| | |
|---|---|
| 项目标号 | |
| 密级： | 绝密 机密 限制 一般 |
| 草稿文档： | |
| 正式文档： | |
| 正式文档修正： | |

# WEB 自动化测试框架

Version 0.5
［文档编号］

项目名称：

项目支持部门：产品测评部

项目负责人：

作者姓名：毕小超

定稿日期：

文档修改记录

| 序号 | 版本 | 修改者 | 主要修改内容 | 日　期 |
|---|---|---|---|---|
| 1 | V0.5 | 毕小超 | 初稿 | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |

# 目 录

# 1. 自动化测试的关键思路介绍

因为软件测试的工作量很大（40% 到 60% 的总开发时间），而又有很大部分适于自动化，因此，测试的改进会对整个开发工作的质量、成本和周期带来非常显著的效果。

首先，谈谈在测试自动化的情况下，带有图形界面的产品的测试用例的设计问题。因为图形界面的输出显示不是很容易做到测试结果自动化比较，所以一般的做法是把图形界面输出的部分单独建立测试用例，以手工运行。而所有非图形输出则可进行自动测试。

下面举出一些测试自动化的例子:

## 1.1 测试个案（test case ，或称为测试用例）的生成

用编程语言或更方便的剧本语言（例如 VBS, Ruby 等）写出短小的程序来产生大量的测试输入（包括输入数据与操作指令）。或同时也按一定的逻辑规律产生标准输出。输入与输出的文件名字按规定进行配对，以便控制自动化测试及结果核对的程序易于操作。

这里提到测试个案的命名问题,如果在项目的文档设计中作统一规划的话,软件产品的需求与功能的命名就应该成为后继开发过程的中间产品的命名分类依据。这样，就会为文档管理和配置管理带来很大的方便，使整个产品的开发过程变得更有条理，更符合逻辑。任何新手半途加入到开发工作中也会更容易进入状态。

## 1.2 测试的执行写控制

单元测试或集成测试可能多用单机运行。但对于系统测试或回归测试，就极有可能需要多台机在网络上同时运行。记住一个这样的原则，在开发过程中的任何时候，如果你需要等候测试的运行结果的话，那就是一个缩短开发时间的机会。

对于单个的测试运行，挖潜的机会在测试的设置及开始运行和结果的对比及显示。有时候，需要反复修改程序，重新汇编和重新测试。这样，每一个循环的各种手工键入的设置与指令所花费的时间，加起来就非常可观。如果能利用 make 或类似的软件工具来帮助，就能节省大量的时间。

对于系统测试或回归测试这类涉及大量测试个案运行的情况，挖潜的机会除了利用软件工具来实现自动化之外，就是怎样充分利用一切硬件资源。往往，

就算是在白天的工作时间内，每台计算机的负荷都没有被充分利用。能够把大量测试个案分配到各台机器上去同时运行，就能节省大量的时间。另外，把大量的系统测试及回归测试安排到夜间及周末运行，更能提高效率。

如果不购买商品化的工具的话，应当遵从正规的软件开发要求来开发出好的软件测试自动化工具。在实践中，许多企业自行开发的自动化工具都是利用一些现成的软件工具再加上自己写的程序而组成的。这些自己开发的工具完全是为本企业量身定做的，因此可用性非常强。同时，也能根据需要随时进行改进，而不必受制于人。当然，这就要求有一定的人力的投入。

在设计软件自动测试工具的时候，路径（path）控制是一个非常重要的功能。理想的使用情况是：这个工具可以在任何一个路径位置上运行，可以到任何路径位置去取得测试用例，同时也可以把测试的结果输出放到任何的路径位置上去。这样的设计，可以使不同的测试运行能够使用同一组测试用例而不至于互相干扰，也可以灵活使用硬盘的空间，并且使备份保存工作易于控制。

同时，软件自动测试工具必须能够有办法方便地选择测试用例库中的全部或部分来运行，也必须能够自由地选择被测试的产品或中间产品采作为测试对象。

## 1.3 测试结果与标准输出的对比

在设计测试用例的时候，必须考虑到怎样才能够易于对此测试结果和标准输出。输出数据量的多少及数据格式对比较的速度有直接影响。而另一方面，也必须考虑到输出数据与测试用例的测试目标的逻辑对应性及易读性，这将会大大有利于分析测试所发现的不吻合，也有利于测试用例的维护。

