



第二章 定位与操作(上海-悠悠)

第二章 定位与操作(上海-悠悠)

2.1 Selector 选择器

- 2.1.1 Selector 选择器
- 2.1.2 CSS 或 XPath 选择器
- 2.1.3 text 文本选择器
- 2.1.4 模糊匹配

2.2 Selector 选择器组合定位

- 2.2.1 实例演示
- 2.2.2 css 与 xpath 组合
- 2.2.3 css 与 text 组合

2.3 官方推荐内置定位器

- 2.3.1 包含的文本 `page.get_by_text()`
- 2.3.2 输入框标签 `page.get_by_label`
- 2.3.3 输入框 `page.get_by_placeholder`
- 2.3.4 title 属性 `page.get_by_title`
- 2.3.5 角色定位 `page.get_by_role`
- 2.3.6 包含的文本 `page.get_by_alt_text`
- 2.3.7 测试 ID `page.get_by_test_id`

2.4 强大的 `get_by_text` 文本选择器

- 2.4.1 适合文本定位的场景
- 2.4.2 模糊匹配和精准匹配
- 2.4.3 xpath 文本定位

2.5 ElementHandle 与 Locator 定位

- 2.5.1 ElementHandle 元素句柄
- 2.5.2 Locator 定位器

2.6 页面元素操作action

- 2.6.1 `fill()` 输入文字
- 2.6.2 Type 输入
- 2.6.3 鼠标点击 `click()`
- 2.6.4 文件上传
- 2.6.5 select 下拉框
- 2.6.6 复选框和单选

- 2.6.7 focus()聚焦给定元素

- 2.6.8 drag_to 拖动

2.7 操作iframe

- 2.7.1 iframe 定位

- 2.7.2 案例演示

- 2.7.3 page.frames 获取页面上所有的iframe

- 2.7.4 frame() 定位方法

2.8 select 选择框

- 2.8.1 select 用法

- 2.8.2 使用示例

2.9 screenshot 截图

- 2.9.1 screenshot 截图

- 2.9.2 截长图

- 2.9.3 捕获图片数据流

- 2.9.4 截取单个元素

2.10 文件上传

- 2.10.1 文件上传

- 2.10.2 使用示例

- 2.10.3 高级操作-事件监听filechooser

2.11 文件下载 download

- 2.11.1 expect_download()

- 2.11.2 download 相关操作

- 2.11.3 使用示例

2.12 监听 download 事件

- 2.12.1 监听下载事件

- 2.12.2 屏蔽下载

2.13 监听dialog处理alert

- 2.13.1 认识alert/confirm/prompt

- 2.13.2 dialog 事件监听

- 2.13.3 dialog 属性和方法

- 2.13.4 prompt提示框

2.14 鼠标悬停 hover

- 2.14.1 鼠标悬停

2.15 定位多个元素

- 2.15.1 click方法

- 2.15.2 first 和 last
- 2.15.3 nth() 根据索引定位
- 2.15.4 count 统计个数
- 2.15.5 all() 定位全部

2.16 locator.filter()过滤定位器

- 2.16.1 使用示例
- 2.16.2 has_text 参数使用
- 2.16.3 按另一个 locator 过滤
- 2.16.4 链式定位器

2.17 无序列表 listitem 定位

- 2.17.1 listitem 水平显示
- 2.17.2 dropdown 下拉显示
- 2.17.3 listitem 定位示例

2.18 scroll 滚动到元素出现的位置

- 2.18.1 click 点击操作
- 2.18.2 滚动到元素出现位置
- 2.18.3 hover 方法

2.19 事件监听与取消

- 2.19.1 等待特定事件
- 2.19.2 添加/删除事件
- 2.19.3 添加一次性事件

2.20 checkbox和radio 相关操作

- 2.20.1 使用场景
- 2.20.2 radio 单选操作
- 2.20.3 checkbox 复选框
- 2.20.4 批量选中checkbox

2.21 css 选择器语法总结

- 2.21.1 css 选择器语法
- 2.21.2 基础语法
- 2.21.3 父子层级定位
- 2.21.4 正则匹配属性

2.22 xpath 语法总结

- 2.22.1 xpath 语法总结
- 2.22.2 基础语法示例
- 2.22.3 层级关系定位

- 2.22.4 text()文本定位
- 2.22.5 index 索引定位
- 2.23 iframe操作元素，监听，执行JS总结
 - 2.23.1 案例页面
 - 2.23.2 iframe 对象的定位
 - 2.23.3 什么是iframe
 - 2.23.4 page.frame_locator(selector) 使用
 - 2.23.5 page.frame(name, url) 的使用
 - 方法一：name 属性可以是iframe 元素的name 或id属性
 - 方法二：url属性定位
 - 2.23.6 iframe 上的动态id属性如何定位
 - 2.23.7 2层iframe 如何操作
 - 2.23.8 page.query_selector(selector).content_frame() 方法
 - 2.23.9 监听 iframe 上的事件
 - 2.23.10 在iframe 上执行JavaScript 代码
- 2.24 svg 元素定位方法总结
 - 2.24.1 svg 元素定位
 - 2.24.2 页面上用多个svg元素
 - 2.24.3 定位svg 上的子元素
 - 2.24.4 使用示例
- 2.25 svg 元素拖拽
 - 2.25.1 拖拽 svg 元素
 - 2.25.2 playwright 拖拽 svg 元素
- 2.26 日历控件操作
 - 2.26.1 日历控件
 - 2.26.2 readonly 的日历控件
- 2.27 在打开的多个标签页窗口灵活切换
 - 2.27.1 使用场景
 - 2.27.2 通过title 判断页面切换
- 2.28 table表格定位与数据获取
 - 2.28.1 table 表格场景
 - 2.28.2 xpath 定位table 表格
 - 2.28.3 获取当前表格总数
 - 2.28.4 获取表格数据

- 2.28.5 判断新增的模块名称在table表格中

- 网易云视频课程

2.1 Selector 选择器

Selector 选择器，也就是通常说的元素定位了，页面上点点点的操作，都是基于元素定位，所以这块是重点需要学的核心内容。

总的来说分四种方式：

- 1.css 选择器
- 2.xpath 语法定位
- 3.text 文本选择器
- 4.组合定位

2.1.1 Selector 选择器

说到元素定位，大家肯定会首先想到 selenium 的八大元素定位，其中xpath 和 css才是最主要的。

playwright 可以通过 CSS selector，XPath selector，HTML 属性（比如 id，data-test-id）或者是 text 文本内容 定位元素。

除了xpath selector外，所有selector默认都是指向shadow DOM，如果要指向常规DOM，可使用*:light。不过通常不需要。

操作元素，可以先定位再操作

```
# 先定位再操作
page.locator('#kw').fill("上海悠悠")
page.locator('#su').click()
```

也可以直接调用fill 和 click 方法，传Selector选择器

```
page.fill('#kw', "上海-悠悠博客")
page.click('#su')
```

一般推荐下面的这种方式

2.1.2 CSS 或 XPath 选择器

可以使用xpath 和 css 元素

```
# CSS and XPath
page.fill('css=#kw', "上海-悠悠博客")
page.click('xpath=//*[@id="su"]')
```

或者可以直接写xpath和css语法，不需要前面的前缀 `css=` 和 `xpath=`，它会自动判断你写的是css还是xpath语法，前提是你语法没有错误。

```
page.fill('#kw', "上海-悠悠博客")
page.click('//*[@id="su"]')
```

XPath 和 CSS 选择器可以绑定到 DOM 结构或实现。

当 DOM 结构发生变化时，这些选择器可能会中断。下面的长 CSS 或 XPath 链是导致测试不稳定的不良做法的示例：

```
page.click("#tsf > div:nth-child(2) > div.A8SBwf > div.RNNXgb > div > div.a4bIc > input")
page.click('//*[@id="tsf"]/div[2]/div[1]/div[1]/div/div[2]/input')
```

2.1.3 text 文本选择器

文本选择器是一个非常实用的定位方式，根据页面上看到的text文本就可以定位了，比如我们经常使用xpath 的文本选择器定位

- 完全匹配文本 `//*[text()="上海悠悠"]`
- 包含某个文本 `//*[contains(text(),"上海悠悠")]`

playwright 封装了text文本定位的方式，也可以支持2种文本定位方式

```
page.click("text=yo yo")
page.click("text='YO YO'")
```

`text=yo yo` 和 `text='YO YO'` 的区别：

- `text=yo yo` 没有加引号（单引号或者双引号），模糊匹配，对大小写不敏感
- `text='YO YO'` 有引号，精确匹配，对大小写敏感

使用示例,比如点击百度上的“新闻” 文本



```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    page = browser.new_page()
    page.goto("https://www.baidu.com")
    print(page.title())
    page.click('text=新闻')
    page.wait_for_timeout(5000)
    browser.close()
```

text 文本除了可以定位 a 标签, 还可以定位 button 按钮

input 标签的button 按钮, 有 value="百度一下" 文本值

```
<input type=button value="百度一下">
```

或者是button 标签的按钮

```
<button>百度一下</button>
```

使用示例:定位百度的搜索按钮

```
元素 控制台 源代码 网络 性能 内存 应用 Lighthouse 记录器 性能数据分析
<input type="hidden" name="tn" value="baiduhome_pg">
<input type="hidden" name="ch" value>
  <span class="btn_wr s_btn_wr bg" id="s_btn_wr">
    <input type="submit" value="百度一下" id="su" class="btn self-btn bg s_btn"> == $0
  </span>
  <span class="tools">
```

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    page = browser.new_page()
    page.goto("https://www.baidu.com")
    print(page.title())
    page.fill("#kw", "上海-悠悠")
    page.click('text=百度一下')
    page.wait_for_timeout(5000)
    browser.close()
```

HTML 属性选择器, 根据html元素的id 定位

```
page.fill("id=kw", "上海-悠悠")
```

2.1.4 模糊匹配

登录按钮的值是 `value="立即登录 > "`, 可以用text文本定位的方式, 模糊匹配到, 这个人性化的设计提高了定位的效率。

```
<input class="btn btn-success btn-block" id="loginBtn" type="submit" value="立即登录 > ">
```

2.2 Selector 选择器组合定位

总的来说分四种方式：

- 1.css 选择器
- 2.xpath 语法定位
- 3.text 文本选择器
- 4.组合定位

组合定位可以使用前面3种定位的任意组合

2.2.1 实例演示

查找目标元素

```

<form id="login-form" action="/api/login" method="POST" role="form" class="form-horizontal fv-form fv-form-bootstrap novalidate">
  <button type="submit" class="fv-hidden-submit" style="display: none; width: 0px; height: 0px;"></button>
  <div class="form-group has-feedback">
    ::before
    <label for="username" class="col-sm-3 col-md-3 col-lg-3 control-label">用&nbsp;  户&nbsp;  名:</label>
    <div class="col-xs-12 col-sm-9 col-md-9 col-lg-9">
      <input type="text" class="form-control" id="username" name="username" placeholder="请输入用户名" data-fv-field="username">
      <i class="form-control-feedback" data-fv-icon-for="username" style="display: none;"></i>
      <small class="help-block" data-fv-validator="notEmpty" data-fv-for="username" data-fv-result="NOT_VALIDATED" style="display: none;">不能为空
      </small>
      <small class="help-block" data-fv-validator="stringLength" data-fv-for="username" data-fv-result="NOT_VALIDATED" style="display: none;">用户名1-30位字符</small>
      <small class="help-block" data-fv-validator="regexp" data-fv-for="username" data-fv-result="NOT_VALIDATED" style="display: none;">用户名不能有特殊字符,请用中英文数字_</small>
    </div>
    ::after
  </div>
  <div class="form-group has-feedback">
  <div class="col-xs-12 col-sm-9 col-md-9 col-lg-9 col-md-offset-3 col-lg-offset-3">
  <div class="clearfix">
  <!-- 清除浮动 -->
  <div class="text-right">
  <div class="input-group-lg">
    <input class="btn btn-success btn-block" id="loginBtn" type="submit" value="立即登录" >
  </div>
</form>

```

不同的selector可组合使用，用 >> 连接

```
# id 属性+ css
page.fill('form >> [name="username"]', "yoyo")
page.fill('form >> #password', "aa123456")
page.click("text=立即登录")
```

form >> [name="username"] 定位方式等价于

```
# page.fill('form >> [name="username"]', "yoyo")
page.locator("form").locator('[name="username"]').fill("yoyo")
```

相当于是根据父元素找到子孙元素了

2.2.2 css 与 xpath 组合

组合定位的时候，css和xpath两种语法可以结合使用

```
# 组合定位 父元素--->子孙元素
page.locator("#login-form >> //*[@id='loginBtn']").click()
```

等价于

```
page.locator("#login-form").locator("//*[@id='loginBtn']").click()
```

2.2.3 css 与 text 组合

```
# 组合定位 父元素--->子孙元素
page.locator("#login-form >> text=立即登录").click()
```

等价于

```
page.locator("#login-form").locator("text=立即登录").click()
```

xpath , css, text 三种定位方式可以任意组合

2.3 官方推荐内置定位器

这些是 playwright 推荐的内置定位器。

- `page.get_by_text()` 通过文本内容定位。
- `page.get_by_label()` 通过关联标签的文本定位表单控件。
- `page.get_by_placeholder()` 按占位符定位输入。
- `page.get_by_title()` 通过标题属性定位元素。
- `page.get_by_role()` 通过显式和隐式可访问性属性进行定位。
- `page.get_by_alt_text()` 通过替代文本定位元素，通常是图像。
- `page.get_by_test_id()` 根据 `data-testid` 属性定位元素（可以配置其他属性）。

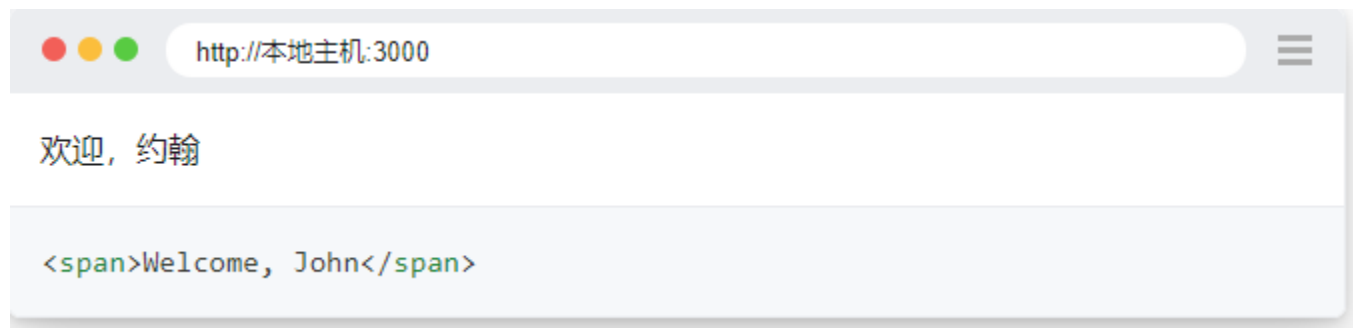
使用示例

```
page.get_by_label("User Name").fill("John")
page.get_by_label("Password").fill("secret-password")
page.get_by_role("button", name="Sign in").click()
expect(page.get_by_text("Welcome, John!")).to_be_visible()
```

2.3.1 包含的文本 page.get_by_text()

根据元素包含的文本查找元素。使用page.get_by_text()时，您可以通过子字符串、精确字符串或正则表达式进行匹配。

例如，考虑以下 DOM 结构。



```
expect(page.get_by_text("Welcome, John")).to_be_visible()
```

设置精确匹配：

```
expect(page.get_by_text("Welcome, John", exact=True)).to_be_visible()
```

与正则表达式匹配：

```
expect(page
  .get_by_text(re.compile("welcome, john", re.IGNORECASE)))
  .to_be_visible()
```

笔记

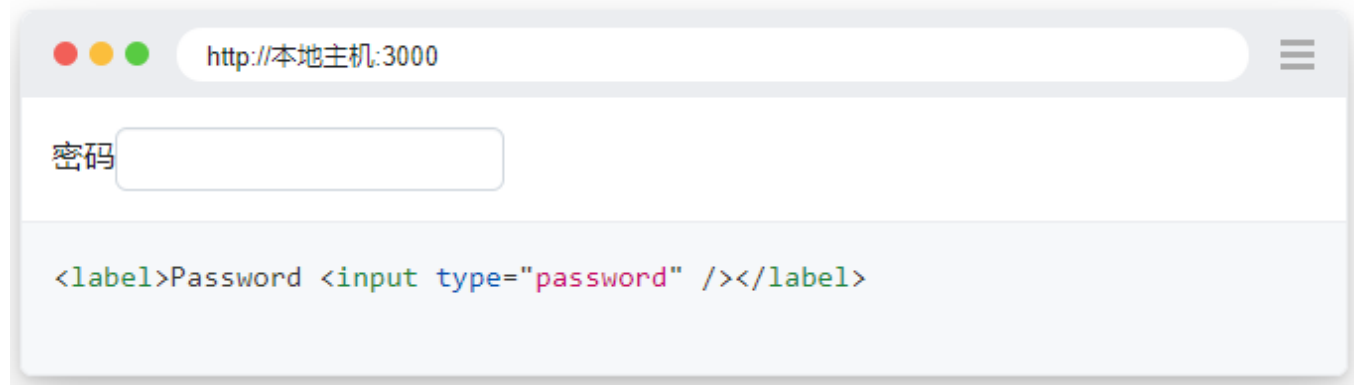
按文本匹配总是规范化空白，即使是完全匹配。例如，它将多个空格变成一个，将换行符变成空格并忽略前导和尾随空格

我们建议使用文本定位器来查找非交互式元素，如div, span, p 等。对于交互式元素，如请button, a, input, 使用角色定位器。

2.3.2 输入框标签 `page.get_by_label`

大多数表单控件通常都有专用标签，可以方便地用于与表单交互。在这种情况下，您可以使用 `page.get_by_label()` 通过其关联标签定位控件。

例如，考虑以下 DOM 结构。



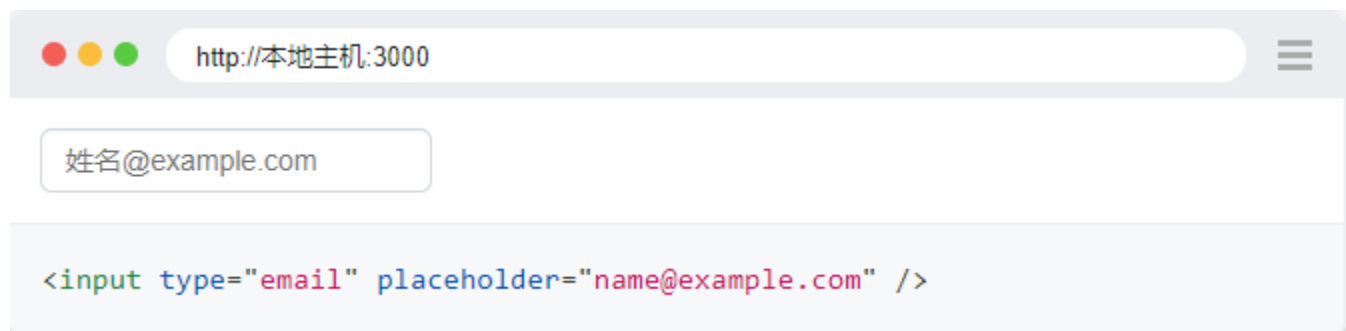
您可以在通过标签文本定位后填写输入：

