

# API TEST

# 写在文档前

声明：

本文档只是个人经验整理，个人理解的内容，或许存在跟实际理论知识有出入的地方，请包涵，谢谢

# 初识Robot Framework

Robot Framework是一个通用的关键字驱动自动化测试框架。测试用例以HTML，纯文本或TSV（制表符分隔的一系列值）文件存储。通过测试库中实现的关键字驱动被测软件。Robot Framework灵活且易于扩展。它非常适合测试有不同接口的复杂软件：用户接口、命令行，Web服务，专有的编程接口等。它具有易于使用的表格来组织测试过程和测试数据。

1	create session	session	http://www.baidu.com	
2	\${resp}	get request	session	/
3	log	\${resp.content}		
4				
5				
6				

（图一）

# 什么是接口测试

接口测试是测试系统组件间接口的一种测试。接口测试主要用于检测外部系统与系统之间以及内部各个子系统之间的交互点。测试的重点是要检查数据的交换，传递和控制管理过程，以及系统间的相互逻辑依赖关系等。

接口有很多种类型，目前用于接口自动化的，就是Web api、Database api等，本次文档重点总结Web api测试，后面文档中提到的接口测试，没有特别指明的一律为Web API测试，以具体实例讲解。

注：Web api测试之前，最基本的就是需要了解什么是web，web api大概是什么，什么是request，什么是response，get请求，post请求，content等，这些基本知识是要走在web api测试之前的。

# 接口测试RF环境配置

多余的就不说了，使用ride，可以直接引入Library，至于怎么安装Library，自行Google.

requestsLibrary、requests安装

requests的官方地址：<https://pypi.python.org/pypi/requests>

requestsLibrary的官方地址：

<https://pypi.python.org/pypi/robotframework-requests/>

requestsLibrary就是用于Robot Framework的测试库，底层基于requests这个工具。安装时，下载tar.gz包安装，先要安装requests，再安装requestsLibrary。

有以上两个包就可以进行api测试了。

注：此文档中有部分内容摘自于道长的《Robot Framework 自动化测试修炼宝典》

# 简单的API测试

- 基本的发送get请求的方式，网上都能查的到，我这边就以现有的例子讲解。
- 如图一：session是一个别名，可以自行命名，是为了在发送get请求时，访问这个session会话，一般create session时，传入的是一个URL的地址，如<http://www.baidu.com> 然而在发送get请求时，如无参数传入，则直接传入"/",此处不能不传，因为原始的get request方法是必须有两个传输传递。
- 发送的get请求返回的是整个页面的response，实例中打印的是resp.content,实际上就跟通过浏览器访问的百度页面，按F12查看的html源码一致。
- 注：在尝试这个例子的时候，要先知道发送get请求的格式以及create session、get request传入参数的形式和返回什么样的数据。

# 一个简单的API文档

- 一个合格的API文档，最基本的就是包涵接口的server，和uri，当然这个说法很多，举个例子说明一下，就如同下面一个地址
- Web\_server:http://172.0.0.1:8080/traffic-analytice-restful
- Web\_uri:/performance/report/actions.html
- 实际上webserver和weburi两部分组成了一个访问这个接口的地址。
- 还要包涵接口访问的形式，是post，get，put还是其他形式。
- 还要包涵接口传递的参数，以及参数的格式和返回值参数的格式，好一点的接口文档会提供给脚本开发人员哪些值可以判断请求是否成功等等。

# 例一 get请求：

- web\_server:http://127.0.0.1:8080/traffic-analytice-restful
- Web\_uri:/ga/api/view/{domain}.html
- 接口业务逻辑：通过domain获取Google Analytics的Views
- Http Method: Get
- 参数: domain: string
- 参数示例: domain=www.igvault.fr
- 检查点: 返回值Content 中code=000002



# 例一脚本演示：

1	create session	session	http://127.0.0.1:8080/traffic-analytics-restful		
2	\${resp}	get request	session	/ga/api/view/www.igvault.fr.html	
3	\${resp_json}	to json	\${resp.content}		
4	should be equal as strings	\${resp_json[code]}	000002		
5					
6					

elapsed time: 0:00:01 pass: 1 fail: 0

command: pybot.bat --argumentfile c:\users\admini~1\appdata\local\temp\RIDE0usy2c.d\argfile.txt --listener C:\Python27\lib\

testProject

testProject.SEMTools

test7

| PASS |

testProject.SEMTools

| PASS |

1 critical test, 1 passed, 0 failed

1 test total, 1 passed, 0 failed

testProject

| PASS |

1 critical test, 1 passed, 0 failed

1 test total, 1 passed, 0 failed

Output: c:\users\admini~1\appdata\local\temp\RIDE0usy2c.d\output.xml

Log: c:\users\admini~1\appdata\local\temp\RIDE0usy2c.d\log.html

Report: c:\users\admini~1\appdata\local\temp\RIDE0usy2c.d\report.html

unexpected error: Exception in thread Thread-1 (most likely raised during interpreter shutdown):

test finished 20160624 17:59:57

20160624 17:59:57.718 : INFO : Get Request using : alias=session, uri=/ga/api/view/www.igvault.fr.html, headers=None

20160624 17:59:57.718 : INFO : \${resp} = <Response [200]>

20160624 17:59:57.721 : INFO : To JSON using : content={"code":"000002","msg":null,"data":[{"bingAccountIds":null,"domain"

20160624 17:59:57.722 : INFO : To JSON using : pretty\_print=False

20160624 17:59:57.722 : INFO : \${resp\_json} = {u'msg': None, u'status': None, u'code': u'000002', u'data': [{u'domain': u'

