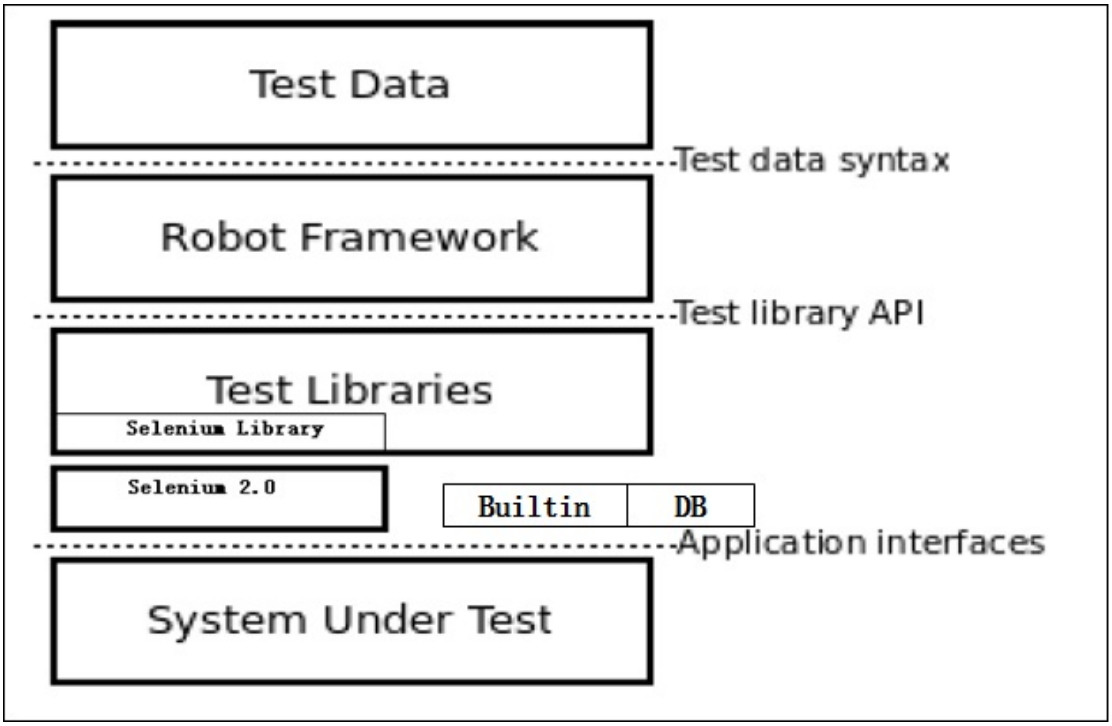
简介

http://robotframework.org/



Robot Framework is a generic test automation framework for acceptance testing and acceptance test-driven development (ATDD). It has easy-to-use tabular test data syntax and utilizes the keyword-driven testing approach. Its testing capabilities can be extended by test libraries implemented either with Python or Java, and users can create new keywords from existing ones using the same syntax that is used for creating test cases.

- Enables easy-to-use tabular syntax for creating test cases in a uniform way.
- Allows using keyword-driven, data-driven and behavior-driven (BDD) approaches.
- Provides ability to create reusable higher-level keywords from the existing keywords.
- Provides easy-to-read reports and logs in HTML format.
- Is platform and application independent.
- The modular architecture supports creating tests even for applications with several diverse interfaces.
- Provides a simple library API for creating customized test libraries.
- Provides a command line interface and XML based outputs for integration into existing build infrastructure (continuous integration systems).
- Provides support for Selenium for web testing, Java GUI testing, running processes, Telnet, SSH, and so on.
- Remote library interface enables distributed testing and implementing test libraries in any programming language.
- Provides tagging to categorize and select test cases to be executed.

- Has built-in support for variables, practical particularly for testing in different environments.

Robot Framework vs. Fitnesse vs. Cucumber

http://www.radekw.com/blog/2009/03/27/automation-frameworks/
http://robotframework.googlecode.com/svn/wiki/publications/Quality_Matters_Q1_2009_Collino.pdf

Recently I researched many testing frameworks in order to choose one to use at work. I spent few days practically testing the following frameworks:

- Robot Framework
- FitNesse
- Cucumber

I chose Robot Framework. It is not perfect, but promising, extensible, and actively developed by the open source community. The following features made me choose this framework over others:

- Keyword driven
- Ability to use re-usable keywords
- Large collection of built-in keywords
- Test cases and suites are HTML documents
- RobotIDE test case editor
- Extendable in Python and Jython
- Reporting using HTML and XML files

All this gives a solid base to extend and wrap around in order to create state-of-the art testing framework. Robot framework is easily installable. Examples and documentation are more than enough to get automation engineers and testers started. It requires some learning and getting used to (especially RobotIDE) but once grasped, creating test cases is easy. Additionally, new keywords can be simply programmed in Python or Jython, or created with existing keywords (macros). Tests are started with a command line tool and can be very simply scheduled using cron.

FitNesse framework is widely used and popular, but is not as extendable as robot framework. It's a testing wiki. Test cases are created as tables in wiki documents and are started by clicking a button on a page. The results show up immediately, but there is no way to store them. Each test table must have a code fixture written in Java. Other languages can be used with a help of additional Fit servers. ~~Being a wiki, FitNesse cannot be used for automatic and non-interactive testing.~~ Automation can be achieved by using command line runner or ant tasks.

Cucumber is a testing environment that allows testers to write tests in a a domain-specific language based on a spoken language. It's behavior-driven, but all used sentences must be mapped in underlying programming language (Ruby by default). It's easy to transcribe users stories into test cases, but it requires automation engineer to work with testers writing every test case. On the other hand, it's a very interesting and amazing way to write unit tests.

目前 robotFramework 支持的测试库

https://github.com/robotframework
https://github.com/bulkan/robotframework-difflibrary

http://robotframework.org/#test-libraries

https://code.google.com/p/robotframework/wiki/TestLibraries

标准测试库：

- BuiltIn
- OperatingSystem
- Screenshot
- Telnet
- Collections
- String
- Dialogs
- Remote
- XML (new in RF 2.7.4)

外部测试库：

This is a list of publicly available test libraries that can be used with Robot Framework but need to be installed separately. Please refer to the individual project pages for information about installing and using them.

- SeleniumLibrary - A web testing library that uses popular Selenium tool internally.
- Selenium2Library - A drop-in-replacement for SeleniumLibrary using newer Selenium 2 WebDriver API.
- Selenium2Library Java port - Java implementation of Se2Lib. Compatible with Jython 2.5.
- watir-robot - A web testing library that uses popular Watir tool via the remote library interface.
- WatinLibrary - A web testing library that uses Watin tool (a .NET port of Watir) via the remote library interface.
- SwingLibrary - A Swing GUI testing library.
- EclipseLibrary - A library for testing Eclipse RCP applications using SWT widgets.
- AutoItLibrary - Windows GUI testing library that uses AutoIt freeware tool as a driver.
- DatabaseLibrary (Java) - A test library that provides common functionality for testing database contents. Implemented using Java so works only with Jython.
- DatabaseLibrary (Python) - Another library for database testing. Implemented with Python and works also on Jython.
- SSHLibrary - A test library that enables SSH and SFTP.
- HTTP test library using livetest
- HTTP test library using Requests

- SudsLibrary - library for testing SOAP-based web services
- AndroidLibrary - Library for Android testing that uses Calabash Android internally.
- IOSLibrary - Library for iOS testing that uses Calabash iOS Server internally.
- Rammbock - Generic network protocol test library.
- How-To: Sikuli and Robot Framework Integration - This is not really a library but these instructions explain how to integrate Sikuli tool with Robot Framework

Found these also, might be useful to add them as well (to the external libraries):

