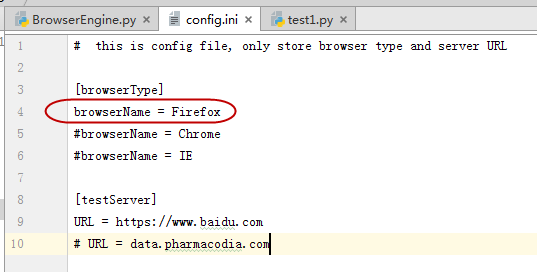
**基于Python的浏览器引擎类的创建和测试调用**

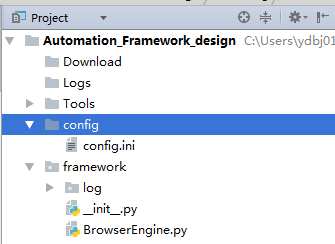
1. 什么是 Cross Browser Testing

在web UI 测试中，一般是要在不同浏览器上做测试，这个叫兼容性测试，在国外，一般叫Cross Browser Testing. 我们在设计自动化测试框架的时候，一般要考虑支持不同的浏览器，当然，主要考虑在Firefox和Chrome上做自动化测试，如果你的环境是windows，IE兼容性和不稳定性，优先级不高，iOS上的safria浏览器只有当你开发环境是mac的时候才考虑。

所以，前面文章我们介绍了python如何读取配置文件config.ini中的数据。我们这样来设计，在框架里写一个浏览器引擎类（BrowserEngine.py）, 这个类第一去解析和读取config.ini中的浏览器类型和测试服务器URL。当浏览器是Firefox，那么就调用方法，打开Firefox去测试，反之，当用户切换到Chrome就去打开Chrome测试。



1. Python 下浏览器引擎类的实现
   1. 先在项目下新建一个python package ： framework
   2. 在framework下新建一个BrowserEngqine.py



* 1. BrowerEngqine.py完整代码

*# -\*- coding:utf-8 -\*-***import** ConfigParser  
**import** os.path  
**from** selenium **import** webdriver  
  
  
**"""  
 \_\_author\_\_ = 'yuangen'  
 这里创建一个浏览器引擎类，用来读取配置文件中的不同浏览器类型，还有一些浏览器Profile设置  
"""  
  
class** BrowserEngine(object):  
  
 dir = os.path.dirname(os.getcwd())  
 chrome\_driver\_path = dir + **'\Tools\chromedriver.exe'** ie\_driver\_path = dir + **'\Tools\IEDriverServer.exe'  
  
 def** \_\_init\_\_(self, driver):  
 self.driver = driver  
  
 **def** get\_driver(self, driver):  
 config = ConfigParser.ConfigParser()  
 file\_path = os.path.dirname(os.getcwd()) + **'/config/config.ini'** config.read(file\_path)  
  
 browser = config.get(**"browserType"**, **"browserName"**)  
 url = config.get(**"testServer"**, **"URL"**)  
  
 **if** browser == **"Firefox"**:  
 driver = webdriver.Firefox()  
 **elif** browser == **"Chrome"**:  
 driver = webdriver.Chrome(self.chrome\_driver\_path)  
 **elif** browser == **"IE"**:  
 driver = webdriver.Ie(self.ie\_driver\_path)  
  
 driver.get(url)  
 driver.maximize\_window()  
 driver.implicitly\_wait(10)  
 **return** driver  
  
 **def** quit(self):  
 self.driver.quit()

代码解释：

1. 代码很简单，就是一个if-elif语句，来判断使用不同浏览器，从而初始化不同浏览器的webdrivere的实例driver.
2. Chrome 和 IE 浏览器需要下载不同driver.exe来支持，需要把这两个插件的地址，告诉IE和Chrome初始化加载这两个插件，否则会报错，由于selenium2.0，google原生支持firefox，到selenium3.0，firefox推出了原生的firefox的浏览器geckodriver.exe，需要把这个文件放到Firefox安装目录，或者放在python的安装根目录，就相当于把geckodriver.exe添加到了path。
3. 代码os.getcwd()是获取当前路径，os.path.dirname(os.getcwd())是获取项目路径。
4. 测试引擎类中方法的调用

浏览器引擎类写好了，我们需要通过单元测试去检测下写的方法是否工作。之前在java+selenium系列，使用testNG，或者jnuit。在这里，我们引入Python风格的unittest单元测试框架。

一开始，我也遇到了各种问题，在调用BrowserEngine.py中的方法总署报错。主要的问题就是，没有弄明白和了解unittest的工作原理和写法。

在testsuites包下创建一个test1.py的文件，代码如下：

*# coding=utf-8***import** unittest  
**from** selenium **import** webdriver  
**from** pageobject.Page **import** HomePage  
**from** framework.BrowserEngine **import** BrowserEngine  
**from** pageobject.pyse **import** \*  
**import** time  
  
**class** TestBadu(unittest.TestCase):  
  
 **def** setUp(self):  
 browser = BrowserEngine(self)  
 self.driver = browser.get\_driver(self)  
 **print** self.driver  
  
 **def** test\_type(self):  
 home\_page = HomePage(self.driver)  
 **print** self.driver  
 home\_page.type\_filter(**"selenium"**)  
 home\_page.click\_submit\_btn()  
 time.sleep(4)  
 **print** self.driver.title  
 **assert** self.driver.title == **u'selenium\_百度搜索'  
  
 def** tearDown(self):  
 self.driver.quit()

通过，在config.ini切换不同浏览器类型和测试地址，发现这个方法调用没有问题，可以正常启动浏览器和打开测试服务器地址。

代码解释1：

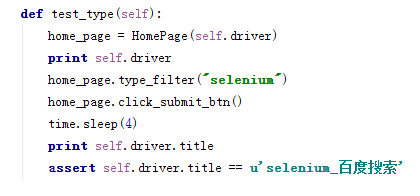
**def** setUp(self):  
 **pass  
def** tearDown(self):  
 **pass**

这两个代码段，类似testNG中的 @beforeClass 和@AfterClass，以后我们可以把初始化的代码写到setUp(self)中去，关闭浏览器操作写在tearDown(self)

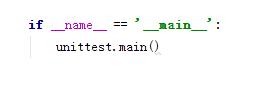
代码解释1：

上面的两个打印print self.driver，只是想证明浏览器引擎类调用的driver和测试脚本页面中的driver是否匹配。

注意：所有测试用例的方法名称都要test开头



只有test开头的才默认是测试用例部分。



这个是启动机制，利用unittest中main()去启动测试用例。Unittest这个框架的基本使用就介绍到这里。