　　本文重点介绍JMeter工具在测试中地位以及其中一些难以理解或者手册中含糊不清的感念，读者可以通过本文了解这些概念，然后再根据自己的需要查阅JMeter中各个组件的具体用法来完成测试工作。

　　1. 简介

　　JMeter是一个专门用于测试C/S应用的桌面测试软件（并不适合于B/S结构，因为它很难模拟用户在browser上的动作，如果需要测试B/S结构的应用，可以选择Selenium这样的工具），主要被用来完成功能测试，压力测试 ，性能测试等工作。

　　JMeter与其它测试软件相比的优势如下：

    它可以帮助测试者很方便地模拟出多用户同时访问服务器的环境（通过Thread Group），这样可以检测出很多平时在单线程环境下无法暴露出来的问题。  
    应用范围很广，几乎所有你能想到的C/S应用它都能够提供了相应的支。JMeter中自己定制了一些特定应用的测试方案，例如对HTTP Server的测试、对数据库的测试、对Java程序的测试等。此外即使Jmeter没有提供当前应用的测试环境，用户也可以同昨BeanShell的方式自行定制。  
    提供了丰富的逻辑控制器，可以允许测试人员很方便地写出一些相对复杂的测试逻辑。  
    提供完善的变量机制以及配置机制，帮助测试人员减轻编写用例的负担，减少重复工作。  
    提供了强大的监控组建，帮助测试人员很方便地得到测试结果统计信息。

　　JMeter的劣势：

    难以针对“正确性”进行测试。虽然JMeter提供了断言机制，但是通常我们的测试在模拟多用户操作，因此某个用户发出一个请求后得到的响应是不可预测的（例如同时对一个数据库表进行读写，虽然我们可以让每个模拟用户将写入的信息存储在某个公共区域，但仍然可能会有问题，因为数据库写入的时间和写入公共区域的时间并不能保证同步），因此如果想通过JMeter验证应用的正确性还是比较麻烦的。通常我们只是利用断言来检查一些较为简单而又重要的信息，例如返回码。  
    没有很好的BeanShell测试机制。在JMeter中，BeanShell是非常重要的一部分，因为通常JMeter定制的测试方案多少与我们的应用有些出入，这时就需要使用BeanShell来完成一些JMeter无法完成的工作。然而BeanShell也是需要保证正确性的，而JMeter并没有对BeanShell的测试提供很好的支持。  
    通过以上的分析，可以发现JMeter更适合找出被测试系统在并发环境下存在的问题。

　　2. JMeter测试用例的基本结构

　　JMeter测试用例的基本结构是一个类似于Windows资源管理器的树形结构，这个树中的每一个节点都由一个元素来表示，因此一个完整的JMeter测试用例实际上是由一个个元素组成的，而测试的执行过程实际上就是这些元素的执行过程。一般而言，JMeter会使用深度优先的方式遍历这些元素，而对于同一层的元素，JMeter会自上至下地执行。

　　在JMeter中有很多种元素，而每种元素在树型结构中都有特定的意义，为了方便理解，这里把这些元素从结构性质上分为三大类：

　　第一类是控制型元素，这类元素通常出现在树型结构的枝节点，它们被用来控制其下第一层元素（注意，不是其下所有元素）的执行，例如控制他们的先后顺序，或者执行哪个或者不执行哪个。通过这类元素我们就可以很方便地动态控制测试用例的执行过程，这些元素就类似于变成语言中的if-else、switch、while等逻辑控制语句。

　　第二类是动作型元素，这类元素是真正发起测试请求的元素，它们通常位于树型结构的底层，每一个动作元素代表一次请求-响应的过程，他们的执行顺序通常被控制型元素管理。

　　第三类是配置型元素，它们只能作为树型结构中的叶子节点，被用于对其作用范围内（作用范围的规则如下：如果该配置元素在控制元素下，则其作用范围为该控制元素下的所有子孙节点；如果该配置元素在动作元素下，则其作用范围仅为这一个动作元素）的所有动作型元素产生一定的影响。这些影响根据具体元素种类而不同，例如改变元素的参数、延迟请求时间、在请求前后加入一些动作、监听请求及其相应。此外，如果某个动作型元素被多个相同的配置型元素影响后，这些配置型元素的效果就会进行Merge，Merge的规则依照元素的功能类型不同而不同（详见第3节）。

