# 功能脚本模块化

对自动化测试脚本进行功能模块化，不仅可以降低脚本维护的成本，而且方便编写组装不同的用例。功能模块化就是把测试脚本按照系统的功能进行拆分写成多个方法或者CLASS文件，比如实例化浏览器与打开系统网址、登陆功能等，甚至一些元素的处理方法也可以进行模块化。例如实例化浏览器与打开系统网址的模块封装，封装好之后，实例化浏览器与打开只需要调用Browser()的browser()方法，参数为浏览器名称以及访问的地址。

from selenium import webdriver  
from time import sleep  
class Browser():  
 def browser(self,browser,url):  
 if browser=='Ie':  
 self.browser=webdriver.Ie()  
 elif browser=='FireFox':  
 self.browser=webdriver.Firefox()  
 elif browser=='Chrome':  
 self.browser=webdriver.Chrome()  
 else:  
 raise ValueError('输入的浏览器不存在！')  
 browser=self.browser  
 browser.maximize\_window()  
 browser.implicitly\_wait(5)  
 browser.get(url)  
 sleep(3)  
 return browser

# 测试脚本与测试数据分离

为了对软件进行全面性的测试，往往需要对不同的数据进行测试。自动化测试中，一般把测试脚本与测试数据剥离，让测试脚本从文件或者数据库中读取测试数据。这样不仅能测试大量的数据，而且使得脚本更加简约，脚本维护也更加方便。

## 文本文件中读取测试数据

从文本文件导入数据使用python的open方法，读取数据后主要重点是对数据进行加工操作，以便提供给测试脚本使用。特别是要灵活使用切片函数split()与除头去尾函数strip()，如果数据是键值对类型，需要进行字典化。

from 进阶.封装登录退出功能.Browser import Browser  
from 进阶.封装登录退出功能.Login import Login  
from 进阶.封装登录退出功能.Logout import Logout  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 测试数据可以通过文本进行读取。  
 with open(r'D:\自动化测试\browser.txt','r',encoding='utf-8') as file:  
 data=file.read().split(',')  
 print(data)  
 with open(r'D:\自动化测试\user.txt','r',encoding='utf-8') as file:  
 users=file.readlines()  
 dict={}  
 for user in users:  
 dict.setdefault(user.strip().split(',')[0],user.strip().split(',')[1])  
 print(dict)  
 for key in dict.keys():  
 browser = Browser().browser(data[0], data[1])  
 Login().login(browser,key,dict[key])  
 Logout().logout(browser)

## Excel文件中读取数据

读取Excel文件中的数据需要借助第三方模块xlrd，xlrd提供了以下常用的方法对Excel文件进行操作：

1. Excel=xlrd.open\_workbook(filename,..)：打开一个Excel表格。
2. Sheet=Excel.sheets()[n]：通过sheets的索引获取sheet。
3. Sheet=Excel.sheet\_by\_index(n)：通过sheets的索引获取sheet。
4. Sheet=Excel.sheet\_by\_name(sheetname)：通过sheet的name获取sheet。
5. row=sheet.row\_values(n)：获取sheet第n+1行数据。
6. col=sheet.col\_values(n)：获取sheet第n+1列数据。
7. row(n).value：获取这一行中第n+1列数据。
8. col(n).value：获取这一列中第n+1行数据。
9. Sheet.cell(n,m).value：获取第n+1行，m+1列数据。
10. Sheet.nrows：工作表的行数。
11. Sheet.ncols：工作表的列数。

import xlrd  
# 实例化Excel对象。  
excel=xlrd.open\_workbook(r'D:\自动化测试\data.xlsx')  
# 通过索引获取sheet。  
sheet=excel.sheets()[0]  
sheet=excel.sheet\_by\_index(0)  
# 通过名称获取sheet。  
sheet=excel.sheet\_by\_name('Sheet1')  
# 获取sheet的行数。  
print(sheet.nrows)  
# 获取sheet的列数。  
print(sheet.ncols)  
# 获取某一行数据。  
row=sheet.row(1)  
print(row)  
# 获取某一列数据。  
col=sheet.col(0)  
print(col)  
# 获取某一行的某列数据。  
print(row[1].value)  
# 获取某一列的某行数据。  
print(col[0].value)  
# 通过sheet获取某行某列数据。  
print(sheet.cell(2,1).value)

