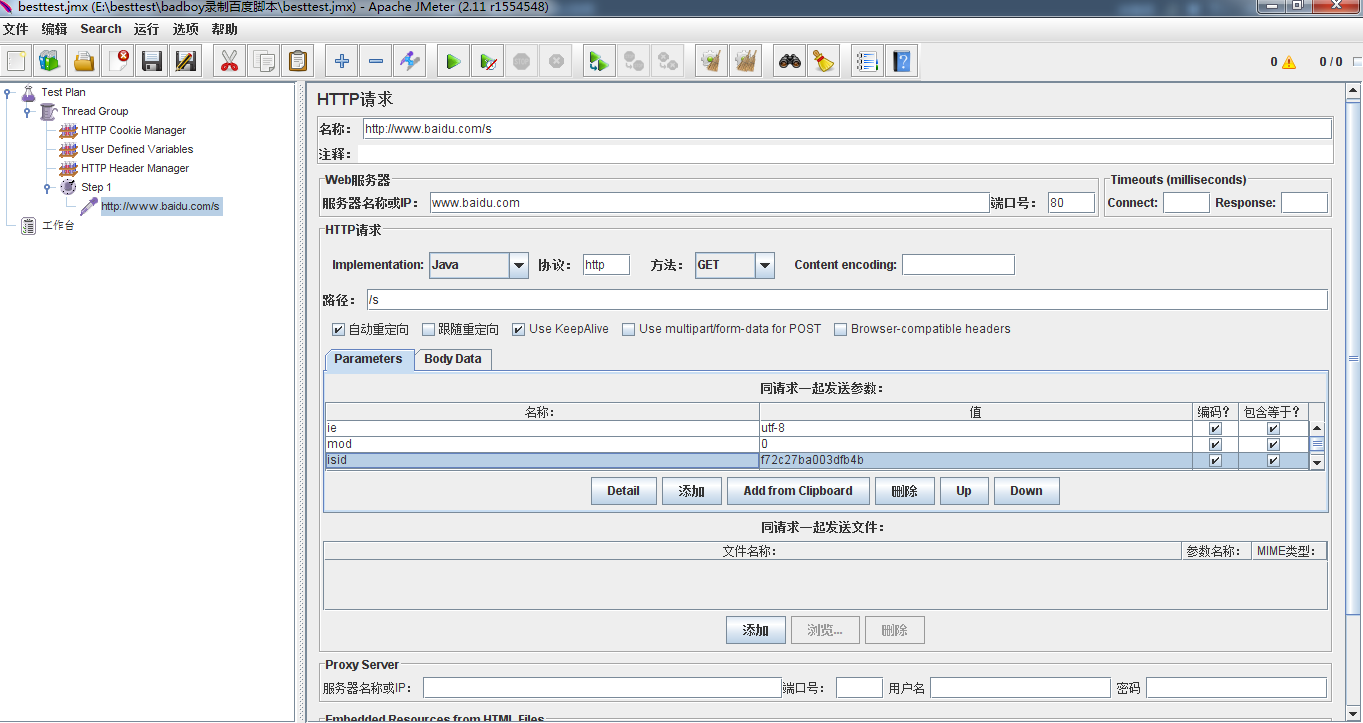


图5-5 JMeter打开利用Badboy录制生成的http脚本

### 利用JMeter代理服务器功能进行录制

（1）**添加录制组件:**在测试计划中添加线程组，在线程组中添加“逻辑控制器”→“录制控制器”。

（2）**设置代理：**在工作台中添加非测试元件→“HTTP代理服务器”，如图5-6所示。

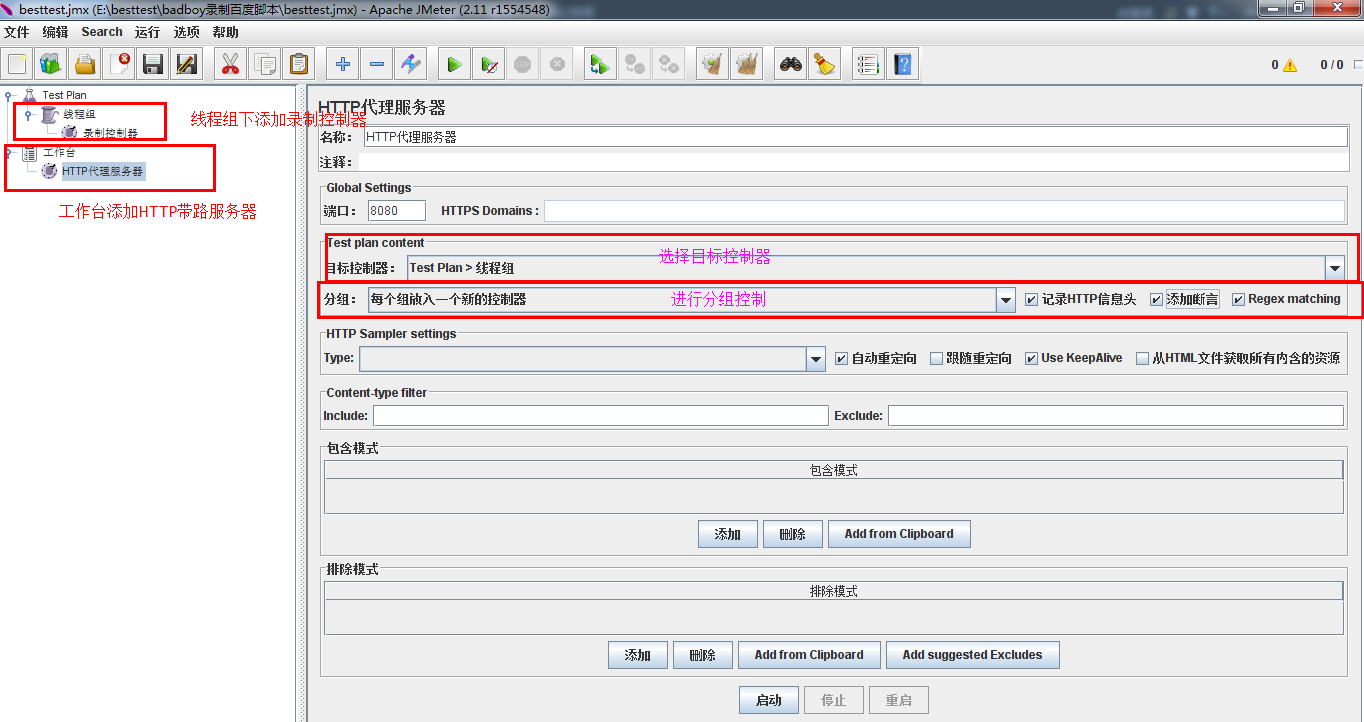


图5-6 JMeter使用代理功能录制HTTP脚本

下面对Jmeter的代理服务器功能界面进行简单介绍。

* 端口：即代理服务器的监听端口，默认为8080，可以自行修改，但要保证不与其他端口冲突；
* **目标控制器选择：录制脚本存放的位置**，选择的是测试计划→线程组，意思就是将脚本保存在测试计划的线程组下；
* **分组：**对请求进行分组，选择的是每个组放入一个新的控制器；
* **记录HTTP信息头：**自动帮我们录制request的head信息，就不需要我们额外单独配置；
* **添加断言：**录制时自动帮我们加入空的检查点，录制完成后需要我们手动添加检查内容；
* **Regex matchi**ng：录制时加入空的正则匹配进行关联，录制完成需要我们手动添加正则表达式，等于是告诉我们哪里需要关联；
* **Type：**选择我们服务对于的HTTP Client的版本；
* **自动重定向（自动跳转）：如果选中该选项，当发送HTTP请求后得到的响应是302/301时，JMeter 自动重定向到新的页面**，比如HTTPS请求跳转，该选项适用于当方法为“HEAD”或者“GET”的时候。自动重定向跳转时候使用的是相对路径，一般用于用户注销登录时候返回主页面或者跳转到其他网站这种情况；还有一种叫做跟随重定向，跟随重定向使用的是相对路径，一般用于用户登录的时候，根据角色转发响应的模块；
* **Use KeepAlive：**当该选项被选中时，JMeter和目标服务器之间使用 Keep-Alive方式进行HTTP通信，也就是在一个请求超时以前，JMeter不会向服务器建立一个新的请求链接，默认是选中状态；
* **从HTML文件获取所以包含的资源：**当该选项被选中时，JMeter在发出HTTP请求并获得返回的HTML文件内容后，会获取HTML中包含的所有资源（图片、flash等），默认不选中；
* **包含模式：与排除模式一样，用来控制排除一些没必要的录制元素；**
* **排除模式：**录制时候许多JS、CSS、PNG、GIF的路径都录制了进来，而这些对于测试脚本是没有用处的，我们可以在HTTP代理服务器的排除模式设置，脚本不记录这些内容。

**（3）添加定时器。**右键”HTTP代理服务器“添加"定时器”→“高斯随机定时器”。

这样设置的目的是，告知JMeter在其生成的HTTP请求中自动增加一个定时器。定时器将会使相应的取样器延迟。延时的规则是：在上一个访问请求被响应并延时了指定的时间后，下一个被定时器影响的取样访问请求才会被发送出去。如果在代理服务器元件里使用了高斯随机定时器，就应该在其中的固定延迟偏移（Constant Delay Offset）设置项里添上${T}（用于自动引用纪录的延迟时间）。

（4**）IE设置。**代理服务器配置好以后，点击启动按钮，代理服务器就会开始记录所接受的HTTP 访问请求。 打开浏览器，打开“Internet选项”→“连接”，将局域网（LAN)设置中的代理服务器设为： localhost，端口为在代理服务器中设的端口：8080。注意，录制完成后，千万要恢复浏览器的代理服务器设置。否则影响正常上网。

（5）开始录制。打开一个IE窗口，输入测试的网址(比如www.goodgoodstudy.cn)，按回车键，测试计划中将录制HTTP的请求。在测试网页上进行操作，测试计划中继续录制HTTP请求。如果是有session判断登录的应用，还需要增加cookie manager配置元素，选中“线程组”，右击后选择“添加”→“配置元素”→“HTTP Cookie管理器”命令。

提示1）IE中局域网的端口号和JMeter中的Proxy Server的端口号要一致,，否则会出问题。2）尽量避免跟其他服务的端口冲突，8080端口比较常用，比如我们本机如果提供了一个用tomcat启动的Web服务，那么就可会导致端口冲突，从而无法录制，因为一本tomcat默认端口是8080，所以尽量换一些不常见的端口来使用。3）如果在进行代理录制时候无法录制，可以尝试解决方法在user.properties文件中配置proxy.cert.alias=anything。

（6）录制完成后， 停止HTTP 代理服务器；在录制控制器元件上单击右键将记录的元件保存为一个文件，用于以后重用，最终录制生成的脚本如图5-7所示。

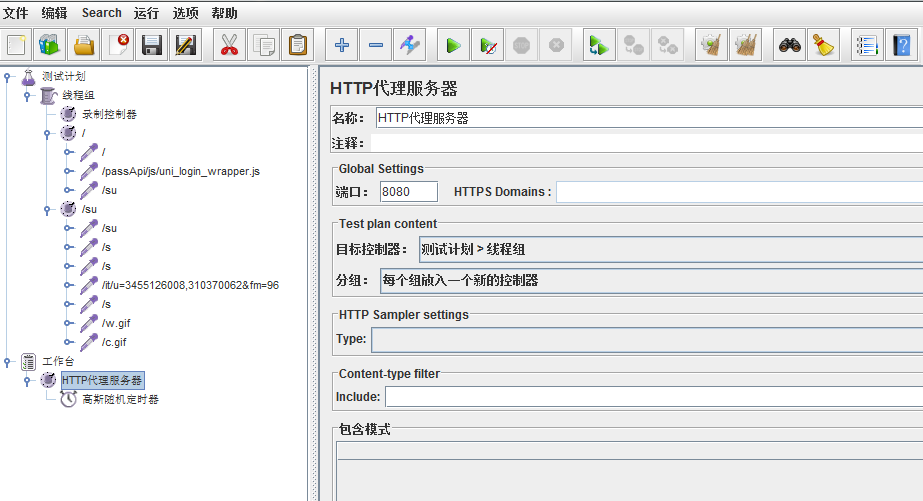


图5-7 JMeter代理服务器录制生成的脚本

### 两种录制方式的区别

（1）利用Badboy录制比较简单，不需要利用代理，录制也与LoadRunner类似，录制的脚本的层次更清晰。经过实践认为使用Badboy进行HTTP脚本录制更加方便和直观。

（2）JMeter代理录制较为复杂，但是能录制到URL相关的静态文件（CSS、JPG等）以及dwr请求，但是Badboy不能。

（3）简单的HTTP协议直接用Badboy录制，不需要我们用JMeter录制后去掉无用的CSS等信息，但是比较复杂的应用，比如dwr请求的，还是得用JMeter代理进行录制。

## JMeter参数化

JMeter也跟LoadRunner一样，有所谓的参数化，在这里学习了JMeter常见的3种参数化方法。

### 使用用户自定义变量-- Test Plan(测试计划)

（1）在Test Plan(测试计划)中点击“用户定义的变量”下的“添加”按钮，开始进行自定义变量的添加。(变量的值要可以是定值也可以是数据文件)

