

实验6 部署自动化测试框架

[官网地址](#)

1. 部署python虚拟环境

[1.1. 创建虚拟环境](#)

[1.2. 激活虚拟环境](#)

[1.3. 退出虚拟环境](#)

[1.4. 删除虚拟环境](#)

2. 下载安装Robot Framework

[2.1. 下载安装RF](#)

[2.2. 下载安装RF-GUI](#)

[2.3. 下载安装RF测试库](#)

[appiumlibrary](#)

[selenium2Library](#)

[rf-browser](#)

[database library](#)

[request library](#)

3. Robot Framework基本操作

[3.1. 建立新项目](#)

[建立新项目](#)

[创建测试套件](#)

[创建测试用例](#)

[GUI与文件目录结构](#)

[3.2. 编辑测试脚本与测试用例](#)

[Test Suite Edit](#)

[导入外部库](#)

[Test Case Edit](#)

[3.2. RF 内建 Library](#)

4. 集成Selenium扩展库完成web自动化测试

[4.1. 对应浏览器驱动配置](#)

[4.2. 封装、引入、调用：工作流与数据流的分离](#)

[Define User Keywords](#)

[封装第一个用户关键字：login](#)

[引入关键字](#)

[Define Variables](#)

[尝试运行](#)

5. OJ系统Web自动化测试：随机应变

5.1. 登录功能

结果展示

5.2. 代码评测功能

二次处理代码文本

登录保存cookie

根据id选择页面

提交代码的流程

线程睡眠的软同步

结果展示

5.3. 注册功能

结果展示

5.4. 修改个人信息

结果展示

5.5. 问题搜索

结果展示

5.6. 状态查看

结果展示

5.7. 排名查看

结果展示

6. 集成RF-Requests库完成接口自动化测试

6.1. 代码评测功能

接口鉴权的准备：从会话中获取cookie

源码解析：何以登录？

登录获得cookie

请求服务端接口

获取请求体结构

检查提交结果

总体调用流程

结果展示

6.2. 注册功能测试

结果展示

7. 总结

官网地址

Robot Framework

1. 部署python虚拟环境

我自己这台计算机上配过很多其他项目得python环境，Lib里面的依赖包非常多，做这个实验的时候发生了很多依赖冲突。所以要单开一个虚拟环境。

1.1. 创建虚拟环境

```
# 找一个部署venv环境的文件夹  
python -m venv rf_venv # 我的叫rf_venv
```

此电脑 > 本地磁盘 (D:) > work_place > st > lab6			
名称	修改日期	类型	大小
rf_venv	2024/5/14 14:10	文件夹	

1.2. 激活虚拟环境

随后要激活这个虚拟环境，不同os上方式不一样，我是win。那么就是：

```
.\rf_venv\Scripts\activate # 执行命令的目录搞清楚。。
```

```
(rf_venv) D:\work_place\st\lab6>
```

1.3. 退出虚拟环境

```
deactivate
```

```
(rf_venv) D:\work_place\st\lab6>deactivate  
D:\work_place\st\lab6>
```

1.4. 删除虚拟环境

```
rm -rf ./rf_venv
```

我就不删了。

2. 下载安装Robot Framework

2.1. 下载安装RF

```
pip install robotframework==6.1.1
```

注意**这里不要下最新版rf**，因为和ride不兼容，log的时候会报错的。兼容版本是6.1.1。

具体参考：

[【Robor framework】 failed: AttributeError: 'Output' object has no attribute '_xmllogger' Log打印报错未解决_testrunneragent.py' failed: attributeerror: 'output- CSDN博客](#)

安装后查看版本可用：

```
robot --version
```

```
(rf_venv) D:\workSpace\st\lab6>robot --version
Robot Framework 6.1.1 (Python 3.11.0 on win32)
```

2.2. 下载安装RF-GUI

Wxpython是python非常有名的一个GUI库，支持python图形化界面，因为RIDE 是基于这个库开发的，所以这个必须安装。

```
pip install wxpython
```

```

C:\Users\Administrator>pip install wxpython
Collecting wxpython
  Downloading wxPython-4.2.1-cp311-cp311-win_amd64.whl.metadata (3.0 kB)
Requirement already satisfied: pillow in c:\python311\lib\site-packages (from wxpython) (9.5.0)
Requirement already satisfied: six in c:\python311\lib\site-packages (from wxpython) (1.16.0)
Requirement already satisfied: numpy in c:\python311\lib\site-packages (from wxpython) (1.24.4)
Downloading wxPython-4.2.1-cp311-cp311-win_amd64.whl (17.8 MB)
----- 17.8/17.8 MB 24.2 MB/s eta 0:00:00
Installing collected packages: wxpython
Successfully installed wxpython-4.2.1

```

RIDE是Robot Framework带图形界面测试编辑器

由于老版RIDE不支持python3，需要安装psutil和最新版的robotframework-ride

```

pip install psutil
pip install -U --pre robotframework-ride

```

```

C:\Users\Administrator>pip install -U --pre robotframework-ride
Collecting robotframework-ride
  Downloading robotframework-ride-2.0.8.1.tar.gz (1.4 MB)
----- 1.4/1.4 MB 2.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting PyPubSub (from robotframework-ride)
  Downloading Pypubsub-4.0.3-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: Pygments in c:\python311\lib\site-packages (from robotframework-ride) (2.17.2)
Requirement already satisfied: psutil in c:\python311\lib\site-packages (from robotframework-ride) (5.9.8)
Requirement already satisfied: wxPython in c:\python311\lib\site-packages (from robotframework-ride) (4.2.1)
Requirement already satisfied: Pywin32 in c:\python311\lib\site-packages (from robotframework-ride) (306)
Requirement already satisfied: pillow in c:\python311\lib\site-packages (from wxPython->robotframework-ride) (9.5.0)
Requirement already satisfied: six in c:\python311\lib\site-packages (from wxPython->robotframework-ride) (1.16.0)
Requirement already satisfied: numpy in c:\python311\lib\site-packages (from wxPython->robotframework-ride) (1.24.4)
Downloading Pypubsub-4.0.3-py3-none-any.whl (61 kB)
----- 61.4/61.4 kB ? eta 0:00:00
Building wheels for collected packages: robotframework-ride
  Building wheel for robotframework-ride (setup.py) ... done
  Created wheel for robotframework-ride: filename=robotframework_ride-2.0.8.1-py3-none-any.whl size=1445701 sha256=3322c8ebdce6b809bb1c755261492d5a153a99d865aefe4c4389e00f8a9dc9ed
  Stored in directory: c:\users\administrator\appdata\local\pip\cache\wheels\6d\8f\04\231c9412b1cc8077cea72cd12fbbc1861e09d3348afe8a623
Successfully built robotframework-ride
Installing collected packages: PyPubSub, robotframework-ride
Successfully installed PyPubSub-4.0.3 robotframework-ride-2.0.8.1

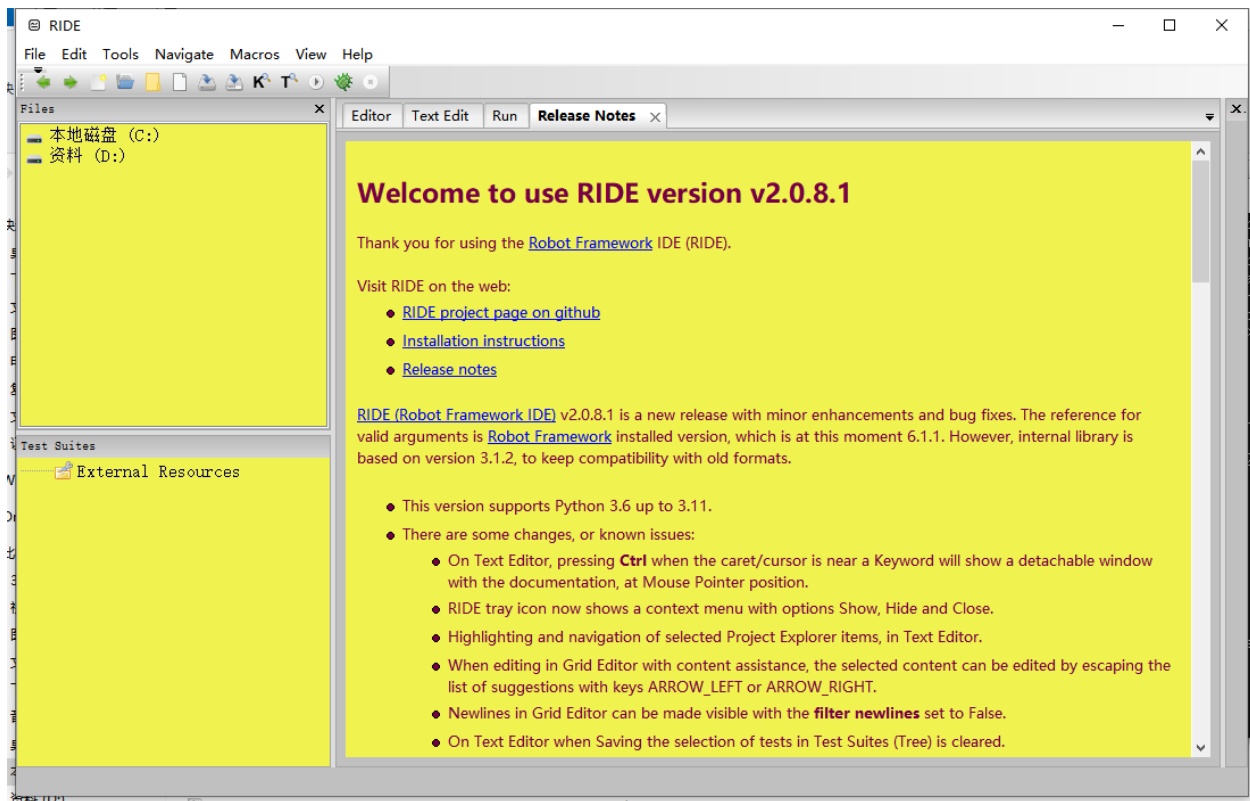
```

如果想启动这个图形化界面，可以运行Python/Scripts下的ride.py

本地磁盘 (C:) > Python311 > Scripts

名称	修改日期	类型	大小
pywin32_testall.py	2024-04-24 21:07	Python File	4 KB
pywsrc.exe	2024-05-14 12:41	应用程序	106 KB
qtpy.exe	2024-04-24 21:07	应用程序	106 KB
rapidocr_onnxruntime.exe	2024-04-24 22:17	应用程序	106 KB
rebot.exe	2024-05-14 12:34	应用程序	106 KB
ride.py	2024-05-14 12:43	Python File	1 KB
ride_postinstall.py	2024-05-14 12:43	Python File	1 KB
robot.exe	2024-05-14 12:34	应用程序	106 KB
rouge.exe	2024-04-12 19:31	应用程序	106 KB
runxldr.py	2024-04-24 22:13	Python File	16 KB
saved_model_cli.exe	2023-10-10 12:32	应用程序	106 KB
send2trash.exe	2024-04-24 21:07	应用程序	106 KB
shortuuid.exe	2024-04-24 22:14	应用程序	106 KB
spacy.exe	2024-04-24 22:18	应用程序	106 KB
streamlit.cmd	2024-04-24 22:18	Windows 命令脚本	1 KB
streamlit.exe	2024-04-24 22:18	应用程序	106 KB
tabulate.exe	2024-04-24 22:13	应用程序	106 KB
tensorboard.exe	2023-10-10 12:32	应用程序	106 KB
tf_upgrade_v2.exe	2023-10-10 12:32	应用程序	106 KB
tflite_convert.exe	2023-10-10 12:32	应用程序	106 KB
tidevice.exe	2024-05-11 14:51	应用程序	106 KB
toco.exe	2023-10-10 12:32	应用程序	106 KB
toco_from_protos.exe	2023-10-10 12:32	应用程序	106 KB
torchrun.exe	2024-04-24 22:15	应用程序	106 KB
tqdm.exe	2024-04-24 22:13	应用程序	106 KB
transformers-cli.exe	2024-04-24 22:17	应用程序	106 KB
ttx.exe	2023-10-10 13:00	应用程序	106 KB
unstructured-ingest.exe	2024-04-24 22:23	应用程序	106 KB
uvicorn.exe	2024-04-24 22:14	应用程序	106 KB
vba_extract.py	2024-04-24 22:13	Python File	3 KB
watchmedo.exe	2024-04-24 22:13	应用程序	106 KB
wavedrompy.exe	2024-04-24 22:14	应用程序	106 KB
weasel.exe	2024-04-24 22:16	应用程序	106 KB
wheel.exe	2023-10-10 12:31	应用程序	106 KB
wsdump.exe	2024-04-24 21:07	应用程序	106 KB
wxdemo.exe	2024-05-14 12:41	应用程序	106 KB
wxdocs.exe	2024-05-14 12:41	应用程序	106 KB
wxget.exe	2024-05-14 12:41	应用程序	106 KB

图形化界面长这样：



2.3. 下载安装RF测试库

appiumlibrary

```
pip install robotframework-appiumlibrary
```

```
Successfully installed Appium-Python-Client-4.0.0 docutils-0.21.2 kitchen-1.2.6 outcome-1.3.0.post0 pysocks-1.7.1 robotframework-appiumlibrary-2.0.0 selenium-4.20.0 sortedcontainers-2.4.0 trio-0.25.0 trio-websocket-0.11.1 typing_extensions-4.11.0 wsproto-1.2.0
```

selenium2Library

```
pip install robotframework-selenium2Library
```

```
Successfully installed robotframework-pythonlibcore-4.4.1 robotframework-selenium2Library-3.0.0 robotframework-seleniumlibrary-6.3.0
```

rf-browser

```
pip install robotframework-browser
```

```
Successfully installed grpcio-1.63.0 grpcio-tools-1.63.0 protobuf-5.26.1 robotframework-assertion-engine-3.0.3 robotframework-browser-18.4.0 wrapt-1.16.0
```

database library

```
pip install robotframework-databaselibrary
```

```
Installing collected packages: robotframework-excellib, robotframework-databaselibrary  
Successfully installed robotframework-databaselibrary-1.4.4 robotframework-excellib-2.0.1
```

request library

```
pip install robotframework-requests
```

```
Successfully installed robotframework-requests-0.9.7
```

