**QTP自动化测试实践**

**（一）**

**第8章 数据驱动**[**测试**](javascript:;)

        测试脚本的开发和维护是[**自动化测试**](javascript:;)的重要环节，适当地调整和增强测试脚本，能提高测试脚本的灵活性，增加测试覆盖面，以及提高应对测试对象变更的能力。数据驱动方式的测试脚本开发是解决这类问题的重要手段。  
        本章介绍如何在自动化测试过程中使用数据驱动的测试脚本开发方式，对测试脚本进行参数化，包括如何使用[**QTP**](javascript:;)的Data Table参数化、Action参数化、环境变量参数化等脚本参数化的方法。

8.1  数据驱动测试方法  
        数据驱动的测试方法要解决的核心问题是把数据从测试脚本中分离出来，从而实现测试脚本的参数化。  
8.1.1  什么时候使用数据驱动测试方法  
        自动化测试对录制和编辑好的测试步骤进行回放，这种是线性的自动化测试方式，其缺点是明显的，就是其测试覆盖面比较低。测试回放的只是录制时做出的界面操作，以及输入的测试数据，或者是脚本编辑时指定的界面操作和测试数据。  
        如何让测试脚本执行时，不仅仅局限于测试录制或编辑时的测试数据呢？数据驱动的测试方式是解决这个问题的最佳方案。数据驱动测试把测试脚本中的测试数据提取出来，存储到外部文件或[**数据库**](javascript:;)中，在测试过程中，从文件动态读入测试数据。  
        注意：如果希望测试的覆盖面更广，或者让测试脚本能适应不同的变化情况，则需要进行测试脚本的参数化，采用数据驱动的测试脚本开发方式。

8.1.2  数据驱动测试的一般步骤  
        通常，数据驱动测试按以下步骤进行：  
        （1）参数化测试步骤的数据，绑定到数据表格中的某个字段。  
        （2）编辑数据表格，在表格中编辑多行测试数据（取决于[**测试用例**](javascript:;)以及测试覆盖率的需要）。  
        （3）设置迭代次数，选择数据行，运行测试脚本每次迭代从中选择一行数据。  
        QTP提供了一些功能特性，让这些步骤的实现过程得以简化。例如，使用“Data Table”视图来编辑和存储参数，如图8.1所示。

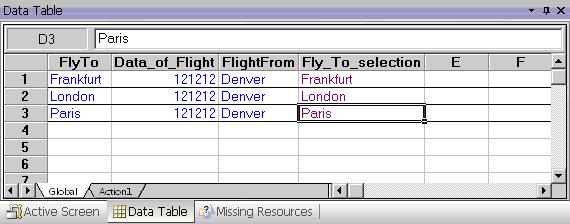


图8.1  Data Table视图  
        另外，还提供“Data Driver向导”，用于协助测试员快速查找和定位需要进行参数化的对象，并使用向导进行一步一步的参数化过程。

（二）

8.2  参数化[**测试**](javascript:;)  
        在[**QTP**](javascript:;)中，可以通过把测试脚本中固定的值替换成参数的方式来扩展测试脚本，这个过程也叫参数化测试，能有效地提高测试的灵活性。  
8.2.1  通过参数化测试来提高测试的灵活性  
        可以通过参数化的方式，从外部数据源或数据产生器读取测试数据，从而扩大测试的覆盖面，提高测试的灵活性。在QTP中，可以使用多种方式来对测试脚本进行参数化，数据表参数化（Data Table Parameters）是其中一种重要的方式，还有环境变量参数化（Environment Variable Parameters）、随机数参数化（Random Number Parameters）等。  
        下面以QTP自带的“Flight”程序为例，介绍如何对测试脚本进行参数化。假设在名为“Flight Reservation”的订票界面中，输入航班信息后，插入订票记录，然后，希望重新打开该记录，检查航班信息中的终点的设置是否正确，录制的测试脚本如图8.2所示。



图8.2  录制的测试脚本  
        提示：对于这样一个测试脚本，仅能检查特定的航班订票记录的正确性，如果希望测试脚本对多个航班订票记录的正确性都能检查，则需要进行必要的参数化。

8.2.2  参数化测试步骤  
        首先，把测试步骤中的输入数据进行参数化，例如航班日期、航班始点和终点等信息。下面，以“输入终点”的测试步骤的参数化过程为例，介绍如何在关键字视图中对测试脚本进行参数化。  
（1）选择“Fly To :”所在的测试步骤行，单击“Value”列所在的单元格，如图8.3所示。



图8.3  设置参数值  
        （2）单击单元格旁边的“<#>”按钮，或按快捷键“CTRL+F11”，则出现如图8.4所示的界面。

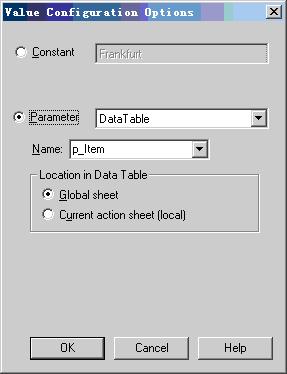


图8.4  选择参数从Data Table读取  
        提示：在这个界面中，选择“Parameter”，在旁边的下拉框中选择“Data Table”，在“Name”中输入参数名，也可接受默认名，在“Location in Data Table”中可以选择“Global sheet”，也可以选择“Current action sheet（local）”，它们的区别是参数存储的位置不同。

（3）单击“OK”按钮，在关键字视图中可看到，“Value”值已经被参数化，替换成了如图8.5所示。



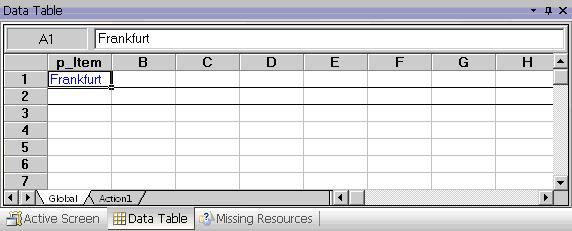
图8.5  参数化后的值  
（4）这时，选择菜单“View | Data Table”，则可看到如图8.6所示的界面。  


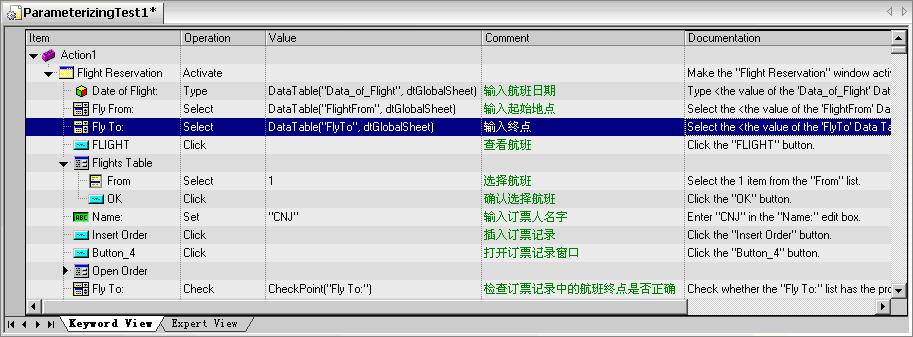
图8.6  Data Table中的参数数据  
        可看到，在“p\_Item”列中有一个默认数据“Frankfurt”，这是参数化之前录制的脚本中的常量，可以在“p\_Item”列中继续添加更多的测试数据。  
提示：可以双击修改“p\_Item”列名，让其可读性更强，例如，改成“FlyTo”。  
（5）把其他几个数据也参数化后，如图8.7所示。  


图8.7  参数化后的测试步骤  
        QTP运行时，就会从如图8.8所示的数据表格中提取数据来对测试过程中的各项输入进行参数化。

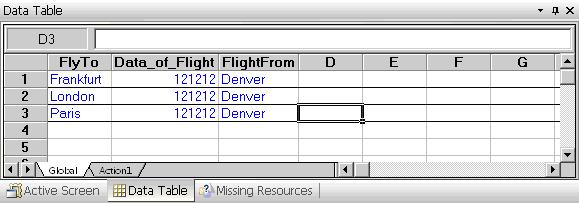


图8.8  Data Table存储的参数值

（三）

8.2.3  使用随机数来进行参数化  
        对于选择航班这个[**测试**](javascript:;)步骤的参数化来说会有所不同，因为航班会跟随所选择的起点和终点而变化，因此，需要做特殊的处理。如下代码所示：  
' 取得航班列表的行数  
ItemCount = Window("Flight Reservation").Dialog("Flights Table").WinList("From").GetItemsCount  
' 随机选取其中一项  
SelectItem = RandomNumber(0, ItemCount)  
' 选择航班  
Window("Flight Reservation").Dialog("Flights Table").WinList("From").Select SelectItem  
        先通过访问GetItemsCount属性，获取航班列表的行数，然后使用RandomNumber随机选取其中一项，最后，再通过Select方法选择航班。参数化后的测试步骤如图8.9所示。

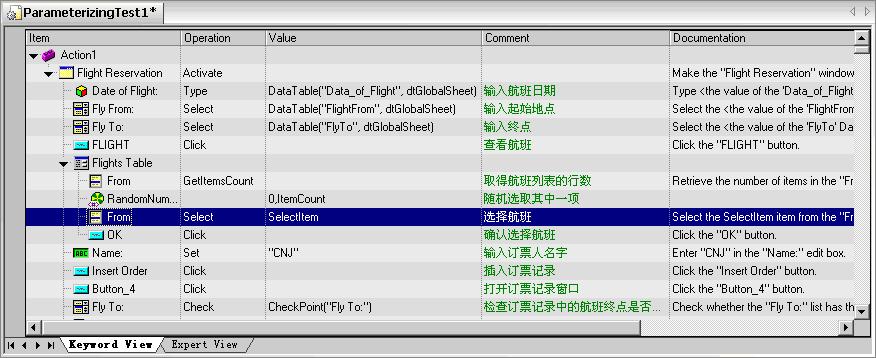


图8.9  参数化后的测试步骤

        提示：使用随机数也是测试脚本参数化的一种重要方法，在[**QTP**](javascript:;)的测试代码中，可用RandomNumber来实现，在关键字视图编辑的界面如图8.10所示，其效果与在脚本中直接编辑是一样的。

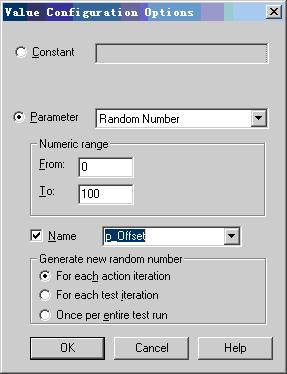


图8.10  选择参数化方式为“Random Number”  
8.2.4  参数化检查点  
        测试脚本的最后一个测试步骤是检查订票记录中的航班终点是否正确，同样需要进行适当的参数化，方法如下：  
（1）单击检查点所在测试步骤的“Value”列中的单元格，如图8.11所示。

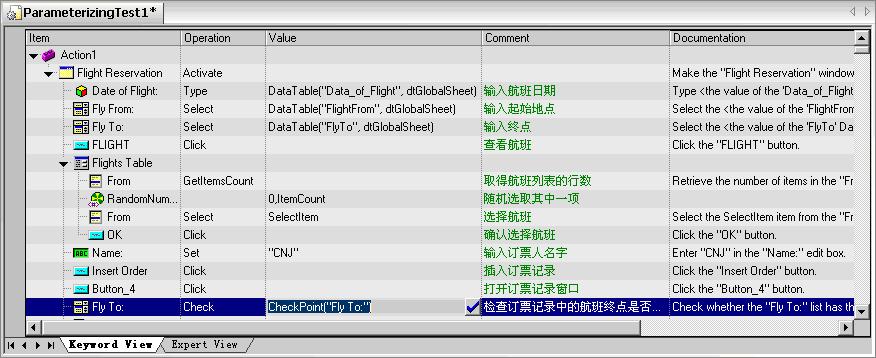


图8.11  设置检查点参数  
（2）单击旁边的按钮，则出现如图8.12所示的界面。

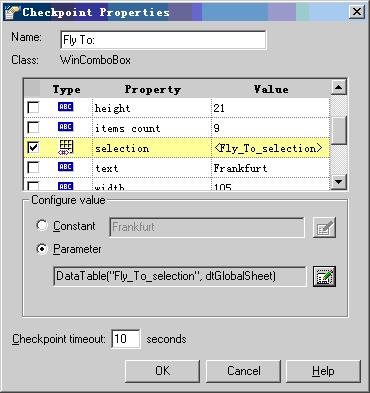


图8.12  检查点属性编辑界面  
（3）在“Configure value”中选择“Parameter”后，可单击“OK”按钮接收默认的设置，也可单击旁边的编辑按钮，在如图8.13所示的界面中，进行参数化的详细设置。

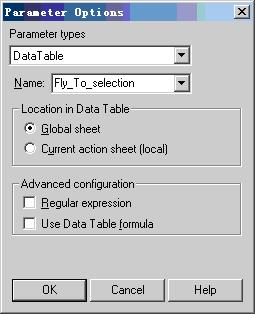


图8.13  参数化的详细设置  
        在“Parameter types”中，选择“Data Table”；可在“Name”修改参数名，或接受默认的命名，产生如图8.14所示的数据列，也可以选择“FlyTo”，因为检查点所指的航班终点得到的预期值应该与测试步骤中选择航班终点时的输入数据一致，否则认为错误。

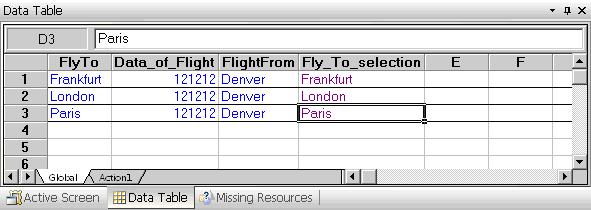


图8.14  Data Table中存储的参数值

（四）

8.2.5  设置数据表格迭代方式  
        把[**测试**](javascript:;)步骤和检查点的参数化[**工作**](javascript:;)都完成后，可得到如图8.15所示的测试步骤。

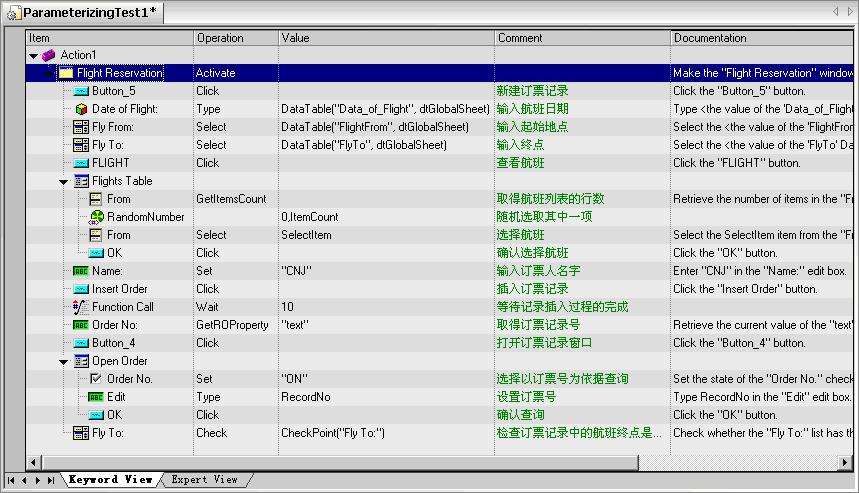


图8.15  参数化后的测试步骤  
        切换到专家视图，可看到如图8.16所示的测试脚本。

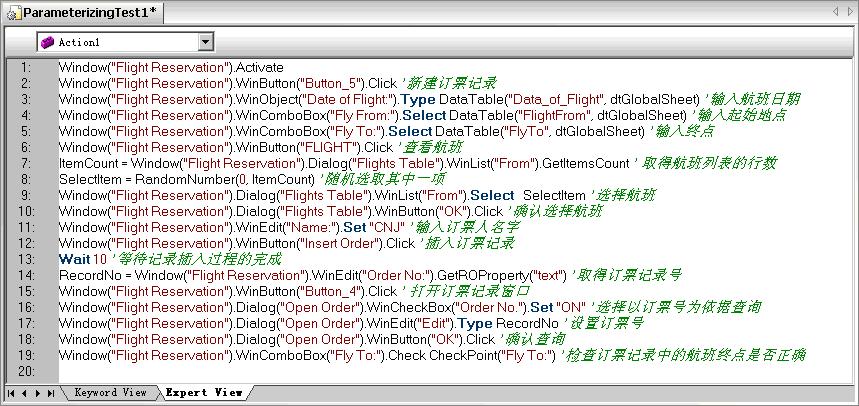


图8.16  参数化后的测试脚本

8.2.5  设置数据表格迭代方式  
        把[**测试**](javascript:;)步骤和检查点的参数化[**工作**](javascript:;)都完成后，可得到如图8.15所示的测试步骤。

