Batman 自动化测试框架全介绍

Batman 框架是我自己搭建的一个测试框架，很简单，也很简洁，给大家做一个介绍，希望后续能够帮助到大家，也非常欢迎大家和我一起来完善这个东西.

- 什么是 Batman

  batman 是一个基于 Watir 测试引擎的测试框架，使用 ruby 语言，主要进行针对 IE 浏览器的 web 前端自动化测试. 这个框架的主要任务就是进行 web 页面的自动化测试，能够完成的任务包括:

  1. 操作页面的元素，例如点击按钮，输入文字，点击图片等等;
  2. 执行 JS 脚本;
  3. 发送 http 请求，并且收集判断返回的内容;
  4. 处理 IE 弹窗，例如 alert 窗口(传说里的#32770 窗体);
  5. 验证界面内容;
  6. 记录测试内容步骤到数据库，并且生成 html 版本的报告;
  7. 发送邮件.

  基本上目前我们在前端测试所需要的操作我们都能够实现，后续还有更多的操作支持，参见后面的路线图的章节.

  地址: http://tool.baidu.com/p/Batman

- Batman 的系统要求

  Batman 是一款工作在 windows 下的测试框架，主要支持 winXP 和 win7，支持的浏览器包括 IE6-9, IE10 和 win8 尚未测试是否支持, 需要的运行环境为 ruby193, ruby2.0 尚未正式支持, 不过已经在评测当中了, 所需要的 gem 包扩:

  - Watir 2.0.3 (测试执行引擎)
  - Markaby (用于动态生成 html)
  - Sqlite3 (用于驱动测试数据库)
  - Rautomation (用于驱动 winForm)
  - Net/http (用于发送 http 请求并收集返回包)
  - net/smtp (用于发送邮件)
  - mailFactory (用于发送邮件)
  - zip/zipfilesystem(用于打包测试附件)
  - Uri (用于处理 url)
  - Base64 (用于编解码)
  - (用于截图)

- Batman 的测试规划
  - 每个测试用例是一个独立的文件

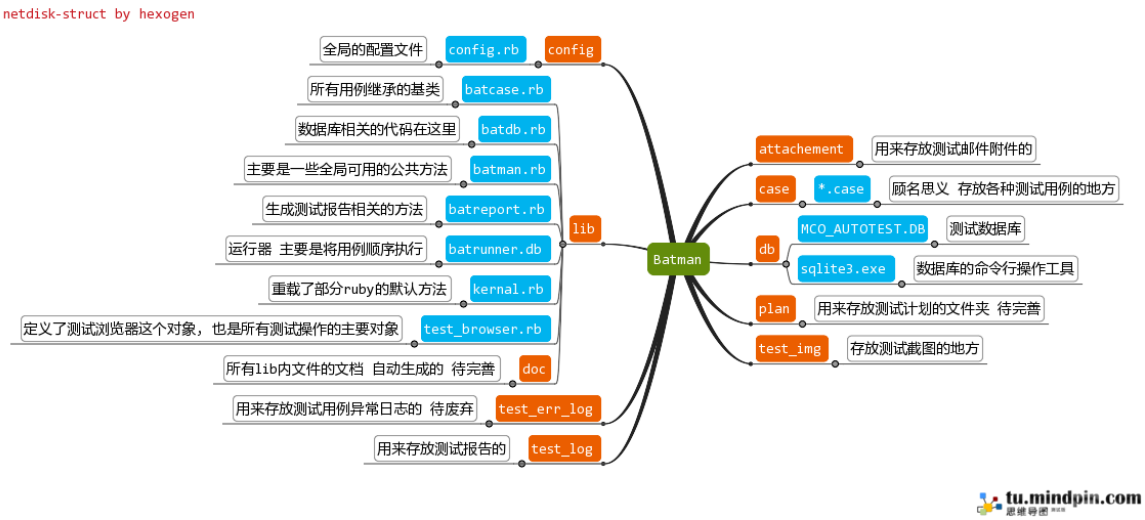
- 每个用例是一个单独的类
- 每个用例类含有三个固定的方法，自动顺序执行
- 每个测试用例为自满足，自我场景布造，自我回收
- 每个用例执行中允许自定义异常处理，原则上有异常终止当前用例的执行，开始下一个用例
- 测试用例执行前会注册到数据库，测试的步骤(包括异常)会写入数据库
- 全部测试用例执行完成后，根据数据库内容生成测试报告，然后根据用户配置，是否打包截图与发送邮件

