目 录

测试架构支撑商业成功（第一部分）

作者：架构师 Jack

什么是测试架构？商业成功的关键是什么？测试仅仅是找 bug 吗？经过多年商业环境中测试工作的经验和实践，我将在本文分享心中的——测试架构支撑商业成功。

首先，请大家看图一，后续的内容将是对图一的一个全面阐述，告诉大家测

图一

什么是商业成功？

以我这些年在工业界的经验和体会，我总结出一个机构的商业目标要成功必然要有两个阶段：第一阶段是吸引客户；第二阶段是留住客户。如果我们不能很好的吸引客户购买你的价值，那就没有任何商业价值了。但是吸引了客户后，为了让商业目标能持续下去，就需要你能继续留住客户，因为留住客户，才能吸引真正大部分的客户，你的商业价值才能越滚越大，才能基业长青。我们可以看 2 个案例：百度靠自己独特的功能，和较高质量的中文搜索结果，吸引了第一批客户，第一批客户又由于搜索结果的满意度，而口碑相传为百度创造了第二批，第

三批客户。阿里巴巴依靠自己独特的功能和定位，以及配合营销的支持，吸引了第一批阿里巴巴的客户，在第一批客户得到满意的服务后，才能支持后续不断扩大着新的客户。倘若阿里巴巴的后台系统质量欠佳，例如：经常导致客户的网上业务中断，或是导致客户的数据丢失，或是客户的网页被黑客修改。那么可以肯定，阿里巴巴的现有客户会放弃阿里巴巴的服务，同时，质量差的口碑也会传播开，导致无法吸引新的客户。那么目前为止无论是百度还是阿里巴巴都在不断的扩大着自己的客户群，就证明了它们两家公司的测试质量还是令人满意的，没有因为质量拖累了商业目标的达成。

那么我们测试人员在这两个阶段如何发挥作用，体现自己的价值呢？

在“吸引客户”阶段：吸引客户的关键是产品的规划设计，规划的产品能解决客户的某些问题，这样自然会有客户愿意尝试产品，并成为第一批客户。在这一阶段，如果你不是一个刚进入工业界的新人，而是一个工业界的“老兵”测试人员那么你可以发挥你过去在其他项目中积累的对客户需求的理解，以一个代表“用户”的角色与老板，与开发负责人一起商讨产品应该具备哪些功能，如何才能满足用户的需求。对于能力很强，经验丰富的测试人员还可以参与产品的架构评审，设计方案质量的把关。确保产品的设计满足最初的需求规划，产品的设计缺陷不会留给用户。我把这个阶段的测试活动都定义为“测试驱动设计”。

在“留住客户”阶段：其实留住客户的贡献，测试人员从一个项目开始就在支持了。从代表“用户”来敲打需求的必要性和需求的正确性，到对产品进行的各种类型的测试，目的都是尽可能的保障产品的质量，不让产品缺陷伤害到客户。我把这个阶段的测试活动定义为“测试驱动开发”。

测试驱动设计——“吸引客户”

此处的测试驱动设计，不是写一段测试代码或一个工具来动态测试产品的设计方案。而是采用静态测试的方式，利用测试人员特有的产品系统经验，与开发人员一起完善产品的设计质量，需求质量。我把测试人员在产品设计阶段对开发人员的影响力，称为“测试驱动设计”。“测试驱动设计”解决的是产品设计质量的需求，确保产品的设计能“吸引客户”。在我的 blog 中有一篇文章，《测试架构师与产品架构师的合作》有更多的介绍。

http://www.51testing.com/index.php?uid-293557-action-viewspace-itemid-179482

这是测试人员未来发展的一个方向，更高价值的体现。成为产品设计的主导，而不只是在最后阶段发现一些程序员粗心大意的错误。

测试驱动开发——“留住客户”

测试驱动开发不仅仅是 TDD 的范畴，不只是测试人员先写好单元测试代码

然后进行 TDD 活动，测试驱动开发还包含后期系统黑盒测试。大多数单元测试只能发现编程技术上或人为粗心的问题，很难发现产品需求和设计方案的问题。因此，我把单元测试和使用一些测试工具对产品进行的黑盒测试都定义为“测试驱动开发”。“测试驱动开发”解决的是开发质量的需求，确保客户不会因产品实现过程中的人为错误而受伤害，离开我们。

“测试驱动设计”如何实施？

需求 / 规划 / 设计 / 编码阶段的测试——“吸引客户”

需求阶段、规划阶段、设计阶段、编码阶段开展的测试在国际上有一个专业的词汇叫“early testing”，大家有兴趣可以在 google 上搜索“early testing”可以有更多的了解。早期测试与我们平时传统测试的区别更多在于测试在产品周期的哪个阶段介入。是开发人员做还是测试人员做更好呢？我依然认为还是专业的测试人员来做更有效，因为测试人员具有更好的发散思维和更严谨的思维，很适合在早期阶段进行测试。虽然欠缺一些产品代码编写的经验，但不影响早期测试的开展。最多在编码阶段的测试活动上效率低一些，但单元测试代码本身编写并不复杂。关于如何开展早期测试，欢迎大家关注我的 blog 文章之一《需求质量如何测试》：

http://www.51testing.com/?uid-293557-action-viewspace-itemid-197367

“早期测试的目的”就是要确保产品后续的活动是在一个正确的方向上。

“测试驱动开发”如何实施？

调测阶段的测试——“留住客户”

后期测试主要就是大部分测试人员日常所做的测试活动，这一阶段的测试活动基本都属于动态测试，测试技术的入门门槛并不高，但却是保障产品质量的最低底线。后期测试包含了平时我们所涉及的：性能测试，压力测试，可靠性测试，兼容性测试，测试方案，测试用例设计，等价类划分等活动。任何一个公司都必须把这些测试活动做扎实，才能守好最后的一道门。相对于早期测试活动的开展，对测试人员能力的要求相对较低，不存在是否测试活动能够有足够技术积累支撑开展的风险，所以成为了主流的测试活动。但是后期测试相对早期测试的不足就是：发现产品 bug 的成本高，发现产品 bug 后修复的成本大和难度大，而且发现一些产品设计缺陷的难度比早期测试大。因此，如果有条件，测试团队应该尽力抽精兵强将在早期测试阶段进行投入，能有多大能力就投入多大，不让“缺陷越早发现，成本越低”成为永远的不去努力的目标。

“后期测试的目的”就是要确保产品实现工作是在既定的方向上得到了无偏差的实现。

静态测试——“吸引客户”

早期测试的主要测试活动都属于静态测试，例如：需求评审，架构评审，代码静态检视，代码静态测试工具都是属于静态测试。静态测试是所有测试活动中发现缺陷时间最多，效率最高，缺陷定位时间最短（发现缺陷了就定位缺陷了），修改缺陷成本最低（只有一些图线或文字描述的修改）。但是静态测试对人能力的要求却更高了，因此目前在大部分中国的公司还无法开展起来。对于已开展静态测试的测试人员，要清楚你正在做的工作是“吸引客户”的第一步，这样会给你更大的责任感，让你更有动力做好静态测试的质量。静态测试的结果就是确保你们在做正确的产品，能满足新客户的需求，吸引到新客户。

动态测试——“留住客户”

动态测试顾名思义就是需要软件动起来，才能进行的测试。我们传统的黑盒测试都属于动态测试，一些动态测试工具能帮我们大大提升黑盒测试的效率，如：动态内存自动检测工具，压力测试工具等。采用动态测试基本都是在产品的后期阶段了，这时测试关注重心是产品得到了正确的实现，能留住老客户，而不会因为修改了一个 bug，引入了新缺陷破坏了老客户原有功能。

目前为止，本文的上半部分结束了。从宏观层面给读者们分析了测试支撑商业目标的主要活动类型，以及各阶段测试活动的价值和意义。我们要在确立测试架构时，可以从以上大类划分测试活动的组合策略，清楚明白后续测试计划所解决的商业问题。

后面的内容将是本文的下半部分内容，从微观层面分解支撑测试架构的具体测试活动。

一、测试架构的国际通用标准和活动

TMM（测试能力成熟度）：是类似于CMMI的一套评估体系，可以帮助我们了解当前所在的测试组织在国际标准中所处的成熟阶段。

ISO9126：是一个国际质量标准，它定义了很多非功能属性的内容。可以帮助我们完善产品非功能属性的设计，以及非功能测试所应该包含的内容。

TPI（Test Process Improvement）：是一个国际测试改进活动，没有唯一的国际标准，但是它提供了一种评估测试活动质量的维度矩阵，一种思想启发。目前国际上有一些测试顾问公司在使用TPI来进行测试咨询服务，识别出测试团队的短木板，给出改进建议。

以上内容在测试架构中的作用和价值是提升测试活动本身的质量。测试活动质量提升后的效果就是产品质量的提升。因此，测试活动的质量如何知道，测试团队提升的方向在哪，都可以参考这些国际通用的方法来帮助我们。

二、常见可用的测试工程方法

需求测试方法——（在产品需求阶段进行测试的方法）

首先，判断需求是否具有可测试性，这是需求测试的最低要求；

其次，对需求描述的完整性，二义性，前后一致性，可修改性，可行性进行评估；

最后，判断需求的必要性；

以上测试方法对人要求较低的是需求的可测试性判断，但也需要有经验的测试人员来实施。而其他内容，则视测试人员的项目经验多少和能力大小来实施。也许需求的测试方法还有许多，但我目前仅用以上方法进行实施，就足够我施展和发挥，而且也还处在不断改进，学习当中。

架构测试方法——（在产品设计阶段进行测试的方法）

借用SEI的经典方法：“质量属性决定架构设计”。测试人员在测试产品架构时，如能从质量属性入手，反推产品架构设计如何来支撑产品的质量属性，则可以发现产品设计上对质量属性支撑不足的缺陷。

同时，可以发挥测试人员积累的异常场景经验和负面场景经验，对架构中的每一个状态或算法进行异常假设和负面假设，则会发现开发人员的设计方案中有不少遗漏。

如能把以上两种方法很好的运用，则测试人员对产品架构质量的贡献已非常巨大了。

基于风险的测试——（在制定测试计划和测试策略之前的测试方法）

核心是测试人员与公司的项目经理，开发人员一起对项目在提交测试前进行风险分析，从风险发生的概率，风险产生的影响两个维度对风险进行打分评级。对于风险系数高的部分作为后期的测试重点，并配置针对性的测试资源和测试方法。基于风险的测试将帮助我们的测试策略能真正关注到产品质量的核心需求，可以大大提升整个测试计划，测试资源的准确性和有效性。

基于模型的测试——（提升测试设计和执行效率的测试方法）

该测试方法解决的问题是：如果需求发生改变，那么测试用例更新，自动化测试脚本更新将会是很大的工作量，而使用基于模型的测试，则几乎可以在瞬间完成用例和测试脚本的更新，用例维护将不再是恶梦。只是基于模型的测试方法对需求描述文字编写的质量要求非常高，且必须符合一定的描述规则，才能自动的将需求转化为模型，模型自动转化为测试用例。

这个大大提升测试设计和执行效率的测试方法目前仅在国外少数机构运用，我也只是看到过演示。如果你的组织能实施基于模型的测试，那么你的组织在需求把握的规范性和质量上已上升到了非常系统的阶段。

探索性测试——（测试执行阶段提升测试质量的测试方法）

探索性测试是测试艺术的体现，也是测试技术中最具有不确定性，也最有意

思的活动，是最有创造性的测试活动。最原始的探索性测试就是简单的“Monkey Test”，但无论多高级的探索性测试都是从“Monkey Test”进化而来。一个测试组织只有“Monkey Test”那是很危险的，因为你的测试质量完全无依无靠，但是一个测试组织没有“Monkey Test”那么也是危险的，因为很多表层以下的问题都没有得到发现。所以，探索性测试应该作为测试用例执行完成后，提升质量的一个必备补充测试技术。它不但能确保系统不少表层下的问题被发现，而且还让测试人员的“发散思维”和“创造力”得到修炼。测试人员要相比开发人员的优势就应该是在“发散思维”和“创造力”上。

基于经验的测试设计——（提高测试经验重用性的测试方法）

业内有一种说法，基于经验的测试设计就是“Checklist”。是的，各种测试所用的checklist是基于经验测试设计的主要体现形式。对组织来说，让有对应测试经验的同事把需要测试的场景经验固化到“Checklist”中，让没有这些“测试场景和方法”的同事也能马上应用上这些测试经验。既为公司保留了宝贵的知识财富，又能让测试覆盖更全面，更系统，提升整体测试质量。

Fuzzing测试——（提升异常测试覆盖率质量的测试方法）

Fuzzing测试有些类似“Monkey Test”，但它与“Monkey Test”的最大区别在于Fuzzing测试的系统性和完整性。Fuzzing测试通常依赖测试工具对测试对象的所有正常场景和异常场景进行覆盖，由于依赖工具进行异常场景覆盖，就可以做到异常组合的全遍历，提升了产品异常测试覆盖率的质量。

等价类划分——（减少测试重复率，提升测试设计和执行效率的测试方法）

对于被测对象的所有场景和输入中，总有一些数据能用一个典型数据来代表，而不需要我们进行所有数据的全遍历。等价类划分技术不只是用于有数字的边界值范围，更可以用于测试用例的划分，测试场景的选取， 测试步骤的等价选择。

正交分析——（提升测试设计覆盖率的测试方法）

简单理解就是要广泛应用功能交互，多个象限边界值交织的测试设计。因为往往某个模块在单一状态下没有问题，但是在多个状态同一时刻作用于一个点时则会暴露出模块设计考虑不足的问题。通常等价类划分和正交分析共同组合使用，使用正交分析可以保障产品应用场景的测试覆盖率，等价类划分则可以对正交分析的结果进行优化，减少重复的测试场景，提高测试效率。

基于场景的分析测试——（确保测试用例不遗漏，测试用例有效性的测试方法）

如何构建测试用例或测试环境？“凭空想象？” 肯定测试结果是不系统，不真实，甚至有时测试还是多余的。最好的方法是：利用需求进行用户场景的分

析，分析用户在一定范围内可能的正常操作和期望，异常操作和期望。然后基于场景分析输出的测试场景，导出测试方案或测试用例。这样编写的测试用例才是真实的，不是多余的或无意义的。既可以保证每个测试用例真正代表了用户需求，证明了每个测试用例的有效性，又可以减少无意义测试用例的产生，或用户真实场景下测试需求的遗漏。

未完待续，下期继续测试架构支撑商业成功（第二部分）……

三、测试的输出

测试计划

测试策略

测试方案

测试用例

测试报告

产品系统知识

四、测试工具

代码覆盖率

内存泄露检测

性能分析

测试工具

Fuzzing工具

模拟器

自动化测试

五、专项测试技术

可靠性测试

可测试性设计

兼容性测试

安全性测试

性能压力测试

IE 对象模型结合 HTML DOM 的 WEB 自动化测试

作者：zzxxbb112

摘要：在这个世界上每天都有数以万计的人在使用 IE，它是全世界使用最广泛的网页浏览器，市场占有率在全球也有着相当大的比例，许多 WEB 应用程序都是基于 IE 来开发，包括现在许多的一些安全控件，密码控件大多都是只支持 IE 浏览器，以及我们使用 QTP 来对 WEB 测试对象进行操作时也基本上使用的是 IE，虽然现在最新的版本开始慢慢支持其他浏览器，但我们的被测系统大多还是基于 IE 来测试的，因此熟悉 IE 自动化模型成了学习 WEB 自动化测试的一项重要任务。

关键字：Internet Explorer，自动化测试，HTML DOM，web

介绍：本文主要介绍如何通过 IE COM 接口结合 HTML DOM 对 IE 浏览器以及 WEB 测试对象的操作，我们都知道 QTP 自动化模型对象（AOM），同样 IE 也支持 COM 接口的自动化模型对象，通过 IE COM 组件我们可以在脱离 QTP 的情况下对 IE 浏览器进行一系列自动化的操作，例如启动浏览器，浏览器地址的跳转，多 IE 窗口操作等等。本章主要介绍 IE COM api 的工作方式以及使用 HTML DOM 来对测试页面上的对象进行自动化。

启动 IE

1. 在 QTP 中启动 IE

```
'在 QTP 中启动 IE
systemutil.run "iexplore.exe"
```

2. 使用 WSH 启动 IE

```
'使用 WSH 启动 IE
Set oShell = CreateObject("wscript.shell")
```

3. 使用 IE COM 对象

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True    '设置可见
oIE.Navigate "http://www.baidu.com"    '跳转 URL
```

小贴士：使用此方法还可以获取窗口的句柄并通过 QTP 来定位浏览器。

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True   '设置可见
oIE.Navigate "http://www.baidu.com"   '跳转 URL
'获取窗口句柄
ieHwnd=oIE.HWND
'通过句柄定位浏览器并关闭
Browser("hwnd:= " & ieHwnd).Close
```

等待页面加载

我们可以使用 IE COM 的 BUSY 属性来确认浏览器是否处于加载状态。并为后续的操作做铺垫，若是没有此步骤，后续的一系列操作可能会无效。因为当 IE 浏览器还有加载完成时，我们对 WEB 测试对象进行操作可能会出现无效的情况，因此为了测试的顺利进行需要在脚本中加入页面等待来保证后续的操作步骤顺利进行。

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True   '设置可见
oIE.Navigate "http://www.baidu.com"   '跳转 URL
'等待 IE 页面加载完毕
While oIE.Busy: Wend
```

接下来我们来对比一下页面加载和没有加载的区别：

A．添加页面加载等待脚本，最后一步 oIE.Document.f.wd.value = "zzxxbb112"先不用管，会在后面的内容中讲解。

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True   '设置可见
oIE.Navigate "http://www.baidu.com"   '跳转 URL
'等待 IE 页面加载完毕
While oIE.Busy: Wend
'百度搜索框输入
oIE.Document.f.wd.value = "zzxxbb112"
```

结果：

B. 不添加页面加载等待的脚本，去掉 While oIE.Busy: Wend 这步

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True   '设置可见
oIE.Navigate "http://www.baidu.com"   '跳转 URL
'等待 IE 页面加载完毕
'While oIE.Busy: Wend
'百度搜索框输入
oIE.Document.f.wd.value = "zzxxbb112"
```

结果：执行出错

原因分析：由于 IE 浏览器跳转 url 后需要一定的加载时间，如果在还没有加载页面时，就直接使用 document 对象是不可以的，必须等待加载完毕，才能够

对 document 对象进行操作，因此程序会抛出一个缺少对象的异常。

遍历所有 IE 窗口

我们可以使用 WINDOWS 的 SHELL.Application 遍历所有打开的 IE 窗口，并在所有打开的窗口中判断其窗口是否为 IE 窗口后存入字典对象集合中。这里就引出一个遍历所有 IE 窗口并且返回一个字典对象集合的 FUNCTION 函数。

