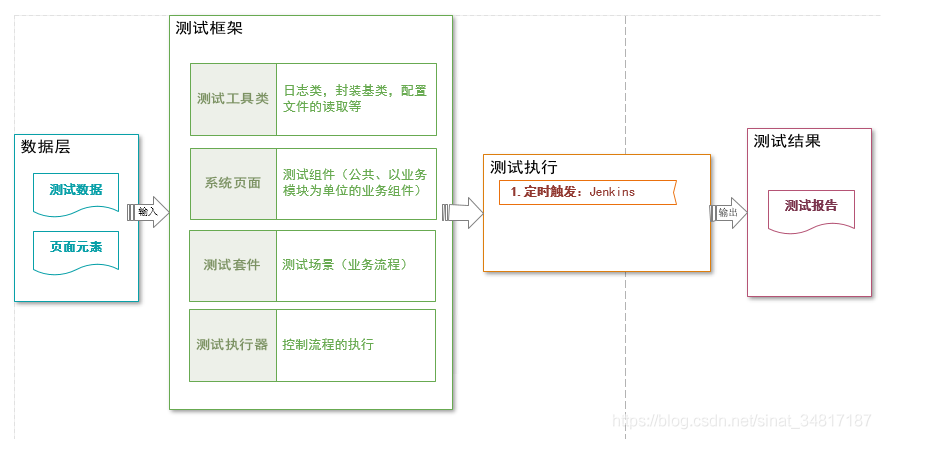
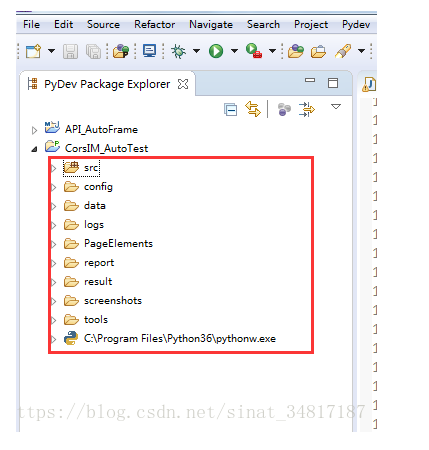
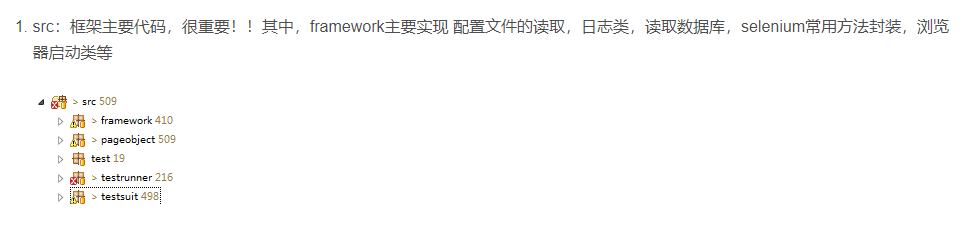
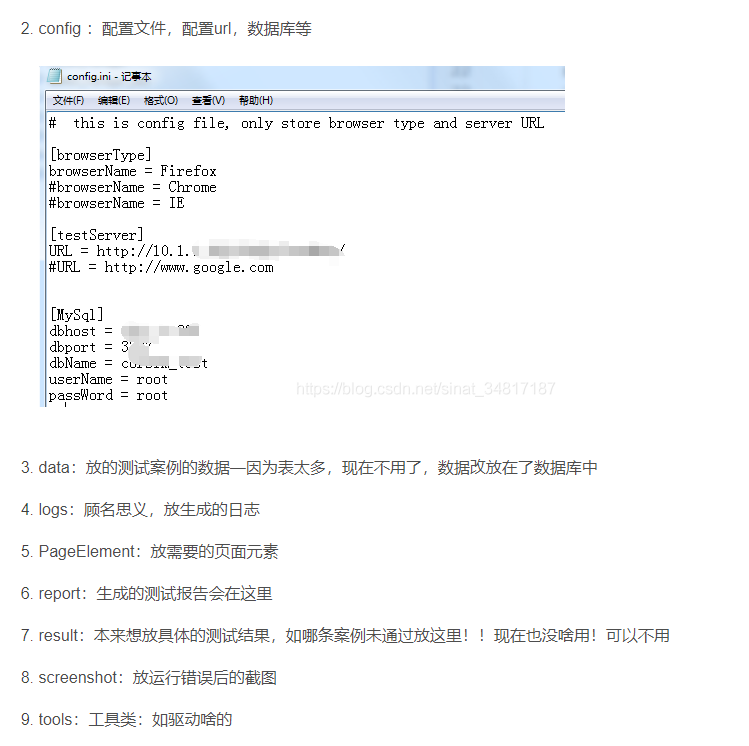
# source\_code