许多时候，要写一些特殊的软件来执行测试结果与标准输出的对比工作，因为可能有部分的输出内容是不能直接对比的（比如，对运行的日期时间的记录，对运行的路径的记录，以及测试对象的版本数据等），就要用程序进行处理。

## 1.4 不吻合的测试结果的分析、分类、记录和通报

上一点所谈到的，用于对测试结果与标准输出进行对比的特殊软件，往往也同时担任对不吻合的测试结果进行分析、分类、记录和通报的任务。

“分析”是找出不吻合的地方并指出错误的可能起因。“分类”包括各种统计上的分项，例如，对应的源程序的位置，错误的严重级别（提示、警告、非失效性错误、失效性错误；或别的分类方法），新发现的还是已有记录的错误，等等。“记录”，是按分类存档。“通报”，是主动地对测试的运行者及测试用例的“负责人”通报出错的信息。

这里提到测试用例的“负责人”的概念。是用以指定一个测试用例运行时发现的缺陷，由哪一个开发人员负责分析（有时是另外的开发人员引进的缺陷而导致的错误）及修复。在设立测试用例库时，各用例均应有指定的负责人。

最直接的通报方法是由自动测试软件发出电子邮件给测试运行者及测试用例负责人。邮件内容的详细程度可根据需要灵活决定。

### 1.5 总测试状况的统计，报表的产生

这些都是自动测试工具所应有的功能。目的是提高过程管理的质量，同时节省用于产生统计数据的时间。

产生出来的统计报表，最好是存放到一个约定的路径位置，以便任何有关人员都知道怎样查阅。同时，可按需要用电子邮件向适当的对象（如项目经理，测试经理和质量保证经理）寄出统计报表。

### 1.6 自动测试与开发中产品每日构建（build ）的配合

自动测试应该是整个开发过程中的一个有机部分。自动测试要依靠配置管理来提供良好的运行的环境，同时它必须要与开发中的软件的构建紧密配合。

在开发中的产品达到一定程度的时候，就应该开始进行每日构建和测试。这种做法能使软件的开发状态得到频繁的更新，以及及早发现设计和集成的缺陷。

为了充分利用时间与设备资源，下班之后进行自动的软件构建，紧接着进行自动测试（这里多数指的是系统测试或回归测试），是一个非常行之有效的方法。如果安排得好，到第二天上班时，测试结果就已经在各人的电子邮箱里面面了，等待着新的一天的开发工作。

## 2. 编码基础介绍

### 2.1 ruby 的由来

松本行弘(Matz)是日本一家开源软件公司的程序员,有 15 年的编程经验。在工作中，他希望有一种比 Perl 强大，比 Python 更面向对象的语言。从 1993 年 2 月，他开始设计一个全新的自己的语言，1994 年 12 月发布了第一个 alpha 版本，并且将这种新语言定名为 Ruby（红宝石）。发展到现在，最新稳定版本是 Ruby 1.8.6 。

## 2.2 ruby 的特性

计算机编程语言的发展总是与飞速变化的世界息息相关的，Ruby 是为了适应变化、提高和完善编程艺术而出现的。

- 完全开源
- 多平台 Ruby 可以运行在 Linux， UNIX， Windows， MS-DOS， BeOS，OS/2…
- 多线程 线程就是指能在一个程序中处理若干控制流的功能。与 OS 提供的进程不同的是，线程可以共享内存空间。
- 完全面向对象
- 不需要内存管理具有垃圾回收（Garbage Collect，GC）功能，能自动回收不再使用的对象。
- 解释执行 其程序无需编译即可轻松执行。
- 功能强大的字符串操作／正则表达式
- 具有异常处理功能
- 可以直接访问 OS Ruby 可以使用（UNIX 的）绝大部分的系统调用。单独使用 Ruby 也可以进行系统编程。
- 动态类型语言 Ruby 的变量没有类型，因此不必为类型匹配而烦恼。
- 动态语言 程序运行中，可以新加入属性，行为，也可以重写方法。
- 支持操作符重写
- 支持无限精度的数字 例如计算 400 的阶乘也轻而易举。
- 丰富的库函数
- 用模块进行混合插入（Mix-in） Ruby 舍弃了多重继承，但拥有混合插入功能。使用模块来超越类的界限来共享数据和方法等。
- 语法简单 它是脚本语言，没有指针，学习曲线比较低。