Ending test: testProject.SEMTools.test7

## 例二post请求：

- web\_server:http://127.0.0.1:8080/traffic-analytice-restful
- Web\_uri:/optimizeTask/schedule.html
- 接口业务逻辑：执行优化任务
- Http Method: Post
- 参数：baseDate: string(执行Rule的数据的开始时间)
- ruleId: string (Rule的ID)
- 参数示例：  
baseDate=20160501&ruleId=57186fd8e4boofeco2b1c887
- 检查点：返回值Content 中code=000002

# 例二脚本演示：

1	Create Session	session	http://52.53.246.23:8080/traffic-a			
2	\${postdata}	Create Dictionary	baseDate=20160401	ruleId=57186fd8e4b00fec02b1c88		
3	\${postheader}	Create Dictionary	Content-Type=application/x-www-			
4	\${response}	post request	session	/optimizeTask/schedule.html	None	\${postdata} \${postheader}
5	\${content_json}	to json	\${response.content}			
6	should be equal as strings	\${content_json['code']}	000002			

```

elapsed time: 0:00:01  pass: 0  fail: 1
command: pybot.bat --argumentfile c:\users\admini~1\appdata\local\temp\RIDE0usy2c.d\argfile.txt --listener C:\Python27\lib\site-packages\robotide\contrib
=====
testProject
=====
testProject.SEMTools
=====
test8
000001 != 000002 | FAIL |
=====
testProject.SEMTools
1 critical test, 0 passed, 1 failed | FAIL |
1 test total, 0 passed, 1 failed
=====
testProject
1 critical test, 0 passed, 1 failed | FAIL |
1 test total, 0 passed, 1 failed
=====

```

```

Starting test: testProject.SEMTools.test8
20160627 11:58:25.922 : INFO : Creating Session using : alias=session, url=http://52.53.246.23:8080/traffic-analytics-restful/, headers={},
20160627 11:58:25.924 : INFO : ${postdata} = {u'baseDate': u'20160401', u'ruleId': u'57186fd8e4b00fec02b1c887'}
20160627 11:58:25.925 : INFO : ${postheader} = {u'Content-Type': u'application/x-www-form-urlencoded'}
20160627 11:58:25.928 : INFO : Starting new HTTP connection (1): 52.53.246.23
20160627 11:58:26.240 : INFO : Post Request using : alias=session, uri=/optimizeTask/schedule.html, data=None, headers={u'Content-Type':
20160627 11:58:26.241 : INFO : ${response} = <Response [200]>
20160627 11:58:26.243 : INFO : To JSON using : content={"code": "000001"}
20160627 11:58:26.243 : INFO : To JSON using : pretty_print=False
20160627 11:58:26.244 : INFO : ${content_json} = {u'code': u'000001'}
20160627 11:58:26.247 : FAIL : 000001 != 000002
Ending test: testProject.SEMTools.test8

```

(图三)

- 由例二可以看出如何进行一个post请求的访问
- 例二中的接口原本就是一个异常接口，故此返回的是0000001，而不是0000002的code，可以看到response的返回值是200，证明此接口访问是正常的，因此由最后的结果可以判断，该接口异常。
- 接口传入的header和data都是字典形式的数据，具体传递的内容，请自行google查看，做api测试要有一定的web相关的知识基础，否则很多地方都很难理解。
- 其实看上去post请求跟get请求的写的脚本的内容大致差不多，知识传递参数的形式不一样，返回的参数也都是是一样的，因此，脚本开发人员可以对部分重复使用的地方，通过ride的关键字进行封装。