## 数据库读取数据

从数据库中读取数据首先需要建立数据库链接，以SQLSERVER为例，需要导入pymssql，建立链接：sql=pymssql.connect('127.0.0.1:1433','sa','000000','intrust')。数据库链接建立后，需要创建游标cursor=sql.cursor()，然后使用游标来执行SQL查询语句cursor.execute(sql)。游标执行查询语句，会把结果存入到游标中，读取数据时，也是通过游标来读取。游标读取数据有以下几种方法：（1）cursor.fetchone()读取一条数据；（2）cursor.fetchmany(n)读取n条数据；（3）cursor.fetchall()读取全部数据。游标与数据库连接在使用完后需要调用close()方法关闭。

import pymssql  
# 实例化数据库连接对象。  
sql=pymssql.connect('127.0.0.1:1433','sa','000000','intrust')  
# 实例化游标。  
cursor=sql.cursor()  
# 游标执行sql语句。  
cursor.execute('SELECT LOGIN\_USER,PASSWORD FROM TOPERATOR WHERE OP\_CODE<1000')  
# 读取数据。  
data=cursor.fetchall()  
print(data)  
cursor.close()  
sql.close()

# 使用随机数据

软件在使用时往往有一些数据要求不能重复，例如注册时不能使用已经存在的用户名、新建客户信息时不能输入已经存在的身份证信息等。自动化测试脚本往往会重复性执行，这样数据的重复性会导致测试脚本执行失败。使用随函数生成随机数据可以帮助解决数据重复性问题，python的随机函数模块为random，常用的方法为random(str,n)：从str中随机取出n个字符组成一个list，使用join函数可以将list转换成str进行转换：

import random  
# random.sample(str,n)：从str中随机取出n个字符组成一个list。  
for i in range(6):  
 print(''.join(random.sample('1234567890',8))+'@qq.com')

# 配置页面元素定位以及元素操作

页面上的元素属性如果发生了变化，可能会造成原测试脚本无法定位到元素，这时候就需要更改测试脚本，或者页面上增加或者减少了元素时也需要更改测试脚本。如果把页面上的元素定位以及元素操作是通过配置来实现的，那么只需要通过更改配置就能够使得原来的脚本也能够完成测试任务，这是一种测试脚本封装的思路。以下介绍两种配置方法：（1）配置文件配置；（2）数据库信息配置。

## 配置文件配置

比较规范的配置文件是.ini文件，.ini文件由节、键、值组成，例如：

[Login]

username=admin

password=000000

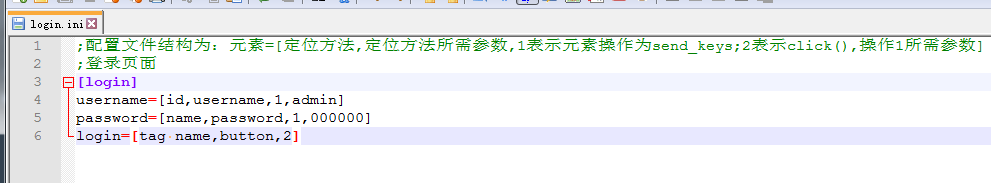
;账号密码信息

[]为节，键与值用=相连，;开头的是注释。

读取.ini文件需要使用第三方模块configparser，使用时先要下载然后导入。

import configparser  
# 实例化配置文件对象。  
config = configparser.ConfigParser()  
# 读取配置文件信息。  
config.read(r'E:\Selenium-Python\进阶\配置页面元素定位方法及操作方法\login.ini')  
# 获取配置文件的节点信息，以列表的形式返回。  
print(config.sections())  
section=config.sections()[0]  
# 获取节点下的键，以列表的形式返回。  
print(config.options(section))  
# 获取节点下的键值对，返回以元组形式的元素组成的列表。  
print(config.items(section))  
# 获取节点下键对应的值。  
print(config.get(section,'username'))

