Table of Contents

简介 1.1

Chapter 1 简明使用手册 1.2

1-1 测试数据, Test Data 1.2.1

1-2 测试用例, Test Case 1.2.2

1-3 测试模板, Test Template 1.2.3

1-4 测试套件, Test Suite 1.2.4

1-5 测试库, Test Library 1.2.5

1-6 变量, Variable 1.2.6

1-6-1 创建变量, Creating Variable 1.2.7

1-6-2 变量文件, Vairable Files 1.2.8

1-6-3 内建变量, Build-in Variables 1.2.9

1-6-4 变量的属性和作用范围 1.2.10

1-6-5 变量的高级特性 1.2.11

1-7 用户关键字, User Keyword 1.2.12

1-7-1 用户关键字参数 1.2.13

1-7-2 变量嵌入到关键字名中 1.2.14

1-7-3 用户关键字返回值 1.2.15

1-7-4 用户关键字Teardown 1.2.16

1-8 资源文件, Resource File 1.2.17

Chapter 2 高级特性 1.3

Chapter 3 实现测试库 1.4

3-1 测试库 1.4.1

3-2 创建测试库:类或模块 1.4.2

3-2-1 测试库名 1.4.3

3-2-2 测试库的参数 1.4.4

3-2-3 测试库的作用范围 1.4.5

3-2-4 测试库版本 1.4.6

3-2-5 测试库的文档说明 1.4.7

3-2-6 测试库工作为Lisnter 1.4.8

3-3 静态API 1.4.9

3-3-1 关键字的名字 1.4.10
3-3-2 关键字的标签 1.4.11
3-3-3 关键字的参数 1.4.12
3-3-4 关键字的参数缺省值 1.4.13
3-3-5 关键字的可变长参数 1.4.14
3-3-6 关键字的自由参数 1.4.15
3-3-7 关键字的参数类型 1.4.16
3-3-8 使用修饰器 1.4.17
3-4 和RF框架通信 1.4.18
3-4-1 停止和继续测试执行 1.4.19
3-4-2 日志消息 1.4.20
3-4-3 Logging编程API 1.4.21
3-4-4 测试库初始化阶段Logging 1.4.22
3-4-5 返回值 1.4.23
3-4-6 线程间通信 1.4.24
3-5 测试库的发布 1.4.25
3-6 动态API 1.4.26
3-6-1 获取关键字名 1.4.27
3-6-2 关键字的运行 1.4.28
3-6-3 获取关键字参数 1.4.29
3-6-4 获取关键字的文档信息 1.4.30
3-6-5 命名参数语法 1.4.31
3-6-6 自由参数语法 1.4.32
3-6-7 小结 1.4.33
3-7 混合型API 1.4.34
3-8 扩展测试库 1.4.35
Chapter 4 常用测试库 1.5
4-1 RF内部模块和内建测试库 1.5.1
4-1-1 BuiltIn库 1.5.2
4-1-2 Collections库 1.5.3
4-1-3 DateTime库 1.5.4
4-1-4 Dialogs库 1.5.5
4-1-5 OperatingSystem库 1.5.6
4-1-6 Process库 1.5.7

4-1-7 Screenshot库 1.5.8
4-1-8 String库 1.5.9
4-1-9 Telnet库 1.5.10
4-1-10 XML库 1.5.11
4-2 第3方测试库 1.5.12
4-2-1 Web自动化 1.5.13
4-2-2 GUI自动化 1.5.14
4-2-3 DB自动化 1.5.15
4-2-4 接口自动化 1.5.16
4-2-5 移动端自动化 1.5.17
4-2-6 敏捷自动化 1.5.18
Chapter 5 框架分析 1.6
5-1 Tips 1.6.1
5-1-1 模块搜索路径, 包(Package) 和 __init__.py 1.6.2
5-1-2 修饰器 1.6.3
5-1-3 迭代器 1.6.4
5-1-4 生成器 1.6.5
5-1-5 包装和授权 1.6.6
5-1-6 描述符 1.6.7
5-1-7 __slots__类属性 1.6.8
5-2 框架结构 1.6.9
5-3 主流程 1.6.10
Chapter 6 扩展框架 1.7
Chapter 7 工具 1.8

序

最近在总结以往的工作: Robot Framework作为在诺西工作时期接触到的框架，其不仅仅是自己最早接触到的大型测试框架，同时还引导自己进入了Python世界。 感情深厚，所以第一个整理工作就从此开始。

网上简单搜索了一下Robot Framework的资料，英文资料居多，中文资料多为用户手册的翻译，并且翻译的内容篇幅不大。 也找到一篇代码阅读解析，不过是比较老版本的代码，看了一些出入颇大。 再次阅读Robot Framework框架过程中，突发奇想，觉得将学习成果整理为文档，定是一件极好的事情。 如果自己的整理能成为填补空白的中文资料，也不枉一份极有意义的工作。

这应该算是自己第一本开源书的写作，大致思路如下:
介绍框架，整理一遍全面但又不失简洁的用户手册；
然后介绍一些进阶的框架使用帮助，主要为测试库的扩展；
最后对一些重要的流程，进行代码的阅读分析。

希望本文既能帮助到他人，也为自己留下重要的学习笔记。

虞科敏
2016/7

---

gitbook.com只会更新第1章节，即简明使用手册。
完整**pdf**版本，将会放在github.com

第1章 简明使用手册

作者: 虞科敏

本节内容可以理解为官方文档《用户手册》的摘录和笔记，主要包括重要的语法，使用方法，以便为后续部分进行重要的测试情景的代码分析做好框架知识准备。

后续章节，请在本开源书的完整版中获得

- 1-1 测试元素, Test Data
- 1-2 测试用例, Test Case
- 1-3 测试模板, Test Template
- 1-4 测试套件, Test Suite
- 1-5 测试库, Test Library
- 1-6 变量, Variable
- 1-6-1 创建变量, Creating Variable
- 1-6-2 变量文件, Vairable Files
- 1-6-3 内建变量, Build-in Variables
- 1-6-4 变量的属性和作用范围
- 1-6-5 变量的高级特性
- 1-7 用户关键字, User Keyword
- 1-7-1 用户关键字参数
- 1-7-2 变量嵌入到关键字名中
- 1-7-3 用户关键字返回值
- 1-7-4 用户关键字Teardown
- 1-8 资源文件, Resource File

测试数据, Test Data

作者: 虞科敏

文件和目录

1. 测试用例文件, Test Case File
   测试用例创建在用例文件中.
   用例文件会自动创建一个包含文件中所定义用例的测试套件, TestSuite.
2. 目录
   包含多个测试文件的目录，形成一个更高一级的测试套件。
   Suite目录拥有从测试文件创建的套件，将它们作为目录Suite的子suite。
   suite目录还可以包含其他的suite目录，这种层级结构可以递归嵌套。
   suite目录可以拥有一个特别的__init__文件。
3. 特别的测试文件 Test Libraries, 包含低级别的关键字
   Resource File, 包含变量Variables，高级别的用户自定义关键字
   Variable File, 提供资源文件外更灵活的创建变量的手段

支持的文件格式

1. HTML
2. TSV
3. Plain TEXT
4. reStructuredText

以Plain Text举例，其他格式请参考用户手册

空格分隔

```
*** Settings ***
Library       OperatingSystem

*** Variables ***
${MESSAGE}    Hello, world!

*** Test Cases ***
My Test
    [Documentation]    Example test
    Log    ${MESSAGE}
    My Keyword    /tmp

Another Test
    Should Be Equal    ${MESSAGE}    Hello, world!

*** Keywords ***
My Keyword
    [Arguments]    ${path}
    Directory Should Exist    ${path}
```

管线和空格分隔

```
| *Setting*  |     *Value*     |
| Library    | OperatingSystem |

| *Variable* |     *Value*     |
| ${MESSAGE} | Hello, world!   |

| *Test Case*  | *Action*        | *Argument*   |
| My Test      | [Documentation] | Example test |
|              | Log             | ${MESSAGE}   |
|              | My Keyword      | /tmp         |
| Another Test | Should Be Equal | ${MESSAGE}   | Hello, world!

| *Keyword*  |
| My Keyword | [Arguments] | ${path}
|            | Directory Should Exist | ${path}
```

数据表格

共计4种表格，以表格中的第一个cell在标识： Settings, Variables, Test Cases, Keywords.
大小写敏感； 单数形式也可以接受，比如Setting, Variable, Test Case, Keyword.

1. Settings
   1) 引入test libraries, resource files, variable files.
   2) 定义suite和case的元数据metadata

2. Variables
   定义变量，可被其他测试数据使用

3. Test Cases
   使用可用的关键字创建测试用例

4. Keywords
   使用已有的低级关键字创建更高级的用户关键字

测试数据解析的规则

1. RF框架解析测试数据时，会忽略一些信息, 细节请参考用户手册。
2. 处理空白字符Whitespace, 转义字符Escaping, 细节请参考用户手册。
3. 空cell的技巧：\ 或者 ${EMPTY}
4. 空格的技巧：\ 或者 ${SPACE}
5. 多行技巧： ...