```
page.get_by_label("Password").fill("secret")
```

输入可能有一个占位符属性来提示用户应该输入什么值。您可以使用 `page.get_by_placeholder()` 找到这样的输入。

2.3.3 输入框 `page.get_by_placeholder`

例如，考虑以下 DOM 结构。



您可以在通过占位符文本定位后填充输入：

```
page.get_by_placeholder("name@example.com").fill("playwright@microsoft.com")
```

2.3.4 title 属性 page.get_by_title

使用page.get_by_title()找到具有匹配 title 属性的元素。

例如，考虑以下 DOM 结构。



您可以在通过标题文本找到它后检查问题数：

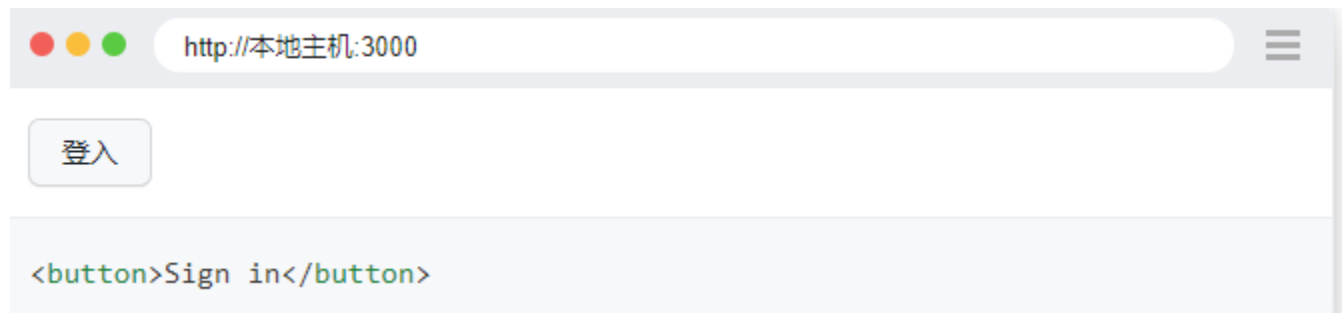
```
expect(page.get_by_title("Issues count")).to_have_text("25 issues")
```

当您的元素具有该title属性时使用此定位器。

2.3.5 角色定位 page.get_by_role

Playwright 带有多个内置定位器。为了使测试具有弹性，我们建议优先考虑面向用户的属性和显式契约，例如page.get_by_role()。

例如，考虑以下 DOM 结构。



button通过名称为“登录”的角色定位元素。

```
page.get_by_role("button", name="Sign in").click()
```

每次将定位器用于操作时，都会在页面中找到一个最新的 DOM 元素。在下面的代码片段中，底层 DOM 元素将被定位两次，一次在每个动作之前。这意味着如果 DOM 由于重新渲染而在调用

之间发生变化，则将使用与定位器对应的新元素。

```
locator = page.get_by_role("button", name="Sign in")

locator.hover()
locator.click()
```

请注意，所有创建定位器的方法（例如`page.get_by_label()`）也可用于`Locator`和`FrameLocator`类，因此您可以将它们链接起来并迭代地缩小定位器的范围。

```
locator = page.frame_locator("my-frame").get_by_role("button", name="Sign in")

locator.click()
```

`page.get_by_role()`定位器反映了用户和辅助技术如何感知页面，例如某个元素是按钮还是复选框。按角色定位时，通常还应传递可访问的名称，以便定位器准确定位元素。

例如，考虑以下 DOM 结构。



```
<h3>Sign up</h3>
<label>
  <input type="checkbox" /> Subscribe
</label>
<br/>
<button>Submit</button>
```

您可以通过其隐含角色定位每个元素：

```
expect(page.get_by_role("heading", name="Sign up")).to_be_visible()

page.get_by_role("checkbox", name="Subscribe").check()

page.get_by_role("button", name=re.compile("submit", re.IGNORECASE)).click()
```

角色定位器包括按钮、复选框、标题、链接、列表、表格等，并遵循ARIA 角色、ARIA 属性和可访问名称的 W3C 规范。

请注意，许多 `html` 元素 `<button>` 都有一个隐式定义的角色，该角色可被角色定位器识别。

请注意，角色定位器不会取代可访问性审核和一致性测试，而是提供有关 ARIA 指南的早期反馈。

何时使用角色定位器

我们建议优先使用角色定位器来定位元素，因为这是最接近用户和辅助技术感知页面的方式。

2.3.6 包含的文本 `page.get_by_alt_text`

所有图像都应该有一个 `alt` 描述图像的属性。您可以使用 `page.get_by_alt_text()` 根据替代文本定位图像。

例如，考虑以下 DOM 结构。



可以在通过文本选项找到图像后单击它：

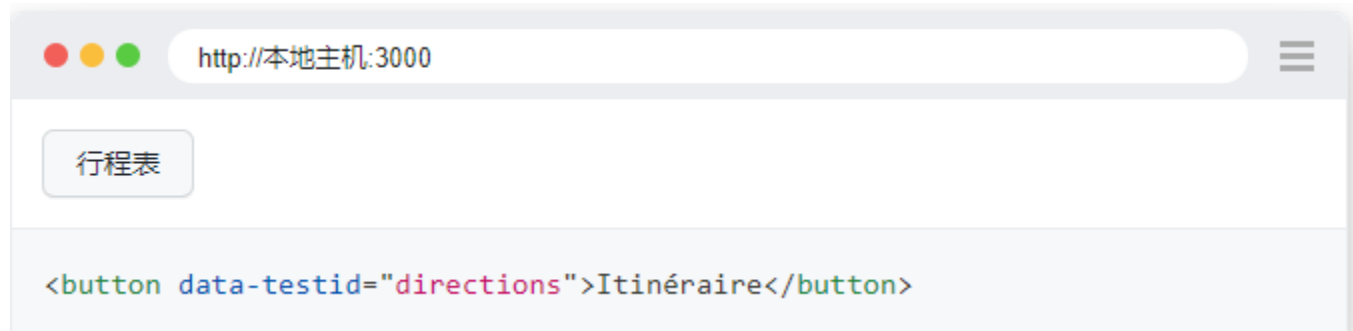
```
page.get_by_alt_text("playwright logo").click()
```

当您的元素支持替代文本（例如 `img` 和 `area` 元素）时使用此定位器。

2.3.7 测试 ID page.get_by_test_id

通过测试 ID 进行测试是最具弹性的测试方式，因为即使您的文本或属性角色发生变化，测试仍会通过。QA 和开发人员应该定义明确的测试 ID 并使用 `page.get_by_test_id()` 查询它们。但是，通过测试 ID 进行的测试不是面向用户的。如果角色或文本值对您很重要，那么请考虑使用面向用户的定位器，例如角色定位器和文本定位器。

例如，考虑以下 DOM 结构。



您可以通过它的测试 ID 找到该元素：

```
page.get_by_test_id("directions").click()
```

设置自定义测试 id

默认情况下，`page.get_by_test_id()` 将根据 `data-testid` 属性定位元素，但您可以在测试配置中或通过调用 `selectors.set_test_id_attribute()` 对其进行配置。

设置测试 ID 以使用自定义数据属性进行测试。

```
playwright.selectors.set_test_id_attribute("data-pw")
```

在您的 html 中，您现在可以使用 `data-pw` test id 而不是 default `data-testid`。



然后像往常一样定位元素：


```
page.get_by_test_id("directions").click()
```

Playwright 提供了一些非常实用的定位方式，也是大家在工作中经常会用到的，比如text文本定位，label 和 placeholder 都是非常实用的定位方式。

2.4 强大的 get_by_text 文本选择器

文本选择器有3种书写方式

- selector 选择器方式 `page.click("text=yo yo")`
- 内置定位器 `page.get_by_text()`
- 原生xpath 文本定位 `//*[@text()='文本']`

2.4.1 适合文本定位的场景

1.a标签

```
<a id="blog" class="menu" href="https://www.cnblogs.com/">我的博客</a>
```

text 文本除了可以定位 a 标签，还可以定位 button 按钮

2.input 标签的button 按钮，有 `value="百度一下"` 文本值

```
<input type="button" value="百度一下">
```

或者是button 标签的按钮

```
<button>百度一下</button>
```

3.纯文本也可以定位

```
<h3>这是文本</h3>
<p>这也是文本</p>
```

2.4.2 模糊匹配和精准匹配

selector 选择器方式 `page.click("text=博客")` 默认是模糊匹配的
以下2个元素都会匹配到

```
<a id="blog1" class="menu" href="https://www.cnblogs.com/">我的博客</a>
<a id="blog2" class="menu" href="https://www.cnblogs.com/">博客</a>
```

精准匹配需使用单引号 `page.click("text='博客'")`

`get_by_text()` 内置定位器

内置定位器 `page.get_by_text('博客')` 默认也是模糊匹配的

精准匹配 `page.get_by_text("博客", exact=True)`

还可以支持正则匹配

```
page.get_by_text(re.compile("welcome, john", re.IGNORECASE))
```

2.4.3 xpath 文本定位

等价于xpath的文本定位方式

完全匹配文本 `//*[text()="上海悠悠"]`

包含某个文本 `//*[contains(text(),"上海悠悠")]`

2.5 ElementHandle 与 Locator 定位

ElementHandle 表示页内 DOM 元素。ElementHandles 可以使用`page.query_selector()`方法创建。

如果你能理解ElementHandle 和 Locator 定位机制，那也就明白了selenium的定位方式 和 playwright 定位方式的本质区别。

2.5.1 ElementHandle 元素句柄

ElementHandle 表示页内 DOM 元素，ElementHandle 实例可以用作`page.eval_on_selector()`和`page.evaluate()`方法中的参数。

可以使用page.query_selector()方法创建。

```
href_element = page.query_selector("a")  
href_element.click()
```

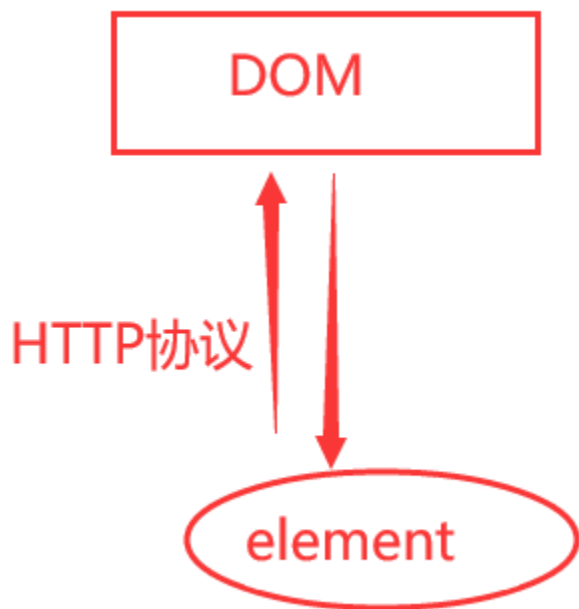
在下面的示例中，句柄指向页面上的特定 DOM 元素。如果那个元素改变了文本或者被 React 用来渲染一个完全不同的组件，句柄仍然指向那个 DOM 元素。这可能会导致意外行为。

```
handle = page.query_selector("text=Submit")  
handle.hover()  
handle.click()
```

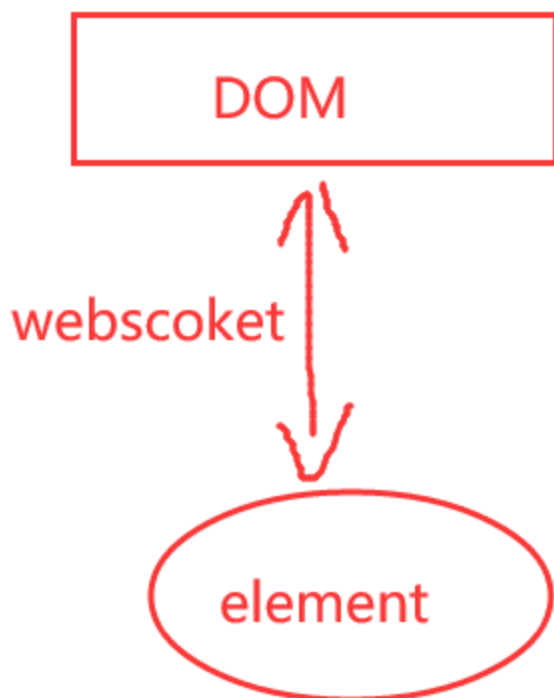
学到这里大家也就明白了，为什么用selenium去循环操作页面元素的时候，只有第一次能操作，当页面刷新时，你定位的对象是上个页面的DOM元素对象。

所以虽然你定位对了，但是句柄发生了改变，也就无法找到指定的元素了。

如果还是不能理解，再给大家讲下 selenium 采用的是 http 协议，获取的元素句柄是固定的，不能实时去获取页面上的元素



playwright 采用的是webscoket 协议，可以实时去获取页面元素，当DOM结构有更新的时候，也能重新获取到



所以不鼓励使用 `ElementHandle`，而是使用 `Locator` 对象和网络优先断言

2.5.2 Locator 定位器

使用定位器，每次 `element` 使用时，最新的 DOM 元素都会使用选择器定位在页面中。所以在下面的代码片段中，底层 DOM 元素将被定位两次。

```
locator = page.get_by_text("Submit")
locator.hover()
locator.click()
```

定位机制不一样，所以 playwright 定位元素的时候比 selenium 更稳定。

2.6 页面元素操作action

Playwright 可以与 HTML 输入元素交互，例如文本输入、复选框、单选按钮、选择选项、鼠标单击、键入字符、键和快捷方式以及上传文件和焦点元素。

2.6.1 fill() 输入文字

使用 `locator.fill()` 是填写表单字段的最简单方法。它聚焦元素并input使用输入的文本触发事件。它适用于 `<input>`, `<textarea>` 和 `[contenteditable]` 元素。

同步示例

```
# Text 文本框输入
page.get_by_role("textbox").fill("Peter")

# 根据label 定位 Date 日期输入
page.get_by_label("Birth date").fill("2020-02-02")

# Time input
page.get_by_label("Appointment time").fill("13:15")

# Local datetime input
page.get_by_label("Local time").fill("2020-03-02T05:15")
```

异步示例

```
# Text input
await page.get_by_role("textbox").fill("Peter")

# Date input
await page.get_by_label("Birth date").fill("2020-02-02")

# Time input
await page.get_by_label("Appointment time").fill("13:15")

# Local datetime input
await page.get_by_label("Local time").fill("2020-03-02T05:15")
```

以下登录框示例, 在不查看元素属性的情况下, 可以直接根据输入框的label 标签定位, 非常的方便



```
# 直接根据label定位
page.get_by_label("用户名:").fill("yoyo")
page.get_by_label("密码:").fill("aa123456")
```

2.6.2 Type 输入

一个字符一个字符地输入字段，就好像它是一个使用locator.type()的真实键盘的用户。

```
# Type character by character
page.locator('#area').type('Hello World!')
```

此方法将发出所有必要的键盘事件，所有keydown, keyup, keypress事件就位。您甚至可以指定delay按键之间的可选操作来模拟真实的用户行为。

大多数时候，page.fill()会正常工作。如果页面上有特殊的键盘处理，您只需要键入字符。

2.6.3 鼠标点击 click()

执行简单的人工点击。

```
# Generic click
page.get_by_role("button").click()

# Double click
page.get_by_text("Item").dblclick()

# Right click
page.get_by_text("Item").click(button="right")

# Shift + click
page.get_by_text("Item").click(modifiers=["Shift"])

# Hover over element
page.get_by_text("Item").hover()

# Click the top left corner
page.get_by_text("Item").click(position={ "x": 0, "y": 0})
```

在幕后，这个和其他与指针相关的方法：

- 等待具有给定选择器的元素出现在 DOM 中（**不用自己去写轮询等待了**）
- 等待它显示出来，即不为空，不display:none，不visibility:hidden（**这个太人性化了，不用去判断元素是否隐藏**）
- 等待它停止移动，例如，直到 css 转换完成
- 将元素滚动到视图中（**这个太人性化了，不用自己去滚动了**）
- 等待它在动作点接收指针事件，例如，等待直到元素变得不被其他元素遮挡
- 如果元素在上述任何检查期间分离，则重试

由此可见，`click()` 方法优化了selenium 点击元素的遇到的一些痛点问题，比如元素遮挡，不在当前屏幕，元素未出现在DOM中或隐藏不可见等不可点击的状态。

使用示例

```
page.get_by_label("用户名:").fill("yoyo")
page.get_by_label("密码:").fill("aa123456")
page.locator("text=立即登录").click()
# page.click("text=立即登录")
```

2.6.4 文件上传

您可以使用`locator.set_input_files()`方法选择要上传的输入文件。

它期望第一个参数指向类型为 的输入元素"file"。数组中可以传递多个文件。
如果某些文件路径是相对的，则它们将相对于当前工作目录进行解析。空数组清除所选文件。