（2）在“名称”里定义变量的名称，比如这里变量名称是“key”；“值”里定义变量所对应的值，比如这里key这个变量对应的值是“好好学习”。

完成自定义变量的添加后，到需要进行参数化的位置进行参数化，比如之前用Badboy录制的百度搜索关键字是“goodgoodstudy”，如图5-8所示。

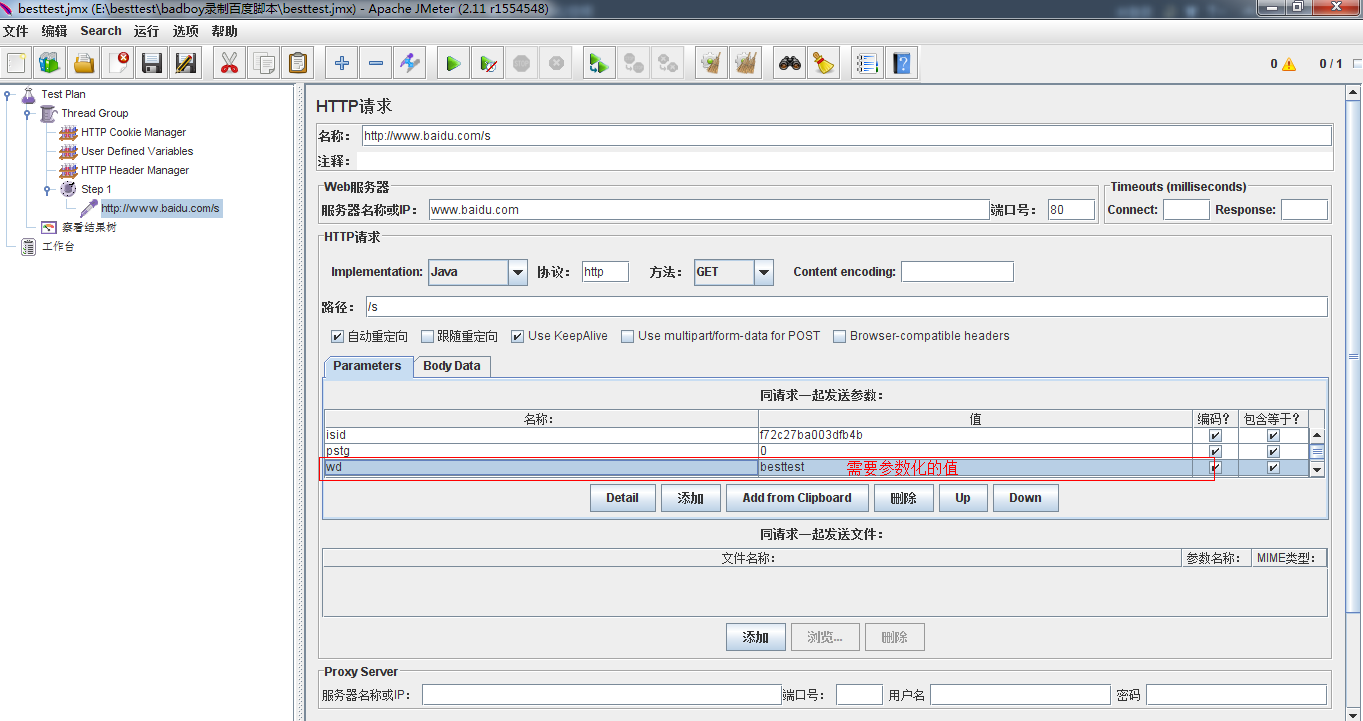


图5-8 需要参数化的地方

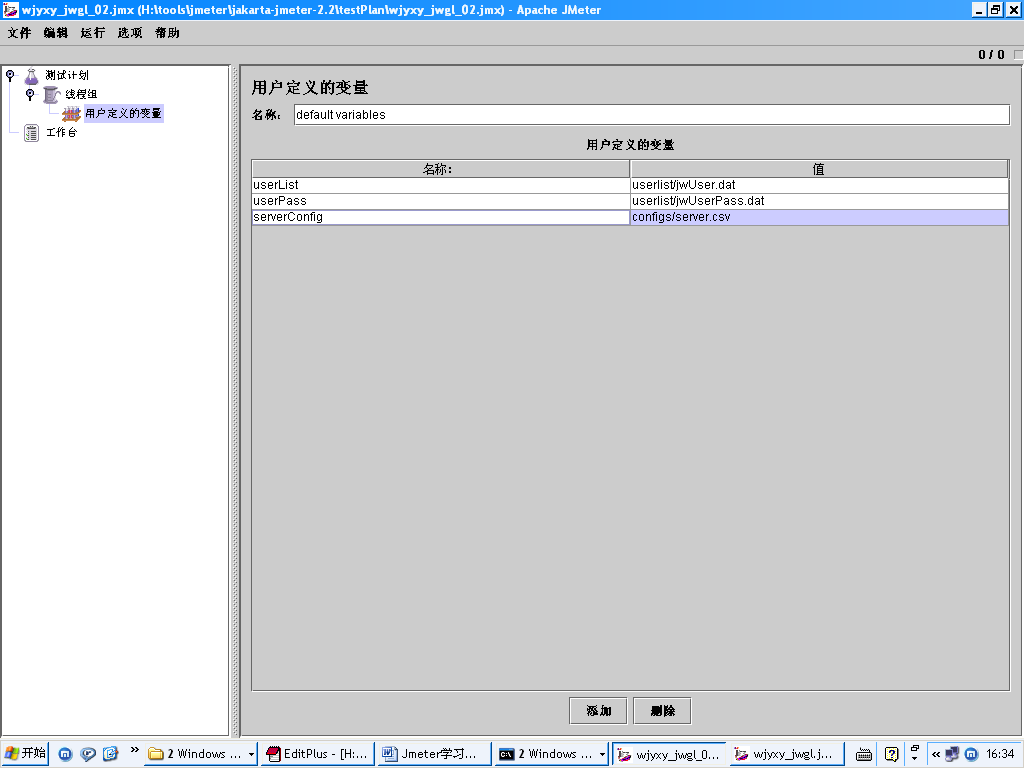
找到需要进行参数化的地方后，将值里面的内容用${key}进行替换（JMeter里的变量都是用${}进行替换），这里的“key”就是刚才在用户自定义变量里设定的变量名，进行参数化替换后，说明在发送请求时候，利用百度进行搜索的关键字就不是之前用Badboy录制时候的“goodgoodstudy”了，而是参数化变量对应的值“其他其他”。

### 用户定义的变量”测试组件，名称：default variables，变量名与值分别如下(可以用绝对路径)

userList userlist/jwUser.dat

userPass  userlist/jwUserPass.dat

serverConfig configs/server.csv



在jakarta-jmeter-2.2\bin目录下新建userlist文件夹，然后在userlist内新建jwUser.dat与jwUserPass.dat文件，然后在这两个文件中分别写入登录系统的用户名和密码，注意要每行一个且一一对应，如：我的jwUser.dat文件内容为

jwadmin

1001300

jwUserPass.dat文件内容为

123

321

### 使用配置元件中的CSV Data Set Config进行参数化设置

在线程组上点击右键选择添加→配置元件→CSV Data Set Config，打开CSV Data Set Config设置界面，如图5-11所示。

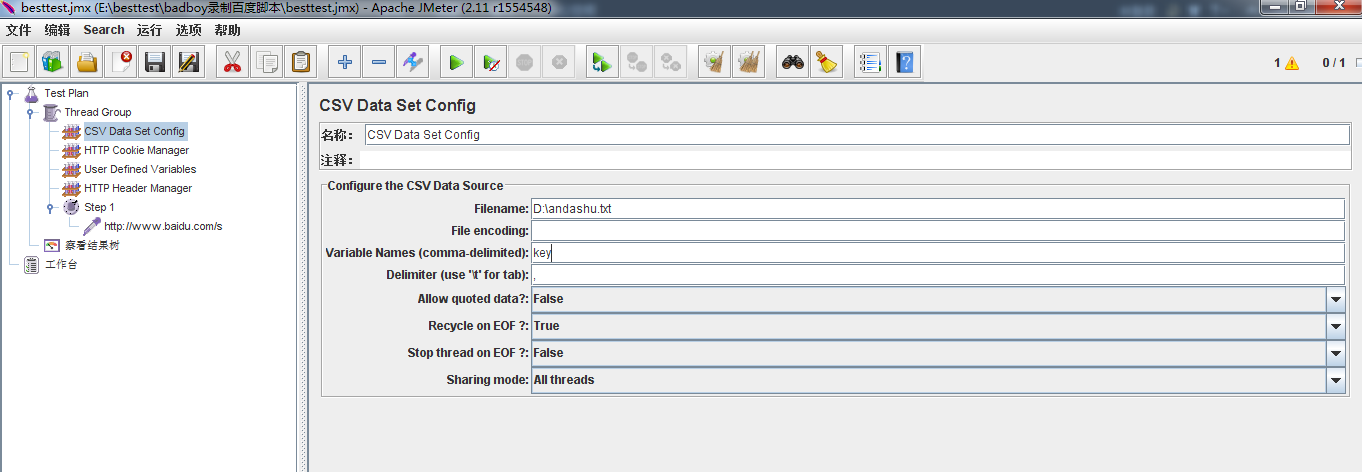


图5-11 CSV Data Set Config参数化设置

在该界面中可以设置以下参数：

（1）**Filename：文件名**，指参数化的文件目录，可以相对或者绝对路径。这里是绝对路径d:\study.txt。如果使用相对路径（如study.txt）则需要将参数化的文件放在当前测试计划所在的路径下；

（2）**File encoding：**文件编码，可以不填，默认的编码方式是ANSI，如果文件文件是用其他编码方式保存的，就需要使用文本文件对应的编码方式，比如GBK或者UTF-8；

（3）**Variable Names(comma-delimited)：**参数名称，这里是“key”，如果有有多个参数，在这里面就写多个参数的名称，每个名称中间用分隔符分割，分隔符在下面的eDelimitet”中定义，比如：key1，key2；

（4）**Delimiter(use ‘\t’ for tab)：**参数文件中多个变量值列用什么进行分隔，\t表示用tab键分割，默认是用逗号进行分隔；

（5）Allow quoted data?(False)：是否允许引用数据，默认为False，表示在引用参数化文件中的值的时候，文件中是什么值就传递什么值到参数中，比如参数化文件有一值是“好好学习”，则引用的时候用的是“好好学习”，如果为True，则引用的时候用到的值就是好好学习，也就是当为True时，将会把参数化文件中双引号里的内容当做响应的参数化取值；

（6） **Recycle on EOF?(true)：结束后是否循环？**选择True或者False，默认是True。CSV Data Set Config是一次读入一行数据，数据分割后存入变量中交给一个线程，当设置的线程数超过参数化文件里的参数化数量时，可以选择数据不够用后是否从头循环开始重新读入数据，为True表示循环读入数据，为False则表示不循环读入数据；

（7**）Stop thread on EOF(true)：到了参数化文件尾处**，是否停止线程，选择：True或者False。

* 当Recycle on EOF 选择True时，Stop thread on EOF选择True或者False没有任何意义。因为之前设置了结束后循环读入数据，这里再来让参数化文件不够后Stop或 Run没有任何意义；
* 当Recycle on EOF 选择flase时：
* Stop thread on EOF选择True，如果设置线程数目为10个， 而参数化文件里只5行数据，那么只会请求5次；
* Stop thread on EOF选择flase，如果设置线程数目为10个， 而参数化文件里只5行数据，那么会请求10次；但是因为第6次取不到参数化数据，所以从第6次之后的请求可能会请求错误；

（8）Sharing mode：共享模式，分为All threads、Current thread group、Current thread，这个地方和[LoadRunner](javascript:;)中的参数化里迭代取值相反，经过试验得出来的结果是：

* **All threads：**选择该项，Test plan中所有线程共享CSV文件中的数据，所有线程按照顺选方式取CSV文件中的不同记录。[测试](javascript:;)计划中所有线程，假如设置了10个线程，线程1取了一次值后，线程2取值时，取到的是csv文件中的下一行，即与线程1取的不是同一行。
* **Current thread group：**选择该项，Test plan中所有线程共享CSV文件中的记录。当前线程组，假设有线程组A、线程组B，A组内有线程A1 到线程An，线程组B内有线程B1到线程Bn。取值的情况是：线程A1取到了第1行，线程A2取第2行，线程B1取第1行，线程B2取第2行。
* **Current thread：**选择该项，每个线程各自独立地使用CSV文件中的记录。假设设置了10个线程，则线程1取了第1行，线程2也取第1行。

注意：线程循环时，去取csv值时，也算入迭代。例如，当设置为Current thread时， 线程1第1次取了第1行，第2次取的就是第2行。

这些设置完成后，在需要进行参数化的地方，将“Variable Names(comma-delimited)”里设置的参数化名进行替换即可，比如将之前用Badboy录制的参数“goodgoodstudy”替换成**${key}**。

### 使用函数助手里的随机函数进行参数化

JMeter中有一个函数助手的功能，里面内置了多个函数，我们可以利用其中的\_Random函数、\_threadNum函数或者CSV相关函数来进行参数化设置。

#### 方法一

（1）在JMeter中选择“选项”→“函数助手对话框”打开函数助手对话框，如图5-9所示。

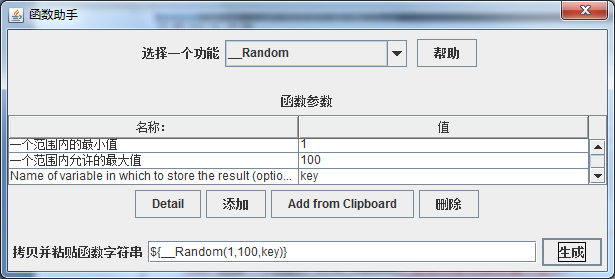


图5-9 JMeter函数助手对话框

（2）选择一个函数，比如\_Random，如图中所示。

1）设定最小值为1。

2）设定最大值为100。

3）函数名称设为key。

4）点击生成将生成一个引用字符串${\_\_Random(1,100,kye)}，在需要进行参数化的地方替换我们生成的随机函数函数就行。

#### 方法二

（1）**选择函数助手：**与方法一类似，打开“函数助手”对话框，选择函数助手的\_CSVRead函数，参见图5-10的第一步。

（2）**在本地生成一个参数化文件**，这里在D盘下面生成了一个“study.txt”文件，JMeter的参数化文件跟LoadRunner的参数化文件有一个显著的区别就是JMeter参数化文件里没有文件列名，比如这里用的参数化文件“study.txt”中内容如下：

Goodgoodstudy，12313213

性能测试博客，123123213

Goodgoodstudy，123123123

这个参数化文件中第一行“Goodgoodstudy”就是参数化的取值了，而不像LoadRunner第一行是参数化文件的列名。

（3）**输入参数化文件的路径：**文件生成后，在如图5-10所示的第二步中输入参数化文件的路径，比如D:\study.txt：

其中：

　　CSV file to get values from | \*alias：要读取的文件路径，应该是绝对路径

　　CSV文件列号| next| \*alias：从第几列开始读取，注意第一列是0

　　点击生成按钮，则生成了函数，在上图中则为：${\_\_CSVRead(D:\login.txt,1)}csvread就是从login.txt文件中读取第二列的参数。以此类推。



图5-10 函数助手之\_CSVRead

##### 注意

1. CSV文件列号项中，默认值为1，实际函数在取值是应该是0对应的第一列。（所以我们写1，哪0行就可以定义列名）
2. 多个参数同时取值时，如（username,password），参数文档中用逗号分隔，每一个参数取参数化文件中对应的列。利用该功能，可以实现不同参数需要不同参数值的问题，只需要在需要参数化的地方将生成的CSVRead函数字符串粘贴过去，然后修改表示第几列的数字就可以了。
3. 这里函数助手最终生成的函数为${\_\_CSVRead(D:\study.txt,0)}，后面需要进行参数化地方的数据用${\_\_CSVRead(D:\study.txt,0)}进行替换即可。
4. jmeter执行的时候，如果有多个线程，顺序读取第一行的数字，如果线程组多余文件中的行数，则循环读取。

#### 函数助手常用函数 ${\_\_functionName(var1,var2,var3)}

其中，\_\_functionName匹配被调用的函数名称。用圆括号包含函数的形参，例如${\_\_time(YMD)}，不同函数要求的参数也不同。有些JMeter函数不要求参数，则可以不使用圆括号，例如${\_\_threadNum}。

如果一个函数的参数中包含逗号，那么必须对逗号进行转义（使用"\"），否则JMeter会把逗号当成参数分隔符。例如：${\_\_time(EEE\, d MMM yyyy)}

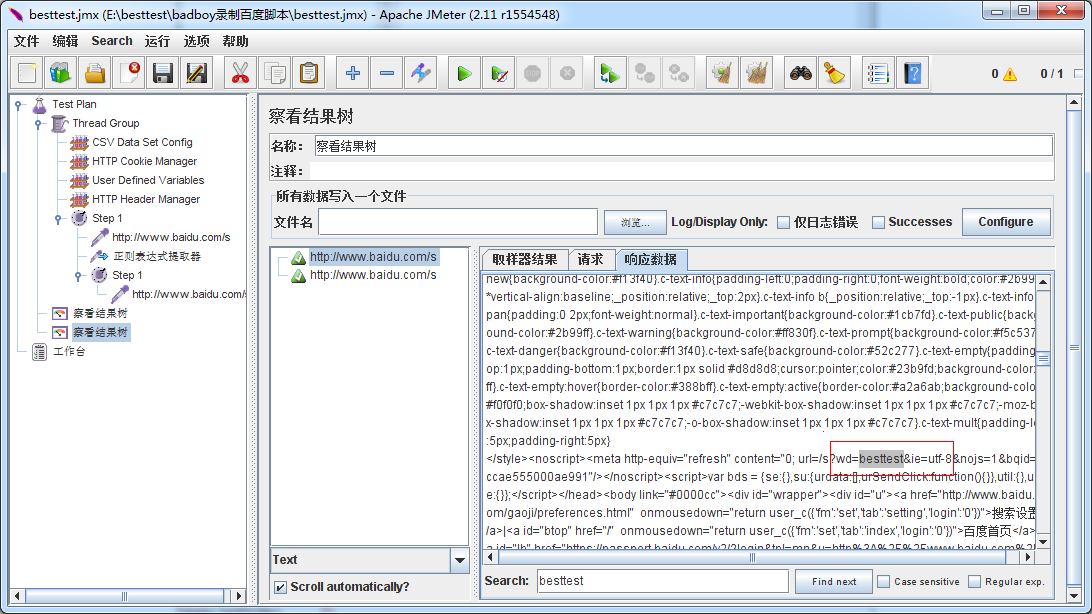
##### 表11-1  JMeter内置函数列表

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 函数类型 | 函数名称 | 注释 |
| Information | [threadNum](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__threadNum) | get thread number |
| Information | [machineName](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__machineName) | get the local machine name |
| Information | [time](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__time) | return current time in various formats |
| Information | [log](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__log) | log (or display) a message (and return the value) |
| Information | [logn](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__logn) | log (or display) a message (empty return value) |
| Input | [StringFromFile](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__StringFromFile) | read a line from a file |
| Input | [FileToString](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__FileToString) | read an entire file |
| Input | [CSVRead](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__CSVRead) | read from CSV delimited file |
| Input | [XPath](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__XPath) | Use an XPath expression to read from a file |
| Calculation | [counter](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__counter) | generate an incrementing number |
| Calculation | [intSum](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__intSum) | add int numbers |
| Calculation | [longSum](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__longSum) | add long numbers |
| Calculation | [Random](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__Random) | generate a random number |
| Scripting | [BeanShell](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__BeanShell) | run a BeanShell script |
| Scripting | [javaScript](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__javaScript) | process JavaScript (Mozilla Rhino) |
| Scripting | [jexl](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__jexl) | evaluate a Commons Jexl expression |
| Properties | [property](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__property) | read a property |
| Properties | [P](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__P) | read a property (shorthand method) |
| Properties | [setProperty](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__setProperty) | set a JMeter property |
| Variables | [split](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__split) | Split a string into variables |
| Variables | [V](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__V) | evaluate a variable name |
| Variables | [eval](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__eval) | evaluate a variable expression |
| Variables | [evalVar](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__evalVar) | evaluate an expression stored in a variable |
| String | [regexFunction](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__regexFunction) | parse previous response using a regular expression |
| String | [char](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__char) | generate Unicode char values from a list of numbers |
| String | [unescape](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__unescape) | Process strings containing Java escapes (e.g. \n & \t) |
| String | [unescapeHtml](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__unescapeHtml) | Decode HTML-encoded strings |
| String | [escapeHtml](http://jakarta.apache.org/jmeter/usermanual/functions.html" \l "__escapeHtml) | Encode strings using HTML encoding |

