目 录

扼杀 QTP 检查点 （连载二）

作者：卢晨之

摘　要：本文主要写了 QTP 检查点在 QTP 实际应用中问题与解决方法。

关键词：检查点

在做自动化脚本录制与回放过程中，只有当用户针对某个功能做的检查点，QTP 才会把这个对象的预期结果做判断，给着人一种“各扫门庭雪”的感觉。或者有些地方会出现类似 Http 错误之类的提示，或者页面出现某些错误但 QTP 却无动于衷，因为你没有对这些做监测点的设置。但这么多的对象做监测点设置是件很麻烦的事情，而你不做，却又会对项目在做验收时候看不到多大的效果。为此我们结合了上节我们的方法，就是自己编写简单的函数对这个页面做判断。

哪么，一个 web 页面出现错误的情况大概都有那些？读者可以自己在脑海中想想在自己的测试工作过程中，都发现了那些类型的页面错误？结合到这点简单的提到一下几点：

1、标题提示错误。这个应该是给用户最直观的。“找不到页面”、“HTTP 错误” 、“找不到服务器”等等；如图-01：

图（01）

2、URL 错误。它涉及到的，可以是当前页面的 URL 错误，或者是 Frame 内部的 URL 错误。错误的标准可以是公司内部自己提供的错误后跳转的 URL，只要出现这个 URL 跳转出来的页面，哪么它也是错误的；如图-02

图（02）

3、页面输出错误。这点是比较泛的，区别于第 2 点就是某些错误的页面它是不会跳转到我们指定的 URL，导致了显示出错。如图-03

图 （03）

4、WinStatusBar 错误提示。这一点在测试的过程中，很多测试人员或者开发人员对这个很不敏感，有些错误不易见的错误我们是可以在这个 WinStatusBar 中看出来的。如图-04

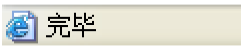


图 （04）

就先说这 4 点，在看到这 4 点后，读者可以先试试如何把这 4 点写成函数。

首先，针对这个错误的比较，我们的前提是错误类型字眼的出现，为此我们先编写一个判断正则表达式的函数。

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' RegExpbld  正则表达式的判断
'
'参数
' patrn                    正则表达式
' strng                    字符串
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function RegExpbld(patrn, strng)
      Dim regEx, retVal                    ' 建立变量。
      Set regEx = New RegExp               ' 建立正则表达式。
      regEx.Pattern = patrn                ' 设置模式。
      regEx.IgnoreCase = true              ' 设置是否区分大小写。
      retVal = regEx.Test(strng)           ' 执行搜索测试。
      RegExpbld=retVal
End Function
```

其次，第 1，2，4 点的函数编写实现难度不高。好比第 1 点中，我们只需要取出 Browser 的标题，并使用上面的表达式函数便可以。

```
RegExp = "错误|error|无法"      '这个是我们简单编写的正则表达式
title=browser("Google").getroproperty("title")
if RegExpbld(RegExp, title) then
    '如果出现这些错误的字样则报错
end if
```

但在第 2 点中，需要提醒大家的是，捕获这个错误的 URL，不仅仅是当前页面的 URL，我们还需要取出在页面中所有 frame 的 URL 做对比。

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' 参数
'error_url          公司内部自定义的错误页面跳转 URL，可以写成数组之类去进行比较
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim FramesObj
Set FramesObj =browser("Google").Page("Google").Object.frames
For i=0 to FramesObj.length-1
        If FramesObj (i).getroproperty("url")= error_url Then
            '如果为 true 就是某个 frame 已经跳转到指定的 URL
      End If
Next
```

再次，我们针对第三点，我们查错的标准是当前页面中是否出现了类似第 1 点中的错误字样："错误|error|无法"。但是如何实现呢，在这里简单介绍 3 个方法。

第一：通过 QTP 自带的 TextUtil 对象去做页面字体捕获

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' QTP_DocIsLive  通过 TextUtil  对象判断页面是否存在某个特定错误字样
'
'参数
'无
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''


Public Function QTP_DocIsLive()
	Dim Src_text,patrn
	Src_text=TextUtil.GetText(0, 20, 20, 2000, 2000)	'窗口的大小需要根据实际去定义
	patrn="错误|error|无法"
	If RegExpbld(patrn,Src_text) Then
		QTP_DocIsLive=true
		Exit Function
	End If
	QTP_DocIsLive=false
End Function
```

优势：原理简单，甚至能查找到不可预知的 msgbox 中的内容。

弊端：QTP 本身对这个 TextUtil 的支持并不是做得很完善，导致了方法是使用过程中，偶尔会与 Window Explorer 冲突，导致了使用函数完后，Window 会出现某些异常情况。

第二：通过浏览器自带的查找功能。

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' New_Page_Docfind  通过 IE  自带查找功能判断页面是否存在某个特定错误字样
'
'参数
' New_Page_Doc      用户自己输入的查找字样，不支持正则表达式，需要自己编写循环查找
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''


Public function New_Page_Docfind(New_Page_Doc)
    Dim oShell
    Set oShell=CreateObject("WSCript.shell")
    oShell.sendkeys    "^f"          '发送”Ctrl+F”调出查找功能
    browser("browser").Window("查找").Page("查找").WebEdit("WebEdit").Set New_Page_Doc
      '设置搜索文本的内容，用户需要自己添加对象或者修改成描述语言
    browser("browser").Window("查找").Page("查找").WebButton("查找下一个(F)").Click
'点击查找按钮
    If    browser("browser").Window("查找").Dialog("Microsoft Internet Explorer").Exist    Then
            browser("browser").Window("查找").Dialog("Microsoft Internet Explorer").Close
            wait(0.1)
            oShell.sendkeys    "%u"
            browser("browser").Window("查找").Page("查找").WebButton("查找下一个(F)").Click
            If browser("browser").Window("查找").Dialog("Microsoft Internet Explorer").Exist    Then
                browser("browser").Window("查找").Dialog("Microsoft Internet Explorer").Close
                New_Page_Docfind=false
                browser("browser").Window("查找").Close
                else
                New_Page_Docfind=true
                browser("browser").Window("查找").Close
            end if
    else
            New_Page_Docfind=true
            browser("browser").Window("查找").Close
    End If
    Set oShell=Nothing
End Function
```

优势：针对某个特定字眼查找速度快，准

弊端：如果特定字眼多，会导致速度下降，界面闪动较为厉害

函数中的某些对象是建立在对象库的基础上的，移植效果差强人意。

不能同时对多个 Frames 做查找，需要点击触发

第三：使用 DocumentElement 对象。

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' New_Page_DocExist 使用 DocumentElement 对象判断页面是否存在某个特定错误字样
'
'参数
' reg       正则表达式
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Public Function New_Page_DocExist(reg)
Dim Page_doc_a,Page_doc_b,Page_doc_c,str,oDesc,frame_obj,logie,child_a
      str=browser("browser").GetROProperty("title")
      If browser("browser").page("index:=0").Exist Then
Set Page_doc_a=browser("browser").page("index:=0").Object.documentElement.all.tags("body")
            For each Element in Page_doc_a
                str=str+"    "+Element.outertext
            Next
        SetPage_doc_b=browser("browser").page("index:=0").Object.documentElement.all.tags("input")
            For each Element in Page_doc_b
                str=str+"    "+Element.value
            Next
            Set Page_doc_c=browser("browser").Page("index:=0").Object.documentElement.all.tags("SPAN")
            For each Element in Page_doc_c
                str=str+"    "+Element.outertext
            Next
            Set Page_doc_a=nothing
            Set Page_doc_b=Nothing
            Set Page_doc_c=nothing
      End if
        frame_count=browser("browser").page("index:=0").object.frames.length
    If frame_count<>0 Then
            Set oDesc = Description.Create()
            oDesc("html tag").Value = "iframe"
            set frame_obj = browser("browser").page("index:=0").ChildObjects(oDesc)
            For i=0 to frame_count-1
            Set Page_doc_a=frame_obj(i).Object.documentElement.all.tags("body")
                    For each Element in Page_doc_a
                            str=str+"    "+Element.outertext
                    next
                    set Page_doc_b=frame_obj(i).Object.documentElement.all.tags("input")
                    For each Element in Page_doc_b
                            str=str+"    "+Element.value
                    Next
```

```
（接上页）                set Page_doc_c=frame_obj(i).Object.documentElement.all.tags("SPAN")
                          For each Element in Page_doc_c
                                    str=str+"    "+Element.outertext
                          next
                          set Page_doc_a=nothing
                          set Page_doc_b=Nothing
                          set Page_doc_c=nothing
                Next
    End If
    str=str+"    "+browser("browser").WinStatusBar("msctls_statusbar32").getroproperty("text") ' WinStatusBar的对象需要自己添加进对象库
    New_Page_DocExist=RegExpbld (reg,str)
End Function
```

优势： 方法编写能把标题捕获，状态栏捕获也加进去

捕获的字样比较全面，较为准确、可移植强

弊端： 使用正则表达式做判断时候时间较长。

看完这 3 个方法后，可能有人会有疑问，那究竟这样真的能够抓到错误吗？在这里，我们只需要再编写多一个截图的功能，并且在跑动脚本过程中只要出现我们定义的“错误”就截图，为的是宁杀错，不放过的精神。值得一提的是，这查找“错误”的方法也适合测试人员在做翻译查找中。因为许多的多语言版本的软件，在翻译方面或者会有漏网之鱼。

细说 LoadRunner 参数化

作者：黄文高

前言

为什么这里说是细说 LoadRunner 参数化，在书和网上到处都能找到关于 LoadRunner 参数化的内容，但是细心的读者不能难发现，虽然现在很多资料都有关于参数化的内容，但写的都不够详细，对于初学者来说是一件很困难的事，而参数化又是编辑脚本最重要的一部分之一，没有学好参数化就不能算是一名合格的性能测试工程师，因此，在这里我将自己理解的关于参数化的内容写出来和初学者共享，希望这份资料对大家学好参数化部分的知识有帮助。

首先：为什么要对脚本进行参数化

a) 为了减少脚本的大小和脚本数量，借助参数化我们可以减少脚本的数量，如果不进行参数化，我们为了达到目标可能要拷贝并修改很多个脚本。

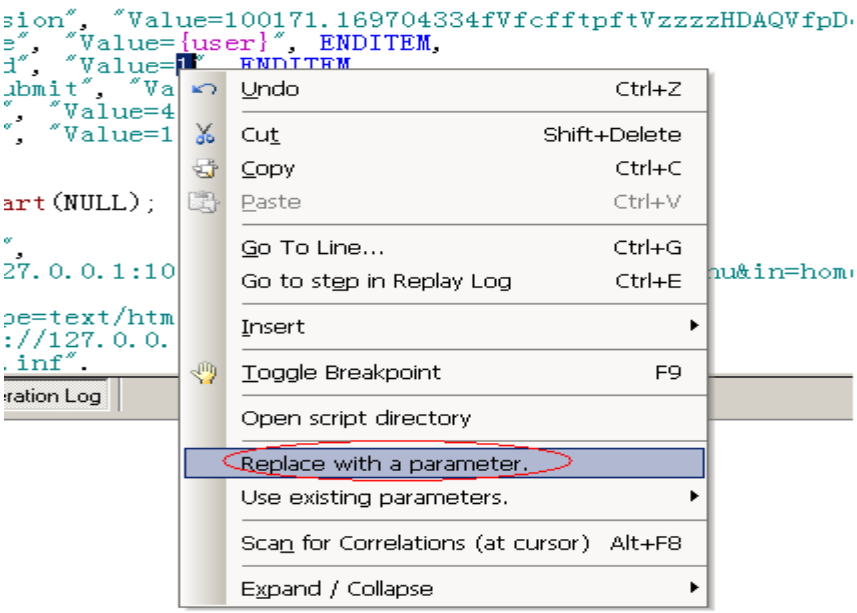
b) 使业务更接近其实的客户的业务，每个虚拟用户使用不同参数值来模拟这样才接近客户的实际情况。

第二：怎么进行参数化

首先在这里先声明一下，下面所有使用的例子都是录制 LoadRunner 中自带的那个例子的注册过程。

这里包括两部分的部分：

a) 编辑脚本，使用参数代替常量；



b) 设置参数的属性和数据源；

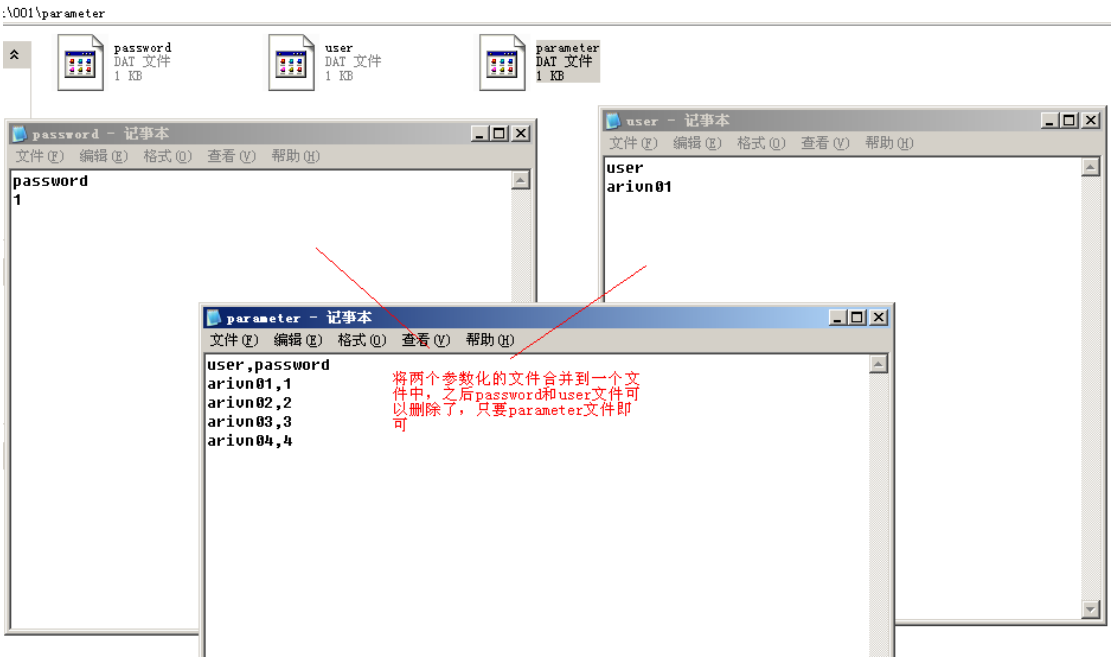
那么如何进行参数化呢？选中要参数化的内容点右键->Replace with a

parameter（如下图）。输入参数化的名称，假设为 password。

这时要我们要注意的一个问题是，当参数化结束后，脚本保存的根目录下会多出一个参数化的文件：

图中多出两个参数化的文件（password 和 user）就是刚才对两个数进行参数化后的文本文件，当然一般的情况下我们不要将这个参数化的文件放到脚本的目录下，而应该是放到一个专门的文件下，这样可以保证参数化文件与脚本分离，如我们新建一个文件夹 parameter，将所有参数化的文本文件都放到这个文件夹下。

这里我们只有两个参数化文件，那么当有很多参数化文件怎么办呢，因为当一个项目很大时，其录制的业务很多时，参数化文件会很多，甚至上几百 MB 时，这时为了方便管理参数化文件和节约空间我们会对参数化文件进行合并到一个文件夹中，如上面两个参数化文件就可以合并，参数化之间用逗号隔开即可，如下图合并好后的参数化文件。



再看一下参数化的属性：

a) 参数类型属性：

1. "Date/Time"（日期/时间）参数类型：

"Date/Time"类型用当前的日期和/或时间替换参数。要指定日期/时间的

格式，可以从菜单列表中选择，或者指定实际需要的格式。该格式应该与脚本中录制的日期/时间格式相对应。还可以单击该对话框中相应的按钮对格式进行添加、删除、还原等操作。

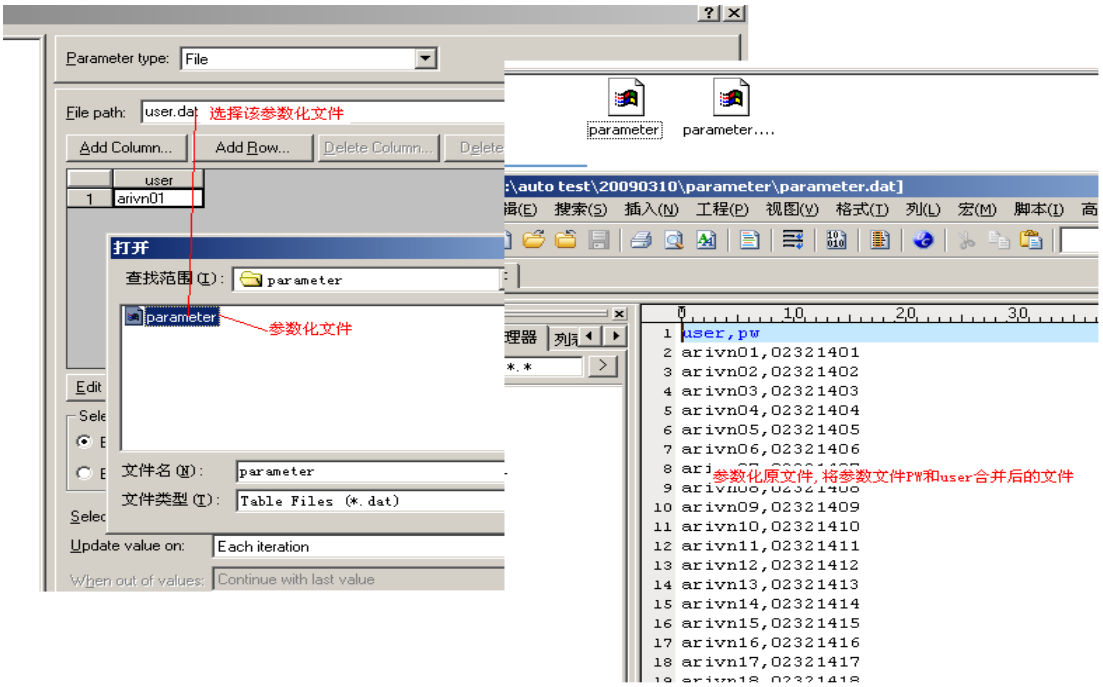
2. "Group Name"（组名）参数类型：

   用 Vuser 组的名称替换参数。创建方案时，要指定 Vuser 组的名称，否则运行 VuGen 的脚本时，组名始终为"无"。但在 VuGen 中运行时，Group Name 将会是 None。

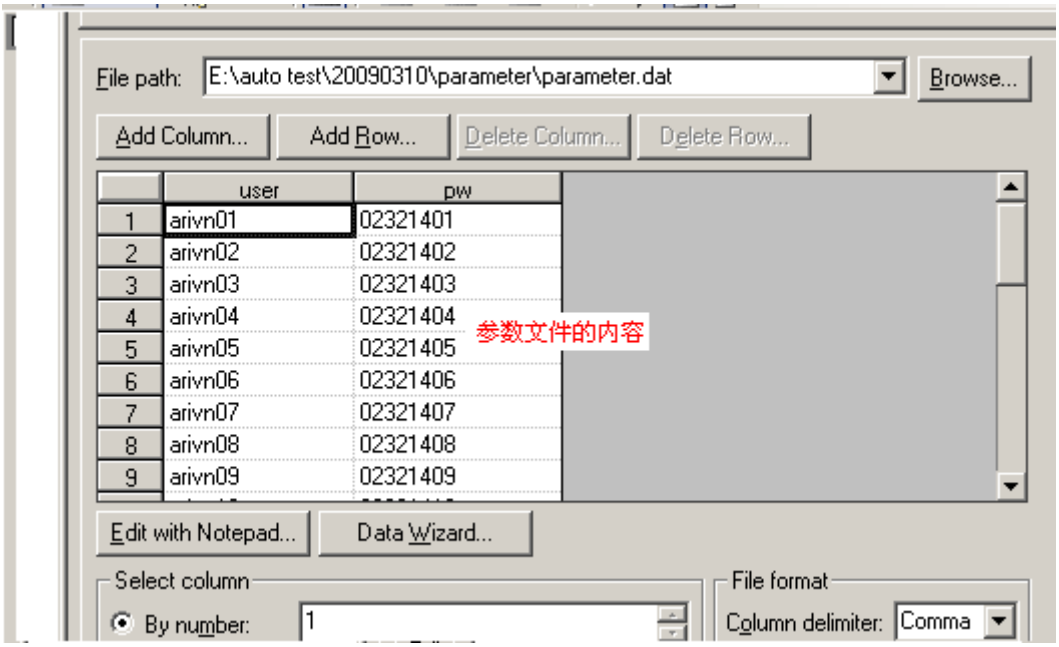
3. "Iteration Number"（迭代编号）参数类型：用当前的迭代编号替换参数。
4. "Load Generator Name"（负载生成器名）参数类型：用 Vuser 脚本的负载生成器名替换参数。负载生成器是运行 Vuser 的计算机。
5. "Random Number"（随机编号）参数类型：用一个随机生成的整数替换参数，可以通过指定最小和最大值，设置随机编号的范围。
6. "Unique Number"（唯一编号）参数类型：用一个唯一编号替换参数。"Block size"（块大小）指明分配给每个 Vuser 的编号块的大小。每个 Vuser 都从其范围的下限（start）开始，在每次迭代时递增该参数值。
7. "Vuser ID"参数类型：LoadRunner 使用该虚拟用户的 ID 来代替参数值，该 ID 是由 Controller 来控制。在 VuGen 中运行脚本时，VuGen 将会是-1。
8. File 参数类型：可以在参数属性中编辑参数文件，也可以直接编辑好参数文件通过路径来选择，还有从现成的数据库中提取。这是参数化最常的一种参数类型。

b) Browse 属性：

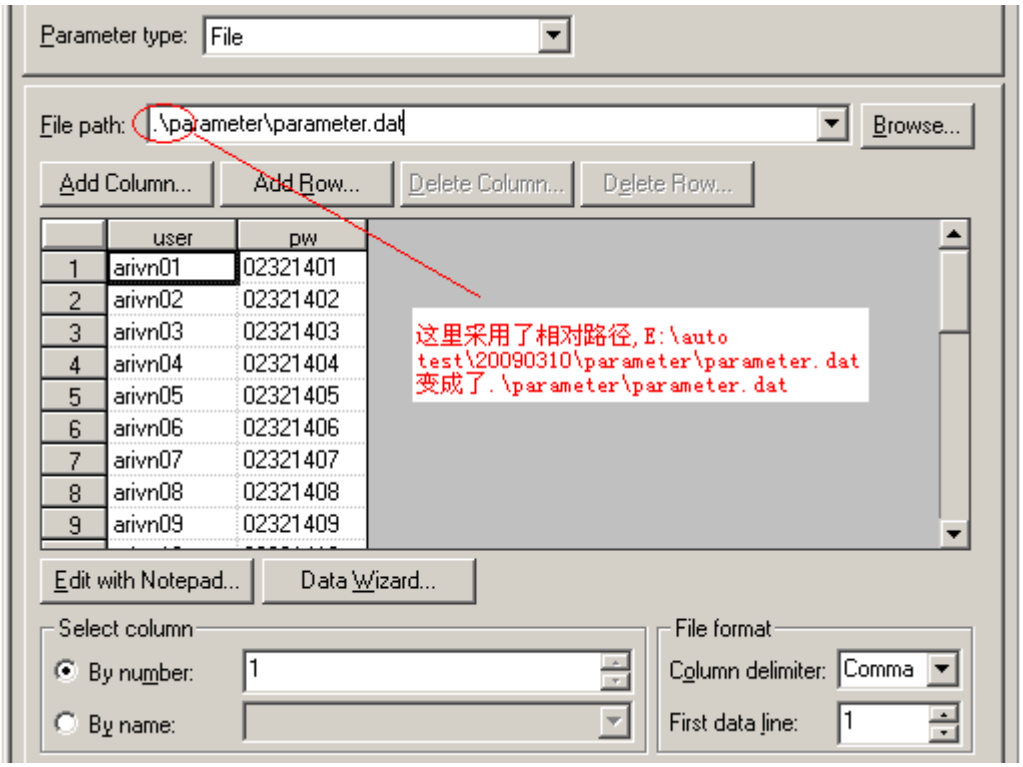
这里是用来选择参数文件的路径，这里要注意的一个问题是，一般我们在做参数化的时候没有单独把参数文件放到一个文件夹下，所以一般我们都没有更改过这块，但我们上面已经讲过，一般都会将参数化文件合并到一个文件下并放到一个专门管理参数的文件夹下，保证参数化文件与脚本分离，这样我们就要选择参数的路径，否则无法读到参数文件中的参数，具体的如下图。



选择好之后，会列出参数化文件中所有的项，如下图：



注意：读者可能会发现，这样如果我们把这个脚本拷贝到另外一台机器上去，这个路径不就出错了吗？也就是我们的脚本可移植性不好，对是的，会出错，因为这里写的是绝对路径，如果换到其它的一个盘或机器，运行就报错了，那么怎么解决这个问题？这里我们采用相对路径来解决这个问题，这是我们 Browse 设置为相对路径，将脚本的根目录使用“．”来代替。见下图，这样就不会出错了。



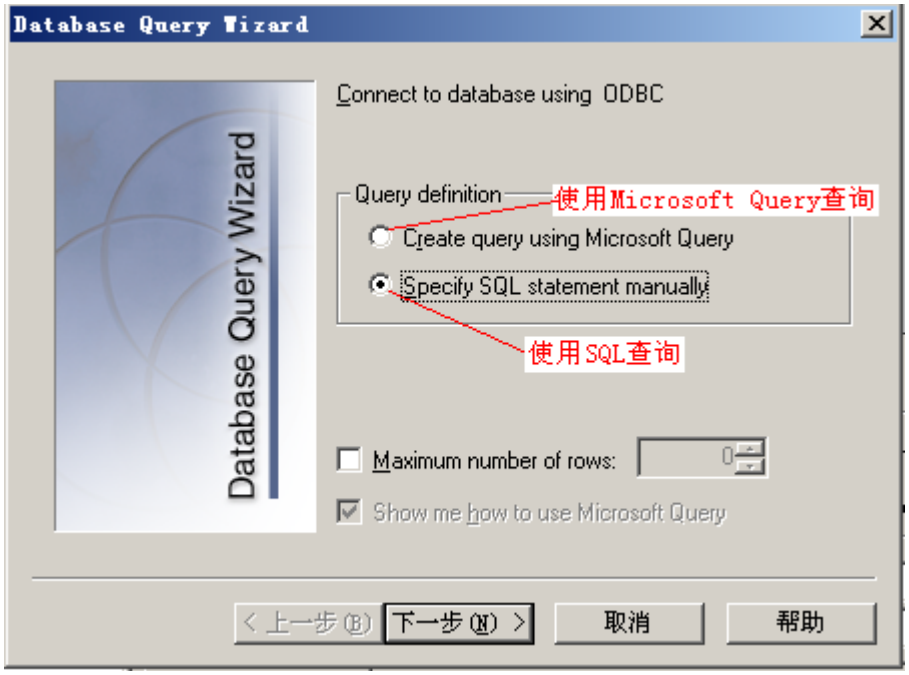
这样就解决了上面的那个问题，具体更好的可移植性了。

c) 导入数据方法：

LoadRunner 允许使用 Microsoft Query 或者指定数据库连接字符串与 SQL 语句，利用参数化从已存在的数据库中导入数据。在这里这两种导入数据的方法的区别在于使用 Microsoft Query 时，不要新建数据源，在导入向导过程中直接连接数据库，而手动指定数据库字符器，需要先做好数据源，现在一般都使用这种方法。

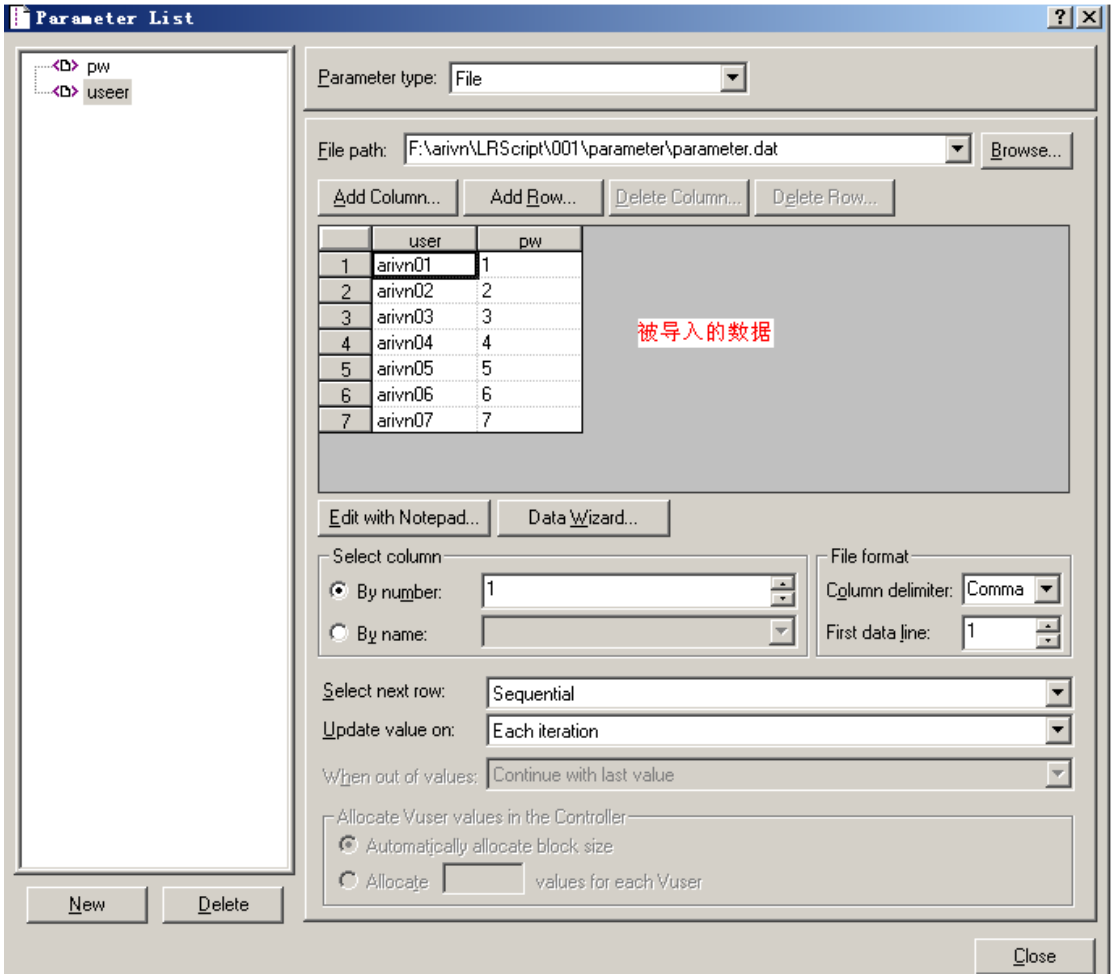
下面我们来看一下使用 Data Wizard 中的 Specify SQL statement manually 如何导入数据库中的数据。

首先建立一个数据源，在这里我个做一个数据源（具体如何建数据源不做解释），名称为 LR，点击 Data Wizard 按钮，选择使用 SQL 语句（如下图）。



接下来是选择数据源和写 SQL 语句（如下图）。

连接成功后，数据将会被成功的导入，下面我们看一下数据库中数据被导入到的情况。



d) Select next now 属性：

注意：这里要注意的是所有的 Select next now 属性选择是针对虚拟用户来说的，也就是这里的策略是针对 Controller 设置的，在调试脚本的过程中是看不出来的，其决定虚拟用户如何选择参数的过程。

- Sequential：虚拟用户 Vuser 按照行顺序的进行读取参数文件中的数据，如果参数文件中没有足够的数据，则返回到第一个值，并一直循环到结束。
- 例：如上图我们这里有 arivn01 到 arivn07 七个数据，假设我们有 10 个 Vuser，那么第 1 个 Vuser 读到的参数为 arivn01，于此类推，到第 8 个 Vuser 的时候，这里表中已经没有数据了，于是又从第一个数据开始读取，故第 8 个 Vuser 读到的数据是 arivn01，第 9 个 Vuser 读到的数据是 arivn02。
- Random：每个 Vuser 从表中随机的读起参数数据。
- Unique：唯一的数，即每个 Vuser 取到的参数均不一致，这里强调了用

户的差异性。

➢ Same link as ＊＊＊：如果一个脚本中定义了多个参数，而有一些参数应该是对应的关系，如上图中的用户名和密码就是对应关系，即密码应该始终和用户名对应，这时就要用到这个选项。

e) Update value on 属性：

注意：这里设置的策略是针对脚本的迭代来讲的，也就是说这里的一切策略其仅仅在脚本迭代次数发挥作用，而对 Vuser 选择参数没有影响。

➢ Each iteration：脚本每迭代一次都访问数据表中的下一个值。

注意：如果在一次迭代过程中，某个参数使用到多次，如下图这个例子中，在一次迭代中使用到两次用户名和密码，这两次使用的同一个数据，而并不是两个数据。

下面是参数化的代码：