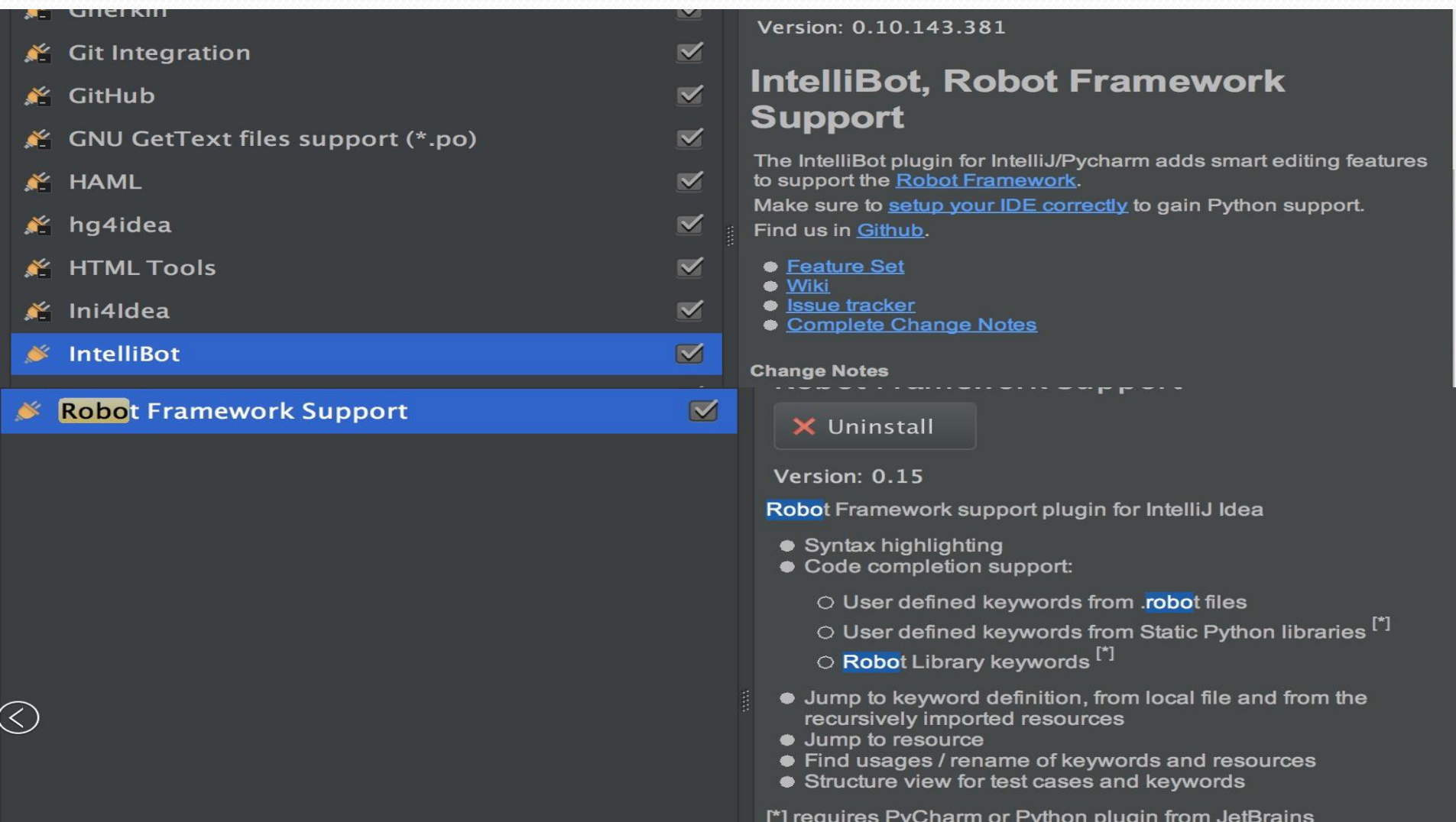
# 两种请求方式小结

- HTTP的请求方式主要有GET,POST,PUT,DELETE，我这里就先只介绍以上两种请求方式，因为put请求和delete请求只会在特定的需求下才会运用，大部分都是get和post请求。
- 不管是那种请求方式，实际上，请求只有一种，那就是发送头文件，只是实现的形式不一样而已。
- 在实际api测试的过程中，要看具体要求，例子中只是说明简单的api使用，但是实际过程中，并不是一条正确的路线走下去，需要制造错误的数据进行测试，看返回的是否为错误的code。

# 利用Pycharm编写RF用例

- 对于初学者来说，使用ride无疑是最好的，因为ride界面看上去很简洁，固定的格式，如同一个excel表格一样，该写啥写啥，需要传入参数的方法后面如果不传入参数会标红，变量的颜色，方法名的颜色等等，我刚接触rf也是从ride开始的，后来我逐渐发现，ride并不能满足我的需求，限制太大，因此我就寻求其他的路，先是使用eclipse安装robot插件，然而不知道哪里出了问题，并没有网络上，其他大神那样，有代码提示功能，关键字颜色变化功能，就如同一个文本，后来也尝试过sublime，但都无果。
- 偶然一次机会，使用pycharm编写库文件，新建了一个.robot文件进行测试时，pycharm提示安装robot相关的插件，结果，安装上之后，只要是.robot后缀的文件，就会带有代码自动提示，格式化，快捷索引等功能，无疑更方便了编写。

# 具体插件

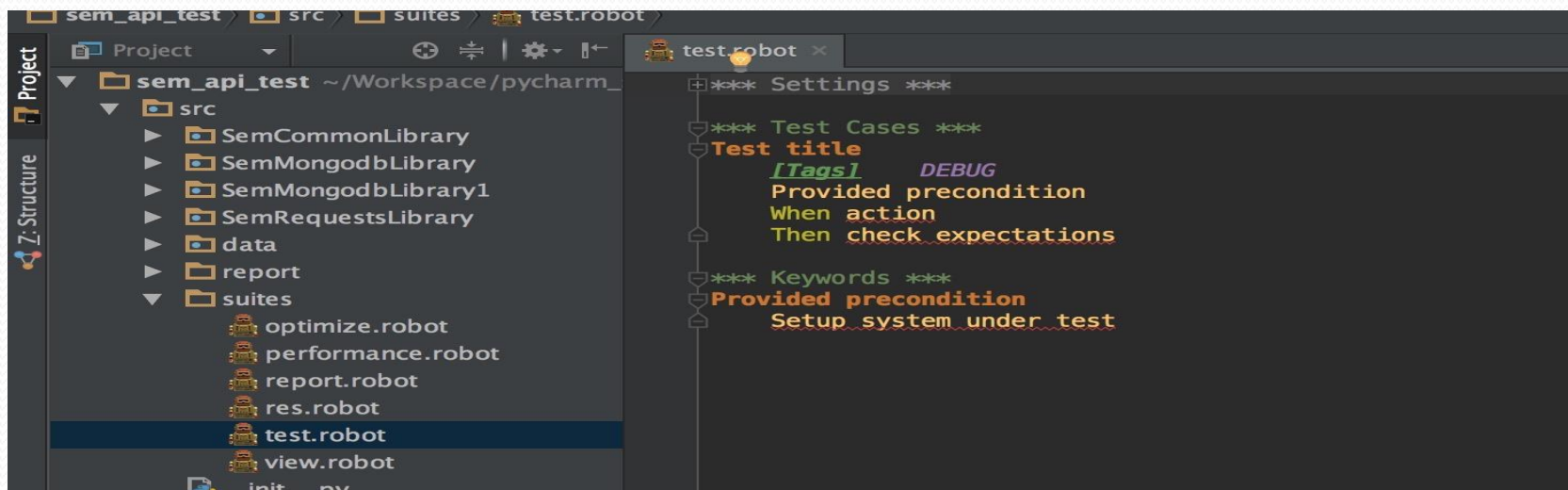


(图四)



# Pycharm插件具体安装方法

- 安装python，pycharm等等就不多说了，这是配置python环境。
- 在pycharm中新建一个python工程，在工程中创建一个.robot为后缀的文件，文件创建完成后，在整个IDE的右上角会弹出匹配到相关文件的插件，询问是否安装，点击安装后，部分电脑可能需要重启pycharm，随后就会看到右键，new的时候，文件列表中有一个robot file，创建好这个robot file后，文件格式就已经自动格式化了，如图五、图六所示：



