# Pytest

## Pytest 与 unittest 区别

### 编写用例上：

Unittest： 测试用例类必须继承 unittest.TestCase类

Pytest：函数 类 方法必须以tes\_开头

### 前置后置

方案一：

沿用了unittest的风格 前置方法 setup teardown

方案二：

Fixture

### 断言

Unittest 继承testCase后有自己的 self.assert

直接使用assert 后面跟 条件表达式 为真或为假

### 收集用例上

Unittest 需要用testload.discover（传入目录 和 文件名） 收集用例 run运行

Pytest:

不需要写测试套件收集用例 自动发现用例，

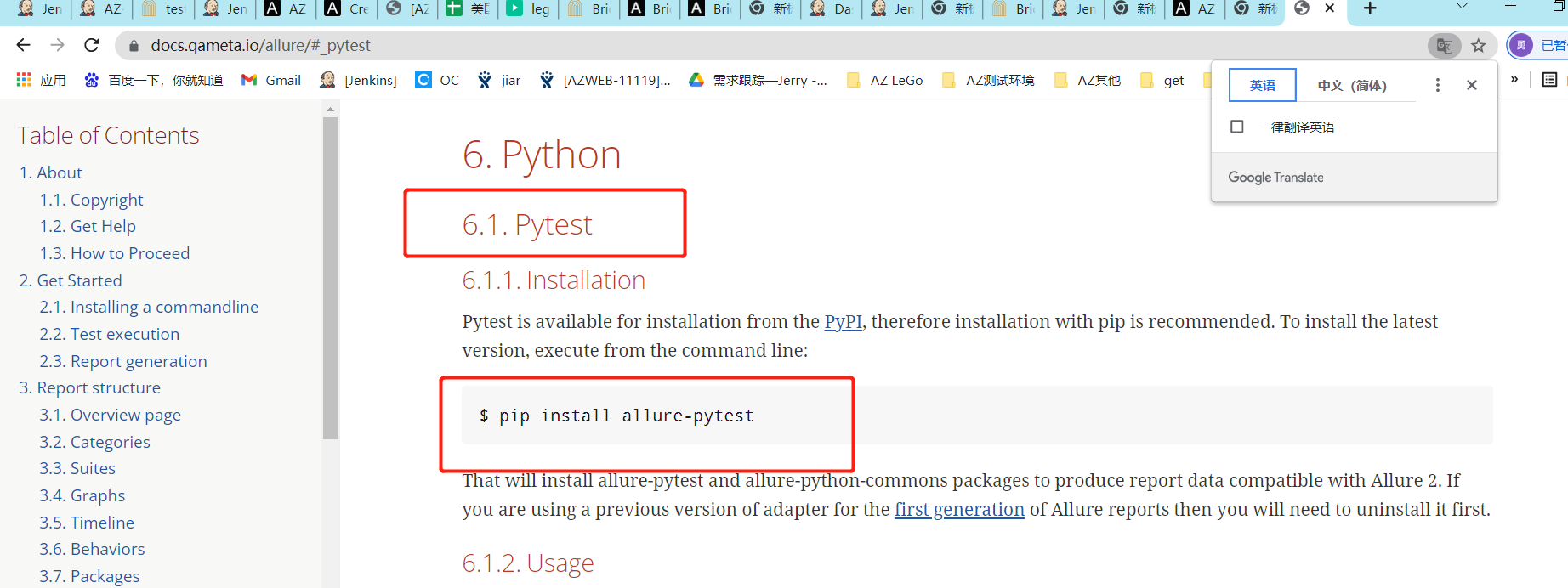
### Pytest 如何自动收集用例 ？

从目录下 搜索以test\_开头的用例

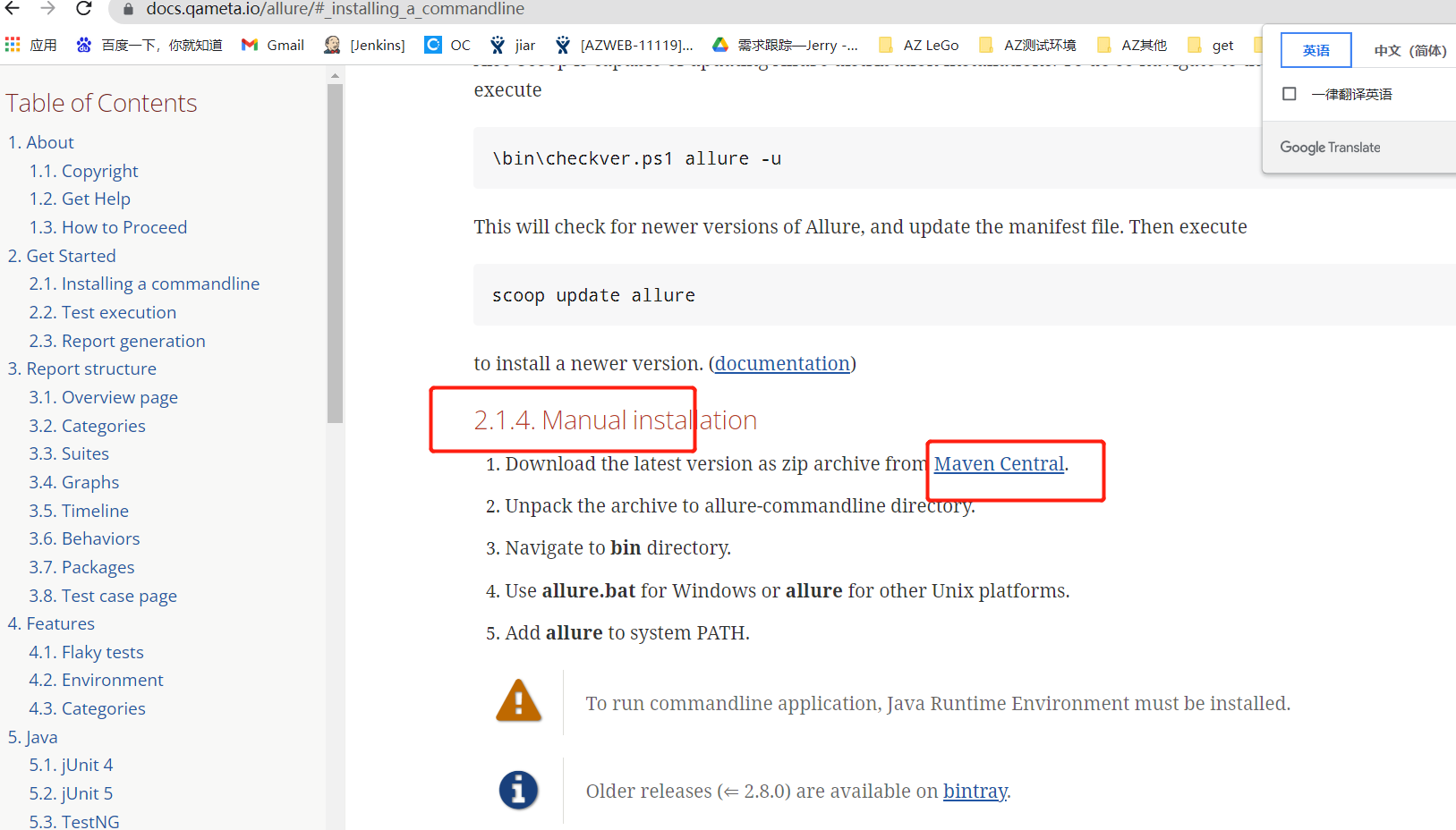
## Pytest allure

Allure官网: <https://docs.qameta.io/allure/#_installing_a_commandline>

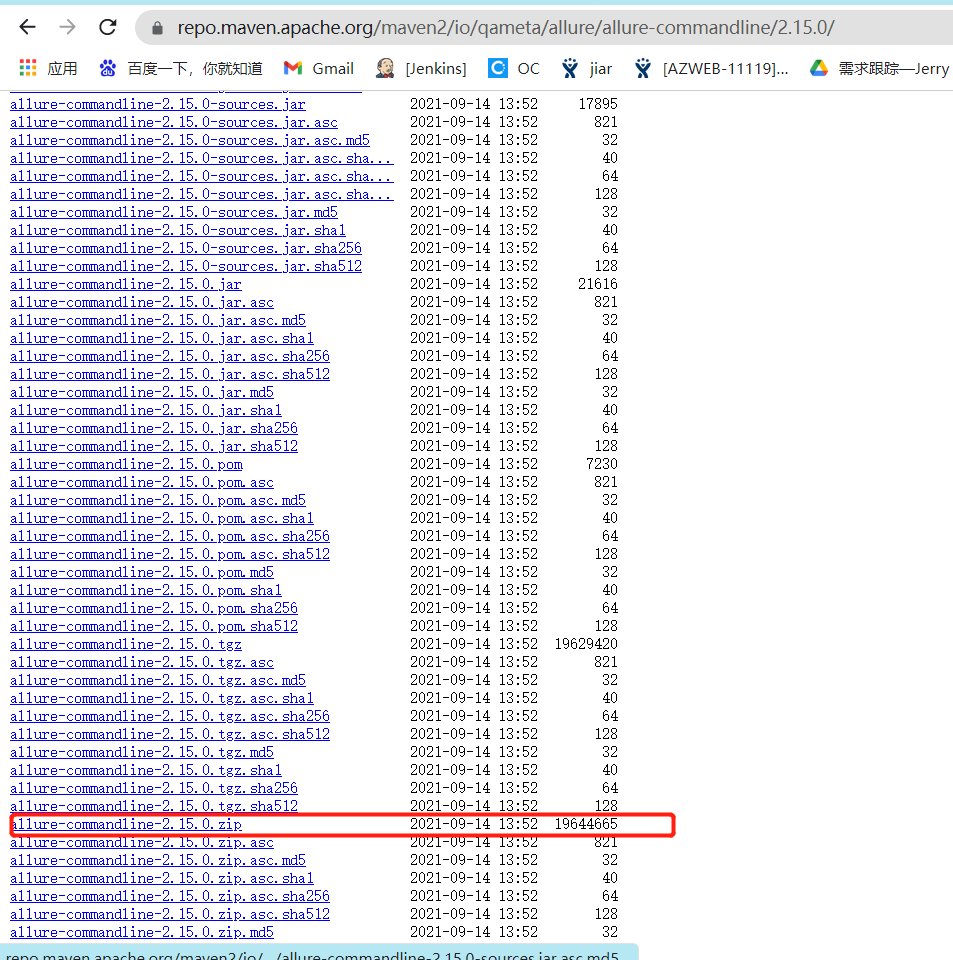
### 安装命令：



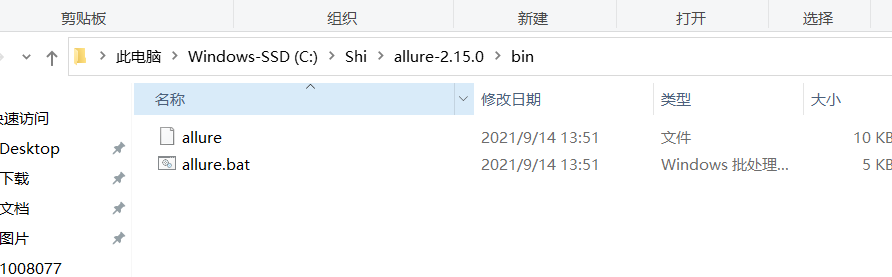
### 下载allure命令行



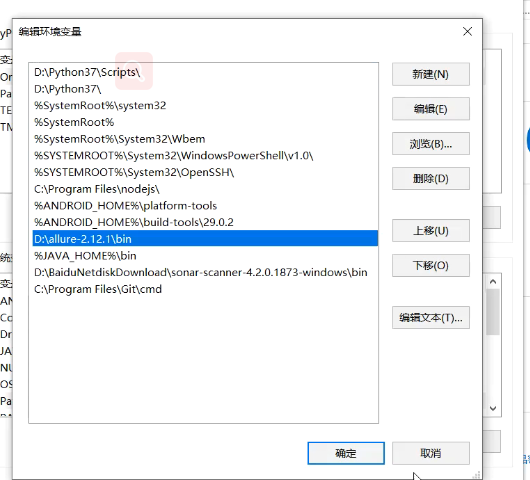
### 下载包



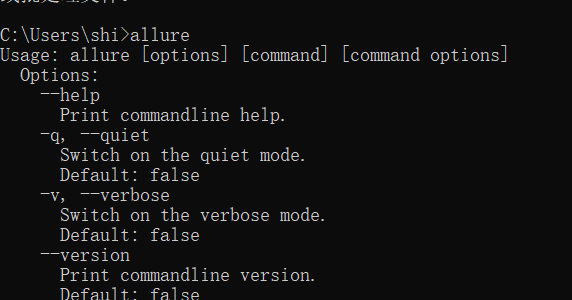
### 解压 进入bin目录



### 把bin目录配置到环境变量



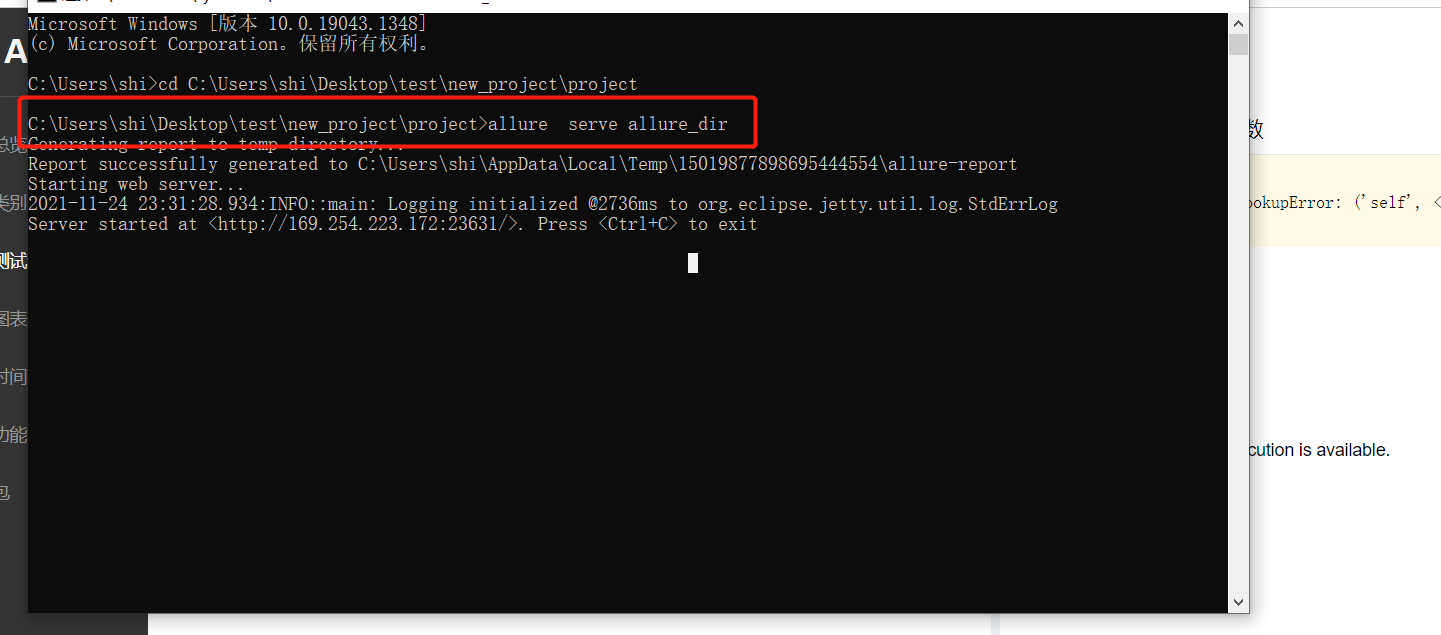
### 打开控制台查看allure



### 控制台 进入 allure文件夹 输入命令

输入命令 拉起 allure报告 html

allure serve allure\_dir



## Fixture

### 定义 方法/函数 级别fixture

# fixture ,scope默认为方法级别，如果需要类/模块级别，需要装饰器传入参数  
#@pytest.fixture(scope="class")  
  
  
#定义 方法级别  
@pytest.fixture*()*def main\_fixture*()*:  
 print*(*'fixture前置条件-----------------》'*)* yield  
  
 print*(*'fixture后置条件-----------------》'*)*

### 调用fixture |函数 or 方法

#普通函数  
@pytest.mark.usefixtures*(*'main\_fixture'*)*def test\_common\_function*()*:  
 print*(*'运行普通函数---------------------》'*)*

### 类里面每个方法都使用fxiture

@pytest.mark.usefixtures*(*'main\_fixture'*)*class Test\_type:  
  
 def test\_function1*(*self*)*:  
 print*(*'运行function1方法---------------------》'*)* def test\_function2*(*self*)*:  
 print*(*'运行function2方法---------------------》'*)* def test\_function3*(*self*)*:  