```
web_submit_data("login.pl",
    "Action=http://127.0.0.1:1080/MercuryWebTours/login.pl",
    "Method=POST",
    "RecContentType=text/html",
    "Referer=http://127.0.0.1:1080/MercuryWebTours/login.pl?username=&passwor
    "Snapshot=t12.inf",
    "Mode=HTTP",
    ITEMDATA,
    "Name=username", "Value={user}", ENDITEM,
    "Name=password", "Value={pw}", ENDITEM,
    "Name=passwordConfirm", "Value={pw}", ENDITEM,
    "Name=firstName", "Value={user}", ENDITEM,
    "Name=lastName", "Value=", ENDITEM,
    "Name=address1", "Value=", ENDITEM,
    "Name=address2", "Value=", ENDITEM,
    "Name=register.x", "Value=74", ENDITEM,
    "Name=register.y", "Value=10", ENDITEM,
    LAST);
```

密码使用两次

用户名使用两次

下面我们来看一下编辑的结果

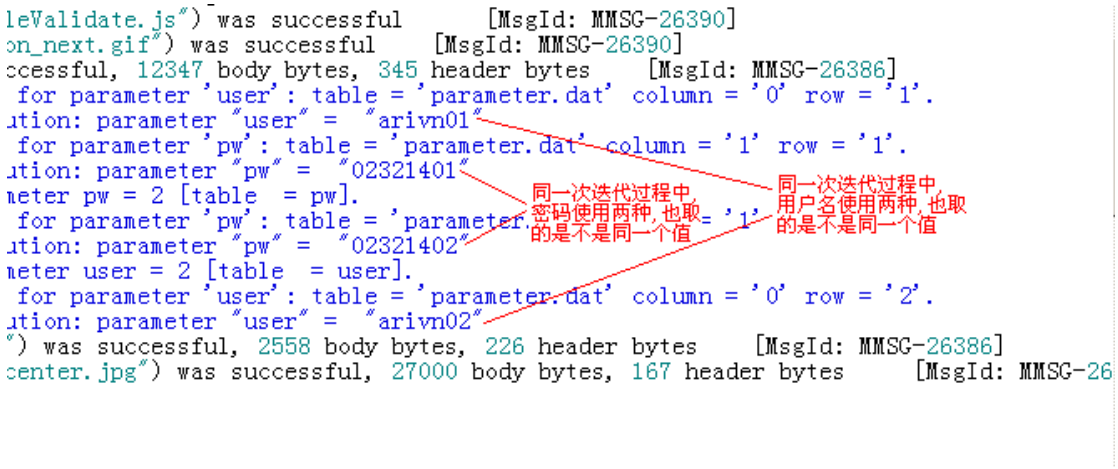
第一次迭代使用表中数据的结果

```
on.c(90): web_concurrent_start was successful    [MsgId: MMSG-26392]
on.c(92): Registering web_url("profileValidate.js") was successful      [MsgId: MMSG-26390]
on.c(100): Registering web_url("button_next.gif") was successful     [MsgId: MMSG-26390]
on.c(108): web_concurrent_end was successful, 12347 body bytes, 345 header bytes     [MsgId: MMSG-26386]
on.c(112): Notify: Parameter Substitution: parameter "user" = "arivn01"
on.c(112): Notify: Parameter Substitution: parameter "pw" = "02321401"
on.c(112): Notify: Parameter Substitution: parameter "pw" = "02321401"
on.c(112): Notify: Parameter Substitution: parameter "user" = "arivn01"
on.c(112): web_submit_data("login.pl") was successful, 2640 body bytes, 226 header bytes     [MsgId: MMSG-26386]
on.c(131): web_url("fma-performance-center.jpg") was successful, 27000 body bytes, 167 header bytes      [MsgId:
ng action Action.
ng iteration 1.
ting iteration 2.
fy: Next row for parameter user = 2 [table  = user].
```

这里在同一次迭代过程中，使用到的用户名和密码都是同一参数，这是第一次迭代

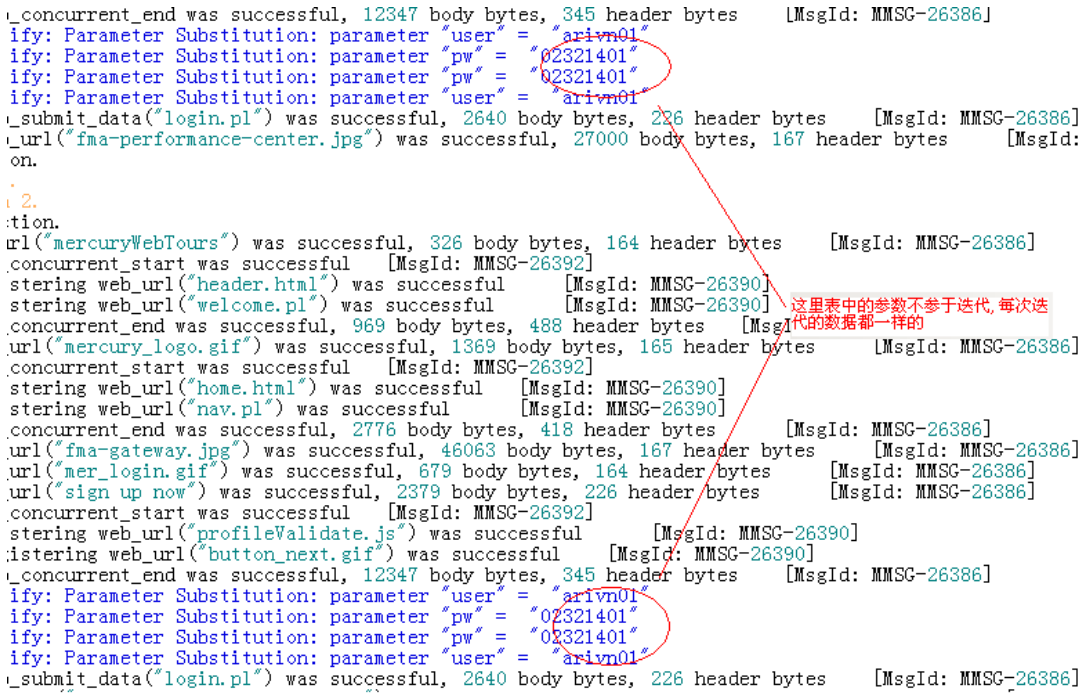
➢ Each occurrence：参数在每次迭代的过程中，参数的值都的更新。

注意：如果一个参数在一次迭代过程中出现多次，即使在同一次迭代过程中也得更新，下面同样看这个例子，其迭代的结果。



➢ Once：在同一个 Vuser 中一直取同一个参数，表中的数据不参于迭代的过程。

还是看我们上面的例子的结果：



到这里参数化的过程已经全部讲完，这里总结一下，参数化过程中要注意的问题：

1) 参数化文件尽可能少，因为参数是放在内存中的，占用了内存的资源；

2) 参数化文件与脚本分离；

3) 参数文件的路径应该以相对路径来取；

4) 一些时候为了使参数更具有真实性，参数应该从数据库中来获得；

5) 参数类型的选择；

6) 参数的数据一般要由业务决定；

后记：

参数化到这里已经彻底讲完了，主要涉及的内容是：

1) 为什么要进行参数化；

2) 如何进行参数化；

3) 参数化过程中要注意那些问题：

以前我也不是很理解到底参数是如何被调用的，现在终于搞定了，当初决定要写是因为很少有书写的这么细，一般都是 2 到 3 页搞定，这对初学者来说是一件很痛苦的事，这里我将我理解的参数化，再加于一些实例展现给大家，希望对大家的学习有帮助。

基于 webserivce 的性能测试方法

作者：刘英

前言

在过去的几年里已经有越来越的公司开始重视 webserivce，并且努力使他们的系统与其他操作系统在他们的防火墙内外进行交互，来使得战略伙伴之间工作得更加顺畅。

Webserivce 并不是一个新鲜的词汇，但是因为他是现在最流行的词汇---特别在 IBM，HP 和 Microsoft 在给他们新的服务器产品做广告的时候，都强调了 Web 技术。那么作为测试工程师我们应该怎样测试这些服务，使它在进入产品之前功能和性能得到满足了？

这篇文章将帮助通过 loadrunner 让你了解基本的 webserivce 功能测试，并且通过 webserivce 性能测试实例的引用，让你了解 webserivce 的基本性能测试方法。

Webserivce 测试需求分析

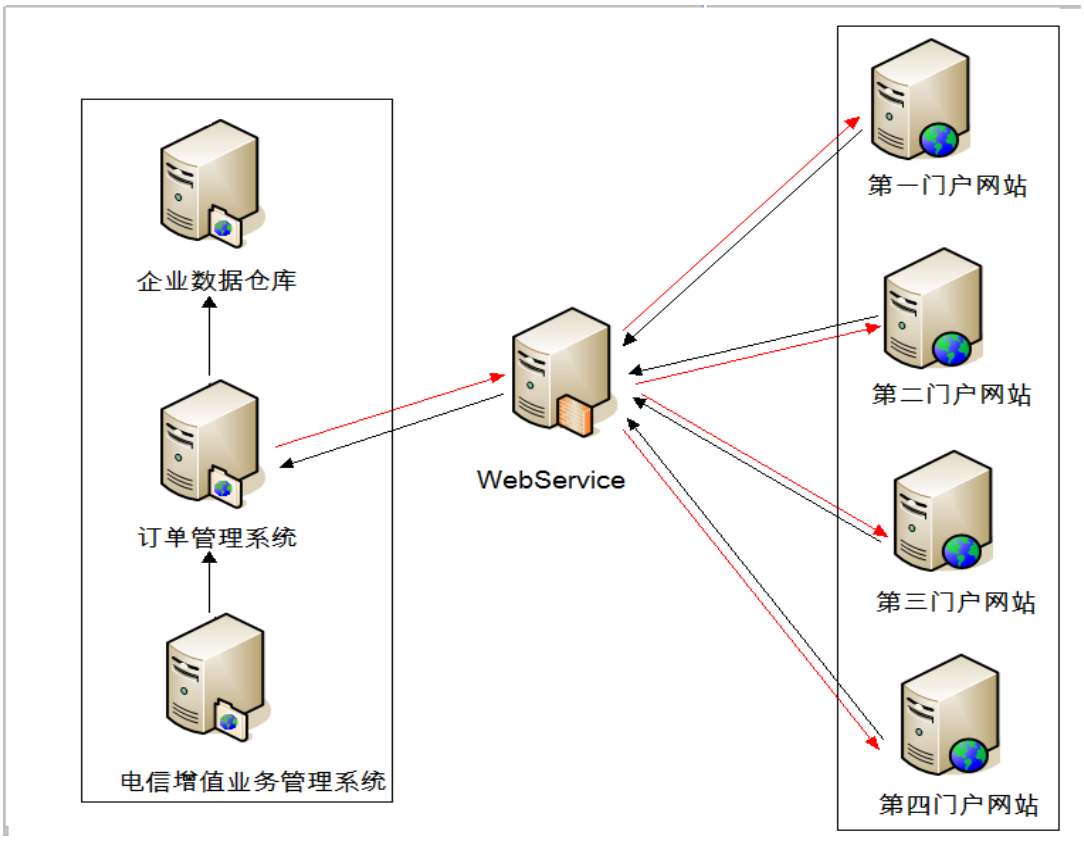


图 1　电信增值业务 webserivce 系统架构图

电信增值业务会通过两种方式来与用户进行交互，一种是通过客户端即-----门户网站来与用户交互，客户通过门户网站订购增值业务，客户端服务器调用电信的业务组件，而业务组件通过 webserivce 和服务端通讯，通过在客户端构建的用户对象，通常对所需的业务进行 soapheader 的验证后，再由 webserivce 传递到服务器端，服务器接收到数据后，对电信的后台服务器中进行业务处理，并将数据存于订单管理系统数据库中，并且将处理后的结果通过 webserivce 返回给门户网站。另一种方式是，客户打电话给客服请求订购增值业务，这时订单管理系统服务器端统一收集请求然后将业务发送给业务组件，那么客服服务器不需要 soapheader 的验证而直接通过 webserivce 和门户网站服务端通讯，来完成下单的处理，再通过业务处理后返回给 webserivce 处理结果，再返回给订单管理系统。

因此我们需要测试的 webserivce 分两种情况：

从门户网站到订单管理系统的 webserivce，带有 soapheader，需要进行并发性能测试

从订单管理系统到门户网站的 webserivce，不带有 soapheader，需要进行单线程的大数据量测试，一个请求接着一个请求的发送。

测试环境

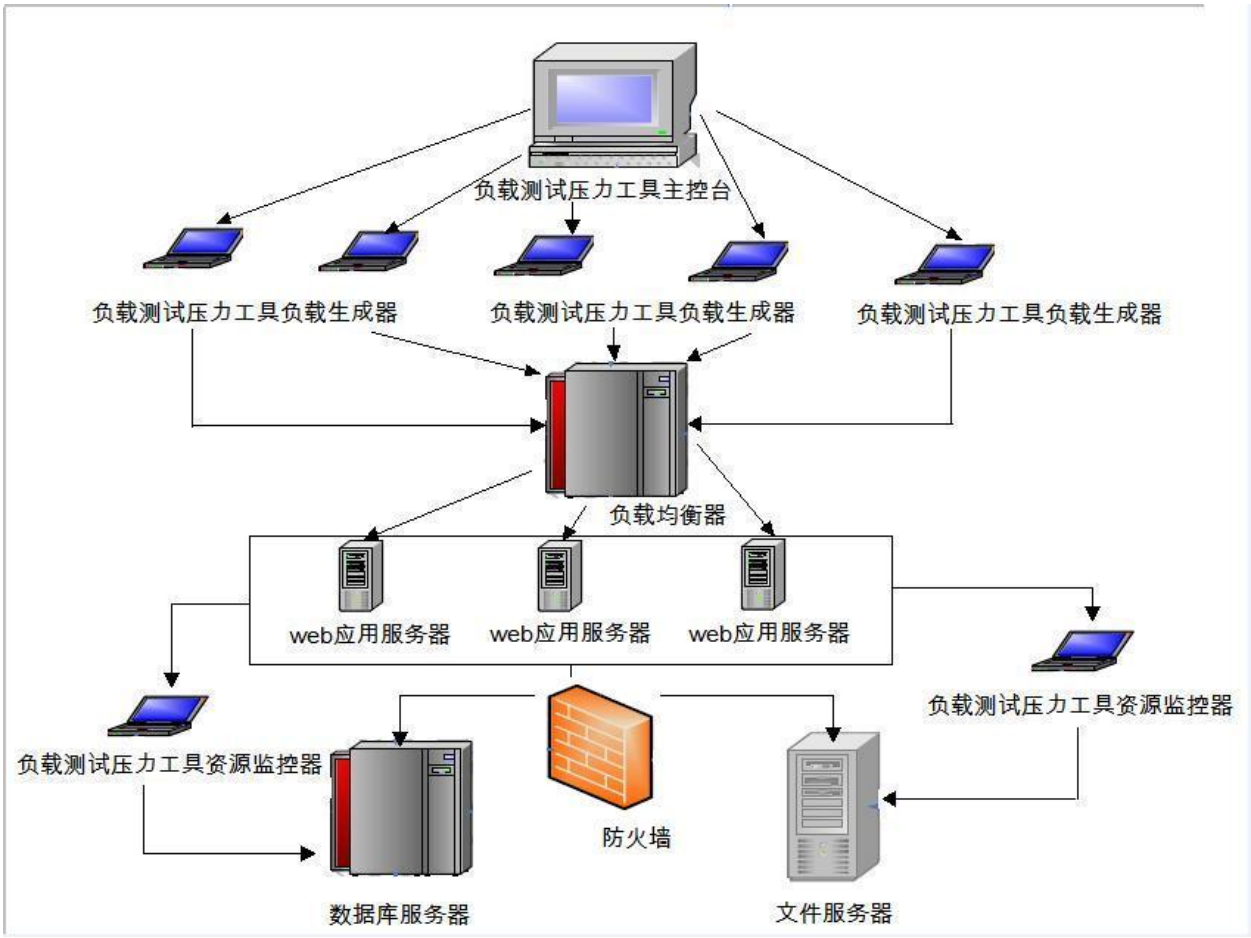


图 2　测试环境拓扑图

测试环境是属于负载均衡环境，并且存在三台 web 应用服务器集群情况下。

1.负载测试做并发测试时由主控台控制负载生成器与服务器集群链接在同一交换机上，压力由负载均衡模块分摊到三台 web 应用服务器上。在集群运行环境下应用服务器和一台数据库服务器和一台文件服务器连接。

2.负载测试做单线程测试时，自由一台测试机与发送给服务器集群链接在统一交换机上，再由压力负载分摊到三台应用服务器上。在集群运行环境下应用服务器和一台数据库服务器和一台文件服务器连接。

测试实施

我们测试的从门户网站到订单管理系统的 webserivce 的 URL：

http://192.168.1.8:4324/portal/getstatus.asmx?wsdl

request xml 为：

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xs
  <soap:Header>
    <tns:AuthenticationHeader>
      <tns:Username>user</tns:Username>
      <tns:Password>pass</tns:Password>
    </tns:AuthenticationHeader>
  </soap:Header>
  <soap:Body>
    <tns:GetServiceStatus>
      <tns:request>
        <tns:userID>b1mmnn45</tns:userID>
        <tns:serviceID/>
      </tns:request>
    </tns:GetServiceStatus>
  </soap:Body>
</soap:Envelope>
```

带soapheader，需要Server验证

Response xml 为：

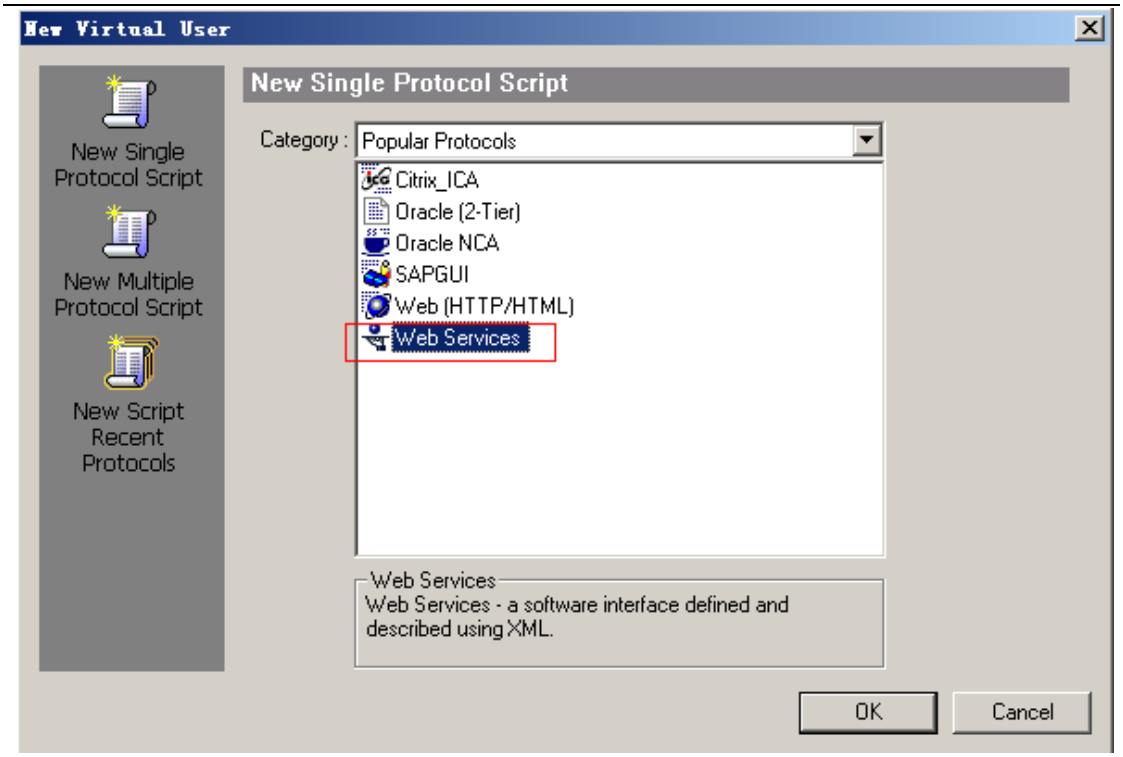
```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:x
  <soap:Body>
    <GetServiceStatusResponse xmlns="http://bell.ca/vas/getServiceStatus">
      <responseCode>0</responseCode>
      <responseDescription>Success</responseDescription>
      <services>
        <serviceID xsi:nil="true" />
        <serviceStatus>0</serviceStatus>
      </services>
    </GetServiceStatusResponse>
  </soap:Body>
</soap:Envelope>
```

返回成功

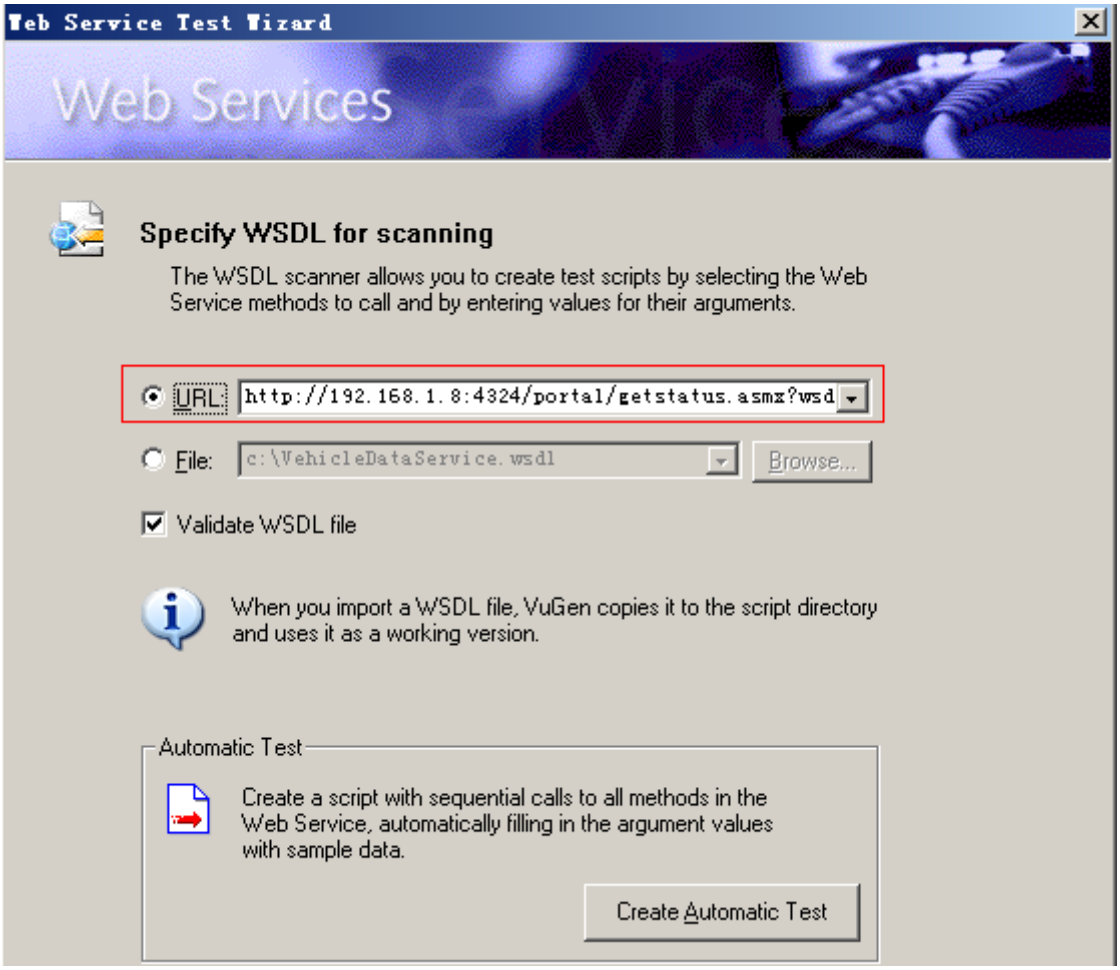
录制脚本

打开 的“Virtual User Generator”

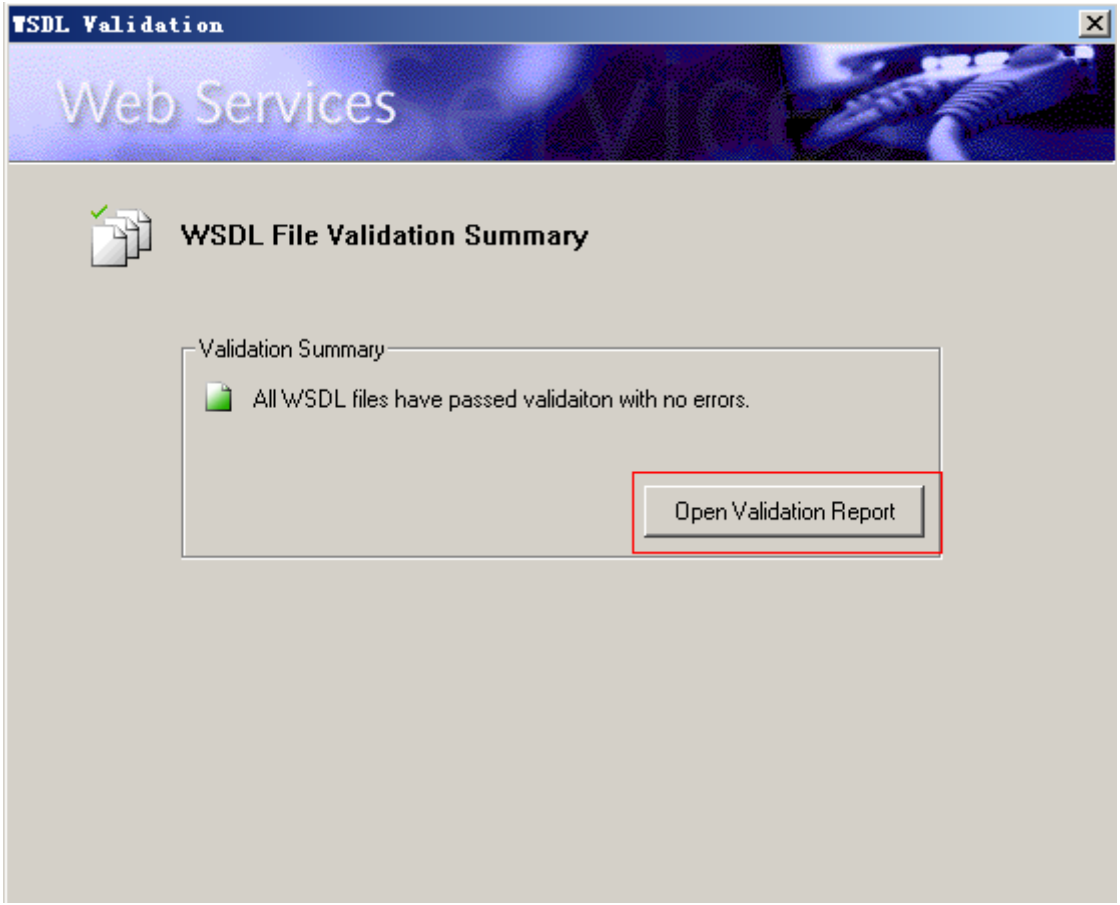
New 一个 virtual user，选择“Web Services”，点击“ok”