Tips: 会被忽略的信息

```
在第一个cell中表名字非法的表格
第一行中第一个cell后的其他内容
第一个表前面的所有内容（如果允许，表之间的内容也会被忽略）
所有的空行（空行一般用来提高可读性）
每行末尾的空cell（除非被转义）
所有单个的\（当不用做转义时）
当#时一个cell中的第一个字符，#后面的所有字符（#可被用来作为注释之用）
在HTML或reStructuredText中的所有格式信息
```

测试用例, Test Case

作者: 虞科敏

基础语法

测试用例在test case table中创建，使用各种合法可用的关键字。
关键字的来源： 从test libraries或者resource file中导入； 在用例文件自身的keyword table中创建。

样例

```
*** Test Cases ***
Valid Login
    Open Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open

Setting Variables
    Do Something    first argument    second argument
    ${value} =    Get Some Value
    Should Be Equal    ${value}    Expected value
```

用例表中的Settings

Force Tags, Default Tags
Test Setup, Test Teardown
Test Template
Test Timeout

样例

```
*** Test Cases ***
Test With Settings
    [Documentation]    Another dummy test
    [Tags]    dummy    owner-johndoe
    Log    Hello, world!
```

参数

关键字的参数，可以类比Python参数来理解，实际上其实现即为Python语言实现，难怪行为也如此相似。 在关键字的文档中也会用类似语法说明出来。 使用Python的同学理解RF中关键字的参数，对比Python的相关行为很容易理解。 将关键字参数和python中的参数进行对比如下。

必选参数, Mandatory arguments

=> 类比python的位置参数，positional arg

```
*** Test Cases ***
Example
    Create Directory    ${TEMPDIR}/stuff
    Copy File    ${CURDIR}/file.txt    ${TEMPDIR}/stuff
    No Operation
```

参数的缺省值, Default values

=> 类比python的默认参数 (arg=value)

```
*** Test Cases ***
Example
   Create File    ${TEMPDIR}/empty.txt
    Create File    ${TEMPDIR}/utf-8.txt         Hyvä esimerkki
    Create File    ${TEMPDIR}/iso-8859-1.txt    Hyvä esimerkki    ISO-8859-1
```

可变长参数, Variable number of arguments

=> 类比python的非关键字可变长参数(元组)（*arg)

```
*** Test Cases ***
Example
    Remove Files    ${TEMPDIR}/f1.txt    ${TEMPDIR}/f2.txt    ${TEMPDIR}/f3.txt
    @{paths} =    Join Paths    ${TEMPDIR}    f1.txt    f2.txt    f3.txt    f4.txt
```

命名参数, Named arguments

=> 类比python的关键字参数，可以直接指定参数名来进行赋值

样例*1* 正常使用举例

```
*** Settings ***
Library    Telnet    prompt=$    default_log_level=DEBUG

*** Test Cases ***
Example
    Open connection    10.0.0.42    port=${PORT}    alias=example
    List files    options=-lh
    List files    path=/tmp    options=-l

*** Keywords ***
List files
    [Arguments]    ${path}=.    ${options}=
    List files    options=-lh
    Execute command    ls ${options} ${path}
```

样例*2* 特殊使用举例

```
*** Test Cases ***
Example
    Run Program    shell=True    # This will not come as a named argument to Run Proce
ss!
    Log    foo\=quux  # this will get the value "foo=quux"

*** Keywords ***
Run Program
    [Arguments]    @{args}
    Run Process    program.py    @{args}    # Named arguments are not recognized from
inside @{args}!
```

自由参数, Free keyword arguments

=> 类比python的关键字可变长参数(字典) (**arg)，其会收集所有使用name=value语法，并且不匹配任何其他类型参数的内容作为kwargs

样例*1*
Run Process关键字(来自Process库), signature: command, *arguments, **configuration
command: 将执行的命令*, its arguments as variable number of arguments (* arguments: 可变长参数
**configuration: 自由参数

```
*** Test Cases ***
Using Kwargs
    Run Process    program.py    arg1    arg2    cwd=/home/user
    Run Process    program.py    argument    shell=True    env=${ENVIRON}
```

样例2
对Run Process关键字的包裹Run Program

```
*** Test Cases ***
Using Kwargs
    Run Program    arg1    arg2    cwd=/home/user
    Run Program    argument    shell=True    env=${ENVIRON}

*** Keywords ***
Run Program
    [Arguments]    @{arguments}    &{configuration}
    Run Process    program.py    @{arguments}    &{configuration}
```

失败

直接使用fails关键字，可以让用例失败: 然后执行test teardown，然后执行下一个用例
使用continuable failures关键字，如果不希望停止用例执行

失败消息: 在fails关键字中可以直接指定失败消息内容，有些关键字也可以指定失败消息内容

样例

```
*** Test Cases ***
Normal Error
    Fail    This is a rather boring example...

HTML Error
    ${number} =    Get Number
    Should Be Equal    ${number}    42    *HTML* Number is not my <b>MAGIC</b> number.
```

用例名字和文档串

直接阅读样例理解即可，不作过多说明

样例

```
*** Test Cases ***
Simple
    [Documentation]    Simple documentation
    No Operation

Formatting
    [Documentation]    *This is bold*, _this is italic_  and here is a link: http://ro
botframework.org
    No Operation

Variables
    [Documentation]    Executed at ${HOST} by ${USER}
    No Operation

Splitting
    [Documentation]    This documentation    is split    into multiple columns
    No Operation

Many lines
    [Documentation]    Here we have
    ...                an automatic newline
    No Operation
```

标签

标签用来分类用例，方便有选择的执行和查看。
标签会出现在报告，日志，统计信息中，提供用例的元数据信息。
使用标签，include和exclude可以容易选择用例，也方便指定关键用例Critical.

Setting Table之强制标签, Force Tags

在测试文件中，本文件中的所有用例都将加上此标签
在suite的__init__文件中，所有的子suite中的用例都将加上此标签

Setting Table之缺省标签, Default Tags

没有Tags定义的用例将加上此标签
不支持suite的__init__文件

Test Case Table之[Tags]

指定本测试用例的标签
同时，它会覆盖缺省标签

命令行中设定标签

--settag

BuiltIn关键字可以在执行过程中动态操纵标签

Set Tags
Remove Tags
Fail
Pass Execution

标签定义时候是自由无限制的； 但是标签有一个normalized过程，将转为全小写，并且移除空格，这个在阅读框架代码是可以看到。
标签可以使用变量来定义。

样例

```
*** Settings ***
Force Tags      req-42
Default Tags    owner-john    smoke

*** Variables ***
${HOST}         10.0.1.42

*** Test Cases ***
No own tags
    [Documentation]    This test has tags owner-john, smoke and req-42.
    No Operation

With own tags
    [Documentation]    This test has tags not_ready, owner-mrx and req-42.
    [Tags]    owner-mrx    not_ready
    No Operation

Own tags with variables
    [Documentation]    This test has tags host-10.0.1.42 and req-42.
    [Tags]    host-${HOST}
    No Operation

Empty own tags
    [Documentation]    This test has only tag req-42.
    [Tags]
    No Operation

Set Tags and Remove Tags Keywords
    [Documentation]    This test has tags mytag and owner-john.
    Set Tags    mytag
    Remove Tags    smoke    req-*
```

保留标签: RF的预留标签都使用**"robot-"**作为前缀，用户不要自己定义以此为前缀的标签。
当前，只有在正常停止测试执行时， "robot-exit"标签会被自动添加到测试用例。

测试固定件Fixture: setup和teardown

Fixture: setup和teardown语义等同于其他测试框架中相似的部件，如果不清楚请自己阅读用户手册。

Fixture: setup和teardown中一般使用单个的关键字。 如果需要关注多个任务，也请创建高级的用户关键字来进行。 另外， 你也可以使用BuiltIn中**Run Keywords**关键字。

teardown的2点特别说明：
即时测试失败， teardown也会被执行， 这里是进行必行执行的clean-up活动的最佳地方
teardown中的关键字，就算其中某些关键字执行失败，其他关键字也会被执行。 这个行为也可以通过**Continue on faliure**系列关键字(比如，Run Keyword And Ignore Error, Run Keyword And Expect Error)来得到。

最方便指定setup和teardown的地方是测试文件的Setting Table中使用Test Setup和Test Teardown进行设置。 T 单独的测试用例， 也可以有它自己的[Setup]和[Teardown]。

样例

```
*** Settings ***
**Test Setup**       Open Application    App A
**Test Teardown**    Close Application

*** Test Cases ***
Default values
    [Documentation]    Setup and teardown from setting table
    Do Something

Overridden setup
    [Documentation]    Own setup, teardown from setting table
    **[Setup]**    Open Application    App B
    Do Something

No teardown
    [Documentation]    Default setup, no teardown at all
    Do Something
    **[Teardown]**

No teardown 2
    [Documentation]    Setup and teardown can be disabled also with special value NONE

    Do Something
    **[Teardown]**    **NONE**

Using variables
    [Documentation]    Setup and teardown specified using variables
    **[Setup]**    **${SETUP}**
    Do Something
    **[Teardown]**    **${TEARDOWN}**
```

测试模板, Test Template

作者: 虞科敏

测试模板，关键字驱动 => 数据驱动

关键字驱动, 用例主体由若干关键字+参数构成
vs.
数据驱动, 用例主体只由Template关键字的参数构成

用途举例:
对每个测试用例，或者一个测试文件中的所有用例，重复执行同一个关键字多次（使用不同数据）
也可以只针对测试用例，或者每个测试文件只执行一次

模板关键字可以接受普通的位置参数，命名参数
关键字名中可以使用参数
不可以使用变量定义模板关键字

样例*1*
[Template]会覆盖Setting Table中的template设置
如果[Template]为空值，意味着没有模板

```
*** Test Cases **
Normal test case
    Example keyword    first argument    second argument

Templated test case
    [Template]    Example keyword
    first argument    second argument
```

样例*2*
对于多行数据，模板关键字会逐行调用执行，一次一行 如果其中有些失败，其他也会执行。
对于普通用例的continue on failure模式，对于模板关键字是缺省行为。

```
*** Settings ***
Test Template    Example keyword

*** Test Cases ***
Templated test case
    first round 1     first round 2
    second round 1    second round 2
    third round 1     third round 2
```

样例*3*

模板关键字支持嵌入参数的语法

关键字名字就作为参数的持有者，在实际执行中这些参数会被模板关键字解析出实际的参数，传递给低级的底层关键字作为参数

```
*** Test Cases ***
Normal test case with embedded arguments
    The result of 1 + 1 should be 2
    The result of 1 + 2 should be 3

Template with embedded arguments
    [Template]    The result of ${calculation} should be ${expected}
    1 + 1    2
    1 + 2    3

*** Keywords ***
The result of ${calculation} should be ${expected}
    ${result} =    Calculate    ${calculation}
    Should Be Equal    ${result}     ${expected}
```

样例*4*

模板关键字名字中的参数个数必须匹配它将使用的参数数量

参数名不需要匹配原始关键字的参数

```
*** Test Cases ***
Different argument names
    [Template]    The result of ${foo} should be ${bar}
    1 + 1    2
    1 + 2    3

Only some arguments
    [Template]    The result of ${calculation} should be 3
    1 + 2
    4 - 1

New arguments
    [Template]    The ${meaning} of ${life} should be 42
    result    21 * 2
```

样例*5* 带有for循环的模板关键字

```
*** Test Cases ***
Template and for
    [Template]    Example keyword
    :FOR    ${item}    IN    @{ITEMS}
    \    ${item}    2nd arg
    :FOR    ${index}    IN RANGE    42
    \    1st arg    ${index}
```

不同的测试用例风格

- 关键字驱动
  - 描述工作流
  - 若干关键字和他们必要的参数
- 数据驱动
  - 针对相同工作流，执行不同的输入数据
  - 只使用一个高级的用户关键字，其中定义了工作流，然后使用不同的输入和输出数据测试相同的场景
  - 每个测试中可以重复同一个关键字，但是test template功能只允许定义以此被使用的关键字
- 行为驱动：
  - 描述工作流
  - Acceptance Test Driven Development, ATDD
  - Specification by Example
  - BDD's Given-When-Then
  - And or But，如果测试步骤中操作较多
  - 支持嵌入数据到关键字名

