2**Web端UI分层自动化测试框架**

因Web端UI自动化和移动端的分层及不断的抽象封装思路类似，就不在赘述了，直接贴分层介绍：

基于Selenium框架运用python语言以及unittest单元测试框架，搭建的Web端的UI自动化框架如下：

WebAuto/:config层: 存放配置文件以及测试数据，把所有的项目的配置均放在这里，用python支持较好的配置文件格式如ini等进行配置。

实现配置和数据与代码分离。

driver层: 存放所需的驱动，如chromedriver、iedriverserver等。

log层: 存放生成的日志文件，包括运行日志Runtime.log和错误日志Error.log等

report层: 存放程序运行生成的html格式报告

screenshot: 存放测试用到的图片以及测试时用例失败截图

src:源码层

common层: 框架级公用方法库

chche.py: 缓存

confparser.py: 配置文件解析器,读取配置文件数据类

dbsever.py: 数据库操作公用类

emailsever.py: 发送邮件服务封装公用类

initdriver.py: 初始化driver类

log.py: 日志记录公用类

...

(如果还有框架级别的公用方法，还可以在该层封装成类，通过面向对象的方式调用即可)

functions层: 用例级公用方法库(元素操作公用方法封装，基于PageObject模式对控件公用方法封装，常用业务操作封装)

baseaction.py: 封装元素常用操作的一些公用方法

login.py: 登录操作

...

(该层主要是封装用例层面的公用方法，常用的操作步骤，针对PageObject思想对不同类型的页面控件元素的操作封装等)

testcase层: 测试用例层

basecase.py: 测试用例基础类

testcase1.py: 测试用例1

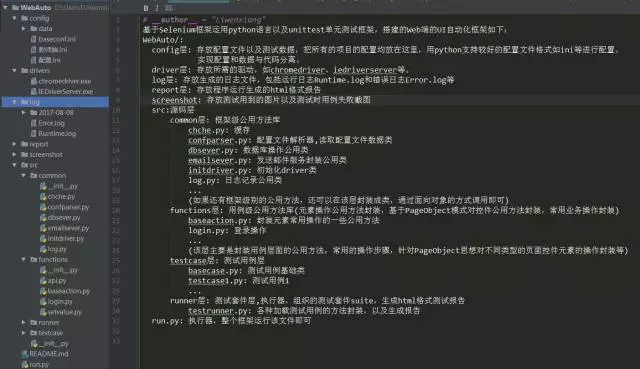
...

runner层: 测试套件层,执行器，组织的测试套件suite，生成html格式测试报告

testrunner.py: 各种加载测试用例的方法封装，以及生成报告

run.py: 执行器，整个框架运行该文件即可

下面是笔者搭建的一个小框架，仅供参考：



3**UI自动化基于Page Object模式设计思想**

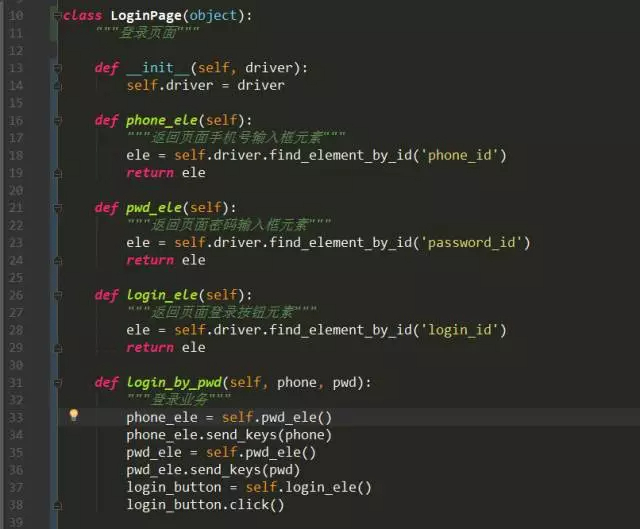
对于很多从手工测试转入自动化测试的朋友来说，把手工测试的功能，逐渐用代码实现自动化，发现其代码并没有什么模式可遵循，在加上UI页面的变动，维护起来也相对麻烦，而Selenium官网推荐了一种自动化构建模式，即Page Object设计模式，将元素操作与用例实现隔离开来，增加用例层的可读性，减少元素属性变化带来的测试用例重构工作，使得用例维护更加容易。下面说下笔者对该模式的理解。

**1. 基于页面的Page Object设计模式**

该设计模式是将每个测试页面抽象成一个页面对象类，把该页面中的元素定位、元素操作、业务流程等都封装在该类的方法中，编写用例时，直接以面向对象的思想调用该页面类中方法，避免了测试人员在用例中直接操作页面元素。