　　3. JMeter中的元素

　　从功能上讲，JMeter的元素分为八大类以及两个特殊元素。这里从使用场合上对这些元素进行叙述，至于具体每个元素什么功能则需要查看JMeter的帮助（具体方法是点击未知的元素，然后选择Help菜单下的Help选项）。

    Test Plan元素：控制型元素。只能存在于树型结构的根节点。它代表了整个测试方案，测试人员可以在这里设置一些全局性的内容，例如全局变量（注意全局变量是Thread Local的，详见第4节）、ClassPath配置（如果希望在Jmeter中调用自己的Java类就需要在这里设置了）等。  
    Thread Group元素：控制型元素。只能存在于Test Plan元素之下。它代表了一组行为相似的用户，通常我们把一类用户的动作放在同一个Thread Group下，这样就可以模拟多这这样的用户了。在这里可以配置模拟用户的个数（线程的个数）、循环次数、执行时间等。  
    Logic Controller：控制型元素。可以存在于Thread Group下任何位子。它用来完成控制其下元素的执行，JMeter提供了很多Logic Controller类型的元素，方便我们在测试中实现基本的逻辑。  
    Config Element: 配置型元素。这些元素被用来改变其作用范围内所有动作元素的配置，利用该类元素可以减少很多测试用例编写中的重复工作，例如可以让一个HTTP Request Defaults元素来配置所有用例中HTTP请求的主机地址以及端口号，这样就无须在每个动作元素中都做这样的配置了。当Merge发生时，如果某个域只有一个Config Element元素有值，则使用该值；如果某个域有多个Config Element元素有值，则使用离动作元素最近的Config Element元素的值（在动作元素节点下的配置元素最近）。  
    Timer: 配置型元素。如果希望控制请求发出的频率，则应该使用Timer延迟这些请求。当Merge发生时，所有Timer的时间会相加。  
    Pre Processor: 配置型元素。用于根据一定条件修改所有被影响的动作元素。当Merge发生时，这些Pre Processor将被依次执行，离动作元素最远的先执行。  
    Post Processor: 同Pre Processor，只是发生在动作元素之后，用于从响应中提取需要的信息。  
    Assertions: 配置型元素。用于对其影响范围内的动作元素的结果进行断言，例如可以断言一些HTTP请求的返回码，如果不通过则系统会记录本次错误。Merge发生时，所有的Assertion都会被判断。  
    Listener: 配置型元素。用于监听其影响范围内所有动作元素，测试结果数据主要由该类元素产生，因此他们非常有用。Merge发生时，所有的Listener监控动作都会被执行。  
    Sampler: 动作型元素。代表一次请求-响应的过程，他们是测试用例中动作的发起者，是测试用例的主要元素。JMeter根据不同的应用预制了很多种动作元素，如果用户觉得仍然不够用甚至可以用BeanShell Sampler写自己的动作。

　　4. JMeter中的变量

　　有的时候我们希望发送成千上万个随机的请求，或者希望本次请求的内容依赖于前几次的请求，那么就需要使用JMeter中的变量（注意，是Variable不是Property），这样就可以用变量来配置每一个请求，这样就可以让同一个Sampler每次都能发出不同的请求。

　　使用变量时，首先必须注意的是，JMeter中的变量是线程独立（Thread Local）的，也就是说虽然我们定义了n个变量，但是在每个线程中都有这n个变量的镜像，他们之间相互独立，互不干扰。

　　当我们希使用变量的时候，首先需要创建所需的变量。在JMeter中创建变量的方式很多，一种途径是通过Test Plan定义全局变量（用于所有的Thread Group，它也是线程独立的），也可以通过Config Element中的User Defined Element来定义不同线程组的局部变量（注意，User Defined Element中定义的变量是用于整个线程组的，无论将这个元素放在哪里都会被应用于整个线程组，这是一个比较特殊的配置型元素），此外当我们在应用中对某个没有创建的变量赋值（后面会讲到赋值）时也会创建该变量。实际上JMeter的源码中是使用Map来实现变量的，因此这些性质也不难理解。