- PhantomJS and Zombie.js libraries (similar to SeleniumLibrary? but headless)
- Generic Network Protocol test library
- REST specific HTTP library
- SoapUI library (similar approach to Sikuli lib, but with SoapUI)
- IOS (Iphone/Ipad) test automation library (uses Calabash IOS Server to drive IOS device)
- Android test automation library (uses Calabash Android Server to drive android device)
- ADB (android) library
- VMWare and HyperV virtual machine libraries
- Library supporting debugging in robot framework
- Diff style file comparison library
- Closure webdriver library
- Sikuli integration libraries here and here (The how-to is helpful, but not exactly a library. Checkout here for code based on the how-to though)
- email library
- Image Comparison library
- HTML Check library (doesn't look maintained though, for shame Janne :D)

安装

https://code.google.com/p/robotframework/wiki/Installation

Windows 下的一键安装：
https://code.google.com/p/robotframework/wiki/Installation#One_Click_Installer

RobotFramework 的中文支持

由于初始并不支持中文，于是在生成测试报告的时候对于中文的关键字无法识别，使用下面的方法可以正常显示中文。

•在[PythonDir]\Lib\site-packages\robot\utils 下的 encoding.py 文件中，找到 def _get_output_encoding(): 原来的编码是'cp437' 将其改成' cp936'(简体中文,gbk) 。

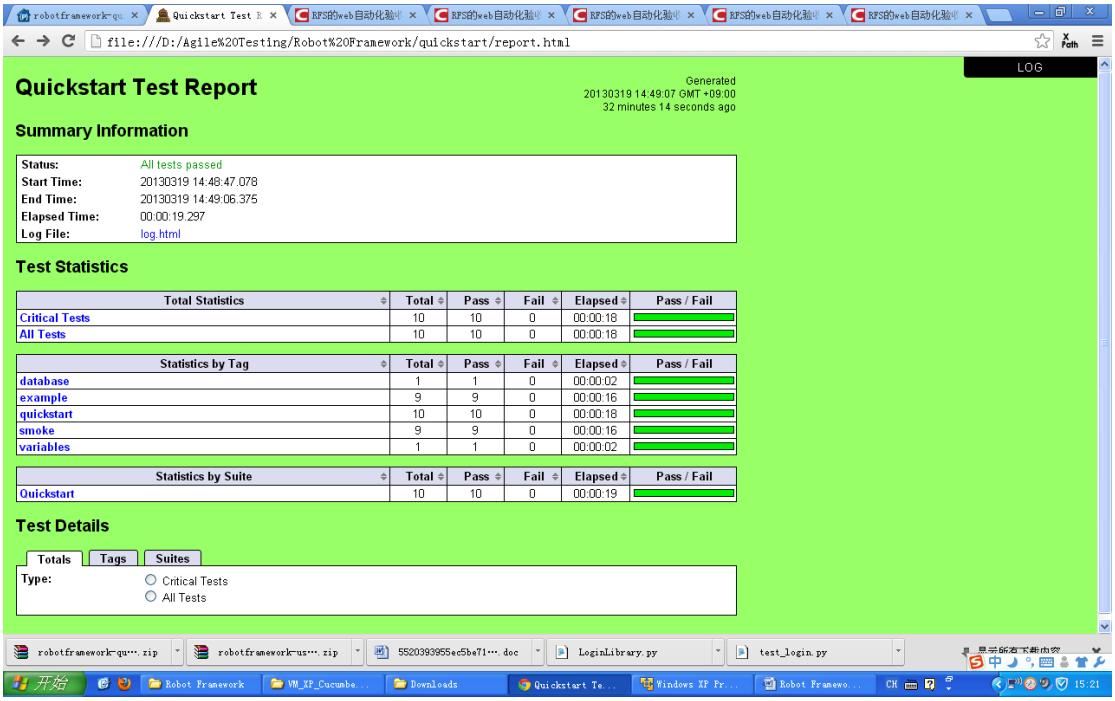
```
def _get_output_encoding():
    # Jython is buggy on Windows: http://bugs.jython.org/issue1568
    if os.sep == '\\' and sys.platform.startswith('java'):
        #return 'cp437'  # Default DOS encoding
        return 'cp936'  # Default DOS encoding
    encoding = _get_encoding_from_std_streams()
    if encoding:
        return encoding
    if os.sep == '/':
        return _read_encoding_from_unix_env()
    #return 'cp437'  # Default DOS encoding
    return 'cp936'  # Default DOS encoding
```

使用指南

http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html?r=2.7.7

入门指南（Python Demo）

http://blog.chinaunix.net/uid-28266860-id-3372629.html
https://code.google.com/p/robotframework/source/browse/doc/quickstart/?name=2.6a1
D:\Agile Testing\Robot Framework\quickstart



标准测试库

https://code.google.com/p/robotframework/wiki/TestLibraries

Operating System 测试库

http://robotframework.googlecode.com/hg/doc/libraries/OperatingSystem.html?r=2.7.7

`OperatingSystem` is Robot Framework's standard library that enables various operating system related tasks to be performed in the system where Robot Framework is running. It can, among other things, execute commands (e.g. `Run`), create and remove files and directories (e.g.`Create File`, `Remove Directory`), check whether files or directories exists or contain something (e.g. `File Should Exist`, `Directory Should Be Empty`) and manipulate environment variables (e.g. `Set Environment Variable`).

参考：
http://robotframework.googlecode.com/svn/wiki/publications/testingexperience01_09_Collino.pdf

Telnet test library（Telnet 库）

`Telnet` is Robot Framework's standard library that makes it possible to connect to Telnet servers and execute commands on the opened connections.

The full library documentation by release is found below. The documentation of the latest minor version (e.g. 2.0.4) is always applicable with earlier minor versions.

- Version 2.7.7
- Version 2.6.3
- Version 2.5.7
- Version 2.1.3
- Version 2.0.4
- Development version

Robot Framework + AutoIt

AutoItLibrary

https://code.google.com/p/robotframework-autoitlibrary/

Overview

AutoItLibrary is a Python keyword library that extends Robot Framework by providing keywords based on the COM interface to AutoIt, a freeware tool for automating the Windows GUI.

In order to do screenshots, the AutoItLibrary requires the Open Source Python Image Library tool PIL.

News

- *2010-04-23* **AutoItLibrary 1.1** released
- *2009-11-17* **AutoItLibrary 1.0** released

Prerequisites

- Install Python and Robot Framework

- If you are not using ActivePython, which comes with the Python for Windows extensions such as win32com, then you'll need to install Mark Hammond's Python for Windows Extensions.
- If you want AutoItLibrary to do screen captures then
    - Install the Open Source Python Image Library tool PIL
- Download the latest AutoItLibrary release

Installation

AutoItLibrary installs its own files and the required parts of AutoIt. To install, unzip the downloaded source release file into a temporary directory on your PC, open a command window in that directory and type:

python setup.py install

NOTE: In Windows Vista and Windows 7 you must do this installation as Administrator!

The installation creates the folder:

C:\RobotFramework\Extensions\AutoItLibrary

on your PC and puts various files into this directory folder.

安装过程提示错误：
C:\AutoItLibrary-1.1\AutoItLibrary-1.1>python setup.py install
%SYSTEMROOT%\system32\regsvr32.exe /S C:\RobotFramework\Python\Lib\site-packages\AutoItLibrary\lib\AutoItX3.dll
AutoItLibrary requires win32com. See http://starship.python.net/crew/mhammond/win32/.
需要下载 pywin32：
http://sourceforge.net/projects/pywin32/files/

Documentation

AutoItLibrary documentation is installed by the installation process into

C:\RobotFramework\Extensions\AutoItLibrary\AutoItLibrary.html

The AutoItX documentation is also installed into this folder as AutoItX.chm. You can also view the Keyword Documentation online.

http://robotframework-autoitlibrary.googlecode.com/svn/tags/robotframework-AutoItLibrary-1.0/doc/AutoItLibrary.html

Tests

The AutoItLibrary installer puts a suite of self-tests here:

C:\RobotFramework\Extensions\AutoItLibrary\tests

To run these tests, which exercise the Windows Calculator GUI, run the RunTests.bat file in the above folder.