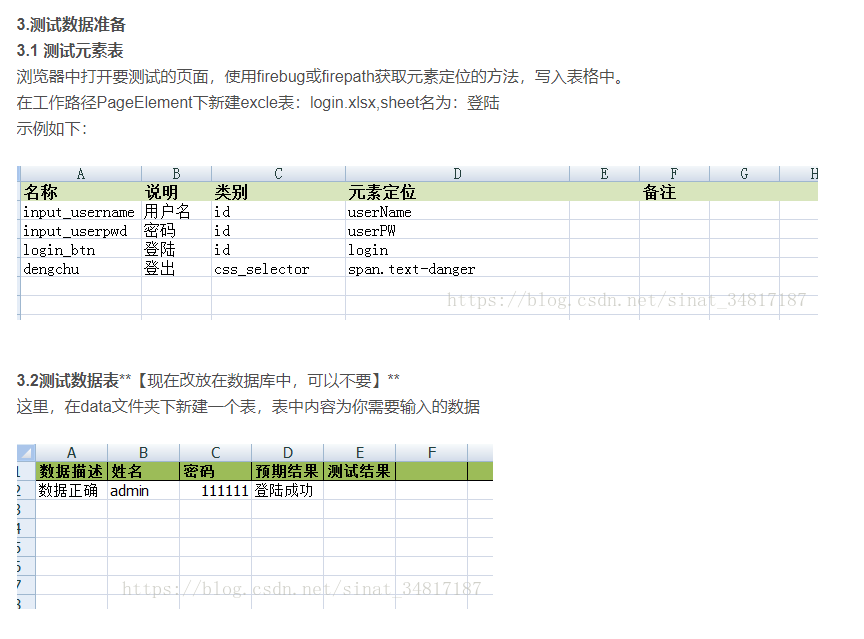
<https://github.com/chenjinwen-1994/framework_demo>