1. appium：移动应用测试库
2. selenium：web应用测试库
3. requests：接口测试库

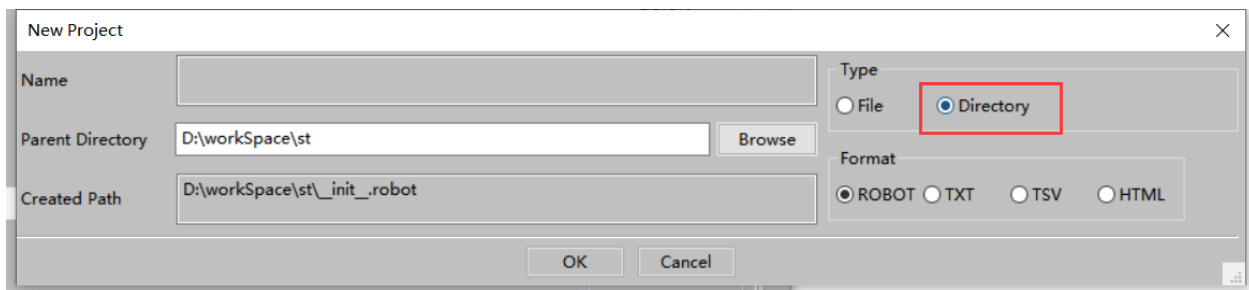
对于我们要测试的SUT系统（hustoj的在线评测的网站），比较适用的Selenium2Library和SeleniumLibrary这两个库。其实有后端源码的情况下，request library也是可以用的。

3. Robot Framework基本操作

3.1. 建立新项目

建立新项目

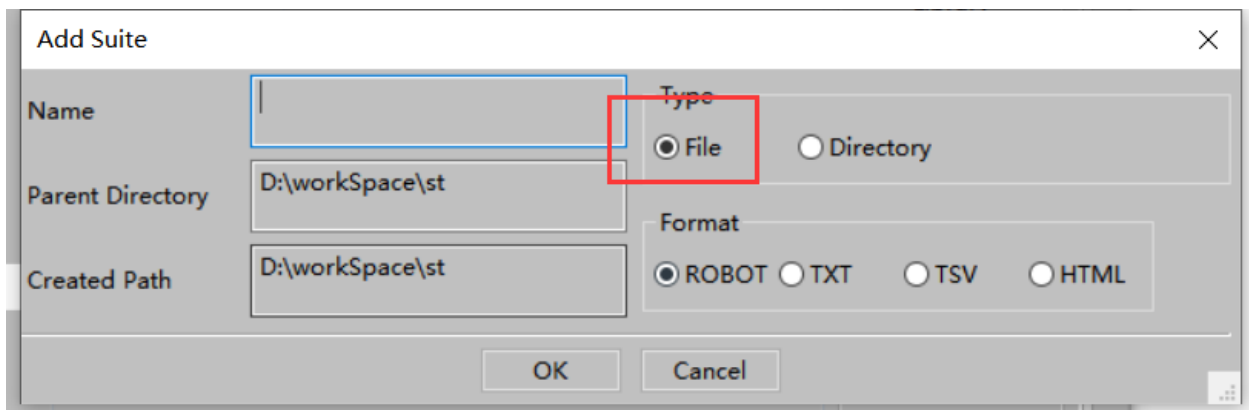
点击File即可选择创建Project



注：Type选择Directory原因是，在Directory的项目下可以创建测试套件，如果选Type为File，则只能创建测试用例，这不利于用例的管理。

创建测试套件

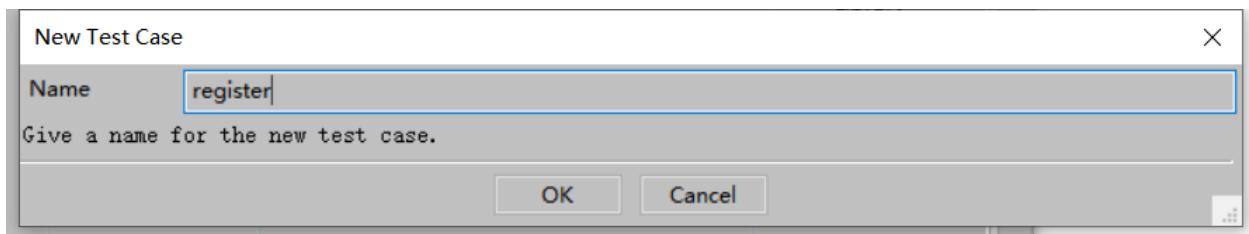
右击项目即可创建test suite



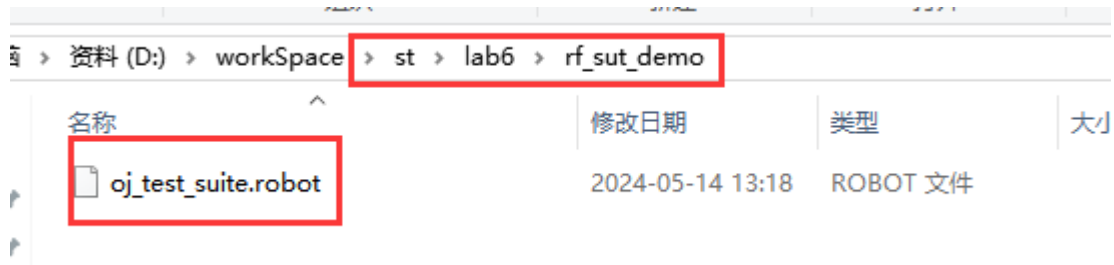
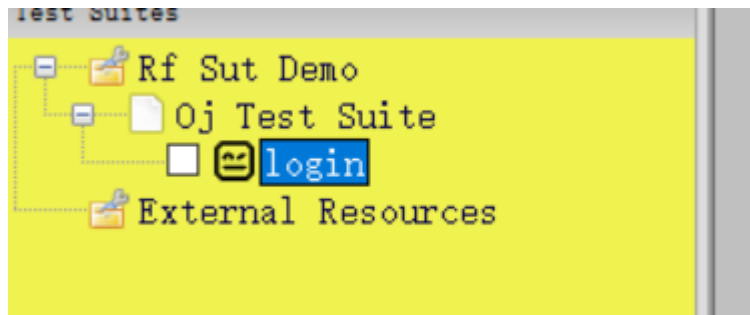
注：Type选择File原因是，测试用例只能在File类型下创建，如果Type为Directory，还得重新再继续建File的测试套件后才能创建测试用例。

创建测试用例

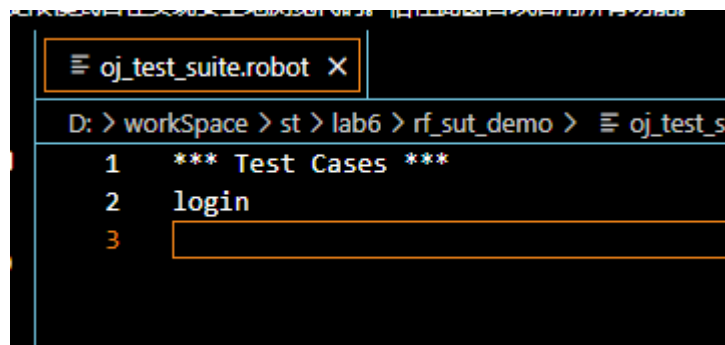
右击test suite即可选择创建test case



GUI与文件目录结构



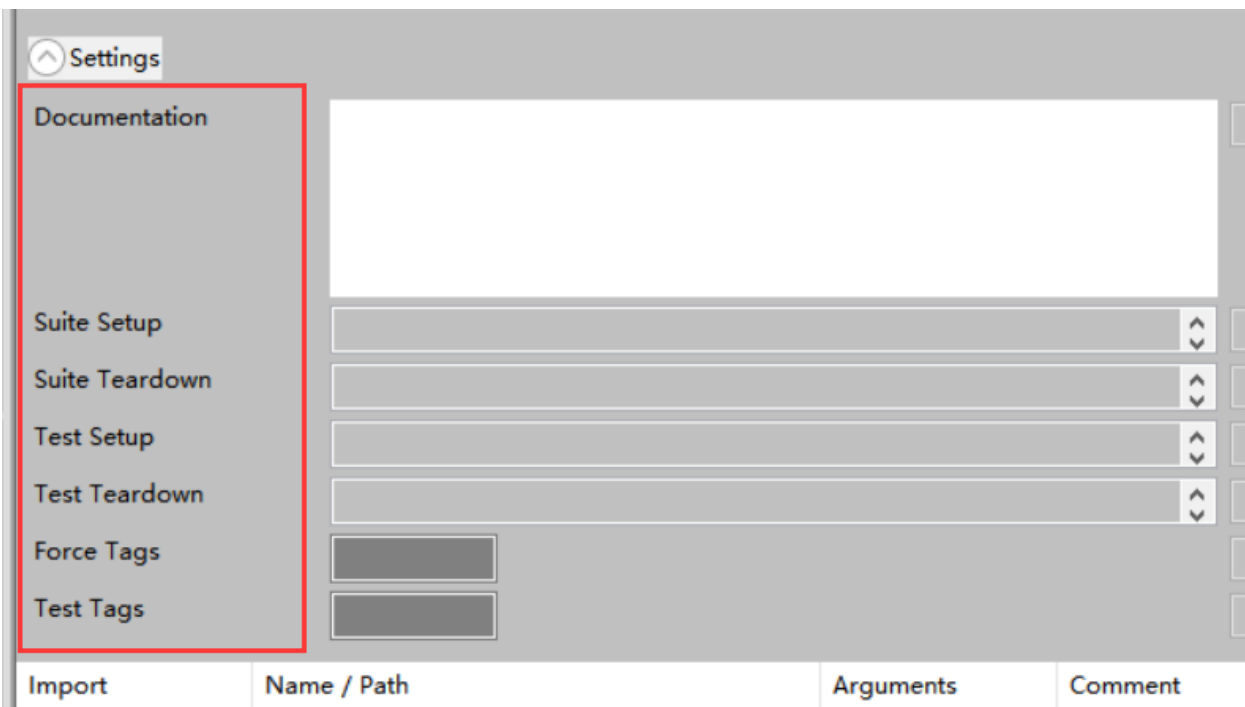
本质上他是通过一个叫做xxx.robot的文件来管理测试脚本以及测试用例的。



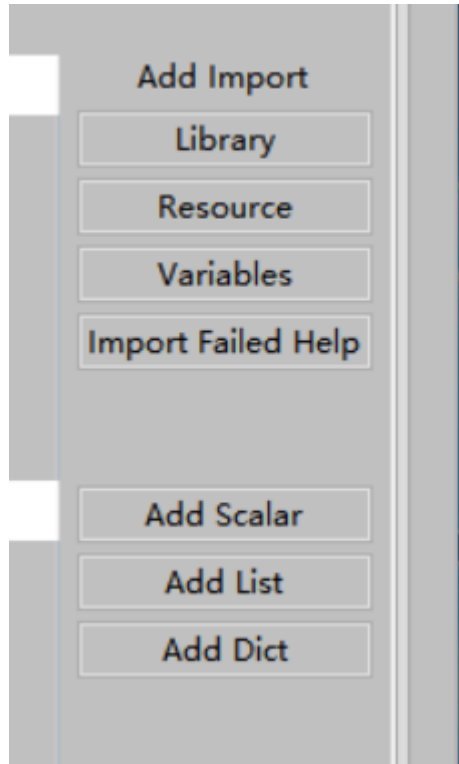
这里面管理了该测试套件下的所有测试用例

3.2. 编辑测试脚本与测试用例

Test Suite Edit













1. Documentation：文档，每一项都有。可以给当前的对象写一些文档说明。
2. Suite Setup:指的是测试套件启动的时候就执行某个关键字。(例如Suite Setup设置了Sleep | 5sec，表示等待5秒，要注意**关键字的参数要使用|分隔**)
3. Suite Teardown：指的是测试套件结束的时候就执行某个关键字。
4. Test Setup：指的就是样例启动的时候执行某个关键字。
5. Test Teardown：指的就是样例结束的时候执行某个关键字。
6. Force Tags：强制标记，给当前项目及项目下的测试套件及套件下的每个测试案例都加上Tag，只能在设置的地方删除。



此外也可以导入一些外部库、资源或者变量，或增加进度条、字典对之类的内容。

导入外部库

Library选项，主要是Python/Lib/site-packages里之前下载的库。

	robotframework_appiumlibrary-2.0.0....	2024-05-14 12:55	文件夹
	robotframework_assertion_engine-3....	2024-05-14 13:02	文件夹
	robotframework_browser-18.4.0.dist-...	2024-05-14 13:02	文件夹
	robotframework_databaselibrary-1.4....	2024-05-14 13:03	文件夹
	robotframework_excellib-2.0.1.dist-in...	2024-05-14 13:03	文件夹
	robotframework_pythonlibcore-4.4.1....	2024-05-14 12:59	文件夹
	robotframework_requests-0.9.7.dist-i...	2024-05-14 13:04	文件夹
	robotframework_ride-2.0.8.1.dist-info	2024-05-14 12:43	文件夹
	robotframework_selenium2library-3....	2024-05-14 12:59	文件夹
	robotframework_seleniumlibrary-6.3....	2024-05-14 12:59	文件夹