(图五)



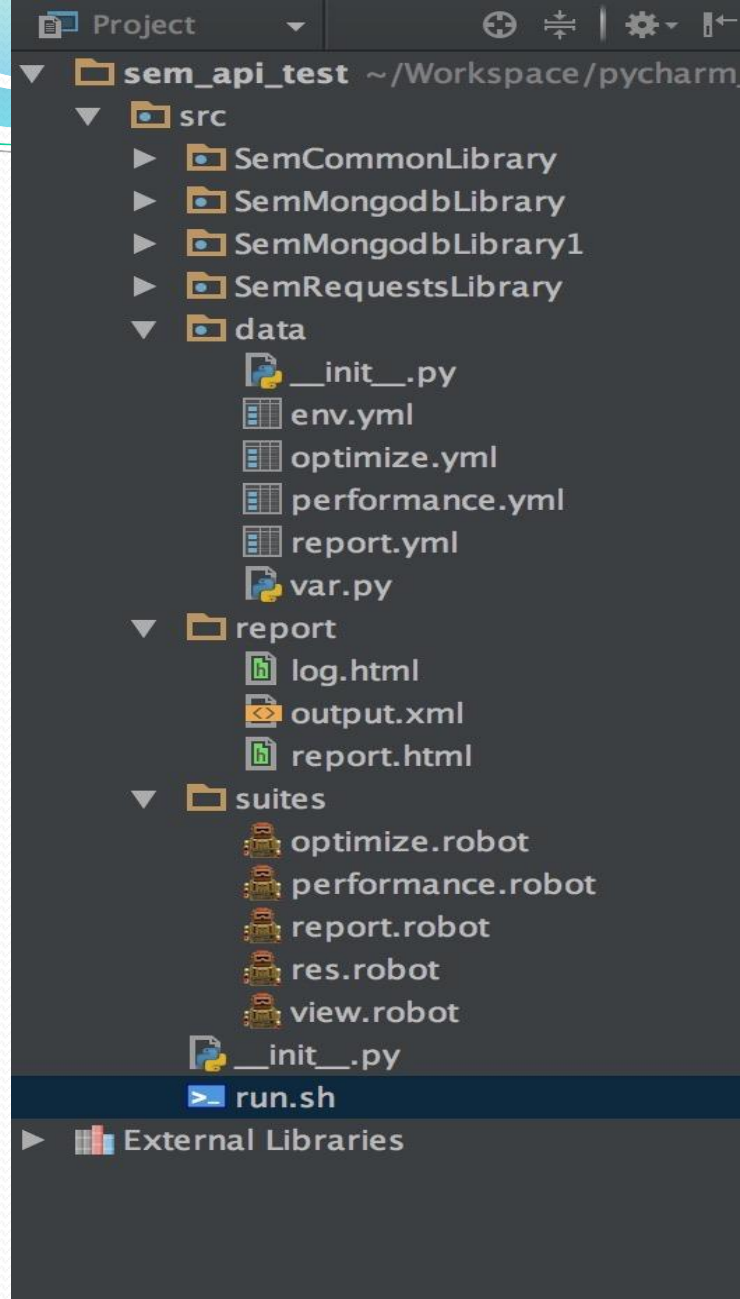


# Pycharm编写RF脚本规范

- 作为一个程序员，尽管是写自动化测试脚本，也制定了一些编写规范：
  1. 进行某个项目接口测试脚本开发时，在pycharm(后面直接称为IDE)中新建该工程，如Sem\_api\_test
  2. 开发中有MVC架构，同样，RF也应该有它自身的架构，因此在测试脚本开发时，采用数据和代码分离，数据单独放在文本文件中或者如同我的项目中放在yaml文件中，使用python脚本进行读取这些数据。
  3. 建议库文件使用自己自定义，可能需要用到从自身的库文件中的方法或者pip安装的Library中的方法，但是个人还是建议自定义，这样有自己的库文件，而且做接口测试，不仅接口变化小，使用的库中的方法几乎也不会有什么变化，因此有一个自己的库文件，对于任何项目几乎都是通用的。
  4. 整理出一个合适的目录架构，报告的存放目录等，也是令人赏心悦目的。

# 目录架构

```
--Project Name
|--src
|   |--FirstLibrary
|   |--SecendLibrary
|   |--data
|   |--report
|   |--suites
|   |--other python file
|   |--other .sh or .bat file
```



(图七)

# 自定义python库

- RF中库文件的定义要符合以下的标准。
  - 新建一个python package 命名为MyLibrary
  - 在MyLibrary文件夹中新建一个version.py文件，用于描述当前测试库的版本信息。
    - VERSION='1.0'
  - 在MyLibrary文件夹中新建一个keywordsl类，可以起名为：testTemp.py，代码如下：
    - class TestTemp(object):
      - def \_\_init\_\_(self):
        - pass
      - def mytest(self,name):
        - return "hello"+name
  - 在MyLibrary中的\_\_init\_\_.py文件中把关键字暴露出来，格式如下：
    - from testTemp import TestTemp
    - from version import VERSION
    - \_version=VERSION
  - class MyLibrary(TestTemp):
    - ROBOT\_LIBRARY\_SCOPE = 'GLOBAL'

# 自定义python库如何生效

- 上一页PPT中，已经创建好了一个简单的库文件，讲这个库文件MyLibrary整个拷贝到python的安装目录/Lib/site-packages/中，如果使用ride的同学，重启一下ride，然后导入MyLibrary，如果使用pycharm的同学，就可以直接Library MyLibrary即可。
- 在Ride或者Pycharm中就已经可以查看到我们定义的mytest方法，当使用这个方法的时候，ride中会提示欠缺一个参数，需要你传入，这个是ride的一个好处，会有红色提示，但是pycharm并没有，然而这并不能影响这个IDE对RF脚本开发提供的便利。
- Ride中选中某个关键字，按住ctrl可以查看到需要传入的参数等信息，Pycharm中则可以按住ctrl左键点击就可以跳转到所定义的.py中的相关类，直接查看源码，并且你还可以在源码中添加方法，这样对项目的功能实现要方便很多。