总结来说 Ruby 吸取了 perl 的正则表达式， python 的简单性可读性，smalltalk 的纯面向对象语法和单继承，LISP 的无穷嵌套的语法，Java 的线程…

## 2.3 Watir 介绍

Watir 封装了对 web 页面元素识别的机制，测试者可以利用 Ruby 在这个框架下根据对象的属性识别，而后写 ruby 代码控制测试逻辑，加入检查点等等以达到自动化测试目的，并且有可以随意自由定制框架和测试逻辑的优点。

## 2.4 下载和安装 ruby + watir

- 首先访问 ruby 官方网站：

http://www.ruby-lang.org/zh_CN/downloads/

找到 Windows 系统

Ruby 1.8.6 一步安装 (md5: 00540689d1039964bc8d844b2b0c7db6) 稳定版 (*推荐*)

点击鼠标右键，另存为…，存入你的本地硬盘，这就完成了下载。

运行下载好的文件ruby186-26.exe出现安装向导界面，

点击next;

点击 I Agree;

点击 next; 出现选择安装位置 界面；

改变你想安装 Ruby 的路径，我选择了 e: 盘；

点击next;

点击Install;

点击next;

点击 Finish,一切 OK，安装完成。

- 访问 Watir 官方网站:

http://wtr.rubyforge.org/install.html

按照网站上的描述进行如下操作:

1. 必须先完成 Ruby 的安装；
2. 打开“运行”，输入 cmd 进入控制台；
3. 输入下面代码；

```
gem update --system
gem install watir
```

4. 等待安装完成，如果安装失败，需要手工添加类库文件:

   http://rubyforge.org/frs/?group_id=126

   下最新的 ZIP 版本

   rubygems-1.3.5.zip

   注：为了让 Watir 认识中文并操作文件上传，需要做下面的修改，

   首先修改 C:\ruby\lib\ruby\gems\1.8\gems\watir-1.5.6\watir\input_elements.rb

文件，找到 FileField 类下的 set 方法,把原来的替换成

system("rubyw -e \"require 'win32ole'; @autoit=WIN32OLE.new('AutoItX3.Control'); waitresult=@autoit.WinWait '选择文件', '', 15; sleep 1; if waitresult == 1\" -e \"@autoit.ControlSetText '选择文件', '', 'Edit1', '#'; @autoit.ControlSend '选择文件', '', 'Button2', '';\" -e \"end\"")

## 2.5 开始第一个小程序

- 安装开发工具，此次选择 eclipse-3.4.2，加上 RDT 插件 org.rubypeople.rdt-1.2.0.21672.zip
- eclipse 下载地址：http://www.eclipse.org/downloads/
- RDT 下载地址：http://update1.aptana.org/rdt/3.2/index.html
- 安装好 eclipse 后，打开 eclipse 面板，点 **Help** -> **Software Updates** -> **Available Software -> Add Site -> Archive ->选中下载好的 org.rubypeople.rdt-1.2.0.21672.zip -> 点 OK 按钮**

准备好开发工具后，让我们进入例子：

模拟打开 Google 的主页，然后在 Google 唯一的那个文本框内输“pickaxe”这个字符串，然后按下“Google 搜索”按钮，之后验证搜索结果的页面中是否包含了“Programming Ruby”这个字符串，并根据结果使用 puts 函数在屏幕上打印不同的信息。

代码如下：

```
require ‘watir’ # the watir controller
# open the IE browser
ie = Watir::IE.new
# Step 1: go to the test site: http://www.google.com
ie.goto (http://www.google.com)
# Step 2: enter ‘pickaxe’ in the search text field
ie.text_field(:name, “q”).set(”pickaxe”) # q is the name of the search field
# Step 3: click the ‘Google Search’ button
ie.button(:name, “btnG”).click # “btnG” is the name of the Search button
# Actual Result: Check that the ‘Programming Ruby’ link appears on the results page
if ie.contains_text(”Programming Ruby”)
puts “Test Passed. Found the test string: ‘Programming Ruby’. Actual Results match
Expected Results.”
else
puts “Test Failed! Could not find: ‘Programming Ruby’”
end
# End of test: Google search
```