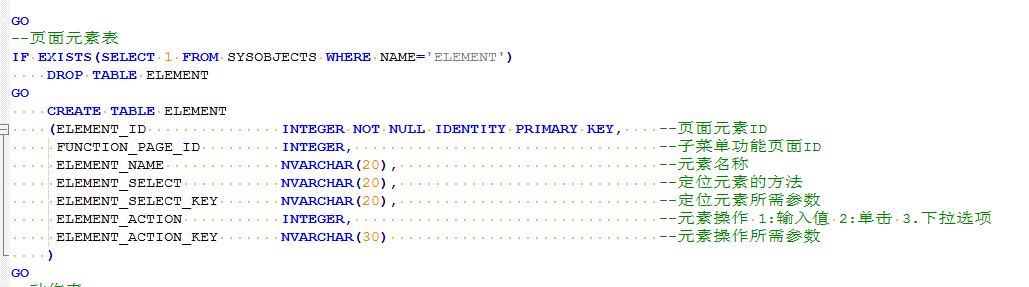
使用配置信息配置查找元素以及元素操作：



from selenium import webdriver  
from time import sleep  
import configparser  
def login(\*args):  
 browser=webdriver.Ie()  
 browser.maximize\_window()  
 browser.implicitly\_wait(5)  
 browser.get('http://localhost:9087')  
 # 循环迭代列表。  
 for i in args:  
 # 拆分列表元素。  
 list=i.split(',')  
 if list[2]=='1':  
 browser.find\_element(list[0],list[1]).send\_keys(list[3])  
 elif list[2]=='2':  
 browser.find\_element(list[0], list[1]).click()  
 sleep(3)  
 browser.quit()  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 实例化配置文件对象。  
 config=configparser.ConfigParser()  
 # 读取配置文件信息。  
 config.read(r'E:\Selenium-Python\进阶\配置页面元素定位以及元素操作\login.ini')  
 list=[]  
 # 取出节点的value值组装成一个列表。  
 for i in config.items(config.sections()[0]):  
 list.append(i[1])  
 login(\*list)

## 数据库信息配置

通过数据库信息配置也是一种比较好的方式，前提是需要对数据库有所了解，因为这里不仅涉及到数据库表的设计，还需要数据库的增删改查等操作。以下是一种比较简单的表，可作为参考：



from selenium import webdriver  
from time import sleep  
import pymssql  
def login(\*args):  
 browser=webdriver.Ie()  
 browser.maximize\_window()  
 browser.implicitly\_wait(5)  
 browser.get('http://localhost:9087')  
 # 循环取出数据。  
 for tuple in args:  
 if tuple[2]==1:  
 browser.find\_element(tuple[0],tuple[1]).send\_keys(tuple[3])  
 elif tuple[2]==2:  
 browser.find\_element(tuple[0], tuple[1]).click()  
 sleep(3)  
 browser.quit()  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 建立数据库连接。  
 with pymssql.connect('127.0.0.1:1433','sa','000000','TEST') as sql:  
 # 实例化数据库索引。  
 with sql.cursor() as cursor:  
 # 查询数据。  
 cursor.execute('SELECT ELEMENT\_SELECT,ELEMENT\_SELECT\_KEY,ELEMENT\_ACTION,ELEMENT\_ACTION\_KEY FROM ELEMENT WHERE FUNCTION\_PAGE\_ID=1')  
 # 从索引中取出所查的数据（数据结果为一个列表）。  
 list=cursor.fetchall()  
 login(\*list)