 print*(*'运行function3方法---------------------》'*)*

### 定义类级别fixture，传入scope=class

#定义类级别fixture,每个类只运行一次此前置后置  
@pytest.fixture*(*scope="class"*)*def class\_fixcure*()*:  
 print*(*'类级别fixture前置条件=============================》'*)* yield  
  
 print*(*'类级别fixtur后置条件=============================》'*)*

### 调用类级别fixture 只能装饰再类名上面，可以结合方法级别的一起用

#类级别  
@pytest.mark.usefixtures*(*'class\_fixcure'*)*#方法级别  
@pytest.mark.usefixtures*(*'main\_fixture'*)*class Test\_type:  
  
 def test\_function1*(*self*)*:  
 print*(*'运行function1方法---------------------》'*)* def test\_function2*(*self*)*:  
 print*(*'运行function2方法---------------------》'*)* def test\_function3*(*self*)*:  
 print*(*'运行function3方法---------------------》'*)*w

### 前置条件 返回数据给用例中使用yield

#定义方法级别fixture  
@pytest.fixture*()*def main\_fixture*()*:  
 print*(*'fixture前置条件-----------------》'*)* yield True**,666** print*(*'fixture后置条件-----------------》'*)*

### 用例使用前置条件返回数据

#普通函数  
@pytest.mark.usefixtures*(*'main\_fixture'*)*def test\_common\_function*(*main\_fixture*)*:  
 print*(*'运行普通函数---------------------》'*)* print*(*'这是用例内取到的 前置返回值：{}'.format*(*main\_fixture*))*

### 类级别 使用前置条件返回值

# #类级别  
@pytest.mark.usefixtures*(*'class\_fixcure'*)*#方法级别  
@pytest.mark.usefixtures*(*'main\_fixture'*)*class Test\_type:  
  
 def test\_function1*(*self**,**class\_fixcure*)*:  
 print*(*'运行function1方法---------------------》'*)* print*(*'这是类前置条件返回的值：{}'.format*(*class\_fixcure*))* def test\_function2*(*self*)*:  
 print*(*'运行function2方法---------------------》'*)* def test\_function3*(*self*)*:  
 print*(*'运行function3方法---------------------》'*)*