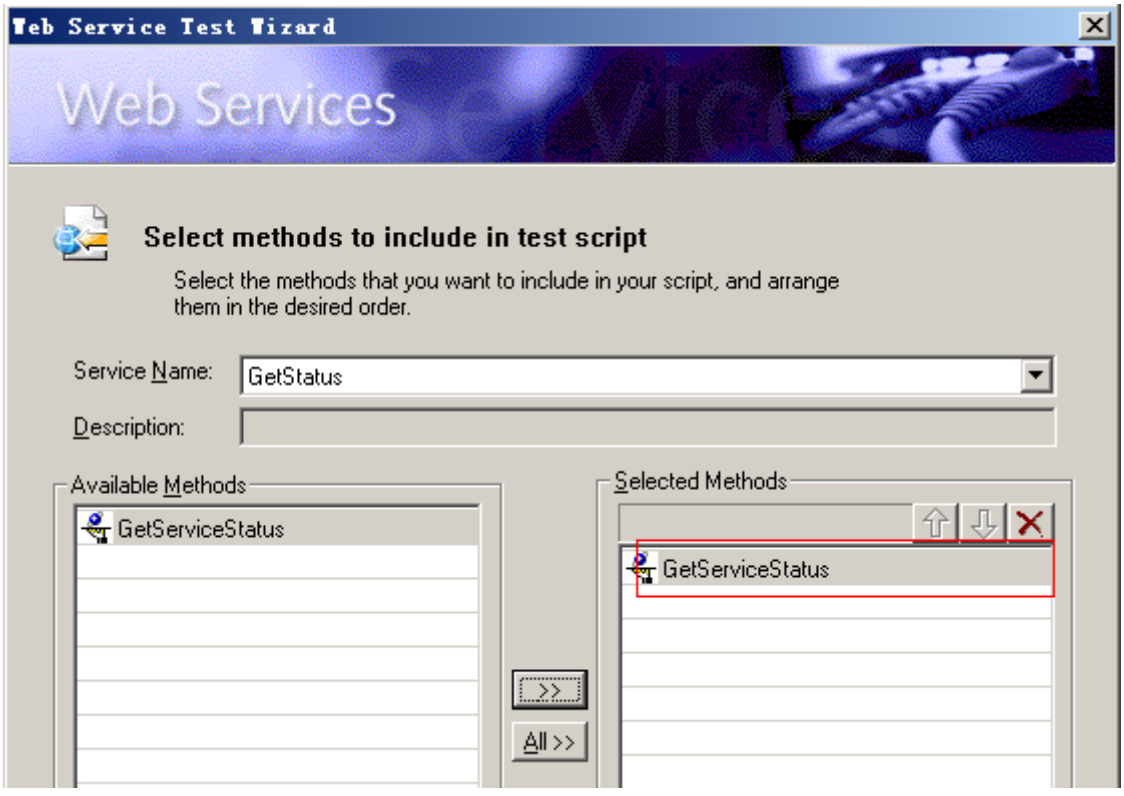
在弹出的脚本页面，选择“ Scan WSDL”，在 URL 中输入要测试的 webservice URL，点击“下一步”



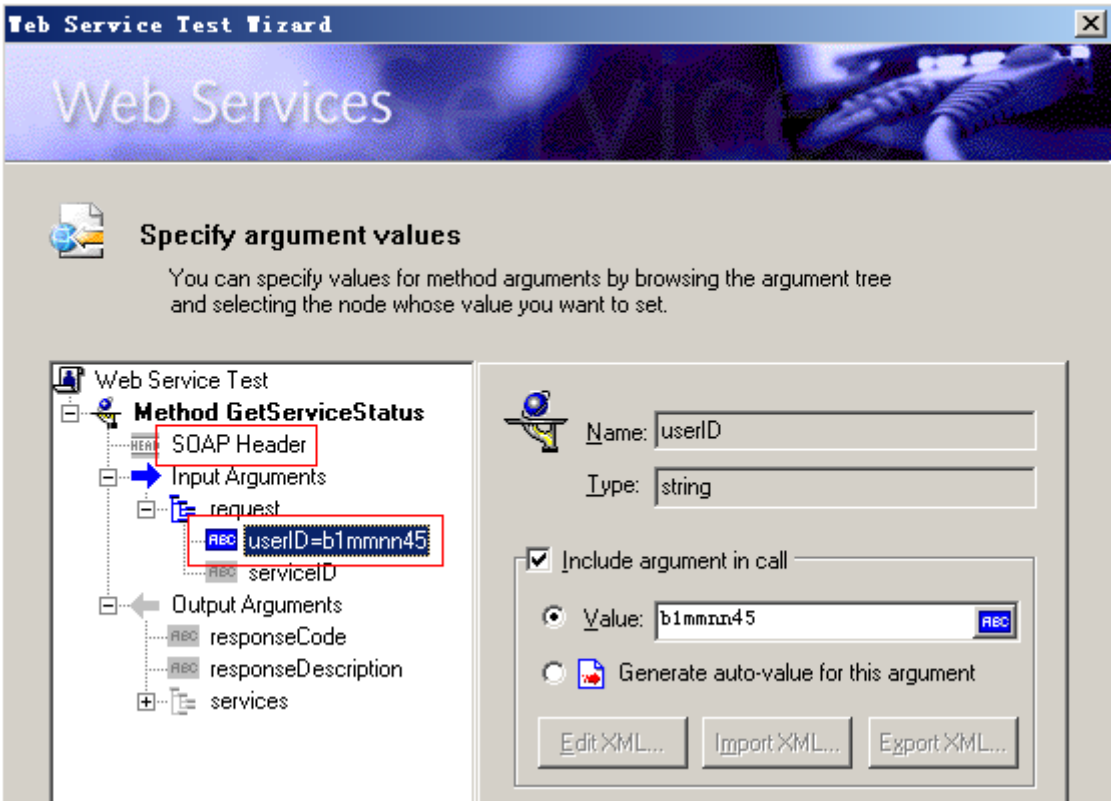
点击“Open Validation Report”来验证 URL 的有效性，点击“下一步”



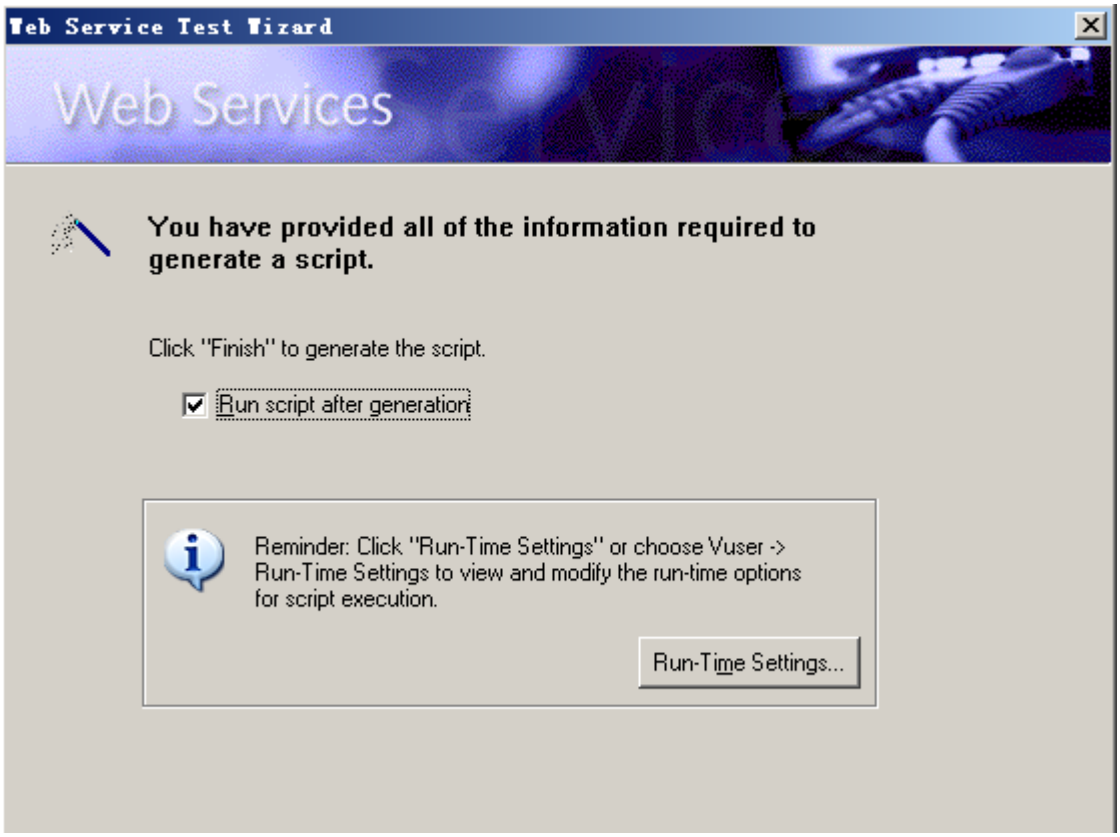
5.选择你要测试 Methods，点击“下一步”



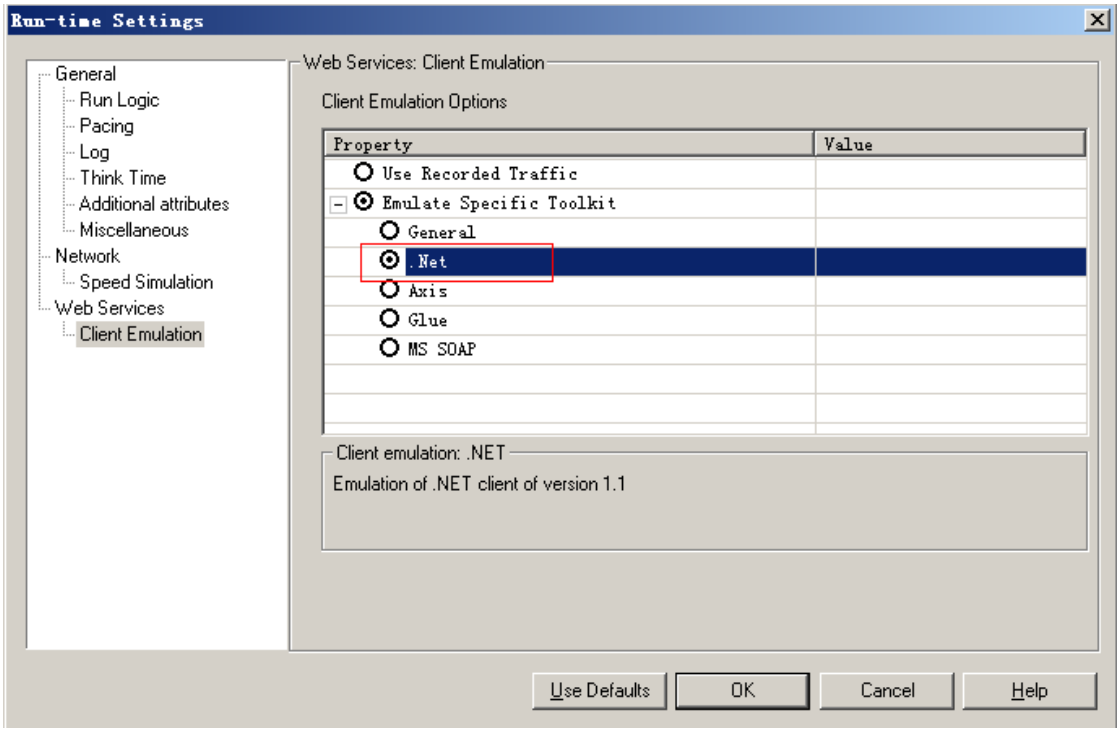
6. 输入 Specify argument values，点击“下一步”



7.勾选“Run script after generation”



设置“ run-time-setting→webserivce→Client Emulation→.Net”点击“完成”，loadrunner 将会自动产生脚本



8.soapheader 的添加

在 script View 模式中可以看到在刚才录制完后，脚本回放成功，但是这并不代表你的 webserivce 的功能正确，你需要查看所保存脚本文件夹目录下\result 1\Iteration1\t1.xml 中的 response 来判断 request 是否成功。上述录制脚本自动回放后的 response 为：

```
<responseCode>98</responseCode>
<responseDescription>Invalid Authentication Credentials</responseDescription>
```

与 response xml 的 success 不同，提示无效的验证错误，这是因为你未输入 soapheader 造成的，那么我们需要自己编写一段脚本来添加 soapheader：

在脚本中添加

```
Action()
{

    web_service_call( "StepName=GetServiceStatus_101",
      "SOAPMethod=GetStatus.GetServiceStatusHttpBinding.GetServiceStatus",
      "SOAPHeader=<soap:Header xmlns=\"http://bell.ca/vas/getServiceStatus\"><AuthenticationHeader><Username>user</U
      "ResponseParam=response",
      "WSDL=http://192.168.1.8:4324/portal/getstatus.asmx?wsdl",
      "UseWSDLCopy=1",
      "Snapshot=t1233629449.inf",
      BEGIN_ARGUMENTS,
      "xml:request=<request><userID>blmmnn45</userID><serviceID></serviceID>"
            "</request>",
      END_ARGUMENTS,
      BEGIN_RESULT,
      END_RESULT,
      LAST);

   // lr_think_time(3);

    return 0;
}
```

即：SOAPHeader=<soap:Header xmlns=\"http://bell.ca/vas/getServiceStatus\"><AuthenticationHeader><Username>user</Username><Password>pass</Password></AuthenticationHeader></soap:Header>

并保持在一行。

再次回放，查看 \result1\Iteration1\t1.xml 中的 response 来将会返回 success。

从订单管理系统到门户网站的 webserivce 的 URL：

http://www.webxml.com.cn/WebServices/ChinaZipSearchWebService.asmx?wsdl

其中 request XML：

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/
  <soap:Body>   不带有soapheader，直接通过soapbody向Server进行验证
    <tns:getAddressByZipCode>
      <tns:theZipCode>200085</tns:theZipCode>
      <tns:userID/>
    </tns:getAddressByZipCode>
  </soap:Body>
</soap:Envelope>
```

Response xml

