### 测试环境配置

#### python环境

https://www.python.org/downloads/windows/

安装时请勾选 Add to PATH

检测是否成功：

win+cmd >>python 会展示python版本

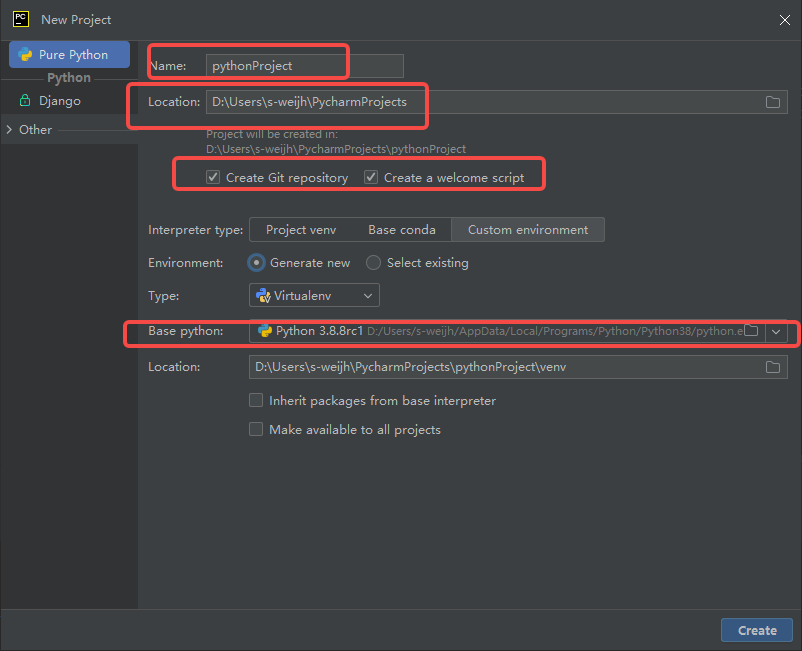
#### Playwright+pytest配置

Pycharm社区版的：

下载地址：

<https://www.jetbrains.com.cn/edu-products/download/download-thanks-pce.html>

#### Pycharm新建项目：



#### playwright环境 terminal

Pycharm terminal中输入

pip list 查看安装的文件

pip install --upgrade pip 升级pip命令

pip install playwright 安装playwright

playwright install 安装内置浏览器chrome，firefox，webkit

新建pythonpackage-

查看安装的文件：pip list

代码整理快捷键：ctrl+alt+L

#### pytest环境 terminal

pip install pytest-playwright

pytest --version #行以下命令来确认插件是否已正确安装

#### 多线程安装

pip install pytest-xdist

#### 多线程启动

在teminal中进入到要执行的文件夹中执行命令:pytest -n 10 testcases/

#### 用例执行加锁，顺序执行配合多线程

pip install filelock

pytest -n 10 -k test\_file #n是执行几个线程， -k模糊执行执行那一条用例

pytest -n 10 -k test\_file执行是需要在要执行文件加中

#### 单线程用例执行加锁

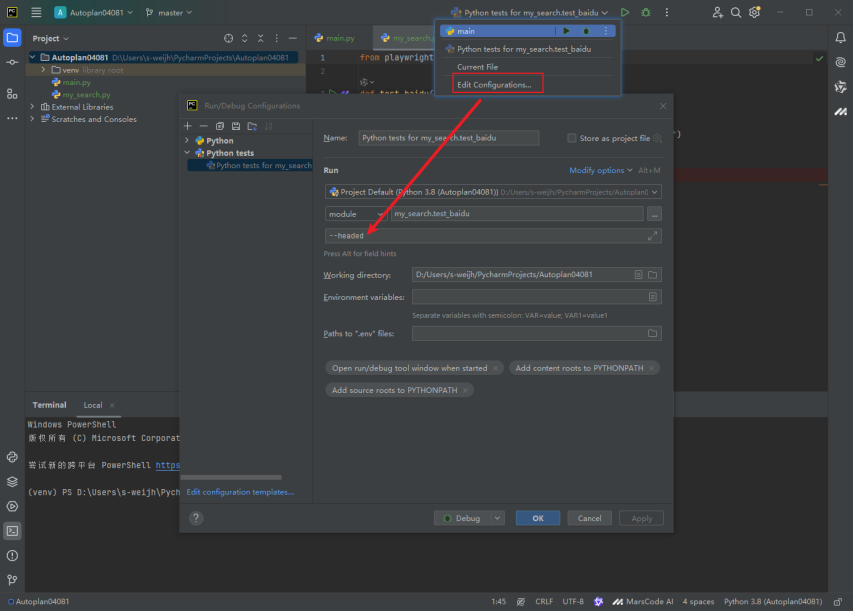
pytest -k filelock -k模糊执行执行那一条用例

#### Pytest查看命令：pytest --help

可以查pytest.ini中pytest内置的参数

#### 运行playwright用有头模式（可见浏览器）：

方式1：运行配置中添加：--headed



方式2：项目根目录下配置pytest.ini

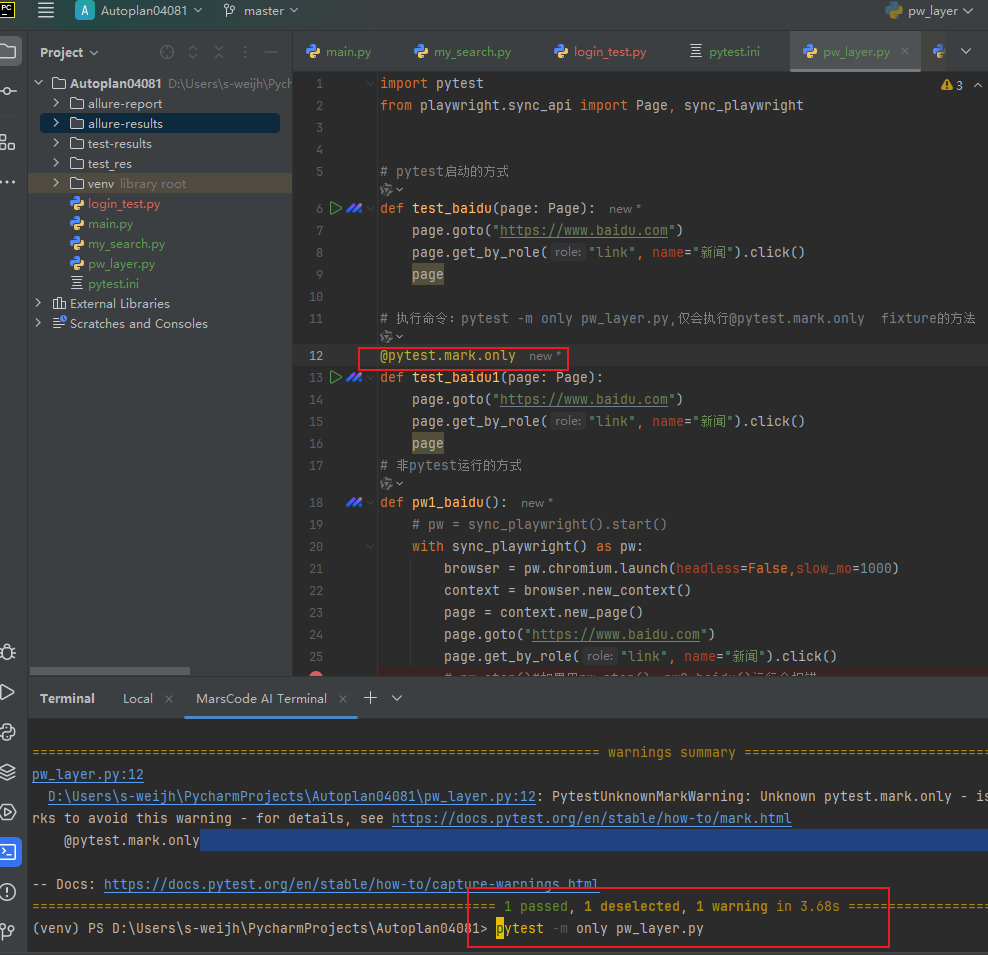
|  |
| --- |
| [pytest] addopts:  --headed  --browser=firefox ; --output=test\_res #参数可以指定测试结果的输出目录 ; --slowmo=1000 # 表示每个操作(点击、输入等)会延迟1000毫秒(1秒)执行 ; --screenshot=only-on-failure # 仅在测试失败时截图（默认值）  --screenshot=on #每次测试都截图（无论成功/失败） ; --screenshot=off #完全禁用截图 ; --video=on #启用视频录制功能，测试运行时会录制视频 ; --html=reports/report.html  # 生成HTML报告，安装:pip install pytest-html  --alluredir=allure-results # 生成Allure报告数据，保存到allure-results目录， ;1. 安装 Allure，npm install -g allure-commandline --save-dev ;2. 生成 Allure 报告，在包含 allure-result 文件夹的项目根目录下，运行以下命令来生成报告： ;allure generate allure-results -o allure-report --clean ;这个命令会读取 allure-results 目录下的数据，并生成一个 HTML 格式的报告在 allure-report 目录中。 ;3. 查看报告:allure open allure-report ;这将在默认的浏览器中打开 Allure 报告。你也可以直接在文件浏览器中打开 allure-report 文件夹，然后双击 index.html 文件来查看报告。 ;4. 清理和再次生成（可选） ;如果你需要清理旧的报告并重新生成，可以使用以下命令： ;allure clean allure-results # 清理结果目录 ;allure generate allure-results -o allure-report --clean # 重新生成报告 |