## JMeter关联

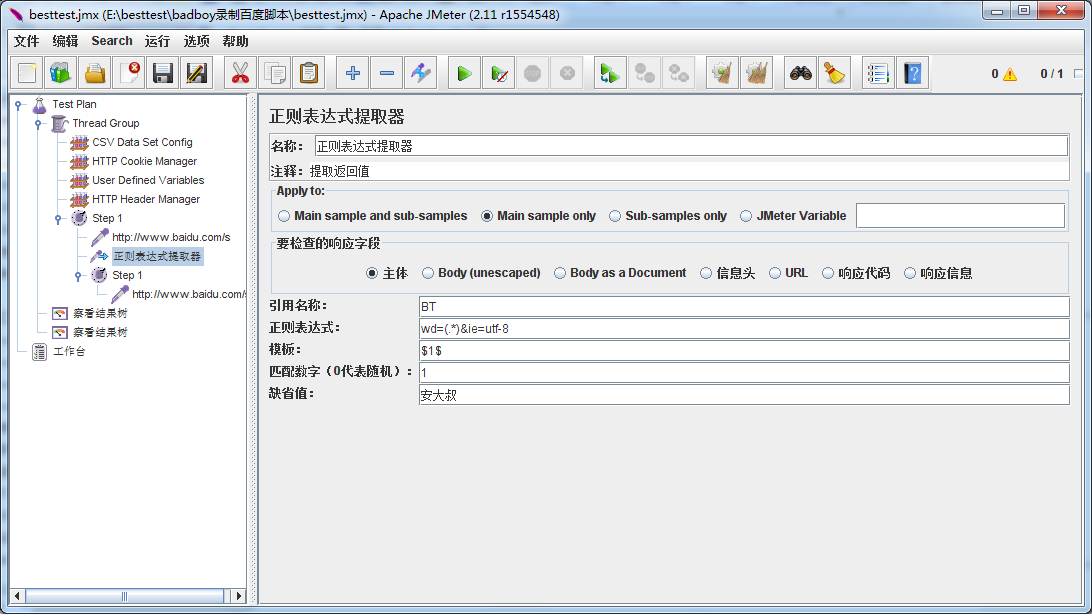
JMeter也有类似于LoadRunner的关联，不过JMeter的关联是用基于正则表达式和XPath Extracto等实现的。

现在想把之前测试结果值中返回的“besttest”（见图5-12）做为关联值获取以便后面需要用到的地方进行使用，比如这里需要将“wd= besttest&ie=utf-8”中返回的“besttest”进行关联。



5-12 JMeter返回值进行关联

首先在需要获得数据的请求上右击添加一个后置处理器→→正则表达式提取器，如图5-13所示：



5-13 JMeter 正则表达式

正则表达式相关界面解释如下：

* **名称：**取一个有代表性的名字即可；--随意
* **注释：**对本次正则表达式的注释，注释自己能理解的意思即可；--随意
* **Apply to：**默认即可；对哪个生效
* **要检查的响应字段(默认主体)：**跟LoadRunner中tree视图有点类似，表示在那哪个地方进行正则表达式的提取，一般选择主体即可；
* **应用名称：**其他地方引用提取值的变量名称，这里如填写的是：BT，那么其他地方具体的引用方式是${BT}；**--关联名**
* 正则表达式（正则表达式中()括起来的部分就是要提取的）：提取的正则表达式主体，这里是wd=(.\*)&ie=utf-8；(.\*)表示要提取的内容，wd=相当于LoadRunner中的左边界，&ie=utf-8相当于LoadRunner中的右边界；,如果碰到返回结果值是多个相同的导致结果这时候用正则中的”?”代替“.\*”,代表找到第一个匹配项后停止,比如parent.location='/bugfree/case/(.+?)';

提取的内容里的内容有：

* name="file" value="(.+?)"> name="file" value="([^"]+)">

上面用到的特殊字符包括如下几个。

1. ( 和 )：封装了待返回的匹配字符串。
2. .：匹配任何字符。
3. +：一次或多次。
4. ?：不要太贪婪，在找到第一个匹配项（左边界）后停止。（如果没有?，在找到第一个">后（左边界），会继续寻找，直到最后一个">，这么做很可能不是测试人员期望的。）
5. \*
6. [^"] 其中[^"]-意味着匹配任何东西（除了"）。在这种情况下，匹配引擎在找到第一个右侧"后，就会停止搜索（只要找到右侧就停止，不用找左侧）。而上面例子中的匹配引擎会去寻找">。

* **提取多个字符串**

假设测试人员期望匹配Web页面的如下部分：name="file" value="readme.txt">，并提取file.name和readme.txt。

一个符合要求的正则表达式：

1. name="([^"]+)" value="([^"]+)"

这会创建两个组合，并可用于JMeter正则表达式模板，形如$1$ 和$2$。

JMeter正则表达式提取器会将组合的值放在指定变量中。

* **关键字**

正则表达式使用特定字符作为关键字，这些字符对正则表达式引擎有特殊意义。在字符串中使用这些字符必须进行转义（使用反斜杠"\"），目的是将它们当成原始字符，而非正则表达式的关键字。下面是关键字和它们的含义。

( )：组合。

[ ]：字符集合。

{ }：重复。

+ ?：重复。

.：任意匹配字符。

\：转义字符。

| -：选择符。

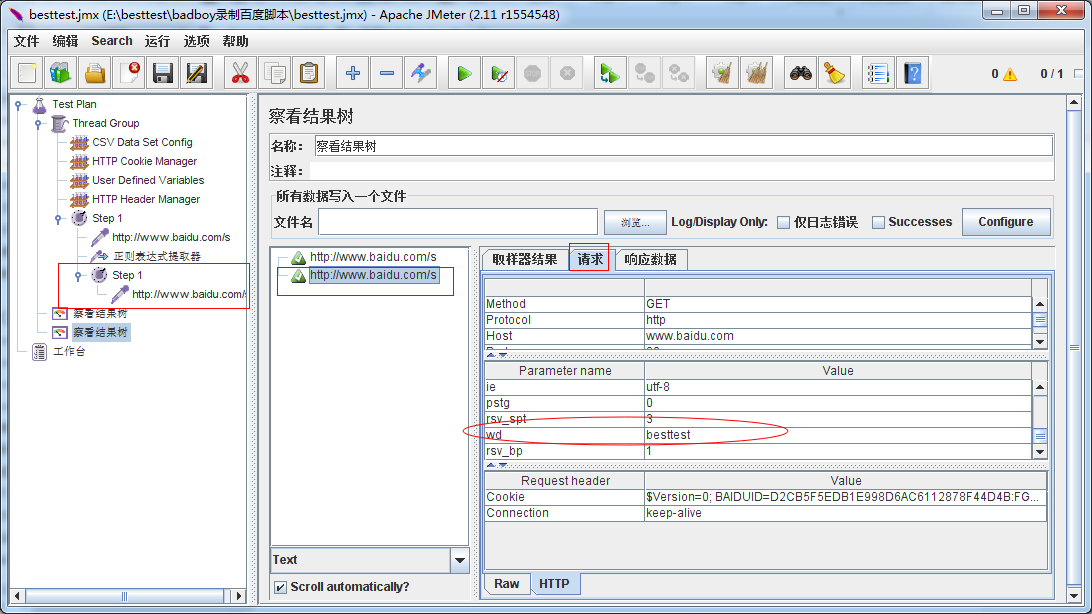
^ $：字符串或行的起始和结尾。

注意，ORO不支持\Q和\E关键字。

* **模板：**模板是使用提取到的第几个值。因为可能有多个值匹配，所以要使用模板。从 1 开始匹配，依次类推。这里只有一个，所以填写 $1$ 即可；用$$引用起来，如果在正则表达式中有多个提取表达式，则可以是$1$，$2$等等，表示解析到的第几个值给提取到的变量，正则表达式的提取模式，值从1开始，值0对应的是整个匹配的表达式；
* **匹配数据（0代表随机）：**0代表随机，-1代表所有，其余正整数表示对应第几个匹配内容，比如4则代表匹配第4个结果值。如果在LR中，取出的值是一个数组，还得处理一下，LR11版本用一个随机的函数就可以不用写大段的代码来处理数组；---提取值（**相当于LR的"Ordinal="/ORD=ALL**）
* **默认值（缺省值）：**如果返回结果中没有匹配值的时候，变量的值采用默认值，比如图5-12中的是“好好学习”，表示在正则表达式提取不到匹配的值的时候，其他地方引用${BT}的值就是“好好学习”；

正则表达式写好之后，把百度请求复制一份，并将复制的请求中“besttest”的值用正则表达式中设置的变量名${BT}进行替换。

完成之后，运行一次脚本，在查看结果树中看第二次发送的请求（第二次请求利用了正则表达式），从图5-13中可以看到请求中wd传递的参数值已经是用正则表达式提取到的“besttest”了。



5-13JMeter 关联运行结果

## JMeter检查点—响应断言

JMeter也跟LoadRunner一样有检查点，不过JMeter的检查点是用断言实现的，JMeterJMeter中集成了BSF断言、响应断言、XML断言等，其中使用的比较多的是响应断言，这里用响应断言跟大家一起学习。

通过“添加”→“断言”→“响应断言”就能添加响应断言了。如果添加了断言就需要在对应的监听器添加“断言结果”才能在执行后查看断言结果是否正确。断言的具体设置如图5-15所示：

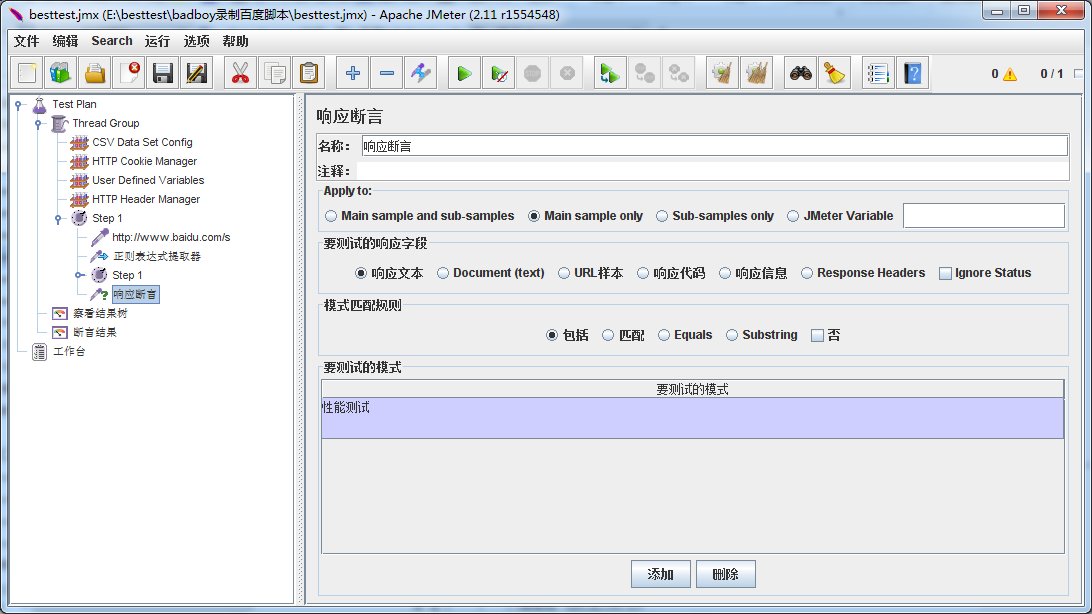


图5-15 响应断言设置

断言相关界面解释如下：

* 名称：该节点的名称，可自行定义。--随意
* 注释：对该节点的注释，有意义即可。--随意
* Apply to：默认即可。
* **要测试的响应字段：**可以选择文本、代码，一般选择文本即可。
* **模式匹配规则：**

包括：返回结果包括指定的内容，一般断言的时候匹配规则选择包括即可；

匹配：返回的结果与指定的内容进行精确匹配，因为响应的内容一般还包含其他的信息，比如HTML语言标签，所以很难准确匹配；

Equals：返回结果与指定结果一致；

Substring：返回返回结果是指定结果的字串；

否：不进行匹配。

* **要测试的模式（断言内容）：**也就是要断言的具体内容，可以是中文或者英文。可以设置多个断言内容，如果设置多个断言内容，则需要多个断言内容都在返回结果值中都找到测试结果才会通过，只要有一个失败则整个测试结果都是失败的。（多个断言时只要前一个出错，就不会再校验后一个断言内容）

之前的脚本中，返回值里有“性能测试”这个字段，这里将“性能测试”设置为断言内容，详情见图5-15。运行测试脚本，在断言结果里如果没有报错则表明断言成功，如果报错则表明断言失败。将断言内容从“性能测试”改为“好好学习”，此时运行脚本后在断言结果中查看就会会看到报错信息，断言结果中报“测试失败”，如图5-15所示。

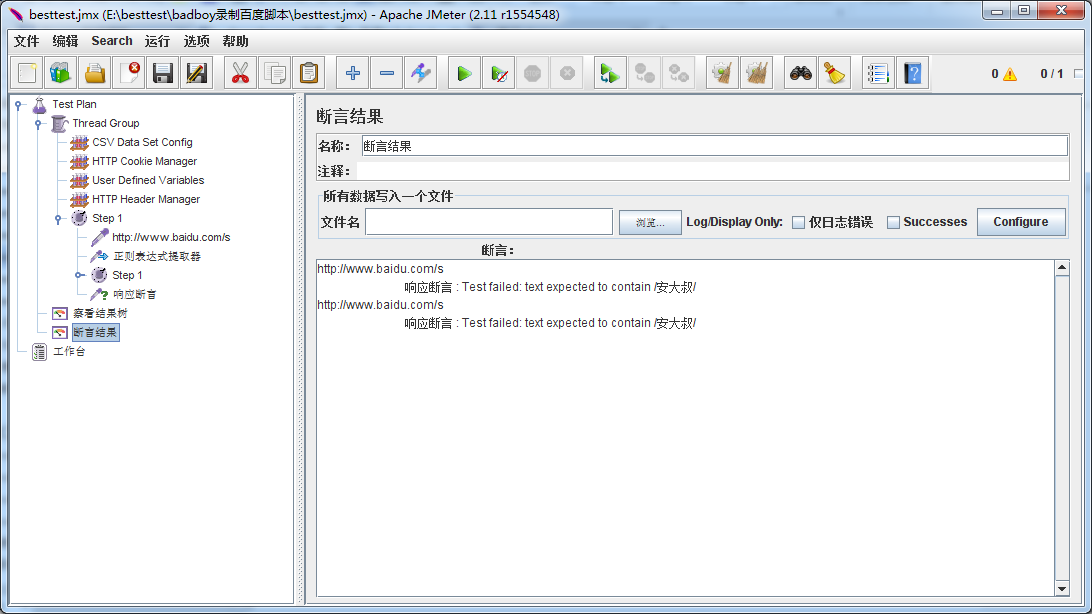


图5-16 断言结果错误

## 思考时间和集合点—定时器

在学完了JMeter的参数化、关联和检查点后，还剩下思考时间和集合点没有学习了，下面就继续来学习JMeter的思考时间和集合点的相关知识。

### . 思考时间—两次请求间停留的间隔

JMeter中也有类似于LoadRunner的思考时间，JMeter的思考时间是用定时器实现的。默认情况下，JMeter线程在发送请求之间是没有间隔的。建议为线程组添加定时器，设定请求之间的间隔时间，一方面是为了更加真实地模拟用户的请求，另一方面也是为了能够减少服务器压力。如果不设定这种延迟，则JMeter可能会在短时间内产生大量请求，导致服务器可能压力过大而瘫痪。但是设置定时器会让作用域内的每一个采样器都在执行前等待一个固定时长，这里需要注意的是不管定时器是放在sampler之前还是sampler之后，定时器都会在sampler之前执行，如果硬是需要在某个位置之后暂停，那么加入“Test Action”采样器即可。