## 2.6 ruby 语法快览

- 注释与分行

```
# 从#开始到行尾是单行注释
puts 3/5
puts 3/5.0
=begin
puts 6/5
puts 6/5.0
多行注释可以用=begin 和 =end ；
实际上，这也是 Ruby 的内嵌文档（Rdoc）注释，类似 javadoc ，
可以用命令 ri 从源文件生产文档。
=end
```

Rdoc 是内嵌在 ruby 代码之中的，可以转换为 html 文档说明。类似 javadoc。ri 是一个命令行程序，用来查看函数说明、类说明。函数说明、类说明应该放置在 =begin 和 =end 之中。“=begin”一定要写在行首，也就是说，这一行的前六个字符是“=begin”，不允许有空格在这之前。

```
# 演示分行
puts 3/5 ; puts 3/5.0
puts "这里演示" \
"连行"
运行结果：
>ruby E3.1-2.rb
0
0.6
这里演示连行
>Exit code: 0
```

Ruby 中用分号“ ; ”来表示一个语句的结束。一行如果有多个语句，每个语句用分号隔开，而最后一个语句可以省略分号。换行符表示一行结束。如果语句太长，可以用“\”连接下一行。

- 分隔符

| Ruby中的常用分隔符如下： 符号 | 名称 | 用途 |
|---|---|---|
| ； | 分号 | 用来分隔一行中的多个语句 |
| （） | 圆括号 | 提高优先级；定义方法时容纳参数列表 |
| | 空格 | 分隔字符；在可省略（）的地方，代替（） |
| ， | 逗号 | 隔开多个参数 |
| . | 点 | 将对象与它的方法隔开 |
| ：： | 紧连的两个冒号 | 域作用符，将模块（类）与它的常量隔开 |

- 关键字

Ruby 中的关键字如下：

模块定义：module

类定义： class

方法定义：def ， undef

检查类型：defined?

条件语句：if ， then ， else ， elsif ，case ，when ，unless

循环语句：for ，in ，while ，until ，next， break ，do ，

redo ，retry ，yield

逻辑判断：not ，and ，or

逻辑值和空值：true ，false ， nil

异常处理：rescue ，ensure

对象引用：super ，self

块的起始：begin/end

嵌入模块：BEGIN ，END

文件相关：__FILE__ ， __LINE__

方法返回：return

别名： alias

- 运算符

| 优先级 | 能否重写 | 运算符 | 描述 |
|---|---|---|---|
| 最高 | Y | [] []= | 数组下标 数组元素赋值 |
| | Y | ** | 乘冥 |
| | Y | ！ ~ + - | 非 位非 一元加 负号 |
| | Y | * / % | 乘 除 模 |
| | Y | + - | 加 减 |
| | Y | >> << | 右移 左移 |
| | Y | & | 位与 |
| | Y | ^ \| | 位异或 位或 |
| | Y | <= < > >= | 小于等于 小于 大于 大于等于 |
| | Y | <=> == === =~ != !~ | 各种相等判断（!= !~ 不能重写） |
| | | && | 短路与 |
| | | \|\| | 短路或 |
| | | .. ... | 区间的开始点到结束点 |
| | | ? : | 三元条件运算符 |
| | | = %= ~= /= -= += \|= &= >>= <<= *= &&= \|\|= **= | 各种赋值<br>例如：a = 5;<br>b += 3( 意思是：b = b+3 )； |

| | | | |
|---|---|---|---|
| | | defined? | 检查类型 |
| | | not | 逻辑非 |
| | | or and | 逻辑或 逻辑与 |

- 标识名和变量的作用域

| 变量 | | | | 常量 |
|---|---|---|---|---|
| 局部变量 | 全局变量 | 实例变量 | 类变量 | 类名称 |
| name | $debug | @name | @@total | PI |
| fishAndChips | $CUSTOMER | @point_1 | @@symtab | FeetPerMile |
| x_axis | $_ | @X | @@N | String |
| thx1138 | $plan9 | @_ | @@x_pos | MyClass |
| _26 | $Global | @plan9 | @@SINGLE | Jazz_Song |