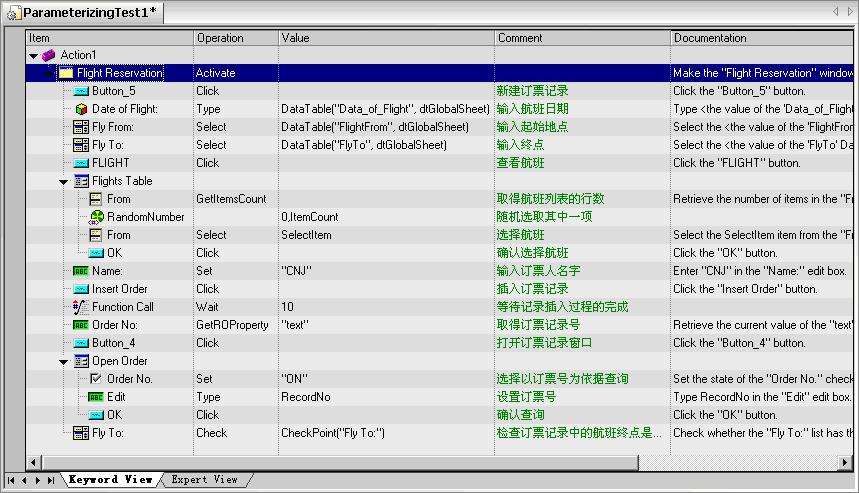


图8.15  参数化后的测试步骤  
        切换到专家视图，可看到如图8.16所示的测试脚本。

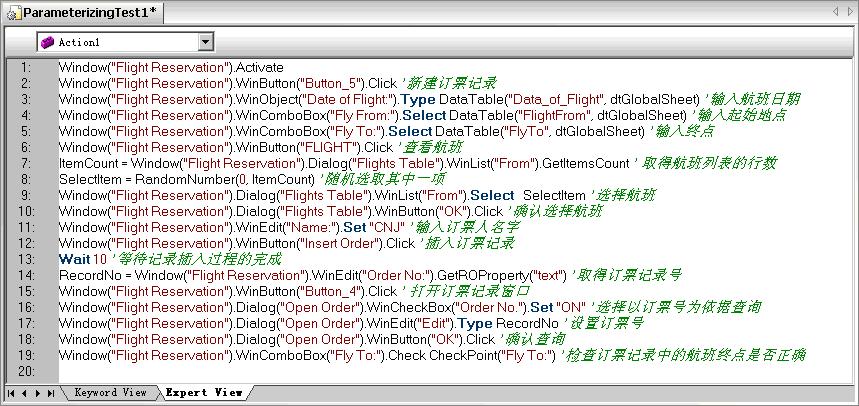


图8.16  参数化后的测试脚本

（五）

8.3  Action测试输入的参数化  
        对于重复使用的[**测试**](javascript:;)用例，可以转换成公共用例，适当参数化后，可被[**其他**](javascript:;)测试用例调用。在[**QTP**](javascript:;)中，可以把Action的输入适当参数化，转换成可重用的测试步骤。  
8.3.1  编辑Action的属性  
        QTP的“Flight”程序中的登录模块的测试步骤是在执行其他测试步骤之前都要经过的测试步骤，因此有“潜力”成为一个可重用的Action。对于如图8.19所示的测试步骤，可以进一步参数化后，成为可重用的测试步骤，被其他Action调用。



图8.19  可重用的测试步骤  
        选择“Action1”所在的行，然后单击鼠标右键，选择菜单“Action Properties”，则出现如图8.20所示的界面。

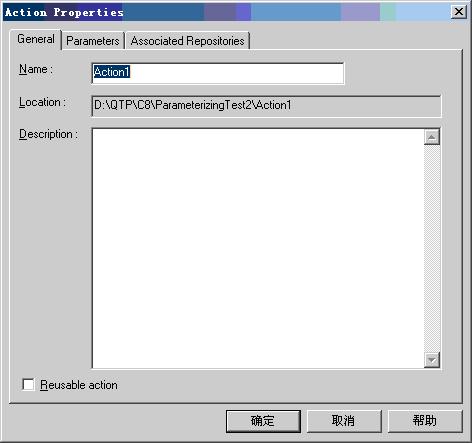


图8.20  Action属性编辑窗口  
        在“Name”中输入新的Action名称，例如“Login\_Action”，在“Description”中输入对Action的描述信息，如图8.21所示。

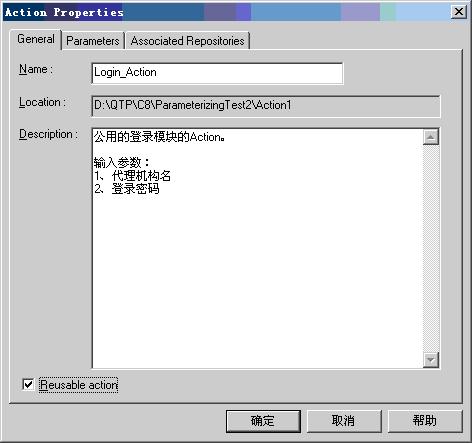


图8.21  编辑Action属性  
        注意：把“Reusable action”勾选上，表示该Action为可重用的测试步骤，是可被其他Action调用的测试步骤。

8.3.2  添加Action的输入参数  
        切换到“Parameters”页，如图8.22所示。单击“+”按钮，添加调用Action需要输入的参数名和类型。

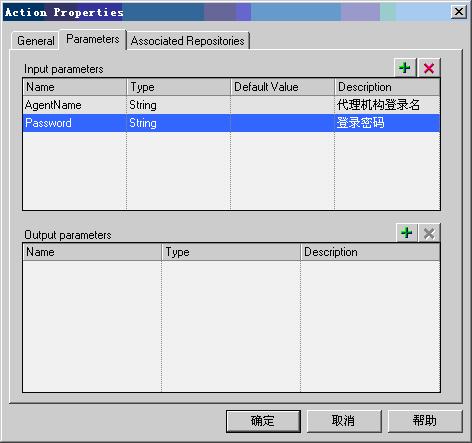


图8.22  添加输入参数  
        提示：在这里，“Login\_Action”需要两个参数，其中，“AgentName”表示代理机构登录名，“Password”表示登录密码。  
        添加完参数后，回到关键字视图，选择“输入代理机构名”所在的测试步骤，单击“Value”列的单元格旁边的“<#>”按钮，出现如图8.23所示的界面。

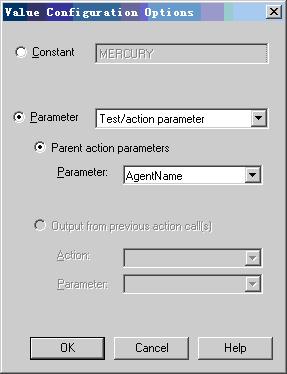


图8.23  设置参数  
        在“Parameter”中，选择“Test/action parameter”，然后，选择刚才编辑好的参数“AgentName”，单击“OK”按钮。重复这个步骤，为“输入登录密码”的测试步骤设置参数，得到如图8.24所示的测试步骤。



图8.24  完成测试步骤

8.3.3  调用Action  
        完成Login\_Action的参数化后，就可以在其他Action中调用这个Action，方法是在Action的测试步骤中，选择菜单“Insert | Call To Existing Action”插入现有的Action，如图8.25所示。



图8.25  选择Action  
        在这个界面的“From test” 中选择“<Current Test>”，在“Action”中选择“Login\_Action”，单击“OK”按钮后，即可插入对“Login\_Action”测试步骤的引用，如图8.26所示。



图8.26  插入对Action的引用  
        选中“Login\_Action”所在的行，单击鼠标右键，选择菜单“Action Call Properties”，出现如图8.27所示的界面。

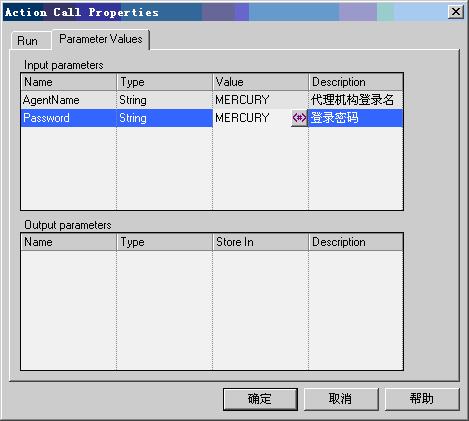


图8.27  设置参数值  
        在“Parameter Value”页中，为每一个参数设置输入的参数值，也可以单击“Value”列旁边的“<#>”按钮，为输入绑定到Data Table中的数据。单击“确定”后，可在专家视图看到如图8.28所示的测试代码：

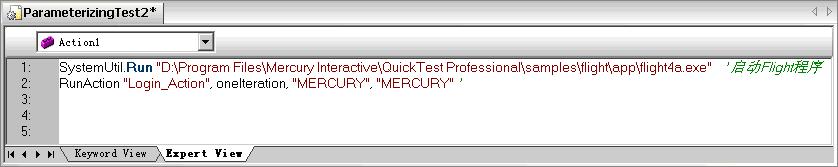


图8.28  使用RunAction方法来调用Action  
        该测试代码使用了RunAction方法来调用“Login\_Action”，输入的第一个参数值和第二个参数值都为“MERCURY”。测试脚本的运行结果如图8.29所示，可看到，“Login\_Action”被成功地调用，测试结果中也列出了调用“Login\_Action”所输入的参数值。

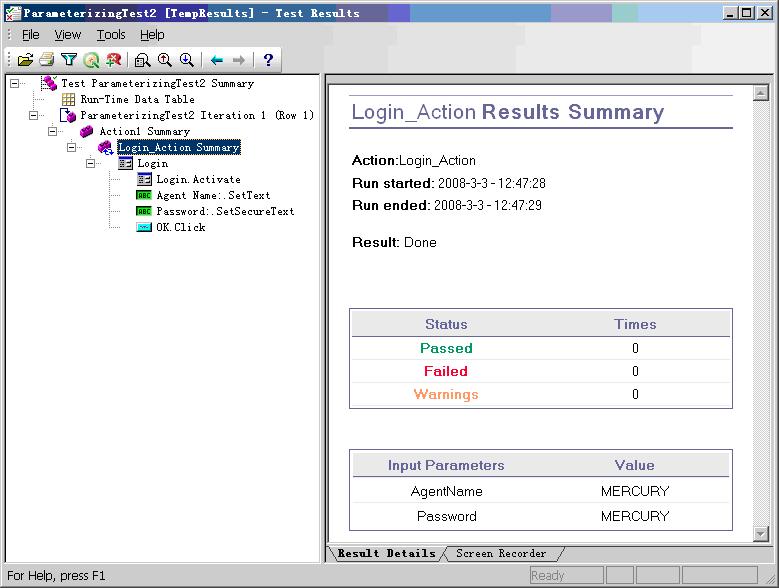


图8.29  测试结果

（六）

8.４  使用环境变量的参数化  
        在[**QTP**](javascript:;)中，除了前面所讲的几种参数化[**测试**](javascript:;)的方式外，还可以使用环境变量来进行测试的参数化。下面介绍如何使用环境变量来参数化如图8.30所示的测试步骤，将其中的“Agent Name”和“Password”的值从定义好的环境变量读入。

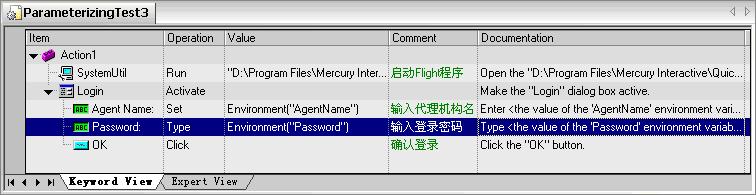


图8.30  待参数化的测试步骤  
8.4.1  定义和设置环境变量  
        在使用环境变量之前，需要定义好环境变量，方法如下：  
（1）选择菜单“File | Settings”，出现如图8.31所示的界面。

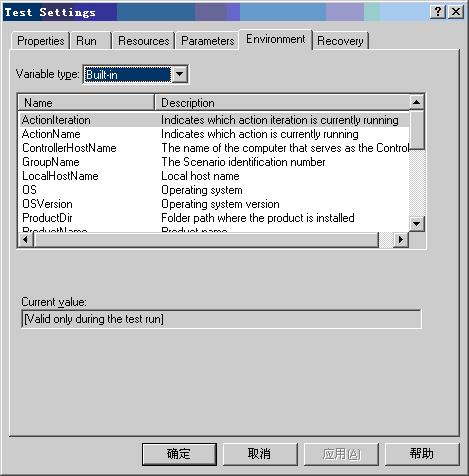


图8.31  环境变量设置  
        （2）在这个界面中，切换到“Environment”页，在“Variable type”中选择“User-defined”，然后，单击旁边的“+”按钮，在如图8.32所示的界面中，定义环境变量名和输入的值。



图8.32  添加环境变量  
（3）重复这个步骤，定义“Password”的环境变量，得到如图8.33所示的结果。

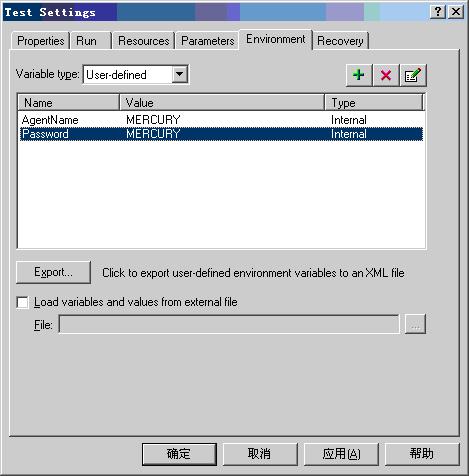


图8.33  成功添加自定义环境变量  
8.4.2  在测试步骤中绑定环境变量值  
        定义好环境变量并设置好其值之后，就可以在测试步骤中使用该环境变量。方法如下：  
（1）在关键字视图中，定位到测试步骤的“Value”列，如图8.34所示。

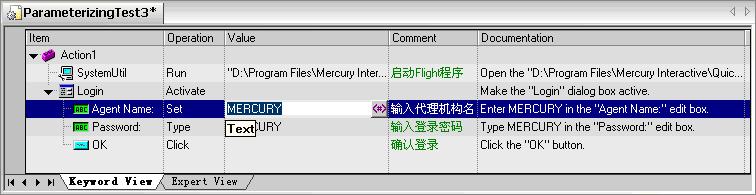


图8.34  定位到测试步骤的“Value”列  
（2）单击旁边的“<#>”按钮，出现如图8.35所示的界面。

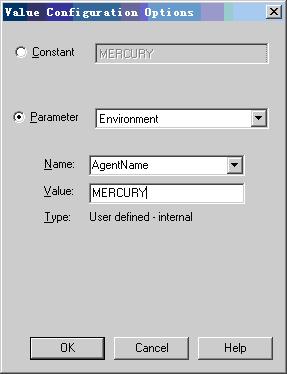


图8.35  参数化选定的值  
        提示：在界面中，选择“Parameter”，并在下拉框中选择“Environment”，在“Name”中选择“AgentName”，在“Value”中输入对应的值。  
（3）重复这个步骤，设置“输入登录密码”的测试步骤所对应的环境变量，如图8.36所示。