### Fixture 共享目录搜索机制

新建一个conftest.py 文件，将fixture 放在其中，这样该项目中的所有测试都能共享其中的fixture

不同的目录下可以创建多个conftest.py文件 就近原则取

优先级 ：就近原则

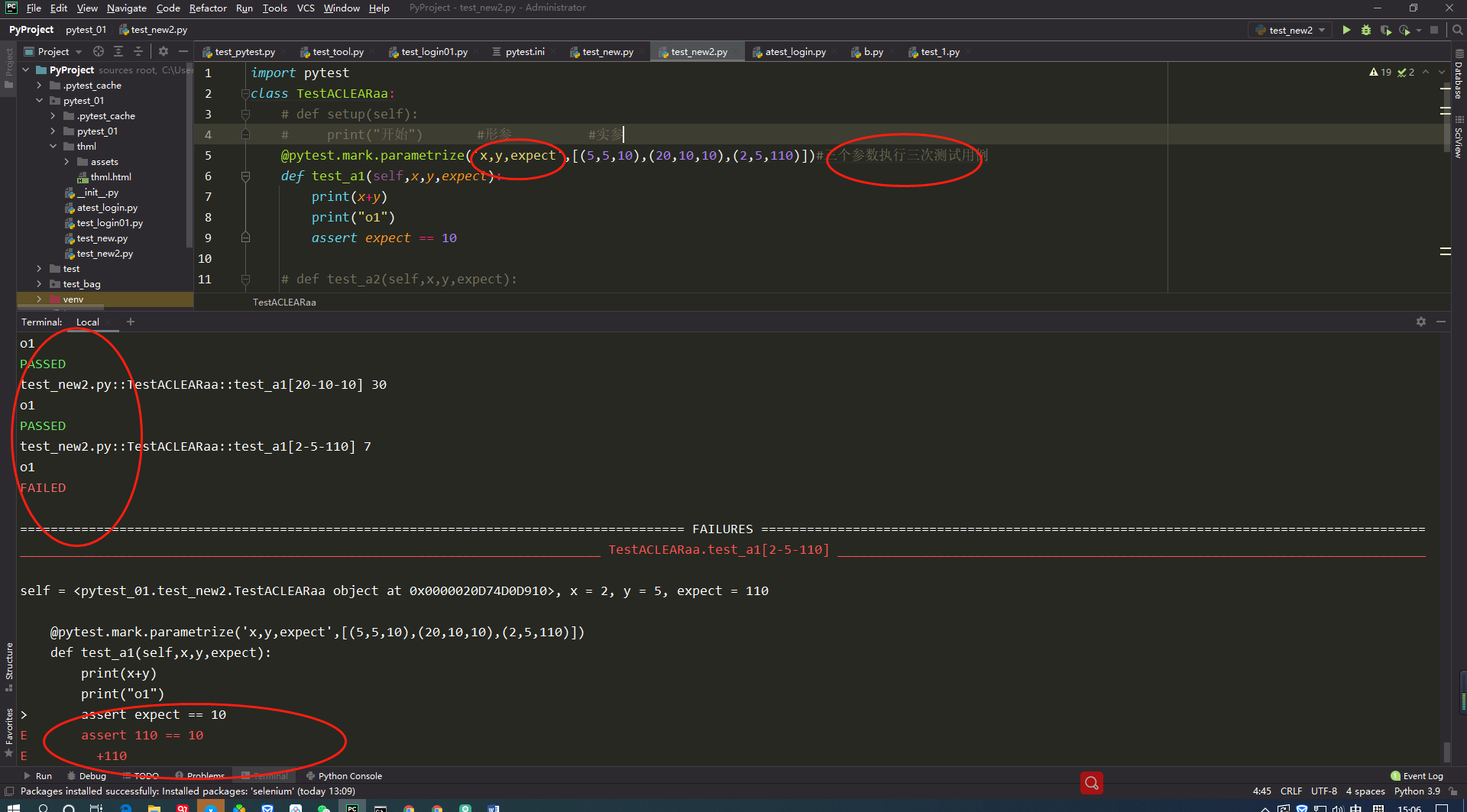
### Fixture 嵌套



## Pytest参数化

实参是以列表包 元祖 or 字典的形式传递的，多少个字典运行执行多少次

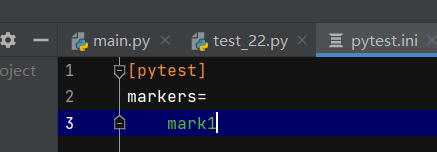
cases1=Excel\_data*(*os.path.join*(*path.excel\_dir**,** 'excel\_data.xlsx'*)***,** 'login'*)*case2=cases1.all\_data*()*#构造数据  
# @unittest.skip  
@pytest.mark.usefixtures*(*'class\_fixture'*)*@pytest.mark.parametrize*(*'case'**,**case2*)*class Test\_login*()*:  
  
 #前置方法  
 def setUp*(*self*)*:  
 pass  
  
 #后置方法  
 def tearDown*(*self*)*:  
 pass  
  
 #登录case  
 def test01\_login\_success*(*self**,**case*)*:  
 new\_log.info*(*"\*\*\*\*\*\*\*\*\* 执行用例{}：{} \*\*\*\*\*\*\*\*\*".format*(*case*[*"id"*]***,** case*[*"case\_name"*]))* self.\_testMethodDoc = case*[*'case\_name'*]* res=send\_request*(*'post'**,**case*[*'url'*]***,**case*[*'data'*])* assert json.loads*(*case*[*'expect'*])[*'code'*]* == jsonpath*(*res.json*()***,**'$.code'*)[***0***]*



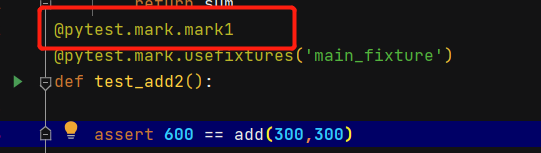
## pytest 打标记

### 创建pytest.ini文件

### 注册标记



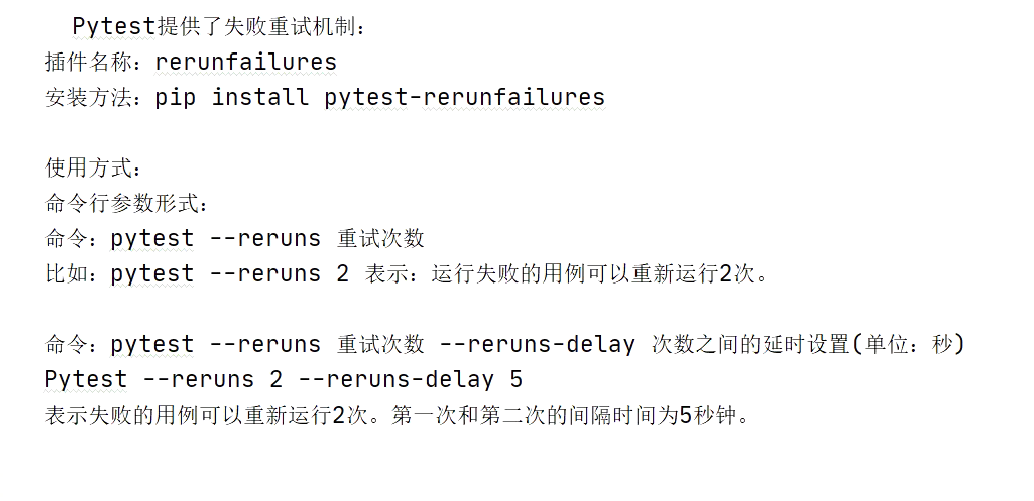
### 2测试用例打上标记



### 命令行执行标记 -m

1. pytest.main*([*'-s'**,**'-v'**,**'-m mark1'*])*

## pytest 失败重跑



pytest.main*([*'-s'**,**'-v'**,**'--reruns'**,**"2"**,**'--reruns-delay'**,**'5'*])*

## Pytest 断言assume

一个可以允许pytest测试用例中执行多个失败的断言的插件（即上面断言1，断言2，断言3都失败的情况下，三个断言都能被执行）

# Html

## 前端：

Html+ccs+javascript

Html:页面的内容表达

Css：样式的布局

Javascript：动态处理页面+用户响应

## Html元素特征

·标签明，属性，文本内容

## 哪些属性所有元素都可以有

style : 内联样式。直接在元素上通过style属性来设置样式。 display:none 不显示。  
id: 身份证。在页面当中，id值不会重复。唯一性。 可能是变动的(每次访问时id不一样)  
class: 类。class值不是唯一的。一个元素可以有多个class值，用空格隔开。  
name: 不唯一。低于id的一个可以用来识别元素的特性。