```
  <PROVINCE>上海</PROVINCE>
  <CITY>上海市区</CITY>
  <ADDRESS>山西北路(单1-295,双2-296)</ADDRESS>
  <ZIP>200085</ZIP>
</ZipInfo>
<ZipInfo diffgr:id="ZipInfo2" msdata:rowOrder="1">
  <PROVINCE>上海</PROVINCE>
  <CITY>上海市区</CITY>
  <ADDRESS>廾封路</ADDRESS>
  <ZIP>200085</ZIP>
</ZipInfo>
<ZipInfo diffgr:id="ZipInfo3" msdata:rowOrder="2">
  <PROVINCE>上海</PROVINCE>
  <CITY>上海市区</CITY>
  <ADDRESS>文安路</ADDRESS>
  <ZIP>200085</ZIP>
</ZipInfo>
<ZipInfo diffgr:id="ZipInfo4" msdata:rowOrder="3">
  <PROVINCE>上海</PROVINCE>
  <CITY>上海市区</CITY>
  <ADDRESS>天潼路(单321-完,双300-完)</ADDRESS>
  <ZIP>200085</ZIP>
```

录制脚本：

录制方式同上，但是由于不带有 soapheader，所以需要在 run-time-setting→webserivce→Client Emulation→MS soap 进行设置。

其他步骤略。

脚本的加强：

从门户网站到订单管理系统的 webserivce 脚本加强.

在脚本中添加事务与集合点，并且参数化 userid。

并且屏蔽掉 lr_think_time(3)，因为这会影响性能测试结果。

2.从订单管理系统到门户网站的 webserivce 脚本则不需要设置事务和集合点，因为模拟的是单线程测试情况，只需要屏蔽掉 lr_think_time(3)。

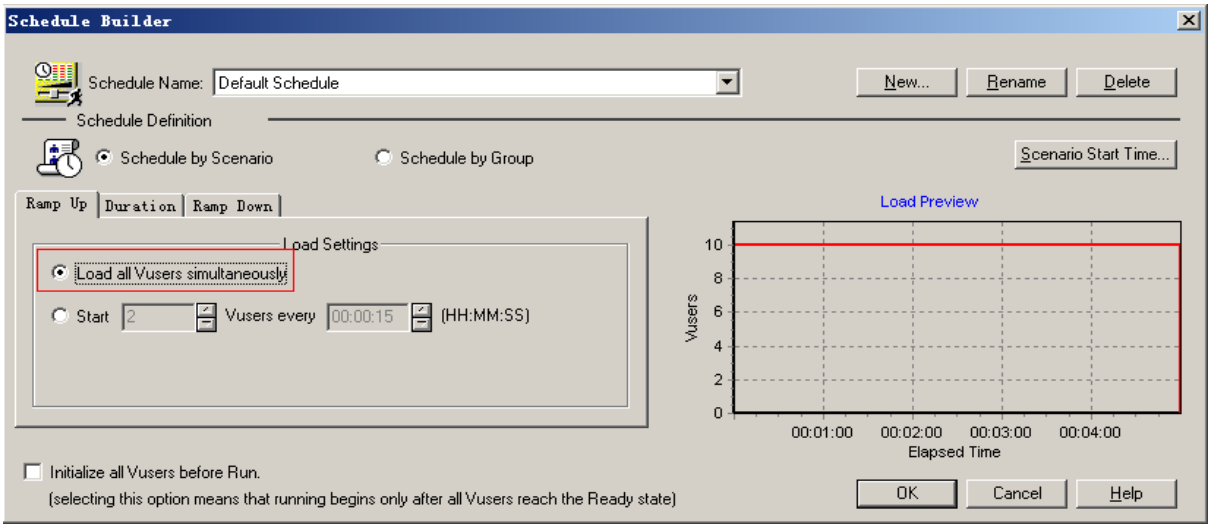
测试场景设计

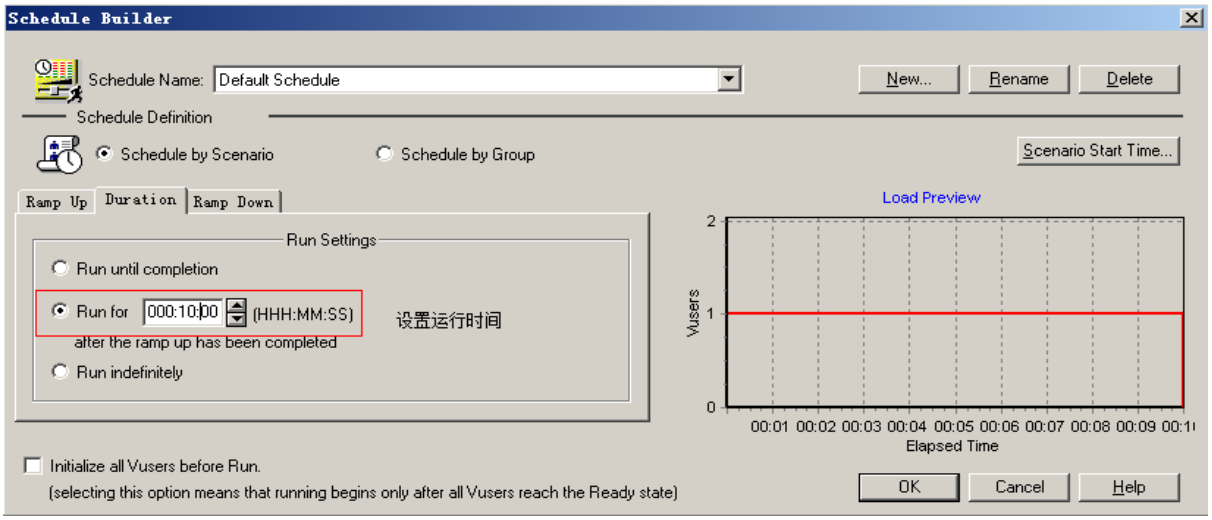
从门户网站到订单管理系统的 webserivce

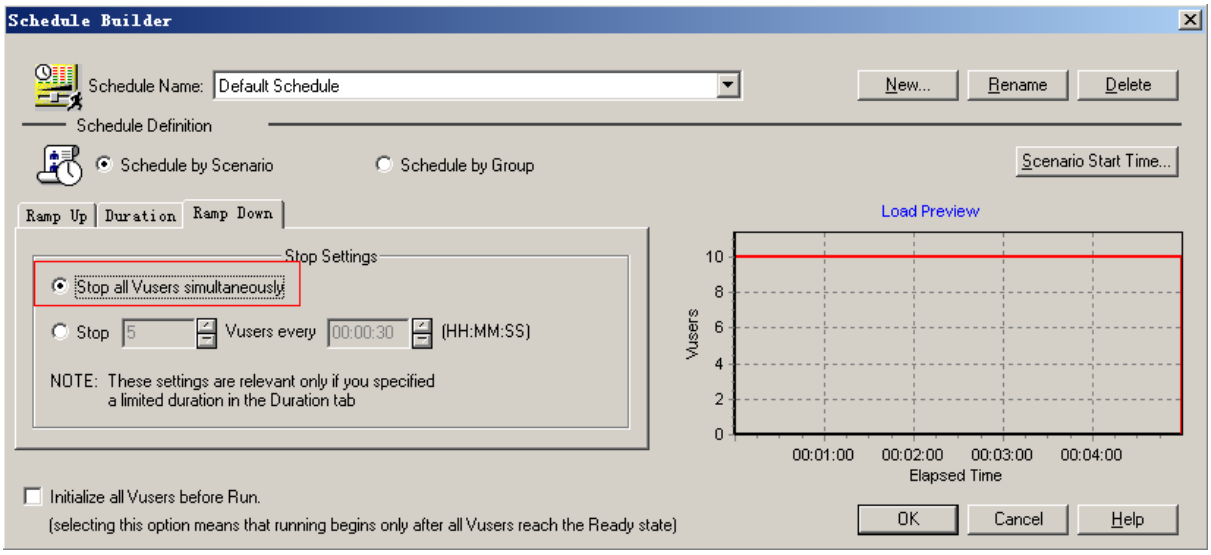
测试接口名称	并发用户数（个）	运行时长（分钟）	备注说明
Getstatus	10	10	10 用户并发，设计 10 个集合点
	20	10	20 用户并发，设计 10 个集合点
	30	10	30 用户并发，设计 10 个集合点
	40	10	40 用户并发，设计 10 个集合点
	50	10	50 用户并发，设计 10 个集合点
	60	10	60 用户并发，设计 10

			个集合点
	70	10	70 用户并发，设计 10 个集合点
	80	10	80 用户并发，设计 10 个集合点

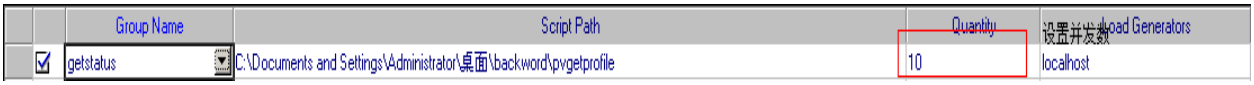
点击 



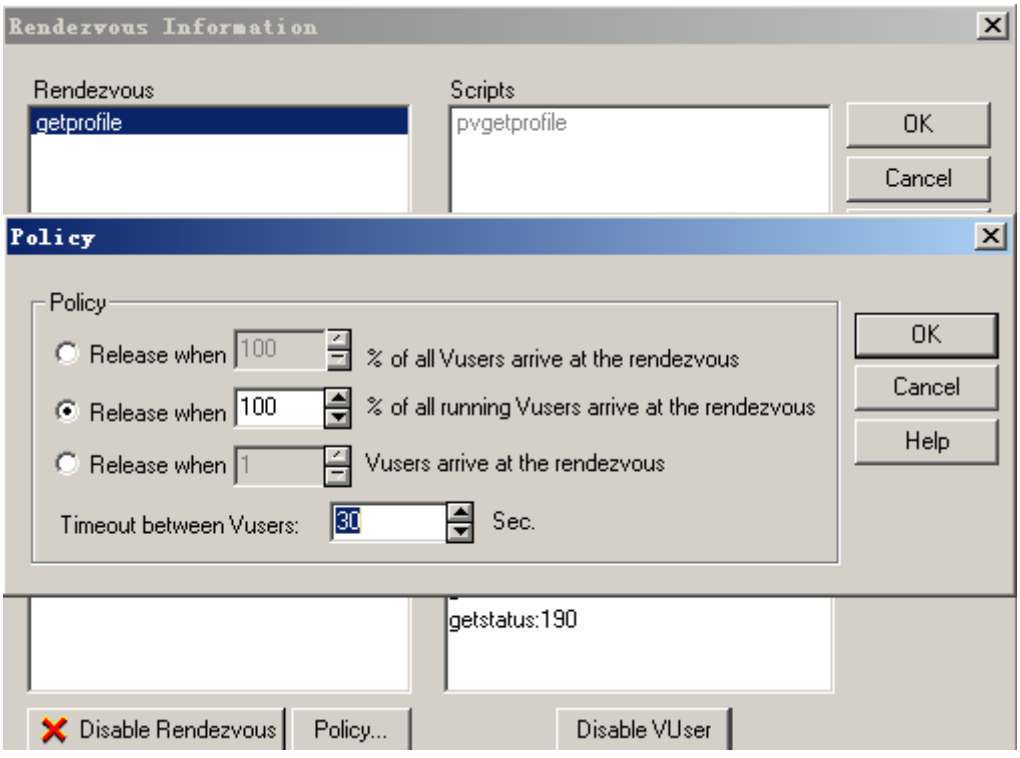




设置并发数



设置集合点

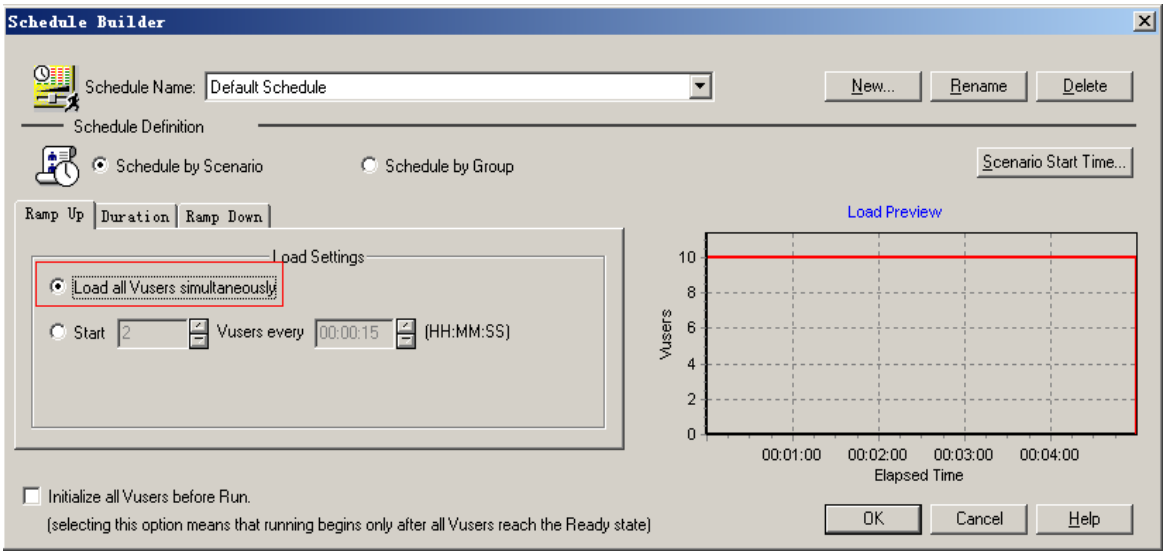


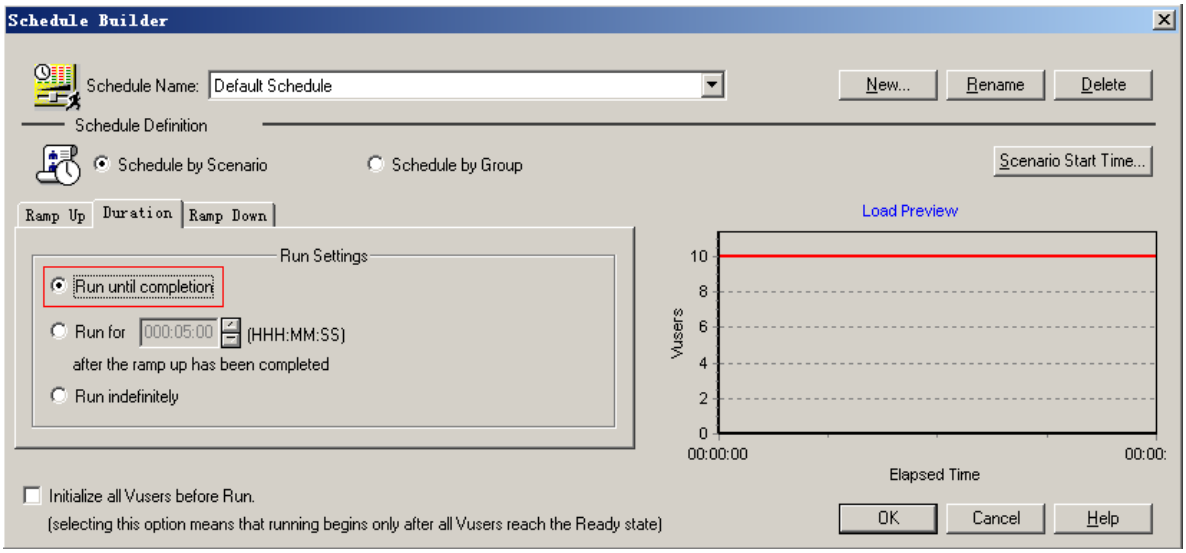
从订单管理系统到门户网站的 webserivce

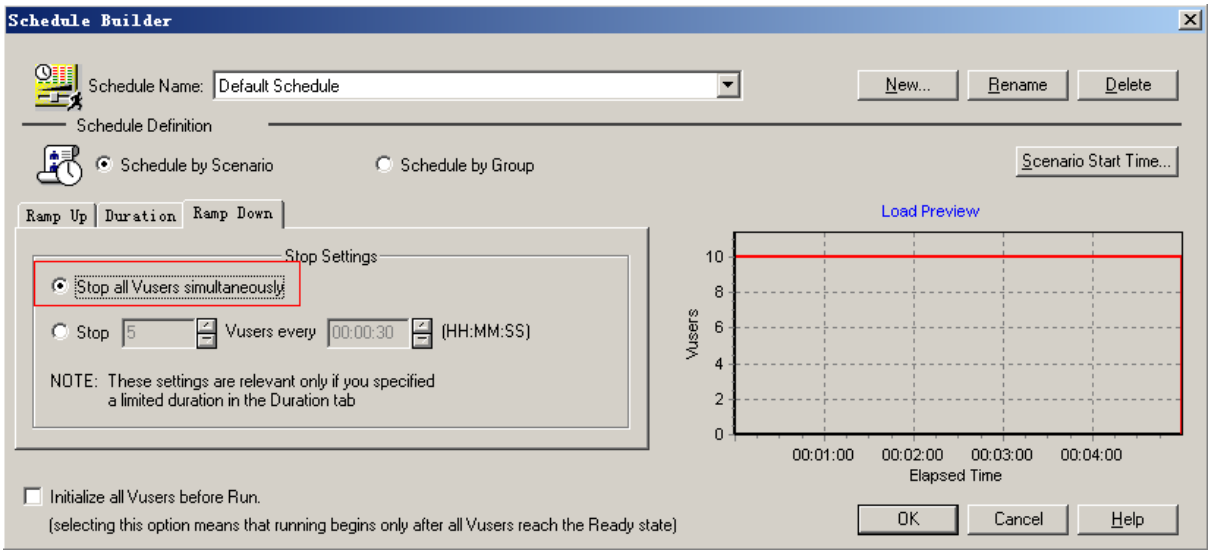
测试接口名称	并发用户数（个）	运行时长（分钟）	备注说明
	1	60	1 个用户迭代 N 次，

GetAddressByZipCode			持续 60 分钟
	1	60	1 个用户迭代 N 次，间隔相同时间持续发送 60 分钟
	1	60	1 个用户迭代 N 次，间隔不同时间持续发送 60 分钟

点击 

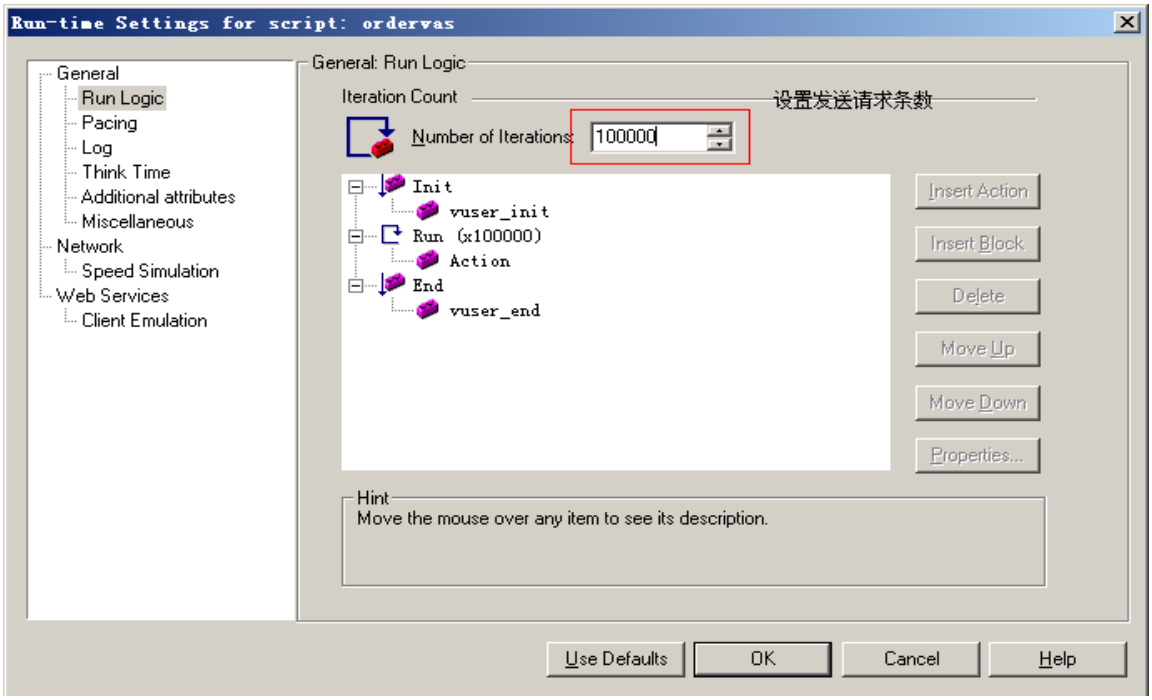


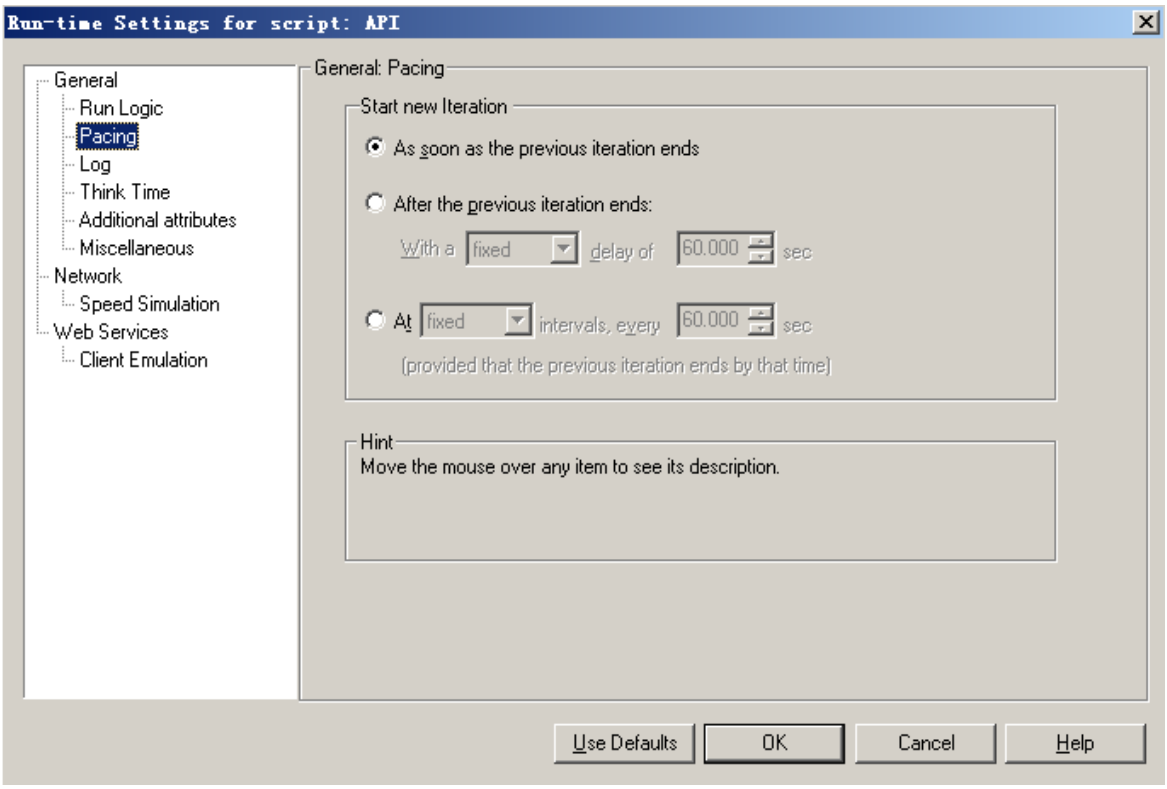




设置单线程数

设置请求数





持续发送情况下：选择 pacing→ As soon as the previous iteration ends

间隔相同时间发送情况下：选择 pacing→ After the previous iteration ends →Fixed→输入设置时间

间隔不同时间发送情况下：选择 pacing→ At→Random→输入设置时间

资源监控：

添加 windows 资源计数器：3 台 web 应用服务器，1 台数据库服务器，及 1 台文件服务器

添加 SQL Server 资源计数器：1 台数据库服务器

添加 Network Delay Time 计数器：测试负载压力工具机到 3 台 web 应用服务器，1 台数据库服务器，及 1 台文件服务器的 Network Delay。

然后，点击 

边运行边查看运行过程中的指标，并在运行 60 分钟后，手动停止场景。

总结

Webserivce 测试方法并不是仅上面所介绍的这些，每个测试工程师的测试方法都会不同。通常，webserivce 源代码并不能被测试工程师看到，webserivce 就

像一个黑盒子一样被我们测试，我们需要关注的是 Requset 和 Response 的 HTTP 的通信。

建立一个 web page 提供给用户来测试一个 webserivce 是最简单方式来验证这个 serivce 所预期的格式。但是得到测试 webserivce 的内部内容必须通过语言的使用，像：Perl，或者自动化工具像 Application Center Test 来提供一个特殊的支持 Requset 和 Response 的 HTTP 数据。你也可以通过 soapsonar，来测试其功能，也可以通过 VSTS 提供的 Loadtest 来进行性能测试，更可以使用 SOAPUI 来测试其功能及性能。

这篇文章只是我们专注 webserivce 通过使用 loadrunner 来测试其功能和性能的方法。主要包括了带有 soapheader 和不带 soapheader 的录制方式及并发情况和单线程大数据量情况下关于 Loadrunner 的运用方法。如果你也对 webserivce 的感兴趣，或许有更好的测试方法，请也拿出来进行交流与讨论。

测试之缘——与 51Testing 一同成长

作者：晒太阳的猫

引言

很早以前我就想总结一下自己的测试生涯，出于很多原因迟迟未能动笔。这次的机会让我燃起了写作的欲望，希望把我走过的点点滴滴分享给所有后来的人。我没有测试的天赋，没有好的学历，也没有好的技术背景，所以我的经历也并不传奇，但确可以带给想从事软件测试工作的朋友一点信心，因为通过这几年的努力，我已经得到了大多数人的认同。

《测试入门篇》

与软件测试结缘要从我的大学生涯说起，也许这是个戏剧性的开始，也是人生道路上的一种巧合，但也许早已命中注定。

高考时我考的不理想，上了一个专科民办学校，糊里糊涂的选择了软件测试专业，然后就开始了我的软件测试生涯。在大学里我遇到了几个好老师，认真教我测试知识，教我做人的道理，现在我们都成了朋友。在他们的热心帮助下，我学到了一些测试的基础知识，也掌握了一些测试技术，可在找工作的时候还是遇到了麻烦。多数工作要求具备工作经验，或者这样那样的能力，证书，学历，而我什么优势也没有。工作迟迟没有着落，眼看着很多同学放弃了软件测试，做起了销售，保险，生意等等，我心里很不是滋味儿。但是对此我不想多说什么，目前很多高校的学生读出来做的都不是本行，但是可惜了啊，软件测试是个好专业。虽然我懂得并不多，但是我不甘心就这样放弃了。事实证明我的坚持也许在某种意义上是正确的，在我毕业后不久软件测试就开始在沿海有火爆的倾向了。

我知道不能一味的等待，必须要行动起来，改变现状，提升自己的能力。于是我抄了一份网上比较在意的一些测试技术要求，想参加一些技术培训班，希望可以在短期内提高自己的能力，找到一份软件测试的工作。成都本地那时候已经开始有培训机构了，但是做测试培训的很少。有一家做测试培训的，我亲自去看了一下，负责人告诉我她以前在某某公司做了 3 年的测试，可以量身培训你欠缺

的测试技术，价格也合理。人是有眼睛的，培训机构面积不大，是在一座居民楼里，我进去的时候就觉得转了好多个弯，出来的时候都是问到路出来的。看到现场的情况后我顿时对这家培训机构失去了信心。还有一家是业界比较有名气的，结果打电话去一问，人家说的成都还没有开展软件测试的培训，要我去北京或深圳才行。

也许这就是缘分，如果再晚一年那家培训机构就有测试的培训了，我也就不会去 51 培训了。其实去 51 也算是巧合，并不是在 51 的网站上了解到的，而是无意间在一个测试的 QQ 群上，听群上的人说起 51 在做培训还不错，一下子心里好激动，马上上 51 看了详细的介绍并打电话咨询。最后听说要电话面试，看能不能入学。要考数据库，数据结构，C 语言，还有网络。我当时心都凉了，对自己能否去培训一点信心都没有，电面的时间约定在一周以后。于是我开始找老师恶补，每天就是在图书馆，教学楼和宿舍。

面试我的人是周峰老师（51 的三个创始人之一），问了我很多问题，面试的时间持续了 1 个小时。问的我耳朵都发烫了，很多问题我都是照本宣科的念。最后周峰老师说你花点时间看看谭浩强的 C 语言吧，一周后再复试一下。电话挂了后我马上和 51 那边的咨询老师（我们喜欢叫她肚子老师）联系，希望知道电面的情况。很快肚子老师告诉我周峰老师说我理论还是知道点，就是基础太差了，编码的基本东西都没掌握。第一次听人这样评价自己，心里像是在滴血，不过还好，肚子老师说我是学软件测试的，老师看了我的简历，还算是对测试有点概念，所以还是给了我机会参加复试。

当时我好担心复试过不了，每天都是带到笔记本去教室编 C 语言的简单程序，努力学好老师所说的欠缺的地方。功夫不负有心人，终于我通过了复试。就这样，在毕业的前夕和一个同学乘火车去上海参加培训。上海对我来说完全是个陌生的城市，在这里我呆了 4 个月，学习了软件测试的入门技术，比如软件测试理论，测试方法实践，测试工具学习等等，其实这些东西我在学校何尝没学过呢？只是老师讲的不够深入，我们实际动手的次数太少了，当然交钱培训的时候人比在学校总要认真很多，这也算是其中一个因素吧。每个阶段学习结束都会考试，大概考了 3 次，记得印象最深刻的是第一次考试，很厚的一本软件测试基础理论，

而且没有复习提纲。在这样的情况下只有硬着头皮从头到尾的看了，也正是这次认真的看书让我把以前学过的软件测试知识都串联在一起了，可能是传说中的量变引起质变吧，我仿佛顿悟软件测试了，我知道从头到尾它是怎么运行的。那次考试我考了 92 分，班上只有一个学员比我高 4 分。

当然考试并不能完全说明问题，这点我承认，但内心的喜悦确实久违了。我开始自己整理学习成果，把自己的体会写成文字的总结，比如“测试基本概念的总结”，“测试质量管理的总结”，“测试方法的总结”等等。我还总结了系统测试，集成测试，单元测试的文档和流程。越是总结越让我脑袋里面有了清晰的线路，所有的知识点都在我脑袋里面融会贯通了一样。就这样我开始微笑着期待找工作的日子。

《测试提高篇》

（一）投简历面试的日子

在 51 的培训结束后我匆匆赶回成都（父母不希望我离家太远）。回成都后我开始充实我的简历，中英文的各写了一份，本人英语不好，东拼西凑算是弄了一篇出来。我开始在网上找要求工作经验一年左右的，技能和我现在差不多匹配的职位。这点是在 51 学到的，老师告诉我们不要害怕要求一年工作经验的，我们系统学习过，不比别人差，去试一试一定没问题的。我大概投了 3 到 4 家公司吧，其实在成都合适自己的公司也并不算很多，毕竟就业要双方都满意。不知道为何我如此固执，一定要找软件测试的工作，一定要是家我也看得上的公司，否则我宁愿继续读书。现在想来自己的固执也许是种坚持，这正是我去 51 培训的动力所在吧！

但一切并不如我想象中的那么如意，我等了一个多星期都没有一家公司通知我面试，我开始害怕自己是不是自视甚高？自己的想法是不是有问题？我在成都整整呆了 10 天，幸好有我老师的收留，其中有一天还是我 21 岁的生日。父母给我打电话，问我工作找到了吗？我总是骗他们说公司面试的过程很长，要很多轮去了，目前还顺利；有公司要我，但是太小我不愿意去。我没有勇气告诉他们没有公司通知我面试，51 的培训并不便宜，加上在上海的吃住绝对是一笔不小的

投入了。这十天可能是我人生中最低谷的日子，我怀疑自己的想法，怀疑自己的能力。

还好仅仅是十天，过后我陆续收到了 3 家公司的面试电话，第一家面试的公司在移动音乐基地，总部在北京，是给移动做外包业务的。公司在成都的规模不算很大，但看起来还是有模有样的。很幸运面试的题目居然都是软件测试理论，这根本难不倒我，我基本上用了 15 分钟就把所有的题目做完了。交卷后技术部的负责人就直接过来找我了，他们对我的答案很满意，和我聊起了 51 培训的很多话题，以及我在学校的一些经历。大家聊的很愉快，最后经理竟然丢下一句让我非常感动的话：“我们公司目前对测试来说还不规范，也没有自动化，都是手工测试，以你的条件可以去尝试下更大的公司。当然如果你愿意留下来我肯定欢迎，至于你想要的薪水完全没问题。”我离开那家公司的时候心都飘飘然了，两天后我就收到这家公司的 Offer 了，准确的说给的比我要的还多。

在去这家公司的当天上午我接到了另外一家公司的电话，让我第二天下午 1 点去面试，这次的面试题目分三个部分，软件测试，数据库应用以及 C 语言的基础程序题。个人感觉这样的考试会比较全面，做了 1 个多小时，题量还是不小。当然 C 语言的题还是没做起，去测试培训后就没有看过 C 语言了，对数据库也稍感陌生。笔试后马上就是技术面试，面试我的大姐姐是个测试负责人，感觉很亲切，并不严肃。她问了我很多细节上的问题，总的来说都是测试方面的细节。由于做过很多实践类的小项目，我自认答的不算太差，大家的交流还是比较愉快的。然后就是测试经理的面试了，大公司就是不一样，面试的次数要多很多，小公司测试负责人面了就算是可以正式入职了。当 HR 问起我薪水上的问题我要的不高，有两个原因：一是我相信去大点的公司会有更高的起点，二是我觉得自己欠缺的确实太多。有公司愿意给你一个机会去学习，我认为已经很不错了，没有必要去盲目的和同学攀比第一份薪水的高低。（当然薪水也不能低到自己还要伸手向父母要钱生活，我觉得这就够了）这次的面试让我很理性的看待工作了。

就在面试的当天下午我接到第三家公司的面试通知，真巧，这两家公司都在同一个地方。次日早上我就去面试了，这次的题目是全英文的，分了很多模块，什么 Unit 的，网络通讯的，好不容易看到 Windows 的题目，勉强做了两个选择

题。我当时的想法是测评中心就是不一样啊，要求高的离谱，我又开始怀疑自己到底会什么了。15 分钟我交卷了，然后又是直接技术面试，我很坦诚的说：“这些都不是我培训所学，也不是学校教的范畴，我确实不会，很抱歉。”没想到面试我的人并不在意这些，他看了一下说：“没关系，主要是看一下你的知识面，另外你是否真的有 Mercury 的 Loadrunner 认证？”我点了点头，交流的不多，大致上他的意思是如果我有认证可以考虑培养我。出门后我很疑惑为何公司看重的是认证，学校或学历而不是能力呢？

最后我选择了第二家面试我的公司，虽然这家给我的起薪水最少，但是我感觉我会学到更多的东西，至少他们更看重能力。

（二）我的第一份工作

顺利签了 Offer，我开始了上班生活。我很珍惜自己的第一份工作，工作的试用期是三个月。很幸运我分到了面试我技术的姐姐手下，又很幸运的分到了电子商务项目组。这就像是种注定的巧合，我只能说公司把我放对了地方，给了我一个好的开始。

工作后我给周峰老师（51 的创始人之一）写过一封电子邮件，主要是请教他刚开始工作的注意事项，以下是邮件的部分正文：

刚开始工作，以下方面可以注意，虽然公司不一定要求你这么规范的做，但你可以有意识去规范的做事情：

1、进行测试用例设计时思路清晰，用例文档写作规范，描述详细清楚；

2、缺陷描述详细规范，有隔离等措施；

3、能对工作中的心得、经验等总结成案例，交给测试经理共享；

4、在测试执行期间能每天规范地提交测试日志；

5、在测试结束后能规范的进行总结，写作测试报告，进行各种测试度量数据的分析。

这封邮件我至今仍然保留，并且我真的做到了。它对我将来的测试工作产生了至关重要的影响，也是为何我会很快得到项目组人员，测试经理和测试负责人赏识与提拔的重要原因。

在试用期间，我第一个做的测试是某项目的界面测试。想起来真的是巧合到

不行，界面测试是我的毕业设计，我对 GUI 测试一直都有学习的兴趣。基本上没费好大的力气就编写完了测试用例，测试工作进展的非常顺利，为此我还写了一个案例《需求了解度对 GUI 测试的影响 20061225》，当然这也得力于 UI 设计人员的技术水平（刚好他是我们公司 UI Team 最优秀的 UI 设计师之一）。正是这样一个好的开始注定了一段好运气。过后我开始协助其他测试人员进行测试执行工作，不久我被分配写小项目的用例。当时公司正准备过 CMMI3，而我学的测试流程基本上是 CMM4 到 5 的水平，所以很快就发现了工作中存在流程上和规范性上的问题，有很多待改进的地方。很感谢我的测试组长，刚好她曾就职的公司在规范性上做的不错，她尊重我的建议，并让我的想法逐步得到实施。要知道现实和理论往往是由差距的，而且差距有时还很大，能有一位好的上级引导你实在是件幸运的事情。不久我又做了一个案例《CMM 标准在测试中的运用 20070124》。每天我总会向测试组长主动报告情况，我会主动询问她今日的工作安排，如果没有安排我可以做些什么，下班前我也会主动的反馈今日的工作成果，如遇到问题我会及时的向她反映。大家不要小看主动汇报的积极作用，这会让你的主管对你产生好感，也会感受到你的专业，对自己以后从事管理工作也做了一个很好的铺垫。后来我做管理工作的时候才知道为何项目负责人希望组员给予自己及时的反馈，因为他们要负责很多事情，有很多人的工作要去检查，而组员如果临时遇到问题他们无法及时知道情况以便调整工作安排。