### Fixture的用法：

#### 要执行方法上加上@pytest.mark.only

执行命令：pytest -m only pw\_layer.py，仅会执行@pytest.mark.only fixture的方法

1 passed， 1 deselected， 1 warning in 3.68s ==



#### 用例启动前优先调用fixture的方法

configtest.py

如下

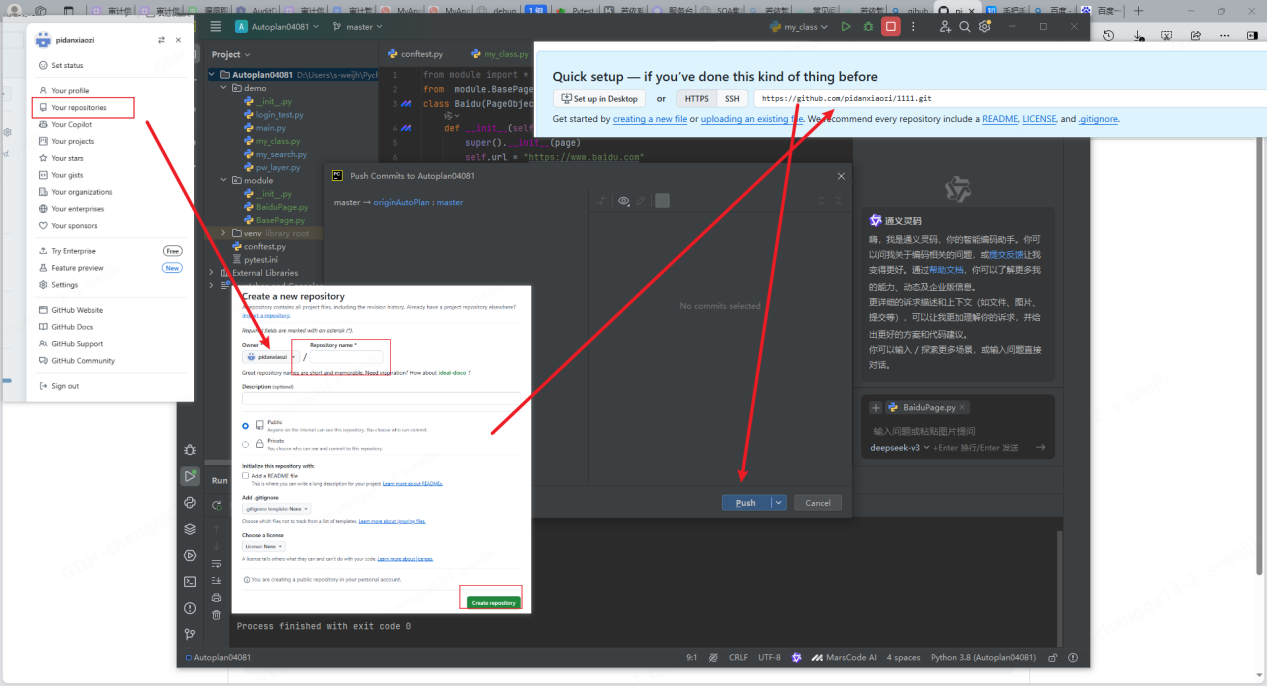
|  |
| --- |
| @pytest.fixture() def hello1():  print("hello")#运行test\_baidu1前会优先 print("hello")  yield  print("\n'bye'")#用例结束后会运行 print("\n'bye'") # 执行命令：pytest -m only pw\_layer.py，仅会执行@pytest.mark.only fixture的方法 @pytest.mark.only def test\_baidu1(page: Page，hello1):  page.goto("https://www.baidu.com")  page.get\_by\_role("link"， name="新闻").click() |
|  |
|  |

