

接口测试工具-SoapUI

修订记录

日期	修订版本	修订内容	作者
101107	0.1	初稿拟制（介绍最常用的功能，部分内容翻译自官方文档）	张蓉
101128	0.2	新增第 7 章内容-实际测试中最常用的场景	张蓉

目录

1. SOAPUI 的介绍.....	5
2. SOAPUI 的安装.....	5
2.1. SOAPUI 下载.....	5
2.2. SOAPUI 安装.....	6
3. 名词解释.....	9
3.1. 基础知识.....	9
3.2. 附加名词解释.....	10
4. 应用场景.....	11
4.1. 导入项目.....	11
4.2. WEB SERVICE MOCKING.....	12
4.3. WEB SERVICE INSPECTION.....	15
4.4. 对 WEB SERVICE 服务进行功能测试.....	17
4.5. 对 WEB SERVICE 服务进行负载测试.....	20
5. 基础讲解.....	22
5.1. 创建工程.....	23
5.2. 测试结构的组织和执行.....	30
5.3. 测试步骤.....	34
5.4. 创建功能测试用例.....	36
5.5. 功能测试断言判断.....	38
5.6. 功能测试用例执行.....	40
5.7. 创建负载测试用例.....	41
5.8. 执行负载测试用例.....	42
5.9. 负载测试断言判断.....	43
5.10. 创建 MockService.....	44
5.11. 编辑 MockService.....	45
5.12. 调用 MockService.....	46
5.13. 自定义模拟响应.....	47
6. 操作技巧.....	50
6.1. 右键点击.....	50
6.2. 测试的管理.....	50
6.3. 命名建议.....	50
6.4. 共享操作管理-SVN.....	51
6.5. 属性操作.....	53

6.6. 接口变化.....	56
6.7. 数据库操作.....	60
6.8. 数据文件操作.....	63
6.9. 循环入参.....	66
6.10. 流程控制.....	69
6.11. 脚本处理.....	73
6.12. 数据初始化-清理.....	74
6.13. 断言操作.....	76
6.14. 定时保存.....	84
6.15. 响应报文.....	85
6.16. 日志查询.....	86
6.17. 导入和检查项目.....	86
6.18. 发布测试报告.....	87
6.19. 加密项目.....	90
7. 测试场景的应用.....	91
7.1. 引入 JAR 包--读取数据源属性.....	91
7.2. 调用 GROOVY 工具类.....	92
7.3. 响应报文处理.....	94
7.4. 动态定位表名.....	96
8. 测试工具的简单对比.....	96

1. SoapUI 的介绍

由于 Web 服务是被程序调用的，一般不会提供界面让最终用户或测试人员直接使用，在 SoapUI 等工具出现之前，测试人员不得不自己编写程序来测试它，这就要求测试人员花费很大的精力了解底层的接口，调用关系和详细的协议，导致他们不能把注意力集中到测试中。

SoapUI 的出现极大的改变了这一局面。作为一个开源的工具，SoapUI 强大的功能、易用的界面，吸引了很多用户。用户可以在 SoapUI 中通过简单的操作完成复杂的测试，不需要了解底层的细节，极大的减轻了工作量。SoapUI 支持多样的测试，例如功能测试，负载测试，回归测试等。到目前为止 SoapUI 的下载量已经超过了 100 万次，成为了 Web 服务测试标准和领先的 Web 服务测试工具。它不仅仅可以测试基于 SOAP 的 Web 服务，也可以测试 REST 风格的 Web 服务，前者是本文介绍的重点。

2. SoapUI 的安装









2.1. SoapUI 下载

SOAPUI 的网站地址是：<http://www.eviware.com/>，网站提供了多种 soapui 安装包的下
载，SoapUI 主要提示三种不同的版本，分别是：

- (1) SoapUI pro 是收费版本，拥有强大的功能，本文主要围绕着这个版本的功能进行阐述，该版本的下载地址是：

<http://www.eviware.com/Download-soapUI/download-soapui-pro.html>

Latest Release
soapUI PRO 3.6.1

 Windows 32-bit installer	» Download
 Linux installer	» Download
 Mac installer	» Download
 Windows binary zip	» Download
<hr/>	
 Windows 64-bit installer (no JRE)	» Download
 Windows binary zip (no JRE)	» Download
 Linux binary zip (no JRE)	» Download
 Mac binary zip (no JRE)	» Download

- (2) SoapUI Trial 是试用版本，拥有的功能和 SoapUI pro 一样，但这个版本在申请成功后仅有 14 天的试用期，建议在开始读本书的时候，可以申请一个试用版进行学习，该版本的下载地址是：

<http://www.eviware.com/Download-soapUI-Trial.html>

Download soapUI Trial

Thank you for your interest in soapUI Pro

In order to obtain an Trial License for soapUI Pro you need to fill out the form below. The information provided will only be available to eviware employees.

IMPORTANT: We will send a license to the e-mail address supplied; please make sure that it is correct. If you have not received a license in 24 hours, please contact eviware.

webmaster@eviware.com

Register for a 14 day soapUI Pro trial license:

E-mail address:

Enter your name:

Company:

After registration you will be redirected to a download page and the License will be emailed to the specified address. Downloading a trial signifies your agreement to eviware contacting you occasionally with promotional material.

- (3) SoapUI 是开源的版本，这个版本我也曾经使用过，这个版本提供的功能较少，但由于它开源，只要你有兴趣，可以到网站下载源代码，根据需要对源代码进行修改，改造成你希望的方式。对于这个版本，如果你熟悉 groovy 的编写，那么在测试过程可以使用 groovy 脚本来丰富你的用例，该版本的下载地址是：

<http://sourceforge.net/projects/soapui/files/>

SourceForge.net > Find Software > soapUI > Browse Files



soapUI by [ebonsu](#), [nreimertz](#), [omatzura](#)

[Summary](#) [Files](#) [Support](#) [Develop](#)

soapUI is the leading desktop application for inspecting, invoking, monitoring, simulating/mockng and functional/load/compliance/surveillance testing of REST/WADL and SOAP/WSDL-based Web Services over HTTP.

Download Now!

soapUI-x32-3.6.1.exe (110.2 MB)



OR

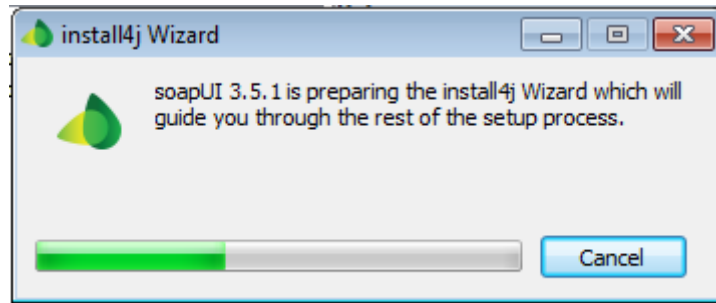
[View all files](#)

Browse Files for soapUI

File/Folder Name	Platform	Size	Date ↓	Downloads	Notes/Subscribe
Newest Files					
com-eviware-soapui-netbeans-module.nbm		37.3 MB	2010-10-21	34	
soapui-idea-plugin-3.6.1.zip		91.4 MB	2010-10-21	18	
soapui-idea-plugin-3.6.1-src.zip		191.1 KB	2010-10-21	4	
soapui-eclipse-plugin-3.6.1.zip		89.4 MB	2010-10-21	53	

2.2. SoapUI 安装

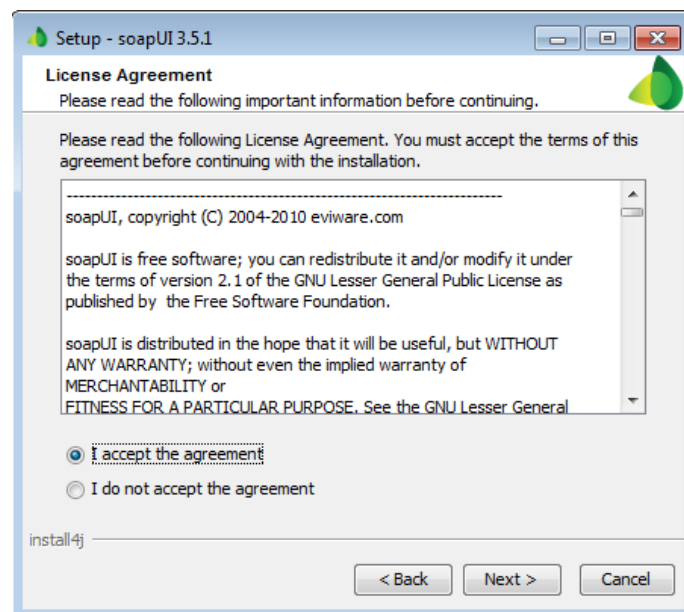
SoapUI 基于 Java 开发，支持多个平台，需要设置 JAVA_HOME 变量指向到相应的 JRE 目录，同时修改 PATH 变量，将 JRE1.6 的 bin 目录添加进去。安装的过程很简单，只要双击已成功下载的 exe 程序，即可开始安装过程：



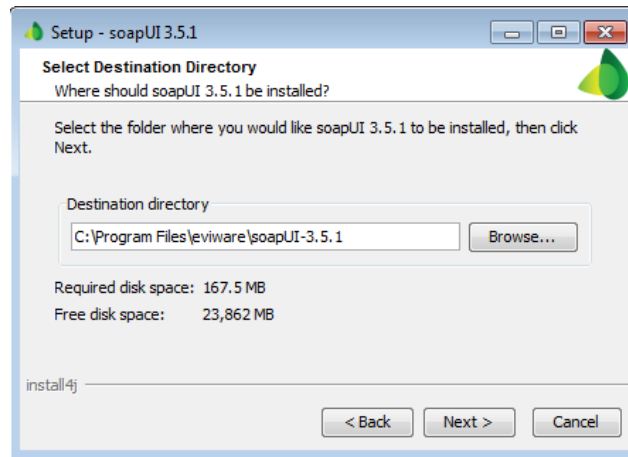
在安装向导完成后，很快可以看到下面的开始对话框：



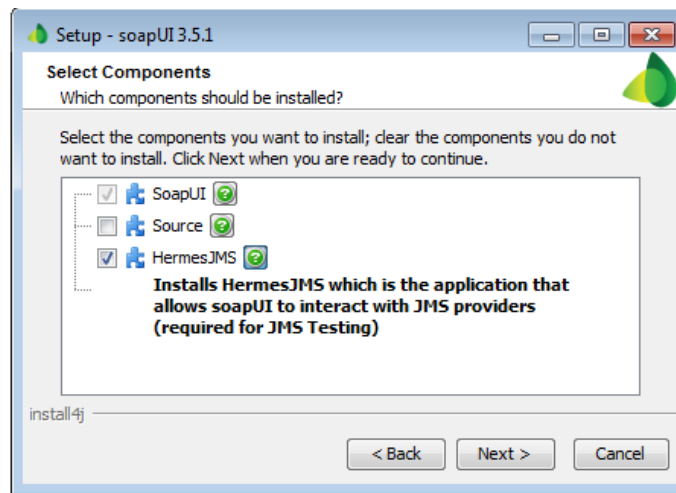
点击 Next 进入下一步，你需要阅读 soapui 要求的协议，并接受协议才能进入下一步：



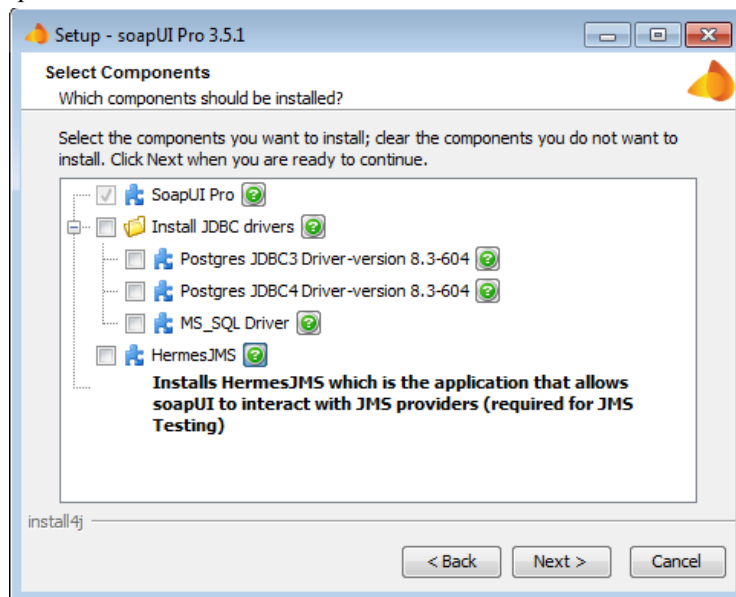
选择接受，并点击 Next 进入下一步，这里要求设置程序安装的路径，soapui 会给出默认路径，如果你不选择，则会默认安装在 C:\Program Files\eviware\soapUI-3.5.1：



接下来，可以根据需要选择要安装的组件：

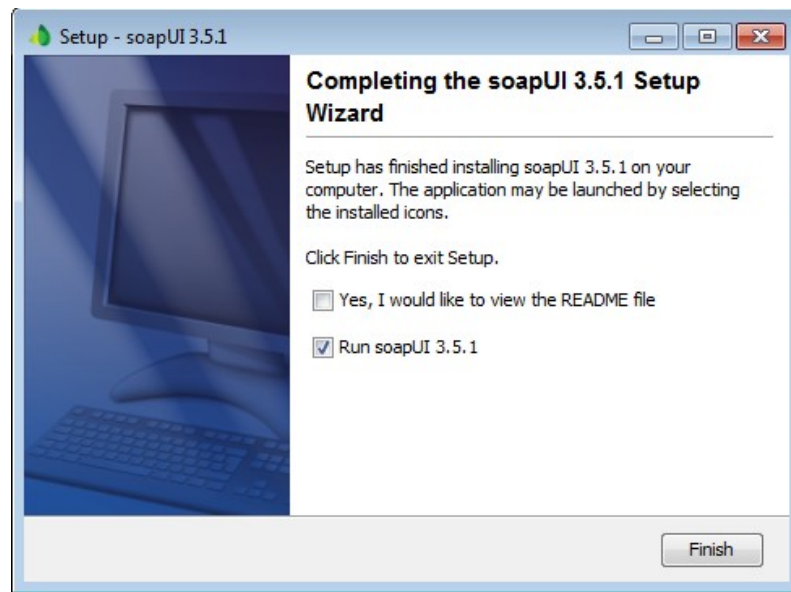


如果是 soapUI Pro 版本，JDBC 的驱动包会提供选择，但是没有源代码可勾选：



如果勾选了 HermesJMS，那么下一步还需要接受 HermesJMS 的协议，接下来只需要默认安装即可，直到出现下面的页面，点击 Finish 完成全部安装过程，接下来你可以开始学

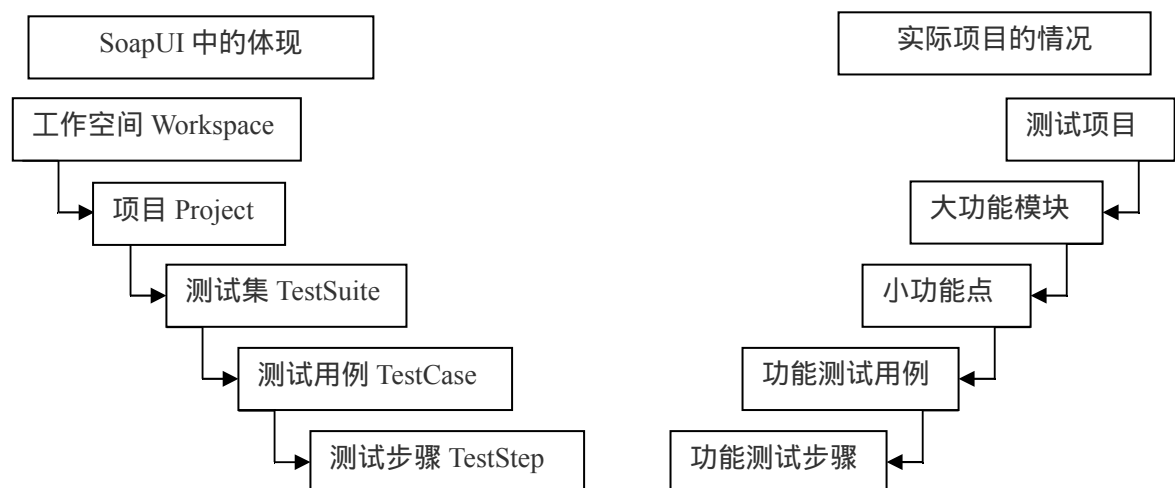
习和使用的过程了。



3. 名词解释

3.1. 基础知识

在 SoapUI 的各种版本里，我们都能深刻地体会到它所提供的测试用例的管理与我们测试项目所需要的层级的映射关系是相当贴近的，对应关系图如下：



SoapUI 里着重引入了以下的概念：

- ✓ 测试步骤 `Test Step`
测试步骤是最小的单位，一个完整的测试用例是由多个测试步骤所组成的，而每一个测试步骤，都需要根据实际的业务要求进行组织。
- ✓ 测试用例 `Test Case`
一个测试用例代表一个完整操作，接口测试的目的，实质在于模拟外部的调用来验证

接口的功能，而接口功能的各个分支则由入参（测试数据）的不同来遍及。

- ✓ 测试集 `Test Suite`

对于测试集，主要是为了区分大功能模块里的不同小功能点而引入的概念，一般一个 `WebService` 都包含有多个接口，此处可根据需要添加测试集。

- ✓ 项目 `Project`

在 SoapUI 里，一个接口对应一个项目（`Project`），这是由 SoapUI 提供的功能所决定的，在每次要测试一个新的接口时，可以右键点击 `WorkSpace` 的名称，从右键菜单中选择 `New soapUI Project` 来引入新的 WSDL。

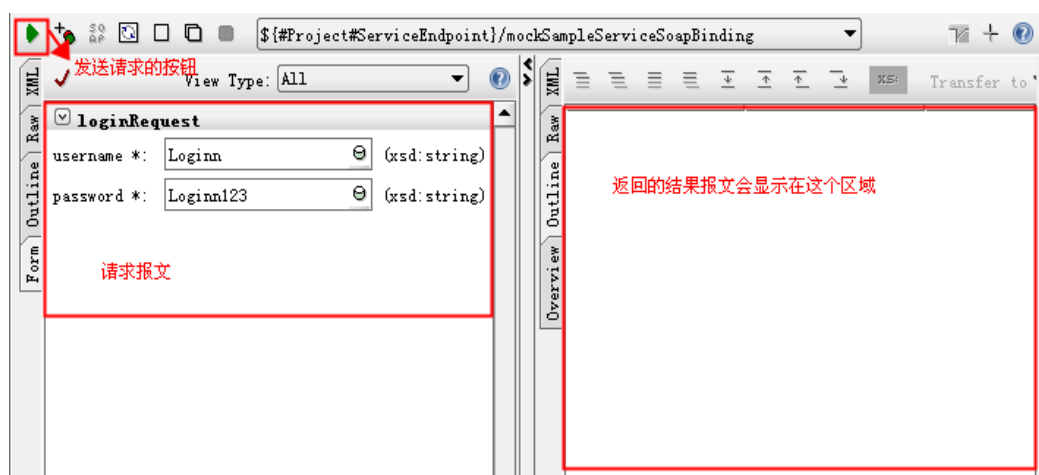
- ✓ 工作空间 `Work space`

对应测试项目的概念，一个测试项目中可能会包含多个 `WebService` 接口，这些接口都同属于一个项目中，由工作空间来管理所有的接口项目。

3.2. 附加名词解释

- ✓ `WSDL`：指网络服务描述语言（`Web Services Description Language`），是一种使用 XML 编写的文档。这种文档可描述某个 `Web service`。它可以规定服务的位置，以及此服务提供的操作（或方法），更加详细的内容可以参考：
http://www.w3school.com.cn/wsdl/wsdl_documents.asp

- ✓ `Request`：SoapUI 通过导入正确的 WSDL，便可以解析接口需要的入参。当对特定的接口创建一个请求时，SoapUI 会帮我们把需要的 SOAP 报文的结构以一定的形式显示出来，此时你只需要输入请求的内容，点击运行，SoapUI 会将我们填写后的 SOAP 报文完整地发送给远程服务接口进行调用。



- ✓ `MokeService`：SoapUI 同样也提供了 Moke 服务的功能，在初期确定完接口出入参、接口名时，我们便可通过 SoapUI 的 Moke 服务功能，人为地模拟系分文档中明确要求的各种情况下的返回参数（根据返回参数的类型编写对应的 SOAP 报文），使测试人员在早期就进行接口测试用例的编写和调试。

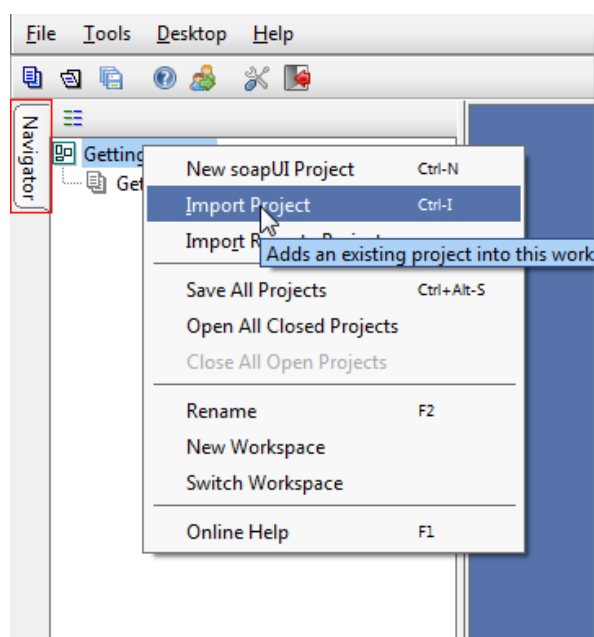
4. 应用场景

无论一个新的应用是复杂还是简单，在开始学习时经常会有一定的困难。为了让你能更容易地开始使用 SoapUI，这边直接引用了 SoapUI 一个经典的例子进行讲解。该例子主要阐述了 SoapUI 的一些基本概念，可以用来作为学习 SoapUI 的出发点，尝试了解该项目，包括 MokeService 模拟服务和测试用例，以些来熟悉 SoapUI 的界面和操作。在这个项目中主要阐述了 SoapUI 的 5 个功能：

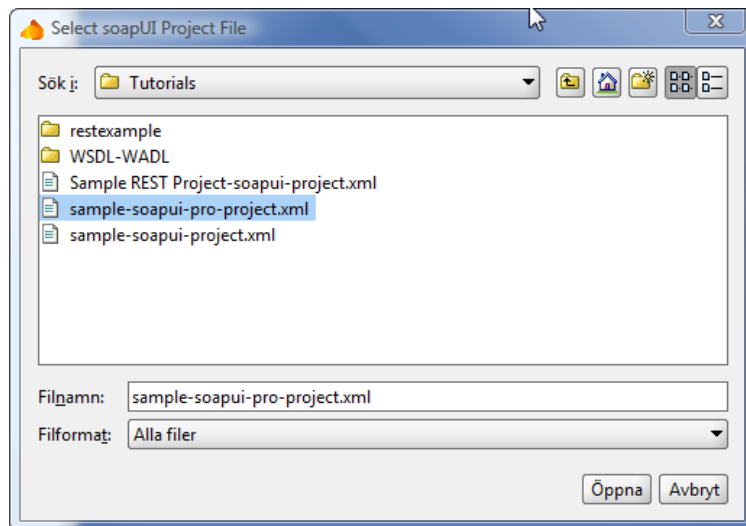
- ✓ 导入项目
- ✓ Web Service Mocking
- ✓ Web Service Inspection
- ✓ 对 Web Service 服务进行功能测试
- ✓ 对 Web Service 服务进行负载测试

4.1. 导入项目

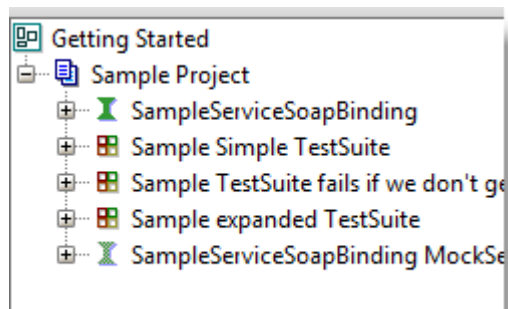
- 右键点击在 Navigator 中的项目节点，选择“Import Project”选项，导入已经存在的项目，此时页面会弹出“Select soapUI Project File”对话框。



- 从安装 SoapUI 的文件夹下的 Tutorials 目录里，选择项目名称为 sample-soapui-project.xml or sample-soapui-project.xml 的项目文件。



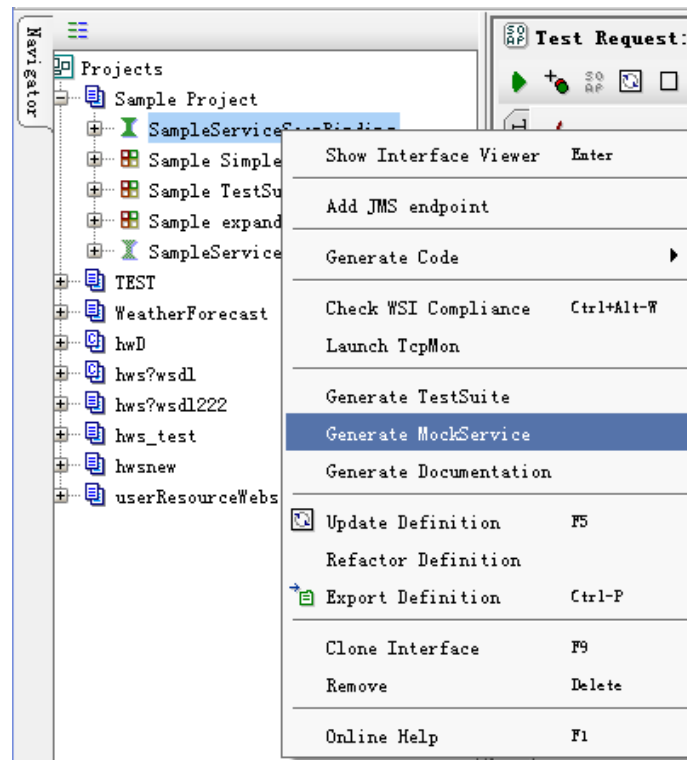
- 名为“Sample Project”的项目将在 SoapUI 的左边栏目显示。此时，我们已经成功地导入项目，接下来开始介绍模拟 Web Service 服务在 SoapUI 是怎么做的。



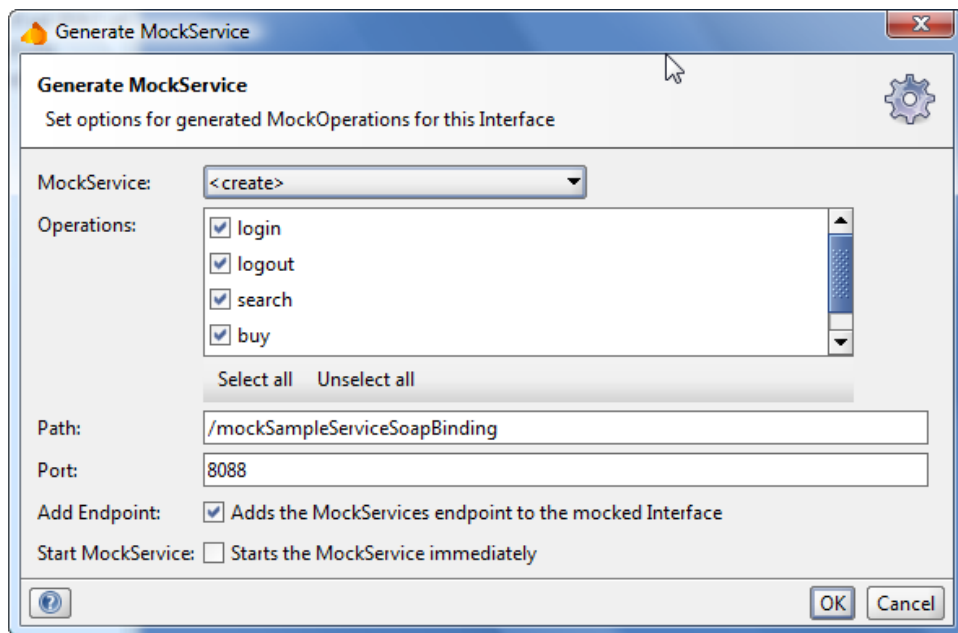
4.2. Web Service Mocking

Web Service Mocking 是在 SoapUI 中伪造或模拟 Web Service 服务接口功能的一个途径，为什么我们需要这样做呢？当 Web Service 服务接口没有启动或接口的编码尚未完成等各种原因导致接口暂时无法使用时，Web Service Mocking 是非常有用的一个功能。总而言之，Web Service Mocking 让你可以在开发开始编码的同时创建你的测试用例，这意味着，当真正的 Web Service 服务接口可以开始测试时，你的用例设计工作可能已经完成，可以直接开始测试。

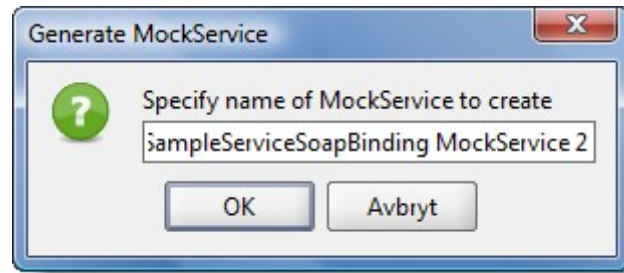
- 添加一个 MockService，右键单击接口集合的名称，如下图所示：



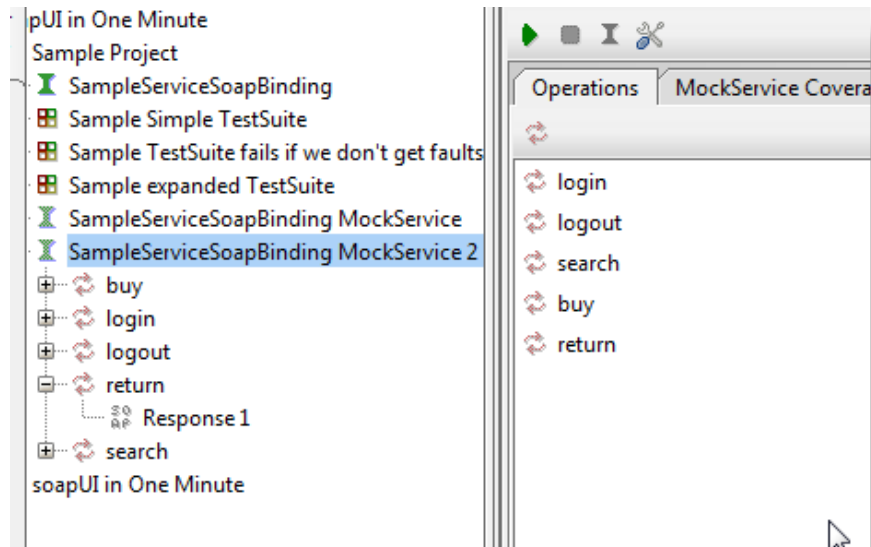
- 弹出“Generate MockService”对话框，可直接使用默认设置的值。



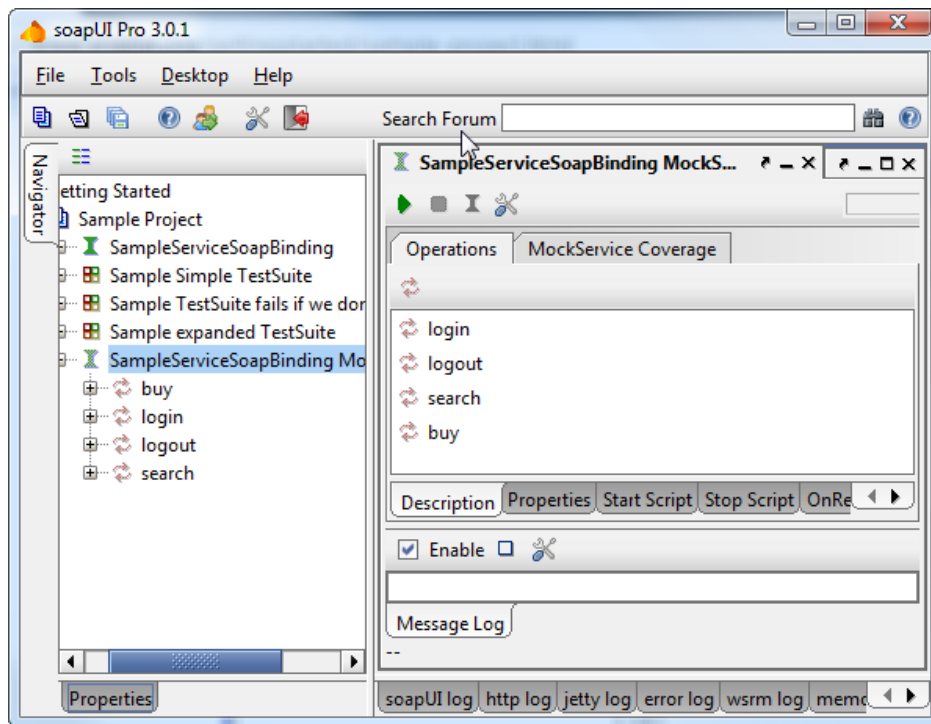
- 输入 MockService 的名称，点击“OK”按钮。




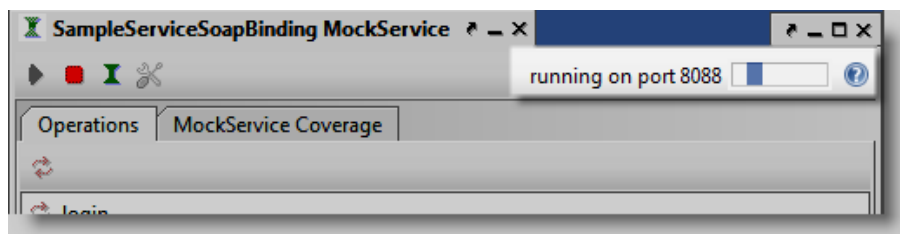
- 经过上面的操作，MockService 已经成功生成，接下来你可以选择如何去响应请求。



- 首先让我们一起看一下，如何让 MockService 运行起来。导入的文件中，已经包含名为“SampleServiceSoapBinding Mock Service”的 MockService，所以接下来，你可以先删除掉你刚才所创建的“SampleServiceSoapBinding Mock Service 2”，打开 MockService 只要双击它即可。



- 在 MockService 中我们可以看到有不同的接口：login, logout, search, buy 共 4 个接口，在例子项目中，你可以看到，所有的响应均采用 SCRIPT 的调度方式进行分发响应，这是最常见的一种调度方式，这里还提供顺序等其它方式供测试人员选择。
- 左键单击执行按钮  开始运行 MockService，现在你可以看到 MockService 运行在 8088 端口。

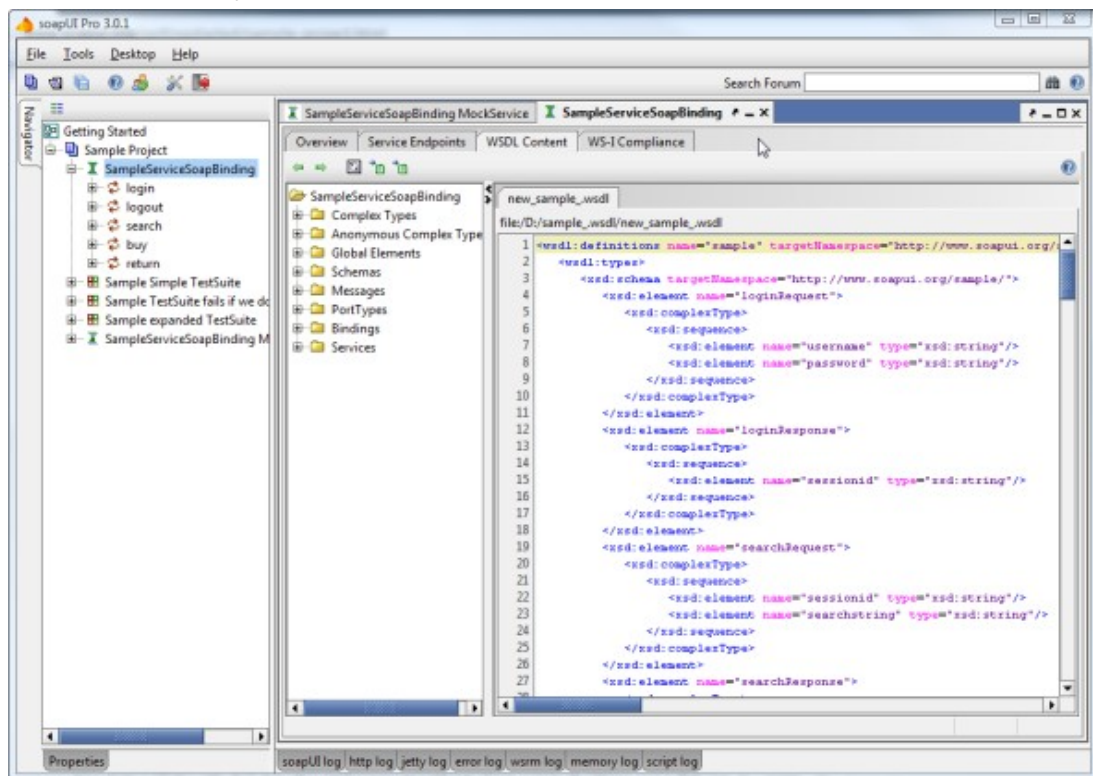


4.3. Web Service Inspection

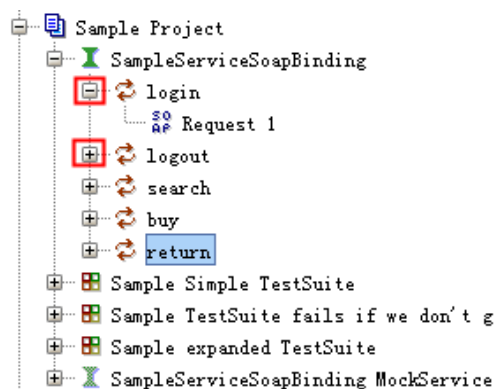
Web Service Inspection 是一个非常好的功能，它能够让你在测试开始执行时就能很容易地了解到你所测试的 Web Service 接口是如何工作，需要什么样的入参才能调用。


- 直接浏览 WSDL 接口的 XML 报文，都是很杂乱的，而且一般 WSDL 报文也都比较复杂，很难直观地看出，因此很少人会这样做，而进一步导致测试人员不想去理解 WSDL。总而言之，由于 WSDL 的复杂而使人们不愿意去读懂它，但其实 WSDL 是规范的，并且你怎么使用它将取决于你对他的认识，而 SoapUI 的接口视图模式是解决这个问题非常好的工具。
- 通过双击“SampleServiceSoapBinding”节点，SoapUI 页面上会打开该节点的信息，切换至 WSDL Content 标签页，如下所示，可通过此处的内容与右边栏“SampleServiceSoapBinding”节点下方的五个接口映射，以此来加深对 WSDL 的接

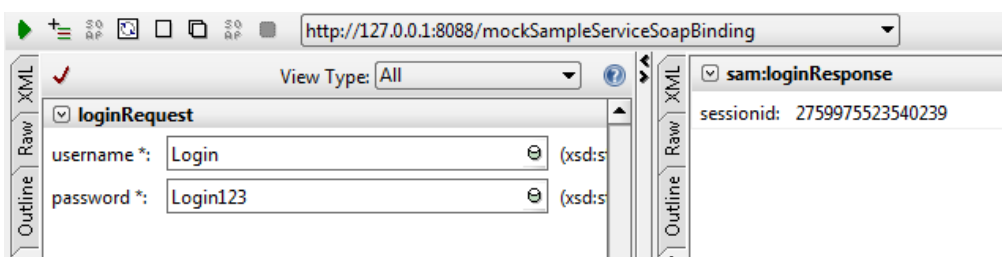
口的理解。



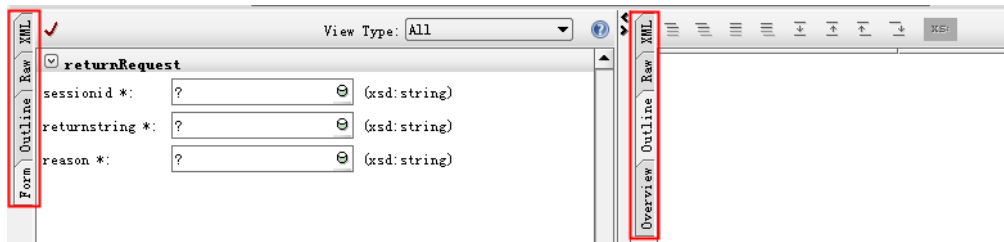
- 单击“SampleServiceSoapBinding”节点下方的接口“login”，展开接口对应已有的请求“Request 1”，双击“Request 1”打开请求页面，你将可以在 SoapUI 的桌面上看到“loginRequest”。