定时器也是有作用域的，当执行一个sampler之前时，所有当前作用域内的定时器都会被执行，如果只希望定时器作用于其中的一个sampler，就把定时器放在该sampler下。如果测试人员为线程组添加了多个定时器，那么JMeter会将这些定时器的时长叠加起来，共同影响作用域范围内的采样器。定时器可以作为采样器或者逻辑控制器的子项，目的是只影响作用域内的采样器。

JMeter常用的定时器有如下几种：

（1）固定定时器：最重用的定时器，比如设置固定定时器时长为2000毫秒，那么在每次sampler运行之前都会固定等待2秒钟。固定定时器的延时不会计入单个sampler的响应时间，但会计入事务控制器的时间。比如“java请求”，定时器相当于LoadRunner中的pacing（迭代之间的间隔时间）；对于“事务控制器”来说，定时器相当于LoadRunner中的think time（真实用户在操作过程中的等待时间）；

（2）高斯随机定时器：定时器的延时时间在指定范围内的正态分布；

（3）Uniform Random Timer：也就是均匀随机定时器，延时时间在指定范围内，并且每个时间取值的概率相同；

（4）Constant Throughput Timer：也就是固定吞吐量定时器，这种定时器在特定场景非常有用，可以让JMeter以指定数字的吞吐量（即指定TPS，JMeter这里要求指定每分钟的执行数，而不是每秒）执行。吞吐量计算的范围可以为指定为当前线程、当前线程组、所有线程组，并且计算吞吐量的依据可以是最近一次线程的执行时间；

（5）Synchronizing Timer：同步定时器也就是JMeter的集合点了，与LoadRunner的结合点基本相当，可以使线程进行等待，当达到指定的线程数后，在一起执行，可以模拟同时的并发。

### 集合点—同步定时器

LoadRunner中集合点可以设置多个虚拟用户等待到一个点，同时触发一个事务,以达到模拟真实环境下多个用户同时操作,实现性能测试的最终目的。JMete中也有集合点这个概念，JMeter中使用Synchronizing Timer实现集合点的功能，模拟多用户并发测试，即多个线程在同一时刻并发请求。将测试元件的Synchronizing Timer添加到请求之前或某个Sampler的子节点，右键菜单依次选择“添加→定时器→Synchronizing Timer”，如图5-17所示。

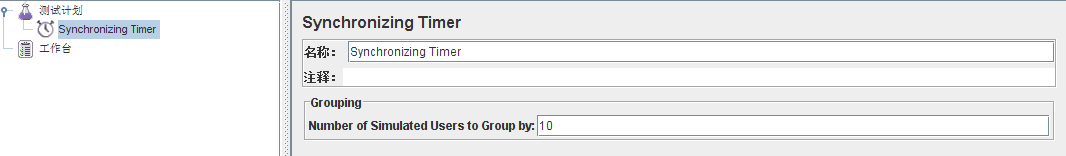


图5-17 JMeter集合点

Synchronizing Timer相关界面解释如下：

* 名称：该节点的名称，可自行定义。
* 注释：对该节点的注释，有意义即可。
* Number of Simulated users to Group by：集合点集合够N个用户才开始执行并发，使用中发现最后一批线程数不够集合点数目时，jmeter会停止不动，如果碰到这种情况，只能杀掉JMeter进程重新执行测试。

## 运行结果分析

JMeter的运行结果都是需要添加监听器，常用的监听器包括聚合报告、图形结果、监视器结果、Aggregate Graph。查看结果树。在运行性能场景的时候，尽量不要同时开启多个监听器，因为监听器会消耗系统资源，更多的监听器也会导致测试结果的失真，所以在运行性能测试场景的时候，尽量选择一个合适的监听器，下面挑选几个比较常用的监听器进行说明。

### 聚合报告（Aggregate Report/ Aggregate Graph）

JMeter的聚合报告，如图5-18所示。

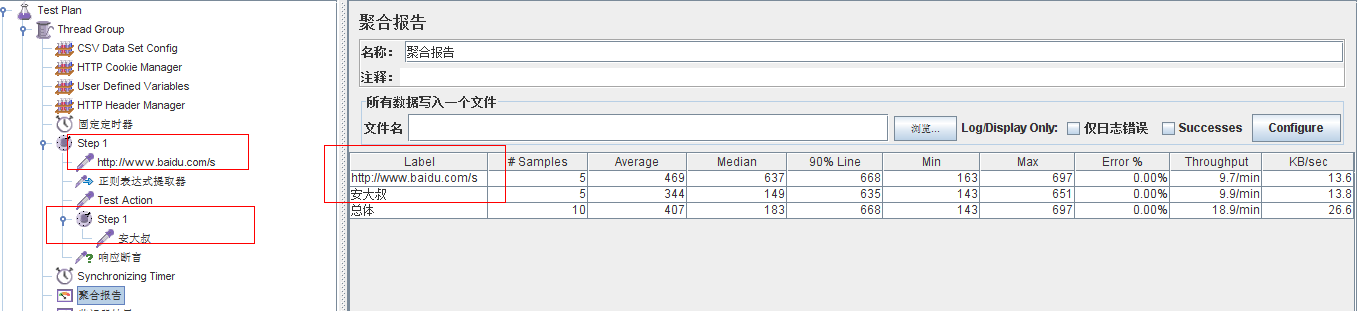


图5-18 jmetet的聚合报告

JMeter聚合报告相关解释如下：

* **Label：请求的名称，**如图5-16中左侧的请求名称有http://www.baidu.com/s和好好学习，聚合报告中也显示这2个名称；
* **Samples：本次测试一共运行了多少个请求，**比如图5-16中有2个请求，迭代5次，一共运行10次；
* **Average：测试结果的平均响应时间**，单位毫秒；
* **Median：测试结果中的**中间值，即50%服务器响应时间低于该值50%的服务器响应时间高于该值；
* **90% Line：**测试结果中90%用户的响应时间不超过该值；
* **Min：**测试结果中最小响应时间；
* **Max：**测试结果中最大响应时间；
* **Error%：**本次测试错误率，性能测试的前提是需要保证业务的正确性的，如果错误率过高，那么性能结果一般也是不能接受的。行业内一般要求是99.99%，当然，这个不是绝对的；
* **Throughput：**吞吐量；默认情况下表示每秒完成的请求数（Request per Second）；
* **KB/Sec：每秒从服务器端接收到的数据量**。

点击“Configure”后会弹出监听器的公告属性，可以对报告保存的结果信息进行修改，建议保存为CSV格式，因为这样会节约系统资源。

“Aggregate Graph”的结果与聚合报告基本是一样的，多了一个“Graph”页，可以以图形化的方式将请求的评价响应时间展示出来，在这里就在多进行介绍。

### 查看结果树

请求、响应数据可以有很多种显示方式，可以选择jsion，html等等显示方式；

### 图形结果

JMeter的图形结果如图5-19所示。



图 5-19 JMeter图形结果

JMeter图形结果相关解释如下：

* 样本数目：总共从服务器返回请求结果的个数；
* 最新样本：最近一次请求的响应时间；
* 吞吐量：服务器每分钟处理的请求数；
* 平均：总运行时间除以发送到服务器的请求数得到的响应时间；
* 中值：运行时间范围内所有响应时间的中间值，代表一半请求的响应时间会高于这个值，一半请求的响应时间会低于这个值；
* 偏离：服务器响应时间变化、离散程度测量值的大小，类似LoadRunner中的标准方差。

### 用表格查看结果树

JMeter可用表格查看结果树，如图5-20所示。

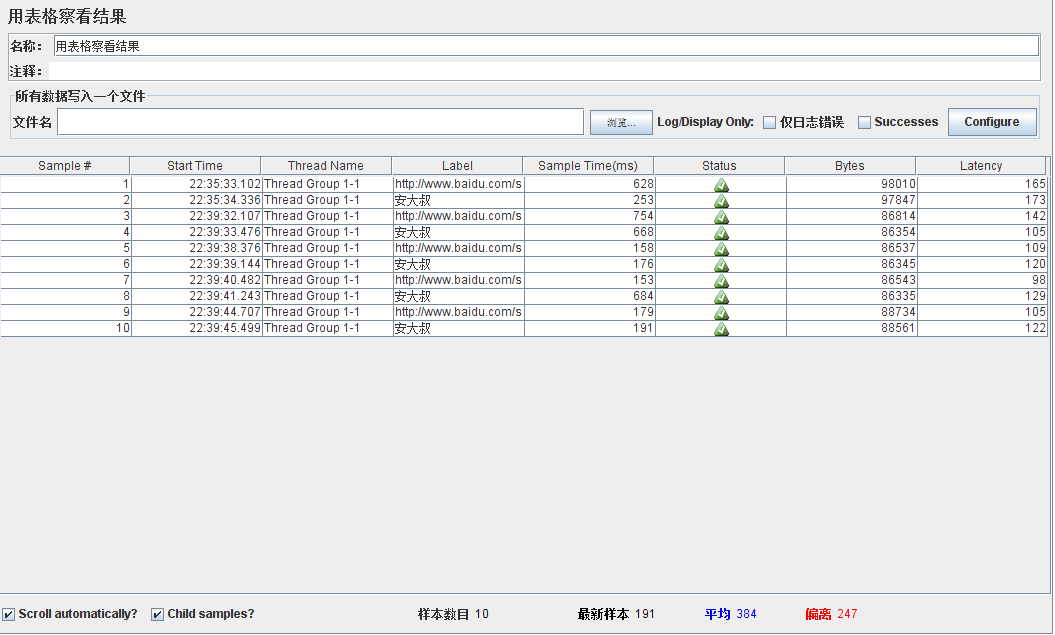


图 5-20 用表格查看结果树

JMeter用表格查看结果树相关解释如下：

* Sample：每个请求的序号；
* Start Time：每个请求运行开始时间；
* Thread Name：每个线程的名称；
* Label：请求名称；
* Sample Time（ms）：每个请求所花时间，单位为毫秒；
* Status：请求状态，如果为勾则表示成功，如果为叉表示失败；但如果是勾，也并不能认为请求就一定完全成功了，因为还得看Bytes的字节数是否是所请求网页的正常大小值，如果不是则说明发生了丢包现象，所以如果status为勾也不代表一定是成功的；
* Bytes：请求的字节数；
* 样本数目：也就是执行了多少次请求；
* 最新样本：表示服务器响应最后一个请求的响应时间；

平均这些参数代表的意思跟图形结果中的一样，在这里就不重复介绍了。

### 断言结果

## JMeter之分布式测试—多台负载机测试

作为一个纯 Java的GUI应用，JMeter对于CPU和内存的消耗还是很惊人的，所以当需要模拟数以千计的并发用户时，使用单台机器模拟所有的并发用户就有些力不从心，甚至还会引起Java内存溢出的错误。不过，JMeter也可以像 LoadRunner 一样通过使用多台机器运行所谓的代理来分担负载产生器自身的压力，并借此来获取更大的并发用户数，在这里只需手动配置一下即可。

Windows下的JMeter分布式配置：

1. **所有负载机jmeter安装：**在所有期望运行JMeter作为负载产生器的机器上安装JMeter，并确定其中一台机器作为 控制器，其他的机器作为代理。
2. **运行所有代理机器上的jmeter-server.bat文件，**这里使用两台机器172.20.80.47和172.20.80.68作为代理服务器；
3. **查找jmeter.properties（JMeter/bin）**：在Controller 机器的JMeter安装目录下找到 bin 目录，再找到 jmeter.properties 这个文件，使用记事本或者其他文字编辑工具打开它；
4. **修改jmeter.properties（JMeter/bin）：**在打开的文件中查找“remote\_hosts=IP:端口”这个字符串，可以找到这样一行“remote\_hosts=127.0.0.1”。其中的 127.0..0.1 表示运行JMeter代理的机器，这里需要修改为“remote\_hosts=172.20.80.47，172.20.80.68”；其中的端口1099不要改动，这是默认的JMeter控制器与代理之间通信的RMI端口号，如果想更换端口号，则根据运行环境进行设置。

1）Windows运行如下命令：

SET SERVER\_PORT=1099

JMeter-SERVER（不添加端口时）

2）linux运行如下命令：

SERVER\_PORT=1099 JMeter-server；

（4）**配置JMeter的环境变量**，如果不配置环境变量，分布式会启动不成功。Windows下JMeter的环境变量的配置如下：

JMeter\_HOME=D:\soft\test\apache-JMeter-2.11；

CLASSPATH=;%JMeter\_HOME%\lib\ext\ApacheJMeter\_core.jar;%JMeter\_HOME%\lib\jorphan.jar;%JMeter\_HOME%\lib\logkit-2.0.jar;

PATH=;%JMeter\_HOME%\bin；

（5）保存文件，并重新启动控制器机器上的JMeter，并进入“启动”→“远程启动”菜单项。就会看到刚才添加的两个代理的地址，选中即可运行，如果想同时启动所有代理，选择远程全部启动即可。

（6）Linux下的JMeter分布式配置。JMeter的分布式可以跟LoadRunner一样将agent代理装在Linux下面，JMeter分布式在Linux下的配置与Windows下配置基本是一样的，唯一不同的就是Linux下的环境变量配置不同，Linux下JMeter环境变量的配置如下：

export JMeter\_HOME=/usr/local/apache-JMeter-2.11

export PATH=$JMeter\_HOME/lib/ext/ApacheJMeter\_core.jar:$JMeter\_HOME/lib/jorphan.jar:$JMeter\_HOME/lib/logkit-2.0.jar:$JMeter\_HOME/bin:$PATH

JMeter分布式控制过程中，各个Agent启动的线程数等于线程组中的配置，不是均分线程组中的配置。

# JMeter JDBC脚本实战

在学习过程中，发现JMeter测试JDBC特别方便，不需要像LoadRunner一样去录制SQLPLUS操作形成Oracle脚本或者自己写JDBC请求来形成脚本。在这里就简单跟大家一起学习下JMeter的测试MySql脚本的开发过程。

## 创建JDBC连接

### 导入jar包

注意一点的是进行JDBC测试都需要依赖相应的Jar包（放入相应的：apache-jmeter-2.11\lib），不同的数据库类型对于不同的Jar包，参考如表5-2。

表5-2 JMeter数据库对应的Jar文件

|  |  |
| --- | --- |
| **数据库** | **对应的Jar文件** |
| Oracle | classes12.zip 或 ojdbc14.jar |
| Sybase | jconn2.jar 或 jconn3.jar |
| Mysql | mysql-connector-java-3.1.1-alpha-bin.jar 或  mysql-connector-java-5.0.2-beta-bin.jar |
| SQLServer 2000 | msbase.jar、msutil.jar、mssqlserver.jar |
| SQLServer 2005 | sqljdbc.jar |
| SQLServer 7.0 | jtds-1.2.2.jar |
| Informix | ifxjdbc.jar |
| DB2 | db2jcc.jar、db2jcc\_license\_cu.jar |