样例*1*

```
*** Settings ***
Test Template    Login with invalid credentials should fail

*** Test Cases ***                USERNAME         PASSWORD
Invalid User Name                 invalid          ${VALID PASSWORD}
Invalid Password                  ${VALID USER}    invalid
Invalid User Name and Password    invalid          invalid
Empty User Name                   ${EMPTY}         ${VALID PASSWORD}
Empty Password                    ${VALID USER}    ${EMPTY}
Empty User Name and Password      ${EMPTY}         ${EMPTY}

*样例2*
*** Test Cases ***
Invalid Password
    [Template]    Login with invalid credentials should fail
    invalid          ${VALID PASSWORD}
    ${VALID USER}    invalid
    invalid          whatever
    ${EMPTY}         ${VALID PASSWORD}
    ${VALID USER}    ${EMPTY}
    ${EMPTY}         ${EMPTY}
```

样例1和样例2都是数据驱动的test template样例。
样例1有命令列，方便阅读理解; test template在setting table中定义; 每行有名字也方便查看结果（如果行数不是太多的话）
样例2在一个用例中完成所有的事情

样例*3*
搜索关键字的时候， 如果full name没有搜索到， Given-When-Then-And-But等前缀会被忽略

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

测试套件, Test Suite

作者: 虞科敏

测试用例 => Test File => 以目录进行组织

Test Case Files

一个文件中用例数建议 <10; 如果用例数过多建议考虑数据驱动

用例文件会自动创建一个包含文件中所定义用例的测试套件, TestSuite

Test Suite Dirctories

test case files被组织为目录， 这些目录形成更高级的suite。

由目录创建的test suite不能有任何直接的测试用例(Test Cases), 但可以包含其他带有测试用例的suite。

这些目录又可以被放进其他目录，形成更高级的suite。 包含的层级深度有没有特别的限制。

当一个测试目录被执行，它包含的文件和目录按以下方式递归处理：

1. 以"."和"_"开始的文件和目录被跳过
2. 目录名为CVS(大小写敏感)被跳过
3. 扩展名非法（有效扩展名: .html, .xhtml, .htm, .tsv, .txt, .rst, or .rest）的文件被跳过
4. 其他文件和目录被处理

   Tips: 如果文件或目录不包含任何测试用例Test Case，会被静默地跳过; 如果需要产生警告，可以在命令行使用 --warnonskippedfiles

初始化文件

目录创建的Suite可以拥有和Test Case File创建的suite相似的设置，这些设置信息被放在Suite初始化文件中。
格式: __init__.ext （Pythoner们是不是觉得很熟悉，RF借用了python的命名方式，不过做的事情和python的__init__.py有些不同)
合法的.ext:

- .html, .htm and .xhtml for HTML
- .tsv for TSV
- .txt and special .robot for plain text
- .rst and .rest for reStructuredText

除了不能拥有Test Case Table，Setting内容有一定限制外， __init__.ext和test case file的结构和语法都保持一致.

在__init__中创建或引入的变量Variables和关键字Keywords，对于其子suite并不可用;如果你需要共享这些变量和关键字， 你需要考虑使用resource file, 它们可以被__init__和test case file同时使用。

__init__.ext的主要用途在于指定类似在test case file中进行的suite相关设置； 另外一些case相关设置也可能在这里进行:

- Documentation, Metadata, Suite Setup, Suite Teardown: 参考test case file部分
- Force Tags: 无条件的指定本目录下的所有用例（直接或递归地）标签
- Test Setup, Test Teardown, Test Timeout: 指定本目录下的所有用例（直接或递归地），可以被更低级别层面的定义覆盖
- Default Tags, Test Template: 不支持！！！！！！！！

样例

```
*** Settings ***
Documentation    Example suite
Suite Setup      Do Something    ${MESSAGE}
Force Tags       example
Library          SomeLibrary

*** Variables ***
${MESSAGE}       Hello, world!

*** Keywords ***
Do Something
    [Arguments]    ${args}
    Some Keyword    ${arg}
    Another Keyword
```

定制化Test Suite

名字

Suite Name来自测试文件名或者目录名。 参考举例来理解，不复杂:
some_tests.html => suite name: Some Tests
My_test_directory => suite name: My test directory

可以添加前缀(pre__, 2个下划线)来指定执行顺序，但是前缀不会被包含在suite name中。
01__some*tests.txt* => *suite name: Some Tests*
*02*_more_tests.txt => suite name: More Tests
Some Tests在More Tests前被执行

命令行选项--name，可以覆盖top-level suite的name定义

文档

可以在Setting Table里用**Documentation**设置: 可以在test case file中，或者高级suite的__init__文件中

命令行选项--doc，可以覆盖top-level suite的documentation定义

样例

```
*** Settings ***
Documentation    An example test suite documentation with *some* _formatting_.
...              See test documentation for more documentation examples.
```

元数据

其他元数据可以在Setting Table里用**Metadata**设置: 以此方式定义的元数据会显示在测试报告和日志中。

命令行选项--metadata，可以覆盖top-level suite的metadata定义

```
*** Settings ***
Metadata    Version        2.0
Metadata    More Info      For more information about *Robot Framework* see http://rob
otframework.org
Metadata    Executed At    ${HOST}
```

Suite Setup和Suite Teardown

测试固定件Fixture: setup和teardown的语义，和其他测试框架没有太多不同。 很多情况也和Tese Case的fixture差不多，可以参考Test Case的相关章节。
也是推荐只使用一个关键字

如果suite setup失败， suite下的所有测试用例和它的子suite都会被标记为失败，并且不会再被执行。 所以suite setup适合用于检查必须满足的前提条件。

suite teardown和test case teardown很类似，也满足continue on failure模式。 如果suite teardown失败, suite下的所有用例被标记为失败, 不管之前这些用例的执行结果如何.

作为setup或者teardown的被执行的关键字名可以为变量Varialbe。

```
小技巧： 可以通过命令行来指定变量，来实现不同环境执行不同的setup和teardown
```

测试库, Test Library

作者: 虞科敏

测试库包含底层关键字，库关键字Library Keywords.

导入库

- 使用Setting Table中的Library Setting导入
  库名字是大小写和空格敏感的
  如果库存在一个包Package中，应提高包含包的全名Full Name
  库可能需要参数进行初始化
  库导入时支持参数缺省值，可变长参数，命名参数等语法，这和关键字的参数语法很像
  库名和参数都可以使用变量

样例

可以在Resource File, Test Case File, Suite __init__ File中导入库，库中的关键字在导入库的文件中可用

```
*** Settings ***
Library    OperatingSystem
Library    my.package.TestLibrary
Library    MyLibrary    arg1    arg2
Library    ${LIBRARY}
```

- 使用关键字**Import Library**

导入的方法和Library Setting很相似
导入的关键字在Import Library关键字被使用的suite中可用

样例

```
*** Test Cases ***
Example
    Do Something
    Import Library    MyLibrary    arg1    arg2
    KW From MyLibrary
```

指定导入的库

使用库的名字

RF框架会在[module search path][1]中根据库名字找寻实现库的类或者模块

理解Module Search Path对于理解库的查找非常重要，这个实际就是使用Python的解释器进行模块搜索的概念，了解Python的同学不难理解。

- Python解释器的安装目录，三方库的安装目录，自动就在搜索路径中。 在Python的搜索路径中的模块，不需要经过其他配置，就能被导入
- 更多的路径信息，可以通过PYTHONPATH, JYTHONPATH, IRONPYTHONPATH环境变量进行设置
- 命令行选项 --pythonpath (-P) 也可以添加更多的地址到搜索路径中
- 通过编程控制sys.path属性
- 如果使用Jython，还可以使用Jython的模块搜索路径或者Java Classpath

样例

```
--pythonpath libs
--pythonpath /opt/testlibs:mylibs.zip:yourlibs
--pythonpath mylib.jar --pythonpath lib/STAR.jar --escape star:STAR
```

[1] 路径搜索和搜索路径
模块的导入需要一个叫做"路径搜索"的过程。 即在文件系统"预定义区域"中查找 mymodule.py 文件(如果你导入 mymodule 的话)。 这些预定义区域只不过是你的 Python 搜索路径的集合。
路径搜索和搜索路径是两个不同的概念, 前者是指查找某个文件的操作, 后者是去查找一组目录。
默认搜索路径是在编译或是安装时指定的。 它可以在一个或两个地方修改。
一个是启动Python的shell或命令行的PYTHONPATH环境变量。 该变量的内容是一组用冒号分割的目录路径。 如果你想让解释器使用这个变量, 那么请确保在启动解释器或执行Python脚本前设置或修改了该变量。
解释器启动之后, 也可以访问这个搜索路径, 它会被保存在 sys 模块的 sys.path 变量里。不过它已经不是冒号分割的字符串, 而是包含每个独立路径的列表。 这只是个列表, 所以我们可以随时随地对它进行修改。 如果你知道你需要导入的模块是什么,而它的路径不在搜索路径里, 那么只需要调用列表的 append() 方法即可, 就像这样: sys.path.append('/home/wesc/py/lib'). 修改完成后, 你就可以加载自己的模块了。 只要这个列表中的某个目录包含这个文件, 它就会被正确导入。

使用库的物理路径

另一种思路是使用库在系统中的路径来指定库
路径以相对于当前test data file所在目录的相对地址来给出，这和Resource Files, Vairable Files很相似
如果库是文件，那么.ext，文件的扩展名必须被包括在路径中，比如Java的.class或.java，Python的.py等
如果库是目录，那么路径应该以"/"结尾

样例

局限: 实现为Python类的库，Python类所在的module名必须和类名相同

```
*** Settings ***
Library    PythonLibrary.py
Library    /absolute/path/JavaLibrary.java
Library    relative/path/PythonDirLib/    possible    arguments
Library    ${RESOURCES}/Example.class
```

库别名

有很多场景，我们使用客户定制化的库名（库别名）会更加方便，或不得不使用定制化的库名:

- 使用不同的参数导入同一个库多次
- 库名太长，比如很多包名很长
- 使用变量在不同的环境中导入不同的包， 但你想使用相同的名字进行引用
- 库名很糟糕，很容易误导人

Library Setting或者关键字Import Keyword都支持库别名

样例*1*
WITH NAME 大小写敏感
指定的名字会在Log中显示
使用关键字全名时，必须使用库别名

```
*** Settings ***
Library    com.company.TestLib    WITH NAME    TestLib
Library    ${LIBRARY}             WITH NAME    MyName
```