- 请求中已经写明了入参：username = Login，password = Login123，点击  按钮提交请求，成功后，可以在右边的“sam:loginResponse”框中看到响应报文。



- 你可以使用不同的显示方式，查看请求报文和响应报文的内容，只需要点击左边的 TAB 标签页即可。



4.4. 对 Web Service 服务进行功能测试

在例子中，已经创建了不同的测试集，测试集包含了不同的用例，一个测试用例又由一个或多个测试步骤组成，SoapUI 会组织成如下的结构：

Project

|-Interface

|-TestSuites

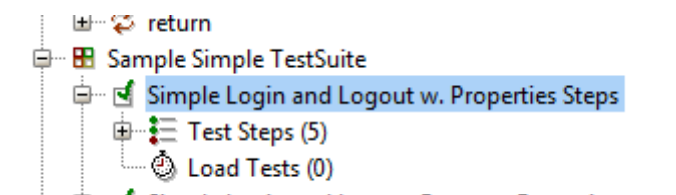
|--TestCases

---TestSteps

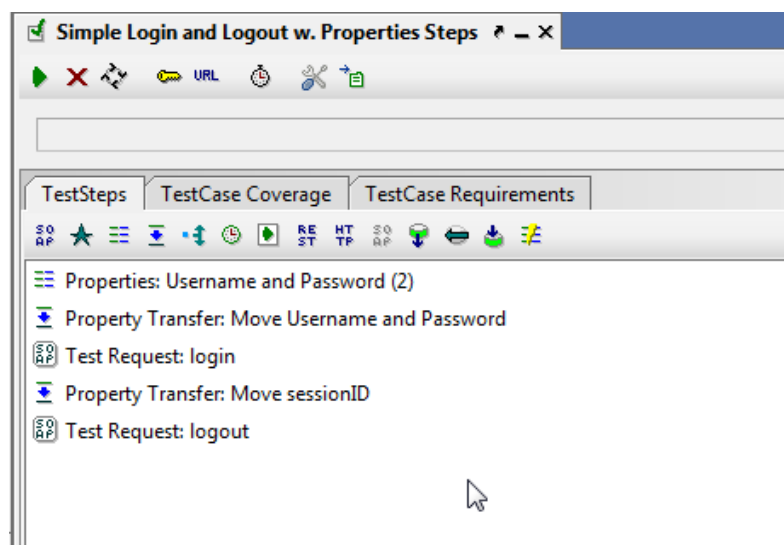
---LoadTests

|-MockServices

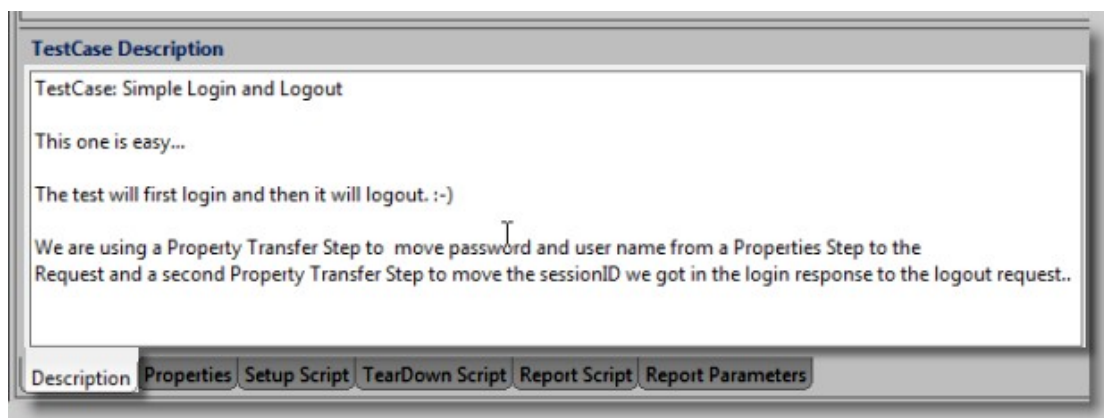
- 通过点击“+”展开“Sample Simple TestSuite”，双击“Simple Login and Logout w. Properties Steps”。



- 你可以看到该测试用例由 5 个测试步骤组成。



- 可以点击“Description”标签查看测试用例描述。



- 该测试用例主要有 3 种类型的测试步骤组成：一个 Properties 测试步骤，两个 TestRequests 和两个 PropertyTransfer 步骤，他们的功能如下：

·Properties：

用来保存属性的值，后面的步骤都可以使用，在用例里“login”接口的入参就是使用属性“Username”和“Password”。

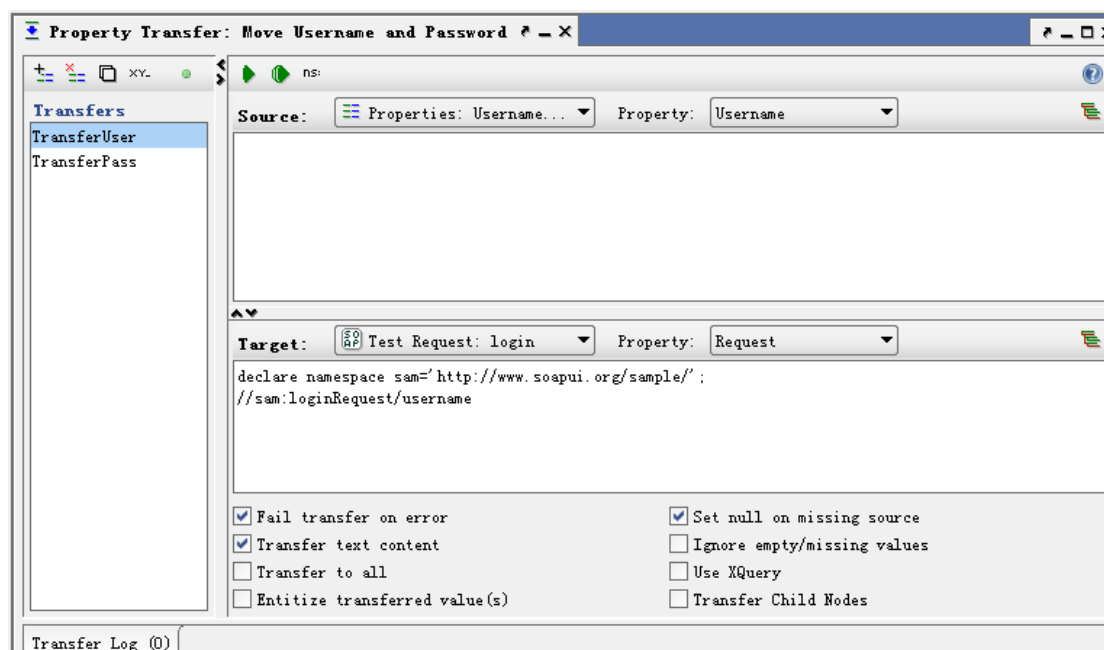
·TestRequests：

发给服务接口的请求，在用例中的请求有“login”和“logout”。

·PropertyTransfer：

一个用来在不同测试步骤间传递属性的步骤，这可以使用的场景有从一个 Properties 的步骤将属性传给一个 TestRequests 的步骤，如用例中：“Move Username and Password”步骤，也可以是将属性从一个请求的出参传递给另一个请求的入参，如用例中：“Move sessionID”步骤。

- 双击测试步骤：“Property Transfer: Move Username and Password”，测试步骤的编辑器将会在 SoapUI 上打开，这个步骤传递了“Username”和“Password”两个属性。



- SoapUI 一个最重要的特点是：断言，断言的使用是为了验证发出的请求是否被正确的响应，校验与预期结果是否一致。打开测试步骤：“Test Request: logout”。在下面的测试步骤中，我们可以看到 4 个断言：

·SOAP Response:

验证响应报文是一个有效的 SOAP 响应。

·Schema Compliance:

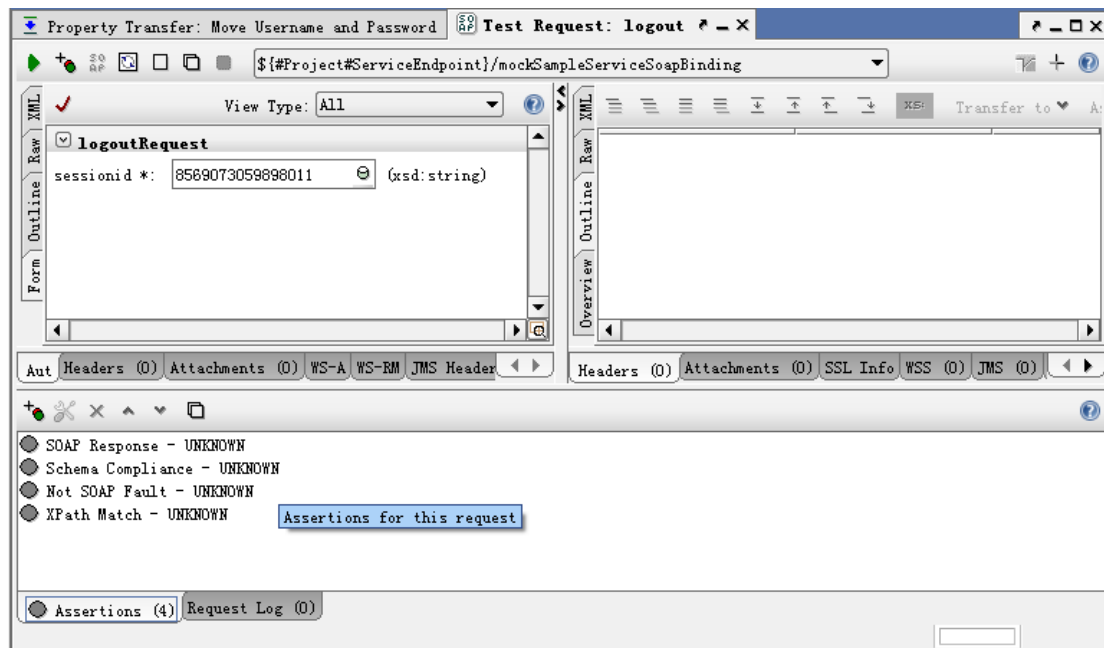
指返回的报文是否符合 WSDL 中所定义的模式。


·Not SOAP Fault:

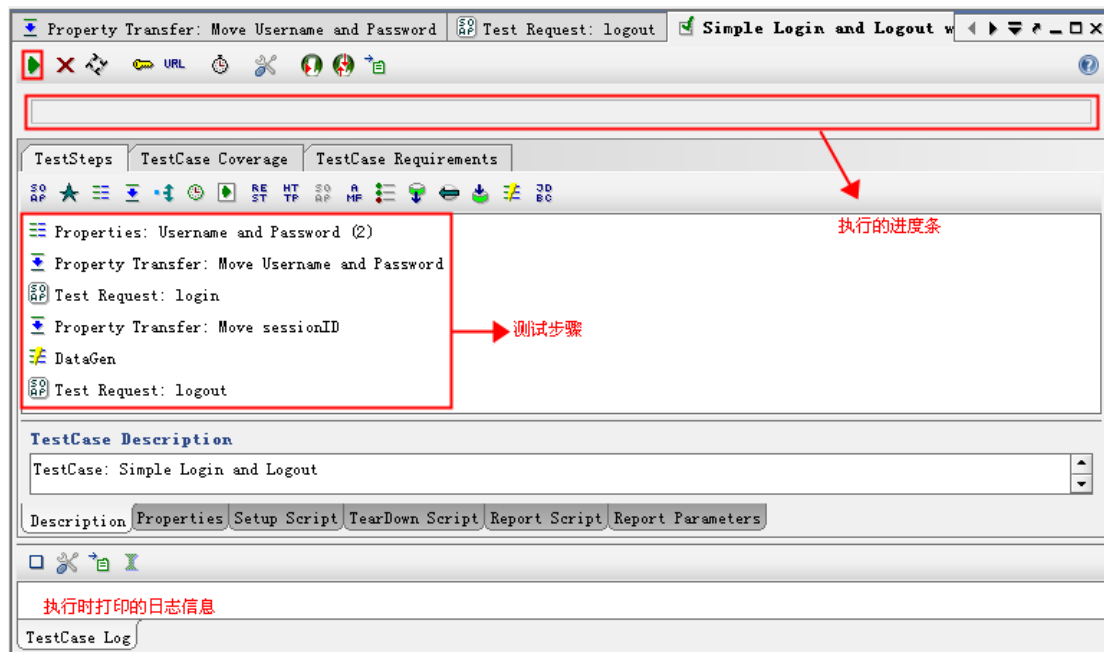
检查响应报文没有包含 SOAP 异常。

·XPath Match:

比较 XPATH 表达式所指定位置上的元素是否与预期值符合。



- 测试执行：在上面我们已经大致了解了整个测试用例，接下来我们可以执行用例了，点击执行按钮 ，测试用例将直接执行，测试结果将会显示在下图所示的执行进度条上，如果完全通过此进度条会显示成绿色，如果执行不通过，会停在失败的执行步骤上，进度条显示成红色，而下方会打印相应的错误日志信息。
注：如果执行时进度条显示成红色，请检查一下你的 MockService 有没有开启，或者在执行完成的测试用例之前，你是否已经执行过“login”请求？如果执行过，请使用“logout”清除掉服务端的会话信息或重启 MockService，再重新执行测试用例。

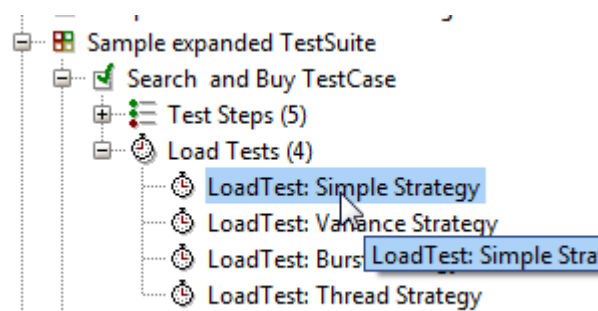


4.5. 对 Web Service 服务进行负载测试

在创建完测试用例后，SoapUI 可以快速地让你创建用例对应的负载测试用例，这是非常实用的功能，越早能够进行负载测试，越早能发现性能问题。在 SoapUI 里创建负载测试只要简单地选择一个功能测试用例，右击并且选择“New LoadTest”即可，是的，就是这么简单！

这使得 SoapUI 的负载测试相当地好用，它可以让你在功能测试完成的情况下，快速地、方便地、随意地检验 Web Service 接口是否能够承载指定的负载量。

- 点击测试集“Sample expanded TestSuite”，展开测试用例“Search and Buy TestCase”的“Load Test”，可以看到，对同一个功能测试用例可以有 4 个不同的负载测试用例，可根据性能测试场景选择不同的负载策略。



- 双击点开其中一个负载用例，可以配置里面的配置项。

·Limit:

表示我们负载测试要持续执行的时间，秒为单位，此处表示要执行 120 秒。

·Threads:

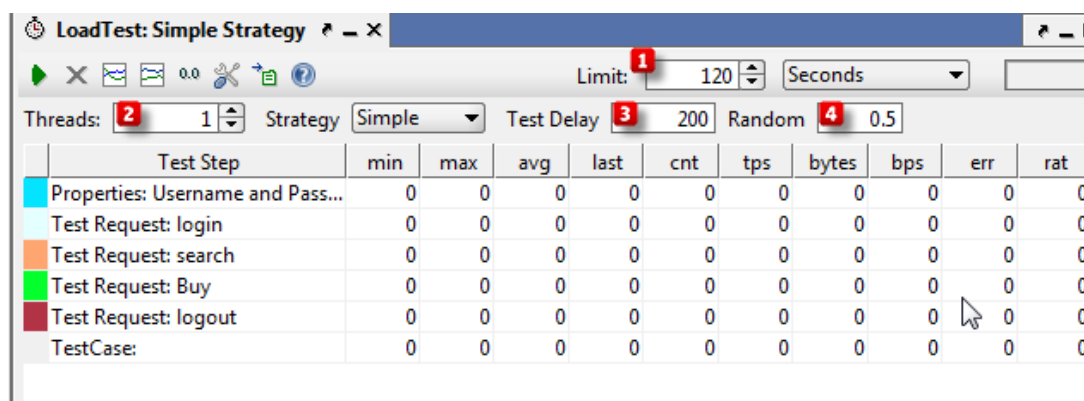
配置负载测试所用的线程数，即一般性能测试中所说的并发数。

·Test Delay:

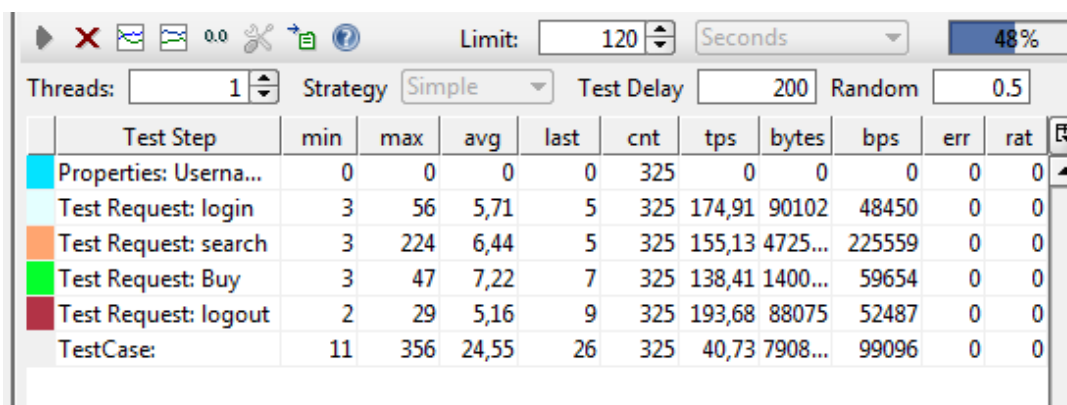
设置测试时线程的休眠时间，即在完成一次完整的用例执行后，开始下一次执行时，线程的休眠时间，以毫秒为单位（1000 毫秒是 1 秒），例子中是 200 毫秒。


·Random:

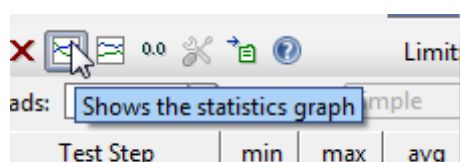
该值的设置是与“Test Delay”的设置结合在一起的，它表示休眠的时间会在“Test Delay”*(1-0.5)=100 毫秒，和“Test Delay”*(1+0.5)=300 毫秒之间波动。此处如果设置为 0，则表示“Test Delay”的值不会随意地变化，直接是初始设置的毫秒数。



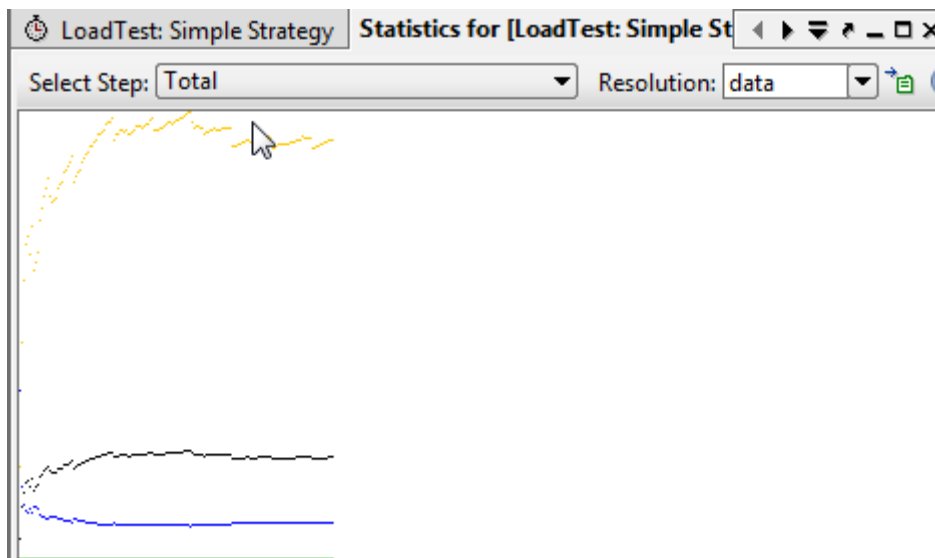
- 现在，让我们开始执行吧，在执行中，你可以看到，测试关注的的数据跟随着测试的进行而持续地发生变化，我们能够得到的数据有：响应时间、每秒吞吐量（tps），错误数（errors）等性能测试中，一般较为关心的数值。



- 你也可以通过点击  图表按钮：



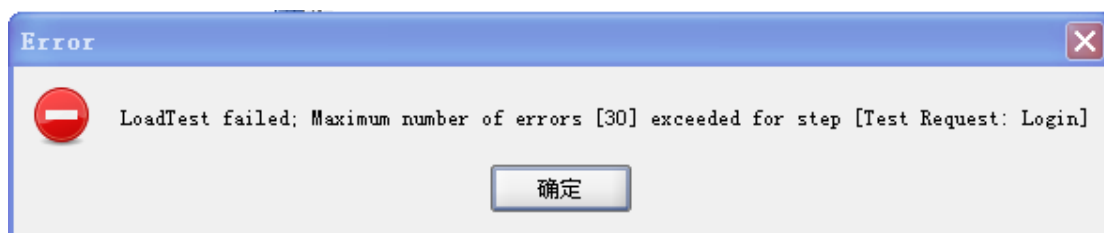
查看测试结果数据的整体走向图：



- 在 SoapUI 的负载测试中，你也可以定义负载测试的断言，一个最经常且重要的断言是：Max Errors。当负载测试中，出现错误，且错误的数量达到 Max Errors 要求的值时，负载测试会停止。

	Name	Step	Details
+	Step Status	- Any -	testStep: - Any -, minRequests: 0, max...
+	Max Errors	- Any -	testStep: - Any -, maxAbsoluteErrors: 1...

- 注：进入测试集“Sample TestSuite fails if we don't get faults”—测试用例“TestCase: Searching after Logging out LoadTests”—负载测试用例“LoadTest with Multiple Tests (will fail)”，双击打开，并开始执行，执行一段时间后，页面弹出错误信息，执行失败，这主要是因为多个用户模拟登录且使用同一个用户名，这在 MockService 所模拟出来的服务接口是不被允许的。



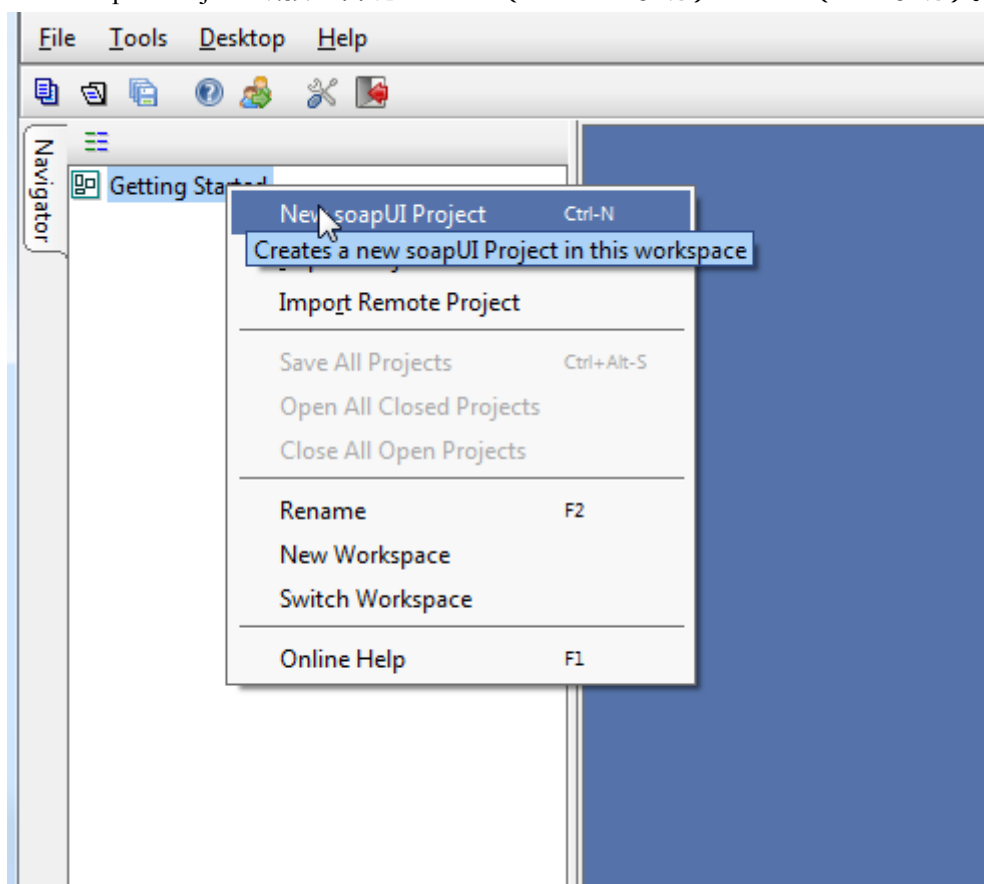
5. 基础讲解

通过上面例子的介绍，我们已经对 SoapUI 能做什么，怎么使用有个大致地了解，但对具体的使用还不够清楚，下面将通过在 SoapUI 中创建一个完整的项目为例子，来介绍完成一个接口测试所要操作的每个步骤。

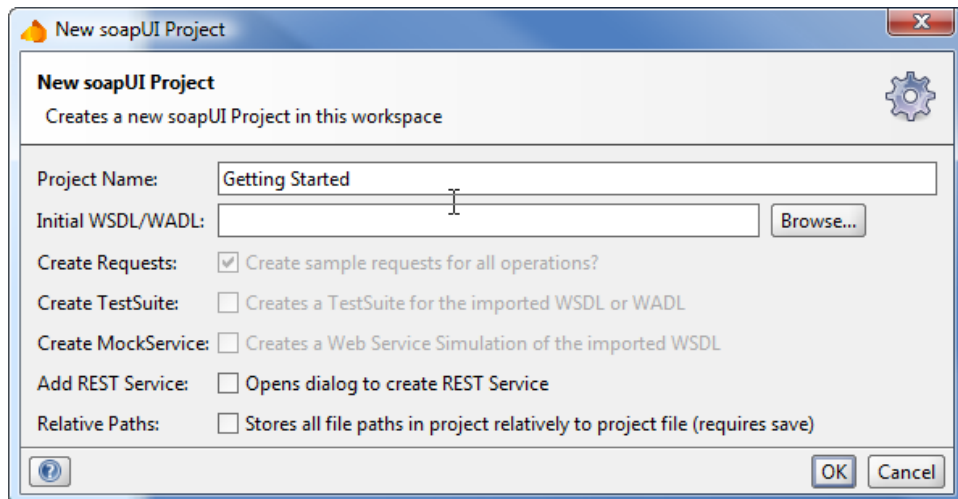
5.1. 创建工程

SoapUI 的项目是一个 SoapUI 测试中最重要的部分，只有创建了工程，我们才能创建对应的功能测试，负载测试，MockServices 等，因此如何通过一个 Web Service 接口来创建一个工程，是最需要了解的一个功能。

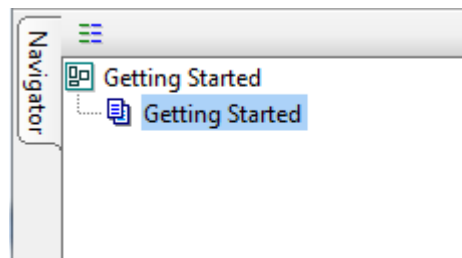
- 在左边栏 Navigator 区域，右键单击工作空间名称，在弹出的菜单中选择“New soapUI Project”或按组合键：Ctrl-N（windows 系统）/ cmd-N（mac 系统）。



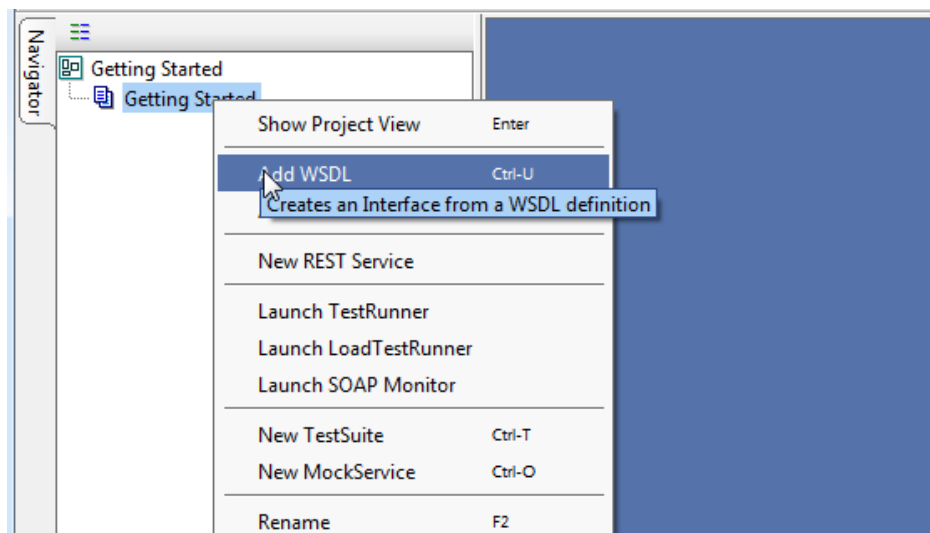
- 页面弹出“New soapUI Project”新建对话框，可以输入项目名（Project Name），选择不填，在之后进行添加。



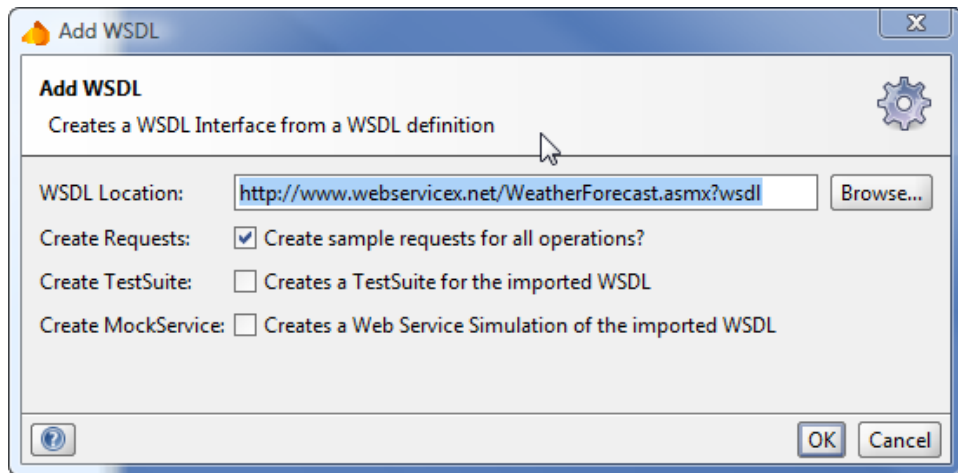
- 点击“OK”后，在左边栏的 Navigator 区可以看到已经成功地导入项目，如果你需要导入一个项目，可参考上文导入例子程序的操作步骤。



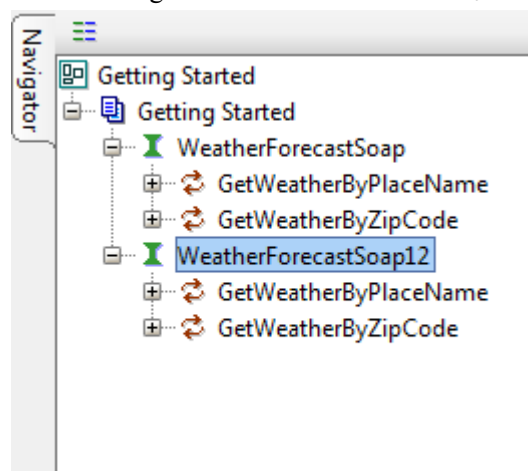
- 接下来需要添加 WSDL，每一个项目都基于一个对应的 WSDL，导入一个 WSDL 不是必须的，但导入以后，你可以轻松地获得 WSDL 包含的所有信息，极大地方便测试用例的编写，右键点击项目名“Getting Started”，在弹出的菜单中，选择“Add WSDL”，页面会弹出“Add WSDL”对话框。



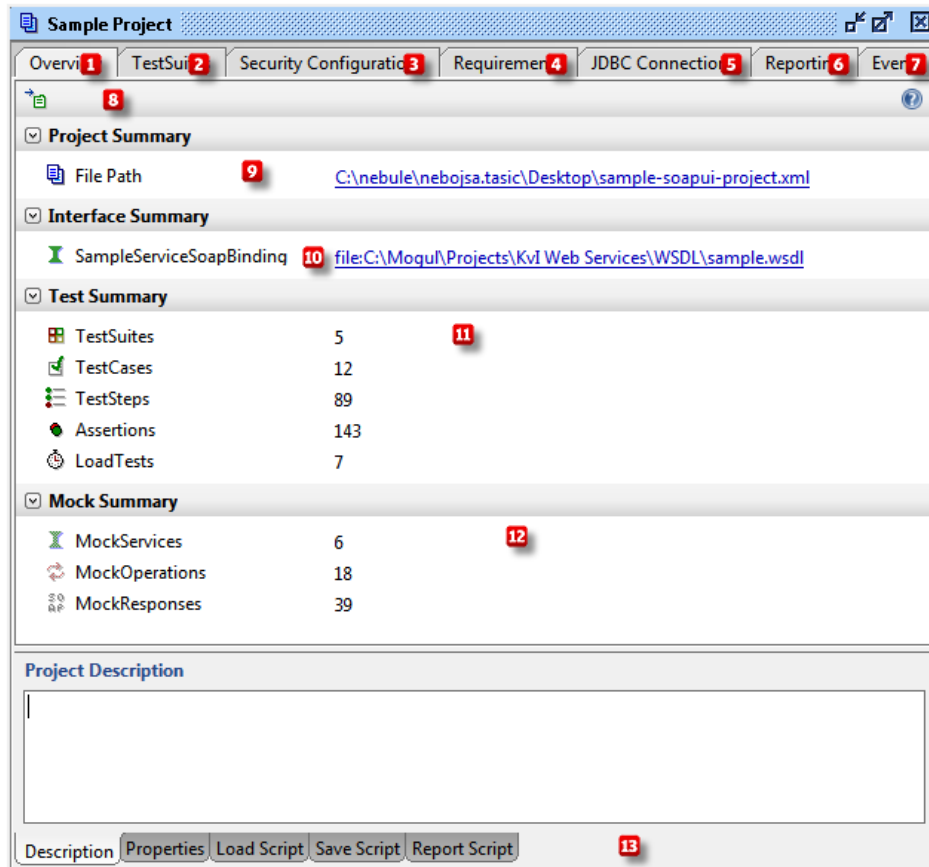
- 输入：<http://www.webservices.net/WeatherForecast.asmx?wsdl>，并点击“OK”。



- 成功后可在左边栏“Navigator”区域看到导入的接口。

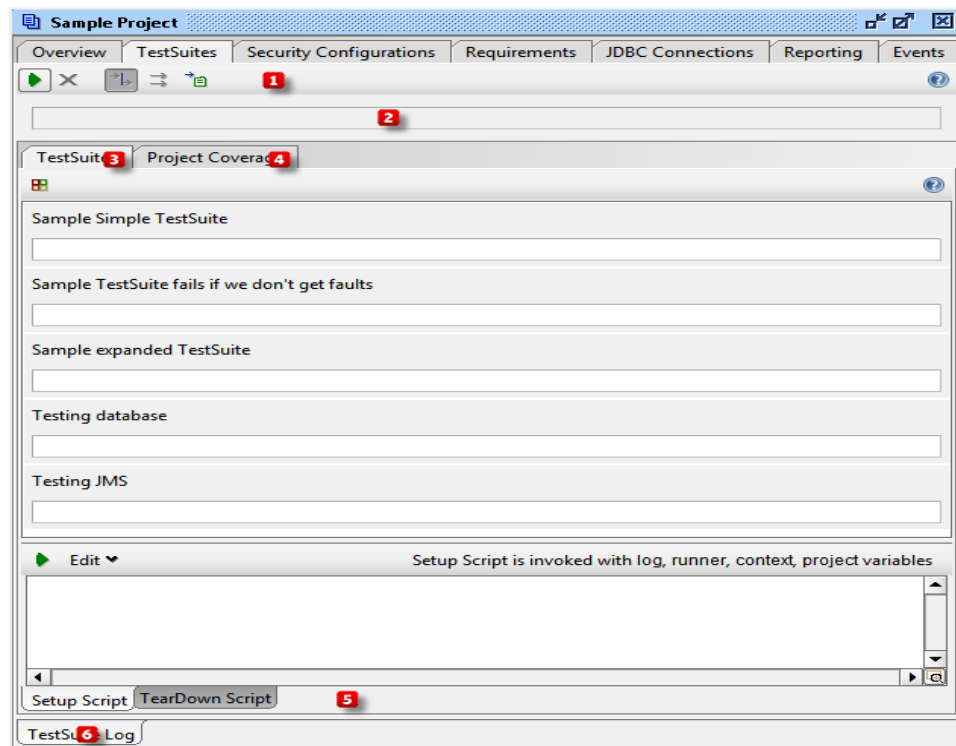


- 你可以双击项目名来打开项目的概要信息窗口，当你的测试资源越来越多时，用这个视图来管理你的所有测试资源是非常重要的，如：JDBC 连接（SoapUI Pro）、安全配置，需求，测试集等信息。



上图所示的项目概要信息包括了大量项目范围的设置和数据信息：

- 1、“Overview”标签页列出了项目所包含了有用的相关数据和度量值。
- 2、“TestSuites”标签页显示了在项目中的所有功能测试集，并且允许你可以按顺序或并行执行测试用例。



(1) 工具栏：从整体上控件测试集的执行。

(2) 整体进度条：显示整体测试集执行的进度。

(3) 测试集进度条：显示单个测试集的执行进度。

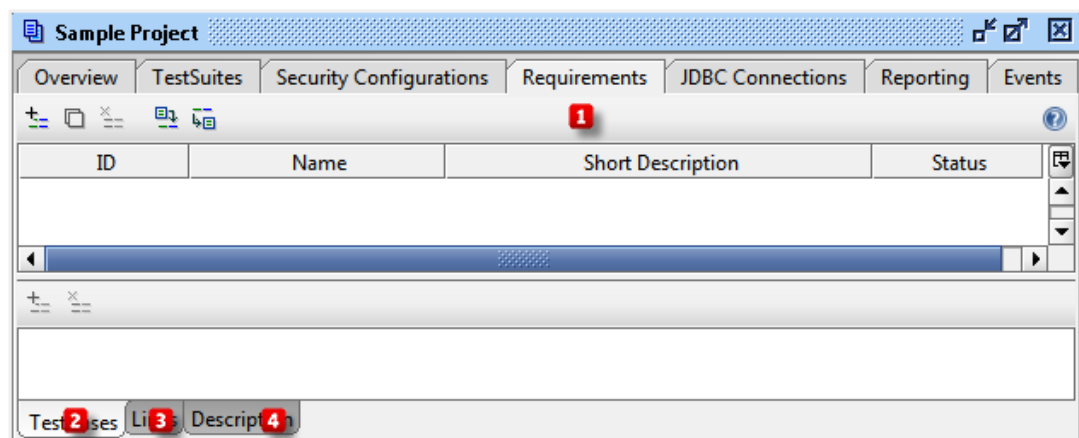
(4) 项目的覆盖率标签：查看测试集的覆盖率。

(5) 脚本：包含 Setup 和 TearDown 脚本，用在开始执行所有的测试集之前和执行完所有的测试集之后，分别运行这两个脚本进行初始化或清理工作。

(6) 在按顺序执行测试集时，会实时地显示测试中的打印日志。

3、“Security Configurations”管理项目范围内，基于 SOAP 的 Web Service 服务的 WS-安全配置。

4、“Requirements”管理项目需要的资源。



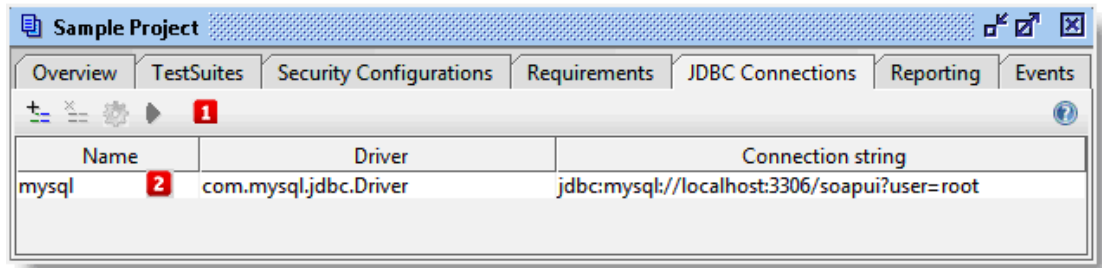
(1) 工具栏：添加、删除、导入、导出 requirements。

(2) TestCases：在这里的每个 ID，在 Testcases 中都可以设置与之关联的测试用例，在设置好关联关系后，可以双击测试用例，即可打开对应的测试用例。