　　当我们希望在某个地方引用一个变量的时候，可以通过${变量名}的语法来获取变量的值。注意，如果这个变量没有被定义，则这个式子就会被当作普通的字符串。

　　修改某个变量值的方法有很多，可以通过BeanShell来修改，也可以通过JMeter中一些特定的元素来修改（例如CSV Date Config Element），还可以使用JMeter函数来定义修改某个变量（具体如何做，见后面的小节）。

　　通常情况下如果我们希望在每次循环中都发出不同的请求，那么可以将可能的请求内容放在一个文件中，并让CSV Date Config Element从中获取相应的值并交给变量，也可以通过BeanShell Sampler用脚本来自己定制变量的值（注意不能使用Pre Processor中的BeanShell PreProcessor来定制变量，Pre Processor是用来修改请求中的域的，这个动作发生在请求被创建以后。也就是说如果我们在BeanShell PreProcessor中定义了一个变量，然后写在请求域中，那么结果就是JMeter先看到了一个没有被赋值的变量，然后把这个${变量名}式子当作字符串处理，然后再执行BeanShell PreProcessor。这一点是很多人容易犯错误的地方），也可以使用Pre Processor直接修改请求中的域，还可以在请求域中写入一个JMeter函数，直接生成需要的值。

　　在有些应用中，我们希望下一个请求的内容依赖于之前的请求。那么我们就可以通过Post Processor将响应中的有用信息抽取出来，然后赋值给一个变量，以便下次使用。

5. JMeter中的属性

　　在JMeter中有Property的概念，他们通常有两种用处：首先他们代表了JMeter的配置信息（存在JMeter目录下的bin/jmeter.properties）；其次他们可以被用来作为MemCache使用，以便线程间能够通信（Properties的get和put方法是线程安全的）。Property的获取、定义、修改一般是通过JMeter函数来完成的，当然也可以通过强大的BeanShell。

　　通常当我们希望多个线程之间有某种依赖关系时可以使用Property，但是这样的需求并不多见，因为我们模拟的用户多半都是相互独立的，他们并不应该知道其他人在干什么。

　　此外我们还可以将一些常量配置在jmeter.properties文件中，以便在测试中随时使用

　　6. JMeter中的函数

　　有的时候我们需要执行一些简单的操作（例如生成一个随机数）而又不希望编写BeanShell的Code，那么可以考虑使用JMeter中内置的一些函数。这些函数的格式通常为${\_\_函数名(参数1，参数2...)}，他们有两种方式来返回自己的执行结果：一种是直接返回，也就是说JMeter会在执行该函数后用函数的返回结果来替换原来调用函数的字符串（例如假设${\_\_method(1，2)}返回world，那么如果我们在域中写hello ${\_\_method(1，2)}，则实际的结果是hello world）。另一种是通过参数（通常是最后一个）指定一个变量后，JMeter会将执行结果存入该变量。具体使用哪种方式获得结果是根据不同的函数而定的，JMeter提供了一个强大的函数生成器（Options菜单中的Function Helper Dialog选项），其中列出了所有的方法以及可能的函数，并且有方便的帮助文档，用户可以通过该生成器来生成所需的函数。

　　这里有必要再次强调JMeter读入一个域（任何一个可以填写内容的空格）的过程：当JMeter读入一个域中的字符串后，会首先查看其中的是否存在能够匹配${...}的字符串，如果有则递归地解析这个${...}中的字符串，直到括号中的字符串不再包含括号为止。然后再解析这个${...}是否为一个函数或变量，如果是则用其结果覆盖原来的${...}字符串，直至将整个字符串解析完毕。例如在输入栏中输入hello ${\_\_method(${num}，2)}，而num变量的值为1，则这个字符串首先被转化为hello ${\_\_method(1，2)}，然后由于${\_\_method(1，2)}的值为world，则最终这个域中的内容为hello world。

　　掌握了以上内容后基本就可以将JMeter中的函数和变量运用自入了，而读者所需做得事情只是查看手册了解那些函数能够提供何种功能。（注意有些函数是不能放入一些特殊的域中的，例如${\_\_threadNum}就不能放在Test Plan的变量定义或者User Defined Element中的变量定义域中，具体原因手册上讲的很明白）