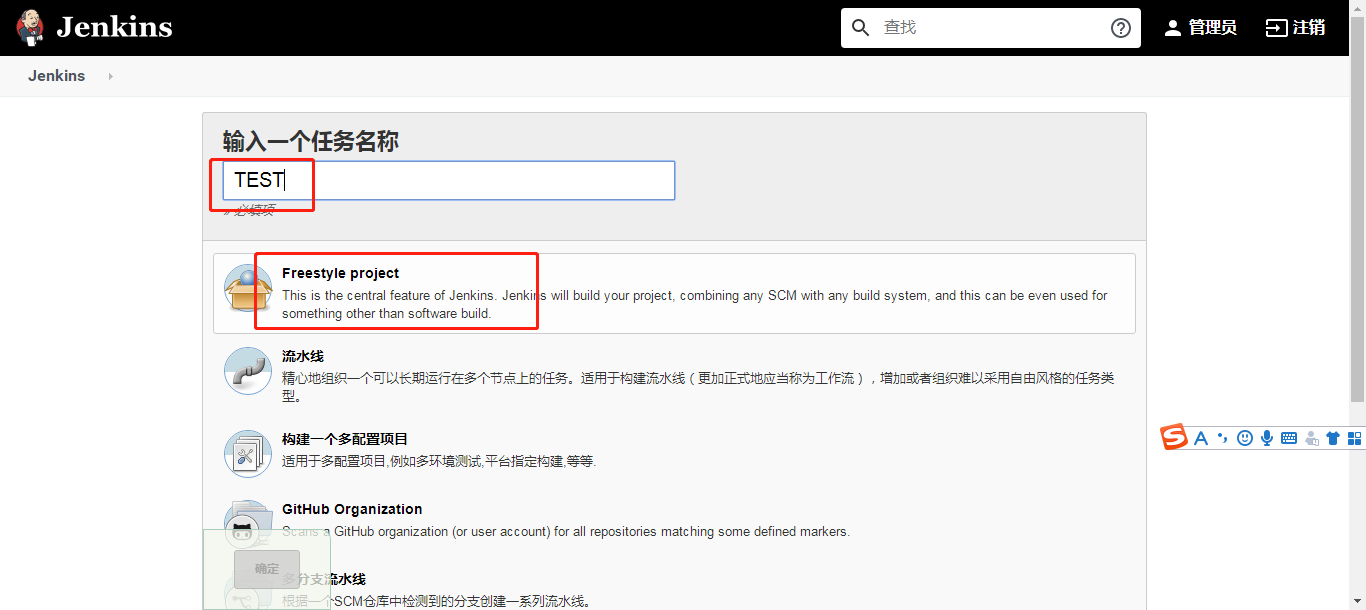
# 持续集成

持续集成是指开发阶段对项目进行持续性自动化编译、测试,以达到控制代码质量的手段，持续集成强调开发人员提交了新代码之后，立刻可以完成自动构建构建、然后进行自动化测试。