Robot Framework + Java

JavaIntegration

https://code.google.com/p/robotframework/wiki/JavaIntegration

安装 Jython

Using test libraries implemented with Java or using Java tools internally requires running Robot Framework on Jython, which in turn requires Java Runtime Environment (JRE). Starting from Robot Framework 2.5, the minimum supported Jython version is 2.5 which requires Java 1.5 (a.k.a. Java 5) or newer. Earlier Robot Framework versions support also Jython 2.2.

Installing Jython is a fairly easy procedure, and the first step is getting an installer from the Jython homepage. The installer is an executable JAR package, which you can run from the command line like `java -jar jython_installer-<version>.jar`. Depending on the system configuration, it may also be possible to just double-click the installer.

NOTE:

Running Robot Framework on Jython using the `jybot` start-up script requires `jython` to be executable directly in the system. This means that you need to make sure it is in PATH.

JavalibCore

https://code.google.com/p/robotframework-javatools/wiki/JavalibCore

Javatoolstest

https://github.com/robotframework/JavatoolsTest

JDave - BDD framework for Java
http://jdave.org/

RobotFramework + C#

https://code.google.com/p/robotframework/wiki/DotNetSupport
https://code.google.com/p/robotframework/wiki/HowToUseCSharp

Installation 安装

Preconditions 必备条件

1. Install .NET Framework. We have only tested with the 4.0 series, but earlier and newer versions should work too assuming you can install IronPython on them.（安装,NET Framework 4.0）
2. Install IronPython. We have tested with 2.6.2 and 2.7 releases but possible newer 2.6.x and 2.7.x releases ought to work too.(安装 IronPython 2.7)
3. Install the source distribution of elementtree module. We have only tested with the 1.2.7 preview release, which should have better IronPython support than 1.2.6. See Installing Python modules section below for installation instructions.
4. If you are using IronPython 2.6.x with Robot Framework 2.6.3 or newer, install IronPython.Zlib module. This module is included into IronPython 2.7 so separate installation is not needed with that release.
5. Optionally, you can add IronPython installation directory into `PATH` to ease running `ipy.exe` from the command line.

Installing Python modules

1. Get and extract the source distribution package of the module you want to install. Alternatively you may be able to checkout the source code directly from the project's version control.(下载 elementtree1.2.7 preview release、robotframework-2.7.7.tar.gz)
2. Go to the extracted or checked out directory from the command line.
3. Install the module with command `ipy setup.py install`. If you do not have `ipy` in `PATH`, you need to use a full path to it. With IronPython 2.6.x you also need to add `--no-compile` option like in `ipy setup.py install --no-compile`.

Robot Framework（安装 robotframework-2.7.7.tar.gz）

1. Get the source code either as downloading a source distribution or by doing a checkout.
2. See Installing Python modules section above for installation instructions.

Running the example

Create a file ipybot.bat as follows. Make sure to replace [IRONPYTHON_INSTALL_FOLDER] with the installation path of IronPython on your system:

```
:: IronPython executable to use.

set ironpython="[IRONPYTHON_INSTALL_FOLDER]\ipy.exe"

:: Path to robot\runner.pypath

set runner="[IRONPYTHON_INSTALL_FOLDER]\lib\site-packages\robot\runner.py"

:: Run Robot on IronPython interpreter

%ironpython% %runner% %*
```

This file emulates the pybot.bat that comes with the installation of the Robot framework on Python. Once all the other files have been created and copied to the same directory, run the command:

ipybot.bat <test file>

System under test（编写被测代码）

The demo application is a very simple C# class whose two methods simply add two numbers and concatenate two strings, which should be enough for the purpose of this exercise.

Combine.cs (to be compiled as a Dll)

```
namespace Test
{
    public class Combine
    {
        public int AddNumbers(int number1, int number2)
        {
            return number1 + number2;
```

```
        }


        public string AddStrings(string string1, string string2)
        {
            return string1 + " " + string2;
        }
    }
}
```

Test library（编写测试库）

A simple test library that can interact with the above using the .Net CLR. The methods' names in that library will act as keywords for the Robot Framework.