Ruby 使用一个约定来帮助它区别一个名字的用法:名字前面的第一个字符表明这个名字的用法。局部变量、方法参数和方法名称应该用一个小写字母开头或者用一个下划线开头;全局变量用美元符作为前缀 $;而实例变量用 @ 开头;类变量用 @@ 开头;类名、模块名和常量应该用大写字母开头。

词首字母后面可以是字母、数字和下划线的任意组合; @ 后面不可以直接跟数字。

Ruby 程序代码现在是用 7 位 ACSII 码来表示，通过语言扩展来支持 EUC, SJIS 或 UTF-8 等 8 位编码系统。Ruby 2.0 版本将支持 16 位的 Unicode 编码。

- 数据类型

Ruby 数据类型有数字，字符串，数组，哈希表，区间，正则表达式。

数字分为整数型（1，0，75 ，1e3），浮点型（2.4 ，7.0 ，0.99）。浮点型数据小数点后必须跟数字（ 1.e3 不可以，1.1e3 可以）。数字可以有前缀：0 表示八进制，0x 表示十六进制，0b 表示二进制（0724，0x5AC4，0b11101）。

字符串是在‘’（单引号）、 “”（双引号）之间的代码。

数组的下标从 0 开始。Ruby 的数组和其它语言不同，数组的每个元素可以是不同的类型：[ 2.4， 99， “thank you”，[ a, b ,c ] ，78 ]。

区间：1..5 表示 1，2，3，4，5 ；

1...5 表示 1，2，3，4 。

- 赋值和条件运算符

Ruby 基本的赋值用 “＝” 来完成，就像如下示例：（在不产生歧义的地方，我用 # 表示答案）

```
a = 1  ; b = 2 + 3 #a=1 ,b=5
```

```
a ,b = b ,a #a=5 ,b=1

a = b = 1 + 2 + 3 #a=6 ,b=6

a = (b = 1 + 2) + 3 #a=6 ,b=3

x = 0 #x=0

a,b,c = x, (x+1), (x+2) #a=0 ,b=1,c=2
```

Ruby 的条件运算符比 Java 更加复杂，看例子:

| | |
|---|---|
| <=> | 比较两个对象的大小，大于、等于、小于 分别返回1,0,-1<br>"aab" <=> "acb" # -1 （第二个 a 的 ASCII 码小于 c）<br>[5] <=> [4,9] # 1 （第一个元素 5 > 4） |
| === | 右边的对象是否在左边区间之内,返回 true, flase<br>puts (0..9)=== 3.14 #true<br>puts ('a'..'f')=== 'c' # true |
| =~（匹配） | 用来比较是否符合一个正则表达式,返回模式在字符串中被匹配到的位置，否则返回 nil |
| !~ (不匹配) | 断言不符合一个正则表达式,返回 true, flase |
| <= < > >= | 小于等于 小于 大于 大于等于 |
| == （等于）<br>！＝ （不等于） | 比较两个对象的值是否相等 ，返回 true, flase<br>a=1；b=1.0；a==b #true |
| eql? | 比较两个对象的值、类型是否相等,返回 true, flase<br>a=1；b=1.0；a.eql?(b) #flase（a为整数型，b为浮点型） |
| equal? | 比较两个对象在内存中地址是否相同,返回 true, flase<br>a=1.0；b=1.0；a.equal?(b) #flase<br>a=1.0；b=a ；a.equal?(b) # true |

- 条件判断语句

判断条件是否相等用“==”，注意不要写成“=”。

一. 单行 if（如果） 语句

```
1) if 条件① then 语句 1; 语句 2 ; 语句… end

2) ( 语句 1; 语句 2 ; 语句… ) if 条件
```

二. 多行 if 语句

```
if 条件

语句 1; 语句 2 ; 语句…

elsif 条件

语句 1; 语句 2 ; 语句…

else

语句 1; 语句 2 ; 语句…

end
```

三. unless(除非) 条件语句：

```
unless 条件 = if not (条件)
```

四. case 分支条件语句

| | |
|---|---|
| case 对象 | 例：x=3 |
| when 可能性1 | case x |
| 语句1; 语句2；语句… | when 1..2 |
| when 可能性2 | print "x=",x,";在 1..2中" |
| 语句1; 语句2；语句… | when 4..9, 0 |
| when 可能性… | print "x=",x,";在4..9,0中,或是0" |
| 语句1; 语句2；语句… | else |
| else | print "x=",x,";其它可能" |
| 语句1; 语句2；语句… | end |
| end | 结果：x=3;其它可能 |

