密级：限制

编号：

**自动化测试操作指南**

目 录

[1 总则 1](#_Toc13577272)

[1.1 目的 1](#_Toc13577273)

[1.2 适用范围 1](#_Toc13577274)

[2 自动化测试框架 1](#_Toc13577275)

[3 框架规范细则 2](#_Toc13577278)

[3.1 页面层 2](#_Toc13577279)

[3.2 用例层 2](#_Toc13577280)

[3.3 测试数据 4](#_Toc13577281)

[3.4 配置数据 4](#_Toc13577282)

[3.5 测试报告 5](#_Toc13577283)

[3.6 运行截图 5](#_Toc13577284)

[3.7 运行日志 6](#_Toc13577285)

[4 命名规范 8](#_Toc13577286)

[5 脚本编写规范 8](#_Toc13577293)

[6 注释规范 9](#_Toc13577296)

[7 附则 9](#_Toc13577300)

自动化测试操作指南

# 总则

## 目的

### 本指南目的在于统一不同项目、系统的自动化测试实现，通过统一设计思路以及脚本、用例的编写细节，方便自动化测试脚本的管理和维护，提升自动化测试的质量和效率。

## 适用范围

### 进行自动化测试实现前，应分析评估项目是否适合自动化、哪些功能模块适合自动化，适合自动化的范围包括：

#### 项目周期长、项目特性相对稳定、变化不大。

#### 每次版本升级测试中，都需要回归测试项目已有功能。

#### 自动化投入成本在接受范围内。

# 自动化测试框架

## 自动化测试框架以page object模式为基础，分为2层：页面层、用例层，如图1，具体规范见第3节。



图 1

### 页面层：系统的每个页面单独为一个类，类中包含该页面的元素以及元素的基本操作、页面业务，供用例层调用；

### 针对客户端产品的特性，页面层可以改成插件层（下同），即元素按不同插件分类，一个插件类可理解为多个页面类的集合；

### 用例层：最终的运行用例，调用页面层提供的元素操作或业务逻辑组成用例；

## 其它输入数据（测试数据、配置数据）、输出数据（运行报告、截图、日志）、共用功能模块（读取测试数据、读取配置数据、输出报告、截图等）按统一格式和位置存放，具体规范见第3节。

# 框架规范细则

## 页面层

### 系统元素（按钮、文本框等）按页面划分，每个页面单独为一个类。

### 类变量为元素定位、类方法为元素操作（点击、输入文本等）或元素操作组成的业务逻辑，元素封装的意义在于，当元素定位发生改变时便于修改，业务逻辑封装的意义在于，减少业务频繁调用带来的代码冗余。

### 每个页面类继承自基页面类，基页面类对元素定位和操作进行了优化和二次封装，供页面类调用，基页面类为通用类，一般不需要修改。

## 用例层

### 用例类为自动化测试用例，按页面或者场景划分，用例类设计基于unittest测试框架，类函数为测试用例，按用例需要可添加初始化函数和收尾函数。

### 每个用例类继承自基用例类，基用例类用于初始化用例变量、封装用例通用函数，供用例类调用，基用例类继承自unittest.TestCase类。

### 用例函数通过调用组合页面层的元素操作和业务逻辑实现用例。

### 用例设计时，应尽可能能减少用例间的耦合关系，提高用例独立性，如果是有关联的脚本，应通过序号编排好执行顺序。

### 每个用例应有一个明确的断言内容，用于验证用例所测试的功能点，如不能直接判断的，应对用例运行结果进行截图并保存。

### 用例一般只做功能的正向逻辑验证。

## 测试数据

### 测试数据统一以csv格式存放于input/data目录下。

### 封装csv模块用于读取数据，存放于common目录下。

### 测试数据在用例基层base.py统一读取和初始化，方便用例调用。

## 配置数据

### 配置数据（测试地址等）统一以ini格式存放。

#配置文件config.ini

[url]

ip = https://

### 封装configparser模块用于读取数据，存放于common目录下。

import configparser

def read\_config(section, key):

"""

输入：section和键名

操作：返回config.ini里对应section下的对应键值

"""

config = configparser.ConfigParser()

config.read("./input/config/config.ini")

return config.get(section, key) #获取指定节点下key的值

### 在用例基层base.py统一读取和初始化，方便用例调用。

## 测试报告

### HTMLTestRunner模块主要用于输出测试报告，存放于common目录下。

### 在run.py中调用运行报告HTMLTestRunner模块运行用例集，将测试报告以html格式统一保存于output/report目录下。

with open('./output/report/' + reportname, 'wb') as fb:

runner = HTMLTestRunner(stream=fb, title="测试报告", description="用例执行情况", retry=1)

runner.run(suite)

## 运行截图

### 封装ImageGrab模块用于输出截图，存放于common目录下，运行截图统一保存于output/screenshots目录下。

from PIL import ImageGrab

from time import strftime

import os

def screenshot(picturename):

"""

输入：截图名称

操作：截图并将图片以对应的截图名称保存

"""

now = strftime("%Y-%m-%d") #获取当前时间

path = './output/screenshots/' + now + 'screenshots/'

if not os.path.exists(path):

os.mkdir(path)

im = ImageGrab.grab()

im.save(path + picturename + ".jpeg")

### 用例里调用函数进行截图并保存。

self.screenshot('xx')

## 运行日志

### 封装logging模块用于输出日志，存放于common目录下，运行日志统一保存于output/log目录下。

import logging

from time import strftime

def write\_log(level, message):