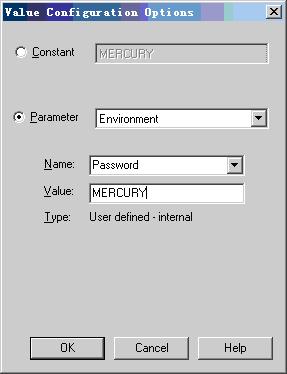


图8.36  设置“输入登录密码”的测试步骤所对应的环境变量  
        设置完后，可得到如图8.37所示的测试步骤。



图8.37  参数化后的测试步骤  
        这样，QTP在运行测试脚本时，就会读取测试步骤所绑定的环境变量值，来执行相应的数据输入动作。

（七）

8.4.3  导出环境变量到XML文件  
        选择菜单“File | Settings”，在如图所示的界面中，单击“Export”按钮，可把当前定义的环境变量导出到XML文件中，如图8.38所示。

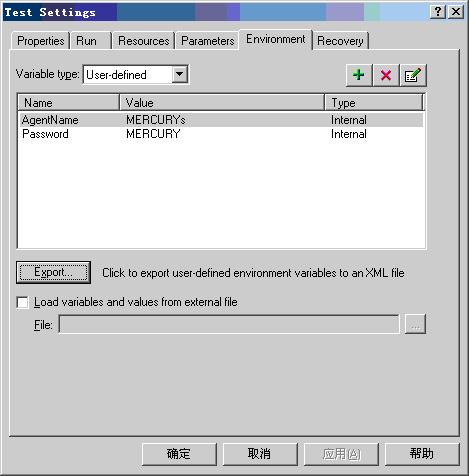


图8.38  导出环境变量  
        导出后的XML文件如图8.39所示。可看到，<Variable>和</Variable>之间是一个个定义好的环境变量，包括变量名和变量值。



图8.39  导出的XML文件  
8.4.4  导入外部环境变量文件  
对于导出的XML文件，可以再次导入，如图8.40所示。

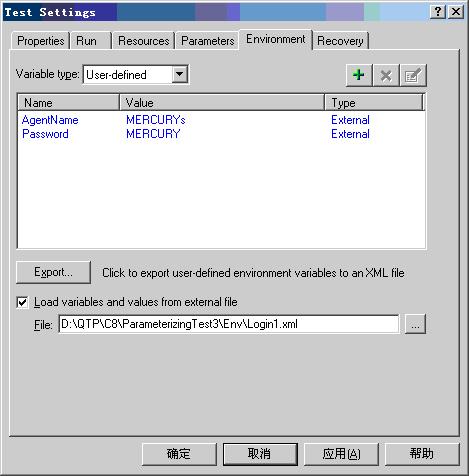


图8.40  导入环境变量  
        也可以在[**测试**](javascript:;)脚本中编写代码来加载，例如，下面的脚本在执行界面的测试步骤之前，先加载D:\QTP\C8\ParameterizingTest3\Env目录中的某个环境变量文件。  
SystemUtil.Run "D:\Program Files\Mercury Interactive\QuickTest Professional\samples\flight\app\flight4a.exe" ' 启动Flight程序  
Environment.LoadFromFile("D:\QTP\C8\ParameterizingTest3\Env\Login2.xml") '加载外部环境变量文件  
Dialog("Login").Activate  
Dialog("Login").WinEdit("Agent Name:").Set Environment("AgentName") ' 输入代理机构名  
Dialog("Login").WinEdit("Password:").Type Environment("Password") ' 输入登录密码  
Dialog("Login").WinButton("OK").Click ' 确认登录

（八）

8.5  使用数据驱动器来参数化[**测试**](javascript:;)  
        为了简化测试脚本参数化的过程，[**QTP**](javascript:;)还提供了名为“Data Driver”的功能，可自动检测脚本中可能需要进行参数化的变量。  
8.5.1  数据驱动器的使用方法  
        “Data Driver”可以帮助测试人员快速找到需要参数化的测试对象、检查点的数据。例如，对于如图8.41所示的录制脚本，选择菜单“Tools | Data Driver”，出现如图8.42所示的界面。

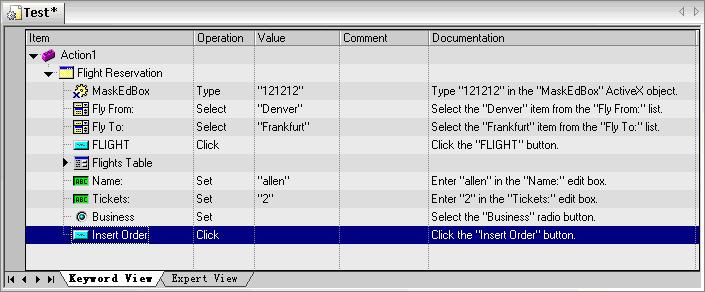


图8.41 待参数化的测试步骤

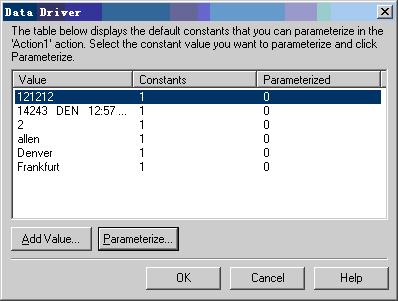


图8.42  数据驱动器  
        在这个界面中，列出了测试步骤中所有可能需要进行参数化的变量。

8.5.2  数据驱动向导  
单击“Parameterize”按钮，出现如图8.43所示的数据驱动向导。



图8.43  数据驱动向导  
单击“下一步”按钮，则出现如图8.44所示的界面。

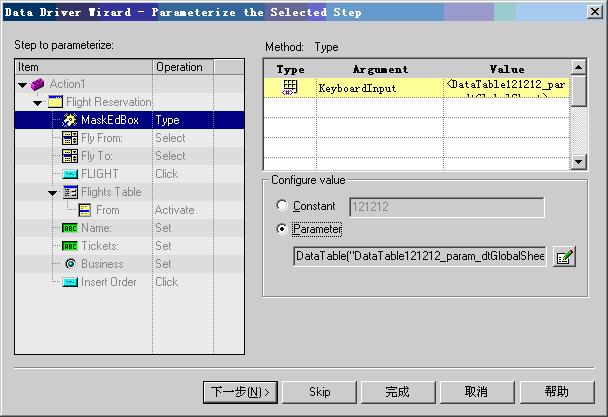


图8.44  参数化选定的测试步骤  
在这个界面中的左边窗口，定位到测试步骤所操作的界面控件，在右边显示参数化的名称和数据，单击“编辑”按钮，可在如图8.45所示的界面中进一步设置参数。

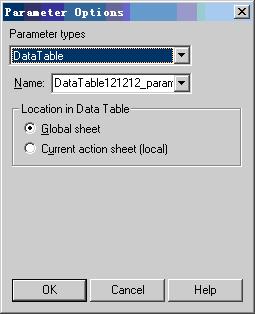


图8.45  参数化设置  
单击“OK”按钮，回到向导界面，单击“下一步”按钮，则出现如图8.46所示的界面，表明测试步骤的参数设置完成。其他测试步骤也可按类似的方式一步步地完成参数化。



图8.46  完成参数化

（九）

第3章软件[**自动化测试**](javascript:;)工具

        软件自动化测试工具是实现软件自动化测试必不可少的关键，因此，选择一个优秀的、适合自己的测试项目实际情况的测试工具是实现成功自动化测试的第一步。本章介绍自动化测试工具的分类，以及如何选择一个合适的自动化测试工具，并且介绍自动化测试工具的基本原理。  
3.1  自动化测试工具类型  
        测试工具的种类很多，有用于管理测试的，有帮助实现测试自动化的，有开源的，有免费共享的。[**软件测试**](javascript:;)工具按照其用途，可大致分成以下几大类：  
 测试管理工具θ  
 自动化θ[**功能测试**](javascript:;)工具  
 性能测试工具θ  
 单元测试工具。θ  
 白盒测试工具。θ  
 测试用例设计工具。θ  
        如果按测试工具的收费方式，又可分为以下几类。  
 商业测试工具。θ  
 开源测试工具。θ  
 免费测试工具。θ  
3.1.1  商业测试工具  
        商业测试工具的特点是需要花钱购买，但是会相对成熟和稳定，并且有一定的售后服务和技术支持。但是，由于其价格昂贵，并不是每一个企业都能负担得起。  
        商业测试工具主要集中在GUI功能测试和[**性能测试**](javascript:;)方面，目前流行的基于GUI的功能自动化测试工具有Robot、[**QTP**](javascript:;)、TestComplete等。各种自动化测试工具实现的功能基本相同，但是在IDE、脚本开发语言、支持的脚本开发方式、支持的控件等方面则有很多不同之处。  
3.1.2  开源测试工具  
        开源软件是指软件的源代码是公开发布的，通常是由自愿者开发和维护的软件。开源测试工具是测试工具的一个重要分支。越来越多的软件企业开始使用开源测试工具。但是开源并不意味着完全的免费，开源测试工具同样需要考虑使用的成本，并且在某些方面可能要比商业测试工具的成本还要高。  
        商业工具的价格在不断地提高。图3.1为WinRunner近几年的价格变化图。

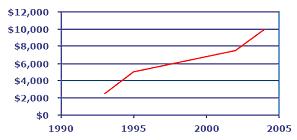


图3.1  WinRunner近几年的价格变化  
        可以看到，价格在不断地增长。这对于那些中小型软件企业而言，无疑加大了测试的成本。开源测试工具相对于商业测试工具拥有以下优势：  
 相对低的成本：大部分开源测试工具可免费使用，只要不做商业用途即可。θ  
 更大的选择余地：可以打破商业测试工具的垄断地位，给测试人员更多的选择空间。θ  
 可自己改造：源代码开放，意味着可对其进行修改、补充和完善，可对其进行个性化改造。θ  
        虽然开源测试工具拥有一定的优势，但是，同时也存在很多不足之处，包括以下几方面。  
 安装和部署相对困难：大部分开源测试工具的安装配置过程比较烦琐，需要测试人员付出一定的努力。θ  
 易用性：开源测试工具在易用性、用户体验方面做得不够完善。θ  
 稳定性：部分开源测试工具的稳定性不够强。θ  
 学习和获取技术支持的难度：大部分开源测试工具不提供培训指导和技术支持服务，联机帮助和用户手册不够完善，增加了测试人员的θ[**学习**](javascript:;)难度。  
3.1.3  自主开发测试工具  
        目前，很多软件测试组织其实已经具备了自己动手开发测试工具的条件：  
 市场对于测试工具的接受程度在不断提高，人们对测试工具的认识不断加强和深入，对测试工具原理的理解不断提高。从脚本化到数据驱动，再到关键字驱动等，很多新的测试工具理念被引入并被广泛接受。θ  
 由于技术的成熟，测试工具变得容易构建。软件系统现在变得更容易测试，可测试性更强，COM、XML、HTTP、HTML等标准化的接口使得测试更加容易进行。托管程序（例如θ[**Java**](javascript:;)、.NET）的反射机制使得查找定位对象，以及捕捉对象和操作对象更加容易。  
         一些开源的框架可以被利用。利用开源框架平台来组合、搭建适合自己测试项目使用的测试平台和测试框架。  
        自己动手开发测试工具的优势有以下方面。  
 购买成本为零。θ  
 简便：只需要开发自己需要的那部分功能。θ  
 个性化：可自己定制需要的功能，随时修改，配置项目组成员的使用习惯。θ  
 可扩展性：可随时增加新的功能。θ  
 可充分利用项目组熟悉的语言开发，利用自己的技术优势。θ  
 可使用自己熟悉的脚本语言，不需要使用商业工具提供的“厂商脚本语言”。θ  
        然而，虽然自己动手设计和开发测试工具有很多好处，但是必须考虑随之而来的成本问题。自己开发测试工具的成本只是开发时间和人员投入的成本，以及维护的成本。当然，如果把测试工具推广到[**其他**](javascript:;)项目组，则也会有学习和培训成本。另外，需要考虑测试工具的实用性，不要做一个大而全的、面面俱到的、很多功能基本上不会被用到的测试工具。

（十）

3.2  [**自动化测试**](javascript:;)工具选型  
        为了保证在一个测试团队中成功地应用某款测试工具，尤其是对于大型商业工具的应用，应该首先进行工具的选型，通过分析实际情况，确定选用范围。对选用范围内的几款测试工具进行试用。根据试用的反馈效果决定最终采用哪款测试工具。在大规模使用工具之前，还应该对测试人员进行全面的工具培训。培训后，正式在项目中应用测试工具，制定相应的测试工具使用策略，并把工具融入测试[**工作**](javascript:;)中。  
3.2.1  测试工具评估  
        测试工具的选型是成功应用测试工具的第一步，测试工具的选型应该注意以下几点：  
        （1）首先，分析项目的特点，软件系统采用的开发工具、语言、技术、平台等。还要结合测试的类型、测试的要求。  
        （2）同时还要了解目前存在的各种测试工具的情况，包括工具的生产厂家、价格、产品特性、技术支持和售后服务情况，还要了解该工具的市场占有率、使用人群等情况，如果是国外厂商生产的测试工具，最好再了解清楚国内的代理机构的情况等。  
        （3）选型的最后一步是编写选型报告。通过综合分析所有收集回来的材料，横向比较测试工具的优势和劣势。  
3.2.2  测试工具试用  
        在初步选型后，可定出几个满足要求的测试工具，然后进行深入的试用工作，应该尽可能尝试测试工具的所有功能，并且可能的话，要尽量在项目的软件系统中尝试。  
        需要制定一份详细的测试工具的试用计划，因为这可能是一项长时间的、需要谨慎进行的工作，尤其是对于那些商业的测试工具，动辄上百万的购置费用。很多公司由于没有谨慎进行前期的选型和试用工作，导致购买的测试工具不适用，或者使用效果不理想，最后被测试人员扔在角落里。  
        注意:不要仅仅听信测试工具销售人员的介绍就轻易购买，一定要自己组织一次详细的试用活动，确认适合在项目中使用，才能购买。  
3.2.3  自动化测试工具的培训  
        确定了选用的测试工具后，正式在测试项目中使用该测试工具之前，还需要组织相关测试人员进行测试工具的培训。测试工具的培训可包括以下内容。  
        （1）测试工具的总体介绍主要给测试人员讲解测试工具包括哪些主要的功能和特性，可用于哪些方面的测试。  
        （2）测试工具操作方法介绍主要给测试人员讲解测试工具的每一项功能的使用方法、操作步骤、注意事项等方面的内容。一般可由工具厂商派遣的技术支持人员进行，也可由熟悉该工具的测试人员来介绍，例如，负责前期测试工具试用的测试人员。  
        （3）测试工具使用实践，则是结合某个具体的例子给测试人员演示测试工具的使用方法和使用经验等。一般可由负责该测试工具试用的测试人员进行。  
        （4）对测试工具相关的测试理论进行讲解的目的是为了让测试人员了解该测试工具的原理，以及工具所应用的领域的相关理论知识，让测试人员在理论知识的指导下能更好、更恰当、更充分、更正确地使用测试工具。  
        技巧:测试工具的培训是成功引入测试工具的关键环节，在正式使用测试工具之前，应该确保测试人员充分掌握测试工具的基本使用方法，避免在使用过程中碰到很多工具操作和使用上的问题，导致测试进度缓慢。