## 元素有哪些状态：

存在 - 存在于html当中。  
可见 - 在存在html的前提之下，有可见和不可见。  
可用 - 在可见的前提之下，元素有可用和不可用

## 8大定位方法

Id：元素的 身份证号唯一的

Name：元素的名字

Class：元素的class属性

Tag：元素的标签名

Link\_test:链接的文本内容

Partial\_link\_text:包含链接的文本内容

Xpath：组合元素的多种特种来定位元素

## 常用的html元素：

### input读取用户输入的元素

#### 常用属性：tetx,radio,file,button

#### H12345~标题

# Chromedriver download

http://npm.taobao.org/mirrors/chromedriver/

# Outo\_Web

## find.element和findelements的区别：

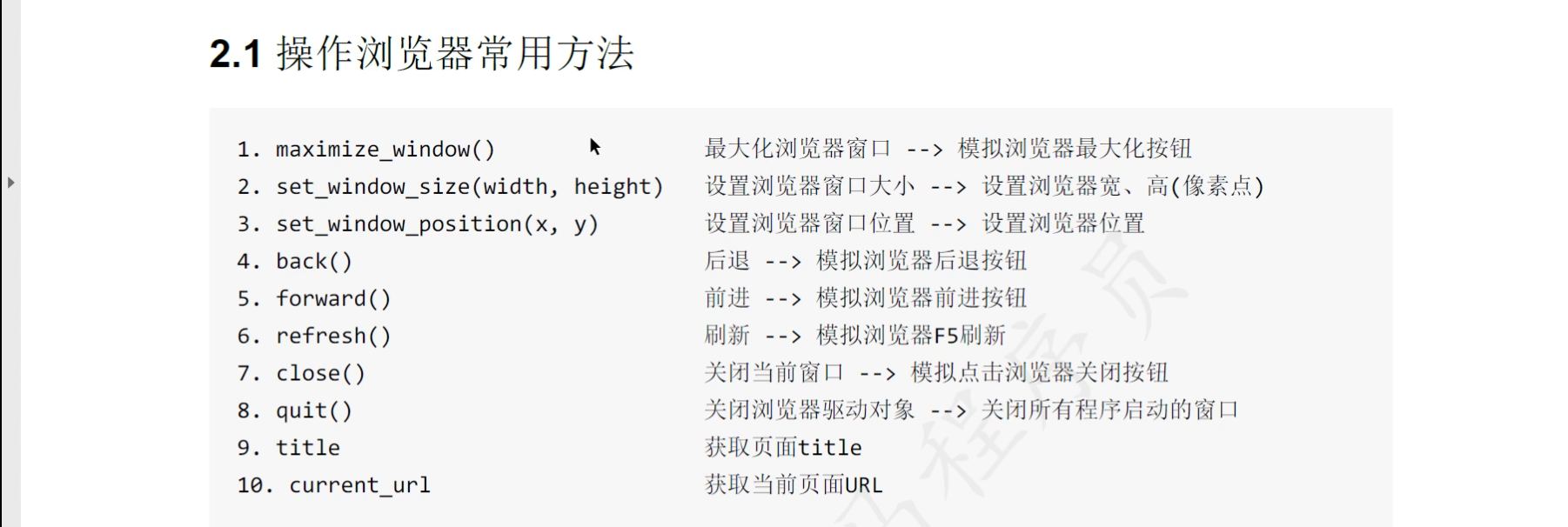
find.element:返回的是一个元素对象

find.element:返回的是一个列表，列表内是webelement

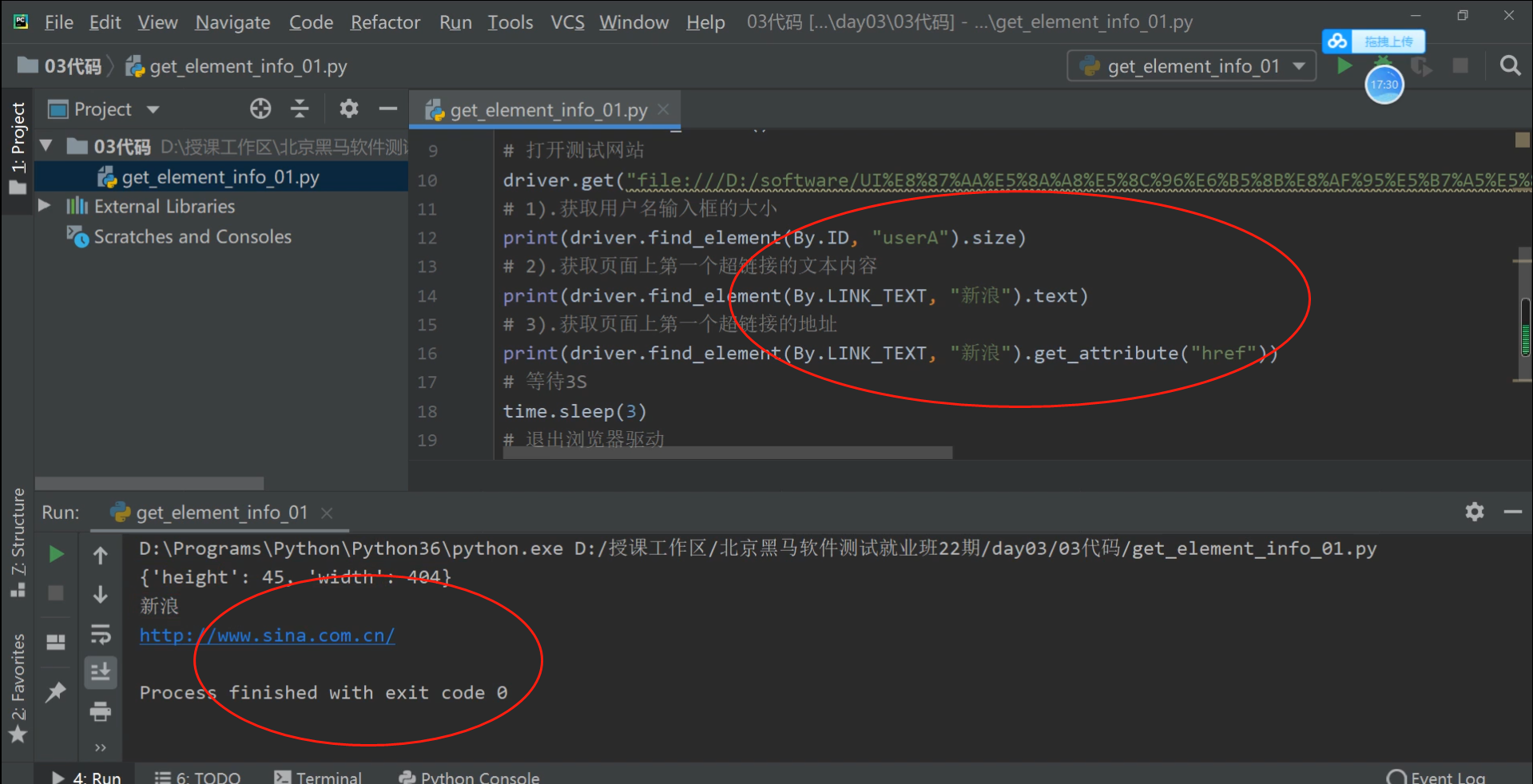
Findelements找不到元素 不会抛出异常，会返回空列表

Findelement 找不元素会抛出异常

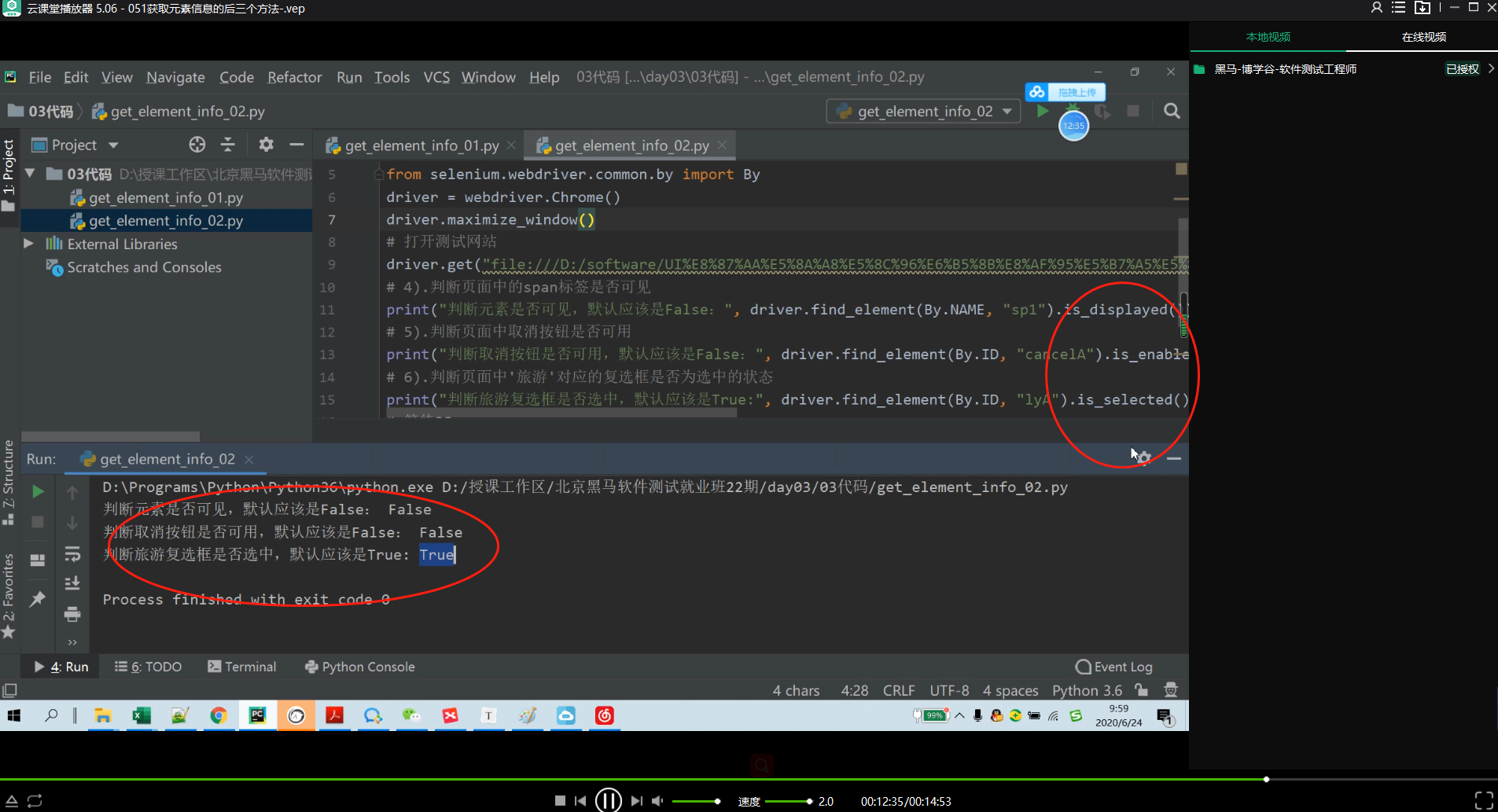
## 操作浏览器常用方法



### 操获取元素的大小/text/属性名



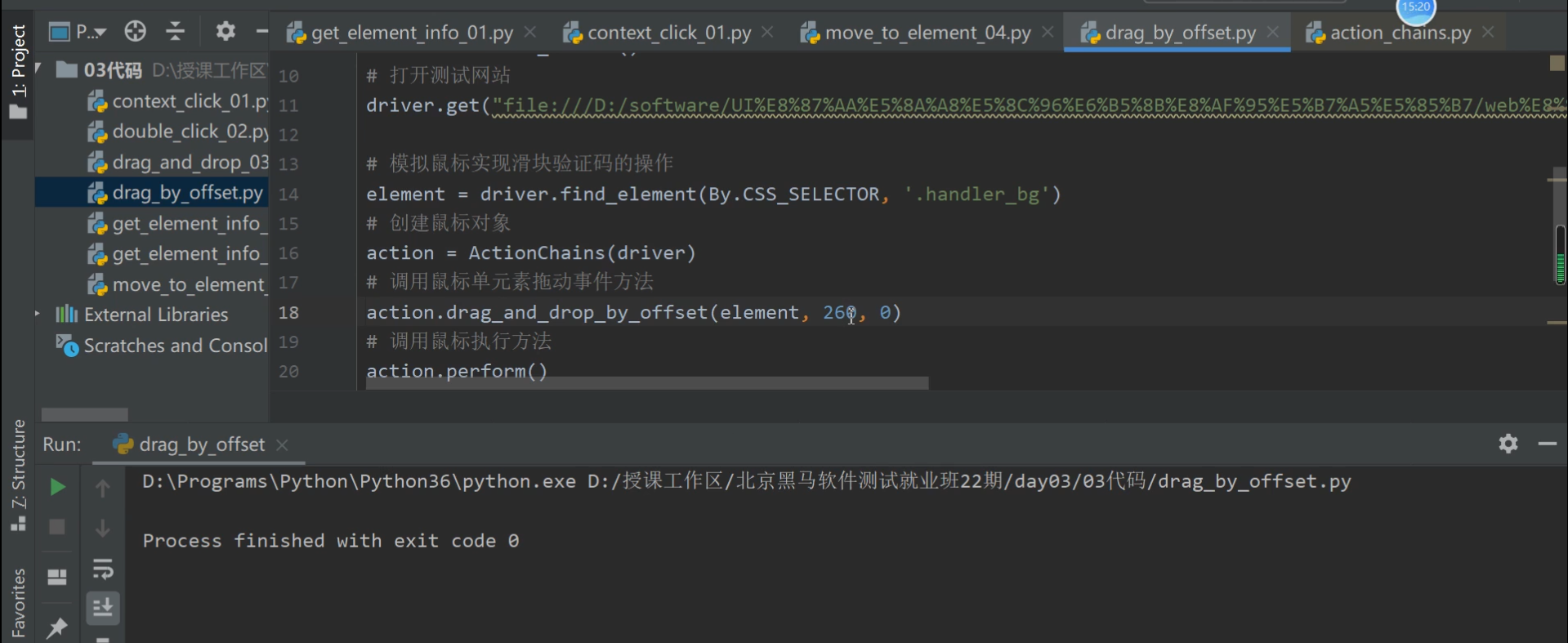