样例*2*
使用不同的参数引入同一个库多次

```
*** Settings ***
Library    SomeLibrary    localhost        1234    WITH NAME    LocalLib
Library    SomeLibrary    server.domain    8080    WITH NAME    RemoteLib

*** Test Cases ***
My Test
    LocalLib.Some Keyword     some arg       second arg
    RemoteLib.Some Keyword    another arg    whatever
    LocalLib.Another Keyword
```

标准库, Standard Library

RF框架提供如下标准库。

```
BuiltIn
Collections
DateTime
Dialogs
OperatingSystem
Process
Screenshot
String
Telnet
XML
```

另外， 除了普通标准库，还存在另一种完全不同的远程库, Remote Library.
远程库不提供任何关键字， 它在RF框架和远程的库实现之间以代理的形式工作.
远程的库实现可以运行在和RF框架不同的其他机器上，甚至可以被并非RF支持的编程语言实现。

外部库, External Library

任何非标准库的测试库，称为外部库。
RF开源社区已经一些常用的测试库，没有打包到框架核心之中. 更多公共外部库，可以访问 http://robotframework.org

```
Selenium2Library
SwingLibrary
```

变量, Variable

作者: 虞科敏

很多测试数据场景会使用变量。 最常见的，比如 用作关键字的参数， Setting的值等。
普通关键字名中不能使用变量，但内建关键字Run Keyword可以用来达到相似的效果。

3种变量类型

- 标量scalars - ${SCALAR}
- 列表lists - @{LIST}
- 字典dictionaries - &{DICT}

另外，环境变量可以直接使用语法 %{ENV_VAR} 进行访问

变量的使用场景举例： 在测试数据中，某字符串经常改变： 使用变量你只需要在一个地方进行改变
创建系统独立或者操作系统独立的测试数据： 变量可以使用命令行进行指定
关键字中需要对象参数，而不仅仅是字符串
不同的库，不同的关键字需要通信： 返回值可以付给变量，然后传给另一个关键字
测试数据的值太长，太复杂

如果测试数据使用了一个不存在的变量，关键字执行会失败
如果不希望获取变量的值，可以进行转义 ${NAME}

变量类型

变量和关键字一样，大小写敏感
空格和下划线会被忽略

建议:
全局变量使用大写，如${PATH} or ${TWO WORDS}
只在某用例或关键字中使用的变量使用小写，如 ${my var} or ${myVar}
最重要的，大小写的风格要保持一致
变量名使用字母表字符(a-z, A-Z)，数字(0-9), 下划线(_)和空格，这也是扩展变量的所明确要求的

标量, scalars - ${SCALAR}

不只是字符串，任何对象实例都可以付给标量变量，比如list
当标量变量是cell中的唯一值时， 变量变量会被它所有的值所代替
当标量变量在cell中和其他元素一起存在时，变量变量会被转换为unicode，然后和其他元素进行合并

```
Tips: 当参数使用命名参数语法被传递给关键字(argname=${var})，标量变量值被用作as-is值， 不会进行转换
，
```

样例*1*
假设 ${GREET} = Hello and ${NAME} = world， 以下两个关键字的效果是一样的

```
*** Test Cases ***
Constants
    Log    Hello
    Log    Hello, world!!

Variables
    Log    ${GREET}
    Log    ${GREET}, ${NAME}!!
```

样例*2* 假设 ${STR} = Hello, world!
${OBJ} = new MyObj()

```
public class MyObj {
    public String toString() {
        return "Hi, tellus!";
    }
}
```

KW1得到字符串 Hello, world!
KW2得到MyObj对象
KW3得到字符串 I said "Hello, world!"
KW4得到字符串 You said "Hi, tellus!"

```
*** Test Cases ***
Objects
    KW 1    ${STR}
    KW 2    ${OBJ}
    KW 3    I said "${STR}"
    KW 4    You said "${OBJ}"
```

列表, lists - @{LIST}

标量变量${EXAMPLE}， 值以"as-is"的方式被直接使用
列表变量@{EXAMPLE}, 单个的list item被分别传递给关键字

样例1

假设 @{USER} = ['robot', 'secret']，关键字Constants和List Variable的效果是一样的

```
*** Test Cases ***
Constants
    Login    robot    secret

List Variable
    Login    @{USER}
```

RF框架存储变量在一个内部存储中，允许使用这些变量作为scalars, lists 或者 dictionaries。

和其他数据一起使用lists

lists可以和其他数据一起，包括和其他list参数一起使用。

样例

```
*** Test Cases ***
Example
    Keyword    @{LIST}    more    args
    Keyword    ${SCALAR}    @{LIST}    constant
    Keyword    @{LIST}    @{ANOTHER}    @{ONE MORE}
```

访问list中的item

@{NAME}[index]访问列表中的单个元素
此处的语法和Python中的list访问是相似的，比如从0开始，比如-1访问倒数第1个元素

样例

```
*** Test Cases ***
List Variable Item
    Login    @{USER}
    Title Should Be    Welcome @{USER}[0]!

Negative Index
    Log    @{LIST}[-1]

Index As Variable
    Log    @{LIST}[${INDEX}]
```

list在setting中的使用

导入libraries和variable files： list变量不能为库名和变量文件名，但可以用作参数
setups和teardowns： list变量不能为关键字名，但可以用作参数
tags： 可以自由使用

```
*** Settings ***
Library         ExampleLibrary      @{LIB ARGS}    # This works
Library         ${LIBRARY}          @{LIB ARGS}    # This works
Library         @{NAME AND ARGS}                   # This does not work
Suite Setup     Some Keyword        @{KW ARGS}     # This works
Suite Setup     ${KEYWORD}          @{KW ARGS}     # This works
Suite Setup     @{KEYWORD}                         # This does not work
Default Tags    @{TAGS}                            # This works
```

字典, dictionaries - &{DICT}

字典变量&{EXAMPLE}, 单个的dict item被分别以命名参数形式()传递给关键字

样例
假设&{USER} = {'name': 'robot', 'password': 'secret'}, 关键字Constants和Dict Variable是等价的

```
*** Test Cases ***
Constants
    Login    name=robot    password=secret

Dict Variable
    Login    &{USER}
```

和其他数据一起使用dict

dict可以和其他数据一起，包括和其他dict参数一起使用。

样例 语法要求，位置参数必须在命名参数之前，所以dict后面只能跟dict或者命名参数

```
*** Test Cases ***
Example
    Keyword    &{DICT}    named=arg
    Keyword    positional    @{LIST}    &{DICT}
    Keyword    &{DICT}    &{ANOTHER}    &{ONE MORE}
```

访问dict中的item

&{NAME}[key]访问字典中的单个元素
此处的语法和Python中的dict访问是相似的
key被认为为字符串，但是非字符串的key也可以被用作变量

样例

```
*** Test Cases ***
Dict Variable Item
    Login    &{USER}
    Title Should Be    Welcome &{USER}[name]!

Variable Key
    Log Many    &{DICT}[${KEY}]    &{DICT}[${42}]
```

dict在setting中的使用

字典变量通常不能用在setting中，除了imports, setups, teardowns，字典可以用作参数

样例

```
*** Settings ***
Library        ExampleLibrary    &{LIB ARGS}
Suite Setup    Some Keyword      &{KW ARGS}     named=arg
```

环境变量, Environment Variables - %{ENV_VAR}

%{ENV_VAR_NAME}直接访问环境变量
环境变量都被认为为字符串
环境变量在测试执行之前，在操作系统中被设置，在执行过程中一直可用

环境变量是全局的，在一个用例中被设置，后续执行的用例都可以使用
对于环境变量的改变，在测试结束后将失效
OperatingSystem.Set Environment Variable
OperatingSystem.Delete Environment Variable

样例

```
*** Test Cases ***
Env Variables
    Log    Current user: %{USER}
    Run    %{JAVA_HOME}${/}javac
```

Java系统属性

如果使用Jython，Java系统属性可视为环境变量被使用
如果Java系统属性和环境变量同名，环境变量将被使用

样例

```
*** Test Cases ***
System Properties
    Log    %{user.name} running tests on %{os.name}
```

创建变量, Creating Variables

作者: 虞科敏

有以下途径可以创建变量：

- 通过Vairable Table
- 在Vairable File中定义
- 使用命令行选项
- 来自关键字的返回值
- 使用内建关键字设置

变量表, Variable Table

最常见的创建变量的地方，就是在Test Case File和Resource File中的Variable Table中。
在Variable Table中创建变量有诸多好处： 和其他测试数据在同一个地方，语法也非常简单；
不足在于变量值可能只能为字符串，并且不能动态创建。

```
Tips: 如果需要克服此问题，可以考虑Variable File
```

创建scalar变量

样例*1*
如果第2列为空，那么空字符串被赋值给变量

```
*** Variables ***
${NAME}         Robot Framework
${VERSION}      2.0
${ROBOT}        ${NAME} ${VERSION}
${ZERO}
```

样例*2*
也支持中间添加"="的语法，但这不是强制要求的

```
*** Variables ***
${NAME} =       Robot Framework
${VERSION} =    2.0
```

样例*3*
如果值太长，可以分为多行和多列
多行和多列会被框架合并起来
缺省的，合并中间会使用空格(等效于" ".join())，也可以通过在第一个cell中使用SEPARATOR=来改变连接字符

```
*** Variables ***
${EXAMPLE}      This value is joined    together with a space
${MULTILINE}    SEPARATOR=\n    First line
...             Second line     Third line
```

创建**list**变量

样例*1*
值从第2列开始
也支持空值，list作为元素，多行等语法
列表的下标从0开始

```
*** Variables ***
@{NAMES}        Matti       Teppo
@{NAMES2}       @{NAMES}    Seppo
@{NOTHING}
@{MANY}         one         two      three      four
...             five        six      seven
```

创建**dict**变量

创建dict变量的语法: "name=value" 或者 已有的Dict变量赋值给新变量
同样key的value，后出现的覆盖先出现的
如果key或value中存在"="，需要进行转义"\="

样例
访问元素的语法(使用Python的同学一定不会陌生): ${VAR.key} 或者 &{USER}[name]
&{MANY}[${3}] 不等价于 ${MANY.3}
字典的key是有顺序的，按照其被定义的顺序
字典按照列表的语法进行使用，实际使用的是字典的key集合: @{MANY}变量 == ['first', 'second', 3]

```
*** Variables ***
&{USER 1}       name=Matti    address=xxx         phone=123
&{USER 2}       name=Teppo    address=yyy         phone=456
&{MANY}         first=1       second=${2}         ${3}=third
&{EVEN MORE}    &{MANY}       first=override      empty=
...             =empty        key\=here=value
```