```
'遍历所有 IE 对象
Function EnumerateIE()
   On Error Resume Next
   '创建一个字典对象并返回所有当前打开的 IE 的集合
   Set EnumerateIE = CreateObject("Scripting.Dictionary")
   '获取所有 WINDOWS 窗口
   Set oWinShell = CreateObject("Shell.Application")
   Set allWindows = oWinShell.Windows
   '遍历每个 WINDOWS 窗口
   For Each oWindow In allWindows
      '检查如果是 IE 就添加字典对象中
      If InStr(1, oWindow.FullName, "iexplore.exe",vbTextCompare) Then
         EnumerateIE.Add oWindow.hwnd, oWindow
      End if
   Next
End Function
```

接着我们就可以对此函数进行应用了，例如关闭所有 IE 窗口。

```
'获取所有 IE 窗口对象的列表
Set allIE = EnumerateIE()
'关闭所有打开的 IE 窗口
For Each oIE In allIE.Items
   oIE.quit
Next
```

结果：通过调用刚才写的函数，成功关闭所有打开的 IE 窗口，同样的效果如果是 QTP 中只需要一行代码就可以完成。

```
Systemutil.CloseProcessByName ("iexplore.exe")
```

利用 DOM 操作测试对象

现在我们已经会使用 IE COM 组件来对 IE 浏览器进行自动化的操作，但是对于浏览器页面中的测试对象 IE COM 是无法对其进行操作的，这个时候我们就需要使用 HTML DOM 来对其进行操作，HTML DOM 是 HTML Document Object Model(文档对象模型)的缩写，它将网页中的各个元素都看作一个个对象，从而使网页中的元素也可以被计算机语言获取或者编辑。接下来我们就来看几个简单的例子：

三种方法对比：

getElementByID，getElementsByName，getElementsByTagName

1. 通过 getElementByID 方法获取定位对象，并对其进行操作：

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True   '设置可见
oIE.Navigate "http://www.baidu.com"   '跳转 URL
'等待 IE 页面加载完毕
While oIE.Busy: Wend
'获取 DOCUMENT 对象
Set oDoc = oIE.Document
'使用 DOM 对测试对象进行操作
With oDoc
    '搜索框输入
    .getElementByID("kw").value = "zzxxbb112"
    '点击百度搜索
    .getElementByID("sb").Click
End With
Set oDoc = Nothing
Set oIE = Nothing
```

结果：运行后首先是打开浏览器跳转百度首页，之后等待加载完成之后，创建 DOM 对象，并使用 getElementByID 获取百度搜索框对象并对其进行输入“zzxxbb112”后，再次通过 getElementByID 方法获取百度搜索按钮并进行点击，最终成功跳转搜索结果。

2. 通过 getElementsByName 方法获取定位对象，并对其进行操作：

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True    '设置可见
oIE.Navigate "http://www.baidu.com"    '跳转 URL
'等待 IE 页面加载完毕
While oIE.Busy: Wend
'获取 DOCUMENT 对象
Set oDoc = oIE.Document
'获取元素名为 WD 的集合
Set oEdits = oDoc.getElementsByName("wd")
'遍历对象并对其进行操作
For Each oEdit In oEdits
    oEdit.value = "zzxxbb112"
Next
'点击百度搜索
oDoc.getElementByID("sb").Click
Set oDoc = Nothing
Set oIE = Nothing
```

结果：使用 getElementsByName 和 getElementByID 不同，getElementByID 是返回的单个对象，而 getElementsByName 返回的是一个元素的集合，需要通过遍历对象才能对其进行操作。

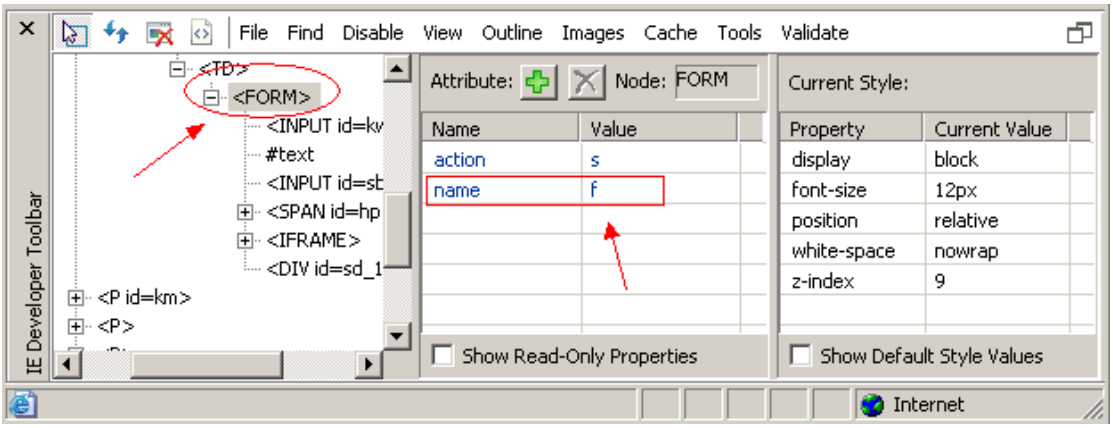
3. 通过 getElementsByTagName 方法获取定位对象，并对其进行操作：

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True   '设置可见
oIE.Navigate "http://www.baidu.com"   '跳转 URL
'等待 IE 页面加载完毕
While oIE.Busy: Wend
'获取 DOCUMENT 对象
Set oDoc = oIE.Document
'获取 TAG 名为 INPUT 的元素集合
Set oEdits = oDoc.getElementsByTagName("INPUT")
'遍历对象并判断文本框对其进行操作
For Each oEdit In oEdits
    If oEdit.type="text" Then
    oEdit.value = "zzxxbb112"
    End If
Next
'点击百度搜索
oDoc.getElementByID("sb").Click
Set oDoc = Nothing
Set oIE = Nothing
```

结果：使用 getElementsByTagName 获取 TAG 名，通过得到相同类型的元素并在遍历中进行判断控件类型并进行操作。

利用 FORM 名来获取对象元素

如果使用 FORM 名来获取对象元素会大大简化我们的脚本，首先我们用 IE DEV 查看下百度的搜索框对应的 FORM 名，在 IE DEV 中查看得到 FORM 名为 f。



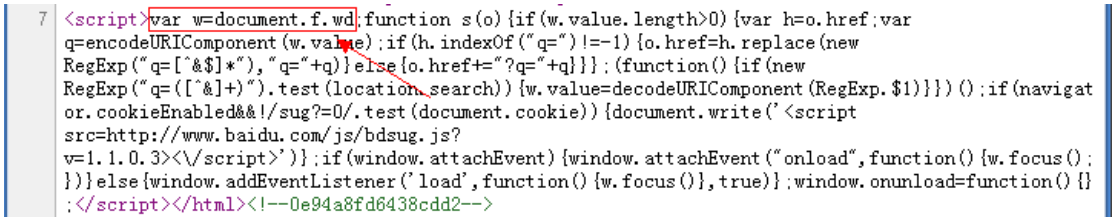

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True    '设置可见
oIE.Navigate "http://www.baidu.com"    '跳转 URL
'等待 IE 页面加载完毕
While oIE.Busy: Wend
'获取 DOCUMENT 对象
Set oDoc = oIE.Document
'获取 FORM 名为 F 下名为 WD 的元素并输入
oDoc.f.wd.value = "zzxxbb112"
'获取 FORM 名为 F 下名为 SB 的元素并点击
oDoc.f.sb.click
Set oDoc = Nothing
Set oIE = Nothing
```

结果：通过以上脚本都可以看到代码非常简易，同样可以达到相同的效果。

访问 WEB 页面的 SCRIPT 脚本变量

通过 DOM 我们可以直接访问 WEB 页面中的 javascript 或者 vbscript 中的变量。

首先打开百度的源文件



我们可以看到在百度源文件中的 javascript 脚本中定义了一个变量为 W 并且赋值为 document.f.wd,从上面的例子我们可以很明显的知道此处的 W 代表的就是百度搜索框这个对象，那么我们其实就可以直接操控 W 这个对象来对自动化百度文本搜索框。

```
'使用 IE COM 启动 IE
Set oIE = CreateObject("InternetExplorer.Application")
oIE.Visible = True    '设置可见
oIE.Navigate "http://www.baidu.com"    '跳转 URL
'等待 IE 页面加载完毕
While oIE.Busy: Wend
'获取 DOCUMENT 对象
Set oDoc = oIE.Document
'获取百度搜索框对象
Set oEdit=oDoc.parentWindow.w
'并对其进行输入
oEdit.value = "zzxxbb112"
'获取 FORM 名为 F 下名为 SB 的元素并点击
oDoc.f.sb.click
Set oDoc = Nothing
Set oIE = Nothing
```

从以上代码我们可以看到只需要通过 parentWindow 来访问 WEB 页面 SCRIPT 中的变量。

Browser 对象转化 Window 窗口对象

我们都知道浏览器窗口之所以能够被 QTP 识别为 Browser 对象是因为 QTP 加载了 WEB-ADDIN 这个插件，若是在启动 QTP 时，没有加载此插件 QTP 就无法把浏览器识别为 Browser 对象，而是把其识别为 Window 对象。每个 Windows 窗口对象都有其唯一的句柄，此句柄就是用于唯一识别的标识，这样我们就可以使用此标识来把 Browser 对象转化为 Window 窗口对象。

```
'浏览器最大化
Public Function BrowserMaximize(oBrw)
    Dim hwnd
    hwnd = oBrw.GetROProperty("hwnd")
    Window("hwnd:=" & hwnd).Maximize
End Function
'浏览器最小化
Public Function BrowserMinimize(oBrw)
    Dim hwnd
    hwnd = oBrw.GetROProperty("hwnd")
    Window("hwnd:=" & hwnd).Minimize
End Function
'浏览器双击
Public Function BrowserDblClick(oBrw, X, Y, MouseButton)
    Dim hwnd
    hwnd = oBrw.GetROProperty("hwnd")
    Window("hwnd:=" & hwnd).DblClick X, Y, MouseButton
End Function

'浏览器大小重设
Public Function BrowserResize(oBrw, width, heigh)
    Dim hwnd
    hwnd = oBrw.GetROProperty("hwnd")
    Window("hwnd:=" & hwnd).Resize width, height
End Function
'浏览器恢复
Public Function BrowserRestore(oBrw)
    Dim hwnd
```

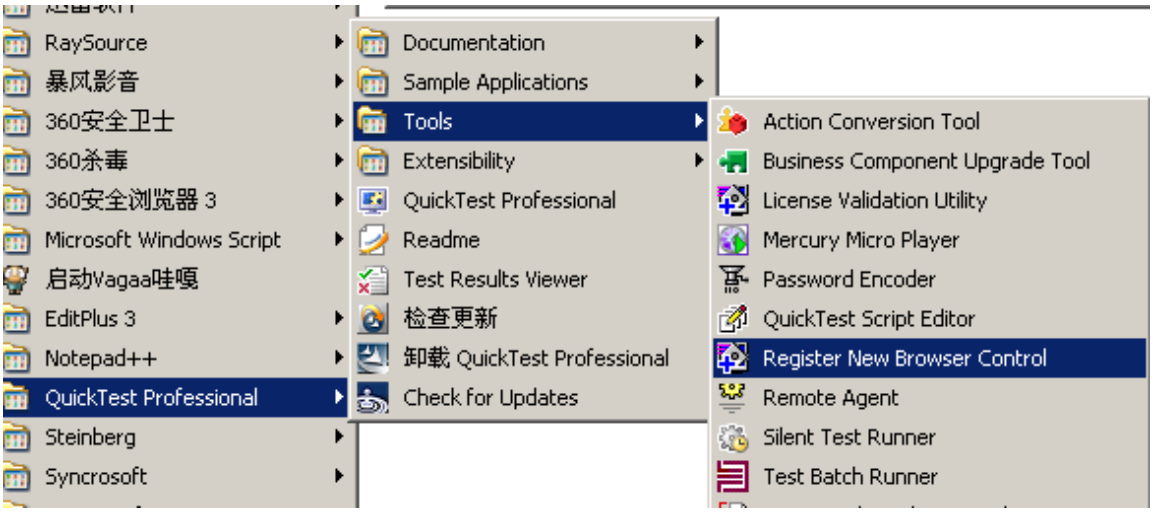
```
    hwnd = oBrw.GetROProperty("hwnd")
    Window("hwnd:=" & hwnd).MouseMove X, Y
End Function
'浏览器移动
Public Function BrowserMove(oBrw, X, Y)
    Dim hwnd
    hwnd = oBrw.GetROProperty("hwnd")
    Window("hwnd:=" & hwnd).Move X, Y
End Function
'获取浏览器句柄
Public Function BrowserhWnd(oBrw)
    hwnd = oBrw.GetROProperty("hwnd")
End Function