### 判断元素是否可用/可见/勾选



### 鼠标操作事件



### 鼠标单元素操作，拖拽元素要大于定位到的元素



## 元素定位

### Xpath

#### 相对路径的用法

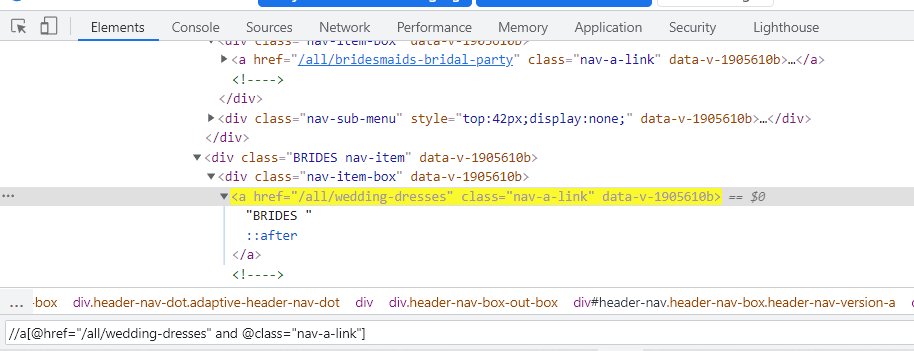
//标签名[@属性名=值]

//标签名[text()=”文本”]

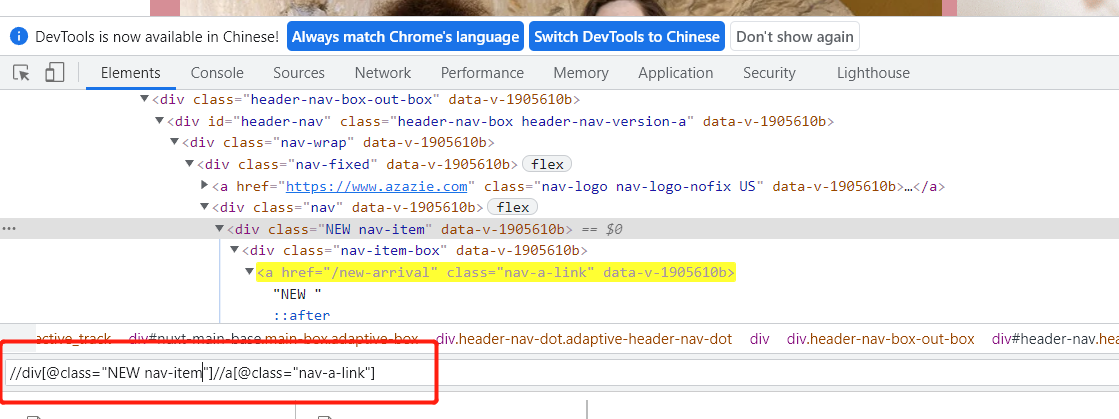
//标签名[contains(text(),“文本“)]

#### 组合条件

//a[@href="/all/wedding-dresses" and @class="nav-a-link"]



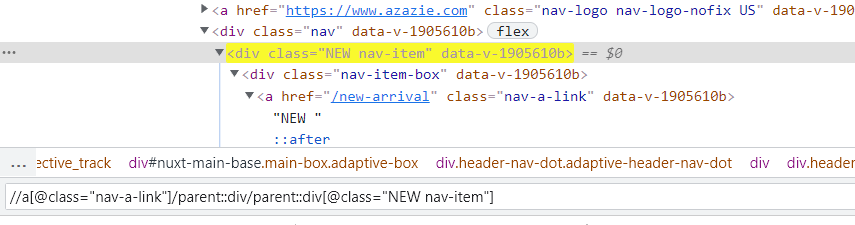
#### 层级定位



#### 轴定位

拿已知元素 找父级元素\同级元素 \*代表标签名

##### parent::/反向找

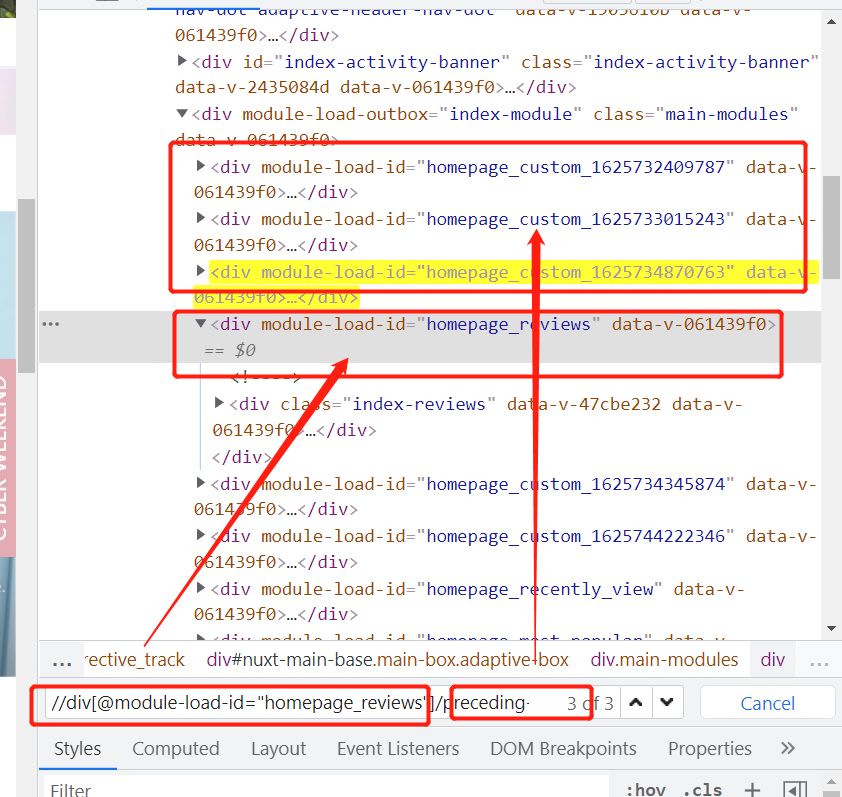


##### preceding-sibling

查找当前元素同级目录前面的元素

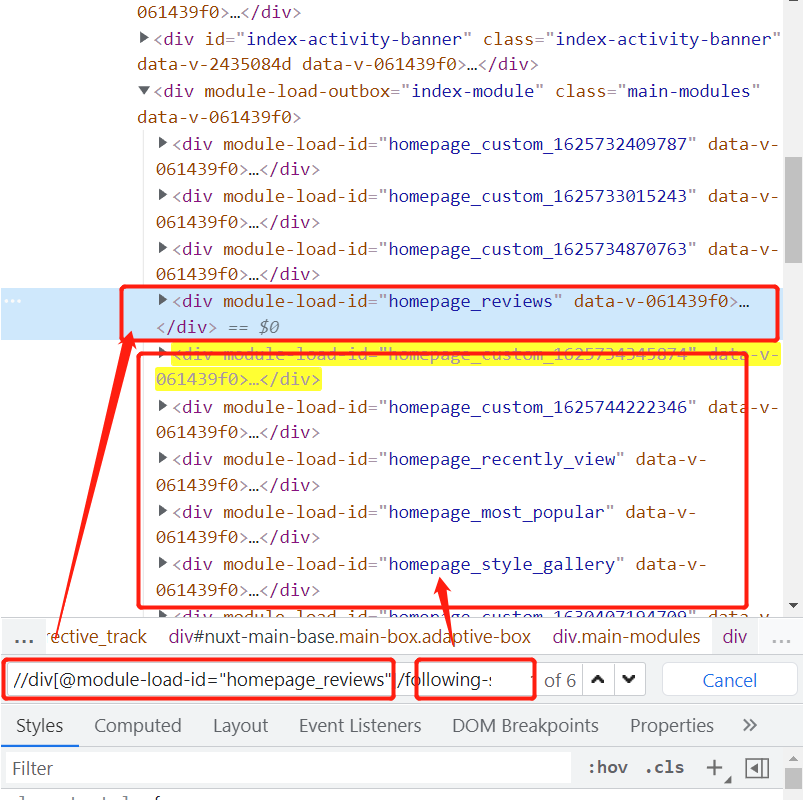
'//div[@class="c-select-dropdown adv-ft-dropdown"]/preceding-sibling::div'*)*

用于 当前元素定位不到 用同级目录指向定位



##### Following-sibling

//input[@id="\_password"]/following::span[@class="help-inline"]