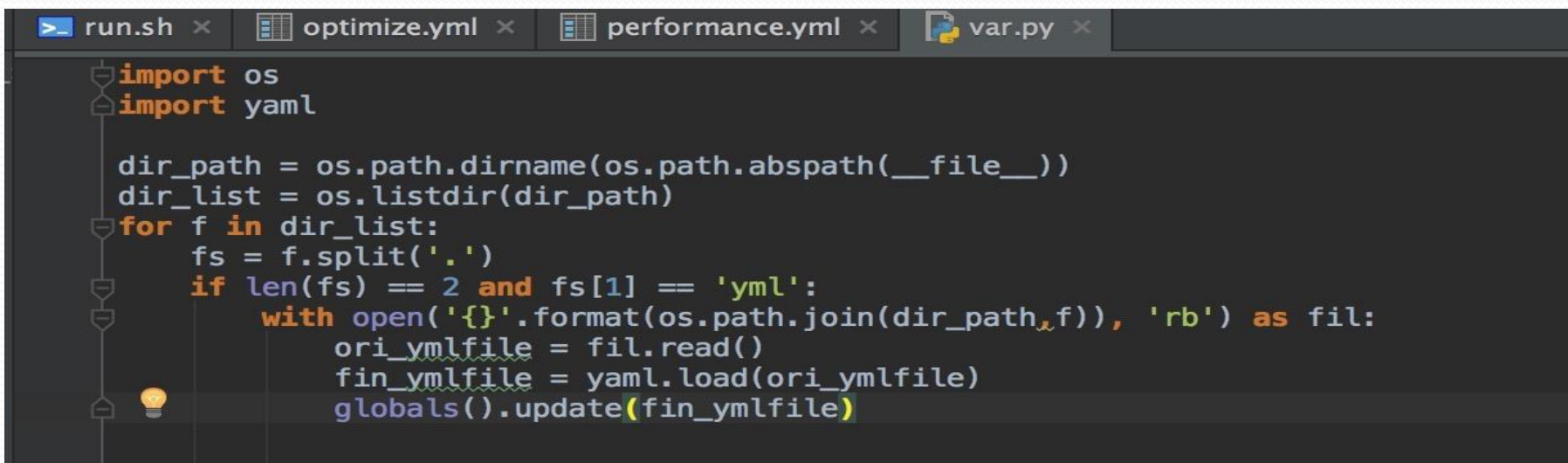


# 为什么要自定义库

- 我们为什么要自定义库，为什么不适用现有的库文件？原因，无它，方便。
- 试想一下，在一个干净的机器上，只有一个robot，我们的测试，并不需要依靠ride或者pycharm，这些只是我们开发脚本的工具，实际只是执行的pybot这个命令。因此我们只需要有robot环境即可。
- 但是仅仅只是robot环境，没有Library，功能会欠缺怎么办？这样我们自定义的Library就派上用场了，所有的方法，包括可能用到的BuiltIn中的方法我们也封装到我们自己的库文件中，这样，我们在测试的时候不需要导入任何其他Library，只需要我们自己的Library就足够了。
- 这样做的好处就是避免了你切换环境的时候，还需要去pip install **Library**，我们要做的只是运行我们事先写好在工程中的脚本，将我们自定义的整个Library拷贝到python安装目录的site-packages目录下。