'注册自定义函数
RegisterUserFunc "Browser", "Maximize", "BrowserMaximize"
RegisterUserFunc "Browser", "Minimize", "BrowserMinimize"
RegisterUserFunc "Browser", "DblClick", "BrowserDblClick"
RegisterUserFunc "Browser", "Resize" , "BrowserResize"
RegisterUserFunc "Browser", "Restore", "BrowserRestore"
RegisterUserFunc "Browser", "MouseMove", "BrowserMouseMove"
RegisterUserFunc "Browser", "Move" ,"BrowserMove"
RegisterUserFunc "Browser", "hWnd" ,"BrowserhWnd"
```

由于 QTP 的 Browser 对象不支持以上这些函数方法，因此我们通过自己写方法然后通过 RegisterUserFunc 函数来注册到 Browser 对象，使其也具有这些方法。

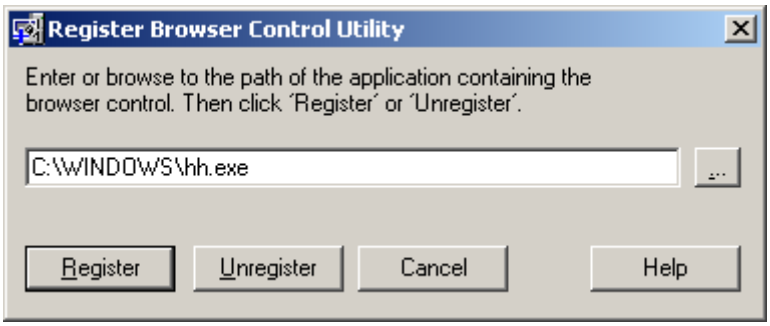
自定义浏览器应用程序

现在有很多 WINDOWS 应用程序都会在其中嵌入 WebBrowser 控件，此控件具有 HTML 相同的显示效果，虽然其核心还是 IE 内核，但是这样的嵌入方式却给自动化测试带来了很大的麻烦，QTP 在此方面也已经考虑到了此处的问题，因此也提供出了一个工具 Register Browser Control Utility 来解决这方面的问题。原先 QTP 是无法识别应用程序中的浏览器对象，哪怕是加载了 WEB 插件，但是使用了 Register Browser Control Utility 注册了对应的应用程序之后 QTP 就可

以成功识别此应用程序中的浏览器对象。下面我们来看一下具体是怎么操作的。



首先我们打开 Register Browser Control Utility 之后：



选择 HH.EXE 点击 Register 之后，重启 QTP，之后随便打开一个 CHM 文件，QTP 就可以成功把 CHM 右边的白色区域识别为 Browser 包括 Browser 对象下的所有对象。

总结

这一章我们主要介绍了利用 IE 的 COM 以及 HTML DOM 来自动化 IE 浏览器以及对浏览器的一些控件对象进行自动化的操作，包括启动 IE、等待页面加载、遍历所有 IE 窗口、利用 DOM 操作测试对象、利用 FORM 名来获取对象元素、访问 WEB 页面的 SCRIPT 脚本变量、Browser 对象转化 Window 窗口对象、自定义浏览器应用程序,这些方法对于我们在自动化测试中也是起到比较重要的作用，并且能够辅助我们更好的完成 WEB 自动化测试，当 QTP 不能达到我们想要达到的目的时，我们就可以使用这些方法来代替或者说来实现我们需要实现的方法，最终使 WEB 自动化测试变得更加的轻松和容易。

参考文献：

《w3school》http://www.w3school.com.cn/htmldom/

利用交叉测试提升软件测试效率

作者：丁志义

摘要：软件测试员长期重复测试同一功能或项目，会出现审丑疲劳，工作效率下降。而交叉测试是提高测试效率，更早更多发现软件缺陷的手段之一。本文就交叉测试意义、如何设计交叉测试、交叉测试使用时点以及交叉测试效果等展开论述，数据表明在正确阶段引入交叉测试可提升软件测试效率，提高软件发版质量。

关键词：交叉测试　软件测试 效率

一、交叉测试意义

一个人长时间干同一件事情，会出现疲劳和厌烦，会丧失激情，甚至不思进取。而软件测试工作一个突出特点就是不断重复枯燥、冗长的测试，所以测试人员在长时间进行同一个模块或同一个功能测试是也及易产生疲劳，发现缺陷迟钝。由于疲劳导致的精力不集中，很容易略过一些重要的测试环节，甚至面对显然的错误也“视而不见”，这是“审丑疲劳”。

同时，在软件开发测试过程中，尤其是对同一产品，同一组开发人员和同一组测试人员长时间配合，测试人员会在开发人员多次说服和解释下而被同化。测试人员对缺陷的判断力会逐渐降低，很多本来是缺陷的问题，由于测试人员对软件“司空见惯”的使用，会不被当成缺陷。

还有研究表明，人有定位效应。社会心理学家曾作过一个试验：在召集会议时先让人们自由选择位子，之后到室外休息片刻再进入室内入座，如此五至六次，发现大多数人都选择他们第一次坐过的位子。这种现象称为定位效应，说明人们习惯上凡是自己认定的，人们大都不想轻易改变它。测试人员的定位效应就是测试过的功能不再进行认真测试：在回归测试时，之前由于进行过认真的测试，往往会认为某些功能是可靠，只要验证一些以前发现的缺陷是否修改完成或者进行主要功能流程验证就可以了。这种现象在多次回归时表现的更加突出。而开发人员在修改缺陷时往往又会引入新的缺陷，测试人员的疏于防范不能更多更早地暴露问题，增加产品质量风险。

测试的主要职责是要在软件发版前尽可能多的发现缺陷，和验证缺陷被修复，保证产品质量。市场要求软件开发节奏越来越快，产品研发周期越来越短，留给测试的时间往往是最容易被压缩挤占的，测试人员数量又不会增加，但发版产品质量必须得到保证。因此只能通过提升测试效率，增加测试产出率，更多更早地暴露和修正软件缺陷来保证产品质量。

在一个测试团队中每个测试员测试项目内容或模块经常是有分工的，且在较

长的时间内是固定的。针对“审丑疲劳”、同化现象以及定位效应等工效下降问题，必须采取措施，提高测试积极性和工作效率。除了将测试用例多次完整执行、必要的士气鼓舞等激励措施外，交叉测试便是一种有效的办法。

交叉测试可以很大程度的避免定位效应。测试人员在互相交换任务时，反复测试某一功能的机会大大减少，从而也就不会“主观的”认为某些功能没有缺陷。因为测试工作内容的变化，对新测试人员来说更有新鲜感、挑战性和刺激性，测试人员对未知领域和未驾御的程序将更乐于测试。测试内容的变化对应配合的开发人员一般也会对应改变，这样“审丑疲劳”和“同化现象”都将大大降低。不同测试人员的不同视角和方法能将隐藏的缺陷快速找出，容易发现更多问题。同时通过交叉测试，测试人员加深对产品的功能的理解和掌握，实现在工作中得到成长和满足。实践表明一个“新手”发现问题会更多更快。

交叉测试也可作为检验前期测试工作成绩的有效手段。

二、交叉测试时机

不是任何时候都可进行交叉测试。当程序的基本功能还没有实现，用户界面还有很多控制问题没有解决，每个测试员每日发现缺陷数量还很多时，不宜进行交叉测试。这个时候各测试员负责测试的模块或功能点功能还不完善，测试员也还没有出现明显的审丑疲劳，开发人员也在忙于修改问题，进行交叉测试没有太大意义。只有当每日发现问题数很少或很均匀时，每个测试员发现的缺陷的趋势稳定甚至下降时，即缺陷数量寥寥不足以开发修改时，可以引入交叉测试。这个时候产品比较稳定，但测试人员经过长时间重复测试，已经出现疲劳懈怠。交叉测试时点要根据当时测试情况和产品稳定状况灵活把握，适时调整，这个时间点一般在联调测试后期，当然也可在进入联调测试前，作为测试工作质量评估而进行交叉抽验。下图给出交叉测试开始时间点示意图：

三、交叉测试原则

除了进行交叉测试的时间点要把握好外，交叉测试还要遵守必要的原则。例如不同的测试员对产品功能的熟悉和掌握程度应该相似，或对将要测试的内容有一定的工作基础。因为测试时间的限制，不允许在正常测试阶段再长时间的学习产品功能，因此对下一轮进行测试内容的轮换，应根据各测试员能力状态合理分配对应的交叉功能程序。只有他对要测试的功能要求和操作步骤清楚才能正常展开测试，少走弯路减少时间的浪费。对产品主要功能和流程性测试，轮换的测试内容和功能点应尽量有衔接性，如有上下由业务关系，这样的交叉几乎是无缝的，同时会增加测试员对产品功能系统性掌握。当然如果测试的内容不是按流程来分配，而是独立的测试项目分散的功能点类例外，可以在上轮中测试某个功能点 A，而下轮测试功能点 B。交叉测试内容应该在设计的周期内完成，然后进入下一个周期内轮动测试。当时间充裕应该进行多次交叉测试，至少保证主要功能流程和每个测试项目被 3 位不同测试员覆盖测试过。以上原则应该在交叉测试设计中得到体现。

四、交叉测试设计

凡事都要有计划，良好的计划是成功的一半。测试也不例外，在软件测试之前要准备和编写完整的测试用例和测试方案，交叉测试方案也应同期给出。在交叉测试方案中应细化交叉功能点或程序或独立的项目。第一轮测试周期起止时间，对应的测试员是谁，该测试员应该对该程序功能最熟悉，大部分问题应该由该测试员发现，第一轮测试完成后该程序功能或项目应该比较稳定，缺陷数量应降低到个数级。也可将从程序的单元验证到进入下一轮抽验这个阶段归为第一轮，因该周期较长，没有明确的开始时间，因此可将下一轮开始时间设置为第一论结束时间，而第一轮开始时间可以为空。第二轮周期中各测试员测试内容遵循上面原则，并和上个周期内容完全不同，时间一般比较短，可以一周作为一个交叉测试周期。该周期内的测试可作为上个周期测试工作质量和工作成绩评估的一种手段。依次进入下个交叉周期。以下为交叉测试设计图例。

某某产品交叉测试方案：

交叉测试轮次 测试员 测试内容	第一轮		第二轮		第三轮		第 N 轮	
	起始日期	结束日期	起始日期	结束日期	起始日期	结束日期	起始日期	结束日期
	(9.10)	11.12	11.13	11.19	11.20	11.26		
流程测试								
BOM 模块主流程	测试员 1		测试员 3		测试员 2			
计划模块主流程	测试员 2		测试员 1		测试员 3			
生产模块主流程	测试员 3		测试员 2		测试员 1			
……								
专项测试								
实施工具测试	测试员 1		测试员 3		测试员 2			
升级工具测试	测试员 2		测试员 1		测试员 3			
安装卸载测试	测试员 3		测试员 2		测试员 1			
……								

五、交叉测试执行

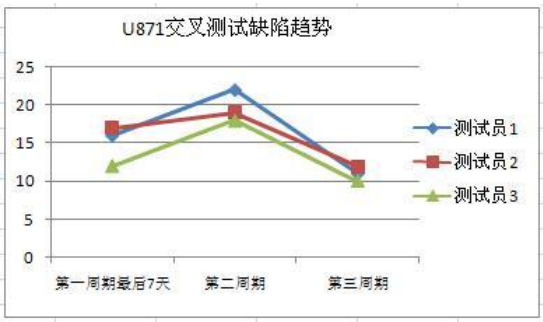
作为软件测试质量保证的重要手段，交叉测试应按测试设计的方案有效执行。实际执行中要根据软件开发和测试进度以及功能稳定性等适当调整交叉测试时间点，测试周期和轮次数也应根据实际情况进行必要修正。无论时间多么紧迫，交叉测试是必要，不经过客户验证的软件是不成熟的，不经过不同测试人员进行测试的软件质量也很难得到保证。不能因为时间紧迫而放弃交叉测试，哪怕是只进行了 1 次交叉测试或者抽验，也会大为减少带到客户那的缺陷数目，降低产品质量风险。

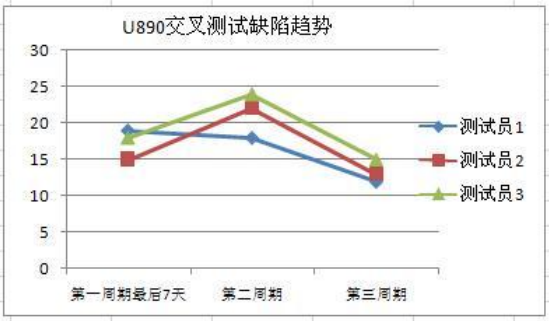
六、交叉测试效果分析

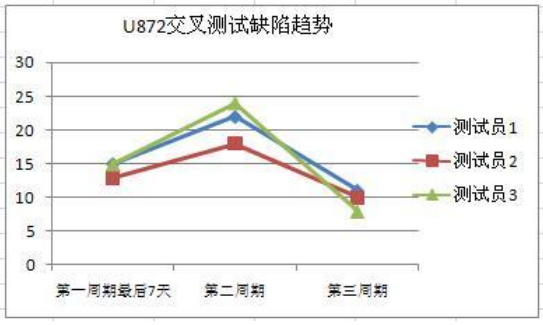
每个测试周期中，测试负责人可以根据缺陷数量和重要等级程度分析前期测试质量，可以在下个周期中进行针对性的调整，例如对质量不稳定的薄弱功能点加大测试资源投入，调整测试人员等。笔者从 2007 年开始对所在团队开发的产品正式引入交叉测试，从这 3 年的测试数据分析，在第二个周期内各测试员发现缺陷的数量明显高于前其同时间段内问题数，且好多问题都是反复问题，显而意见的问题，证明在第一周期的最后几天内原测试人员因为这样或那样的原因确实有些“审丑疲劳”。而在接下来的第三轮动周期中，问题数量有所下降，逐步趋向个级，以新发现的缺陷、更深层次和复杂应用场景缺陷居多。经过这样的轮动测试，软件中的缺陷会越来越少，最终达到产品进入集成测试要求。

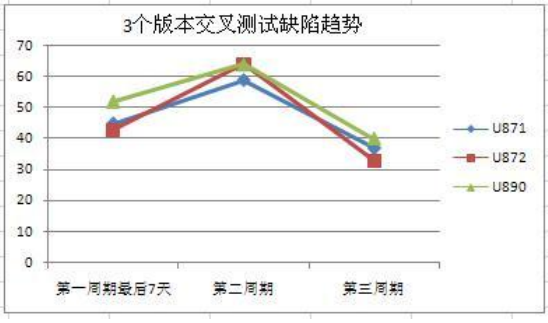
以下为 U871、U872、U890 这 3 个版本软件测试中交叉测试问题统计和图表分析：

U871 交叉测试缺陷数据			
测试员	第一周期最后 7 天	第二周期	第三周期
测试员 1	16	22	11
测试员 2	17	19	12
测试员 3	12	18	10
U872 交叉测试缺陷数据			
测试员	第一周期最后 7 天	第二周期	第三周期
测试员 1	15	22	11
测试员 2	13	18	10
测试员 3	15	24	8
U890 交叉测试缺陷数据			
测试员	第一周期最后 7 天	第二周期	第三周期
测试员 1	19	18	12
测试员 2	15	22	13
测试员 3	18	24	15
3 个版本交叉测试缺陷数据			
版本	第一周期最后 7 天	第二周期	第三周期
U871	45	59	37
U872	43	64	33
U890	52	64	40









七、扩大交叉测试使用范围

交叉测试能更多更早发现产品缺陷，因此除了在各相对独立的开发组内使用外，还可以延伸到整个软件产品内进行。当前大的软件开发都会分成若干个业务组，例如 ERP 软件开发，一般分为财务组、供应链组、制造组等。各开发组内可以在单元验证后期或联调测试阶段进行交叉测试，在进入集成测试后，由于不同业务组实现的程序功能之间有着上下游关系，有接口和流程要求，为保证数据传递的正确性，功能一致连贯性，也可在不同业务组间进行交叉测试。当然参与交叉测试人员不宜过多，不宜全部组间轮换，参与的测试员的选取也应当以了解甚至熟悉对方业务流程为主。通过这样的衔接交叉测试，产品流程接口才更稳定更可靠。

C 单元测试框架

作者：Anil Kumar、Jerry St.Clair 译者：黄志霞

1. 引言

1.1 概述

CUnit 是在 C 语言环境中用来编写、管理和运行单元测试的系统，它是用户通过编写测试代码进行连接的静态库。

CUnit 用一个简单的框架来建立测试结构，为常用数据类型的测试提供了丰富的断言语句，并为运行测试和报告测试结果提供各种接口，包括通过代码自动运行测试并报告测试结果的接口，还有使用户可以动态运行测试并查看结果的交互式接口。

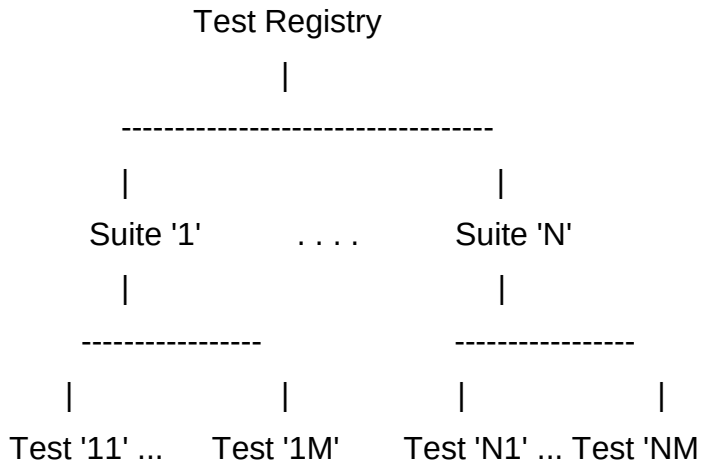
在下面的头文件中说明对用户非常有用的数据类型和函数：

头文件	含义
#include <CUnit/CUnit.h>	测试用例中的断言宏指令和其他的框架头文件
#include <CUnit/CUError.h>	错误处理的函数和数据类型由 CUnit.h 自动加载
#include <CUnit/TestDB.h>	数据类型定义和函数操作 由 CUnit.h 自动加载
#include <CUnit/TestRun.h>	运行测试并报告测试结果的数据类型和函数的定义
#include <CUnit/Automated.h>	自动实现 xml 输出
#include <CUnit/Basic.h>	非交互式 输出到 stdout
#include <CUnit/Console.h>	交互式 控制台方式
#include <CUnit/CUCurses.h>	交互的 控制台方式(*nix)
#include <CUnit/Win.h>	Windows 界面（尚未实现）

1.2 结构

CUnit 由一个与平台无关的框架和各种用户接口组成。核心框架实现对 Registry、测试包和测试用例的管理。通过用户接口方便地使用这个框架来运行测试并查看结果。

CUnit 和通常的单元测试框架是一样的：



一个活动的测试 Registry（相当于一个测试单元）可以注册多个包（Suite，可视为程序中独立的模块），每个包（Suite）又挂载多个测试用例（Test case）。包可以有 setup 和 teardown 函数，在其执行前后自动进行调用。通过一个函数调用可以运行 Registry 中所有的包或用例，也可以选择一些包或用例来运行。

1.3 测试基本流程

使用 CUnit 的基本步骤如下；

写测试函数(必要的时候包括测试组的初始化和清空)

初始化测试- CU_initialize_registry()

向测试中添加包 - CU_add_suite()

向包中添加测试用例 - CU_add_test()

用合适的接口运行测试，如控制台 e.g. CU_console_run_tests

清理测试 - CU_cleanup_registry

1.4 CUnit API 第 2 版的变化

在所有的公有名称前加上前缀“CU_”，以减少用户代码中命名的冲突。以前的版本用的是不带前缀的名称，和现在不同。原来的名称不再使用，原有代码的编译要引用 USE_DEPRECATED_CUNIT_NAMES 文件。

原有的 API 函数将在文件相关部分进行详细说明。

2. 写 CUnit 测试用例

2.1 测试函数

CUnit 测试函数是 C 语言函数。

void test_func(void)

测试函数的内容没有任何的限制，但不能修改 CUnit 的框架（如加入 Suite 或用例，修改 Registry，或初始化测试运行）。一个测试函数还可以调用其他的测试函数（当然也不能影响到测试框架）。当注册一个 Test 的时候，如果 Registry

在运行，则被测函数就会被调用。

例如：返回两个整型数的最大值的测试函数为

```
int maxi(int i1, int i2)
{
   return (i1 > i2) ? i1 : i2;
}
void test_maxi(void)
{
   CU_ASSERT(maxi(0,2) == 2);
   CU_ASSERT(maxi(0,-2) == 0);
   CU_ASSERT(maxi(2,2) == 2);
}
```

2.2 CUnit 断言

CUnit 提供了一套测试逻辑判断的断言。通过 CUnit 进行判断断言是否成立，在测试运行结束的时候得到测试结果。

每个断言测试一个逻辑条件，若值为 False，测试不通过。一个断言执行结束，测试函数继续往下运行，直到出现 FATAL 类的断言，测试函数会终止并立即返回。FATAL 类型的断言要谨慎使用。一旦 FATAL 断言执行失败，测试函数将无法 cleanup。但是，一般的 Suite cleanup function 不受影响。

也有一些特殊的断言不需要逻辑测试而得到 pass 和 fail 的判断，这在测试流程控制和其他不需要逻辑判断的环境中很有帮助：

```
void test_longjmp(void)
    {
       jmp_buf buf;
       int i;

       i = setjmp(buf);
       if (i == 0) {
          run_other_func();
          CU_PASS("run_other_func() succeeded.");
       }
       else
          CU_FAIL("run_other_func() issued longjmp.");
    }
```

由已注册过的测试函数调用的函数在使用断言时没有特别的限制，其使用的断言属于主调函数。并且可以使用 FATAL 断言，在失断言失败时会终止当前测试函数和整个调用链的执行。

CUnit 已经定义的断言包括：

#include <CUnit/CUnit.h>

CU_ASSERT(int expression) CU_ASSERT_FATAL(int expression) CU_TEST(int expression) CU_TEST_FATAL(int expression)	断言表达式值为 True 或非零
CU_ASSERT_TRUE(value) CU_ASSERT_TRUE_FATAL(value)	断言 value 值为 TRUE 或非零
CU_ASSERT_FALSE(value) CU_ASSERT_FALSE_FATAL(value)	断言 value 的值为 FALSE 或零
CU_ASSERT_EQUAL(actual, expected) CU_ASSERT_EQUAL_FATAL(actual, expected)	断言期望值==实际值
CU_ASSERT_NOT_EQUAL(actual, expected)) CU_ASSERT_NOT_EQUAL_FATAL(actual, expected)	断言期望值！=实际值

CU_ASSERT_PTR_EQUAL(actual, expected) CU_ASSERT_PTR_EQUAL_FATAL(actual, expected)	断言指针实际指针=期望指针
CU_ASSERT_PTR_NOT_EQUAL(actual, expected) CU_ASSERT_PTR_NOT_EQUAL_FATAL(actual, expected)	断言指针实际指针！=期望指针
CU_ASSERT_PTR_NULL(value) CU_ASSERT_PTR_NULL_FATAL(value)	断言指针为空
CU_ASSERT_PTR_NOT_NULL(value) CU_ASSERT_PTR_NOT_NULL_FATAL(value)	断言指针不为空
CU_ASSERT_STRING_EQUAL(actual, expected) CU_ASSERT_STRING_EQUAL_FATAL(actual, expected)	断言字符串实际值与期望值相同
CU_ASSERT_STRING_NOT_EQUAL(actual, expected) CU_ASSERT_STRING_NOT_EQUAL_FATAL(actual, expected)	断言字符串实际值与期望值不同
CU_ASSERT_NSTRING_EQUAL(actual, expected, count) CU_ASSERT_NSTRING_EQUAL_FATAL(actual, expected, count)	断言实际字符串和期望字符串的字符数目相等
CU_ASSERT_NSTRING_NOT_EQUAL(actual, expected, count) CU_ASSERT_NSTRING_NOT_EQUAL_FATAL(actual, expected, count)	断言实际字符串和期望字符串的字符数目不相等
CU_ASSERT_DOUBLE_EQUAL(actual, expected, granularity) CU_ASSERT_DOUBLE_EQUAL_FATAL(actual, expected, granularity)	断言 |实际值-期望值| <=区间长度 (这个断言在使用时必须加载 math 库)
CU_ASSERT_DOUBLE_NOT_EQUAL(actual, expected, granularity) CU_ASSERT_DOUBLE_NOT_EQUAL_FATAL(actual, expected, granularity)	断言 |实际值-期望值| >区间长度 (这个断言在使用时必须加载 math 库)

CU_PASS(message)	带有说明信息的执行成功的断言 （不含逻辑判断的测试）
CU_FAIL(message) CU_FAIL_FATAL(message)	带有说明信息的执行失败的断言 （不含逻辑判断的测试）

2.3 V1 被取代的断言

下面的断言是第二版已不用的断言，在使用这些断言时，原有的代码的编译要引用已定义的 USE_DEPRECATED_CUNIT_NAMES 文件，而作用和第一版是一样的。

#include <CUnit/CUnit.h>

原名称	等效的新名称
ASSERT	CU_ASSERT_FATAL
ASSERT_TRUE	CU_ASSERT_TRUE_FATAL
ASSERT_FALSE	CU_ASSERT_FALSE_FATAL
ASSERT_EQUAL	CU_ASSERT_EQUAL_FATAL
ASSERT_NOT_EQUAL	CU_ASSERT_NOT_EQUAL_FATAL
ASSERT_PTR_EQUAL	CU_ASSERT_PTR_EQUAL_FATAL
ASSERT_PTR_NOT_EQUAL	CU_ASSERT_PTR_NOT_EQUAL_FATAL
ASSERT_PTR_NULL	CU_ASSERT_PTR_NULL_FATAL
ASSERT_PTR_NOT_NULL	CU_ASSERT_PTR_NOT_NULL_FATAL
ASSERT_STRING_EQUAL	CU_ASSERT_STRING_EQUAL_FATAL
ASSERT_STRING_NOT_EQUAL	CU_ASSERT_STRING_NOT_EQUAL_FATAL
ASSERT_NSTRING_EQUAL	CU_ASSERT_NSTRING_EQUAL_FATAL
ASSERT_NSTRING_NOT_EQUAL	CU_ASSERT_NSTRING_NOT_EQUAL_FATAL
ASSERT_DOUBLE_EQUAL	CU_ASSERT_DOUBLE_EQUAL_FATAL
ASSERT_DOUBLE_NOT_EQUAL	CU_ASSERT_DOUBLE_NOT_EQUAL_FATAL

3. The Test Registry

3.1 概要