### 本地文件上传github的方法

点击右上角的头像旁边的小三角，选择Your repositories，NEW

填写repositorname，提交后，

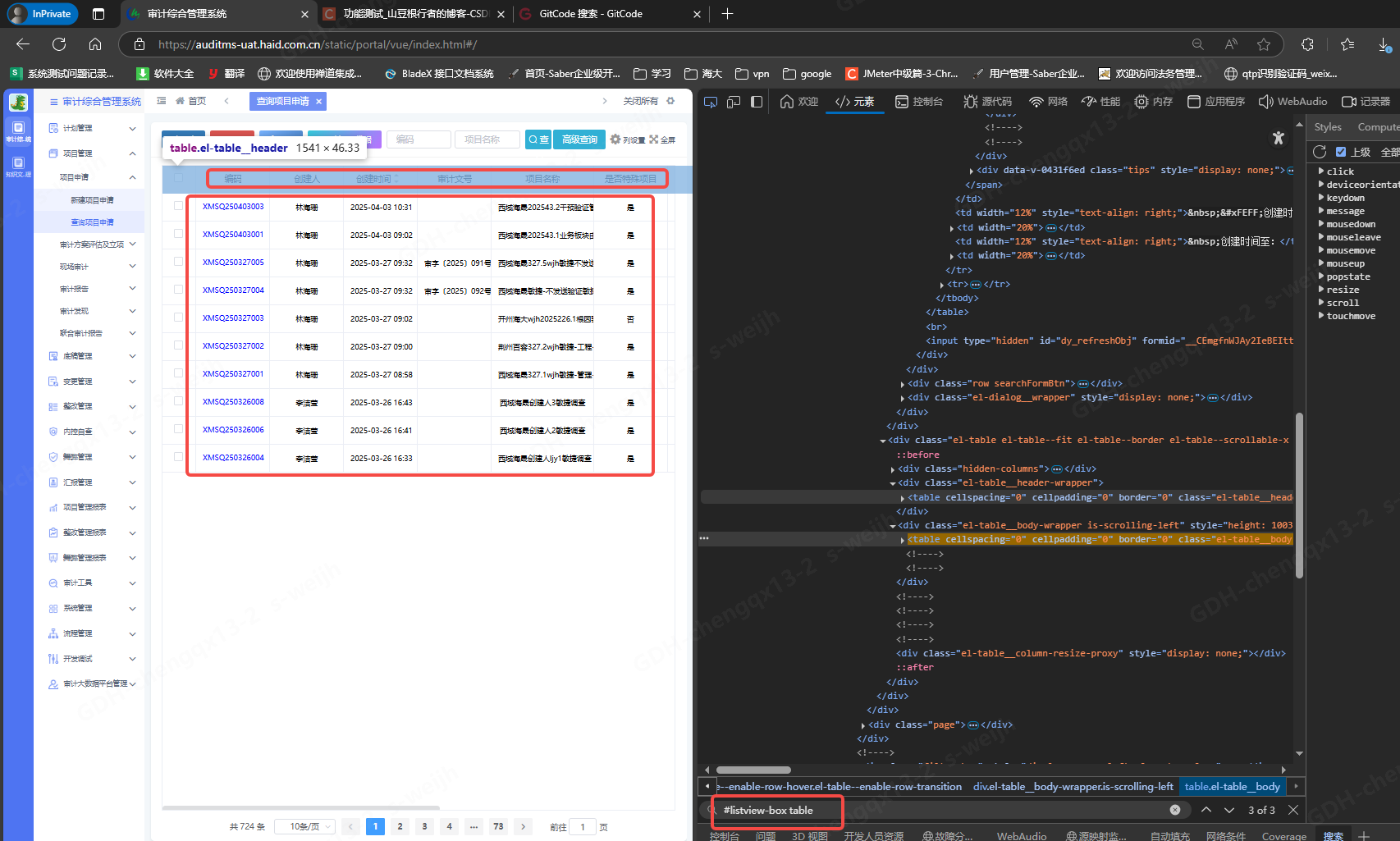
Pycham选择文件push，填写新建项目中复制下来的giturl



表格的封装：

找到表格的总div：后方输入table回显是处这个div下有几个表格如：

#listview-box table

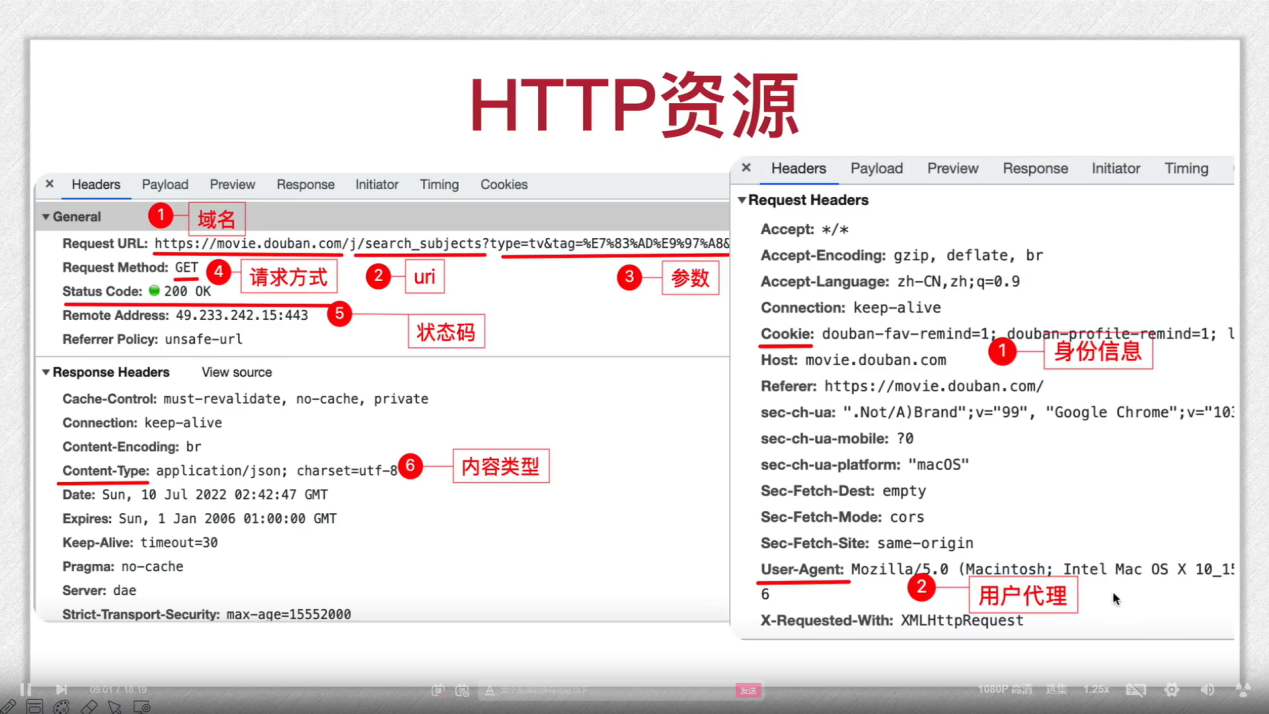


#### http

###### http三次握手



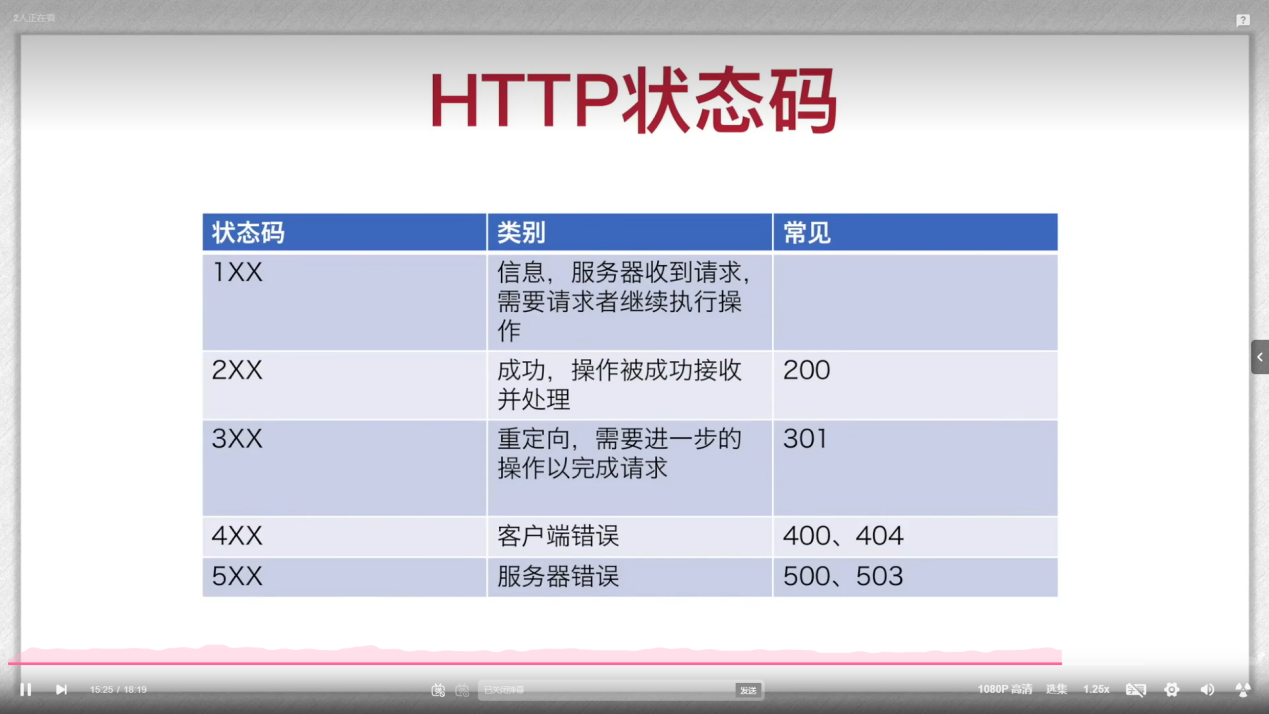
###### http资源



###### Http请求方式



###### Http状态码



#### Excel

序号自增：在需要生成序号的单元格中输入公式 =ROW()-1

#### Python

有返回值函数（fruitful functions）

无返回值函数（void functions）

进程（processes）和事件（events）

回溯（traceback）

回溯中的函数顺序，与堆栈图中的函数顺序一致。出错时正在运行的那个函数则位于回溯信息的底部。

#### Python命名规范

项目名称：大驼峰或中横线，

文件和文件夹：小写+下划线，

类：大驼峰，

方法和函数和变量：大写字母+下划线，

变量：小写+下划线，

常量：全大写+下划线，

实例化：在\_\_ini\_\_文件中

### playwright：

#### 简单录制命令：

* 1. playwright codegen https://auditms-uat.haid.com.cn/static/login/login.html
     1. 录制某个网址并记录在login\_test.py：

playwright codegen --target python -o login\_test.py -b chromium <https://auditms-uat.haid.com.cn/static/login/login.html>

#### playwright codegen方法

* 'assert visibility' 断言元素可见
* 'assert text' 断言元素包含特定文本
* 'assert value' 断言某个元素具有特定值

#### Assert Visibility

在 Playwright 的 `codegen` 工具中，`assert visibility` 是一种断言功能，用于验证页面上的某个元素是否可见。以下是使用 `assert visibility` 的步骤：

1. \*\*启动 Codegen\*\*：

- 使用命令 `playwright codegen` 启动测试生成器。你可以指定一个 URL 作为参数，或者在浏览器中手动输入 URL。

2. \*\*进行页面交互\*\*：

- 在打开的浏览器窗口中，执行你想要记录的操作，例如点击按钮或填写表单。

3. \*\*生成断言\*\*：

- 在 Playwright Inspector 的工具栏中，选择 `assert visibility` 图标。

- 然后点击页面上你想要验证可见性的元素。

- Playwright 将生成相应的断言代码，确保该元素在页面上是可见的。

4. \*\*复制生成的代码\*\*：

- 完成所有操作后，点击 Playwright Inspector 中的“复制”按钮，将生成的代码复制到你的编辑器中。

使用 `assert visibility` 可以帮助你在测试中确保页面元素的可见性，从而验证页面的正确渲染和布局。

##### Playwright codegen中的pick localtor

在 Playwright 的 `codegen` 工具中，使用“选择定位器”（Pick Locator）功能可以帮助你快速生成用于测试的定位器代码。以下是具体步骤：

1. \*\*启动 Codegen\*\*：

- 运行 `playwright codegen` 命令并提供要测试的网站 URL。例如：

```bash

npx playwright codegen demo.playwright.dev/todomvc

```

- 这将打开一个浏览器窗口和 Playwright Inspector。

2. \*\*停止录制\*\*：

- 在 Playwright Inspector 中，点击“录制”按钮以停止录制。此时，“选择定位器”按钮将出现。

3. \*\*选择定位器\*\*：

- 点击“选择定位器”按钮。然后将鼠标悬停在浏览器窗口中的元素上，查看每个元素下方高亮显示的定位器。

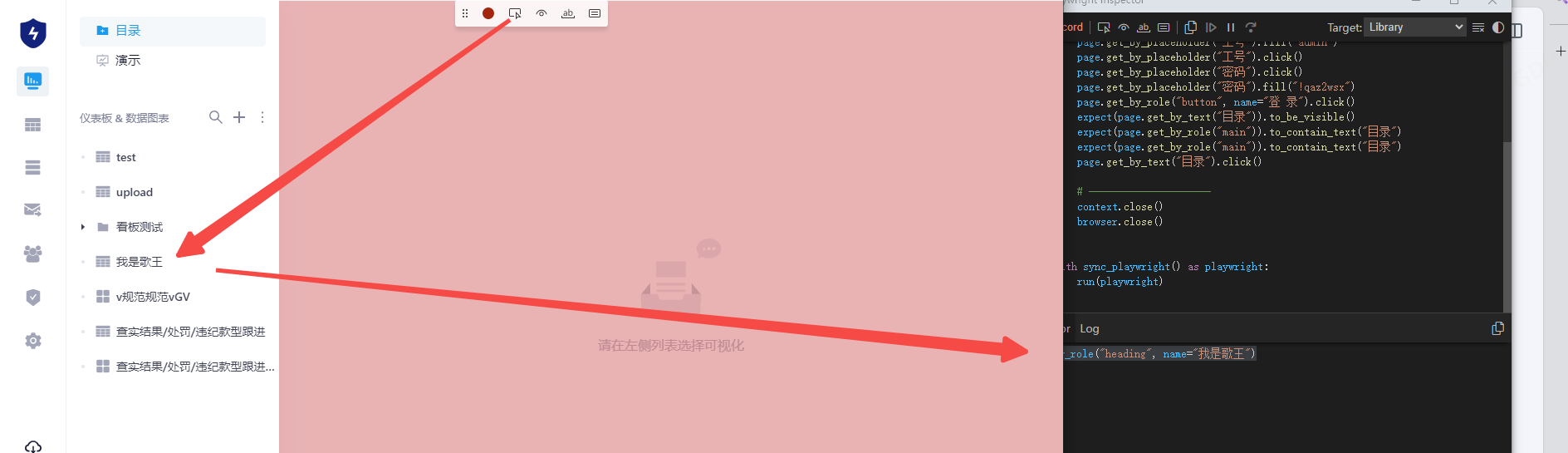
- 点击你想要定位的元素，该元素的定位器代码将显示在“选择定位器”按钮旁边的字段中。

4. \*\*编辑和复制定位器\*\*：

- 你可以在字段中编辑定位器以进行微调。

- 使用复制按钮将定位器代码复制到剪贴板，然后粘贴到你的测试代码中。

通过这些步骤，你可以快速生成和使用 Playwright 的定位器，从而简化测试脚本的编写过程.