### 添加一个“JDBC Connection Configuration”配置元件

在测试计划下添加线程组，在线程组下从“配置元件”中添加一个“JDBC Connection Configuration”，具体信息参考图5-21。

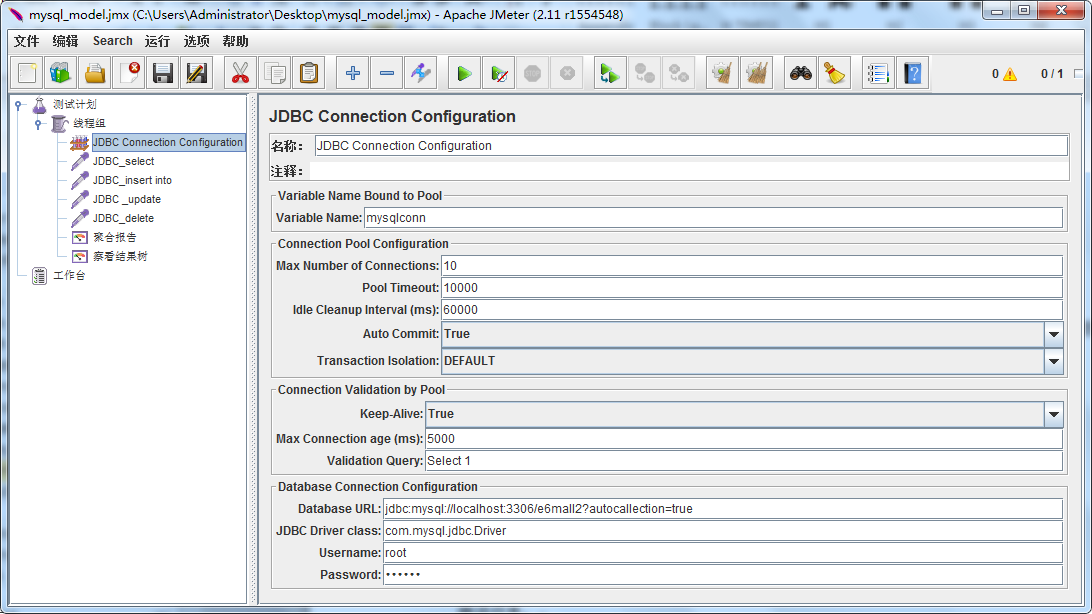


图5-21 JDBC Connection Configuration配置

各参数属性解释如下：

* Variable Name：JDBC创建的连接池的名称，这里测试的是Mysql，所以设置为MySql，这里一定要注意要跟JDBC Request中的Variable Name保持一致；
* Max Number Connection：JDBC连接池最大连接数，如果该值设置为0，表示线程之间独立不使用共享线程池，如果要使用共享线程池，确保该值大于等于最大线程数；
* Pool Timeout：创建连接池时的超时时间，如果超过这个时间，系统会抛出错误信息；
* Idle Cleanup Interval (ms)：连接池的空闲时间，默认是1分钟；
* Auto Commit：是否自动提交，默认选择为True；
* Transaction Isolation：数据库事物隔离级别，默认为default。MySql数据库事物隔离级别有4种，分别是Read Uncommitted、Read Committed、Repeatable Read、Serializable。级别越低，支持的并发往往会更大，并且系统开销也会越小，附录2对MySql数据库的事物隔离级别有详细的介绍；
* Keep-alive：是否开启Keep-alive机制，如果开启，则需要设定keep-alive的超时时间。
* Max Connection Age (ms)：keep-alive的超时时间，在连接超时之前，新的请求会重用之前建立的连接而不会重新创建新的链接，如果超过这个超时时间，则会断开原有连接，新的请求会重新创建新的链接；
* Validation Query：测试JDBC连接是否畅通，默认是“select 1 from default”；
* Database URL：JDBC连接字符串，这里测试的是Mysql，则可以写成jdbc:mysql://host[:port]/dbname；其他类型数据库对应的Driver class以及Database URL参考表5-1；

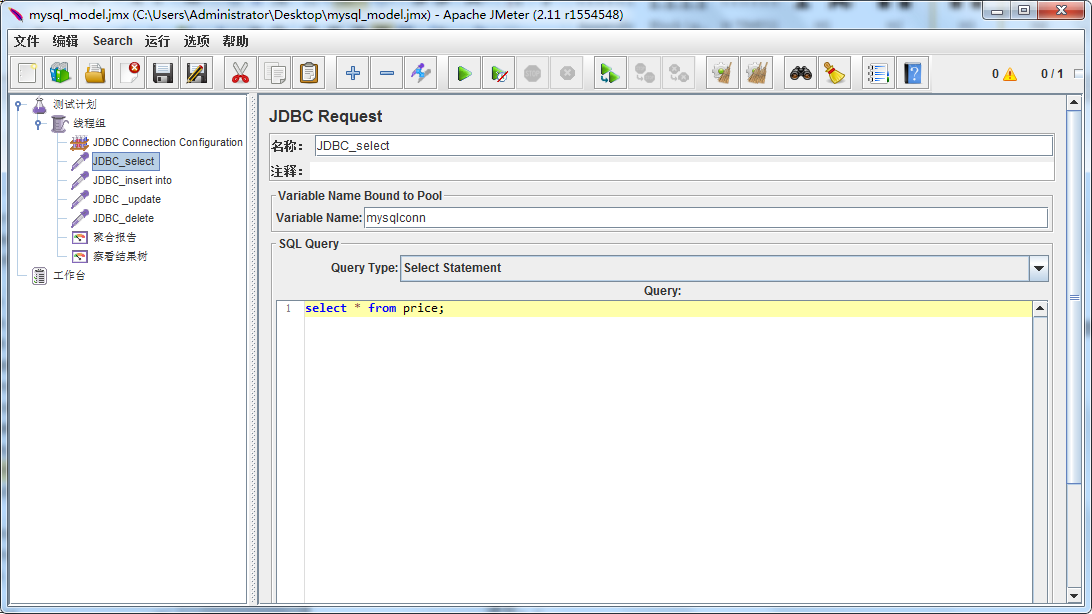
表5-1 JMeter数据库驱动名以及对应的URL

|  |  |  |
| --- | --- | --- |
| **Database** | **Driver class** | **URL** |
| **MySQL** | **com.mysql.jdbc.Driver** | **jdbc:mysql://host[:port]/dbname** |
| **PostgreSQL** | **org.postgresql.Driver** | **jdbc:postgresql:{dbname}** |
| **Oracle** | **oracle.jdbc.OracleDriver** | **jdbc:oracle:thin:@//host:port/service OR** |
| **jdbc:oracle:thin:@(description=(address=(host={mc-name})(protocol=tcp)(port={port-no}))(connect\_data=(sid={sid})))** |
| **Ingres (2006)** | **ingres.jdbc.IngresDriver** | **jdbc:ingres://host:port/db[;attr=value]** |
| **SQL Server (MS JDBC driver)** | **com.microsoft.sqlserver.jdbc.SQLServerDriver** | **jdbc:sqlserver://host:port;DatabaseName=dbname** |
| **Apache Derby** | **org.apache.derby.jdbc.ClientDriver** | **jdbc:derby://server[:port]/databaseName[;URLAttributes=value[;...]]** |
| **Sybase** | **com.sybase.jdbc2.jdbc.SybDriver** | **jdbc:sybase:Tds:hostip:port/dbname** |
| **DB2** | **com.ibm.db2.jcc.DB2Driver** | **jdbc:db2://hostip:port/dbname** |
| **Informix** | **com.informix.jdbc.IfxDriver** | **jdbc:informix-sqli://hostip:port/dbname:informixserver=<dbservername>** |
| **MongoDB** | **com.mongodb.jdbc.MongoDriver** | **jdbc:mongo://192.168.152.132:29847/database\_name** |

* JDBC Driver class：JDBC驱动的类名，这里测试MySql，所以设置的是com.mysql.jdbc.Driver。不同类型的驱动名称可参考表5-1；
* Username：数据库用户名；
* Password：数据库密码。

## 创建JDBC测试用例

右键单击左侧的线程组，在弹出的菜单中选择：添加→Sampler→JDBC Request，就会创建一个新的JDBC测试用例。按照如下方式设置属性：



* **Variable Name：**使用的JDBC连接池名字。需要与“JDBC Connection Configuration”中“Variable Name”保持一致，否则会报错；
* **Query Type：**SQL查询的类型，主要包括Select 、Update、Callable Statement、Commit、Rollback等。
* 多个查询语句(不使用参数的情况下)可以放在一起顺序执行，需要设置Query Type为Callable Statement，然后顺序输入select语句，不用加go或者分号。如果Query Type设置为Select Statement，则只执行第一个SQL语句。
* 如果SQL语句中使用参数变量，那么Query Type 需要设置为：Select Statement，只执行第一条SQL语句。变量可以直接在在JDBC Request下方的Parameter Values中输入${变量名称},${变量名称}。也可以使用文件本地文件，也可以通过CSV Data Set Config进行参数化；
* 对于Insert、Update、Delete语句，需要设置Query Type为：Update Statament。数据修改语句中可以使用参数，而且可以顺序执行多个修改语句。
* Query：SQL查询语句，这里写需要执行的SQL语句，这里写的是；

insert into bf\_test\_user(username,password,realname,email) values('${username}','e10adc3949ba59abbe56e057f20f883e','${username}','${username}@test.com');

--update bf\_test\_user set realname='study' where username='${username}';

--delete from bf\_test\_user where username='${username}';

* Parameter values：SQL查询的参数，这里介绍通过在CSV文件中读取参数的方式。这里设置为 ：

${\_\_CSVRead(D:/goodgoodstudy/studytest.csv,0)}

其中的0表示CSV文件的第0列。如果SQL查询语句有多个参数，可以在“Parameter values”中用逗号分隔继续写。例如：

${\_\_CSVRead(D:/goodgoodstudy/studytest.csv,0)},${\_\_CSVRead(D:/goodgoodstudy/studytest.csv,1)}

* Parameter types：SQL查询参数的类型。这里设置为VARCHAR，如果有多个参数使用逗号分隔。

## 添加监听器

创建完成测试用例之后，就可以创建监听器查看我们的结果数据。

右键选择“测试计划”，点击“添加”→“监听器”→“查看结果树”，可以看到每一次请求的详细信息，包括SQL语句、参数的内容、响应的结果、相应的时间等等。建议在调试脚本的时候，首先添加“查看结果树”以查看脚本发送的请求是否成功，如果不成功也可以通过该监听器查看返回内容，从而查找错误原因。

脚本通过添加“查看结果树”监听器调试无误后，就右键选择“测试计划”，点击“添加”→“监听器”→“Aggregate Graph”，然后删除“查看结果树”监听器。运行一段时间性能测试场景后，就可以在“Aggregate Graph”监听器里查看总体的运行情况，包括平均响应时间、吞度量、错误率等。

# JMeter SOAP脚本实战Web Service

JMeter可以用来很方便的对SOAP（包括LoadRunner中的 Web Service）协议进行进行测试。

SOAP 使用 RPC（远程过程调用）和消息传递来建立通信服务，SOAP RPC 定义了用于表示远程过程调用和应答的协议。SOAP 协议本身仅仅定义了消息的交换结构，它可以和许多现存因特网协议结合在一起使用，其中包括超文本传输协议（ HTTP），多用途网际邮件扩充协议（MIME），Java 消息服务（JMS）以及简单邮件传输协议（SMTP）等。

根据 JMS 的规范，消息交换有 2 种方式：消息发布 / 订阅方式和点对点方式。由这两种交换方式所建立的消息收发系统都是异步的，即 JMS 客户机可以发送消息而不必等待回应。JMeter 同样提供了两种 Sampler 分别建立对这两种服务的调用：SOAP/XML-RPC Request 和 JMS Point-to-Point。前者使用互联网中最为广泛的超文本传输协议（ HTTP）而后者使用 JMS 协议，JMS 是 Java 平台面向消息中间件的技术规范，用它来提供创建、发送、接收、读取消息的服务。

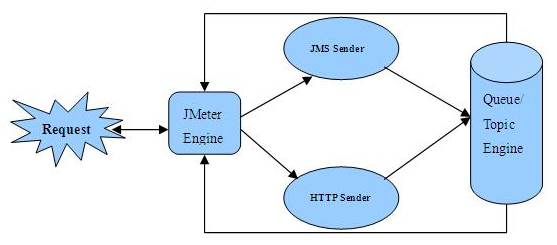


图5-22 JMeter 框架基于不同协议时，对 SOAP 消息的调用机制流程

JMeter测试SAOP的方法很简单，通过线程组→SOAP/XML-RPC Request就能形成简单的SOAP请求测试用例（见图5-23），这里用邮政编码的Web Service的“getAddressByZipCode”接口进行测试，邮政编码Web Service地址为：http://www.webxml.com.cn/WebServices/ChinaZipSearchWebService.asmx。

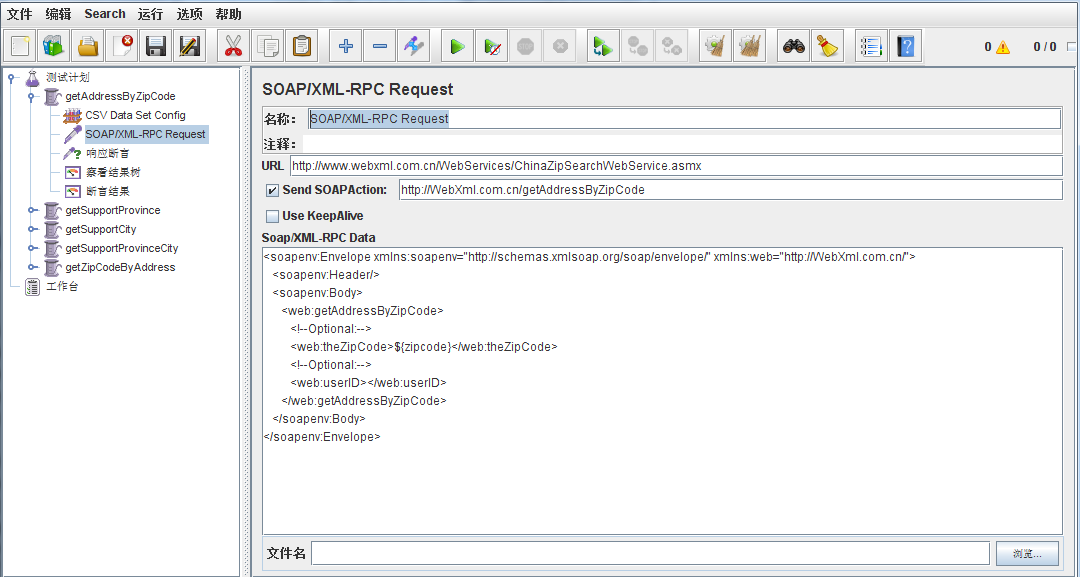


图5-23 JMeter测试SOAP测试

* URL：邮政编码的Web Servic URL地址；
* Send SOAPAction：SOAP请求的目的地，这里填写是“http://WebXml.com.cn/getAddressByZipCode”，其实就是接口的URL地址；
* Use KeepAlive：是否开启KeepAlive机制，KeepAlive的作用可以参考本文5.4章节的JDBC脚本实战中关于KeepAlive的解释。
* SOAP/XML-RPC Date：请求的主体数据，请求内容如下：

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:web="http://WebXml.com.cn/">

<soapenv:Header/>

<soapenv:Body>

<web:getAddressByZipCode>

<!--Optional:-->

<web:theZipCode>100011</web:theZipCode>

<!--Optional:-->

<web:userID></web:userID>

</web:getAddressByZipCode>

</soapenv:Body>

</soapenv:Envelope>

添加“查看结果树”后，就能看到响应数据中返回的邮编100011适用的各地区的地址。

对于刚接触SAOP请求的人来说，可能不知道SOAP请求以及SOAPAction怎么填写，在这里可以用SaopUI快速方便地获取这些信息。

SaopUI是一款针对SOAP的功能/性能测试工具，分为开源以及商业版本，下载地址为http://www.soapui.org/，下载开源版本即可。

打开SoapUI，选择“File”→“new soapUI Project”，输入“Project Name”→“Initial WSDL/WADL”输入URL地址（比如：[http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx?wsdl），完成之后点击OK](http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx?wsdl）-%3eOK)按钮。

完成之后如图5-24所示，会看到该web service的接口以及接口默认的请求。图5-23中“Request”左侧即SAOP请求的主体，参数初始化默认显示为“？”，这些”？“就是接口参数，改成需要测试的数值即可，“Request”右侧为Response结果。

将SoapUI请求主体复制到JMeter的SOAP/XML-RPC Date中形成JMeter SOAP脚本的请求主体。

然后查看SoapUI请求中的“Raw”视图，找到“RAW”试图中的“SOAPAction”值，比如这个天气预报的SOAPAction值为 <http://WebXml.com.cn/getRegionCountry>。将该SOAPAction的内容复制到图5-23中JMeter SAOP脚本中的“Send SOAPAction”中，这样整个JMeter中SOAP测试用例就形成了。