```
# Select one file
page.get_by_label("Upload file").set_input_files('myfile.pdf')

# Select multiple files
page.get_by_label("Upload files").set_input_files(['file1.txt', 'file2.txt'])

# Remove all the selected files
page.get_by_label("Upload file").set_input_files([])

# Upload buffer from memory
page.get_by_label("Upload file").set_input_files(
    files=[
        {"name": "test.txt", "mimeType": "text/plain", "buffer": b"this is a test"}
    ],
)
```

如果当前没有输入元素（它是动态创建的），您可以处理page.on("filechooser")事件或在您的操作中使用相应的等待方法：

```
with page.expect_file_chooser() as fc_info:
    page.get_by_label("Upload file").click()
file_chooser = fc_info.value
file_chooser.set_files("myfile.pdf")
```

2.6.5 select 下拉框

`<select>` 使用locator.select_option()选择元素中的一个或多个选项。您可以指定选项value，或label选择。可以选择多个选项。

```
# Single selection matching the value
page.get_by_label('Choose a color').select_option('blue')

# Single selection matching the label
page.get_by_label('Choose a color').select_option(label='Blue')

# Multiple selected items
page.get_by_label('Choose multiple colors').select_option(['red', 'green', 'blue'])
```


2.6.6 复选框和单选

使用 `locator.set_checked()` 是选中和取消选中复选框或单选按钮的最简单方法。 `input[type=checkbox]` 此方法可与 `input[type=radio]` 和元素一起使用 `[role=checkbox]`。

```
# Check the checkbox
page.get_by_label('I agree to the terms above').check()

# Assert the checked state
assert page.get_by_label('Subscribe to newsletter').is_checked() is True

# Select the radio button
page.get_by_label('XL').check()
```

2.6.7 focus() 聚焦给定元素

对于处理焦点事件的动态页面，您可以使用 `locator.focus()` 聚焦给定元素。

```
page.get_by_label('password').focus()
```

2.6.8 drag_to 拖动

您可以使用 `locator.drag_to()` 执行拖放操作。此方法将：

- 将鼠标悬停在要拖动的元素上。
- 按鼠标左键。
- 将鼠标移动到将接收放置的元素。
- 松开鼠标左键。

```
page.locator("#item-to-be-dragged").drag_to(page.locator("#item-to-drop-at"))
```

如果您想精确控制拖动操作，请使用较低级别的方法，如 `locator.hover()`、`mouse.down()`、`mouse.move()` 和 `mouse.up()`。

```
page.locator("#item-to-be-dragged").hover()
page.mouse.down()
page.locator("#item-to-drop-at").hover()
page.mouse.up()
```

如果您的页面依赖于dragover正在调度的事件，则您至少需要移动两次鼠标才能在所有浏览器中触发它。要可靠地发出第二次鼠标移动，请重复mouse.move()或locator.hover()两次。操作顺序是：悬停拖动元素，鼠标向下，悬停放置元素，第二次悬停放置元素，鼠标向上。

2.7 操作iframe

iframe 是web自动化里面一个比较头疼的场景，在Selenium中处理 iframe 需要切换来切换去非常麻烦。

在playwright中，让其变得非常简单，我们在使用中无需切换iframe，直接定位元素即可。

2.7.1 iframe 定位

可以使用 page.frame_locator() 或 locator.frame_locator() 方法创建 FrameLocator 捕获该 iframe 中检索和定位元素。

使用示例：

```
locator = page.frame_locator("my-frame").get_by_text("Submit")
locator.click()
```

使用frame_locator() 定位到iframe上，然后继续在上面使用locator方法定位元素

iframe 定位器是严格的。这意味着如果有多个元素与给定的选择器匹配，则对 iframe 定位器的所有操作都会抛出异常。

```
# Throws if there are several frames in DOM:
page.frame_locator('.result-frame').get_by_role('button').click()

# Works because we explicitly tell locator to pick the first frame:
page.frame_locator('.result-frame').first.get_by_role('button').click()
```

以下代码段在带有 id 为 my-frame 的 iframe 中定位带有文本“提交”的元素，例

如 `<iframe id="my-frame">`：

```
locator = page.frame_locator("#my-iframe").get_by_text("Submit")
locator.click()
```

2.7.2 案例演示

以163 邮箱为例



由于iframe 元素 id 属性是动态可变的 id="x-URS-iframe1676960382133.3657" 可以使用xpath的 contains 模糊匹配，或者css的正则匹配

语法	描述
<code>\$('[name^="value"]')</code>	匹配 name 以 value 开头的元素
<code>\$('[name\$="end"]')</code>	匹配 name 以 end 结尾的元素
<code>\$('[class*="text"]')</code>	匹配class属性包含text的元素

代码示例

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    page = browser.new_page()
    page.goto("https://mail.163.com/")

    # 操作 iframe 上的元素
    # frame = page.frame_locator("iframe[id^=x-URS-iframe]")
    # xpath 模糊匹配
    frame = page.frame_locator('//iframe[contains(@id, "x-URS-iframe")]')
    frame.locator('[name="email"]').fill('yoyoketang')
    frame.locator('[name="password"]').fill("123456")
    frame.locator('#dologin').click()

    page.wait_for_timeout(5000)
    browser.close()

```

当定位到有多个时候，last 将定位器返回到最后一个

```
frame_locator().last
```

first 匹配第一个

```
frame_locator().first
```

还可以使用index索引

```
frame_locator().nth(index)
```

2.7.3 page.frames 获取页面上所有的iframe

获取全部iframes

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    page = browser.new_page()
    page.goto("https://mail.163.com/")
    print(page.frames)
    for f in page.frames:
        print(f)

    page.wait_for_timeout(5000)
    browser.close()

```

运行结果

```

[<Frame name= url='https://mail.163.com/'>, <Frame name=frameforlogin url='about:blank'>, <Frame name=frameJS6 url=
<Frame name= url='https://mail.163.com/'>
<Frame name=frameforlogin url='about:blank'>
<Frame name=frameJS6 url='about:blank'>
<Frame name=x-URS-iframe1676963051544.6213 url='https://dl.reg.163.com/webzj/v1.0.1/pub/index_d12_new.html?cd=%2F%2F'>]

```

2.7.4 frame() 定位方法

还有一种frame() 定位方法，可以根据name属性和url属性匹配

```

frame = page.frame(name="frame-name")
frame = page.frame(url=r".*domain.*")
# Interact with the frame
frame.fill('#username-input', 'John')

```

page.frame 和 page.frame_locator 使用差异

page.frame_locator("") 返回的对象只能用locator() 方法定位元素然后click()等操作元素
page.frame() 返回的对象能直接使用fill() 和 click() 方法

2.8 select 选择框

select 选择框是页面上常见的场景

2.8.1 select 用法

select 元素示例

```
<select multiple>
  <option value="red">Red</div>
  <option value="green">Green</div>
  <option value="blue">Blue</div>
</select>
```

`<select>` 使用`locator.select_option()`选择元素中的一个或多个选项。您可以指定选项`value`，或`label`选择。可以选择多个选项。

```
# Single selection matching the value
page.get_by_label('Choose a color').select_option('blue')

# Single selection matching the label
page.get_by_label('Choose a color').select_option(label='Blue')

# Multiple selected items
page.get_by_label('Choose multiple colors').select_option(['red', 'green', 'blue'])
```

用法

```
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

# single selection matching the value or label
element.select_option("blue")
# single selection matching the label
element.select_option(label="blue")
# multiple selection for blue, red and second option
element.select_option(value=["red", "green", "blue"])
```


2.8.2 使用示例

下拉框选择


新增模块

模块名称:

所属项目:

test 

模块描述:

90c7f90b8b2846d4826efbff1
7fe8af78937349248d16e331b
1wq
接口
test 

从option 中选一个

```
<select id="project" name="project_id" class="selectpicker form-control show-tick" data-live-search="true" data-size="5" title="请选择">
  <option class="bs-title-option" value>请选择</option>
  <option value="56">ed2008d7e06d4560a17474d29</option>
  <option value="55">ed8192ecbf18417b8387d82eb</option>
  <option value="53">eqwrqw</option>
  <option value="43">9d05496be6c24796891009902</option>
  <option value="42">a9a6ba0a4cd944fcbaebff857</option>
  <option value="41">f8990a444e8d49138ea6a2d79</option>
  <option value="40">90c7f90b8b2846d4826efbff1</option>
  <option value="39">7fe8af78937349248d16e331b</option>
  <option value="4">1wq</option>
  <option value="2">接口</option>
  <option value="1">test</option>
</select>
```

示例代码

方法一，先定位select元素，再定位选项

1.根据选项名称定位

```
select = page.get_by_label("所属项目:")
select.select_option("test")
```

2.根据index 索引定位

```
select = page.get_by_label("所属项目:")
select.select_option(index=1)
```

3.根据label 标签定位

如下select

```
<select name="test" id="t" onchange="change(this)" >
  <option value="1" label="第一" selected="selected">第一</option>
  <option value="2" label="第二">第二</option>
  <option value="3" label="第三">第三</option>
  <option value="4" label="第四">第四</option>
</select>
```

option 有label 属性的情况下可以使用label 标签定位

```
select = page.get_by_label("选择:")
select.select_option(label="第四")
```

方法二，通过page对象直接调用

```
page.select_option("select#project", "test")
```

类似于page.fill和page.click的用法

2.9 screenshot 截图

playwright 除了可以截取当前屏幕，还可以截长图，也可以对某个元素截图，这点非常棒。

2.9.1 screenshot 截图

这是捕获屏幕截图并将其保存到文件中的快速方法：

```
page.screenshot(path="screenshot.png")
```

使用示例


```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    context = browser.new_context() # 创建上下文, 浏览器实例1

    page = context.new_page() # 打开标签页
    page.goto("https://www.baidu.com/")
    print(page.title())
    page.screenshot(path="screenshot.png")
```

于是就可以看到截图了



2.9.2 截长图

设置 `full_page=True` 参数 `screenshot` 是一个完整的可滚动页面的屏幕截图，就好像你有一个非常高的屏幕并且页面可以完全容纳它。

```
page.screenshot(path="screenshot.png", full_page=True)
```

2.9.3 捕获图片数据流

您可以获取包含图像的缓冲区并对其进行后处理或将其传递给第三方像素差异工具，而不是写入文件。

```
screenshot_bytes = page.screenshot()  
print(base64.b64encode(screenshot_bytes).decode())
```

2.9.4 截取单个元素

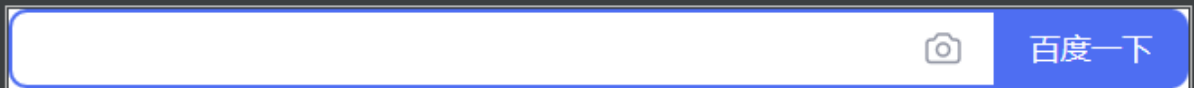
有时截取单个元素的屏幕截图很有用

```
page.locator(".header").screenshot(path="screenshot.png")
```

使用示例,截图百度页面的form 表单输入框和搜索按钮

```
from playwright.sync_api import sync_playwright  
# 上海悠悠 wx:283340479  
# blog:https://www.cnblogs.com/yoyoketang/  
  
with sync_playwright() as p:  
    browser = p.chromium.launch(headless=False, slow_mo=1000)  
    context = browser.new_context() # 创建上下文, 浏览器实例1  
  
    page = context.new_page() # 打开标签页  
    page.goto("https://www.baidu.com/")  
    print(page.title())  
    page.screenshot(path="screenshot.png")
```

于是最终截图效果



2.10 文件上传

如果你之前用过selenium, 肯定遇到过文件上传头疼的事, 有些控件是input输入框, 可以直接

传本地文件地址，然而有些需要弹出本地文件选择器的时候就不好处理了。
playwright 控件优雅的处理了文件上传操作，在这里一切都变得如此简单了。

文件上传主要有2种情况：

1.input 标签的,并且 `type="file"`

2.非input的文件上传框

2.10.1 文件上传

您可以使用`locator.set_input_files()`方法选择要上传的输入文件。它期望第一个参数指向 `<input >` 的输入元素"file"。

数组中可以传递多个文件。如果某些文件路径是相对的，则它们将相对于当前工作目录进行解析。空数组清除所选文件。

```
# Select one file
page.get_by_label("Upload file").set_input_files('myfile.pdf')

# Select multiple files
page.get_by_label("Upload files").set_input_files(['file1.txt', 'file2.txt'])

# Remove all the selected files
page.get_by_label("Upload file").set_input_files([])

# Upload buffer from memory
page.get_by_label("Upload file").set_input_files(
    files=[
        {"name": "test.txt", "mimeType": "text/plain", "buffer": b"this is a test"}
    ],
)
```

如果您手头没有输入元素（它是动态创建的，非input类型的文件上传框），您可以处理 `page.on("filechooser")` 事件或在您的操作中使用相应的等待方法：

```
with page.expect_file_chooser() as fc_info:
    page.get_by_label("Upload file").click()
file_chooser = fc_info.value
file_chooser.set_files("myfile.pdf")
```

几个操作方法

- `file_chooser.element` 返回与此文件选择器关联的输入元素。

- `file_chooser.is_multiple()` 返回此文件选择器是否接受多个文件。
- `file_chooser.page` 返回此文件选择器所属的页面。

设置与此选择器关联的文件输入的值。如果其中一些`filePaths`是相对路径，那么它们将相对于当前工作目录进行解析。对于空数组，清除所选文件。

```
file_chooser.set_files(files)
file_chooser.set_files(files, **kwargs)
```

几个参数

- `files` `pathlib.Path`
- `no_wait_after` 启动导航的操作正在等待这些导航发生并等待页面开始加载。您可以通过设置此标志来选择退出等待。您仅在特殊情况下才需要此选项，例如导航到无法访问的页面。默认为`false`。
- `timeout` 以毫秒为单位的最长时间，默认为 30 秒，传递0以禁用超时。可以使用 `browser_context.set_default_timeout()`或`page.set_default_timeout()`方法更改默认值。

2.10.2 使用示例

第一种场景：文件上传是input 输入框，并且类型是 `type="file"` 的情况



The screenshot displays a web browser interface at the top with a form containing two fields: "文件名称" (File Name) with a text input placeholder "输入文件名称", and "选择文件" (Select File) with a button labeled "选择文件" and the text "未选择任何文件" (No file selected). Below the browser window, the Chrome DevTools "Elements" panel shows the HTML structure of the form. The HTML code is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head> </head>
  <body>
    <form action method="post">
      <div>
        <label for="file_name">文件名称</label>
        <input id="file_name" name="file_name" type="text" placeholder="输入文件名称">
      </div>
      <div>
        <label for="file">选择文件</label>
        <input id="file" type="file" name="file" == $0
      </div>
    </form>
  </body>
</html>
```

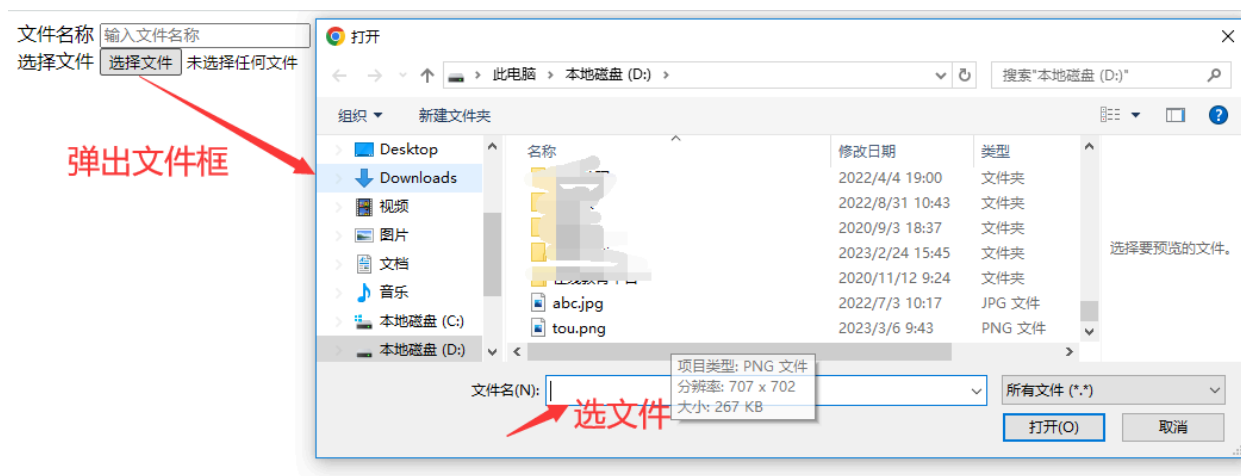
The `<input id="file" type="file" name="file" == $0` line is highlighted in blue, and the `type="file"` attribute is enclosed in a red box, indicating the selected file input element.