（十一）

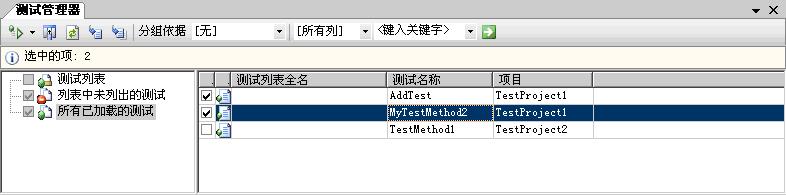
3.3  [**自动化测试**](javascript:;)工具的原理  
        测试工具的优势在于可部分地替代人工的测试过程，能重复不断地执行，能精确判断数值和字符对象。自动化测试工具把[**测试用例**](javascript:;)用自动的方式执行，例如，自动地产生数据，自动地打开应用程序，自动地查找控件，自动地输入数据，自动地操作控件，自动地收集测试结果，自动地与预期结果进行比较等。  
        自动化[**功能测试**](javascript:;)工具可基于GUI层面进行测试，也可基于代码层面进行测试。只要实现了自动化执行测试用例，自动化地检查测试数据的测试工具，替代人工进行测试步骤的执行，从而验证应用程序是否满足了特定功能的测试工具，都可以称为自动化功能测试工具。  
3.3.1  基于代码层面的功能自动化测试工具  
        基于代码层面的功能自动化测试工具主要是一些[**单元测试**](javascript:;)工具，例如[**JUnit**](javascript:;)、NUnit、MSTest等，这些工具直接访问被测试的应用程序的代码，对其中的类和函数进行调用，输入各种测试数据，检查函数的返回值，通过比较返回值与期待的值是否一致来判断测试是否通过。图3.2所示的是Visual Studio.NET 2005中的单元测试管理界面。  


图3.2  Visual Studio.NET 2005中的单元测试管理界面  
        这种类型的工具主要实现了测试代码框架产生的自动化，例如，下面代码是Visual Studio.NET 2005中的单元测试框架MSTest为某个类的方法自动产生的单元测试代码框架：  
// 以下代码由 Microsoft Visual Studio 2005 生成。  
// 测试所有者应该检查每个测试的有效性。  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
using System;  
using System.Text;  
using System.Collections.Generic;  
using AUT;  
namespace TestProject1  
{  
    /// <summary>  
    ///这是 AUT.Form1 的测试类，旨在包含所有 AUT.Form1 单元测试  
    ///</summary>  
    [TestClass()]  
    public class Form1Test  
    {  
        private TestContext testContextInstance;  
        /// <summary>  
        ///获取或设置测试上下文，上下文提供有关当前测试运行及其功能的信息。  
        ///</summary>  
        public TestContext TestContext  
        {  
            get  
            {  
                return testContextInstance;  
            }  
            set  
            {  
                testContextInstance = value;  
            }  
        }

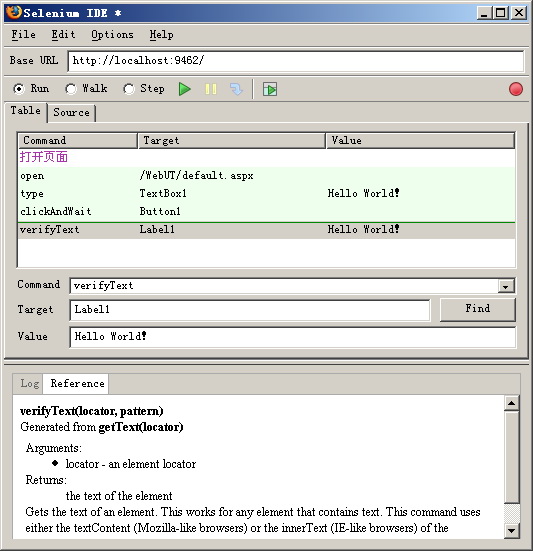
        #region 附加测试属性  
        //  
        //编写测试时，可使用以下附加属性:  
        //  
        //使用 ClassInitialize 在运行类中的第一个测试前先运行代码  
        //  
        //[ClassInitialize()]  
        //public static void MyClassInitialize(TestContext testContext)  
        //{  
        //}  
        //  
        //使用 ClassCleanup 在运行完类中的所有测试后再运行代码  
        //  
        //[ClassCleanup()]  
        //public static void MyClassCleanup()  
        //{  
        //}  
        //  
        //使用 TestInitialize 在运行每个测试前先运行代码  
        //  
        //[TestInitialize()]  
        //public void MyTestInitialize()  
        //{  
        //}  
        //  
        //使用 TestCleanup 在运行完每个测试后运行代码  
        //  
        //[TestCleanup()]  
        //public void MyTestCleanup()  
        //{  
        //}  
        //  
        #endregion

        /// <summary>  
        ///Add (int, int) 的测试  
        ///</summary>  
        [DeploymentItem("AUT.exe")]  
        [TestMethod()]  
        public void AddTest()  
        {  
            Form1 target = new Form1();  
            TestProject1.AUT\_Form1Accessor accessor =  
new TestProject1.AUT\_Form1Accessor(target);  
            int i = 0; // TODO: 初始化为适当的值  
            int j = 0; // TODO: 初始化为适当的值  
            int expected = 0;  
            int actual;  
            actual = accessor.Add(i, j);  
            Assert.AreEqual(expected, actual, "AUT.Form1.Add 未返回所需的值。");  
            Assert.Inconclusive("验证此测试方法的正确性。");  
        }  
    }  
}  
        在代码框架的背后，单元测试框架负责查找和调用被测试的类和方法，通过代码反射机制可以访问到被测试代码中的所有方法和属性。另外，单元测试框架会提供一系列的Assert类，使用这些Assert类可以简化测试结果检查、判断的工具。  
        提示:在执行单元测试时，单元测试框架负责加载包含测试类的程序集文件，通过查找里面的测试类和测试方法标识来加载测试方法，例如，上面代码中的“[TestMethod()]”就是用于标识其中的测试方法。

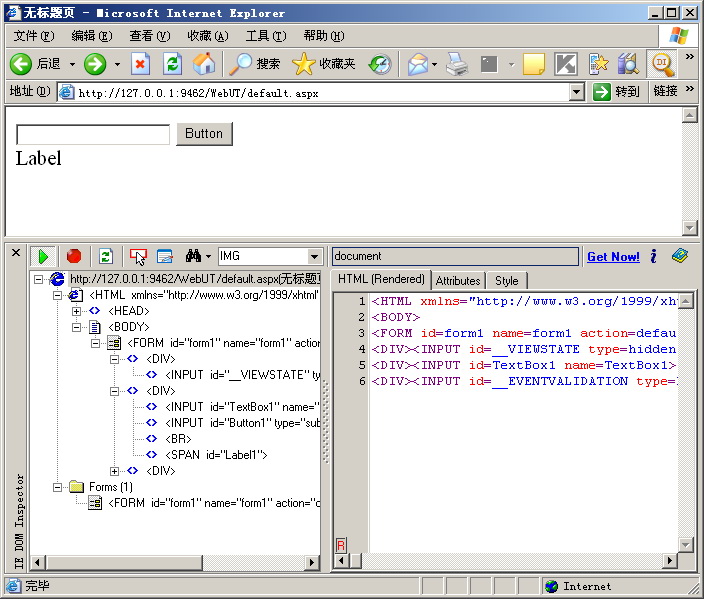
（十二）

3.3.2  基于浏览器和DOM对象模型的功能[**自动化测试**](javascript:;)工具

        另外一种自动化的[**功能测试**](javascript:;)工具是基于浏览器和DOM对象模型开发的，例如[**Selenium**](javascript:;)、Watir等，这些测试工具直接访问Web浏览器，利用脚本语言操纵浏览器和Web页面中包含的DOM对象，从而达到模拟用户控制浏览导航、页面元素的操纵等效果，并且直接获取DOM对象的属性，从而获得Web页面元素的各种属性，通过这些属性可判断测试步骤的结果是否正确。图3.3所示的是可作为插件嵌入到Mozilla Firefox浏览器中的Selenium IDE的测试界面。

  
图3.3  Selenium IDE的测试界面

        HTML DOM（Document Object Model）是一个HTML文档的编程接口，它定义了HTML的标准对象集合，并且定义了标准的访问和操纵HTML对象的方式。HTML DOM接口让测试人员可以访问和操纵HTML文档的内容。图3.4所示的界面是使用了一个名为“IE DOM Inspector”的工具查看到的Web页面中的DOM对象。

   
图3.4  IE DOM Inspector的界面

        如果熟悉和了解DOM的原理，那么完全可以自己动手编写一个基于浏览器和DOM的Web页面自动化测试工具，例如，下面的C#代码就是一个简单的例子：

|  |
| --- |
| using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Text; using System.Windows.Forms; using System.Diagnostics; using System.Threading; // 引用Microsoft.mshtml的HTML接口 using mshtml; // 引用IE对象 using SHDocVw;  namespace WebAutomatedTest1 {     public partial class Form1 : Form     {         static AutoResetEvent documentComplete = new AutoResetEvent(false);         public Form1()         {             InitializeComponent();         }         private void button1\_Click(object sender, EventArgs e)         {             InternetExplorer ie = null;             // 启动IE的进程             Process p = Process.Start("iexplore.exe", "about:blank");             // 等待一段时间，让IE启动             Thread.Sleep(3000);             if (p == null)             {                 MessageBox.Show("不能启动IE！");                 return;             }             SHDocVw.ShellWindows allBrowsers = new SHDocVw.ShellWindows();             // 附加到IE进程             int i = 0;             while (i < allBrowsers.Count && ie == null)             {                 InternetExplorer browser = (InternetExplorer)allBrowsers.Item(i);                 if (browser.HWND == (int)p.MainWindowHandle)                     ie = browser;                 ++i;             }             if (ie == null)             {                 MessageBox.Show("不能附加到IE！");                 return;             }             ie.DocumentComplete += new DWebBrowserEvents2\_DocumentCompleteEventHandler(ie\_DocumentComplete);             object nil = new object();              ie.Navigate("[http://127.0.0.1:9462/WebUT/default.aspx",ref](http://127.0.0.1:9462/WebUT/default.aspx%22,ref)nil,ref nil,ref nil,ref nil);             documentComplete.WaitOne();             HTMLDocument Doc = (HTMLDocument)ie.Document;              HTMLInputElement textBox = (HTMLInputElement)Doc.getElementById("TextBox1");             textBox.value = "123";              HTMLInputElement button = (HTMLInputElement)Doc.getElementById("Button1");             button.click();             // 验证,如果Label1的值等于123，则表示测试通过              HTMLSpanElement label = (HTMLSpanElement)Doc.getElementById("Label1");             if (label.innerText == "123")             {                 MessageBox.Show("测试通过！");             }             else             {                 MessageBox.Show("测试不通过！");             }         }         private static void ie\_DocumentComplete(object pDisp, ref object URL)         {             documentComplete.Set();         }     } } |

（十三）

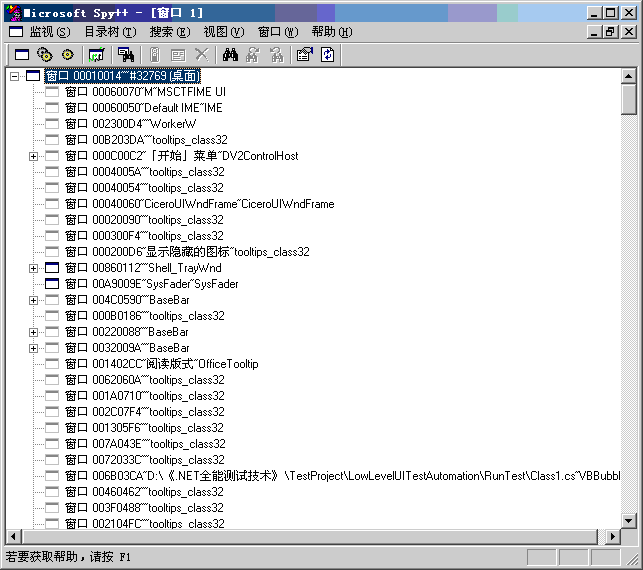
　3.3.3  基于GUI对象识别的[**测试**](javascript:;)**工具**原理  
　　目前，大部分[**自动化功能测试**](javascript:;)工具，尤其是商业的测试工具，都是基于GUI对象识别[**技术**](javascript:;)来设计的。基于GUI层面的测试需要与各种界面元素打交道，而且不同的编程语言和开发工具开发的应用程序在界面的表现、事件的响应上都略有不同，因此，设计基于GUI层面的自动化[功能测试](http://www.51testing.com/)工具会更为复杂些。  
　　[**Windows**](javascript:;)的API中封装了很多可用于[**自动化测试**](javascript:;)编程的函数，例如FindWindow、GetWindowRect等函数。这些函数可在编程语言或脚本代码中进行调用，从而实现自动化测试编程。例如，下面的C#代码封装了user32.dll中的FindWindow函数， 实现查找指定类型和名称的控件的功能：

|  |
| --- |
| using System; using System.Collections.Generic; using System.Text; using System.Runtime.InteropServices;  namespace LowLevelGUITest {     public class Automation     {         // 引入user32.dll中的FindWindow函数         [DllImport("user32.dll", CharSet = CharSet.Auto)]         static extern IntPtr FindWindow(string lpClassName, string lpWindowName);         // 查找控件方法         public IntPtr FindControl(string ClassName,string ControlName)         {             // 返回控件句柄             return FindWindow(ClassName, ControlName);         }     } } |

　　可用下面的C#代码来调用这个封装的函数：

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)      {          Automation automation = new Automation();          // 查找名为AUT的窗体          IntPtr p = automation.FindControl("WindowsForms10.Window.8.app.0.378734a","AUT");          // 判断是否找到控件          if (p == IntPtr.Zero)          {              Console.WriteLine("找不到指定的控件！");          }      } |

　　如果想进一步了解GUI对象识别的原理，可以打开Visual Studio.NET 2005自带的Spy++，如图3.5所示。

[](http://www.51testing.com/batch.download.php?aid=10828)

　　Spy++中的搜索查找控件功能就是这种对象识别原理的体现。例如，在图3.6所示的界面中，可以看到某个指定的窗口的句柄、标题、类。

[](http://www.51testing.com/batch.download.php?aid=10829)

3.3.4  反射机制在自动化功能测试工具中的使用  
　　在基于GUI对象识别和控制的自动化测试工具中，过去一直依赖于Windows API函数的调用。而随着新的编程语言和平台的出现，涌现了很多新的语言特性，这些语言特性可用于自动化测试工具的设计，例如反射机制就是其中一项技术。  
　　程序集包括模块，模块包含类型，类型包含成员。在.NET和Java中，都提供了反射的机制，反射提供了封装程序集、模块和类型的对象，可以通过反射动态地创建类型的实例，将类型绑定到现有对象，或者从现有对象中获取类型，然后调用类型的方法或访问其字段和属性。  
　　反射机制可被用在测试中，通过反射来加载被测试程序，获取被测试程序的各种属性，触发被测试程序的各种事件，从而达到自动化测试的目的。例如，下面的C#代码通过反射机制读取程序中textBox1控件的Text属性：

|  |
| --- |
| // 获取控件属性         static object GetControlPropertyValue(string controlName,  string propertyName)         {             if (AUTForm.InvokeRequired)             {                 Thread.Sleep(1000);                 return AUTForm.Invoke (new GetControlPropertyValueHandler(GetControlPropertyValue),  new object[] { controlName, propertyName });             }             // 获取类型             Type t1 = AUTForm.GetType();             // 获取类型中的成员             FieldInfo fi = t1.GetField(controlName, BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Static | BindingFlags.Instance);             object ctrl = fi.GetValue(AUTForm);             Type t2 = ctrl.GetType();              // 获取成员中的属性             PropertyInfo pi = t2.GetProperty(propertyName, BindingFlags.Public |  BindingFlags.NonPublic | BindingFlags.Static | BindingFlags.Instance);              // 返回控件属性值             return pi.GetValue(ctrl, new object[0]);         }         delegate object GetControlPropertyValueHandler(string controlName, string propertyName); |

　　而下面的C#代码则通过反射机制调用控件的方法，模拟用户点击按钮的过程：

|  |
| --- |
| // 模拟用户点击按钮         InvokeMethod("button1\_Click", new object[] { null, new EventArgs() });         // 调用控件方法         static void InvokeMethod(string methodName, params object[] parms)         {             if (AUTForm.InvokeRequired)             {                 Thread.Sleep(1000);                 AUTForm.Invoke(new InvokeMethodHandler(InvokeMethod),  new object[] { methodName, parms });                 return;             }             // 获取类型             Type t = AUTForm.GetType();             // 获取类型中的指定方法             MethodInfo mi = t.GetMethod(methodName, BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Static | BindingFlags.Instance;             // 调用方法             mi.Invoke(AUTForm, parms);         }        delegate void InvokeMethodHandler(string methodName, params object[] arms); |