```
#include <CUnit/TestDB.h> (由<CUnit/CUnit.h>自动加载)
  typedef struct CU_TestRegistry
  typedef CU_TestRegistry*  CU_pTestRegistry
  CU_ErrorCode      CU_initialize_registry(void)
  void              CU_cleanup_registry(void)
  CU_pTestRegistry CU_get_registry(void)
  CU_pTestRegistry CU_set_registry(CU_pTestRegistry pTestRegistry)
  CU_pTestRegistry  CU_create_new_registry(void)
  void     CU_destroy_existing_registry(CU_pTestRegistry* ppRegistry)
```

3.2 内部结构

Registry 是包（Suite）和相关测试（Test）的组合。CUnit 保持一个 Registry 处于激活状态，当有测试加入时 Registry 发生改变。激活状态的 Registry 中的 suites 就是当用户运行所有的测试用例时运行的 suites。

CUnit 的 registry 是一个在<CUnit/TestDB.h>中声明的数据结构 CU_TestRegistry。它包括 Registry 中所有的测试包和测试用例，还有一个指向已注册的测试包链表的头指针。

```
typedef struct CU_TestRegistry
  {
    unsigned int uiNumberOfSuites;
    unsigned int uiNumberOfTests;
    CU_pSuite     pSuite;
  } CU_TestRegistry;
typedef CU_TestRegistry* CU_pTestRegistry;
```

通常，用户只需要在测试前后分别进行初始化和 cleanup 操作，但是在必要的时候需提供函数进行 Registry 操作。

3.3 初始化

CU_ErrorCode CU_initialize_registry(void)

CUnit Registry 在使用前必须进行初始化，CU_initialize_registry()的调用必须在所有函数调用之前完成，否则会很可能导致崩溃。

返回错误状态代码：

CUE_SUCCESS	初始化成功
CUE_NOMEMORY	存储分配失败

3.4 清理

void CU_cleanup_registry(void)

当测试完成时，用户调用 void CU_cleanup_registry(void)清理 Registry 并释放占用的存储空间，这是测试调用的最后一个函数(使用 CU_initialize_registry() 或 CU_set_registry()激活 Registry 除外)。

调用 CU_cleanup_registry()失败将导致内存泄露，它能连续被调用多次而不出现错误。请注意：调用这个函数将销毁 Registry 中所有的测试包和相关的测试。在清除 Registry 之后，指向 Suites 和 Tests 的指针应该建立起联系。

调用 CU_cleanup_registry()只影响 CUnit 框架的内部 CU_TestRegistry。销毁用户的任何 Registry 只能由用户完成，可以通过显式调用 CU_destroy_existing_registry()实现，也可以隐式地用 CU_set_registry() 激活新的 Registry，或再次调用 CU_cleanup_registry()来实现。

3.5 其他 Registry 函数

其他的测试函数原本用做内部使用和测试，但通常用户也会用到这些函数，所以应该有所了解。主要有：

CU_pTestRegistry CU_get_registry (void)

返回指向当前活动的 Registry 的指针。该 Registry 是一个 CU_TestRegistry 类型的变量，宜通过 API 函数进行操作而不是直接操作内部 Registry。返回的指针属于框架，在调用 CU_cleanup_registry() 或 CU_initialize_registry()时不能使用。

CU_pTestRegistry CU_set_registry (CU_pTestRegistry TestRegistry)

用一个指定的 Registry 代替当前活动的 Registry，返回指向原来 Registry 的指针。调用者负责销毁旧的 Registry，通过显式调用 CU_destroy_existing_registry() 来返回指针。另一种方式，可以用 CU_set_registry()激活 Registry 或在调用 CU_cleanup_registry()时隐式地销毁 Registry。但显式地销毁当前活动的 Registry 需要很谨慎，因为这很可能导致多次释放同一个 Registry 的内存甚至造成崩溃。

CU_pTestRegistry CU_create_new_registry (void)

创建一个 Registry 并返回指向该 Registry 的指针。新的 Registry 不包含任何的 Suite 和 Test，销毁新创建的 Registry 由调用函数负责。

void CU_destroy_existing_registry(CU_pTestRegistry* ppRegistry)

销毁 Registry 并释放其占用的所有内存，它内部的 Suite 和 Test 也同时被

销毁，不能针对当前活动的 Registry 调用该函数进行销毁（如通过 CU_get_registry() 返回 CU_pTestRegistry 类型的指针），因为当调用 CU_cleanup_registry()时会多次释放同一块内存。当参数为 null 时该函数没有意义。

3.6 V1 被取代的数据类型和函数

下面是在第二版中已经取消的数据类型和函数，若要这些函数或数据类型名称，用户代码的编译要引用已定义的 USE_DEPRECATED_CUNIT_NAMES 文件。

#include <CUnit/TestDB.h> （由< CUnit/CUnit.h>自动加载）。

旧名称	等效的新名称
_TestRegistry	CU_TestRegistry
_TestRegistry.uiNumberOfGroups PTestRegistry->uiNumberOfGroups	CU_TestRegistry.uiNumberOfSuites CU_pTestRegistry->uiNumberOfSuites
_TestRegistry.pGroup PTestRegistry->pGroup	CU_TestRegistry.pSuite CU_pTestRegistry->pSuite
PTestRegistry	CU_pTestRegistry
initialize_registry()	CU_initialize_registry()
cleanup_registry()	CU_cleanup_registry()
get_registry()	CU_get_registry()
set_registry()	CU_set_registry()

4. 管理测试用例和测试包

要通过 CUnit 运行测试，必须将测试添加进已注册到 Registry 的测试包中。

4.1 内容概要

```
#include <CUnit/TestDB.h> (由 <CUnit/CUnit.h>自动加载)
   typedef struct CU_Suite
   typedef CU_Suite*   CU_pSuite
   typedef struct CU_Test
   typedef CU_Test*   CU_pTest
   typedef void (*CU_TestFunc)(void)
   typedef int   (*CU_InitializeFunc)(void)
   typedef int   (*CU_CleanupFunc)(void)

   CU_pSuite CU_add_suite(const char* strName,
                          CU_InitializeFunc pInit,
                          CU_CleanupFunc pClean);

   CU_pTest   CU_add_test(CU_pSuite pSuite,
                          const char* strName,
                          CU_TestFunc pTestFunc);

   typedef struct CU_TestInfo
   typedef struct CU_SuiteInfo
   CU_ErrorCode CU_register_suites(CU_SuiteInfo suite_info[]);
   CU_ErrorCode CU_register_nsuites(int suite_count, ...);
```

4.2 向 Registry 加入 Suite

CU_pSuite CU_add_suite(const char* strName, CU_InitializeFunc pInit, CU_CleanupFunc pClean)

用指定的名称、初始化函数和清除函数创建包 Suite。新的包注册到 Registry 并属于该 Registry，而 Registry 必须在所有测试包加入之前完成初始化。当前不支持创建独立于 Registry 之外的 Suite。

包的名称必须是 Registry 内唯一。而初始化和清除函数并不是必须包含的，它们以指针的形式分别传递给 Suite 中的测试运行前后调用到的函数，这样包就可以 setup 或 teardown 临时变量来运行测试。这些函数不含参数并在执行成功时返回 0(否则返回非 0)。如果包不需要这两个函数，给 CU_add_suite()传递参数 null。

返回指向新的 Suite 的指针，用于向包中加入 Test。如果发生错误，返回 NULL。相应的错误代码如下：

CUE_SUCCESS	包创建成功
CUE_NOREGISTRY	Registry 尚未被初始化
CUE_NO_SUITENAME	包名称为空.
CUE_DUP_SUITE	包名称重名
CUE_NOMEMORY	存储空间分配失败

4.3 向包中添加测试

CU_pTest CU_add_test (CU_pSuite pSuite, const char* strName, CU_TestFunc pTestFunc)

用指定的名称和测试函数创建测试用例，注册到指定的测试包，这个测试包必须是已经用 CU_add_suite()创建的。当前不支持创建独立于 Suite 之外的 test。

Test 的名称必须是包内唯一。参数 pTestFunc 不能为空，并且指向一个在测试运行中调用的函数，测试函数没有参数和返回值。

返回指向测试用例的指针，在创建测试用例的过程中发生错误，返回 NULL。CUnit 定义的错误码如下：

CUE_SUCCESS	包创建成功
CUE_NOSUITE	指定的包为空或者不可用
CUE_NO_TESTNAME	suite 名称为空.
CUE_NO_TEST	指针为空或不可用
CUE_DUP_TEST	测试用例的名称重名
CUE_NOMEMORY	存储空间分配失败

4.4 管理测试的捷径

#define CU_ADD_TEST(suite, test) (CU_add_test(suite, #test, (CU_TestFunc)test))

这个宏命令根据测试函数的名称自动产生唯一的测试用例并把它加入到指定的测试包中。返回值用来确认是否执行成功。

CU_ErrorCode CU_register_suites(CU_SuiteInfo suite_info[])

CU_ErrorCode CU_register_nsuites(int suite_count, ...)

对于包含 Test 和 Suite 的测试体系， 管理 Test 同 Suite 的联系以及 Suite 注册的的工作既繁杂又易错。CUnit 制定了专用于管理 Suite 和 Test 的系统，有效地把包的注册和 Test 的组合集中起来，并减少了用户编写错误检验代码的工

作量。

Test 会首先组成 CU_TestInfo 类型的数组(在 <CUnit/TestDB.h>中定义)：

```
CU_TestInfo test_array1[] = {
   { "testname1", test_func1 },
   { "testname2", test_func2 },
   { "testname3", test_func3 },
   CU_TEST_INFO_NULL,
};
```

每个数组元素包含属于一个测试用例的唯一的名称和测试函数，根据宏命令 CU_TEST_INFO_NULL 定义，每个数组的结尾元素都必须是一个 null 值。在 CU_TestInfo 数组中的测试用例组成了注册到一个 Suite 的用例集。

Suite 的信息定义在一个或多个 CU_SuiteInfo（在<CUnit/TestDB.h>中定义）数组中：

```
CU_SuiteInfo suites[] = {
   { "suitename1", suite1_init-func, suite1_cleanup_func, test_array1 },
   { "suitename2", suite2_init-func, suite2_cleanup_func, test_array2 },
   CU_SUITE_INFO_NULL,
};
```

数组中的每个元素都包含唯一的名称和包初始化函数、包清除函数以及包的 CU_TestInfo 数组。同样，在给定的包不需要初始化函数和清除函数时，它们可以为 null，数组的最后一个元素必须为 null 值，可以用 CU_SUITE_INFO_NULL 宏命令。

CU_SuiteInfo 数组中定义的所有测试包都通过一个语句进行注册：CU_ErrorCode error = CU_register_suites(suites)；

在包或 Test 的注册过程中出现错误时返回错误代码，同 suite registry 和 test addition 操作返回的错误码是相同的。调用 CU_register_nsuites()函数可以在一个语句中注册多个 CU_SuiteInfo 数组：

CU_ErrorCode error = CU_register_nsuites(2, suites1, suites2);

函数参数包含 CU_SuiteInfo 数组，其个数不定，第一个参数表示 CU_SuiteInfo 数组的数目。

4.5 V1 被取代的数据类型和函数

下面是在第二版中已经取消的数据类型和函数，若要这些函数或数据类型名称，用户代码的编译要引用已定义的 USE_DEPRECATED_CUNIT_NAMES 文件。

#include <CUnit/TestDB.h> (由< CUnit/CUnit.h>自动加载)

旧名称	等效的新名称
TestFunc	CU_TestFunc
InitializeFunc	CU_InitializeFunc
CleanupFunc	CU_CleanupFunc
_TestCase	CU_Test
PTestCase	CU_pTest
_TestGroup	CU_Suite
PTestGroup	CU_pSuite
add_test_group()	CU_add_suite()
add_test_case()	CU_add_test()
ADD_TEST_TO_GROUP()	CU_ADD_TEST()
test_case_t	CU_TestInfo
test_group_t	CU_SuiteInfo
test_suite_t	不对等使用 CU_SuiteInfo
TEST_CASE_NULL	CU_TEST_INFO_NULL
TEST_GROUP_NULL	CU_SUITE_INFO_NULL
test_group_register	CU_register_suites()
test_suite_register	不对等使用 CU_register_suites()

5. 运行测试

5.1 概述

```
#include <CUnit/Automated.h>
void            CU_automated_run_tests(void)
CU_ErrorCode CU_list_tests_to_file(void)
void            CU_set_output_filename(const char* szFilenameRoot)
#include <CUnit/Basic.h>
typedef enum      CU_BasicRunMode
CU_ErrorCode      CU_basic_run_tests(void)
CU_ErrorCode      CU_basic_run_suite(CU_pSuite pSuite)
CU_ErrorCode CU_basic_run_test(CU_pSuite pSuite, CU_pTest pTest)
void         CU_basic_set_mode(CU_BasicRunMode mode)
CU_BasicRunMode CU_basic_get_mode(void)
void      CU_basic_show_failures(CU_pFailureRecord pFailure)
#include <CUnit/Console.h>
void CU_console_run_tests(void)
#include <CUnit/CUCurses.h>
void CU_curses_run_tests(void)
#include <CUnit/TestRun.h> (由<CUnit/CUnit.h>自动加载)
   unsigned int CU_get_number_of_suites_run(void)
   unsigned int CU_get_number_of_suites_failed(void)
   unsigned int CU_get_number_of_tests_run(void)
   unsigned int CU_get_number_of_tests_failed(void)
   unsigned int CU_get_number_of_asserts(void)
   unsigned int CU_get_number_of_successes(void)
   unsigned int CU_get_number_of_failures(void)

   typedef struct CU_RunSummary
   typedef CU_Runsummary* CU_pRunSummary
   const CU_pRunSummary CU_get_run_summary(void)
   typedef struct CU_FailureRecord
   typedef CU_FailureRecord*   CU_pFailureRecord
   const CU_pFailureRecord CU_get_failure_list(void)
   unsigned int CU_get_number_of_failure_records(void)
```

Cunit 能运行测试包中注册的所有的测试，也能运行单个的包或测试。在每次执行过程中，Framework 记录 Suite 和 Test 个数，执行的断言数目以及通过和失败的情况。请注意：在每个测试初始化的时候会清空测试结果记录（即便本次测试失败仍然会清空结果记录）。

尽管 CUnit 提供了测试函数来运行 Suites 和 Tests，用户仍希望通过使用简单的用户界面，响应用户和 CUnit framework 之间的交互并输出测试结果及详细说明。

下面是 CUnit 库中包含的集中用户接口：

接口	平台	含义
Automated	所有平台	非交互 输出到 xml 文件
Basic	所有平台	非交互 可选输出到 stdout
Console	所有平台	可交互 用户控制的控制台模式
Curses	Linux/Unix 平台	可交互 用户控制的 curses 模式

如果这些接口不能满足使用，可用<CUnit/TestRun.h>中定义的 API 实现，参照各个接口的实现代码。

5.3 自动模式

自动模式是非交互的，用户初始化测试运行并输出结果到 xml 文件。注册的测试和测试包也会输出到 XML 文件中。

以下函数构成自动模式下的 API：

void CU_automated_run_tests(void)

运行注册的所有测试包的所有测试，测试结果输出到名为 ROOT-Results.xml.的文件，文件名 ROOT 可通过 CU_set_output_filename() 修改，或者使用默认的 CUnitAutomated-Results.xml。请注意如果没有重命名，原来的测试结果记录会被覆盖。

测试结果支持文档类型的文件(如：CUnit-Run dtd)和 XSL 格式的文件(如 CUnit-Run xsl)，可从共享的子目录和安装路径查看。

CU_ErrorCode CU_list_tests_to_file(void)

在文件中列出注册的 Suite 和相关的 Test，文件名为 ROOT-Listing.xml。ROOT 可通过 CU_set_output_filename() 修改，或者使用默认的 CUnitAutomated-Results.xml。请注意如果没有重命名，原来的测试结果记录会被覆盖。

列表文件支持文档类型的文件(如：CUnit-List.dtd)和 XSL 格式的文件(如

CUnit-List.xsl)，可从共享的子目录和安装路径查看。

请注意：列表文件不能通过 CU_automated_run_tests()自动生成，当需要此文件的时候用户代码必须显式地定义一个 listing。

void CU_set_output_filename(const char* szFilenameRoot)

通过 szFilenameRoot 分别同-Results.xml 或-Listing.xml 连接，为输出结果和列表文件设定输出文件名称。

5.4 Basic 模式

Basic 模式也是非交互的，输出结果到 standard output 中。接口能独立运行 Suite 或 Test，并且在每次运行过程中可以通过用户代码控制显示输出的类型，为用户访问 API 提供便利。提供以下的公有函数：

CU_ErrorCode CU_basic_run_tests(void)

运行注册的 Suite 中所有的 Test，返回测试运行过程中的第一个错误码，由当前的运行代码通过 CU_basic_set_mode()来控制输出的类型。

CU_ErrorCode CU_basic_run_suite(CU_pSuite pSuite)

运行指定的包中的所有测试，返回测试运行过程中的第一个错误码，由当前的运行代码通过 CU_basic_set_mode()来控制输出的类型。

CU_ErrorCode CU_basic_run_test(CU_pSuite pSuite, CU_pTest pTest)

运行指定的包中的一个测试，返回测试运行过程中的第一个错误码，由当前的运行代码通过 CU_basic_set_mode()来控制输出的类型。

void CU_basic_set_mode(CU_BasicRunMode mode)

设置 basic 运行模式，控制测试运行过程中的输出，选项包括：

CU_BRM_NORMAL：打印失败结果和运行信息。

CU_BRM_SILENT：只在出现错误时打印输出。

CU_BRM_VERBOSE：输出详细运行信息。

CU_BasicRunMode CU_basic_get_mode (void)：

返回当前 basic 运行模式代码。

void CU_basic_show_failures(CU_pFailureRecord pFailure)：

所有失败的信息打印输出到 stdout，与运行模式无关。

5.5 交互的控制台模式

控制台模式是可交互的，客户端所要完成的工作就是初始化控制台会话，且用户交互式地控制测试的运行，包括选择已注册的 Suite 和 Test 并运行，查看测试结果。使用 void CU_console_run_tests(void) 开始控制台会话。

5.6 交互的 Curses 模式

Curses 模式也是交互的，客户端所要完成的工作就是初始化 Curses 会话，且用户交互式地控制测试的运行，包括选择并运行已注册的 Suites 和 Tests，查看测试结果。使用这种模式要在应用程序中连接 Curses 库。用 void CU_curses_run_tests(void) 开始 Curses 会话。

5.7 取得测试结果

界面可以显示测试运行的结果，但是代码有时需要直接访问测试结果，包括运行过程的各种统计信息和失败的信息列表。请注意：每次新的测试开始或者进行初始化或者清除时都会覆盖原有的测试结果。访问测试结果的函数有：

unsigned int CU_get_number_of_suites_run(void)

unsigned int CU_get_number_of_suites_failed(void)

unsigned int CU_get_number_of_tests_run(void)

unsigned int CU_get_number_of_tests_failed(void)

unsigned int CU_get_number_of_asserts(void)

unsigned int CU_get_number_of_successes(void)

unsigned int CU_get_number_of_failures(void)

这些函数报告了在上一次测试过程中执行通过或失败的 Suite、Test 和断言的数目。如果 Suite 的初始化或者清空函数的返回值不为空，Suite 运行失败。如果 Test 中的任何一个断言执行不通过，则测试运行失败。最后三个函数指的都是断言。分别运行 CU_get_registry()->uiNumberOfSuites 和 CU_get_registry()->uiNumberOfTests 取得 Suite 或 Test 的数目。

const CU_pRunSummary CU_get_run_summary(void)

快速获得测试结果数据统计，返回指向包含统计结果的结构的指针，数据类型在<CUnit/TestRun.h>定义(由<CUnit/CUnit.h>自动加载)。