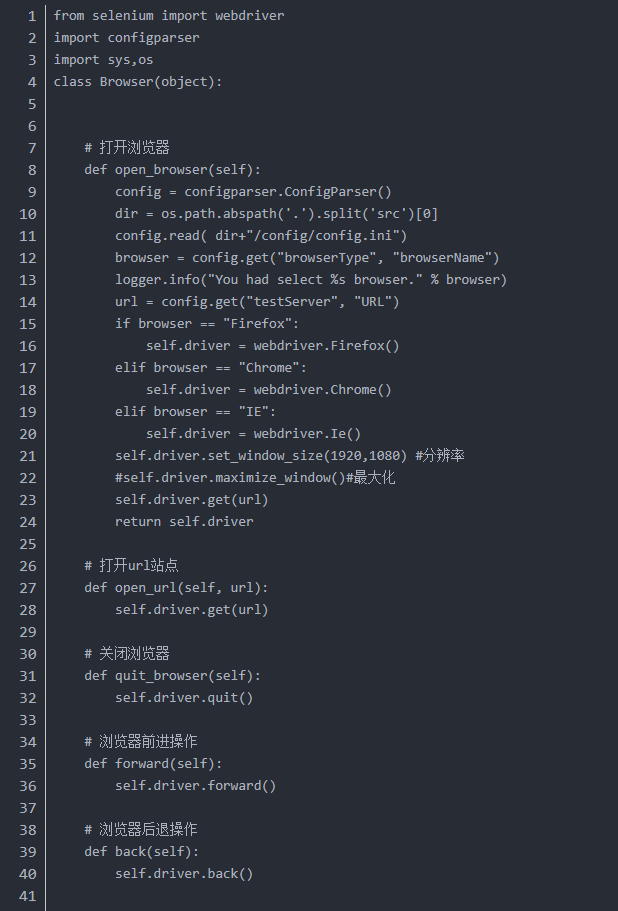


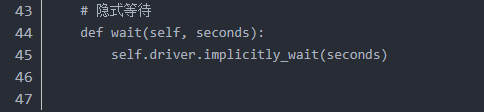






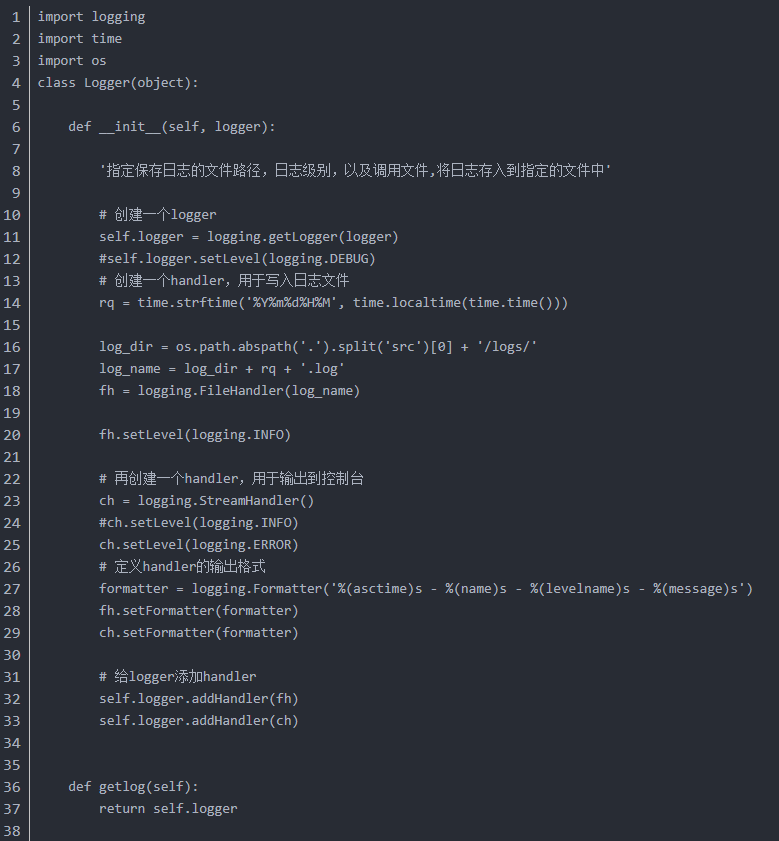