Settings			
Import	Name / Path	Arguments	Add Import
Library	Selenium2Library		Library
Library	SeleniumLibrary		Resource
			Variables

注意，在导包时一定要和文件夹的名字一致，包括大小写

Resource：加载资源，主要是工程相关的资源文件。

Variables：加载变量文件，不怎么用，可暂时忽略。

Test Case Edit

1. Documentation：文档，每一项都有。可以给当前的对象写一些文档说明。
2. Setup：指的是案例启动的时候执行某个关键字。
3. Teardown：指的是案例结束的时候执行某个关键字。

4. **Timeout**：设置每一个测试案例的超时时间，超时则失败，并停止案例运行。
5. **Template**：测试模版，这是可以指定某个关键字为这个测试套件下所有TestCase的模版，这样所有的TestCase就只需要设置这个关键字的传入参数即可。
6. **Tags**：标记某个测试用例。在Run区中Only Run Tests with these Tags和Skip Tests with these Tags，会通过这个标志来识别是否运行或跳过用例。

3.2. RF 内建 Library

RF的内建库，主要有这么几个：

1. **BuiltIn** - 提供了大量通用的关键字，比如断言、循环、条件语句等。
2. **Collections** - 用于处理列表和字典等集合类型的关键字。
3. **DateTime** - 提供了与日期和时间相关的关键字。
4. **Dialogs** - 允许测试从终端用户那里获取输入。
5. **Easter** - 提供了一些有趣的复活节彩蛋关键字，通常用于演示。
6. **OperatingSystem** - 提供了与操作系统交互的关键字，比如文件和目录操作。
7. **Process** - 允许管理和监控外部进程。
8. **Remote** - 使得在远程服务器上运行测试成为可能。
9. **Reserved** - 包含了一些保留的关键字，这些关键字用于内部处理。
10. **Screenshot** - 提供了截图功能，可以捕获测试执行过程中的屏幕内容。
11. **String** - 提供了用于处理字符串的关键字。
12. **Telnet** - 允许通过Telnet与服务器进行交互。
13. **XML** - 提供了用于处理XML文件和数据的关键字。

比如Log（控制台打印关键字），就属于BuiltIn。如果想要查看某个关键字，可以F5快捷键，或者在Tools中找到Search Keyword，查找某个关键字的用法及其对应的库。

另外，如果想自动补全或者在编辑时快速检索某个关键词，可以使用快捷键**shift+ctrl+space**。

之后我们可以看到在指定路径下的报告（包括报告、日志、控制台信息等）。

板	组织	新建	打开	选择	
> 此电脑 > 资料 (D:) > workSpace > st > lab6 > demo > target					
	名称	修改日期	类型	大小	
kDowr	Lab6-Console-20240517-091548.txt	2024-05-17 9:15	文本文档	3 KB	
	Lab6-Log-20240517-091548.html	2024-05-17 9:15	Microsoft Edge ...	237 KB	
all_ver	Lab6-Message-20240517-091548.log	2024-05-17 9:15	文本文档	2 KB	
rs	Lab6-Output-20240517-091548.xml	2024-05-17 9:15	XML 源文件	2 KB	
s	Lab6-Report-20240517-091548.html	2024-05-17 9:15	Microsoft Edge ...	242 KB	
s (x86					

并且，我们能在控制台信息中（Log输出的位置），看到这条消息。

```
Starting test: Lab6.Demo.Demo Suite.logSomething
20240517 09:45:18.352 : INFO : BuiltIn Log Keyword Log
```

之后的report中，可以看到测试用例的通过情况。

Lab6 Report

Generated
20240517 09:15:49 UTC+08:00
4 minutes 29 seconds ago

Summary Information

Status: All tests passed
Start Time: 20240517 09:15:49.023
End Time: 20240517 09:15:49.108
Elapsed Time: 00:00:00.085
Log File: [Lab6-Log-20240517-091548.html](#)

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	1	1	0	0	00:00:00	<div></div>
Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						<div></div>
Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Lab6	1	1	0	0	00:00:00	<div></div>
Lab6.Demo	1	1	0	0	00:00:00	<div></div>
Lab6.Demo.Demo Suite	1	1	0	0	00:00:00	<div></div>

Test Details

All Tags Suites Search

Suite:

Test:

Include:

Exclude:

Search Clear Help

而RF的内建库远不止这一个关键字。例如我们可以通过Set Variable关键字定义变量，通过Get Time关键字获取系统时间，或者是将一个数据转换为二进制串后输出出来，以及一些os层面的关键字，例如调用某个关键字等。而因为表格是按行来的，所以可以在某一行仅定义一个注释，也是可以的。

Log	BuiltIn Log Keyword Log	# 打印			
# 定义变量					
\${a}	Set Variable	20			
Log	\${a}				
# 获得系统时间					
\${time}	Get Time				
# 验证两个值是否相等					
\${assert_res}	Run Keyword And Return Status	Should Be Equal	\${a}	20	
Log	\${assert_res}				
# 验证一个字符串是否包含另					
Should Contain	I love software test!	love			
# os库					
运行关键字并忽略错误					
Run Keyword And Ignore Error	Log	Man!			
# 将数据转换为二进制格式					
\${bs}	Convert To Binary	\${20}			
Log	\${bs}				

结果如下。

```
elapsed time: 0.0000  pass: 1  skip: 0  fail: 0
Console log
-----
Lab6
1 test, 1 passed, 0 failed
-----
Debug:  D:\workspace\atl\lab6\demo\target\demo\Lab6-Message-20240517-095526.log
Output:  D:\workspace\atl\lab6\demo\target\demo\Lab6-Output-20240517-095526.xml
Log:    D:\workspace\atl\lab6\demo\target\demo\Lab6-Log-20240517-095526.html
Report:  D:\workspace\atl\lab6\demo\target\demo\Lab6-Report-20240517-095526.html
Test finished 20240517 09:55:27
```

```
20240517 09:55:26.992 : INFO : ${a} = 20
20240517 09:55:26.993 : INFO : 20
20240517 09:55:26.993 : INFO : ${time} = 2024-05-17 09:55:26
20240517 09:55:26.995 : INFO : ${assert_res} = True
20240517 09:55:26.996 : INFO : True
20240517 09:55:26.997 : INFO : Man!
20240517 09:55:26.998 : INFO : ${bs} = 10100
20240517 09:55:26.999 : INFO : 10100
Ending test: Lab6.Demo.Demo Suite.logSomething
```

4. 集成Selenium扩展库完成web自动化测试

web自动化测试主要是指对于web界面的自动化测试，也就是在不知道接口的情况下，获取界面元素，进行点击、输入等。

跟实验三的selenium+webdriver很像，只不过这次我们不再需要编写脚本。

主要依赖于之前下载过的seleniumLibrary

```
(rf_venv) D:\workSpace\st\lab6>pip show robotframework-selenium2Library
Name: robotframework-selenium2library
Version: 3.0.0
Summary: Web testing library for Robot Framework
Home-page: https://github.com/robotframework/Selenium2Library
Author: Tatu Aalto
Author-email: aalto.tatu@gmail.com
License: Apache License 2.0
Location: D:\workSpace\st\lab6\rf_venv\Lib\site-packages
Requires: robotframework-seleniumlibrary
Required-by:
```

4.1. 对应浏览器驱动配置

一般常用的浏览器有：













1. IE（当然现在也很少有人用了，都是一些老Web应用，只能跑在IE上）
2. Edge（微软）
3. Firefox（火狐）
4. Chrome（谷歌）

无论是哪种浏览器，在利用rf+selenium进行自动化测试，都需要为其安装驱动，否则selenium没法启动对应的浏览器。






例如我用的是火狐，火狐的驱动的地址：

<https://github.com/mozilla/geckodriver/releases/>

找到一个兼容当前机器上firefox浏览器的驱动版本，下载下来（根据自己机器情况来）：

▼ Assets 12			
	geckodriver-v0.33.0-linux-aarch64.tar.gz	2.93 MB	Apr 3, 2023
	geckodriver-v0.33.0-linux32.tar.gz	3.02 MB	Apr 3, 2023
	geckodriver-v0.33.0-linux32.tar.gz.asc	833 Bytes	Apr 3, 2023
	geckodriver-v0.33.0-linux64.tar.gz	2.93 MB	Apr 3, 2023
	geckodriver-v0.33.0-linux64.tar.gz.asc	833 Bytes	Apr 3, 2023
	geckodriver-v0.33.0-macos-aarch64.tar.gz	1.85 MB	Apr 3, 2023
	geckodriver-v0.33.0-macos.tar.gz	2.05 MB	Apr 3, 2023
	geckodriver-v0.33.0-win-aarch64.zip	1.47 MB	Apr 3, 2023
	geckodriver-v0.33.0-win32.zip	1.54 MB	Apr 3, 2023
	geckodriver-v0.33.0-win64.zip	1.59 MB	Apr 3, 2023
	Source code (zip)		Apr 3, 2023
	Source code (tar.gz)		Apr 3, 2023

然后解压得到exe文件，把这个exe文件放在自己正在使用的python环境的根目录下。例如我正在使用的虚拟环境中。

	Include	2024/5/14 14:10	文件夹	
	Lib	2024/5/14 14:10	文件夹	
	Scripts	2024/5/17 14:00	文件夹	
	geckodriver.exe	2023/4/2 22:00	应用程序	3,794 KB
	pyenv.cfg	2024/5/14 14:10	Configuration 源...	1 KB

这样驱动环境就配完了。

4.2. 封装、引入、调用：工作流与数据流的分离

假设现在我们有一个过程，这个过程可以完成一个系统的某个功能点的测试（例如登录），那么这个过程首先涉及了测试时的**工作流程**。其次涉及测试时需要使用的**数据**。

举个例子，登录过程涉及：

工作流程：

1. 获取到登陆界面
2. 定位到用户名、密码输入框

数据流：

1. 登陆界面的url
2. 浏览器驱动

3. 向输入框输入
4. 定位到登录按钮
5. 点击登录按钮

3. 用户名
4. 密码

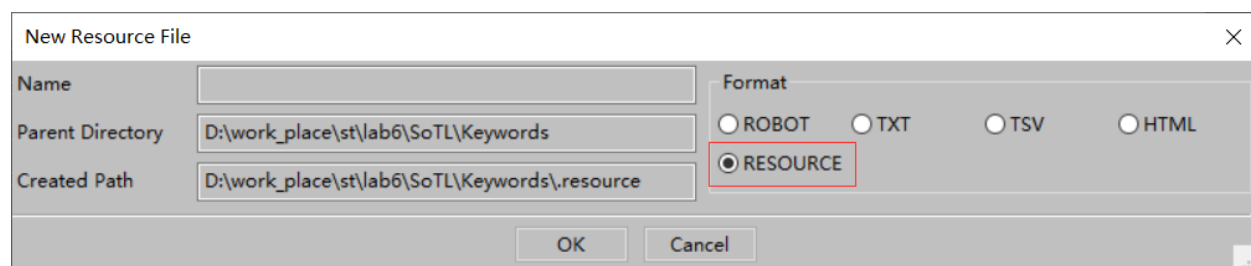
如果我们把工作流和数据流全放在一个文件中，那么下一次我们想更换一个测试用例，就需要完整的复制一遍这一套东西到另一个文件运行，产生了大量冗余。当这个冗余非常庞大后，将完全无法管理。

因此工作流和数据流最好分离开来，我们先封装好一些基础的工作流程以及一些基本的、通用的变量，随后引入到一个测试用例的文件中，并进行调用，这样便大大减少冗余。

Define User Keywords

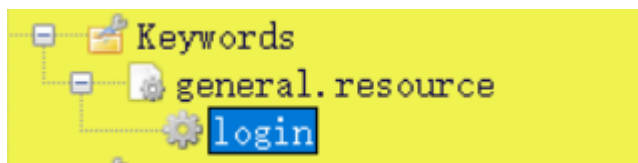
我们知道rf支持很多关键字，而这些关键字都是官方库或者扩展库的。而用户也可以自定义自己的关键字，相当于封装一个函数。

例如，现在我想封装一个登录的函数（这个函数将在很多地方用到，例如，登录功能本身，或者是提交代码进行测评，需要在登录状态下进行），在各处调用，如何做？



右击工作目录，选择New Resource，即可建立Resource类型的文本文件，这个文件可以保存用户自定义的Keywords，或者是Variables等。

随后右击这个Resource文件，选择New User Keywords，创建用户关键字。例如我这里创建的就是login 关键字。

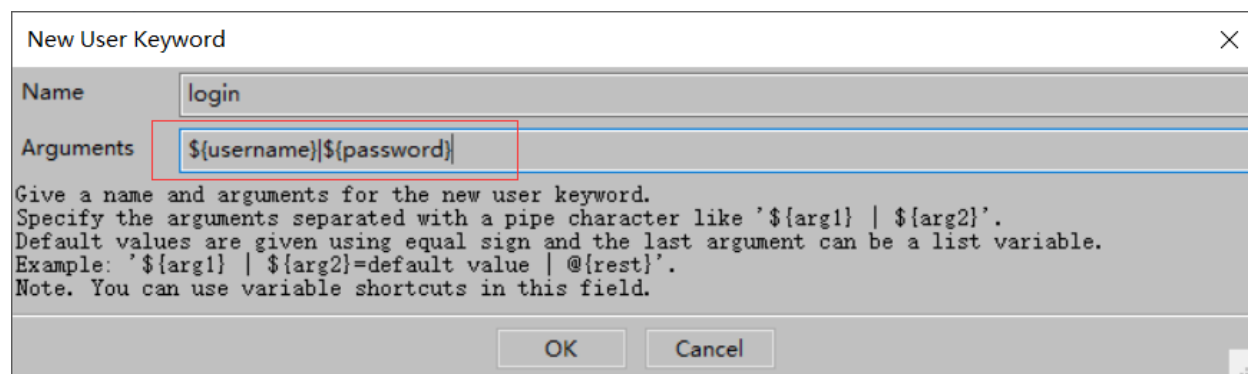


这个login和一个测试用例差不多，但区别是它可以到处引入，然后被调用。

现在来看一个简单的例子，即登录功能的Keywords封装。

封装第一个用户关键字：login

一个关键字就像一个函数，可以有入参，也可以有返回值。显然login是需要知道用户名密码这两个变量的，而剩下的部分（登陆界面url，浏览器驱动，页面元素定位器等），这些就都是常量了，可以从其他文件中引入，也可以写死，不用当作入参。