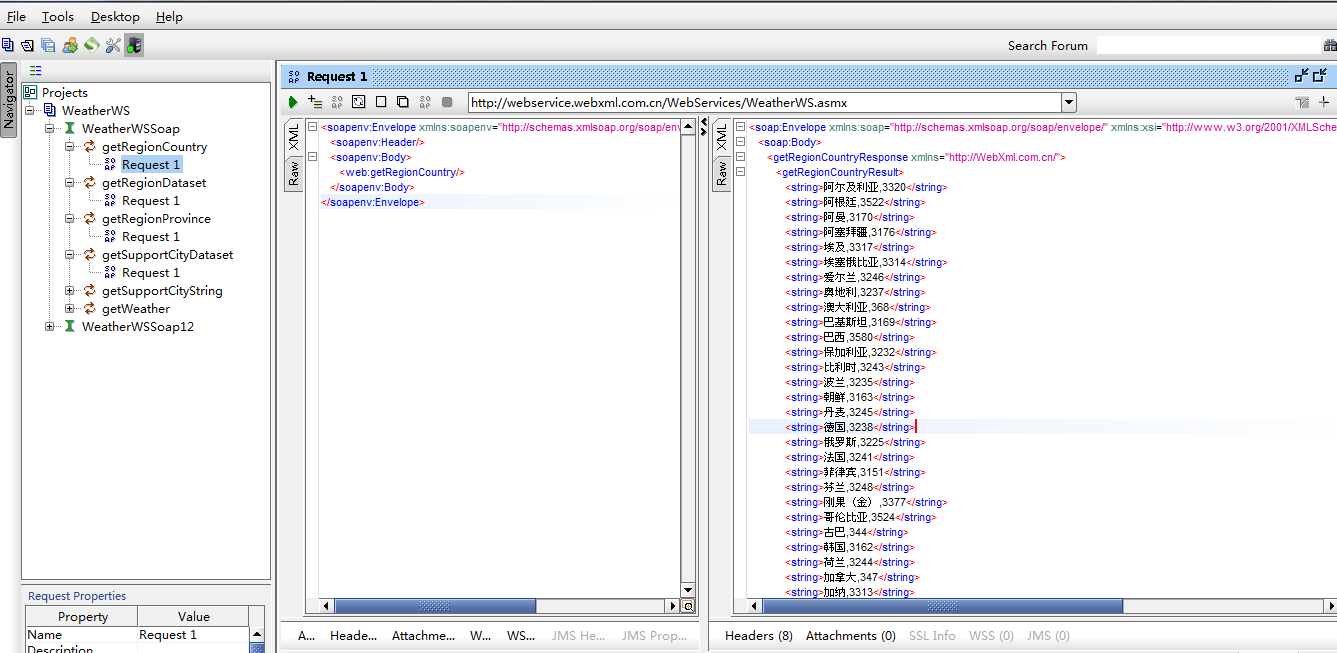


图5-24 SoapUI请求界面

# 基于jmeter java协议的接口测试

## 通用方法

### 核心步骤

1. 创建一个Java工程；
2. 将JMeter的lib目录下的jar文件(ApacheJMeter\_java.jar/ApacheJMeter\_core.jar)添加进此工程的Build Path；
3. 创建一个类并实现JavaSamplerClient接口或继承AbstractJavaSamplerClient，并重写；
4. public Arguments getDefaultParameters(){Arguments.addArgument("参数名", "默认值")};设置可用参数的默认值；显示到Jmeter的参数列表中，第一个参数为参数默认的显示名称，第二个参数为默认值
5. public void setupTest(JavaSamplerContext arg0){可为空、new SampleResult()}：每个线程测试前执行一次，做一些初始化工作+获取参数等(只实行一次!!)；
6. public SampleResult runTest(JavaSamplerContext arg0){ SampleResult().setSampleLabel()/SampleStart()/SampleEnd()/setSuccessful(true) }：开始测试，从arg0参数可以获得参数值；
7. public void teardownTest(JavaSamplerContext arg0){可为空}：测试结束时调用；
8. Export为Runnable Jar File；
9. 将此jar包放入JMETER\_HOME\lib\ext目录；
10. 以管理员身份打开JMeter；
11. 创建线程组、Java Request、查看结果树，进行测试；

### 实例2: extends AbstractJavaSamplerClient

1. 在eclipse里面新建一个工程：JavaForJMeter
2. 把{Jmeter\_home}\lib\ext目录下的所有jar(ApacheJMeter\_java.jar/ApacheJMeter\_core.jar)添加进此工程的Build Path；
3. 添加要测试的类Hello，代码如下：

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

package com.test.webservice;

public class Hello {

public String sayHello()

{

return "Hello";

}

public String sayHelloToPerson(String s)

{

if(s == null || s.equals(""))

s = "nobody";

return (new StringBuilder()).append("Hello ").append(s).toString();

}

public int sum(int a,int b)

{

return a+b;

}

}

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

1. 添加类perftest，并继承AbstractJavaSamplerClient，并添加代码如下：

注意：System.out等的输出会显示在启动JMeter时的命令窗口内，但过多的输出会影响性能的准确性。

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

package com.test.webservice;

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;

import com.test.webservice.Hello;

public class perftest extends AbstractJavaSamplerClient {

private String a;

private String b;

/\*\* Holds the result data (shown as Response Data in the Tree display). \*/

private String resultData;

// 这个方法是用来自定义java方法入参的。

// params.addArgument("num1","");表示入参名字叫num1，默认值为空。显示到Jmeter的参数列表中，第一个参数为参数默认的显示名称，第二个参数为默认值

//设置可用参数列表及的默认值；

public Arguments getDefaultParameters() {

Arguments params = new Arguments();

params.addArgument("num1", "");

params.addArgument("num2", "");

return params;

}

//每个线程测试前执行一次，做一些初始化工作；

public void setupTest(JavaSamplerContext arg0) {

}

//开始测试，从arg0参数可以获得参数值；

public SampleResult runTest(JavaSamplerContext arg0) {

a = arg0.getParameter("num1");

b = arg0.getParameter("num2");

SampleResult sr = new SampleResult();//请求开始，可以放到setupTest中  
　　　　 sr.setSampleLabel( "Java请求哦");//在聚合报告中显示的请求名称

try {

sr.sampleStart();// jmeter 开始统计响应时间标记

Hello test = new Hello();

// 通过下面的操作就可以将被测方法的响应输出到Jmeter的察看结果树中的响应数据里面了。

resultData = String.valueOf(test.sum(Integer.parseInt(a), Integer

.parseInt(b)));

if (resultData != null && resultData.length() > 0) {

sr.setResponseData("结果是："+resultData, null);

sr.setDataType(SampleResult.TEXT);

}

// System.out.println(resultData);

sr.setSuccessful(true); //jmeter统计为成功

} catch (Throwable e) {

sr.setSuccessful(false); //jmeter统计为失败

e.printStackTrace();

} finally {

sr.sampleEnd();// jmeter 结束统计响应时间标记

}

return sr;

}

//测试结束时调用；

public void teardownTest(JavaSamplerContext arg0) {

// System.out.println(end);

// System.out.println("The cost is"+(end-start)/1000);

}

// main只是为了调试用，最后打jar包的时候注释掉。

/\*

public static void main(String[] args)

{ // TODO Auto-generated method stub

Arguments params = new Arguments();

params.addArgument("num1", "1");//设置参数，并赋予默认值1

params.addArgument("num2", "2");//设置参数，并赋予默认值2

JavaSamplerContext arg0 = new JavaSamplerContext(params);

perftestbbb test = new perftestbbb();

test.setupTest(arg0);

test.runTest(arg0);

test.teardownTest(arg0);

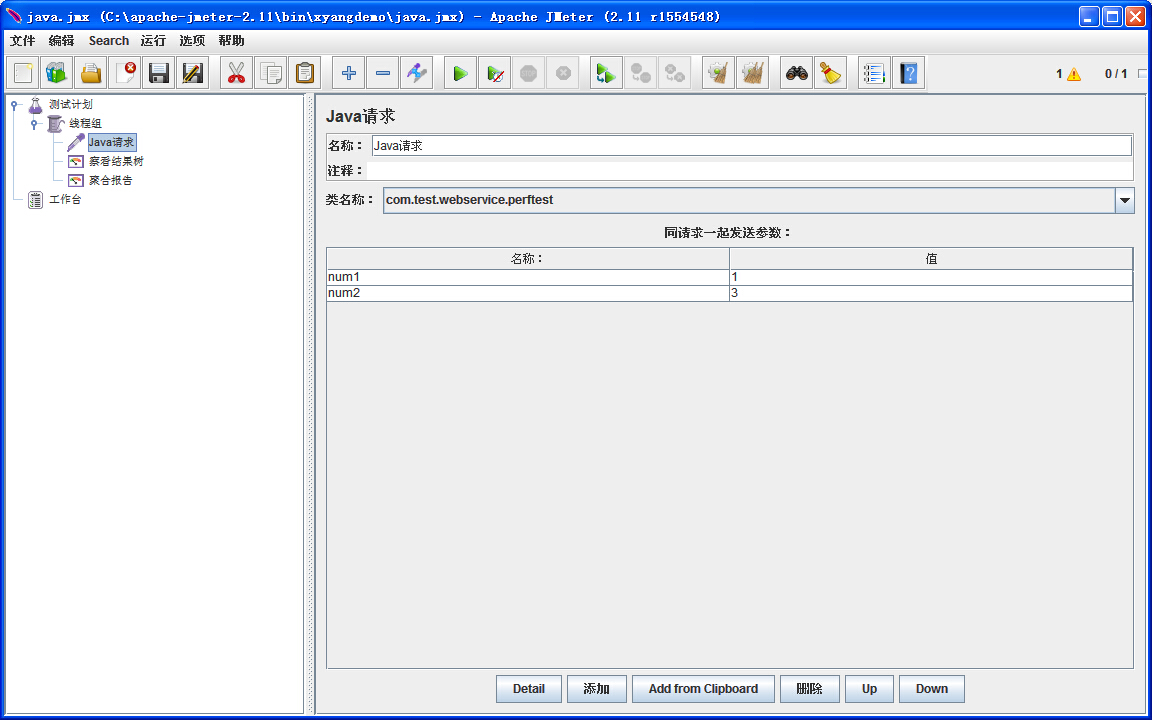
}

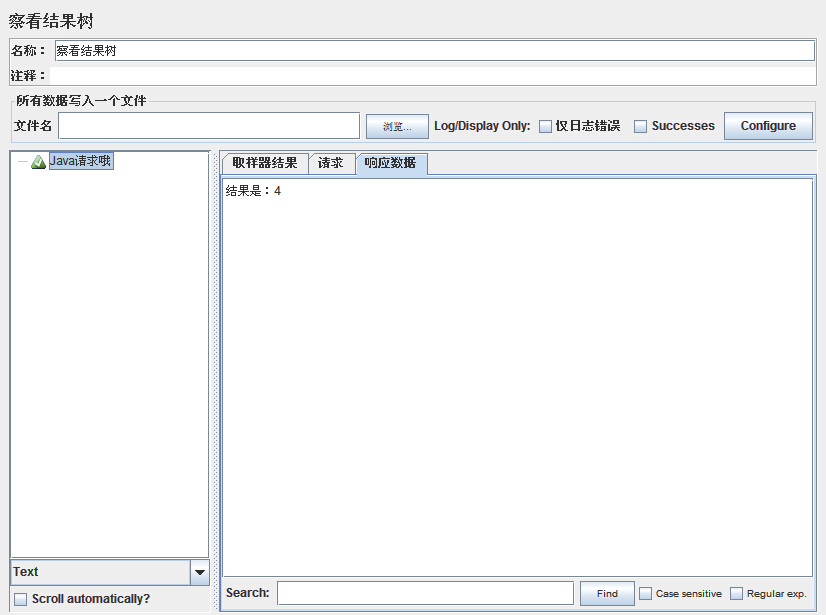
\*/

}

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

1. Export为Runnable Jar File：hello.jar
2. 将此hello.jar包放入JMETER\_HOME\lib\ext目录；
3. 打开JMeter,添加线程组，添加Java请求，查看结果树。





### 实例：implements JavaSamplerClient

服务为：将输入的两个参数通过IO存入文件；

#### 打开MyEclipse，编写Java代码

##### 服务：

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

package test;

import java.io.File;

import java.io.PrintWriter;

public class OutputService {

public static void output(String filename, int a, int b) throws Exception {

PrintWriter out = new PrintWriter(new File(filename));

out.write(a + ":" + b);

out.close();

}

}

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

##### 测试类：

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

package test;

import org.apache.jmeter.config.Arguments;

import org.apache.jmeter.protocol.java.sampler.JavaSamplerClient;

import org.apache.jmeter.protocol.java.sampler.JavaSamplerContext;

import org.apache.jmeter.samplers.SampleResult;

//服务为：将输入的两个参数通过IO存入文件；

public class PerformenceTest implements JavaSamplerClient {

private SampleResult results;

private String a;

private String b;

private String filename;

// 设置传入的参数，可以设置多个，已设置的参数会显示到Jmeter的参数列表中

public Arguments getDefaultParameters() {

Arguments params = new Arguments();

params.addArgument("filename", "0");// 设置参数，并赋予默认值0

params.addArgument("a", "0");// 设置参数，并赋予默认值0

params.addArgument("b", "0");// 设置参数，并赋予默认值0

return params;

}

// 初始化方法，实际运行时每个线程仅执行一次，在测试方法运行前执行

public void setupTest(JavaSamplerContext arg0) {

results = new SampleResult();

}

// 测试执行的循环体，根据线程数和循环次数的不同可执行多次

public SampleResult runTest(JavaSamplerContext arg0) {

b = arg0.getParameter("b"); // 获取在Jmeter中设置的参数值

a = arg0.getParameter("a"); // 获取在Jmeter中设置的参数值

filename = arg0.getParameter("filename"); // 获取在Jmeter中设置的参数值

results.sampleStart();// jmeter 开始统计响应时间标记

try {

OutputService test = new OutputService();

test.output(filename, Integer.parseInt(a), Integer.parseInt(b));

results.setSuccessful(true);

// 被测对象调用

} catch (Throwable e) {

results.setSuccessful(false);

e.printStackTrace();

} finally {

results.sampleEnd();// jmeter 结束统计响应时间标记

}

return results;

}

// 结束方法，实际运行时每个线程仅执行一次，在测试方法运行结束后执行(空)

public void teardownTest(JavaSamplerContext arg0) {

}

/\*

public static void main(String[] args) {

// TODO Auto-generated method stub

Arguments params = new Arguments();

params.addArgument("a", "0");// 设置参数，并赋予默认值0

params.addArgument("b", "0");// 设置参数，并赋予默认值0

params.addArgument("filename","abc.txt");

JavaSamplerContext arg0 = new JavaSamplerContext(params);

PerformenceTest test = new PerformenceTest();

test.setupTest(arg0);

test.runTest(arg0);

test.teardownTest(arg0);

}

\*/

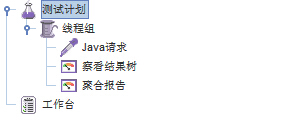
}

[[说明: 复制代码](javascript:void(0);)](javascript:void(0);" \o ""复制代码" )

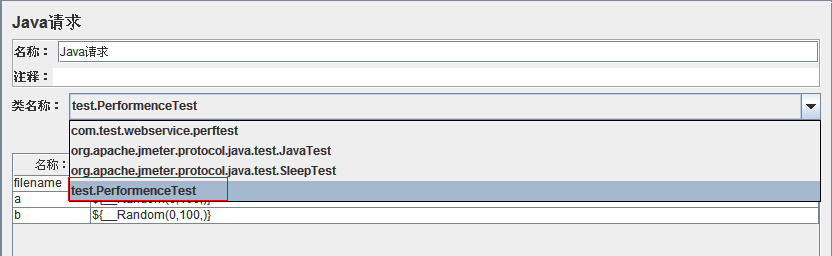
#### 生成jar包，放在JMeter\_HOME/lib/ext目录下。

#### 设置JMeter

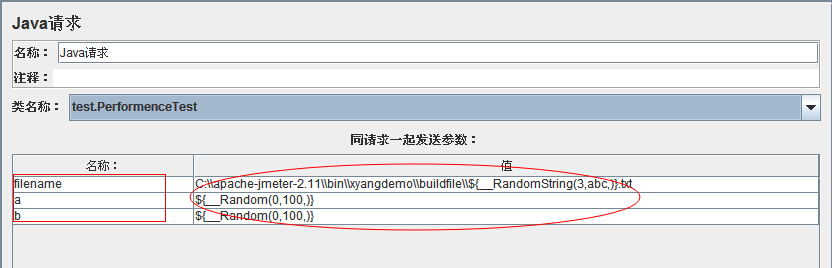
以管理员身份打开JMeter，并创建Java Request后,结构如下图所示：