就这样，我顺利的转为正式员工。

（三）第一年我做了些什么

在上一个主题中我提到我进入电子商务项目组是种幸运，遇见一位好的测试负责人引导也是种幸运。那我就先来讲讲为何会觉得幸运，这是我这一年成长中的两个助推因素。

首先说一下项目组，该项目组的多数项目都是做 WEB 的系统测试，而几乎我学的所有的系统测试方法都适合 WEB 测试，这让我充分展现了在用例设计上的优势。可以很自信地说当时我设计出来的用例是很有技术含量的，不久写了一个案例《测试方法在 WEB 中的应用 20070404》，总之算是入对项目组了。再说一下带我的测试负责人（其实现在她已经是我朋友兼室友了），她在公司的测试团队

里面级别是很高的，有着多年的工作经验以及其管理方面的卓越才，我在她手下工作的日子十分愉快。我每天上班都是充满活力，总是微笑，总是充满热情。那种发自内心的快乐让我对软件测试着迷，也许正是这份快乐感染了很多同事，很自然和同事们打成一片了。

很快公司组织了 QTP 的远程培训，我有幸参加，然后负责给全公司的测试人员做推广培训。这是我第一次讲课，同事们给了我很大的鼓励和支持。这也为我今后的发展道路多了一种选择，也许我本来就很适合做培训呢？

过后不久带我的测试负责人就要调到其他项目组做项目经理了，这是她要走的路，我虽然害怕以后要独立面对测试中面临的诸多问题，但也许这更是一种机会。我开始独立负责中型规模的项目，包括测试计划，测试用例编写，测试执行，并且开始带新来的实习生。虽然还不能胜任测试负责人的职位，但是在这个过程中让我体会了太多。由于是自己负责，我可以充分的把我所学的展现出来，我把所学的理论变为现实，并且创造了不错的绩效，然后再通过实际的工作不断的完善，修正和扩展自己所学的理论。那种感觉我无语言表，如果非要表达什么，我只能说我的选择和付出都是值得的。

通过一年的努力（含试用期）我在测试用例设计思路上算是轻车熟路，后来公司的测试用例实践培训也是我在负责。对于工作中所学到的我都喜欢做成案例分享给项目组的成员，有些适合的就做全员的分享。我喜欢知识共享的感觉，也喜欢和同事交流技术，在我眼里知识的运用比知识的拥有更重要，这是我读大学时学到最重要的思想之一。虽然我的技术在高手眼里就是菜鸟级别，但是只要能为工作所用我觉得就是好的。在测试文档的编写上我尽量趋于规范，完全可以达到公司对 CMMI3 的要求，我一直是以 4 或 5 的标准来完成自己的测试文档编写工作。并且在较为重大的项目结束后我会做总结，把项目中自己没考虑到的 BUG，其他同事考虑到的记录下来，经常分析自己在设计上和执行上存在的不足，这为我积累了丰富的 WEB 测试经验。我总是认真的记录 BUG 描述，特别是重现步骤，这样便于开发人员定位问题，也便于时间长了自己不记得 BUG 的真实情况，也为后期我做 BUG 分析和数据度量打下了很好的基础。

这里我不想说太多的感谢，感谢我所有的测试老师，感谢 51 的老师，感谢

带我的测试负责人，以及测试经理对我的肯定，总之这一年让我学到的太多。

（四）开始弥补不足

俗话说万丈高楼平地起，有好的基础才能走得更远，这是我为何觉得自己成长很快的原因。一年后我开始思考自己的不足，希望有一天可以成为一名优秀的测试工程师。首先我不会编码，也没有测试度量数据分析的经验，英文也不好，对 WEB 测试的了解也仅限于皮毛。这一切都成为我继续进步的阻碍，那段时间我很喜欢逛论坛，去看一些前沿技术和一些优秀测试工程师写的文章。51 的论坛好大，也让我觉得测试技术好深，只是它比开发更容易入门罢了，要做好其实并不容易。我必须要找到一个突破口，什么事情都要有个循序渐进的过程。

机会总是给有准备的人，刚好公司想要发展性能测试，打算成立性能测试小组，当然我不是最佳人选，负责人是一个有三年性能测试工作经验的人，但由于我做过 QTP 的培训又有 Loadrunner CPE 的认证，加上我的争取，最终把我也规划进了性能测试工作的范畴。就这样开始了半工作半学习技术的模式，我开始跟到性能测试组长做项目，抽空看性能测试的资料。从开始的脚本录制到后来的自动化用例设计，我上手都很快，你会体现出手工测试用例设计熟练的优势，记得有老师告诉我手工测试用例都写不好的人难道自动化测试用例就能写好吗？我不知道这句话是否正确，但事实证明我学习的很快，并且写出来的用例让负责人较为满意。不久我开始制定性能测试的流程和在公司推行的方案。

在一切开始走入正轨的时候我升职成为了一名测试负责人，我要开始熟悉新项目组的业务，工作的重心也就发生了改变，我开始做测试管理，开始带初级测试人员。大多时候计划没有变化快，性能测试工作必须放下一段时间。必须做权衡的调整，要知道这是一家公司，有时候还是应该多为公司想一下，当它需要你的时候不能只考虑到自己的发展（原本我是计划先做 2 到 3 年的技术积累后再从事测试管理工作）。在成为测试负责人后我开始学习配置管理，编写新项目组的业务手册，开始为项目组的测试做规划，也趁此机会开始为测试度量数据分析做准备。我开始更有责任感的工作，毕竟是要带人做事了。其实带的第一个人并不是在我做测试负责人的时候，而是在我做一个中型项目带实习生的时候。我急切的希望他能写出好的用例，能够做计划和测试执行。我忽略了能力的成长是需要

一个过程的，现在想起来都觉得对他很歉意，当时对他的要求较为严格。当然后来他还是很感激我对他规范的管理，现在他在另外一家公司有了更好的发展。后期我也负责过实习生的测试基础培训工作，在管理方面也就更有经验了。

也许公司真的对我格外的厚待，知道我英文不好居然要求用英文编写用例，用英文写测试报告，测试计划。当然需求说明本身是英文的，所以这个依然不变，其余所有文档都要求英文了。这让我必须正视自己最害怕的英语，人都是逼出来的嘛，渐渐也就开始习惯用英文写用例了，但是用英文写测试报告，想表达清楚我要说的意思对我来说还是件很困难的事情。不过万事开头难，头已经开了，相信量变一定会引起质变的。

尾声

当然这一切的一切只是一个开始，它见证了我从最初的无知到现在的稍微懂事。我还在继续努力寻求适合自己的发展之路，以后会和大家分享这段经历。当然这期间并不像描述的这样一帆风顺，我也有过迷茫，也有过失落，也有过无奈。这也是在今年年初最集中的情绪反应，我一直在调整自己，让自己保持好的心态，寻求一种生活和工作的平衡。

作者简介

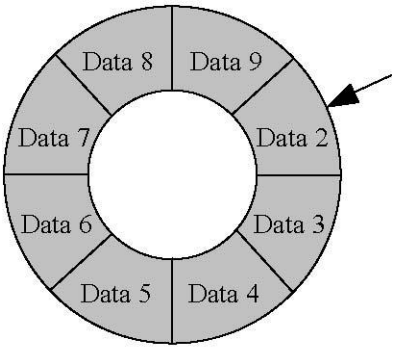
本人在校主修软件测试课程，接受过 51testing 的系统培训，拥有 Mercury LoadRunner CPE 认证。近两年外企软件测试工作经验，熟悉 WEB 测试，有扎实的测试基础和较强的业务理解能力。现任公司某项目组测试负责人，有一定测试管理和培训经验。

利用环形缓冲区日志分析错误

译者：李容

引言

本文包含了一系列相关的模型。所有的模型均涉及到的是，一个程序员，你怎样掌握你所需要的信息，理解并解决了软件错误。该方案围绕着使用环形缓冲区关键日志来记录一个系统的生命周期。



所谓的环形缓冲区就是一个指定了大小的缓冲区，当他被使用完后，系统会增加另一个区块来覆盖第一个。上图显示一个环形缓冲区可容纳 8 个数据文件。当第九个已经增加后，按照顺序，产生的第九个会覆盖第一个，再增加一个时，依次类推，它将覆盖第二个……

这些模型有两个特点：

1、 易用性；

2、 随着时间的推移，形成一个系统的日志文件，从而提高调试效率。

问题	解决方案	模型名称
在错误不能稳定重现的时候，如何使得你的用户在遇到错误后能够正确的描述错误信息？	在环形缓冲区中存储一个关于系统操作的历史文件。	环形缓冲区日志
用户在问题发生很长时候后，才关注了这个问题。	如果该程序显示单一状态，则它应转存日志。	转存环形缓冲区的错误
没有得到用户的信任	关于是否向您发送日志，允许他们作一个合理的决定。	用户友好的日志
如何知道日志中记录什么样的信息？	直到你有一个更好的主意，才登录。	边缘检查
	日志中最后的错误要容易找到。	捷径
缓冲区总是太早的被填满。	不同详细程度的两份日志。	两个记录点

一、环形缓冲区的日志

客服给你转来一个客户。她很郁闷。她直接给你说，系统崩溃了。然后你问“在这个问题发生前，你还记得你曾做过什么样的操作？” 但是她不记得了。

作为答复，在产品中增加日志。关键事情和状态改变都被记录到一个硬盘文件中。然后告知客服，请他们告知用户，在反馈问题的时候，应当以电子邮件方式将日志文件一并反馈。

随着时间的流逝。让你吃惊的是，从外地发回的错误报告往往不包含日志。你向客服询问原因，他们说，用户错误的认为关闭日志对系统的性能提升是一个很大的帮助。

时间越长。该项目的系统维护小组会指导您从产品中删除日志文件。在一个关键的客户网站，您的日志填满硬盘。一夜之间，相对于产品，他们的费用下跌了大约 6 倍。

问题

在错误不能稳定重现的时候，如何使得你的用户在遇到错误后能够正确的描述错误信息？

正文

这些模式最适合于在用户的计算机运行独立的应用程序。其中的一些并不适用是因为大多数的处理是在远程服务器上。分布式应用程序需要另外的模式。

要点

没有持续关注一个问题，而用户由此提交的问题报告，可能是不完整和不准确的；

即使是准确和完整的用户报告也没能提供所有的相关信息。在其他人（用户除外）的机器上执行这个程序。

即使你在问题被发现的时候检查出了项目的冻结状态，但是重要的信息已经丢失。即使你知道所有变量的当前值，已经不是他们过去的值了。堆栈跟踪包含一些过去的历史操作，但不是全部。

将每一次操作记录到磁盘上，这个代价将相当昂贵。这将导致程序的执行效率缓慢下降，也将消耗磁盘空间。

如果让用户在没有问题时关注日志，他们会比较恼火。

因此

在环形缓冲区中记录关键事件、包括用户的操作。提供了一种环形缓冲区被写入到磁盘的方法（并在其后附有一封电子邮件）。这就是所谓的“倾销日志”。当发现一个问题时允许用户手动转存日志。这样使程序找到自己的问题并自动转存日志变得有可能（见自卸环形缓冲区的错误）。如有需要，则提供一个工具来恢复日志中的残余失事程序（例如，在 Unix 中，从一个核心文件）。

Linux 的环形缓冲区的例子是根据命令 dmesg（ 8 ）［Linux 系统]转录的。下面的日志是在启动过程中显示的：

Console: 16 point font, 400 scans

Console: colour VGA+ 80x25, 1 virtual console (max 63) pcibios_init : BIOS32 Service Directory structure at 0x000f73d0 pcibios_init : BIOS32 Service Directory entry at 0xfd7a0 pcibios_init : PCI BIOS revision 2.10 entry at 0xfd9b6 Probing PCI hardware. Warning : Unknown PCI device (8086:7180). Please read include/linux/pci.h Warning : Unknown PCI device (8086:7181). Please read include/linux/pci.h Calibrating delay loop.. ok - 333.41 BogoMIPS Memory:

Linux 的环形缓冲区默认 8192 字节长。

Trace.java［ Marick97 ］是一个开源的 Java 跟踪子系统，它存储了多达 500 个轨迹（日志）在信息清单中。当增加到 501 个信息时，原来 500 中排序第一的将会被删除并进行一定的垃圾回收。当被转存后，该环形缓冲区将会是以下格式的一系列信息：

=== 1997/09/22 13:52:09 (Controller.event:Controller.java:26) USE gui:

Linedraw 选定 Transarc 的 Encina 分布式事务处理系统［ Transarc97 ］采用了 64K 的追踪缓冲区。当某个特定追踪信息发布到日志中，一个进程退出或者发出一个信号时，系统就会根据指令将存放在追踪缓冲区的邮件转存到另外一个文件。

结果

当发生了一个错误后，用户可以为你提供更多的有用信息。

什么是有用的信息？见边缘检查、捷径和两个记录点。

根据日志文件记录的信息，可以得知用户是不是有误操作。见转存环形缓冲区的错误和用户友好的日志。

注意，在与此是相反的，Linux 操作系统中没有一个固定的内存量分配给该缓冲区。数量固定的信息更容易使用，在大内存环境下也没有不良后果，所以跟踪邮件的大小可能大多是相同的。

二、转存环形缓冲区的误差

从用户那里接到一份缺陷报告和缺陷日志，你会非常失望的发现日志中没有

一点有用的信息。原因在哪里呢？用户篡改了程序并试图恢复错误，在放弃之前准备寻求客服帮助，是谁告诉她转存日志。

问题

此刻，由于一个长期的问题（或可能是早已被发现的问题）让用户决定在系统中使用日志。但关键的信息可能还是会丢失。

要点

通常来讲，在错误发生前日志中记录的信息比错误发生了所记录的信息更为有用。

日志是以固定的格式存储，一些信息在错误发生前就可能已经丢失，有可能包括最重要的信息[2]。

如果是用户手工转存日志，则很有可能不会那么及时的处理。

因此

只要是有可能，应该让程序自动在适当的时间内转存日志，而不必等待用户来手工操作。

如果程序因为发现一个错误自动转存到日志，您可以告诉用户具体的问题是什么以及在哪里可以找到日志文件。这听起来似乎比较复杂：

有时，用户将不会注意到这样的错误。例如：它可能只会导致某种短暂的屏幕闪或一直没反应，直到用户不耐烦而直接重启。实际上，关于该错误的提示比错误本身更具破坏性。

重复同样错误的提示更让人生厌。

不管是不再重复提示或对这些做一个汇总，都需要小心的处理这个事情。Linux 和其它版本的 Unix 内核会输出“最后消息重复 343 次”，而不是显示所有 343 个实例。

还注意到，部分环形环缓冲器首先是要节省磁盘空间。如果该程序确实失控，最终可能需要每秒多次转储较大的环形磁盘缓冲区，这样就可以起到将其挂起的目的。

通常，环形缓冲区的大小可在启动或运行时对齐进行修改。但是，对于已经丢失的信息，这样来做也于事无补。

例如

在最初设计的产品 Trace.java 中，设计了一个最高级别的例外处理程序，用它来捕获在例外情况下所有未处理的代码堆栈。当这个最高级别的例外处理程序捕获到未处理的例外信息后，它自动将日志转储。像这样的事件所导致的错误和数组溢出登录非常相似。（在转储日志之前，对堆栈处理记录进行一次回溯，因此所发生的错误的确切位置就非常明确了。）

有人鼓励程序员在他们的代码中进行注释。经过注释的例外处理程序将触发转储日志，就是采用上文所述的机制。

在 Encina 的跟踪设施中，在进程（可能是意外）关闭时，所记录的日志可自动转储。它们也可以在一个特定的信息（如错误消息）发生时进行转储。这对于将来很难重现的情况下，该信息对于遇到故障的用户是非常有用的。当下一次遇到该问题时，分析已转储的错误日志可以得到更多的有用信息。

结果

有时您仍然依赖于用户所转储的环形缓冲区，即使在执行该转储计划时因为人为原因而没有执行。

这个时候，就需要发挥“社交工程”，用以说服用户向您发送日志。并及时查看用户友好日志。

三、用户友好的日志

您在产品中添加了一个环形缓冲区分析日志，并热切期待着关于第一个错误报告的一个有用的附件，不管是谁发送的。

因此你增加了一个特性：当程序执行了 DUMPS 环形缓冲区的错误，它将通过互联网转储日志在您的网站，而不是将其备份到用户的计算机磁盘上。如果客服人员泄漏了网址正好被总理浏览到，您可能会被解雇。

问题

如何说服客户给你发送日志文件？

要点

你所在的行业可能是做各方面的事情，这样会使的用户认为他的隐私没被尊重。所以他们不相信你。

自动检索的日志记录的意义不是很大，特别是因为你没有权力控制其他程序员可以选择的日志。

用户希望可能控制告诉你的信息。

程序内部很多数据都是隐藏的，如何向用户解释？

重复审查一个复杂的记录是因为其他方式的计算机太难使用。

因此

做个一个能够被用户理解的日志消息。里面没有隐藏数据（如长字符串的数字，可能是编码的文本）。这种日志最好是格式化的，使其具有易扫描性，从而使读者可以快速地确认问题。如果文章看起来很生涩，那么将不会有人愿意去看，更不用说反馈日志文件。如果有巴洛克式的内容（如 Java 程序包名称，文件名，和行号），在一个专门的地方对信息进行解释，用户很可能会找到它。

鼓励程序员在编写日志消息时考虑个人隐私。例如，没有理由要求在明文中记录密码。存储电子邮件地址是可以避免的。记录日志的主要目的就是记录用户似乎忘记的事情。她的电子邮件并不是其中。网址是有必要记录的——如果用户处处点击，由于他们是如此的相似，以至于用户并不了解他们的意义——在错误报告发送之前，如果用户能够查看一下那简直是太好不过了。

例如

当 Netscape 浏览器崩溃时，Netscape 公司的质量反馈代理[Netscape]就会弹出。他会直接将本次崩溃发送至 Netscape（并不会询问你是否需要发送一封邮件）。在发送之前，您将有机会看到发送数据的内容，这个内容明确的标出：“发送的消息中将不会包括曾经访问过的网址，电子邮件或地址，密码，个人资料等敏感信息。” 然而，至少有一个我认识的人不信任它。他宁愿亲自将日志通过电子邮件发送，而不相信一个程序自身获取的这些信息。这也许是过于偏执，但用户所考虑的和我们的本意是不相关。

我不知道此人是否注意到，您可以将代理人的数据存储到一个文件中。然而，在存储时，它有 187425 字节那么长，包括十六进制和 ASCII 堆栈转储，程序指令，进程列表，设备驱动程序列表，DLL 的名单，等等。内容就像下面的一样：