##### Playwright codegen中的assert text

在 Playwright 的 `codegen` 工具中，使用“断言文本”（Assert Text）功能可以帮助你验证页面上的元素是否包含特定的文本。以下是具体步骤：

1. \*\*启动 Codegen\*\*：

- 运行 `playwright codegen` 命令并提供要测试的网站 URL。例如：

```bash

npx playwright codegen demo.playwright.dev/todomvc

```

- 这将打开一个浏览器窗口和 Playwright Inspector。

2. \*\*停止录制\*\*：

- 在 Playwright Inspector 中，点击“录制”按钮以停止录制。

3. \*\*生成文本断言\*\*：

- 在工具栏中选择“断言文本”图标。

- 然后点击页面上你想要验证文本的元素。Playwright 将生成相应的断言代码，确保该元素包含特定的文本。

4. \*\*查看生成的代码\*\*：

- 生成的断言代码将显示在 Playwright Inspector 中。你可以查看并复制这段代码，然后将其粘贴到你的测试脚本中。

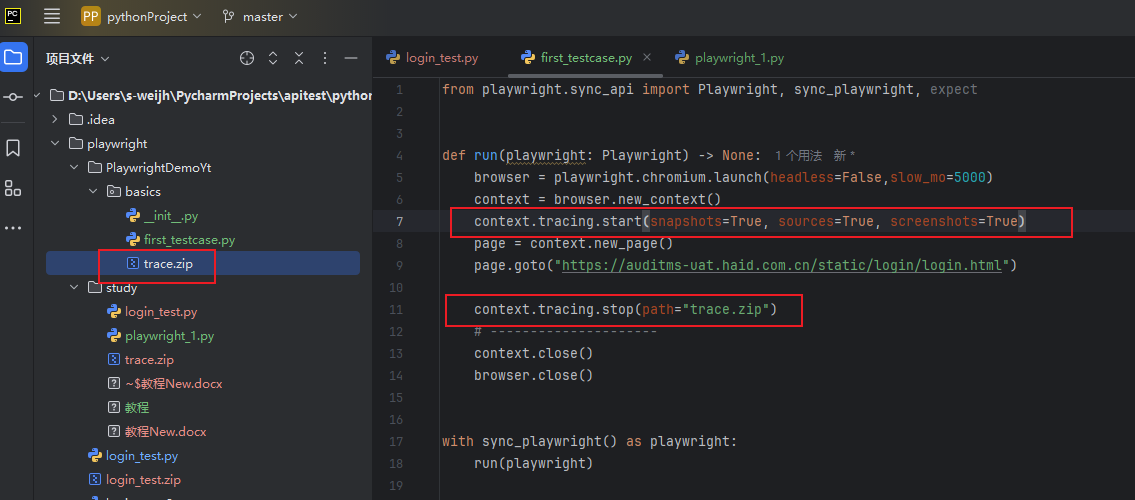
通过这些步骤，你可以快速生成用于验证页面元素文本的断言代码，从而确保页面内容的正确性。

#### 运行代码

配置运行速度（slow\_mo=5000）

browser = playwright.chromium.launch(headless=False，slow\_mo=5000)

#### 运行记录跟踪trace



#### 关于trace：

##### 启动

# 启动playwright driver进行

p = sync\_playwright().start()

print("1")

# 启动浏览器，返回Browser类型对象

browser = p.chromium.launch(headless=False)

print("2")

context = browser.new\_context()

context.tracing.start(snapshots= True，sources= True，screenshots= True)

3.2#代码执行完成后执行 保存的位置

context.tracing.stop(path="trace.zip")

#查看

playwright.exe show-trace trace.zip

4、CSS选择器定位元素

5、获取元素文本并打印

text1 = page.locator("div").all()

for text2 in text1:

print(text2.inner\_text())

##### 怎么查看这个跟踪文件呢？有2种方法：

直接访问 [trace.playwright.dev](https://trace.playwright.dev/" \t "_blank) 这个网站，上传 跟踪文件

执行命令 playwright show-trace trace.zip

### 常用方法API

### 获取所有的元素对应的 locator 对象，使用 all方法

locators = page.locator('.plant').all()

##### 有时，我们只需要得到某种表达式对应的元素数量 ，可以使用 count方法，如下

count = page.locator('.plant').count()

##### 定位

鼠标双击：

page.dblclick()

获取元素焦点

#获取元素并聚焦它。如果没有匹配的元素，则方法等待匹配元素出出现在DOM中

page.focus('#su')

鼠标悬停

#就是鼠标放在按钮上，此方法针对那种浮框操作

page.hover('//\*[@id="u1"]/\*[text()="设置"]')

设置复选框取消或选中

#布尔值为True，如果未选中则选中，布尔值为False是，如果未送选中则就显示未选中

page.set\_checked('//\*[text()="全部语言"]'，False)

取消已选中复选框取

#确保元素是复选框或单选框。如果该元素已取消选中，则此方法立即返回

page.uncheck('//\*[text()="仅简体中文"]')

获取元素属性值

#返回元素属性值

page.get\_attribute('#kw'，'name')

获取内部文本

page.inner\_text('//\*[@id="s-hotsearch-wrapper"]//\*[@data-index="2"]//\*[@class="title-content-title"]')

获取内部HTML

page.inner\_html('//\*[@id="s-hotsearch-wrapper"]///\*[@data-index="2"]')

获取文本内容

page.text\_content('//\*[@id="s-hotsearch-wrapper"]1//\*[@data-index="2"]')

截图

#baidu.png存放至当前文件夹下的cases文件夹种

page.screenshot(path='./cases/baidu.png')

填写文本并触发键盘事件

#为文本中的每个字符发送一个keydown、keypress/input和keyup事件

page.type("#kw"， "hello")

输入键盘操作

#获取按钮元素，输入键盘操作

page.press('#su'， 'Enter')

设置select下拉选项

#与值匹配的单个选择

page.select\_option(\'select#colors\"， \"blue\")

#与标签匹配的单个选择

page.select\_option(\'select#colors\"， label=\"blue\")

#多项选择

page.select\_option(\'select#colors\"， value=[\"redI\"， \"green\"， \"blue\"])

调度事件

type可传:"click"， "dragstart"

page.dispatch\_event('#su'， 'click')

##### 检查点(断言)

文字内容断言

#获取文本内容，进行断言

content = page.text\_content('[target="\_blank"]:firstc-child')

assert content == “新闻"

内部文字断言

#获取内部文字，进行断言

text = page.inner\_text('[target="\_blank"]:first-child')

assert text== “新闻"

属性断言

#获取属性值，进行断言

attribute = page.get\_attribute('#su'， 'value')

assert attribute== "百度一下"

复选框断言

page.hover('//\*[@id="u1"]/\*[text()="设置"]')

page.click('//\*[@id="s-user-setting-menu"]//\*[teext()="搜索设置"】')

#复选框状态，进行断言

checked=page.is\_checked('//\*[text()="全部语言"]')

assert checked

js表达式断言

#JS表达式，进行断言

js\_content = page.locator('[data-index="4"]>a>[clas:s="title-content-title"]').text\_content(

assert js\_content=="长津湖超战狼2成中国影史票房冠军"

内部HTML断言

#内部HTML，进行断言

html = page.inner\_html('[class="hot-title"]')

assert"百度热搜" in html

元素可见断言

#元素可见性，进行断言

visible = page.is\_visible('#su')

assert visible

启动状态断言

#启用状态(元素存在可点击)，进行断言

enabled = page.is\_enabled('#su')

assert enabled

直接对比断言

assert page.title()=="百度一下，你就知道"

**playwright还提供了自定义断言，这一块我还没有实操过，有兴趣可的可继续研究下：**

# 断言本地存储值

user\_id = page.evaluate("() => window.localStorage.getItem('user\_id')")

assert user\_id

# 断言输入元素的值

value = page.locator('#search').input\_value()

assert value == 'query'

# 断言计算样式

font\_size = page.locator('div').evaluate('el => window.getComputedStyle(el).fontSize')

assert font\_size == '16px'

# 断言列表长度

length = page.locator('li.selected').count()

第41章BasePage+PageObject

我的文件: D:\Users\s-weijh\PycharmProjects\apitest\pythonProject\playwright\playwright0-1