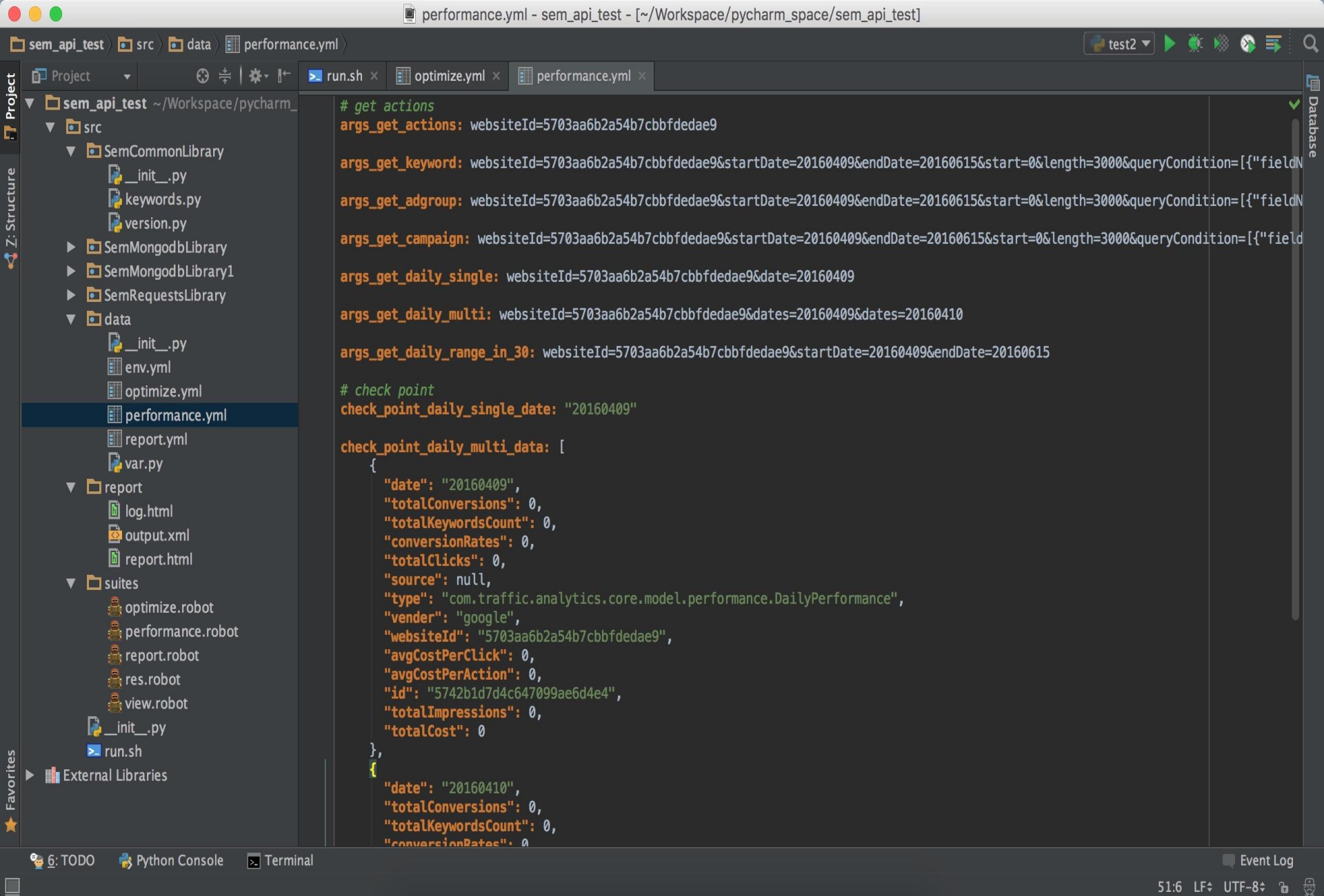
# data目录下文件的作用

- 如以上ppt的图七所示，\_\_init\_\_.py文件的存在，表明data目录是一个python package，\_\_init\_\_.py文件中没有任何内容；data目录下大部分文件后缀为yaml，这里我使用的是yaml格式的文件，只是为了存储数据，可以使用.txt等。yaml文件中的数据格式你可以自行定义，写一个json的也行，写一个key:value的也行，怎么方便怎么来，如图九。
- 在data文件夹内还有一个特殊的文件，是一个.py的文件，其作用就是data目录下.yaml为后缀的文件读取的全局变量中，这样的好处就是在所有的suite中就可以使用\${args\_get\_actions}获取到相应的值。就算后期有改动也是改动yaml文件中的这个值，而不需要改动脚本代码。



```
run.sh x optimize.yaml x performance.yaml x var.py x
import os
import yaml

dir_path = os.path.dirname(os.path.abspath(__file__))
dir_list = os.listdir(dir_path)
for f in dir_list:
    fs = f.split('.')
    if len(fs) == 2 and fs[1] == 'yaml':
        with open('{}'.format(os.path.join(dir_path, f)), 'rb') as fil:
            ori_yamlfile = fil.read()
            fin_yamlfile = yaml.load(ori_yamlfile)
            globals().update(fin_yamlfile)
```



(图九)



# Report 目录

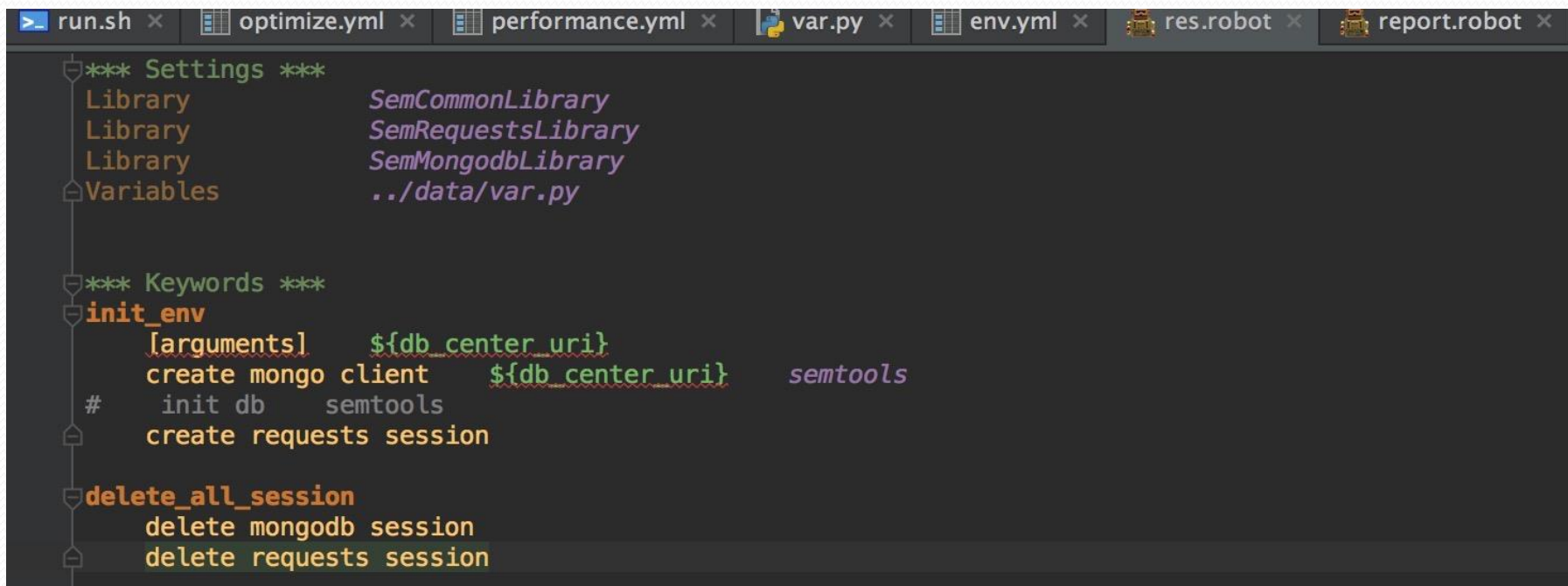
- 这个目录没有什么好说的，就是一个存放报告的目录，报告不一定要存放在这个目录，我这边是为了方便，所以这里创建了一个report目录，后续在执行pybot脚本时，使用-d参数指定到这个目录，这样output.xml、log.html、report.html这三个文件就会存放在这个目录下，当然有特别需求的可以存放在其他地方。