在创建这个用户关键字时，入参用`${}`标识，用`|`分隔。

假设我们知道了登陆界面url以及浏览器驱动等常量（之后我会讲如何从数据文件中引入这些变量）。现在就重点关注 workflow 吧。

1. 加载登录页面
2. 定位输入框，输入用户名密码
3. 定位并点击登录按钮
4. 若登陆失败，会弹出对话框，因此我们可以采用是否有对话框来检验登陆成功与否。
5. 返回登录状态信息。

#	打开浏览器并加载网页			
Open Browser		\${login}	\${ff}	
#	输入用户名密码			
Input Text	id = username		\${username}	
Input Password	id = password		\${password}	
#	点击登录按钮			
Click Button	name = submit			
#	检查登录是否成功			
Sleep	3			
\${alert}	Run Keyword And Return Status	Execute Javascript	return !!window.alert	
Run Keyword If	\${alert}	Log	用户\${username} 登录成功	
...	ELSE	Log	用户\${username} 登陆失败	
Return From Keyword	\${alert}			

这个里面的关键字都是SeleniumLibrary或者RF内建库暴露的关键字，网上搜下文档就会用了，而且语法非常脚本化，几乎就是自然语言了，这里对于这些关键字的学习我就不介绍了，因为很简单。

唯一值得一提的，是页面元素定位方式，例如：

```
id = username
id = password

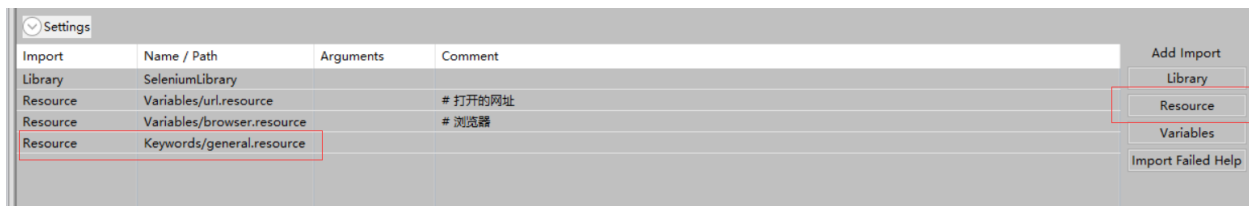
name = submit
```

这对应了css样式中的选择器。

```
<i class="user icon">...</i>
<input name="user_id" placeholder="用户名（学号）" type="text" id="username"> == $0
</div>
```

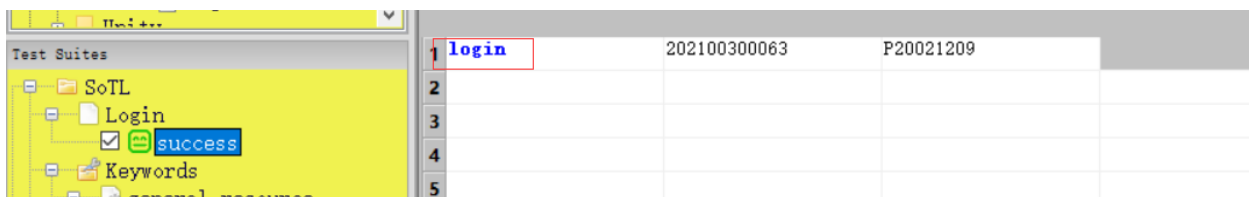
引入关键字

显然这个工作流程，是每个测试用例都需要使用的，因此我们可以在所有做登录测试的测试套件中引入他，这样所有登陆测试的测试用例全部都能用这个关键字。



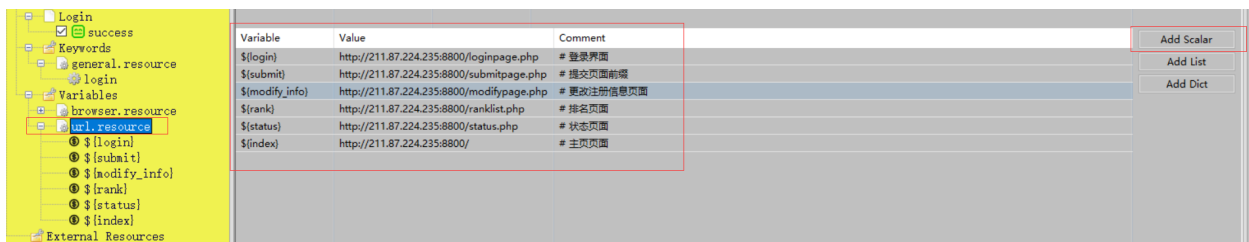
只要引入了一个resource文件，其中包括的所有User Keywords或者是Variables都会被引入。

这样就可以在测试用例中调用了。



Define Variables

定义变量跟定义关键字差不多。



创建完resource文件后，直接在里面添加变量即可。随后再把这个resource文件引入测试套件，里面的变量就都可用了。

尝试运行

在工作流、数据流准备、封装、引入、调用这一整套流程结束后，我们就可以试着运行一个测试用例了。

运行后，会自动启动firefox浏览器。



运行结果如下：

```
Starting test: SoTL.Login.success
20240517 14:45:04.193 : INFO : Opening browser 'firefox' to base url 'http://211.87.224.235:8800/loginpage.php'.
20240517 14:45:13.480 : INFO : Typing text '202100300063' into text field 'id = username'.
20240517 14:45:13.643 : INFO : Typing password into text field 'id = password'.
20240517 14:45:13.695 : INFO : Temporarily setting log level to: NONE
20240517 14:45:13.724 : INFO : Log level changed from NONE to INFO.
20240517 14:45:13.725 : INFO : Clicking button 'name = submit'.
20240517 14:45:16.966 : INFO : Slept 3 seconds
20240517 14:45:16.973 : INFO :
Executing JavaScript:
return !!window.alert
Without any arguments.
20240517 14:45:17.017 : INFO : ${alert} = True
20240517 14:45:17.022 : INFO : 用户202100300063登录成功
20240517 14:45:17.027 : INFO : Returning from the enclosing user keyword.
Ending test: SoTL.Login.success
```

断言通过，登陆成功。（这里我先不给测试报告，等之后我把测试用例都完善好，统一测完给测试报告）

5. OJ系统Web自动化测试：随机应变

接下来就是灵活发挥的时间。这个web应用的各个页面各有不同，定位元素的方式，检验测试结果的方式也大有不同，需要随机应变。

5.1. 登录功能

这个之前介绍过了，具体的测试用例我就用我之前实验三设计好的了。这里不再赘述。

1	# 打开浏览器并加载网页				
2	Open Browser	\${login}	\${ff}		
3	# 输入用户名密码				
4	Input Text	id = username	\${username}		
5	Input Password	id = password	\${password}		
6	# 点击登录按钮				
7	Click Button	name = submit			
8	# 检查登录是否成功				
9	\${alert}	Run Keyword And Return Status	Execute Javascript	return !!window.alert	
10	Run Keyword If	\${alert}	Log	用户\${username} 登录成功	
11	...	ELSE	Log	用户\${username} 登陆失败	
12	Return From Keyword	\${alert}			
13					
14					

这里返回登陆成功的状态（用是否弹出对话框判断是否登录成功），便于断言。

结果展示

Test Statistics							
Total Statistics		Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests		5	5	0	0	00:00:30	
Statistics by Tag		Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Login		5	5	0	0	00:00:30	
Statistics by Suite		Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Lab6		5	5	0	0	00:00:30	
Lab6.WebAutomation		5	5	0	0	00:00:30	
Lab6.WebAutomation.Login		5	5	0	0	00:00:30	
Test Details							
All Tags Suites Search							
Status: 5 tests total, 5 passed, 0 failed, 0 skipped							
Total Time: 00:00:29.784							
Name	Documentation	Tags	Status	Message	Elapsed	Start / End	
Lab6.WebAutomation.Login.pwd_err		Login	PASS		00:00:05.738	20240518 12:30:38.130	
Lab6.WebAutomation.Login.success		Login	PASS		00:00:05.478	20240518 12:30:32.651	
Lab6.WebAutomation.Login.uname_err		Login	PASS		00:00:06.056	20240518 12:30:38.129	
Lab6.WebAutomation.Login.uname_format_err		Login	PASS		00:00:06.359	20240518 12:30:49.925	
Lab6.WebAutomation.Login.uname_not_exist		Login	PASS		00:00:06.153	20240518 12:30:49.925	

我用的几个测试用例：

- 1. 成功登录
- 2. 用户名不匹配

3. 密码不匹配
4. 用户不存在
5. 用户名格式错误

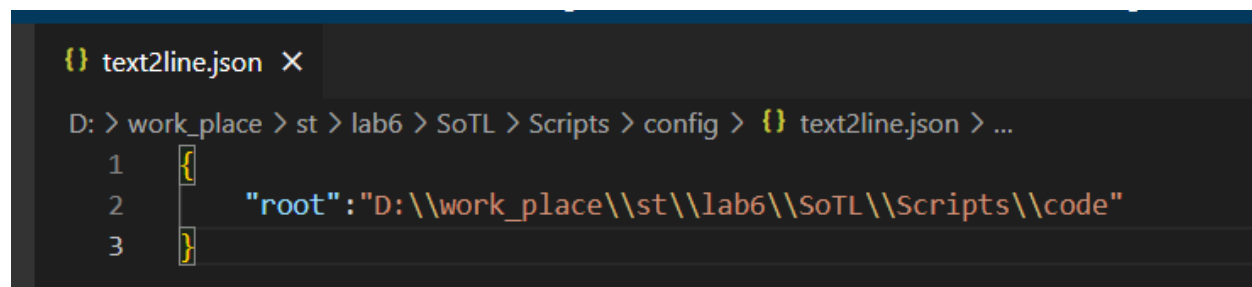
5.2. 代码评测功能

如何获取输入框并进行输入，我在实验3中讲过一遍了。

二次处理代码文本

在rf中，去除代码中的换行符比较麻烦。因此我选择另写脚本处理代码文本(之前在实验三时就是内嵌的代码，处理文本的换行符)。不去除换行符的话，这个在线编辑器会自动补全代码（例如大括号），造成不必要的CE。

首先我先创建一个配置文件，把所有代码源文件的根目录放在里面。



```
{ } text2line.json X
D: > work_place > st > lab6 > SoTL > Scripts > config > { } text2line.json > ...
1  {
2    "root": "D:\\work_place\\st\\lab6\\SoTL\\Scripts\\code"
3  }
```

随后我读取这个配置文件，然后遍历这个目录下的所有代码源文件，然后读取每个源文件中的文本，将其拼接为一个字符串，然后通过正则表达式替换掉他的换行符，随后重新写入，这样就得到了仅有一行的代码源文件，可以将其作为参数，填入到textarea中。

```
import os
import re
import json
import sys

def text2line(file_path:str):
    print(f'正在处理任务:${file_path}')
    with open(file_path,"r",encoding = 'utf-8') as file:
        text = file.readlines()
        file.close()
```

```

with open(file_path, "w", encoding = 'utf-8') as file:
    text = ''.join(text)
    text = re.sub(r'\r|\n', '', text)
    print(text)
    file.truncate(0)
    file.write(text)

if __name__ == '__main__':

    config_path = sys.argv[1]

    with open(config_path, 'r', encoding = 'utf-8') as file:
        config = json.load(file)

    root = config['root']
    assert os.path.isdir(root)

    files = os.listdir(root)

    for src in files:
        text2line(os.path.join(root, src))

```

```

管理员: C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19044.4412]
(c) Microsoft Corporation. 保留所有权利。

D:\work_place\st\lab6\SoTL\Scripts>python .\text2line.py D:\work_place\st\lab6\SoTL\Scripts\config\text2line.json
正在处理任务:$D:\work_place\st\lab6\SoTL\Scripts\code\1000_CE
正在处理任务:$D:\work_place\st\lab6\SoTL\Scripts\code\1000_MLE
正在处理任务:$D:\work_place\st\lab6\SoTL\Scripts\code\1003_TLE
正在处理任务:$D:\work_place\st\lab6\SoTL\Scripts\code\1004_RE
正在处理任务:$D:\work_place\st\lab6\SoTL\Scripts\code\1005_WA
正在处理任务:$D:\work_place\st\lab6\SoTL\Scripts\code\1006_PE
正在处理任务:$D:\work_place\st\lab6\SoTL\Scripts\code\1008_AC

```

随后我们用：

```
(Get-Content -Path .\1008_AC).Count
```

的命令，查看文件中的代码行数。

```
PS D:\work_place\st\lab6\SoTL\Scripts\code> (Get-Content -Path .\1008_AC).Count
1
```