在playwright0-1下新建目录:demo，

将除conftest.ty 和 pytest.ini 以外的文件放入demo

在playwright0-1下新建目录:testcase，用来存放测试用例

在playwright0-1下新建目录:module

在module 下新建python文件:\_\_ini\_\_.py 用来导入依赖

在module 下新建python文件:BasePage.py

### Web定位方式Lighthouse：

Devtools中选择Recorder，随意录制要获取的web页面，找到【点击】后，点击下拉框，找到selectors，点击这个图标，会有4中定位方式，1. css，2. xpath，3. pierce，4. 文本定位

### Pytest-allure的安装

#### Pycharm中安装命令:

pip install allure-pytest

#### Github中下载搜索allure2,点击Releases,选择需要下载的版本进行下载,下载完成后解压缩

#### 还需要安装java环境

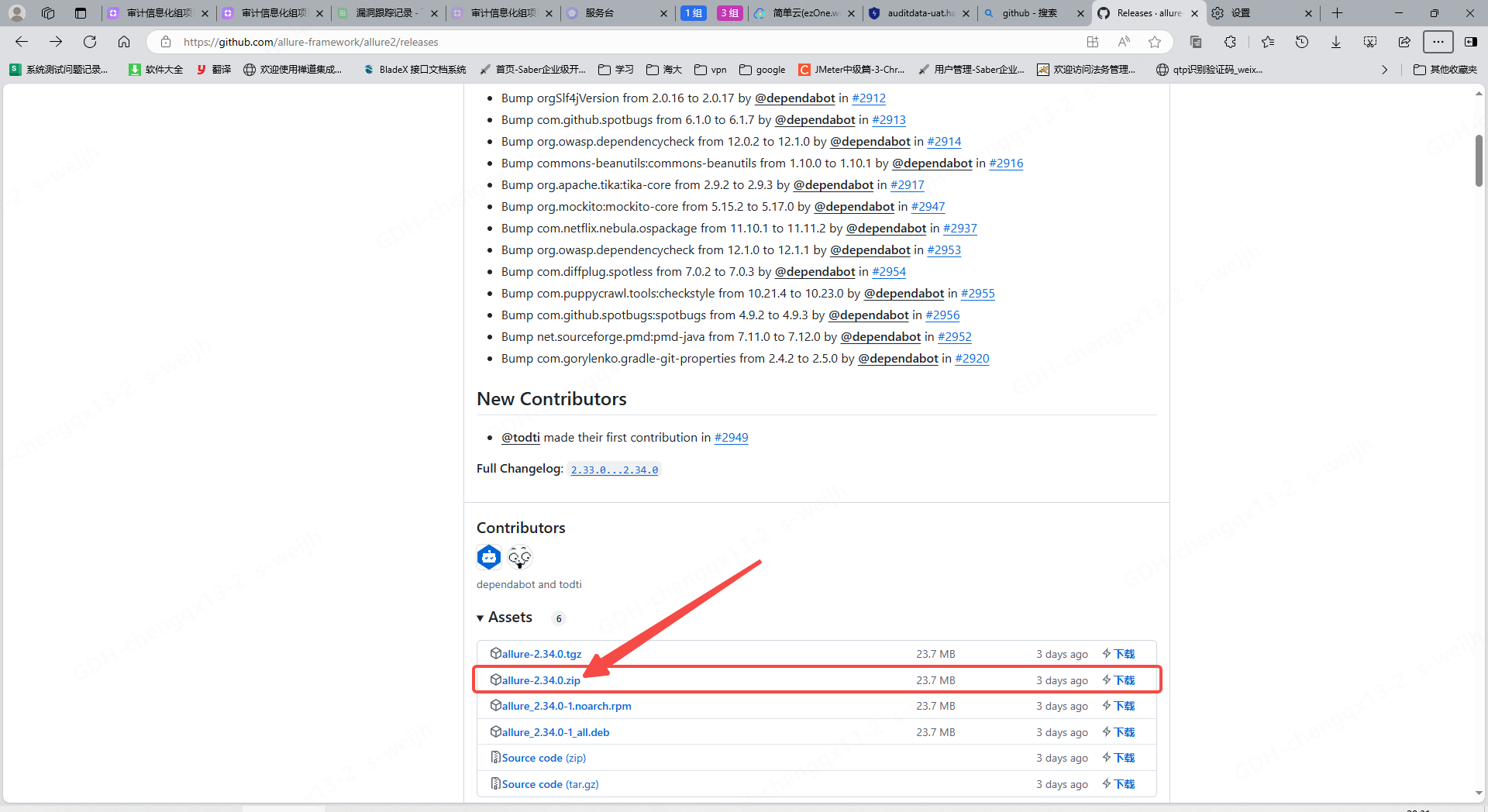
#### Pytest.ini中配置测试结果存放的路径

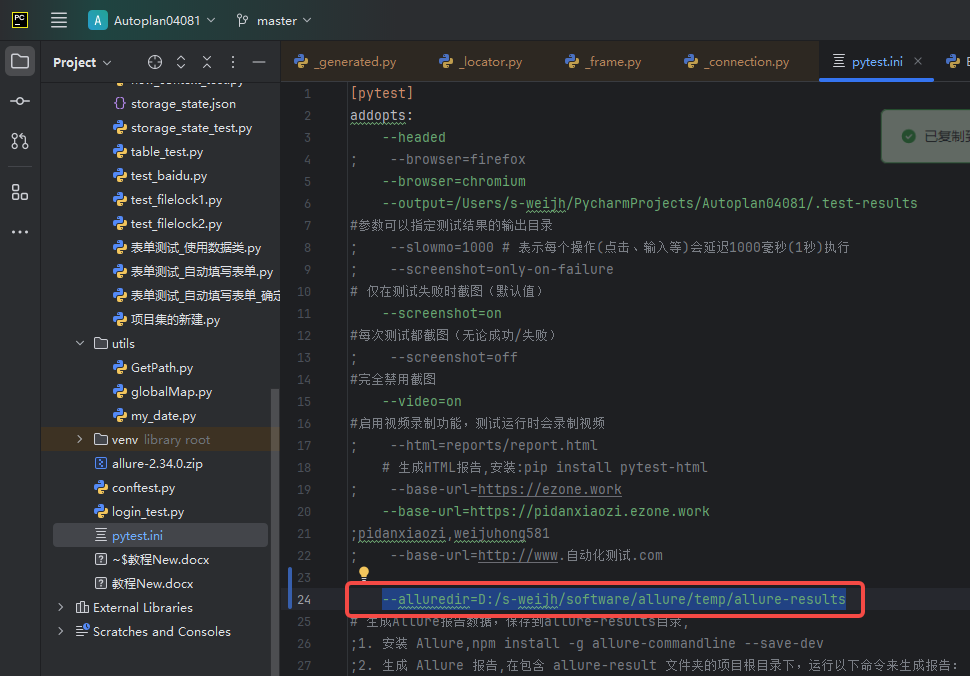
--alluredir=D:/s-weijh/software/allure/temp/allure-results111

#### 查看

执行完成以后,在路径D:/s-weijh/software/allure/temp中输入cmd中执行命令

allure serve allure-{文件名称}





#### 测试用例的配置:test\_case/\_\_init\_\_

|  |
| --- |
| from allure import severity as 用例级别,step as 测试步骤,title as 用例名称,description as 用例描述 from allure\_commons.types import Severity 阻塞=Severity.BLOCKER 严重=Severity.CRITICAL 普通=Severity.NORMAL 不重要=Severity.TRIVIAL 轻微=Severity.MINOR |

#### 日志和报告---allure的失败截图附加,conftest.py中添加方法

|  |
| --- |
| #conftest.py  def pytest\_runtest\_makereport(item):  outcome = yield  report = outcome.get\_result()  if report.failed:  try:  for context in item.funcargs['browser'].contexts:  for page in context.pages:  if page.is\_closed():  continue  bytes\_png=page.screenshot(timeout=5000, full\_page=True)  allure.attach(bytes\_png,f"失败截图---{page.title()}")  except:  ... |