在Java请求中发现了自己新建的测试类：



我们在JMeter中发现参数有三个：

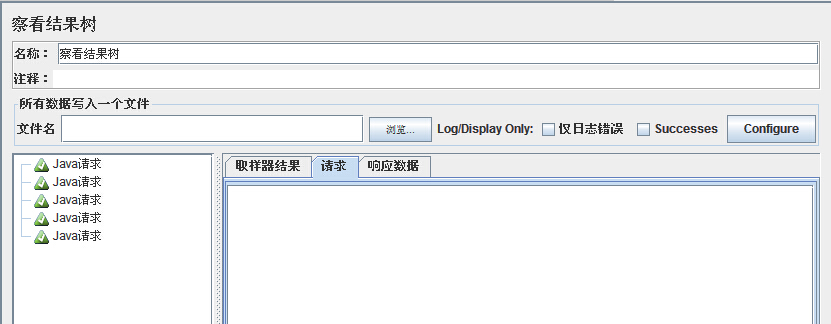


在这里我们使用到了两个函数：

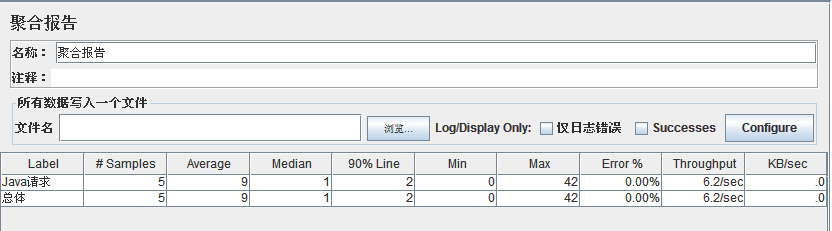
(1)\_Random生成随机整数；

(2)\_RandomString生成随机字符串；

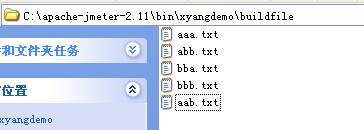
在线程组中设置并发线程数为5，保存后即可运行；



聚合报告：



结果成功，成功生成了5个文件：(表示执行没有问题)



#### 遇到的问题

3.1 Export为Jar File导致Java Request的类名称无法找到；

3.2 JMeter java.lang.OutOfMemoryError: PermGen space问题：

在JMeter.bat 中修改：

set HEAP=-Xms512m -Xmx1024m

set NEW=-XX:NewSize=128m -XX:MaxNewSize=128m

set SURVIVOR=-XX:SurvivorRatio=8 -XX:TargetSurvivorRatio=50%

set TENURING=-XX:MaxTenuringThreshold=2

set RMIGC=-Dsun.rmi.dgc.client.gcInterval=600000 -Dsun.rmi.dgc.server.gcInterval=600000

set PERM=-XX:PermSize=256m -XX:MaxPermSize=512m

3.3 JMeter需要以管理员身份打开，否则会出现：(文件位置)



### JMeter Java Sampler介绍

#### implements JavaSamplerClient接口或extends AbstractJavaSamplerClient，并重写：

implements JavaSamplerClient/extends AbstractJavaSamplerClient

|  |  |
| --- | --- |
| Arguments | **getDefaultParameters**() 用于获取界面的参数(给JAVA Sample传入的参数） |
| SampleResult | **runTest**(JavaSamplerContext context) 类似于LR的Action，result.sampleStart()一个事务开始，result.sampleEnd()一个事务 结束 |
| void | **setupTest**(JavaSamplerContext context) 初始化方法，类似于LR的init和Junit中的setUp() |
| void | **teardownTest**(JavaSamplerContext context) 类似于LR的end和Junit中的tearDown() |

#### 执行的先后顺序为：

Arguments **getDefaultParameters**()

--> void **setupTest**(JavaSamplerContext context)

--> SampleResult **runTest**(JavaSamplerContext context)

--> void **teardownTest**(JavaSamplerContext context)

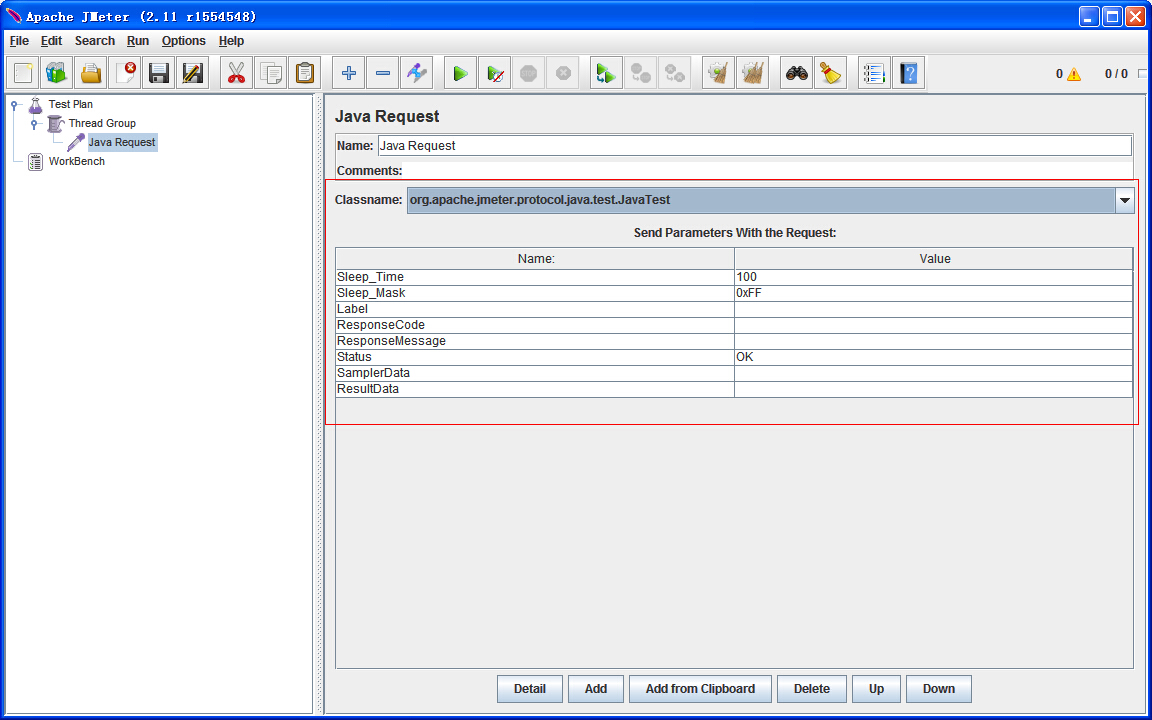
#### 常用的方法：

1. Arguments.addArgument("name", "value")    定义参数
2. JavaSamplerContext.getArgument("name", "value") 获取定义的参数
3. SampleResult.setSampleLabel( "Java请求哦");//在聚合报告中显示的请求名称
4. SampleResult.sampleStart()    定义事务的开始，类似于LR的lr\_start\_transaction，和LR一样事务间不要放无关代码
5. SampleResult.sampleEnd()    定义事务的结束，类似于LR的lr\_end\_transaction
6. SampleResult.setSuccessful(true/false)    设置运行结果的成功或失败，Jmeter统计成功失败的次数，在聚合报告中能够体现。
7. SampleResult().setResponseData("结果是："+resultData, null);--响应数据出现在查看结果数，调试的时候用到
8. SampleResult().setDataType(SampleResult.TEXT);-- 数据结果格式（跟上一条一起

### 自带Java Request Sampler

JMeter默认实现了两个Java Request Sampler：SleepTest Sampler和JavaTest Sampler。

JavaTest Sampler不向服务器发送数据请求，一般被用作调试目的使用。



## jmeter java实现http协议 PerfmonTest代码

### 源码：类HttpRequest(Get/Post)

**package** com.util;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.io.PrintWriter;

**import** java.net.URL;

**import** java.net.URLConnection;

**import** java.util.List;

**import** java.util.Map;

**public** **class** HttpRequest {

/\*\*

\* 向指定URL发送GET方法的请求

\*

\* **@param** url

\* 发送请求的URL

\* **@param** param

\* 请求参数，请求参数应该是 name1=value1&name2=value2 的形式。

\* **@return** URL 所代表远程资源的响应结果

\*/

**public** **static** String sendGet(String url, String param) {

String result = "";

BufferedReader in = **null**;

String urlNameString="";

**try** {

**if**(param.length()>0){

urlNameString = url + "?" + param;

// System.out.printf("urlNameString的值是：：%s",urlNameString);

}**else** {

urlNameString = url;

//System.err.printf("urlNameString的值是：：%s",urlNameString);

}

//url

URL realUrl = **new** URL(urlNameString);

// 打开和URL之间的连接

URLConnection connection = realUrl.openConnection();

// 设置通用的请求属性

connection.setRequestProperty("accept", "\*/\*");

connection.setRequestProperty("connection", "Keep-Alive");

connection.setRequestProperty("user-agent",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");

// 建立实际的连接

connection.connect();

// 获取所有响应头字段

Map<String, List<String>> map = connection.getHeaderFields();

// 遍历所有的响应头字段

**for** (String key : map.keySet()) {

System.*out*.println(key + "--->" + map.get(key));

}

// 定义 BufferedReader输入流来读取URL的响应

in = **new** BufferedReader(**new** InputStreamReader(

connection.getInputStream(),"UTF-8"));

String line;

**while** ((line = in.readLine()) != **null**) {

result += line;

}

} **catch** (Exception e) {

System.*out*.println("发送GET请求出现异常！" + e);

e.printStackTrace();

}

// 使用finally块来关闭输入流

**finally** {

**try** {

**if** (in != **null**) {

in.close();

}

} **catch** (Exception e2) {

e2.printStackTrace();

}

}

**return** result;

}

/\*\*

\* 向指定 URL 发送POST方法的请求

\*

\* **@param** url

\* 发送请求的 URL

\* **@param** param

\* 请求参数，请求参数应该是 name1=value1&name2=value2 的形式。

\* **@return** 所代表远程资源的响应结果

\*/

**public** **static** String sendPost(String url, String param) {

PrintWriter out = **null**;

BufferedReader in = **null**;

String result = "";

**try** {

URL realUrl = **new** URL(url);

// 打开和URL之间的连接

URLConnection conn = realUrl.openConnection();

// 设置通用的请求属性

conn.setRequestProperty("accept", "\*/\*");

conn.setRequestProperty("connection", "Keep-Alive");

conn.setRequestProperty("user-agent",

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");

conn.setRequestProperty("Content-Type", "application/json");

// 发送POST请求必须设置如下两行

conn.setDoOutput(**true**);

conn.setDoInput(**true**);

// 获取URLConnection对象对应的输出流

out = **new** PrintWriter(**new** OutputStreamWriter(conn.getOutputStream() ,"UTF-8"));

// 发送请求参数

out.print(param);

// flush输出流的缓冲

out.flush();

// 定义BufferedReader输入流来读取URL的响应

in = **new** BufferedReader(

**new** InputStreamReader(conn.getInputStream(),"UTF-8"));

String line;

**while** ((line = in.readLine()) != **null**) {

result += line;

System.*out*.println(result);

}

} **catch** (Exception e) {

System.*out*.println("发送 POST 请求出现异常！"+e);

e.printStackTrace();

}

//使用finally块来关闭输出流、输入流

**finally**{

**try**{

**if**(out!=**null**){

out.close();

}

**if**(in!=**null**){

in.close();

}

}

**catch**(IOException ex){

ex.printStackTrace();

}

}

**return** result;

}

### 解析：发送get/post请求类（HttpRequest）

[通过java.net.URLConnection发送HTTP请求的方法](http://www.cnblogs.com/nick-huang/p/3859353.html)

[JDK中的URLConnection参数详解](http://www.blogjava.net/supercrsky/articles/247449.html)

#### **JDK中的URLCONNECTION参数详解**

##### **分类：HTTP请求粗分为两种，一种是GET请求，一种是POST请求**

1. **get请求可以获取静态页面，也可以把参数放在URL字串后面，传递给servlet;**
2. **post与get的不同之处在于post的参数不是放在URL字串里面，而是放在http请求的正文内。**

##### **大概步骤：使用Java发送这两种请求的代码大同小异，只是一些参数设置的不同。步骤如下：**

1. **通过统一资源定位器（java.net.URL()）获取连接器（java.net.URLConnection()）及URLConnection的子类：HttpURLConnection()**
2. **设置请求的参数httpUrlConnection.setDoOutput()/.setRequestMethod("POST")**
3. **发送请求httpUrlConnection.connect();**
4. **以输入流的形式获取返回内容**
5. **关闭输入流**

**要重新发送数据时需要重新创建连接、重新设参数、重新创建流对象、重新写数据、**

**// 重新发送数据(至于是否不用重新这些操作需要再研究)**

##### **创建连接：URLConnection的对象问题: (URL/ (URLConnection/ HttpURLConnection )URL. openConnection())**

**URLConnection的对象,如下代码示例:**

**// 下面的index.jsp由<servlet-mapping>映射到**

**// 一个Servlet(com.quantanetwork.getClientDataServlet)**

**// 该Servlet的注意点下边会提到**

**URL url = new URL("http://localhost:8080/TestHttpURLConnectionPro/index.jsp");**

**URLConnection rulConnection = url.openConnection();**

**// 此处的urlConnection对象实际上是根据URL的**

**// 请求协议(此处是http)生成的URLConnection类**

**// 的子类HttpURLConnection,故此处最好将其转化**

**// 为HttpURLConnection类型的对象,以便用到**

**// HttpURLConnection更多的API.如下:**

**//转化**

**HttpURLConnection httpUrlConnection = (HttpURLConnection) rulConnection;**

##### **设参数：（配置）> HttpURLConnection对象参数问题 (httpUrlConnection.setDoOutput()/setDoInput()/ setUseCaches()/ setRequestProperty()/setRequestMethod("POST"))**