### Css选择器 无法定位文本

https://www.w3school.com.cn/cssref/css\_selectors.asp



## By类

from selenium.webdriver.common.by import By

link\_element=By.LINK\_TEXT**,**'贴吧'  
element.find\_element*(*\*link\_element*)*.click*()*

## 等待

### 强制等待

1. time.sleep*(***3***)*

强制休息 代码停止运行

### 隐性等待

针对全局代码的 每当元素找不到时就会进行等待 找到后再运行

driver.implicitly\_wait*(***5***)*

### 显示等待

38秒

1. 导入webdriverwait 类
2. 导入expected\_conditions EC
3. 创建 webdriverwait 对象 传入driver 最大等待时间
4. Wait对象 调用 until方法,传入 ec.等待条件(传入元素)
5. #创建waite 对象传入 driver 最大等待时间  
   wait = WebDriverWait*(*driver**, 20***)*#定义元素表达式  
   element=*(*By.ID**,**'js\_login'*)*#wait对象调用until方法，传入 EC.(元素条件（元素表达式）)  
   wait.until*(*EC.visibility\_of\_element\_located*(*\*element*))*

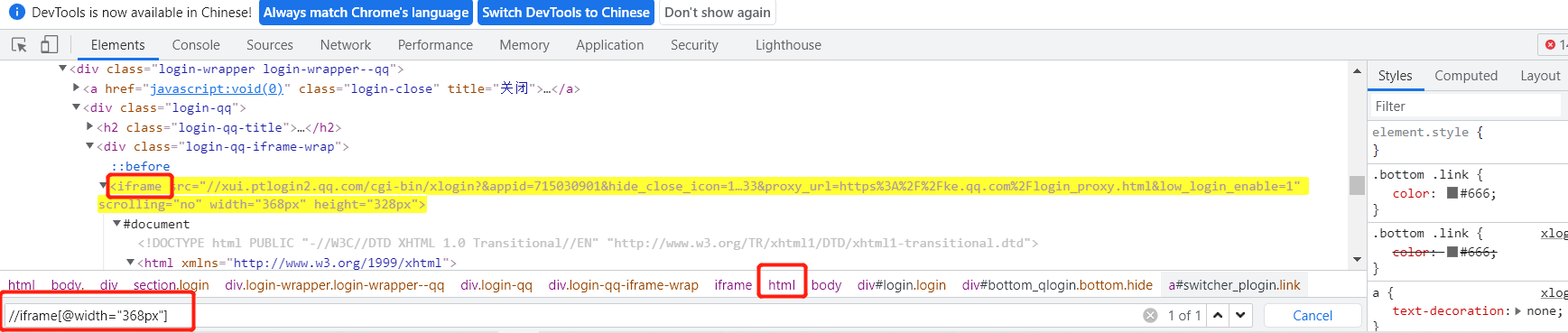
#### 等待条件

##### 元素存在：presence\_of\_element\_located

##### 元素可见：visibility\_of\_element\_located

##### 元素可点击：element\_to\_be\_clickable

## Iframe 网页内嵌网页



### iframe无属性 通过下标获取iframe

# #iframe无属性 通过下标获取iframe  
# iframe=driver.find\_elements(By.TAG\_NAME,'iframe')[2]  
#driver.switch\_to.frame(iframe)

### iframe无属性 通过元素定位返回iframe

# #iframe无属性 通过元素定位返回iframe  
iframe=driver.find\_element*(*By.XPATH**,**'//iframe[@width="368px"]'*)*driver.switch\_to.frame*(*iframe*)*

### iframe有具体的name和 id 属性

（id和name的属性值如果是动态变化的数字 就不能使用）

driver.switch\_to\_iframe("id or name 的属性值")

### 退出iframe 到默认列表页

#退回到初始html页  
driver.switch\_to.default\_content*()*

### 退回到上一级 iframe

Driver.switch\_to.parent.frame()

## Switch\_to.window

#获取网页所有窗口句柄  
win\_handle=driver.window\_handles  
  
print*(*'打印当前句柄：{}'.format*(*driver.current\_window\_handle*))*print*(*win\_handle*)*#切换到指定句柄  
driver.switch\_to.window*(*win\_handle*[***1***])*

## 弹窗处理

### 普通弹窗

alert=driver.switch\_to.alert #创建弹窗对象

alert.accept() #点击弹窗中的【确定】

三、点击取消

alert=driver.switch\_to.alert #创建弹窗对象

alert.dismis() #点击弹窗中的取消【】

四、双弹窗

alert=driver.switch\_to.alert #创建弹窗对象

driver.switch\_to.alert.dismis() #点击二次弹窗的取消

driver.switch\_to.alert.accept() #点击二次弹窗的确定

### 显示通知弹窗

#### 属于浏览器的设置 可以在打开浏览器时用代码配置浏览器

options = webdriver.ChromeOptions*()*prefs = *{* 'profile.default\_content\_setting\_values':*{* 'notifications':**2** *}  
}*options.add\_experimental\_option*(*'prefs'**,**prefs*)*driver = webdriver.Chrome*(*options = options*)*

## select 类处理下拉框



## 下拉框处理/鼠标悬浮

from selenium.webdriver.common.action\_chains import ActionChains

#获取要移动的元素对象  
ele=driver.find\_element*(*\*elc*)*#创建鼠标对象  
mouse=ActionChains*(*driver*)*#移动到指定对象 保存  
mouse.move\_to\_element*(*ele*)*.perform*()*

## 鼠标操作方法·

### Move\_to\_element

### Click

### Double\_click

### Click\_and\_hould

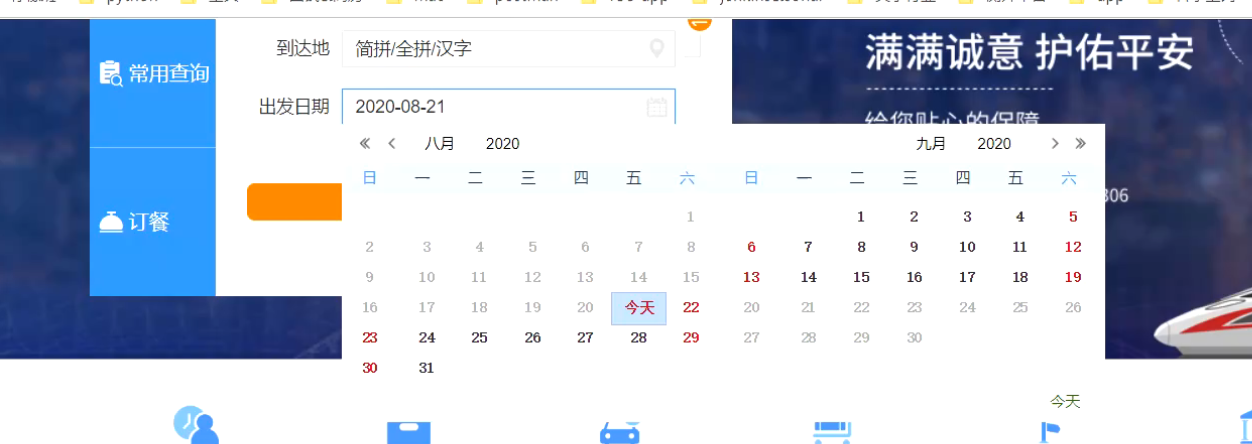
## JSDom对象

语言要对html进行操作 必须要经过dom对象

Javascript 通过dom对象对html的样式 和元素进行增删改查

### 应用场景

有些输入框选择日期 不支持输入



### Js 获取元素对象

var a=document.getElementById("kw")



### Js 修改元素值





### Js12306有id执行js

#访问链接  
driver.get*(*'https://www.12306.cn/index/'*)*els=driver.find\_element*(*By.XPATH**,**'//input[@id="train\_date"]'*)*js="a=document.getElementById(\"train\_date\");" \  
 "a.readOnly=false;" \  
 "a.value=\"2021-11-15\""  
driver.execute\_script*(*js*)*

无id执行js

### #不知元素id js定位方法

#访问链接  
driver.get*(*'https://www.12306.cn/index/'*)*els=driver.find\_element*(*By.XPATH**,**'//input[@id="train\_date"]'*)*#不知元素id js定位方法  
js1="arguments[0].value=\"2021-11-15\""  
driver.execute\_script*(*js1**,**els*)*



Ui自动框架设计思想

一个页面一个类

## 页面截图

#当前时间  
current\_time=time.strftime*(*'%Y-%m-%d-%H-%M-%S'**,**time.localtime*())*file\_path=os.path.join*(*error\_picture\_dir**,**'{}\_{}'.format*(*page\_action**,**current\_time*))*self.driver.save\_screenshot*(*file\_path*)*