以登录页面为例：

第一步，将该页面单独封装一个页面类LoginPage，包含该页面内元素的定位和操作等，代码如下：



第二步，调用LoginPage页面类来编写用例



**2. 基于页面控件的Page Object设计模式**

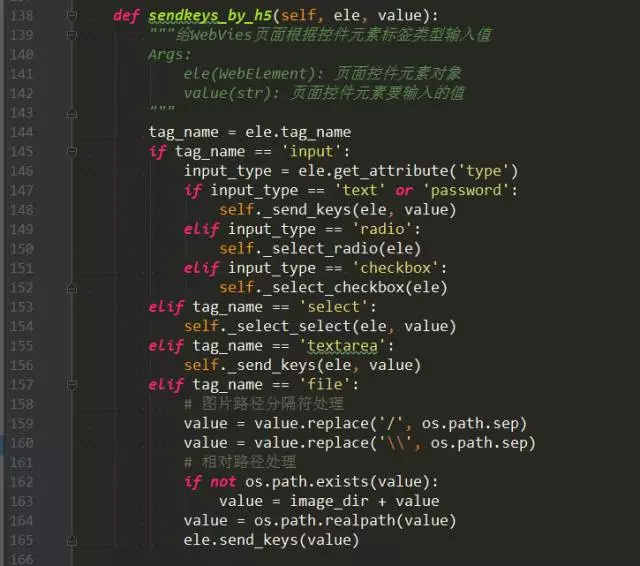
该设计模式是将测试页面中的控件抽象成不同的类，把控件中元素的定位、元素操作、元素行为等封装在类方法中，编写用例时，同样是以面向对象的思想调用该控件类中方法。比如页面中常用的控件包括元素输入框、导航栏、分页、筛选框等等，然后把元素输入、导航、分页、筛选等封装成类对象，然后根据元素具有的属性封装在类方法中，即分页的类中包含跳转上一页，下一页，输入页码跳转等方法，导航类中有跳转导航页的方法等等。

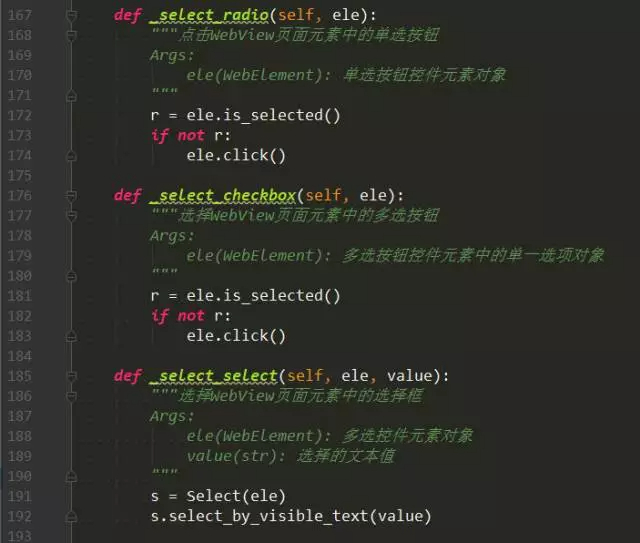
下面以该页面元素输入控件举例：



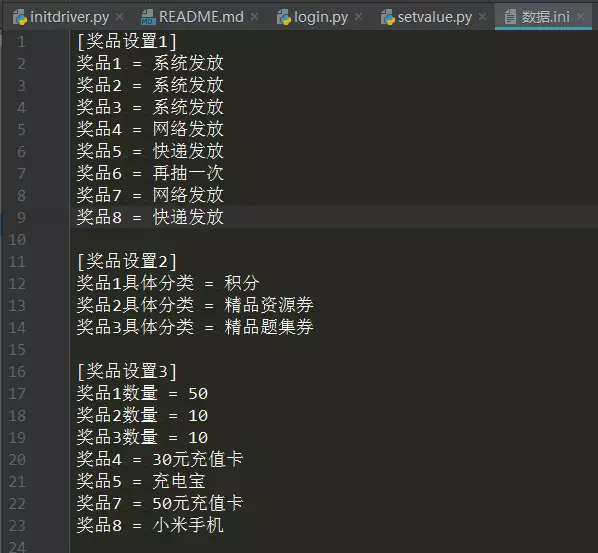
第一步，将元素输入控件封装，笔者封装的代码可能还需要大家熟悉下html的知识，我是先定位到元素，然后根据元素的输入框类型，比如input（text，password，radio，checkbox等）、select、textarea、file等，然后输入值，思路代码如下：

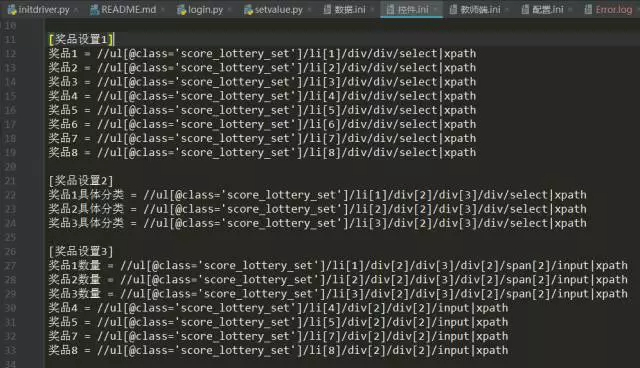






第二步，在用例实现中调用该方法，把输入的数据放在ini文件中，代码如下







用例中输入奖品信息只需要调用setvalues方法即可，不需要重复的去sendkeys，这样是不是清晰很多，同时所有页面的输入值操作都可以调用该控件类。

**3. 两种方法的比较**

第一种，是将每个页面看做一个页面对象，而当页面多了，每个页面中多会有重复的元素及操作，这样就必须在每个页面对象类中写一遍，相对代码量大，重用性低，相对不易维护。

第二种，是将页面不同类型的控件看做一个页面对象，因为页面并不是最小的UI单元，控件才是，当页面通用的元素变化后，只需要修改对应的控件类即可，而不用所有页面都去修改，这样岂不是更易维护。

当然选用哪种模式还需要结合产品具体情况，笔者推荐大家使用第二种基于控件的Page Object模式。