**对connection对象的一切配置（那一堆set函数） 都必须要在connect()函数执行之前完成**

**// 设置是否向httpUrlConnection输出，因为这个是post请求，参数要放在**

**// http正文内，因此需要设为true, 默认情况下是false; （get/post的区别）**

**httpUrlConnection.setDoOutput(true);**

**// 设置是否从httpUrlConnection读入，默认情况下是true;**

**httpUrlConnection.setDoInput(true);**

**// Post 请求不能使用缓存**

**httpUrlConnection.setUseCaches(false);**

**// 设定传送的内容类型是可序列化的java对象**

**// (如果不设此项,在传送序列化对象时,当WEB服务默认的不是这种类型时可能抛java.io.EOFException)**

**httpUrlConnection.setRequestProperty("Content-type", "application/x-java-serialized-object");**

**// 设定请求的方法为"POST"，默认是GET**

**httpUrlConnection.setRequestMethod("POST");**

##### **连接：从上述第2条中url.openConnection()至此的配置必须要在connect之前完成，**

**httpUrlConnection.connect();**

##### **创建流对象： HttpURLConnection连接问题：（httpUrlConnection.getOutputStream()）**

**// 此处getOutputStream（获取输出流）会隐含的进行connect(即：如同调用上面的connect()方法，**

**// 所以在开发中不调用上述的connect()也可以)。**

**OutputStream outStrm = httpUrlConnection.getOutputStream();**

##### **写数据： HttpURLConnection写数据与发送数据问题： (向HTTP请求传入参数)**

**// 现在通过输出流对象构建对象输出流对象，以实现输出可序列化的对象。**

**ObjectOutputStream objOutputStrm = new ObjectOutputStream(outStrm);**

**// 向对象输出流写出数据，这些数据将存到内存缓冲区中**

**objOutputStrm.writeObject(new String("我是测试数据"));**

**// 刷新对象输出流，将任何字节都写入潜在的流中（些处为ObjectOutputStream）**

**objOutputStm.flush();**

**// 关闭流对象。此时，不能再向对象输出流写入任何数据，先前写入的数据存在于内存缓冲区中,**

**// 在调用下边的getInputStream()函数时才把准备好的http请求正式发送到服务器**

**objOutputStm.close();**

**// 调用HttpURLConnection连接对象的getInputStream()函数, (获取请求报文)**

**// 将内存缓冲区中封装好的完整的HTTP请求电文发送到服务端。**

**InputStream inStrm = httpConn.getInputStream(); // <===注意，实际发送请求的代码段就在这里**

**// 上边的httpConn.getInputStream()方法已调用,本次HTTP请求已结束,下边向对象输出流的输出已无意义，**

**// 既使对象输出流没有调用close()方法，下边的操作也不会向对象输出流写入任何数据.**

**// 因此，要重新发送数据时需要重新创建连接、重新设参数、重新创建流对象、重新写数据、**

**// 重新发送数据(至于是否不用重新这些操作需要再研究)**

**objOutputStm.writeObject(new String(""));**

**httpConn.getInputStream();**

##### **总结：**

**a) HttpURLConnection的connect()函数，实际上只是建立了一个与服务器的tcp连接，并没有实际发送http请求。**

**无论是post还是get，http请求实际上直到HttpURLConnection的HttpURLConnection().getInputStream()这个函数里面才正式发送出去。**

**b) 在用POST方式发送URL请求时，URL请求参数的设定顺序是重中之重，**

**对connection对象的一切配置（那一堆set函数）**

**都必须要在connect()函数执行之前完成。而对outputStream的写操作，又必须要在inputStream的读操作之前。**

**这些顺序实际上是由http请求的格式决定的。**

**如果inputStream读操作在outputStream的写操作之前，会抛出例外：**

**java.net.ProtocolException: Cannot write output after reading input.......**

**c) http请求实际上由两部分组成，**

**一个是http头，所有关于此次http请求的配置都在http头里面定义，**

**一个是正文content。**

**connect()函数会根据HttpURLConnection对象的配置值生成http头部信息，因此在调用connect函数之前，**

**就必须把所有的配置准备好。**

**d) 在http头后面紧跟着的是http请求的正文，正文的内容是通过outputStream流写入的，**

**实际上outputStream不是一个网络流，充其量是个字符串流，往里面写入的东西不会立即发送到网络，**

**而是存在于内存缓冲区中，待outputStream流关闭时，根据输入的内容生成http正文。**

**至此，http请求的东西已经全部准备就绪。在getInputStream()函数调用的时候，就会把准备好的http请求**

**正式发送到服务器了，然后返回一个输入流，用于读取服务器对于此次http请求的返回信息。由于http**

**请求在getInputStream的时候已经发送出去了（包括http头和正文），因此在getInputStream()函数**

**之后对connection对象进行设置（对http头的信息进行修改）或者写入outputStream（对正文进行修改）**

**都是没有意义的了，执行这些操作会导致异常的发生。**

##### **Servlet端的开发注意点：**

**a) 对于客户端发送的POST类型的HTTP请求，Servlet必须实现doPost方法，而不能用doGet方法。**

**b) 用HttpServletRequest的getInputStream()方法取得InputStream的对象，比如：**

**InputStream inStream = httpRequest.getInputStream();**

**现在调用inStream.available()（该方法用于“返回此输入流下一个方法调用可以不受阻塞地**

**从此输入流读取（或跳过）的估计字节数”）时，永远都反回0。试图使用此方法的返回值分配缓冲区，**

**以保存此流所有数据的做法是不正确的。那么，现在的解决办法是**

**Servlet这一端用如下实现：**

**InputStream inStream = httpRequest.getInputStream();**

**ObjectInputStream objInStream = new ObjectInputStream(inStream);**

**Object obj = objInStream.readObject();**

**// 做后续的处理**

**// 。。。。。。**

**// 。。。 。。。**

**而客户端，无论是否发送实际数据都要写入一个对象（那怕这个对象不用），如：**

**ObjectOutputStream objOutputStrm = new ObjectOutputStream(outStrm);**

**objOutputStrm.writeObject(new String("")); // 这里发送一个空数据**

**// 甚至可以发一个null对象，服务端取到后再做判断处理。**

**objOutputStrm.writeObject(null);**

**objOutputStrm.flush();**

**objOutputStrm.close();**

**注意:上述在创建对象输出流ObjectOutputStream时,如果将从HttpServletRequest取得的输入流**

**(即:new ObjectOutputStream(outStrm)中的outStrm)包装在BufferedOutputStream流里面,**

**则必须有objOutputStrm.flush();这一句,以便将流信息刷入缓冲输出流.如下:**

**ObjectOutputStream objOutputStrm = new ObjectOutputStream(new BufferedOutputStream(outStrm));**

**objOutputStrm.writeObject(null);**

**objOutputStrm.flush(); // <======此处必须要有.**

**objOutputStrm.close();**

**HttpURLConnection是基于HTTP协议的，其底层通过socket通信实现。如果不设置超时（timeout），在网络异常的情况下，可能会导致程序僵死而不继续往下执行。可以通过以下两个语句来设置相应的超时：**

**System.setProperty("sun.net.client.defaultConnectTimeout", 超时毫秒数字符串);**

**System.setProperty("sun.net.client.defaultReadTimeout", 超时毫秒数字符串);**

**其中： sun.net.client.defaultConnectTimeout：连接主机的超时时间（单位：毫秒）**

**sun.net.client.defaultReadTimeout：从主机读取数据的超时时间（单位：毫秒）**

**例如：**

**System.setProperty("sun.net.client.defaultConnectTimeout", "30000");**

**System.setProperty("sun.net.client.defaultReadTime**

**Java中可以使用HttpURLConnection来请求WEB资源。**

**HttpURLConnection对象不能直接构造，需要通过URL.openConnection()来获得HttpURLConnection对象，示例代码如下：**

**String szUrl = "http://www.ee2ee.com/";**

**URL url = new URL(szUrl);**

**HttpURLConnection urlCon = (HttpURLConnection)url.openConnection();**

**HttpURLConnection是基于HTTP协议的，其底层通过socket通信实现。如果不设置超时（timeout），在网络异常的情况下，可能会导致程序僵死而不继续往下执行。可以通过以下两个语句来设置相应的超时：**

**System.setProperty("sun.net.client.defaultConnectTimeout", 超时毫秒数字符串);**

**System.setProperty("sun.net.client.defaultReadTimeout", 超时毫秒数字符串);**

**其中： sun.net.client.defaultConnectTimeout：连接主机的超时时间（单位：毫秒）**

**sun.net.client.defaultReadTimeout：从主机读取数据的超时时间（单位：毫秒）**

**例如：**

**System.setProperty("sun.net.client.defaultConnectTimeout", "30000");**

**System.setProperty("sun.net.client.defaultReadTimeout", "30000");**

**JDK 1.5以前的版本，只能通过设置这两个系统属性来控制网络超时。在1.5中，还可以使用HttpURLConnection的父类URLConnection的以下两个方法：**

**setConnectTimeout：设置连接主机超时（单位：毫秒）**

**setReadTimeout：设置从主机读取数据超时（单位：毫秒）**

**例如：**

**HttpURLConnection urlCon = (HttpURLConnection)url.openConnection();**

**urlCon.setConnectTimeout(30000);**

**urlCon.setReadTimeout(30000);**

**需要注意的是，笔者在JDK1.4.2环境下，发现在设置了defaultReadTimeout的情况下，如果发生网络超时，HttpURLConnection会自动重新提交一次请求，出现一次请求调用，请求服务器两次的问题（Trouble）。我认为这是JDK1.4.2的一个bug。在JDK1.5.0中，此问题已得到解决，不存在自动重发现象。out", "30000");**

## socket协议 PerfmonTest代码

### 源码：发送socket请求的类、方法

/\*\*

\* 发送消息

\* **@param** ip

\* **@param** port

\* **@param** msg

\* **@throws** Exception

\*/

**private** **void** sendMsg(String ip, **int** port, String msg) **throws** Exception{

sid = **new** Socket(ip, port);

in = **new** BufferedReader(**new** InputStreamReader(sid.getInputStream()));

out = **new** PrintWriter(sid.getOutputStream(), **true**);

out.println(msg);

in.readLine();

System.*out*.println("success");

}

## 

# JMeter使用注意事项

（1）JMeter 2.11版本支持的JDK为1.6以上版本，如果发现无法打开JMeter，先检查是否JDK版本不正确；

（2）为了保证测试数据的尽可能准确，尽量使用非GUI进行测试，比如使用JMeter -n -t test.jmx -l test.jtl来进行测试；

（3）尽可能少使用监听器，因为过多的监听器会消耗更多的系统资源；

（4）尽量把相似的取样器放在循环内，使用变量（CSV Data Set Config）修改这些Sampler，而不是重复添加Sampler。或者可能会用到Access Log Sampler（在这里不能使用Include Controller，因为它会将指定文件中的所有测试元件直接添加到测试计划中来）；

（5）不要使用函数测试模式（Functional Mode）；

（6）以CSV格式输出测试结果，尽量不要使用XML格式，因为CSV更节约系统资源；

（7）如果需要保持测试数据结果，则仅保存你需要的数据，这样会节约系统资源；

（8）断言最好在调试脚本时候使用，在运行测试场景时候应该禁用断言，这样的测试结果会更准确；

（9）负载测试期间不要使用“查看结果树”“用表格查看结果”监听器，这两个监听器最好只在调试脚本时使用；

（10）如果测试需要大量数据（特别是随机产生的数据），可以提前准备好测试数据放到数据文件中，以CSV Data Set Config方式读取，这样就能避免在测试运行阶段去创建这些数据从而浪费资源；

（11）大并发量的负载测试，尽量在多台机器上运行多个非GUI JMeter实例，进行分布式测试；

（12）如果测试过程中报内存不足，则在JMeter的bin目录下找到JMeter的启动文件JMeter.bat（linux下是JMeter.sh），打开该文件并修改该文件中的JVM内存设置，JVM内存修改为比如：HEAP="-Xms512m -Xmx512m"，最大堆内存跟最小堆内存保持一致，这样可以减小频繁创建销毁引起的内存开销；

（13）在测试SOAP协议并使用SoapUI时，如果报出类似“Error loading [http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl]: org.apache.xmlbeans.XmlException: java.io.CharConversionException: Characters larger than 4 bytes are not supported: byte 0xb1 implies a length of more than 4 bytes ”这样的错误，解决办法是在SoapUI的安装路径下找到.vmoptions文件，在末尾加入 “-Dfile.encoding=UTF8” 便可解决该问题。

# 附录A 正则表达式全集

正则表达式有多种不同的风格。表A-1是在[PCRE](http://zh.wikipedia.org/w/index.php?title=PCRE&action=edit&redlink=1" \o "PCRE（页面不存在）)中元字符及其在正则表达式上下文中的行为的一个完整列表，适用于Perl或者Python编程语言（grep或者egrep的正则表达式文法是PCRE的子集）：

表A-1 正则表达式全集

|  |  |
| --- | --- |
| 字符 | 描述 |
| \ | 将下一个字符标记为一个特殊字符、或一个原义字符、或一个向后引用、或一个八进制转义符。例如，“n”匹配字符“n”。“\n”匹配一个换行符。串行“\\”匹配“\”而“\(”则匹配“(” |
| ^ | 匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置 |
| $ | 匹配输入字符串的结束位置。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置 |
| \* | 匹配前面的子表达式零次或多次。例如，zo\*能匹配“z”、“zo”以及“zoo”。\*等价于{0,} |
| + | 匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,} |
| ? | 匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”中的“do”。?等价于{0,1} |
| {n} | n是一个非负整数。匹配确定的n次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o |
| **{*n*,}** | *n*是一个非负整数。至少匹配*n*次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o\*” |
| **{*n*,*m*}** | *m*和*n*均为非负整数，其中*n*<=*m*。最少匹配*n*次且最多匹配*m*次。例如，“o{1,3}”将匹配“fooooood”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格 |
| **?** | 当该字符紧跟在任何一个其他限制符（\*,+,?，{*n*}，{*n*,}，{*n*,*m*}）后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串“oooo”，“o+?”将匹配单个“o”，而“o+”将匹配所有“o” |
| **.** | 匹配除“\*n*”之外的任何单个字符。要匹配包括“\*n*”在内的任何字符，请使用像“(.|\n)”的模式 |
| **(pattern)** | 匹配pattern并获取这一匹配的子字符串。该子字符串用于向后引用。所获取的匹配可以从产生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中则使用$0…$9属性。要匹配圆括号字符，请使用“\(”或“\)” |
| **(?:pattern)** | 匹配pattern但不获取匹配的子字符串，也就是说这是一个非获取匹配，不存储匹配的子字符串用于向后引用。这在使用或字符“(|)”来组合一个模式的各个部分是很有用。例如“industr(?:y|ies)”就是一个比“industry|industries”更简略的表达式 |
| **(?=pattern)** | 正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，“Windows(?=95|98|NT|2000)”能匹配“Windows2000”中的“Windows”，但不能匹配“Windows3.1”中的“Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始 |
| **(?!pattern)** | 正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如“Windows(?!95|98|NT|2000)”能匹配“Windows3.1”中的“Windows”，但不能匹配“Windows2000”中的“Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始 |
| **(?<=pattern)** | 反向肯定预查，与正向肯定预查类似，只是方向相反。例如，“(?<=95|98|NT|2000)Windows”能匹配“2000Windows”中的“Windows”，但不能匹配“3.1Windows”中的“Windows” |
| **(?<!pattern)** | 反向否定预查，与正向否定预查类似，只是方向相反。例如“(?<!95|98|NT|2000)Windows”能匹配“3.1Windows”中的“Windows”，但不能匹配“2000Windows”中的“Windows” |
| **x|y** | 匹配x或y。例如，“z|food”能匹配“z”或“food”。“(z|f)ood”则匹配“zood”或“food”。 |
| **[xyz]** | 字符集合（character class）。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。特殊字符仅有反斜线\保持特殊含义，用于转义字符。其他特殊字符如星号、加号、各种括号等均作为普通字符。脱字符^如果出现在首位则表示负值字符集合；如果出现在字符串中间就仅作为普通字符。连字符 - 如果出现在字符串中间表示字符范围描述；如果如果出现在首位则仅作为普通字符 |
| **[^xyz]** | 排除型（negate）字符集合。匹配未列出的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin” |
| **[a-z]** | 字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符 |
| **[^a-z]** | 排除型的字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符 |
| **\b** | 匹配一个单词边界，也就是指单词和空格间的位置。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er” |
| **\B** | 匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er” |
| **\cx** | 匹配由x指明的控制字符。例如，\cM匹配一个Control-M或回车符。x的值必须为A-Z或a-z之一。否则，将c视为一个原义的“c”字符 |
| **\d** | 匹配一个数字字符。等价于[0-9] |
| **\D** | 匹配一个非数字字符。等价于[^0-9] |
| **\f** | 匹配一个换页符。等价于\x0c和\cL |
| **\n** | 匹配一个换行符。等价于\x0a和\cJ |
| **\r** | 匹配一个回车符。等价于\x0d和\cM |
| **\s** | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v] |
| **\S** | 匹配任何非空白字符。等价于[^ \f\n\r\t\v] |
| **\t** | 匹配一个制表符。等价于\x09和\cI |
| **\v** | 匹配一个垂直制表符。等价于\x0b和\cK |
| **\w** | 匹配包括下划线的任何单词字符。等价于“[A-Za-z0-9\_]” |
| **\W** | 匹配任何非单词字符。等价于“[^A-Za-z0-9\_]” |
| **\x*n*** | 匹配*n*，其中*n*为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，“\x41”匹配“A”。“\x041”则等价于“\x04&1”。正则表达式中可以使用ASCII编码 |
| **\*num*** | 向后引用（back-reference）一个子字符串（substring），该子字符串与正则表达式的第num个用括号围起来的子表达式（subexpression）匹配。其中num是从1开始的正整数，其上限可能是99。例如：“(.)\1”匹配两个连续的相同字符 |
| **\*n*** | 标识一个八进制转义值或一个向后引用。如果\*n*之前至少*n*个获取的子表达式，则*n*为向后引用。否则，如果*n*为八进制数字（0-7），则*n*为一个八进制转义值 |
| **\*nm*** | 标识一个八进制转义值或一个向后引用。如果\*nm*之前至少有*nm*个获得子表达式，则*nm*为向后引用。如果\*nm*之前至少有*n*个获取，则*n*为一个后跟文字*m*的向后引用。如果前面的条件都不满足，若*n*和*m*均为八进制数字（0-7），则\*nm*将匹配八进制转义值*nm* |
| **\*nml*** | 如果*n*为八进制数字（0-3），且*m和l*均为八进制数字（0-7），则匹配八进制转义值*nm*l |
| **\u*n*** | 匹配*n*，其中*n*是一个用四个十六进制数字表示的Unicode字符。例如，\u00A9匹配版权符号（©） |

# 附录B MySql数据库事物隔离级别

在Mysql数据库调优中，总会提到数据库事物隔离级别，但很多测试人员不懂数，在这里进行简要说明。数据库事务隔离级别分为4中：

**1. Read Uncommitted（读取未提交内容）**

       在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。  
**2. Read Committed（读取提交内容）**

       这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别 也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select可能返回不同结果。  
**3. Repeatable Read（可重读）**

       这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

**4. Serializable（可串行化）**  
       这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

         这四种隔离级别采取不同的锁类型来实现，若读取的是同一个数据的话，就容易发生问题。例如：

         脏读（Drity Read）：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。

         不可重复读（Non-repeatable read）：在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

         幻读（Phantom Read）：在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

         在MySQL中，实现了这四种隔离级别，分别有可能产生问题如下表B-1所示所示：

表B-1 数据库隔离级别限制



1. [↑](#footnote-ref-0)
2. 2JDK最好保证为1.6以上版本，1.6以下版本在使用时候可能无法启动JMeter，JDK下载地址为http://www.oracle.com/technetwork/java/javase/downloads/index.html。 [↑](#footnote-ref-1)
3. 3 JMeter也适用于Linux系统，Linux下用JMeter.sh启动。 [↑](#footnote-ref-2)
4. 4时JMeter选择的语言为简体中文，如果不是简体中文大家可以在“选项”—“语言”中选择简体中文即可。 [↑](#footnote-ref-3)