## Git

### Command



## Jenkins 集成Allure 配置

### Jenkins 三大概念

#### Job

任务配置，执行任务，任务结果展示

#### Plugins

插件

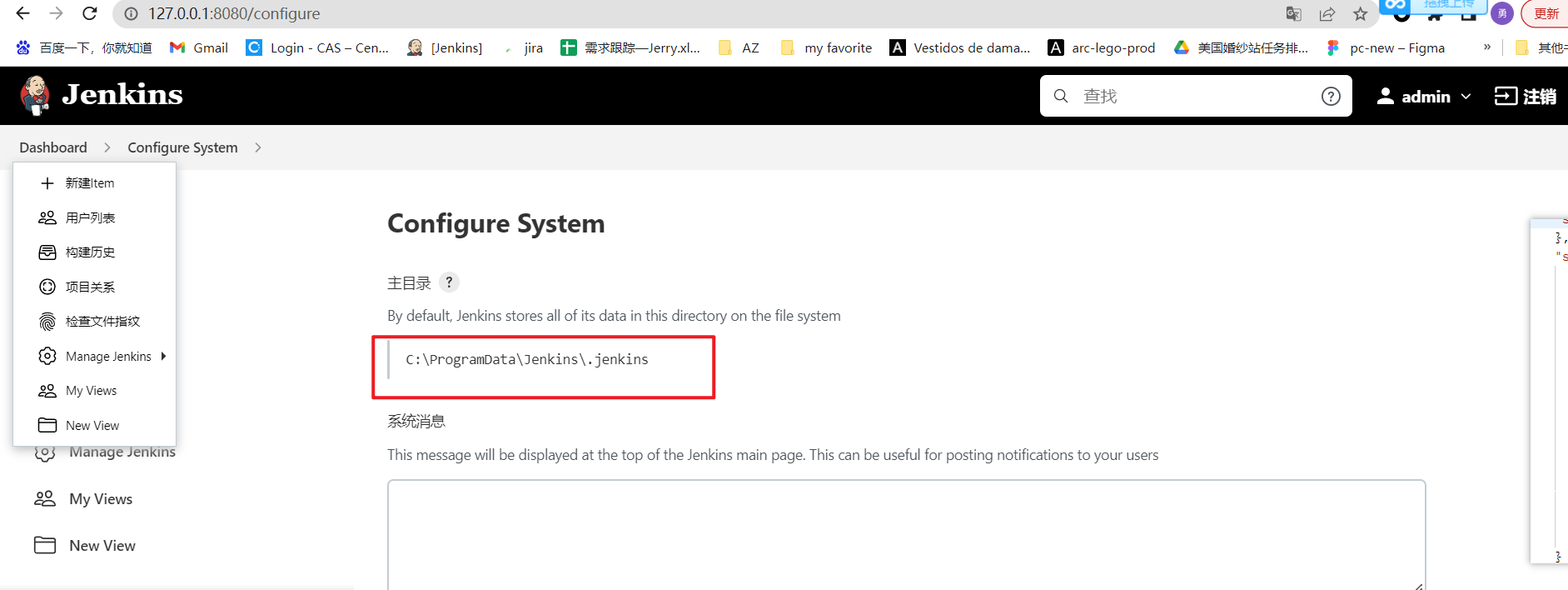
#### Work space、

工作空间 用来存放 任务输入文件、产出文件 比如git下载的项目文件

### 工作空间

Jenkins是用文件夹的形式管理的 不需要数据库

查看工作空间主目录



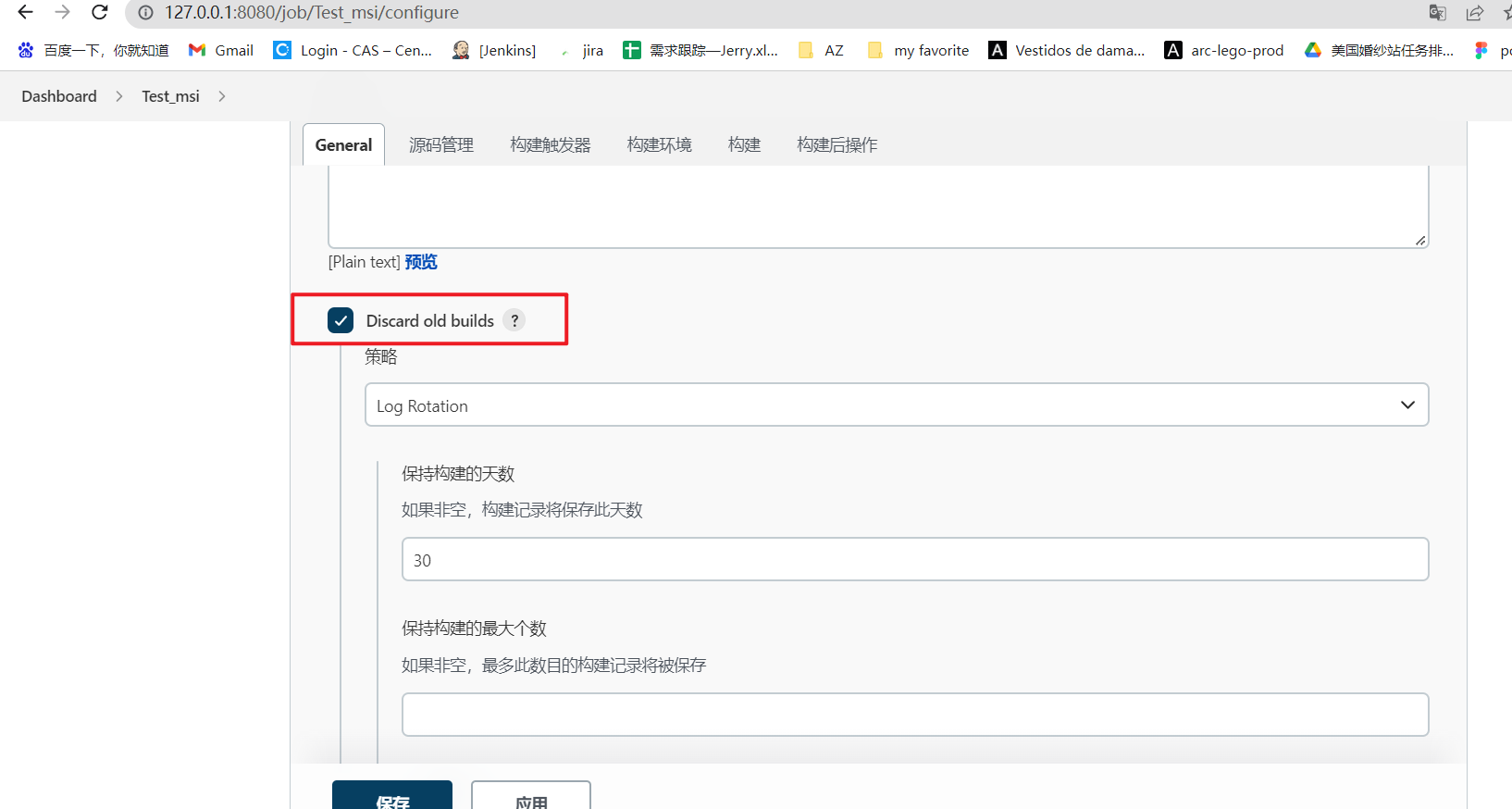
### Job配置

#### Git 配置

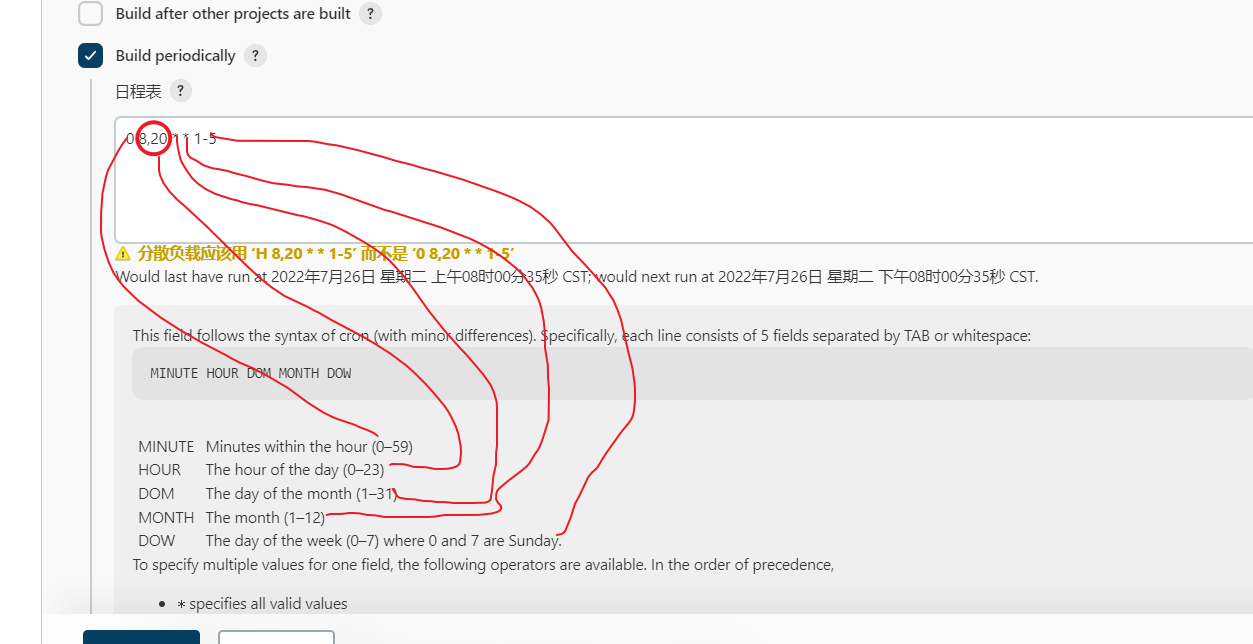
配置 git 库项目地址 ：跑job时会把项目文件下载到 本地.jenkins\work space文件夹内