可以直接定位输入框，用 `set_input_files('myfile.pdf')` 方法上传文件路径，类似于selenium的`send_keys('文件路径.xx')`

```
page.goto("*****1")

page.get_by_label("文件名称").fill("yoyo")
# 不点开文件框的情况下
page.get_by_label("选择文件").set_input_files('tou.png')
```

第二种场景：如果不是input输入框，必须点开文件框的情况（selenium上没法实现的操作）



可以使用 `page.expect_file_chouser()` 监听到弹出框，在弹出框上输入文件路径

```
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

# 点击选择文件，输入文件
with page.expect_file_chouser() as fc_info:
    page.get_by_label("选择文件").click()
page.pause()
file_chouser = fc_info.value
file_chouser.set_files(r"D:\tou.png")
```

在运行过程中你是感知不到文件选项框弹出来的

异步代码示例

```
async with page.expect_file_chooser() as fc_info:
    await page.get_by_text("Upload file").click()
file_chooser = await fc_info.value
await file_chooser.set_files("myfile.pdf")
```

2.10.3 高级操作-事件监听filechooser

当应该出现文件选择器时触发此操作，例如在单击 `<input type=file>` .可以通过使用 `file_chooser.set_files()`设置输入文件来响应它，之后可以上传这些文件。

```
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

page.on("filechooser", lambda file_chooser: file_chooser.set_files(r"D:\tou.png"))

# 点击选择文件按钮，会触发 filechooser 事件
page.get_by_label("选择文件").click()
```

`page.on("filechooser",)` 会自动监听filechooser 事件，只要有点击了选择文件按钮，就会自动触发。

2.11 文件下载 download

文件下载操作

2.11.1 expect_download()

当浏览器上下文关闭时，所有属于浏览器上下文的下载文件都会被删除。
下载开始后会发出下载事件。下载完成后，下载路径可用：

```
with page.expect_download() as download_info:
    page.get_by_text("Download file").click()
download = download_info.value
# wait for download to complete
path = download.path()
```

2.11.2 download 相关操作

1.取消下载。如果下载已经完成或取消，则不会失败。成功取消后，`download.failure()`将解析为'canceled'.

```
download.cancel()
```

2.删除下载的文件。如有必要，将等待下载完成。

```
download.delete()
```

3.返回下载错误（如果有）。如有必要，将等待下载完成。

```
download.failure()
```

4.获取下载所属的页面。

```
download.page
```

5.下载路径

如果下载成功，则返回下载文件的路径。如有必要，该方法将等待下载完成。该方法在远程连接时抛出。

请注意，下载的文件名是随机 GUID，使用`download.suggested_filename`获取建议的文件名。

```
download.path()
```

返回`NoneType|pathlib.Path` 类型

6.将下载复制到用户指定的路径。在下载仍在进行时调用此方法是安全的。如有必要，将等待下载完成。

```
download.save_as(path)
```

7.返回此下载的建议文件名。

它通常由浏览器根据 `Content-Disposition` 响应标头或 `download` 属性计算得出。请参阅whatwg上的规范。不同的浏览器可以使用不同的逻辑来计算它。

```
download.suggested_filename
```

8.返回下载的 url。

```
download.url
```

2.11.3 使用示例

比如有个页面有下载地址

```
<body>
  <h1>下载文件</h1>
  <a href="https://www.python.org/ftp/python/3.10.10/python-3.10.10-embed-amd64.zip">点我下载</a>
</body>
```

代码示例

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def run(playwright):
    chromium = playwright.chromium
    browser = chromium.launch(headless=False, slow_mo=3000)
    page = browser.new_page()
    page.goto(r'*****down.html')
    with page.expect_download() as download_info:
        page.get_by_text("点我下载").click()
    download = download_info.value
    # wait for download to complete
    print(download.url) # 获取下载的url地址
    # 这一步只是下载下来, 生成一个随机uuid值保存, 代码执行完会自动清除
    print(download.path())
    # 最终可以用save_as 保存到本地
    download.save_as(download.suggested_filename)

    browser.close()

with sync_playwright() as playwright:
    run(playwright)
```


这样在本地就会看到下载成功的文件

2.12 监听 download 事件

download 事件在附件下载开始时发出。用户可以通过传递访问下载内容的基本文件操作实例

使用语法

```
page.on("download", handler)
```

Download 对象由页面通过page.on(" download ")事件分派

```
# 文件下载
with page.expect_download() as download_info:
    page.get_by_text("点我下载").click()
download = download_info.value
# wait for download to complete
print(download.url) # 获取下载的url地址
# 这一步只是下载下来，生成一个随机uuid值保存，代码执行完会自动清除
print(download.path())
# 最终可以用save_as 保存到本地
download.save_as(download.suggested_filename)
```

2.12.1 监听下载事件

比如有个页面有下载地址

```
<body>
  <h1>下载文件</h1>
  <div>
    <a href="https://www.python.org/ftp/python/3.10.10/python-3.10.10-embed-amd64.zip">点我下载</a>
  </div>
  <div>
    <a href="https://registry.npmmirror.com/-/binary/chromedriver/111.0.5563.64/chromedriver_win64.exe" >点我下载</a>
  </div>
  <div>
    <a href="https://registry.npmmirror.com/-/binary/chromedriver/111.0.5563.64/chromedriver_linux64.tar.gz" >点我下载</a>
  </div>
</body>
```

代码

```

from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    page = browser.new_page()
    page.goto("file:///D:/wangyiyun/web_ui/section2/down.html")

    def download_handler(download):
        # 保存文件到本地
        print(download.url) # 获取下载的url地址
        print(download.suggested_filename) # 文件名称
        download.save_as(download.suggested_filename)

    page.on("download", download_handler)

    # 触发第一个文件下载
    page.get_by_text('文件1').click()
    # 继续下载第二个文件
    page.get_by_text('文件2').click()
    page.wait_for_timeout(60000)
    browser.close()

```

如果使用 `page.on("download", handler)` 方式处理文件下载，会有个问题：由于代码运行太快，可能下载文件没下载完成，就直接关闭了上下文，保存到本地代码就会报错。所有需要等待他下载完成，下载时间我们无法控制，所以这种方式有缺陷。

2.12.2 屏蔽下载

如果是通过创建上下文的方式，打开的浏览器窗口，可以在 `new_context` 设置 `accept_downloads` 参数，默认是 `True` 接受下载事件。

设置 `accept_downloads=False` 可以屏蔽下载事件，点击下载按钮的是不会触发下载事件了。

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    # 创建上下文
    context = browser.new_context(accept_downloads=False)
    page = context.new_page()
    page.goto("file:///D:/wangyiyun/web_ui/section2/down.html")

    # 不会触发下载
    page.get_by_text('点我下载').click()

    page.wait_for_timeout(60000)
    browser.close()
```

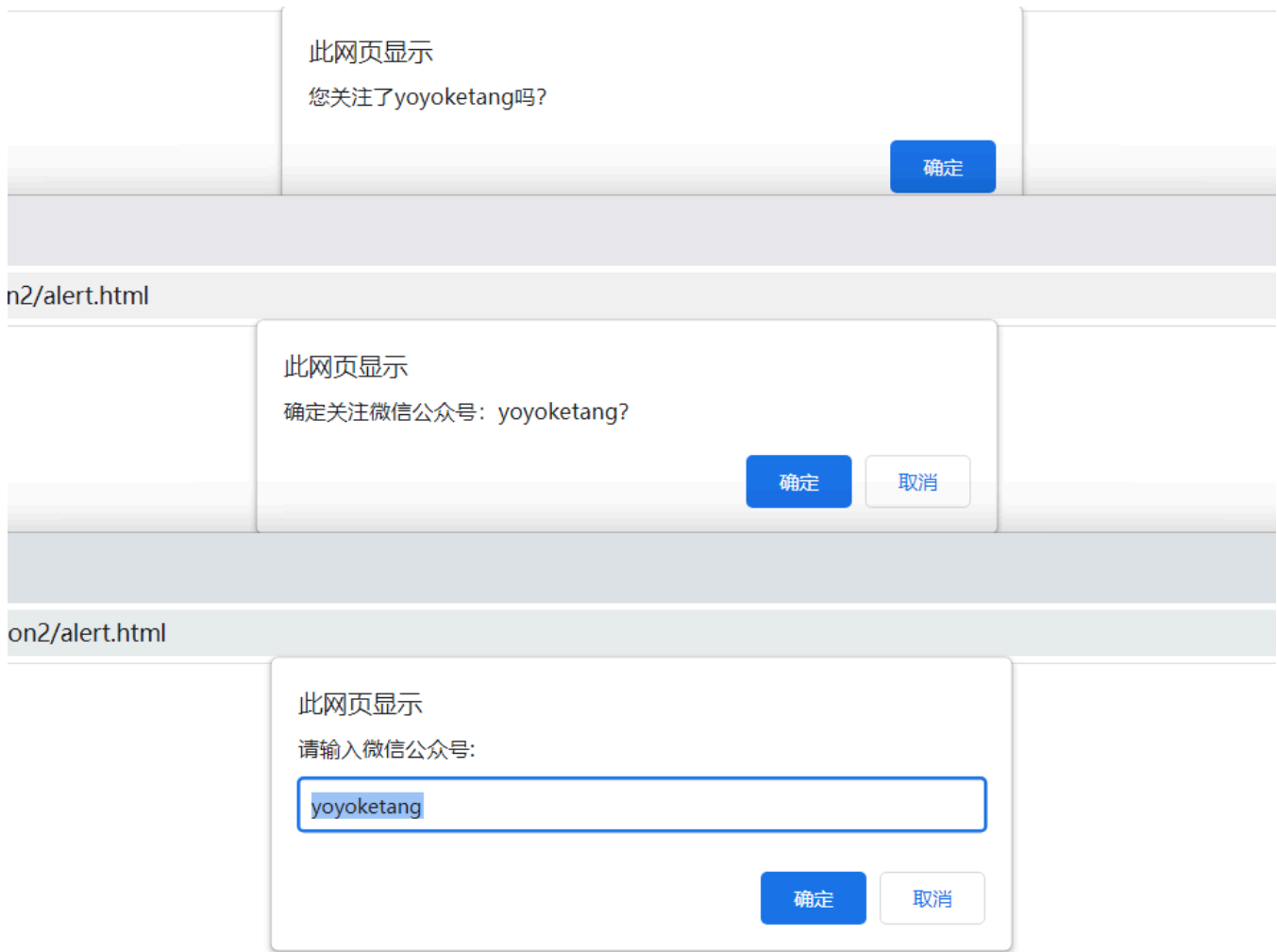
2.13 监听dialog处理alert

网页上的alert 弹出框你不知道什么时候弹出来，selenium 处理alert弹出框的方式是先判断有没有alert 再处理，并且只能处理这一次。

playwright 框架可以监听dialog事件，不管你alert 什么时候弹出来，监听到事件就自动处理了。

2.13.1 认识alert/confirm/prompt

1.如下图，从上到下依次为alert/confirm/prompt，先认清楚长什么样子，以后遇到了就知道如何操作了。



html 相关代码

```
<body>

<input id = "alert" value = "alert" type = "button" onclick = "alert('您关注了yoyoketang吗? ');"/>

<input id = "confirm" value = "confirm" type = "button" onclick = "confirm('确定关注微信公众号: yoyoketang? ');"/>

<input id = "prompt" value = "prompt" type = "button" onclick = "var name = prompt('请输入微信公众号: ');"/>

</body>
```

2.13.2 dialog 事件监听

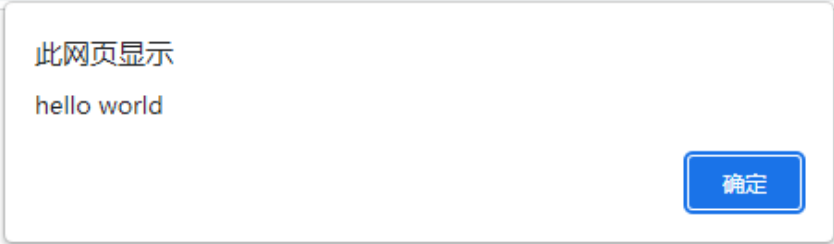
当出现 JavaScript 对话框时发出，例如alert、prompt或。侦听器必须dialog.accept()或dialog.dismiss()对话框 - 否则页面将冻结等待对话框，并且单击等操作将永远不会完成。

```
page.on("dialog", lambda dialog: dialog.accept())
```

注意:当没有`page.on("dialog")`侦听器存在时，所有对话框都会自动关闭。

用法

```
page.on("dialog", handler)
```



此网页显示
hello world

确定

使用示例

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def run(playwright):
    chromium = playwright.chromium
    browser = chromium.launch(headless=False, slow_mo=3000)
    page = browser.new_page()
    # 没监听时，它自动会关闭
    page.evaluate("alert('hello world')")
    browser.close()

with sync_playwright() as playwright:
    run(playwright)
```

当监听器存在时，它必须`dialog.accept()`或`dialog.dismiss()`对话框 - 否则页面将冻结等待对话框，并且单击等操作将永远不会完成。

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def handle_dialog(dialog):
    """监听后处理"""
    print(dialog.message)
    dialog.dismiss()

def run(playwright):
    chromium = playwright.chromium
    browser = chromium.launch(headless=False, slow_mo=3000)
    page = browser.new_page()
    page.on("dialog", handle_dialog)
    page.evaluate("alert('hello world')")
    browser.close()

with sync_playwright() as playwright:
    run(playwright)

```

通过 `page.on("dialog", handler)` 监听到dialog 事件，可以获取到dialog.message内容

2.13.3 dialog 属性和方法

`accept()`当对话框被接受时返回。

```

dialog.accept()
dialog.accept(**kwargs)

```

参数 `prompt_text`（可选), 要在提示中输入的文本。如果对话框 `type` 没有提示，则不会产生任何影响。

`default_value`, 如果对话框是提示的，则返回默认提示值。否则，返回空字符串。

```

dialog.default_value

```

`dismiss` 关闭对话框

```

dialog.dismiss()

```

message 获取对话框中显示的消息

```
dialog.message
```

type返回对话框的类型，可以是 alert , beforeunload , confirm 或 prompt 其中一个。

```
dialog.type
```

2.13.4 prompt提示框

prompt()有两个参数：

- **text**: 要在对话框中显示的纯文本
- **defaultText**: 默认的输入文本

```
<input id = "prompt" value = "prompt" type = "button" onclick = "var name = prompt('请输入微信公众号名');"
```

可以通过 dialog.type 判断到是 prompt 框，就可以给输入框输入内容

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False, slow_mo=1000)
    context = browser.new_context(accept_downloads=True)
    page = context.new_page()
    page.goto("file:///D:/wangyiyun/web_ui/section2/alert.html")

    def handle_dialog(dialog):
        """监听后处理"""
        print(dialog.message)
        print(dialog.type)
        print(dialog.default_value)
        if dialog.type == 'prompt':
            dialog.accept(prompt_text='hello world')
        else:
            dialog.dismiss()
    page.on('dialog', handle_dialog)

    page.wait_for_timeout(60000)
    browser.close()
```

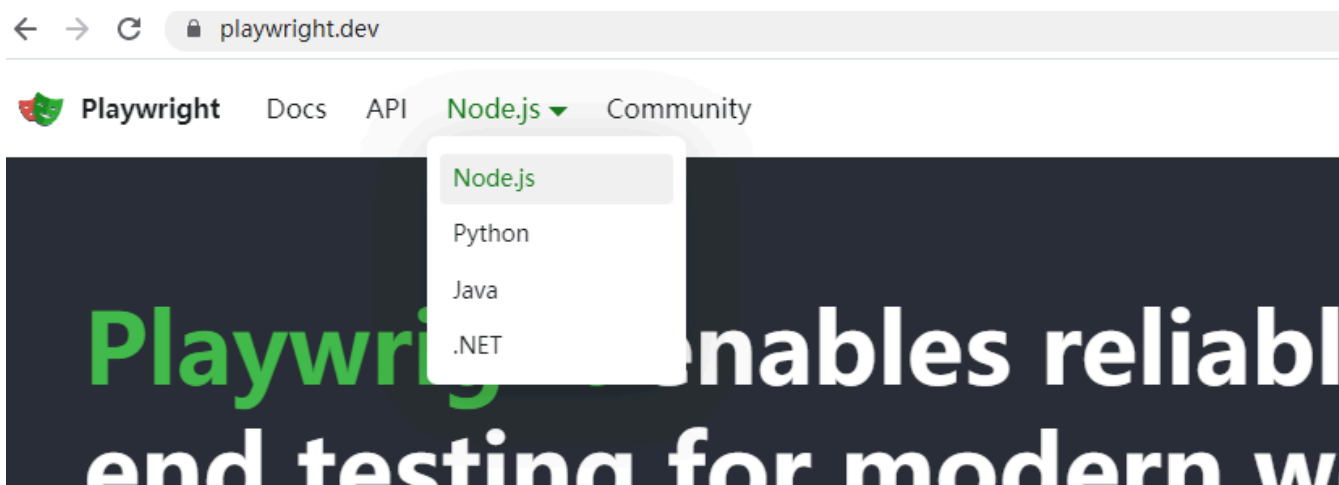
2.14 鼠标悬停 hover

鼠标悬停到某个元素上后出现一些选项，这是很常见的操作了，playwright 操作鼠标悬停非常简单，只需调用hover() 方法。

2.14.1 鼠标悬停

打开官网<https://playwright.dev/>

鼠标悬停到按钮上才会出现选项



代码示例

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://playwright.dev/", timeout=60000, wait_until="domcontentloaded")

    dropdown = page.locator('.dropdown--hoverable')
    # 鼠标悬停
    dropdown.hover()
    # 点选项
    dropdown.locator('.dropdown__link >> text=python').click()

    page.pause()
    browser.close()
```

上面网站加载很慢，可以换成百度的试试

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://www.baidu.com/")

    dropdown = page.locator('#s-user-setting-top')
    # 鼠标悬停
    dropdown.hover()
    # 点选项
    page.get_by_text('搜索设置').click()

    page.pause()
    browser.close()

```

2.15 定位多个元素

我们一般定位到页面上唯一的元素再进行操作，有时候一个元素的属性是一样的，会定位到多个元素

2.15.1 click方法

当定位到页面唯一元素的时候，可以调用click方法

```

<div>
  <label>标签:
    <input type="checkbox" id="a1">旅游
    <input type="checkbox" id="a2">看书
    <input type="checkbox" id="a3" checked="">学习
    <input type="checkbox" id="a4">学python
  </label>
</div>

```

如果直接通过id定位到，可以直接调用click方法

```

a1 = page.locator('#a1')
print(a1) # <Locator>
a1.click()

```

如果通过 `type="checkbox"` 属性定位，会定位到多个元素

```
a1 = page.locator('[type="checkbox"]')
print(a1)    # <Locator>
a1.click()
```

此时会抛出异常

```
playwright._impl._api_types.Error: Error: strict mode violation: locator("[type=\"checkbox\"]") re
  1) <input id="a1" type="checkbox"/> aka get_by_label("标签: \n          旅游\n
  2) <input id="a2" type="checkbox"/> aka get_by_role("checkbox").nth(1)
  3) <input id="a3" checked type="checkbox"/> aka get_by_role("checkbox").nth(2)
  4) <input id="a4" type="checkbox"/> aka get_by_role("checkbox").nth(3)
===== logs =====
waiting for locator("[type=\"checkbox\"]")
```

在异常里面会非常清晰的看到，按给的定位方式有4个元素被定位到，所以不能直接调用click的方法

2.15.2 first 和 last

前面提到如果定位到多个元素，可以用first 和 last 取第一个和最后一个