详细的测试运行流程后面会更详细的说明

- Case 的结构和规约

  - 每个测试用例是一个类
  - 这个类的名字需要与所在的 rb 文件同名，大小写完全一致
  - 所有的 case 都继承自 BatCase 这个类
  - 每个 case 有且只有三个方法: setup， test， tear_down
  - 三个方法系统会按照顺序执行， 顺序是 setup-->test-->tear_down
  - 建议三个方法的规划为： setup 布造测试场景， test 进行测试， tear_down 进行各种资源回收
  - case 的基类中， 在 setup/test/tear_down 之前和之后 还有前置和后置方法，测试用户看不到也不需要进行操作

- Batman 的结构



- Batman 的运行流程

在默认的执行全部测试用例的情况下，执行测试的顺序:

1. 清理上次的日志
2. 扫描当前 case 目录下全部的用例，生成 case 列表
3. 注册本次运行到数据库
4. 注册所有的 case 到数据库
5. 开始顺序执行 case

i. 注册 case
ii. 执行 case 的测试方法的前置/后置方法(测试用户无感觉)
iii. 执行测试方法
iv. 注册 case 结果
v. 结束

6. case 执行完毕，注册测试信息到数据库
7. 生成测试报告
8. 生成测试邮件附件
9. 发送邮件

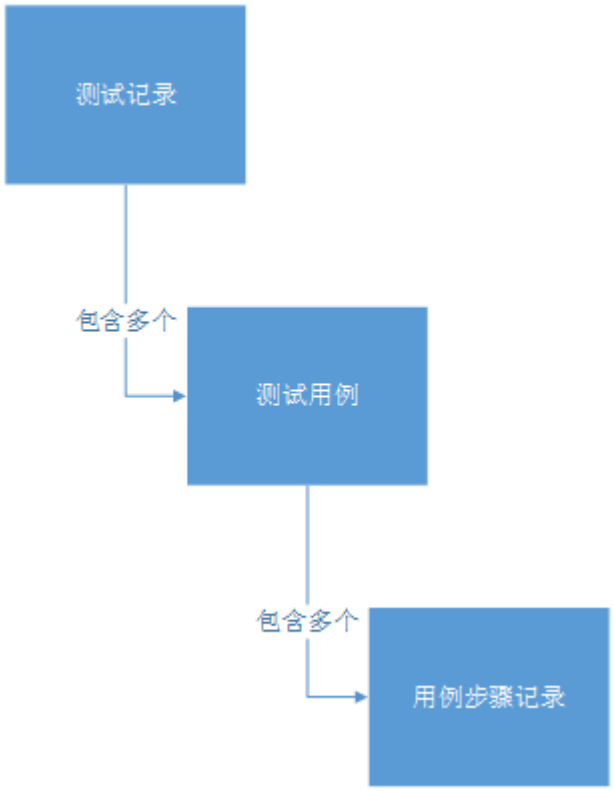
- 关键设计
  1. 测试用例的执行器
     其实很简单，测试用例本身是一个类，测试方法的入口就是用例类的构造函数，所以只需要实例化这个类，运行指定的方法就可以将这个用例运行了，运行器的原理就是将 case 列表中的类进行批量化的实例化，具体做法是利用 ruby 语言的动态增加方法的特点，代码的实现如下:

```
@case_list.each do |case_name|
        @total_code += "#{case_name.downcase}_instance = #{case_name}.new.run_test; "
end
```

     测试用例列表的信息被从上面的代码中取到了@total_code 中，再利用动态修改函数内容的特性加入到 runner 中:

```
run_code = <<-eval_end
def run_all mgr
        begin
                #{@total_code}
        rescue Exception => e
                puts e.class
                puts e
        end
end
eval_end
# puts run_code
Batman::BatRunner.class_eval  run_code     # add
```

     在这段代码中，我们利用@total_code 生成一个代码段 run_code，然后利用 class_eval 方法将这段 code 加入到类 BatRunner 中，运行这个方法就可以运行全部的测试用例了.
  2. 数据库设计
     数据库的设计模型很简单:



每次运行自动化测试都是一条测试记录，每次测试包含多个测试用例，每个用例拥有一个或者多个步骤，

数据库的设计比较简单，主要包含 3 张表:

test_log: 主要记录当前的测试记录，包括本次运行有多少 case， 成功多少， 失败多少等等.