这样我们就准备好了代码文本。

登录保存cookie

想进行代码评测，必须要在登陆状态下进行。但webdriver每次启动的浏览器，都不带之前的cookie，所以需要先登录保存cookie，才能接着进入评测功能。

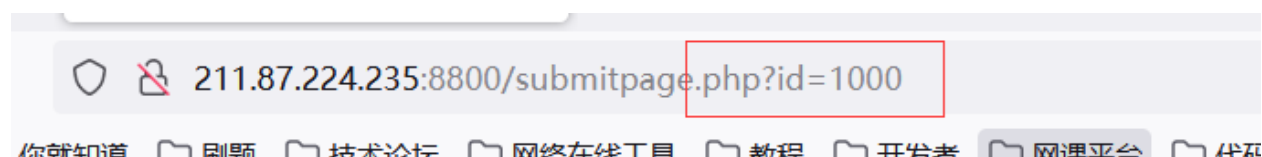


所以每次提交代码前，我们都先调用一遍login的过程，先存上cookie再跳转提交代码界面：

1	<code>\${token}</code>	<code>login</code>	<code>\${username}</code>	<code>\${password}</code>	
2	<code>submit code</code>	1008	<code>\${1008_AC}</code>	2	# 0 1 2 3 C C++ Java Python
3	<code>check submit result</code>	<code>\${username}</code>	正确		
4					

根据id选择页面

我们每次执行的测试用例可能用了不同的题目。所以根据submitpage的要求：要在get参数中带一个id=xxx的参数指定页面：

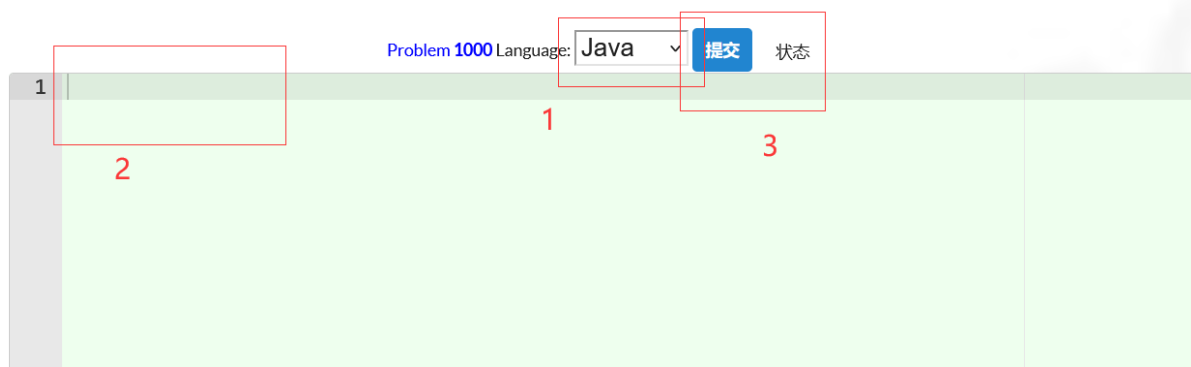


1	# 拼接url		
2	<code>\${submit_url}</code>	Set Variable	<code>\${submit}?id=\${id}</code>
3	# 打开对应页面		
4	Go To	<code>\${submit_url}</code>	

提交代码的流程

三步走：

1. 选择语言
2. 填写代码
3. 点击提交



选语言很简单，F12看一下这个框id叫啥就行：

```
<input id="problem_id" type="hidden" value="1000" name="problem_id">
<span id="language_span">
  Language:
  <select id="language" name="language"
    onchange="reloadtemplate($(this).val());">...</select>
  <button id="Submit" class="ui primary icon button">提交</button>
```

# 填充语言选择			
Select From List By Index	id = language	<code>\${lang}</code>	# 这个lang是索引，而非值

选代码文本框就复杂了，这里我个我做的视频讲解链接：

山东大学2021级软件测试实验3：谈一个用了一下午才解决的测试模块——怎样找到OJ系统代码编辑器的文本框_哔哩哔哩_bilibili

总之，要输入代码，一定有一个能接收输入的地方。我们F12点击一下，可以看到这么一长串里面的元素，只有两个可以输入交互：

```
if="false"> ... </button> event
▼ <pre id="source" class=" ace_editor ace-xcode" style="width:
  90%; height: 747px; font-size: 18px;" cols="180" rows="16">
  event
  <textarea class="ace_text-input" wrap="off"
    autocorrect="off" autocapitalize="none" spellcheck="false"
    style="opacity: 0; left: -100px;"></textarea> event
  > <div class="ace_gutter" aria-hidden="true"> ... </div> event
  ▼ <div class="ace_scroller" style="left: 43px; right: 0px;
    bottom: 0px;"> event
    > <div class="ace_content" style="margin-top: 0px; width:
      768px; height: 788.2px; margin-left: 0px;"> ... </div>
    </div>
    > <div class="ace_scrollbar ace_scrollbar-v" style="display:
      none; width: 22px; bottom: 0px;"> ... </div> event
    > <div class="ace_scrollbar ace_scrollbar-h" style="display:
      none; height: 22px; left: 43px; right: 0px;"> ... </div> event
    > <div style="height: auto; width: auto; top: 0px; left: 0px;
      visibility: ...absolute; white-space: pre; font: inherit;
      overflow: hidden;"> ... </div>
    </pre>
    <input id="hide_source" type="hidden" name="source" value="">
    <style> ... </style>
    > <div class="row"> ... </div>
  </form>
</center>
```

第一个是textarea，另一个是input框。

而我第一次就选错了，我以为是input框。为什么呢？因为我翻网站源码时，看到了这个：

```

        document.getElementById("frmSolution").submit();
    }

    function do_submit() {
        if (using_blockly)
            translate();
        if (typeof (editor) != "undefined") {
            $("#hide_source").val(editor.getValue());
        }
        var mark = "<?php echo isset($id) ? 'problem_id' : 'cid'; ?>";
        var problem_id = document.getElementById(mark);
        if (mark == 'problem_id')
            problem_id.value = '<?php if (isset($id))
                echo $id ?> ';
            problem_id.value = '<?php if (isset($cid))
                echo $cid ?> ';
        document.getElementById("frmSolution").target = "_self";
        document.getElementById("frmSolution").submit();
    }
    var handler_interval;

```

那么可以看到，这个input框的id就是hide_source。所以真正最后被提交到后端的代码的文本来源，就来自于这个input框。

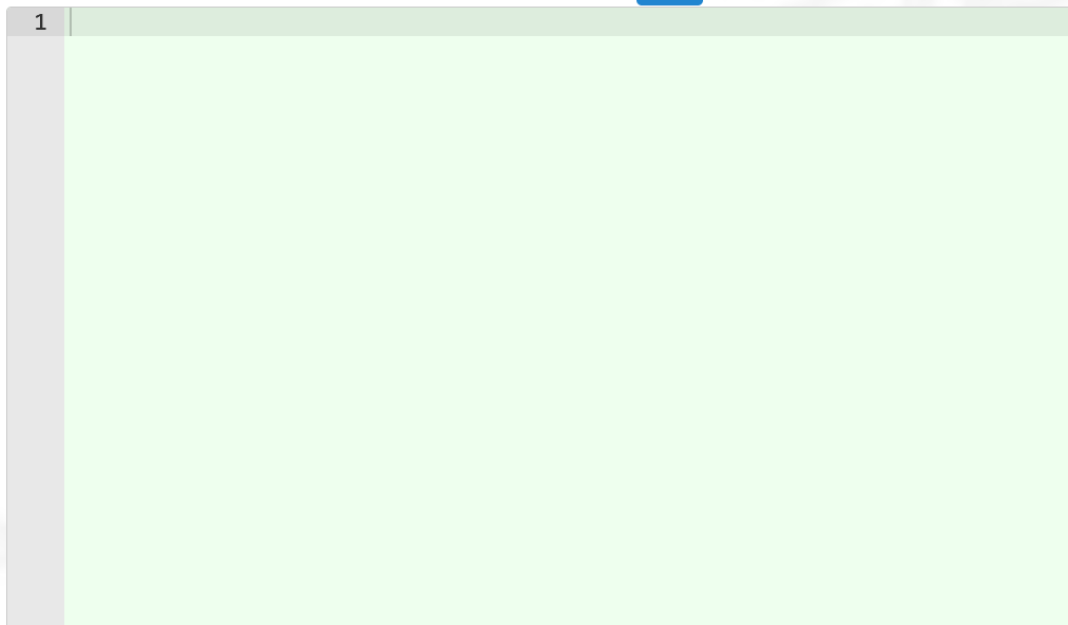
所以一开始我的想法是，修改这个input框的value，然后点提交按钮就行了。但是这样有个问题：

```

$("#hide_source").val(editor.getValue());

```

我们可以看到input框的val值，来自于editor的getValue，那么editor是谁呢？经过我翻阅源码，editor指的就是代码编辑器组件：



所以input框的值来自于这个编辑器，当用户在编辑器输入时，他会自动更新到这个hidden的框中。

可以这样证明，把这个input框的type从hidden改成text。可以发现，显示出来的input框的内容和代码编辑器中的内容一样。



因此只要我们动不到代码编辑器，每时每刻都会把input框的内容更新成代码编辑器的内容，也就是空。

那么怎么办呢？还剩下一个可以用作输入的，也即textarea。

不过需要注意的是，textarea的css样式中有这么一段：

```
<div class="ace gutter" aria-hidden="true">...</div> event  
<textarea class="ace_text-input" wrap="off"  
autocorrect="off" autocapitalize="none" spellcheck="false"  
style="opacity: 0; left: 80px; height: 20px; width:  
10.666/px; top: 0px;"></textarea> event
```

style.opacity = 0，意味着这个文本框是没有体积的。而webdriver得原则就是，没有体积的容器是不能输入的！

因此我们要修改这个容器，使得其体积不为0，随后就能向里面输入了。

而在rf中，仅仅是修改opacity还不够，因为这个容器的left初始值是-100，不在父容器内。

```
cols="180" rows="16"> event
<textarea class="ace_text-input" wrap="off" autocorrect="off" autocapitalize="none"
spellcheck="false" style="opacity: 0; left: -100px;"></textarea> event
```

所以这次我们还得把这个子容器的left改为0，让他能在父容器中被访问，才能进行交互，填充代码。

7	# 填充代码			
8	\${text_area}	Get WebElement	tag = textarea	# 获取textarea元素
9	\${js_code}=	Catenate	SEPARATOR=;	arguments[0].style.opac
				=
				1;arguments[0].style.le
				= 0;
10	Execute JavaScript	\${js_code}	ARGUMENTS	\${text_area}

这里利用Excute JavaScript关键字执行js脚本，修改css属性。

线程睡眠的软同步

先来说还有啥问题。因为我想做的是，每次自动提交跑完之后，自动获取结果，然后输出出来：

提交编号	用户	昵称	题目编号	结果	内存	时间	语言	代码长度	提交时间
2198	202100300063	lyh	1006	正确	20744 KiB	260 ms	Java/Edit	1656 bytes	2024-04-28 18:38:17

不过这里有个问题，我连着执行7个测试用例，而oj系统对于判题而言是要排队的。所以可能第二个题还没判完，在排队的时候，我这边脚本自动去获取结果了，那么获取的就是上一个测试用例的结果。这样就串了。

所以我采用线程睡眠的方式来实现同步：

4	Sleep	10	# 10s内测评必定结束
---	-------	----	--------------

不过为啥说是软同步呢？因为假设我一个测试用例，排队排了无限长的时间，这个时候无论睡多久，获取的都是上一个的值。

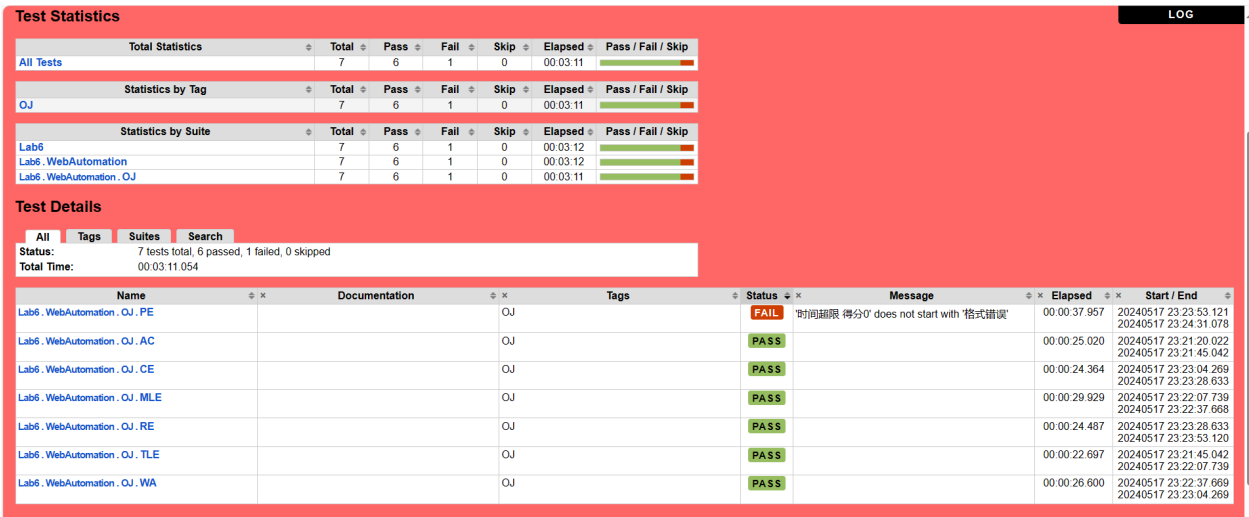
不过大部分时候这种情况不会发生，所以就这样吧。

结果展示

我在每个测试中都加入了断言，判断评测结果是否和应该出现的结果对的上。判断方式是应该出现的结果是真正结果的前缀。

5	@{tds}	Get WebElements	class = label	
6	\${res}	Set Variable	\${tds}[0]	
7	\${result}	Get Text	\${res}	
8	Log	代码评测结果为\${result}		
9	Should Start With	\${result}	\${should_be_result}	