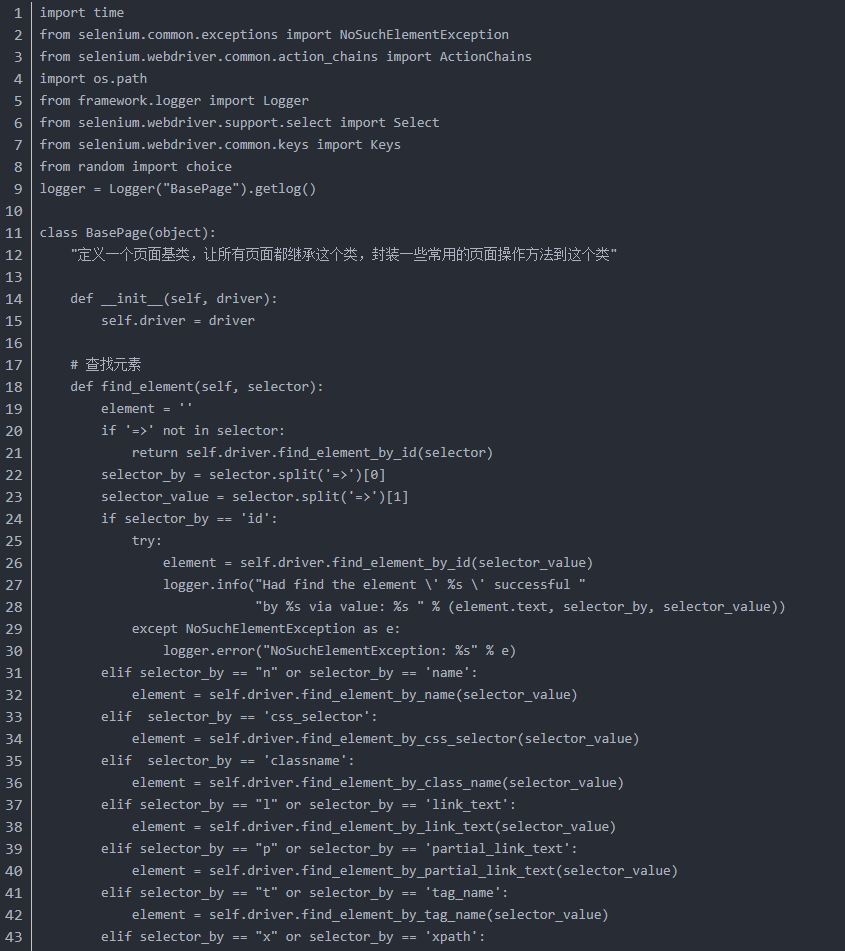
from selenium import webdriver  
import configparser  
import sys, os  
  