```
a1 = page.locator('[type="checkbox"]')
a1.first.click() # 点第一个
a1.last.click()  # 点最后个
```

2.15.3 nth() 根据索引定位

前面报错的时候会看到画面有 nth() 可以根据索引找到第几个元素

```
2) <input id="a2" type="checkbox"/> aka get_by_role("checkbox").nth(1)
3) <input id="a3" checked type="checkbox"/> aka get_by_role("checkbox").nth(2)
4) <input id="a4" type="checkbox"/> aka get_by_role("checkbox").nth(3)
```

使用示例

```
a1 = page.locator('[type="checkbox"]')
a1.nth(0).click()
a1.nth(3).click()
```

2.15.4 count 统计个数

使用count 可以统计元素的个数

```
a1 = page.locator('[type="checkbox"]')
print(a1.count()) # 4
```

2.15.5 all() 定位全部

定位全部元素，批量操作

```
all = page.locator('[type="checkbox"]')
for loc in all.all():
    loc.click()
```

2.16 locator.filter()过滤定位器

locator().filter() 和 locator().locator() 有什么区别？

- locator().locator() 是从已经定位到的元素中，继续查询子元素或子孙元素。
- locator().filter() 是从已经定位到的元素中，根据选项缩小现有定位器的范围，可以按文本或定位器过滤。

2.16.1 使用示例

```
<div id="filter">
  <div class="item">
    <a href="">python</a>
  </div>
  <div class="item">
    <a href="">playwright</a>
  </div>
  <div class="item">
    <a href="">selenium</a>
  </div>
  <div class="item">
    <a href="">yoyoketang</a>
  </div>
</div>
```

根据class=item 定位到四个元素

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("file:///D:/wangyiyun/web_ui/section2/demo.html")
    # item
    x = page.locator('.item')
    for item in x.all():
        print('item ----- ', item.inner_html())

    # filter 筛选
    y = x.filter(has_text='python')
    print('yyyy', y.inner_html())

    # locator 是继续查找子元素
    z = x.locator('text=python')
    print('zzzz', z.inner_html())
```

运行结果

```

item -----
    <a href="">python</a>

item -----
    <a href="">playwright</a>

item -----
    <a href="">selenium</a>

item -----
    <a href="">yoyoketang</a>

yyyy
    <a href="">python</a>

zzzz python

```

2.16.2 has_text 参数使用

locator.filter()此方法根据选项缩小现有定位器的范围，例如按文本过滤。它可以链接多次过滤。



比如百度页面的链接，根据div元素定位到多个a标签的，继续缩小范围，按文本过滤

```

page.goto("https://www.baidu.com")

page.locator('#s-top-left>a').filter(has_text="新闻").click()

```

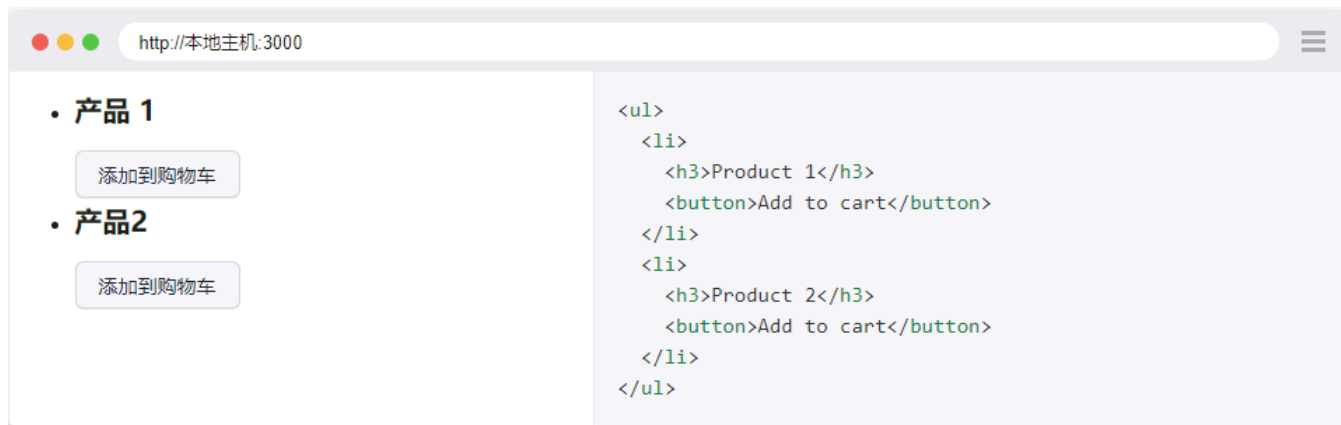
在locator 定位方法里面也可以传一个 has_text 参数，示例如下

```
page.goto("https://www.baidu.com")
page.locator('#s-top-left>a', has_text="新闻").click()
```

在 locator 传 has_text 参数和使用 `filter(has_text="新闻")` 作用是等价的

2.16.3 按另一个 locator 过滤

定位器支持仅选择具有与另一个定位器匹配的后代的元素的选项。因此，您可以按任何其他定位器进行过滤，例如 `locator.get_by_role()`、`locator.get_by_test_id()`、`locator.get_by_text()` 等。



```
page.get_by_role("listitem").filter(
  has=page.get_by_role("heading", name="Product 2")
).get_by_role("button", name="Add to cart").click()
```

我们还可以声明产品卡以确保只有一个

```
expect(
  page.get_by_role("listitem").filter(
    has=page.get_by_role("heading", name="Product 2")
  )
).to_have_count(1)
```

请注意，内部定位器是从外部定位器开始匹配的，而不是从文档根目录开始。

2.16.4 链式定位器

您可以链接创建定位器的方法，例如 `page.get_by_text()` 或 `locator.get_by_role()`，以将搜索范围缩小到页面的特定部分。

在此示例中，我们首先通过定位测试 ID 创建一个名为 product 的定位器。然后我们按文本过滤。我们可以再次使用产品定位器来获取按钮的角色并单击它，然后使用断言来确保只有一个产品带有文本“Product 2”。

```
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

product = page.get_by_role("listitem").filter(has_text="Product 2")

product.get_by_role("button", name="Add to cart").click()
```

2.17 无序列表 listitem 定位

listitem 是无序列表，ul 和 li 标签组合，跟前面讲的 select-option 不一样。在页面上会有2种显示方式

- 1.水平显示的列表
- 2.dropdown 方式，一般需要鼠标悬停，出现对应的列表

2.17.1 listitem 水平显示

看bootstrap3官方的显示 <https://v3.bootcss.com/>

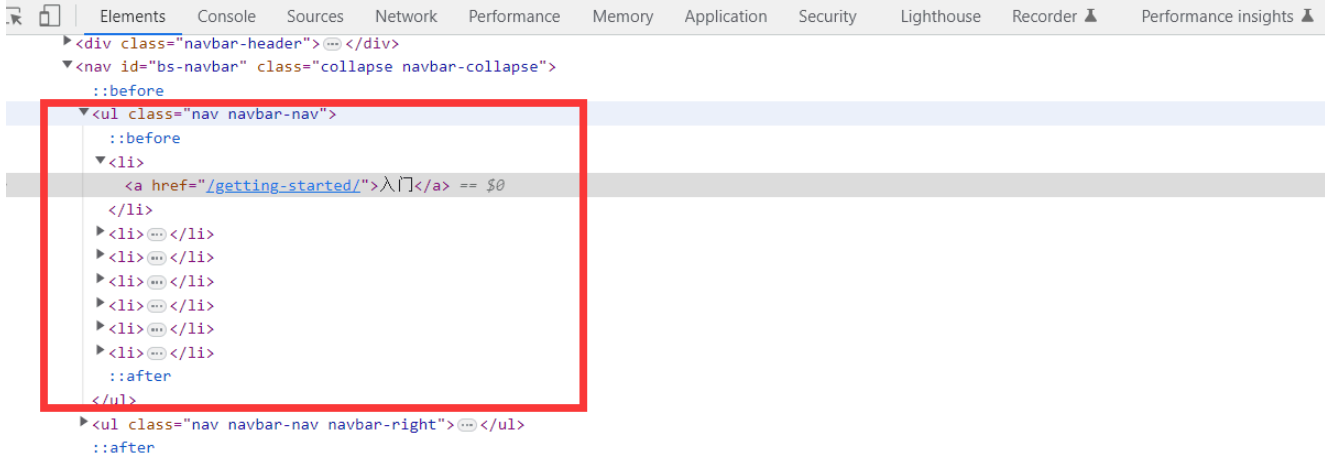
ul.nav.navbar-nav 525.83 × 50

DevTools is now available in Chinese!

Always match Chrome's language

Switch DevTools to Chinese

Don't show again



使用示例

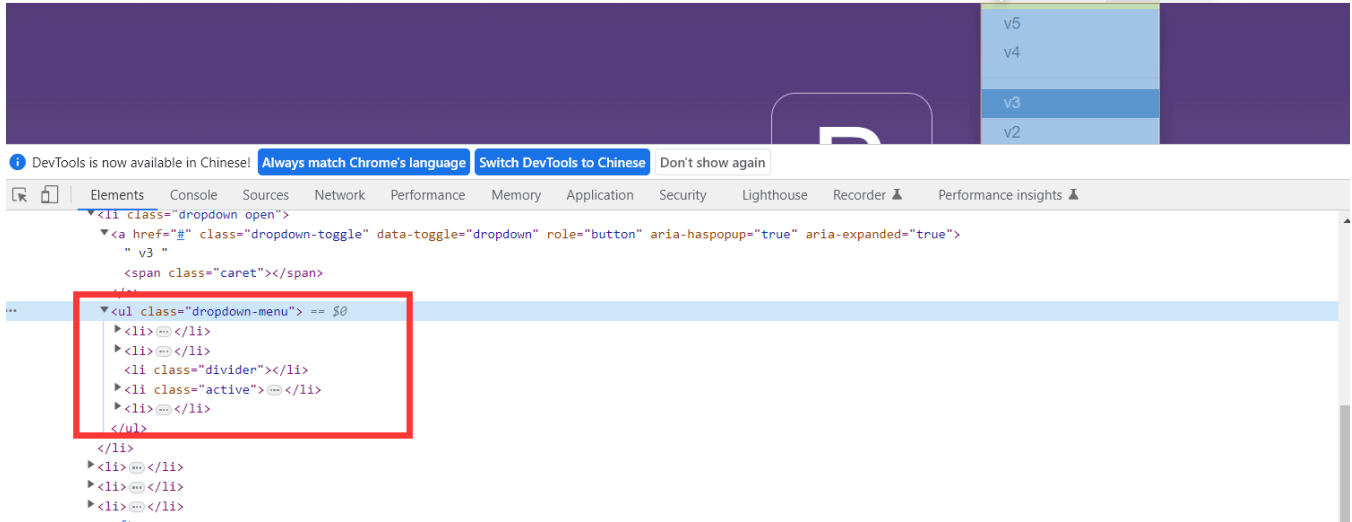
```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://v3.bootcss.com/")

    # listitem 定位
    page.get_by_role('listitem').filter(has_text='入门').click()
    page.pause()
```

2.17.2 dropdown 下拉显示

dropdown 方式的选项，一般需要点击一下，或者鼠标悬停到上面出现列表



```
from playwright.sync_api import sync_playwright
```

```
with sync_playwright() as p:
```

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

```
    page = browser.new_page()
```

```
    page.goto("https://v3.bootcss.com/")
```

```
    # dropdown 先点开
```

```
    dropdown = page.locator('.dropdown')
```

```
    dropdown.click()
```

```
    dropdown.get_by_role('listitem').filter(has=page.get_by_text('v4')).click()
```

```
    page.pause()
```

2.17.3 listitem 定位示例

如果li 里面的按钮文本都是一样的，可以用listitem+filter 过滤定位器组合

对于ul-li的元素，可以用listitem 的角色定位方式

```
<ul>
  <li>
    <h3>Product 1</h3>
    <button>Add to cart</button>
  </li>
  <li>
    <h3>Product 2</h3>
    <button>Add to cart</button>
  </li>
</ul>
```

配合 `locator.filter()` 过滤选择器一起使用

```
page.get_by_role("listitem").filter(has_text="Product 2").get_by_role(
  "button", name="Add to cart"
).click()
```

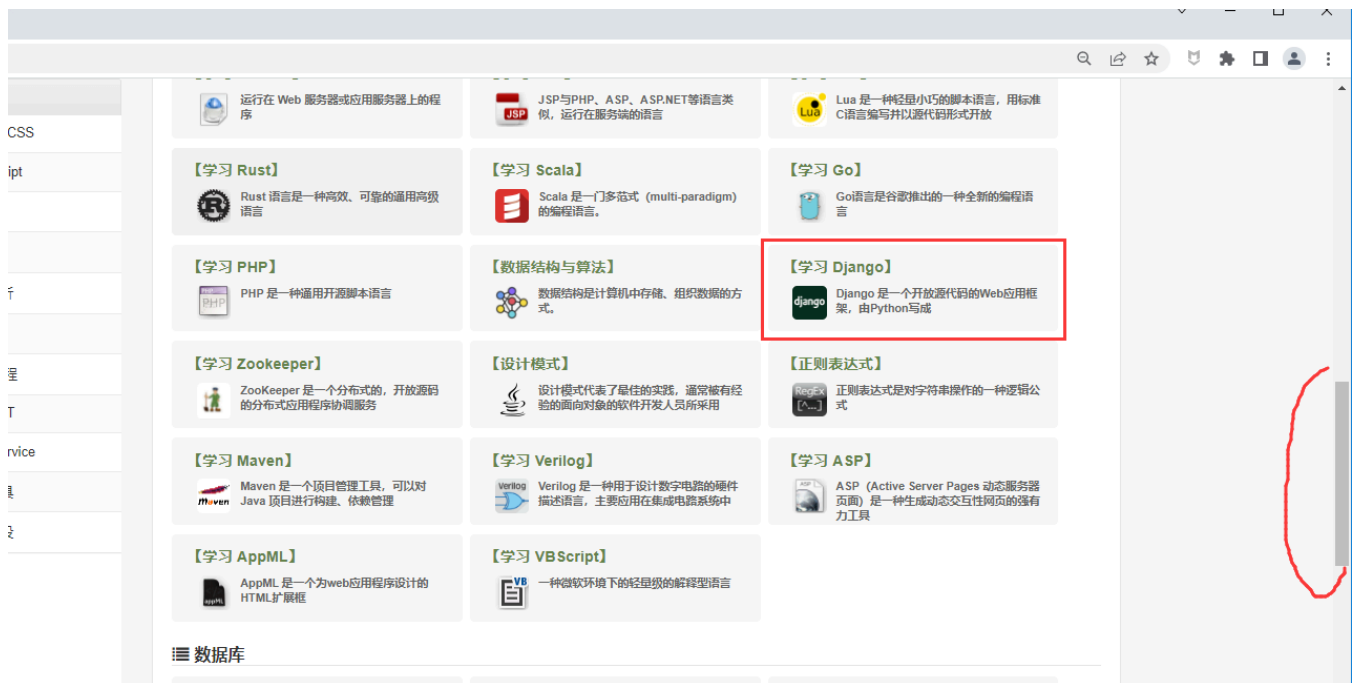
2.18 scroll 滚动到元素出现的位置

当页面超过屏幕的高度时候，需要滚动到元素出现的位置，让元素处于可视的窗口上才能去操作元素。

playwright 在点击元素的时候，会自动滚动到元素出现的位置，这点是非常人性化的。

2.18.1 click 点击操作

比如我需要点如下图中的按钮，是需要先滚动右侧滚动条后元素才会出现



playwright 在点击元素的时候，会自动滚动到元素出现的位置

```
from playwright.sync_api import sync_playwright

with sync_playwright() as pw:
    browser = pw.chromium.launch(headless=False, slow_mo=2000)
    page = browser.new_page()

    page.goto("https://www.runoob.com/")

    # 点击的时候会自动滚动
    page.get_by_text('【学习 Django】').click()

    browser.close()
```

2.18.2 滚动到元素出现位置

如果我们仅仅是让元素出现到窗口的可视范围，可以使用 `scroll_into_view_if_needed()` 方法，它会让元素出现在屏幕的正中间

```
from playwright.sync_api import sync_playwright

with sync_playwright() as pw:
    browser = pw.chromium.launch(headless=False, slow_mo=2000)
    page = browser.new_page()

    page.goto("https://www.runoob.com/")

    # 滚动元素到屏幕可视窗口
    page.get_by_text('【学习 Django】').scroll_into_view_if_needed()
```

2.18.3 hover 方法

hover 方法是把鼠标放到元素上，它也会自动去页面上找到元素，让它出现在可视窗口

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as pw:
    browser = pw.chromium.launch(headless=False, slow_mo=2000)
    page = browser.new_page()

    page.goto("https://www.runoob.com/")

    # 鼠标放到元素上
    page.get_by_text('【学习 Django】').hover()
```

playwright 在操作元素的时候，都会自动去让元素出现在可视窗口。大部分情况不需要我们去操作滚动条。

2.19 事件监听与取消

Playwright 允许监听网页上发生的各种类型的事件，例如网络请求、子页面的创建、文件上传，文件下载，dialog 等。

2.19.1 等待特定事件

大多数时候，脚本需要等待特定事件的发生。下面是一些典型的事件等待模式。

使用`page.expect_request()`等待具有指定 url 的请求：

```
with page.expect_request("**/*logo*.png") as first:
    page.goto("https://v3.bootcss.com/")
print(first.value.url)
```

等待弹出窗口：

```
with page.expect_popup() as popup:
    page.get_by_text("xxx").click()
popup.value.goto("https://v3.bootcss.com/")
```

2.19.2 添加/删除事件

有时，事件在随机时间发生，而不是等待它们，它们需要被处理。Playwright 支持用于订阅和取消订阅事件的传统语言机制：

添加事件使用 `page.on('event', handle)`

```
def print_request_sent(request):
    print("Request sent: " + request.url)

def print_request_finished(request):
    print("Request finished: " + request.url)

page.on("request", print_request_sent)
page.on("requestfinished", print_request_finished)
page.goto("https://wikipedia.org")
```

删除事件使用 `page.remove_listener("event", print_request_finished)`