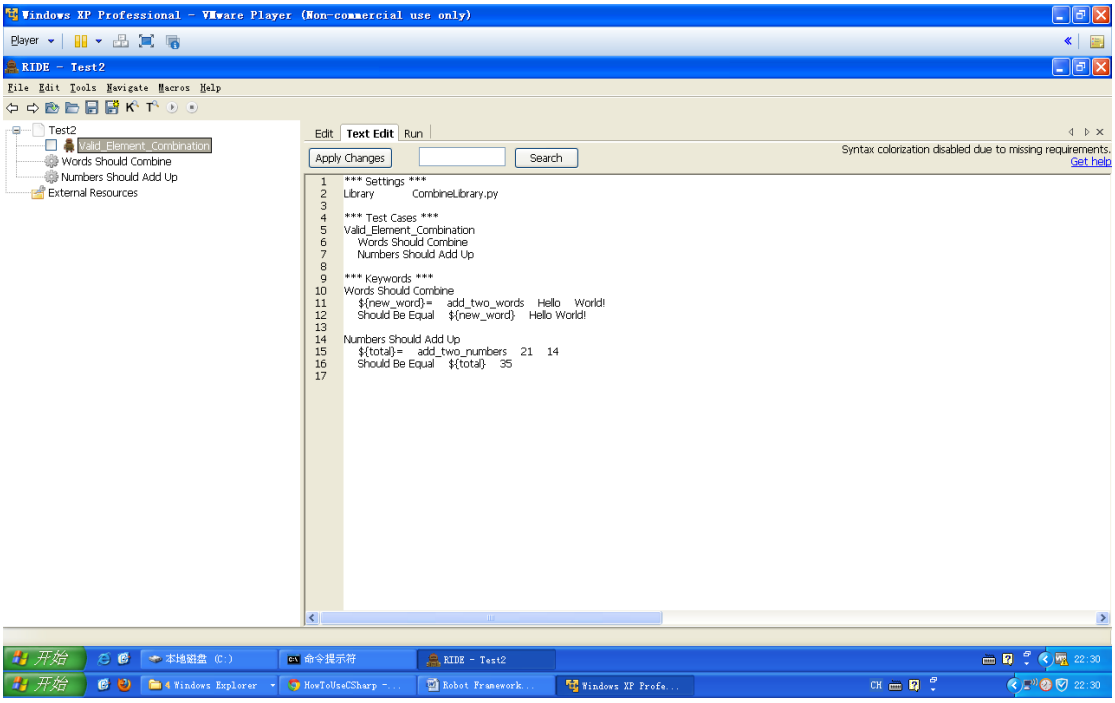
CombineLibrary.py

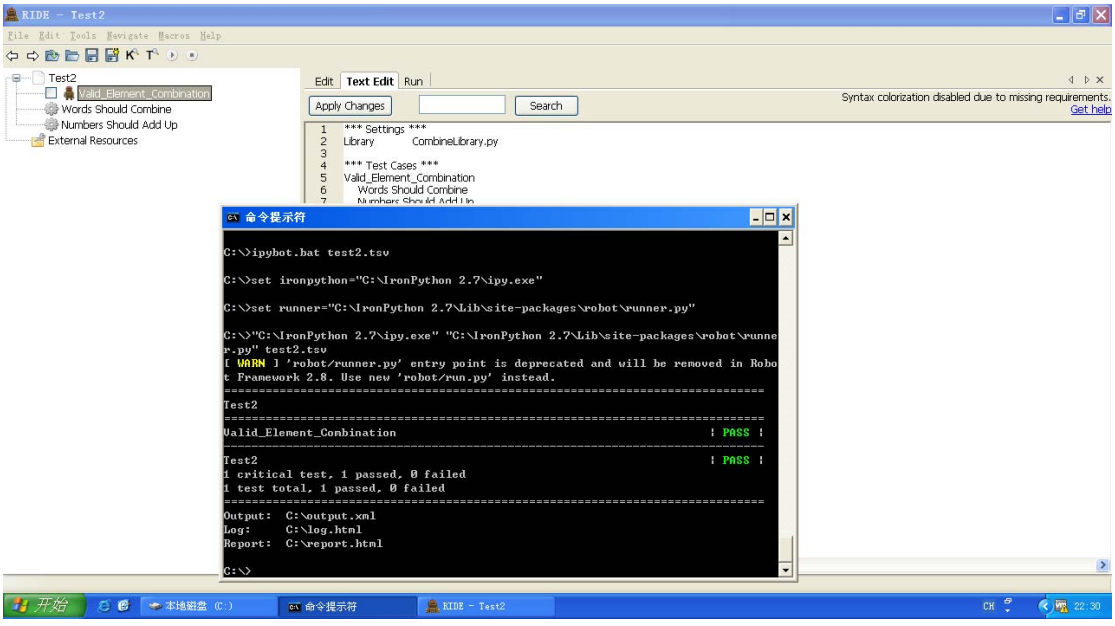
```
import clr
clr.AddReferenceToFileAndPath('Combine.dll') #include full path to Dll if required
from Test import Combine


def add_two_numbers(num1, num2):
        try:
                intNum1 = int(num1)
                intNum2 = int(num2)
        except:
                raise Exception("Values must be integer numbers!")
        cmb = Combine()
        total = cmb.AddNumbers(intNum1, intNum2)
        return str(total)


def add_two_words(word1, word2):
        cmb = Combine()
        return cmb.AddStrings(word1, word2)
```

Test cases（编写测试用例）

*** Settings ***				
Library	CombineLibrary.py			
*** Test Cases ***				
Valid_Element_Combination				

	Words Should Combine			
	Numbers Should Add Up			
*** Keywords ***				
Words Should Combine	${new_word} =	add_two_words	Hello	World!
	Should Be Equal	${new_word}	Hello World!	
Numbers Should Add Up	${total} =	add_two_numbers	21	14
	Should Be Equal	${total}	35	





Robot Framework + Selenium

先安装：

Java

wxPython

robotframework-ride

robotframework-selenium2library

https://code.google.com/p/robotframework-seleniumlibrary/

SeleniumLibrary is a Robot Framework test library that uses the popular Selenium web testing tool internally. It provides a powerful combination of simple test data syntax and support for different browsers. In addition to standard web testing, the library also supports testing Adobe Flex/Flash applications.

See the ***SeleniumLibrary demo*** for executable example test cases and reports, look at the library documentation for information about the provided keywords, and consult the installation instructions if you want to use the library yourself.

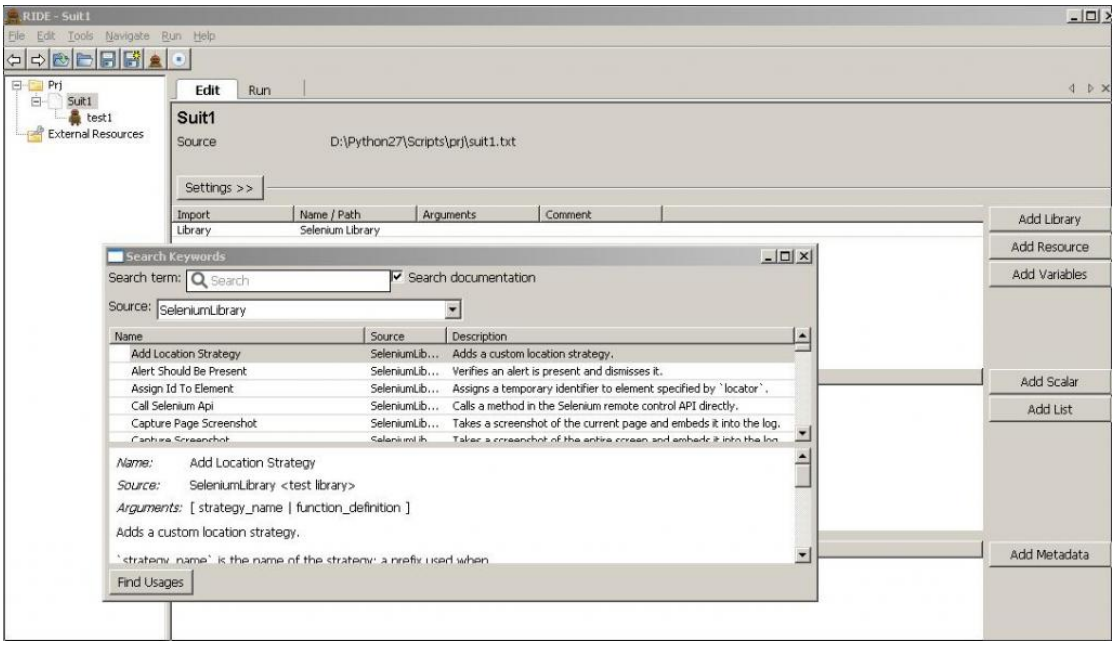
Selenium 1 vs. Selenium 2

SeleniumLibrary internally uses Selenium Remote Controller API that is part of Selenium 1. If you want to use the new Selenium 2 WebDriver API, you should look at Selenium2Library that is, for most parts, a drop-in-replacement for SeleniumLibrary.

According to http://seleniumhq.org, the old Remote Controller API is officially deprecated in favor of the new WebDriver API. As a result also***SeleniumLibrary is deprecated*** and no new releases are expected. New users should use the already mentioned Selenium2Library and existing users should start to plan migrating to it.

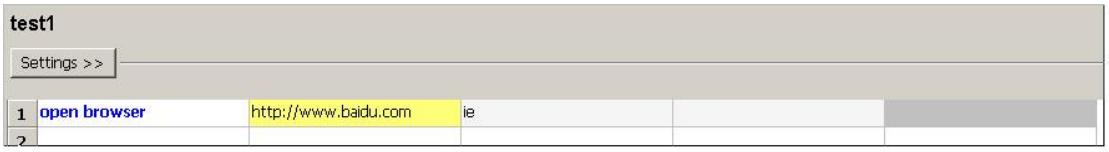
参考：
http://cgmblog.sinaapp.com/html/204.html
http://blog.csdn.net/tulituqi/article/details/7592711
http://blog.csdn.net/tulituqi/article/details/6834037

添加 SeleniumLibrary：



设置启动 Selenium Server：

编写关键字驱动测试脚本：



对应的文本文件是:

*** Settings ***

Suite Setup Start Selenium Server

Suite Teardown StopSelenium Server

Library SeleniumLibrary

*** Test Cases ***

case1

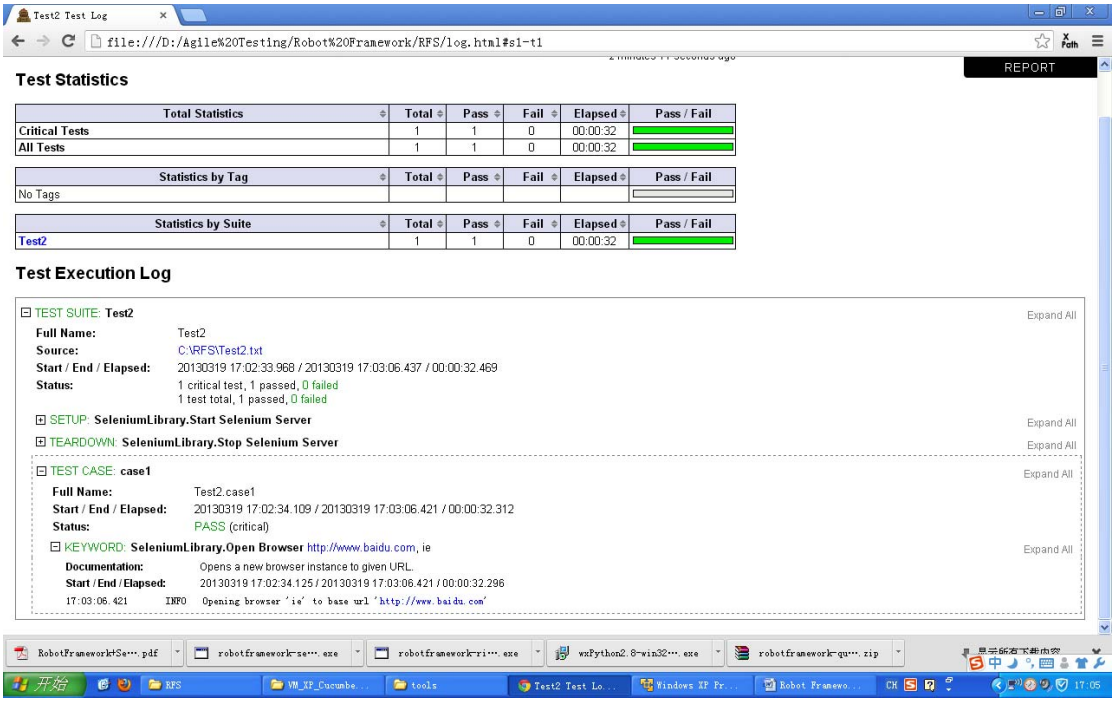
open browser http://www.baidu.com ie

按 F8 执行测试脚本

命令行执行方式：

Pybot Test2.txt

执行会在目录中生成报告文件



Robot Framework 测试 WebService

Robot Framework + SoapUI

https://github.com/pavlobaron/robotframework-soapuilibrary

robotframework-sudslibrary

https://github.com/ombre42/robotframework-sudslibrary

SudsLibrary is a library for functional testing of SOAP-based web services. SudsLibrary is based on Suds, a dynamic SOAP 1.1 client.

This library is in a beta development stage.

See the Keyword Documentation for more information about using the library.

安装 pip：

http://blog.csdn.net/maowenbin/article/details/6622307

再安装 robotframework-sudslibrary

```
pip install ez_setup

pip install robotframework-sudslibrary
```

测试脚本例子：

```
*** Settings ***
Library                 SudsLibrary
Library                 XML
Library                 Collections

*** Test Cases ***
TestCase1
    Create Soap Client      http://www.webservicex.net/Statistics.asmx?WSDL
    ${dbl array}=       Create Wsdl Object      ArrayOfDouble
    Append To List      ${dbl array.double}     2.0
    Append To List      ${dbl array.double}     3.0
    ${result}=      Call Soap Method        GetStatistics       ${dbl array}
    Should Be Equal As Numbers      ${result.Average}       2.5
```

帮助文档：

http://ombre42.github.com/robotframework-sudslibrary/doc/SudsLibrary.html

Robot Framework 测试 HTTP

robotframework-httplibrary

https://github.com/peritus/robotframework-httplibrary

robotframework-httplibrary is a Robot Framework test library for all your HTTP needs. It uses livetest (which, in turn uses the famouswebtest library underneath).

安装：

```
pip install --upgrade robotframework-httplibrary
```

使用方法：

API documentation can be found at http://peritus.github.com/robotframework-httplibrary/, here is an example on how to use it:

Setting	Value
Library	HttpLibrary.HTTP

Test Case	Action	Argument
Example		
	[Documentation]	Follows a Redirect
	Create HTTP Context	httpstat.us
	GET	/302
	Response Status Code Should Equal	302
	Follow Response	
	Response Body Should Contain	generating different HTTP codes

You can view a report and a log of this test executed that looks like this:

对中文支持不大好！

robotframework-requests

https://github.com/bulkan/robotframework-requests

RequestsLibrary is a Robot Framework test library that uses the Requests HTTP client.

You need to have requests installed

NOTE: Only support requests 0.9.0 or lower

```
pip install -U requests==0.9.0
```

Now install robotframework-requests

```
pip install -U robotframework-requests
```

Settings	
Library	Collections
Library	RequestsLibrary
Test Cases	
Get Reque	

sts				
	Create Session	github	http://github.com/api/v2/json	
	Create Session	google	http://www.google.com	
	${resp}=	Get	github	/
	Should Be Equal As Strings	${resp.status_code}	200	
	${resp}=	Get	github	/user/search/bulkan
	Should Be Equal As Strings	${resp.status_code}	200	
	${jsondata}=	To JSON	${resp.content}	
	Dictionary Should Contain Value	${jsondata['users'][0]}	Bulkan Savun Evcimen	

robotframework-restlibrary

https://code.google.com/p/robotframework-restlibrary/

REST Library is a test library for HTTP/REST, primarily designed for Robot Framework.

It enables HTTP dialogues using HTTP verbs, setting request headers and verifying the response headers and body.

It includes (optional) support for parsing the body (currently requiring lxml). Apart from XML support it works with both Python and Jython.

Install

You can install the latest trunk by invoking:

```
$ sudo easy_install http://robotframework-restlibrary.googlecode.com/svn/trunk#egg=RestLibrary-dev
```

Or manually by checking out the source code and then do:

```
$ sudo python setup.py install
```

Robot Framework 与 DB

Robotframework-Database-Library（Python）

http://franz-see.github.io/Robotframework-Database-Library/

安装：
easy_install robotframework-databaselibrary

使用方法：
http://blog.sina.com.cn/s/blog_654c6ec70100u9fr.html

在 Robot Framework 的主页上有 2 个 DataBase 相关的扩展库，一个是 Java 实现，一个是 Python 实现。Java 实现文档做的比较好，有详细说明，也有样例文档。但是 Python 实现的文档太少了，搞通它得费一番周折。在折腾了一段时间后我终于把它调通顺了，做个记录，也希望能帮助别人迅速解决问题。需要说明的是：我是用的 oracle 数据库。

- **工作原理**

任何一个 Robot Framework 的 Library 基本上都是一个双层结构：外层的皮，实现标准接口供 Robot Framework 调用；里面的瓤，实现具体的功能，提供 API 供外层的皮进行封装。

DataBaseLibrary 也不例外，你从它的主页 Download 标签处下载的只是一个皮。

为了使它能够真正工作，你还得下载一个瓤---------一个符合 Python 数据库接口规范的库文件。

两部分缺一不可。

- **安装介绍**

前提是你已经安装好了 Robot Framework 环境，如果没有，可参考下面文章：

http://blog.sina.com.cn/s/blog_654c6ec70100tkxn.html

1.先下载安装瓤，我们从这个主页能够找到所有支持 Python 数据库规范的实现：

http://wiki.python.org/moin/DatabaseInterfaces

由于要连接 Oracle ，我这里要进入 Oracle 的支持页面

http://wiki.python.org/moin/Oracle

选择第二个 cx_oracle 的主页，进入下载列表

选择合适的版本下载并安装。我选择的是 Windows x86 Installer (Oracle 10g, Python 2.6)