注意：

Ruby 里，nil 和 false 为假，其它都为真； puts "is true" if 5 #is true

str="false" ; puts "is true" if str #is true

- 循环语句

一. while（当…） 循环

```
while 条件

语句 1; 语句 2；语句…

end
```

二. 单行 while 循环

```
( 语句 1; 语句 2；语句… ) while 条件
```

三. until（直到…） 循环

| | |
|---|---|
| 我们想输出数字1到9<br>a=1<br>while a <10<br>print a," "<br>a=a+1<br>end<br>#1 2 3 4 5 6 7 8 9 | a=1<br>until a >=10<br>print a," "<br>a=a+1<br>end<br>#1 2 3 4 5 6 7 8 9 |

从这两个小程序可以看出： until 条件 = while not (条件)

四. for…in 循环

```
for 变量 in 对象

语句 1; 语句 2 ; 语句…

end
```

对象可以是数组，区间，集合…，看程序：

```
for i in 1..9

print i," "

end

#1 2 3 4 5 6 7 8 9
```

五． break ， next & redo & retry

在循环体内，如果遇到：

```
break ，跳出当层循环；
```

next ，忽略本次循环的剩余部分，开始下一次的循环；

redo ，重新开始循环，还是从这一次开始；

retry ，重头开始这个循环体。

## 六. 求 50 以内的素数。

```
#求 50 以内的素数
for i in 2..50 #50 以内
f=true #起始假定每个数都是素数
for p in 2...i #比自身小的正整数（1 和自身除外）
if i%p==0 #如果能整除
f=!f #那么这个数不是素数
break #并且跳出这层循环
end # if 结束
end #内层循环结束
print i," " if f #如果这个数保持起始假定，则打印
end #外层循环结束
#2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

## 七. times , upto , downto , each ,step

```
3.times { print "Hi!" } #Hi!Hi!Hi!
1.upto(9) {|i| print i if i<7 } #123456
9.downto(1){|i| print i if i<7 } #654321
```

```
(1..9).each {|i| print i if i<7} #123456

0.step(11,3) {|i| print i } #0369
```

- 异常与线程

与 Java 中的 try…catch…finally…throw 相对应，Ruby 中用 begin/end …rescue…ensure … raise 来处理异常，retry 可以用在 rescue 中。可以只用 rescue 或是 ensure，两者都使用时，rescue 必须在 ensure 前。

```
Begin

  #执行代码

Rescue

  #异常抓取

Ensure

  #程序出口

End
```

## 2.7 Watir 语法快览

**文本框：**

```
<INPUT id="email" name="_fmu.u._0.e" value="" />
方法 1： ie.text_field(:id,'email').set("文本内容")
方法 2： ie.text_field(:name, 'email').set("文本内容")
方法 3： ie.text_field(:name,"email").clear
```

**下拉框：**

<SELECT name="cert_no">

<OPTION value="身份证">身份证</OPTION>

</SELECT>

方法 1： ie.select_list(:name,"cert_no").select("身份证")

方法 2： ie.select_list(:name,"cert_no").clearSelection

**超链接:**

<a href = "http://www.google.cn/">google</a>

方法 1：ie.link(:text,"google").click

方法 2：ie.link(:url,"http://www.google.cn/").click

**复选框：**

<input type = "checkbox" name = "checkme" value = "1">

方法 1： ie.checkbox(:name,"checkme").set

方法 2： ie.checkbox(:name,"checkme").clear

方法 3： values = ie.checkbox(:name,"checkme").value

<input type = "checkbox" name = "checkme" value = "2">

#多个同名的复选框处理

方法 1： ie.checkbox(:name,"checkme","2").set

方法 2： ie.checkbox(:name,"checkme","2").clear

**单选框：**

<input type = "radio" name = "clickme" id = "1">

方法 1： ie.radio(:name, "clickme").set

方法 2： ie.radio(:name, "clickme").clear

**一般按钮：**

<input type = "button" name = "clickme" value = "Click Me">

方法 1： ie.button(:value, "Click Me").click

方法 2： ie.button(:name,"clickme").click

**submit 按钮：**

<form action = "submit" name = "submitform" method = "post">

<input type = "submit" value = "Submit"></input>

</form>

方法： ie.button(:value."Submit").click

**图片按钮：**

<form action ="submit" name = "doitform" method = "post">

<input type = "image" src = "images/doit.gif" name = "doit">

</form>

方法：ie.button(:name, "doit").click

**Form 中无按钮：**

<form action = "login" name = "loginform" method = "get">

<input name = "username" type = "text"></input>

</form>

方法 1：ie.form(:name,"loginform").submit

方法 2：ie.form(:action,"login").submit

**获取隐含对象值：**

<INPUT type=hidden value="您的 Email" name="field1">

方法：values = ie.hidden(:name,'field1').value

**获取窗口对象：**

方法 1： ie2 = Watir::IE.attach(:url,'http://www.google.cn/') #根据 URL 获取

方法 2： ie3 = Watir::IE.attach(:title,'Google') #根据窗口标题获取

方法 3： ie4 = Watir::IE.attach(:title, /google.cn/) #正则表达式匹配获取

**URL 编码：**