```
typedef struct CU_RunSummary
        {
          unsigned int nSuitesRun;
          unsigned int nSuitesFailed;
          unsigned int nTestsRun;
          unsigned int nTestsFailed;
          unsigned int nAsserts;
          unsigned int nAssertsFailed;
          unsigned int nFailureRecords;
        } CU_RunSummary;
typedef CU_Runsummary* CU_pRunSummary;
```

这个结构和其返回的指针属于 CUnit Framework，用户不能进行释放或改变。请注意：一旦其他的测试被初始化，指针有可能不能使用。

const CU_pFailureRecord CU_get_failure_list(void)

取得上次测试运行过程中失败的信息列表（没有错误返回 null）。返回值的类型在<CUnit/TestRun.h>（由<CUnit/CUnit.h>自动加载）中声明，每一条失败信息包括位置和类型。

```
typedef struct CU_FailureRecord
        {
          unsigned int   uiLineNumber;
          char*          strFileName;
          char*          strCondition;
          CU_pTest       pTest;
          CU_pSuite      pSuite;

          struct CU_FailureRecord* pNext;
          struct CU_FailureRecord* pPrev;

        } CU_FailureRecord;
        typedef CU_FailureRecord*   CU_pFailureRecord;
```

这个结构和其返回的指针属于 CUnit Framework，用户不能进行释放或改变。请注意：一旦另一个测试被初始化，指针有可能无法使用。

unsigned int CU_get_number_of_failure_records(void)

取得由 CU_get_failure_list()返回的失败列表中 CU_FailureRecords 类型的记录的数目。请注意：可能会比失败的断言的数目多，因为包括了 Suite 的初始化和清空。

5.8 V1 被取代的数据类型和函数

下面是在第二版中已经取消的数据类型和函数，在用到这些函数或数据类型名称时，原有的代码的编译要引用 USE_DEPRECATED_CUNIT_NAMES 文件。

旧名称	等效的新名称
automated_run_tests()	CU_automated_run_tests() plus CU_list_tests_to_file()
set_output_filename()	CU_set_output_filename()
console_run_tests()	CU_console_run_tests()
curses_run_tests()	CU_curses_run_tests()

6. 异常处理

6.1 概要

```
#include <CUnit/CUError.h> (由<CUnit/CUnit.h>自动加载)
  typedef enum CU_ErrorCode
  CU_ErrorCode     CU_get_error(void);
  const char*      CU_get_error_msg(void);

  typedef enum CU_ErrorAction
  void                CU_set_error_action(CU_ErrorAction action);
  CU_ErrorAction CU_get_error_action(void);
```

6.2 CUnit 出错处理

大多数的 Cunit 函数在发生错误的时候，都会给出相应的错误码来说明错误信息。有些函数会返回错误码，也有些函数只是给出了错误码但不将错误码作为返回值。Cunit 提供了本身的错误处理函数，主要是以下两个：

CU_ErrorCode CU_get_error(void)

const char* CU_get_error_msg(void)

第一个函数返回错误代码本身，第二个函数返回描述错误的信息。错误代码是一个 CU_ErrorCode 型的枚举变量。主要有以下几种错误代码：

错误代码	含义
CUE_SUCCESS	成功
CUE_NOMEMORY	存储分配失败
CUE_NOREGISTRY	registry 未初始化
CUE_REGISTRY_EXISTS	试图在 CU_cleanup_registry()前执行 CU CU_set_registry()
CUE_NOSUITE	CU_pSuite 指针为空
CUE_NO_SUITENAME	CU_Suite 名称不存在
CUE_SINIT_FAILED	Suite 初始化失败
CUE_SCLEAN_FAILED	Suite 清除失败
CUE_DUP_SUITE	不允许 suite 重名
CUE_NOTEST	请求的 CU_pTest 指针为 NULL.
CUE_NO_TESTNAME	CU_Test 名称不存在
CUE_DUP_TEST	不允许测试用例重名
CUE_TEST_NOT_IN_SUITE	测试未在指定的包中注册
CUE_FOPEN_FAILED	打开文件出错
CUE_FCLOSE_FAILED	关闭文件出错
CUE_BAD_FILENAME	文件名不合法.
CUE_WRITE_ERROR	写文件出错

6.3 Behavior Upon Framework Errors

默认情况下在出现错误时会设置错误代码而程序继续执行，但很多时候我们希望在出现错误时程序停止执行或应用程序退出，错误行为可由用户设定，有两个函数可用：