  
class Browser(object):  
  
 # 打开浏览器  
 def open\_browser(self):  
 config = configparser.ConfigParser()  
 dir = os.path.abspath('.').split('src')[0]  
 config.read(dir + "/config/config.ini")  
 browser = config.get("browserType", "browserName")  
 logger.info("You had select %s browser." % browser)  
 url = config.get("testServer", "URL")  
 if browser == "Firefox":  
 self.driver = webdriver.Firefox()  
 elif browser == "Chrome":  
 self.driver = webdriver.Chrome()  
 elif browser == "IE":  
 self.driver = webdriver.Ie()  
 self.driver.set\_window\_size(1920, 1080) # 分辨率  
 # self.driver.maximize\_window()#最大化  
 self.driver.get(url)  
 return self.driver  
  
 # 打开url站点  
  
 def open\_url(self, url):  
 self.driver.get(url)  
  
 # 关闭浏览器   
  
 def quit\_browser(self):  
 self.driver.quit()  
  
 # 浏览器前进操作  
 def forward(self):  
 self.driver.forward()  
  
 # 浏览器后退操作  
 def back(self):  
 self.driver.back()  
  
 # 隐式等待  
 def wait(self, seconds):  
 self.driver.implicitly\_wait(seconds)

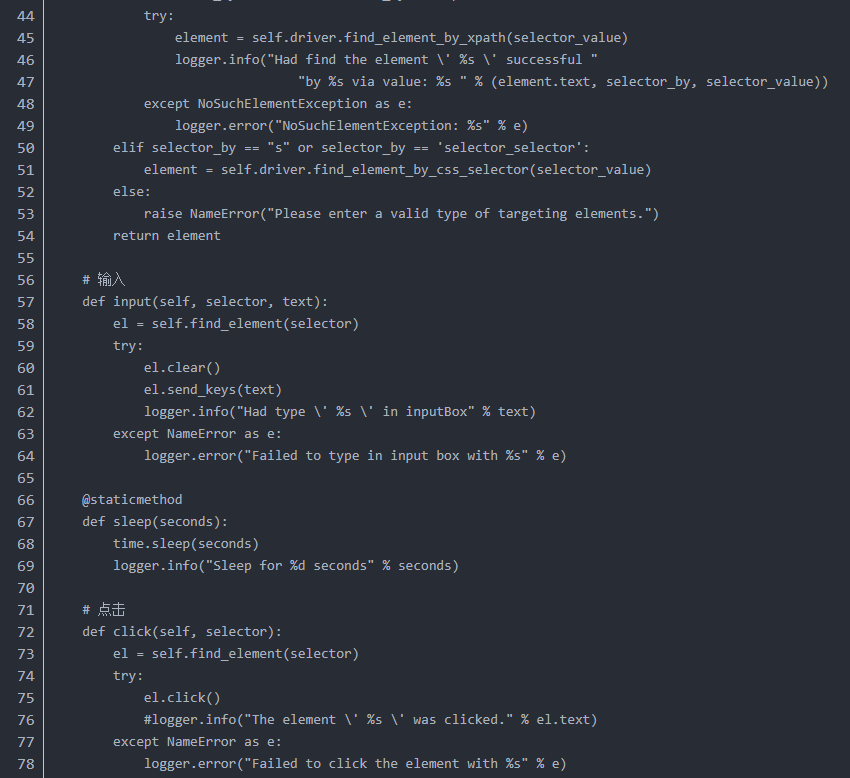
**4.3 框架之-----日志配置 logger类**

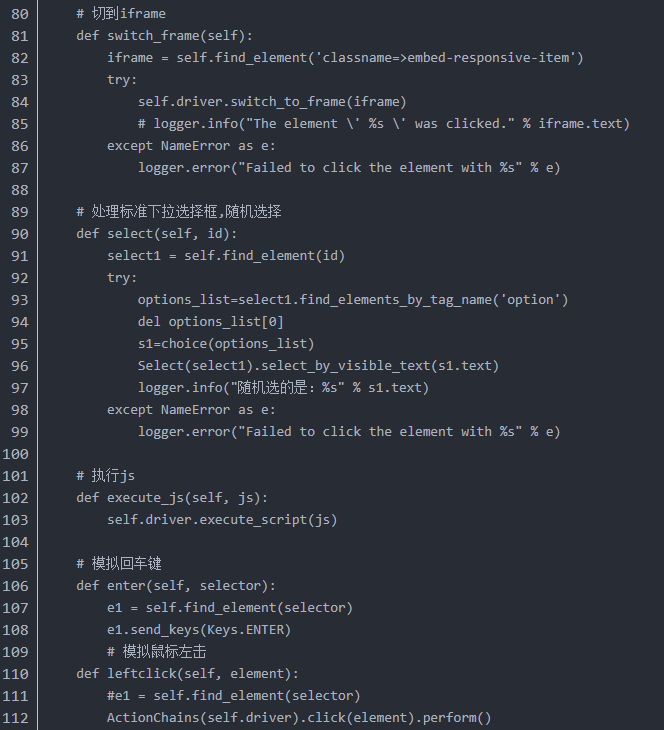


import logging  
import time  
import os  
class Logger(object):  
  
 def \_\_init\_\_(self, logger):  
 *'指定保存日志的文件路径，日志级别，以及调用文件,将日志存入到指定的文件中'* # 创建一个logger  
 self.logger = logging.getLogger(logger)  
 # self.logger.setLevel(logging.DEBUG)   
 # 创建一个handler，用于写入日志文件  
 rq = time.strftime('%Y%m%d%H%M', time.localtime(time.time()))  
  
 log\_dir = os.path.abspath('.').split('src')[0] + '/logs/'  
 log\_name = log\_dir + rq + '.log'  
 fh = logging.FileHandler(log\_name)  
  
 fh.setLevel(logging.INFO)  
  
 # 再创建一个handler，用于输出到控制台  
 ch = logging.StreamHandler()  
 # ch.setLevel(logging.INFO)   
 ch.setLevel(logging.ERROR)  
 # 定义handler的输出格式  
 formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')  
 fh.setFormatter(formatter)  
 ch.setFormatter(formatter)  
  
 # 给logger添加handler  
 self.logger.addHandler(fh)  
 self.logger.addHandler(ch)  
  
 def getlog(self):  
 return self.logger











import time  
from selenium.common.exceptions import NoSuchElementException  
from selenium.webdriver.common.action\_chains import ActionChains  
import os.path  
from framework.logger import Logger  
from selenium.webdriver.support.select import Select  
from selenium.webdriver.common.keys import Keys  
from random import choice  
  
logger = Logger("BasePage").getlog()  
  
  
class BasePage(object):  
 *"定义一个页面基类，让所有页面都继承这个类，封装一些常用的页面操作方法到这个类"* def \_\_init\_\_(self, driver):  
 self.driver = driver  
  