变量文件, Vairable Files

变量文件支持各种变量的创建: 任何对象被指派给变量; 动态创建变量等
变量文件典型使用Python模块来实现
更多关于变量文件的介绍，请参考下一个章节。

命令行, Command Line Option

命令行选项可以设置变量:

- 单个设置变量: --variable (-v)
- 使用Variable File: --variablefile (-V)

  Tips: --variable (-v) 优先于 --variablefile (-V)

通过命令行设置的变量，对于所有测试执行是全局可用的。 它们会覆盖通过Variable Table创建或者导入的Variable File创建的同名变量。

样例*1*
语法： --variable name:value
只能设置scalar变量，值只能为字符串
可以使用--escape对字符进行转义

以下样例的结果如下:
${EXAMPLE} 值: value
${HOST} 和 ${USER} 值分别为: localhost:7272 和 robot
${ESCAPED} 值: "quotes and spaces"

```
--variable EXAMPLE:value
--variable HOST:localhost:7272 --variable USER:robot
--variable ESCAPED:Qquotes_and_spacesQ --escape quot:Q --escape space:_
```

返回值, Return Value from Keyword

关键字返回值可以被赋值给变量。这个变量又可以被传递给其他关键字。 这样实现不同关键字之间的相互通信。
这种变量的设置和使用，和其他方式创建的变量是一样的。只是这种变量的作用域范围只局限在它们被创建的局部作用域。即在一个Test Case里创建的变量，不能在另一个Test Case中被使用。

scalar变量的赋值

样例*1*
"="不是强制要求的。
这种创建变量的方式，在定义用户关键字时也是一样的。

```
*** Test Cases ***
Returning
    ${x} =    Get X    an argument
    Log    We got ${x}!
```

样例*2*
尽管被赋值给scalar变量，但是，如果数据为list-like的，那么你可以将其按照list变量进行使用。
同样地，如果数据为dict-like的，也可以将其按照dict变量进行使用。

```
*** Test Cases ***
Example
    ${list} =    Create List    first    second    third
    Length Should Be    ${list}    3
    Log Many    @{list}
```

list变量的赋值

样例
如果关键字返回值为list或者list-like的值，可以将其赋值给list变量。

```
*** Test Cases ***
Example
    @{list} =    Create List    first    second    third
    Length Should Be    ${list}    3
    Log Many    @{list}
```

因为所有的变量都被存储在同一个名字空间(namespace), 值被赋值给scalar变量或者list变量实际上是没有太大区别。
区别在于:
当创建一个list变量时，RF框架会验证值是否为list或list-like的，存储的值将会是从返回值创建的新的list对象。
当创建一个scalar变量时，RF框架不会进行任何验证，返回的值将会原样被存储起来。

dict变量的赋值

样例
如果关键字返回值为dict或者dict-like的值，可以将其赋值给dict变量。

```
*** Test Cases ***
Example
    &{dict} =    Create Dictionary    first=1    second=${2}    ${3}=third
    Length Should Be    ${dict}    3
    Do Something    &{dict}
    Log    ${dict.first}
```

因为所有的变量都被存储在同一个名字空间(namespace), 值被赋值给scalar变量，之后又按照dict变量来进行使用，这样做也是可以的。
但赋值给dict变量有这些好处:
RF框架会像验证list变量一样，执行dict变量的相应验证。
RF框架会将值转换为特殊的dict: 可排序的; 可以使用语法${dict.first}访问单个元素。

多个变量的同时赋值

如果关键字返回值为list或list-like的对象，可以将单个值赋值给不同的scalar变量，或者scalar变量+list变量。

样例
假设关键字"Get Three"返回[1, 2, 3], 创建的变量和值如下情况:
${a}, ${b}, ${c} 值分别为: 1, 2, 3
${first} 值: 1, @{rest} 值: [2, 3]
@{before} 值: [1, 2], ${last} 值: 3
${begin} 值: 1, @{middle} 值: [2], ${end} 值: 3

```
*** Test Cases ***
Assign Multiple
    ${a}    ${b}    ${c} =    Get Three
    ${first}    @{rest} =    Get Three
    @{before}    ${last} =    Get Three
    ${begin}    @{middle}    ${end} =    Get Three
```

内建变量设置关键字, Set Test/Suite/Global Variable

内建变量设置关键字，可以在用例执行期间动态设置关键字
如果变量已经存在，值将会被覆盖;
如果变量不存在，新变量会被创建

- Set Test Variable
  作用域: 当前测试用例中
- Set Suite Variable
  作用域: 和测试文件中Variable Table或者从Variable File导入等效;
  其他Suite, 包括子Suite, 不可见
- Set Global Variable
  作用域: 设置后的所有用例和Suite都可见;
  和--variable 或 --variablefile等效

  关键字"Set Xxx Variable"直接设置变量到相应的作用域，没有返回值
  关键字"Set Variable"使用返回值设置本地变量

变量文件, Vairable Files

作者: 虞科敏

变量文件提供了强大的变量创建和共享的机制。 它支持各种变量的创建: 任何对象被指派给变量; 动态创建变量等
变量文件强大的功能，是因为典型地，它是使用Python模块(或者Python类，Java类)来实现的

在Variable File创建变量的2种方法

- 直接创建变量

模块的属性，直接成为变量。比如在模块中定义

```
MY_VAR = 'my value'
```

创建${MY_VAR}， 值为'my value'

- 使用特殊函数

特殊的获取变量函数, 返回作为字典的变量。方法可以带有参数，此机制创建变量非常灵活。

```
get_variables
getVariables
```

另外， 除了Python模块，也可以使用Python类或者Java类实现， 框架会实例化这些类作为变量。 创建这种对象实例的变量，方法也和上面创建变量的2种方法一致。

导入和使用Varialbe File

在Setting中导入

Variable File的导入和Resource File的导入相似
Path先以相对要求导入的文件所在目录的相对路径进行； 如果没有找到，会在Python的模块搜索路径中查找
导入的路径和参数，都支持使用变量

样例

```
*** Settings ***
Variables    myvariables.py
Variables    ../data/variables.py
Variables    ${RESOURCES}/common.py
Variables    taking_arguments.py    arg1    ${ARG2}
```

```
Tips:
导入的变量在执行导入操作的测试文件中有效
如果多个文件导入存在重名变量情况，最早导入的变量有效
在Variable Table或者命令行选项创建的变量，可能覆盖Variable File创建的变量
```

通过命令行, **Command Line**导入

命令行选线 --variablefile 也可以使用Variable File
从命令行导入的Varialbe File，作用域范围是全局可用的。这个通过--varialbe选项设置的变量情况是相似的。 如果通过 --variablefile 和 --varialbe 创建的变量名存在冲突，--variable选项创建的变量将会被保留。

样例
文件通过path被引用
如果需要参数，使用":"将参数添加在path后面; 也可以使用";"分隔path和参数

```
--variablefile myvariables.py
--variablefile path/variables.py
--variablefile /absolute/path/common.py
--variablefile taking_arguments.py:arg1:arg2
--variablefile "myvariables.py;argument:with:colons"
--variablefile C:\path\variables.py;D:\data.xls
```

```
Tips
从命令行导入与从Setting中导入Variable File，路径搜索的规则是一致的
```

直接创建变量

如果熟悉Python的同学，对于Variable的导入可能不难理解。 Varialbe File就是以导入Python模块的机制在进行导入。
Variable File导入时候，模块中除了以"_"开始的属性，其他所有全局属性都会被导入作为变量。

```
Tips
变量名是大小写敏感的
建议: 全局变量请使用全大写名称
```

样例*1*
在此定义的所有变量都可以被当做scalar变量来进行使用 @{STRINGS}为list变量，
$STRINGS为一个list
&{MAPPIN}为Gdict变量， $MAPPING为一个包含dict所有key的list

```
VARIABLE = "An example string"
ANOTHER_VARIABLE = "This is pretty easy!"
INTEGER = 42
STRINGS = ["one", "two", "kolme", "four"]
NUMBERS = [1, INTEGER, 3.14]
MAPPING = {"one": 1, "two": 2, "three": 3}
```

样例*2*
为了更明确的定义list变量或者dict变量，可以使用前缀"LIST__" 或者 "DICT__"。 前缀不会不会作为变量名的一部分，它们的作用是告诉RF框架，变量将会是"list-like"或者"dict-like"类型的，框架会执行相应的验证检查。
访问dict变量的值: ${FINNISH.cat}

```
from collections import OrderedDict

LIST\_\_ANIMALS = ["cat", "dog"]
DICT\_\_FINNISH = OrderedDict([("cat", "kissa"), ("dog", "koira")])
```

样例*3*
样例1和样例2创建的变量，可以采用样例3的Variable Table来创建。

```
*** Variables ***
${VARIABLE}           An example string
${ANOTHER VARIABLE}   This is pretty easy!
${INTEGER}            ${42}
@{STRINGS}            one          two          kolme        four
@{NUMBERS}            ${1}         ${INTEGER}   ${3.14}
&{MAPPING}            one=${1}     two=${2}     three=${3}
@{ANIMALS}            cat          dog
&{FINNISH}            cat=kissa    dog=koira
```

使用对象实例作为值

在Variable Table创建变量，变量值局限于字符串或其他基本类型
在Variable File中创建变量，没有这个局限

样例*1*
${MAPPING}的值为Hashtable对象，其中有2个值。

```
from java.util import Hashtable

MAPPING = Hashtable()
MAPPING.put("one", 1)
MAPPING.put("two", 2)
```

样例2 创建了和样例1相似的${MAPPING}，本样例中为python字典实例。
另外, ${OBJ1}和${OBJ2}，值为Variable File中定义的MyObject类的实例。

```
MAPPING = {'one': 1, 'two': 2}

class MyObject:
    def __init__(self, name):
        self.name = name

OBJ1 = MyObject('John')
OBJ2 = MyObject('Jane')
```

动态创建变量

由于Variable File本质上是编程语言进行变量的创建，所以可以达到动态创建变量的效果。

样例*1*
使用python库中随机函数，每次设置不同的值

```
import os
import random
import time

USER = os.getlogin()                # current login name
RANDOM_INT = random.randint(0, 10)  # random integer in range [0,10]
CURRENT_TIME = time.asctime()       # timestamp like 'Thu Apr  6 12:45:21 2006'
if time.localtime()[3] > 12:
    AFTERNOON = True
else:
    AFTERNOON = False
```

样例*2*
可以使用外部数据源(比如数据库，文件，甚至询问用户)来设置变量值

```
import math

def get_area(diameter):
    radius = diameter / 2
    area = math.pi * radius * radius
    return area

AREA1 = get_area(1)
AREA2 = get_area(2)
```