```
void CU_set_error_action(CU_ErrorAction action)
CU_ErrorAction CU_get_error_action(void)
```

错误行为代码是在<CUnit/CUError.h>中定义的 CU_ErrorAction 型的枚举变量，主要有以下几种错误行为代码：

错误码	含义
CUEA_IGNORE	（默认）在错误出现时继续运行
CUEA_FAIL	在错误出现时停止运行
CUEA_ABORT	错误出现时应用程序退出

6.4 V1 被取代的变量和函数

下面是是在第二版中已经取消的数据类型和函数，在用到这些函数或数据类型名称时，原有的代码的编译要引用 USE_DEPRECATED_CUNIT_NAMES 文件。

旧名称	等效的新名称
get_error()	CU_get_error_msg()
error_code	None. Use CU_get_error()

面向对象的测试用例设计思想

作者：大傻

摘要：

如何设计好测试用例一直是困扰我们测试人员一个问题，设计结构清晰、易读、可维护性强测试用例更是我们测试人员一个大难题，更是一个头疼的事情。

这段我时间我一直在考虑这个问题，通过实践模索，我个人认为测试用例设计应该也有自己的设计思想，虽然现在还没有一套测试用例设计方面思想，但是程序设计已经很多成熟的设计思想如面向对象、原型方法、结构设计等，我们为什么不能借鉴呢？为什么不能把测试人员的【测试用例】当作开发人员【程序代码】来设计呢？

关键词：用例、设计、思想、面向对象

解决方案：

测试用例设计我们可以通过过程和方法两方面来考虑，并且借鉴程序设计过程和方法来设计我们的测试用例，让我们测试用例结构清晰、易读、可维护性强，提高设计测试用例质量和效率。

一、过程

我们可以参见程序设计过程跟我用例设计过程进行对比。请大家不要误会的是，这里的对应不是阶段、时间的对应，而是程序设计过程和测试用例设计过程中活动对照。

编号	开发过程	用例过程	我们做什么？
1	需求分析	测试需求	根据需求拆分、合并、整理成我们测试需求。
2	概要设计	初步设计	根据测试需求组装、拆分成我们测试用例大纲，并确定测试大纲包含各种测试有效并且是主导性分支。 要设计多少测试用例（模块），其中哪些测试用例可以做成公用用例（接口）
3	说细设计	用例编写	根据本文 3.2 方法和大家熟知的白盒、黑盒测试用例设计方法来设计我们的测试用例。
4	代码编写	用例执行	根据设计并且评审好的测试用例来执行测试。

这个过程我要重点说一下第 2 个过程初步设计,初步设计好坏直接影响以后用例复杂度，我们需要尽量减少用例冗余度。如下表【原设计测试用例】有点冗余了。

测试需求	自费病人、公费用病人挂号收费
原设计测试用例	030101 初诊自费无账户病人 030102 复诊自费有账户病人 030103 初诊公费无账户病人 030104 复诊公费有账户病人
精简后测试用例	030103 自费病人 030104 公费病人

设计测试用例大纲时必须考虑主导问题，不然测试用例变得非常复杂了。如上表我们就可以病人性质（自费 or 公费）为主导，初诊、复诊、有无账户作为测试用例里的因素来设计，那么用例会更简洁清晰，执行效率更高、更有效。

二、方法

面向对象的测试用例设计方法，实际上就是借鉴面向对象的程序设计方法，将我们测试数据或测试用用例当作一个个对象，每个数据、测试用例可以根据需要再进行组合成新的数据集或测试用例，每个测试用例也可以将自己子用例当作方法处理。这些数据、测试用例、数据集、子用例也可以根据需要定义成私有还是公用的，并作为其它测试用引用或作为前提条件。

我们可以将用例中各中元素，根据当前系统业务，加上自己一点想象力有机组合，把它设计成一个个公用对象、公用函数、变量、结构体等，下表简单举例几种情况。

测试用例中内容	可设计对象	举例说明	
数据	变量	用户属性，病人属性，厂家属性等	实例 1
	结构体	药品列表，费用列表，用户列表等	实例 2
	……		
公用测试用例	公用函数	界面规范，安全规范，SQL 语句规范等	实例 3
	公用对象	查询控件，报表设置控件、折旧方法对象等	实例 4
	……		
……			

实例 1：病人属性医保性质，原来两种“省医保”，“市医保”，我们测试用例编写快结束的时候，客户说要增加一个“区医保”，病人医保性质几乎要涉及到 60%的测试用例，要以前我们可能要花好几天来更新所有测试用例，现在只要花上两小时就可以了，只要在医保性质测试用例增加一个“区医保”政策及算

法的子用例就行了。

实例 2：药品信息我们设计成一个数据集，有一次项目进行了设计变更药品信息表中要加一个批次的字段，我们只要在药品信息数据集中增加批次即可，更新测试用例只花几分钟的时间。

实例 3：如果我们有一个产品升级项目上，离上次升级已经有 3 年了，原有测试用例有好多涉及界面规范，安全规范，SQL 语句编写规范内容，这几年这些规范了也更新好几个版本，如果我采有面向对象的测试用例设计方法我们只要更新一下用例库规范内容就行了。

例子 4：最近我们再进行资产系统升级，我们将折旧方法设计公用测试用例。后面折旧方法中工作量法有变更，我就只要修改一下了折旧方法公用测试用例中工作量法子用例就好了。

三、实例

测试用例名称：医技用药申领

数据对象：

标识	费用名称	单价	数量	金额	自负比例
A	阿莫西林胶囊	32.20	2	64.40	50%
B	芬必得胶囊	2.75	3	8.25	
C	表阿霉素针	248.00	1	248.00	
D	乙胺丁醇片	21.00	4	84.0	70%
E	汞	30.82	2	61.64	
F	乳果糖口服液	39.3	1	39.30	
【用药单 1】	A 至 F				
【用药单 1】	A+C+D+F				

用例编号	YJ01_01	用例名称	用药申领及提交
前提条件	医技系统选项目设置发药药房，病区有未出院病人		

步骤名称	输入描述	预期结果	实际结果
1	通过住院号或床号调出病人，输入【用药单 1】，点击保存	保存成功，病人账户未记账	
2	退出医药申领模块，重新打开，输入【步骤 1】中住院号或床号	显示【用药单 1】	
3	操作【步骤 2】点击提交	对应药房收到【用药单 1】，病人账户未记账，提交的用药记录不能修改	
4	操作【步骤 3】，然后操作【步骤 2】	已提交的用药记不能修改	

用例编号	YJ01_02	用例名称	药房医技用药发药
前提条件	完成 YJ01_01，医技科室设置为药房发药科室		

步骤名称	输入描述	预期结果	实际结果
1	打开普通医嘱发药，选择发药方式和发药病区	显示【用药单 1】对应的发药信息	
2	操作【步骤 1】点击发药	药房库存减少【（A+B+C+D+E+F）数量】，病人账户减少【（A*0.5+B+C+D*0.7+E+F）金额】	
3	操作【步骤 1】，只发【C+D】，点击发药	药房【(C+D)数量】数据库存减少，病人账户减少【（C+D*0.7）金额】	
4	操作【步骤 3】，再操作【步骤 3】	显示【A+B+E+F】的发药信息	
5	操作【步骤 4】，点击发药	药房【(A+B+E+F)数量】库存减少，病人账户减少【（A*50+B+E+F）金额】	

RDP 协议介绍

作者：Tim Chase 译者：肖艳霞

HP 公司发布的 LoadRunner 9.0 版本引进了几个新的协议，其中一个是微软远程桌面协议（Microsoft Remote Desktop Protocol ，RDP）。该协议允许录制终端服务器会话的脚本，从脚本的角度来看，它与 LoadRunner 早期版本的“Citric ICA”协议相似。许多在 Citrix 协议的脚本录制过程中使用的好方法可用于 RDP 会话录制中。在深入研究 LoadRunner 如何执行 RDP 协议之前，我们先浅谈一下 RDP 究竟是什么。

RDP 是什么？

终端服务器是微软 Windows 服务器的一项附加服务，它允许你发起从本地电脑到服务器的会话。当你通过本地电脑登录服务器时，你可运行服务器上的所有进程。该项服务还会提供给客户端一种可视化操作界面，在界面上可实时显示服务器运行状态。RDP 就是这样一种用于连接本地客户端到终端服务器，保持两者之间会话的协议。

在录制脚本之前，我们需要理解 RDP 如何工作，这一点很重要。当一个用户登录到一台终端服务器机器后，基本上，该用户可对服务器上运行状态、所运行的程序和进程一目了然。而对于自己的每个操作，输入或点击，用户也可从即时刷新的操作界面上看到反馈结果。正因为如此，LoadRunner 无法动态捕捉屏幕的变化，它所能获取当前服务器状态。所以需要记住这一点，录制脚本时必须插入同步点，以验证脚本回放结果是否正确。

计划

因为 RDP 是一种基于可视化操作界面的脚本录制协议，在开始录制脚本之前需要详细计划业务流程，这一点很重要。在准备录制脚本前，以独立于 LoadRunner 的角度，反复对业务流程进行检查，以避免录制脚本时出错。否则，脚本录制完成后，若脚本存在错误，我们虽然可以查看并修改脚本，但因为 RDP 所录制的操作包含大量的鼠标点击，窗口同步和键盘输入，要看懂由这些操作所转化的脚本语句可不是一件容易事情，所以事前的充分准备可帮助减少事后修改所带来的工作量，最终可节省时间，并可使所录制脚本更清晰易读。

录制等级

录制 RDP 脚本时，其中一个要注意的录制设置项是录制等级，该项设置决定使用哪个等级去录制脚本。默认设置是“高”，这意味着录制时捕捉到的任何输入都将作为“rdp_type”被写入脚本，比如双击鼠标将被记录为“rdp_mouse_double_click”，同时任何修饰键（Control 键、Alt 键、Shift 键）

的点击都会被记录入步骤。下一级是“低”，该等级的主要不同之处在于输入被记录为“rdp_key”（一次敲击一个键）。最后一级是“原始的”，当使用向上键、向下键、鼠标上、鼠标下功能时才生成脚本，同时所有修饰键的点击都不会被记录。一般开始录制时都会选择最高级，这会帮助增加脚本的可读性和可修改性。当录制脚本遇到困难时，则向下降一个等级。

位图同步

脚本录制完后，如何成功回放 RDP 脚本将会一项复杂工作，关键是使正确实现操作同步。RDP 中所谓的同步就是在屏幕上框出一部分区域，当所有用户都执行到该区域内操作时，脚本才能继续执行。在创建 RDP 协议同步点时需记住以下几点：

1、同步点在可录制脚本时或脚本录制完后创建。录制脚本时，可使用录制工具条上的按钮插入同步点。脚本录制完后，可通过树形结构中的快照创建同步点。

2、保持小面积的同步区域，把注意力集中在用户必须同时执行的操作上。同步区域越小，同步问题将越少。

3、如果脚本回放时出现同步失败，Loadrunner 将会弹出一个对话框。该对话框允许你对比录制时快照和回放时快照。如果回放时快照是正确的，你可在脚本中修改同步点设置，否则，你需要停止回放，并开始调试脚本。

4、RDP 同步具有一种叫做容忍度的特性，它允许同步中存在一定的灵活性。该容忍度可设为低、中、高和精确等级。容忍度等级越高，录制时快照和回放时快照的相似度越高。如果回放时同步因为不明确的原因而多次失败，可尝试降低容忍度。

坐标转移

RDP 协议的其中一项最有趣的特性是坐标转移。有时候目标物在应用中的位置发生变化，对于其它协议，仅有两个选择，测试员必须大篇幅修改脚本或重新录制脚本。RDP 协议尝试自动修正这个问题。它一旦发现一个目标物的坐标发生了变化，则修改与该目标物相关的功能的脚本或其它依赖于该目标物而存在的功能的脚本。例如，假设需要录制一个包含一个弹出窗口的应用软件的脚本，在该窗口用户需要输入名字并点击 Enter 键，若回放时弹出窗口的位置发生变化，RDP 协议不但会调整该窗口的所有功能所对应的脚本，还会对输入名字和 Enter 键点击这两个功能所对应的脚本进行调整。

不好之处

RDP 协议存在几个问题，首先它不支持 Windows XP 操作系统，若你尝试在 XP 操作系统录制脚本，LoadRunner 不会报错，却给你一个空白脚本。为了

正常录制脚本，LoadRunner 需要在运行 Windows 终端服务器的机器上录制脚本。

第二个问题是 RDP 协议不支持目标物的识别，LoadRunner 不提供被捕捉目标物的任何信息，这意味着脚本将会完全依赖位图同步。另一个不利之处你检查脚本时可发现的错误数量非常有限。目标物识别对于错误检查的影响是很大的，因为你可以从屏幕中获取文字信息以确保应用软件的显示结果与预期相符。

结论

RDP 和 Citrix 协议很相似。熟识 Citrix 协议的人可以很快掌握 RDP 协议的使用。RDP 协议还具有坐标转移和容忍度等其它协议没有的特性，而这些特性都很有帮助。该协议是 LoadRunner 9.0 的新增功能，因此它仍然是很不成熟的。随着越来越多的人开始使用它，我希望该项功能在未来可得到不断更新和完善，同时也希望不久之后，LoadRunner 的技术工程师可以将目标物识别技术并入 RDP 协议，以改善其可用性。

作者介绍：Tim Chase 是 Loadtester 团队的一名资深系统工程师，他同时也是 LoadRunner 的权威产品顾问和导师，读者可通过“tchase @ Loadtester.com”的 Email 地址联系他。

Toad Data Modeler 使用说明

作者：李元

简单介绍：

Toad Data Modeler 是 Quest 公司专业的数据库建模工具，以下是该软件的主要功能。

Key Features

- Physical Model
- Logical Model
- Support for Various Databases
- Reverse Engineering
- SQL/DDL Script Generation
- Alter Scripts Generation (Oracle 9i， Oracle 10g， Oracle 11g， MS SQL Server 2005 and MS SQL Server 2008)(差异脚本)
- HTML/RTF Reports (including Comparison reports)
- Version Manager(版本控制)
- Model Explorer
- Model Merge(模型合并)
- Model Compare(模型比较)
- Model Verification(模型检查)
- Package Explorer
- Script Explorer
- Autolayout
- Workspaces and Designers
- Editable Forms
- Dockable Panes
- Modeless Dialogs/Forms
- Categories
- Message Explorer
- Undo/Redo
- To-Do List
- Zoom， Loupe， Model Overview features and many more…

虽然是个专业工具，但是使用起来还是比较方便的，下面我就如何使用这个工具做个详细的讲解，希望对那些想要做数据库版本控制，以及规范化管理开发过程中的数据库资源的人有所帮助。

文档内容：

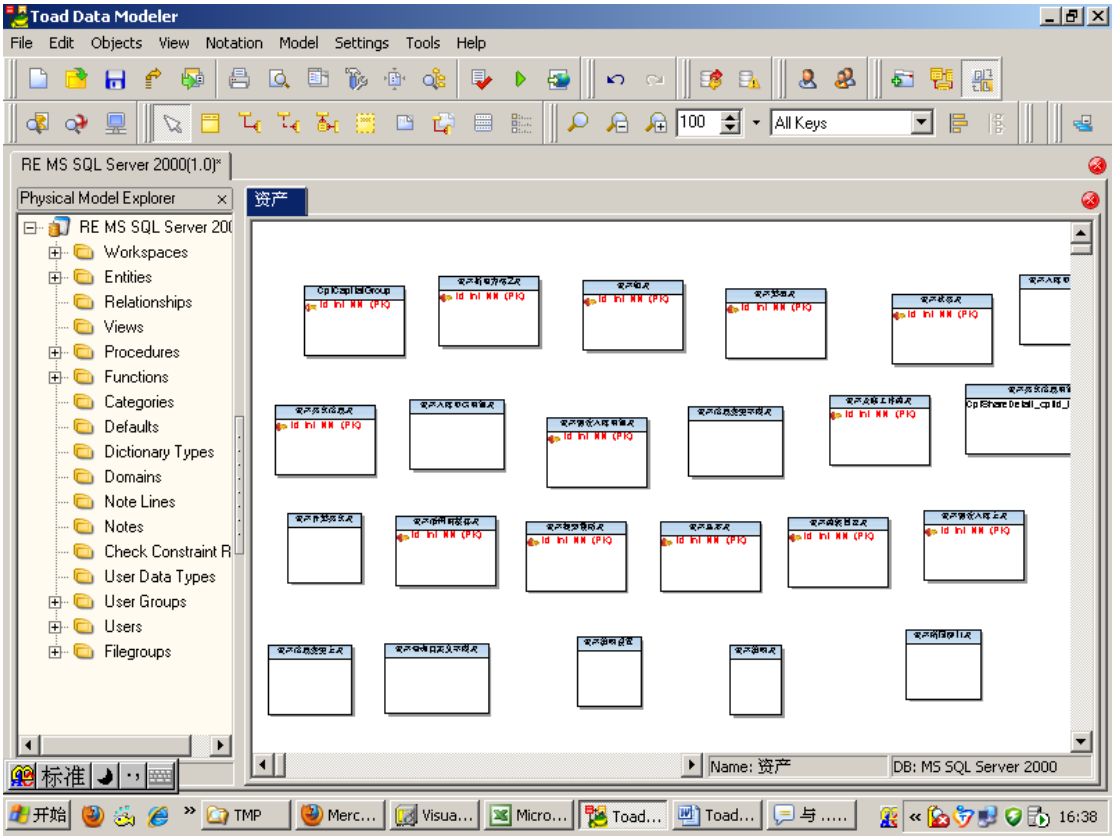
1．如何使用工具连接一个现有的数据库，产生模型文件。

2．如何根据自己编辑的模型，产生对应的脚本，或者连接数据库后，直接产生实体数据库。

3．如何通过工具自带的功能，做到数据库版本控制。

4．如何在开发过程中，使用工具，产生数据库的升级脚本。

这个软件是商业软件，像大多牛 X 公司一样，Quest 公司也提供了 15 天的免费试用版，大家可以下载，本文档使用的是 3.3.8.11 版本。界面如下图所示。

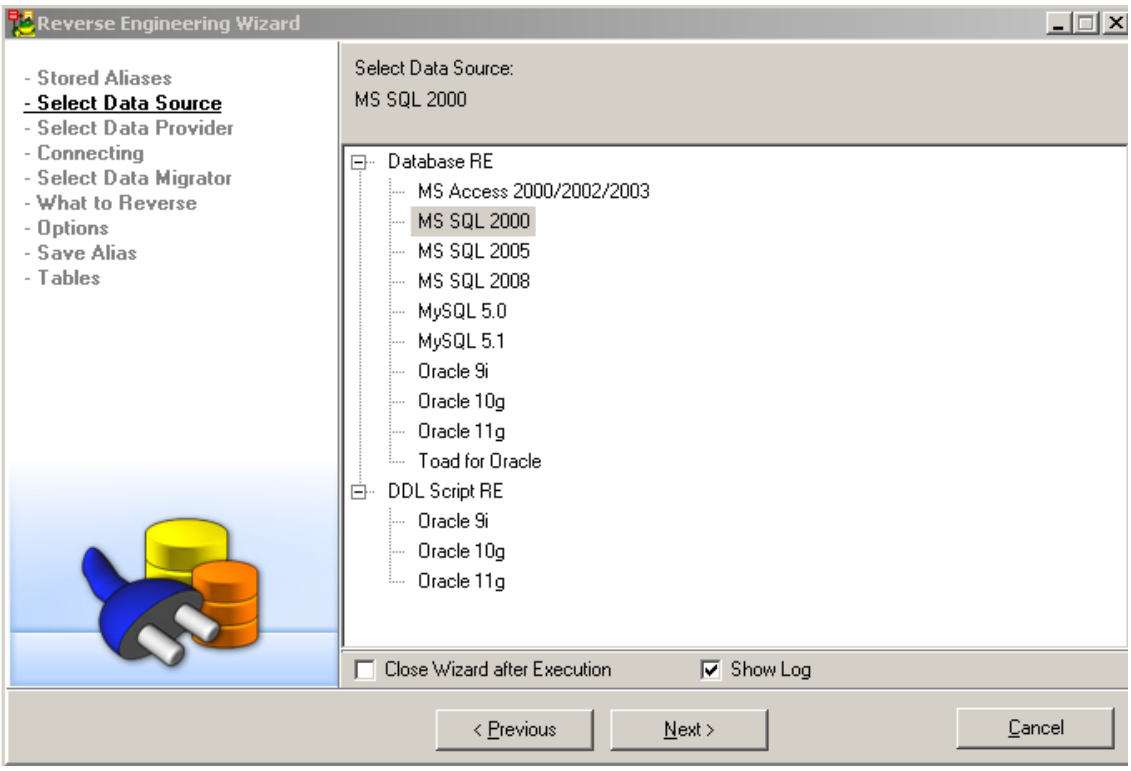


软件界面

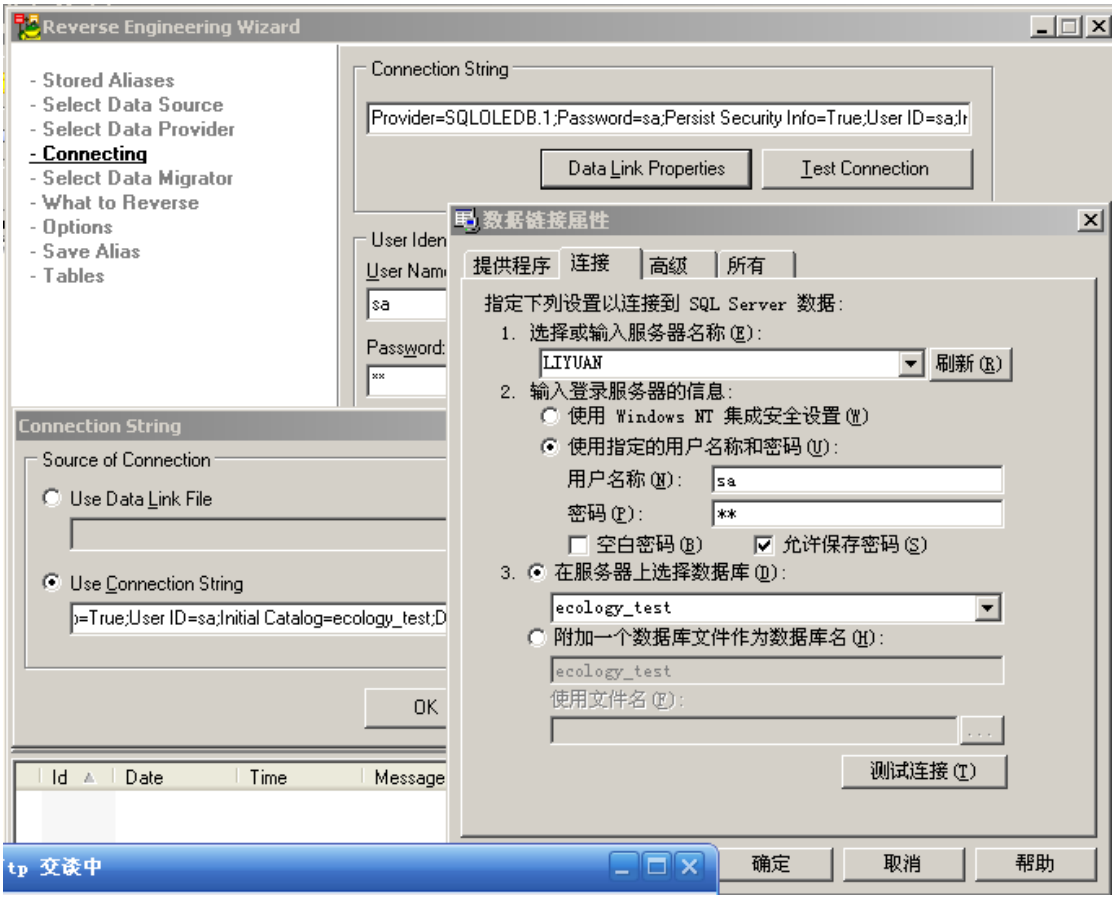
1. 导入数据库

首先我们来看一个最实用的功能，导入数据库模型。这个功能是很普遍的一个功能，大多数据库建摸工具都有，我在这里就简单说一下如何使用。

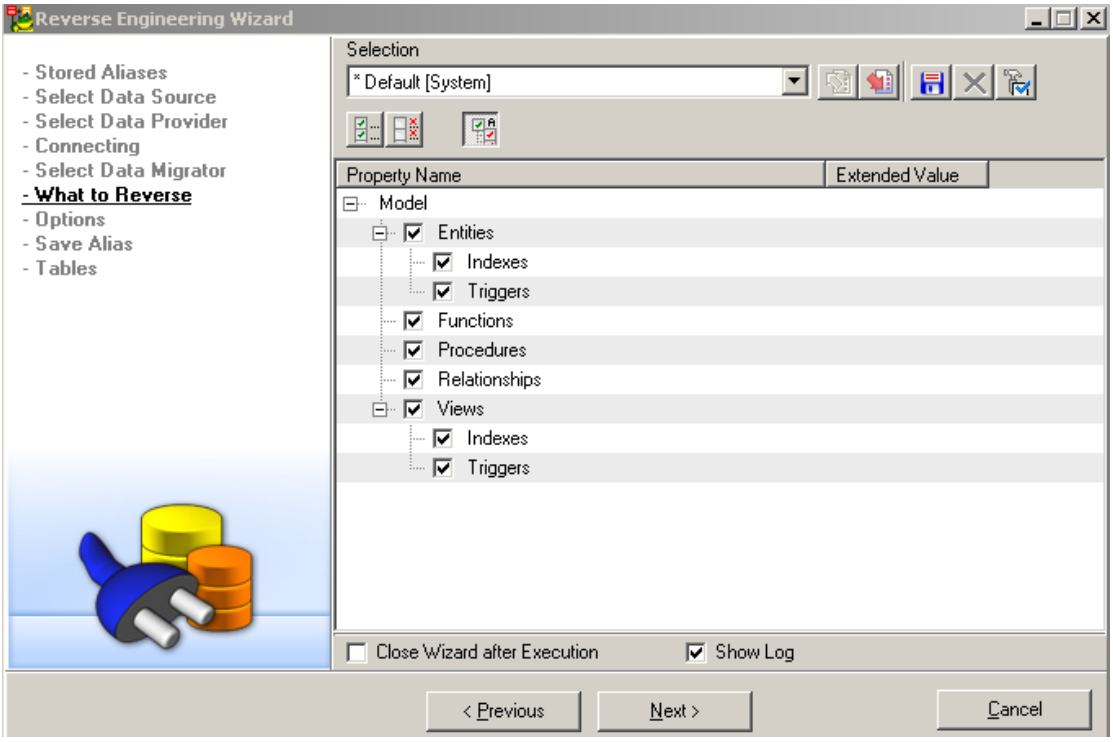
打开菜单 File－Reverse Engineering，点击下一步，出现如下图的操作界面：



选择需要的数据源，这里使用 MS SQL2000，点击 Next，出现连接设置界面如下图，选择对应的数据库。

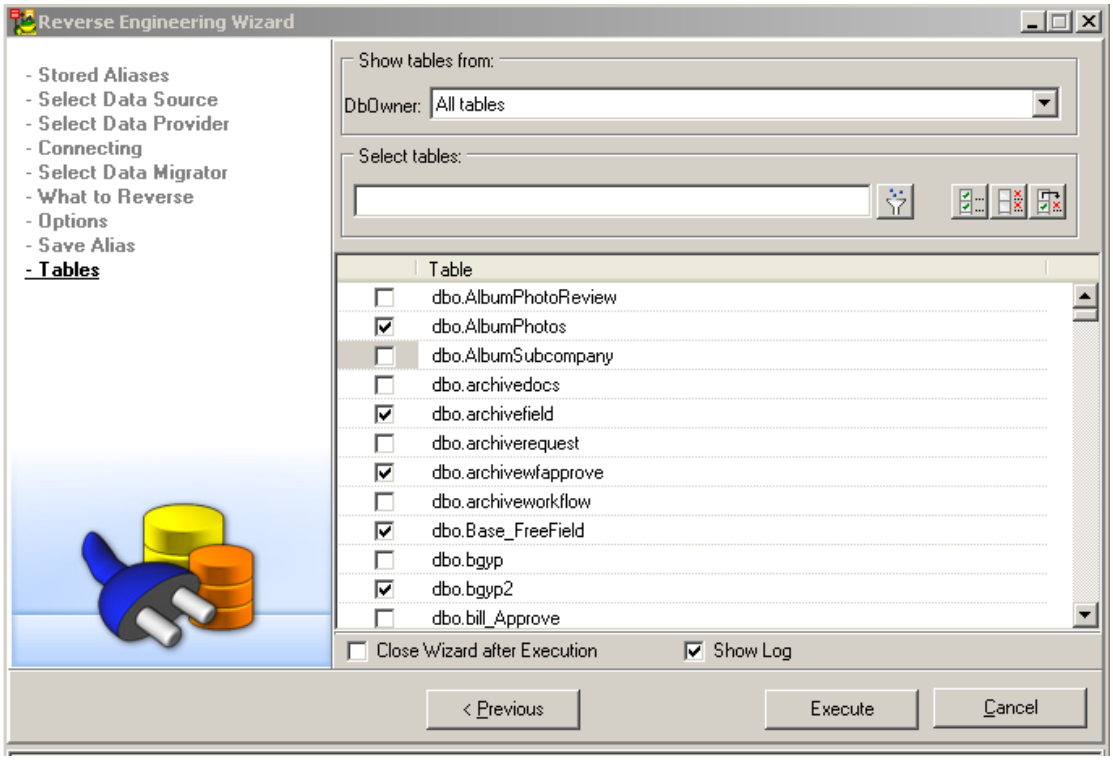


连接正确后，下一步，出现如下图的界面，系统提示选择需要反转的数据库对象。



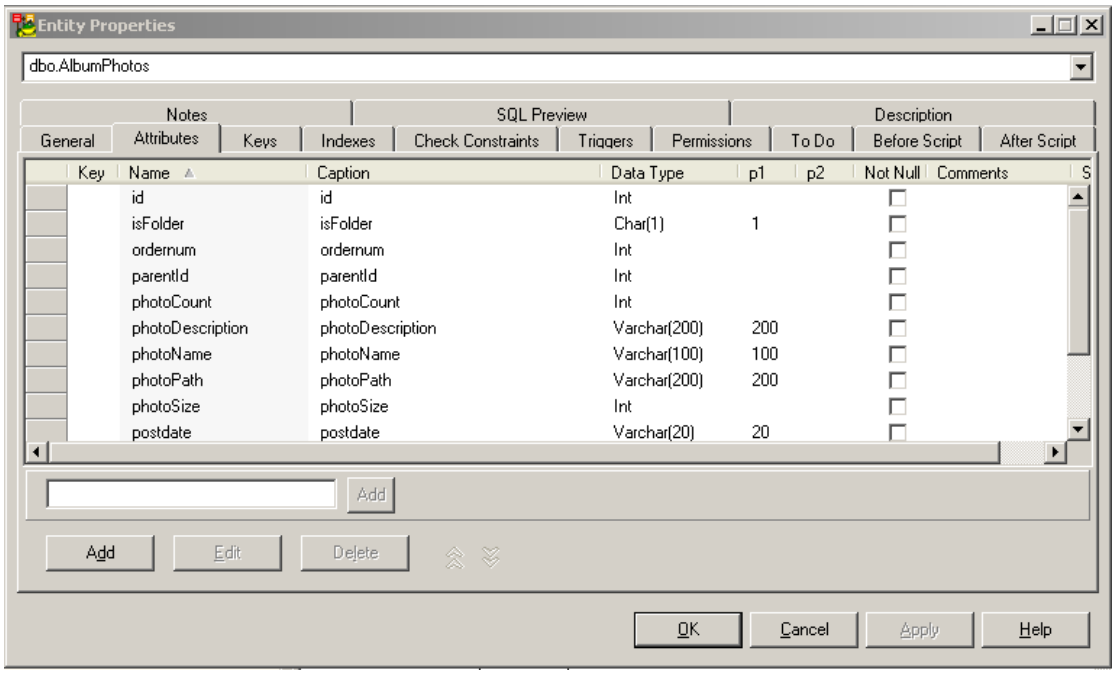
上图中有 Indexes（索引），Triggers(数据表)，Functions(函数)，Procedures（存储过程），等等。

之后都是下一步，没有特殊的要求，保持默认选项，最后如下图。



系统根据连接上的数据库信息，列出了所有可产生的对象，用户可以根据需要选择部分或者全部产生数据模型。选择完后，点击 Execute 按钮，执行操作。耐心等待一会，这个过程是比较慢的，尤其是数据库结构复杂的话。最终产生的结果，如开始那个图。至此，数据结构模型就导入成功了，你可以执行 File－Save Model 菜单，把导入成功后模型保存下来，以供以后使用。

补充一下:下图是编辑实体属性的界面，TDM 是个很强大的工具，有很全面的实体属性设置功能，而且都是通过图型化的方式实现的，使用比较简单，这个留给读者自己研究。



这里简单来说一下几个功能主菜单：

File 里的功能都是比较普遍的功能，这里不详细说明了；

Edit 里是工具使用中的一些操作功能，比如粘帖，复制，撤消什么的，这里不做详细说明；

Objects 里是实体对象的新增菜单，以及一些其他功能，也不详细说明；

View 不详细说明；

Notation 不详细说明；

Model 对模型的操作功能菜单都在这个主菜单中，是工具的重要菜单；

Setting 里面有个 Options 菜单，是系统功能设置的重点；

Tools 一些重要的功能工具在这个里面，比如版本控制的等等。

2. 导出数据库

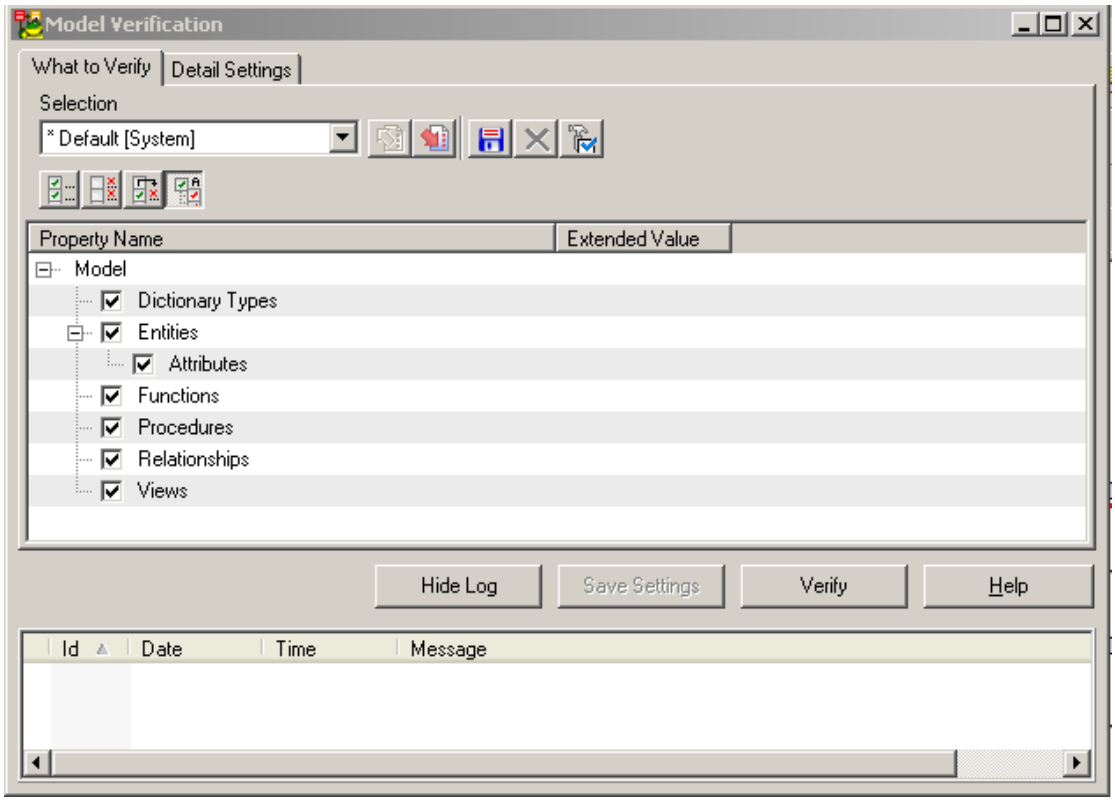
前面我们说了导入数据库的方法，现在我们来说一下如何导出数据库。

导出数据库分为两种：

一种就是产生数据库脚本文件。再放在数据库中去执行，产生实体数据库；另外一种是直接连接已有的数据库产生对应的实体数据库（遗憾的是 TDM 貌似没有提供这个功能，希望在以后的新版本中会出现）。

在介绍导出数据库前，我们先来看另外一个常用功能，Model–Verify Model（模型验证），简单来说，就是验证当前模型的逻辑关系是否正确，如果有不合理的逻辑，系统会提示。

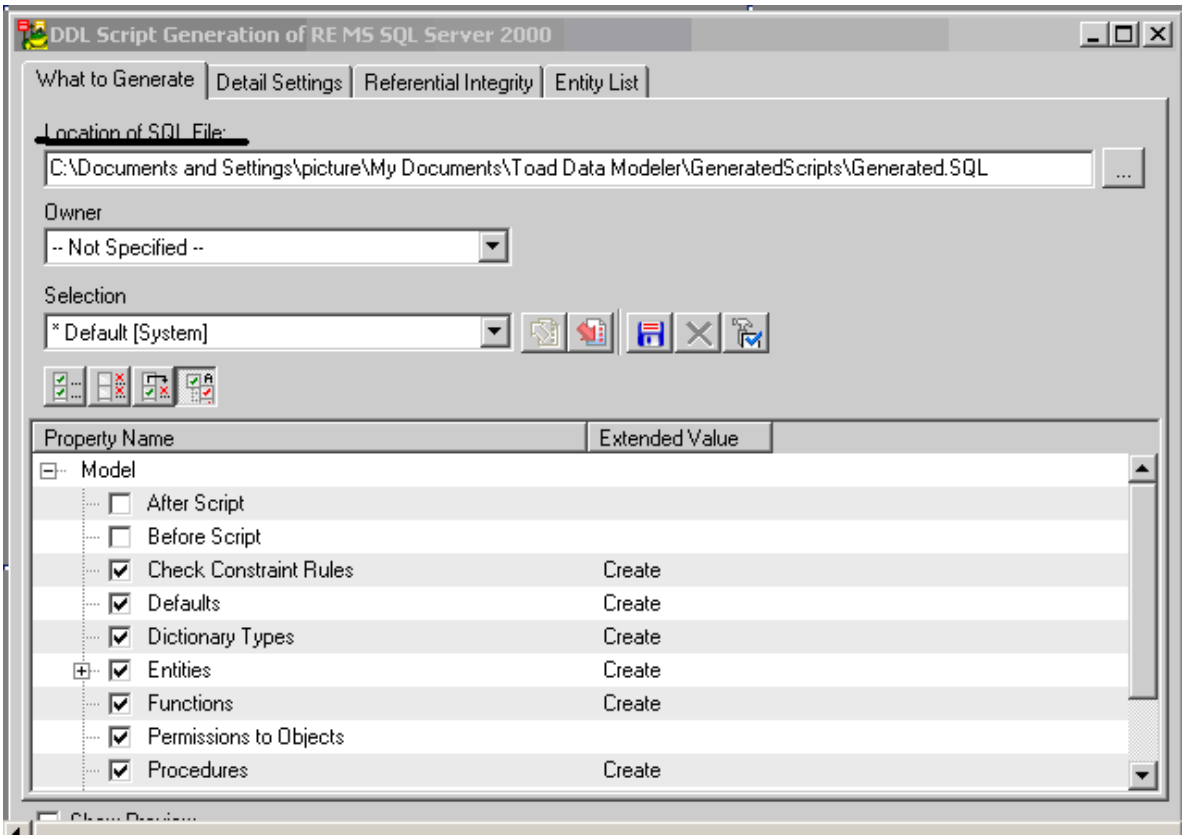
下图是 Model Verification 功能界面，中间部分是选择需要检查的对象，下面是系统的检查过程中的日志信息。



一般来说，在模型导出前，对模型做一下逻辑验证，这样能保证导出后的脚本创建数据库的成功。

下面我们来介绍一下如何使模型产生对应的脚本。

在当前打开了一个模型的情况下，用户可以点击 Model－Generate DDL Script 菜单，如下图所示，该页面有四个标签页，分别是 What to Generate（产生什么）、Detail settings（细节设置）、Referential Integrity（不理解这个标签页功能）、Entity list（模型对象列表）。



Location of SQL file 是产生脚本的存放位置，图中是系统默认的一个路径也可以通过 Setting-Option 菜单里的设置来调整这个路径。

Owner 是产生脚本在数据库里执行的时候所拥有者的意思，一般来说选择 Not Specify（不指定）。

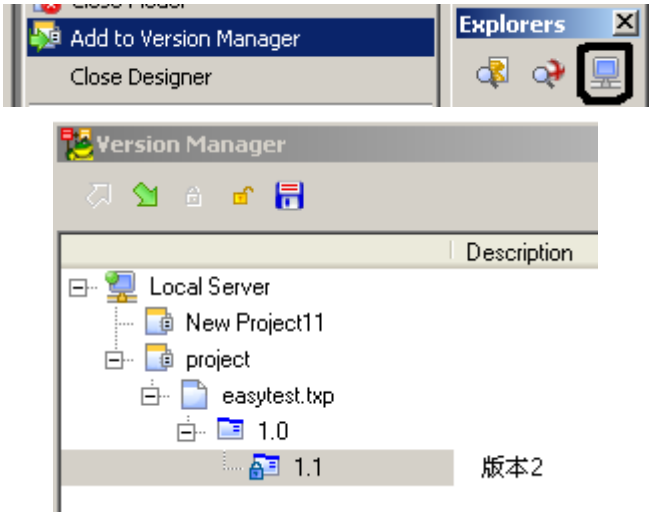
Selection 这个是对于下面的选择项的功能，用户可以保存下面的选择项，或者选择已有的选择配置文件，来快速，方便的完成筛选内容的设置。

完成相关的设置后，点击 Generate，对应的脚本就产生了，路径如图中所示。

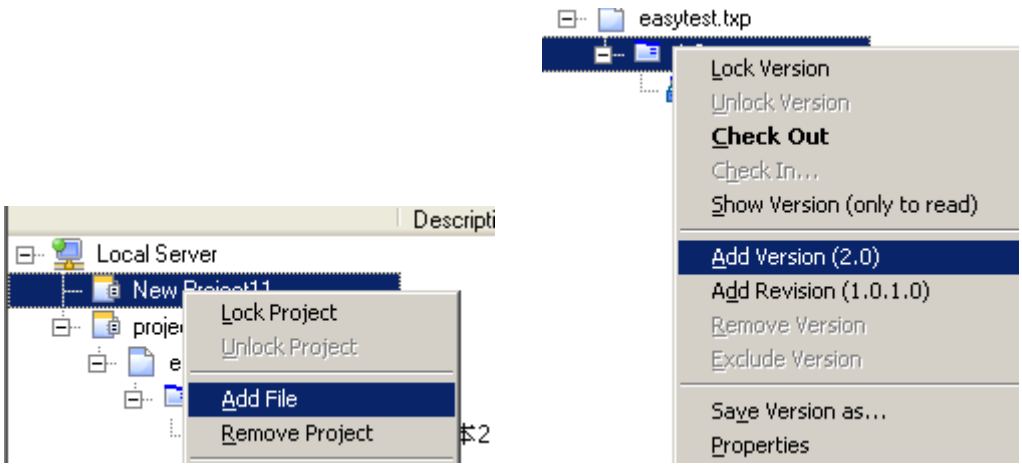
3. 数据库版本控制

下面我们来看一下版本控制功能，首先版本控制功能默认工具是不开启的，要到 Setting－Options 页面中，General 页勾选 Expert Mode（专家模式），启用该功能。

这样在操作页面中，就会有 Version Manager 功能按钮，或者通过 File－Add to Version Manager 功能进入下面页面。

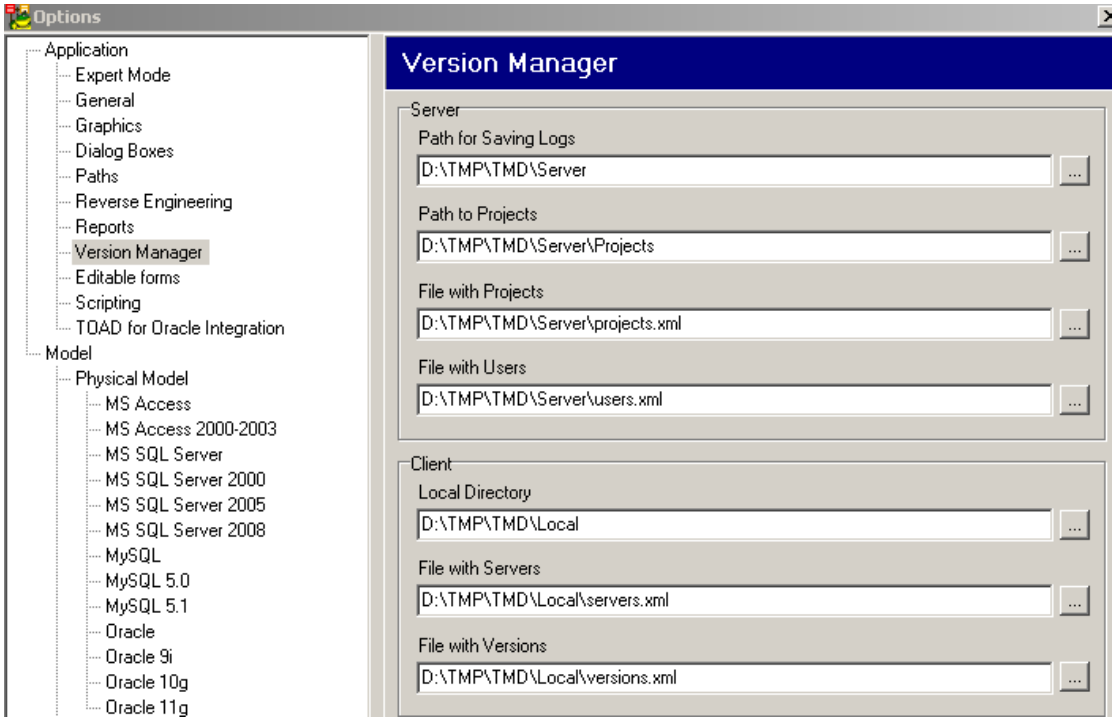


上图中分为三部分，先是定义一个版本控制的项目，如图中的 project 或 New Project11，然后通过右击项目，选择 Add File 功能，添加项目内容－已有的模型文件，比如这里的 easytest，这样，系统自动产生了一个模型初始版本，如图中的 1.0，然后我们可以通过右键功能，比如锁定，解锁，check out，check in 等菜单，来维护这个项目里的模型版本。如下图。



关于如何对版本控制做操作，这里就不详细说明了，与普通版本控制软件大致是一样的，要特别说明的是，TDM 也支持服务器－客户端模式。

如何设置 TDM 的服务器－客户端的版本控制管理：



以上是关于版本控制的路径设置页面，主要分为两部分，服务端和客户端，对于服务端的目录，比如上图中的 D:\TMP\TDM\Server，如果把它共享到网络上，网络上任意一个 TDM 的客户端程序，通过设置上图中的 Version Manager-Server，客户端就能识别到 Server 下的 project 里的版本。

注意：

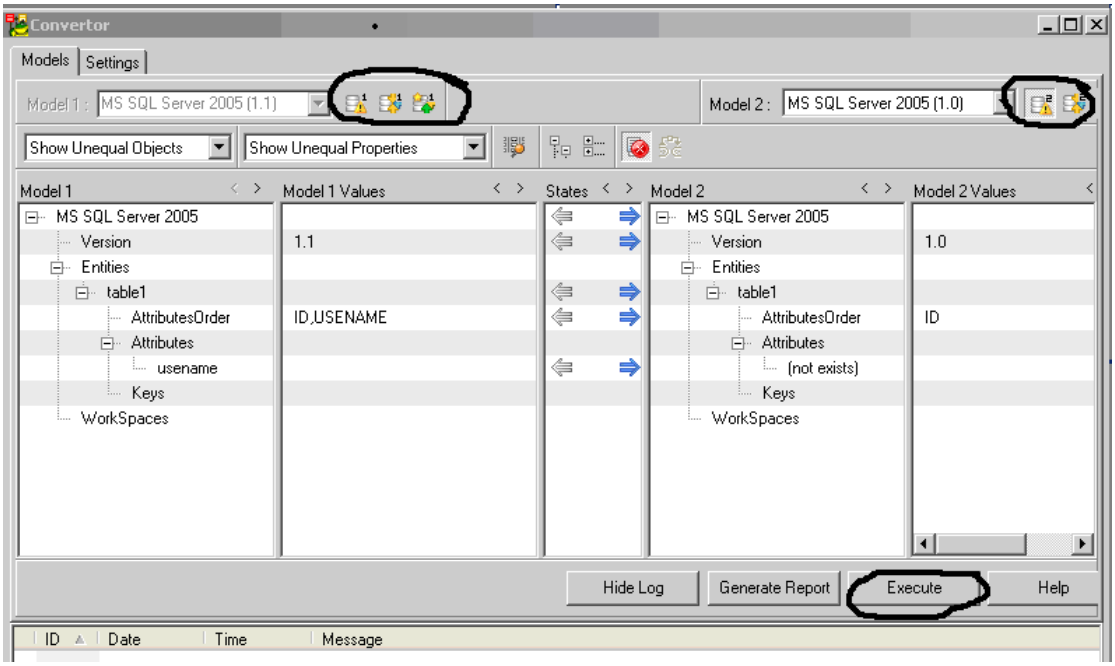
当然，如果要做到客户端能够更新项目版本，那就必须要对该 Server 目录有写的权限。

补充一下，如果是 TDM 服务器更换，或者是系统盘损坏需要重新部署，那也是很简单的事，只要把上图中的 TDM\Server 整个目录复制出来，放到需要的地方，然后用 TDM 去连接这个目录，整个项目控制又能生效了，非常方便，个人觉得比 VSS 什么的要方便很多。

4. 产生数据升级脚本(Alter Scripts)

最后我们要讨论的是一个许多 IT 公司都会碰到的问题，那就是数据库的升级脚本管理问题。对于一个已经投放市场的软件产品来说，数据库的更新肯定是必不可少的，不管是数据库的结构，还是程序需要的初始化数据，都是在不断变化的，这个时候如果没有一个工具的支持，完全靠人为的去维护，那是非常繁琐，并且及其容易出错，下面我们就来看一下 TDM 怎样帮助我们快速，正确，高效的管理数据库不同版本。

下图是 TDM 的 Convertor(变换器)界面。



- alter script generation (for Oracle 9i， Oracle 10g， Oracle 11g， MS SQL Server 2005 and MS SQL Server 2008)

以上文字说明，差异脚本功能，只针对 Oracle9i， Oracle10g， Oracle11g，MSSQL2005， MSSQL2008 数据库版本。

如上图中的标示部分，以下是 TDM 自带 help 的说明：

Generate Alter Script for Model 1

Enable to generate change script for Model 1.(Gets selected differences from Model 2 and generates alter scripts to modify database based on Model 1.)

从模型 2 上得到不同的内容 and 产生升级脚本为了修改模型 1。

Save Merged Model 1

Differences from Model 2 will be merged with Model 1 (Model1 + differences from Model2).

模型 2 的不同内容合并到模型 1 中。

Save Merged Model 1 to New Model

Differences from Model 2 will be merged with Model 1 and a new model will be created (Model1 + differences from Model2).

模型 2 的不同内容合并到模型 1 中，并且产生新的版本。

Generate AlterScript for Model 2

Enable to generate change script for Model 2.(Gets selected differences from Model 1 and generates alter scripts to modify database based on Model

2.)

从模型 1 上得到不同的内容 and 产生升级脚本为了修改模型 2。

Save Merged Model 2

Differences from Model 1 will be merged with Model 2 (Model2 + differences from Model1)

模型 1 的不同内容合并到模型 2 中。

Save Merged Model 2 to New Model

Differences from model 1 will be merged with Model 2 and a new model will be created (Model2 + differences from Model1).

模型 1 的不同内容合并到模型 2 中，并且产生新的版本。

Show Log(显示日志)- Shows log at the bottom of the form.In Log,you can see messages related to Model Merge,for example.

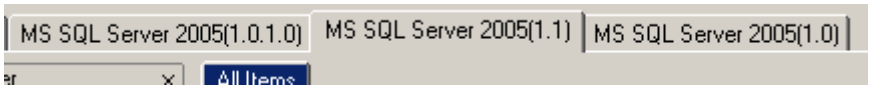
Generate Report (产生报告)- Generates HTML report wizard to generate alter report containing the changes between the two models.

Execute (执行)- Executes selected action(the same icon is used for Model Merge and Alter Script Generation features).

下面我们就一个具体的例子来演示一下如何产生两个差异数据库的升级脚本。

有一个数据库 A，存在两个版本，分别是 V1。0 和 V1。1，后者比前者相同表中多了某个字段 1，现在要在版本 v1。0 的基础上，产生升级脚本，首先我们要明确几点。：

1) Convertor 界面中，Model1 的对象是不能用户选择的，也就是说一打开 Convertor 界面，Model1 就是固定死了，是根据工作区中当前击活的数据库版本，如下图：



点击 Alter-Scripting，Model Merge 菜单或者按钮。

2) 我们要把 1.0 版本的数据库修改成 1.1 版本，那么就是要修改 Model2，所以选择 ，执行 Execute，产生升级脚本。系统会根据 Options-Path-General-File with Generated Script 的路径产生对应的脚本文件。

产生的脚本文件是 sql 格式，格式比较专业，用户可以直接在查询分析器中执行，或者也可以批量执行。

3) 对于升级脚本，我们还可以通过 Generate Report 产生升级的报告，作为历史记录。对于数据结果的修改记录，对于团队的开发效率是非常重要的。

4) Convertor 页面的功能范围只是针对表的结果变化，视图，存储过程，函数的变化，都不会产生差异脚本。(这个是我自己猜的，没有试过，有兴趣的 TX 可以自己尝试一下)

至此，四个功能讲解完毕，总的来说 TDM 是一个专业全面的数据库工具，虽然说第三方的数据库建模工具很多，有商业的 PD，ROSE，Visio，有开源的 ERDesigner NG，OpenSystemArchitect 等等，我也没一一用过，总的感觉 TDM 是个不错的数据库工具。好了今天就到这里，希望有更多的牛 X 公司提供更好的工具。

如何做好 SAP 自动化测试

作者：卢晨之

本文主要研究了如何使用脚本语言或主流自动化测试工具 QTP 在 SAP GUI 上展开自动化的自动化测试。探讨了它们的弊端与优势，还有如何结合。

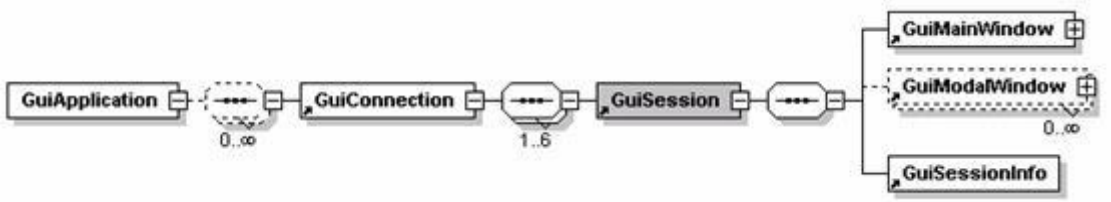
前言：

SAP GUI 编写的语言是 ABAP（Advanced Business Application Programming），一般情况下外部公司会购买 SAP 的部分模块并在原来 SAP GUI 的基础上做二次开发，所以大部分的 SAP 程序都是由原始版本放射出来的子版本，同样它也继承了 SAP 的大部分基层类、方法。

当然，如何对一个 GUI 程序做自动化测试，我们需要了解的还有很多，还包括了它的构成，类对象，方法等等。

了解 SAP GUI 的结构

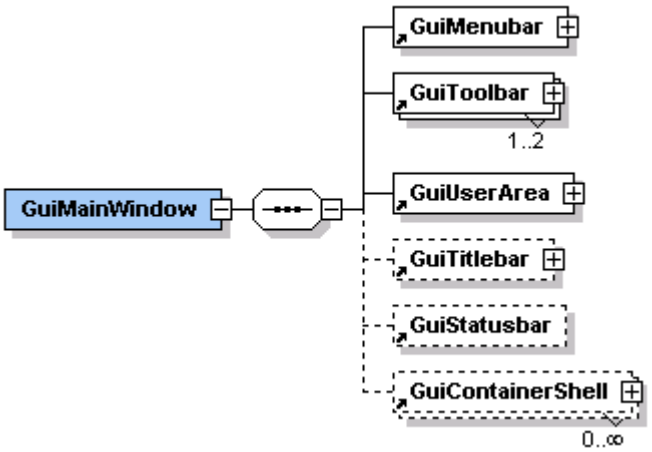
SAP GUI 主要分成 2 层，一层是后台控件（Top level administrative objects），另一层是前台控件（Top level user interface objects）。在后台控件中，存在着一个层次鲜明的关系，正如下图所示：



透过上面图片我们可以看到，后台控件的关系分别：GuiApplicaiton，GuiConnection，GuiSession；前台控件包括了：GuiMainWindow，GuiModalWindow，GuiSession 等等。而这些层次关系也是非常重要，因为我们需要靠它来得到我们需要操作的对象，下面是 VBS 脚本片段（GetSession.vbs）：