（十四）

　3.3.5  UI Automation在自动化[**功能测试**](javascript:;)工具中的使用  
　　随着.NET 3.0和Vista的发布，[**微软**](javascript:;)提供了一套崭新的[**Windows**](javascript:;)界面框架，名为“UI Automation”，UI Automation在所有支持Windows Presentation Foundation（WPF）的[**操作系统**](javascript:;)中都可以使用。  
　　UI Automation提供了一套对Windows界面的可编程的访问机制，同时也为测试自动化工具提供了一个新的思路。例如，在使用[**QTP**](javascript:;)的.NET插件对WPF控件进行测试时，就可以使用AutomationElement来访问控件属性，再例如，下面脚本通过AutomationElement访问Test4窗体控件的Name属性：  
'　　通过AutomationElement访问控件属性  
　　msgbox WpfWindow("Test4").AutomationElement.Current.Name  
　　3.3.6  QTP对象识别技术  
　　有了前面的基础，就不难理解QTP的[**工作**](javascript:;)原理了。QTP同样是通过查找应用程序界面中的各个控件的属性来判断是否与测试对象匹配，例如QTP中提供的“Object Spy”功能，就可以把某个界面控件的基本属性列举出来，如图3.7所示。  
　　另外，还可以根据控件的类型，把其拥有的可操作的方法列举出来，如图3.8所示。  
针对不同平台和语言编写的控件，QTP会采用不同的对象识别方法，但是共通的一点是，通过查找对象属性，依据该控件与[**其他**](javascript:;)控件能区分的属性来判断其身份，例如控件的类名、控件的文本等。QTP中的“Object Identification”界面中提供了一个可供测试员调整的对象识别配置器，如图3.9所示。

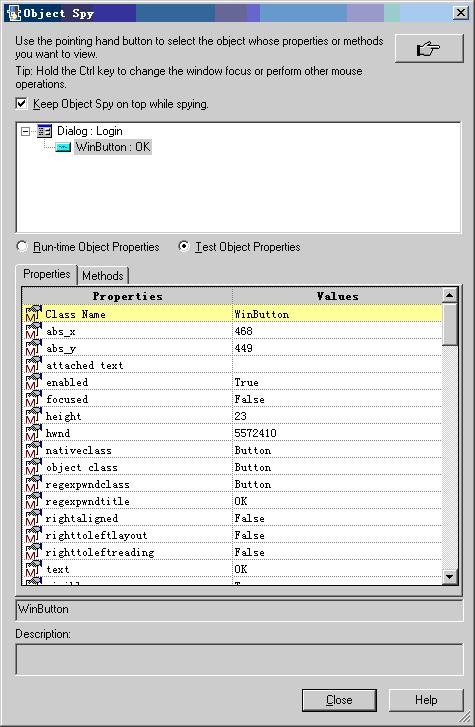
[](http://www.51testing.com/batch.download.php?aid=10878)

　　图3.7  QTP的“Object Spy”功能

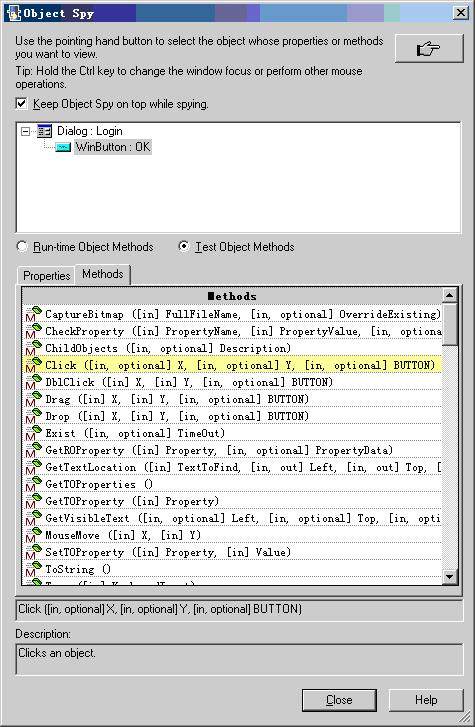
[](http://www.51testing.com/batch.download.php?aid=10879)

　　图3.8  使用“Object Spy”查看控件可操作的方法

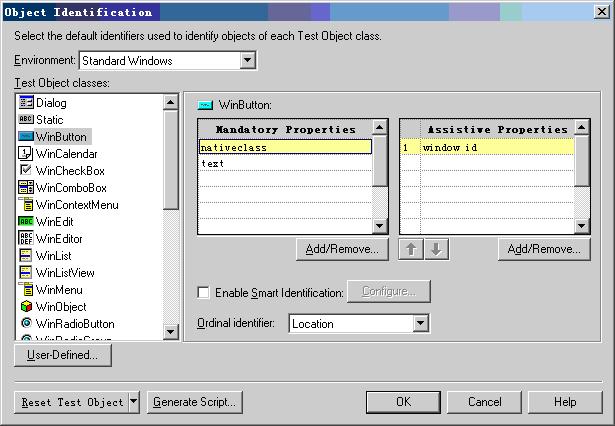
[](http://www.51testing.com/batch.download.php?aid=10880)

　　图3.9  QTP的“Object Identification”界面

　　提示：其中“Mandatory Properties”中的属性是强制性的判断准则，“Assistive Properties”中的属性用于辅助性地判断。  
　　单击“Add/Remove”按钮，可在如图3.10所示的界面中添加更多的属性用于唯一地标识和区别某个控件实例。

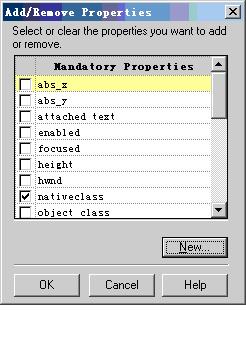
[](http://www.51testing.com/batch.download.php?aid=10881)

　　图3.10  添加属性

（十五）

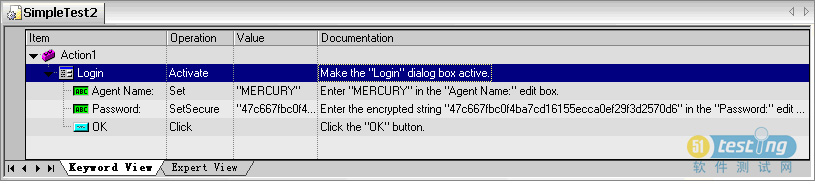
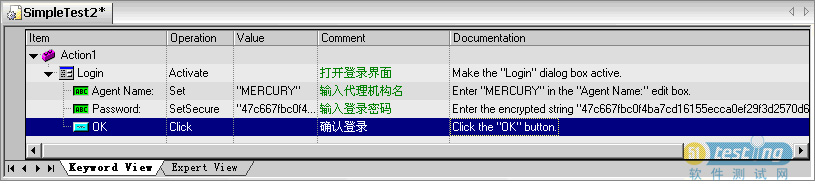
　前面的章节介绍了[**QTP**](javascript:;)的安装和设置，并且利用录制功能创建了一个基本的[**自动化**](javascript:;)**测试**脚本。在本章，将基于这个基本的测试脚本介绍QTP基本功能的使用，包括如何利用QTP提供的各种辅助功能来创建测试脚本，编辑和调试测试脚本，运行测试脚本，以及查看和分析测试结果。  
　　5.1  QTP基本功能的使用  
　　QTP的基本功能包括两大部分：一部分是提供给初级用户使用的关键字视图；另一部分是提供给熟悉VBScript脚本编写的自动化测试工程师使用的专家视图。但是，并没有严格的区分，在实际的自动化测试项目中完全可以两者结合着使用。  
　　5.1.1  QTP自动化测试的基本过程  
　　使用QTP进行自动化测试的基本过程与使用[**其他**](javascript:;)自动化测试工具进行自动化[**功能测试**](javascript:;)的过程基本是一致的，一般包括以下5个步骤：  
　　（1）录制测试脚本。利用QTP先进的对象识别、鼠标和键盘监控机制来录制测试脚本，测试人员只需要模拟用户的操作，像执行手工测试的测试步骤一样操作被测试应用程序的界面即可。  
　　（2）编辑测试脚本。主要包括调整测试步骤、编辑测试逻辑、插入检查点（CheckPoint）、添加测试输出信息、添加注释等。  
　　（3）调试测试脚本。利用“Check Syntax”功能检查测试脚本的语法错误，利用QTP脚本编辑界面的调试功能检查测试脚本逻辑的正确性。  
　　（4）运行测试脚本。可运行单个“Action”，也可批量运行测试脚本。  
　　（5）分析测试结果。使用QTP的测试结果查看工具查看测试结果，检查测试运行过程的正确性。  
　　5.1.2  在关键字视图中编辑测试脚本  
　　对于录制下来的测试脚本，需要进一步修改和调整，这些[**工作**](javascript:;)可在关键字视图（Keyword View）中进行，也可在专家视图（Expert View）中进行。  
　　对于新手而言，关键字视图可能会更容易理解和使用一些，因为它不需要理解和熟悉测试脚本语言。例如，通过查看如图5.1所示的关键字视图，可以很容易地知道这个测试经过了4个测试步骤，分别是：  
　　（1）让Login窗口处于激活状态。  
　　（2）在“Agent Name”输入框中输入字符串“MERCURY”。  
　　（3）在“Password”输入框中输入密码。  
　　（4）单击“OK”按钮。  
　　

　　图5.1  查看测试步骤  
　　技巧：在录制过程中，QTP为每一步操作自动生成操作文档，放到关键字视图的“Documentation”列中显示出来，在“Documentation”列的列头单击鼠标右键，选择“Copy Documentation to Clipboard”可以导出测试步骤文档，生成测试用例，用于指导手工测试的进行。  
　　5.1.3  在关键字视图中为测试步骤添加注释  
　　虽然QTP能为每一个录制的测试步骤自动生成文档，但是，未必能满足对测试脚本的理解上的要求，因此，还需要为测试步骤添加必要的注释。方法是：首先在关键字试图的表格列头中单击鼠标右键，选择“Comment”，则会多出一列名为“Comment”的列，在这一列中可为每一个测试步骤添加注释，如图5.2所示。



5.1.4  在关键字视图中添加测试信息的输出  
　　如果想在某个测试步骤完成后，输出相应的测试信息到测试报告中，则可单击鼠标右键，选择“Insert Step | Report”，出现如图5.3所示的界面。

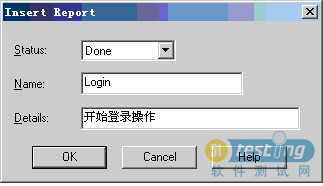


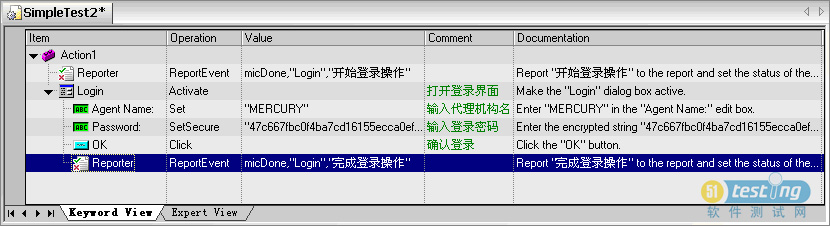
　　图5.3  插入报告  
　　在界面中的“Status”下拉框中，选择写入测试报告的状态（可以是Done、Passed、Failed、Warning这4种状态中的一种）；在“Name”中输入信息摘要；在“Details”中输入详细的测试步骤信息。  
　　例如，可在激活Login界面之前报告“开始登录操作”，在单击OK按钮后报告“完成登录操作”，如图5.4所示。  
　　

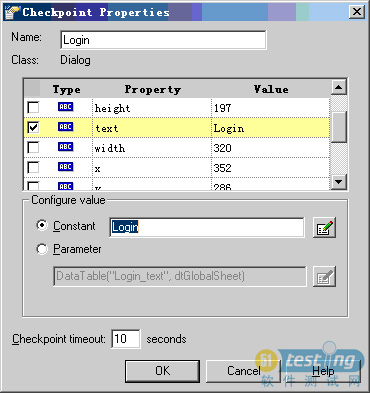
　　图5.4  添加报告  
　　5.1.5  插入检查点（Checkpoint）  
　　如果想检查Login窗口的名称是否为“Login”，如果窗口名称不是“Login”，那么认为测试不通过。这可在关键字视图中通过QTP的检查点插入功能来实现。方式如下：  
　　（1）首先，定位到激活Login窗口所在的测试步骤。  
　　（2）然后单击鼠标右键，选择菜单“Insert Standard Checkpoint”，出现如图5.5所示的界面。  
　　

　　图5.5  插入标准检查点  
　　（3）在界面中，选择需要检查的属性，例如，选择“Enabled”属性，设置为“True”，选择“Text”属性，设置为“Login”。单击“OK”按钮后，则可在关键字视图中看到新添加的检查点步骤，如图5.6所示。

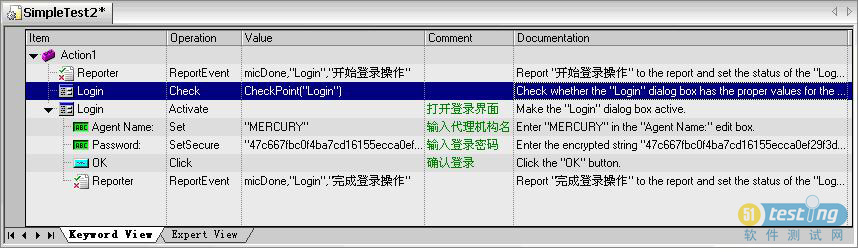


　　图5.6  添加新的检查点步骤

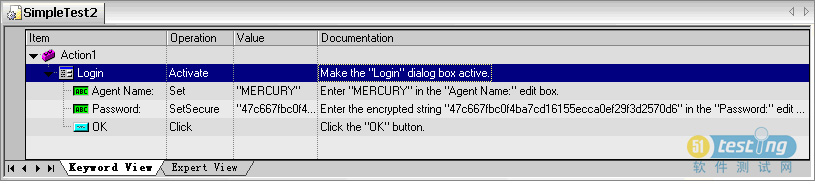
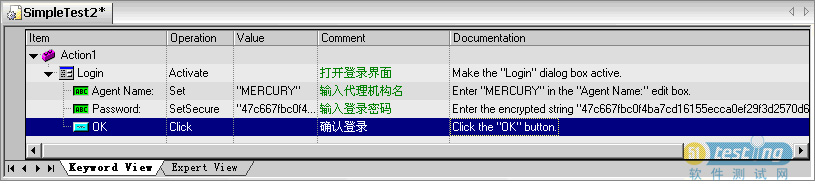
　　前面的章节介绍了QTP的安装和设置，并且利用录制功能创建了一个基本的自动化测试脚本。在本章，将基于这个基本的测试脚本介绍QTP基本功能的使用，包括如何利用QTP提供的各种辅助功能来创建测试脚本，编辑和调试测试脚本，运行测试脚本，以及查看和分析测试结果。  
　　5.1  QTP基本功能的使用  
　　QTP的基本功能包括两大部分：一部分是提供给初级用户使用的关键字视图；另一部分是提供给熟悉VBScript脚本编写的自动化测试工程师使用的专家视图。但是，并没有严格的区分，在实际的自动化测试项目中完全可以两者结合着使用。  
　　5.1.1  QTP自动化测试的基本过程  
　　使用QTP进行自动化测试的基本过程与使用其他自动化测试工具进行自动化功能测试的过程基本是一致的，一般包括以下5个步骤：  
　　（1）录制测试脚本。利用QTP先进的对象识别、鼠标和键盘监控机制来录制测试脚本，测试人员只需要模拟用户的操作，像执行手工测试的测试步骤一样操作被测试应用程序的界面即可。  
　　（2）编辑测试脚本。主要包括调整测试步骤、编辑测试逻辑、插入检查点（CheckPoint）、添加测试输出信息、添加注释等。  
　　（3）调试测试脚本。利用“Check Syntax”功能检查测试脚本的语法错误，利用QTP脚本编辑界面的调试功能检查测试脚本逻辑的正确性。  
　　（4）运行测试脚本。可运行单个“Action”，也可批量运行测试脚本。  
　　（5）分析测试结果。使用QTP的测试结果查看工具查看测试结果，检查测试运行过程的正确性。  
　　5.1.2  在关键字视图中编辑测试脚本  
　　对于录制下来的测试脚本，需要进一步修改和调整，这些工作可在关键字视图（Keyword View）中进行，也可在专家视图（Expert View）中进行。  
　　对于新手而言，关键字视图可能会更容易理解和使用一些，因为它不需要理解和熟悉测试脚本语言。例如，通过查看如图5.1所示的关键字视图，可以很容易地知道这个测试经过了4个测试步骤，分别是：  
　　（1）让Login窗口处于激活状态。  
　　（2）在“Agent Name”输入框中输入字符串“MERCURY”。  
　　（3）在“Password”输入框中输入密码。  
　　（4）单击“OK”按钮。  
　　