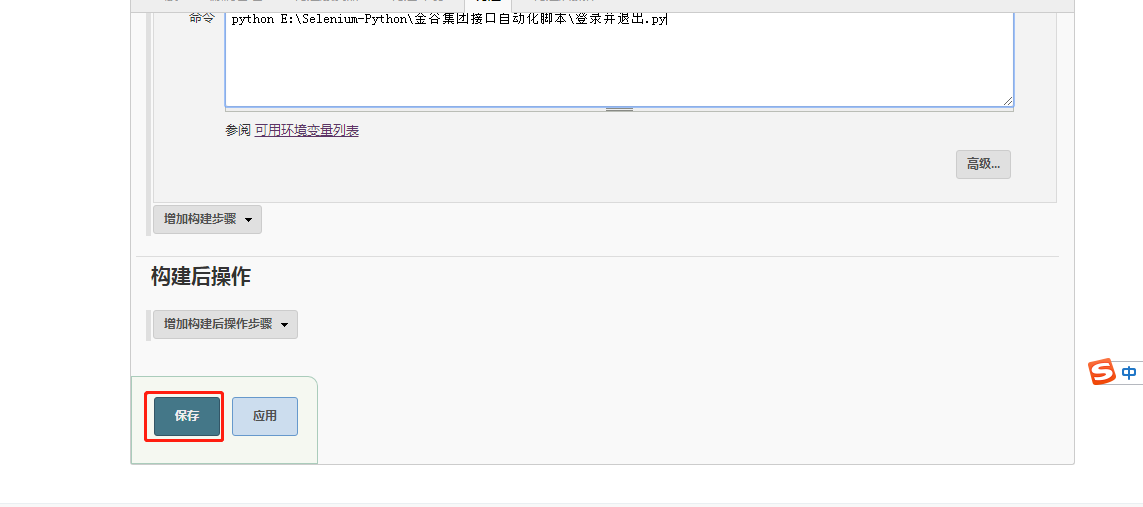
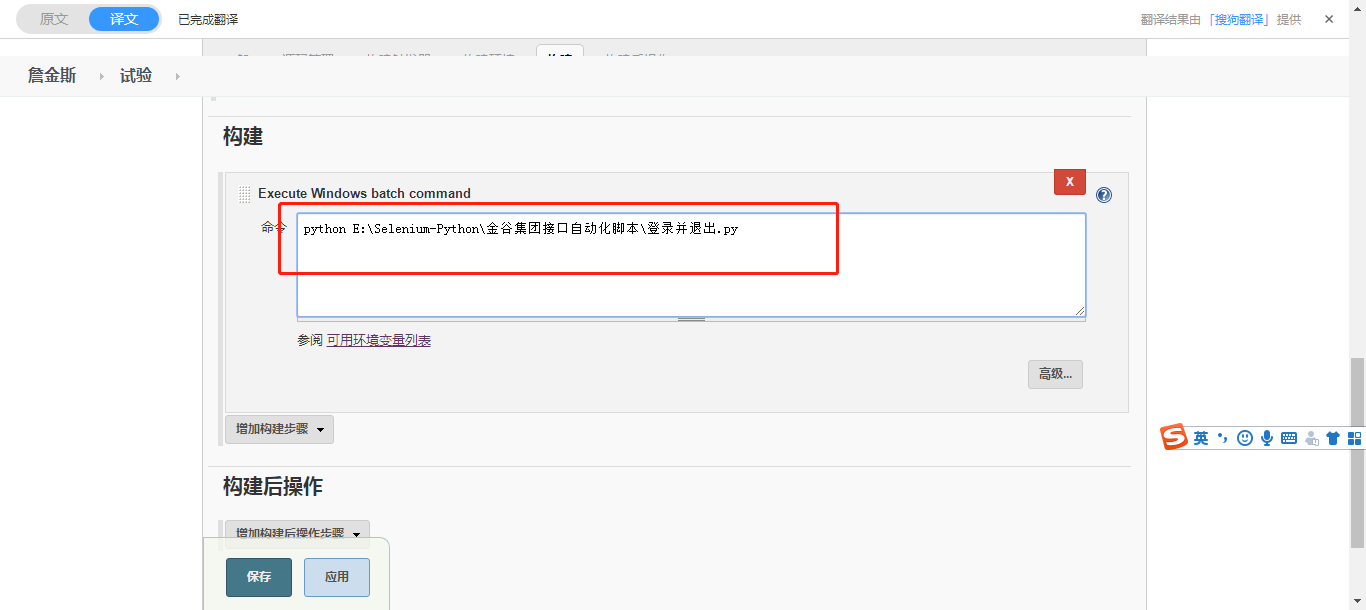
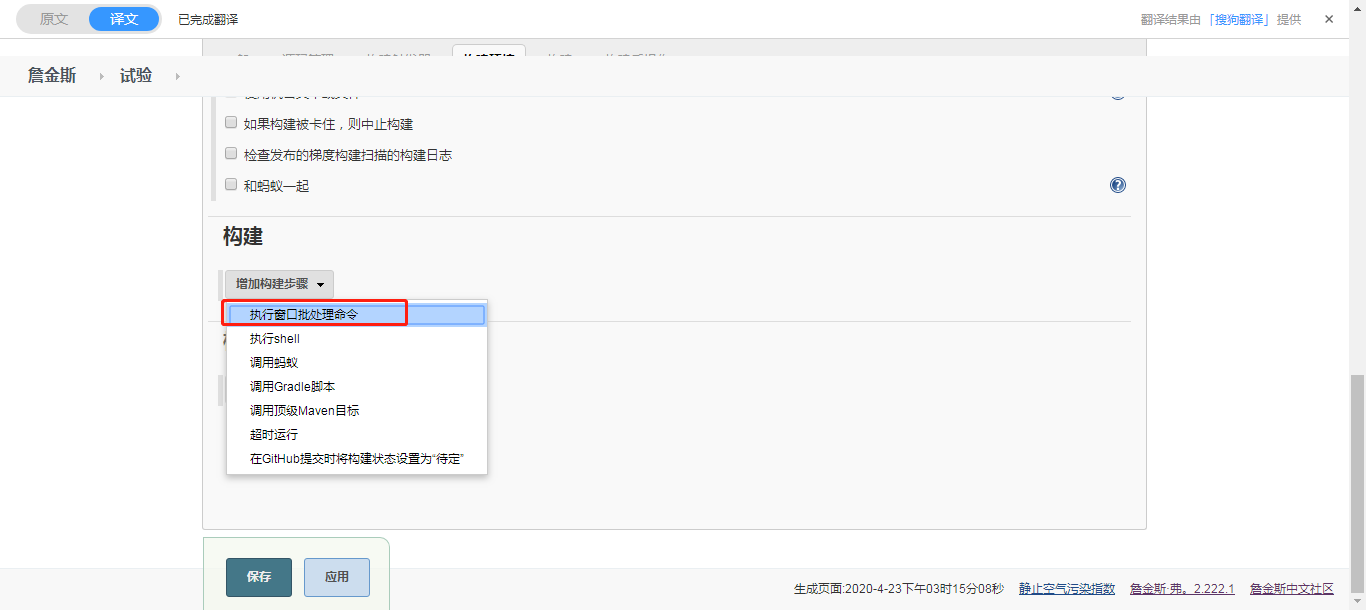
Jenkins是一个开源的、提供友好操作界面的持续集成(CI)工具，主要用于持续、自动的构建/测试软件项目、监控外部任务的运行。Jenkins用Java语言编写，可在Tomcat等流行的servlet容器中运行，也可独立运行。通常与版本管理工具(SCM)、构建工具结合使用。常用的版本控制工具有SVN、GIT，构建工具有Maven、Ant、Gradle。

## 创建简单的Jenkins任务

点击创建一个新任务，输入任务名称，选择自由风格的任务，点击确定。



选择构建步骤执行window批处理命令，输入执行语句，然后保存生成任务。



生成任务后，点击立即构建，如果构建成功则会显示蓝色的小圆标失败的话会显示红色的小圆标。失败的话，可以点击构建记录，进入到控制台查看输出信息查找构建失败的原因。