"""

输入：日志等级、日志内容

操作：按日志等级记录日志内容

"""

now = strftime("%Y-%m-%d") #获取当前时间

logname = './output/log/' + now + "log.log"

logging.basicConfig(

level=logging.DEBUG,

format='%(asctime)s : %(levelname)s %(message)s',

datefmt='%H:%M:%S',

filename=logname,

filemode='a')

# 输出日志级别

if level == 'debug':

logging.debug(message)

elif level == 'info':

logging.info(message)

elif level == 'warning':

logging.warning(message)

elif level == 'error':

logging.error(message)

elif level == 'critical':

logging.critical(message)

### 用例里调用函数进行日志记录。

self.write\_log('info', '用例1-添加xx完成')

框架具体规范可参考图2和图3。



图 2

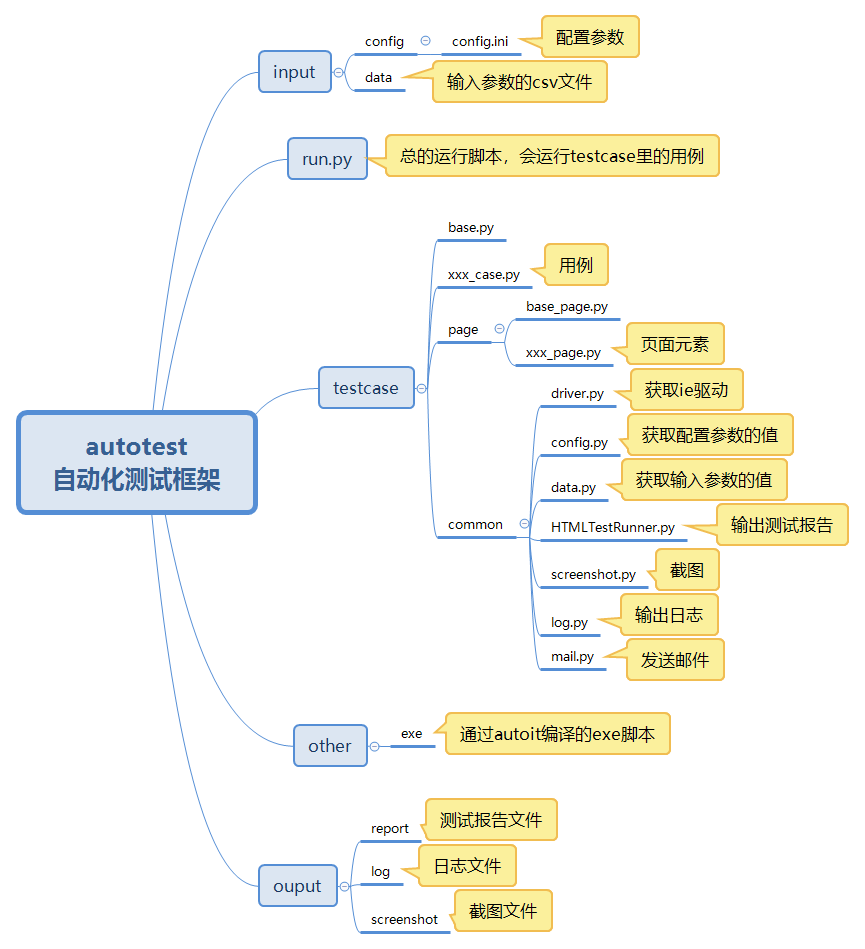


图 3

# 命名规范

## 页面类文件名以“\_page”为后缀，页面类名以“Page”为后缀，如：cert\_info\_page、InfoPage。

## 页面元素变量名以“\_type\_loc”为后缀，type用于区分元素类型，具体见表1，元素操作函数名以“\_type”为后缀，业务逻辑函数名结构应是动词+名词，动词小写，名词首字母大写，且能反应函数功能。

|  |  |
| --- | --- |
| type | 类型 |
| iframe | iframe |
| button | 按钮 |
| text | 文本输入框 |
| select | 多选框 |
| radio | 单选框 |

## 用例类文件名以“\_ case”为后缀，用例类名以“Case”为后缀，如：cert\_03\_business\_case、BusinessCase（证书业务系统-证书业务流程用例）。

## 测试用例函数名以“test\_”为前缀，如test\_cert\_ register（用例-新申请）。

## 常量的命名应全部用大写，使用"\_"作为单词间的分隔符，单词尽量使用全称，如SUCCESS=1。

## 变量、参数命名应简单明了，如果包括两个或两个以上的单词时，首单词字母小写，其他单词首字母大写，如identityType=1。

# 脚本编写规范

## 不同函数之间必须加上空行；在函数内部，变量声明块和实现块之间应使用空行间隔；实现块的内部，应通过空行来标识一个功能段。

## 严格执行缩进，变量声明块不缩进，实现块必须保证全部缩进；对于基本的控制结构来说，必须要有缩进，如 IF、DO、FOR、WHILE块。

# 注释规范

## 脚本模块应注释脚本编写信息，包括脚本编写人、编写时间、脚本描述。

"""

脚本信息：登录页面

编写人：XXX

编写时间：2019-7-9

"""

## 类变量应注释该变量的含义。

login\_button\_loc = (By.CLASS\_NAME, 'bodySiginBtn') #登录按钮

## 类函数应注释该函数的功能，包括输入参数的含义以及具体执行的操作。

"""

输入：

操作："""

# 附则

本文件自公布之日起施行，由测试组负责解释。