选择导入的变量

为了避免不被需要的变量被导入，有如下手段。 其实熟悉Python的同学可以看出来，都是Python的菜啊。

样例*1*
以"_"开头的属性将不会被导入

```
import math

def _get_area(diameter):
    radius = diameter / 2
    area = math.pi * radius * radius
    return area

AREA1 = _get_area(1)
AREA2 = _get_area(2)
```

样例2
熟悉Python的同学，一定不会陌生，这和__init__.py 中加入 __all__ 变量的语法很像. 该变量包含执行"from-import all"语句时应该导入的模块名字. 它由一个模块名字符串列表组成.

```
import math

__all__ = ['AREA1', 'AREA2']

def get_area(diameter):
    radius = diameter / 2.0
    area = math.pi * radius * radius
    return area

AREA1 = get_area(1)
AREA2 = get_area(2)
```

特殊函数

如果在Variable File中存在特殊函数(get*variables*或者*getVariables)*，可以通过特殊函数来得到变量。
如果特殊函数存在，*RF*框架会调用此函数，来获取函数。 函数的预期返回应该是*Python*的*dict* 或者 *Java*的*Map*， 其中*key*为*Variable*名称， *value*为*Variable*值。 其他的规则，和直接创建变量的情形一样*:* 可以创建*scalar, list, dict*各种类型变量*;* 也支持特殊前缀*"LIST_"* 和 "DICT__"，等。

样例*1*

```
def get_variables():
    variables = {"VARIABLE ": "An example string",
                 "ANOTHER VARIABLE": "This is pretty easy!",
                 "INTEGER": 42,
                 "STRINGS": ["one", "two", "kolme", "four"],
                 "NUMBERS": [1, 42, 3.14],
                 "MAPPING": {"one": 1, "two": 2, "three": 3}}
    return variables
```

样例*2*
特殊函数也支持参数
在测试数据中，参数在path cell后面的cells中给出*;* 在命令行中，参数通过":"或者";"给出

```
variables1 = {'scalar': 'Scalar variable',
    'LIST\_\_list': ['List','variable']}
variables2 = {'scalar' : 'Some other value',
              'LIST\_\_list': ['Some','other','value'],
              'extra': 'variables1 does not have this at all'}

def get_variables(arg):
    if arg == 'one':
        return variables1
    else:
        return variables2
```

使用**Python**类或者**Java**类来实现**Variable File**

由于Variable File导入需要使用文件系统的path进行，使用类来实现Varialbe File有一些限制:

- Python类必须和它所在的module同名
- Java类必须存在于缺省包
- Java类的路径必须以.java或.class结尾，类文件必须存在

无论使用哪种语言实现，框架都会使用无参构造函数创建类实例，通过类实例得到变量。
和直接使用模块一样，变量可以在实例中直接定义，也可以通过特殊方法(get_variables或者getVariables)得到。

如果变量在类实例中直接被定义，为了避免实例方法用来创建变量，全部可调用的属性将会被忽略。

实例1
2个版本都创建了如下变量:
来自类属性的 ${VARIABLE} 和 @{LIST}
来自实例属性的 ${anotherVariable}

Python版本

```
class StaticPythonExample(object):
    variable = 'value'
    LIST__list = [1, 2, 3]
    _not_variable = 'starts with an underscore'

    def __init__(self):
        self.another_variable = 'another value'
```

Java版本

```
public class StaticJavaExample {
    public static String variable = "value";
    public static String[] LIST__list = {1, 2, 3};
    private String notVariable = "is private";
    public String anotherVariable;

    public StaticJavaExample() {
        anotherVariable = "another value";
    }
}
```

实例2
2个版本都使用动态方法创建了变量${dynamic variable}

Python版本

```
class DynamicPythonExample(object):

    def get_variables(self, *args):
        return {'dynamic variable': ' '.join(args)}
```

Java版本

```
import java.util.Map;
import java.util.HashMap;

public class DynamicJavaExample {

    public Map<String, String> getVariables(String arg1, String arg2) {
        HashMap<String, String> variables = new HashMap<String, String>();
        variables.put("dynamic variable", arg1 + " " + arg2);
        return variables;
    }
}
```

内建变量, Build-in Variables

作者: 虞科敏

OS变量

${CURDIR} - Test Data File所在目录的绝对路径
${TEMPDIR} - 系统临时目录的绝对路径(Linux "/tmp"; Window "c:\Documents and Settings\\Local Settings\Temp")
${EXECDIR} - 测试执行开始目录的绝对路径
${/} - 操作系统目录路径分隔符(Linux "/"; Window "\")
${:} - 操作系统路径环境变量的分隔符(Linux ":"; Window ";")
${\n} - 操作系统文件行分隔符(Linux "\n"; Window "\r\n")

样例

```
*** Test Cases ***
Example
    Create Binary File    ${CURDIR}${/}input.data    Some text here${\n}on two lines
    Set Environment Variable    CLASSPATH    ${TEMPDIR}${:}${CURDIR}${/}foo.jar
```

数字变量

数字变量语法，用来创建整数或者浮点数。
当关键字希望得到一个实际的数字，而不是字符串时，应该使用数字变量语法。

样例*1*

```
*** Test Cases ***
Example 1A
    Connect    example.com    80       # Connect gets two strings as arguments

Example 1B
    Connect    example.com    ${80}    # Connect gets a string and an integer

Example 2
    Do X    ${3.14}    ${-1e-4}        # Do X gets floating point numbers 3.14 and -0.
0001
```

样例*2*
数字进制的前缀(注: 本语法不是大小写敏感的)
0b或者0B - 2进制

0o或者0O - 8进制
0x或者0X - 16进制

```
*** Test Cases ***
Example
    Should Be Equal    ${0b1011}    ${11}
    Should Be Equal    ${0o10}      ${8}
    Should Be Equal    ${0xff}      ${255}
    Should Be Equal    ${0B1010}    ${0XA}
```

布尔变量

样例
布尔变量不是大小写敏感的: ${true} == ${True}, ${false} == ${False}

```
*** Test Cases ***
Boolean
    Set Status    ${true}               # Set Status gets Boolean true as an argument

    Create Y    something   ${false}    # Create Y gets a string and Boolean false
```

空值变量

样例
空值变量不是大小写敏感的, Null和None同义: ${null} == ${Null} == ${none} == ${None}

```
*** Test Cases ***
None
    Do XYZ    ${None}                   # Do XYZ gets Python None as an argument

Null
    ${ret} =    Get Value    arg        # Checking that Get Value returns Java null
    Should Be Equal    ${ret}    ${null}
```

空格和空字符串变量

空格变量: ${SPACE} == " "
多个空格语法: ${SPACE * 2} == " \ "

空字符串变量: ${EMPTY} = \

样例*1*
空格和空字符串样例

```
*** Test Cases ***
One Space
    Should Be Equal    ${SPACE}          \ \

Four Spaces
    Should Be Equal    ${SPACE * 4}      \ \ \ \ \

Ten Spaces
    Should Be Equal    ${SPACE * 10}     \ \ \ \ \ \ \ \ \ \

Quoted Space
    Should Be Equal    "${SPACE}"        " "

Quoted Spaces
    Should Be Equal    "${SPACE * 2}"    " \ "

Empty
    Should Be Equal    ${EMPTY}          \
```

样例2
空list和空dict样例

```
*** Test Cases ***
Template
    [Template]    Some keyword
    @{EMPTY}

Override
    Set Global Variable    @{LIST}    @{EMPTY}
    Set Suite Variable     &{DICT}    &{EMPTY}
```

自动变量

自动变量被RF框架创建和修改，在测试执行过程中值可能会变量; 另外，某些自动变量在执行过程中并非总是可用。
修改自动变量，并不能对变量的初始值产生影响。
但可以用某些内建关键字来修改某些自动变量的值。

变量名	含义	可用范围
${TEST NAME}	当前测试用例的名字	Test case
@{TEST TAGS}	当前测试用例的标签(按字母序)。可以使用"Set Tags"和"Remove Tags"关键字修改	Test case
${TEST DOCUMENTATION}	当前测试用例的文档说明。可以使用"Set Test Documentation"关键字修改	Test case
		Test

		teardown
${TEST MESSAGE}	当前测试用例的消息	Test teardown
${PREV TEST NAME}	前一个测试用例的名字。如果还没有用例被执行，值为空字符串	Everywhere
${PREV TEST STATUS}	前一个测试用例的状态: Pass或FAIL。如果还没有用例被执行，值为空字符串	Everywhere
${PREV TEST MESSAGE}	前一个测试用例的错误消息	Everywhere
${SUITE NAME}	当前Suite的全名	Everywhere
${SUITE SOURCE}	Suite的文件或目录的绝对路径	Everywhere
${SUITE DOCUMENTATION}	当前测试Suite的文档说明。可以使用"Set Suite Documentation"关键字修改	Everywhere
&{SUITE METADATA}	当前测试Suite的元数据。可以使用"Set Suite Metadata"关键字修改	Everywhere
${SUITE STATUS}	当前测试Suite的状态: Pass或FAIL	teardown
${SUITE MESSAGE}	当前测试Suite的消息, 包括统计信息	Suite teardown
${KEYWORD STATUS}	当前测试关键字的状态: Pass或FAIL	User keyword teardown
${KEYWORD MESSAGE}	当前测试关键字的错误消息	User keyword teardown
${LOG LEVEL}	当前的日志级别	Everywhere
${OUTPUT FILE}	输出(output)文件的绝对路径	Everywhere
${LOG FILE}	日志(log)文件的绝对路径。如果没有日志文件，值为空字符串	Everywhere
${REPORT FILE}	报告(report)文件的绝对路径。如果没有报告文件，值为空字符串	Everywhere
${DEBUG FILE}	调试(debug)文件的绝对路径。如果没有调试文件，值为空字符串	Everywhere
${OUTPUT DIR}	输出(output)文件所在目录的绝对路径	Everywhere

```
Tips:
${SUITE SOURCE}, ${SUITE NAME}, ${SUITE DOCUMENTATION}, &{SUITE METADATA}在libraries和v
ariable files被导入时已可用
```

变量的属性和作用范围

作者: 虞科敏

属性, Properties

命令行设置的变量

- 命令行设置的变量拥有在测试开始执行前，所有能设置变量的最高优先级: 可以覆盖Test Case File中
- Variable Table, 导入的Resource File和Variable File的相关设置
- --variable (-v) 优先于 --variablefile (-V)
- 如果设置同一个变量多次，后面的设置会覆盖前面的设置
- 可以在启动脚本中设置变量，这种情况会覆盖命令行选项的设置同名变量
- 如果在多个变量文件中设置同名变量，第一次设置的变量具有最高优先级(先导入优先原则)