 # 查找元素  
 def find\_element(self, selector):  
 element = ''  
 if '=>' not in selector:  
 return self.driver.find\_element\_by\_id(selector)  
 selector\_by = selector.split('=>')[0]  
 selector\_value = selector.split('=>')[1]  
 if selector\_by == 'id':  
 try:  
 element = self.driver.find\_element\_by\_id(selector\_value)  
 logger.info("Had find the element \' %s \' successful "  
 "by %s via value: %s " % (element.text, selector\_by, selector\_value))  
 except NoSuchElementException as e:  
 logger.error("NoSuchElementException: %s" % e)  
 elif selector\_by == "n" or selector\_by == 'name':  
 element = self.driver.find\_element\_by\_name(selector\_value)  
 elif selector\_by == 'css\_selector':  
 element = self.driver.find\_element\_by\_css\_selector(selector\_value)  
 elif selector\_by == 'classname':  
 element = self.driver.find\_element\_by\_class\_name(selector\_value)  
 elif selector\_by == "l" or selector\_by == 'link\_text':  
 element = self.driver.find\_element\_by\_link\_text(selector\_value)  
 elif selector\_by == "p" or selector\_by == 'partial\_link\_text':  
 element = self.driver.find\_element\_by\_partial\_link\_text(selector\_value)  
 elif selector\_by == "t" or selector\_by == 'tag\_name':  
 element = self.driver.find\_element\_by\_tag\_name(selector\_value)  
 elif selector\_by == "x" or selector\_by == 'xpath':  
 try:  
 element = self.driver.find\_element\_by\_xpath(selector\_value)  
 logger.info("Had find the element \' %s \' successful "  
 "by %s via value: %s " % (element.text, selector\_by, selector\_value))  
 except NoSuchElementException as e:  
 logger.error("NoSuchElementException: %s" % e)  
 elif selector\_by == "s" or selector\_by == 'selector\_selector':  
 element = self.driver.find\_element\_by\_css\_selector(selector\_value)  
 else:  
 raise NameError("Please enter a valid type of targeting elements.")  
 return element  
  
 # 输入  
 def input(self, selector, text):  
 el = self.find\_element(selector)  
 try:  
 el.clear()  
 el.send\_keys(text)  
 logger.info("Had type \' %s \' in inputBox" % text)  
 except NameError as e:  
 logger.error("Failed to type in input box with %s" % e)  
  
 @staticmethod  
 def sleep(seconds):  
 time.sleep(seconds)  
 logger.info("Sleep for %d seconds" % seconds)  
  
 # 点击  
 def click(self, selector):  
 el = self.find\_element(selector)  
 try:  
 el.click()  
 # logger.info("The element \' %s \' was clicked." % el.text)   
 except NameError as e:  
 logger.error("Failed to click the element with %s" % e)  
  
 # 切到iframe  
  
 def switch\_frame(self):  
 iframe = self.find\_element('classname=>embed-responsive-item')  
 try:  
 self.driver.switch\_to\_frame(iframe)  
 # logger.info("The element \' %s \' was clicked." % iframe.text)   
 except NameError as e:  
 logger.error("Failed to click the element with %s" % e)  
  
 # 处理标准下拉选择框,随机选择  
  
 def select(self, id):  
 select1 = self.find\_element(id)  
 try:  
 options\_list = select1.find\_elements\_by\_tag\_name('option')  
 del options\_list[0]  
 s1 = choice(options\_list)  
 Select(select1).select\_by\_visible\_text(s1.text)  
 logger.info("随机选的是：%s" % s1.text)  
 except NameError as e:  
 logger.error("Failed to click the element with %s" % e)  
  
 # 执行js  
  
 def execute\_js(self, js):  
 self.driver.execute\_script(js)  
  
 # 模拟回车键  
 def enter(self, selector):  
 e1 = self.find\_element(selector)  
 e1.send\_keys(Keys.ENTER)  
 # 模拟鼠标左击  
  
 def leftclick(self, element):  
 # e1 = self.find\_element(selector)  
 ActionChains(self.driver).click(element).perform()  
  
 # 截图，保存在根目录下的screenshots   
 def take\_screenshot(self):  
 screen\_dir = os.path.dirname(os.path.abspath('../..')) + '/screenshots/'  
 rq = time.strftime('%Y%m%d%H%M%S', time.localtime(time.time()))  
 screen\_name = screen\_dir + rq + '.png'  
 try:  
 self.driver.get\_screenshot\_as\_file(screen\_name)  
 logger.info("Had take screenshot and saved!")  
 except Exception as e:  
 logger.error("Failed to take screenshot!", format(e))  
  
 def isElementExist(self, xpath):  
 flag = True  
 driver = self.driver  
 try:  
 driver.find\_element\_by\_xpath(xpath)  
 return flag  
 except:  
 flag = False  
 return flag