安装好了以后你的 python\Lib\site-packages 下多了一个 cx_Oracle.pyd 文件，这个目录下同时也多了个文件夹

cx_Oracle-5.1-py2.6.egg-info，cx_oralce 的使用文档就在里边。

2.再下载安装皮，我们从这个地方下载 DatabaseLibrary 的皮，这是个压缩文件。

http://github.com/franz-see/Robotframework-Database-Library/tarball/master

下载解压后，在命令行进入解压目录，执行 setup.py install

安装好了以后 DatabaseLibrary 这个目录也被安装到了 site-package 目录下。

安装就算完成了。

- **使用**

在 Ride 下编辑

Setting	Value			
Library	DatabaseLibrary			

Test Case	Action	Arguments		
ConnectTest	Connect To Database Using Custom Params	cx_Oracle	'username','password','your TNS Name'	
	@{rs}	query	select * from table1	
	Log many	@{rs}		
	Disconnect from Database			

这样我们就完成了第一个选择 table1 的操作。

需要说明的是 Connect To Database Using Costom Params 的参数：由于我使用了 cx_Oracle 这个库，所以第一个参数指定它；第二个参数是连接串 用户名，密码，和要连接的 Oracle 数据库的本地服务命名（与你 oracle 目录\network\admin\tnsnames.ora 文件中的描述相对应）。

cx_Oracle 依赖 Oracle 的客户端环境，这点需要注意。

运行，得到 Pass：）

- **中文显示问题**

运行后查看 Log，傻眼了。表中的中文全部变成了一串乱码。翻阅资料，原来是 cx_Oracle 的默认编码不是 Utf-8。需要人工转换一下。而 DataBaseLibrary 又没有提供转换函数，为了解决这个问题，我们得对 DataBaseLibrary 进行扩展了。打开安装目录下的 query.py 文件，在后面加一个函数

```
def decode(self,customstr):
    return customstr.decode('utf-8')
```

对所有需要变成中文的地方执行 Decode 关键字，就能得到中文了：

${a}	decode	${rs[0][2]}

这样还是有些麻烦，其实我们可以重写 query 关键字，让它把所有的值都做一下 decode。

- **扩展问题**

DatabaseLibrary 提供的关键字挺少，有时候不够用（例如不能 insert），这时候我们就需要对它进行扩展了。扩展原理同上，可以按需扩展：）

robotframework-dblibrary（JAVA）

参考：
http://thomasjaspers.github.io/robotframework-dblibrary/DatabaseLibrary_v20.html#Check Content For Row Identified By Where Clause

http://www.51testing.com/?uid-276050-action-viewspace-itemid-831122

RobotFramework 能用 Python 和 Jython 两条腿走路。但有的时候你得选一条。今天就碰上个问题，为了整合其它模块必须用 Java 实现的 DataBaseLibrary

其实它很简单，记录步骤如下：

1.在 RF 的主页找到 DataBaseLibrary（Java）的主页。
http://code.google.com/p/robotframework-dblibrary/
下载 dblibrary-1.0.jar 这个库文件

2.在 Oracle 的主页的 download 中找到 JDBC 的驱动程序下载下来（不同版本驱动不同，我用的 10g 的驱动 class12.jar），或者从你安装的 Oracle 目录的 jdbc 子目录中找到驱动备用。

3.将这两个 jar 文件全部加入环境变量 CLASSPATH

4.在 RF 里编写测试脚本

连接的例子如下:

Setting	Value
Library	org.robot.database.keywords.DatabaseLibrary

Test Case	Action	Arguments		
Test	Connect to DataBase	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@xxx.xxx.xxx.xxx:1521:sid	username
	...	password		

具体使用情况可见用户手册：

http://robotframework-dblibrary.googlecode.com/svn/tags/robotframework-dblibrary-1.0/doc/DatabaseLibrary.html

两个 Tip：

1.为了便于多人使用，建议使用 bat 动态在 Classpath 加入 Jar 包。而不是直接写到环境变量中（这样慢慢的你的 classpath 就乱死了)。

给一个参考例子：

```
set CP=%CLASSPATH%
for %%j in (*.jar) do ( call :set_cp %%j )
set CLASSPATH=%CP%

jybot --outputdir %%  --suitestatlevel 1 %*

goto :eof

:set_cp
set CP=%CP%%%\%1;
goto :eof
```

2.这个 classLibrary 是开源的，写的很粗糙，觉得不爽可以自己改一改，它的代码 host 在 google code 上

可用 SVN checkout

http://code.google.com/p/robotframework-dblibrary/source/checkout

Robotframework 与 SSH

https://code.google.com/p/robotframework-sshlibrary/

简介

SSH Library is a test library for Robot Framework. It enables execution of commands on a remote machine over an SSH connection. It also offers possibility to transfer files from and to a remote machine.

SSH Library works with both Python and Jython interpreters, but it requires SSH modules to be installed separately. With Python you needParamiko (which in turn requires PyCrypto), and with Jython Trilead SSH is required. For more information see installation instructions.

Keyword documentation.

An example test suite can be downloaded and executed.

News

2012-08-31 **Version 1.1** released.

2010-11-08 **Version 1.0** released.

2009-03-05 **Version 0.9** released.

安装

Preconditions

When using Python

A precondition for using SSH Library with Python is having SSH module named paramiko installed. Paramiko itself has some other requirements listed in installation instructions (mainly PyCrypto) and these must naturally be installed too. PyCrypto binaries for Windows are available here for 32 bit Windows and here for 64 bit Windows.

In Windows, paramiko has to be installed from the source distribution, by unzipping the download, and then running `python setup.py install`in the unzipped directory.

When using Jython

Paramiko doesn't work with Jython, but Trilead SSH for Java can be used instead. The JAR package containing Trilead SSH tool must be found from the CLASSPATH during the test execution.

See Robot Framework user guide for more information about setting the environment for test execution.

用 jybot 命令行执行

Installing SSH Library

SSH Library is installed differently on Windows and on other operating systems.

Windows

Installing using Windows binary installer is really straightforward, just double click the provided Windows installer and follow the instructions.

Other operating systems

On other operating systems SSH Library must be installed using a source distribution. The procedure is following:

1. Extract the source distribution package
2. Open console and go the resulting directory
3. Run command `python setup.py install`

Root privileges may be required for the installation.

Robot Framework 测试 C

https://code.google.com/p/robotframework/wiki/HowToUseC

Introduction

This simple example demonstrates how to use C language from Robot Framework test libraries. The example uses Python's standard ctypes module, which requires that the C code is compiled into a shared library. This version is implemented and tested on Linux, but adapting it to other operating systems would require only changing compilation and name of the produced shared library.

Getting and running the example

The example is available on the download page as robotframework-c-example-<date>.zip. After downloading and unzipping the package, you should have all the files in a directory `robotframework-c-example`.

To try out the example first run `make` in the directory that was created when you unzipped the package. This will create library `liblogin.so`, a shared library that is needed to use `ctypes` module. The example tests can be executed using command `pybot LoginTests.tsv`. All the test should pass.