#### 第八十一章用例级重试机制--pytest-rerun的安装和基本使用

需要注意:rerun 3,总共会跑4次11

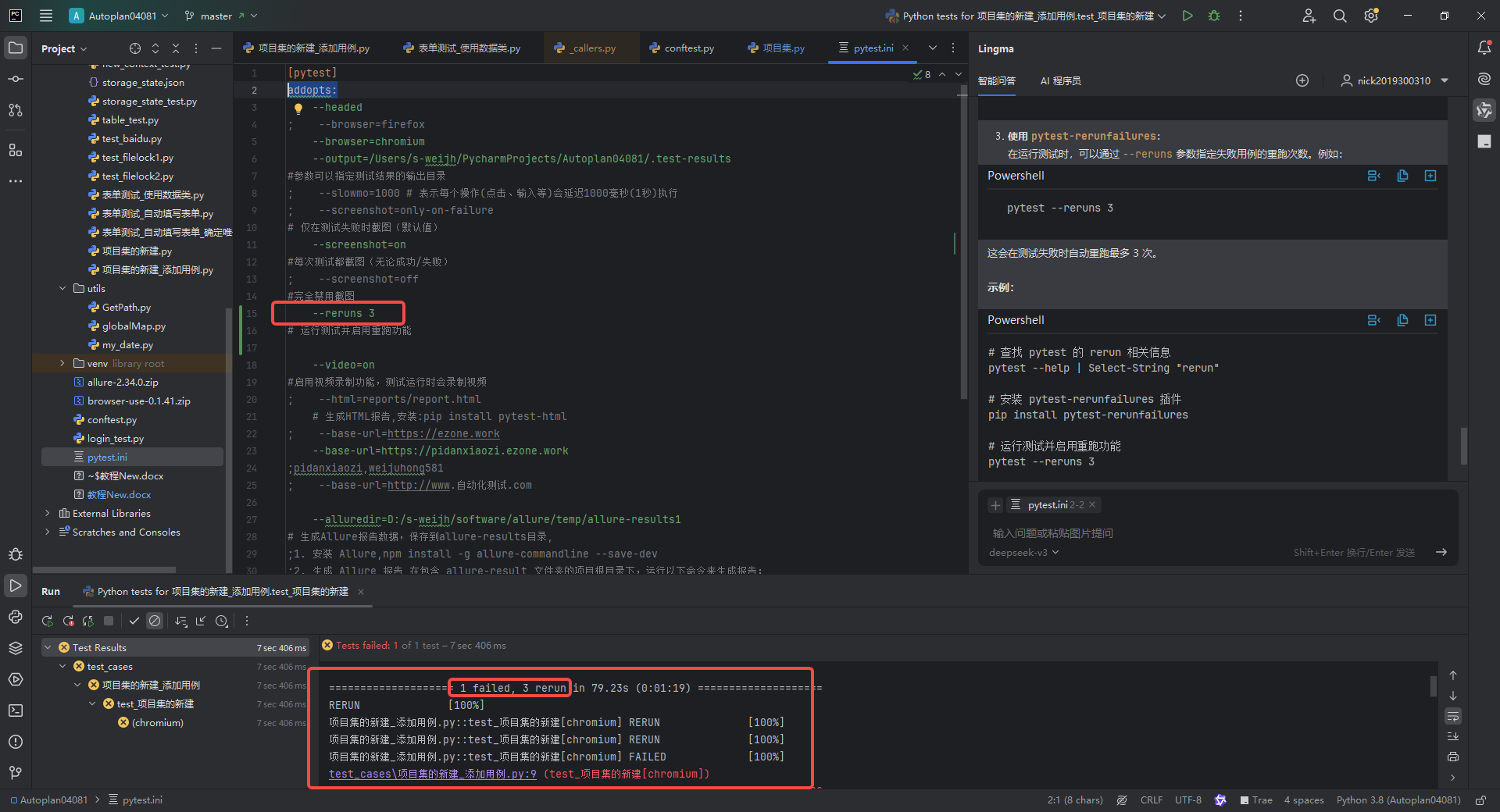
Pycharm终端执行:pip install pytest-rerunfailures

说明查看:pytest --help | Select-String "rerun"

pytest.ini下添加配置

|  |
| --- |
| [pytest]  addopts:  --reruns 3 |





#### Pytest执行次数的控件pytest-repeat

**安装**pytest-repeat,pycharm终端:pip install pytest-repeat

执行的命令:pytest 项目集的新建.py --count=5

pytest test\_file.py::test\_function\_name --count=N

#### 第八十二章用例级重试机制--根据轮次执行不同操作的原理

### 审计综合管理系统

#### 一级-二级-三级

计划管理-年度计划-新建

计划管理-年度计划-查询

计划管理-执行计划-新建

计划管理-执行计划-查询

项目管理-项目申请-新建

项目管理-项目申请-查询

项目管理-审计方案评估及立项-新建

项目管理-审计方案评估及立项-查询

项目管理-现场审计-新建

项目管理-现场审计-查询

项目管理-审计报告-新建

项目管理-审计报告-查询

项目管理-审计发现-新建

项目管理-审计发现-查询

项目管理-联合审计报告-新建

项目管理-联合审计报告-查询

底稿管理--新建底稿复核

底稿管理--查询底稿复核

底稿管理--查询底稿清单

变更管理-计划转让-新建计划转让

变更管理-计划转让-查询计划转让

变更管理-计划变更-新建计划变更

变更管理-计划变更-查询计划变更

变更管理-项目变更-新建项目变更

变更管理-项目变更-查询项目变更

变更管理-审计发现变更-新建审计发现变更

变更管理-审计发现变更-查询审计发现变更

变更管理-工程整改变更-新建工程整改变更

变更管理-工程整改变更-查询工程整改变更

变更管理-管理整改计划变更-新建管理整改计划变更

变更管理-管理整改计划变更-查询管理整改计划变更

变更管理-问题分类变更-新建问题分类变更

变更管理-问题分类变更-查询问题分类变更

整改管理-根因管理-根因主数据

整改管理-根因管理-旧根因主数据

整改管理-根因管理-共性根因描述主数据

整改管理-根因管理-查询共性根因描述处理情况

整改管理-根因管理-根因业务流程主数据

整改管理-根因管理-查询根因业务流程处理情况

整改管理-工程整改-查询工程整改

整改管理-重分类根因-新建重分类根因

整改管理-重分类根因-重分类根因明细

整改管理-重分类根因-查询重分类根因

整改管理-公司整改计划-新建公司整改计划

整改管理-公司整改计划-查询公司整改计划

整改管理-大区整改计划-新建大区共性问题整改计划

整改管理-大区整改计划-查询大区共性问题整改计划

整改管理-整改跟进-新建整改跟进

整改管理-整改跟进-查询整改跟进

整改管理-整改跟进-查询整改跟进发布情况

整改管理-整改跟进复核-新建整改跟进复核

整改管理-整改跟进复核-查询整改跟进复核

内控自查-自查项目-新建自查项目

内控自查-自查项目-查询自查项目

内控自查-自查整改计划-新建自查整改计划

内控自查-自查整改计划-查询自查整改计划

内控自查-自查整改落地反馈-内控自查整改落地反馈

内控自查-自查整改落地反馈-查询自查整改落地反馈

舞弊管理-线索登记-新建线索登记

舞弊管理-线索登记-查询线索登记

舞弊管理-查实结果登记-新建查实结果登记

舞弊管理-查实结果登记-查询查实结果登记

汇报管理-工作周报-新建工作周报

汇报管理-工作周报-查询工作周报

项目管理报表--敏捷项目调查跟踪表

项目管理报表--项目进度统计

项目管理报表--已完成项目统计

项目管理报表--项目进度(项目阶段统计)

项目管理报表--项目进度(人力资源统计)

项目管理报表--前期重点项目情况

项目管理报表--计划实际对比表

项目管理报表--近阶段报告发出情况

项目管理报表--近阶段评议情况

项目管理报表--本周发出报告及评议情况

整改管理报表--审计报告流程进度

整改管理报表--统计问题根因共性识别

整改管理报表--公司整改台账

整改管理报表--大区共性整改台账

整改管理报表--分子公司层面整改概况

整改管理报表--未识别共性问题整改概况

整改管理报表--共性问题整改概况

舞弊管理报表--线索台账

舞弊管理报表--违纪人员台账

审计工具-历史项目中标价格-新建历史项目中标价格

审计工具-历史项目中标价格-查询历史项目中标价格

审计工具-签证-合同价对比-新建签证-合同价对比

审计工具-签证-合同价对比-查询签证-合同价对比

审计工具-钢筋混凝土造价指标-新建钢筋混凝土造价指标

审计工具-钢筋混凝土造价指标-查询钢筋混凝土造价指标

审计工具-专业项目-新建专业项目

审计工具-专业项目-查询专业项目

审计工具-历史造价指标-新建历史造价指标

审计工具-历史造价指标-查询历史造价指标

审计工具-工程项目招标单价分析测算-新建工程项目招标单价分析测算

审计工具-工程项目招标单价分析测算-查询工程项目招标单价

系统管理-邮件管理-新建邮件信息

系统管理-邮件管理-查询邮件信息

系统管理-会议管理-新建会议信息

系统管理-会议管理-查询会议信息

系统管理-用户管理-组别信息

系统管理-用户管理-部门信息

系统管理-用户管理-用户信息

系统管理-用户管理-角色信息

系统管理-用户管理-职类信息

系统管理-用户管理-能力系数维护表

系统管理-主数据管理-工程审计风险点

系统管理-主数据管理-供应商维护信息

系统管理-主数据管理-风险等级扣分

系统管理-主数据管理-数据字典

系统管理-主数据管理-被审计单位

系统管理-主数据管理-审计各业务循环

系统管理-主数据管理-工程审计问题分类

系统管理-主数据管理-整改跟进运行期

系统管理-权限管理-新建数字化权限申请

系统管理-权限管理-查询数字化权限申请

系统管理-需求管理-查询需求申请

系统管理-需求管理-新建需求申请

流程管理--我的待办

流程管理--经办跟踪

流程管理--我的待阅

流程管理--我的已阅

流程管理--仪表分析

#### 区分项目流程

特殊项目 不创建【审计方案评估及立项】和【现场审计】

是和否特殊项目+工程审计，无【意见反馈】

非特殊项目【审计报告】创建流程：

管理审计

项目申请选，【非特殊，管理审计】新建项目结束来到》审计方案评估及立项-新建审计方案评估及立项(需要注意的是: 预览解析内容必须存在，不然不会在现场审计出现)，流程审核结束自动流转到【查询现场审计】》