```
require 'cgi'

string = "URL 编码"

string = CGI::escape(string)

puts string                # 转换结果： URL%B1%E0%C2%EB
```

**URL 解码：**

```
require 'cgi'

string = "URL%BD%E2%C2%EB"

string = CGI::unescape(string)

puts string                # 转换结果：URL 解码
```

# 3. 框架介绍

```
------AutoTest
        |
        |-doc/
        |包含自动化脚本相关文档介绍
        |
        |-include/
        |包含所有基本方法(include/common/)
        |测试数据(include/img， include/datas)
        |页面元素程序(include/page/)
        |命名规范：必须小写开头，必须有后缀 axxxx_page.rb
        |包含点击 IE 弹出对话框的方法(include/util)
        |用来调用基本方法(include/all_includes.rb)
        |
        |-init/
        |包含配置文件(config.rb)
        |程序入口执行文件(setup.rb)
        |
        |-log/
        |包含程序运行时产生的所有日志文件
        |
        |-report/
        |测试结果产生的文档
```

|
|-testcase/
|所有的测试用例主程序
|命名规范：必须小写开头，必须有后缀 axxxx_testcase.rb
|
|-run_suites.rb
|运行所有测试用例的主程序

config.rb 文件的描述:

```
#==============================基本配置==========================

# 打开自动过滤运行 Pass 过的用例

$turn_on_case_check = false

# 打开删除 cooking 的功能

$kill_cooking = true

# 打开 email 发送测试报告功能

$need_email = false

# 打开生成报表功能

$created_report = true



#email 配置接收邮件人员

$email_list = ['bixiaochao@snda.com', 'majiangtao@snda.com']

#email 服务器的基本配置

$email_server = "psmtp.gawab.com"

$mail_user = "sdna_test@gawab.com"

$mail_pasw = "840331"
```

```
$email_server_port = "25"

#email 内容所需要的全局变量

$all_other_text = Array.new

$tol_email_datas = nil

#报表功能需要的全局变量

#{ 用例 1-->[状态，运行时间，内容], 用例 2-->[状态，运行时间，内容] }

$case_states = Hash.new

#{ 类名   --> {$case_states, $case_states}}

$all_case_states = Hash.new



#==============================测试站点配置=========================

# NEWPAY--银行卡充值页

$newpay_url = "http://test.wpay.sdo.com"

$login_url = "http://dev.pt.sdo.com:828/loginPT.asp"

$user_name = "autotest2009"

$user_password = "840331"



#WEB 数据库

$web_sql_site = "http://test.direct.sdo.com/WebSql/page/SelectDb.aspx"

$data_base_name = "192.168.169.4"

$data_base_user_name = "tester"

$data_base_user_psw = "tester"
```

```
#补发货地址
$post_bank_order_site = "http://61.172.247.16:8087/"
```

run_suites.rb 文件使用描述:

1、增加新的用例类;

2、脚本的总运行文件。

```
#在文件内增加新的测试用例，方式如下：
class RunSuites
    puts "Start Run Preferable Suites:"
    def self.suite
        suite = Test::Unit::TestSuite.new("Run AutoTest Suites:")
        suitr << 测试用例类名称.test
        #suite << BankOrderPageTestCase.test
        return suite
    end
end
```

测试用例类名称,需要在 testcase 文件夹内，找到对应的用例文件 xxxx_testcase.rb