(3) Links: 可以设置链接到有用的引用或是在项目管理系统中的实际需求。

(4) Description: 为测试用例添加描述。

5、“JDBC Connections”管理项目范围内可以被使用在多个场景的 JDBC 连接。




(1) 工具栏: JDBC 连接工具栏, 包括添加、删除、配置和测试连接按钮。

(2) JDBC 表格: 包括 JDBC 配置信息的表格。

6、“Reporting”管理所有全局的和项目范围内的报告模板和参数。

7、“Events”管理项目范围内用来增强测试执行的 event-handlers (事件控制)。

8、 创建报告的按钮: 通过点击按钮可以创建报告。

9、“Project Summary”项目文件所存放的路径。

10、“Interface Sumamry”WSDL 文件所存放的路径。

11、“Test Summary”统计和预览测试用例数据。

12、“Mock Summary”统计和预测模拟服务的数据。

13、“Overview Inspectors”: 包含描述、属性、加载、保存、报告等。

·Load Script:

在项目加载后调用, 可以被使用来初始化一些会话、数据等。

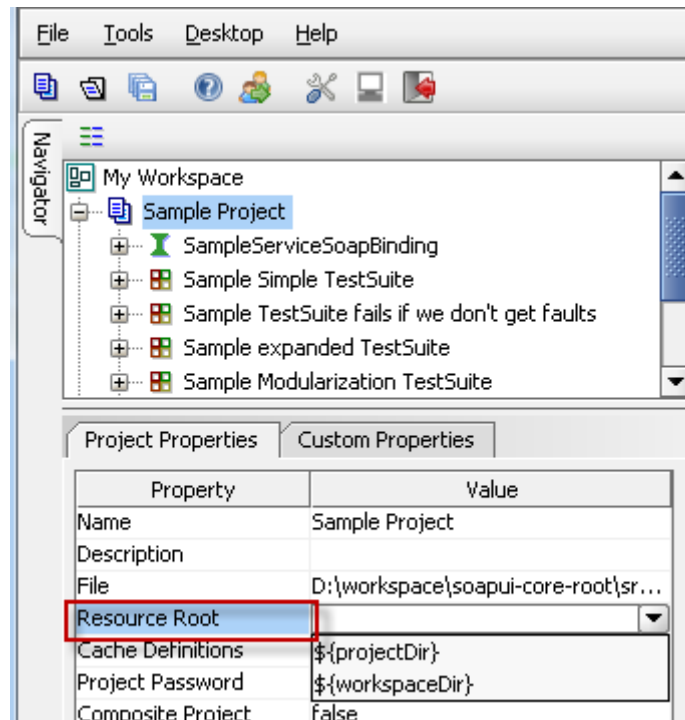
·Save Script:

在项目被保存前调用, 允许你执行一些自定义的清理任务, 如清空密码或测试资源等。

·Report Script:

当生成项目报告的时候被调用。

同样的, 在左边栏的下半部分还包含了项目的属性和自定义属性两个标签页:



使用外部文件来保存属性、测试数据等是一个相当普遍的方式，可以在“Resource Root”指定的地址存放整个工作空间所有项目需要使用的数据文件等资源，这里指定一个相对路径是更好的方式，因为项目有可能会进行迁移，指定相对路径的情况下，我们就可以不用在每次迁移都要改变资源存放的路径。例如：指定“Resource Root”的值为“d:\data”，这里可以指定到一个已经创建好的目录，在这个目录下有一个文件“testdata.xls”，那么在执行的时候，路径会被解析为：d:\data\testdata.xls。

上图，通过下拉框，我们可以看到两个参数：

·\${projectDir}：

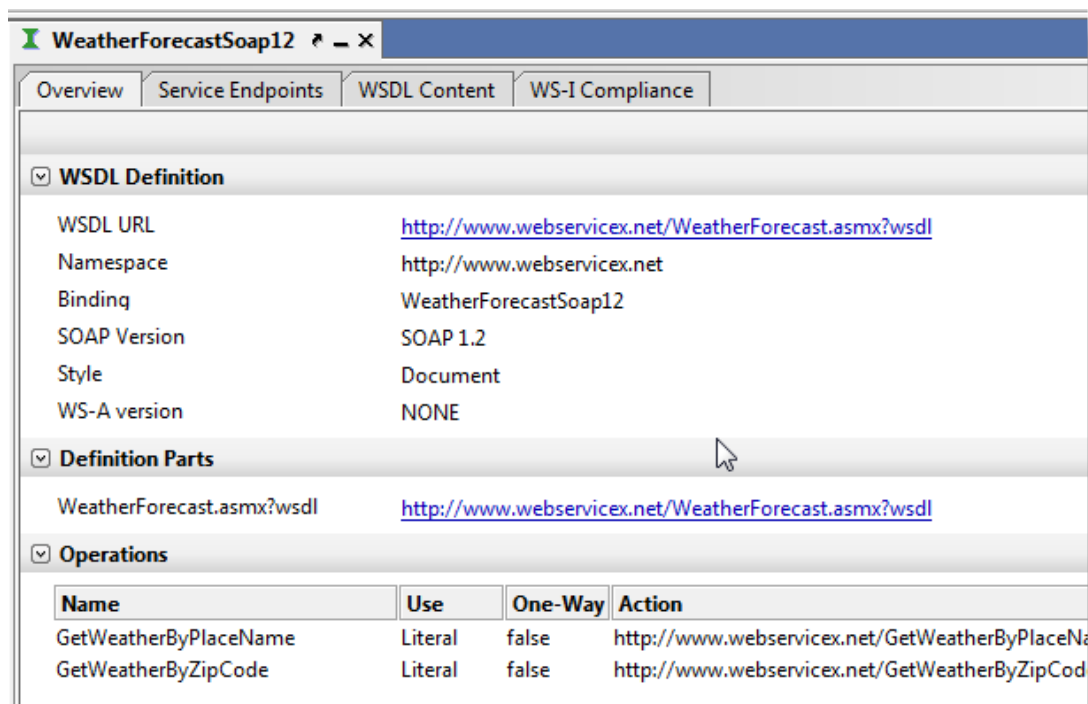
项目所在路径。

·\${workspaceDir}：

工作空间所在路径。

例如：指定“Resource Root”的值为：\${projectDir}/data，那么在执行时，上面所提到的 excel 文件的路径会被解析为：<项目所在路径>/data/testdata.xls。



- 你也可以通过双击接口名来查看接口的概要信息，从此窗口可以了解到 WSDL 接口的信息，对于浏览和检查 WSDL 是非常有用的。

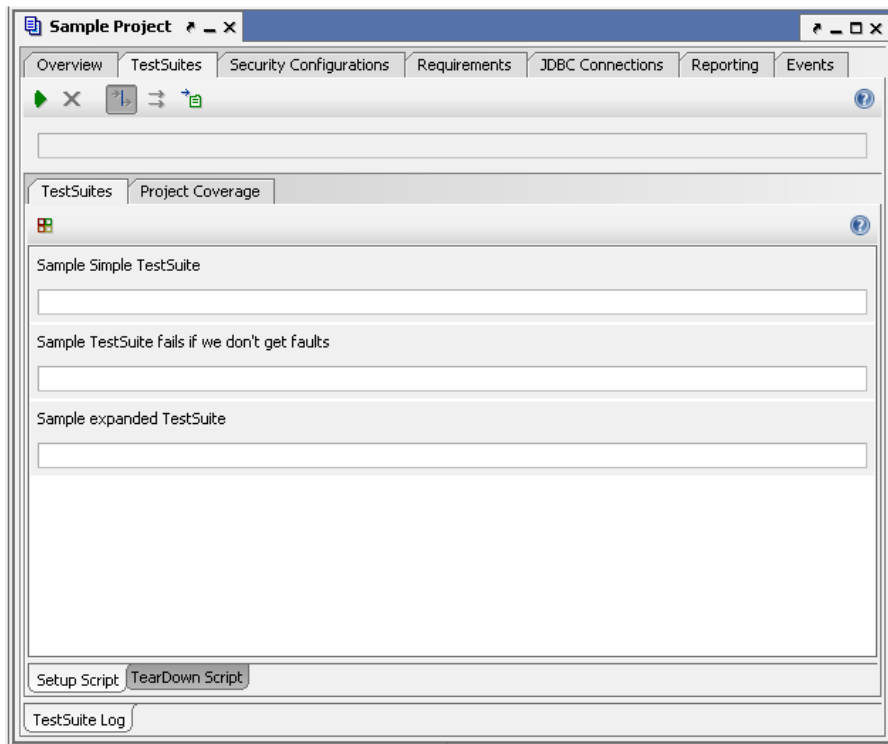


5.2. 测试结构的组织和执行


SoapUI 将功能测试用例组织为三层结构：测试集—测试用例—测试步骤。

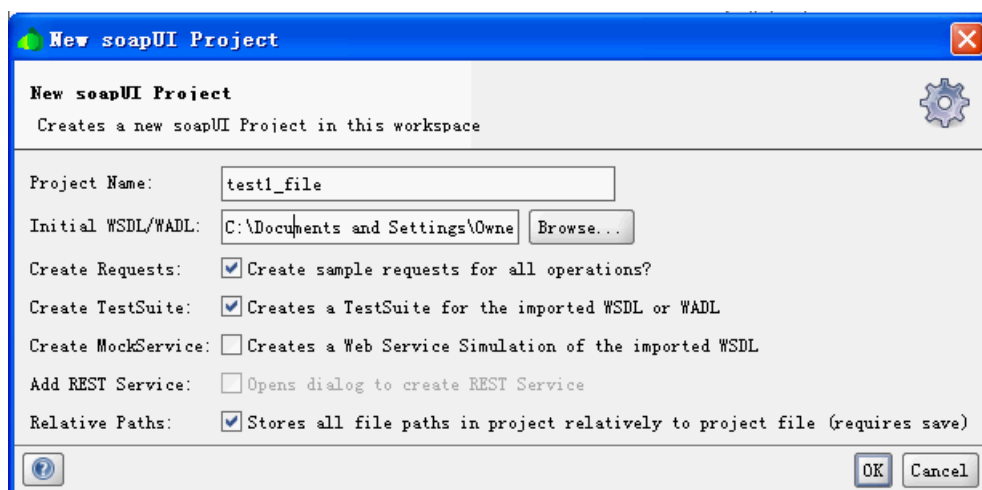
- 1、一个测试集是一组测试用例的集合，这组测试用例主要是针对同一个逻辑功能模块。在一个项目中为支持大量的测试场景，可以创建任何数量的测试集。
- 2、一个测试用例是一组测试步骤的组合，这一组测试步骤组合起来主要是为了测试服务某一个特定的功能，你可以根据需要在一个测试用例中添加测试步骤，这里的使用是很灵活的，更多的内容留给读者们自己研究。
- 3、测试步骤是功能测试的“积木”，它们被添加到测试用例中，用来控制、执行测试步骤和检验被测服务的功能。



在项目中创建测试集是为了将测试用例集合到同一个逻辑单元中。举例：假如有一个银行系统，则它至少有查询账务的功能，又有付款的功能，那么，我们会将这二个功能归到两个不同的测试集中，再通过组织测试用例来进行测试。你的项目可以包含任意数量的测试集，并且你可以通过双击项目名称，在打开的概要信息窗中选择“TestSuites”标签来执行这些测试集（顺序执行或并行执行，可选择工具栏的  按顺序执行图标或  并行执行图标）。

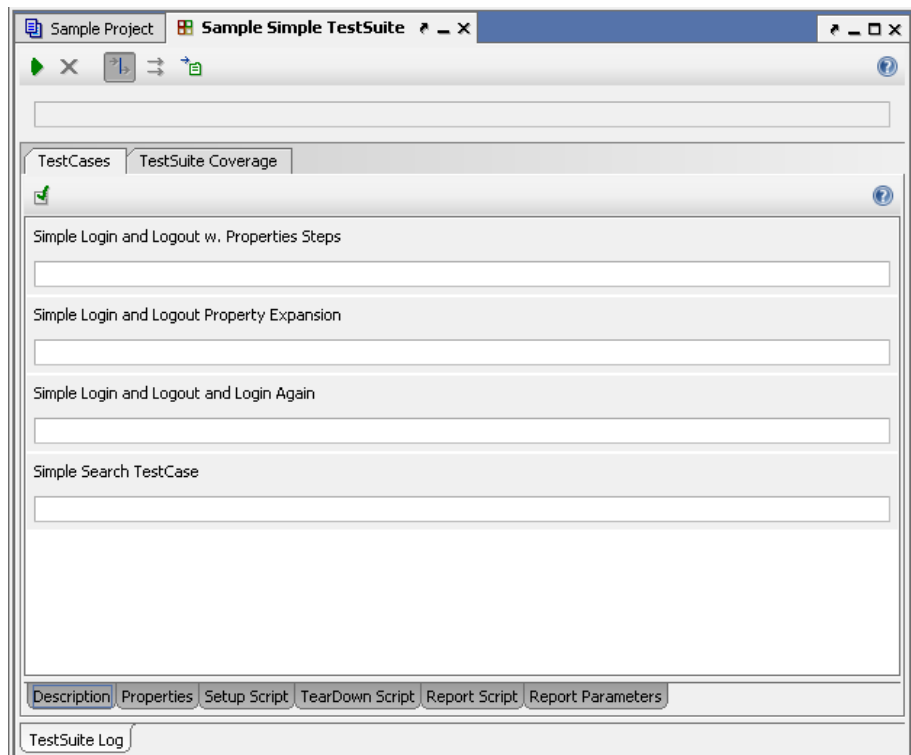


创建一个新的测试集可通过以下方式创建：


- 1、通过右键单击项目名称的弹出菜单，选择“New TestSuite”
- 2、右键点击上图“TestSuites”标签页中任何位置，在弹出的菜单，选择“New TestSuite”
- 3、通过上图的  图标也能创建一个新的测试集
- 4、测试集还可以在初始创建项目，导入 WSDL 或输入 WSDL 的 URL 后，勾选“Create TestSuite”，勾选此选项，系统会自动帮我们为每个接口都创建一个测试集，后续可根据测试用例需要进行调整即可。



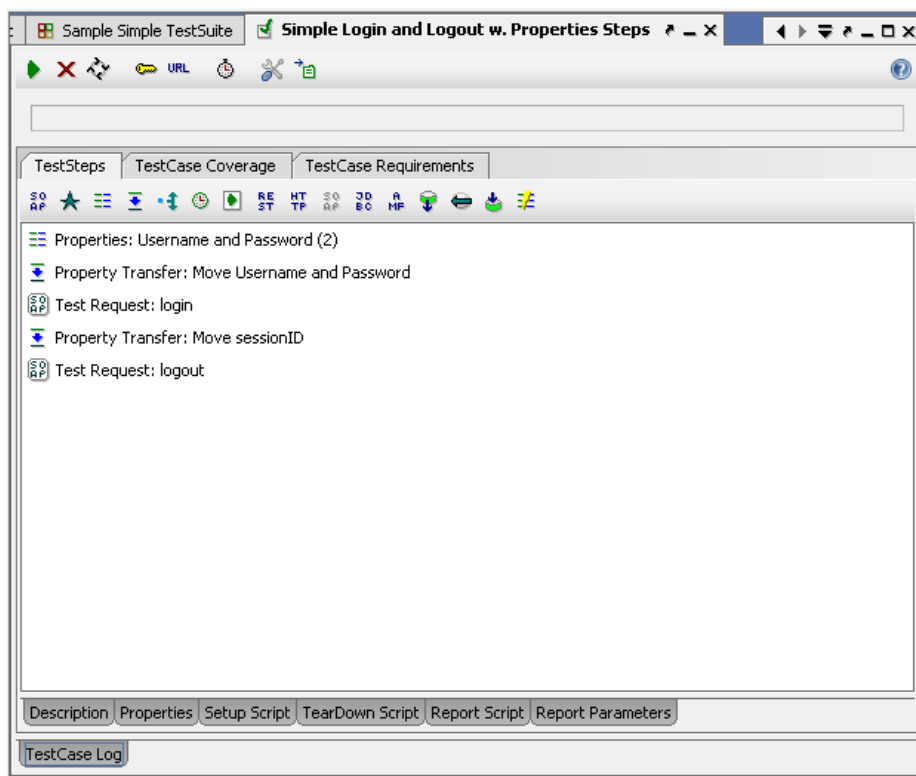
一个测试集可以包括任意数量的测试用例，执行时可以按顺序执行也可以并行执行，可以通过点击测试集窗口中工具栏的  （按顺序执行）图标或  （并行执行）图标来选择执行的顺序。



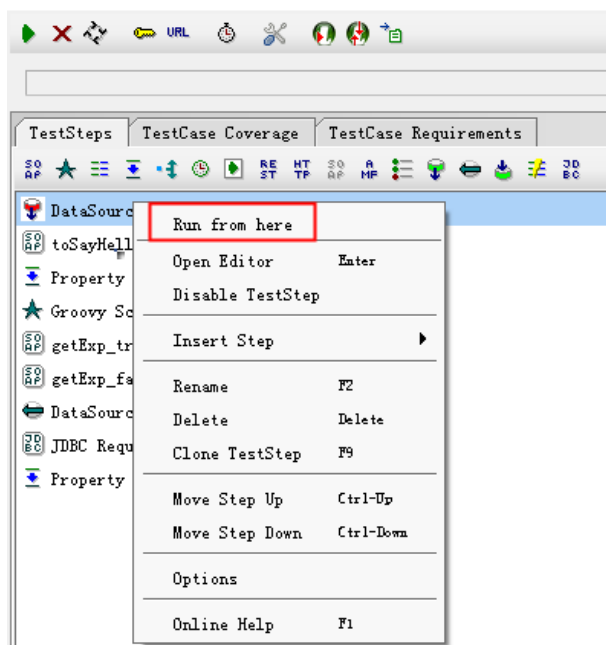
创建测试用例的方式与创建测试集的方式基本一致，主要有以下几种：

- 1、 右键单击项目中的测试集名称，在弹出的右键菜单中选择“New TestCase”
- 2、 点击上图创建测试用例图标： 
- 3、 在上图中“TestCases”标签页中任何位置，右键点击，选择弹出菜单中的“New TestCase”

一个测试用例包含一定数量的测试步骤，目的是为了向服务接口发起请求并对测试结果进行校验。一个测试用例在执行的时候，它的执行顺序与步骤所显示的顺序一致，如果有循环或分支，那么需要在测试步骤或测试脚本中进行配置。



右键单击上图某一测试步骤，弹出的右键菜单中包含“Run from here”选项，该选项方便在测试用例编写时，测试用例的调试。



Run from here：从所选择的测试步骤开始执行测试用例，非常方便调试。

Open Editor：打开所选择的测试步骤编辑窗口，双击上图的测试步骤也能打开编辑窗口。

Enable or Disable：测试步骤可以被启动，也可以被禁用，允许你有选择地执行不同的测试步骤，当你选择 Disable 测试步骤时，该步骤将变成灰色。

Insert Step：在当前选择的步骤之前插入一个新的测试步骤。

Rename and Delete：快速地重命名或删除测试步骤，在一个测试用例中，测试步骤的名称要惟一。

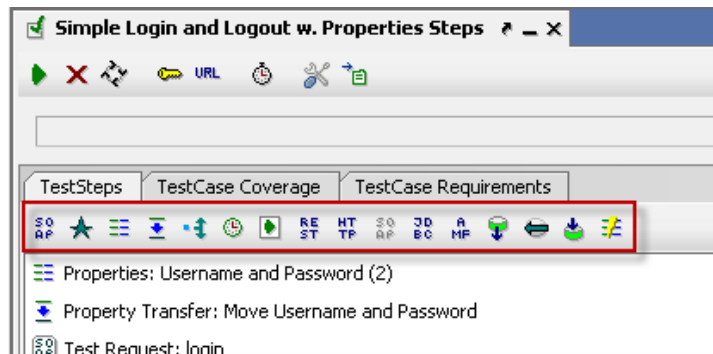
Move or Clone to other TestCases：测试步骤可以被复制或移动到当前工作空间的其它测试用例中，如果目标测试用例与要复制的测试步骤不在同一个项目中，那么你还需要复制接口到目标项目中，SoapUI 会引导复制接口的操作。

Move up or down：你可以通过这个功能移动测试步骤的执行顺序。

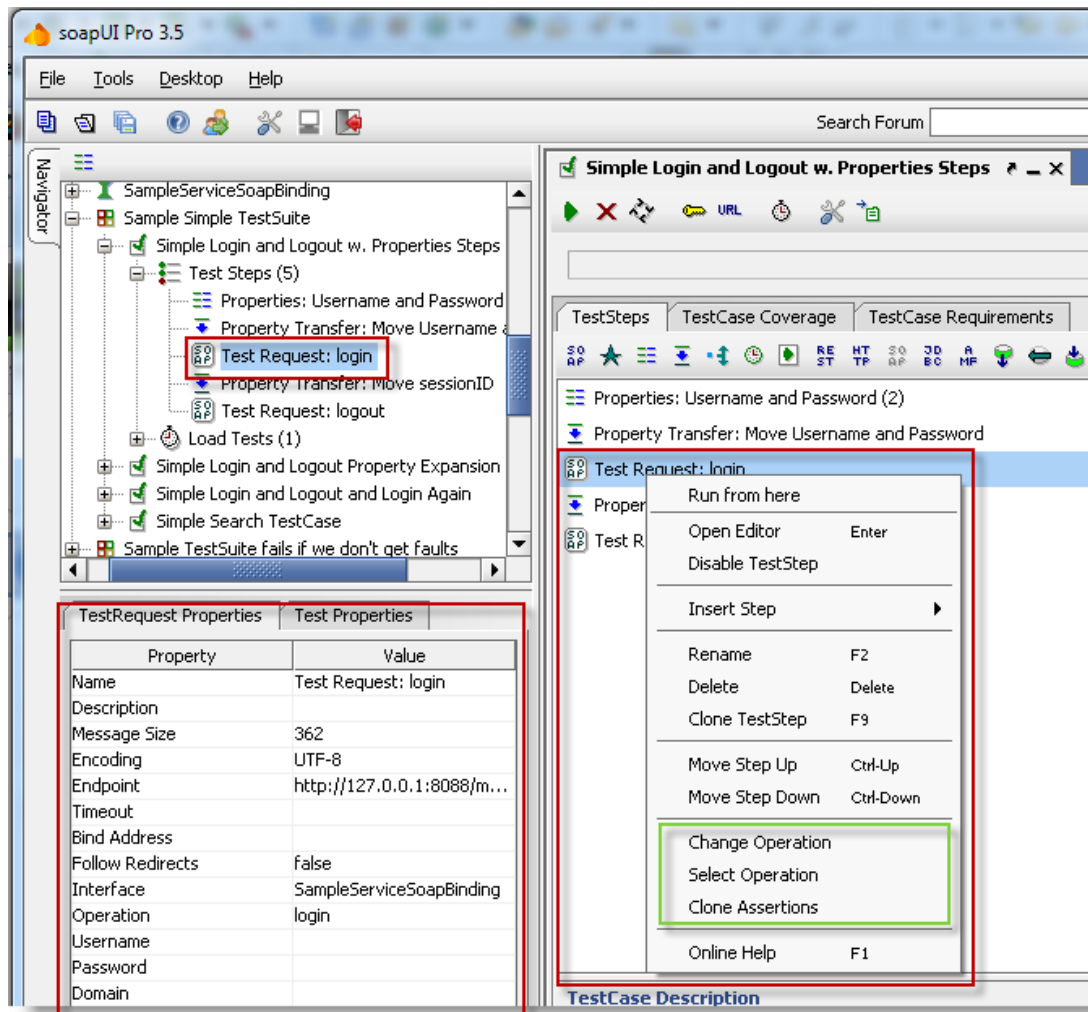
你可以选择多个测试步骤，右击时可提供批量删除、启用、禁用所选择步骤的操作。

5.3. 测试步骤

正如上一节所说：测试步骤是功能测试的“积木”，每一个测试步骤都在验证被测服务的功能。测试步骤默认情况下是按顺序执行，但存在几种可能的分支：循环、甚至调用其它的测试用例，即当有需要时会有复杂的测试用例。任意数量的测试步骤都可以被添加到一个测试用例里，添加时：1、可以通过右键单击测试用例名称，在弹出菜单中选择“Add Step”；2、也可以通过右键单击测试步骤名称，在弹出的菜单中选择“Insert Step”；3、下图为双击测试用例名称打开的测试用例信息框，红包方框内的图标是各种测试步骤，点击对应的图标会弹出相应测试步骤添加框。



选择一个测试步骤，右键单击后的弹出菜单显示了该用例所具有的行为，且左边底部也显示了相应测试步骤的属性，可以直接进行设置。下图中绿色方框中的所标记的行为是 SOAP 请求测试步骤所特有的，其它的选项则是所有的测试步骤都具有。如下图所示：



用例里都会有一些主要的测试步骤来验证接口的功能，它们既可以发送请求给接口，也可以接收接口返回的结果报文，且响应报文可以经由一个标准的断言机制进行校验。你可以很轻松地组合一个测试用例的所有步骤，并且能够在它们之间很方便地共享数据，例如你也许会使用一个标准的 HTTP 请求登录到一个服务上，然后使用接收到的 HTTP 响应报文作为入参，用 SOAP 发起请求，最后再使用 JDBC 的测试步骤验证数据库的结果数据。

测试步骤—属性相关

属性测试步骤一般用来管理需要参数化的属性，测试步骤主要有：

Properties：允许你定义任意数量的属性，属性可以从文件读出或写入，可以用来参数化请求、断言等。

Property Transfer：允许你在测试步骤间传递或抽取属性值，如你可以从一个响应信息中抽取一个值，然后通过 DataSink 测试步骤写入到一个外部文件。

DataGen：允许你创建一个计数器、随机值等的动态的属性。

测试步骤—数据相关

SoapUI Pro 添加了一些测试步骤主要用来与外部数据源交互，可读可写，主要有：

DataSource：允许你从一些外部源文件中读到属性值，外部源文件包含数据库、EXECL 文件、directories 等，之后可通过使用属性传递等将值作为请求入参或验证响应报文等。

DataSource Loop：与 DataSource 成对出现，主要用在当 DataSource 有多行数据时，可通过 DataSource Loop 循环得到每一行的值来作为入参发起调用，在 SoapUI 中这是一

种基于数据驱动测试。

DataSink：允许你将属性值写到外部存储文件中，如数据库、EXECL 文件等，以供后续分析和处理。

测试步骤—执行流程

虽然在测试用例中测试步骤的执行是顺序的，但 SoapUI 提供了一些测试步骤允许我们可以做分支、循环等：

Conditional Goto：检查返回的响应报文中指定的值，并跳转到步骤中符合条件配置的目标步骤。

Delay：根据配置的毫秒数暂停测试步骤的执行。

Run TestCase：转去执行指定的目标测试用例，当某些测试步骤需要在一些用例运行前被执行，可以使用这种方式进行处理。

DataSource Loop：正如上面所提到的，根据 DataSource 所配置的行数，以每行为一次入参，循环地执行测试用例中特定的某几个步骤。

测试步骤—其它类型

SoapUI 还提供了其它类型的测试步骤，让你可以做任何需要做的事情。Script TestStep 测试步骤让我们可以写任意的脚本（包括 groovy 和 javascript）做几乎所有需要做的事情，一般会在以下几个场景使用：

一些无法通过默认的断言机制实现的复杂报文的校验。

复杂的分支或循环。

以数据驱动来生成测试步骤的情况，如从数据库中动态生成测试步骤。

集成外部系统来读写数据。

触发外部的活动或进程，如发送邮件或启动程序。

与用户的交互（对话框等）来得到输入或控制执行。

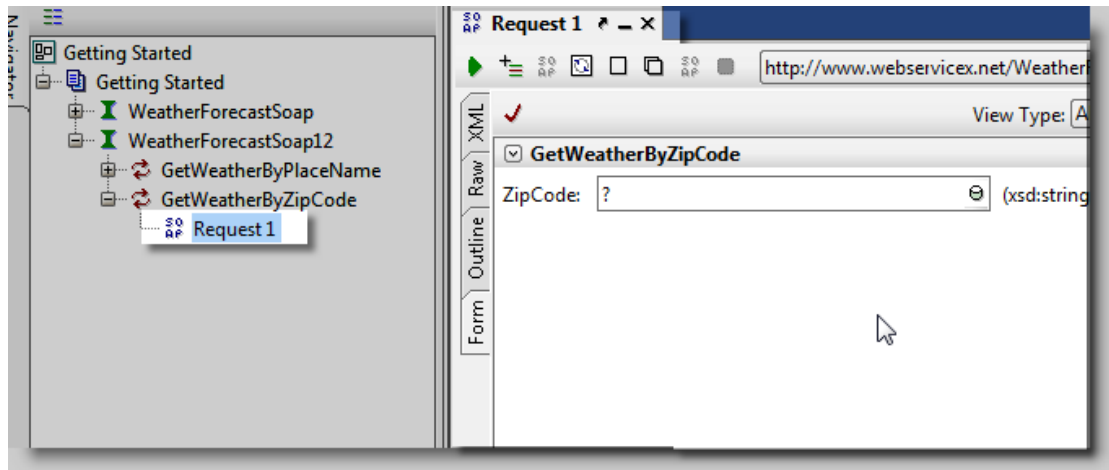
.....


SoapUI 提供强大的功能，基本上可以满足你所有服务接口的功能测试需求。

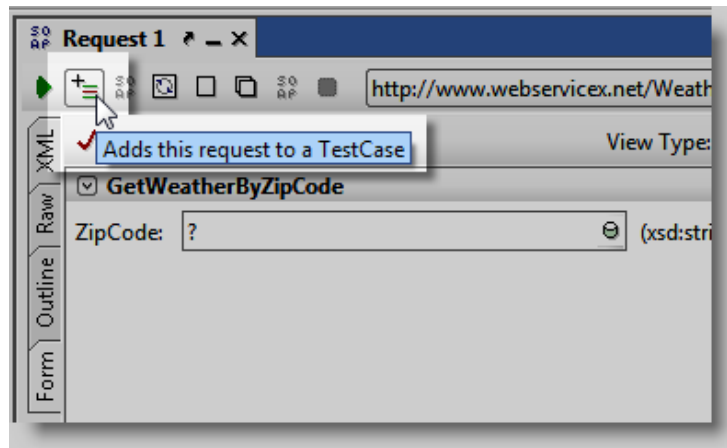
5.4. 创建功能测试用例

在导入 WSDL 后，一个项目已经创建完毕，本节主要围绕着如何创建一个请求展开描述。

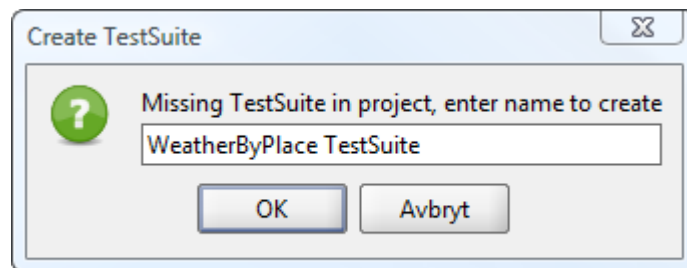
- 展开 Web Service 接口，选择“GetWeatherByZipCode”的“Request1”，双击打开进行编辑（此处是使用 SoapUI Pro 版本），请求报文区域会列出需要的入参名及类型。



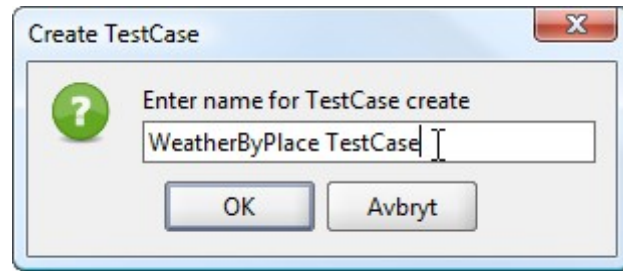
- 通过添加按钮  将该请求添加到测试集的测试用例中。



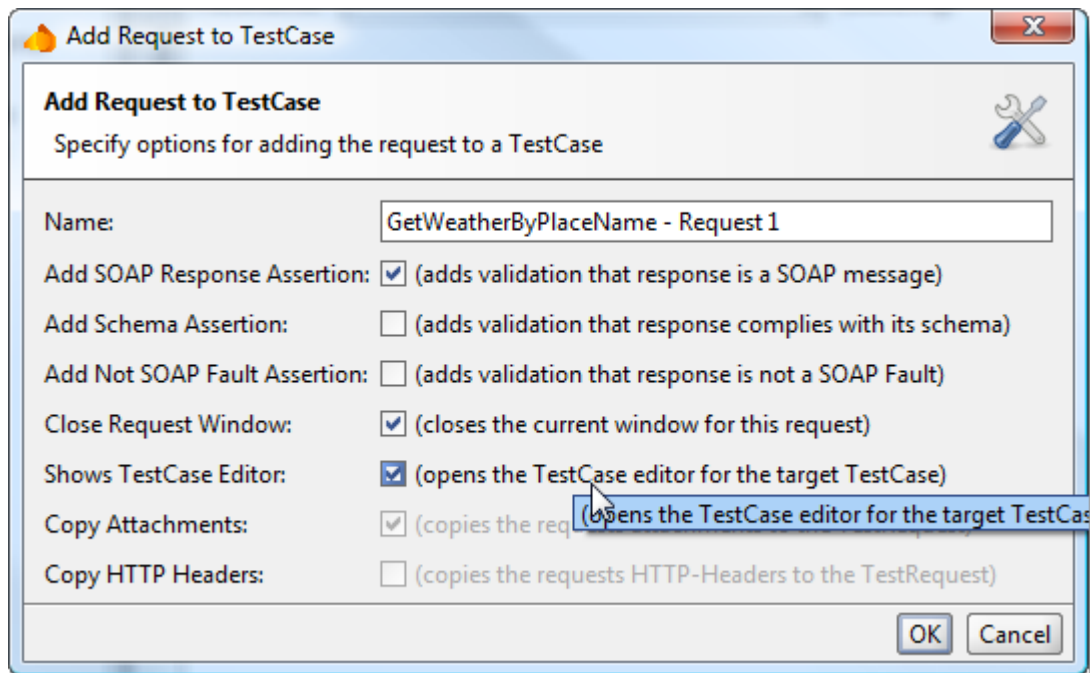
- 添加一个请求到测试用例时，会先打开一个创建测试集的对话框，在“Create TestSuite”对话框中，输入测试集的名称，点击“OK”即可。



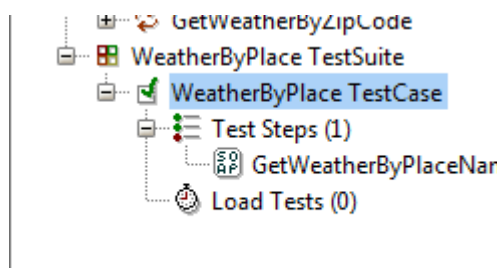
- 在创建测试用例的对话框中，输入测试用例的名称，这样就根据测试集包括测试用例的关系创建了用例。



- 最后弹出的“Add Request to TestCase”对话框，把刚才的请求复制一份直接添加到测试用例中，作为测试用例中的一个测试步骤。



完成上面的所有步骤后，页面左边栏正确显示了所有的层级关系，可以看出，在添加一个请求到测试用例时，系统会先创建一个测试集、再创建一个测试用例，最后才会把请求作为一个测试步骤添加到测试用例中：

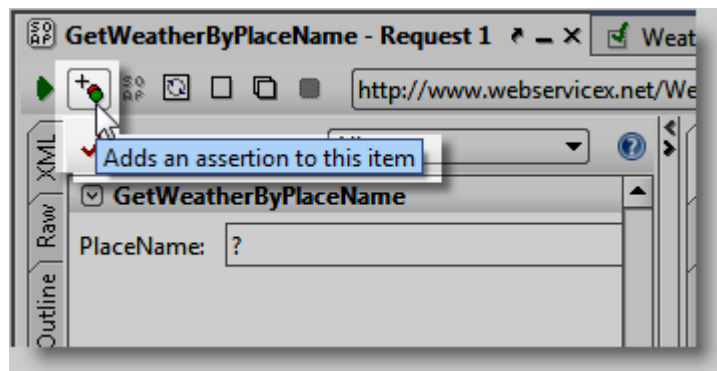


5.5. 功能测试断言判断

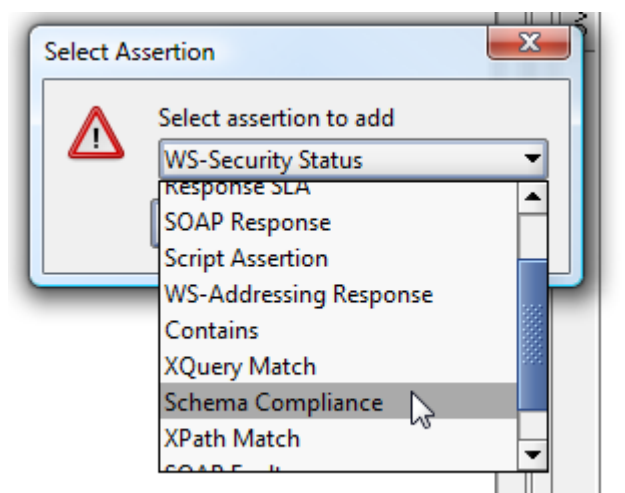
发起请求后，我们要关注接口提供的功能是否正确，那么就需要对响应的报文进行相应的验证，断言就是为了解决这个问题，对响应报文提供了多种方式的验证，来保证测试

结果的正确。程序中提供了多种方式的断言：Schema Compliance，Simple Contains，Simple Not Contains，SOAP Fault，Not SOAP Fault，SOAP Response，Response SLA，XPath Match，XQuery Match，Script Assertion，WS-Security Status，WS-Addressing Response Assertion，WS-Addressing Request Assertion。

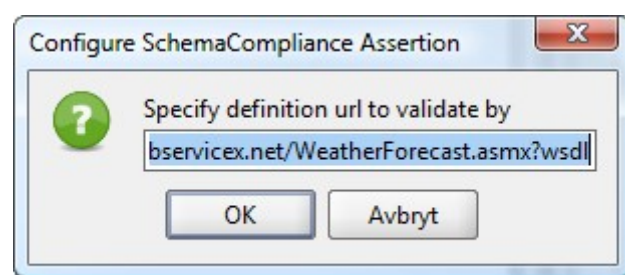
- 双击打开上一节创建的请求编辑页面，在请求编辑页面，单击添加断言按钮来添加断言：



- 在“Select Assertion”对话框中，从下拉框选择“Schema Compliance”方式来创建断言：



在“Configure Schema Compliance Assertion”配置 URL 的地址，此处会默认显示之前导入的 WSDL 地址，如果不需要改变，可直接点击“OK”。

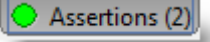


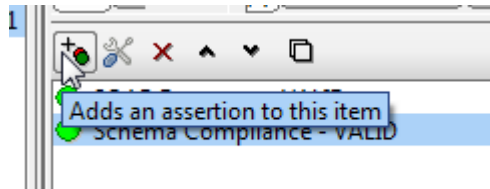
一旦你添加了断言，在执行该请求时，断言就会根据创建时的配置来验证响应报文的内容，如果验证通过，请求名称前面的图标 SOAP 图标就会变成绿色，否则将以红色显示。