```
page.remove_listener("requestfinished", print_request_finished)
page.goto("https://www.openstreetmap.org/")
```

2.19.3 添加一次性事件

如果某个事件需要处理一次，有一个方便的 API：

```
page.once("dialog", lambda dialog: dialog.accept("2021"))
page.evaluate("prompt('Enter a number: ')")
```

以上代码 `dialog` 事件仅处理一次。

2.20 checkbox和radio 相关操作

单选框和复选框相关操作总结

`locator.click()` 点击操作

`locator.check()` 选中

`locator.uncheck()` 不选中

`locator.set_checked()` 设置选中状态

`locator.is_checked()` 判断是否被选中

2.20.1 使用场景

radio 和 checkbox 使用场景

```
<div>
  <label>性别:
    <input type="radio" name="sex" id="man" checked>男
    <input type="radio" name="sex" id="woman">女
  </label>
</div>
<div>
  <label>标签:
    <input type="checkbox" id="a1"> 旅游
    <input type="checkbox" id="a2">看书
    <input type="checkbox" id="a3" checked >学习
    <input type="checkbox" id="a4" >学python
  </label>
</div>
```

2.20.2 radio 单选操作

radio 是单选，如果男已经是选择状态，那么点击它是不会改变状态的，只能点另外一个radio 改变状态。

方法1：`click()` 点击

```
# radio 操作
status1 = page.locator('#man').is_checked()
print(status1)
# 选择 女
page.locator('#woman').click()
print(page.locator('#woman').is_checked())
```

方法2：check()

```
# 选择 女
page.locator('#woman').check()
print(page.locator('#woman').is_checked())
```

方法3：set_checked()需传checked 参数，布尔值

```
# 选择 女
page.locator('#woman').set_checked(checked=True)
print(page.locator('#woman').is_checked())
```

另外一种写法, 调用page对象相关方法

```
page.check('#woman')

page.set_checked('#woman', checked=True)
```

需注意的是，如果男本身就是选择状态，去设置unchecked 状态，会报错: Clicking the checkbox did not change its state

```
page.locator('#man').uncheck()
```

报错内容

```
result = next(iter(done)).result()
playwright._impl._api_types.Error: Clicking the checkbox did not change its state
===== logs =====
waiting for locator("#man")
```


2.20.3 checkbox 复选框

checkbox 复选框跟 radio 操作的区别在于，如果已经被选择了，再点击会被取消选中，所以不会有前面的报错。

click 是点击操作，未选中的时候，点击就会被选中。

```
# checkbox 操作
page.locator('#a1').click()
print(page.locator('#a1').is_checked())
```

如果想让元素必须是选择状态(不管之前有没被选中)，可以使用check() 或 set_checked() 方法

```
page.locator('#a1').check()
print(page.locator('#a1').is_checked())
```

set_checked() 方法

```
# checkbox 操作
page.locator('#a1').set_checked(checked=True)
print(page.locator('#a1').is_checked())
```

2.20.4 批量选中checkbox

定位全部CheckBox 批量选中

```
# checkbox 操作
box = page.locator('[type="checkbox"]')
for item in box.all():
    item.check()
```

2.21 css 选择器语法总结

2.21.1 css 选择器语法

`$()` 是jQuery 的语法 css 语法是括号里面的

语法	描述
<code>\$(this)</code>	选取当前 HTML 元素
<code>\$("*")</code>	选取所有元素
<code>\$("p")</code>	匹配 <code><p></code> 标签元素
<code>\$("#kw")</code>	匹配属性 <code>id="kw"</code> 的元素
<code>\$(".info")</code>	选取 class 属性为 <code>info</code> 的元素
<code>\$(p.intro)</code>	选取 class 为 <code>intro</code> 的 <code><p></code> 元素
<code>\$(p:first)</code>	选取第一个 <code><p></code> 元素
<code>\$(ul li:first)</code>	选取第一个 <code></code> 元素的第一个 <code></code> 元素
<code>\$(ul li:last)</code>	选取每个 <code></code> 元素的最后一个 <code></code> 元素
<code>\$("[href]")</code>	选取带有 <code>href</code> 属性的元素
<code>\$(a[target='_blank'])</code>	选取所有 <code>target</code> 属性值等于 <code>"_blank"</code> 的 <code><a></code> 元素
<code>\$(a[target!='_blank'])</code>	选取所有 <code>target</code> 属性值不等于 <code>"_blank"</code> 的 <code><a></code> 元素
<code>\$(":button")</code>	选取所有 <code>type="button"</code> 的 <code><input></code> 元素 和 <code><button></code> 元素
<code>\$(tr:even)</code>	选取偶数位置的 <code><tr></code> 元素
<code>\$(tr:odd)</code>	选取奇数位置的 <code><tr></code> 元素
<code>\$('[name^="value"]')</code>	匹配 name 以 <code>value</code> 开头的元素
<code>\$('[name\$="end"]')</code>	匹配 name 以 <code>end</code> 结尾的元素
<code>\$('[class*="text"]')</code>	匹配class属性包含text的元素
<code>\$('#demo>p')</code>	子代选择器,通过父元素找子元素
<code>\$('#demo p')</code>	后代选择器,通过父元素找子孙元素
<code>\$('#p1+div')</code>	兄弟相邻选择器,通过定位当前元素，找到同一层级的下一个兄弟元素
<code>\$('#p1~div')</code>	同辈选择器，通过定位当前元素，找到同一层级的该元素后面的全部兄弟

语法	描述
<code>\$('p, div')</code>	p标签和div标签元素

2.21.2 基础语法

基本定位语法

- tag 不加任何符号, 定位元素标签
- #id 加 # 号, 定位 id 属性
- .class 加点 . 定位 class 属性

```
<input type="text" class="form-control" id="username" name="username" placeholder="请输入用户名" data-bbox="97 342 1000 360"/>
```

定位上面的 input 输入框, 可以有以下方式

1.根据 input 标签定位, 一般标签不唯一

```
$('input')
```

2.根据id定位

```
$('#username')
```

3.根据class属性定位

```
$('.form-control')
```

4.其它属性定位, 比如name属性

```
$('[name="username"]')
```

placeholder 属性定位

```
$('[placeholder="请输入用户名"]')
```

5.多个属性联合定位

tag标签和id组合

```
$('#input#username')
```

tag标签和class组合

```
$('#input.form-control')
```

tag标签和name属性组合

```
$('#input[name="username"]')
```

name和class组合

```
$('.form-control[name="username"]')
```

name和placeholder组合

```
$('#[name="username"][placeholder="请输入用户名"]')
```

2.21.3 父子层级定位

层级定位主要有2个语法

1. 大于号 `$('#demo>p')` ,大于号只能找到该元素的子元素
2. 空格 `$('#demo p')` ,找到该元素的子孙元素

有时候你需要定位的元素属性很少, 或者很多重复的, 这就需要往上找父元素

`<h3>上海-悠悠接口自动化平台 </h3>`

```
<form id="login-form" action="/api/login" method="POST" role="form" class="form-horizontal fv-form fv-form-l
novalidate="novalidate">
  <button type="submit" class="fv-hidden-submit" style="display: none; width: 0px; height: 0px;"></button>
  <div class="form-group has-feedback">
    ::before
    <label for="username" class="col-sm-3 col-md-3 col-lg-3 control-label">用&nbsp;  户&nbsp;  名:</label>
    <div class="col-xs-12 col-sm-9 col-md-9 col-lg-9">
      <input type="text" class="form-control" id="username" name="username" placeholder="请输入用户名" data-
ame"> == $0
      <i class="form-control-feedback" data-fv-icon-for="username" style="display: none;"></i>
      <small class="help-block" data-fv-validator="notEmpty" data-fv-for="username" data-fv-result="NOT VALI
```

先定位父级元素, 再定位子孙元素

先定位form这一层，再找子孙元素

```
$('#login-form input#username')
```

根据父元素找子元素，用大于号

```
</div>
<div class="form-group has-feedback">...</div>
<div class="col-xs-12 col-sm-9 col-sm-offset-3 col-md-9 col-md-offset-3 col-lg-9 col-lg-offset-3">...</div>
<div class="clearfix">...</div>
<!-- 清除浮动 -->
<div class="text-right">...</div>
<div class="input-group-lg">
  <input class="btn btn-success btn-block" id="loginBtn" type="submit" value="立即登录" > "> ==
</div>
</form>
</div>
/div>
12...
```

语法示例

```
$('.input-group-lg>input')
```

2.21.4 正则匹配属性

css 的正则匹配有3个

语法	描述
<code>\$('#[name^="value"]')</code>	匹配 name 以 value 开头的元素
<code>\$('#[name\$="end"]')</code>	匹配 name 以 end 结尾的元素
<code>\$('#[class*="text"]')</code>	匹配class属性包含text的元素

使用示例

```
$('#iframe[id^=x-URS-iframe]')
```

2.22 xpath 语法总结

2.22.1 xpath 语法总结

根据节点查找相关语法

表达式	描述
/	从根节点选取（取子节点）。
//	相对节点（取子孙节点）。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。
[index]	选取属于 bookstore 子元素的第一个 book 元素 /bookstore/book[1]
text()	文本定位, //*[text()="页面文本"]
contains	模糊匹配, //*[contains(@xxx, "index")], //*[contains(text(), "文本")]

2.22.2 基础语法示例

1.定位所有input标签

```
$x('//input')
```

2.根据id属性定位

```
$x('//*[@id="username"]')
```

3.根据class属性定位

```
$x('//*[@class="form-control"]')
```

4.根据placeholder 属性定位

```
$x('//*[placeholder="请输入用户名"]')
```

5.根据2个属性组合定位

```
$x('//*[placeholder="请输入用户名" and @class="form-control"]')
```

2.22.3 层级关系定位

一个/ 是父子层级关系

2个// 是子孙元素关系

1.根据父元素定位子元素

```
</div>
▶<div class="form-group has-feedback">...</div>
▶<div class="col-xs-12 col-sm-9 col-sm-offset-3 col-md-9 col-md-offset-3 col-lg-9 col-lg-offset-3">...</div>
▶<div class="clearfix">...</div>
<!-- 清除浮动 -->
▶<div class="text-right">...</div>
▼<div class="input-group-lg">
  <input class="btn btn-success btn-block" id="loginBtn" type="submit" value="立即登录" > ==
</div>
</form>
</div>
/div>
</div>
```

```
$x('//div[@class="input-group-lg"]/input')
```

2.根据父元素定位子孙元素，相对层级

```
<h3>上海-悠悠接口自动化平台 </h3>
<div class="login-form">
  <form id="login-form" action="/api/login" method="POST" role="form" class="form-horizontal fv-form fv-form-l
  novalidate="novalidate">
    <button type="submit" class="fv-hidden-submit" style="display: none; width: 0px; height: 0px;"></button>
    <div class="form-group has-feedback">
      ::before
      <label for="username" class="col-sm-3 col-md-3 col-lg-3 control-label">用&nbsp;户&nbsp;名:</label>
      <div class="col-xs-12 col-sm-9 col-md-9 col-lg-9">
        <input type="text" class="form-control" id="username" name="username" placeholder="请输入用户名" data-
        ame"> == $0
        <i class="form-control-feedback" data-fv-icon-for="username" style="display: none;"></i>
        <small class="help-block" data-fv-validator="notEmpty" data-fv-for="username" data-fv-result="NOT VALI
```

先定位父级元素，再定位子孙元素

```
$x('//form[@id="login-form"]//input[@name="username"]')
```

3.也可以根据子元素，定位父元素

```
$x('//input[@id="loginBtn"]/..')
```

2.22.4 text()文本定位

xpath 文本定位

```
$x('//*[text()="没有账号? 点这注册"]')
```

contains 包含属性

```
$x('//*[contains(text(), "点这注册")]')
```

其它属性包含

```
$x('//*[contains(@class, "group-lg")]')
```

2.22.5 index 索引定位

查找父元素下的第几个子元素 (如果结果不是同一个父元素的子元素，下标取值无效)

新闻 hao123 地图 贴吧 视频 图片 网盘 更多

关于百度 About Baidu 使用百度前必读 帮助中心 企业推广 京公网安备11000002000001号 京ICP证030173号 互联网新闻信息服务许可证11220180C



查找父元素下第一个子元素，xpath的索引下标是从1开始

```
$x('//*[id="s-top-left"]/a[1]')
```

2.23 iframe操作元素，监听，执行JS总结

总结关于iframe 的定位，iframe上元素的操作（输入框，点击等），iframe 上的事件监听 与iframe上执行JS脚本的总结。

2.23.1 案例页面

home.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

  <h3>双层iframe</h3>

  <iframe name="yoyo" src="http://47.108.155.10/login.html" style="height: 300px; width: 100%"
  <iframe id="iframe-auto" src="down.html" style="height: 300px; width: 100%" frameborder="0"><
  <iframe id='yoyo2' src="iframe.html" style="height: 800px; width: 100%" frameborder="0"></ifr

</body>
<script>
  iframe= document.getElementById('iframe-auto')
  iframe.setAttribute('id', 'iframe-auto'+(Math.random()*10000000).toString().slice(0, 10) )
</script>

</html>

```

down.html


```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>带iframe的页面</title>
</head>
<body>

  <h3>带iframe的页面</h3>
  <div>      <input id = "alert" value = "alert" type = "button" onclick = "alert('删除后无法还
  </div>

  <iframe src="down.html" style="height: 300px; width: 100%" frameborder="0"></iframe>

</body>
<script>

</script>
</html>

```

2.23.2 iframe 对象的定位

定位iframe 对象，总的来说有四种方法

- page.frame_locator(selector) 通过page对象直接定位iframe 对象，传selector 选择器参数
- page.locator(selector).frame_locator(selector) 通过page对象定位某个父元素，通过locator定位frame_locator(selector)
- page.frame(name, url) 通过page对象直接定位iframe 对象，传name 或者url参数
- page.query_selector(selector).content_frame() 通过query_selector方式，定位到元素，转成frame 对象

page 对象还有2个跟frame 相关的方法

- page.frames 获取page对象全部iframes,包含page本身的frame对象
- page.main_frame 获取page的main_frame （page对象本身也是一个frame对象）

.....

Browser

BrowserContext

BrowserType

CDPSession

ConsoleMessage

Dialog

Download

ElementHandle

Error

FileChooser

Frame

FrameLocator

JSHandle

.....

Sync Async

```
from playwright.sync_api import sync_playwright
```

```
def run(playwright):
    firefox = playwright.firefox
    browser = firefox.launch()
    page = browser.new_page()
    page.goto("https://www.theverge.com")
    dump_frame_tree(page.main_frame, "")
    browser.close()
```

```
def dump_frame_tree(frame, indent):
    print(indent + frame.name + '@' + frame.url)
    for child in frame.child_frames:
        dump_frame_tree(child, indent + "  ")
```

```
with sync_playwright() as playwright:
    run(playwright)
```

使用示例

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def dump_frame_tree(frame, indent):
    print(indent + frame.name + '@' + frame.url)
    for child in frame.child_frames:
        dump_frame_tree(child, indent + "  ")

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("file:///C:/Users/dell/Desktop/home.html")
    dump_frame_tree(page.main_frame, "")
    browser.close()
```

于是可以看到层级结构如下

```
@file:///C:/Users/dell/Desktop/home.html
yoyo@http://47.100.155.10/login.html
iframe-auto6396692.22@file:///C:/Users/dell/Desktop/down.html
yoyo2@file:///C:/Users/dell/Desktop/iframe.html
@file:///C:/Users/dell/Desktop/down.html

Process finished with exit code 0
```

也就是主页面（主页面其实也可以看成一个iframe 对象）下有3个iframe，其中最后一个iframe2下又嵌套了一层iframe。

下面示例用到了3个基本方法

- page.main_frame 获取page对象本身的 frame 对象
- page.frames 获取page对象全部frames 包含page本身的frame对象
- frame.child_frames 获取frame下的全部子 frame 对象

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("file:///C:/Users/dell/Desktop/home.html")
    print('获取page对象本身的frame对象')
    print(page.main_frame)
    print('获取page对象全部frames 包含page本身的frame对象 ')
    print(page.frames)
    print('获取page对象子frame ')
    print(page.main_frame.child_frames)
    browser.close()
```

运行结果

```
获取page对象本身的frame对象
<Frame name= url='file:///C:/Users/dell/Desktop/home.html'>
获取page对象全部frames 包含page本身的frame对象
[<Frame name= url='file:///C:/Users/dell/Desktop/home.html'>, <Frame name=yoyo url='http://127.0.0.0/login.html'>, <Frame name=iframe-auto2834976.42 url='file:///C:/Users/dell/Desktop/down.html'>]
获取page对象子frame
[<Frame name=yoyo url='http://127.0.0.0/login.html'>, <Frame name=iframe-auto2834976.42 url='file:///C:/Users/dell/Desktop/down.html'>]
```

从运行结果可以看出，iframe 对象有2个重要的属性name和url, 可以直接打印出来看看

```
for frame in page.frames:  
    print(frame.name)  
    print(frame.url)
```