　　图5.1  查看测试步骤  
　　技巧：在录制过程中，QTP为每一步操作自动生成操作文档，放到关键字视图的“Documentation”列中显示出来，在“Documentation”列的列头单击鼠标右键，选择“Copy Documentation to Clipboard”可以导出测试步骤文档，生成测试用例，用于指导手工测试的进行。  
　　5.1.3  在关键字视图中为测试步骤添加注释  
　　虽然QTP能为每一个录制的测试步骤自动生成文档，但是，未必能满足对测试脚本的理解上的要求，因此，还需要为测试步骤添加必要的注释。方法是：首先在关键字试图的表格列头中单击鼠标右键，选择“Comment”，则会多出一列名为“Comment”的列，在这一列中可为每一个测试步骤添加注释，如图5.2所示。



（十六）

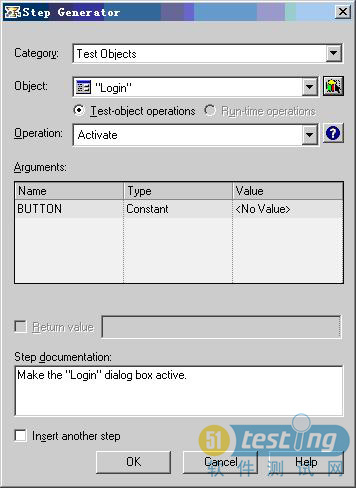
　5.1.6  在关键字视图中插入新的[**测试**](javascript:;)步骤  
　　如果想在打开登录界面并激活Login窗口的测试步骤之后，而在输入代理机构名的测试步骤之前，先单击“Help”按钮查看帮助，则需要使用插入测试步骤的功能。  
　　（1）选择“打开登录界面”所在的测试步骤，单击鼠标右键，选择菜单“Insert Step | Step Generator”，则出现如图5.7所示的界面。  
　　

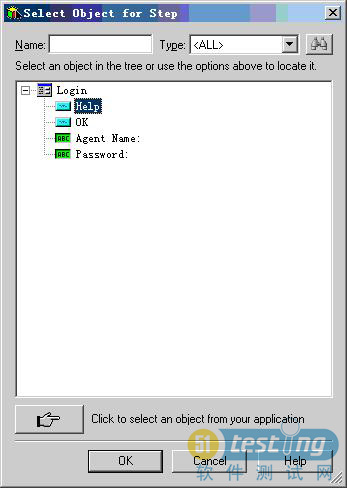
　　图5.7  测试步骤产生器  
　　“Category”包括“[**Test**](javascript:;) Objects”、“Utility Objects”和“Functions”，分别代表如下含义：  
　　Test Objects：测试对象，即被测试应用程序的界面上的控件元素。  
　　Utility Objects：工具对象，是[**QTP**](javascript:;)内建的各种用于编写测试脚本、辅助建立测试逻辑的工具类对象。  
　　Functions：各种函数，包括库函数、内建函数、本地脚本函数。  
　　（2）在这里，选择“Test Objects”，然后单击“Object”下拉框旁边的图标按钮，出现如图5.8所示的界面，在界面中选择“Help”对象，然后单击“OK”按钮。  
　　

　　图5.8  为测试步骤选择测试对象  
　　注意：如果“Help”对象不在列表中，则可单击界面中的手型按钮，然后移动到Flight程序的“Login”界面，从中选择测试对象“Help”按钮，则出现如图5.9所示的界面。单击“OK”按钮，把“Help”按钮对象添加到测试对象列表中。



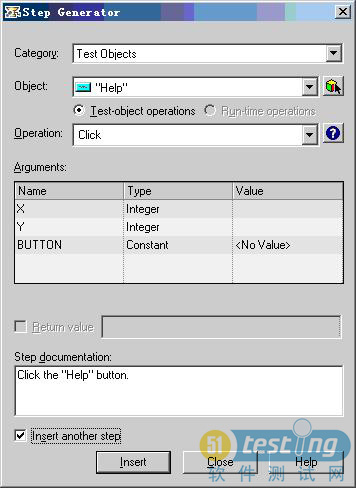
　（3）返回“Step Generator”界面后，在“Operation”的下拉框中选择“Click”，并把“Insert another step”选项勾选上，如图5.10所示。  
　　

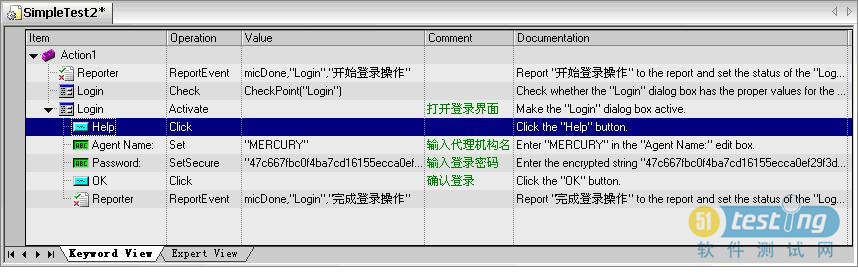
　　图5.10  设置测试步骤属性  
　　（4）单击“Insert”按钮，然后单击“Close”按钮，返回关键字视图，可看到新的测试步骤已经被添加，如图5.11所示。  
　　

　　图5.11  添加新的测试步骤  
　　5.1.7  在专家视图中编辑测试脚本  
　　在QTP中，选择菜单“View | Expert View”，可看到如图5.12所示的测试脚本。

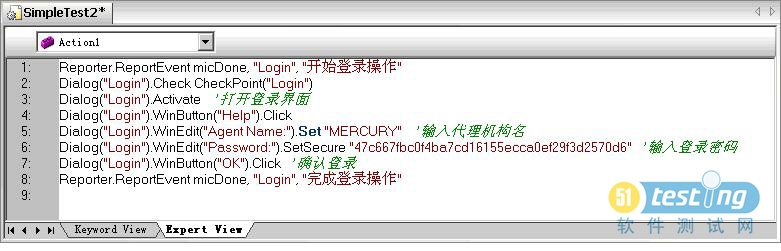


　　图5.12  专家视图  
　　“Expert View”是一个强大的VBScript的脚本编辑器，在这里，可以直接编写测试脚本的代码，适合熟悉VBScript语言、有较好的编码技巧的自动化测试工程师使用。  
　　5.1.8  脚本编辑器的使用  
　　QTP提供的脚本编辑器支持“语法感知”功能，例如，在代码中输入“Dialog("Login")”后加点，则自动显示一个下拉列表，从中可选取属于“Login”测试对象所包含的所有属性和方法，如图5.13所示。

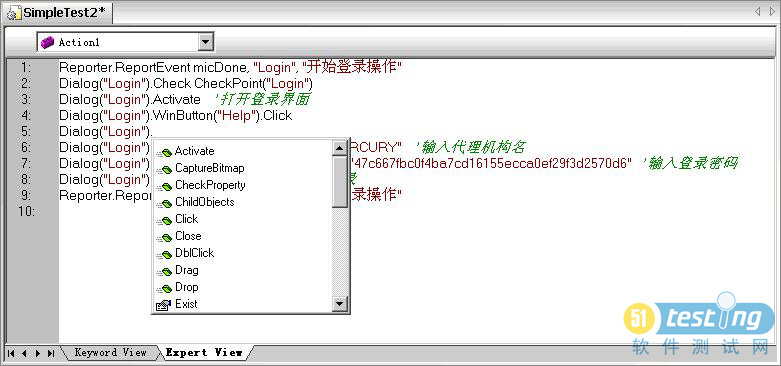


　　图5.13  脚本编辑器  
　　提示：这种功能对于自动化测试工程师编写测试脚本有很大的帮助，能减少很多拼写的错误，以及减轻记忆的难度。

（十七）

5.2  编辑[**测试**](javascript:;)脚本象

　　在掌握了[**QTP**](javascript:;)的关键字视图和专家视图的基本使用方法后，就可以综合使用这两个测试视图，结合对象库、函数库等辅助手段来编辑测试脚本。

5.2.1  识别对象

　　编辑测试脚本的第一步是识别测试对象，因为基于GUI的[**自动化测试**](javascript:;)主要是围绕着界面的控件元素来进行的。QTP针对不同语言开发的控件，采取不同的对象识别[**技术**](javascript:;)，根据加载的插件来选择相应的控件对象识别的依据。在QTP中，选择菜单“Tools | Object Identification”，出现如图5.14所示的界面。

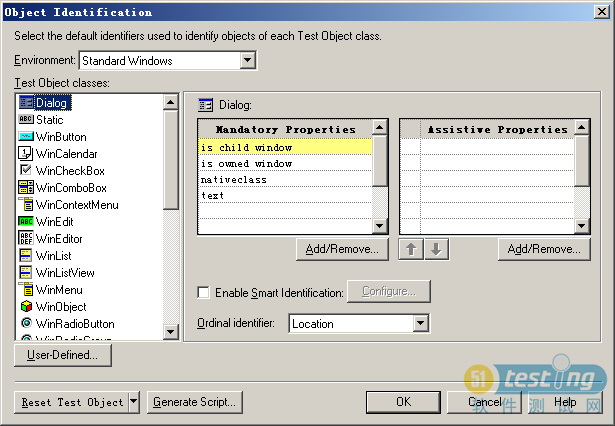


　　图5.14  对象识别定义界面

　　在界面中可看到各种标准[**Windows**](javascript:;)控件对应的对象识别方法，例如，对于Dialog控件，使用的是“is child window”、“is owned window”、“nativeclass”和“text”这四个控件对象的属性来区别出一个唯一的Dilalog控件对象。

　　可以单击“Add/Remove”按钮，在如图5.15所示的界面中，选择更多的控件属性来唯一识别控件。



图5.15  选择或移除属性

差一页

5.2.3  对象库管理

　　另外一种观察和了解测试程序的界面控件元素，以及它们的层次关系的方法是通过对象库（“Object Repository”）。在QTP中，选择菜单“Resources | Object Repository”，出现如图5.19所示的界面。

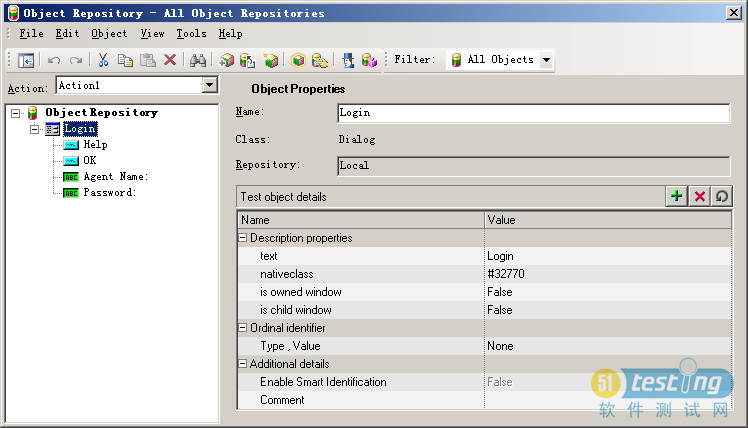


　　图5.19  对象库管理

　　QTP在录制测试脚本的过程中会把界面操作涉及的控件对象都自动添加到对象库中，但是那些未被鼠标点击或键盘操作的界面控件则不会添加到对象库中。

5.2.4  把测试对象添加到对象库中

　　在对象库管理界面中，选择菜单“Objects | Add Object to Local”，然后选择测试程序界面中的某个控件，例如，选择Flight程序的“Login”界面中的“Cancel”按钮，则出现如图5.20所示的界面。

　　单击“OK”按钮，把测试对象添加到对象库中，如图5.21所示。



　　图5.20  对象选择

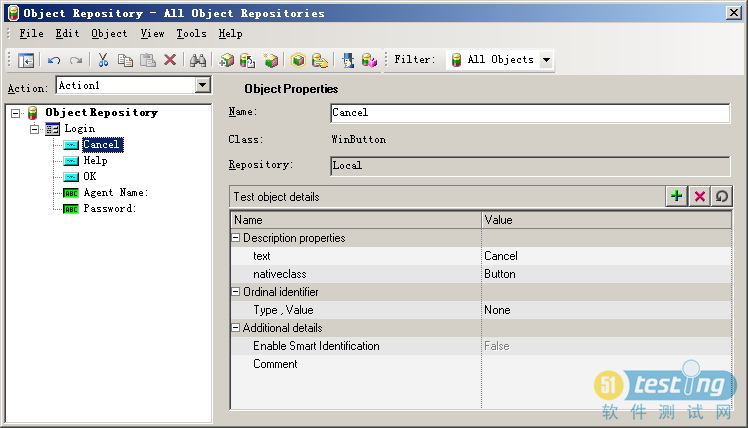


图5.21  添加测试对象到对象库

从右边的对象属性窗口中可看到，对于“Cancel”这个“WinButton”类型的控件，QTP使用“text”和“nativeclass”两个属性来唯一区分和识别。这两个属性将作为测试脚本运行时找到测试程序界面上的相应控件的依据。

　　界面中的某些控件对象是有层次关系的，例如，按钮、输入框等控件包含在窗口控件中，在添加测试对象到对象库时，可以选择窗口对象，然后在如图5.22所示的界面中选择“All object types”，单击“OK”按钮，则会把选择的窗口对象中的所有控件对象添加到对象库中。

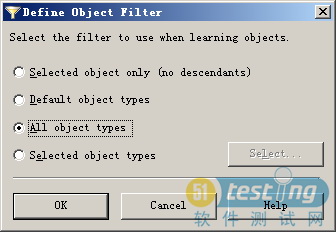


　　图5.22  定义对象过滤

5.2.5  导出对象库文件

　　测试对象作为资源，可导出到文件中，以方便其他测试脚本的使用，方法是在对象库管理界面中，选择菜单“File | Export Local Object”，存储到某个文件夹中。

（十八）

　5.2.6  在[**测试**](javascript:;)脚本中访问对象库的测试对象  
　　把界面的控件作为测试对象添加到对象库中之后，就可以把这些测试对象作为测试资源来访问，例如，在关键字视图中可从对象库中选择需要的测试对象，如图5.23所示。



　　图5.23  从对象库选择测试对象  
　　在下拉列表中选择“Object from repository”，出现如图5.24所示的界面，在界面中选择对象库中的测试对象。