```
[ C70] 00 00 09 01 00 00 87 3E 00 00 A8 49 00 00 2D 00 [.......>...I..-.] [ C80] 00 00
65 6D 61 63 73 2E 65 78 65 00 70 A3 20 EA [..emacs.exe.p. .] [ C90] B9 D0 BF 01
00 70 F4 0E 00 60 F4 0E 42 05 00 00 [.....p...`..B...]
```

这对于一个多疑的用户，可能内容太长，太详细，或过于隐蔽。

Trace.java 输出到一定的长度，以确保程序员所提供的数据显示在同一行中，在作者看来，奇怪的换行符会误导其他人：

```
=== 1997/09/22 13:52:09 (Controller.event:Controller.java:26) USE gui:
linedraw selected
```

Encina 的跟踪例子显示，生涩的文字和更隐蔽的数字：

```
1 22753 96/01/01-08:17:51.003038 a0040437 W Unable to bind to server
/.:/branch1/server/sfsServer1 (DCE-rpc-0214: not registered in endpoint map)
```

但他们的用户是高级系统管理员。 Trace.java 的应用程序的对象是普通百姓。（该产品从未完成 beta 测试。）

结论

用户可能仍然不会给你发送日志文件。然而，这里有额外的好处：如果日志文件易读，用户可能会自己将问题解决。

四、边缘检查

背景

该系统提供了一个环形缓冲区和转储它的工具。

问题

系统怎么知道要存储哪些信息？

要点

日志太过简单意味着至关重要的信息将永远不会被记录在案。

日志记录的太过详细，则可能使得的重要信息将增加被覆盖的几率。

日志记录的太过详细，还需要程序员花时间添加一些永远都用不上的记录报表。

事前很难知道我们需要什么。

通常，一份关于用户的操作记录是非常有用的，至少它可以表明是用户的什么操作导致了该问题。

程序员会使用日志信息。有些程序员往往是对特殊子系统负责。例如，一个程序员可能是负责网络，一个是负责安全核心，一个是负责图形用户界面。子系统通常是明显大于一个单独的类;因为他们有自己的程序包。

在一个单一的子系统中存在着多项漏洞。这些问题中有一些是因为两个子系统交互影响而导致的。因此，相当大一部分调试信息会给相应人提供错误报告和日志。

因此

如果你不知道应该记录一些什么，那就记录用户界面事件。在大多数程序员看来，用户界面的问题可能会反应与其他子系统之间的关系。在用户界面层所反应出来的问题还不能确定的定位到问题在日志中的位置。例如，大多数程序员并不想知道，特别是在哪里点击了鼠标，点击后在什么地方出现一个对话框，他们更愿意知道是什么操作导致整个对话框被修改了。

下一步，转换成一个子系统的要求，导致它来执行重要的处理。可能你不会记录所得到的结果，这种状态是由相同操作所导致的。你更愿意记录是原因，这种子系统功能状态的改变是由什么操作导致的，特别是如果该状态会影响子系统今后的处理（例如，如果它不仅仅是挂起等待处理而是导致一些其他的问题）。

环形缓冲区日志条目所标记的名称应该根据产生这些日志记录的子系统而得到。明显的，这就与程序员的代码具有更加密切的关系。（测试也往往利用这些信息。）

例如

在 GNU Emacs 的编辑 Emacs 认为，view-lossage 功能显示了环形缓冲区，记录按键，鼠标点击，鼠标是否移动（但不是鼠标移动到哪里），其中所被选定菜单项是什么。下面就是一个样例：

C-x o C-SPC escape > escape w C-g C-hadribble return C-x o C-n C-n C-n C-n C-x o escape x shellreturn down-mouse-1 mouse-movement mouse-1 double-down-mouse-1 mouse-1 escape > C-x C-b C-x o C-n C-n C-n f (lisp SPC rules) C-j escape x view - lossage return

这个功能（和相关的“断点文件”，它可以记录所有的按键）可以用在没有任何其他级别的记录中。

Trace.java 允许子系统记录。该子系统的名字在跟踪输出记录中显示，使之易于略读和过滤。在它本来的产品中是书面描述的，用户界面子系统不记录按键和鼠标动作。相反，它主要记录完成用户登录的操作对其他子系统造成的影响。这样的结果是大大减少每秒在环形缓冲区中经过的日志记录条目，所以默认大小为 500 的消息缓冲区是足以捕捉由于序列的用户操作所导致大多数错误。

默认情况下，其他子系统没有记录其他的访问。但是，有一种机制（两个记录点中有描述），要求有更详细的日志。一些，但不是所有的子系统，会记录跨子系统间的访问（以及其他类似详细程度的事件）。一些测试人员习惯了这种方式。这样不需要大量的不相关的细节描述，这种日志仍然可以具有相当的可读性。

结论

这是一个完美的解决方案，主要有以下几个原因：

1、 如果只登录图形界面，重要的信息将丢失。

2、 随着环形缓冲区填满，新的但无用的信息仍然会覆盖旧的和有价值的信息。

3、 程序员（用户决定是否通过邮件发送日志）将仍然会看一些从来都没有关系的所记录的信息，希望有一天这将是有益的。

捷径和两个记录点有助于解决遗留的问题。

五、捷径

背景

在系统中已经存在了一些日志文件，但却不够详细并且存放在错误的地方。你明白你将有一天重新部署该系统或发布新的可执行文件，所以是值得花时间来

改进日志存储。

问题

如何改进系统的存储日志？

要点

日志是帮助排除故障的。无用的日志信息将有朝一日放慢某一程序员人的调试。

但是，增加一个无用的信息将仅仅意味着将增加更多的无用信息，它可能导致重要信息从环形缓冲区中消失。

有句俗语讲，一个错误聚集的地方，意味着可以寻找更多的错误的最好地方，就是你已经发现了许多错误的地方。

决定在日志消息中存放什么内容并不总是很容易。很难预测这将在以后的用处，特别是对别人也是有益的。

因此

不要对其以后的有用性报多么大的希望。想想过去你所解决的问题就可以了。是不是存在一系列事件导致问题难以重现？你在故障定位、查找可能的错误代码时有问题？如果是这样，增加日志状态，这样会使工作更容易一些。

还要记录有价值的关键变量。这些变量是，那些不同的值可能会导致显示不同的执行路径。这将有助于后来的程序员理解为什么程序运行到这样一个点。

让人厌恶的是，所增加的日志记录是针对特定的错误，而不是解决错误的途径。例如，如果关键的错误是错误的特定实例变量，而你现在所记录的是它在其他地方参与计算或使用的日志。这可能是对将来的错误并无用处，并有可能将有用的详细信息推出环形缓冲区。

以此类推，你应该指出错误的普遍性性与关联性，而不是一个片面的结论，因为，没人知道将来错误究竟会怎么样呈现。

当经过很长一段时间，当你修复了一个子系统中的错误，你可能会发现某些日志没有一点用处。考虑要取消他们。不过，这个时候要小心了，对你没有用处或许对其他人是有用的。

例如

像许多跟踪软件包, Trace.java 具有多层次的记录（见两个记录点）。 在客户的环形缓冲区中，“相近的” 的信息可以被记录在一个较高层次上。在调试环境运行时，一个具体的细节问题可以进行相应的唯一记录。这些信息对不能重现的问题将无补于事，但它们也不会在日志中覆盖其他的信息。他们有助于调试可重现的问题，在这方面并无害处。

结论

系统变的更加容易理解。这种理解可能是千变万化的，即使是添加这个日志文件的人。这些日志文件可能对其他人也是没用的。可能对变化子系统（如极限编程[Beck99] ）之间的人们具有帮助。因此，可以和测试人员讨论日志信息，从另一个方面讲，他们可能会大量使用。

六、两个记录点

背景

您将经常忘记的重要信息记录到环形缓冲区。

问题

开始是一个特殊的时刻。在系统或子系统启动时，关于系统的环境的重要信息会被自然的发现并记录。在环形缓冲区填满时你如何避免失去这一信息？

更笼统地说，可能有些有价值的信息，你永远都不想放弃它。这怎么办呢？

要点

有些旧的消息必须要维护。

判断未来日志信息的重要性是困难的。没有人知道，是一个旧的信息重要呢，还是一个新的信息重要呢，甚或是同样重要呢？

在多处查找信息，这比较混乱。

因此

提供不同标准的日志。记录更加详细的日志，其对应更多的日志条目。例如，一个“里程碑”是一个系统生命周期中所记录日志条目中的一件大事（如一个子系统的初始化或关闭，或建立一个连接预计持续的时间）。里程碑发生的频率很低。低优先级“使用”的日志条目将记录重要的用户界面的事件（如申请一个对话框）。这样的操作会经常发生。低于“使用”级别的是“事件”级别，这是用

来记录子系统接口通道。这种活动水平低于使用，原因是一个单一的用户界面的事件可能会导致一连串的界面点和活动项目。

提供了两种日志，一种是环形缓冲区，另外一种是永久的硬盘存储。

信息以一个合适的优先级分别进入环形缓冲区和永久记录。进入到环形缓冲区，是因为人们还是希望能够按顺序读出它。他们不希望在其他记录中查找是否有更高的优先级信息发生在某一特定环形缓冲区中。

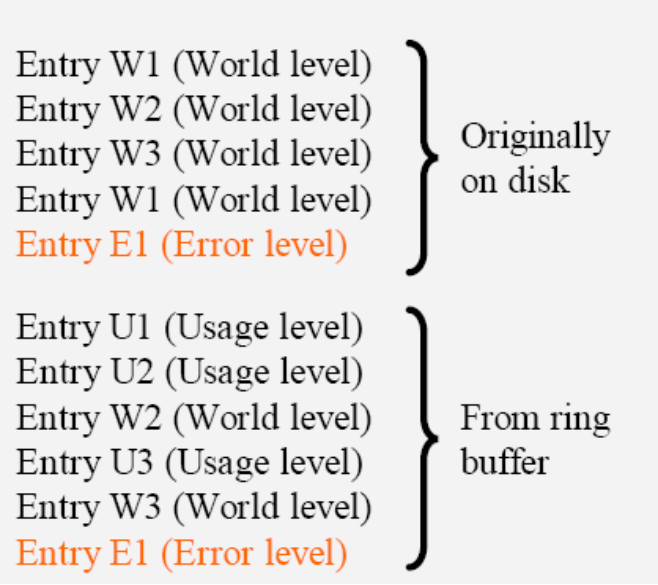
如果代码在自卸环形缓冲区的错误，它应该转储到永久记录中。而这也将会成为单一的档案，用户也可能会向您发送。同样的，如果用户手动转储了日志。

为了避免填补磁盘，永久日志中应该有一个大小，但应远大于环形缓冲区。

例如

Trace.java 基本上是上文所述的结构。优先级是：错误，警告，提示（系统主要的里程碑），使用（用户界面的事件，而不是鼠标事件），事件（内部接口点），调试信息和详细信息。默认情况下，错误，警告，提示的信息都转到指定的位置，这三个，再加上使用信息，记录到环形缓冲区中。其余的信息不做处理，除非他的记录级别被更改。

输入项目混乱是导致环形缓冲区中记录转储到磁盘记录。例如，您可能有这一系列这样的输入项目：



设置明确的分隔符（“环形缓冲区日志从这里开始！”，“环形缓冲区日志到此结束！”）在这两种类型的输出项目之间， 再按时间进行排序，但是这仍然

不能令人满意。一种简单实用的就是按时间戳对输出项目进行排序，然后再删除重复的条目，这样或许最终所形成的产品是经过早期 beta 测试的。

Encina 的东西和这个大部分是一样的。它含有以下跟踪类：检查，错误，严重的，输入项目，事件，中度的，自卸。默认情况下，检查，错误和严重的信息会存储到一个磁盘文件和控制台。其他信息进入环形缓冲区。这些默认值可以更改。不像 Trace.java，Encina 的类别不是全局标准；他们是可以单独切换的。

结论

这样就存在一个专门的位置，用以描述系统问题。它包括起初的高优先级的输入，在结尾处有更详细的描述。

待探索的一个变量

默认情况下，Trace. java 对永久记录并没有限制，但它也可以强制限制在 N 字节内。如果日志文件的增长高于 N/2 字节，它就自动关闭的，并将其重新命名为备份文件，然后重新将本文件作为一个空文件。这是一个简单的近似一个环形缓冲区：在最坏的情况下，你仍然保留有 N/2 字节的记录（或完整的记录，如果它一直都没有达到极限）。

这样的处理方式可能会失去宝贵的启动信息。再想想，有三个文件的话将会更好。第一次将存储原始 N/3 字节的记录，后两个将存储最近 2N/3 的信息 。

感谢

谢谢 PLoP 导师，凯尔布朗，感谢他的建议。谢谢 Ralph Johnson 和我讨论较早不成功版本。本文完成在 PLoP 2000；感谢 Rossana Andrade，odd Coram，Brian Foote，Terry Fujimo，Robert S. Hanmer，William Opdyk，Carlos O'Ryan 和 Juha Pärssinen 。

参考资料

[Beck99]

Kent Beck, Extreme Programming Explained: Embrace Change, 1999. [Emacs]

Free Software Foundation, Gnu Emacs Manual.

http://www.gnu.org/manual/emacs-20.3/emacs.html [Linux]

dmesg(8)manpage,http://howto.tucows.com/man/man8/dmesg.8.html [Marick97] Brian Marick, “The Trace.java User’s Guide”, http://www.visibleworkings.com/trace/Documentation/Trace.html [Netscape] Netscape Corporation, “Netscape Quality Feedback System for Netscape Communicator 4.5”. http://home.netscape.com/communicator/navigator/v4.5/qfs1.html [Transarc97] Transarc Corporation, Encina Administration Guide, Volume 1, 1997. http://www.transarc.com/Library/documentation/txseries/4.2/aix/en_US/html/aetga1/aetga118.htm

嵌入式应用软件的测试策略

一个微型应用系统白皮书

译者：肖艳霞

这篇文章的主要目的是希望能为那些测试和使用嵌入式应用软件的人们提供有用的建议和想法。测试和嵌入式应用软件都是广泛的领域，所以这些建议只能适用于一般情况，不可能对每个人都有用。本文这篇文章首先介绍一些术语，以帮助作者和读者形成共识，接下来会主要谈论以下方面：测试技术，测试工具和测试策略。

测试技术这一块谈论软件测试的一些方法和使这些方法更有效的一些技术，主要包括有：回归测试，基于代码测试和功能性测试。

测试工具这一块介绍几种可增进和支持测试的工具，集中谈论其中的两种。所谓的测试支持工具是指可用于帮助测试员提高工作效率，但本身不可直接执行测试的工具。这些工具通常都是测试员可使用的通用工具。自动化测试工具是指为某种测试类型而特定设计或为某个测试阶段提供帮助的工具。

测试策略这一块研究软件测试的某些扩展问题和讨论这些问题该如何解决。主要包括：何时开始测试，如何处理软件需求不全甚至无明确需求的情况，如何处理测试时间不足的情况，和如何改善总体测试过程以提高总体测试效率等。

一、测试术语

为了精确的谈论测试问题和方法，我们需要事先对测试的一些术语进行定义。在测试中，我们经常会说到“Bug”这个词，但实际上很多人对"bugs"这个词没有很清晰的理解，不明确知道它指的是什么。对于程序设计者，一个 Bug 可能就是源代码错误，而对于一个测试者，它可能意味着代码输出与预期不符。出于这个原因，我们必须进行术语定义，以便于这篇文章的所有读者明确了解我们所谈论的问题。

软件错误：是软件的设计错误，可能是源于需求规格说明书对需求的错误定义或缺少定义，程序员对需求或界面的错误理解，甚至可能是发生在错误时间的中断。这是软件问题的根本源头。

软件缺陷：指计算机程序源代码中的错误。它们由软件错误引入，通常指：错别字、错误的逻辑，错误的等式，甚至一些无关紧要的小问题，如多余的代码。

软件失效：指是软件预期结果与实际输出结果不符。这些不符可能是系统总线信息输出的字节错误，针对特定输入的错误输出，或无法满足需求说明书或连接其它设备的接口的时间要求

失效模式：指软件失效的普遍表现形式，例如，多种软件失效可能导致软件组件无法满足时间需求，另一个例子就是多种软件失效可引发系统冲突。

处理软件错误的方法有很多种，本文将讨论其中的两种基础方法

错误检测或移除：指通过查找软件失效来找到软件错误，然后在源代码中改正这些错误的过程。这种方法成本高，并且需要花费很多时间。

错误避免：指努力地从源头上防止软件错误的出现。通过确认源头的软件错误，且从一开始就尝试防止这些错误出现，这样错误检测或移除方法中需要的成本和时间就可免去，或至少可减少一部分。这种方法不会经常起作用，也不会经常被应用到实际应用中，不过在测试过程中有意识地使用这种方法可以逐渐提升测试质量，可以减少测试时间和提高测试效率。

二、测试技术

我们将会研究软件测试的三种普通类型：

回归测试：在已经发现的软件失效基础上查找更多的失效。这种测试类型的测试效果与用于测试的测试用例的关注点与健壮性，还有指定测试结果的关注点数量直接相关。

基于代码测试：基于源代码的结构和组织查找软件失效。很多时候，采用这种测试类型的测试员需要了解被测试软件的需求，同时要对源代码有详细的了解。这种测试类型归结到白盒测试范围。

功能测试：基于软件的使用目的和特征查找软件失效，这种测试类型的测试依据是用户和客户的需求。

这三种测试类型是相辅相成，互为补充的，这意味着一个完整的测试过程应包括这三种测试类型，并且每种测试类型都在应在合适的测试阶段使用。每种类型都有各自的关键优点和缺点。如果测试过程中，每种测试类型得到很好的应用，

它们的优点可以弥补其它类型的缺点。

回归测试

需要进行回归测试几种原因是：第一，确保旧的失效已经被修正，开发人员在修正旧缺陷的过程中可能引入新缺陷，或只是部分地修正一个缺陷，或完全无法修正一个缺陷，这些现象很普遍。即使开发人员在修正缺陷时高度集中精神，非常小心，还是难免会偶尔出现这些问题，因此，任何时候开发人员修正了缺陷，测试人员都需要对软件重新测试一遍，这样做是值得的。

进行回归测试的第二个原因是为了归纳通用的失效模式。软件的另一个特征是很多时候同一问题症状或软件失效是由多个软件缺陷引起的，所以同一个失效可以出现在不同的时间阶段，可由代码中不同缺陷引发。如果在测试通用的失效模式时，回归测试足够强壮，测试员可能会在运行软件，验证旧失效是否修复的过程中发现新失效。

最后，一种非常常见的软件缺陷表现形式就是结构缺陷。当错误的源代码文件被引入到软件构建过程时，或当应该被引用的源代码文件因疏忽而漏掉了时，这两种情况都会导致结构缺陷的出现。当一份不是最新版本的源代码文件被引用到软件构建中时，适当进行回归测试可找出问题根源，从而可大大减少大家的工作量。

回归测试的缺点是，进行回归测试之前，你必须得至少测试整个软件一遍，回归测试是根据之前的测试结果查找缺陷的，当回归测试得到结果与之前测试得到的结果不一致时，就表示存在缺陷。基于这个原因，回归测试在软件的首次测试或测试软件新完成的部分时，效果是不理想的，即使之前测试得到的系统缺陷可能有时会有用。另外，回归测试对于测试执行方式是敏感的，不同的方式会有不同的结果，如果回归测试只是一味地对一个失效单纯地重复测试，将不利于发现其它相关失效和部分修复的失效。

下面是回归测试的几种策略：

1、尽量使用自动化测试。这样可以有效降低回归测试的成本，因此使得它更容易操作和发挥更大的作用。自动化测试在后面会有更详细的介绍，这里只是

指出根据被测软件的特性和操作环境，自动化回归测试的有几种处理方式。一般来说，自动化测试所需的成本会比手工测试的高几倍，但考虑到回归测试的性质，是针对同一样东西反复进行测试的，自动化测试可以反复被使用，每次使用并不会增加它自身的成本，而手工测试每执行一次，就会增加一倍的成本。

2、设计鲁棒性（即健壮性）强的回归测试用例，不要每次回归测试都只限于确认之前测试发现的失效是否修复。根据失效的类型、发现问题所在的软件结构和失效的功能性方面，设计一套测试用例，用于从不同角度测试软件，以发现更多的相关失效。举个例子，假设一个导航软件在北极使用时存在问题，那么设计一个回归测试，首先在整个北极地区运行软件，然后扩展到北极的方圆 100 码之内，之后在距离北极一英里之内的一系列平行轨道运行软件，最后在南极重复同样的测试。当你从北极到南极时，在格林威治子午线上横跨赤道，和从几个不同的方位跨越国际日期变更线，这样回归测试以能发现一整类的缺陷和极端导航坐标的缺陷及可能同时会发现几个其它的缺陷（或者至少能发现开发人员做这样的测试时找到的问题）。另一个例子：假如在两个模块之间的接口存在一个缺陷，那么设计一个回归测试，通过几种不同的能引发失效的方式和不能引发失效的方式测试这个接口。

3、在软件每个版本发布时都要进行回归测试。这样你对于每个版本状态都能进行检查，使得你可把握每个版本的状态，从而避免浪费时间在存在问题多的版本或正常版本中存在问题多的部分。

4、以合理的顺序执行回归测试，随着回归测试用例库的不断增大，以一个合理的顺序执行测试，使得在这一版本的测试在进行同时，项目的其它工作可以不受影响，继续进行。大规模回归测试的典型执行顺序是：首先执行整体测试，以确认当前版本的所有模块和功能是否已经完成并且是可执行的，然后执行具体的测试，以确认最近两个或三个版本的问题是否已经修复，之后再在所有被修改过的模块上执行一系列测试，以检测这些模块是否出现新问题，最后执行整套回归测试剩下的其它测试。

基于代码测试

基于代码测试（常被视作白盒测试或玻璃盒测试）是根据被测源代码的组织

架构，并且以大型系统的软件预期需求为补充来设计的。因为它依赖于对代码的认知，所以无论哪个测试员，在进行基于代码测试时，必须对于软件和软件工程知识的有详细了解，这一点很关键，同时也要有良好的软件测试知识。

基于代码测试在使软件统一运行，确保所有软件组件按设计者的意愿工作，确保所有组件能正常组合在一起，及测试软件内在符合性这几方面很有用。基于代码测试也能有效地测试模块间和软件子系统之间的接口。

基于代码测试对于代码覆盖率高度敏感。基于代码测试要求一定比例的“关键”代码被执行，如果测试没有涉及到关键代码的执行，那这个测试是达不到要求的，也不是针对该软件的测试。如果对软件关键代码的界定范围不全或界定错误，甚至是随意界定的（例如，允许测试员挑选最容易测试的部分来测试），那么关键代码还是没有被完全测试。最后，要求 100%的代码都被执行，这通常都是不现实的。因为死代码，或除了软件自身发生灾难性故障之外，其它任何条件都无法到达的代码，它们在系统或子系统中可能是不可到达的。大部分这些代码只是在开发人员的单元测试中才可到达。

基于代码测试不需要在目标硬件环境下进行测试，在软件开发环境下，或使用模拟环境代替目标环境进行基于代码测试，这种方式通常是合乎实际并且是合乎常理的。基于这个原因，相对于使用主机环境，使用那些专为测试环境而设计的工具（例如：UNIX 或 DOS 主机工具）来支持部分或全部的基于代码测试，可能会比较好。

使用覆盖率分析工具的好处是可以确保基于代码测试确实是按开发人员的意愿来执行尽量多的代码。这样通过在进行测试的同时提供有用的反馈给测试员，来提高测试的质量，并且允许对测试进行修改以提高代码覆盖率。

基于代码测试的缺点是它对代码的依赖性很强。只是简单的执行一行代码并不能告诉你这行代码是否正确甚至那部分代码是否第一次被执行，即使在测试前确认代码的哪些部分在这之前已经被执行，这对测试可能有帮助，但也无法检测出软件是否有功能性需求未实现或没被正确地实现。而这个问题可以通过以下方法解决：首先进行功能确认测试以检测是否存在某些功能性需求未被实现，其次使用工具例如目录测试工具（见下文）以确认代码测试是否得以正确地实施，这

样做比只是单纯的执行代码效果要好。

基于代码测试并不能代替功能性测试或回归测试。它不能够检测出代码缺漏的缺陷，并且可能存在部分代码是不可到达的，尤其是在子系统和系统级测试中。

基于代码测试在测试代码是否实现全部功能需求的问题，或测试某些系统问题如系统响应时间需求，硬件接口，和负载或压力测试等方面存在不足，而以上测试中某些测试可能是在子系统级测试中完成，所以有必要去确认软件的所有子系统在功能上可以有效结合在一起工作。

进行基于代码测试时可考虑的策略有：

让开发人员做基于代码测试。对被测代码要有详细的了解，这是基于代码测试所要求的，而软件开发人员对代码已经有详细的了解，由他们执行基于代码测试，就可以节省了解代码所需的时间。开发人员经过一定的培训后，执行基于代码测试不会有很大的难度，并且可以在这过程中慢慢提高自己的编码质量（因为他们不得不接受并处理那些因他们失误而造成的后果，从而使得他们在以后的编码过程中尽量避免这些失误）。开发人员做基于代码测试，从而测试员不再需要经历学习和了解整套代码的过程，这样可以节省很多时间，并且可以防止在测试设计过程中因误解代码而不正确设计测试。另外，由开发人员执行代码测试往往会使得对编码质量的关注得到加强，因为直到他们交付经过测试的代码，开发人员才能根据时间安排进行下一步，所以开发人员在检测代码方面的时间相对充裕。然而，由开发人员执行基于代码测试也存在某些弊端。

由开发人员执行基于代码测试存在两个潜在问题，这两个潜在问题就是开发人员可能只是测试他们认为代码实现的功能，而不是测试代码实际实现的功能时，和他们可能只是象征式的执行部分代码而不是对整套代码彻底的执行一遍。避免这两个问题的基本原则是让开发人员有充裕的时间进行测试，并且对开发人员进行有效培训，他们知道需要测试什么和怎样测试。另外，测试目录可以提升基于代码测试的效率。

1、使用测试目录。测试目录是一个标准集合，这个集合描述了不同类型的软件和不同特性的代码需要测试的内容。测试目录刚开始可能会比较粗糙，但在整个软件生命周期内，会随着项目工程的进展而越来越具体，越来越能体现被测

试软件的特性。这些测试目录还是一个良好的基于代码测试知识储存库，能为团队中其它做基于代码测试的人员提供帮助。这样，当面对一种常见的基于代码测试情况，他们不必重复别人操作。关于测试目录的一个很好的例子（通常都会有用）可见 Brian Marick 写的相关书籍。（更多说明见下文）

2、在软件发生变化的同时更新测试相关资料。任何时候软件发生了变化，与软件相关的基于代码测试也要更新相关文档。这个措施可以避免基于代码测试变得过时，而不得不重测的情况发生。

3、使测试自动化。在切实可行的情况下，使用自动化工具来执行基于代码测试，使得版本更新时的回归测试变得简单。这样可以节省测试团队中每一个人的时间和精力，同时可以提供测试定义和测试结果的精确说明。

功能性测试

功能性测试（有时被称为黑盒测试）是基于软件的需求，包括精确的软件需求说明书和软件的执行环境，并且以对这个软件的架构的大体认识和这个软件在更大系统中的角色为补充。虽然这种测试不要求测试员详细了解软件代码（这一点对于基于代码测试很重要），却要求测试员对软件的需求、软件的用途和软件将被如何使用这几方面有非常详细的了解。

功能性测试的最大特点是它独立于代码，并且能从实际使用环境这个角度去测试软件。这两种特性使得它能发现回归测试和基于代码测试所不能发现的问题。如果软件存在整体上的问题，这种问题最有可能是通过功能性测试来发现。如果软件存在软件需求实现不完整的问题或软件需求被开发人员错误实现的问题，则一个高水平的功能性测试可以发现这些问题。

同样地，功能性测试的以上两种特性使得它对软件需求高度敏感。如果软件的需求说明书不够完善，或需求说明书经常作修改，这些将会影响功能性测试的质量，并且会对限制功能性测试的效力。另外，高水平的功能性测试需要参与测试的测试工程师密切关注软件需求。如果测试工程师得不到足够的资源去详细分析软件需求或了解软件情况，测试质量，甚至于软件产品的质量都会受影响。功能性测试实际就是测试软件需求的覆盖率。

功能性测试在执行代码方面的效力是很弱的。高水平的功能性测试很少会达

到很高的代码覆盖率，这是因为条件分支的数目必要由开发人员来考虑和编码。另外，创建通过功能来测试代码的条件需要很高的成本，而基于代码测试的测试员在编码环境下很容易完成代码的测试。这就是功能性测试不能取代基于代码测试的原因。

进行功能性测试时可考虑的策略有以下：

1、设计测试并使其文档化，首先和首要的，对软件需求和软件运行环境进行合理评估，在此基础上设计功能性测试。在测试过程中把需求分配成可内聚的几块，并且不过于强调软件的架构。最好，使执行的测试适当地文档化，这一点很关键。如果测试未能适当地文档化，则我们很难确保测试能达到足够的需求覆盖率，因此很难确保软件的所有需求是否都已经得到了满足。设计和文档化测试的过程往往可发现测试本身的问题。以软件需求为基准进行测试，这使测试员可在需求上的错误和矛盾成为软件的缺陷之前发现它们。最后，这样使得团队的管理人员可以在必要的时候增加新的测试人员，而这些新加入的测试人员不需要经过一段长期的学习过程去了解被测软件。

2、使用系统化的思维方式。当设计功能性测试时，要对被测软件作用的系统有整体的认识。从系统的整体需求中获取负载，效率，和性能等指标数据。设计测试以确保软件能满足以上需求指标，且能显示软件的性能处于哪级水平。识别并确认软件和系统其它部分的接口。

3、站在用户的角度去考虑测试。从用户的角度去了解系统，和从用户的角度去测试软件。用户期望系统实现哪些功能？用户将如何使用这个系统？用户需要怎样组合软件特性和信息？解雇这些问题将对设计功能性测试和核查以确保软件需求说明书无误有帮助。

4、对极端情况进行测试。研究软件需求说明书的极端条件，并对系统的极限进行测试。在极端条件下测试功能以确保软件在这些极端条件仍可正常运行。使软件超负荷运行直到其失效，然后观察其如何失效。它是否是失效是否以一种可接受的方式发生，并且系统会提示问题所在？或仅是崩溃，故障发生时没有提供任何的提示信息和警告信息？功能性测试应该不仅是测试软件正常运行的情况，还要测试软件的失效情况，以判断哪些失效是否可接受。压力测试和负载测

试需要花费很多的人力和时间去设计和执行，但是，这些测试对于确保嵌入式软件能满足实际使用环境的用户需求是至关重要的。

5、快杀测试用于功能性测试缺乏时间和资源的情况。大体来说，这种测试方法工作原理是功能性测试员执行测试，直到他们找到一个严重程度足以阻止被测版本发布的缺陷，以至于有理由“枪毙”这个软件。这个方法存在一些非常严重的问题，而其中最严重的是使得测试人员和开发人员都要承担风险。因为没有任何一个软件版本经过完整的测试，软件也往往从未经过完整的测试，因此无法衡量软件的质量改进过程。这种方法成功与否在于测试员能否快速地找到缺陷。如果测试周期持续的时间足够长，测试员迟早能找到所需缺陷，而无论软件能否正常运行，它都会被测试人员否决。最后，随着开发人员被迫不断修改软件，而测试人员被迫不断地“枪毙”软件，测试人员和开发人员之间形成了一种恶性竞争关系，这种关系会对软件产生不好的影响。非到最后没有其它方法可使用的情况，才考虑使用这种方法。经常使用这种方法必将在软件开发过程中引入其它潜在问题。

6、迭代测试往往与快杀测试组合使用，作为解决快杀测试的固有问题的一种手段。迭代测试的工作原理是对软件的一系列版本进行测试，而不是尝试对软件单个版本完成所有的测试。因此测试员要做的是对每个版本进行研究，尝试确定软件被修改的部分，并且首先测试这些部分。然后测试软件的其它最近未被测试过的部分。理论上说，经过测试软件一系列版本，软件的所有部分都经过了测试。但实际上，这种方法存在很严重的问题，首先，很多时候测试员并不清楚软件的哪些部分被修改了，接下来也就无法确定哪些修改需要测试和怎样测试这些修改。第二，不存在足够的时间去遍历软件的所有版本。最后，测试员往往很难或甚至不可能去估计修改的代码的影响范围，从而据此确定软件哪些部分需要测试，哪些部分可以忽略。

三、测试工具

测试工具可以是任何由简单 shell 脚本创建的程序，这些工具可针对复杂的测试场景生成易于理解报告，这些场景使用了先进的自动化测试和分析工具。这篇文章尝试取两个极端间的平衡点，研究那些当前可用并易用的工具，或那些单

纯针对某一个项目而开发，开发难度适中或容易纳入他们的设计的工具。

在这方面，我们将研究工具的两个大类，支持工具，这种工具不直接用于测试，但通过处理测试中一些琐碎问题，能帮助降低测试员的工作难度和提升他们的工作效率；自动化测试工具，这种工具本身直接协助测试。

通常来说，一种工具的功用或功效与这种工具的成本直接相关，但，这种说法并不总是正确的。我们有可能会找到或开发一些能极大的促进生产力而同时成本又可接受的工具。在任何情况下，当我们考虑是否购买或开发一种工具时，重要的是要对如何使用这种工具有清晰的认知，和了解使用这种工具在进度控制和质量改进方面的预期回报。

支持工具

支持工具一般都价格适中，可用于提升测试员的工作效率，因此，其价格与价值是对等的。这种工具的成本效益比率通常都是高的，在某些情况下，这种工具的开发者开发它只为了方便他们自己在某一方面的使用。

一些有用的支持工具包括如下：

1、数据库工具对于测试人员来说是多用途工具中最有用的一种，在功用方面排名只在文字编辑器之后。数据库工具在收集特殊数据和将这些集合成适用的格式。相关数据库测试应用软件包括缺陷跟踪、需求分析和分配和形成测试目录。作者中见过的数据库工具其它一些更奇特的用法包括测试数据生成，分布式测试控制，以及故障/错误分析等。在考虑选用哪种数据库管理工具以用于测试时，应关注的关键特性有：易用性，灵活性，报告生成能力和可编程性。

2、缺陷跟踪工具用于跟踪软件所检测出失效的修复过程，查明潜伏缺陷的性质和确认根据缺陷代码所需要做的修正。理想情况下，这些工具需要具有通过 E-mail 的自动发送报告这一功能，需要集成配置管理系统和客户需求变更跟踪系统，以提供出现在最初报告中的问题轨迹跟踪和核查不同阶段的测试。大部分工具都不具备上面所说的所有功能，且使用一个简单的带数据库的缺陷跟踪工具相对容易。然而，这些功能应被视为可取的，并应在可能的情况下使用。

3、配置管理工具对于测试员来说有以下两方面的作用。首先，这种工具帮助测试员精确界定正在测试的内容，从而使得更详细的跟踪软件缺陷和软件变更

变得可能。第二，这种工具本身在具有跟踪自动化测试过程和手工测试文档化过程的功能，通过这一功能可确认指定的测试是否执行了正确版本的软件。