断言的标签中同时会显示该请求已创建的断言个数，通过点击标签，可查看对应

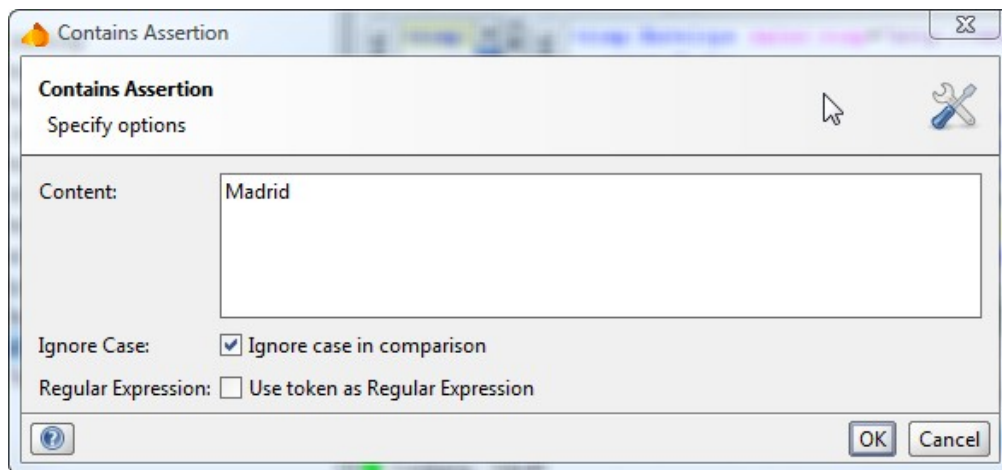
的断言信息。

5.6. 功能测试用例执行

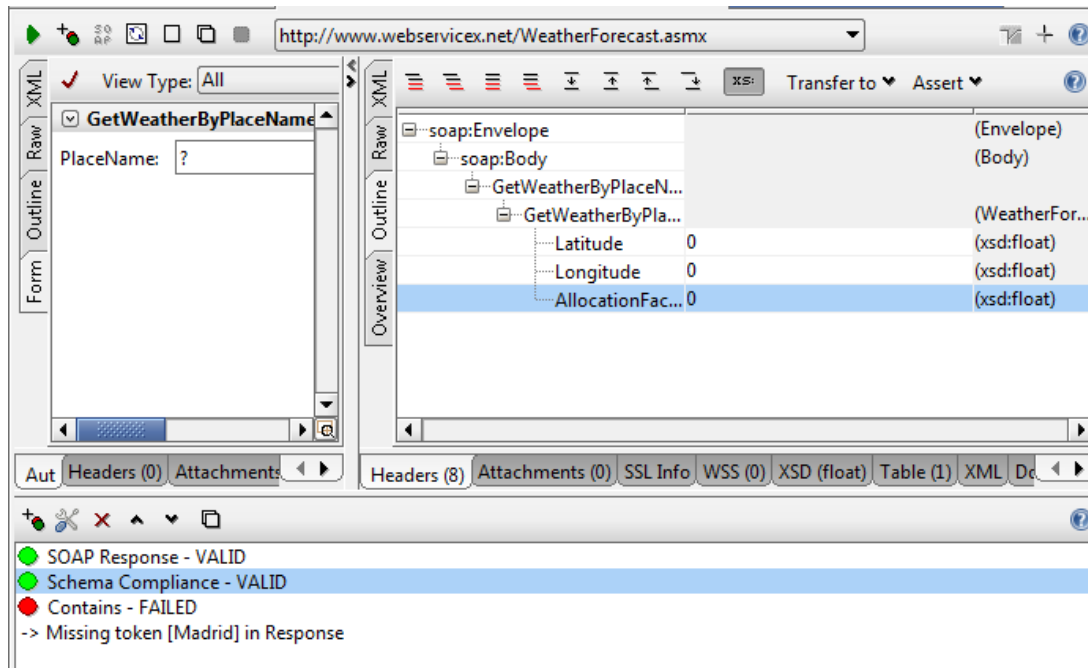
- 在用例执行之前，先给上面的测试步骤再增加一个断言，点击  标签，显示断言信息框，点击添加断言按钮



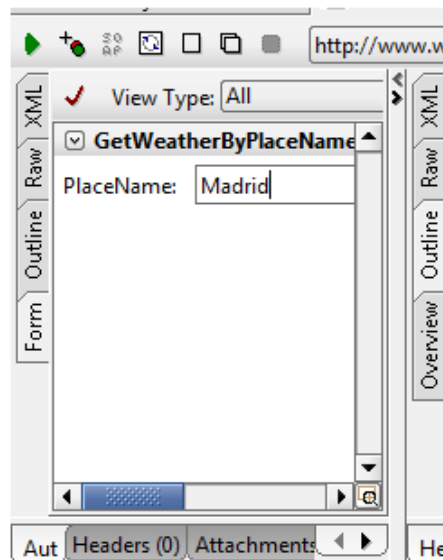
- 我们将添加一个最简单的检查响应报文内容的断言，名为“Contains”断言，“Contains”断言检查整个从服务器返回的响应报文，包括 XML 都可以作为查找的内容，作为一个断言的方式，它比较生硬但非常有用。在弹出的“Contains Assertion”对话框中，我们添加了要查找的内容“Madrid”，由于不清楚内容的格式，因为选择忽略大小写。



- 点击执行按钮，我们可以看到断言失败，断言失败时，SoapUI 会自动弹出断言住处框，并将断言失败的信息打印在断言下方，方便调试。



这是预期的结果，因为我们发出的请求中入参没有使用正确的入参，从上图中可以看到入参 PlaceName 是使用？作为入参，此处我们修改为“Madrid”，并发起请求，这时返回的结果报文中便包含了“MADRID”，因此断言通过，校验成功，图标均显示为绿色。



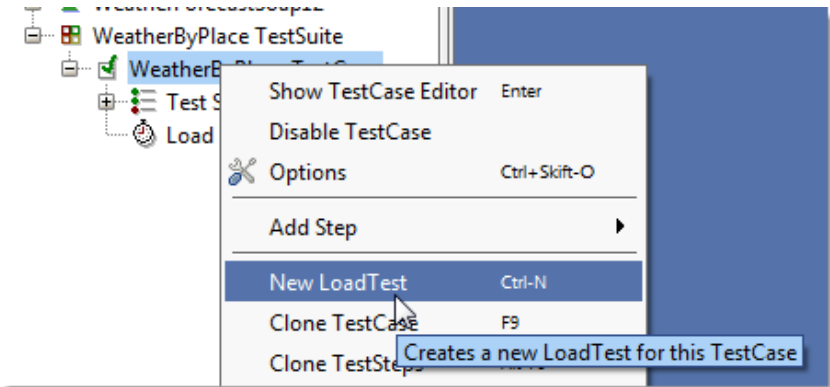
- 上面的请求可以通过双击测试用例，打开测试用例概要信息框进行执行，测试用例方式的执行，它会执行用例里的每个步骤。也可以通过双击测试集，打开测试集概要信息框进行执行，测试集中会显示出所有的测试用例，执行的时候会根据配置，按顺序执行或并行执行所有用例。

5.7. 创建负载测试用例

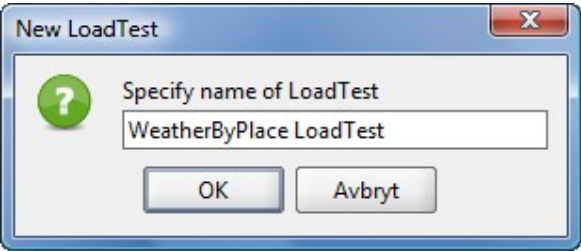
在我们创建完一个测试用例后，我们可以快速地创建对应的负载测试用例，并且

能很方便地更新用例的内容，主要原因是因为负载测试用例的生成是由已经存在的功能测试用例直接产生的。你可以通过这个功能验证 Web Services 接口在不同负载场景下的性能，也可以通过同时运行多个负载测试用例，观察在运行的同时是否会影响彼此的功能。

- 右键单击已经创建好的测试用例，在弹出的右键菜单中选择“New LoadTest”。

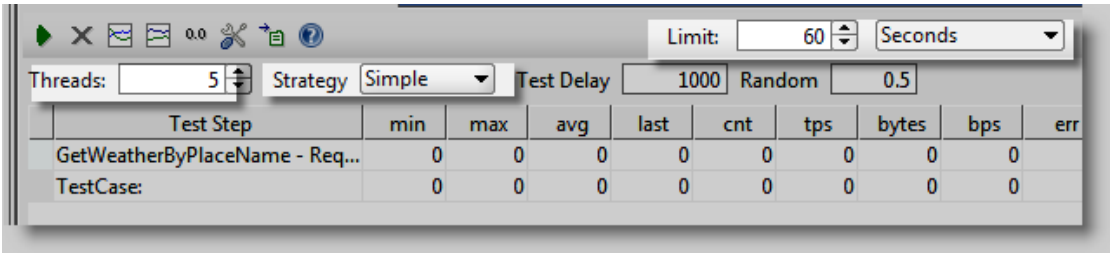


- 在弹出的“New LoadTest”对话框中，填入用例名称，点击“OK”即可。

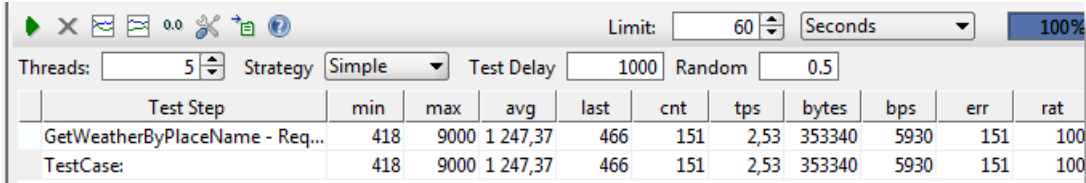


5.8. 执行负载测试用例

在负载测试用例创建完成之后，还需要配置相应的负载策略，配置窗口如下图：

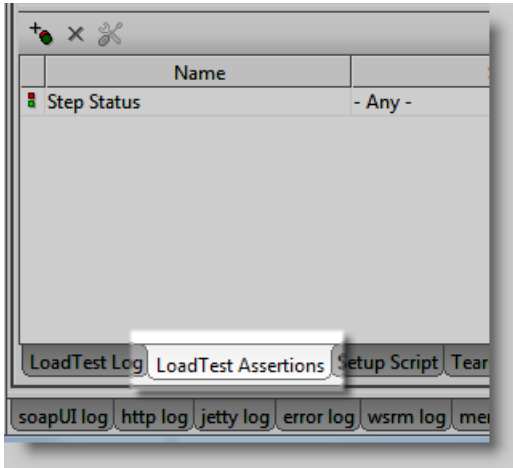



可以根据测试场景进行相应的修改，开始执行以后，我们可以看到在中间的统计图表开始收集测试结果数据，并实时更新。

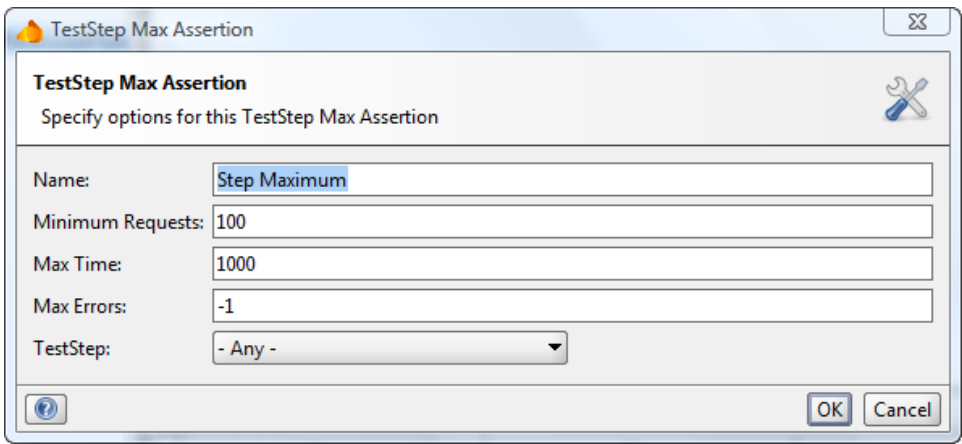


5.9. 负载测试断言判断

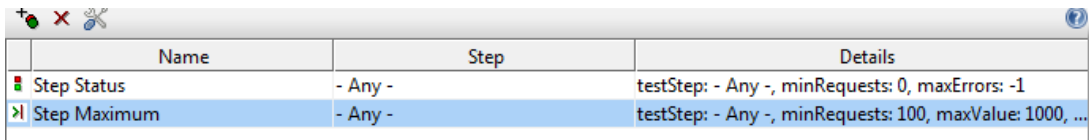
- 在“LoadTest”编辑框中，选择编辑器底部的“LoadTest Assertion”标签。





- 点击添加断言按钮 ，添加一个负载测试的断言，在弹出的对话框中，选择“Step Maximum”类型的断言，“Max Time”设置一个请求最大的响应时间（单位为毫秒），表示如果超过这个时间，则负载测试停止。“Minimum Requests”设置允许发出的最大请求数，“Max Errors”设置允许发生的最大错误数，“TestStep”可从下拉框选择上面设置的值是对用例中的哪个测试步骤生效，选择“Any”表示对任意的步骤生效。



保存成功后，在断言信息框中会添加一条断言：



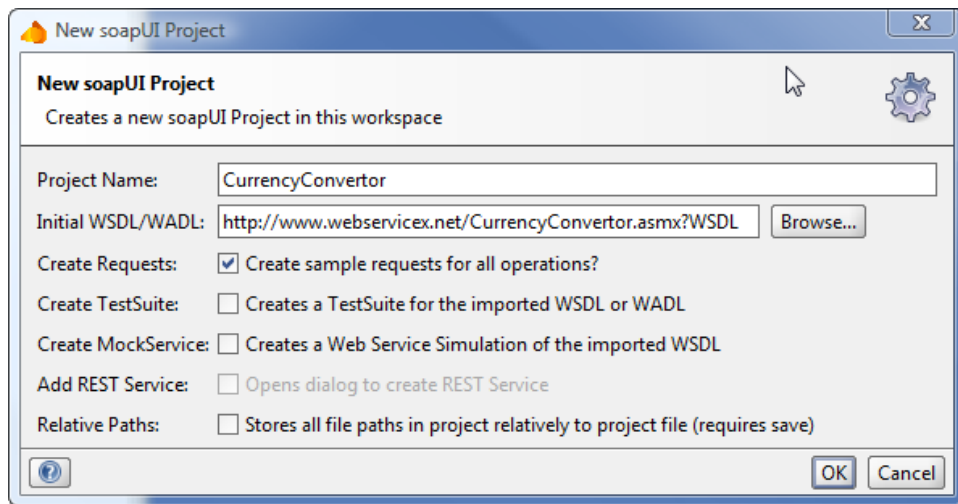
	Name	Step	Details
	Step Status	- Any -	testStep: - Any -, minRequests: 0, maxErrors: -1
	Step Maximum	- Any -	testStep: - Any -, minRequests: 100, maxValue: 1000, ...

5.10. 创建 MockService

Web Service 模拟服务在测试工具库中是非常有用的工具，它解决了一个问题：当 Web Service 接口还未编码完成的时候，我怎么去准备我的测试用例？Web Service 模拟服务就是为了解决这个问题，它允许你在真正的服务接口完成之前创建一个模拟或近似的服务接口。

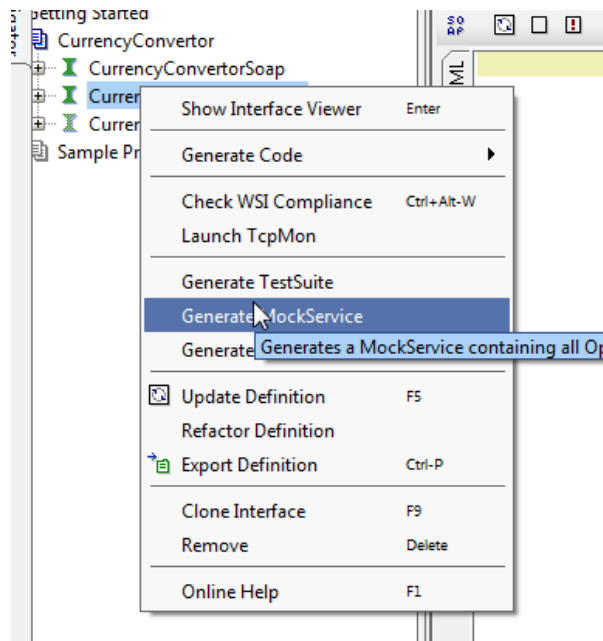
- 新创建一个项目，使用下面的 wsdl：

<http://www.webservice.net/CurrencyConverter.asmx?WSDL>

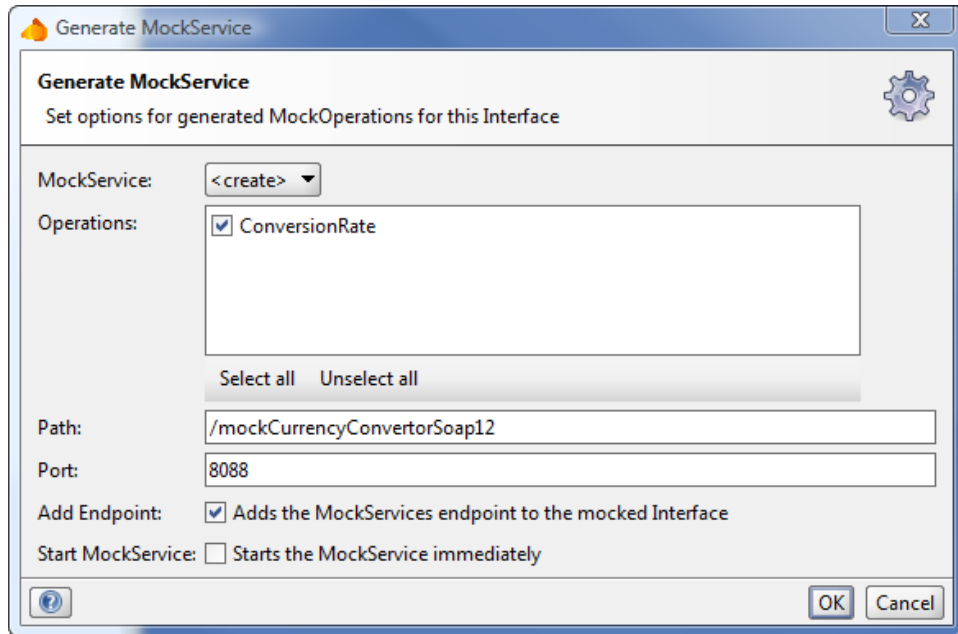


上面使用的服务接口压力比较大，如果你在打开 WSDL 页面无法打开时，Web Service 接口可能已经当掉。为了验证接口是否正在工作，可以复制上面的 wsdl 地址到浏览器中进行访问，如果你的浏览器没有收到响应信息，那么服务已经关掉了。

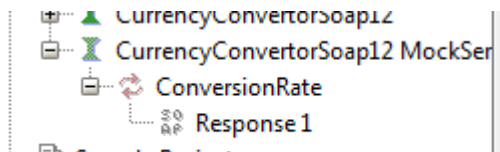
- 创建成功后，在左边窗口中可以看到名为“CurrencyConverter”的项目，右键单击其中一个接口，并在弹出菜单中选择“Generate MockService”



- 在“Generate MockService”对话框中，可以定义服务访问路径“Path”和端口“Port”，如果该接口有多个方法，方法名都会在“Operations”中列出。



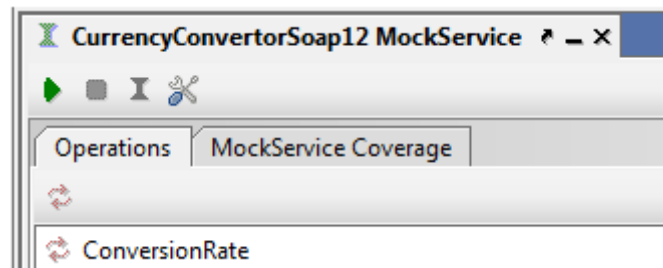
- 点击“OK”按钮后，再新弹出的对话框中填入“MockService”的名称即可，在创建完“MockService”后，你的“MockService”必须还要有一个操作和一个对应的请求。



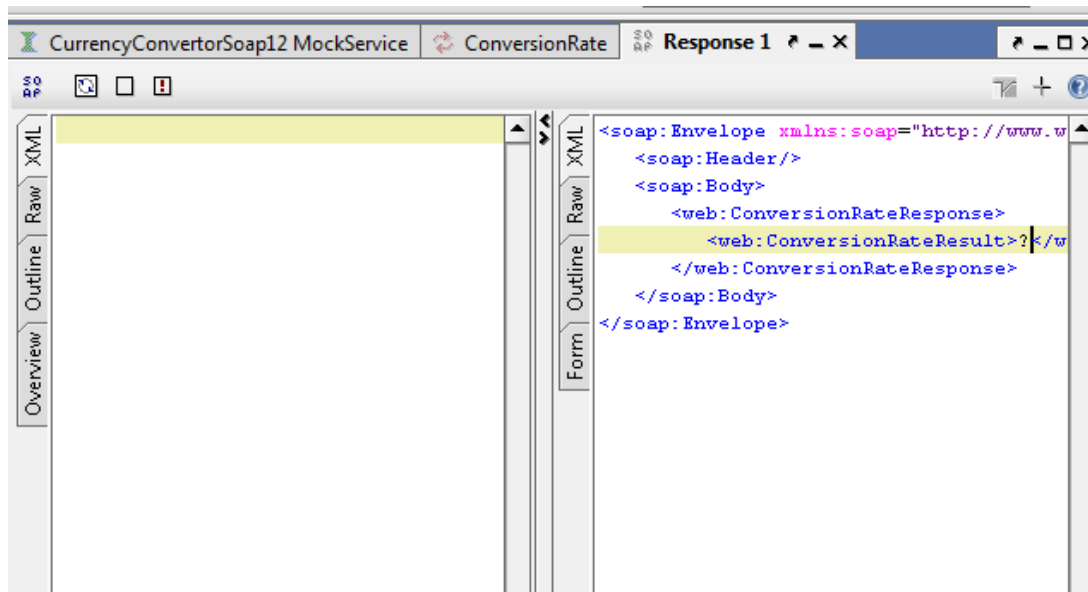
正如你所看到的，“MockService”前的图标是灰色的且尚未激活。这是表示“MockService”还没有运行，至此，我们已经成功地创建了一个“MockService”。

5.11. 编辑 MockService

- 双击上节创建的“MockService”，打开编辑器，在编辑器中，我们可以看到接口对应的所有操作，正如请求一样，响应日志会记录所有发过来被处理的请求。

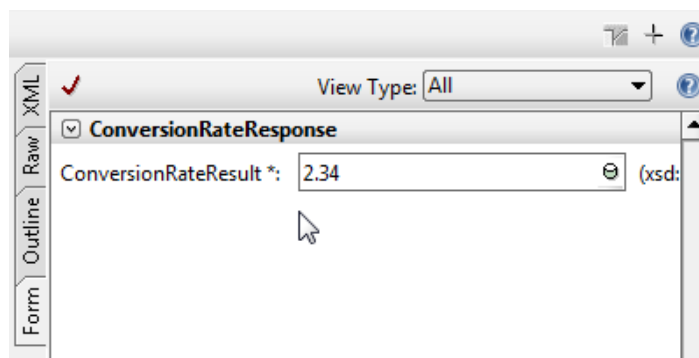


- 双击上图的操作“ConversionRate”，以查看我们在“MockService”中定义的模拟响应。正如你所看到的，我们只有一个响应，双击该响应并进行编辑。




模拟响应编辑器与请求编辑器是非常相似的，如果你有发起过请求，并成功地收到响应，此时请求编辑器接收的响应将可以直接作为参考，在打开响应页面时，会直接显示我们最后一次请求收到的响应，包括内容和 HTTP 报文头。极大地方便了模拟响应的编写。

- 选择响应报文的显示模式为“Form”（在响应的左部标签中可进行切换），编辑“ConversionRateResult”，使用其它的值代替？。




5.12. 调用 MockService

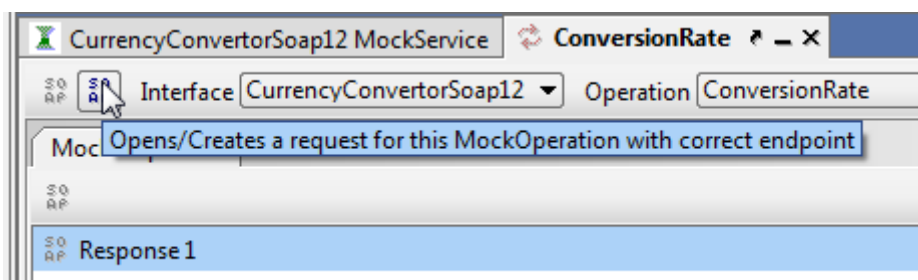
首先我们必须启动“MockService”，点击启动按钮 ，可直接在我们之前定义的路径和端口启动模拟服务。



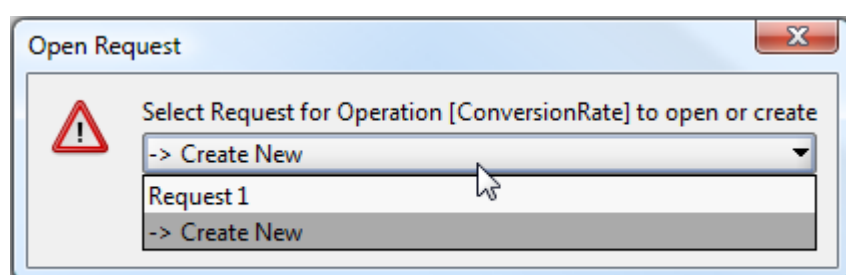
上图的红色方框的位置显示了我们的模拟服务正在运行中，此时启动的按钮变灰，而

停止的按钮可以使用，当你需要停止服务运行时，可直接点击停止按钮.

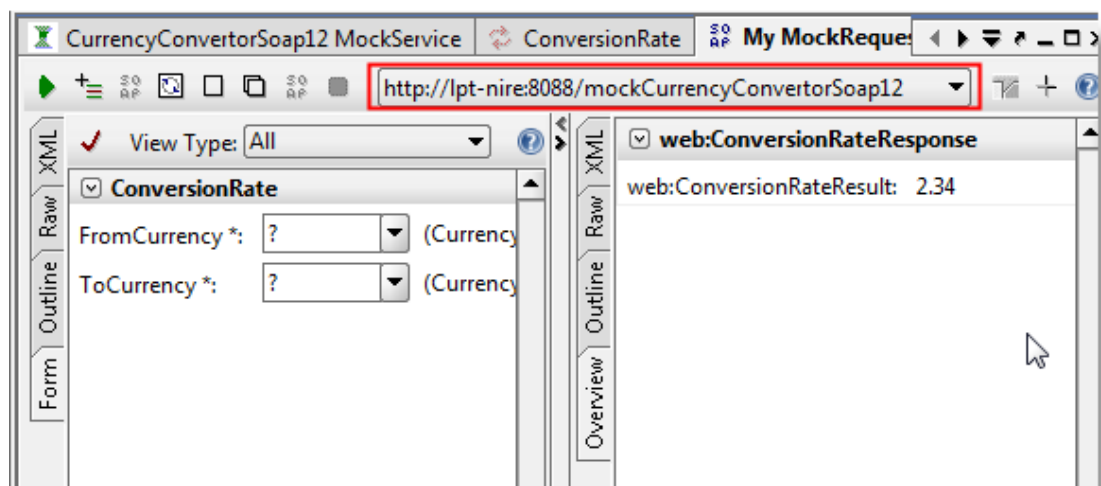
在模拟响应编辑器中，点击创建请求的按钮：



这将直接打开“Open Request”对话框，可以从下拉框中为你的操作选择已有的请求或是创建一个新的请求，并给出请求的名称即可。



当你直接打开上一步创建的请求时，SoapUI 将自动地改变请求所访问的接口地址为你所创建的“MockService”地址，点击执行查看返回的结果，可以看出响应的结果是我们之前所编辑的响应内容。



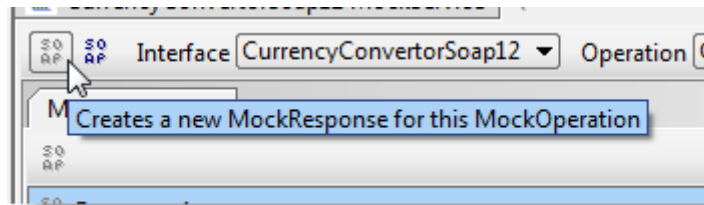
若切换回之前的模拟响应编辑器页面，你会发现请求报文已经填充了，且为刚才创建的请求，在这里可以看到提交的请求内容和响应内容。如果你需要更新或创建响应报文，可以直接进行修改，修改完成后执行请求即可，无需重启。

5.13. 自定义模拟响应

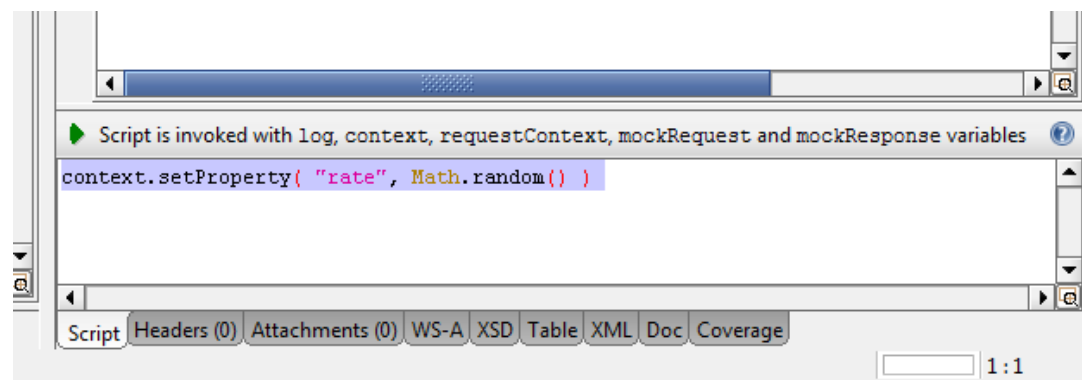
对于一个接口我们可能需要它动态地返回不同的结果，上面我们是创建了一个返回固定值的响应，但其实我们更希望一个响应可以返回动态的值，这个需求我们是通过 groovy

脚本来实现的。对于多个响应报文，当我们发送请求过去时，我们也希望“MockService”能够返回对应的响应，即我们也要有定义返回响应的需求，而不是每次都返回同一个响应。

首先，我们还需要再创建一个模拟响应：

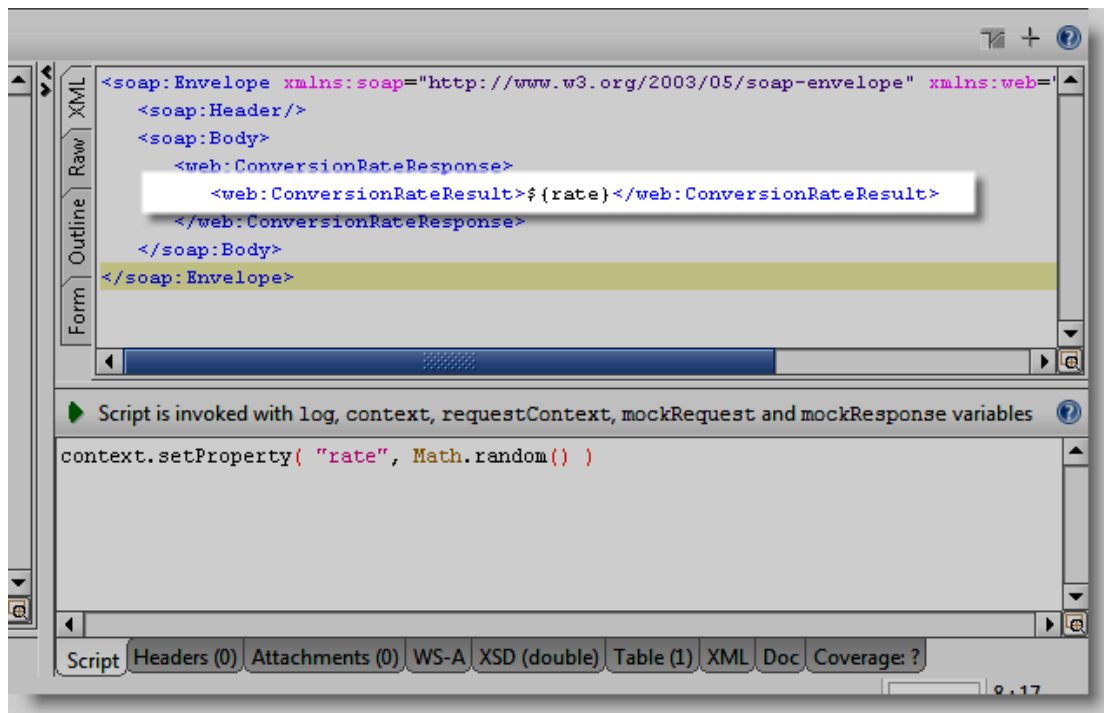


命名为“Groovy Response”（可随意），接下来需要写一个脚本来作为响应，点击“Script”标签，点开脚本编辑器，输入内容：`context.setProperty("rate", Math.random())`，如下：

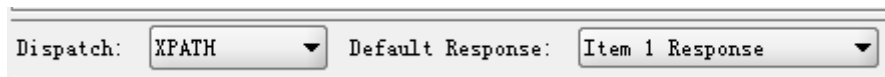


上面的脚本设置了一个名为“rate”的属性，赋给它一个随机数。

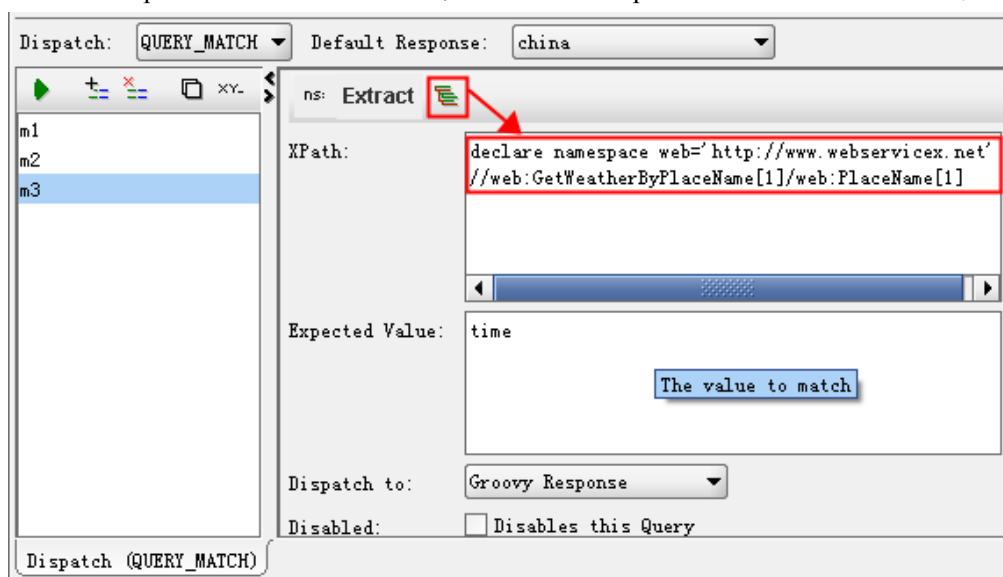
接下来，我们需要在响应报文中，把这个属性使用到返回值里，替换？为：`${rate}`，这种使用方式在 SoapUI 中被称为“Property expansion”。“Property expansion”的应用场景很广，不仅可以直接引入响应报文的任一元素中，其它的所有步骤的任何元素也都是可以使用“Property expansion”的方式来得到值。



我们已经完成了脚本，并且脚本返回的结果也会被插入到响应中，接下来我们就可以定义响应报文的分发形式了。SoapUI 中的分发机制允许你能够指定不同的模拟响应。默认的模拟响应是使用顺序分发的方式，即一个接一下。当然可以定义的更复杂一些，即让我们发送的响应完全取决于接收到的请求。打开模拟服务编辑器选择“Dispatch”方法为“XPath”，正如你所看到的，当匹配不到时，SoapUI 也提供了默认响应的设置项。



还可以设置更为复杂的方式，如选择“Dispatch”方法为“QUERY_MATCH”。“XPath”的值可以通过下图所示的图标进行选择。下图所示的配置含义为：当“XPath”所指示位置上的元素的值与“Expected Value”的值相等时，则会返回“Dispatch to”所设置的模拟响应。



6. 操作技巧

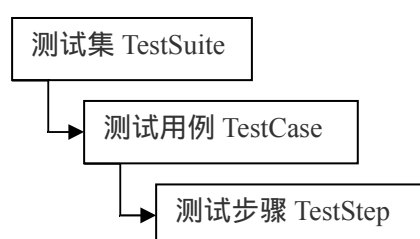
通过学习使用 Eclipse 写代码的方式进行接口测试的过程中，一方面体会到写代码方式的灵活性，另一方面也体会到写代码方式所遇到的一些问题，因此下文主要是针对性地对前一种方式列出 SoapUI 所提供的解决方案。

6.1. 右键点击

SoapUI 是一套很有尝试的测试组件，它能做很多事情，但很多人都没有发现，有很大一部分功能都是集成在右键菜单中，所以无论你在用 SoapUI 做什么操作，右击你所有操作的项目，查看弹出的右键菜单可以提供给你使用的功能，你可能会很惊讶。这虽然是个很基础的操作，但一旦你了解到这一点，使用起 SoapUI 会发现简单很多。

6.2. 测试的管理

在 SoapUI 中提供了特定的结构来管理我们的测试，通过使用这个经过实践得到的最好的结构，你可以受益良多。这样的测试结构不难，现在有很多的测试人员都在使用这样的结构。结构如下：



在 SoapUI 里，所有的测试都需要由上面的结构进行管理，这使用例的重用更简单，你可以克隆或复制测试，同样也可以很方便地生成负载测试用例，且易于管理。如果将一个请求添加到一个测试用例中，这当然也很简单，只需要右键单击请求，选择“Add to TestCase”即可。

6.3. 命名建议

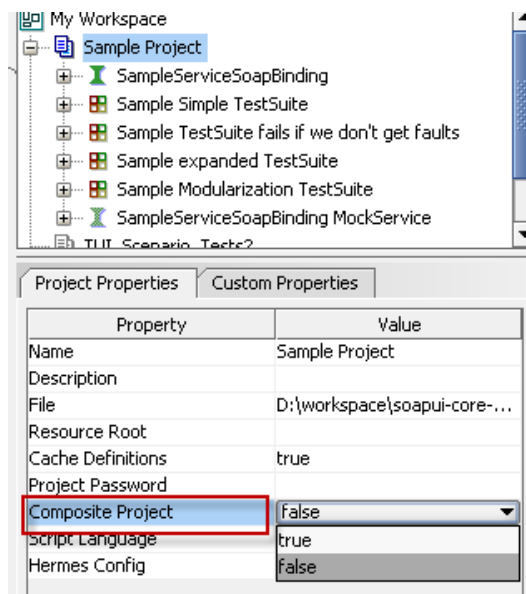
我们已经了解到使 SoapUI 定义的测试结构的重要性，接下来我们要关注的是如何提高测试的可读性。当你创建一个新的测试集时，SoapUI 会提供给你一个很简单的名字，在某此情况下这是很方便的，但当你想创建大量的测试集时，让 SoapUI 命名一个测试集如“TestSuite 3”或“TestSuite 349”可能在刚开始时，你还能记住每个用例是做什么内容，但三个月后呢？一年后呢，也许你早已经忘记“TestSuite 349”这个用例集是测什么的了。

为此，还不如花一些额外的时间去定义好你的测试集的名字，从名字就能知道他在测试什么内容，如“TS01 校验客户姓名”。当然测试用例、测试步骤的命名也应遵循易懂的规范 SoapUI 关于标题长度的限制，基本能够符合我们的需求，可以很长，长到足以让你清楚地描述你所想要的测试名称。

6.4. 共享操作管理-SVN

创建完一个项目，在保存时，一般会被默认保存为一个 XML 文件，而这个文件包含接口、测试、模拟服务、脚本等所有的信息。如果项目不包含任何外部文件的引用，那么它的项目文件是可以很方便地进行传递。如果你有一个很大的项目需要用 SoapUI 来进行测试，那么就会有多个测试人员同时进行编写用例的操作，除了通过 SVN 来进行项目脚本的同步外，当一个测试人员在编写用例时，由于是对同一个文件进行操作，那么另外一个测试人员就必须等待这个测试人员写完才能开始进行另一个模块的编写，否则在代码合并的时候对于单一的一个文件势必会有很多代码冲突而不利于维护。