## other python file

- 这个python文件的作用是将上面我们创建的FirstLibrary, SecondLibrary拷贝到python安装目录/Lib/site-packages/目录下，具体的实现代码可以参考google下，我这边暂时没有现成的代码展示。
- 注:编写Python脚本和python库都需要一定的python功底，因此在做自定义python库时，请先恶补python编程。

# suites目录

- suites目录是脚本存放的目录，从图七中可以看出，该文件下存放的都是.robot文件，.robot文件前面也提到了，实际上跟ride创建的txt一样，可以理解成都是文本。
- 这样在每个.robot文件中编写脚本即可。需要注意的是，如果Library太多的话，创建一个如图七中res.robot的存在，也就是ride中的关键字，在res文件中，我一次性导入了多个Library，并且创建关键字init\_env用来初始化相关的操作，这样在其他的.robot脚本的文件中，使用source res.robot就等于导入了多个Library，并且可以在res.robot中定义的关键字。



```
*** Settings ***
Library          SemCommonLibrary
Library          SemRequestsLibrary
Library          SemMongodbLibrary
Variables        ../data/var.py

*** Keywords ***
init_env
    [arguments]    ${db_center_uri}
    create mongo client    ${db_center_uri}    semtools
    #    init db    semtools
    create requests session

delete_all_session
    delete mongodb session
    delete requests session
```

- 暂时不需要理解图十中那些方法的作用，因为这个是我现在所做项目的业务需要，当然如果你不同于我这样做也是可以的，在每个.robot文件中使用Library导入库没有任何问题。
- 图十中所示的是一个user keyword文件，文件中不包含 **\*\*\*Test Case\*\*\***，因此这个文件再执行时，并不会有任何结果输出，也不会计入到执行用例的条数中去。
- 正常的用例文件是包含**\*\*\*Test Case\*\*\***的，并且在脚本文件中可以正常调用user keyword中的关键字，比如图十中的init\_env关键字
- 我这里的设计思想就是，需要初始化的数据在这个user keyword中写好，在robot脚本中，是用Test Setup中使用这个init\_env方法，这样在执行这条用例的时候，环境就已经是初始化完成，就不需要在每个脚本中去编写调用了。
- 注：Test Setup相关知识请[google Setup, Teardown](#)

# 详细说明-Library

- 这里我使用Get请求作为说明
- 写Library之前，先要懂python这是前提，当然也可以用Jython，但是我没有用过，我这里暂时就介绍python编写Library。
- 在之前，我们必须在脚本中这样写：`create session session url`
- 然而，我们可以在自定义的库中，将创建的会话注册到当前的缓存中去，具体代码如图十一。
- 以上的代码编写好之后，当Library拷贝到site-packages目录下后，可以正常运行时，我们在res.robot这个文件中有两个方法，一个是init\_env,一个是delete\_all\_session,在init\_env中，我们调用定义的方法create request session，可以看到这里不需要传入任何参数，因为我们定义的方法中并没有任何形参；在delete\_all\_session中，我们删除这个会话，调用的是我们定义好的delete\_requests\_session。到这里，我们只是在user keywords中定义使用了这两个方法，如何在我们脚本中使用呢？这就涉及到上一页中的setup和teardown了，Test setup中，我们使用init\_env，这样该需要的环境都已经创建好，在Teardown中调用delete，这样测试完毕后，就删除会话，保持测试环境的纯净。

```
res.robot x site-packages/.../keywords.py x SemRequestsLibrary/keywords.py x view.r
#!/usr/bin/env python
# encoding:utf-8
import ...

_req_alias = "SemRequests"

class SemRequestsKeywords(object):
    def __init__(self):
        self._cache = robot.utils.ConnectionCache('No sessions created')
        self.builtin = BuiltIn()

    def create_requests_session(self, req_alias=_req_alias):
        """
        创建会话
        """
        req = requests.Session()
        self._cache.register(req, alias=req_alias)

    def delete_requests_session(self):
        """
        清空会话
        """
        self._cache.empty_cache()

    def get(self, url, params={}, req_alias=_req_alias):
        req = self._cache.switch(req_alias)
        return req.get(url, params=params)

    def post_request(self, url, params, headers, req_alias=_req_alias):
        req = self._cache.switch(req_alias)
        return req.post(url, params=params, headers=headers)
```

(图十一)