4、其它的工具同样是有用的：报告生成和格式化工具用于报告测试结果和生成正确格式的测试数据；计算机语言工具用于编写测试代码，生成测试工具，且有时其本身可生成测试；和其它工具用于分析测试数据，分配测试结果和执行其它的功能等，在这不胜枚举。

自动化测试工具

随着应用软件变得越来越复杂，软件测试相应得也就得更复杂。另外，嵌入式应用软件的测试往往需要接受过高深的专业知识培训和经验非常丰富的测试工程师。对于大部分的 Windows 应用软件来说，具有最基本测试能力的测试员都可测试，但对于在医院中使用医疗设备，甚至是办公室中使用的程控交换机，嵌入这些机器的软件需要经过高水平的测试，因为需要的测试成本也就更高。

不同的自动化测试工具往往都有其独特的功效，有效的运用自动化测试工具可使不可能的测试工作成为可能，并且能保证经过测试的软件版本可成功发布。测试工具永远都不可能取代测试工程师，然而它们可以帮助成倍提高测试工程师的工作效率和允许更多测试用例得到审查。

自动化测试工具的范围很广，从临时为某一被测软件创建的工具，和测试人员用于生成输入数据，通过内部的功能将数据压缩、整理和分析为一个实际可用形式的工具，到先进的特为测试而设计的生产级测试工具，比如代码覆盖率分析工具，内存分析工具，和图形用户界面测试工具将在这一节的余下部分讨论。

代码覆盖率分析工具

代码覆盖率分析工具用于基于代码测试阶段，用来跟踪软件的哪些部分代码已经被执行，和这些代码的覆盖程序如何。这些通常通过使用某种的工具来追踪（嵌入到测试环境的硬件设备或被添加到可执行代码的软件或这两种工具的集成）测试过程中代码的执行情况。接着对数据进行归纳分析，并将分析结果汇报给用户，使用户可确定哪些代码已被执行和哪些代码未被执行。

通常情况下，代码覆盖率分析工具可以提供以下三方面的信息：功能覆盖率，这只能说明一个功能在测试中是否已被执行，而不能说明哪些功能已被执行；分

支覆盖率，这可告诉我们代码中的判定条件的哪些分支已被执行（都不是，真，假，或都是）；条件覆盖率，这可告诉我们哪些条件语句的组合已被执行，哪些未被执行。

在选择代码覆盖率分析工具时要考虑的一个关键要素是这个工具本身会否干扰代码的执行，如何有干扰的话，这个工具会对代码的性能产生很大的影响，因为这种干扰会对测试过程中的代码执行情况产生影响，特别是对嵌入式应用软件中实现实时性功能的代码有着更大的影响，从而导致测试中获取的性能数据未能真实地反映被测软件本身的性能情况。基于这个原因，测试员应该在确定如何使用这种工具和将这种工具用于何处时充分考虑其干扰影响。基于硬件的代码覆盖率分析工具对代码的性能的影响作用非常小，但这种工具通常都很贵，并且对一次可测试的代码量有着严格限制。

内存分析工具

与动态存储器相关的缺陷是最难发现和最难修正的缺陷之一。动态指针的行为是不可预测的，并且经常会不稳定，从而导致缺陷难于重现和难于修正。使用内存分析器，可以快速和高效地的检测和修正内存一整类的缺陷。这种内存分析工具对加快开发速度和提升产品质量有着重要的意义，因为它使任何时候的功能性测试不受内存类缺陷的影响。

内存分析器的另一个特性是它们可用于追踪每个功能的内存消耗情况，识别空指针的使用和其它产生缺陷的条件。这些工具非常适用于内存跟踪。

和代码覆盖率分析工具相似，内存分析器通过软件工具和随机存取存储器的相关部分起作用，因此，基于软件的内存分析器可能会对代码性能产生很大的影响。我们在选择使用哪种内存分析器时可能要考虑这种影响。基于硬件的内存分析器确实存在，然而它们也往往更加昂贵，并且它们受限于运行环境，并适用于所有的运行环境。

我们可以主要地通过编写特殊版本的内存分配和重分配程序来创建内存分析器，这程序可用于测量内存和追踪内存使用情况。虽然这些工具不像一些商用工具那样多功能，但它们非常适用于软件调试。

性能分析工具

计算机程序的性能对于嵌入式软件来说通常是一个需要重点关注的问题。当在软件的执行过程中有必要插入一个中断或在一个帧时间内保持不衰退，软件所消耗的时间量是变得很关键。即使软件的时间预算不紧，但从用户操作的响应时间来说，速度通常都是一个重要的考虑因素。当软件在任何条件下，速度都成了一个问题时，可通过提高关键功能的作用效率来改进软件，这一步将关系到软件能否成功发布。问题是，我们应对软件的作哪部分调整来使速度达到最优化？大部分的开发人员在确定哪部分代码需要优化方面存在困难，这一点已经被有效证实。基于这个原因，这时候我们需要性能分析工具的帮助。

性能分析工具并不是强迫开发人员去猜测，模拟，或估算哪部分代码需要优化，而是提供我们一份详细数据，这份数据是关于软件的执行时间是如何消耗的，在何时消耗，和其中多少的时间是程序运行所消耗的。当我们手头上有一份消耗时间最多的程序清单时，确定软件的哪部分需要优化时间和该如何优化将变得相对简单很多。

性能分析工具关键原理是，对于大部分应用软件，少量一部分代码消耗大部分的执行时间。这些代码在软件所有代码中所占的比例最大可达 20%，最少可达 5%，不同的软件类型有不同的比例。性能分析工具可帮助测试员查找和定位这些代码，并且还可用于集成调试器，仿真器或其它工具，以确认为什么这些代码需要消耗这么多的时间和怎样提升系统运行速度。没有性能分析工具，开发人员往往花费大量的时间去改进代码，却对提高软件性能作用不大或没有作用。

图形用户界面测试工具和测试脚本

为什么我要在这讨论用于嵌入式软件的图形用户界面测试工具呢？因为很多嵌入式软件都提供与计算机相连的这种或那种界面接口，作者自己的公司生产的嵌入式设备同时提供与 UNIX 操作系统和 Windows 操作系统连接的界面接口，以便于用户操纵设备和从设备获取信息。其次，当 shell 脚本和宏服务不运行时，图形用户界面测试工具可用于录制脚本，在开发环境上执行测试。最后，通过录制测试员在功能性测试的手工操作，图形用户界面测试工具比较容易重现和精确记录软件在一个失效被发现前发生的状况，从而提高功能性测试的效率。

一种典型的图形用户界面测试工具包括脚本录制和回放功能，有执行测试时的快照功能（以用于测试完成后的分析，以及与以前的或未来的测试结果相比照。），可允许测试员使用某种编程语言编写测试脚本，可组织和安排测试，还有测试员可自主安排一系列测试的创建和运行次序。

虽然图形用户界面测试工具录制/回放的功能本身很少创建可用的测试脚本，但它们非常适用于录制和回放可导致软件明显失效的用户操作。插入被测软件的脚本工具很少能录制到导致灾难性故障发生的操作，独立的脚本工具则非常适用于捕捉可导致灾难性故障发生关键操作。另外，对那些导致灾难性故障的关键操作的回放，或故障本身可极大地帮助开发人员检测和修改缺陷。

图形用户界面测试工具也可用于运行经过设计和编码开发的用户界面测试。一般来说，这些工具接受以某种测试执行语言编写的脚本化测试用例。脚本经过详细和具体定义，可用于执行和确认嵌入式设备的功能。虽然它通常不被用于录制确认设备功能的操作，却非常适用于对指定时间和指定部位的受控输入进行编码设计。

很多嵌入式设备都没有图形用户界面，而图形用户界面测试工具是在图形用户界面上进行操作的，但这些设备可被插入可执行的测试脚本，这些脚本可为用户提供一种宏观调控能力，或许有时候它可能需要在代码中作一些细小的改变以被执行，但在一般情况下，这种修改的成本是通常都是很低的。当功能性测试员可以通过执行脚本来自动运行测试和记录测试结果时，他们可以节省不少时间，所以说这点成本是非常值得的。很多时候，如果嵌入式系统不能插入某种形式脚本，其组件的确认测试是很难完成的，这时候，把系统进行脚本化的最后方法是把被测试系统或组件嵌入到一个测试环境中，这样可在不影响可交付使用的软件下执行脚本。

无论是使用脚本还是使用图形用户界面测试工具，计划自动化测试一个关键的考虑要素是进行自动化测试对项目成本和时间进度的影响。进行脚本自动化测试的所需绝对最低成本大概是手工执行测试所需成本的两倍，如果测试员对测试环境或自动化工具不熟悉，自动化测试所需的成本可不止是手工测试的两倍。然而，对于一个需要反复执行的测试，编写和修改测试脚本所需的成本与测试自动

运行或测试员只需稍稍关注测试过程而节省的成本相比，相形见绌。另外很多时候，一些测试条件只能通过使用脚本创建，手工无法创建，在这种情况下，脚本化的成本需要与软件未经过那些条件的测试就交付所存在的风险相比较，从而得出需不需要花费成本创建脚本。

四、测试策略

当计划和管理软件测试时需要考虑一些的范围比较广泛的问题，这些问题，虽然很少涉及与测试相关的技术性的问题，却对测试效果的好或差有着很大的影响，并且因此对被测试软件的质量也有很大影响。以下将会介绍这些问题中的一些：

测试目标

测试过程应以在软件产品交付客户使用之前尽量减少其缺陷为目标。达到这个目标的方法应该不只是发现缺陷和报告它们的存在。一个需牢记的关键要素是，在软件开发过程中，缺陷越早被发现，则它们在软件最终发布前就越有可能被修复。这是因为这样可以有更多的可用时间可安排，并因此可在这些问题产生更大影响之前有着更大的灵活度去修复它们。

开发人员对测试人员

基于这个原因，开发人员和测试人员之间的消极竞争是一个严重的问题。如果开发人员迟迟不肯告诉测试人员与被测系统相关的信息或只肯告诉测试人员少量的信息，又或者如果测试人员不肯把他们的经验传授给开发人员（甚至到了不肯告诉开发人员他们正在测试软件的哪些功能和如何测试这些功能的地步），这两个团队之间就出现了消极竞争，并且不管哪个团队赢了，软件质量都得不到保障。

测试人员的早期加入

测试人员参与到软件的早期开发过程是很重要的，这可提供测试人员充足的时间计划和准备测试，确认和创建或获取所需的测试工具，和确保软件测试尽量彻底。这同样对开发人员有好处，当测试人员在进行测试需求分析时，他们通常都会对软件的设计和实施提出问题，这可帮助开发人员审查软件设计和实施是否存在问题。测试员介入早期开发过程的很多益处来自这个阶段，因为测试人员在

对他们提出的问题进行跟踪解答的过程中，不可避免地将发现和报告软件需求规格说明书的错误，编码错误，甚至可能在一些错误导致故障发生之前将它们检测出来。如果测试人员没有参与到软件的早期开发过程中，这将意味着软件产品将不能经过最好的、彻底的测试，和意味着在项目时间进度安排的最后阶段执行测试时项目不得不延期。这样几乎可以肯定，软件交付使用时间将会延后，并且交付的软件将会存在很多问题。

测试文档化

适当的文档化对于成功组织长远的软件测试是必要的。如何测试过程没有形成有用记录文档，在整个测试过程中始终使用相同的方式执行测试，和对测试程序做合理的修改，或甚至确认哪些功能已经经过测试或为什么要测试哪些功能，这些都是不可能的。缺少描述测试标准，测试方法，测试程序和测试机制等的有用文档，将加大引入新的测试人员到测试团队的难度和时间成本，并且使得临时或在有需要时增加人手变得比较困难。

一套好的，基本的测试文档至少应包括以下的内容：

1、一份包含测试设计所需的源信息，测试假设，测试覆盖率（是需求覆盖率还是代码覆盖率要视具体情况而定）等信息，和关于如何启动和操作测试所需工具的信息，以及任何其它相关信息的测试记录。

2、一份用于讲解测试目的和软件初始化条件或安装问题的，描述测试本身的，和提供可用于评估测试结果的信息的测试文档。

3、一份测试报告，是通过详细地记录每次执行测试所获取的信息，包括软件缺陷，测试自身存在的问题，和一些系统辨识信息，如版本号等所形成的。

从 IEEE 计算机协会可获取更多的详细和全面的系列标准。

谁来执行测试？

开发人员和测试人员都应该对软件进行测试。软件的每一块都应有一个或多个团队负责测试，检测缺陷同时也应该是开发人员的职责工作的一部分，他们应该执行低级别的测试，基于代码测试和一些回归测试。并且还可能需要一个或几个团队去执行一个完整系统的集成测试、可用性测试/可维护性测试、配置测试和可靠性测试以及其它一些形式的测试，在这无法一一列举。软件开发过程值得

关注的一点是：在软件产品开发的每个阶段，软件测试都应该和软件开发的平行进行，通过对软件产品进行测试，确认其是否具备进入到下一阶段的条件。如果软件开发和软件测试不能平行进行，故障和错误潜伏的时间可能远远超出原本必要的时间，从而增加项目的成本和开发时间。

缺陷检测/移除对缺陷避免

也许，对于公司产品的长期质量保证来说，测试团队的最重要的贡献并不来自测试本身。如果我们重点关注那些不同软件项目都出现的故障，可能无需测试，就可避免一些故障或错误的出现。这里的目标并不是直接提高测试的质量，而是提高待测软件的质量，这样无需测试，就可直接影响项目的质量。

软件代码总是会存在一些错误，然而，可以使用以上讨论的内存分析工具等工具来防止或大幅度减少某几类的错误进入到功能测试阶段。另外，追踪故障曾经在哪处地方出现，和尝试确认故障是由哪些错误引起的并防止这些错误的出现，可以显著地提高代码的质量。故障分析可引导软件机构从根本上改进他们处理事情的方式，而不只是针对某一个项目的。

分析故障以发现引发故障的错误和避免这些错误的出现或寻找消除这些错误的方法，可以持续地提升产品质量，并且长期来看，可以减少由源代码质量问题所引起的项目延期和成本超支等情况的发生。

进行故障分析的第一步相对比较简单，只是简单地往 BUG 追踪过程添加一些信息，以允许分析问题的根源。这些信息应包括那部分代码出现了失效和这些代码存在什么问题等信息。还需包括出错代码的编写者名字，和其它一些关于故障如何发生和为何发生的信息。当经过几个软件版本的数据积累后，搜索其中有着共同主题或故障发生原因的数据。通常，软件的那些经过多次修改的代码会被检索出来，重新设计这些代码可以显著提升软件的质量，同时可加快项目开发进度。可能有些界面接口缺少相关的说明文档，或是难以理解的，从而引发不少问题，这时需要重写这些接口，使它们得以简化以利于将来的使用。错误的出现也可能和代码没有直接的关系，例如，可能因为软件运行环境存在过多的干扰，或对于特殊代码，开发人员缺乏相关的培训。

当这些错误被逐渐消除后，继续收集数据和评估这些错误移除后产生的作

用，以及确认更多错误的来源。这种处理软件故障和故障的源头错误的方法被称为故障避免。这种方法的基本原理是，在故障首次出现之前消除它比等它成为麻烦后再发现和修改它所花费的成本会少很多。随着越来越多的错误被确认和阻止出现，更多的错误将会出现，并且可得到修正。这种方法的净作用是将会极大地节省项目时间和成本。

测试启动

有时候，测试工程师可能会被要求帮助一个没有正式测试流程的公司建立测试。这可以是一项非常艰巨的任务，同时也可以一项有趣和有价值的任务。以下几个想法可能会有助于测试启动：

首先，确定那公司希望开始测试的原因，原因有很多种，可能因为是最近软件出了一个灾难性的事故，或是因为那公司希望对软件做某些调整，或是因为客户要求进行测试等。作者发现至少会这样一个原因，就是一个公司希望开始测试是因为它们的开发人员坚持为客户提供优质的产品是必需的。确定什么原因促使一个公司的决策阶层认识到测试的必要性，可以让我们知道哪种形式的测试是最适合的，同时它也可以指示出未来问题将会出现在软件哪些功能上。

其次，制定出一个关于如何开展测试的总体规划。从简单处着手，以简单规模的一种测试开始（通常是功能性测试）。往往最好的方法是一个新的工程上建立测试的原型，这可使开发人员事先得到非正式的测试经验，也可允许测试人员的早期加入，并且使得测试人员和公司有机会对测试流程进行试验，从而总结出适用的测试流程以在全公司推广使用。计划中应包括所有所需的人力资源和培训需求，测试所需的工具和从哪可获取这些工具，和测试所包括内容和如何实施的总体描述。在制定测试计划时，开发人员的参与很重要，这样可确保他们了解测试将会进行哪些操作和他们需要提供哪些协助。在开始制定计划之前，要从开发团队和项目管理人员处获取足够的信息。

当原型测试在进行时，要及时记录哪些功能可正常工作，哪些功能出现问题。与开发人员和项目管理人员保持密切联系，以让他们知道测试流程是如何进行的。在测试过程中尽量按照计划来进行，当然随着原型测试的推进，应允许对计划进行相应的调整，这一点很重要。事情并非总可按任一方的计划进行，因此对

计划的调整和修正是必要的。

在原型测试完成之后，和项目管理人员、测试人员、开发人员和其它的涉及的参与者对测试做总结分析，以确认测试的哪部分是成功的，和还需要如何改进测试，把这些都融入到计划中，使它们成为计划的一部分。然后判断是否还需要在另外的项目重复原型测试，否则测试流程可在全公司推行使用。

当确定了测试流程可继续在全公司推行使用后，从新的项目开始，一次只致力于在一个项目上执行测试流程。在旧的项目引入测试流程相对会比较困难，因为这些工程可能会缺少支持完整测试的基础架构。逐项进行的决定必须是在充分考虑如何在旧项目上实施测试后作出，且必须知道这并不容易。

当一个新的项目被引入到测试流程后，要研究被测项目的特别之处。即使是一个小公司，它所开发的项目之间也可存在很大的差异，而这些差异必须成为设计测试方法时的考虑因素。

需求不足或很少时的测试

项目没有明确定义的（或任何）需求规格说明书就开始实施，这很常见，特别是在小公司里面。即使是在大型企业，在他们认为不需要规范领域，也可能存在项目缺少需求规格说明书的情况。在这种情况下如何进行测试是一个常见的问题。

在项目缺少需求规格说明书的情况，并不存在一个做好测试简单快捷的方法，因为需求规格说明书对功能性测试的效力有很大的推动作用。其中关键的一点是要注意保持对需求来源进行追踪，和从这些需求源头上可衍生出哪些需求。这将大大简化对需求合法性的验证。以下是在以往测试成功过的几种策略：

把用户手册当作需求规格说明书使用，这种方法是有效的和符合要求的。如果用户手册提供了足够的细节信息并且被信息组织编排得很有条理，使用它和使用需求规格说明书的效果几乎是一样的。关键是学会在用户手册上系统化地查找需求，确认和追踪这些需求。另外，经常需要从其它的来源获取需求信息来对用户手册进行补充，因为用户手册很少会包含压力和响应时间方面等精确数目信息。

把设计文档当作需求规格说明书使用，大部分的开发人员至少会在某处的文

件上记录或保存系统的一些相关信息。查找出这些设计文档后，它可作为需求的一个源头，特别是对于那些在用户手册上无法找到的硬性指标需求。关键是要小心选择可用作需求的信息，要注意避免把设计信息当作需求。销售文档同样也可以这样用。

与人交谈。小项目的一个常见问题是“两只腿的需求”，这是指长驻客户公司的应用软件技术支持人员，他们围绕描述软件的用途与客户反复沟通。通常这些技术支持人员都会写下一些信息，这些信息可用作需求，但多半时候这些信息用处不大，这需要和他们坐下来谈论一下这个系统要实现哪些功能。另外，走出去和开发人员，系统的实际用户，甚至购买产品的客户等交谈。在每次会谈中，及时作笔记，然后把这些笔记作为进行测试的基础资料。

使用常识。使用常识这个方法应该是所有其它方法都失败后的最后选择。虽然使用常识可能会发现问题，但使用这种方法会引发所发现故障的重要性和关联性的争论，甚至这个缺陷实际上是不是缺陷也可能需要论证。

测试准备不足情况下的测试

测试准备时间不足在软件行业来说很普遍，包括嵌入式软件行业。测试准备时间不足的影响因素数目随软件与上一版本软件或与相关的软件项目的差异大小而定，相应处理这些问题的难度可能在从绝对不可能处理到难度很小的范围内变化。

当这些情况下要求对某个软件进行测试时，首要的工作是弄清楚这个软件是一个怎样的软件和它是做什么的，从而对于充分测试软件时需要测试什么形成某种程度的大体认识，以及弄清楚测试这个软件的成果会对其它项目被测项目时产生怎样的影响。一个关键的考虑因素可能是根本上这个软件是否可测，如果没有足够的时间去为一个全新的产品作测试做准备，唯一可行的办法是向项目管理人员报告这个软件达不到测试的条件，让其决定如何处理这个问题。如何项目管理人员做出的决定是宣布这个产品达不到测试的条件，测试人员需提供详细的信息以解释为什么这个软件达不到测试的条件和需要作哪些改进使它达到测试的条件。

如果软件具有可测性，开始回归测试，如果可行的话，开始自动化测试，如

果可行和合适的话，可把先前开发的系统拿来为这个项目服务，但在说明测试结果时需要额外地谨慎。

确定使用哪种类型的功能性测试资源。如果使用某种资源可增强现有功能性测试的效力，可使用该资源作为测试依据，否则，直接把需求规格说明书或用户手册上作为测试依据。当测试时使用用户手册时，需要根据用户手册上的例子和说明来推测测试条件。

如果被测软件是不可测的，不要测试它。说明其不可测的原因及需要补充什么条件你才能测试它。需要向开发人员和项目管理人员都说明这些情况，以确保下一步采取适当的行动。

时间不足的测试

测试的另一个常见的问题在项目时间不足的情况下执行测试。如果执行测试的时间严重不足，取消回归测试和基于代码测试，直接跳到功能性测试阶段。如果执行全面的功能性测试的时间也不足，是使用快杀测试方法，还是简单地向上级报告测试时间不足，无法完成测试，从而推掉这项测试任务，需要在这两者之间做一个抉择。这两个方法都是解决问题的容易方法，但都会给你带来麻烦。如果测试团队就这样推掉太多项目的测试，将存在一个严重的问题，就是测试队伍是否还有存在的必要，项目还有没有测试的必要。如果使用快杀测试方法，最后可能会以测试团队，开发团队和项目管理人员都受到惩罚而结束，同时软件产品的质量却没有得到多少的提高。

当决定使用快杀测试方法时，最好的方法是结合迭代测试方法进行测试，并且把项目当作一个新的项目来系统化地测试代码的不同部分，这样至少可确保功能需求的覆盖。虽然这样做还是存在风险，但从长远来看它可能是唯一可行的方法。

五、结论

那么，所有上面所谈论的意义何在？希望这篇文章的读者可以从中获取到一些对他们有用的知识。无论是学到如何挑选和使用一套测试工具以帮助加快测试速度，还是学会如何往你已经在进行中的测试添加健壮性强的回归测试或基于代码测试，甚至是学会如何在开发团队中开展测试。总之，作者的目的是给读者提

供一些可用于提高测试的效率和成功率的想法。

希望读者已经把握以下几个关键点：首先，测试只采用一种技术是无法完成全面的有效测试的。第二，有效的选择测试工具可以显著地提高测试的效率，使用测试工具所获得收益与支出的价格成本和培训所花费的人力和时间成本相比，往往是值得的。最后，希望读者能思考一下测试策略的全局性问题和找到适用于提高读者所在公司的测试成功率的长远测试策略。采用以下两个关键的策略将获得明显的收益：建立一套方面好的，实用的与测试相关的标准文档，并且发展一套致力于避免缺陷的全面方法，而不满足于仅查找和修正缺陷。

读者若对本文简略提到的一些观点有兴趣，想获得更多信息，可在下面找到这些信息的来源，另外，欢迎通过电子邮件或普通邮件随时与作者联系，可对本文提供您的宝贵意见，也可就一些测试问题向作者咨询，或与作者讨论感兴趣的测试领域。

更多的信息：

与测试相关的信息来源有很多。在测试领域内，对测试的研究开发正在持续进行，不少公司正在研究生产用于或适用于测试的新产品，并且新的测试技术正在原有的技术基础上不断发展。最近两本谈论测试不同方面的书籍是：

软件测试的艺术（The Craft of Software Testing），Brian Marick 编写，Prentice Hall1995 年出版，ISBN：0-13-177411-5。这是一本很好地讲解了基于代码测试的书籍，所提供的信息远多于本文所能提供的。本文所提到的关于基于代码测试几种方法都源自这种书。这本书的附录还包括一个稍加调整和扩展即可用于嵌入式软件测试的测试目录。

测试计算机软件（Testing Computer Software）（第二版），Cem Kaner，Jack Falk，和 Hung Quoc Nguyen 编写，Van Nostrand Reinhold1993 年出版，ISBN：0-442-01361-2。这本书涵盖的内容非常广泛，包括测试的组织和管理。书中对于测试设计和测试控制有很好的和全面的讨论。

另外，还有测试新闻组，测试协会，和包含测试讨论的普通协会。在这些测试协会中，其中最好的是位于美国波特兰俄勒冈州的太平洋西北部软件质量协会（Pacific Northwest Software Quality Conference），其测试新闻组网址是

comp.software.testing。另外，comp.software-eng 上有大量测试人才在交流，comp.risks 上经常会有一些与嵌入式软件测试相关的文章发表。

如何聘用优秀的性能测试工程师？

译者：陈能技

朋友最近需要聘请一些性能测试工程师，问我：“一个优秀的性能工程师需要怎样的素质？”在我思前想后最终想到一些东西之后，我觉得有必要跟大家分享一下我的看法，你也可以加入你的意见，这些都有助于老板们在雇佣性能测试工程师时作出精明的选择。

需要声明的是：这里列出的东西目的不是要作为人力资源部的正式职位描述，而更多的是关于原则和概念。

技能方面

你需要一位了解最新的计算机技术和概念的人。他需要熟练地安装操作系统（包括 Windows、Linux 等），自己动手设置网络，为什么这些是重要的呢？因为他往往在工作中需要自己搭建一个测试的实验环境。

网络知识要点——你需要一位全面了解 OSI 模型的人，他应该知道 TCP/IP，需要知道 DNS、DHCP、WINS、路由/交换器/网路集线器，并且知道他们的工作原理。为什么这些是需要的呢？因为他可能需要用到网络嗅探工具来定位网络瓶颈所在，那么很明显，他需要知道自己在“嗅探”什么。作为性能测试工程师，在碰到一些简单的网络问题时应该能自己解决，而不需要把负责网络的工程师拉过来帮忙，他应该能自己解决类似 LoadRunner 中 Controller 和 Load Generator 之间的连接问题，只要知道网络接入、IP 地址设置等常见的问题就能解决。

协议——最低要求是：他能够对项目产品中用到的那些协议轻易地创建测试脚本。当然，最好是掌握更多的协议，有各种各样的协议测试脚本开发经验，例如 Winsock、COM、HTTP、Citrix 等，因为不知道什么时候也许就能用上这些东西了。

虽然我不要求他是一位“代码狂”或者开发爱好者，但是他应该可以看懂 HTML、ASP、JSP、JAVA、C 等代码，并且可以弄懂代码中的来龙去脉。因为这些东西不仅对于测试脚本开发来说是需要的，而且对于定位代码瓶颈尤为重要，很明显，他对代码懂得越多，能发现的问题就越多。

SQL 方面的知识（包括查询语句、存储过程、索引、数据库管理、备份还原等）。数据库是复杂应用系统中造成主要瓶颈的几个原因之一。在这方面找出造成瓶颈的原因一般来讲是 DBA 的事情，但是如果你的性能测试工程师对此一窍不通，也不知道如何与数据库打交道，则他可能就把一些关键的东西忽略掉了。

他需要“统观全局”。他应该知道自己在 SDL（软件开发生命周期）中的角色。他应该知道开发人员、项目经理、QA 和系统管理员都是做什么事情的，并且知道如何跟他们打交道。有时候，你可能会发现有些技术方面很强的人，他们在自己的“小天地”中很牛，但是也就仅仅看到自己的那片“小天地”，而不知道对其他人在组织层面上的影响。

他应该能非常熟练地使用你们公司所选择的测试工具。如果他掌握了其中一个，其他的也会比较容易掌握，但是最好是选择那些至少有一年实际使用经验的人。

非技能方面

跟上面说的技能同样重要的是：性能测试工程师应该懂业界常用的性能测试、性能调优、容量规划方法和过程。而不仅仅懂得按一个按钮执行测试。

他应该掌握一套计划、测试和调优最佳实践和方法论，并且可以根据公司的实际情况进行调整、定制。当然，如果聘用的是入门级别的人则不需要这些也可以，但是他们需要按照公司的现状来建立这方面的内容。

一个好的工程师应该永远都是一名顾问，即使他的顾客是内部人员。如果他不把每个人都当成是顾客的话，也许你把他请进来就直接造就了一场与开发组的不间断的战争。他应该是温顺的、懂得变通的、能承受压力并保持冷静的、对人尊重有礼貌的人。

性能测试小组的终极目标应该是让产品发布前的每个人都保持最佳状态，为发布一款性能优越的产品而努力。作为回报，每个人都喜欢性能测试小组的人，因为他们帮助大家让程序跑得越来越快。其实这就意味着你聘请的性能测试工程师必须拥有良好的沟通能力，他们应该被认为是帮助别人的协作者，而不是被看作整个项目的“挡路石”。

我会比较喜欢引入那些有激情的人。我一般会找那些不断扩充自己知识的

人。当他们来到一个项目中，对于那些没有接触过的协议，他们会感到很兴奋，因为这意味着他们有机会学到一些新的东西。他们会与其他公司的、网络上的性能测试工程师保持联系，构建一个属于自己的良好的技术支持系统。他们经常参加各种各样的活动、用户组和会议。

最后，还有其他一些简单的要点：他应该有能力提醒和催促、要求别人做一些事情，而且是在能让别人心甘情愿地、乐意地接受的情况下。他应该尽自己所能去帮助别人，即使需要牺牲自己很多的时间，花费大量精力。他应该非常乐意分享知识。他知道什么时候做领导者，什么时候做跟随者。这些都是在聘用时需要考察的内容。

小结

我非常幸运可以碰到很多真正优秀的性能测试工程师。那些全职的顾问往往都保持忙碌的状态，因为在对待工作方面他们有相似的素质。我想更多的是他们的非技能方面的能力，而不是他们的技能方面的能力，让他们保持领先、站在更高的位置。关于一个好的性能测试工程师的素质要求，你有没有什么其他的建议了？如果有，欢迎发表、一起讨论。

译者后记：

这是 2004 年的一篇文章，但是今天看起来仍然非常实用。最近大环境不是很好，很多人尤其是大学毕业生找起工作来都比较吃力，这篇文章是关于如何招聘一名优秀的性能测试工程师，但是我觉得更加适合那些希望找性能测试方面工作的人读一读。看自己在技能方面、非技能方面还有哪些欠缺。

一个合格的性能测试工程师所要求的知识面是非常广的，要想成为一名优秀的性能测试工程师，没有一两年的努力是无从谈起的。很多知识我们在大学也学了，但是没有掌握，出来找工作时才后悔当初没有好好学。当然，还有很多实用的技术方面的东西在当今的大学教育体制下很难学到，需要个人努力或者通过后续的培训掌握到。另外，正如文中提到的，很多非技能方面的东西是需要工作中慢慢体会和掌握的。

如何在不可能的期限内完成无文档的软件测试？

译者：成韩丽

前言

我对于这次的讲演没有太多的想法——大家的意见多一些，我只是提出了自己的见解。毕竟，这对于测试者而言是一个持续的在不断的投诉中进行的测试……

现在我们开始吧。你是如何在一个比较紧张的期限里完成无文档的软件测试？

我认为答案取决于你所处的环境。你最好的解决方案可能是技术上的、行政上的或者是 resume-ical。这些都取决于你以及你们公司如何陷入这个混乱的，如果这算是一个混乱的话。

一、什么引起了这样的情况？

让我们带着几个问题来认识这个原因：

- 为什么软件没有文档？
- 为什么你只有如此少的时间？
- 对于这个客户，质量问题是什么？

二、为什么你只有这么少的时间？

可能有以下几种可能

- 发布的时间推进竞争
- 现金流转推进发布日期
- 关键财政里程碑推进完成日期
- 相信你从来都是不满意的，从而使你的时间表与此无关
- 相信整个测试组是不在控制之中的，而且需要紧缩预算来控制
- 相信你有一些错误的优先事项（例如文书工作好于寻找缺陷的能力）
- 相信测试是不相关的
- 或者对于最终用户你缺少常规上的关注
- 也许你有适量的时间

三、这个客户的质量问题是什么？

- 内部的，“内部”外包的
- 外部的，定制的
- 外部的，大包装的，包装的
- 外部的，大众市场的，包装的
- 这个市场的质量要求是什么？他们失败的成本是什么？

——用户群体、软件商店、销售电话、电话支持服务

从长远来看，很好的理解一个市场的质量要求，会帮助你提升你的信誉和权威，因为它影响到贵公司的成本。

四、行政上的方法：购买时间

如果你处理的是一个个别的超出控制的项目，在这样的情况下有很多的方法。

如果你处理的是一个常规的（涉及政策的或者公司有标准的），这样的情况下就没有太多的方法。

五、购买时间：延迟过早的发布-2

走高端的道路

- “我希望在胸前看到刀子而不是背后。”—Trip Hawkins，电子艺术的先驱
- 真诚的交流
- 避免使人们总是惊讶
- 从来都不要破坏项目
- 不要变成“敌人”：如果你是令人讨厌的、自私的，你将会变成一个工具被有些人利用，并且你将会是随意被别人支配的人。

六、购买时间：延迟过早的发布-3

- 寻找表面上的制止者。如果可以的话，奉献你的专家小组，以完成这项任务。
- 广泛的分发推迟的缺陷清单
- 考虑书写中期和后期项目的评估：

- 测试的程度（按区域）
- 过失的程度（按区域对剩余缺陷的预测）
- 延期的缺陷
- 可能在黑盒/白盒测试以及审核测试的经验以外的

七、购买时间：延迟过早的发布-4

- 做经常性的有效的现状报告。如下列表：
  - 你的团队的问题（由于交付给你，所以事情放慢或者被阻止了等将影响你得进展，在哪些领域你是落后或者提前于计划）
  - 项目问题
  - 缺陷统计
- 寻找盟友（考虑质量相关的成本）

八、如何在发布时签署文件？

- 不要签署。
- 不接受有权签署。
- 当你（明显）不是 QA 的时候不要假扮 QA。
- 将自己定位为技术信息支持者。
  - 发布预发布报告，提供稳定的产品的数据以及完成测试范围的数据。
  - 发布一份报告，在报告中展示出客户看到产品后会被杂志审评（或者其他第三方）可能发现的问题。
  - 让想过早的承担责任的人拥有这项权利，因为你的责任是向他们提供他们需要的信息，以便他们做出决定。

九、技术上的方法

- 查找参考文件（你可以得到大量的数据，即使你不能得到规范。）
- 重复利用项目的材料。
- 为探索测试做计划
- 列出自动化成本效益
- 利用工具快速的找出缺陷
  - 页面的语法检查程序，spider 等等。

- 便利视察
  - （当这些缺陷找到你之前获取它。）
- 覆盖合理的问题

十、查找参考文件

- 凡是出现规格的文献
- 软件变更备忘录，这些记录了每个新的内部版本的程序
- 客户使用手册草稿（包括以前版本的手册）
- 产品文献
- 发布风格指南和用户界面标准
- 第三方产品兼容性测试套件
- 发布的规章
- 内部的备忘录（例如项目经理给工程师描述的功能的定义）
- 营销演示，产品管理销售的概念