最终是挂了一个测试用例。



可以看到，AC\CE\RE\WA\MLE\TLE都没问题。而PE（格式错误）却输出TLE。并且我利用web ui界面进行测试，又输出了RE（都是用一样的代码）。

提交编号	用户	昵称	题目编号	结果	内存	时间	语言	代码长度	提交时间
6923	202100300063	lyh	1006	运行错误 得分0	1065 KiB	1887 ms	Java/Edit	1532 bytes	2024-05-17 23:44:23

可以发现1006题的评测表现很不稳定，实验三是输出了AC，自动化测试是TLE，手动测试是RE，是存在缺陷的。

5.3. 注册功能

这个也是一个很trivial的part。

Open Browser	\${register}	\${ff}		
Input Text	name = user_id	\${uname}		
Input Text	name = nick	\${nickname}		
Input Text	name = password	\${pwd}		
Input TEXT	name = rptpassword	\${repeat}		
Input Text	name = school	\${school}		
Input Text	name = email	\${email}		
Click Button	name = submit			
\${alert}	Run Keyword And Return Status	Execute Javascript	return !!window.alert	
Run Keyword If	\${alert}	Log	用户\${uname} 登录成功	
...	ELSE	Log	用户\${uname} 登陆失败	
Return From Keyword	\${alert}			

把该填的内容填上（当然传进来的参数也可以是空），然后提交后返回注册成功与否的结果就行了。

结果展示

Test Statistics

Total Statistics

All Tests

Total

6

Pass

6

Fail

0

Skip

0

Elapsed

00:00:44

Pass / Fail / Skip

Statistics by Tag

Register

Total

6

Pass

6

Fail

0

Skip

0

Elapsed

00:00:44

Pass / Fail / Skip

Statistics by Suite

Lab6

Lab6.WebAutomation

Lab6.WebAutomation.Register

Total

6

Pass

6

Fail

0

Skip

0

Elapsed

00:00:44

Pass / Fail / Skip

Test Details

AllTagsSuitesSearch

Status:6 tests total, 6 passed, 0 failed, 0 skipped

Total Time:00:00:44.178

Name

Documentation

Tags

Status

Message

Elapsed

Start / End

Lab6.WebAutomation.Register.pwd_rpt_err

Register

PASS

00:00:07.719

20240518 12:58:40.434
20240518 12:58:48.153

Lab6.WebAutomation.Register.pwd_too_short

Register

PASS

00:00:06.844

20240518 12:58:33.587
20240518 12:58:40.431

Lab6.WebAutomation.Register.school_too_long

Register

PASS

00:00:10.954

20240518 12:58:48.153
20240518 12:58:59.107

Lab6.WebAutomation.Register.success

Register

PASS

00:00:06.100

20240518 12:58:14.824
20240518 12:58:21.024

Lab6.WebAutomation.Register.uname_format_err

Register

PASS

00:00:06.364

20240518 12:58:21.025
20240518 12:58:27.389

Lab6.WebAutomation.Register.uname_too_short

Register

PASS

00:00:06.197

20240518 12:58:27.389
20240518 12:58:33.586

这里我用的测试用例如下：

1. 成功注册
2. 用户名格式错误
3. 用户名太短
4. 密码太短

- 5. 密码重复错误
- 6. 学校名太长

5.4. 修改个人信息

和注册一样trivial。

1	Go To	\${modify_info}		
2	Input Text	name = nick	\${nickname}	
3	Input Text	name = opassword	\${opwd}	
4	Input Text	name = npassword	\${npwd}	
5	Input Text	name = rptpassword	\${rpt}	
6	Input Text	name = school	\${school}	
7	Input Text	name = email	\${email}	
8	Click Button	name = submit		
9	\${alert}	Run Keyword And Return Status	Execute Javascript	return !!window.alert
10	Run Keyword If	\${alert}	Log	用户\${uname} 修改成功
11	...	ELSE	Log	用户\${uname} 修改失败
12	Return From Keyword	\${alert}		
13				

先登录，然后跳转，该填的填了（不修改的话填原来的值就可以），然后点击提交。查看并返回结果。

结果展示

Test Statistics

All Tests

Statistics by Tag

Modify

Total Statistics

Statistics by Suite

Lab6

Lab6.WebAutomation

Lab6.WebAutomation.ModifyInfo

Total

Pass

Fail

Skip

Elapsed

Pass / Fail / Skip

5

4

1

0

00:00:37

5

4

1

0

00:00:37

5

4

1

0

00:00:37

5

4

1

0

00:00:37

5

4

1

0

00:00:37

Test Details

AllTagsSuitesSearch

Status:5 tests total, 4 passed, 1 failed, 0 skipped

Total Time:00:00:36.555

Name

Documentation

Tags

Status

Message

Elapsed

Start / End

Lab6.WebAutomation.ModifyInfo.school_too_long

Modify

FAIL

True != False

00:00:08.941

20240518 13:45:52.514
20240518 13:46:01.455

Lab6.WebAutomation.ModifyInfo.npwd_too_short

Modify

PASS

00:00:06.873

20240518 13:45:37.799
20240518 13:45:44.672

Lab6.WebAutomation.ModifyInfo.pwd_err

Modify

PASS

00:00:06.303

20240518 13:45:31.496
20240518 13:45:37.799

Lab6.WebAutomation.ModifyInfo.rpt_err

Modify

PASS

00:00:07.840

20240518 13:45:44.673
20240518 13:45:52.513

Lab6.WebAutomation.ModifyInfo.success

Modify

PASS

00:00:06.598

20240518 13:45:24.898
20240518 13:45:31.496

这里我用到的几个测试用例：

- 1. 成功修改
- 2. 旧密码错误
- 3. 新密码过短
- 4. 重复密码错误
- 5. 学校名称过长

可以看到，还是老缺陷，学校名过长，应该是不能修改成功的，但是这里修改成功了。和注册时的校验不一致。

5.5. 问题搜索

跳转页面，填上数据，模拟回车即可。

注意这个网站的逻辑是如果没有输入，那么不能在对应的框内回车，否则视为输入空字符串。所以这里写了个条件分支，判断输入是否为空，在不为空的输入框内模拟回车。

Open Browser	\${problemset}	\${ff}		
Input Text	name = search	\${title}		
Input Text	name = id	\${id}		
IF	'\${id}' != '\${EMPTY}'			
	Press Key	name: id	\\13	
ELSE				
	Press Key	name: search	\\13	
END				

结果展示

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	5	5	0	0	00:00:33	

Statistics by Tag

Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
5	5	0	0	00:00:33	

Statistics by Suite

Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip	
Lab6	5	5	0	0	00:00:33	
Lab6_WebAutomation	5	5	0	0	00:00:33	
Lab6_WebAutomation_ProblemSearch	5	5	0	0	00:00:33	

Test Details

AllTagsSuitesSearch

Status:5 tests total, 5 passed, 0 failed, 0 skipped

Total Time:00:00:32.773

Name	Documentation	Tags	Status	Message	Elapsed	Start / End
Lab6_WebAutomation_ProblemSearch_ByExistID		Problem	PASS		00:00:06.894	20240518 14:31:47.182 20240518 14:31:54.076
Lab6_WebAutomation_ProblemSearch_ByNotExistID		Problem	PASS		00:00:06.264	20240518 14:31:54.077 20240518 14:32:00.341
Lab6_WebAutomation_ProblemSearch_FuzzyByTitle		Problem	PASS		00:00:06.402	20240518 14:32:00.342 20240518 14:32:06.744
Lab6_WebAutomation_ProblemSearch_None		Problem	PASS		00:00:06.695	20240518 14:32:13.264 20240518 14:32:19.959
Lab6_WebAutomation_ProblemSearch_PreciseByTitle		Problem	PASS		00:00:06.518	20240518 14:32:06.745 20240518 14:32:13.263

可以看到，无论是否有搜索结果，这套搜索流程都是可以正常完成的。（没有搜索结果是因为根本不存在对应问题）

5.6. 状态查看

填写四个参数（当然也可以不填，不过选择框必须要选），然后提交表单进行查询。

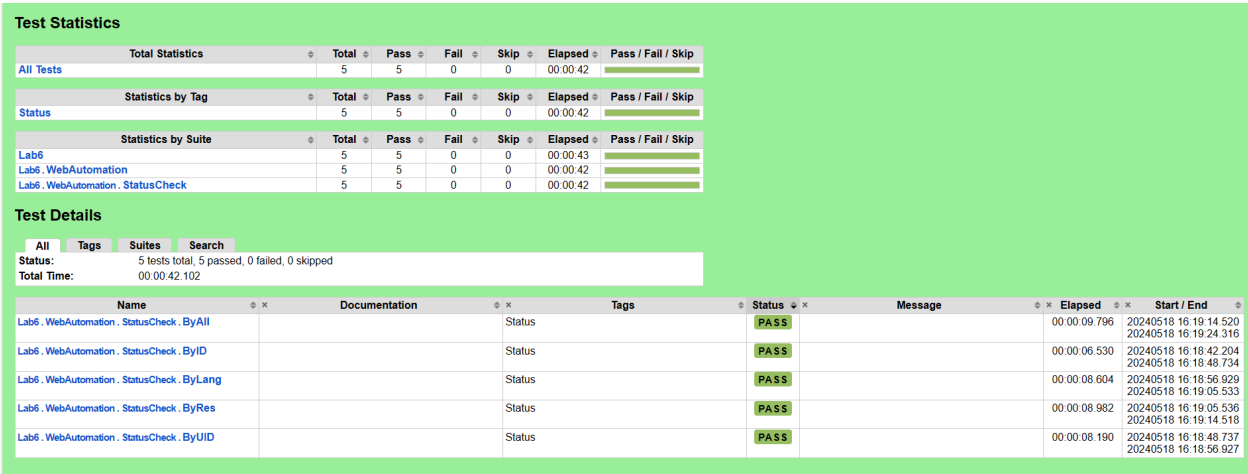
1	Open Browser	<code>\${status}</code>	<code>\${ff}</code>	
2	Input Text	<code>name = problem_id</code>	<code>\${id}</code>	
3	Input Text	<code>name = user_id</code>	<code>\${uid}</code>	
4	Select From List By Index	<code>name = language</code>	<code>\${lang}</code>	
5	Select From List By Index	<code>name = jresult</code>	<code>\${res}</code>	
6	Click Button	<code>tag = button</code>		
7				

结果展示

我用的测试用例如下：

- 1. 根据用户id
- 2. 根据题目id
- 3. 根据语言
- 4. 根据结果种类

5. 所有综合



可以看到这个功能是没有问题的。

5.7. 排名查看

填写用户id，然后点击查看。（这里支持模糊匹配：前缀匹配）

1	Open Browser	<code>\${rank}</code>	<code>\${ff}</code>
2	Input Text	<code>name = prefix</code>	<code>\${uid}</code>
3	Click Button	<code>tag = button</code>	
4			

结果展示

我用的测试用例是：

- 1. 精准id匹配
- 2. 模糊id匹配

Test Statistics							
Total Statistics		Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests		2	2	0	0	00:00:13	
Statistics by Tag		Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Rank		2	2	0	0	00:00:13	
Statistics by Suite		Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Lab6		2	2	0	0	00:00:14	
Lab6 .WebAutomation		2	2	0	0	00:00:14	
Lab6 .WebAutomation .RankCheck		2	2	0	0	00:00:14	
Test Details							
<div> <div>All</div> <div>Tags</div> <div>Suites</div> <div>Search</div> </div> <div> <div>Status:</div> <div>2 tests total, 2 passed, 0 failed, 0 skipped</div> </div> <div> <div>Total Time:</div> <div>00:00:13.274</div> </div>							
Name	Documentation	Tags	Status	Message	Elapsed	Start / End	
Lab6 .WebAutomation .RankCheck .AccurateID		Rank	PASS		00:00:06.706	20240518 16:29:11.269 20240518 16:29:17.975	
Lab6 .WebAutomation .RankCheck .FuzzyID		Rank	PASS		00:00:06.568	20240518 16:29:17.976 20240518 16:29:24.544	

可以看到这个查询也没有问题。

6. 集成RF-Requests库完成接口自动化测试

这个主要依赖于robotframework-requests库，用来直接对服务端接口进行请求。

6.1. 代码评测功能

接口鉴权的准备：从会话中获取cookie

第5节中我们使用selenium扩展库时，进行测试的方式是模仿用户点击、输入等操作。因此当时我们并没有显式地保存cookie，而是选择利用Go To跳转页面，从而使用保存在浏览器应用进程内存中的cookie。

不过现在我们需要直接请求服务端接口了，就需要自己在请求中附上cookie。这一步是所有需要鉴权的接口都要用到的。包括最重要的代码评测功能。因此我们第一步要做的，就是完善这个获取cookie的过程。

源码解析：何以登录？

首先我们翻阅源码：<https://github.com/zhblue/hustoj.git>，查看trunk/web下的loginpage.php文件：

```
<form id="login" action="login.php" method="post" role="form"
class="form-horizontal" onSubmit="return jsMd5();" >

function jsMd5(){
    if($("#input[name=password]").val()=="") return false;
    $("#input[name=password]").val(hex_md5($("#input[name=password]").val()));
}
```

```
    return true;
}
```

可以看到，登录表单提交时会先执行一个叫做jsMd5()的函数，用来将密码加密传到服务端进行校验，随后会请求login.php这个接口，用post的方式发送请求体中的用户名密码。