在Test Case File的Vaiable Table中设置的变量

- 对于本文件中的所有Test Case可用
- 会覆盖文件中通过导入Reousrce File和Variable File创建的同名变量
- 本文件中其他的表(如Setting Table可以使用本文件中Vairable Table中创建的变量)

通过导入Resource File和Variable File设置的变量

- 在Test Data中设置的变量中，优先级最低
- 来自Resource File的变量 和 来自Variable File的变量，优先级相同
- 对于来自Resource File和Variable File的同名变量，采用"先导入优先原则"
- 如果一个Resource File导入其他Rsource File和Variable File, 采用"本地优先于导入"的原则
- 从Resource File和Variable File导入的变量，对于导入它们的文件中的Variable Table不可用

  Tips: Variable Table的处理先于Setting Table的处理(Resource File和Variable File在Setting Table中被导入)

在执行过程中设置的变量

在执行过程中设置的变量(通过关键字返回值设置，或者使用内建变量设置关键字)会覆盖其作用域内的同名变量
但其不会影响作用域外的变量

内建变量

内建变量拥有最高优先级，不能被Vairable Table或者命令行所覆盖； 但它们可能被框架重置数字变量是个例外: 当变量不能被找到时，它们会被动态解析； 这样它们可能被覆盖，但不推荐这样做！

作用范围, Scope

全局范围

以下变量具有全局范围:

```
通过命令行选项创建的变量
内建关键字"Set Global Variable"设置(和修改)全局变量
内建变量
```

全局范围变量，建议使用全大写命名

Test Suite范围

以下变量具有Test Suite范围:

```
Varaible Table创建的变量
通过Resource File和Variable File导入的变量
内建关键字"Set Suite Variable"设置(和修改)的变量
```

Test Suite范围变量，建议使用全大写命名

Test Case范围

以下变量具有Test Case范围:

```
内建关键字"Set Case Variable"设置(和修改)的变量
```

Test Case范围变量，建议使用全大写命名

本地(Local)范围

以下变量具有Local范围:

```
通过扩展关键字的返回值创建的变量，用户自定义关键字使用其作为参数
```

Local范围变量，建议使用全小写命名

变量的高级特性

作者: 虞科敏

暂缺

用户关键字, User Keyword

作者: 虞科敏

基础语法

在Keyword Table中， 组合使用已有的关键字，创建更高一级的关键字，被称为用户关键字。区别于在测试库中实现的低级库关键字。
创建用户关键字的语法，和创建Test Case很相似。

用户关键字可以在以下位置被创建:

- Test Case File
- Resource File
- Suite的__init__ File

样例
使用底层库关键字，或者其他用户关键字，定义新的用户关键字
通常，关键字名位于第2个cell中； 对于返回值赋值语法，关键字名位于返回值变量后1个cell(第3个cell）中
用户关键字可以带参数，也可以不带参数

```
*** Keywords ***
Open Login Page
    Open Browser    http://host/login.html
    Title Should Be    Login Page

Title Should Start With
    [Arguments]    ${expected}
    ${title} =    Get Title
    Should Start With    ${title}    ${expected}
```

Keywork Table的设置项(Setting)

样例

```
[Documentation]
Used for setting a user keyword documentation.
[Tags]
Sets tags for the keyword.
[Arguments]
Specifies user keyword arguments.
[Return]
Specifies user keyword return values.
[Teardown]
Specify user keyword teardown.
[Timeout]
Sets the possible user keyword timeout. Timeouts are discussed in a section of their o
wn.
```

关键字名

用户关键字的名称，在定义的第一行中指定
关键字名可以很长，以描述关键字主要功能为目标

文档串

关键字文档串，和用例文档串很相似
文档工具Libdoc能从Resource File中提取出正式的关键字文档，文档串会作为重要的说明性信息
文档串的第一行也会出现在测试日志中
以 "*DEPRECATED*" 作为文档串的开头，可以标记关键字过期。

标签

关键字标签使用 "[Tags]" 进行设置，这和用例标签很相似。
设置(Setting)中的Force Tags和Default Tags不会影响关键字的标签。

文档工具Libdoc提取出的正式关键字文档中，关键字标签也会被显示。
命令行选项 --removekeywords 和 --flattenkeywords 支持根据标签选择关键字。
和用例标签一样，RF框架预留标签前缀 **"robot-"** 作为特殊特性的用途。除非用户希望激活这些特性，否则请不要自己定义以此为前缀的标签。

样例
在文档串的最后一行，可以指定标签。
样例中定义的2个用户关键字，标签是一样的，都是3个标签: my, fine, tags。

```
*** Keywords ***
Settings tags using separate setting
    [Tags]    my    fine    tags
    No Operation

Settings tags using documentation
    [Documentation]    I have documentation. And my documentation has tags.
    ...                Tags: my, fine, tags
    No Operation
```

用户关键字参数

作者: 虞科敏

参数使用 "[Arguments]" 进行设置， 参数名设置使用变量的语法，如 ${arg}
参数命名和框架无关，应该能对参数作用具有很好的描述性
参数建议使用小写形式，如 ${my_arg}, ${my arg}, ${myArg}

位置参数

位置参数的格式在调用时候的数量，必须和关键字签名的参数数量相同

样例

```
*** Keywords ***
One Argument
    [Arguments]    ${arg_name}
    Log    Got argument ${arg_name}

Three Arguments
    [Arguments]    ${arg1}    ${arg2}    ${arg3}
    Log    1st argument: ${arg1}
    Log    2nd argument: ${arg2}
    Log    3rd argument: ${arg3}
```

缺省值

缺省值语法: ${arg}=default
缺省值中可以包含test范围，suite范围，global范围的变量，但不可以包含关键字local范围的变量

```
Tips:
缺省值语法是空格敏感的，在"="前不允许空格；在"="后的空格被认为是值的一部分
```

样例
如果几个带有缺省值的参数，只有部分参数值需要被重置，可以使用命名参数语法:
arg=newvalue
命名参数语法中，参数名不要使用${}进行修饰; 而值可以是字符串，也可以是变量。

在下面的测试用例中， arg2=new value即为命名参数语法.
关键字"Two Arguments With Defaults"的arg1参数值为缺省值， arg2参数的值为"new value "

```
*** Test Cases ***
Example
    Two Arguments With Defaults    arg2=new value

*** Keywords ***
One Argument With Default Value
    [Arguments]    ${arg}=default value
    [Documentation]    This keyword takes 0-1 arguments
    Log    Got argument ${arg}

Two Arguments With Defaults
    [Arguments]    ${arg1}=default 1    ${arg2}=${VARIABLE}
    [Documentation]    This keyword takes 0-2 arguments
    Log    1st argument ${arg1}
    Log    2nd argument ${arg2}

One Required And One With Default
    [Arguments]    ${required}    ${optional}=default
    [Documentation]    This keyword takes 1-2 arguments
    Log    Required: ${required}
    Log    Optional: ${optional}

 Default Based On Earlier Argument
    [Arguments]    ${a}    ${b}=${a}    ${c}=${a} and ${b}
    Should Be Equal    ${a}    ${b}
    Should Be Equal    ${c}    ${a} and ${b}
```

可变长参数

在关键字签名中，位置变量后面，跟上list变量 @{varargs} 作为可变长参数 也可以和缺省值语法混合使用，由后面的list变量接收所有未被匹配认领的位置参数(非命名参数)
list变量可以接受 0 到 多个参数值
以上这些用法， @{varargs} 其实和python里面的非关键字可变长参数(tuple, *args)和其他参数之间的交互是一致的

样例
对于关键字"Required, Default, Varargs":
如果超过1个参数值被指定，参数${opt}将总是使用指定值，而不会使用缺省值。即时给定值为${EMPTY}也会赋值给${opt}参数。
另外，请注意list参数如何在:FOR语法中被使用。

```
*** Keywords ***
Any Number Of Arguments
    [Arguments]    @{varargs}
    Log Many    @{varargs}

One Or More Arguments
    [Arguments]    ${required}    @{rest}
    Log Many    ${required}    @{rest}

Required, Default, Varargs
    [Arguments]    ${req}    ${opt}=42    @{others}
    Log    Required: ${req}
    Log    Optional: ${opt}
    Log    Others:
    : FOR    ${item}    IN    @{others}
    \    Log    ${item}
```

自由参数

在关键字签名中，位置变量和可变长变量后面，跟上dict变量 &{kwargs} 作为自由参数
自由参数应该是关键字签名中的最后一个参数
最后的dict变量，将会接收所有未被位置参数匹配的命名参数
以上这些用法，&{kwargs} 其实和python里面的关键字可变长参数(dict, **kwargs)和其他参数之间的交互是一致的。

样例

```
*** Keywords ***
Kwargs Only
    [Arguments]    &{kwargs}
    Log    ${kwargs}
    Log Many    @{kwargs}

Positional And Kwargs
    [Arguments]    ${required}    &{extra}
    Log Many    ${required}    @{extra}

Run Program
    [Arguments]    @{varargs}    &{kwargs}
    Run Process    program.py    @{varargs}    &{kwargs}
```

变量嵌入到关键字名中

作者: 虞科敏

除了常规的方法: 参数在关键字名后面的cell中指定，也可以将变量嵌入到关键字名之中。
这样做最大的好处是，眞正意义上的并且简洁的语句，可以作为关键字名。

基础语法

嵌入变量语法，在关键字定义时，不在明确的使用[Arguments]指定参数。
在变量名中使用的参数，在关键字内部被解析出来使用，它们的值依赖于如何调用关键字的。

变量嵌入语法的关键字，使用和其他关键字区别不大。 除了:
在关键字名中，空格和"_"不会被忽略掉; 关键字名不是大小写敏感的

变量嵌入语法的关键字，不支持参数缺省值 和 可变长参数
变量嵌入语法，只支持用户关键字

样例*1*
理想的变量嵌入，如下的关键字，我们可以定义一个关键字，可以执行"Select cat from list"或者"Select dog from list"等关键字。
"Select cat from list"， 那么${anmial}=cat
"Select dog from list"， 那么${anmial}=dog

嵌入参数的关键字名非大小写敏感， select x from list == Select x fromlist

```
*** Keywords ***
Select ${animal} from list
    Open Page    Pet Selection
    Select Item From List    animal_list    ${animal}
```