```
Dim SapGuiAuto
Dim Application
Dim connection
Dim session
If Not IsObject(application) Then
    Set SapGuiAuto  = GetObject("SAPGUI")
    Set application = SapGuiAuto.GetScriptingEngine
End If
If Not IsObject(connection) Then
    Set connection = application.Children(0)
End If
If Not IsObject(session) Then
    Set session    = connection.Children(0)
End If
If IsObject(WScript) Then
    WScript.ConnectObject session,     "on"
    WScript.ConnectObject application, "on"
End If
```

上面的代码片示例了获取 GuiApplicaiton，GuiConnection，GuiSession。了解完了后台控件的关系图后，我们需要了解多一个前台控件的关系，就是我们所见到的 SAP GUI，它们的关系如下图罗列的：



在SAP GUI中，一般情况下每个Session下面就只有一个GuiMainWindow，而在这个对象下面，就都是我们能看到的对象，包括了 GuiMenuBar，GuiToolBar，GuiUserArea 等等。如何获取到它们？后面我们将再做介绍。

SAP GUI 对象模型

SAP GUI 的对象模型是面向对象的设计模式，它主要包括了 4 大个基类：GuiComponent，GuiContainer，GuiVComponent,，GuiVContainer。大部分对象继承了其中的一个或者多个类，但有小部分的对象是没有继承的，例如 GuiScrollBar，GuiComboBoxEntry，GuiSessionInfo 等等。

纯脚本与 SAP GUI

有了前面对 SAP 了解的基础上，我们再使用 SAP 自带的 SAPGUIScripting API 为我们服务（SAPgui\SAPguihelp\SAPGUIScripting.chm），透过这个文档，我们能够获取到这些对象的方法，属性，以便展开我们的自动化测试工作。在实际自动化过程中，我们的操作主要都是针对对象，所以获取对象自然是我们一项非常重要的工作之一。因此，如何获取到 SAP GUI 的前台对象，也是我们需要学习的。

获取对象的几个主要方法：

1. findById 这是获取 SAP 对象中，一个比较重要和常用的方法，传递进去的参数是对象的 ID，而这里的 ID 我们也需要注意的是，这个对象的 ID 其实是一个路径，所以在不同父对象中使用 findById 时候，ID 会有所不同，就好比下列代码中，同样是获取 GuiToolBar，但 ID 却是不同。