```
$LOAD_PATH.unshift File.join(File.dirname(__FILE__), '..') if $0 == __FILE__
require 'init/setup'
require 'testcase/testcase_base'
```

```
class BankOrderPageTestCase < BaseTestCase

    def setup #所有用例的入口

      kill_all_cookie #删除 COOKIE 的方法

      @begin_time = Time.now #记录开始时间

    End



    def teardown

      $end_time = Time.now - @begin_time #记录结束时间

    End



    def self.test

        suite = Test::Unit::TestSuite.new('Test Case Suite')

        tests = [

        'bank_order_page_001'

        ]

        tests = check_testcase_run("#{self.name}", tests) #过滤掉上次 Pass 的用例

        tests.each do |test|

            suite << new(test)

        end

        return suite

    end
```

```
=begin
        测试用例，将描述放在这里，变量，用例的作用
=end
    def bank_order_page_001
          # 对此用例的操作进行简短描述,必须有值
          $content = "无登陆情况下，兴业银行下订单,充值量和积分显示正确"
        #实际要执行的用例主体
        # ..................
    end
    end
```

datas 文件夹下放入准备好的测试数据，如：

1、输入完成后，需要多按一个回车；

2、在用例内使用$datalog.get_data_by_key(用例编号名,"关键字")，如：

game_name = $datalog.get_data_by_key("bank_order_page_001", "game_name") #拿到之前定义的游戏名称

```
<?xml version='1.0' encoding="GB2312"?>
<datas>
<bank_order_page_001>
<game_name>热血传奇</game_name>
<area_name>传 奇 一 区</area_name>
```

```
<account_name>小超</account_name>

<account_type>游戏帐号</account_type>

<money>5.00</money>

<content>4 个元宝</content>

<integral>20</integral>

<bank_name>兴业银行</bank_name>

</bank_order_page_001>

<bank_order_page_002>

<game_name>热血传奇</game_name>

<area_name>传 奇 一 区</area_name>

<account_name>小超</account_name>

<account_type>游戏帐号</account_type>

<money>5.00</money>

<content>4 个元宝</content>

<integral>20</integral>

<bank_name>中国工商银行</bank_name>

</bank_order_page_002>
```

页面元素文件放在 include/page 文件夹下，必须是小写开头，必须带后缀，如 axxxxx_page.rb 。

```
require 'watir'   #必须引用'watir'框架
```

```
class BankOrderPage < BasePage

# 页面元素

# ============================================= #

    def url

        return "/BankOrder/DepositOrder.aspx?type=bank&sel_game="

    end

# 页面整体使用方法

#========================================== #

    # 选择输入充值产品名

    # name：字符类型

    def enter_game_name(name)

        pay_game_text.set(name)

        @ie.div(:id, 'SelectDiv').table(:id, 'listtb')[1][1].click

    end

End
```

注：report 文件夹不需要手动删除，每次运行前，会先删除旧的报告，再生产新的测试报告。

Common 文件夹下需要说明的文件：

1、 base_function.rb 包含所有基本公用方法；

2、 db_common.rb 包含操作 DB 的公用方法；

3、 email_common.rb 包含发 email 邮件的方法；

4、 logger_factory.rb 包含数据的读写和测试报告的生成方法；

5、 system.rb 包含 ruby 系统的基本方法的重写；

Page 文件夹下需要说明的文件：

1、 base_page.rb 包含基类的页面元素；

Unit 文件夹下需要说明文件：

1、 clickDialog.rb 描述点掉 windows 窗体的方法，需要用 base_function.rb 里的 check_dialog 方法调用；

Testcase 文件夹下需要说明文件：

1、 testcase_base.rb 重写了 Watir 里测试框架，增加日志。

# 4. 总结

夜色已深，霓虹消失，浮华散尽，一切都回归本来面目。我在思索，我们应该把肩膀降得低一些，再低一些，好让后面的人踏着向前。明天，太阳照常升起，一切新生力量又将开始新的一天。

我努力使这文档面向没有编程经验的人，书中许多内容来自网络，我已经没法一一历数出处，再次感谢你和他——生活在网络的人们，再次感谢这个网络世界，让我们共同进步。

2009 年 8 月 25 日