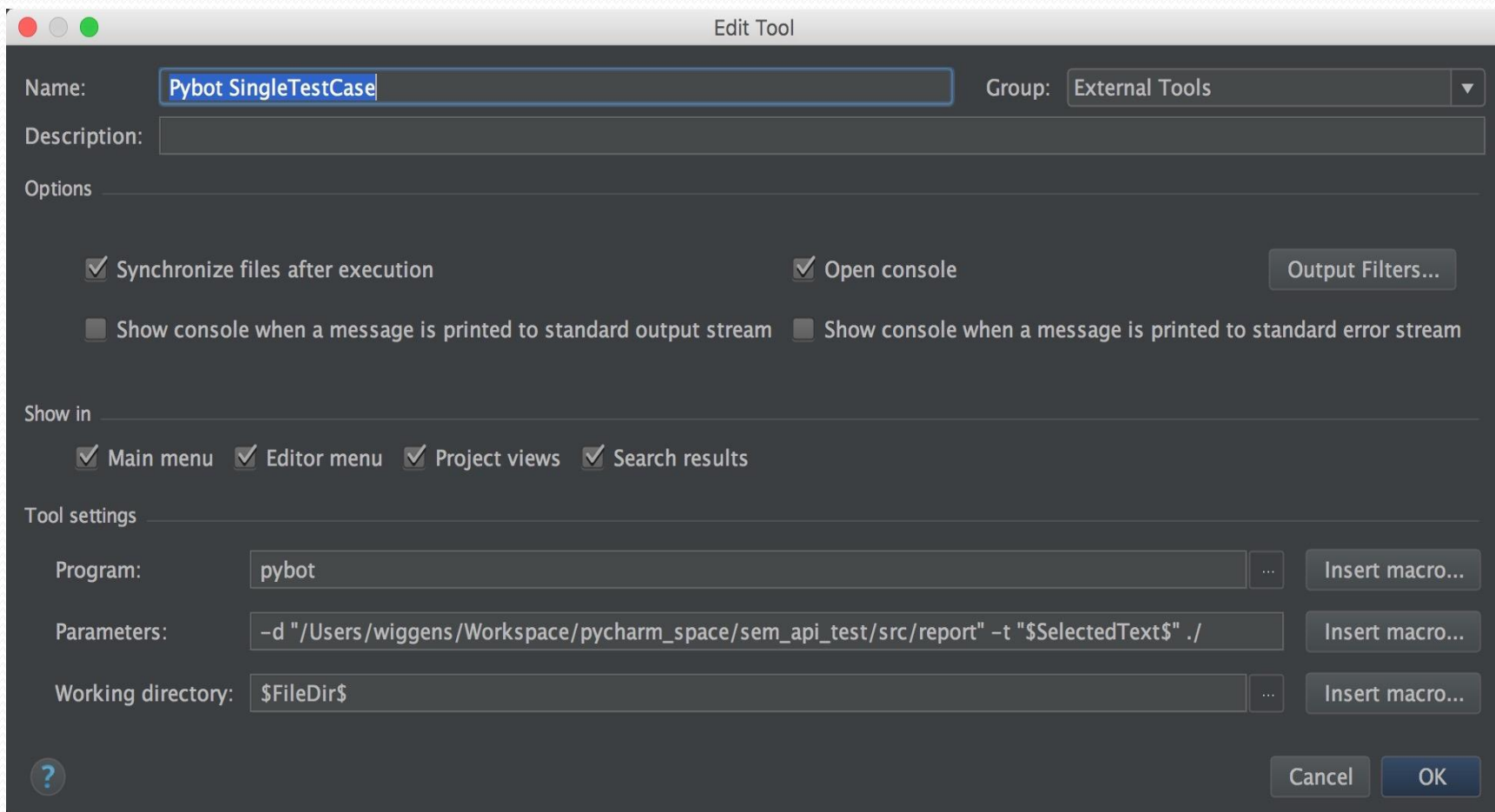
# 详细说明-TestCase

- 新建.robot文件，在ride中，是默认的表格形式，在pycharm中，可以用四个空格代替。
- 具体TestCase的编写需要看具体需求，接口测试如果接口正常，则返回response中status\_code必定是200,其他的比较方式需要参考接口文档。
- 前文有说到yaml文件，并且附有一个解析yaml文件的python文件，并且python文件中最后将解析出来的所有数据都update到了全局，因此所有yaml文件中的key: value形式的参数，都是全局参数，所以在写TestCase是可以直接使用\${key}即可获取到参数，让然，插件会有下红波浪线提示，忽略它，因为yaml文件这里只是一个当做txt的文本容器，并没有实质性的编译，因此Pycharm IDE保存后会自动编译，编译的时候会提示找不到这个参数，但是运行的时候，由于python会解析yaml，所以不会有异常。
- 注：这里要注意的是，yaml中文件格式要求不严格，但是必定是准守key: value的格式，:后面有空格，而且是英文状态下的冒号，如果这些编写不正确，会导致整个项目运行异常。

# Pycharm中运行TestCase

- 此配置的方法我就不具体说明了，参考下面的连接：
- <http://www.daveze.cc/2016/05/05/%E5%9C%A8pycharm%E4%B8%AD%E6%94%AF%E6%8C%8A%E6%9C%AC/>

www.daveze.cc/2016/05/05/在pycharm中支持robotframework脚本/



# 其他说明

- 本文档只是一篇自我总结的文档，具体内容都是个人理解，若有误，请纠正，并且文档并不面面俱到的详细，只是记录了实际过程中的一些经验。因此不喜勿喷。
- 看完本文档后，可能并不能让你直接就会接口测试了，其实本人也对接口测试才刚接触，本文档中没有提及到发送请求的参数处理以及返回响应的参数处理，这个同样也需要有一点web开发的功底，因此做API测试并不像使用selenium做UI功能测试，API测试需要涉及到很多底层的东西。
- 接口测试要求的环境是纯净的，起码我目前接手的项目是这样要求的，因此就需要CI环境的支持，jenkins配合代码仓库、tomcat等自动下拉代码，自动编译，自动打包，自动发布，最后才是自动化接口测试。
- 接口测试远远不止我所写出来的这些内容，很多的内容我也没有接触到，因此没有实际的操作经验，我也没法写出来。
- 在接口测试中，有一部分没有讲解到，那就是数据库的操作，我所做的项目是mongodb数据库的，各个公司不一样，因此暂时没有列出来，等后期空闲时再自我整理。在测试过程中，api返回的数据需要跟当前数据库中的数据进行比对，这样判断是否是返回正确的数据，还有一部分就是需要往数据库中插入一些假数据，这些假数据是走在测试之前的，因此数据库部分也是很重要的，同样的坐数据库的操作，包括数据库Library的开发，也是需要一定的数据库开发的经验。



谢谢