流程到【现场审核】时，提交提示请填写【离场】》

【汇报管理-工作周报-新建工作周报】，关联该项目全部填写提交》

查询工作周报，选择刚新建的周报，点击调试》

调试完成返回【现场审计】流程，完成离场填写后，提交即可

工程审计

项目申请选，【非特殊，工程审计】新建项目结束来到》

审计方案评估及立项-新建审计方案评估及立项，流程审核结束自动流转到【查询现场审计】》

【现场审计】完成流程》

审计报告即可宣选择该项目（注意：如果关联该项目需要先将底稿管理中的查询底稿复核通过）

#### 日常工作

##### 需求评审

##### 测试计划

##### 测试方案

##### 测试用例编写

##### 测试数据创建

##### 任务进度汇报

##### 项目进度汇总

##### 版本跟新汇报

##### 工单系统

运维问题解答

运维问题排查

运维问题维护

Ideapycharm：playwright+pytest

Postman

###### 部署工具：zadig

运行状态和项目环境

原型图figma

服务日志列表SOA

### 自动化实施方案

#### UI自动化实施方案

##### 主要特性

1. 效率：

* 并发:主要使用：pytest-xdist
* 执行端：单机-多进程-一线程跑一进程
* 避免强制等待：影响效率
* 预制数据和数据准备方式：UI和接口的方式创建数据

1. 稳定性

* 根据被测对象设计的等待
* 主要是方法中判断请求

1. 重试机制

* 用例级：用例中出现报错，根据配置重试次数以及等待时间
* 方法级：方法中出现报错，根据配置重试次数以及等待时间

1. 隔离

* 主要用在多用户并发登录，可通过conftest.py配置，各个用例直接是解耦的
* pytest的默认隔离层级

@pytest.fixture(scope="session",autouse=True)

def test\_init(base\_url):

global\_map=GlobalMap()

global\_map.set("baseurl",base\_url)

env = re.search("(https://)(.\*)(.ezone.work)",base\_url).group(2)

global\_map.set("env",env)

#Fixture作用域控制

@pytest.fixture(scope="function") # 默认function级别

def clean\_data():

return Database.create\_clean\_snapshot() # 每个测试函数获得独立数据快照

##### 功能

###### 登录的持久化

通过登录的持久化storage\_state方法，将登录信息保存到storage\_state.json，再每次登录是优先判断是否有登录的文件，文件是否为我所需要的租户，拿到数据后判断登录是否已经过期，如果未过期则使用，过期则重新登录，从而提高一定的效率

路径：Autoplan04081\test\_cases\storage\_state\_test.py

def test\_storage\_state(browser: Browser) -> None:

# my\_page = PageIns(page)

# my\_page.登录页.登陆方法("pidanxiaozi","weijuhong581")

# my\_page.page.context.storage\_state(path="storage\_state.json")#存储当前用户的登录状态

# my\_page.page.context = my\_page.page.browser.new\_context(storage\_state="storage\_state.json")#使用存储的登录状态

context = browser.new\_context(storage\_state="storage\_state.json") # 使用指定的存储状态创建一个新的浏览器上下文。

page = context.new\_page()

page.goto("https://pidanxiaozi.ezone.work/workbench")

###### User data持久化

减少数据请求量，因为UI方面的请求会占用大量网络IO可能会出现瓶颈，导致报错，影响测试执行结果

可使用命令行参数控制

路径：Autoplan04081\data\_module\auth\_data.py

class MYData:

def \_\_init\_\_(self, local=True, excel=None, yaml=None, feishu=None):

self.local = local

self.excel = excel

self.yaml = yaml

self.feishu = feishu

def userinfo(self):

user = ""

if self.excel:

pass

# todo 把excel转换成字典的方法

elif self.yaml:

pass

# todo 把yaml转换成字典的方法

elif self.feishu:

pass

# todo 把feishu转换成字典的方法

else:

user = { "pidanxiaozi":{

"测试员1":{"username": "pidanxiaozi","password": "weijuhong581"},

"项目经理1":{"username": "winni","password": "playwright001"}}

}

return user

###### 多context的使用

不同用户同时执行操作，且存在用户交互的情况

###### 同跳转结合

登录后页面直达，获取配置文件中base\_url采用拼接当前模块类的url，实现页面直达，减少调试时间和资源的小号

路径：Autoplan04081\test\_cases\new\_context\_test.py

def test\_new\_context(new\_context) -> None:

# 使用new\_context() 方法创建新的上下文

context1 :BrowserContext = new\_context(storage\_state="storage\_state.json")

context1.new\_page().goto("/workbench/myapproval")

context2: BrowserContext = new\_context()

context2.new\_page().goto("/workbench/myapproval")

context1

@pytest.mark.browser\_context\_args(storage\_state="storage\_state.json")

def test\_new\_context(page:Page):

page.goto("/workbench/myapproval")

###### 数据类的使用

数据的转换与维护：

标准化

数据处理更灵活，通过配置的方式

如用户数据类

路径：auth\_data.py

class MYData:  
  
 def \_\_init\_\_(self, local=True, excel=None, yaml=None, feishu=None):  
 self.local = local  
 self.excel = excel  
 self.yaml = yaml  
 self.feishu = feishu  
  
 def userinfo(self):  
 user = ""  
 if self.excel:  
 pass  
 # todo 把excel转换成字典的方法  
 elif self.yaml:  
 pass  
 # todo 把yaml转换成字典的方法  
 elif self.feishu:  
 pass  
 # todo 把feishu转换成字典的方法  
 else:  
 user = { "pidanxiaozi":{  
 "测试员1":{"username": "pidanxiaozi","password": "weijuhong581"},  
 "项目经理1":{"username": "winni","password": "playwright001"}}  
 }  
  
 return user

##### 维护

###### pom的封装

1. BasePage的封装
2. 用于被页面类继承，pom页面类的基本定义，对应页面名称，对应的url，对应的功能
3. 通用功能封装，如登录功能、按钮点击、搜索的封装、表格的封装、表单的封装
4. BaseLocators的封装

常见元素定位的封装

路径：Autoplan04081\module\BasePage.py

class PageObject:

def \_\_init\_\_(self, page: Page) -> None:

self.page = page

self.url = ""

from module.locators import Locators

self.locators = Locators(self.page)

def navigator(self):

self.page.goto(self.url)

def table(self, 唯一文字, 表格序号=-1):

return Table(self.page, 唯一文字, 表格序号)

def click\_button(self, button\_name, timeout=30\_000):

# self.page.get\_by\_role("button").filter(has\_text=button\_name).click(timeout=timeout)

button\_loc = self.page.locator("button")

for 单字符 in button\_name:

button\_loc = button\_loc.filter(has\_text=单字符)

button\_loc.click(timeout=timeout)

# def search(self, 搜索内容: str, placeholder=None):

# if placeholder:

# self.page.locator(f"//span[@class='ant-input-affix-wrapper']//input[contains(@placeholder,'{placeholder}')]").fill(

# 搜索内容)

# else:

# self.page.locator(".ant-input-affix-wrapper input").fill(搜索内容)

# self.page.wait\_for\_load\_state("networkidle") # 等待网络空闲状态

def 表单\_文本框填写(self, 表单项名称: str, 需要填写的文本: str, 表单最上层定位: Locator = None,

timeout: float = None):

# 等待目标元素加载完成

# self.page.wait\_for\_selector("input,textarea", state="visible")

if 表单最上层定位:

表单最上层定位.locator(self.locators.表单项中包含操作元素的最上级div(表单项名称)).locator(

"input,textarea").locator("visible =true").last.fill(需要填写的文本, timeout=timeout)

else:

self.locators.表单项中包含操作元素的最上级div(表单项名称).locator("input,textarea").locator(

"visible =true").last.fill(需要填写的文本, timeout=timeout)

# def w使用new\_context登录并返回实例化的page(new\_context, 用户别名):

# baseurl = "通过globalmap回去baseurl"

# 被测环境 = "通过对字符串截取或正则拿到环境,比如https://pidanxiaozi.ezone.work"

# 用户名 = "通过用户信息获取用户名"

# 密码 = "通过用户信息获取密码"

# with f"这里使用filelock进行文件锁,锁的文件名为{被测环境}-{用户别名}.lock":

# if f".temp/{被测环境}-{用户别名}.json存在,判断方法os.path.exists":

# context = new\_context(storage\_state=f".temp/{被测环境}-{用户别名}.json")

# page = context.new\_page()

# my\_page = "PageIns实例化page,获得所有页面的控制方法"

# # 需要新增一个首页的封装

# my\_page.首页.navigate()

# 登录成功 = "判断下是成功登录上了,还是登录失败"

# if not 登录成功:

# my\_page.登录页.navigate()

# my\_page.登录页.登录(用户名, 密码)

# f"把storage\_state保存为.temp/{被测环境}-{用户别名}.json"

# else:

# context = new\_context()

# page = context.new\_page

# my\_page = "PageIns实例化page,获得所有页面的控制方法"

# my\_page.登录页.登录(用户名, 密码)

# f"把storage\_state保存为.temp/{被测环境}-{用户别名}.json"

#

# return page

1. 各个页面的封装

一般分为两层：元素定位，方法

路径：Autoplan04081\module\登录页.py

class 登录页\_类(PageObject):

def \_\_init\_\_(self, page):

super().\_\_init\_\_(page)

self.url = "/signin"

self.用户名输入框 = self.page.get\_by\_placeholder("用户名/邮箱/手机号")

self.密码输入框 = self.page.get\_by\_placeholder("密码")