test_case: 主要记录测试中包含的测试用例，包括所属的测试记录, case 步骤的数目，结果等等

case_log: 主要记录测试用例的执行步骤，包括用户输出的备注，结果等相关数据

- 如何写一个 case

  举个栗子给大家说说怎么写 case:

  首先创建一个新的 ruby 文件， 然后输入文件头， 都是统一的:

```
# encoding: utf-8
require File.dirname(__FILE__) + "/../lib/batcase.rb"
```

  然后创建一个类， 继承自 BatCase， 名字需要与文件名一直， 且大小写敏感.

```
class CLChangeViewType < Batman::BatCase
```

  然后创建三个方法， setup/test/tear_down， 这三个方法会依次执行

```
def setup
  # login the pan
  @wangpan = Batman::BatBrowser.new
  @wangpan.login_pan
  @wangpan.log "*****************************************"
  @wangpan.log "case CLChangeViewType started: check the CLChangeViewType function"
end
```

```
def test
```

```
def tear_down
```

在 setup 方法中，建议包含如下的操作:

新建浏览器

打开测试网站

登陆测试账号

部造测试场景

例如:

```
# login the pan
@wangpan = Batman::BatBrowser.new
@wangpan.login_pan
@wangpan.log "*****************************************"
@wangpan.log "case CLChangeViewType started: check the CLChangeViewType function"
```

在 test 方法中，建议包含主要的操作和验证内容

例如:

```
# step 1
# get the file_name
sleep 0.5
file_name = Array.new
(0..4).each do |rand_index|
  file_name << (@wangpan.get_text_with_index("class", "inline-file-col", rand_index)).to_s
end
# change the ViewType
# barCmdViewList
# barCmdViewSmall
@wangpan.click "id", "barCmdViewSmall"
# compare the file_name
file_name.each do |past_name|
  if @wangpan.assert_has_str(past_name) then
    @wangpan.log "find the file:#{past_name} in the page @ icon view"
  else
    @wangpan.log "can not find the file:#{past_name} in the page @ icon view, test fail."
    raise "can not find the file:#{past_name} in the page @ icon view, test fail."
  end
end
```

在 tear_down 方法中，建议包含如下操作:

清理测试痕迹(恢复数据等等)
退出测试账号
退出浏览器
清除 cookie
例如:

```
@wangpan.logout_pan
@wangpan.clear_cookie
@wangpan.log "clean cookie from tear_down"
@wangpan.log "case CLChangeViewType over."
@wangpan.close
```

- 如何定制一个 test_browser
  - 首先确定一个被测对象的类型:
    - 如果是无账号差别网页(比如大搜索), 需要建立数据池, 数据驱动测试
    - 如果是账号相关应用, 比如网盘, 需要建立登陆方法, 登出方法, 封装基本的测试方法(业务相关),然后在进行测试
    - 如果是多账号相关的, 比如多用户的公共软件(查话费或者什么的), 封装公共登陆组件, 然后也需要构造数据池
    - 如果是多账号多沟通的话, 比如微博等需要测试关注/被关注的 case, 可以实例化多个 test_browser 的实体进行互相操作
    - 如果确认系统的类别, 然后需要优先封装基础流程需要的方法, 后续如果有一个方法被需要超过三次, 则可以考虑进行公共化
    - 建议创建 gui_map
- 路线图

  这个测试框架还是大有可为的, 后续还有很多的想法会慢慢的实现, 现在想实现的内容还有下面这些:

  i. 支持数据库检查, 操作完后可以直接访问数据库进行校验
  ii. 主动点击, 通过确定控件的相对坐标, 移动鼠标, 进行主动点击, 目的是完成一些前端无法操作的内容
  iii. 组件 map, 自动生成界面组件的 map, 降低人工维护的成本
  iv. 多进程, 多个 case 一起执行 降低测试的时间消耗
  v. 独立进程监控 ie 弹窗, 主要是防止 js 的中断

- 安装与部署
  - 安装 ruby 1.9.3
  - 安装 devkit
  - 安装相关的 gem 包,其中 watir 的版本必须是 2.0.3
  - 拷贝 test_suite 到任意目录
  - 编写 case
  - 在 plan 目录下运行 ruby go.rb 来执行测试