为了解决上面所说的场景，SoapUI Pro 添加了一个 Composite Project 的属性来解决这个问题：

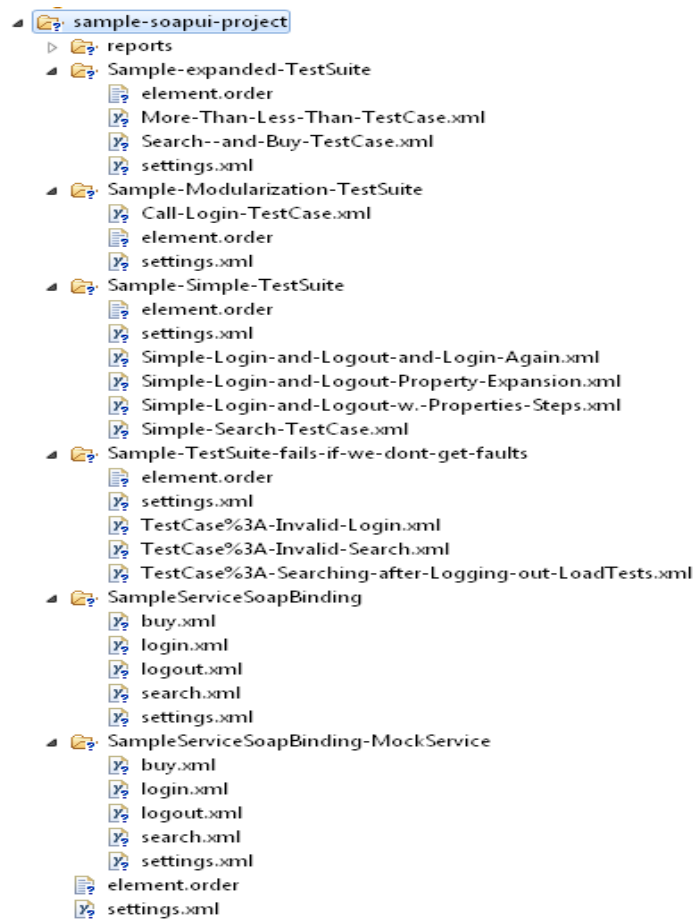


将 Composite Project 属性值从 false 改为 true，然后保存项目，结果我们可以看到 SoapUI 帮我们把项目保存成为如下形式：

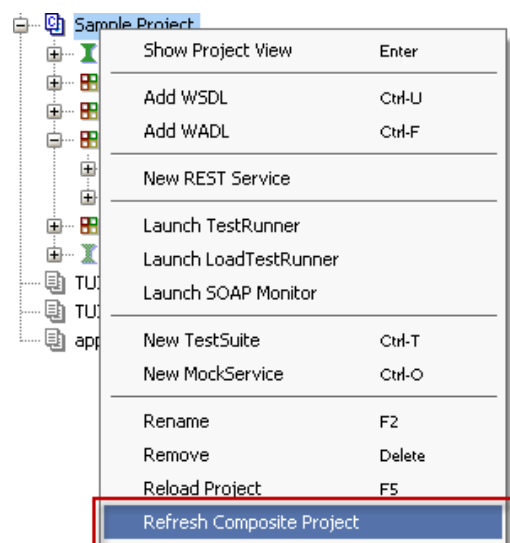
<项目的文件夹>

- * settings.xml – 项目级别的配置文件
- * element.order – 保存项目所有测试集执行顺序的文件
- * reports folder – 包含自定义的报告和子报告
- * <测试集对应的文件夹> - 每个测试集会有一个独立的文件夹
 - o settings.xml – 测试集的配置文件
 - o element.order – 保存测试集中所有测试用例执行顺序的文件
 - o <测试用例对应的 xml 文件> - 每个测试用例会有一个独立的 XML 文件，包含测试用例和其相关的配置，测试步骤和负载测试
- * <MockService 对应的文件夹> - 每个 MockService 会有一个独立的文件夹
 - o settings.xml – MockService 的配置文件
 - o <模拟操作对应的 XML 文件> - 包含所有的配置和模拟响应
- * <WSDL 接口对应的文件夹> - 每个 WSDL 接口会有一个独立的文件夹
 - o settings.xml – 接口的配置文件
 - o <操作对应的 xml 文件> - 包含所有的配置和操作发出的请求

例如，将例子“Sample Project”这个项目，设置 Composite Project 属性为 true 后，文件的层级关系显示如下图：

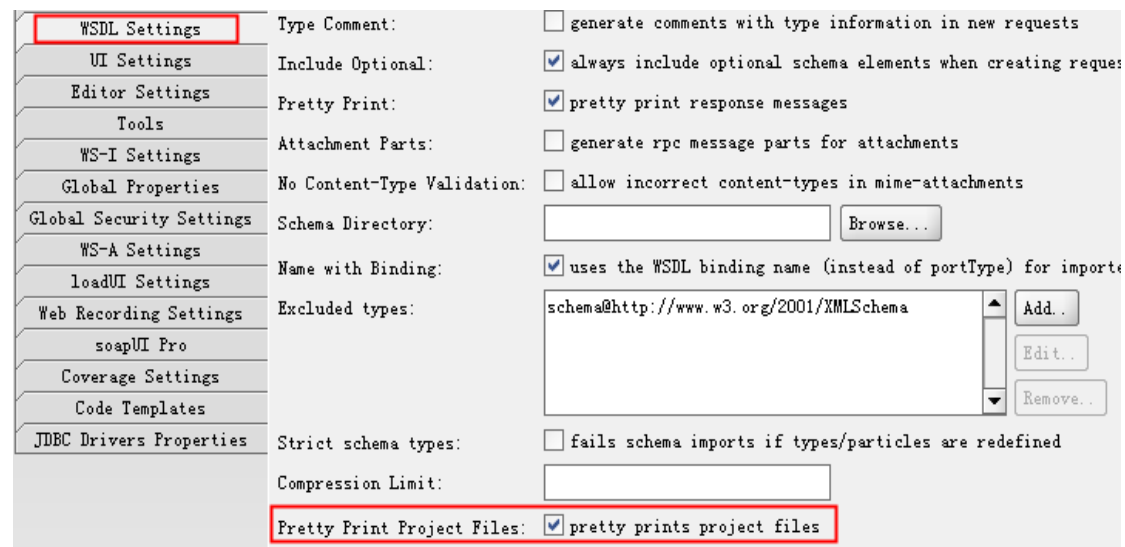


经过这样的转换，我们就可以通过 SVN 来管理测试代码的版本了，每个测试人员可根据实际分配到的模块并行地编写测试代码，编写完成后提交到服务器，其它测试人员可以通过 SVN 同步最新的代码到本地磁盘，然后在 SoapUI 中通过右击项目名称，选择弹出菜单项：Refresh Composite Project 来完成代码从本地磁盘更新到 SoapUI 编辑器这一过程：



如果希望在之后，将项目可以简单快速地又重新合并为一个 XML 文件，那么可以设

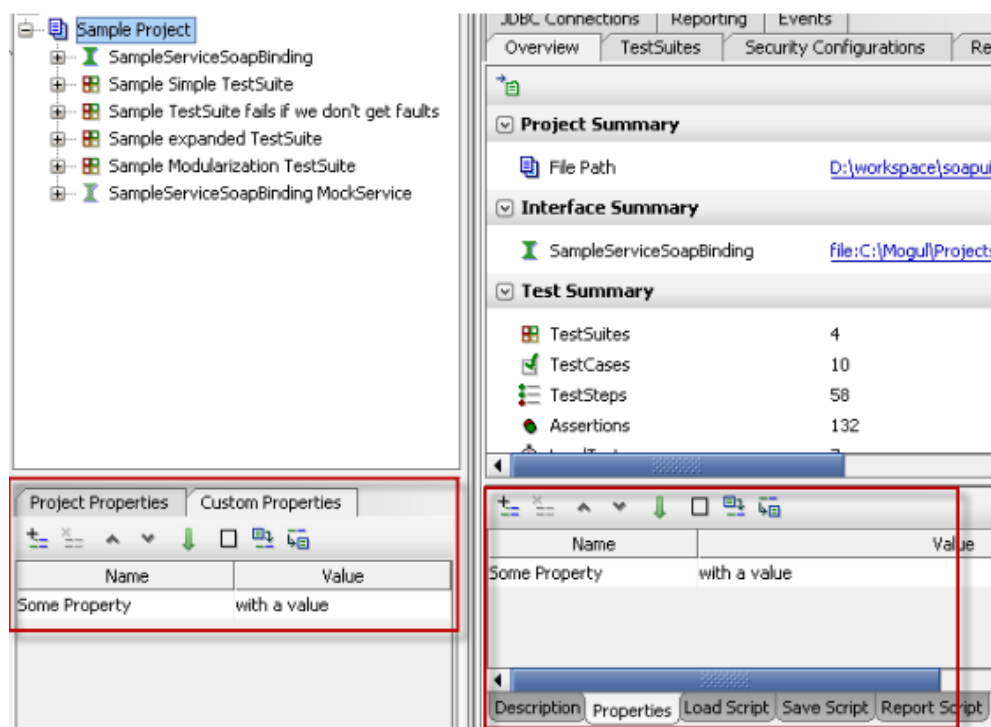
置 File-Preferences-WSDL Settings，勾选属性 Pretty Print Project Files（但这会增加项目文件的大小）。



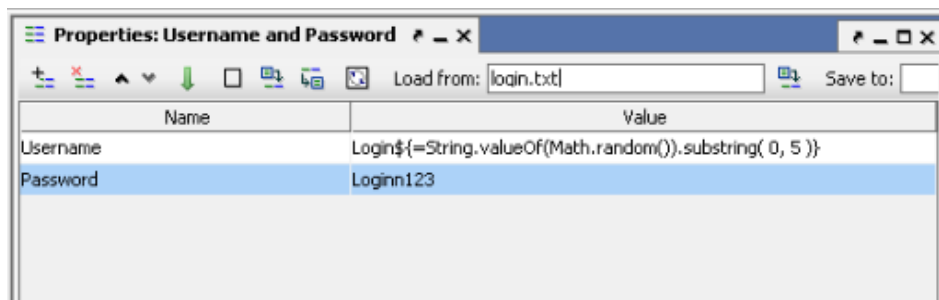
6.5. 属性操作

属性是 SoapUI 中一个非常重要的特性，因为有关属性的定义和操作尤为重要。我们可以在以下列出的范围中自定义属性值：

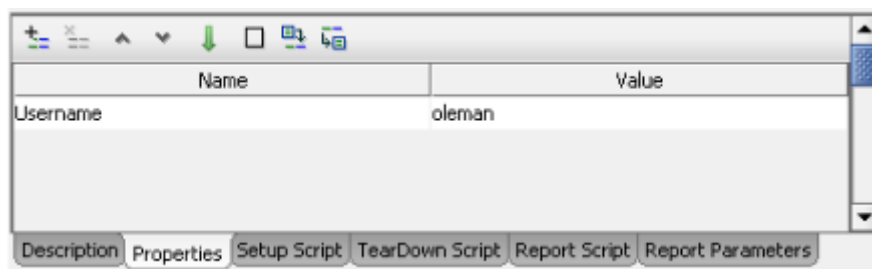
- ✓ 在项目、测试集、测试用例范围中，可以通过相应的属性标签页来添加属性



- ✓ 在 Properties 测试步骤中



- ✓ 在 DataGen 测试步骤中
 - ✓ 以下两个测试步骤中的配置：
 - 在 DataSource 测试步骤中，执行数据驱动测试的场景
 - 在 DataSink 测试步骤中，将属性值保存到一个外部存储介质的场景
- 进一步，大部分的测试步骤也提供了一些属性可用来读写，如：
- ✓ Script 测试步骤提供了一个名为 Result 的属性，它的内容为最后一次执行该脚本所得到的值，且脚本中提供了一些方法用来读写属性，如测试集定义了一个属性：



我们可以通过下面的代码访问该属性值：

```
// get username property from TestSuite
def username=testRunner.testCase.testSuite.getPropertyValue( "Username" )
```

也可以通过下面的代码改变该属性值：

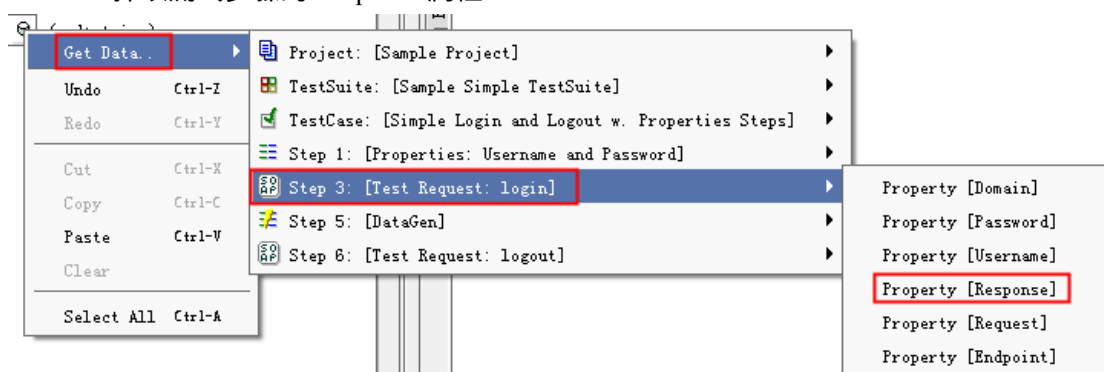
```
// write the username to the HTTP Request
testRunner.testCase.testSteps["HTTP Request"].setProperty( "Username", username )
```

- ✓ 所有的 Request 步骤都提供了一个 Response 属性，它的内容为最后一次接收到的响应报文

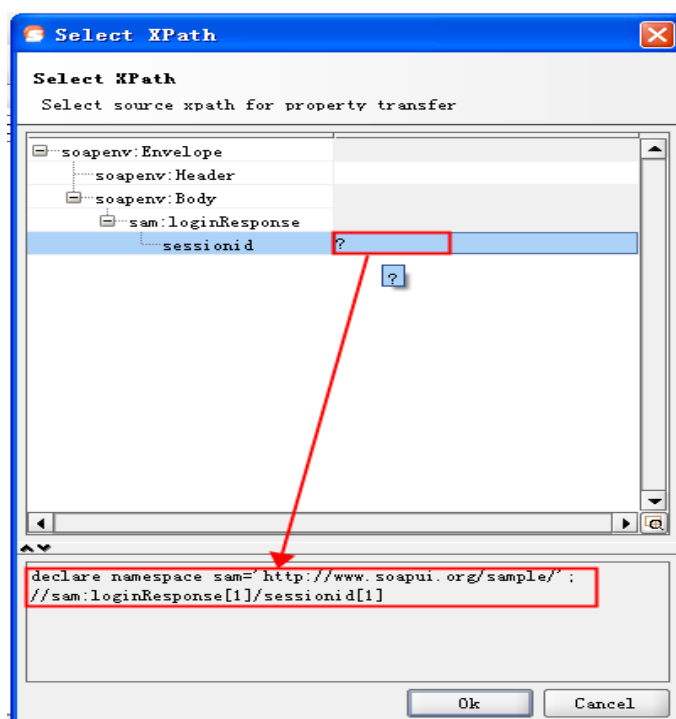
在 SoapUI 中继断言之后最为经常使用的另一个特性就是属性传递操作，最常使用的场景是：你希望将响应报文中的值传递给另一个请求，举个例子，你登录完之后会获得一个 sessionId，在这之后的一系列请求都必须用到这个 sessionId，SoapUI 提供了两种方式让我们来解决这个问题：属性传递（“Property Transfer”）和（“Property expansion”），这两种方式使用中有一些差别，这里先介绍使用方式。在 SoapUI Pro 版本中使用这个功能是相当简单的，在 SoapUI 开源版本中使用这个功能会稍难一些，接下来我们将来学习怎么来使用它。

下面是之前例子的快照，看一下是如何来完成属性传递（“Property Transfer”）操作的。

择该测试步骤的 Response 属性：



- 2、之后页面会弹出“Select XPath”步骤，点击 ? 所在的区域，页面下方会出现该元素的位置，点击 OK 即可。



经过上面的介绍，对这两个操作的使用应该都可以清楚了，笔者认为这两个操作的使用场景是有区别的，“Property Expansion”比较方便，适用于当一个变量会在之后很多个测试步骤中使用，用这种方式可以更加快捷，但可读性差一些，而“Property Transfer”需要先定义一个测试步骤，再为每个传递定义一次变量，并做完上面的 3 步操作，但相应可读性强一些。

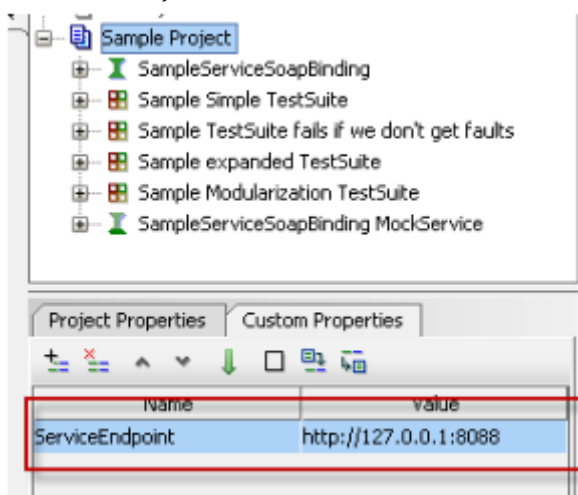
6.6. 接口变化

接口的变化，相信做过接口测试的人都会遇到，接口测试环境迁移了，接口的出入参变化了等等，像这样子的情况，我们都需要去修改代码中相应的配置文件，去修改代码中接口的调用和返回等。针对这样的情况，SoapUI 也给我们提供了相应的解决方案：

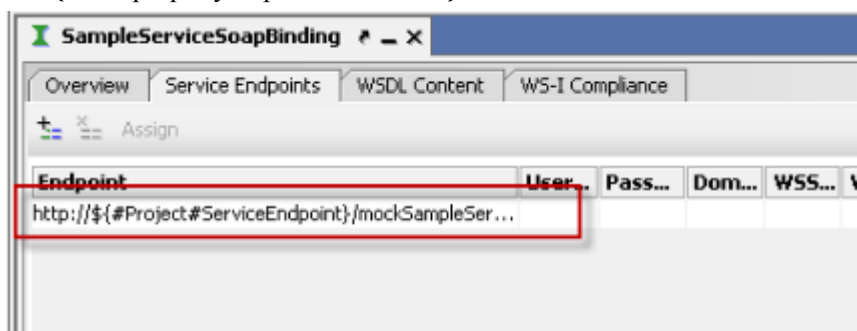
地址改变

- 1、给项目定义一个属性，用来存放服务的地址（在开始编写用例时，就需要进行设

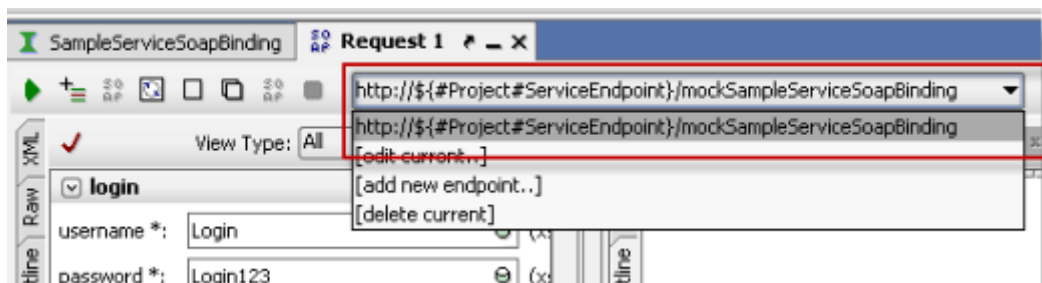
置，这里推荐第一种方式）：




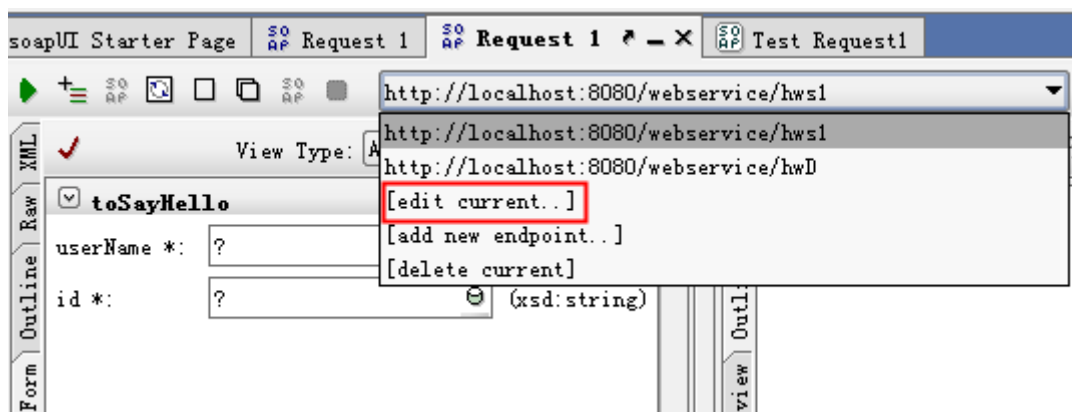
- 2、双击项目名称，打开项目概要窗口，切换到“Service Endpoints”标签页，配置属性的值（使用 property-expansion 的方式）：



- 3、确保请求中的 URL 地址是正确的，则之后创建的请求都会指向上图所配置的地址：

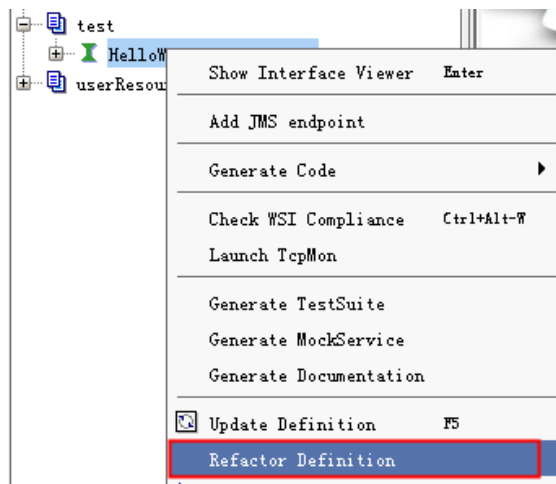


- 4、如果在项目进行到一定阶段了，发现接口地址有变化，可以打开接口（接口的图标为 ）下方的请求，通过使用“edit current”选项，直接编辑当前的接口地址。

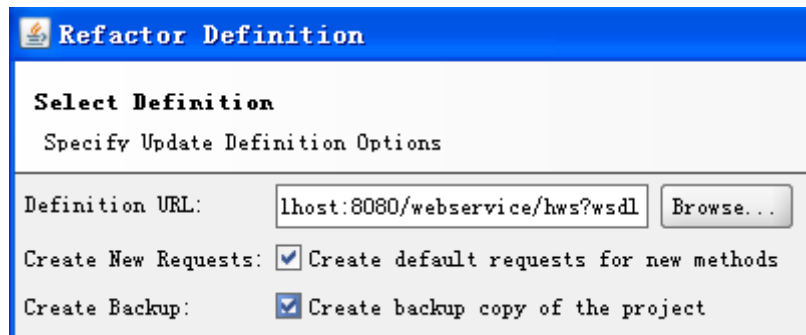


入参改变

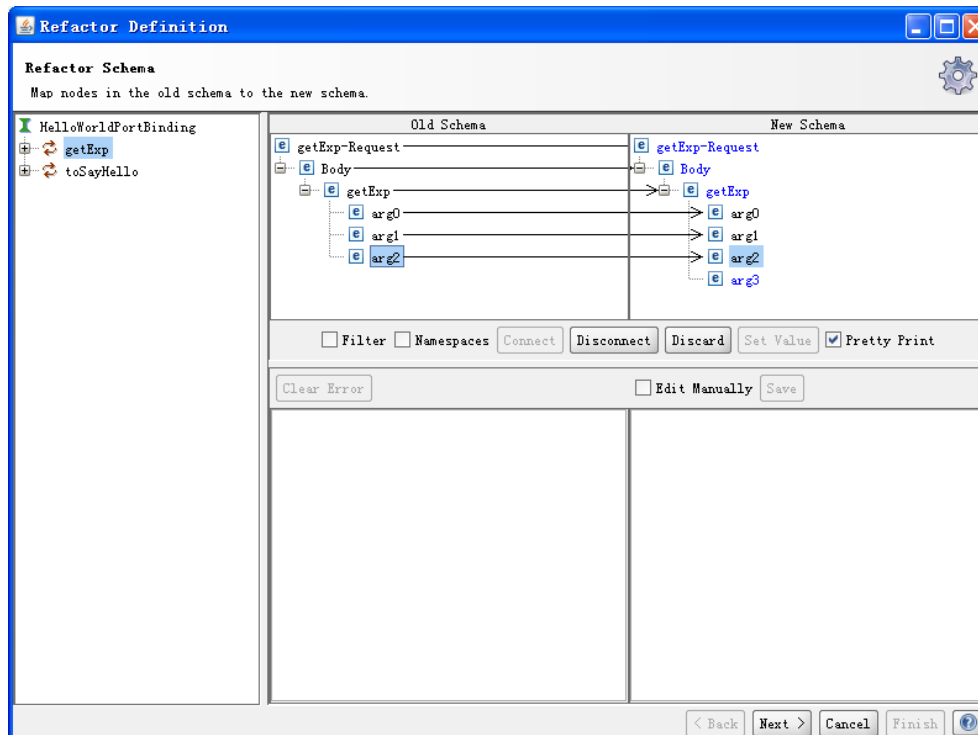
- 1、右键单击接口名称，选择弹出的右键菜单“Refactor Definition”选项：



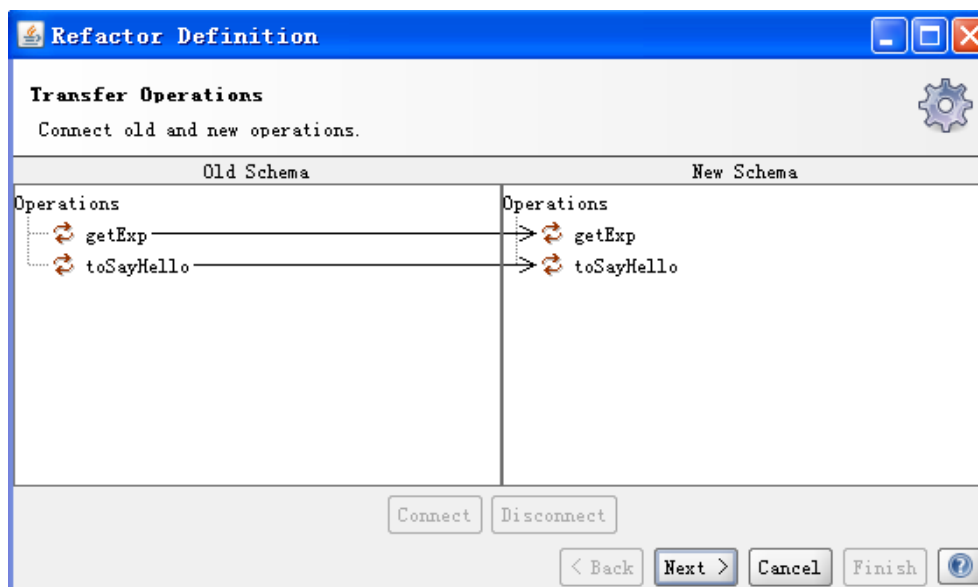
- 2、下一步，确定 URL 是所测试的服务接口地址，根据个人需要勾选下面的两个选项，一般建议都勾选：



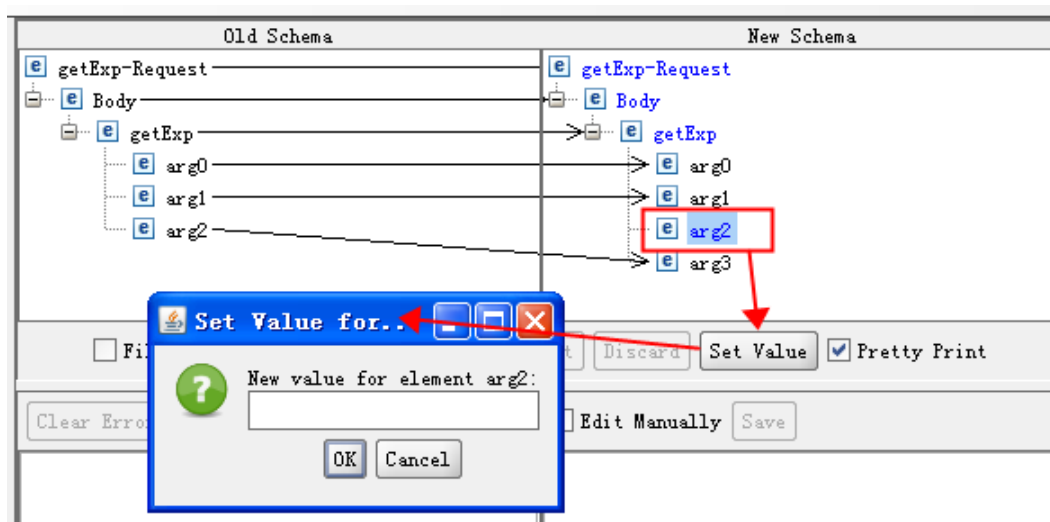
- 3、在“Refactor Definition”配置页面，我们可以看到对应的 WSDL 中有几个接口，通过左边栏，选择接口，查看是否有参数的变化，下图中右边上半区域分为两块内容，一块为“Old Schema”，列出旧接口的入参，另一块为“New Schema”，会列出新接口的入参，我们可以根据需要重新进行配置：



- 4、下图，当方法名称变了，可以通过配置下面的步骤快速地迁移测试用例，（单击旧的方法名拖动到新的方法名即可完成连线操作）：

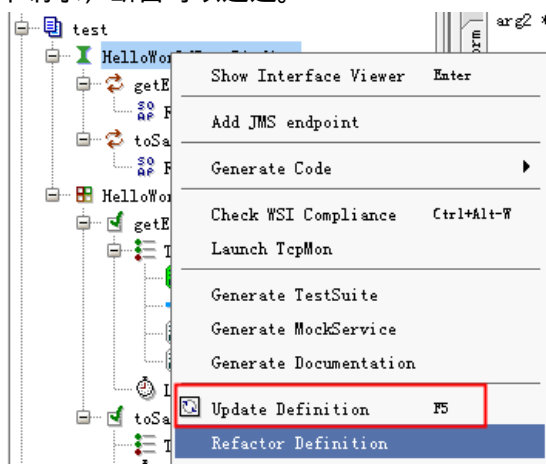


- 5、下图，我们将原接口的入参 arg2 中的值赋给入参 arg3（操作上，只要将鼠标点选旧参数，再拖动到新参数即可完成连线操作），而新接口的入参 arg2，我们可以使用“Set Value”快速地给接口指定一个默认值，确定后，SoapUI 会帮我们把所有原来已经设计好的测试用例，将本来是 arg2 的值赋给 arg3，将 arg2 的值设置为下图所设置的值，快速地帮我们重建测试用例的入参：



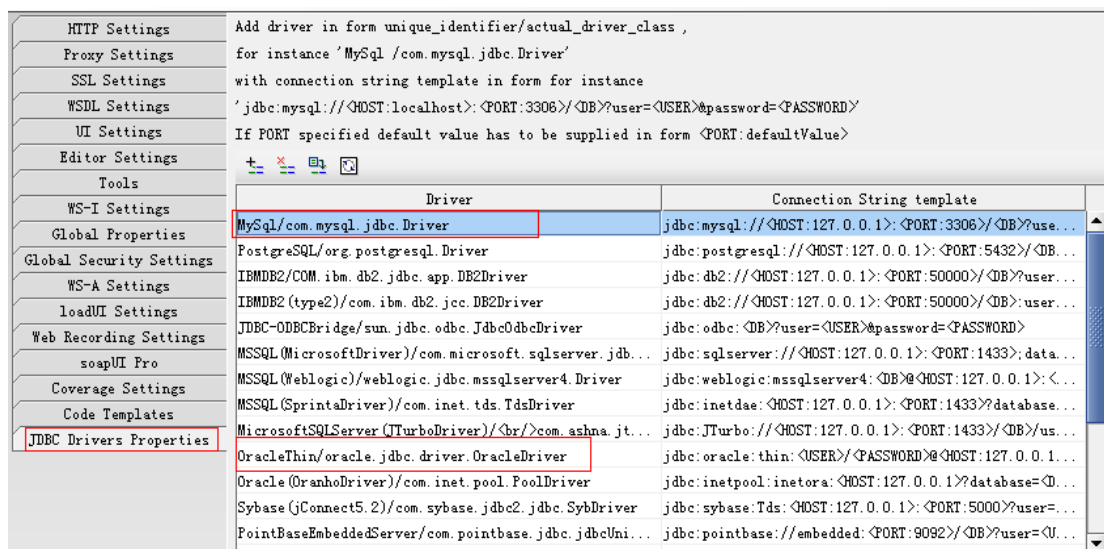
出参改变

- 1、对于出参的变化，笔者建议使用下面的方式进行处理，首先，给测试用例的请求加上一个“Schema Compliance”断言，如果返回的出参结果与之前的定义不一致时，这个断言会失败，当这个断言失败的时候，我们可以很快就清楚是由于出参发生了变化。
- 2、当出参发生变化时，我们不需要像入参改变时，做重新映射的步骤，只需要右键单击接口名称，选择弹出菜单中的选项“Update Definition”，更新一下 WSDL 即可这时再运行一下请求，断言可以通过。

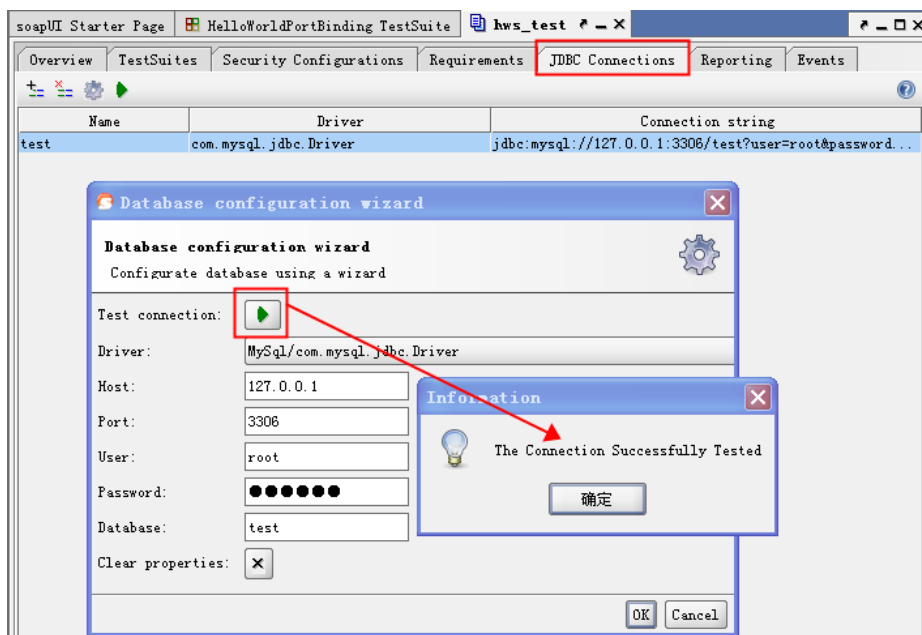


6.7. 数据库操作

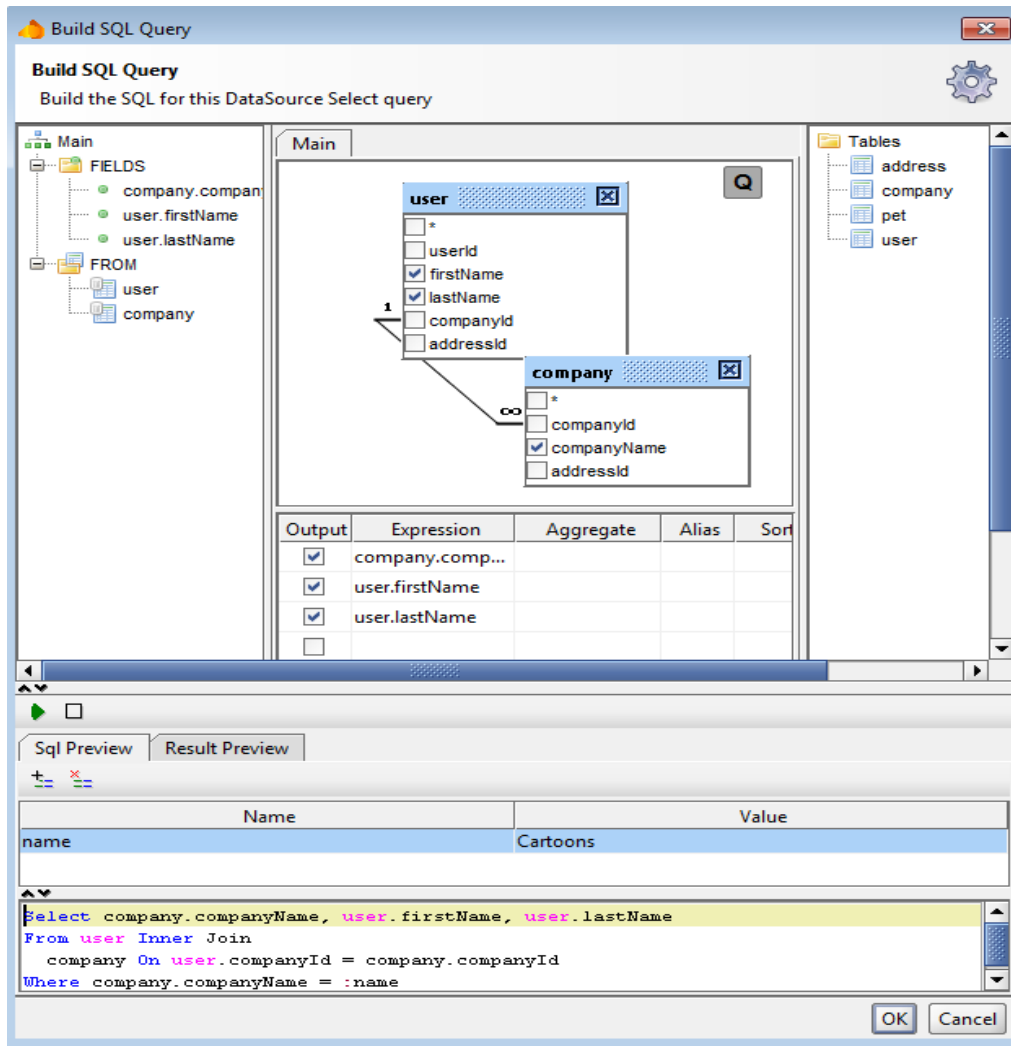
在项目中提供了一个用来配置 JDBC 数据库连接的选项，因此你可以在测试中使用 JDBC 数据源，JDBC 数据接收器（JDBC DataSink）和 JDBC 请求步骤。为了能够配置数据库连接，就必须有驱动程序和连接串，可通过菜单 File-Preferences-JDBC Drivers Properties 配置，里面已经提供了一些连接串模板，你也可以根据需要添加新的连接串。一般较为经常使用的数据库主要是：ORACLE 和 MYSQL 两种数据库，驱动程序分别为：ojdbc14.jar 和 mysql-connector-java-5.1.13-bin.jar，在使用 JDBC 前需要先把驱动程序放到指定位置：“D:\Program Files\eviware\soapUI-Pro-3.6\lib”文件夹下。



驱动程序放好后，须重启 SoapUI 后方可使用。接下来，我们通过双击项目名打开项目概要信息，切换到“JDBC Connections”标签页，通过 新增数据源，如下图，我们创建了一个 Mysql 的数据源，填入用户名、密码、数据库，再使用 测试数据源是否可连接，连接测试成功则返回成功提示。