System under test

The demo application is a very simple login system, implemented using C, that validates the given user name and password and returns the status. There are two valid username password combinations: `demo/mode` and `john/long`.

```
#include <string.h>
#define NR_USERS 2

struct User {
    const char* name;
    const char* password;
};
const struct User VALID_USERS[NR_USERS] = { "john", "long", "demo", "mode" };

int validate_user(const char* name, const char* password) {
    int i;
    for (i = 0; i < NR_USERS; i++) {
        if (0 == strncmp(VALID_USERS[i].name, name, strlen(VALID_USERS[i].name)))
            if (0 == strncmp(VALID_USERS[i].password, password, strlen(VALID_USERS[i].password)))
                return 1;
    }
    return 0;
}
```

Test library

A simple test library that can interact with the above program using `ctypes` module. The library provides only one keyword `Check User`.

```
from ctypes import CDLL, c_char_p

LIBRARY = CDLL('./liblogin.so')  # On Windows we'd use '.dll'
```

```
def check_user(username, password):
    """Validates user name and password using imported shared C library."""
    if not LIBRARY.validate_user(c_char_p(username), c_char_p(password)):
        raise AssertionError('Wrong username/password combination')



if __name__ == '__main__':
    import sys
    try:
        check_user(*sys.argv[1:])
    except TypeError:
        print 'Usage:  %s username password' % sys.argv[0]
    except AssertionError, err:
        print err
    else:
        print 'Valid password'
```

The `if __name__ == '__main__'` block above is not used by the executed tests, but it allows using the library code as a tool for manual testing. You can test this handy behavior by running, for example, `python LoginLibrary.py demo mode` or `python LoginLibrary.py demo invalid` on the command line.

Test cases

*** Settings ***			
Library	LoginLibrary.py		
*** Test Case ***			
Validate Users			
	Check Valid User	john	long
	Check Valid User	demo	mode
Login With Invalid User Should Fail			
	Check Invalid User	de	mo
	Check Invalid User	invalid	invalid
	Check Invalid User	long	invalid
	Check Invalid User	${EMPTY}	${EMPTY}
*** Keyword ***			
Check Valid User	[Arguments]	${username}	${password}

	Check User	${username}	${password}
Check Invalid User	[Arguments]	${username}	${password}
	Run Keyword And Expect Error	Wrong username/password combination	Check User
	...	${username}	${password}

Robot Framework 测试 C++

参考：
http://stackoverflow.com/questions/2854831/using-robot-framework-for-atdd

We have been using Robot Framework at my place of work for several over a year now with moderate success. Like the poster, we also do C++ work. We took some time to evaluate Robot against Fitnesse/Slim and, at the time, both solutions were good, but the deciding factors were (as of ~2009):

- It was clearer how Robot and its reports would scale to large projects
- It wasn't obvious how to version control Fitnesse artifacts

From a technical perspective, we have been using SWIG to bridge between Robot and C++. We wrap our test fixtures in SWIG and link it with the production code under test - giving us a python module that can be imported by Robot.
We use the .txt format for Robot input almost exclusively - we've found that this version controls better, it's easier to read, and we simply weren't using the advanced features of HTML (which is where we started). In addition, we are using the "BDD Style" Robot syntax as well. We use GoogleMock with some wrappers to help us set expectations which are checked during the teardown of each Robot test.
As far as organizing the tests, we have settled on the following approach (with inspiration from Dale Emery's approach given here):

- Major functional hierarchy is represented by a folder structure.
- A feature-ish sized thing is described in a Robot test file name.
- A description of each part of that feature is used at the Robot test case name.
- An example is given as the steps in the test case.
- The example text is broken down into steps using Robot "keywords".
- The test fixture drives the production code.

For example, a phone might have something like this:

```
//  PhoneProject/MakingCalls/DialAPhoneNumber.txt

*** Test Case ***
A user can dial a US number with an area code, up to 10 digits
```

```
   Given a phone without any numbers dialed
   Expect the attempted phone number to be 123-456-7890
   When a user enters the numbers 1234567890


// A separate MakingCallsKeywords.txt or something similar
*** Keyword ***
Given a phone without any numbers dialed              CreateNewDialer
Expect the attempted phone number to be ${phoneNumber}  ExpectDialedNumber
${phoneNumber}
When a user enters the numbers ${numbersEntered}       DialNumbers
${numbersEntered}

// MakingCallsFixture.cpp  (which gets wrapped by SWIG)

std::wstring MakingCallsFixture::DialNumbers(const std::wstring& numbersEntered)
{
   ... Drive production code ...
}

// CreateNewDialer, ExpectDialedNumber also go here
```

We would then pair this up with unit tests that cover the corner conditions (e.g. ensure no more than 10 digits are allowed) - or maybe that would be another acceptance test (executable spec) depending on who is reading the tests and their familiarity with the domain.

We create a draft of these specs and review with domain experts and the team prior to beginning work. This has helped flush out a number of misunderstandings early on.

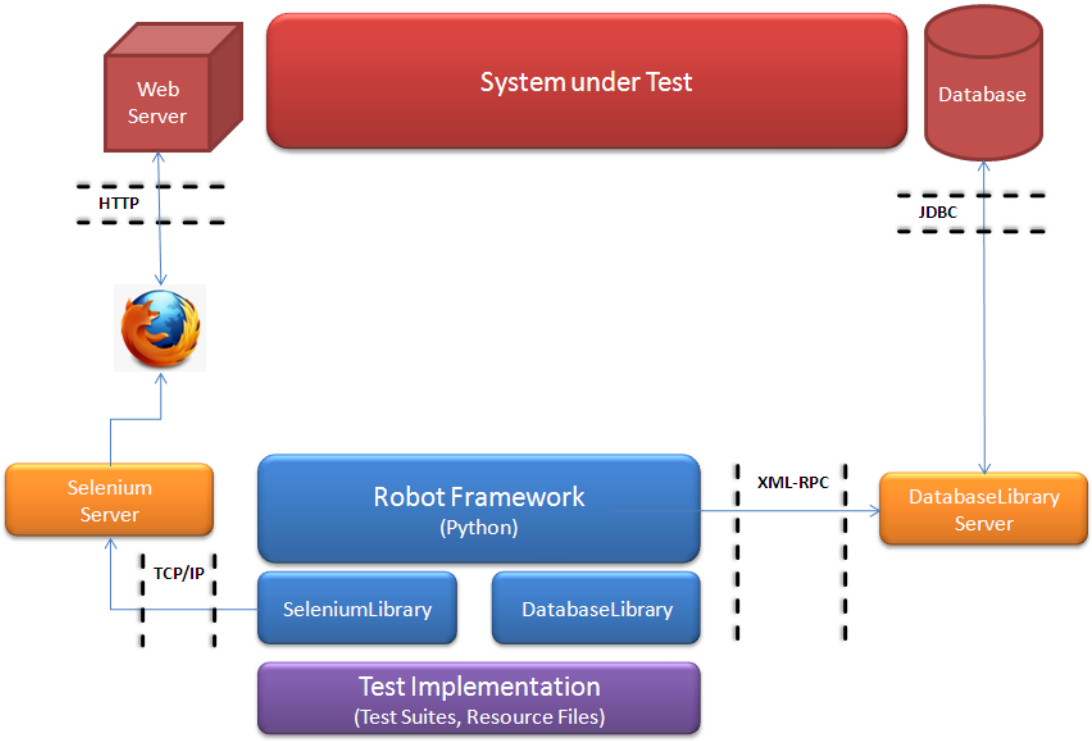
自定义 Test Library

参考：
http://blog.codecentric.de/en/2012/06/robot-framework-tutorial-writing-keyword-libraries-in-java/

Robot Framework 分布式测试

https://code.google.com/p/robotframework/wiki/RemoteLibrary

http://blog.codecentric.de/en/2012/04/robot-framework-tutorial-a-complete-example/
参考：



RobotFramework 与 BDD

http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html?r=2.7.6#different-test-case-styles

Behavior-driven style

It is also possible to write test cases as requirements that also non-technical project stakeholders must understand. These *Executable Requirements* are a corner stone of a process commonly called Acceptance Test Driven Development (ATDD).

One way to write these requirements/tests is *Given-When-Then* style popularized by Behavior Driven Development (BDD). When writing test cases in this style, the initial state is usually expressed with a keyword

starting with word *Given*, the actions are described with keyword starting with *When* and the expectations with a keyword starting with *Then*. Keyword starting with *And* may be used if a step has more than one action.