```
Set SapGuiAuto  = GetObject("SAPGUI")
Set application = SapGuiAuto.GetScriptingEngine
Set connection = application.Children(0)
Set session     = connection.Children(0)
session.children(0). findById ("tbar[0]")
session. findById (“wnd[0]/tbar[0]”)
```

2. FindAllByName 与 FindAllByNameEx 这 2 个方法是通过传递进对象名称与类型来查找当前对象下面的子对象，并返回一个对象的集合 GuiComponentCollection。其中的 FindAllByName 与 FindAllByNameEx 的区别是传进去的对象类型名称或类型码（SAP 中每个对象类型都有自己对应的类型码，例如 GuiToolBar 是 101，GuiCheckBox 是 42），而这 2 个参数中，类型可以为空。

3. FindByName 与 FindByNameEx 区别于第 2 中的 2 个方法，第 3 的这 2 个方法是直接返回了符合条件第一个对象，是一个 GuiComponent 而非 GuiComponentCollection 。 如果通过代码去了解它， 就是 FindAllByName().Item(0) 是一样的。

4. Children 与 Parent 属性 无论是在 SAP 或者 Web 或者.net 与 VB，Children 与 Parent 这 2 个属性都是极为常见，它们获取到的分别是 GuiComponentCollection 与 GuiComponent，也就是一个集合与一个对象，我们同样可以通过它们去获取我们想要的对象并做操作。

打开 SAP 脚本录制开关

SAP 公司是一家非常人性化与具有远瞩的公司，它同时考虑到 SAP GUI 的自动化在项目中的设施，所以 SAP GUI 后台就已经提供了简单的脚本生成工具，就如同微软的宏录制一样。而这个脚本录制之前，我们需要把开关打开，并且设置好录制完毕后脚本的名字。

这个开关就放在了 GuiSession 下面，属性分别是 Record 与 FileRecord。它们的操作顺序是先把 FileRecord 设置好(这里只需要设置文件名字，路径是默认为 Documents and Settings\XXXX\SapWorkDir)，再把 Record 设置为 1 并开始录制，到最后再把开关关闭。

TestRecord.Vbs：

```
Set SapGuiAuto  = GetObject("SAPGUI")
Set application = SapGuiAuto.GetScriptingEngine
Set connection = application.Children(0)
Set session      = connection.Children(0)
session.Recordfile="LuchenzhiTest.Vbs"
session.Record="1"   '关闭需要设置为 0
```

录制与修改脚本

录制完毕后，我们把开关关闭，在文件夹下可以看到如下代码片(LuchenzhiTest.Vbs)：

```
session.findById("wnd[0]").maximize
session.findById("wnd[0]/tbar[0]/okcd").Text = "Test"
WScript.Sleep 100
session.findById("wnd[0]").sendVKey 0
session.findById("wnd[0]/usr/ctx").Text = “LuchenzhiTest”
WScript.Sleep 100
session.findById("wnd[0]/usr/ctxt").caretPosition = 14
session.findById("wnd[0]/tbar[1]/btn[8]").press
session.findById("wnd[1]/usr/sub/ctxtZ").Text = "2"
WScript.Sleep 100
session.findById("wnd[1]/usr/sub/ctxtZ").SetFocus
session.findById("wnd[1]/usr/sub/ctxtZ").caretPosition = 1
WScript.Sleep 100
session.findById("wnd[1]/tbar[0]/btn[0]").press
```

但透过上面录制的方法，我们会看到我们的操作都被转化成为 VBS 的脚本，它的优点就是生成速度快，文件小，运行简单快捷。但它存在着下面的几点缺点：

1. 缺乏稳定性。这点是所有脚本与自动化测试所需要面临的。我们需要更多的判断处理，事务等待，对象查找操作等等。

2. 方法单一，维护量大。生成的脚本中，都是使用了 FindById 的方法去查找对象，而会导致如果程序的对象发生变化，或者 ID 发生变化，都需要人员维护与修改。自然，我们可以结合上文中提到的其他方法去获取对象，来确保脚本运行顺畅。

但尽管如此，缺乏良好的对象管理模式是非常吃力的。

3. VBS 缺乏强大的 IDE 支持。在没有很强大 IDE 支持情况下，脚本中途出错，维护与 Debug 都有可能让脚本重头再跑，让效率大大降低。

4. 需要跟多人力去建立庞大的方法库。如前面几点所述，我们为了确保它的稳定，还需要另外编写更多的方法去支持它，例如事务等待，对象响应，错误处理，检查点等等。这些都是需要很多的资源投入，包括了前期搭建与后期维护。

QTP 在 SAP GUI 中的应用

上文大概介绍了 VBS 纯脚本语言如何在 SAP GUI 自动化领域中的应用，同样我们发现了很多它的缺点与弊端。但我们不能够否认它的作用，当它与 QTP 结合的时候，它们就能互补彼此的缺点。

QTP SAP solutions add-in

QTP 的 SAP solutions add-in 对 SAP 的控件支持，算得上是理想。而这个理想的主要原因是 SAP 的控件比较单一，不像.Net 的第三方控件如此之多。我们使用这个插件基本能满足我们在业务中的大部分操作与自动化的实现。而 QTP 的另一个优势就是它的对象库，这个能很好的管理我们的对象与维护，同时省去了我们非常多的代码量，区别于纯 VBS 脚本。

可是不幸的是，当我们去用这个控件时候，我们会发现，QTP 封装这块原来做得非常粗糙，划分得太大，而不是细。就如同我们用 QTP 能够识别了五大洲但却识别不了里面的国家，而我们要操作国家，只能和这些大洲的洲长去联系，告诉它我们现在要让你们某个国家干什么，给，这是这个国家的 ID 和它要干的事情。

调用 SAP GUI 对象的方法与属性

我们能够使用.Object 去调用 SAP GUI 对象的自身方法，而 QTP 做的就是让我们更快的找到这个对象并帮我们管理好它。所以当我们使用这个组合的时候，有很多的问题我们都能迎刃而解。

实例 1：SAP solutions add-in 中并没有 MouseMove 或者 Click(x,y)的方法，而我们通过把 SAP 转化为 Window 对象的方法来实现。

```
l=Sapguisession("Session").SAPGuiWindow("SAP").SAPGuiButton("Log").Object.left
h=Sapguisession("Session").SAPGuiWindow("SAP").SAPGuiButton("Log").Object.Top
hwnd=sapguisession("Session").SAPGuiWindow("SAP").Object.handle
window("hwnd:="&hwnd).MouseMove l,h
```

实例 2：SAP solutions add-in 中对 SapGuiTable 的封装得并不细，我们能操作的都只是点击某个 Cell 与设置，或者控制选中某个 Row，Column。但现在我们能够获取到 Cell，或者对列宽度设置，或者滚轮滚动等等做操作。在 SAP 中，GuiTable 是由 Rows 与 Columns 交叉组成的，因此需要获取某个 Cell 我们就有 2 种方法获取到，Rows.item(1).item(0)与 Columns.item(0).item(1)是一样的。

```
Set Cell=Sapguisession("Session").SAPGuiWindow("Change").SAPGuiTable("All
").Object.columns.item(1).item(0)  '获取了第 2 列第 1 行 的 Cell
Sapguisession("Session").SAPGuiWindow("Change").SAPGuiTable("All").Object.column
s.item(1).width=100                '设置第 2 列的宽度
sapguisession("Session").SAPGuiWindow("Change").SAPGuiTable("All").Object.Vertical
Scrollbar.Position=1               '我们可以通过循环递增的方法设置它的 Position 来
实现滚动的效果，模拟鼠标拉动滚轮。
```

实例 3：QTP 封装的 SAPGUITabled 中，SelectRow 传递进去的参数是一个整数，而选择的的就是 Table 中所有数据（包括当前没罗列在 Table 中）的第几行。所以当我们翻页时候，我们要选择当前页面的第一条数据却显得力不从心，可是如果我们这么写呢：

```
Sapguisession("Session").SAPGuiWindow("Change").SAPGuiTable("All").Object.Rows.Item(0).Selected=True
```

所以当我们这写的时候，选择的就是罗列在当前 Table 中，可见数据的第一条，而不是第一页的第一条。

实例 4：如果你已经读到了前文中的一句话“一般情况下每个 Session 下面就只有一个 GuiMainWindow”，你就知道，这样的理论知识有时候是能够让你在实际应用中发挥它的用途。在 SAP GUI 中，弹跳窗口一直是一个捉摸不定，不可预期的对象，而我们会发现，在 Session 的属性中，有一个 ActiveWindow 的属性，它返回的就是当前活动窗口的类型。所以只要我们判断属性不是 GuiMainWindow 而是其他类型的窗口，我们可以归纳为信息或者错误提示窗口等等并做相应处理。

透过上面的 4 个实例，可以让我们更深刻的了解 QTP 与 SAP GUI 对象自身方法，属性结合后的好处与用途，这也会让你不再因为 QTP 对 SAP GUI 支持不好而感觉力不从心。

总结

上文介绍了 VBS 或者 QTP 在 SAP GUI 自动化领域的应用，包括了对象操作，对象获取等等，以及 QTP 在 SAP GUI 的高级应用。在纯脚本的 VBS 中，存在的问题，到了 QTP 应用中同样存在 2 个弊端，其中最主要的就是事务等待的处理与异常处理。这个是做好 SAP 自动化测试的一个瓶颈，也是一个技术关卡。需要读者们更多的积累与实践，才能把这方面处理好，处理得稳妥，否则是在实际项目中，SAP 的自动化很难做到无人看守状态。

如果您在实际项目中遇到问题，可以和作者共同探讨与研究。（MSN：luchenzhi@hotmali.com）