嵌入参数匹配过多

确保调用关键字时候，使用的参数匹配正确的值，是最具挑战和需要一定技巧的。
"Select ${city} ${team}" 如果使用 "Select Los Angeles Lakers" 调用， 就不会工作得很好。

一种解决方法是使用引号: Select "${city}" "${team}" => Select "Los Angeles" "Lakers"

另一个更强大，但也更复杂的方法是使用 定制正则表达式
请参考后面章节

如果情况变得复杂，难于管理，使用普通的位置参数可能才是一个更好的选择。

用户正则表达式, custom regular expression

嵌入参数语法进行值匹配的内部机制，其实是 正则表达式匹配。
缺省的，每个参数位置是使用正则表达式".*"来进行替换，这个正则表达式会匹配所有的字符串。

使用正则表达式的语法: ${arg:regexp}
比如，参数值只能匹配数字，可以写为: ${arg:\d+}

样例*1*
使用引号"，有时候可以帮助解决匹配过多的问题。
但是下面定义的关键字，I execute "ls" with "-lh" 既可以匹配第1个用户关键字，也可以匹配第2个用户关键字。 匹配有歧义，会导致测试失败。

```
*** Test Cases ***
Example
    I execute "ls"
    I execute "ls" with "-lh"

*** Keywords ***
I execute "${cmd}"
    Run Process    ${cmd}    shell=True

I execute "${cmd}" with "${opts}"
    Run Process    ${cmd} ${opts}    shell=True
```

样例*2*
"${cmd: "+}" - cmd参数匹配非"的字符
${a:\d+} ${operator:[+-]} ${b:\d+} - a和b参数匹配数字，operator匹配+或-
${date:\d{4}-\d{2}-\d{2}} - date参数匹配 4数字-2数字-2数字

```
*** Test Cases ***
Example
    I execute "ls"
    I execute "ls" with "-lh"
    I type 1 + 2
    I type 53 - 11
    Today is 2011-06-27

*** Keywords ***
I execute "${cmd:[^"]+}"
    Run Process    ${cmd}    shell=True

I execute "${cmd}" with "${opts}"
    Run Process    ${cmd} ${opts}    shell=True

I type ${a:\d+} ${operator:[+-]} ${b:\d+}
    Calculate    ${a}    ${operator}    ${b}

Today is ${date:\d{4\}-\d{2\}-\d{2\}}
    Log    ${date}
```

支持的正则表达式语法

由于RF框架使用Python实现，Python的 re模块 支持标准的正则表达式语法，参数的正则表达式语法也继承了这些语法能力
正则表达式的 (?...) 扩展，在参数正则表达式语法中不被采用
如果正则表达式语法不合法，会导致相应的关键字失败

```
(pattern) 匹配pattern, 并获取这一匹配
(?:pattern) 匹配pattern, 但是不获取匹配结果，即为非获取匹配，不进行存储供以后使用
(?=pattern) 正向预查，在任何匹配 pattern 的字符串开始处匹配查找字符串, 也为非获取匹配； 预查不消耗字符
(?!pattern) 反向预查，在任何不匹配 pattern 的字符串开始处匹配查找字符串，也为非获取匹配； 预查不消耗字符
```

特殊字符的转义

} => \} - 在正则表达式中定义出现次数需要{}语法，所以在正则表达式中出现，需要转义
\ => \\ , 更保险的是4个(即 \\\\) - \在Python正则表达式中有特殊含义，如果需要字面上的\字符，需要转义

在嵌入参数正则表达式使用变量

除了文本的匹配，RF框架会自动加强匹配能力，去匹配变量值
总是可以将变量 和 带正则表达式的嵌入参数关键字 一起使用

样例
以下测试用例Example的关键字，会匹配之前定义的关键字 I type ${a:\d+} ${operator:[+-]} ${b:\d+} 和 Today is ${date:\d{4}-\d{2}-\d{2}}

```
*** Variables ***
${DATE}    2011-06-27

*** Test Cases ***
Example
    I type ${1} + ${2}
    Today is ${DATE}
```

行为驱动开发范例, BDD Example

当进行BDD风格测试用例时，希望采用高级的语句风格的关键字

样例
如前所述，Given-When-Then-And-But 可以是，也可以不是关键字名的一部分

```
*** Test Cases ***
Add two numbers
    Given I have Calculator open
    When I add 2 and 40
    Then result should be 42

Add negative numbers
    Given I have Calculator open
    When I add 1 and -2
    Then result should be -1

*** Keywords ***
I have ${program} open
    Start Program    ${program}

I add ${number 1} and ${number 2}
    Input Number    ${number 1}
    Push Button     +
    Input Number    ${number 2}
    Push Button     =

Result should be ${expected}
    ${result} =    Get Result
    Should Be Equal    ${result}    ${expected}
```

```
Tips: RF中BDD编程风格的特性是受到Cucumber框架的启发添加的. 我的下一个开源书计划正好是Python-Cucumber: Lettuce :)
```

用户关键字返回值

作者: 虞科敏

和库关键字一样，用户关键字也可以带有返回值
最典型的返回值定义，是使用[Return]进行设置
也可以使用内建关键字: Return From Keyword 或者 Return From Keyword If
不管采用哪种方式返回值，在测试用例中，返回值都可以被赋值给变量，然后传递给其他关键字

[Return]

语法: [Return] cell 后紧跟 被返回值cell

关键字可以返回多个值，语法: [Return] cell 后紧跟 多个值的cells

这些值可以被赋值给:
一次性赋值给多个scalar变量
1个list变量
多个scalar变量+1个list变量

样例

```
*** Test Cases ***
One Return Value
    ${ret} =    Return One Value    argument
    Some Keyword    ${ret}

Multiple Values
    ${a}    ${b}    ${c} =    Return Three Values
    @{list} =    Return Three Values
    ${scalar}    @{rest} =    Return Three Values

*** Keywords ***
Return One Value
    [Arguments]    ${arg}
    Do Something    ${arg}
    ${value} =    Get Some Value
    [Return]    ${value}

Return Three Values
    [Return]    foo    bar    zap
```

使用特殊关键字返回

特殊关键字: Return From Keyword 或者 Return From Keyword If
可以在关键字中间有条件地返回值
这些关键字可以带可选的返回值，返回值的处理行为和[Return]一样

样例
关键字"Return One Value"展示和[Return]一样的功能
关键字"Find Index"展示在:For循环中有条件返回值的功能

```
*** Test Cases ***
One Return Value
    ${ret} =    Return One Value  argument
    Some Keyword    ${ret}

Advanced
    @{list} =    Create List    foo    baz
    ${index} =    Find Index    baz    @{list}
    Should Be Equal    ${index}    ${1}
    ${index} =    Find Index    non existing    @{list}
    Should Be Equal    ${index}    ${-1}

*** Keywords ***
Return One Value
    [Arguments]    ${arg}
    Do Something    ${arg}
    ${value} =    Get Some Value
    Return From Keyword    ${value}
    Fail    This is not executed

Find Index
    [Arguments]    ${element}    @{items}
    ${index} =    Set Variable    ${0}
    :FOR    ${item}    IN    @{items}
    \    Return From Keyword If    '${item}' == '${element}'    ${index}
    \    ${index} =    Set Variable    ${index + 1}
    Return From Keyword    ${-1}    # Could also use [Return]
```

用户关键字Teardown

作者: 虞科敏

可以使用[Teardown]设置用户关键字的Teardown过程。
Teardown一般是一个单一的关键，常常可能是另一个用户关键字。
即使用户关键字失败，Teardown过程也会被执行。
和其他的Teardown行为一样， 其中的步骤不管其他步骤是否成功，都会被执行。
用户关键字Teardonw任何步骤失败，都会导致所在的测试用例失败，后面的测试步骤将不会再被执行。
作为Teardown中被执行的用户关键字名，可以通过变量传入。

样例

```
*** Keywords ***
With Teardown
    Do Something
    [Teardown]    Log    keyword teardown

Using variables
    [Documentation]    Teardown given as variable
    Do Something
    [Teardown]    ${TEARDOWN}
```

资源文件, Resource File

作者: 虞科敏

在Test Case File和suite的__init__ File中定义的用户关键字和变量只能在创建它们的文件中被使用。
Resource File提供了一个共享用户关键字和变量的手段。
Resource File的结构和Test Case File非常相似，但其中不能有Test Case Table。

导入和使用Resource File

Resource File通过Setting Table中的Source命令进行导入，在Source Cell后，紧跟提供Resource File Path信息的cell。

如果path为绝对路径，将会被直接使用
如果path为相对路径，path先以相对要求导入的文件所在目录的相对路径进行； 如果没有找到，会在Python的模块搜索路径中查找
导入的路径支持使用变量

路径中可以包含参数，建议使用这项功能已使path尽量独立于系统，如${RESOURCES}/login_resources.html 或 ${RESOURCE_PATH}
另外，在windows系统中，路径中的/会被自动替换为\

Resource File定义的关键字和变量，只在导入它的文件中可用
在Resource File导入的库Libraries，Resource File, Variable File，这些文件定义的关键字和变量，也是可用的

Resource File结构

Resource File的结构和Test Case File非常相似，但其中不能有**Test Case Table**。
另外，Resource File中的Setting Table只能包含**import settings** (Library, Resource, Variables) 和 **Documentation**。

Variable Table和Keyword Table的使用，和Test Case File完全一样。

如果多个资源文件中存在同名的用户关键字，为了引用这些用户关键字，需要在关键字名前加上资源文件的名字(不带后缀)。 如，myresources.Some Keyword 和 common.Some Keyword。

如果多个资源文件中存在同名的变量，采用"先导入优先原则"。

文档串

Resource File中定义的用户关键字可以使用[Documentation]设置文档串。
Resource File本身也可以在Setting Table中定义自己的文档串 (和Suite中一样)。

工具Libdoc和RIDE都会使用这些文档串，当它们打开资源文件时，这些文档串就会被它们使用。

关键字的文档串的第1行在它们被执行时，会被log记录，但其他的Resource File文档串在执行时会被忽略。

样例

```
*** Settings ***
Documentation     An example resource file
Library           Selenium2Library
Resource          ${RESOURCES}/common.robot

*** Variables ***
${HOST}           localhost:7272
${LOGIN URL}      http://${HOST}/
${WELCOME URL}    http://${HOST}/welcome.html
${BROWSER}        Firefox

*** Keywords ***
Open Login Page
    [Documentation]    Opens browser to login page
    Open Browser    ${LOGIN URL}    ${BROWSER}
    Title Should Be    Login Page

Input Name
    [Arguments]    ${name}
    Input Text    username_field    ${name}

Input Password
    [Arguments]    ${password}
    Input Text    password_field    ${password}
```