　　7. JMeter中的BeanShell

　　在多数情况下，JMeter提供的功能是不够我们使用的，我们的测试用例中可能会存在一些比较复杂的逻辑，而这些逻辑又不发通过简单的函数来实现，那么我们就必须动用强大的BeanShell了（JMeter有一些元素专门用于在测试的不同地方加入BeanShell脚本，如BeanShell Sampler）。不幸的是，JMeter的手册中并没有介绍BeanShell如何使用，而是把责任全部推给了BeanShell的网站，BeanShell的网站中的确有完整的BeanShell使用手册，但是我们总不希望为了做测试又去学习一个脚本语言，因此这里给出了一些BeanShell的简单应用，如果觉得不够的话再去查看更加复杂的应用。

　　7.1 BeanShell快速上手

　　BeanShell是面向Java的脚本语言，因此如果你想在完全不会BeanShell的前提下使用他，那么直接编写Java code就可以了。唯一需要注意的是，BeanShell支持若类型的变量，也就是不用指定变量类型，只要给他赋值就好了，BeanShell知道这个变量的类型。无论这个若类型变量在哪里被使用，其后的代码都能访问这个变量，这与强类型俄变量不同。例如：

　　view plaincopy to clipboardprint?

　　// Arbitrary code block

　　{

　　y = 2;      // Untyped variable assigned

　　int x = 1;  // Typed variable assigned

　　}

　　print( y ); // 2

　　print( x ); // Error! x is undefined.

　　// Arbitrary code block

　　{

　　y = 2;      // Untyped variable assigned

　　int x = 1;  // Typed variable assigned

　　}

　　print( y ); // 2

　　print( x ); // Error! x is undefined.

　　7.2 JMeter内置变量

　　JMeter在它的BeanShell中内置了变量，用户可以通过这些变量与JMeter进行交互，其中主要的变量及其使用方法如下（JMeter文档并没有对该部分内容进行详细讲解，这里也会说明他们分别对应于JavaDoc中的哪个类）：

    vars：这个变量实际引用了JMeter线程中的局部变量容器（本质上是Map），因此可以通过put和get方法访问JMeter中的变量。这个变量是所有内置变量中最有用的，它是测试用例与BeanShell交互的桥梁。对应于org.apache.jmeter.threads.JMeterVariables  
    props：该变量引用了JMeter的配置信息，它的使用方法与vars类似，但是只能put进去String类型的值，而不能是一个对象。对应于java.util.Properties。  
    ctx：该变量引用了当前线程的上下文，理论上通过这个东西我们几乎可以控制当前线程相关的一切，不过这要求使用者非常熟悉JMeter的源码。对应于org.apache.jmeter.threads.JMeterContext。

　　7.3 在BeanShell中使用Java类

　　有的时候我们希望发送数个请求，而这些请求的内容是通过一个Java类来实现的，这时就需要让JMeter来调用这个Java类，而唯一的途径就是通过BeanShell。

　　为了能够加载Java类，我们首先需要在Test Plan中添加自定义Java类所在的Class Path，然后就可以在BeanShell中import这个Java类，最后再将获得的内容放入vars变量中，这样就可以在其他地方通过${...}的方式获取该变量的信息了。

　　例如，创建了一个com.linhao.A类在/root/Java文件夹下，该类有一个方法hello()返回一个"Hello World"字符串，那么如果想在请求中使用这个方法返回的字符串，则进行如下操作：

　　首先在Test Plan下添加ClassPath为/root/Java

　　然后在请求前添加一个BeanShell Sampler并写入如下代码：

　　view plaincopy to clipboardprint?

　　import com.linhao.A;

　　vars.put("word"， A.hello());

　　import com.linhao.A;

　　vars.put("word"， A.hello());

　　最后在需要使用该字符串的地方写${word}，这样最终这个域将被替换为Hello World

　　7.4 JMeter下BeanShell的调试

　　BeanShell也是代码，应此也可能会有错误，而JMeter又是一个图形界面的程序（也有命令行模式，但并不变于在设计用例阶段使用），因此很多时候用例的设计者并不能确定BeanShell中某些变量是否正确。一个简单的办法是使用BeanShell创建一个对话框，然后将需要检测的变量值显示在对话框中。代码如下：

　　view plaincopy to clipboardprint?