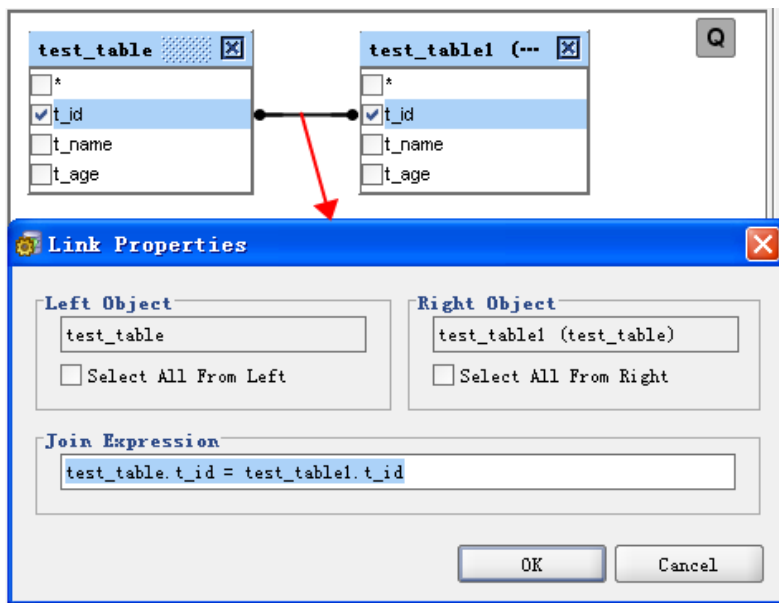


创建完数据源后，页面会提示是否要生成 SQL 语句，确定后进入 SQL 配置页面：



在上图的 Main 窗口，右键点击空白区域弹出菜单，或添加对象、表、添加或复制连接操作等。

- ✓ 表连接：当两个对象（表）会使用一个外键进行关联时，需要添加一个连接关系，可通过单击其中一个对象的字段，拖拽到另一个对象的字段上，即可完成连接操作，右击连接，可以进行相应的设置。



同时该窗口也提供了以下的功能：

- ✓ 设置对象的别名
- ✓ 选择输出字段
- ✓ 排序
- ✓ 定义查询条件
- ✓ 定义 group by 字段
- ✓ 插入子查询：可定义一个子查询作为某字段的查询条件

Outp...	Expression	Aggreg...	Alias	Sort Type	Sort Or...	Grouping	Criteria	Or...	Or...
<input checked="" type="checkbox"/>	company.co...					<input type="checkbox"/>	= :name		
<input checked="" type="checkbox"/>	user.firstName					<input type="checkbox"/>			
<input checked="" type="checkbox"/>	user.lastName			Ascendi...	1	<input type="checkbox"/>			
<input type="checkbox"/>						<input type="checkbox"/>			

窗口底部包含两个标签页：

- ✓ SQL preview：可以查看由上面的图表连接所生成的 SQL，可以添加属性用来作为 SQL 语句的条件。
- ✓ Result preview：显示了当前 SQL 的查询结果。

Sql Preview		Result Preview (5 rows)	
companyName	firstName	lastName	
Cartoons	Merry	Popins	
Cartoons	Micky	Mouse	
Cartoons	Minie	Mouse	
Cartoons	Ben	Ten	

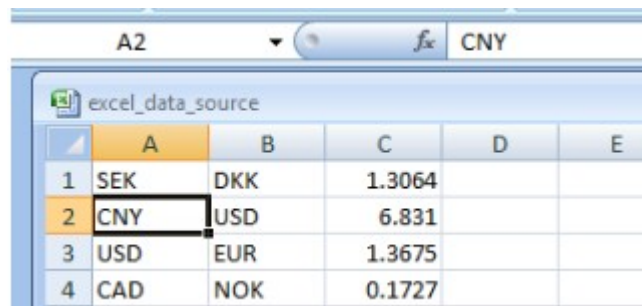
6.8. 数据文件操作

数据文件的操作在测试中是使用频率非常高的一个功能，你可能会将测试数据（输入、


预期输出等）存储到一些外部文件（数据库、Excel、XML 文件等）上，然后在测试中反复地使用这些数据来运行测试用例。例如，需要测试一个用户查询信息的服务，我们有用户的名单，这些名单存储在数据库或外部文件上，这时你就需要使用这些数据做为接口的入参来测试接口的功能。

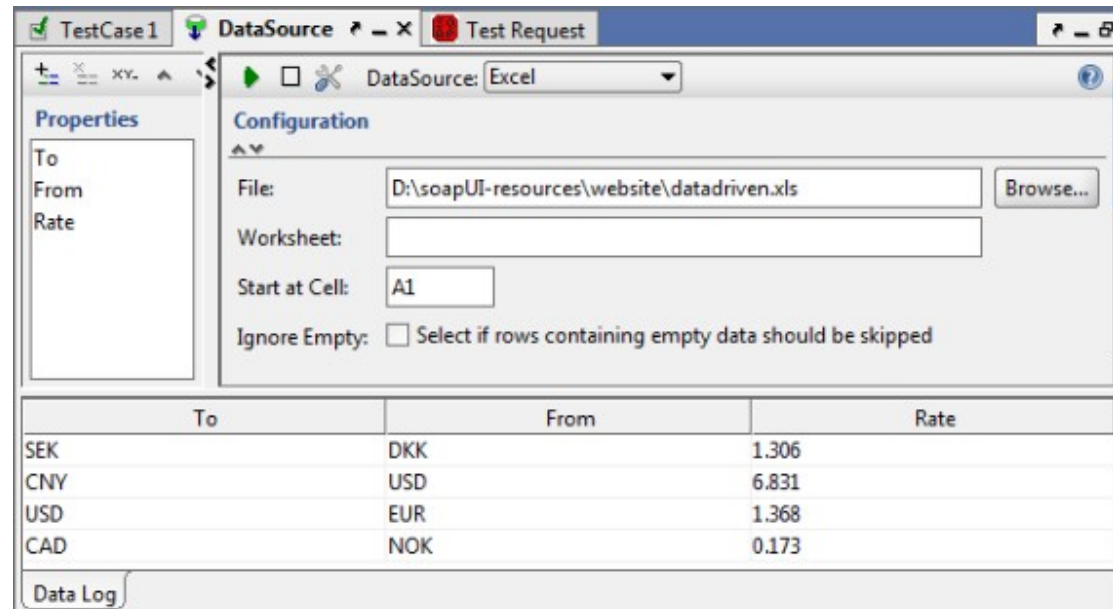
在 SoapUI Pro 版本，创建一个数据驱动测试脚本是非常容易的，“DataSource”测试步骤可以用来将外部测试数据（数据库、Excel、XML 文件等）读入到 SoapUI 的标准属性中且这些数据在后续的请求中可以通过“Property-Transfers”或“Property-Expansions”的方式直接使用，然后通过一个“DataSource Loop”测试步骤来循环地读取数据源“DataSource”的值，并执行相应的测试步骤。在本节中，我们主要介绍数据文件的读写操作。

首先我们需要有一个 Excel 文件，里面需要有对应的数据内容：



	A	B	C	D	E
1	SEK	DKK	1.3064		
2	CNY	USD	6.831		
3	USD	EUR	1.3675		
4	CAD	NOK	0.1727		

创建一个“DataSource”步骤，下拉框中选择“Excel”（根据要导入的 Excel 中的列数，需要配置相应的属性个数），在下拉框中选择“Excel”，使用“Browse”选择 Excel 文件，设置 Cell（Excel 中第一个数据的位置），点击  执行测试读取数据源操作，查看在底部“Data Log”区域是否返回数据：



Properties

To
From
Rate

Configuration

DataSource: Excel

File: D:\soapUI-resources\website\datadriven.xls [Browse...]

Worksheet:

Start at Cell: A1

Ignore Empty: ☐ Select if rows containing empty data should be skipped

To	From	Rate
SEK	DKK	1.306
CNY	USD	6.831
USD	EUR	1.368
CAD	NOK	0.173

Data Log

下面将不再详细介绍各种数据源的创建过程，直接贴出图以供大家参考，下图为 Groovy 的方式：

DataSource: Groovy		
Configuration		
<pre> result["p1"] = Math.random() result["p2"] = Math.random() result["p3"] = Math.random() </pre>		
p1	p2	p3
0.5835844717857255	0.0080526346361115	0.8554820520643669
0.08338020384402434	0.6300404209440476	0.7379197748924214
0.6457519894301996	0.819482643010397	0.28346056840477274
0.3931859693906665	0.33135568344102684	0.22294988810545646
0.18118570540033052	0.29473569596485705	0.8571086837171448
0.6629576967142092	0.3970423465023155	0.5508595123874068
0.642358624311169	0.03310944407286731	0.7353330271890406
0.7285439526974317	0.2555681381471667	0.7025992538075703
0.26795328525752593	0.4578874848200336	0.81751369714604
0.2283775430317656	0.9432793315729869	0.30522703251357

下图为 XML 的方式：

DataSource: XML	
Configuration	
Source Step:	login - Request 1
Source Property:	Response
Row XPath:	declare namespace sam="http://www.soapui.org/sa //sam:loginResponse[1]/sessionid[1]
Column XPath:	a/text()

下图为外部文件（txt）的方式：

DataSource: File	
Configuration	
File:	C:/Documents and Settings/Owne Browse...
Separator:	,
Charset:	gbk
Trim:	<input checked="" type="checkbox"/> Trim values
Quoted Values:	<input type="checkbox"/> Values are quoted

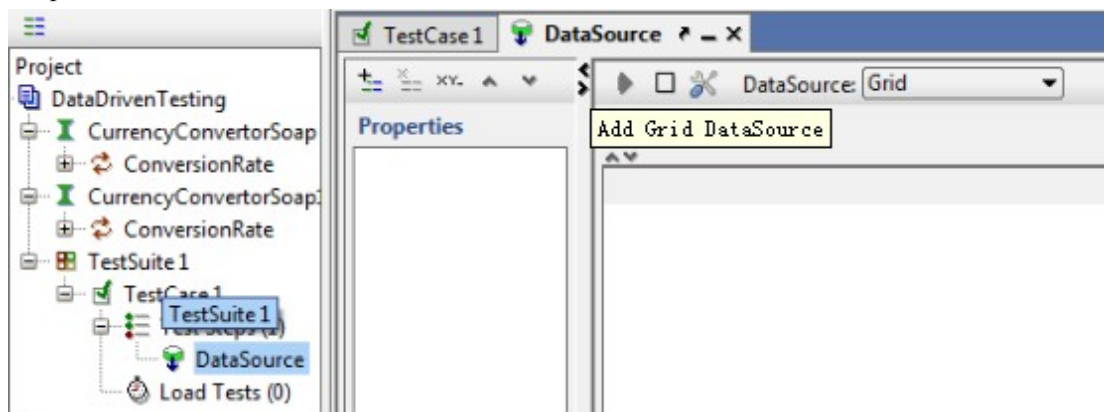
a	b	c	d


6.9. 循环入参

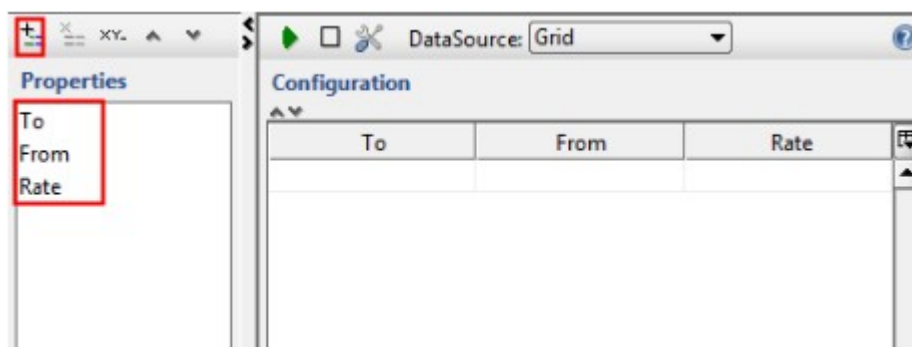
通过上一节介绍，我们了解到，使用循环读取数据源时，主要遵循以下三个步骤：

“DataSource”步骤

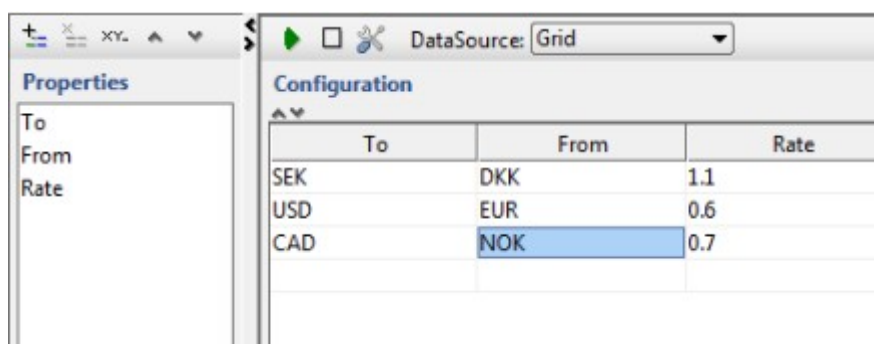
读取外部文件的数据，并赋给属性，首先，我们创建一个空的测试用例，然后添加一个“DataSource”步骤，从下拉框中选择“Grid”的数据源形式（这种形式的数据源，可直接在 SoapUI 中添加数据，无须从外部文件中读取）：



接下来我们添加了 3 个属性到数据源中：To, From, Rate, 通过  按钮添加：

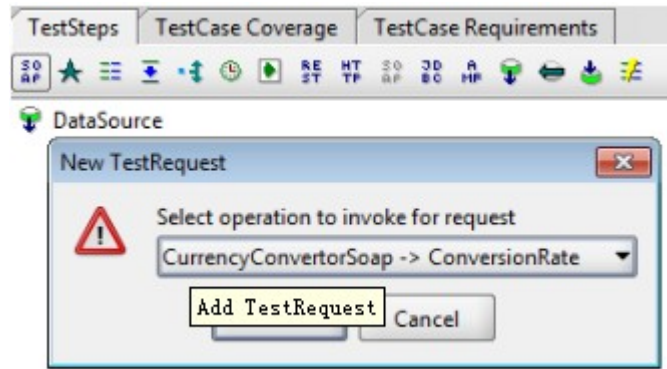


当我们添加一个属性时，对应的右边区域会显示网格状的数据，之后，我们可以直接在上面添加数据，这样就完成了一个数据源的创建过程：

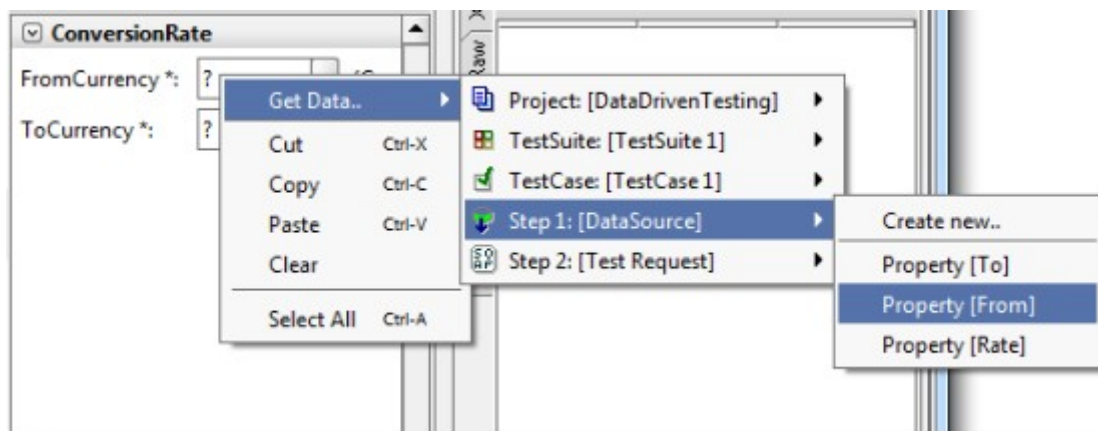


TestSteps（可有任意个）

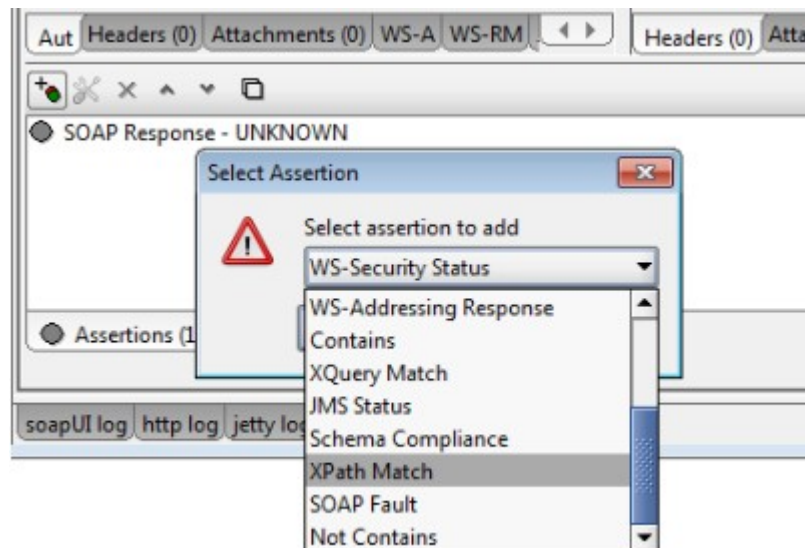
通过读取 1 中的属性，作为服务接口测试步骤的入参。我们需要添加一个请求到用例中，用来访问所要测试的服务接口，选择创建 SOAP 请求，然后选择我们所要测试的方法，点击“OK”后，创建成功：



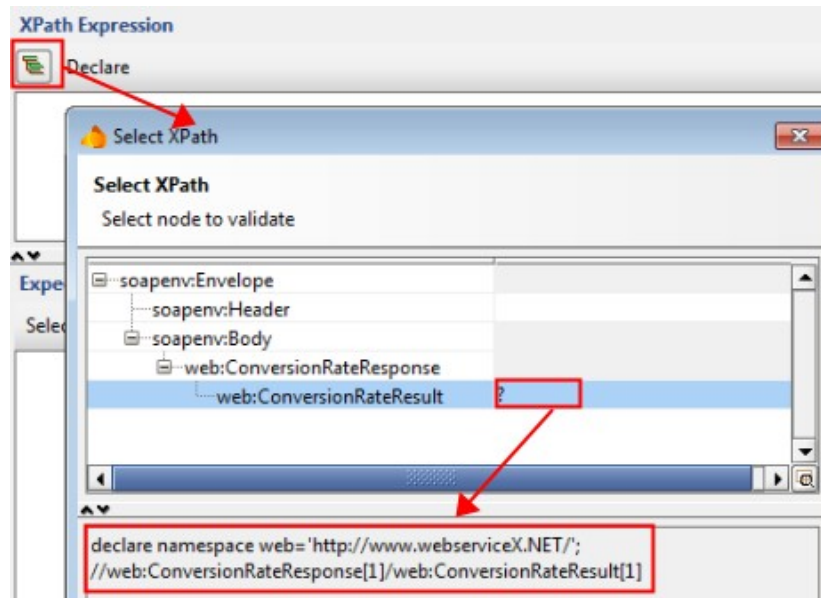
请求被添加到用例中后，我们需要打开请求，给请求填入参值，入参中我们设置它是从“DataSource”步骤读取：



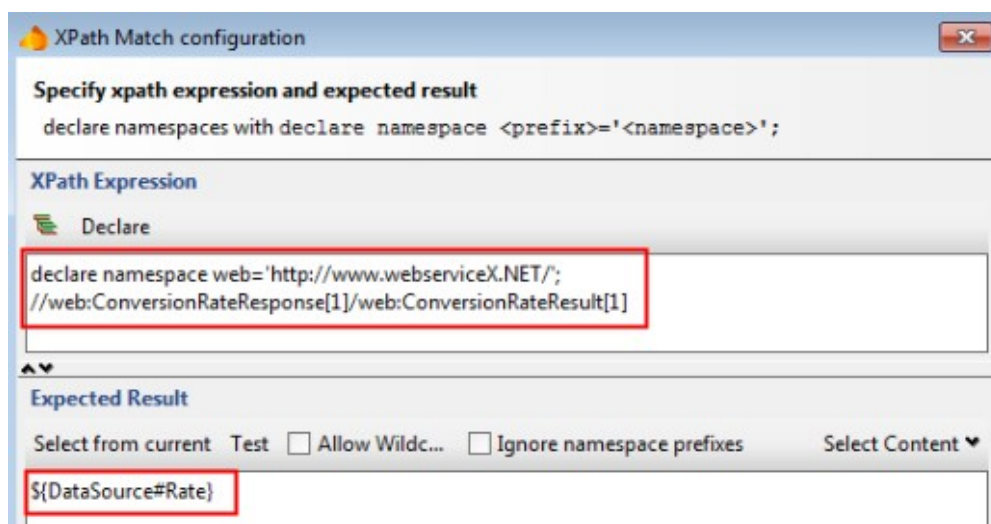
选择好两个入参好，我们还需要添加“XPath Match”断言，确认服务接口所返回的汇率是正确的：



在弹出的配置窗口中，我们选择了节点“web:ConversionRateResponse”，即我们所关心的汇率值：



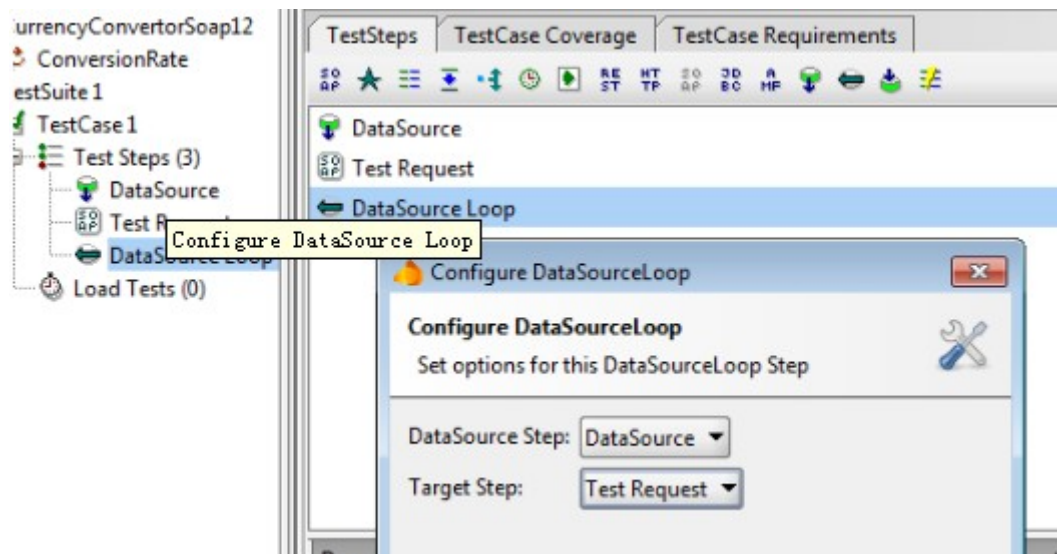
在选择好要校验的元素后，我们还需要配置校验的内容，此处，我们的预期值是前“DataSource”步骤的“Rate”属性，因此配置后的界面如下：



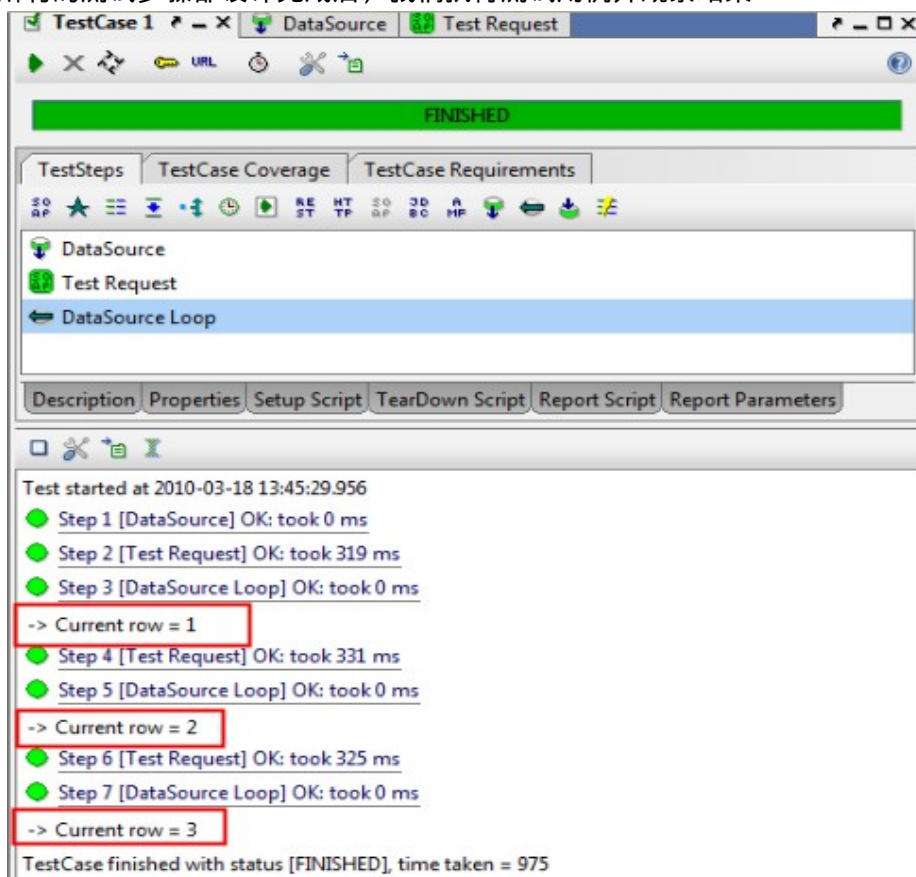
“DataSource Loop”

从“DataSource”一次读取一行数据，循环执行上面所创建的测试步骤，循环次数由“DataSource”中的数据行数决定。通过上面的配置，我们的请求入参读取了“DataSource”中的属性“To”，“From”，而断言的判断值使用了“Rate”，所缺的只有一个“DataSource Loop”步骤。

添加一个“DataSource Loop”步骤，设置“DataSource Step”和“Target Step”的值，分别表示要使用哪个数据源进行循环，读取数据后要执行哪个测试步骤（只有加了这个步骤，才能循环地取到所有的测试数据，否则永远只能读到第一行）：



在所有的测试步骤都设计完成后，我们执行测试用例并观察结果：

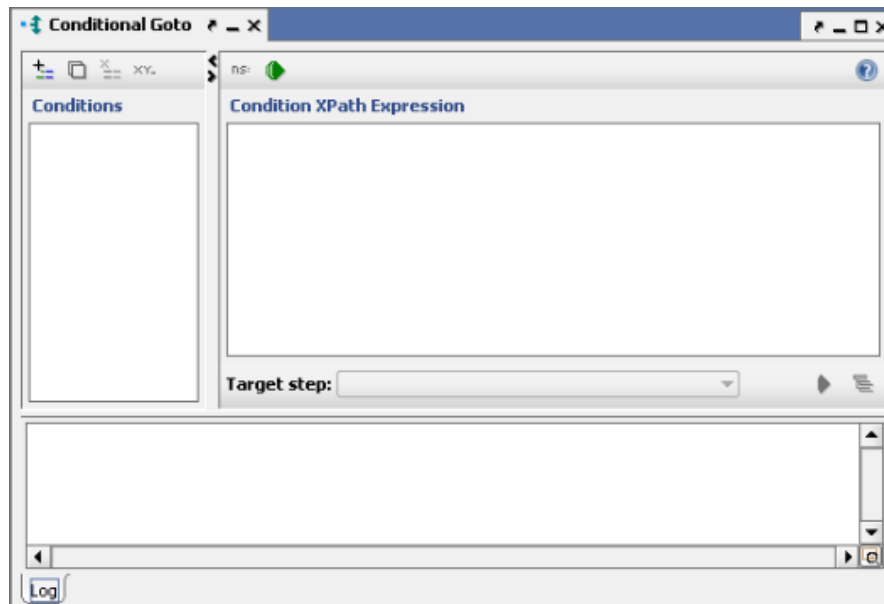



6.10. 流程控制

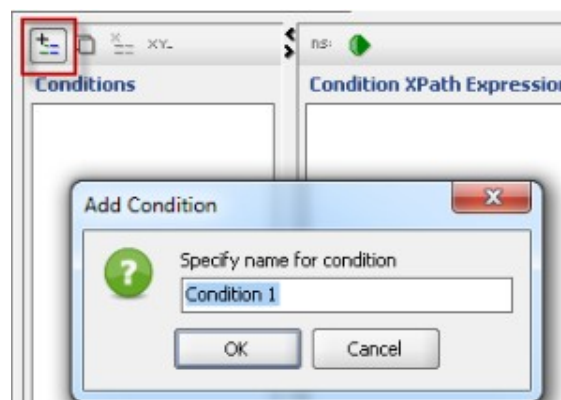
测试用例在执行时是按顺序执行测试步骤的，但有很多情况下，我们是需要通过返回的结果进行判断后，使用循环或分支执行某些测试步骤的，SoapUI 提供了三种方式让我们可以完成这样的需求：

测试步骤 — “Conditional Goto”

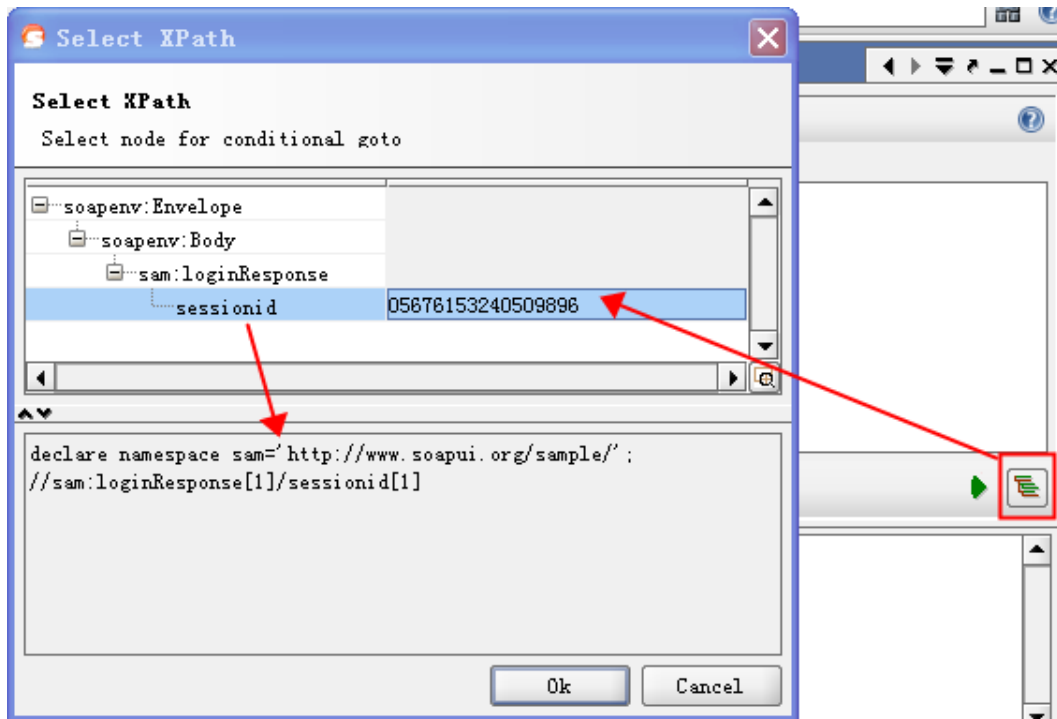
这个步骤让我们可以在设计测试用例时，使用分支的形式，它可以在“Condition XPath Expression”区域中使用一些 XPath 表达式来判断最近一次响应的报文中的某些字段值，如果 XPath 表达式返回“TRUE”，那么“Condition Goto”测试步骤会转去执行指定的目标测试步骤。



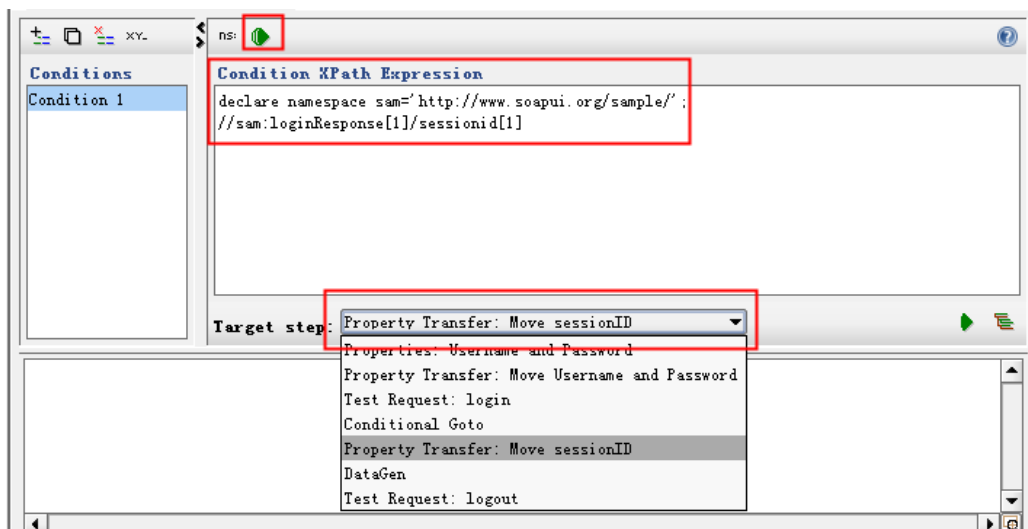
左上部分包含了定义条件变量的按钮：，在创建好条件变量后，“Condition XPath Expression”区域即可编辑，底部 log 区域会打印执行日志。首先，我们需要添加一个条件变量：



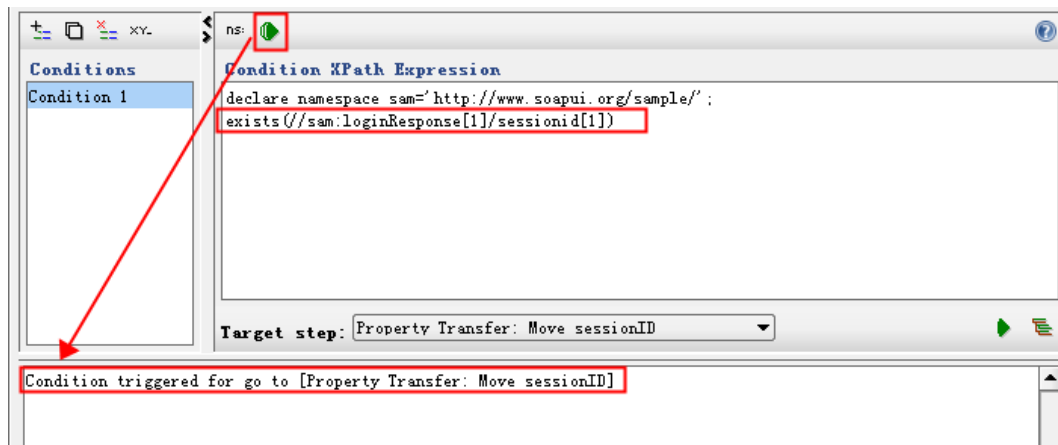
点击“OK”，然后通过 XPath 向导按钮，选择我们所要检查的节点：



弹出的结果报文是最后一次响应所接收到的报文，上图是一个 SOAP 请求的响应，可以很方便地直接选择 sessionid 节点进行校验，选择完毕后，点击 OK 回到主编辑窗口：



如上图所显示，我们还需要对“Condition XPath Expression”区域的内容进行改造，使其变成一个 XPath 表达式判断语句，添加 exists 表达式，然后点击 执行按钮。



“Log”区域显示的日志，表示条件已经被成功触发。

测试步骤—“Script”

“Script”步骤也提供了控制测试用例流程的功能，允许你可以跳转到测试用例中的任意步骤，或直接执行测试步骤，它主要使用了两个方法：

1、testRunner.gotoTestStepByName("nameofteststep")这个方法将流程转到指定的测试步骤中，但必须等当前的脚本执行完毕，不会立即转去执行脚本中指定测试步骤：

```
if( Math.random() > 0.5 )
    testRunner.gotoTestStepByName( "Request 1" )
else
    testRunner.gotoTestStepByName( "Request 2" )
//do something else
.....
```

2、testRunner.runTestStepByName("nameofteststep")会执行指定的测试步骤，见例子：

```
if( Math.random() > 0.5 )
    testRunner.runTestStepByName( "Request 1" )
else
    testRunner.runTestStepByName( "Request 2" )
//do something else
.....
```

这个脚本会先去执行指定的测试用例，在执行完毕后，会继续执行脚本中的内容。下面是一个循环的脚本，它会执行十次随机的请求后，才执行之后的脚本，这里使用“gotoTestStepByName”也会有同样的结果：

```
// run ten random requests
for( i in 1..10 )
{
    if( Math.random() > 0.5 )
        testRunner.runTestStepByName( "Request 1" )
    else
        testRunner.runTestStepByName( "Request 2" )
}
//do something else
.....
```

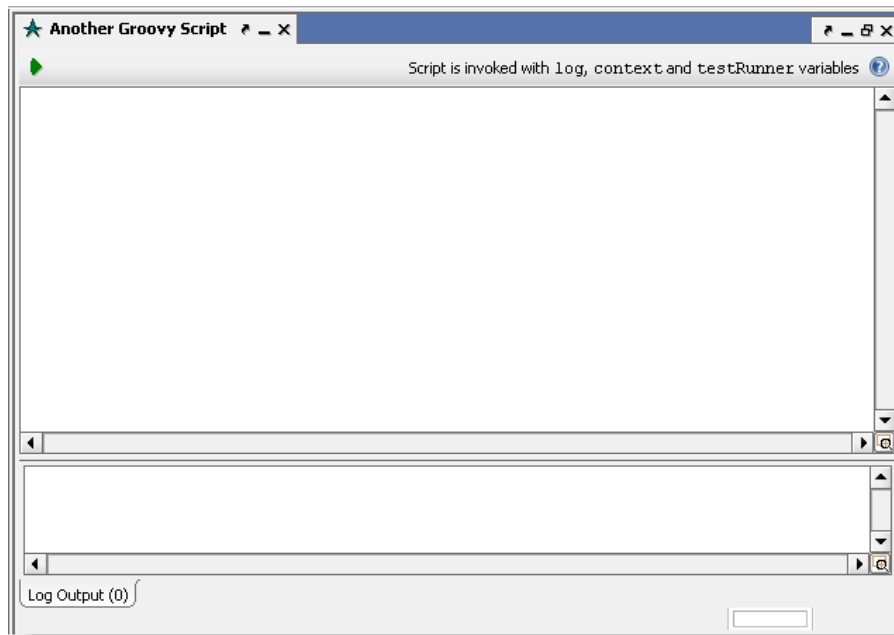
测试步骤—“DataSource Loop”


这个步骤与“DataSource”测试步骤绑定，主要用在数据驱动测试的场景中，此处不展开，可参考上一节内容。

6.11. 脚本处理

编写脚本是 SoapUI 很重要的一个功能，它允许你可以根据你的特定需求去定制你的测试执行，这些脚本都能访问“context”相关的变量和一个日志对象。

在 SoapUI 中“Script”步骤是非常有用的，JRE, soapUI 等相关的 API 都可以被使用，创建一个“Script”步骤：



顶部的  按钮用来执行当前的脚本，底部的 Log 显示脚本执行时打印的日志，正如最上面所列的提示，我们可以在脚本中直接使用三个变量：

testRunner

testCaseRunner 对象，是访问项目中变量、结果等的入口，是用来执行测试用例的对象，在测试用例中循环地执行各个测试步骤。它提供了相关的测试执行方法和对象（测试用例的属性），最普通的使用场景如下：

1. 使用 testRunner.testCase 可以访问和操作项目中所有的对象。
2. 使用 testRunner.fail(...) (or testRunner.cancel)中止正在运行时，发生错误的测试用例。
3. 使用 testRunner.gotoStepByName(...)或 testRunning.runTestStepByName(...)来跳转到测试用例中的其它测试步骤。