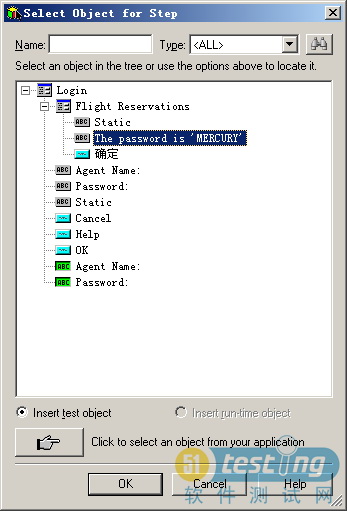


　　图5.24  为测试步骤选择对象  
　　而在专家视图中，同样可以访问到对象库的测试对象，以及它们的属性和方法，如图5.25所示。

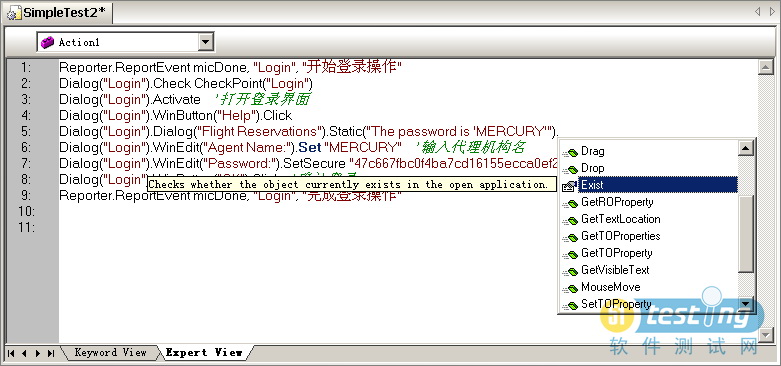


 　　图5.25  在专家视图访问对象库的测试对象

　　5.2.7  添加新的Action  
　　在[**QTP**](javascript:;)中，“Action”相当于测试脚本的文件，可使用Action来划分和组织测试流程，例如，把一些公用的操作放到同一个Action中，以便重用。  
　　如果想在当前Action的某个测试步骤之后添加新的Action，则可选择菜单“Insert | Call to New Action”，出现如图5.26所示的界面。

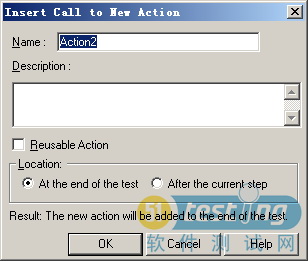


　　图5.26  添加新的Action  
　　在“Name”中输入Action的名称，例如“Action\_Help”，在“Description” 中输入对该Action的描述，例如“处理Help窗口”，在“Location”中选择“After the current step”，然后，单击“OK”按钮，返回关键字视图，则可看到新的名为“Action\_Help”的Action已经成功被添加，如图5.27所示。

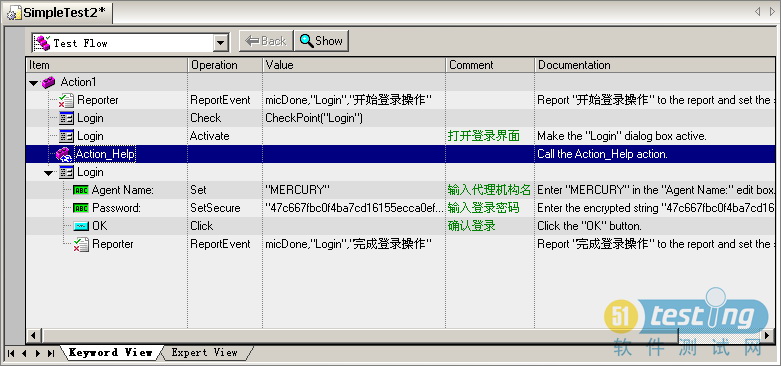
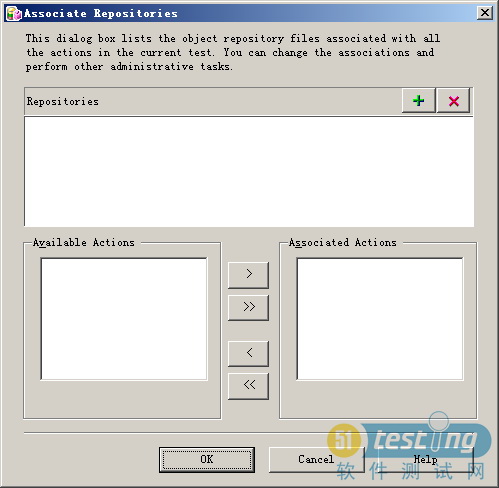


　　图5.27  成功添加“Action\_Help”  
　　5.2.8  关联Action的对象库  
　　双击新添加的Action，可在该Action中添加新的测试代码。由于QTP为每一个Action生成相应的测试文件和目录，而对象库作为资源，也是与Action绑定的，因此，新添加的Action不能直接使用前一个Action中的测试对象。  
　　解决方法有两种，一种是通过录制新的测试脚本来产生新的测试对象库，另外一种是通过关联前一个Action所导出的对象库文件来使用其测试对象。下面介绍第二种方法：  
　　（1）选择菜单“Resources | Associate Repositories”，出现如图5.28所示的界面。



差一页

（2）在“Function definition”中的“Name”输入框中输入函数的名称，在Type中选择“Function”，在“Scope”中选择“Public”，在“Description”中输入函数的描述信息，例如“用于演示如何利用Function Definition Generator创建函数”，如图5.33所示。

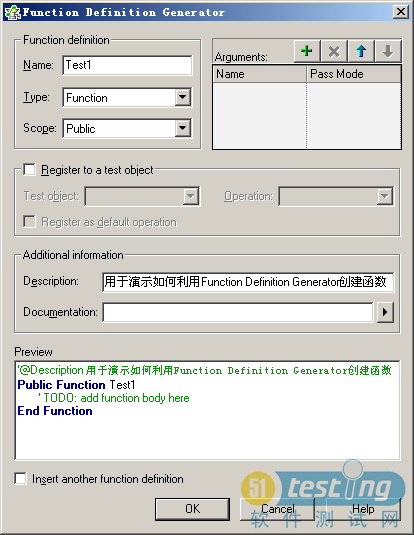


　　图5.33  定义函数  
　　提示：如果函数有输入参数，则可以在“Arguments”中单击“+”按钮进行添加。  
　　（3）单击“OK”按钮，则会在当前Action的测试代码中，添加如图5.34所示的函数框架代码。



　　图5.34  产生函数框架代码  
　　在这里可以简单地写一个函数，用于往测试报告中添加一条信息，代码如下所示：

|  |
| --- |
| ['@Description](mailto:%27@Description) 用于演示如何利用Function Definition Generator创建函数 Public Function Test1        ' TODO: add function body here     Reporter.ReportEvent micDone ,"Test1" ,"Test1" ' 往测试报告中添加一条信息 End Function |

　　（4）添加完函数代码后，就可以在测试代码中调用这个函数，调用方法有两种，一种是在专家视图中编写代码调用，只需要简单地输入函数名即可，如下面代码所示：

|  |
| --- |
| Test1 ' 调用函数Test1 Dialog("Login").WinButton("Help").Click '打开帮助界面 Dialog("Login").Dialog("Flight Reservations").Activate Dialog("Login").Dialog("Flight Reservations").WinButton("确定").Click ' 单击确定按钮关闭帮助界面  ['@Description](mailto:%27@Description) 用于演示如何利用Function Definition Generator创建函数 Public Function Test1        ' TODO: add function body here     Reporter.ReportEvent micDone ,"Test1" ,"Test1" ' 往测试报告中添加一条信息 End Function |

　　另外一种方法是在关键字视图中，通过选择菜单“Insert | Step Generator”，在如图5.35所示的界面中，选择“Category”为“Functions”，选择“Library”为“Local script functions”，然后选择“Operation”为“Test1”即可。



图5.35  通过Step Generator来添加函数调用  
　　添加对Test1的调用后，在关键字视图中可看到，新加了一个测试步骤“Function Call”，如图5.36所示

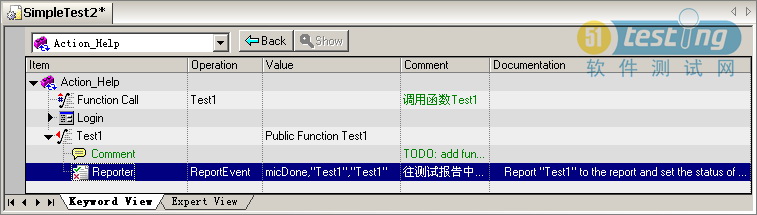
。  
　　

　　图5.36  添加“Function Call”测试步骤

（十九）

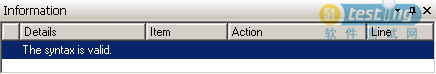
5.3  运行和调试[**测试**](javascript:;)脚本  
　　编辑好测试脚本后，在运行测试之前，可利用[**QTP**](javascript:;)的语法检查功能和脚本调试功能对测试脚本的逻辑进行检查。  
　　5.3.1  语法检查  
　　选择菜单“Tools | Check Syntax”，或者按快捷键“CTRL + F7”对测试脚本进行语法检查，如果语法检查通过，则在“Information”界面（可通过选择菜单“View | Information”打开）提示如图5.37所示的信息。  
　　

　　图5.37  语法检查通过  
　　如果语法检查发现有问题，则会在Information界面中列出详细的信息，如图5.38所示，包括语法错误的信息描述，出现在哪个Action的哪行代码，双击该提示信息，将转到相应的测试脚本的代码行。

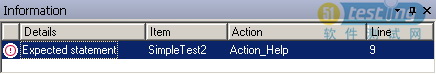


　　图5.38  语法检查的详细信息  
　　5.3.2  使用断点  
　　语法检查通过后，可直接运行测试脚本，也可设置断点对脚本进行调试。例如，在如图5.39所示的代码行中设置断点（快捷键是F9）。  
　　

　　图5.39  设置断点  
　　然后，按F5键运行测试脚本，运行过程中将在断点所在的代码行停住，如图5.40所示。



　　图5.40  运行脚本到断点处  
　　5.3.3  单步调试  
　　这时候，可以选择菜单“Debug | Step Over”，或者按快捷键“F10”跳到下一行代码，也可以选择“Debug | Step Into”，或者按快捷键“F11”进入代码行中所调用的函数“Test1”。如图5.41所示。



　　图5.41  单步调试  
　　5.3.4  调试查看器的使用  
　　在调试过程中，可选择菜单“View | Debug Viewer”，在如图5.42所示的界面中，输入测试对象属性或变量，查看其值。

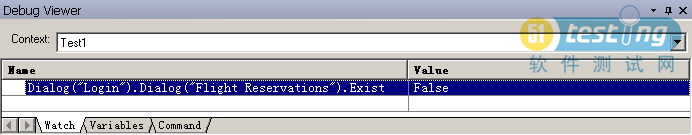


　　图5.42  调试器  
　　如果安装了Visual Studio.NET 2008，则可以增强QTP的调试能力，在“Debug Viewer”中可以查看到对象的大部分属性，如图5.43所示。

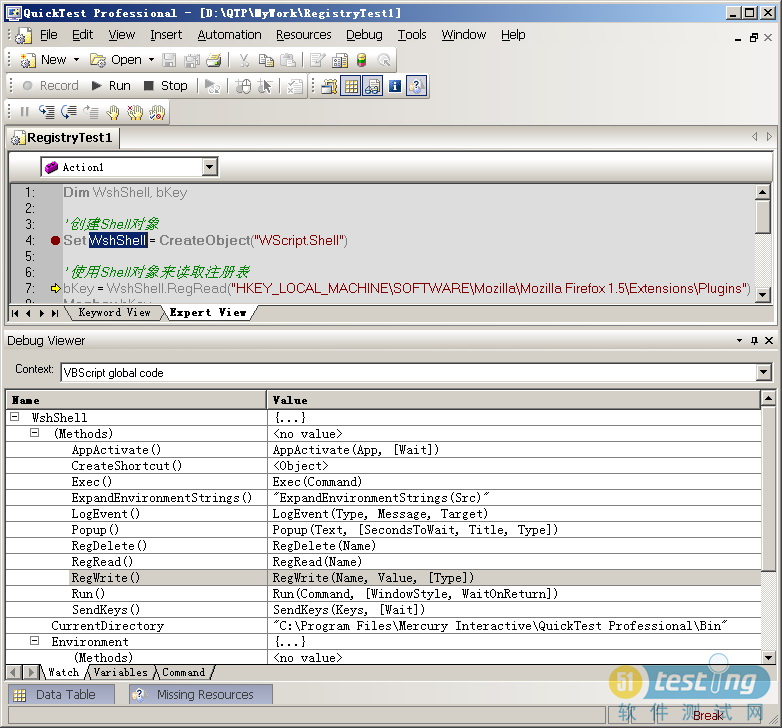
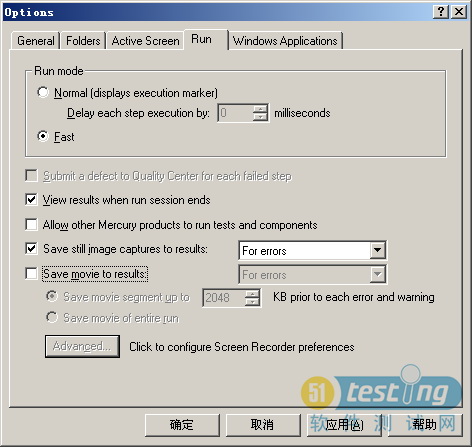


　　图5.43  增强的调试器  
　　技巧：可以通过安装Visual Studio.NET 2008来增强QTP调试能力，也可以不安装，仅仅把其中一个名为PDM.DLL的文件拷贝到“C:\Program Files\Common Files\Microsoft Shared\VS7DEBUG”目录中，然后注册一下即可，注册方法是在命令行中输入“RegSVR32 “C:\Program Files\Common Files\Microsoft Shared\VS7DEBUG\pdm.dll"”。  
　　5.3.5  运行整个测试  
　　对测试脚本进行语法检查和调试都无误后，可以按F5键运行整个测试脚本。在运行测试之前，可以对运行做必要的设置，以便满足测试的要求。选择菜单“Tools | Options”，出现如图5.44所示的界面。



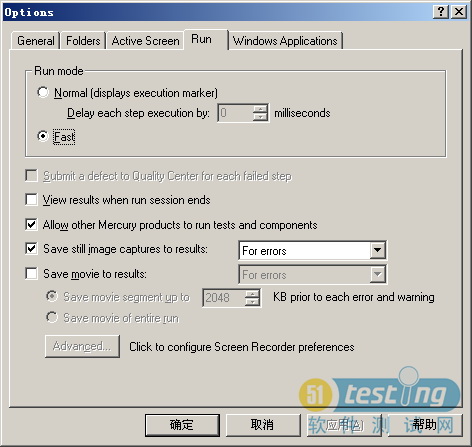
在“Run”页中的“Run mode”，可选择运行模式为“Normal”或“Fast”。如果选择“Fast”，则QTP以尽可能快的速度运行测试脚本中的每一个测试步骤；如果选择“Normal”，则可以进一步设置测试运行过程中每一个步骤直接的停顿时间，这种设置有利于测试人员在QTP执行测试的过程中查看测试的整个过程，看是否如预期的设计一样执行测试。  
　　提示：如果想在测试运行完毕后，自动打开测试结果界面，则需要确保“View results when run session ends”选项被勾选上。  
　　5.3.6  运行部分测试  
　　如果有多个Action，则可以定位到需要运行的Action，然后，选择菜单“Automation | Run Current Action”来仅仅运行当前的Action，这种方式有利于单独运行Action，查看单个Action测试执行的情况，有利于定位当前Action运行的问题。  
　　还有另外一种方式可以用于运行部分的测试，方法是选中某个测试步骤，然后单击鼠标右键，选择菜单“Run From Step”，可以从当前选中的测试步骤开始运行测试，也可以选择“Run To Step”，开始测试并运行到当前所选的测试步骤。  
　　5.3.7  批量运行测试  
　　可以使用QTP自带的工具“Test Batch Runner”来批量运行测试脚本。为了让“Test Batch Runner”可以运行测试脚本，需要在QTP中选择菜单“Tools | Options”，打开如图5.45所示的界面，确保“Allow other Mercury products to run tests and components”选项被勾选上。  
　　

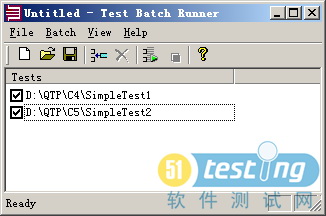
　　图5.45  设置运行属性  
　　然后，通过选择“开始 | 所有程序 | QuickTest Professional | Tools | Test Batch Runner”启动Test Batch Runner，如图5.46所示。  
　　