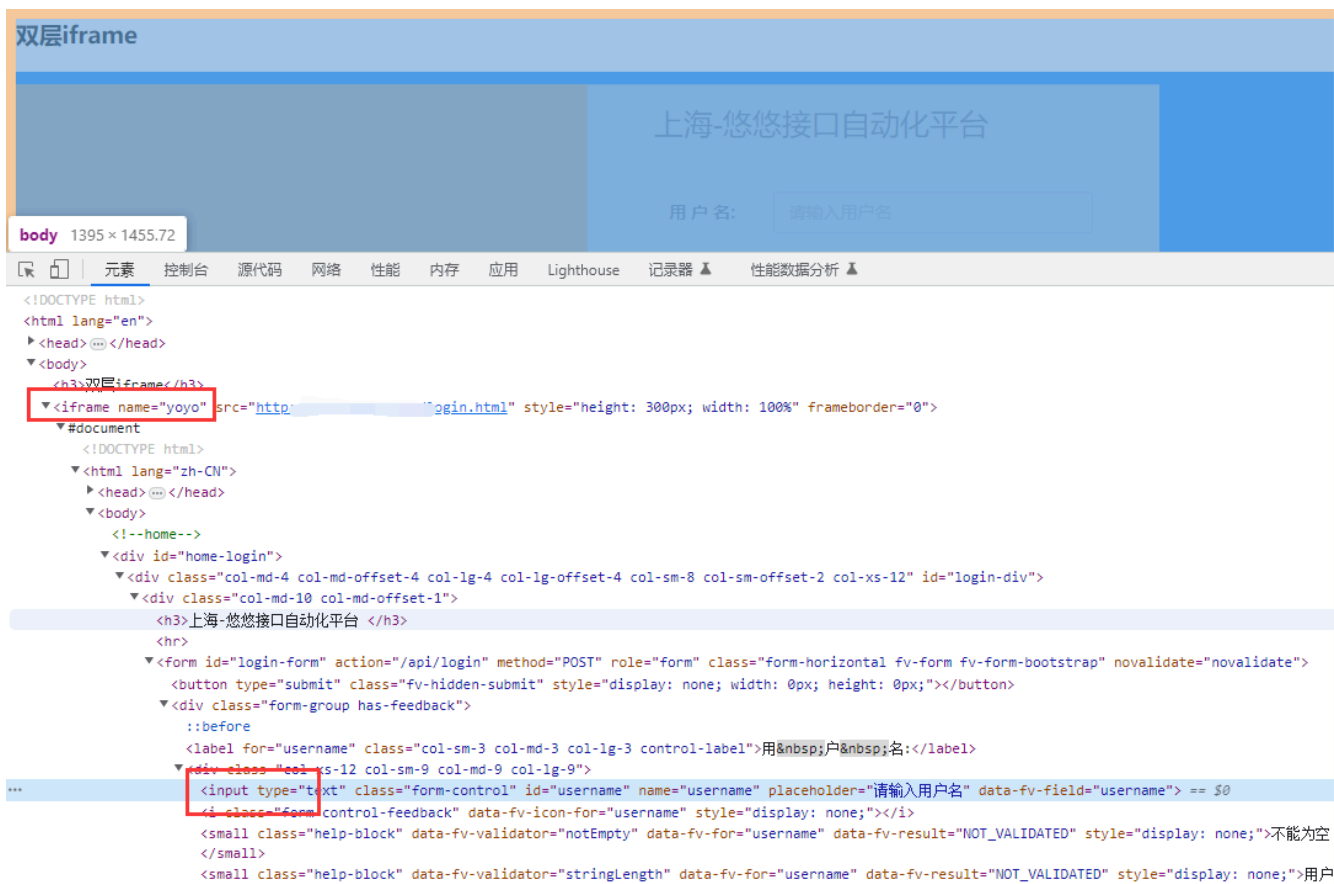
运行结果

```
file:///C:/Users/dell/Desktop/home.html  
yoyo  
http://47.108.155.10/login.html  
iframe-auto9389828.64  
file:///C:/Users/dell/Desktop/down.html  
yoyo2  
file:///C:/Users/dell/Desktop/iframe.html  
  
file:///C:/Users/dell/Desktop/down.html
```

从结果可以看出。iframe 元素的name和id属性，都会被作为那么属性打印出来，如果2个属性都没有，那么获取的name属性为空字符

2.23.4 page.frame_locator(selector) 使用

如下图定位iframe上的输入框，并输入内容



基本语法

```
page.frame_locator(selector)
```

使用示例，可以直接定位到frame对象，继续定位元素操作

```
# 直接定位输入
```

```
page.frame_locator('[name="yoyo"]').locator("#username").fill('yoyoketang')
page.frame_locator('[name="yoyo"]').locator("#password").fill('123456')
```

或者先定位到iframe对象，再通过frame对象操作

```
# frame 对象操作
```

```
frame = page.frame_locator('[name="yoyo"]')
frame.locator("#username").fill('yoyoketang')
frame.locator("#password").fill('123456')
```

只需要定位到frame 对象，后面的元素定位操作都基本一样了。

也可以通过先定位外层的元素，再定位到frame对象，使用基本语法

```
page.locator(selector).frame_locator(selector)
```

2.23.5 page.frame(name, url) 的使用

page.frame 和 page.frame_locator 使用差异

- page.frame_locator("") 返回的对象只能用locator() 方法定位元素然后click()等操作元素
- page.frame() 返回的对象能直接使用fill() 和 click() 方法

page.frame(name, url) 方法可以使用frame的name属性或url属性定位到frame对象

方法一：name 属性可以是iframe 元素的name 或id属性

使用name属性定位示例



```
# name 属性定位
frame = page.frame(name="yoyo")
print(frame)
```

运行结果

```
<Frame name=yoyo url='http://47.108.155.10/login.html'>
```

name 属性不能模糊匹配，只能绝对匹配字符串

iframe元素没有name属性，有id属性，也可以用

```

    <iframe name="yoyo" src="http://.../login.html" style="height: 300px; width: 100%" frameborder="0">
      <#document
        <!DOCTYPE html>
        <html lang="zh-CN">...</html>
      </iframe>
    <iframe id="iframe-auto2298420.99" src="down.html" style="height: 300px; width: 100%" frameborder="0">...</iframe>
    <iframe id="yoyo2" src="iframe.html" style="height: 800px; width: 100%" frameborder="0">
      <#document
        <!DOCTYPE html>
        <html lang="en"> == $0
          <head>...</head>
          <body>...</body>
        </html>
      </iframe>
    <script>...</script>

```

```

# id 属性也能定位
frame = page.frame(name="yoyo2")
print(frame)

```

运行结果

```

<Frame name=yoyo2 url='file:///C:/Users/dell/Desktop/iframe.html'>

```

方法二：url属性定位

url属性的值，就是我们看到页面上的src属性，可以支持模糊匹配

```

# url 支持模糊匹配
frame = page.frame(url='login.html')
print(frame)

```

2.23.6 iframe 上的动态id属性如何定位

有时候，我们看到的iframe的id不是固定的，是动态的一个id，每次刷新，它的值都不一样（一般前面一部分是固定的）

```

<!DOCTYPE html>
<html lang="en">
  <head> ... </head>
  <body>
    <h3>双层iframe</h3>
    <iframe name="yoyo" src="http://.../login.html" style="height: 300px; width: 100%" frameborder="0">
      <#document>
    </iframe>
    <iframe id="iframe-auto5547682.23" src="down.html" style="height: 300px; width: 100%" frameborder="0">
      <#document>
        <!DOCTYPE html>
        <html lang="en">
          <head> ... </head>
          <body> == $0
            <form id="login-form" action="/api/login" method="POST"> ... </form>
            <h1>下载文件</h1>
            <a href="https://registry.npmmirror.com/-/binary/chromedriver/113.0.5672.24/chromedriver_win32.zip">点我下载</a>
          </body>
        </html>
      </iframe>
    <iframe id="yoyo2" src="iframe.html" style="height: 800px; width: 100%" frameborder="0"> ... </iframe>
    <script> ... </script>
  </body>
</html>

```

可以用css的正则匹配元素属性

css的正则匹配

语法	描述
<code>\$('[name^="value"]')</code>	匹配 name 以 value 开头的元素
<code>\$('[name\$="end"]')</code>	匹配 name 以 end 结尾的元素
<code>\$('[class*="text"]')</code>	匹配class属性包含text的元素

使用示例

```

# css 正则匹配属性
frame = page.frame_locator('[id^="iframe-auto"]')
print(frame)
frame.locator('#username').fill('yoyo')

```

也可以使用xpath的contains 包含属性

```

# xpath的contains 包含属性
frame = page.frame_locator('//*[contains(@id, "iframe-auto")]')
print(frame)
frame.locator('#username').fill('yoyo')

```

2.23.7 2层iframe 如何操作

iframe 下嵌套另外一个iframe

```
▼ <body>
  <h3>双层iframe</h3>
  <iframe name="yoyo" src="http://.../login.html" style="height: 300px; width: 100%" frameborder="0">...</iframe>
  <iframe id="iframe-auto5547682.23" src="down.html" style="height: 300px; width: 100%" frameborder="0">...</iframe>
  ▼ <iframe id="yoyo2" src="iframe.html" style="height: 800px; width: 100%" frameborder="0">
    ▼ #document
      <!DOCTYPE html>
      ▼ <html lang="en">
        <head>...</head>
        ▼ <body>
          <h3>带iframe的页面</h3>
          ▼ <iframe src="down.html" style="height: 300px; width: 100%" frameborder="0">
            ▼ #document
              <!DOCTYPE html>
              ▼ <html lang="en">
                <head>...</head>
                ▼ <body>
                  <form id="login-form" action="/api/login" method="POST">...</form> == $0
                  <h1>下载文件</h1>
                  <a href="https://registry.npmmirror.com/-/binary/chromedriver/113.0.5672.24/chromedriver_win32.zip">点我下载</a>
                </body>
              </html>
            </iframe>
          <script> </script>
        </body>
      </html>
    </iframe>
  </body>
```

解决办法没什么技巧，一层一层定位即可

```
# 一层一层定位
frame = page.frame_locator('#yoyo2').frame_locator('[src="down.html"]')
frame.locator('#username').fill("yoyoketang")
```

2.23.8 page.query_selector(selector).content_frame() 方法

还有一个不怎么常用的方法，使用 `page.query_selector(selector)` 元素句柄的方式定位到iframe，然后使用`content_frame()`方法切换到iframe对象上
语法规则

```
page.query_selector(selector).content_frame()
```

使用示例

```
# query_selector 方法
iframe = page.query_selector('[src="down.html"]').content_frame()
print(iframe)
```

2.23.9 监听 iframe 上的事件

事件可以通过page对象直接监听到



使用示例

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def handler(dialog):
    print("监听dialog 事件")
    print(dialog.message)
    # dialog.accept()

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("file:///C:/Users/dell/Desktop/home.html")

    page.on('dialog', handler)

    # 一层一层定位
    frame = page.frame_locator('#yoyo2')
    frame.locator('#alert').click()

    page.pause()
    browser.close()

```

运行结果可以看到page.on()可以截图到iframe上的事件

```

监听dialog 事件
删除后无法还原!

```

其它的下载事件，文件上传监听方法都一样。

2.23.10 在iframe 上执行JavaScript 代码

page.evaluate(js代码) 方法可以直接在page对象上执行JavaScript 代码

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def handler(dialog):
    print("监听dialog 事件")
    print(dialog.message)
    # dialog.accept()

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("file:///C:/Users/dell/Desktop/home.html")

    page.on("dialog", handler)

    # 执行JavaScript
    page.evaluate("alert('hello world')")

    page.pause()
    browser.close()

```

在iframe上执行JavaScript代码，需在iframe对象上执行

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("file:///C:/Users/dell/Desktop/home.html")

    iframe = page.frame(url='down.html')
    # 执行js 给输入框输入内容
    js = "document.getElementById('username').value='yoyo';"
    iframe.evaluate(js)

    page.pause()
    browser.close()

```

如果需要在iframe上执行js代码，必须使用page.frame()方法定位到iframe对象，才有 `iframe.evaluate(js)` 方法执行js。

`page.frame_locator()` 方法只能定位元素操作元素，没有执行js的方法

2.24 svg 元素定位方法总结

SVG英文全称为Scalable vector Graphics，意思为可缩放的矢量图，这种元素比较特殊，需要通过 `name()` 函数来进行定位。

2.24.1 svg 元素定位

demo.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <div id="box1">
    <svg width="500" height="500">
      <linearGradient id="linear">
        <stop stop-color="red" offset="0" />
        <stop stop-color="yellow" offset="1" />
      </linearGradient>
      <text x="10" y="20" fill="url(#linear)">上海-悠悠
        <tspan font-weight="bold" fill="green">hello</tspan>
      </text>
      <!-- 绘制圆形circle -->
      <circle cx="380" cy="300" r="100" fill="red"
        stroke="green" stroke-width="10" stroke-opacity="0.3">
      </circle>
      <!-- 绘制椭圆ellipse -->
      <ellipse cx="200" cy="200" rx="108" ry="50" ></ellipse>
    </svg>
  </div>
  <div id="box2">
    <svg width="200" height="500" xmlns="http://www.w3.org/2000/svg">
      <defs>
        <linearGradient id="linearGradient">
          <stop stop-color="red" offset="0" />
          <stop stop-color="blue" offset="1" />
        </linearGradient>
        <radialGradient id="radialGradient" cx="0.25" cy="0.25" r="0.5">
          <stop stop-color="pink" offset="0" />
          <stop stop-color="orange" offset="1" />
        </radialGradient>
        <pattern id="pattern" x="0" y="0" width=".25" height=".25">
          <rect x="0" y="0" width="50" height="50" fill="skyblue"
            stroke="black" stroke-width="1" />
          <rect x="0" y="0" width="25" height="25"
            fill="url(#linearGradient)" />
          <circle cx="25" cy="25" r="20"
            fill="url(#radialGradient)" fill-opacity=".5" />
        </pattern>
      </defs>
      <rect x="0" y="0" width="200"
        height="200" fill="url(#pattern)" />
    </svg>
  </div>
</body>
</html>
```

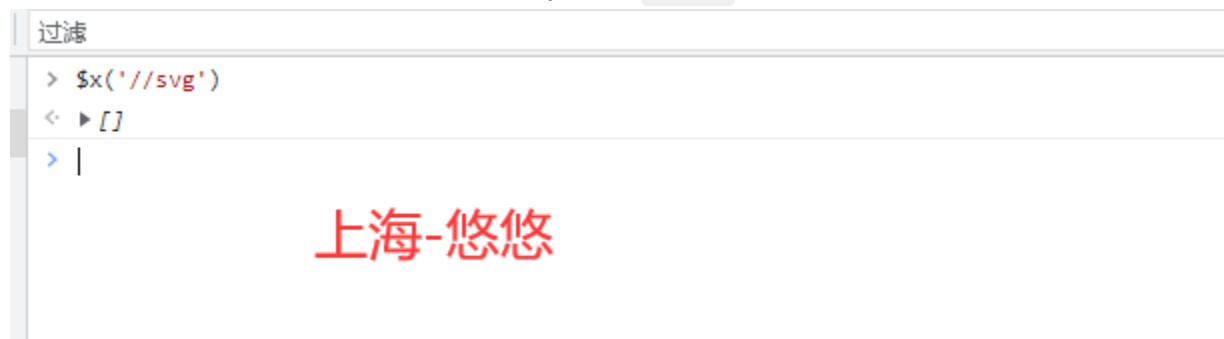
```
</svg>
</div>
</body>
</html>
```

如下看到的svg 标签，就是svg元素

上海-悠悠 hello



用普通的标签定位，是无法定位的,如xpath的 //svg



只能通过 name() 函数来定位 `//*[name()='svg']`

```

> $x('//svg')
< ▶ []

> $x('//*[name()="svg"]')
< ▼ (2) [svg, svg] ⓘ
  ▶ 0: svg
  ▶ 1: svg
    length: 2
  ▶ [[Prototype]]: Array(0)
>

```

2.24.2 页面上用多个svg元素

如果页面上用多个svg元素，通过 `//*[name()="svg"]` 会定位全部的svg元素，为了区分定位具体的哪个，可以通过父元素的区分

```
//*[id="box1"]//*[name()="svg"]
```

```

> $x('//*[name()="svg"]')
< ▶ (2) [svg, svg]

> $x('//*[id="box1"]//*[name()="svg"]')
< ▶ [svg]

> $x('//*[id="box2"]//*[name()="svg"]')
< ▶ [svg]
>

```

除了用父元素区分，也可以用其它属性组合，svg属性加其它属性，用 `and` 组合

```
//*[name()="svg" and @width="500"]
```

```

> $x('//*[name()="svg" and @width="500"]')
< ▶ [svg]

> $x('//*[name()="svg" and @width="200"]')
< ▶ [svg]
>

```

2.24.3 定位svg 上的子元素

如果需要定位svg 下的子元素，如下图的text属性

```

    <div id="box1">
      <svg width="500" height="500">
        <linearGradient id="linear"></linearGradient>
        <text x="10" y="20" fill="url(#linear)"></text> == $0
        <!-- 绘制圆形circle -->
        <circle cx="380" cy="300" r="100" fill="red" stroke="green" stroke-width="10" stroke-opacity="0.3">
        <!-- 绘制椭圆ellipse -->
        <ellipse cx="200" cy="200" rx="108" ry="50"></ellipse>
      </svg>
    </div>
    <div id="box2">
      <svg width="200" height="500" xmlns="http://www.w3.org/2000/svg">
        <defs></defs>
        <rect x="0" y="0" width="200" height="200" fill="url(#pattern)"></rect>
      </svg>
    </div>

```

跟前面定位方式一样，还是通过name() 函数来定位子元素的标签

```
//*[@id="box1"]//*[name()="svg"]//*[name()="text"]
```

```

> $x('//*[@id="box1"]//*[name()="svg"]//*[name()="text"]')
< [text] ⓘ
  ▶ 0: text
    length: 1
    [[Prototype]]: Array(0)
>

```

再举个例子，定位circle 标签

```
//*[@id="box1"]//*[name()="svg"]//*[name()="circle"]
```

```

> $x('//*[@id="box1"]//*[name()="svg"]//*[name()="circle"]')
< [circle] ⓘ
  ▶ 0: circle
    length: 1
    [[Prototype]]: Array(0)
>

```

2.24.4 使用示例

svg 元素定位示例

```
from playwright.sync_api import sync_playwright

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto('file:///C:/Users/dell/Desktop/demo.html')

    # svg 元素定位
    svg1 = page.locator('//*[id="box1"]//*[name()="svg"]')
    print(svg1.get_attribute('width'))

    svg2 = page.locator('//*[id="box2"]//*[name()="svg"]')
    print(svg2.get_attribute('width'))
```