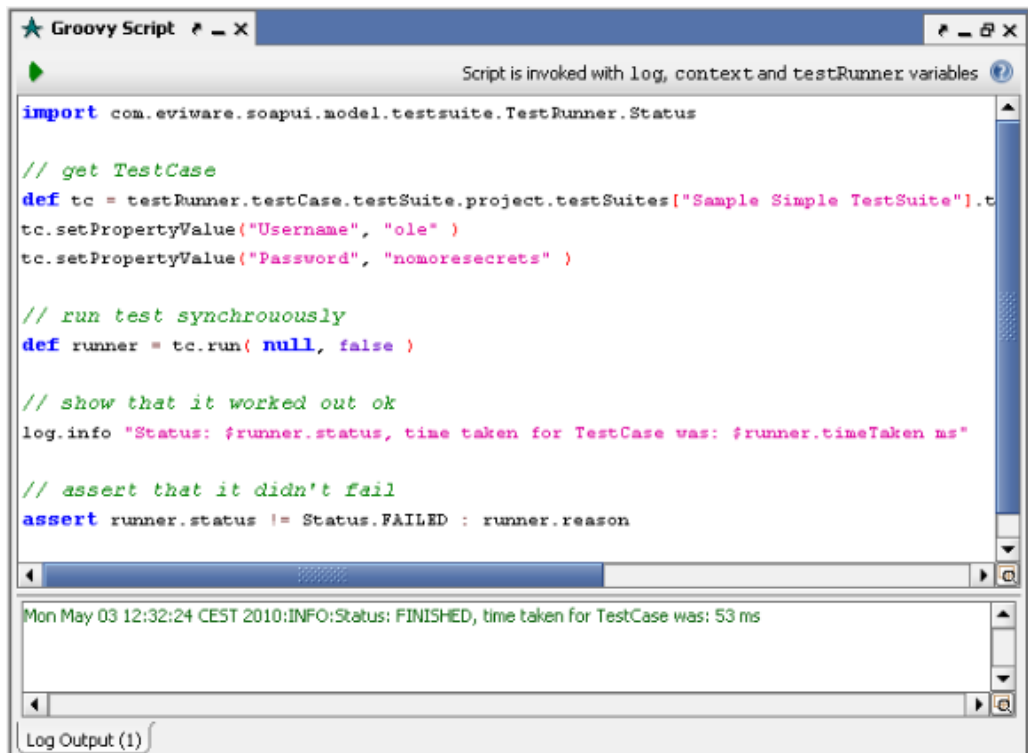
context

TestcaseRunContext 对象，具有上下文相关的属性，主要使用在存储值，以用在后续或相关的脚本中，如：context.myProperty = "Hello"，将会在上下文中创建一个名为“myProperty”且被赋值为“Hello”的属性，在随后的测试脚本中，你可以通过 log.info(context.myProperty)进行访问。经常使用在循环上，主要是通过保存相应的计数器等来控制测试步骤的执行。

log

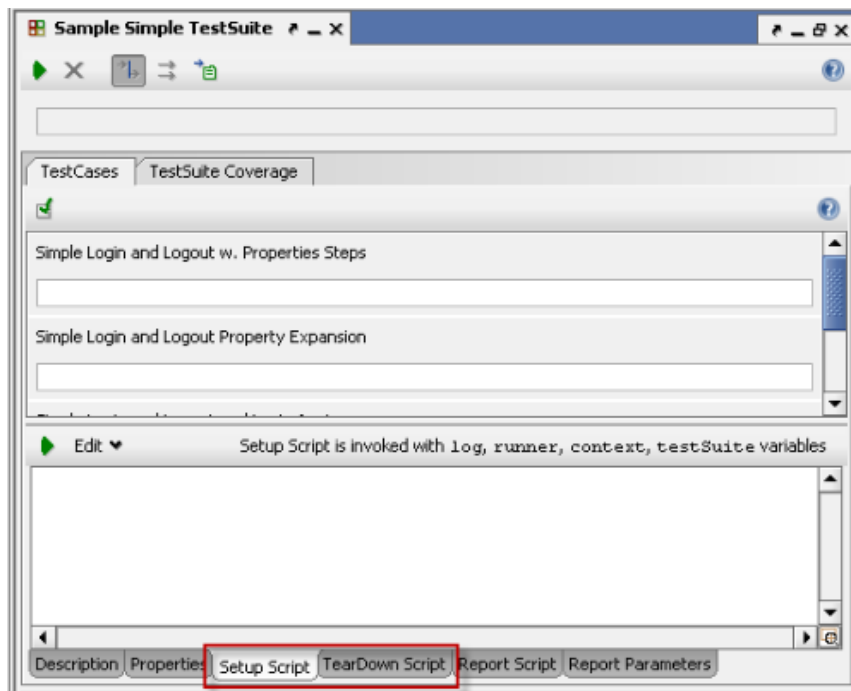
一个标准的 log4j 日志对象。

下图是一个 groovy 脚本编写的测试步骤：




6.12. 数据初始化-清理

在项目、测试集、测试用例中都可以使用初始化“Setup”和清理“TearDown”脚本，在打开项目、测试集、测试用例时，都可以看到“Setup”和“TearDown”脚本的编辑器，可以直接在编辑器里编辑脚本：



通过脚本名称，我们不难看出，这两个脚本会在用例、测试集或项目运行前（Setup）和运行后（TearDown）执行，它们主要使用在测试的准备和测试后数据的清理（如关闭资源、创建报告等），接下来用一个简单的例子进行说明：在 Setup 脚本中创建一个 JDBC 连接，并保存到上下文中，测试脚本中使用了 connection 去执行一些与数据库访问的操作，最后由 TearDown 脚本来关闭数据库连接，这对我们是非常方便的。

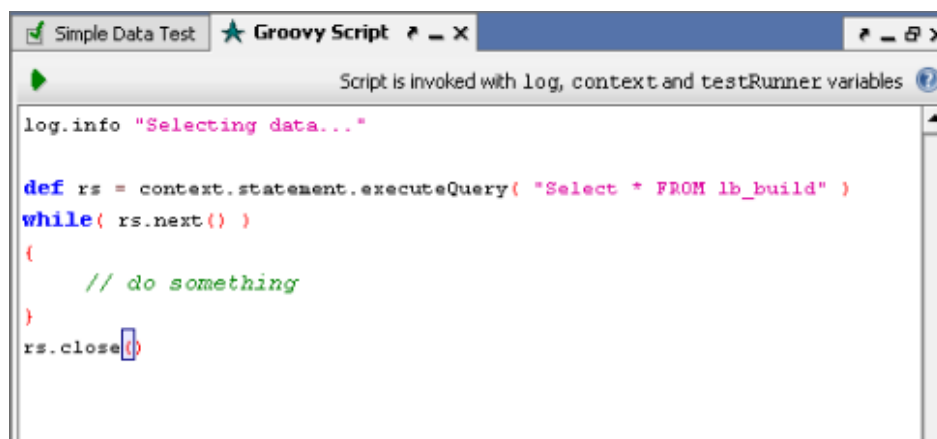
Setup 脚本如下（使用 GroovyUtilsPro 类来创建一个项目级别的 JDBC 连接）：



```
log.info "Preparing connection"

def utils = new com.eviware.soapui.support.GroovyUtilsPro( context )
context.dbConnection = utils.getJdbcConnection( "StorageDB" )
context.statement = context.dbConnection.createStatement()
```

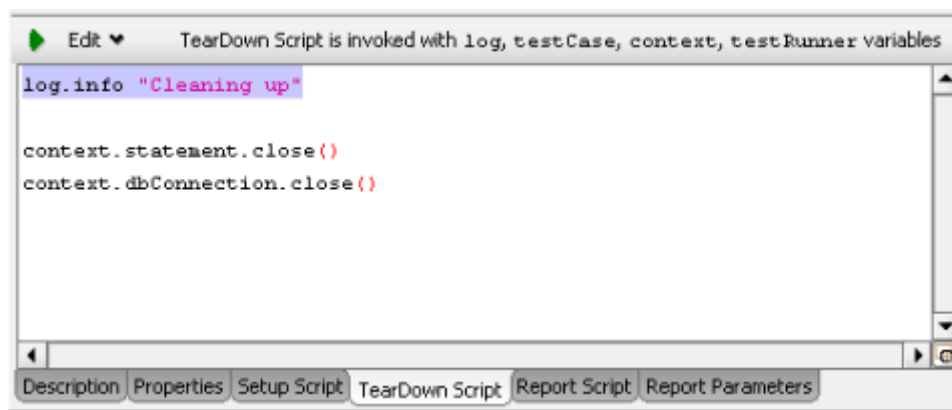
测试步骤中使用 script 类型来创建，使用上面创建的连接执行数据库操作：



```
log.info "Selecting data..."

def rs = context.statement.executeQuery( "Select * FROM lb_build" )
while( rs.next() )
{
    // do something
}
rs.close()
```

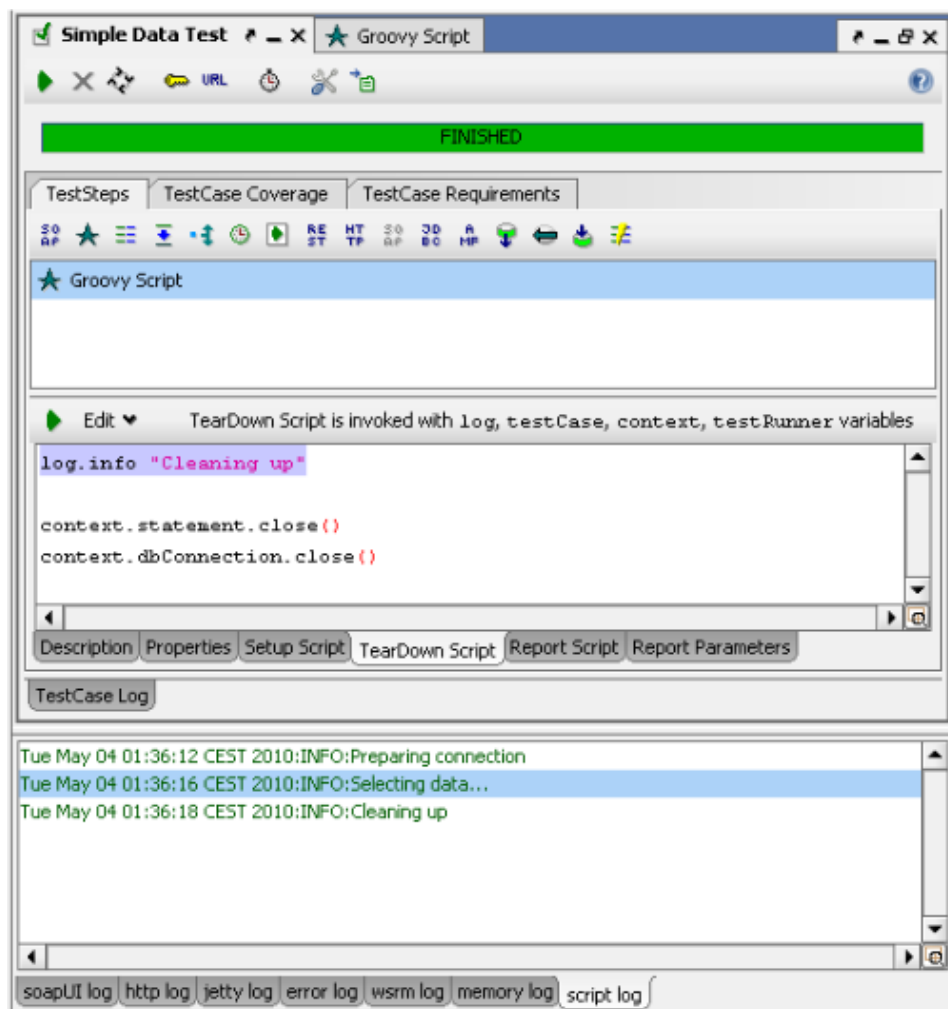
最后通过 TearDown 脚本来关闭数据库连接：



```
log.info "Cleaning up"


context.statement.close()
context.dbConnection.close()
```

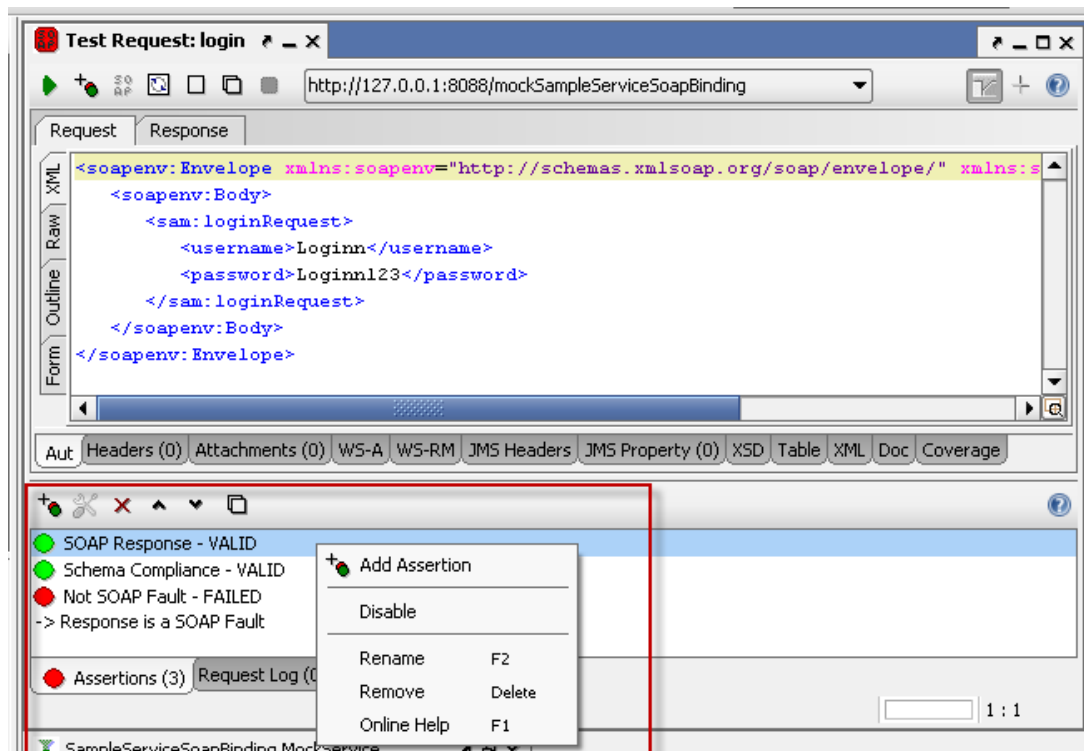
完成之后，我们执行整个测试用例：



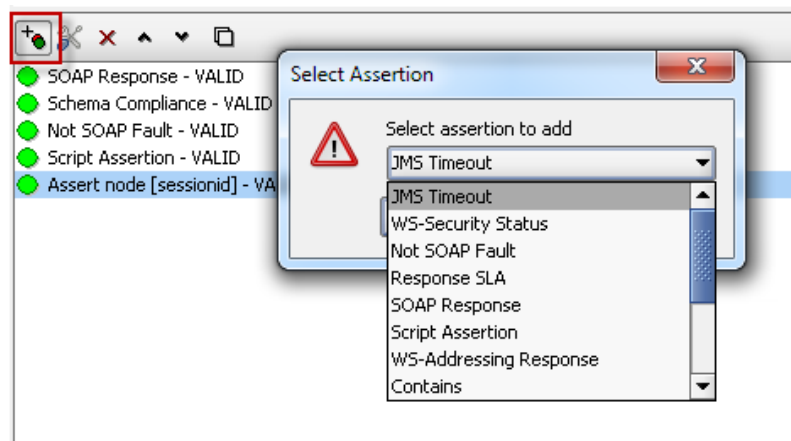
6.13. 断言操作

在 SoapUI 的测试中我们会发现基本上每个步骤都会包含断言。没有断言的话你如何能确认你的测试执行是正确的？断言是为了检查你的接收是否如你所愿，所以简单地讲，当你做一个手工的测试并查看响应，如要确认返回一个名字，那么你就是在“断言”那个名字。换句话说，手工的测试包含手工的断言。

创建一个断言，可以先进入请求/响应编辑器中（只有测试用例中的请求才能插入断言，一般的接口请求是不能创建断言的），通过点击断言标签  **Assertions (3)**，进入断言信息框中，之后可以通过点击添加按钮创建一个新的断言。



在上图所示的快照中，你可以看到 3 个断言，其中有 2 个断言通过，而最后一个失败。在断言窗口上部的工具栏中，提供了添加、配置、删除、移动和复制断言的功能，右键单击断言窗口时，弹出的右键菜单也同样提供了这些功能。当双击创建好的断言时，会弹出该断言的配置窗口。创建断言时，页面会弹出断言的类型以供选择：



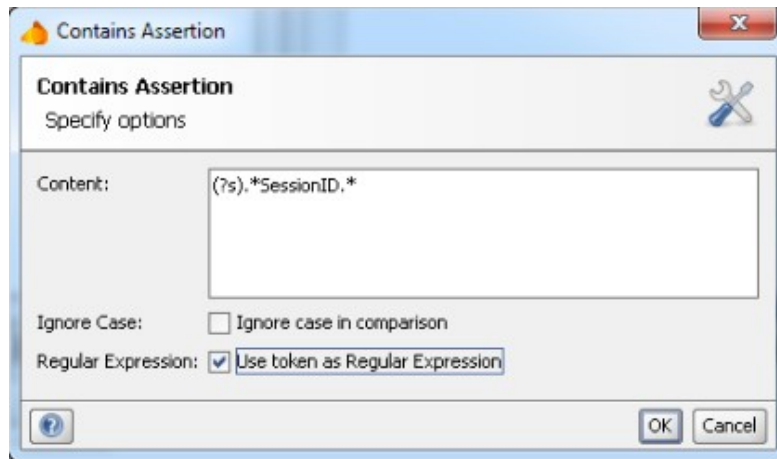
SoapUI 提供了大量的验证接收报文的断言，一些断言是专门为某些指定的测试步骤所使用，也有一些是为所有测试步骤所通用，这些通用的断言主要有：

Contains

检查指定的字符串是否存在。这个断言检查一些文本内容是否包含在接收到的响应报文中，它的配置对话框如下，下图中我们看到，检查的文本中是可以包含正则表达式的，在这里还可以支持“Property-Expansion”的验证。

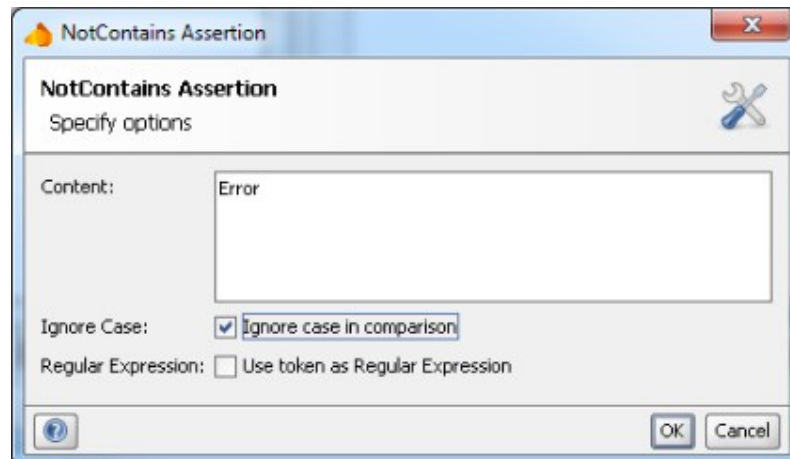
关于能够支持的正则表达式的结构可详见：

<http://www.jdocs.com/javase/7.b12/java/util/regex/Pattern.html>



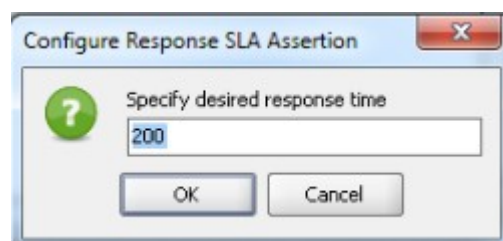
Not Contains

检查指定的字符串是否存在。与 Contains 断言相反，这是一个检查不存在某文本串的断言，配置窗口如下，检查响应报文中不能存在“Error”，忽略大小写匹配：



Reponse SLA

检查响应时间是否在指定的时间内，如果没有在指定的时间内，那么断言会失败，配置页面非常简单，只需要填入数字即可，单位是毫秒：



填入的值可支持“Property-Expansion”的形式，允许我们可以通过一个变量来控制该值。

SoapUI 将接收到的响应报文在内部直接转换为 XML，因为能够引入 XQuery 和 XPath 这两种非常方便的断言方式，接下来主要是介绍这两种断言方式。

XPath Match

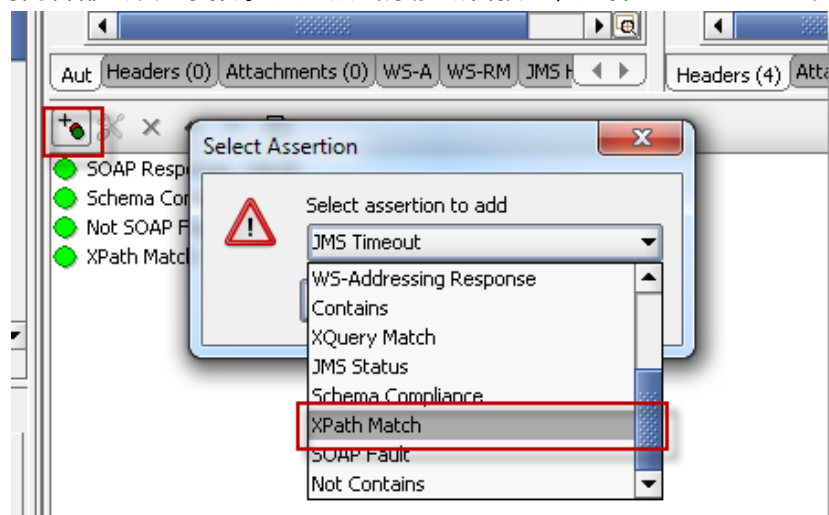
比较 XPath 表达式所指定位置上的元素是否与预期值符合。如果值匹配则断言通过，否则失败，让我们看前面的例子中，所要验证的 login 接口返回的响应报文：

```

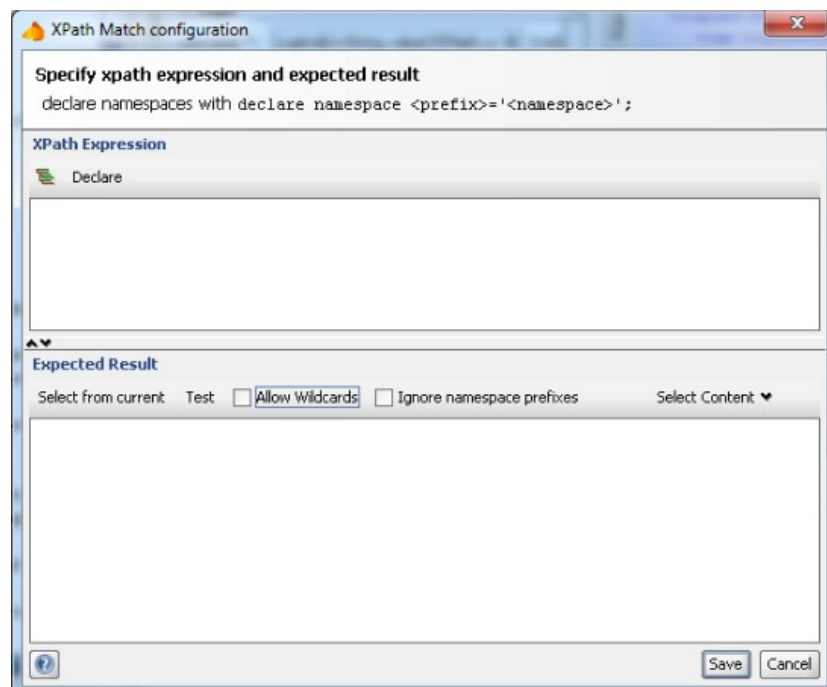
1. <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2.   xmlns:sam="http://www.example.org/sample/">
3.   <soapenv:Header/>
4.   <soapenv:Body>
5.     <sam:loginResponse>
6.       <sessionid>10873286937963711</sessionid>
7.     </sam:loginResponse>
8.   </soapenv:Body>
9. </soapenv:Envelope>


```

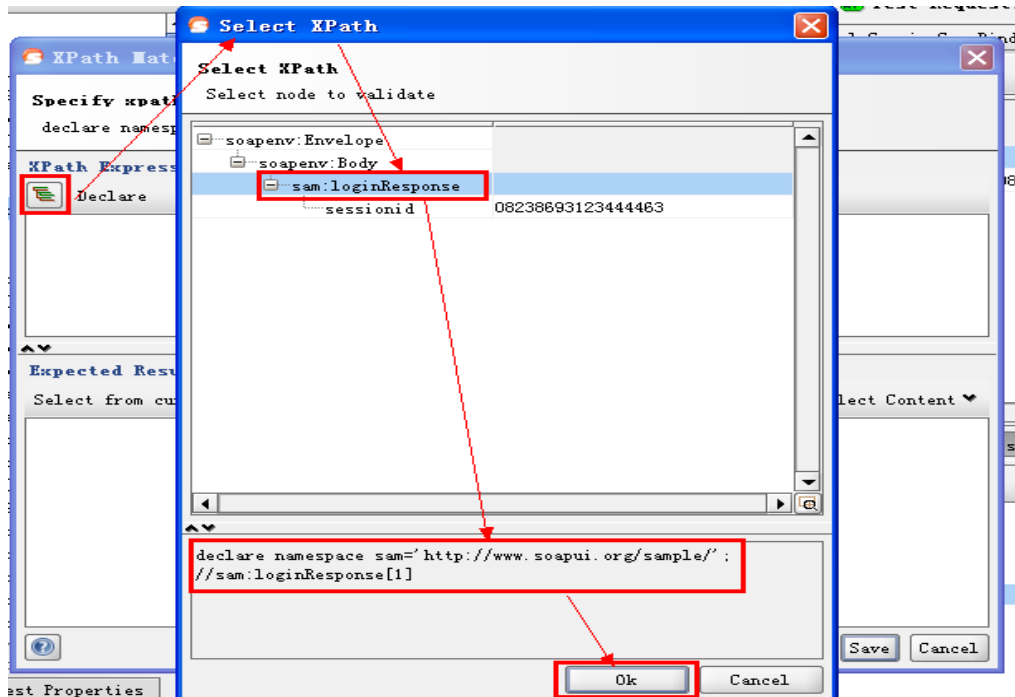
实际上，我们只需要验证 loginResponse 中的结果 sessionid 这个元素的值，且它在每次请求时值的内容都会发生变化。通过点击添加断言按钮，选择“XPath Match”类型的断言：



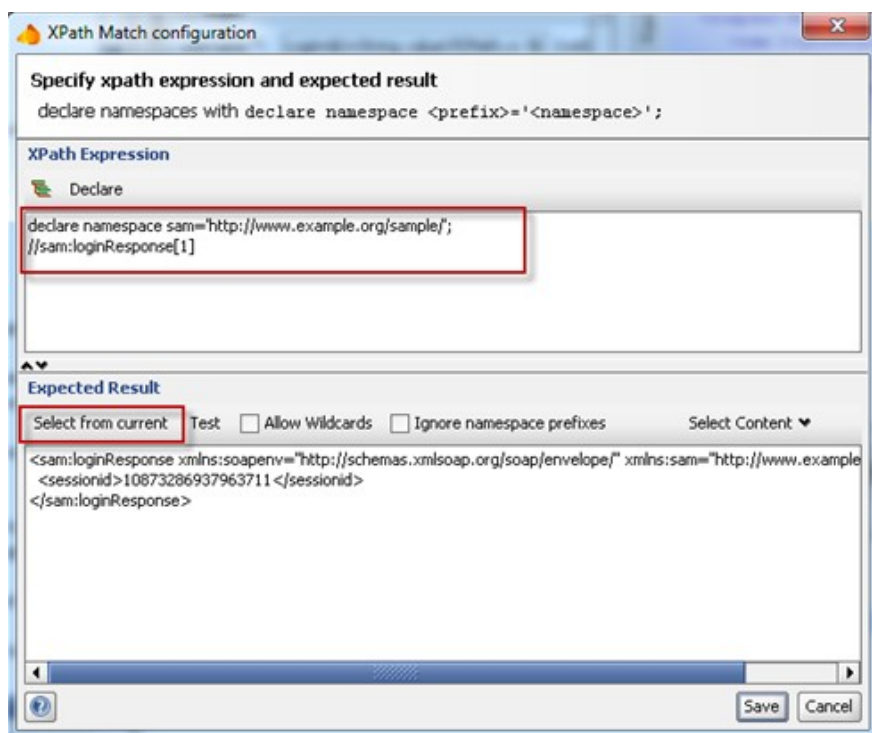
点击“OK”后，一个空白的尚未配置的对话框将被打开：



对话框被分为两个区域，上面的部分填写 XPath 表达式，下面的部分填写预期的结果值。下一步从 loginResponse 中选择所要验证的元素，通过  按钮可以直接查看结果报文的结构，直接选择所要关注的元素，SoapUI 会自动帮你填充 XPath 表达式到上面的区域中：



通过上面填充完上面区域的内容后，再点击下面区域“Select from current”可以直接帮我们最后一次的响应报文中对应的上面区域的 XPath 位置上的值直接选择出来：



正如你所看到的，这样的断言，在后续的新请求中，都会将 XPath 表达式所指定位置的值与预期的结果值进行匹配，如果值不匹配，那么断言就会失败。

很显然，上面有个问题，由于每次请求 sessionid 的值都会变化，因此按照上面设置的预期结果，断言将会失败，因此我们选择了“Allow Wildcard”允许通配符的模式进行设置：



要匹配任何元素都可以，只要设置的 XPath 路径正确即可。

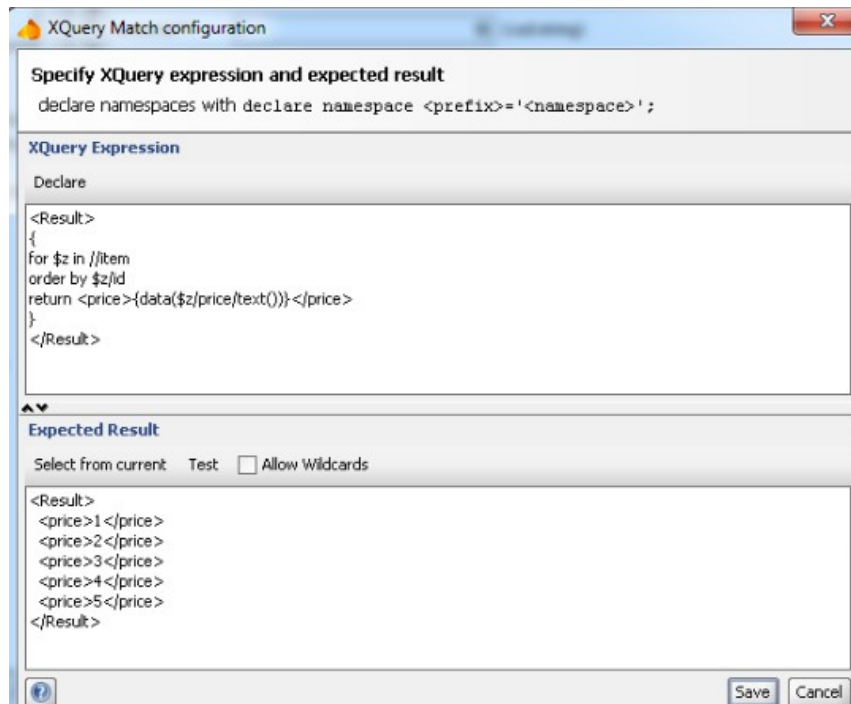
XQuery match

比较通过 XQuery 表达式提取的结果是否与预期值符合。XQuery 断言的工作机制与 XPath 相似，惟一与 XPath 不同的是 XQuery 是使用 XQuery 的表达式来选择 XML 进行验证，对于复杂的验证有一定的明显优势：1、可以选择所需要的节点并将其合并到同一个 XML 结果中，使得断言更易于管理，2、结果中的元素可以排序这一特性允许你创建一个不依赖于元素顺序的断言等。

我们需要对下图所示的结果报文进行校验：



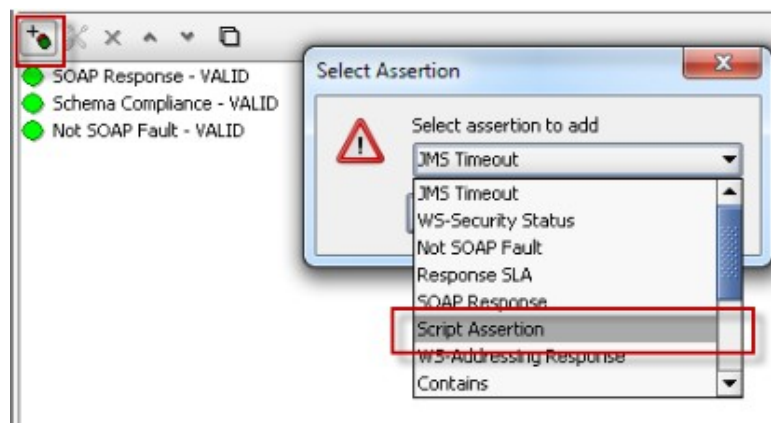
我们一般只能验证里面的元素值，如想验证所有的 price，按照之前所了解的内容，只能关注于验证每个 price 是否存在，但是无法按顺序得到值，而 XQuery 可以解决这个问题：



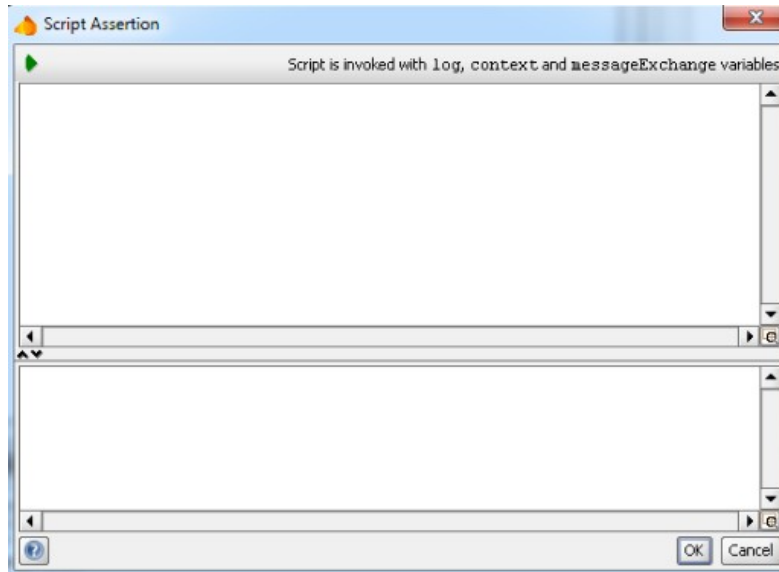
上面区域中的 XQuery 表达式选择了所有的值，按照 id 排序，并把它们抽取到一个临时的 XML 结果中，通过“Select from current”得到表达式处理后的响应。


Script

执行特定的脚本来验证接收的报文，包含接收报文的内容，HTTP 报文头等。在项目中的验证脚本是使用 groovy 或 javascript 脚本语言进行编写的，添加 script 脚本断言到你的测试步骤中：



点击“OK”后会打开一个配置对话框：



上面区域是一个标准的 SoapUI 脚本编辑器，按钮  是用来验证最后一次接收到的响应报文，并通过一个弹出窗口来显示结果，底部区域显示了脚本对应的日志输出信息。messageExchange 这个对象提供了最后一次请求/结果的很多属性，供 script 编写脚本时使用。下面通过一个简单的例子说明：

1、验证响应报文中 HTTP 报文头的内容：

```
// check for the amazon id header
assert messageExchange.responseHeaders["x-amz-id-1"] != null
```

2、验证响应时间：

```
// check that response time is less than 400 ms
assert messageExchange.timeTaken < 400
```

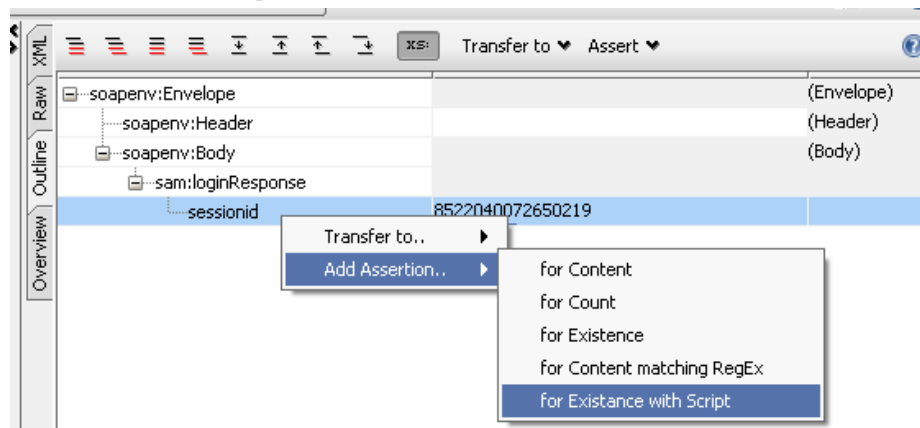
3、验证响应报文中的节点：

```
// check that we received 2 attachments
assert messageExchange.responseAttachments.length == 2
```

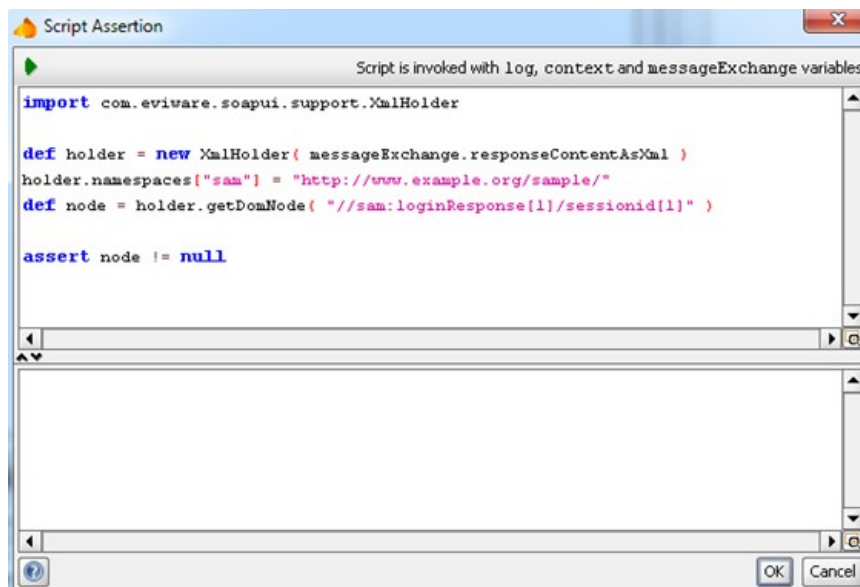
4、验证指定的 XML 元素在响应报文中：


```
// check for RequestId element in response
def holder = new XmlHolder( messageExchange.responseContentAsXml )
assert holder["//ns1:RequestId"] != null
```

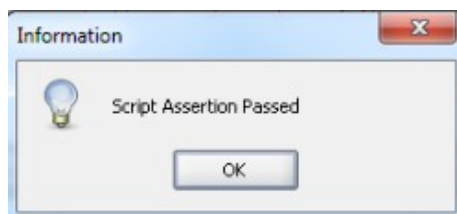
另外还有一种添加 script 脚本验证响应报文的方式：



选择“for Existence with Script”，SoapUI 将直接帮我们创建好下面的验证脚本：



点击  按钮，页面返回验证结果信息：



为方便某些测试步骤断言的使用，SoapUI 也提供了专门用于这些类型上的断言，这里主要介绍常用的类型：

SOAP 形式的测试步骤

Schema Compliance：根据 WSDL 所定义的内容验证响应报文和包含的 XML 模式。

SOAP Response：验证响应报文是一个有效的 SOAP 响应。

SOAP Fault：检查响应报文包含 SOAP 异常。

Not SOAP Fault：检查响应报文没有包含 SOAP 异常。

JDBC 形式的测试步骤

JDBC Timeout：验证从目标数据库接收到结果数据的时间是否在指定值内。

JDBC Status：验证没有 JDBC 相关的错误发生。

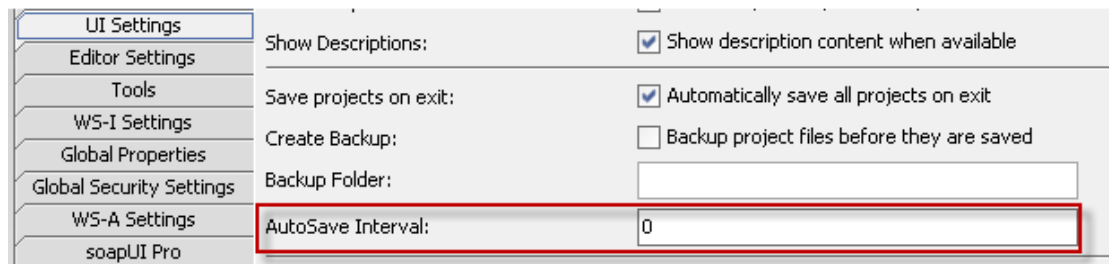
MockResponse 形式的测试步骤

Schema Compliance：根据 WSDL 所定义的内容验证响应报文和包含的 XML 模式。

SOAP Request：验证请求报文是一个有效的 SOAP 请求。

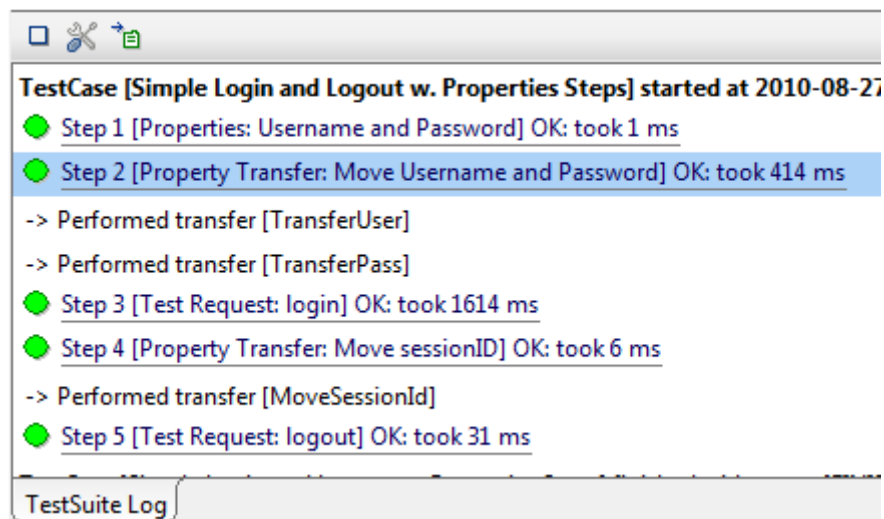
6.14. 定时保存

SoapUI 中提供了一个可以自动保存项目变更的功能，它是通过 File-Preferences-UI Settings 进行设置的。自动保存时间间隔的单位是分，当配置此参数时，他将会自动地保存所有开起来的项目。