　　图5.46  Test Batch Runner主界面  
　　在Test Batch Runner中，选择“Batch | Add”来选择并添加需要运行的测试脚本，然后，选择菜单“Batch | Run”来批量运行列表中的所有测试脚本。

（二十）

**5.4  分析**[**测试**](javascript:;)**结果**

[**自动化测试**](javascript:;)的最后一个步骤就是运行测试并查看测试结果，这个步骤也是非常重要的，测试员根据测试结果来判断测试是否通过，检查测试脚本是否正确地完成了测试。

**5.4.1  选择测试运行结果的存储位置**

　　在[**QTP**](javascript:;)中，按F5键运行测试脚本，会出现如图5.47所示的对话框。

　　在这里，可以选择测试运行结果存储的位置，如果选择“New Run results folder”，可以为本次测试选择一个目录用于存储测试结果文件；如果选择“Temporary run results folder”，则QTP将运行测试结果存放到默认的目录中，并且覆盖上一次该目录中的测试结果。

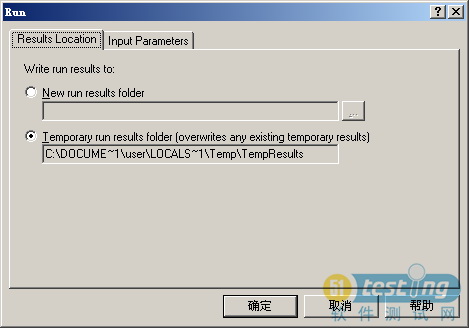


　　图5.47  运行设置对话框

　　提示：如果希望保存每次测试运行的结果，则应该选择“New Run results folder”，如果测试脚本处于调试和检查分析阶段，觉得没必要保存每次运行的测试结果，则可以选择“Temporary run results folder”。

**5.4.2  查看概要测试运行结果**

　　测试脚本运行结束后，可在如图5.48所示的界面中查看概要的测试结果信息，包括测试的名称、测试开始和结束的时间、测试脚本运行的迭代次数、测试通过的状态等。

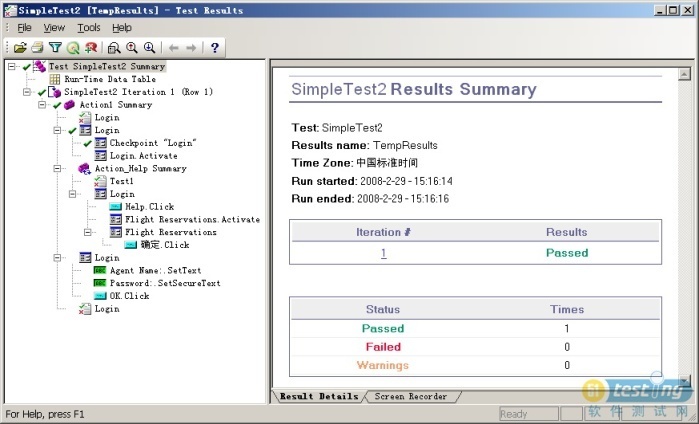


　　图5.48  查看概要测试结果

**5.4.3  查看检查点的结果**

　　在测试结果的左边窗口中，用树型结构展示了所有测试步骤，如果选择节点 Checkpoint “Login”，则可以看到如图5.49所示的检查点结果。

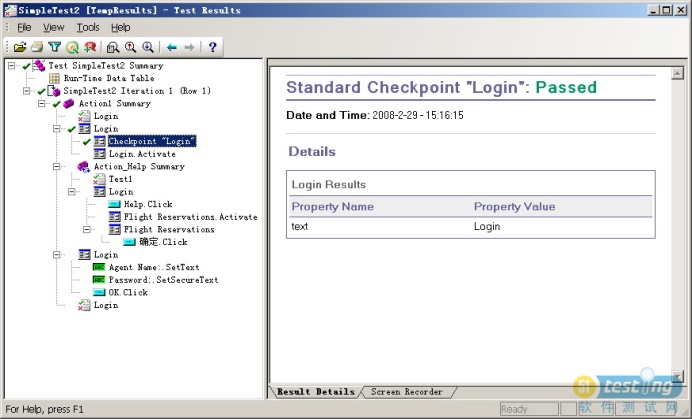


　　图5.49  查看检查点结果

　　检查点检查的是Login窗口的“text”属性是否等于“Login”，在这里可看到，其检查结果是 “Passed”，表示检查通过。

**5.4.4  查看测试过程的截屏**

　　如果设置了运行时保存截屏的选项，则可以在测试结果的“Screen Recorder” 中，查看测试步骤对应的界面截屏，如图5.50所示。

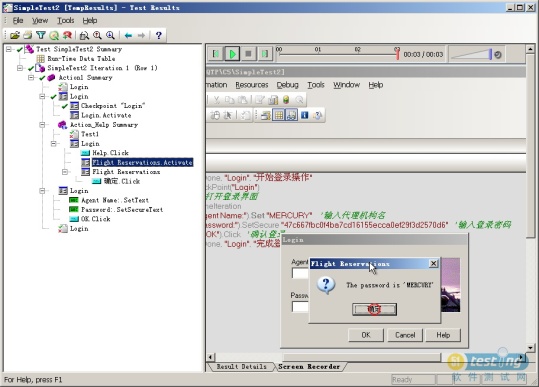


　　图5.50  测试结果的 “Screen Recorder”

　　如果 “Save movie to results”设置为“Always”，则会把所有测试过程的操作录制下来，单击“Play”按钮，就可以像放电影一样把测试过程回放出来。这样可以直观地看到测试的过程，如果测试过程出现问题，则可以查看出现问题前的界面操作，看是什么界面操作或数据的输入导致了错误的出现。

　　屏幕截屏的设置方法是：在QTP中，选择菜单 “Tools | Options”，在如图5.51所示的界面中，把 “Save movie to results”勾选上，并且在下拉框中选择“Always”。

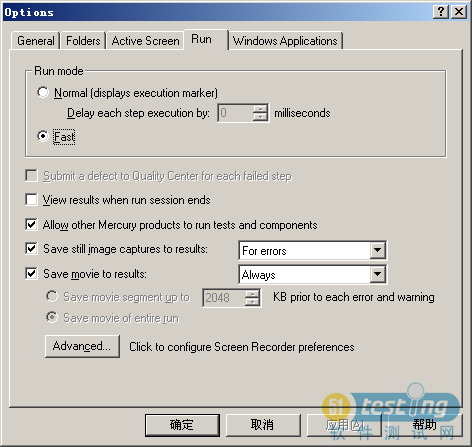


　　图5.51  屏幕截屏的设置

　　注意：选择“Always”将全程截屏，会占用比较大的存储空间。

（二十一）

**5.5  利用Active Screen来编辑**[**测试**](javascript:;)**脚本**

　　Active Screen是[**QTP**](javascript:;)独有的一个功能，可用于录制测试脚本后辅助编辑和维护测试脚本。Active Screen既是测试步骤的截图，可用于直观地描述测试步骤，也是一个测试对象的信息库，存储了测试步骤中测试对象的信息，这些信息可用于添加测试步骤，插入检查点。

**5.5.1  查看Active Screen**

　　Active Screen是在录制过程中，QTP为每一个测试步骤录制的截屏，可通过选择菜单“View | Active Screen”来查看每个测试步骤的Active Screen，如图5.52所示。

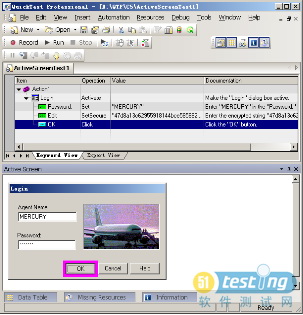


　　图5.52  查看每个测试步骤的Active Screen

　　在“Active Screen”中显示的截屏对应的是，单击“OK”按钮的测试步骤。Active Screen的截图可用于后期添加测试步骤、插入检查点等[**工作**](javascript:;)。

**5.5.2  Active Screen的设置**

　　基于Active Screen的截屏所对应的界面，可以在录制完成后参数化对象的值、插入检查点、输出值，即使是在应用程序未被启动和激活的情况下。这是因为Active Screen存储了测试界面相关的信息。可以调整Active Screen截屏时存储测试界面相关信息的粒度。在QTP中，选择菜单“Tools | Options”，出现如图5.53所示的界面。

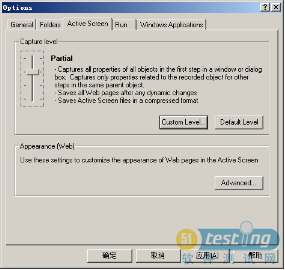


　　图5.53  调整Active Screen截屏时存储测试界面相关信息的粒度

　　在“Capture Level”中，可以设置Active Screen捕获信息的级别，包括4个级别：

　　（1）Complete：捕获和存储所有信息，包括每个步骤中应用程序的所有激活的窗口、对话框、[**Web**](javascript:;)页面等。

　　（2）Partial：这是默认设置，捕获和存储第一个步骤所执行的应用程序窗口，以及该窗口下属的对象属性。

　　（3）Minimum：仅捕获和存储录制的对象及其父亲对象的属性。

　　（4）None：屏蔽Active Screen捕获和存储对象属性的功能。

　　存储越多的信息，对于后期编辑测试脚本而言，会更加容易，但是存的信息越多，录制过程则会越慢，而且需要更多的存储空间。

（二十二）

**5.5.3  在Active Screen中选择并添加对象到对象库**

[**QTP**](javascript:;)在录制脚本时，仅会将[**测试**](javascript:;)步骤所涉及的对象添加到对象库，例如，录制Flight程序的登录功能时，如果是正常路径，则仅会涉及4个对象，如图5.54所示。

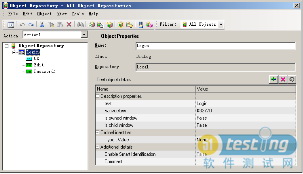


　　图5.54  对象库

　　如果想在后期添加对象，除了可以再次打开应用程序的登录界面，在对象库中添加所需的测试对象外，还可以利用Active Screen捕获的界面信息来把测试对象添加到对象库中去。例如，可在如图5.55所示的Active Screen界面中，把鼠标移到“Help”按钮所在的位置。



　　图5.55  Active Screen界面

　　然后，单击鼠标右键，选择“View/Add Object”，则会出现如图5.56所示的界面。



　　图5.56  选择对象

　　单击“OK”按钮，出现如图5.57所示的对象属性界面。

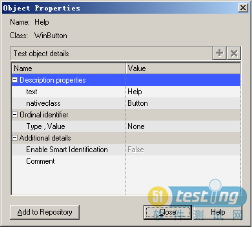


　　图5.57  对象属性界面

　　在这个界面中，显示出了“Help”按钮的对象属性。单击“Add to Repository”按钮，可把“Help”按钮添加到对象库中，按钮变成“View in Repository”，如图5.58所示。

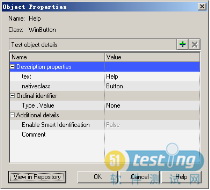


　　图5.58  查看对象属性

　单击“View in Repository”按钮，可看到“Help”按钮已经被添加到对象库中，如图5.59所示。

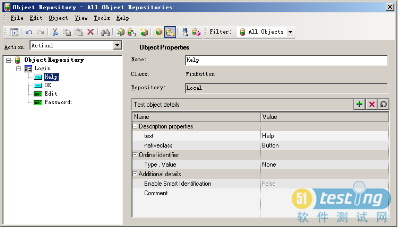


　　图5.59  “Help”按钮被添加到对象库

**5.5.4  选择Active Screen中的对象并输出其对象属性**

　　可以在Active Screen中选择某个对象并在测试结果中输出其对象属性值，例如，如果想在如图5.60所示的Active Screen所对应的测试步骤之前，添加一个测试步骤用于输出“Static”对象的某些属性值，则可以把鼠标移到图片所在的位置。

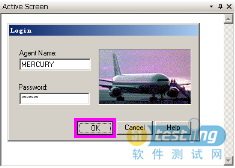


　　图5.60  Active Screen界面

　　单击鼠标右键，选择“Insert Output Value”，出现如图5.61所示的界面。

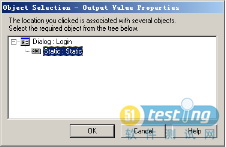


　　图5.61  对象选择

　　单击“OK”按钮，则出现如图5.62所示的界面。



　　图5.62  编辑输出值属性

　　在这个界面中，选择需要输出的属性，然后，单击“OK”按钮，则会插入一个输出对象属性值的测试步骤，如图 5.63所示。

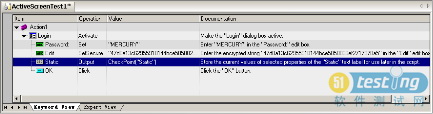


　　图5.63  插入输出对象属性值的测试步骤

　　运行测试脚本，可得到如图5.64所示的结果。

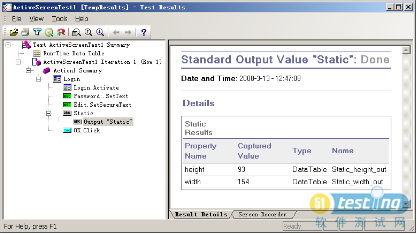


　　图5.64  测试结果

（二十三）

**5.5.5  在Active Screen中插入检查点**

　　可以在Active Screen中选择某个对象并为其插入检查点，例如，如果想在单击“OK”按钮之前插入一个位图检查点，则可以在如图5.65所示的Active Screen中，把鼠标移到图片所在的位置。

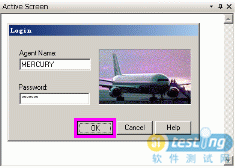


　　图5.65  Active Screen界面

　　然后，单击鼠标右键选择“Insert Bitmap Checkpoint”，则出现如图5.66所示的界面。



　　图5.66  选择对象

　　在这个界面中，单击“OK”按钮，出现如图5.67所示的界面。

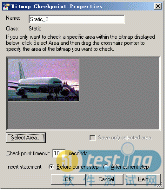


　　图5.67  编辑位图检查点的属性

　　在这个界面中，可选择位图的区域，选择检查点超时的时间，选择在当前[**测试**](javascript:;)步骤之前添加检查点还是在当前测试步骤之后添加。然后，单击“OK”按钮，得到如图5.68所示的测试步骤。

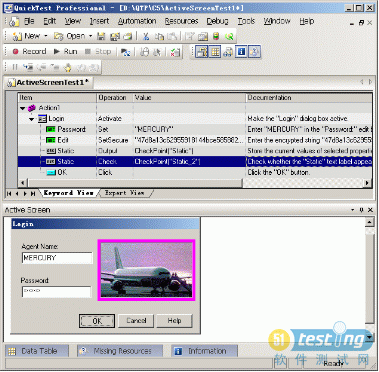


　　图5.68  成功添加测试步骤

**5.5.6  在Active Screen中选择对象并添加测试步骤**

　　可以在Active Screen中选择某个对象并为其添加一个测试步骤，例如，如果想在单击“OK”按钮之前，先单击“Help”按钮，则可以在如图5.69所示的Active Screen中，把鼠标移到“Help”按钮所在的位置。

　　然后，单击鼠标右键选择“Step Generator”，则出现如图5.70所示的界面。

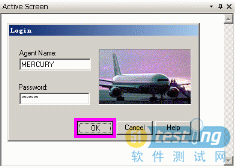
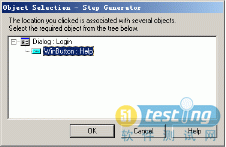
    

　　图5.69  Active Screen界面                      图5.70  选择对象

　　在这个界面中，单击“OK”按钮，出现如图5.71所示的界面。

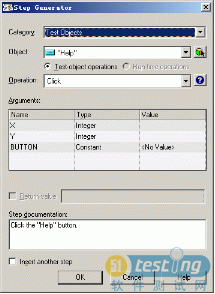


　　图5.71  “Step Generator”界面

　　在这个界面中，选择“Category”为“Test Objects”，在“Object”中选择“Help”，在“Operation”中选择“Click”，然后，单击“OK”按钮，则可添加一个测试步骤，如图5.72所示。



　　图5.72  添加一个测试步骤