那么我们在传输数据时也需要使用这个js库函数进行加密，而这在request库中很难做到。

因此最终我选择的解决方案是：**selenium与request库综合使用**。先确认一下这个功能点的测试流程：

1. 登录拿到cookie。并保存在会话中。
2. 将cookie附在请求头中，请求判题的接口
3. 查询结果，判断与预期结果是否一致。

在这个过程中，登录拿到cookie用selenium做。请求判题接口用rf-requests库做，查询结果用selenium做（其实也可以用rf-requests库做，不过之前第5节中我已经封装好了一个查询结果的selenium的关键字，直接复用了）。

换句话说，我们利用**浏览器维护内存中的cookie**的功能，**维护session会话不要掉线**，从而在登陆后可以直接**使用拿到的cookie请求服务端接口而不被鉴权模块拦截**。

登录获得cookie

这里的脚本和之前第5节中的脚本就有了差别，当时第5节我们判断是否登录成功后就把状态码返回了，但现在接口测试中，需要cookie。所以我使用了Get Cookies这个关键字，获得所有cookie。

1	# 打开浏览器并加载网页			
2	Open Browser	\${login}	\${ff}	
3	# 输入用户名密码			
4	Input Text	id = username	\${uid}	
5	Input Password	id = password	\${pwd}	
6	# 点击登录按钮			
7	Click Button	name = submit		
8	# 检查登录是否成功			
9	\${alert}	Run Keyword And Return Status	Execute Javascript	return !!window.alert
10	Run Keyword If	\${alert}	Log	用户\${uid} 登录成功
11	...	ELSE	Log	用户\${uid} 登陆失败
12	&{cookies}	Get Cookies	True	# 以字典形式获取cookie
13	Return From Keyword	&{cookies}		
14				
15				

请求服务端接口

首先我们需要维护一个Session会话，让服务端认为我们之前是进行过登陆的。他会在这个Session的上下文中搜寻cookie，如果找到能够通过鉴权的cookie，那么我们就可以请求服务端接口了。

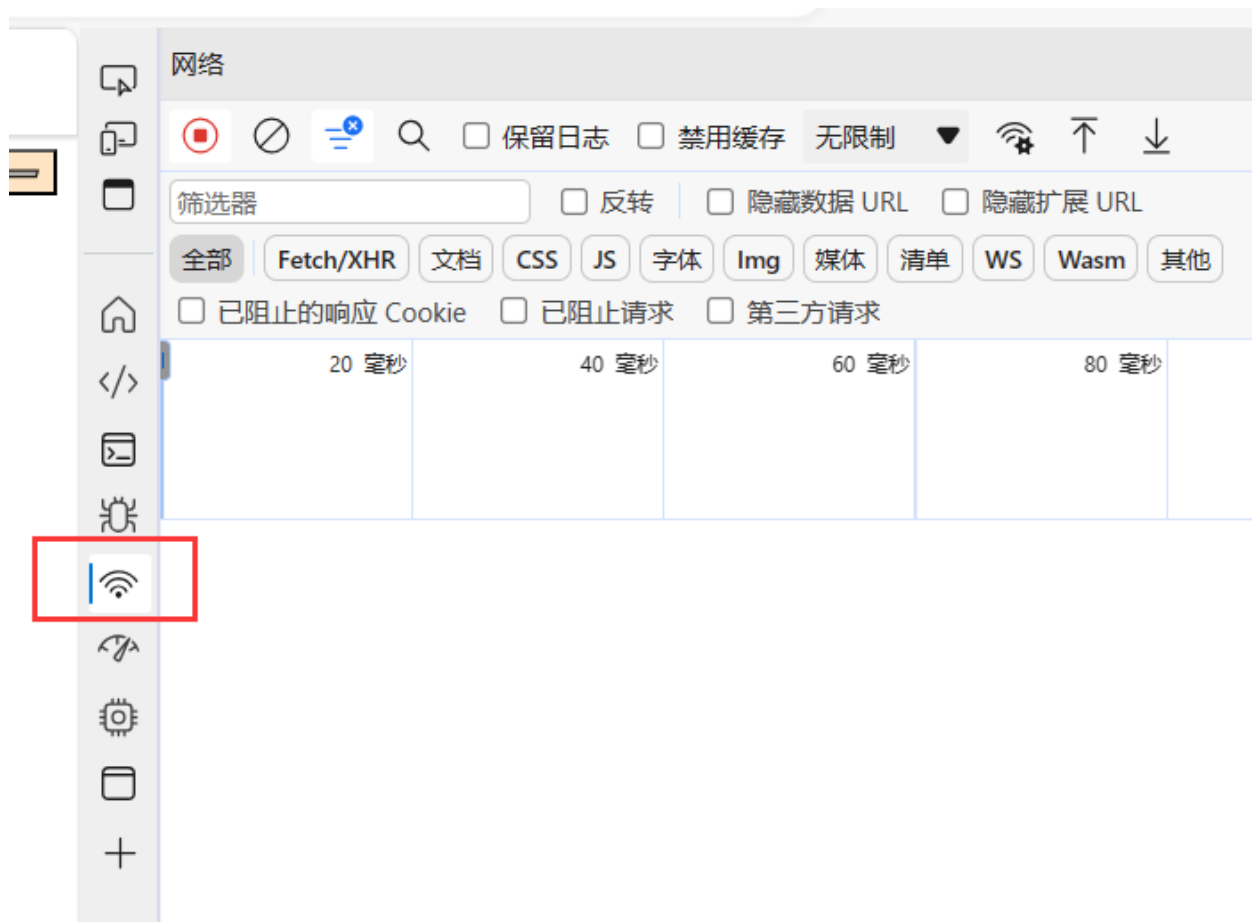
1	Create Session	oj	\${ip}	\	\${cookies}
2	Log	oj			
3	\${payload}=	Create Dictionary	id=\${id}	language=\${lang}	source=\${source}
4	\${response}=	POST On Session	oj	/submit.php	data=\${payload}
5	Log	\${response.content}			
6	Status Should Be	200	# 应该得到正确响应		
7					
8					

这里我通过rf-requests库的Create Session关键字，创建了一个名为oj的session，其中服务器地址为\${ip}，并且我把之前获取到的cookies置入了这个session。

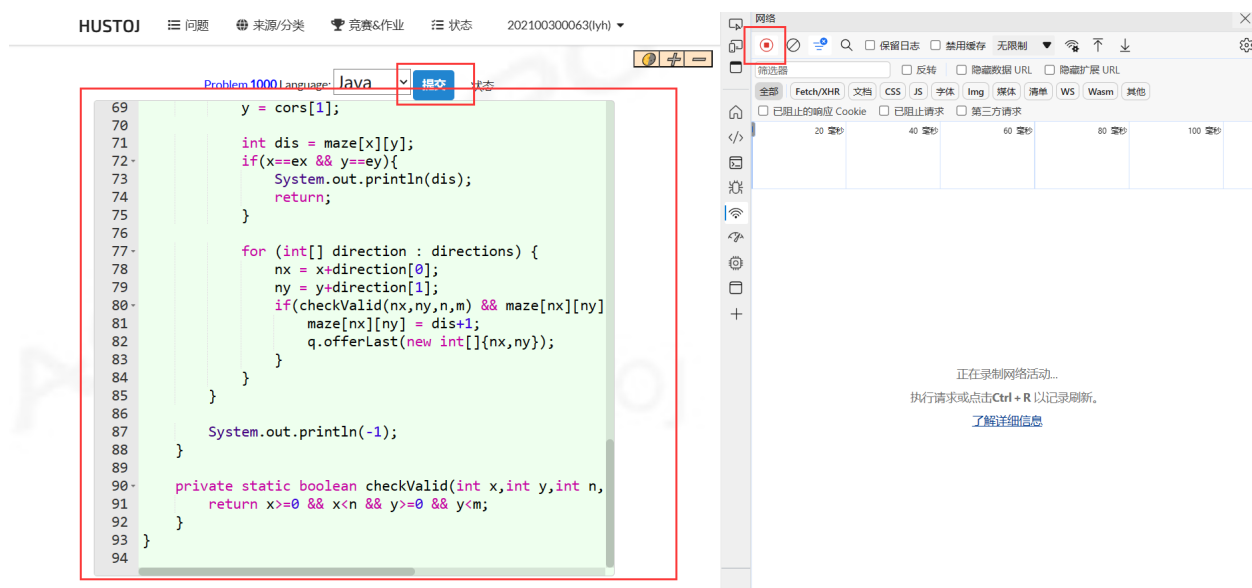
此外，我们需要构建一个请求体，用来将提交题目时的信息置入负载中。那么此时一个很关键的问题是，请求体长成什么样？

获取请求体结构

首先我们先f12打开开发者工具，找到网络选项，这个功能可以监测所有的网络请求。



点击录制。然后我们填好语言和代码后，点击提交。



录制结束后，即可看到所有请求。

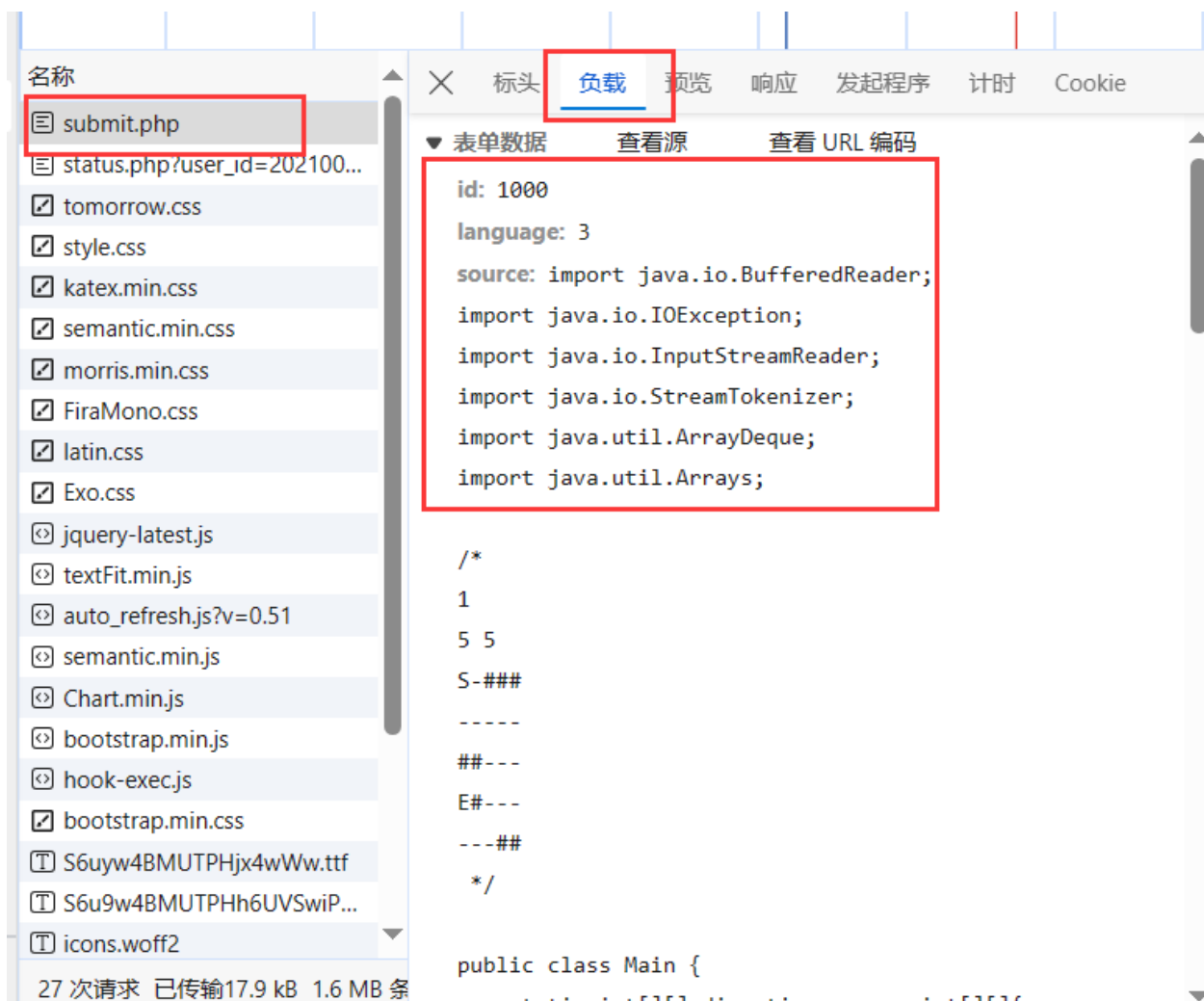
名称	状态	类型	发起程序	大小	时间	腹...	时间线
submit.php	302	doc...	其他	414 B	17 ...		
status.php?user_id...	200	doc...	:8800/submi	8.5 kB	22 ...		
tomorrow.css	200	styl...	:8800/status	0 B	0 ...	(me...	
style.css	200	styl...	:8800/status	0 B	1 ...	(dis...	
katex.min.css	200	styl...	:8800/status	0 B	0 ...	(me...	
semantic.min.css	200	styl...	:8800/status	0 B	5 ...	(dis...	
morris.min.css	200	styl...	:8800/status	0 B	0 ...	(me...	
FiraMono.css	200	styl...	:8800/status	0 B	0 ...	(me...	
latin.css	200	styl...	:8800/status	0 B	0 ...	(me...	
Exo.css	200	styl...	:8800/status	0 B	0 ...	(me...	
jquery-latest.js	200	script	:8800/status	0 B	0 ...	(me...	
textFit.min.js	200	script	status.php?u	0 B	2 ...	(dis...	
auto_refresh.js?v=0...	200	script	status.php?u	0 B	2 ...	(dis...	
semantic.min.js	200	script	status.php?u	0 B	4 ...	(dis...	
Chart.min.js	200	script	status.php?u	0 B	3 ...	(dis...	
bootstrap.min.js	200	script	status.php?u	0 B	2 ...	(dis...	
hook-exec.js	200	script	hook.js:1	6.9 kB	134 ...		
bootstrap.min.css	200	styl...	style.css:1	0 B	0 ...	(me...	
S6uyw4BMUTPHjx...	200	font	latin.css	0 B	0 ...	(me...	
S6u9w4BMUTPHh...	200	font	latin.css	0 B	0 ...	(me...	
icons.woff2	200	font	semantic.mi	0 B	0 ...	(me...	