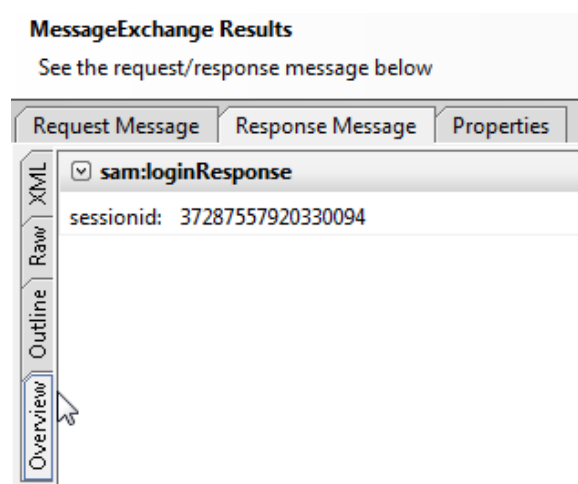


6.15. 响应报文

当你执行完一个测试用例时，你可以一步一步地观察测试结果日志：TestCase Log，如果是在测试集中执行测试用例，则可通过：TestSuite Log 查看。



直接点击上图的每个测试步骤，可直接弹出实际的请求报文、响应结果、属性等信息，当在执行测试失败时，这对于排查问题是很方便的。



6.16. 日志查询

SoapUI 的日志通常能告诉你发生了什么事，使用日志去查看发生了什么问题是一个好了解方式。让我们来看一个很普遍的例子，你发送了一个请求，但没有从 SoapUI 的响应窗口中得到任何响应信息。SoapUI 中的大量日志能够告诉你错误的原因，你可以创建一个 HTTP 测试，URL 使用：<http://www.ghiklj.com>，然后发送一个请求，你将不会收到任何响应，但你如果看日志，你将看到如下报错：

```
Fri Jul 30 15:57:08 CEST 2010:ERROR:Exception in request: java.net.UnknownHostException:
www.ghiklj.comFri Jul 30 15:57:08 CEST 2010:ERROR:An error occurred [www.ghiklj.com], see error
log for details
Fri Jul 30 15:57:08 CEST 2010:INFO:Error getting response for [HTTP Test Request];
java.net.UnknownHostException: www.ghiklj.com
```

Looking in the Error Log you see this:

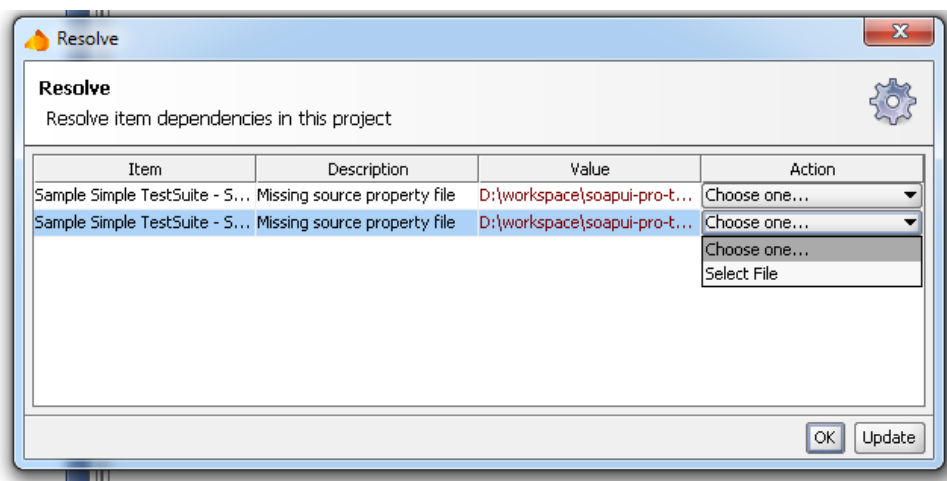
```
Fri Jul 30 15:57:08 CEST 2010:ERROR:java.net.UnknownHostException: www.ghiklj.com

java.net.UnknownHostException: www.ghiklj.comat java.net.PlainSocketImpl.connect(Unknown
Source)
at java.net.SocksSocketImpl.connect(Unknown Source)
at java.net.Socket.connect(Unknown Source)at java.net.Socket.connect(Unknown Source)
at java.net.Socket.<init>(Unknown Source)at java.net.Socket.<init>(Unknown Source)at
org.apache.commons.httpclient.protocol.DefaultProtocolSocketFactory.createSocket(DefaultProtoco
at
org.apache.commons.httpclient.protocol.DefaultProtocolSocketFactory.createSocket(DefaultProtoco
org.apache.commons.httpclient.HttpConnection.open(HttpConnection.java:707)at
com.eviware.soapui.impl.wsdl.support.http.SoapUIMultiThreadedHttpConnectionManager$HttpConnecti
org.apache.commons.httpclient.HttpMethodDirector.executeWithRetry(HttpMethodDirector.java:387)
at org.apache.commons.httpclient.HttpMethodDirector.executeMethod(HttpMethodDirector.java:171)
at org.apache.commons.httpclient.HttpClient.executeMethod(HttpClient.java:397)
at
com.eviware.soapui.impl.wsdl.submit.transports.http.HttpClientRequestTransport.sendRequest(Http
at com.eviware.soapui.impl.wsdl.WsdlSubmit.run(WsdlSubmit.java:122)
at java.util.concurrent.Executors$RunnableAdapter.call(Unknown Source)
at java.util.concurrent.FutureTask$Sync.innerRun(Unknown Source)
at java.util.concurrent.FutureTask.run(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
at java.lang.Thread.run(Unknown Source)
```

上图是在错误日志的一个异常信息，这并不代表 SoapUI 发生错误，通过仔细观察上面的日志，你将会很快意识到是因为你所输入的 URL 地址不存在所导致，所以通过读日志能够帮助我们快速地发现错误。

6.17. 导入和检查项目

如果你已经有一个项目文件，你可以通过“Import Project”选项轻松地导入到你的工作空间中。在导入后，SoapUI 会做一个检查，来确认所有必须的外部依赖文件是否可用（这个过程称为“resolving”），如果在检查过程中发现错误，将会弹出一个“Resolve”对话框显示错误的内容，并提供修改的选项。



在上图例子中，两个属性文件丢失了，你可以通过下拉框来选择正确的文件，当你设置好外部文件后，则可以通过点击“OK”进行保存即可。


如果需要手动地进行检查，则可以使用右键单点击项目名，在弹出的右键菜单中，选择“Resolve”选项，进行人工促发检查。

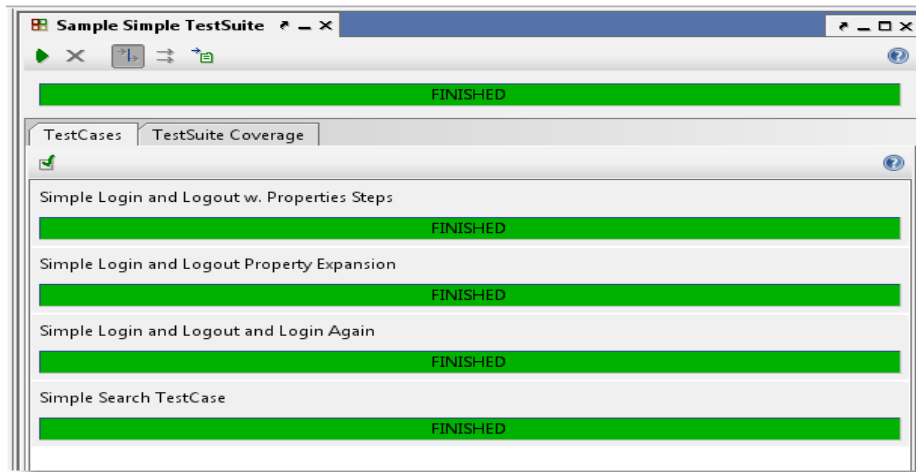
SoapUI 允许你通过 HTTP 协议导入项目文件，发布项目是很简单的，文章中下一节有提到可参考。使用这种方式，其它人可以很随意地导入项目文件。选择“File”菜单中的“Import Remote Project”选项可以通过指定的 URL 导入项目文件，导入的方便程度就像在直接使用本地文件导入一样。因为无法重新将更新保存到 URL 网页上，所以任何改变均只被保存在本地。

6.18. 发布测试报告

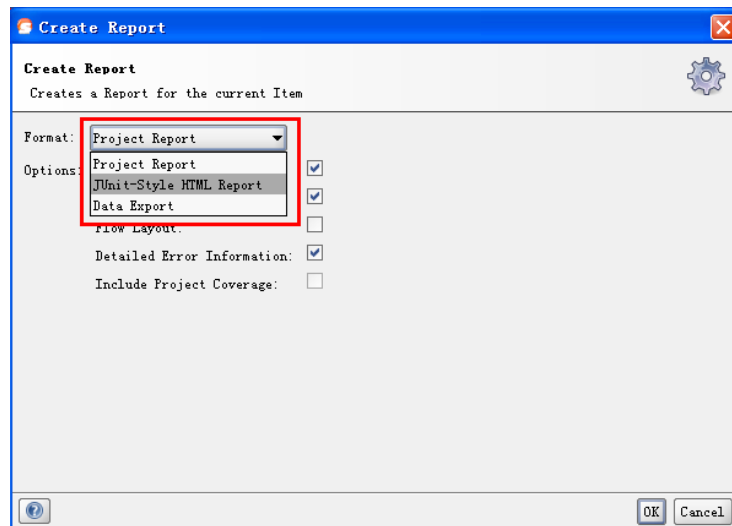
SoapUI 提供了三个类型的报告，分别可以在项目、测试集、测试用例和负载测试中产生

- 1、可打印的报告：可以被打印或保存成 PDF，HTML，Word，RTF，Excel 等，允许你创建和定义任意形式的报告。
- 2、数据导出：可以导出报表数据到 XML 或 csv 格式的文件，如果你需要将测试数据导入到其它工具中，形成自定义的报告，那么这将是一个很有用的功能。
- 3、HTML 报告：将简单的功能测试结果概要信息输出到 HTML 形式上（在负载测试中不产生）

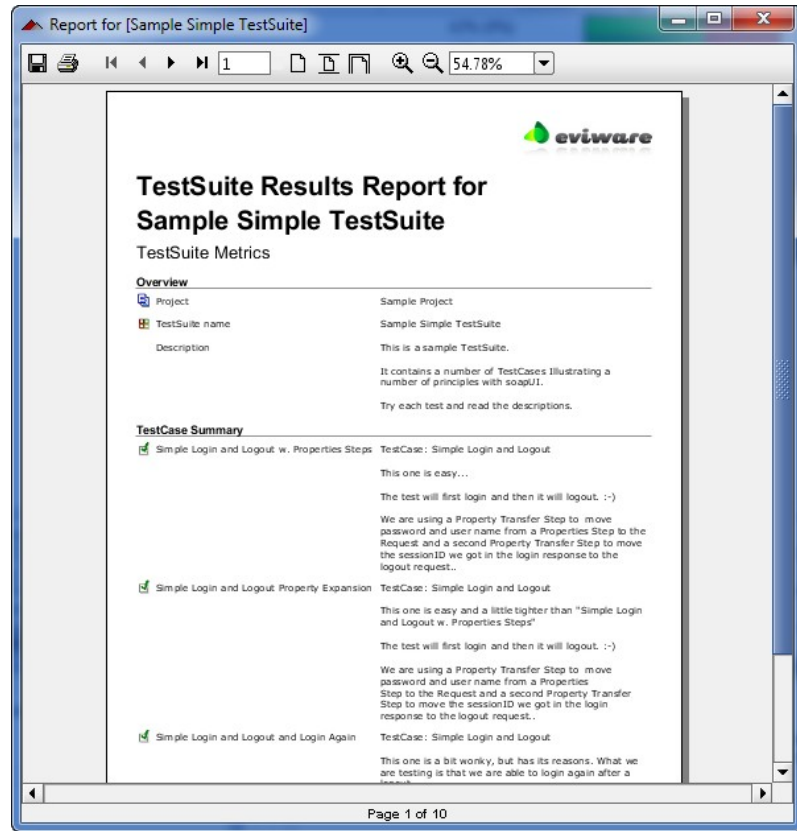
打开项目、测试集、测试用例和负载测试窗口，一般可以看到生成报告的按钮 ：




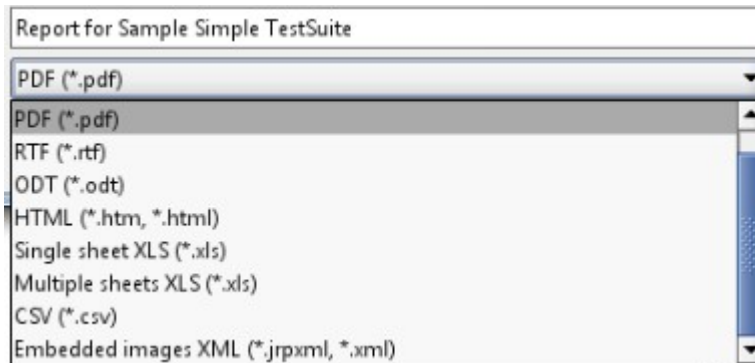
在测试运行完后，点击生成报告按钮，会弹出“Create Report”对话框，在 Format 中可以选择生成哪种格式的报告：



点击 OK 后，生成报告，根据 Format 设置格式的不同，生成的报告内容显示的样式有所差别，下图是使用 Project Report 的格式生成的报告：



点击  图标，会弹出打印的格式，以供选择：



如果使用第二种格式 Junit-Style HTML report 格式生成，可直接由浏览器打开，查看结果：

soapUI Test Results

Summary

TestCases	Failures	Errors	Success rate	Time
11	6	0	45.45%	6.113

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Projects

Note: Project statistics are not computed recursively, they only sum up all of its testsuites numbers.

Name	TestCases	Errors	Failures	Time (s)	Time Stamp	Host
Sample Project	11	0	6	6.113		

Project Sample Project

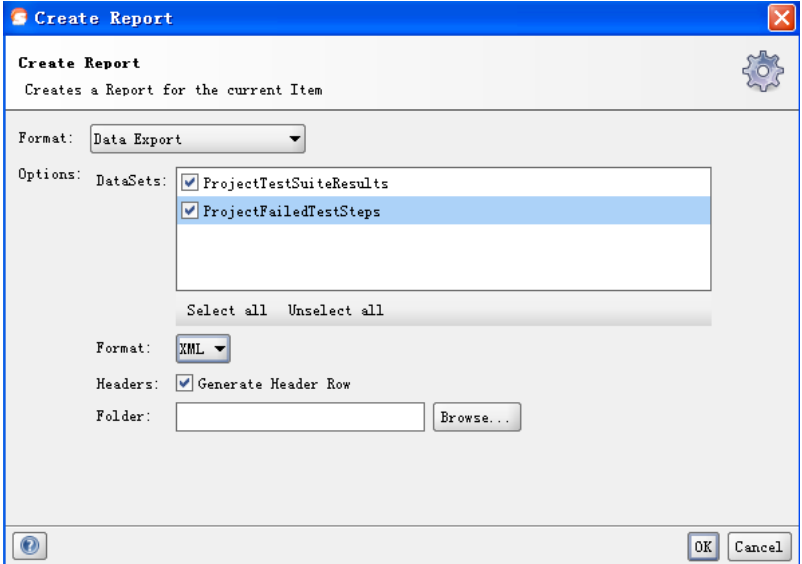
Name	TestCases	Errors	Failures	Time (s)	Time Stamp	Host
Sample Project.Sample TestSuite fails if we don't get faults	3	0	1	0.407		
Sample Project.Sample Simple TestSuite	4	0	1	5.323		
Sample Project.Sample expanded TestSuite	4	0	4	0.383		

Back to top

TestSuite Sample Project.Sample expanded TestSuite

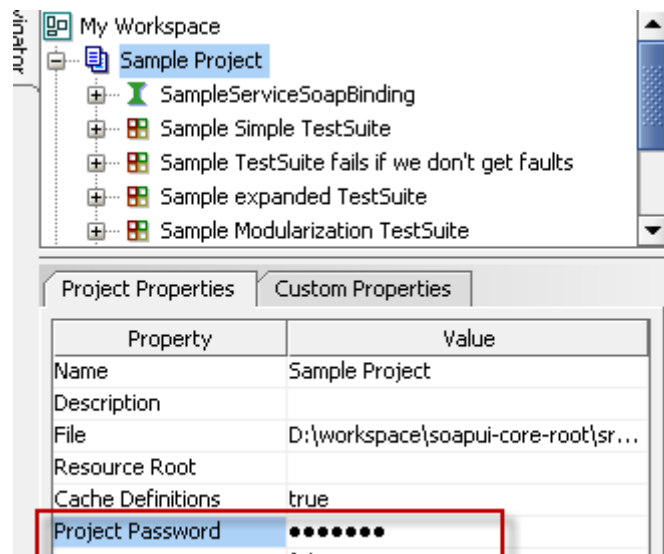
TestCase	Status	Type
Search and Buy TestCase	Failure	Failing due to failed test step Test Request: login Failed [Not SOAP Fault] Response is a SOAP Fault

最后一种则可直接导出成文件，供其它工具导入分析测试结果：

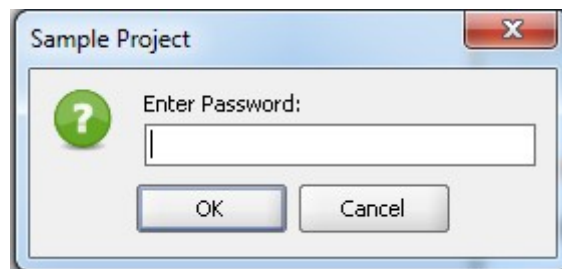


6.19. 加密项目

单击项目名称，在左边栏下半部分的项目属性标签页中，有一个名称为：Project Password 的属性，设置该数据的值，可以让你加密整个项目，不仅加密了 XML 文件，而且在 SoapUI 中要访问时还需要输入密码：



当打开 SoapUI，要访问有设置密码的项目时，项目会要求你输入密码，输入正确的密码，即可打开项目进行编辑。



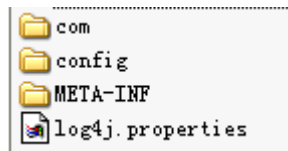
如果想移除密码只需要清空 Project Password 的值并保存项目即可。一定要记住密码，否则没有任何方式可以寻回被加密了的项目。

7. 测试场景的应用

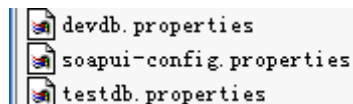
7.1. 引入 jar 包--读取数据源属性

在测试中，遇到的最大的问题就是开发库和数据库不是同一个库，因此，在项目或升级包合并，代码到测试环境后，之前在开发环境上写的测试代码就必须修改里面的数据库连接属性，以保证正确地读到相应的数据库。

为解决这个问题，引入自动化中对这个问题的处理方式，自动化中有一个全局的配置文件用来控制是读取开发库还是读取测试库配置，以表名作为入参，直接从数据库配置文件读取该表名所对应的数据库链接串、用户名、密码等创建数据库连接所必需的参数。此处采用 Jar 包：soapuidbutil.jar 的形式引入 SoapUI 中，将 Jar 包直接放到：D:\Program Files\evaware\soapUI-Pro-3.6\lib 文件夹下，重启即可，该 Jar 包的结构如下：

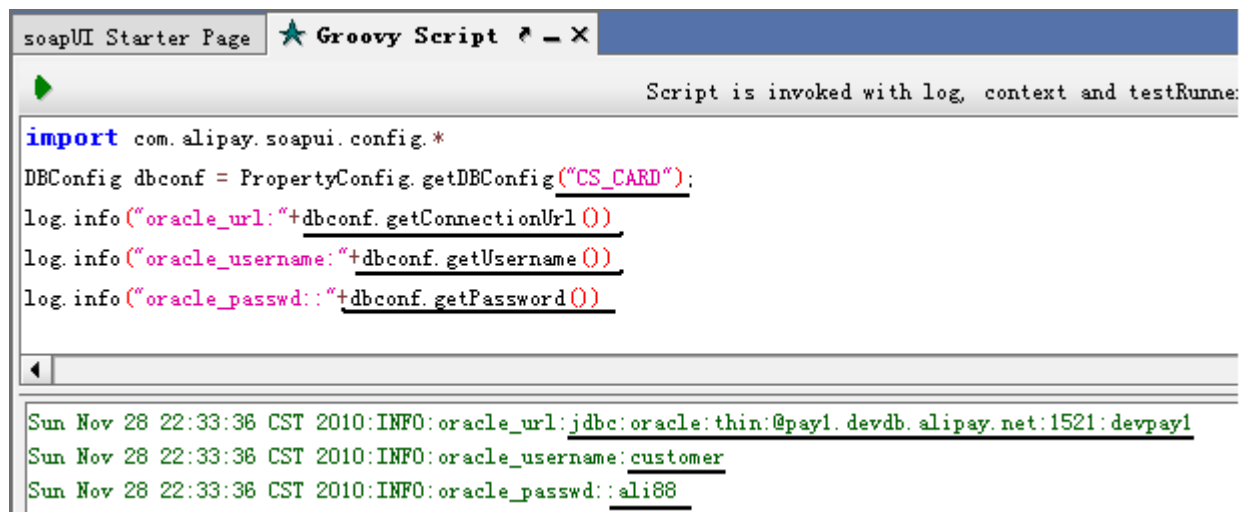


config 目录下包含三个配置文件：



soapui-config.properties 只提供了一个参数：check_testdb = false，当设置 false 时，将读取 devdb.properties 开发库配置文件，而设置成 true 时，则读取 testdb.properties 测试库配置文件。devdb.properties 和 testdb.properties 配置文件的内容与自动化测试中数据库配置文件的内容完全一致。

调用 Jar 包的过程如下：



需要通过 import 导入相应的 Jar 包，然后将表名填在上图的 CS_CARD 所在的位置，之后可通过 dbconf.getConnectionUrl() 等方法获得数据库的连接串等属性。

7.2. 调用 groovy 工具类

在测试中，最经常遇到的情况就是需要将预期结果与实际结果进行比较，且一般预期结果与实际结果都是 list 的形式，因此会有一些公共的方法需要提取出来，放在 D:\Program Files\eviware\soapUI-Pro-3.6\bin\scripts\soapui\utils 下以供程序直接调用，下面的代码是一个比较方法，方法提供 3 个入参：预期结果 List，实际结果 List，日志 log（传入该参数是为了在调用的方法中可将过程中的一些变量输出在 SoapUI 的控制台上，方便调试），比较方法代码如下：

```
package com.alipay.utils

import com.eviware.soapui.SoapUI
import org.apache.log4j.Logger

class Compare{
```

```

def static compare(List actualData,List expectedData,Logger log){
    def actualList = []
    def expectedList = []
    if(actualData != [] && expectedData != []){
        //对比数组长度是否一致
        if(actualData.size() == expectedData.size()){
            for(i in 0..actualData.size() - 1){
                if(actualData[i].toString() != expectedData[i].toString()){
                    log.info("-----")
                    log.info("实际值: " + actualData[i])
                    log.info("预期值: " + expectedData[i])
                    actualList << actualData[i]
                    expectedList << expectedData[i]
                }
            }
            if(actualList == []) {
                log.info("实际值与预期值一致")
                assert true
            }
            else{
                log.info("实际值与预期值不一致, 实际值: [${actualList}] 预期值: [${expectedList}]")
                assert false
            }
        }
        else{
            log.info("实际值与预期值元素个数不一致, 实际值: [${actualData}] 预期值: [${expectedData}]")
            assert false
        }
    }
    else{
        if (actualData == expectedData) {
            log.info("返回空结果")
            assert true
        }
        else {
            log.info("校验失败, 实际值与预期值不一致, 实际值: [${actualData}] 预期值: [${expectedData}]")
            assert false
        }
    }
}

```

```
}
```

SoapUI 中调用示例代码如下：

```
//调用兑入接口返回的流水与数据库存放的流水比较
```

```
Compare c = new Compare()
```

```
c.compare(actualData, expectedData, log)
```

这里还引入了自动化脚本中对 ORACLE 数据库的工具类，调用示例代码如下：

```
import com.alipay.utils.OracleOperatorUtil
```

```
//创建连接
```

```
OracleOperatorUtil deal = new OracleOperatorUtil()
```

```
def con = deal.oracle_connect("generalpoint.gpt_ex_inflow_instruction",log)
```

```
//执行 UPDATE 语句
```

```
def sql = """update generalpoint.gpt_ex_inflow_instruction t set  
t.partner_id='2088101010468271',
```

```
t.provider_code = 'ALIPAY1',t.sub_provider_code = 'ALIPAY-USER1'
```

```
where t.out_instruction_id='3129056994162903'"""
```

```
def result = deal.oracle_update(con, sql, log)
```

```
log.info("result:"+result)
```

```
//执行查询语句
```

```
sql="""select * from generalpoint.gpt_ex_inflow_instruction t where rownum<10"""
```

```
result = deal.oracle_query(con, sql, log)
```

```
log.info("result:"+result)
```

```
//执行删除语句
```

```
sql="""delete from generalpoint.gpt_ex_inflow_instruction t where  
t.instruction_id='201011240000131424'"""
```

```
result = deal.oracle_delete(con, sql, log)
```

```
log.info("result:"+result)
```

```
//关闭数据库连接
```

```
deal.oracle_disconnect(con, log)
```

7.3. 响应报文处理

对于通过调用 Webservice 接口所返回的 XML 报文，在提取里面的元素存放在 List 中时，都需要根据结果报文定义相应的 namespace 等，此处列出一个处理案例以供大家参考：

```
//调用兑入接口返回的结果
```

```
def response = context.expand( '${exchangeIn - Request#Response#declare namespace  
soap='http://schemas.xmlsoap.org/soap/envelope/'; //soap:Envelope[1]}') )
```

```
def xml = new XmlSlurper().parseText(response).declareNamespace(ns1:  
'http://api.exchange.service.common.pointprod.alipay.com',soap:  
'http://schemas.xmlsoap.org/soap/envelope/')
```

```
//定义 LIST
```

```
List actualData = []
```

```
List tempArray = []
```

```
//从返回报文中抽取值，放入 LIST
```

```

xml.'soap:Body'. 'ns1:exchangeInResponse'. 'ns1:out'. result. each {
    tempArray=[]
    tempArray << it.balance.cent.text()
    tempArray << it.bizDate.text()
    tempArray << it.bizTime.text()[0..18]
    tempArray << it.bizType.text()
    tempArray << it.budgetCode.text()
    tempArray << it.instructionId.text()
    tempArray << it.memo.text()
    tempArray << it.partnerId.text()
    tempArray << it.partnerInstructionId.text()
    tempArray << it.partnerUserName.text()
    tempArray << it.pointcoreInstructionId.text()
    tempArray << it.providerCode.text()
    tempArray << it.subBizType.text()
    tempArray << it.subProviderCode.text()
    tempArray << it.userId.text()
    tempArray << it.pointAmount.cent.text()
    actualData << tempArray
}

```

而对于 JDBC 请求返回的简单的 XML 报文，处理的方式会更简单一些，如下：

```

//查询 gpt_ex_inflow_instruction 兑入流水表返回的结果
def responseAsXml = context.expand( '${JDBC
Request#ResponseAsXml#//Results[1]/ResultSet[1]}' )
def ResultSet1 = new XmlParser().parseText(responseAsXml)
//定义 LIST
List expectedData = []
List tempArray = []
//从返回报文中抽取值，放入 LIST
ResultSet1.Row.each{
    tempArray << it.BIZ_DATE.text()
    tempArray << it.BIZ_TIME.text()[0..18]
    tempArray << it.BIZ_TYPE.text()
    tempArray << it.BUDGET_CODE.text()
    tempArray << it.INSTRUCTION_ID.text()
    tempArray << it.MEMO.text()
    tempArray << it.PARTNER_ID.text()
    tempArray << it.OUT_INSTRUCTION_ID.text()
    tempArray << it.PARTNER_USER_NAME.text()
    tempArray << it.POINTCORE_INSTRUCTION_ID.text()
    tempArray << it.PROVIDER_CODE.text()
    tempArray << it.SUB_BIZ_TYPE.text()
    tempArray << it.SUB_PROVIDER_CODE.text()
    tempArray << it.USER_ID.text()
}

```

```
tempArray << it.POINT_AMOUNT.text()
add_amount = it.POINT_AMOUNT.text()
expectedData << tempArray
}
```

7.4. 动态定位表名

在测试中经常碰到：根据 userid 的倒数第二位，或倒数第二、三位来确定需要查询的流水表的需求，SoapUI 中可以使用 DataGen 步骤来解决这个问题，Type 选择 Script：

Properties

Name	Type	Mode	Shared	Last
mini_account_log_pt	Script	STEP	<input type="checkbox"/>	mini_account_log_pt_91
gpt_budget_use	Script	STEP	<input type="checkbox"/>	gpt_budget_use_03
gpt_deposit_instruction	Script	STEP	<input type="checkbox"/>	gpt_deposit_instruction_03
biz_no	Script	STEP	<input type="checkbox"/>	3129100006294903

Configuration

Property Script Script is invoked with context, testRunner, log variable

```
def account_no = testRunner.testCase.testSuite.project.getPropertyValue("account_no")
account_no = account_no[-6,-5]
def gpt_deposit_instruction = "mini_account_log_pt_" + account_no
log.info(gpt_deposit_instruction)
return gpt_deposit_instruction
```

- 变量定义在 project 级别，使用方法：
testRunner.testCase.testSuite.project.getPropertyValue 来读取属性值。
- 变量定义在 testStep 级别，使用方法：
testRunner.testCase.testSteps["DataSource"].getPropertyValue 来读取属性值。
- 变量定义在 testCase 级别，使用方法：
testRunner.testCase.setPropertyValue("user_id",userid)来设置属性值。

8. 测试工具的简单对比

使用 LoadRunner 提供的 Webservice 协议对同一个接口进行性能测试。

- 不加校验的脚本（脚本名称：LR_1）如下：

```
//@oolong 11/29/2010
Action()
{
    lr_start_transaction("here_start");
    web_service_call( "StepName=test1_101",
        "SOAPMethod=RequestJaxRPCService.RequestJaxRPC.test1",
        "ResponseParam=response",
        "WSDL=C:/Documents and Settings/Owner/桌面/RequestService.wsdl",
        "UseWSDLCopy=1",
```



```

        "Snapshot=t1264818214.inf",
        BEGIN_ARGUMENTS,
        "xml:sss=<sss><string></string></sss>",
        "id=aff",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
    lr_end_transaction("here_start", LR_AUTO);
    return 0;
}

```

- 加了校验的脚本（脚本名称：LR_2）如下，下面的脚本对返回的结果进行校验，类似 SoapUI 里提供的断言：

```

//@oolong      11/29/2010
Action()
{
    char com[] = "123";
    lr_start_transaction("here_start");
    web_service_call("StepName=test1_101",
        "SOAPMethod=RequestJaxRPCService.RequestJaxRPC.test1",
        "ResponseParam=response",
        "WSDL=C:/Documents and Settings/Owner/桌面/RequestService.wsdl",
        "UseWSDLCopy=1",
        "Snapshot=t1264818214.inf",
        BEGIN_ARGUMENTS,
        "xml:sss=<sss><string></string></sss>",
        "id=aff",
        END_ARGUMENTS,
        BEGIN_RESULT,
        "test1Return[1]=Param_result",
        END_RESULT,
        LAST);
    if(strcmp(lr_eval_string("{Param_result}"),com)==0)
    {
        lr_end_transaction("here_start", LR_AUTO);
        lr_vuser_status_message("成功");
    }
    else
    {
        lr_end_transaction("here_start", LR_FAIL);
        lr_error_message(lr_eval_string("{Param_result}"));
    }
    return 0;
}

```

- 负载测试的场景与 SoapUI 的场景一致：100 用户并发，持续运行 10 分钟，没有设置思考时间。对 LR_2 脚本进行性能测试后，发现响应时间比使用 SoapUI 进行测试的响应时间长，因此把校验过程注释掉，使用 LR_1 又进行了一次负载测试。
- LR 可以提供的结果分析图表较多，以下列出几个示意图：

Analysis Summary

Period: 30-01-2010 10:57:46 - 30-01-2010 11:08:04

Scenario Name: D:\Program Files\Mercury\LoadRunner\scenario\test.lrs
Results in Session: C:\Documents and Settings\Owner\Local Settings\Temp\res\res.lrr
Duration: 10 minutes and 18 seconds.

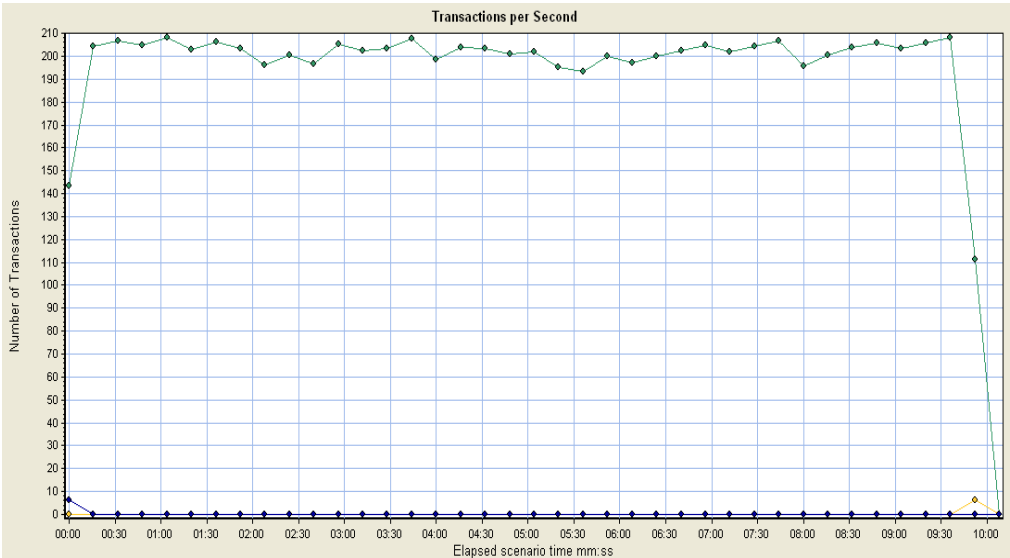
Statistics Summary

Maximum Running Vusers:	100	
Total Throughput (bytes):	109,483,911	
Average Throughput (bytes/second):	176,872	
Total Hits:	120,642	
Average Hits per Second:	194.898	View HTTP Responses Summary

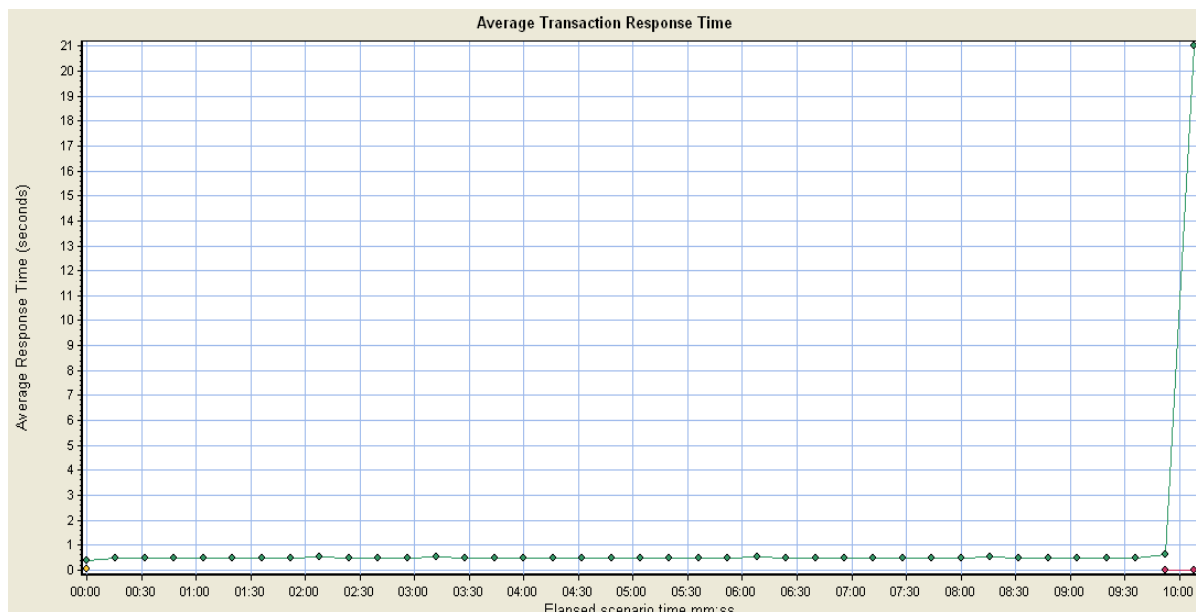
Transaction Summary

Transactions: Total Passed: 241,484 Total Failed: 4 Total Stopped: 0							Average Response Time	
Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Action Transaction	0.006	0.491	48.003	1.383	2.983	120,642	2	0
here_start	0.006	0.491	48.003	1.383	2.983	120,642	2	0
vuser_end Transaction	0	0	0	0	0	100	0	0
vuser_init Transaction	0	0.047	0.582	0.095	0.122	100	0	0

TPS 图如下：



平均事务响应时间如下：



可以看到由 LR 得到的结果，图表丰富，数据完整，提供了更好、更直观的说明作用。

■ 比较上面的负载测试结果数据

脚本名称	平均响应时间	总事务数	TPS
SoapUI 脚本	291.45MS	205856	339.04
LR 1	0.491S	120646	194.898
LR 2	0.606S	96636	159.464

由上表及图表，可以得到以下结论：

- ✓ SoapUI 是专门针对 webservice 接口的测试工具，在对同一个 webservice 接口进行测试时，SoapUI 表现出来的性能更优越，更真实、更接近地反映接口的性能指标。
- ✓ SoapUI 在发送请求时，是直接以组装好的 soap 报文进行发送，而 LR 是使用 web_service_call 方法，从方法传入相应的参数，由 LR 组装为 soap 报文后，再发往接口进行调用，因此 LR 在组装报文时，会有相应时间的耗费。LR 脚本中创建的事务，就包含了这段组装报文的时间，因此响应时间会比 SoapUI 的响应时间更长。LR 与 SoapUI 的差别应该还有更多，在此我尚未研究的更深入。
- ✓ 对于 LR，在测试中若增加对返回结果的校验，也会耗费一定的时间，从上面的数据可以看出，时间差大约 0.12s 左右，这也与校验中使用的方法有关系，如果方法高效的话，这个时间差也将更少。
- ✓ SoapUI 提供的结果数据的分析不如 LR 那么详细与全面，但对于接口级的测试已足够。

目前 webservice 接口有多种语言可以实现，除了 JAVA、C++，当前还有遇到 WCF，这类接口生成的 WSDL，LR 是无法直接读到接口的入参与出参，读取时直接失败，抛出异常，暂找不到解决方法。而使用 SoapUI，笔者已经测试过，可支持 java、c++，wcf 编写的 webservice 接口。