- 缺陷报告（他们的反应）
- 逆向的工程师得计划
- 和以下几方面的人见面，例如：
  - 开发主管
  - 技术资料编写者
  - 客户服务人员
  - 事项专家
  - 项目经理
- 阅读头文件、源文件、数据库表定义文件
- 阅读你所使用的所有的第三方工具的规格和缺陷表
- 原型，实验室的原型
- 行业专家
- 与最终版本的开发人员会面
- 查看以前版本的客户通话记录，在外地发现的缺陷都有哪些？
- 可用性测试结果

- Beta 版本的测试结果
- 为了你的产品中的缺陷以及一些共同的缺陷，或者是你的平台的缺陷，查看 Ziff-Davis 急救中心的 CD 和其他技术支持的 CD
- BugNet 杂志/网站的共同缺陷
- 在新闻小组、在线信息服务论坛等，寻找对于你的产品的缺陷报告以及其他产品的缺陷报告，并且讨论有些功能如何实现。
- 本地化指南（或许在你的平台上本地化产品已经发行）
- 得到兼容性设备名单，至少应该从理论上的市场营销的环境中得到
- 查看兼容性产品以找出他们的缺点（然后找出在你的产品中的这些缺点），它们是如何设计那些你不理解的功能的，以及他们是如何解释那些设计的。看看 listserv 的、新闻的以及 BugNet 的等等。
- 与你相似的产品进行准确的对比
- 内容参考资料（如地图以检查你的在线地理计划）

十一、可重复使用的测试矩阵

输入数据栏															
		无	磅值 LB	UB 的值	LB 的值减一	UB 的值加一	0	负数	数字或者字符的 LB 值	数字或者字符的 UB 值	空的区域（清空默认值）	字符可能的最大值	没有数字		

十二、项目组管理的方法

- 你能够增加人数吗？
  - 他们会允许你增加吗？
  - 你能吸引他们吗？

——做一个工作流程分析指出任务的哪一部分能够分离，并且将这部分委派给新来的人员。

- 井井有条
  - 使用功能概述
  - 或者使用详细的项目计划
  - 创建和发布明确范围的报告
- 取得关键流程的控制
  - 例如：发布管理

风险测试——四个常见问题的解决方案

译者：沈佳容

风险测试可谓是一项困难的技术，既须要预估可能导致一件产品发生故障的原因及后果的严重程度，还要开发执行测试用例来证实该产品是否真的会由于这个原因而发生故障。这个过程在特定的开发环境中执行起来会更加困难，因为还要顾虑到时间、资源的压力，以及可能存在的各种各样不同的观点。尽管如此，这还是一项值得深究的技术。因为它关注风险，将精力集中于测试任务的重心，即快速发掘重要问题。

通过我个人经验总结，发现测试人员和测试经理在运用风险测试方法的时候常会碰到类似的问题。就像下面四个典型问题，让我们一起来解决它们。

问题 1：“每个人都认为风险测试属于管理范畴。”

我经常听到有人讨论风险测试的时候将重点仅仅局限于风险测试管理——如何根据产品风险来决定测试策略——怎样的测试活动是最有效的，以及我们应该执行到什么程度才算合适。例如您可能根据风险决定了人员的分工——四个测试人员执行网站的性能和扩展测试，仅一人执行功能测试。尽管这可能是个好策略，但是仅根据这点，测试人员知道怎么做吗？是否有必要进行风险测试设计呢？

解决方法 1：同样基于风险进行测试设计

我认为风险测试不仅包括风险测试管理，还包括风险测试设计。

进行测试设计的时间点位于管理之后。它是一个测试用例设计的过程，用于挖掘特定产品风险的相关信息。重点关心的是产品是否会像我们所担心的那样，由于某种原因而真的碰到故障。输出结果是一系列的测试用例。

完成这一任务的其中一种方法就是建立一份风险目录（也可称之为 Bug 分类）。这是一张很简单的表格，列有产品可能碰到的各种种类的 Bug。我所说的 Bug，即指任何可能威胁到产品价值的东西，包括故障、失败，或线程环境。我编写风险目录常用的格式是“[什么]可能导致[什么]”。如果我有更多的信息，如

导致问题的原因，或问题可能受到何种因素影响等，我都可能将其做到目录里面去。当我针对每个风险都进行了一番详细描述后，这个目录就算完工了。但是通常情况下我更偏向于使用简单的短语，或词组碎片来进行风险描述。

Cem Kaner 发行过一本名为 Testing Computer Software（测试电脑软件）的书，其附录内有一份非常详尽的风险目录。但该目录所列内容未能涵盖后期的网络时代。之后， Cem 的一位学生， Giri Vijayaraghavan，对原有的目录进行了补充，加入了一部份非常实用的风险目录，该部分目录是专门针对电子商务网站购物车进行测试的，其中包括有以下几项：

- 包含有购物车数据库的系统没有足够的存储空间
- 磁盘故障/硬盘崩溃，以及购物车数据库内其他不可撤销的媒体数据损坏所导致的数据丢失
- 购物车数据库备份的损坏

Giri 列表里的每一项都与购物车可能进行的功能测试有着直接或间接的关系。尽管这不是一份测试目录，但是我们可以这样问自己“我们所进行的何种测试中发现的问题是与该目录有关的？”这个问题并不难回答，因为这份目录已经足够细致了。

我所担心的是如果我总是靠其他人的风险目录，是不是太无能了。所以，我选择不完全照搬 Giri 的分类，而是会花上几小时的时间完全熟悉产品的功能集，然后自己思考其可能存在的风险，制作自己的目录。完成后再与 Giri 的目录作对比，察看是否有重点被我遗漏了。因为这样做，我能更深刻地理解 Giri 的目录，同样也是对自己工作的负责，甚至还有可能发现 Giri 所未能发现的风险。

当您制作完成风险目录后，就能决定哪些项目是值得测试的。然后为其设计测试用例，来回答这些问题：我的产品是否可能发生这样的问题？如果可能，其结果的严重程度如何？

问题 2：“我的风险目录简直乱七八糟”

定位和分析风险的方法有很多。我喜欢讨论风险分析的推理过程，以及如何让这些过程发挥作用。但是当您和其他人一起制作风险目录的时候，情况又不一

样了。这一过程不仅仅是推理过程，而且还是团队合作的过程。不同的想法会磨出各种不同的火花，而这些火花会发生交汇，同样也会发生冲突。

我的同事 Bret Pettichord 在近期的项目中碰到了这样的事情。“会议上，我们和测试组共同制作风险目录，”他回忆道：“我们有六个人，很快就生成了一份很长的风险目录。困难的是如何对这份列表进行管理，如何将这么多项的工作进行下去。”

他说：“我们有许多项都是模糊不清的。甚至还有‘不明所以’的，就是先前不知道谁因为什么原因将其加入了列表，当我们后来再回忆的时候却忘记了。另外，我们列表里还有各种不同种类的东西，包括风险、测试用例、产品功能和质量标准（如可扩展性或安全性）。我们需要对何谓风险，以及风险与结果的关系有更清楚的了解。”

Bret 经常会提到“风险”。因为测试就是有关查找产品问题的，而风险测试就是专门针对产品风险方面的问题。能明白这点就是个好的开始。在头脑风暴中所得到的想法往往比较笼统，但事实上这并没有错，因为头脑风暴的核心目的是尽可能不要遗漏掉重要的想法，甚至可能涵盖上百个奇异的想法。问题是您不可能将头脑风暴所得出的目录直接生成测试计划。“我们习惯于参照测试用例或 Bug 列表来进行工作，” Bret 说：“所以当我们开始做事的时候，就要先对头脑风暴所得出的目录进行检查筛选，理出工作的头绪。但是我们又不知用何种方法来进行。”他所面临的挑战就是如何将这份目录转换为实际可行的东西。

解决方法 2a：分类

我们可以按照头脑风暴中所讨论事情的不同来对目录进行分类。最简单的就是使单张目录中的事情仅仅与一种行动相关连。这样我就可以针对一张目录来进行工作，而不会混淆。

我使用的目录就像这样：

产品风险目录：产品可能出现的问题（如：“在常规登陆的情况下，服务器太慢” ）。由于产品风险可能驱使生成测试用例或使用某种测试技术。在风险分析阶段，产品任何特定的需求都有可能是一个风险。对于任何需求而言，只要不得到满足就可能有风险。如果某项需求不可能有风险，或有风险的可能性极小，

就没有必要把它作为风险列入表内，或者将其和其他风险组合一起列入，这样会比较有用。

您所要做的：对于表内的项而言，您只要选择以下一条执行即可：

- 建立测试用例以评估风险
- 进一步对风险进行研究（不进行必要的测试）
- 接受风险（根本无须测试）
- 将风险提交给其他人（也许让开发者对产品进行重定位，以减少风险）

在项目进行过程中，您可以将产品的风险目录作为报告的基础。当项目完全结束，管理者更可以从风险目录的状态看出您的工作成果，以使其对产品有足够的信心。

风险因素目录：可能造成或加大产品风险的环境（如：开发者以前没有接触过网站服务器）。您不必因为一个风险因素而进行测试，而是需要在存在风险因素的环境下进行测试。例如，“功能 X 非常复杂”并不意味着该产品存在特定的问题。而是表示在这种复杂的代码环境下，任何种类的问题发生的可能性会更大。另一种风险因素是“威胁”—— 由于某种环境或输入使产品收到某种程度的负面影响，并使产品出现了潜在的弱点。要区别是风险因素还是产品风险，最好的方法就是先问问自己，如果将其作为缺陷提交的时候会有什么结果。如果结果非常明确，是“非 Bug”，那么它更有可能是风险因素。

您所要做的：因为风险因素会导致产品风险，您就可以利用该目录中的项来推测出可能存在的产品风险，及其潜在的严重程度。同时还能帮您思考降低风险的方法（尽管这可能超出了测试的范畴）、待处理程序的易测性，以及可执行的测试用例。例如，“瘫痪的客户端”可能是头脑风暴中得出的风险因素，我准备将其列入风险因素目录，然后设想瘫痪的客户端会带来怎样的问题？我们如何可以察觉到这种环境？我们如何可以创建这种环境？

风险区域目录：相似的产品风险的集合（如：性能）。危险区域内大多情况下可能存在多个产品风险。在风险分析阶段，质量标准的一些类别，如兼容性或性能，都能被视为风险区域。换句话说，没有满足兼容性的标准，我们就需要将“兼容性”（如果您喜欢还可以使用“不兼容”）作为风险区域包含在内。

您所要做的：该列表中的项可作为风险测试计划的标题，下面可包含特定的风险集。风险区域可以帮助您总结风险和相关的测试战略。对于任何风险区域而言，您应该能够设想出产品发生故障的各种可能性。如果您不能，那么它可能就不是真正的风险区域，或是您所收集的信息不够充分。

问题目录：一张包揽所有需要进一步调查或解答的总表，用于推动项目进行（如：我们是否有足够的预算用来购买一个负载测试工具？）。

您所要做的：解决问题，收集更多相关信息，或直接忽略它们。这是项目管理中的常规事务。

产品部件或功能目录：可能与风险有关的部件（如：网络服务器）。有些部件可能与一些很复杂的风险相关联。试图找出那些部件并做成目录。

您所要做的：您可以在风险测试管理中，使用该目录定位到产品的具体部件。我喜欢总结出一到两页的部件目录，并为每项标注资源需求为“高”、“普通”或“低”。当头脑风暴中提到某一个产品部件时，你们可以就该部件是否仅为“普通”风险进行讨论（通常情况下，我假定每项都是需要测试的），或者该部件的风险是否高于其他部件。

测试设想目录：测试的建议（如：负载测试）。针对风险所提出的测试设想是值得记录下来的，其很有可能对以后的工作会有帮助。当风险分析过程中有人提出测试用例，就需要追根究源，尽可能分析出其所对应的风险是什么。

您所要做的：将列表中的项转换为测试战略或具体的测试用例。

项目风险目录：影响项目结果的风险，非必要的产品操作（如：未能发送需求文档）。当在产品风险分析过程中谈论到类似该类风险时，请将其记录进目录。

您所要做的：作为测试管理，推进项目或监督质量可能不是您的本职工作。尽管如此，当头脑风暴中有人提出项目风险时，还是需要记录的。后期，再看看列表中的那些项，看看哪些已经威胁到了项目中的测试部件（测试经理已将其列入问题目录），哪些会威胁到后期的项目（您可以向项目经理或其他人员提出）。

风险降低：为了降低风险您所需做的事。这有可能已经超出了项目测试的范围。

您所要做的：如果您对自己的项目不是很有把握，就最好将某些有用的想法

告诉项目经理，让他来进行下一步的操作。

以上这些可能并不是一份完整的目录。如果您在头脑风暴过程中发现有其他的内容需要另外归类，您可以重新建一个新的目录分类。对于所有新的目录分类，您都应该明确其作用，否则不建也罢。

您可以在进行头脑风暴时建立这些目录，待头脑风暴结束后再进行目录分类，或者仅仅针对一类目录进行头脑风暴。当然，您也可以将所有都集成在一张目录内，只要您清楚自己所要做的事。

解决方法 2b：明确参考框架。

要记住何谓风险，有一种定义我比较喜欢：一件事件造成破坏或伤害的可能性或概率。对于任何产品风险而言，我所要立即知道的就是：“会发生什么？发生的可能性多大？”这些问题就决定着原因和影响。人们之所以会搞混风险列表的原因之一就是未能明确原因和影响。

例如“错误输入”，这是原因还是影响？错误输入是否是可能导致错误的事件，还是用户界面设计简单所造成的影响？或是由于其他已发生问题的子系统的输出所导致的？此项是否有可能成为产品风险，全决定于您的参考框架，并没有一个确切的答案。人们所感知的产品都有一条长而曲折的因果链，而所有的测试都是围绕着该链进行着。

你可以通过对普通链的理解，以保证您测试的有效性：受害者 <— 问题 <— 弱点 <— 威胁

■ 受害者：受到问题影响的某人。到项目的末期，可能除了那些对人们有影响的 Bug 外，其余都不重要了。

■ 问题：产品的某些方面未能符合我们的预期。（也称之为“错误”，但严格来说，某些问题可能并非错误。）

■ 弱点：在某种环境下，产品中存在的问题，或存在问题的某方面（也称为“故障”）。

■ 威胁：一些环境或产品的外部输入，会发生问题，或会引发脆弱产品中存在的某些问题。

把这些词连在一起，我们可以说：产品由于受到某种威胁而出现弱点，而由

于该弱点所引发的问题使操作者成为了受害者。这是关于风险的一个简短的故事，我们所要做的，就是在想到和风险有关的事情时，准确地寻找到其在这个故事中的位置，然后试图发掘出其他故事情节。要做到这点，您需要设置您的参考框架：首先确定您所讨论的是什么产品或什么子系统——是什么东西存在弱点、面对威胁、存在问题，且影响着受害者？

参考范例

我们假设您的团队进行了一次头脑风暴，结果是一个长长的目录，就像这样：

1、 性能和可用性

2、 非常大量的事务

3、 对于网站服务没有完全理解!!!!

4、 双曲线树部分不可能进行自动化回归测试。对于源代码也没有控制力。

5、 网络服务失败

用两分钟时间对其进行分类。

1、性能和可用性	风险区域。接下来需要收集更多的相关信息。
2、非常大量的事务	可能是风险驱动、测试想法、产品风险或产品性能。还需要连接上下文进一步分析。
3、对于网站服务没有完全理解!!!!	听起来像风险因素，即某人认为这点比较重要。也可能是一种论点、项目风险，或产品性能。需要更多信息。
4、双曲线树部分不可能进行自动化回归测试。对于源代码也没有控制力。	两个分项被捆绑在一起：第一项看上去像一个论点，或一个项目风险，第二个像是项目风险，也有可能是风险因素。两者结合在一起，也可能提醒需要对双曲线树部分测试特别注意（如果我们已决定该项为测试重点）
5、网络服务失败	可能是任何意思。要通过上下文明确该意思。

我相信其中的大多数项都可以通过扩充或内容设定，变得更加充实。在某些案例中，通过项目团队内的讨论可以为原目录引申出数十条新项，并且可以为测试提供更好的建议。

问题 3：“我项目组内没有人愿意讨论风险。”

风险是可怕的。如果您是一个设计者，且刚刚成功说服大家在产品中使用一项未经试验的新技术，您可能就不愿意当众承认其可能带来的风险。如果您是一个客户，可能不愿意听到您即将使用的产品存在很多风险。如果您是公司法务，您可能会因为风险目录而惶惶不安，唯恐其会引来一张法院传票。我曾经见到有些高层管理人员希望，且假设所有的风险都处于“已管控”状态，其实“已管控”就意味着这些风险已经被排除了。

让人们去直面风险可能是件困难的事情。即便他们能够面对，让他们在某个理性团队内讨论风险也是很难的。您很有可能被看成“消极派”，形成误解。

解决方法 3：将注意点从风险转移到选择

不可能对所有东西进行同样程度的测试。即使您能做到，也会花费大量人力、物力。风险测试人员的标语是：让我们做出明智的选择。

如果人们都不喜欢讨论风险，好吧，那就讨论选择。没有人可以逃避选择。我们应该测试什么？需要进行怎样的测试？为什么针对这个需要进行更多的测试？除了勉强人家来讨论风险外，还可以来讨论一下这些问题。

另一种策略是独自进行风险分析，然后宣布您选择测试些什么。仔细聆听管理人员和开发人员的反应。这就是“石头汤”的方法——先假装煮石头来诱使人家提供煮汤的原料，最终成就一锅鲜汤。如果人们可以对我的风险目录发表想法，或是来反对我的选择，我就能得到更多关于风险的信息。

问题 4：“我不知道其实那个不是风险。但那真不是风险吗？”

每个风险开始时都是非常轻微的。在进行测试前，您不会知道产品会有哪些 Bug。您甚至不会知道 Bug 产生的可能性有多大。您可能只知道或许会出现某种问题。而就是这样才会给我们风险分析带来困难，因为不确定性可能感觉就像是产品风险。

但是您所知道的风险和您害怕可能成为的风险又是不一样的。就像您看到一只大黄蜂在围着您的卧室转，您知道自己会有被蜇的危险，这与您不确定是否自己家中有黄蜂是不同的。

进行这些区别又有什么用呢？其实，在当我们试图区分风险严重程度时就用得上了。当您试图比较两种风险的严重等级，而其中一种是您所了解的，对于另一种风险您所掌握的信息还非常不够，如此就很有可能高估那类信息不够的严重程度，致使风险分析整体的可靠性下降。

解决方法 4：根据对风险的了解程度的不同，进行分类

当我对产品风险进行分类的时候，我会使用两种尺度：重要性以及已获信息程度。我经常用“多”、“一些”、“少”、“无”来区分已获得的信息。（我总喜欢谦虚地评价）

通过用信息掌握的程度对风险进行分类，可以使我们的目录更加全面。当我们评估风险时，对已知风险的评估也会更有信心。如果说我们将一个了解不详的风险评估为严重，就意味着我们对其有顾虑。通过进一步的研究，包括测试，我们可以获得关于该风险更多的信息。而我们测试人员的目标就是将所有重要的风险转换为熟悉的风险，即对该风险的各方面都了如指掌。这样或许我们会发现有些风险可能根本不算风险，或是确定某一风险非常严重，需要管理人员和开发者着重关注，并采取行动来改善该产品这一薄弱环节。

透露个风险测试的大秘密！

已经介绍了很多问题的解决方法，现在我再给出一个很有帮助的建议：风险测试期间，错误的风险分析是一个不断完善的过程。之所以这么说，是因为如果您所进行的测试不是完全基于风险的话，就很有可能会漏掉很重要的问题。那么您就会说，下次让我们集中更多精力来测试这个问题。学习的过程是循序渐进的。所以当您刚开始对一条新的产品线进行风险测试时，不要对自己太苛刻。一些风险您或许可以预期，但更多的可能不行。没有方法可以预知每个风险，只有更加缜密的思考，并让人们了解到您的结论是通过分析和经验累计而来的。

一次的失败会为我们未来的测试铺路——只要我们用心去吸取经验。切忌 Mark Twain 说过的一句话：“我们要学着从经历的世事中仅提炼出智慧，摒弃其他——就像坐在火炉上的猫，它再不会第二次坐或添那滚烫的火炉——因为它提炼出了智慧；但它同时也再也不敢接近任何一个冷却的火炉——这是它所不该提炼出来的。”

Web 应用的安全性测试入门

译者：Yuyifan

介绍

随着越来越多的重要数据都存储在 web 应用上，以及网络事务数量的增长，适当的基于网络应用程序的安全性测试也变得相当重要。安全性测试是指机密的数据确保其机密性（例如，不是将其暴露给不恰当的不被授权的个人或用户实体）以及用户只能在其被授权的范围进行操作（例如，一个用户不应该能够单方面有权限屏蔽掉网站的某一功能，一个用户不应该能够在某种无心的状态下改变网络应用程序的功能）的这样一个过程。

一些在安全性测试中运用到的重要的术语

在我们说的更深入之前，了解一些网络应用程序的安全性测试中使用频繁的术语会很有帮助：

1、什么是“易受攻击性”？

这是网络应用程序的一个软肋。造成这个”软肋”的原因，可能是程序中的 bug，一种注入（SQL/脚本代码）或者已存在的病毒。

2、什么是“URL 处理”？

一些网络应用程序通过 URL 在客户端（浏览器）和服务器端之间进行额外信息的传递。有时在 URL 中改变信息可能会导致服务器不可预期的结果。

3、什么是“SQL 注入”？

是指通过网络应用的用户接口插入 SQL 指令到服务器所执行的查询中去的过程。

4、什么是“XSS（跨站式脚本攻击）”？

当一个用户通过 Web 应用接口插入 HTML 代码，而这种嵌入其中的代码对于其他用户是可见的，被称做 XSS。

5、什么是“欺骗”？

完全仿造网站或电子邮件的赝品被称为欺骗。

安全性测试方法：

为了对 Web 应用提供一次有效的安全性测试，安全测试人员应当对 HTTP 协议有很好的认知。对于客户端(浏览器)和服务器端之间如何运用 HTTP 通信有很好的了解是非常重要的。除此之外，测试人员需了解最基本的 SQL 注入以及跨站式脚本攻击。如果运气好的话，在 Web 应用上发现的安全漏洞的数目不会很多。不管怎样，能够对所发现的问题精确具体地描述安全漏洞能提供相当大的帮助。

1. 密码破解：

Web 应用上的安全测试可由“密码破解”开始。为了登录应用程序的非公开领域，可以通过猜测用户名/密码或者利用一些密码破解工具来达到相同的目的。常用用户名和密码列表与开源密码破解工具一样有可用价值。如果一个 web 应用不强制要求复杂的密码（例如，包含字母，数字和特殊字符，包含最小字符长度要求），那么这个用户名和密码不用花费很长的时间就可以被破解。

如果用户名和密码被不加密的储存在 Cookie 文件中，入侵者可以通过不同的方法盗用 Cookie 文件里面的信息，比如用户名和密码。

更多信息请关注文章“Website Cookie Testing”。

2. HTTP GET 方法上的 URL 操作

测试人员应当检查应用程序是否通过查询字符串传递重要信息，这应当发生在应用程序通过 HTTP GET 方法在客户端和服务器之间传递信息的时候。信息是通过查询字符串的参数传递的。测试人员可以修改查询字符串的参数值来检查服务器是否接受了传递过来的信息。

通过 HTTP GET 请求的用户信息传递给服务器进行身份验证或数据提取。攻击者可以操作每一个通过 GET 请求传递过来的输入变量，以获得所需的信息或进行数据破坏。在这种情况下，任何由应用程序或 web 服务器产生的不寻常的行为都是攻击者进入应用的大门。

3. SQL 注入：

第二件应该在 SQL 注入中检查的是，在应用程序的任何一个文本框输入一个单引号应当是非法的。反之，如果测试人员遇到一个数据库错误，表示用户输

入插入了一些查询并被应用程序所执行。这样的话，该应用容易受到 SQL 注入的攻击。

SQL 注入攻击是非常危险的，因为攻击者可以从服务器数据库获取一些重要的信息。检查你在 Web 应用上的 SQL 注入点，找出你代码库中那些能够直接在数据库上接收用户输入而执行的 MySQL 查询的代码。

如果用户输入数据被用作查询数据库，攻击者可以注入 SQL 语句或者伪装 SQL 语句为用户输入从数据库提取重要信息。即使攻击者成功的摧毁了应用，在浏览器上显示了 SQL 查询错误，他们仍可以获取他们所找寻的信息。鉴于这种情况，用户输入的特殊字符应当被适当的处理或者避免。

4. 跨站式脚本攻击(XSS)：

测试人员应当额外的检查对 Web 应用的跨站式脚本攻击。任何 HTML，比如<HTML>或者任何脚本，比如<SCRIPT>，应用程序都不应当接受。不然，该应用很容易受到跨站式脚本的攻击。

攻击者可以利用此方法在被攻击者的浏览器上执行恶意代码或者 URL。经此跨站式脚本攻击，攻击者能够利用像 JavaScript 之类的脚本来盗取用户 Cookie 以及存储在 Cookie 中的信息。

很多 Web 应用从不同页面的一些变量里传递或提取用户信息。

比如：http://www.examplesite.com/index.php?userid=123&query=xyz

攻击者可以轻松的传递一些恶意输入或者如同查询参数的<脚本代码>，把重要的用户/服务器数据公布在浏览器上。

重要提示：

在安全性测试过程中，测试人员应当特别注意不要去修改如下的任何一条：

- ✧ 应用程序或服务器的任何配置信息
- ✧ 服务器端处于执行状态的服务
- ✧ 由应用程序管理的现有的用户或用户数据
- ✧ 除此之外，应避免在生产环境上做安全性测试。

安全性测试的目的是发现 Web 应用的易受攻击性，这样开发人员能及时地移除这些缺点，从而保护 Web 应用和数据在非授权范围的安全性。

恰到好处的测试框架

译者：赵岗耀

在加州 Sunnyvale 做测试管理一年后，我准备再次跳槽。我来到弗吉尼亚州，在那里我已经被可靠软件技术雇用，该公司因致力于实现更高品质软件的工具和咨询而闻名。我将花费大部分的时间在我的咨询角色上，这是一个和软件管理完全不同的工作。但这将是一个有趣的工作，然而，不同的是因为我从来没有在一个可靠性领先于市场、或者甚至符合市场的环境中工作过。

我因我的非常好的质量模型（“更好的品质：超越口号”《计算机》，1997年8月，页码：96-98）而出名，所以可能看起来比较奇怪，RST（译者注：可靠软件技术的简写）聘用了我。起初，我以为CEO（首席执行官）Jeff Payne已终止他的《计算机》订阅，因此没有看到我的质量见解。但事实证明，RST相信，一个非常高的可靠性标准——对他们来说——只不过是更好的质量的一部分。

对于质量的同样的思考适用于，无论我们把我们置身于质量等级的何处：在生命周期的每一点，我们必须比较产品当前质量和进一步改进的成本和价值。（编者语：我要区分最好的方法和其他方法，这些方法指出，我们应该测试，直到管理部门从我变冷的、麻木的手指中撬走产品。）

时间的意义

在跳槽到 RST 后，我面对的挑战是能够阐明、或许量化，为什么某些要求的测试技术可能会对一个任务苛刻或生存苛刻的项目产生效用，如果是的话，多少测试是必需的。大家都知道，在原则上和实践上来说，穷尽的测试都是不可能的。但依据实际情况，充足的测试是可能的。

我要区分最好的方法和其他方法，这些方法指出我们应该做我们能想到的各种测试，并确定我们找到的每一个错误，我应该测试，直至管理部门从我变冷的、麻木的手指中撬走产品。我所可以告诉的是，这是一个方法，它从不知道和不关心实则需要指明的时间。

这里有大量的来自“穷尽论者”观点的对更好（测试）的批评。但是穷尽论是不负责任：测试经理拒绝面对穷尽的测试是不可能的选择这样一个事实，而不

是拒绝寻求一项不可能的标准的测试。这是一个纯粹的行政庇护：当管理部门将产品运送给必然的反对者时，测试经理对于在实地发现的每一个错误可以归咎于管理部门："我告诉过他们，它需要更多的测试！"

在测试行业，我们实际上是在为如何知道需要指明的时间而努力。在过去几年中，Cem Kaner（《测试计算机软件》，第 2 版，国际汤普森计算机出版社，1993 年；和《拙劣的软件：当软件失败时该怎么办》，John Wiley & Sons 出版，纽约，1998 年）、Brian Marick（《软件测试技艺》，学徒讲堂，上鞍河，新泽西州，1995 年）和我一直在讨论和争辩它。

我们和越来越多的测试人员、测试经理和顾问开会和工作，他们认为自己是最好的提议者，但我相信时间最适合提出恰到好处的测试的特定模型。

恰到好处的测试的定义

任何情况下，恰到好处的测试会提出问题，"我怎么知道是否我正在做的，或已经完成的正确类型的测试已经足够？"遗憾的是，这里没有客观的或严格的算法来回答这个问题，但是我们在尝试回答这个问题时，可以确定哪些因素将被考虑。我们至少可以开始建立一种围绕这个问题的启发式框架。事实上，我的更好质量的一般模型也能适用于更好的测试。我从它借用了某些要素，但在这里我提出了一个更具体的框架。

作为第一步，我将定义专业术语：

恰到好处的测试是用合理的成本，开发一个充分的质量评估的过程，以使对产品作出明智的和及时的决定。

让我来解释这个定义。我主张，在较高的水平，把测试的价值看作一个动态的四个部分是有益的：

- **产品质量的评估**（它的准确性和完整性如何？）。
- **测试成本**（它的合理性如何？它在项目的限制范围内吗？是否有一个良好的投资回报率，如每次测试获得丰富的信息？）。
- **决定**（评估服务于项目和业务的作用如何？）。
- **所有上述的时间**（它足够快所以有益吗？）。

测试还支持做出决定以外的业务流程。但在这个小段中，我把任何测试客户

对测试结果的使用归总于“决定”栏下，如创建正确的营销资料或提高我们的能力来提供技术支持。

一般来说，如果质量评估越精确，测试就越好，测试成本就越低，作出决定的基础就更好，时限就越短。完美的测试将即刻、便捷地提供正确的信息，允许任何业务部门作出有关产品的必要的决定。

根据这一定义，完美的测试在一些急迫的项目中很容易实现。一个例子来自我的同事 Doug Hoffman，他曾处于这样一种状况，他被告知，他做的测试不会影响产品发送的决定。因此，他宣布测试完成。

在另外的情况下，也许持续的测试会提供一个有益的技术支持或为其他一些类型的企业决定提供依据。这一点是指，当测试和作出决定不再有联系，并且不再提供数据以供将来使用，那么测试就没有任何用途了。

另一种太常见的情况出现在，当一个组织或监管机构需要某些特定的测试或测试产品，即使它们对项目没有多少或根本没有帮助。虽然我可能明白他们的行政需要，这样的测试产品还是很少如我建议的那样去做。恰到好处的测试是有意识的和有目的的测试，而不是迷信活动和宗教仪式。我已经看到了的大多数测试计划，可以撕毁和扔掉，绝对不影响测试项目或任何利益相关者。（编者语：在生命周期的每一点，我们必须比较产品当前质量和进一步改进的成本和价值。）

在许多情况下，测试计划被编写是因为有人说：“规则书说我们应该有这样一个东西。”几年前，我自己写过几个，所以我的意思不是在宣称好像我从没被“阴暗势力”所诱惑。但是，作为不断成长的专业人士，我们应该争取贡献更多的价值和更少的杂乱给我们的项目。

评估的组成

认真讨论恰到好处的测试的问题首先包括评估四部分的定义，然后决定它们作为一个整体，是足够得好，还是值得通过改进测试过程而加以改进。您可以为我能想到的任何测试方法套用这个分析。

1、评估产品质量

- 我们如何评估和报告产品的质量？
- 我们相信我们的质量评估从我们的意见看是合理的吗？

- 当我们需要知道时，我们能知道明示的和隐含的产品需求吗？
- 在产品中的重要问题被创建后，我们如何迅速发现他们？
- 我们的测试覆盖了我们需要覆盖的产品的方方面面吗？
- 我们充分利用了各种不同的测试技术或质量信息的来源来消除我们测试覆盖的差距吗？
- 产品中可能具有的我们不知道的重要问题的可能性有多大？
- 那些本应该我们的测试首先发现，但却通过我们测试过程以外的其他手段报告的问题是什么？

2、评估测试成本

- 测试要花费多少？多少钱我们能承受？
- 我们怎样才能消除我们的测试覆盖中不必要的冗余？
- 是什么使得难以（并且因此昂贵）进行测试？
- 产品如何才能被制作得更易于测试？
- 是否有工具或技术，可使过程更有效率或成效？
- 我们很早或者等到很晚才开始测试，总的来说，哪个将使测试不太昂贵？

3、检测测试如何更好的支持决策

- 测试过程知道管理部门、开发者、或其他客户需要做出的各种决定吗？
- 测试过程关注潜在的产品和项目的风险吗？
- 测试过程与变更控制过程和项目管理相关吗？
- 测试报告递送及时吗？
- 测试报告以易于理解的格式传达吗？
- 测试过程和测试的结果一样被传达吗？我们报告包含其中的我们的评估和我们的信心的基础吗？
- 测试过程服务于技术支持、出版、销售、或任何其他应该使用质量评价的业务过程的需要吗？

4、时间安排如何？

本模式的其他三部分的每个方面都是时间驱动的。这就是问题所在：我们从

来没有足够的时间来做每一件事，所以我们所做的每一件事都是在与时间赛跑。

归总

在评估的下一部分，考虑你对以上问题的答案，并提问："我们的测试有多好？"

我们的测试有多好？

- 针对上述问题，测试过程有一些迫切的问题吗？
- 如果产品质量低于管理者想达到的目标，我们的测试过程能充分警示他们吗？
- 一些类别的潜在问题是不能容忍的吗，如果是的话，我们相信我们的测试过程将找到所有这些问题吗？（编者语：测试经理拒绝面对穷尽的测试是不可能的选择这样一个事实，而不是拒绝寻求一项不可能的标准的测试。）

当更多测试者由衷地选择面对和我们不期而遇的得失时，我们将建立一个足够好的行业。

值得改进吗？

- 什么战略我们可以使用以改进测试？
- 我们如何能够实施这些战略？我们知道如何进行吗？
- 改进测试需要多少成本或遇到多少困难？是最有效地利用资源吗？
- 我们能从现在开始以后不断改进吗？我们能在可接受的时间框架内实现改进吗？
- 改进如何才会适得其反，例如引入错误，损伤士气，或使其他项目挨饿？
- 我们应该特别改进什么？改进它有其他方面的益处（如更好的士气）吗？
- 改进将产生显著的不同吗？

我喜欢可以适用于任何类型的软件项目情况的模式。我保证这一组问题是值得考虑的，无论项目是生命苛刻的或者只是市场苛刻的。

然而，通过比较测试披露的信息和其他方式（如客户体验）所披露的信息，事后回答这些问题常常很容易。在以市场为导向的软件世界，大多数软件都经过

开发和发行的许多次迭代，所以事后的认识可能很充足。

迭代提供了一个评估和改进测试过程的方式。但在任务苛刻和生存苛刻的世界里，面临的挑战是如何知道，在做第一次测试时，测试是恰到好处的。在今后几年我们将为这一挑战而努力。

所有这一切我的主要关注，除了对 RST 有用外，是帮助软件测试专业工作自身脱离政策、主观性、和自我防护，而应用结构和推理到困难的、多层面的问题中。

当更多测试者由衷地选择面对和我们不期而遇的得失时，我们将建立一个足够好的行业。

James Bach 是可靠软件技术一个主要的 SQA（译者注：软件质量保证）人员。通过 j.bach@computer.org 可联系他。