　　JFrame. frame. = new JFrame( a ); //a is a variable to be checked

　　frame.setVisible(true);

　　JFrame. frame. = new JFrame( a ); //a is a variable to be checked

　　frame.setVisible(true);

　　这样当测试执行到这里的时候就可以知道变量a的值了。

　　虽然BeanShell很强大，但是它毕竟是一个脚本语言，因此如果测试参数的生成逻辑很复杂，则还是应该把主要逻辑放在Java类中，然后让BeanShell去调用，BeanShell只用来处理一些较为简单的操作

8、Jmeter使用之常用函数介绍

"\_Random"函数

　　功能：这个函数是从某个数字段随机读取数据替换参数，可以利用在测试需要添加多条数据记录而且某些字段需要唯一性的测试脚本中，随机生成的参数是数字

　　作用：可以用来实现参数化http请求发送的参数，使得在Jmeter运行时参数化了的参数在每个线程去不同的随机数。如果某个参数每次发送请求的时候不能重复才可以存储到数据库中，则用这个功能十分方便。

　　使用：

　　1.点击Jmeter的“选项”，选择“函数助手对话框”（或者使用快捷键“Ctrl＋F”），在“选择一个功能”的下拉框中选择“\_Random”。

　　2.配置“\_Random”函数，第一个参数是“一个范围内的最小值”，即所要取的随机数的最小值，我们设置成1；第二个参数是“一个范围内的最大值”，即所要取的随机数的最大值，我们设置成100；第三个参数是“函数名称”，即用于存储在测试计划中其他的方式使用的值，我们设置成Random。设置好上面的三个参数后，点击“生成”按钮，这样就会在对话框的最下面生成一个字符串“${\_\_Random(1，100，Random)}”，在我们编写的脚本中，找到要替换的参数，把它的值换成前面生成的字符串就可以了，然后每次运行的时候，这个参数会变成一个1到100之间的随机数。

　　"\_counter"函数

　　功能：这个函数是一个计数器，用于统计函数的使用次数，它从1开始，每调用这个函数一次它就会自动加1，它有两个参数，第一个参数是布尔型的，只能设置成“TRUE”或者“FALSE”，如果是TRUE，那么每个用户有自己的计数器，可以用于统计每个线程歌执行了多少次。如果是FALSE，那就使用全局计数器，可以统计出这次测试共运行了多少次。第二个参数是“函数名称”

　　格式：${\_\_counter(FALSE，test)}

　　使用：我们将“\_counter”函数生成的参数复制到某个参数下面，如果为TRUE格式，则每个线程各自统计，最大数为循环数，如果为FALSE，则所有线程一起统计，最大数为线程数乘以循环数

　　"\_time”函数

　　功能：Jmeter运行时取当前时间到变量，利用该功能，可以将某个参数增加time函数，然后用该参数作为查询条件查询，然后以该参数作为断言，这样可以使得断言更精确，因为时间实时变化的，使用该方法，需要注意的是，要先添加一个全局的用户参数，具体在断言操作中有描述。

　　格式：${\_\_time(MMddmmss，TEST)}

　　使用：添加jmeter的time函数，选择选项——函数助手，然后选择\_\_time函数，如下图所示：

　　在“ormat string for SimpleDateFormat (optional)”中输入值“MMddhhmmss”，表示取月日时分秒。

　　然后，点击生成，则生成了time参数。

　　"\_intSum”函数

　　功能：用于计算多个整数的和，可以是计算正整数和负整数的和，它有N个参数，最少有3个参数，最多不限。最后一个参数是函数名称，前面的其它参数是要求和的整数。这个函数在函数对话框中只显示3个参数，如果要计算多个整数，可以通过添加参数实现，不过最后一个参数一定要是函数名称。再添加的参数会在函数名称后面，这个时候，需要我们手动将函数名称参数放到最后一个。

　　格式：${\_\_intSum(1，100，test)}  
StringFromFile

1.函数： ${\_StringFromFile(文件名.dat)}

2.外部文件的格式必须为.dat

3.默认读取位置为/bin下，也可以自己设定文件存放路径，如： ${\_StringFromFile(e:user.dat)}，从e盘下读取user.dat文件中的数据

4.每次读取文件中的一行

5.如果我有多个user文件，想一起读取，文件名分别为user1  user2。函数写为：${\_StringFromFile(user#.dat，，1，2)}，从默认目录下读取user1和user2的文件

6.如果一个文件想读取多次，可以写成：${\_StringFromFile(user.dat，，，2)}，user1文件读取2次(来源 www.iocblog.net)