# self.登录按钮 =self.page.get\_by\_role("button", name="登录")

self.通知铃铛 = self.page.get\_by\_text("P", exact=True).first

def 登陆方法(self,用户名输入框,密码输入框) -> None:

self.navigator()

self.用户名输入框.fill(用户名输入框)

self.密码输入框.fill(密码输入框)

self.click\_button("登录")

expect(self.通知铃铛).to\_be\_visible()

###### 中文标识符

引入中为你命名插件：ChinesePinyin-CodeCompletion Helper

主要涉及名称的地方为：模块（项目管理），类（项目管理\_类），方法（test\_获取项目名称），变量（self.创建时间）等可汉化对象

###### 日志和报告

1. 报告使用Allure
2. 截图，录像和trace，涉及log策略，失败重试次数，首次执行失败用例进行截图，第二次执行失败用例会被录像记录，第三次会使用trace进行记录，避免每次使用trace占用服务内存过多
3. Console日志下数据，调试使用，或服务中jenkins运行下的报错日志

###### 配置

配置文件的形式pytest.ini

[pytest]

addopts:

--headed#有头模式运行

--browser=firefox#指定浏览器driver

--output=/Users/s-weijh/PycharmProjects/Autoplan04081/.test-results#参数可以指定测试结果的输出目录

; --slowmo=1000 # 表示每个操作(点击、输入等)会延迟1000毫秒(1秒)执行

; --screenshot=only-on-failure# 仅在测试失败时截图（默认值）

--screenshot=on#每次测试都截图（无论成功/失败）

; --screenshot=off#完全禁用截图

--video=on#启用视频录制功能，测试运行时会录制视频

; --html=reports/report.html # 生成HTML报告,安装:pip install pytest-html

; --base-url=https://ezone.work

--base-url=https://pidanxiaozi.ezone.work

###### 持续集成

主要使用jenkins

#### 公共功能的封装

##### 首页封装

class 登录页\_类(PageObject):

def \_\_init\_\_(self, page):

super().\_\_init\_\_(page)

self.url = "/signin"

self.用户名输入框 = self.page.get\_by\_placeholder("用户名/邮箱/手机号")

self.密码输入框 = self.page.get\_by\_placeholder("密码")

# self.登录按钮 =self.page.get\_by\_role("button", name="登录")

self.通知铃铛 = self.page.get\_by\_text("P", exact=True).first

def 登陆方法(self,用户名输入框,密码输入框) -> None:

self.navigator()

self.用户名输入框.fill(用户名输入框)

self.密码输入框.fill(密码输入框)

self.click\_button("登录")

expect(self.通知铃铛).to\_be\_visible()

##### 登录持久化与登录页的封装

登录通过storage\_state方法进行持久化操作

# def 使用new\_context登录并返回实例化的page(new\_context, 用户别名):

# baseurl = "通过globalmap回去baseurl"

# 被测环境 = "通过对字符串截取或正则拿到环境,比如https://pidanxiaozi.ezone.work"

# 用户名 = "通过用户信息获取用户名"

# 密码 = "通过用户信息获取密码"

# with f"这里使用filelock进行文件锁,锁的文件名为{被测环境}-{用户别名}.lock":

# if f".temp/{被测环境}-{用户别名}.json存在,判断方法os.path.exists":

# context = new\_context(storage\_state=f".temp/{被测环境}-{用户别名}.json")

# page = context.new\_page()

# my\_page = "PageIns实例化page,获得所有页面的控制方法"

# my\_page.首页.navigate()

# 登录成功 = "判断下是成功登录上了,还是登录失败"

# if not 登录成功:

# my\_page.登录页.navigate()

# my\_page.登录页.登录(用户名, 密码)

# f"把storage\_state保存为.temp/{被测环境}-{用户别名}.json"

# else:

# context = new\_context()

# page = context.new\_page

# my\_page = "PageIns实例化page,获得所有页面的控制方法"

# my\_page.登录页.登录(用户名, 密码)

# f"把storage\_state保存为.temp/{被测环境}-{用户别名}.json"

#

# return page

登录页\_类

class 登录页\_类(PageObject):

def \_\_init\_\_(self, page):

super().\_\_init\_\_(page)

self.url = "/signin"

self.用户名输入框 = self.page.get\_by\_placeholder("用户名/邮箱/手机号")

self.密码输入框 = self.page.get\_by\_placeholder("密码")

# self.登录按钮 =self.page.get\_by\_role("button", name="登录")

self.通知铃铛 = self.page.get\_by\_text("P", exact=True).first

def 登陆方法(self,用户名输入框,密码输入框) -> None:

self.navigator()

self.用户名输入框.fill(用户名输入框)

self.密码输入框.fill(密码输入框)

self.click\_button("登录")

expect(self.通知铃铛).to\_be\_visible()

##### 表格功能封装

class Table:

def \_\_init\_\_(self, page: Page, 唯一文字: str, 表格序号: int = -1):

self.page = page

self.page.wait\_for\_load\_state("networkidle") # 等待网络空闲状态

# 定位Ant Design表格容器

# 1. 先通过.ant-table-wrapper类名定位所有表格容器

# 2. 再通过filter筛选包含指定文本(唯一文字)的表格

# 3. 最后通过nth选择指定序号(表格序号)的表格，-1表示最后一个

self.table\_div = self.page.locator(".ant-table-wrapper").filter(has\_text=唯一文字).nth(表格序号)

self.table\_header\_tr = self.table\_div.locator("//thead/tr")

# 获取表头文字定位

def get\_header\_index(self, 表头文字: str) -> int:

return self.table\_header\_tr.locator("th").all\_text\_contents().index(表头文字)

# 获取行文字定位

def get\_row\_locator(self, 行元素定位: Locator) -> Locator:

return self.table\_div.locator("tr").filter(has=行元素定位)

def get\_cell\_locator(self, 表头文字or列序号: str or int, 行元素定位or行序号or行文字:

Locator or int or str) -> Locator:

if isinstance(表头文字or列序号, str):

列序号 = self.get\_header\_index(表头文字or列序号)

else:

列序号 = 表头文字or列序号

if isinstance(行元素定位or行序号or行文字, Locator):

行定位 = self.get\_row\_locator(行元素定位or行序号or行文字)

elif isinstance(行元素定位or行序号or行文字, str):

行定位 = self.table\_div.locator("tr").filter(has\_text=行元素定位or行序号or行文字)

elif isinstance(行元素定位or行序号or行文字, int):

行定位 = self.table\_div.locator("tbody").locator("//tr[not(@aria-hiden='true')]").nth(行元素定位or行序号or行文字)

return 行定位.locator("td").nth(列序号)

# 使用空获取随机行的数据,使用行的index获取指定行的数据,使用定位器获取指定行的数据

def get\_row\_dict(self,行元素定位or行序号: Locator or int = "random"):

if isinstance(行元素定位or行序号, int):

tr = self.table\_div.locator("tbody").locator("tr").locator("visible ='true'").nth(行元素定位or行序号)

elif isinstance(行元素定位or行序号, Locator):

tr = self.table\_div.locator("tr").filter(has=行元素定位or行序号)

else:

all\_tr = self.table\_div.locator("tbody").locator("tr").locator("visible ='true'").all()

tr = random.choice(all\_tr)

td\_text\_list = tr.locator("td").all\_text\_contents()

header\_text\_list = self.table\_header\_tr.locator("th").all\_text\_contents()

row\_dict = dict(zip(header\_text\_list, td\_text\_list))

return row\_dict

# 使用列表名称获取当前列内容

def get\_col\_list(self,表头文字: str) -> list:

index = self.get\_header\_index(表头文字)

all\_list = self.table\_div.locator("tbody").locator("tr").locator("visible ='true'").all()

col\_list = []

for tr in all\_list:

col\_list.append(tr.locator("td").nth(index).text\_content())

return col\_list

##### 表单的封装

###### 封装原因

1. POM封装中有大量的表单填写，如果每个表单都进行封装，比较耗费时间
2. 前端组件如果变化，更好维护，应对变化更加灵活

###### 封装思路

### 封装各个操作方法

1. 封装填写文本框
2. 下拉框单选(多选)
3. radio单选
4. switch开关
5. 时间日期
6. 时间日期区间
7. 等等
8. 综合为一个方法，接收字典解包传入的kwargs
9. 数据转换模型
10. 表单填写提示
11. 数据批量处理
12. 采用python自带的dataclass数据类，做中间层
13. 自动判断操作对象类型
14. 自动判断要操作的表单
15. 填写完成后，返回实际填写的数据

路径：Autoplan04081\data\_module\项目集数据类模块.py

@dataclass

class 项目集数据类\_新建项目集(As\_dict):

项目集名称: str = "自动化创建的项目集\_时间戳"

项目集周期: str = ""

父项目集: str = ""

子项目集: str = ""

路径：Autoplan04081\module\locators.py

class Locators:

def \_\_init\_\_(self, page):

self.page = page

def button\_按钮(self, text, nth=-1) -> Locator:

button = self.page.locator("button")

for word in text:

button = button.filter(has\_text=word)

return button.locator("visible=true").nth(nth)

def below\_元素下方紧邻的元素(self, 要找的元素类型="\*") -> Locator:

return self.page.locator(f"xpath=/following:{要找的元素类型}[position()=1]")

def 表单项中包含操作元素的最上级div(self, 字段名: str) -> Locator:

最终上级元素locator = self.page.locator("label").locator("visible=true").filter(

has=self.page.get\_by\_text(字段名)).locator(self.below\_元素下方紧邻的元素())

return 最终上级元素locator

#### 接口自动化实施方案