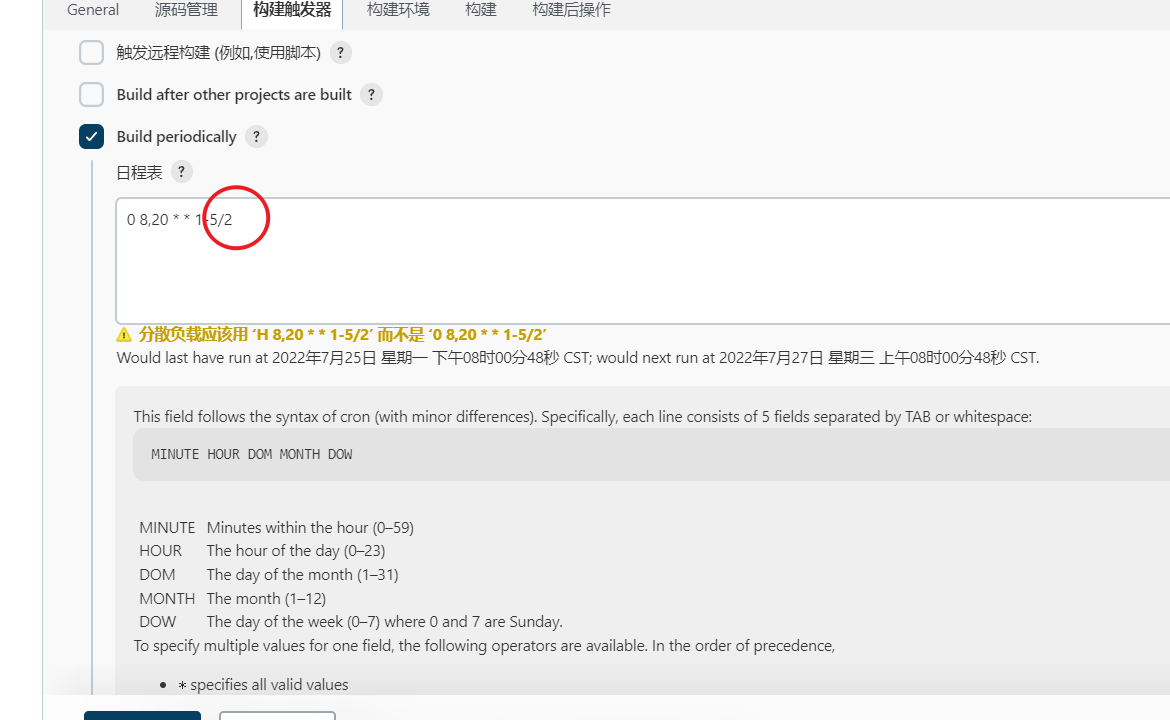
#### 工作空间定时清理



#### 定时构建



周1到周5 两天1次



#### 构建py文件

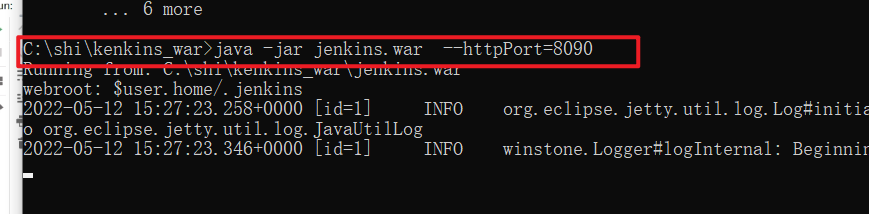
cd C:\Users\15572\Desktop\test\.virtualenv\new\_api-Q1fkvOT7\Scripts

python C:\ProgramData\Jenkins\.jenkins\workspace\Test\_msi\goods\_filter.py



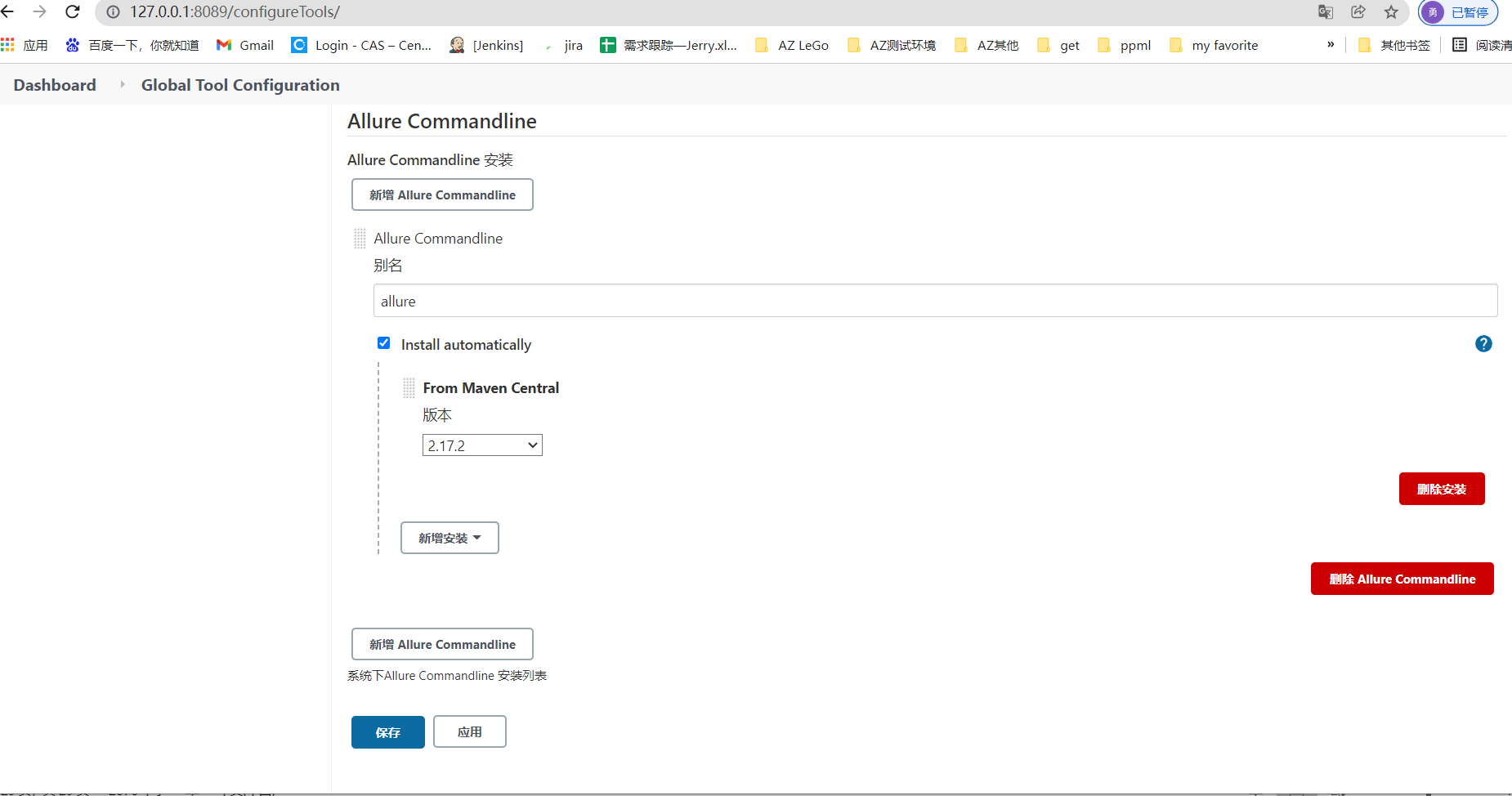
1. 下载jenkins 包
2. 本地运行enkins
3. 控制台启动 Jenkins

java -jar jenkins.war --httpRort=8089

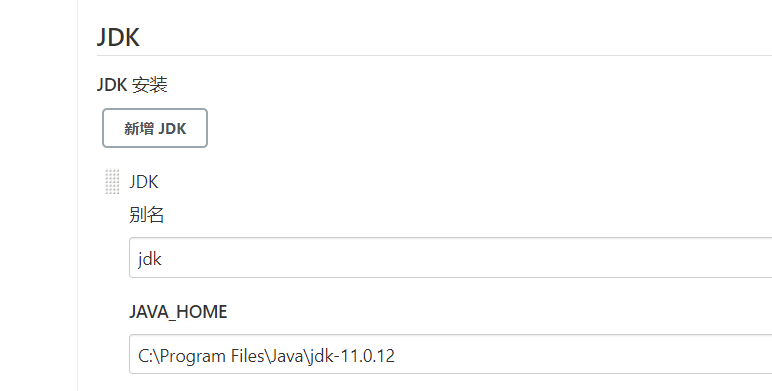


jenkins 下载allure插件

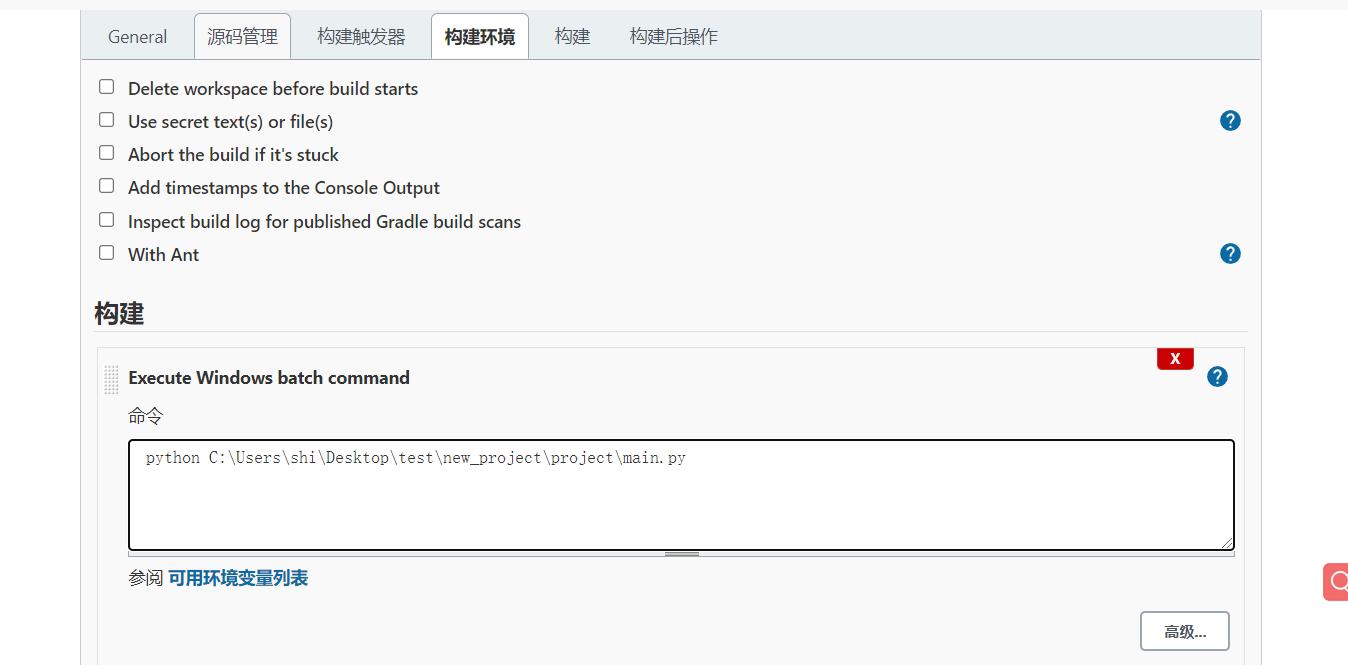
1. 进入jenkins 全局工具配置
2. 配置安装 allure



1. 配置jdk



1. 添加构建触发器



1. 添加构建后操作相对路径