27 次请求 已传输17.9 kB 1.6 MB 条资源 完成: 611 毫秒 DOMContentLoaded: 456 毫秒 加载: 613 毫

翻看源码，得知提交表单后请求的是submit.php这个接口：

```
<form id=frmSolution action="submit.php" method="post" onsubmit='do_submit()>
<?php if (isset($id)){>
```

查看submit.php这个请求的负载：



这回我们就知道了，请求体的格式是：

```
{
  id:
  language:
  source:
}
```

因此我们在构造请求体时，也需要用这样的格式：

```
{ ${payload}= Create Dictionary id=${id} language=${lang} source=${source}
```

随后我们基于现在的session会话发post请求，请求体为刚刚构造的payload。

<code>\${response}=</code>	POST On Session	oj	/submit.php	data=\${payload}
----------------------------	-----------------	----	-------------	------------------

最后，无论我们的提交得到一个什么样的判题结果（AC,WA,MLE,TLE等等.....），这个请求都是应该得到响应的，所以最后我加了个断言，响应码理应是200。

Status Should Be	200	# 应该得到正确响应
------------------	-----	------------

检查提交结果

这一部分在第5节中已经展示过了。此处不再赘述。

总体调用流程

1	<code>&{cookies}</code>	login	<code>\${uid}</code>	<code>\${pwd}</code>	获取cookie
2	Log	<code>\${cookies}</code>	# 打印cookie		
3	<code>\${id}</code>	Convert To Integer	1000		
4	<code>\${lang}</code>	Convert To Integer	3	# 0 1 3 6 C C++ Java Python	
5	submit code	<code>\${id}</code>	<code>\${lang}</code>	<code>\${1000_AC}</code>	<code>&{cookies}</code>
6	check submit result	<code>\${uid}</code>	正确		带着cookie请求
7	检查判题结果是否正确				
8					

注意在选择语言时，我们传入的是option元素的value。这个value并不是option在父级select元素中的索引。而是：

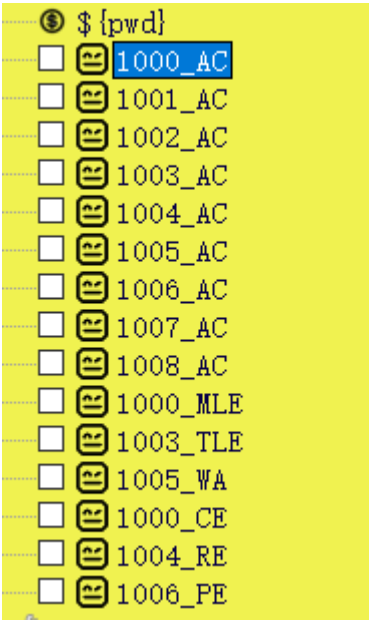
```
<option value="0"> C </option> slot
<option value="1"> C++ </option> slot
<option value="3" selected> Java </option> slot
<option value="6"> Python </option> slot
```

手动赋过值的。

结果展示

在5.18日晚我做代码评测功能测试时，出现了这么个情况：同一份代码交上去会有3个不同的评测结果，比如原先AC的代码再交一次是MLE，再交一次变RE了。可能评测姬内部崩掉了（注意时间，5.18晚）

因此这次我的接口测试挂了一多半测试用例：



我先是准备了所有题目的AC代码，然后准备了MLE\TLE\WA\CE\PE\RE的代码，结果如下：

AllTagsSuitesSearch

Status:15 tests total, 7 passed, 8 failed, 0 skipped

Total Time:00:03:05.721

Name	Documentation	Tags	Status	Message	Elapsed	Start / End
Lab6.ApiAutomation.OJ.1000_AC		PostOJ	FAIL	'运行错误 得分0' does not start with '正确'	00:00:10.393	20240518 21:24:33.897 20240518 21:24:44.290
Lab6.ApiAutomation.OJ.1000_CE		PostOJ	FAIL	'答案错误 得分0' does not start with '编译错误'	00:00:13.909	20240518 21:26:55.600 20240518 21:27:09.509
Lab6.ApiAutomation.OJ.1001_AC		PostOJ	FAIL	'运行错误 得分0' does not start with '正确'	00:00:10.026	20240518 21:24:44.291 20240518 21:24:54.317
Lab6.ApiAutomation.OJ.1003_AC		PostOJ	FAIL	'运行错误 AC:0%' does not start with '正确'	00:00:10.857	20240518 21:25:04.659 20240518 21:25:15.516
Lab6.ApiAutomation.OJ.1006_AC		PostOJ	FAIL	'运行错误 得分0' does not start with '正确'	00:00:12.051	20240518 21:25:37.487 20240518 21:25:49.538
Lab6.ApiAutomation.OJ.1006_PE		PostOJ	FAIL	'运行错误 得分0' does not start with '格式错误'	00:00:15.407	20240518 21:27:24.221 20240518 21:27:39.628
Lab6.ApiAutomation.OJ.1007_AC		PostOJ	FAIL	'编译错误' does not start with '正确'	00:00:12.359	20240518 21:25:49.538 20240518 21:26:01.897
Lab6.ApiAutomation.OJ.1008_AC		PostOJ	FAIL	'编译错误' does not start with '正确'	00:00:11.731	20240518 21:26:01.898 20240518 21:26:13.629
Lab6.ApiAutomation.OJ.1000_MLE		PostOJ	PASS		00:00:12.960	20240518 21:26:13.630 20240518 21:26:26.590
Lab6.ApiAutomation.OJ.1002_AC		PostOJ	PASS		00:00:10.342	20240518 21:24:54.317 20240518 21:25:04.659
Lab6.ApiAutomation.OJ.1003_TLE		PostOJ	PASS		00:00:13.969	20240518 21:26:26.591 20240518 21:26:40.560
Lab6.ApiAutomation.OJ.1004_AC		PostOJ	PASS		00:00:10.841	20240518 21:25:15.516 20240518 21:25:26.357
Lab6.ApiAutomation.OJ.1004_RE		PostOJ	PASS		00:00:14.710	20240518 21:27:09.510 20240518 21:27:24.220
Lab6.ApiAutomation.OJ.1005_AC		PostOJ	PASS		00:00:11.129	20240518 21:25:26.357 20240518 21:25:37.486
Lab6.ApiAutomation.OJ.1005_WA		PostOJ	PASS		00:00:15.037	20240518 21:26:40.563 20240518 21:26:55.600

可以看到好多应该AC的代码交上去不对，更离谱的是还有编译错误，但是点到编译信息里又什么都没有。

可以看出来，这次接口测试，证实了评测机的缺陷。

6.2. 注册功能测试

刚刚测了一个需要cookie鉴权的，现在再来测一个不需要cookie的好了。注册功能。

1	\$(payload)=	Create Dictionary	user_id=\${uid}	nick=\${name}	password=\${pwd}	rptpassword=\${rpt}	school=\${school}	email=\${email}
2	\$(response)=	POST	\$(ip)/register.php	data=\${payload}				
3	Log		\$(response.content)					
4	Status Should Be	200	# 应该得到正确响应					
5	\$(bytes_seq)	Evaluate	'\${should_contain}'.enc					
6	Should Contain		\$(response.content)	\$(bytes_seq)				
7								

首先把负载填好，然后直接调Post请求即可（这里不需要创建session，因为注册本来就是无状态无上下文的功能）

最后，如果注册成功，应该会有个

```
20240519 12:24:06.160 : INFO :  
POST Response : url=http://211.87.224.235:8800//register.php  
status=200, reason=OK  
headers={'Server': 'nginx/1.18.0 (Ubuntu)', 'Date': 'Sun, 19 May 2024 04:  
body=<script>history.go(-2);</script>
```

如果注册失败，会有个：

```
20240519 12:24:06.175 : INFO :  
POST Response : url=http://211.87.224.235:8800//register.php  
status=200, reason=OK  
headers={'Server': 'nginx/1.18.0 (Ubuntu)', 'Date': 'Sun, 19 May 2024 0  
body=<script language='javascript'>  
alert('User ID can only contain NUMBERS & LETTERs!\n');  
history.go(-1);
```

我们可以靠这两个断言注册结果是否是应该出现的结果。先传入一个应该出现的结果should_contain，注册成功是-2，注册失败是alert。随后转为字节串，和response的content属性比较，看看后者是否包含前者即可。

结果展示

AllTagsSuitesSearch

Status:6 tests total, 6 passed, 0 failed, 0 skipped

Total Time:00:00:00.098

Name	Documentation	Tags	Status	Message	Elapsed	Start / End
Lab6.ApiAutomation.Register.pwd_rpt_err		PostRegister	PASS		00:00:00.014	20240519 12:24:06.215 20240519 12:24:06.229
Lab6.ApiAutomation.Register.pwd_too_short		PostRegister	PASS		00:00:00.017	20240519 12:24:06.198 20240519 12:24:06.215
Lab6.ApiAutomation.Register.school_too_long		PostRegister	PASS		00:00:00.015	20240519 12:24:06.229 20240519 12:24:06.244
Lab6.ApiAutomation.Register.success		PostRegister	PASS		00:00:00.020	20240519 12:24:06.144 20240519 12:24:06.164
Lab6.ApiAutomation.Register.uname_format_err		PostRegister	PASS		00:00:00.016	20240519 12:24:06.164 20240519 12:24:06.180
Lab6.ApiAutomation.Register.uname_too_short		PostRegister	PASS		00:00:00.016	20240519 12:24:06.181 20240519 12:24:06.197

注册功能还是都没有问题的。

7. 总结

作为一个程序员，我认为robot framework是一款不太好用的软件。虽然是一个自动化测试框架，但实际上开发效率远低于用户自己通过第三方库API编写测试脚本的开发效率。有如下原因：

1. IDE：
 - a. **GUI的稳定发行版和最新发行版不一致**：如果你要用RIDE这个GUI，首先Robot version 7.0的最新发行版内核甚至不能兼容这个GUI，是需要降版本的。最关键的是这在官网的文档中根本没提到，我在stkof上的一个犄角旮旯里找到的解决方案。这说明这款软件的运营维护以及相关生态是不够成熟的。
 - b. **自动补全（代码提示）操作不便**：如果你想查看相似的变量、关键字，必须使用快捷键查看（无论你把快捷键绑在哪了，总归要按快捷键，默认的ctrl+shift+space的快捷键实在太麻烦了），而市面上有点用户量的高级程序语言IDE的自动补全和代码提示都是全自动的，做得好的还能集成Copilot这样的外挂级别的辅助工具。RIDE根本没得比。如果你用Text Edit，那么根本就没有代码补全功能。
 - c. **编辑时卡顿**：打开文件、编辑保存、切换选项卡时会卡顿。
 - d. **选项卡不能多开**：Editor、Text、Run选项不能多开，还没有快捷键切换。只能一个个来回用鼠标点击切换。
 - e. **读磁盘慢**：每次向磁盘写入日志或者报告后，如果这个写入位置在打开文件的文件树下，都会询问用户是否读磁盘，非常繁琐。而且读磁盘很慢。最关键的是，读完磁盘会把之前的编辑界面和文件树的GUI全关掉，用户还原工作场景时会遇到很大的麻烦。

f. **代码层级结构差**：Editor居然是用一个表格来编写代码，在表格中填写关键字和变量。

g. **容易卡死、闪退**

2. API：

a. **和高级程序语言脱轨**：这个软件暴露了一套API（关键字），实际上这套API不依托于任何高级程序语言（虽然它是用CPython实现的，但是它的语法以及保留关键字和Python完全不像，反而跟Matlab或者R语言甚至是PHP有点像），导致程序员用户会有很大的学习成本。之前用selenium+webdriver时，可以说我压根没有任何学习成本。但是对于这个软件，我要学习一整套API和基础的数据结构的使用方法。

b. **纯面向过程**：这套API除了支持封装（关键字）之外，不支持任何类、继承、多态，仅支持纯面向过程的写法。

3. 数据管理：

a. **数据源格式过于单调**：无论是什么文件格式，数据源的格式都非常单调，必须依赖于它规定的编写方式进行解析，也就是类似于***** Keywords ***** 和 ***** Variables ***** 这样的格式。如果能用高级程序语言编写，我可以**读文本文件、通过第三方库加载配置文件、直接开静态容器把常量写死**等方式读取数据源，而不是僵硬地按照某种固定格式编写数据然后再用鼠标点来点去把这些数据文件引入项目。

b. **变量类型混乱不堪**：对于变量的引用有\$\\&\\@这些标识符，代表变量、字典、列表等。用\$时，可以随意引用字符串、整型等基本类型，但是列表和字典这样的数据结构就完全不能用\$符号。这种**介于弱类型和强类型之间的类型**让人叫苦不迭。

综上所述，这个软件，功能测试不及selenium+webdriver，压力测试不及Jmeter。我还翻了这个软件的开源库。

<https://github.com/robotframework/robotframework>

他在有着171个contributors的同时仅有不到1w的个人user。而且实际上并没有多少企业在用这个测试工具，大部分还是Jenkins集成JUnit或者TestNG这样的自动化测

试框架。这个实验我觉得是一次学习和尝试，了解一些新的工具。但是如果谈及到成熟的生产力工具，可能还是会另有选择。