Example test cases using behavior-driven style

Test Case	Step
Valid Login	Given login page is open
	When valid username and password are inserted
	and credentials are submitted
	Then welcome page should be open

Ignoring *Given/When/Then/And* prefixes

Prefixes *Given*, *When*, *Then* and *And* are dropped when matching keywords are searched, if no match with the full name is found. This works for both user keywords and library keywords. For example, *Given login page is open* in the above example can be implemented as user keyword either with or without the word *Given*. Ignoring prefixes also allows using the same keyword with different prefixes. For example *Welcome page should be open* could also used as *And welcome page should be open*.

Embedding data to keywords

When writing concrete examples it is useful to be able pass actual data to keyword implementations. User keywords support this by allowing embedding arguments into keyword name.

RobotFramework + JavatoolsTest
https://github.com/robotframework/JavatoolsTest
JDave - BDD framework for Java
http://jdave.org/

Given/When/Then And Example Tables Using the Robot Framework
http://blog.codecentric.de/en/2009/11/givenwhenthen-and-example-tables-using-the-robot-framework/

命令行执行方式

```
pybot|jybot|ipybot [options] data_sources
python|jython|ipy -m robot.run [options] data_sources
python|jython|ipy path/to/robot/run.py [options] data_sources
java -jar robotframework.jar [options] data_sources
```

例：

Pybot TestSuites.txt

pybot -t "TestCase1" -d "C:\RobotFramework\results" ITF_Demo.txt

All command line options

This appendix lists all the command line options that are available when executing test cases with `pybot` or `jybot`, and when post-processing outputs with `rebot`.

Command line options for test execution

-N, --name *<name>*

Sets the name of the top-level test suite.

-D, --doc *<document>*

Sets the documentation of the top-level test suite.

-M, --metadata *<name:value>*

Sets free metadata for the top level test suite.

-G, --settag *<tag>*

Sets the tag(s) to all executed test cases.

-t, --test *<name>*

Selects the test cases by name.

-s, --suite *<name>*

Selects the test suites by name.

-i, --include *<tag>*

Selects the test cases by tag.

-e, --exclude *<tag>*

Selects the test cases by tag.

-c, --critical *<tag>*

Tests that have the given tag are considered critical.

-n, --noncritical *<tag>*

Tests that have the given tag are not critical.

-v, --variable *<name:value>*

Sets individual variables.

-V, --variablefile *<path:args>*

Sets variables using variable files.

-d, --outputdir *<dir>*

Defines where to create output files.

-o, --output *<file>*

Sets the path to the generated output file.

-l, --log *<file>*

Sets the path to the generated log file.

-r, --report *<file>*

Sets the path to the generated report file.

-x, --xunitfile *<file>*

Sets the path to the generated XUnit compatible result file.

-b, --debugfile *<file>*

A debug file that is written during execution.

-T, --timestampoutputs

Adds a timestamp to all output files.

--splitlog Split log file into smaller pieces that open in browser transparently.

--logtitle *<title>*

Sets a title for the generated test log.

--reporttitle *<title>*

Sets a title for the generated test report.

--reportbackground *<colors>*

Sets background colors of the generated report.

-L, --loglevel *<level>*

Sets the threshold level for logging. Optionally the default visible log level can be given separated with a colon (:).

--suitestatlevel *<level>*

Defines how many levels to show in the *Statistics by Suite* table in outputs.

--tagstatinclude *<tag>*

Includes only these tags in the *Statistics by Tag* table.

--tagstatexclude *<tag>*

Excludes these tags from the *Statistics by Tag* table.

--tagstatcombine *<tags:title>*

Creates combined statistics based on tags.

--tagdoc *<pattern:doc>*

Adds documentation to the specified tags.

--tagstatlink *<pattern:link:title>*

Adds external links to the *Statistics by Tag* table.

--removekeywords *<all|passed|for|wuks>*

Removes keyword data from the generated log file.

--listener *<name:args>*

Sets a listener for monitoring test execution.

--warnonskippedfiles

Show a warning when an invalid file is skipped.

--nostatusrc Sets the return code to zero regardless of failures in test cases. Error codes are returned normally.

--runemptysuite

Executes tests also if the top level test suite is empty.

--runmode *<mode>*

Sets the execution mode for this test run. Valid modes are ExitOnFailure, SkipTeardownOnExit, DryRun, and Random:<what>.

-W, --monitorwidth *<chars>*

Sets the width of the console output.

-C, --monitorcolors <*on|off|force*>

Specifies are colors used on the console.

-K, --monitormarkers <*on|off|force*>

Specifies are console markers (`.` and `F`) used.

-P, --pythonpath <*path*>

Additional locations where to search test libraries from when they are imported.

-E, --escape <*what:with*>

Escapes characters that are problematic in the console.

-A, --argumentfile <*path*>

A text file to read more arguments from.

-h, --help Prints usage instructions.

--version Prints the version information.

Command line options for post-processing outputs

-N, --name <*name*>

Sets the name of the top level test suite.

-D, --doc <*document*>

Sets the documentation of the top-level test suite.

-M, --metadata <*name:value*>

Sets free metadata for the top-level test suite.

-G, --settag <*tag*>

Sets the tag(s) to all processed test cases.

-t, --test *<name>*

Selects the test cases by name.

-s, --suite *<name>*

Selects the test suites by name.

-i, --include *<tag>*

Selects the test cases by tag.

-e, --exclude *<tag>*

Selects the test cases by tag.

-c, --critical *<tag>*

Tests that have the given tag are considered critical.

-n, --noncritical *<tag>*

Tests that have the given tag are not critical.

-d, --outputdir *<dir>*

Defines where to create output files.

-o, --output *<file>*

Sets the path to the generated output file.

-l, --log *<file>*

Sets the path to the generated log file.

-r, --report *<file>*

Sets the path to the generated report file.

-x, --xunitfile *<file>*

Sets the path to the generated XUnit compatible result file.

-T, --timestampoutputs

Adds a timestamp to all output files.

--splitlog Split log file into smaller pieces that open in browser transparently.

--logtitle *<title>*

Sets a title for the generated test log.

--reporttitle *<title>*

Sets a title for the generated test report.

--reportbackground *<colors>*

Sets background colors of the generated report.

-L, --loglevel *<level>*

Sets the threshold level to select log messages. Optionally the default visible log level can be given separated with a colon (:).

--suitestatlevel *<level>*

Defines how many levels to show in the *Statistics by Suite* table in outputs.

--tagstatinclude *<tag>*

Includes only these tags in the *Statistics by Tag* table.

--tagstatexclude *<tag>*

Excludes these tags from the *Statistics by Tag* table.

--tagstatcombine *<tags:title>*

Creates combined statistics based on tags.

--tagdoc *<pattern:doc>*

Adds documentation to the specified tags.

--tagstatlink *<pattern:link:title>*

Adds external links to the *Statistics by Tag* table. table in outputs.

--removekeywords *<all|passed|for|wuks>*

Removes keyword data from the generated outputs.

--starttime *<timestamp>*

Sets the starting time of test execution when creating reports.

--endtime *<timestamp>*

Sets the ending time of test execution when creating reports.

--nostatusrc Sets the return code to zero regardless of failures in test cases. Error codes are returned normally.

--processemptysuite

Processes an output file even if the top level test suite is empty.

-E, --escape *<what:with>*

Escapes characters that are problematic in the console.

-A, --argumentfile *<path>*

A text file to read more arguments from.

-h, --help Prints usage instructions.

--version Prints the version information.

参考资料

https://code.google.com/p/robotframework/wiki/Publications