2.25 svg 元素拖拽

本篇讲下关于svg元素的拖拽相关操作。

2.25.1 拖拽 svg 元素

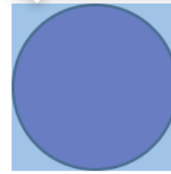
如图所示，svg下的circle元素是可以拖动的

代码下载:<https://www.yiibai.com/downloads/svg/drag.zip>

下载到本地后，浏览器打开snap_events.html

!Pasted image 20230426214434.png

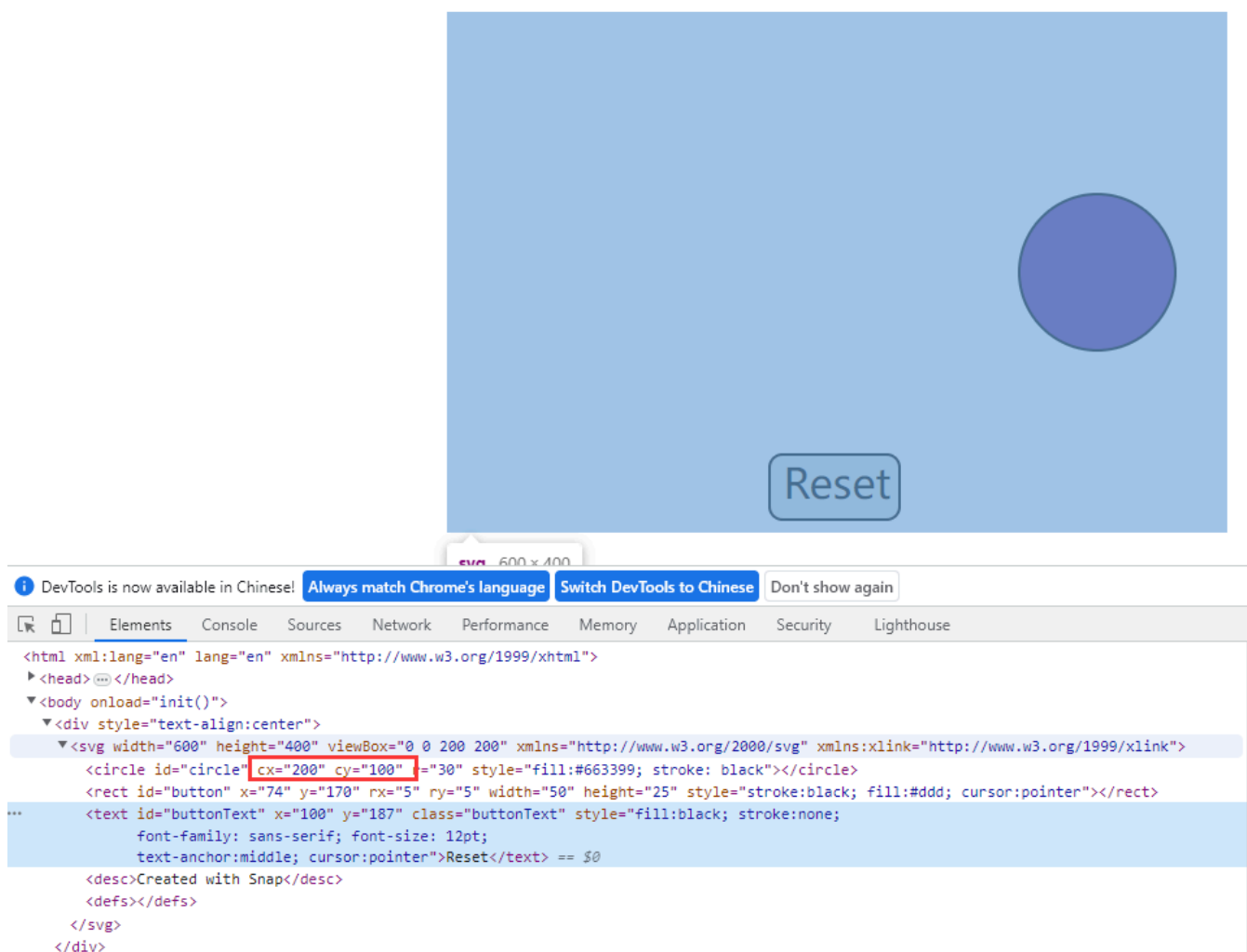
circle#circle 120 × 120



Reset



比如往右拖动 100 个像素，那么cx的值由原来的 `cx="100"` 变成 `cx="200"`



通过鼠标是可以实现拖拽操作的

2.25.2 playwright 拖拽 svg 元素

先定位目标元素，通过 mouse 鼠标事件操作


```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto('file:///C:/Users/dell/Desktop/drag/snap_events.html')

    # svg元素定位
    circle = page.locator('//*[name()="svg"]/*[name()="circle"]')
    print(circle.bounding_box())
    box = circle.bounding_box()
    # svg元素拖拽
    page.mouse.move(x=box['x']+box['width']/2, y=box['y']+box['height']/2)
    page.mouse.down()
    page.mouse.move(x=box['x']+box['width']/2+100, y=box['y']+box['height']/2)
    page.mouse.up(button="middle")
```

如果不确定鼠标down事件有没触发，可以加个元素的事件监听，看console日志

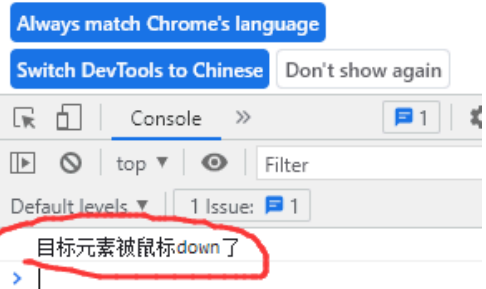
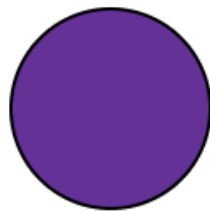
```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto('file:///C:/Users/dell/Desktop/drag/snap_events.html')

    # svg元素定位
    circle = page.locator('//*[@name="svg"]/*[name="circle"]')
    print(circle.bounding_box())
    box = circle.bounding_box()
    # 添加事件监听
    circle.evaluate('node => node.addEventListener("mousedown", '
        'function(){console.log("目标元素被鼠标down了");});')

    # svg元素拖拽
    page.mouse.move(x=box['x']+box['width']/2,
        y=box['y']+box['height']/2)
    page.mouse.down()
    page.mouse.move(x=box['x']+box['width']/2+100,
        y=box['y']+box['height']/2)
    page.mouse.up(button="middle")
```

Reset




```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto('http://124.70.221.221:8200/users/login/')
    page.locator('#username').fill('123@qq.com')
    page.locator('#password_1').fill('123456')
    page.locator('#jsLoginBtn').click()

    page.goto('http://124.70.221.221:8200/users/userinfo/')
    # 日历控件直接输入
    page.locator('#date_day').fill('2021-04-01')

    page.pause()
```

2.26.2 readonly 的日历控件

有些日历控件是 readonly 属性，也就是不能直接输入的

```
<input type="text" id="birth_day" name="birthday" value="2023-04-02" readonly="readonly">
```

这种情况先要执行JavaScript 去掉元素的 readonly 属性，再输入

JavaScript 操作日历控件

```

from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto('http://124.70.221.221:8200/users/login/')
    page.locator('#username').fill('123@qq.com')
    page.locator('#password_1').fill('123456')
    page.locator('#jsLoginBtn').click()

    page.goto('http://124.70.221.221:8200/users/userinfo/')

    # 去掉元素的readonly属性
    js1 = 'document.getElementById("birth_day").removeAttribute("readonly");'
    page.evaluate(js1)
    # 直接给输入框赋值
    js2 = 'document.getElementById("birth_day").value="2021-04-01";'
    page.evaluate(js2)

    page.pause()

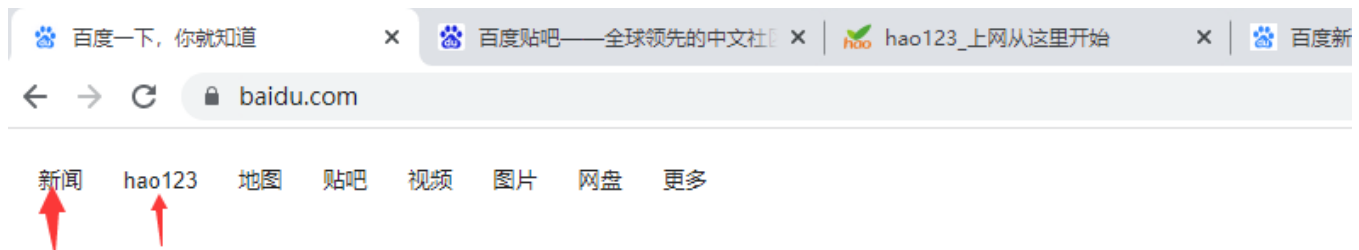
```

2.27 在打开的多个标签页窗口灵活切换

当页面打开了多个标签页后，如何切换到自己需要的标签页上呢？

2.27.1 使用场景

以百度首页为例，当打开多个标签页后，如何切换到自己想要的页面上操作。



通过 `context.pages` 可以获取到所有的page对象，每一个page对象就代表一个标签页实例

```
from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto('https://www.baidu.com')

    # 点开多个标签页
    for link in page.locator('#s-top-left>a').all():
        link.click()

    # 遍历page对象
    for i in context.pages:
        print(i.title())
```

运行结果

```
百度一下，你就知道
hao123_上网从这里开始
百度地图
好看视频--轻松有收获
百度贴吧—全球领先的中文社区
百度新闻—海量中文资讯平台
百度图片-发现多彩世界
```

2.27.2 通过title 判断页面切换

可以写个公共的函数，通过 title 或者 url 地址判断切换到自己想要的页面

```

from playwright.sync_api import sync_playwright
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

def switch_to_page(context, title=None, url=None):
    """切换到指定title 名称 或 url 的 标签页"""
    for item_page in context.pages:
        if title:
            if title in item_page.title():
                # 激活当前选项卡
                item_page.bring_to_front()
                return item_page
        elif url:
            if url in item_page.url:
                # 激活当前选项卡
                item_page.bring_to_front()
                return item_page
    else:
        print("not found title or url")
    return context.pages[0]

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False)
    context = browser.new_context()
    page = context.new_page()
    page.goto('https://www.baidu.com')

    # 点开多个标签页
    for link in page.locator('#s-top-left>a').all():
        link.click()

    # 打开多个tab 标签页, 切换
    page1 = switch_to_page(context, title='hao')
    print(page1.title())

```

2.28 table表格定位与数据获取

定位table 表格内容以及获取table 表格数据。

列表
gtalk.py
管理
模块

<input type="checkbox"/>	ID	td.col-md-2 191.75 × 43.5	所属项目	测试人员	创建时间	
<input type="checkbox"/>	20	登录	test	yoyo	2023-03-01 13:10:12	20
<input type="checkbox"/>	3	查询个人信息	接口	yoyo	2023-02-18 16:48:54	20

元素 控制台 源代码 网络 性能 内存 应用 Lighthouse 记录器 性能数据分析

```

<div class="bootstrap-table bootstrap3">
  <div class="fixed-table-toolbar"></div>
  <div class="fixed-table-container" style="padding-bottom: 0px;">
    <div class="fixed-table-header" style="display: none;"></div>
    <div class="fixed-table-body">
      <div class="fixed-table-loading table table-bordered table-striped table-condensed table-hover" style="top: 37.5px;"></div>
      <table id="table" data-toolbar="#toolbar" class="table table-bordered table-striped table-condensed table-hover">
        <thead class="bg-info"></thead>
        <tbody>
          <tr data-index="0">
            <td class="bs-checkbox" style="width: 36px;"></td>
            <td class="col-md-1">20</td>
            <td class="col-md-2">== $0
              <a href="javascript:;" onclick="EditViewById(0)" title="编辑">登录</a>
            </td>
            <td class="col-md-2">test</td>
            <td class="col-md-2">yoyo</td>
            <td class="col-md-2">2023-03-01 13:10:12</td>
            <td class="col-md-1">20</td>
          </tr>
          <tr data-index="1">
            <td class="bs-checkbox"></td>
            <td class="col-md-1">3</td>
            <td class="col-md-2">查询个人信息</td>
            <td class="col-md-2">接口</td>
            <td class="col-md-2">yoyo</td>
            <td class="col-md-2">2023-02-18 16:48:54</td>
            <td class="col-md-1">20</td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>

```

于是套用上面的语法

```
$x('//*[id="table"]/tbody/tr[1]/td[3]')
```

制台 源代码 网络 性能 内存 应用 Lighthouse 记录器 性能数据分析

过滤

```

> $x('//*[id="table"]/tbody/tr[1]/td[3]')
< ▼ [td.col-md-2] ⓘ
  ▶ 0: td.col-md-2
    length: 1
    [[Prototype]]: Array(0)
>

```

示例2：定位第一行的删除按钮

```
$x('//*[id="table"]/tbody/tr[1]/td[8]/a[2]')
```

<input type="checkbox"/>	ID	模块名称	所属项目	测试人员	创建时间	更新时间	a.btn.btn-xs.btn-danger
<input type="checkbox"/>	20	登录	test	yoyo	2023-03-01 13:10:12	2023-03-01 13:10:12	✎ ✕
<input type="checkbox"/>	3	查询个人信息	接口	yoyo	2023-02-18 16:48:54	2023-02-18 16:48:54	✎ ✕

```
<tbody>
  <tr data-index="0">
    <td class="bs-checkbox" style="width: 36px;"><input type="checkbox"/></td>
    <td class="col-md-1">20</td>
    <td class="col-md-2">
      <a href="javascript:;" onclick="EditViewById(0)" title="编辑">登录</a>
    </td>
    <td class="col-md-2">test</td>
    <td class="col-md-1">yoyo</td>
    <td class="col-md-2">2023-03-01 13:10:12</td>
    <td class="col-md-2">2023-03-01 13:10:12</td>
  </tr>
  <tr data-index="1">
    <td class="col-md-2" style="min-width: 100px; max-width: 260px; white-space: nowrap; text-overflow: ellipsis; overflow: hidden; text-align: center;">
      <a href="javascript:;" class="btn btn-xs btn-info" style="margin: 0 5px;" onclick="EditViewById(0)" title="编辑"><input type="button" value="编辑"/></a>
      <a href="javascript:;" class="btn btn-xs btn-danger" style="margin: 0 5px;" onclick="DeleteById(20)" title="删除"><input type="button" value="删除"/></a>
    </td>
  </tr>
  <tr data-index="1"></tr>
  <tr data-index="2"></tr>
  <tr data-index="3"></tr>
</tbody>
```

<input type="checkbox"/>	ID	模块名称	所属项目	测试人员	创建时间	更新时间	操作
<input type="checkbox"/>	20	登录	test	yoyo	2023-03-01 13:10:12	2023-03-01 13:10:12	<div> <div>a.btn.btn-xs.btn-danger 24 × 22</div> <div>   </div> </div>
<input type="checkbox"/>	6	查询个人信息	test		2023-02-18	2023-02-18	<div>   </div>

过滤

```
> $('//*[@id="table"]/tbody/tr[1]/td[8]/a[2]')
```

▼ [a.btn.btn-xs.btn-danger] ⓘ

- ▶ 0: a.btn.btn-xs.btn-danger
 - length: 1
 - [[Prototype]]: Array(0)

2.28.3 获取当前表格总数

如何获取当前表格有几行呢？可以定位全部的tr 元素，计算tr 的个数，就是总行数了

```
$x('//*[@id="table"]/tbody/tr')
```

<input type="checkbox"/>	ID	模块名称	所属项目	测试人员	创建时间	更新时间	操作
<input type="checkbox"/>	20	登录	test	yoyo	2023-03-01 13:10:12	2023-03-01 13:10:12	 
<input type="checkbox"/>	3	查询个人信息	接口	yoyo	2023-02-18 16:48:54	2023-02-18 16:48:54	 
<input type="checkbox"/>	2	注册	test	yoyo	2023-02-18 16:48:49	2023-02-18 16:48:49	 
<input type="checkbox"/>	1	登录	接口	yoyo	2023-02-18 16:48:42	2023-02-18 16:48:42	 

控制台 源代码 网络 性能 内存 应用 Lighthouse 记录器 性能数据分析

过滤

默认级别 1 个问题

```
> $x('//*[@id="table"]/tbody/tr')
▶ (4) [tr, tr, tr, tr]
```

playwright 获取table表格总行数示例

```
n = page.locator('//*[@id="table"]/tbody/tr')
print(n.count()) # 统计个数
```

2.28.4 获取表格数据

示例1:获取表格第1行的数据

```
n = page.locator('//*[@id="table"]/tbody/tr[1]')
print(n.inner_text()) # 获取第一行数据
# 20      登录      test      yoyo      2023-03-01 13:10:12      2023-03-01 13:10:12
```

示例2:获取第3列的数据

```
# 上海悠悠 wx:283340479
# blog:https://www.cnblogs.com/yoyoketang/

# 获取第3列数据
a = page.locator('//*[@id="table"]/tbody/tr/td[3]')
for td in a.all():
    print(td.inner_text())
```

运行结果

登录
查询个人信息
注册
登录

示例3:获取第1行第3列数据

```
b = page.locator('//*[@id="table"]/tbody/tr[1]/td[3]')
print(b.inner_text()) # 登录
```

2.28.5 判断新增的模块名称在table表格中

获取第3列的数据放到list列表，断言新增的名称在列表里

<input type="checkbox"/>	ID	模块名称	所属项目	测试人员	创建时间	更新时间	操作
<input type="checkbox"/>	20	登录	test	yoyo	2023-03-01 13:10:12	2023-03-01 13:10:12	 
<input type="checkbox"/>	3	查询个人信息	接口	yoyo	2023-02-18 16:48:54	2023-02-18 16:48:54	 
<input type="checkbox"/>	2	注册	test	yoyo	2023-02-18 16:48:49	2023-02-18 16:48:49	 
<input type="checkbox"/>	1	登录	接口	yoyo	2023-02-18 16:48:42	2023-02-18 16:48:42	 

```
# 断言新增模块名称 "登录" 在列表里
loc_table_module = self.add_project.page.locator(
    '//*[@id="table"]/tbody/tr/td[3]'
)
# 获取文本
all_module_names = [i.inner_text() for i in loc_table_module.all()]
print(f"获取页面 table 表格-模块名称列全部内容: {all_module_names}")
assert "登录" in all_module_names
```

网易云视频课程

网易云视频完整课程地址<https://study.163.com/course/courseMain.htm?courseId=1213382811&share=2&shareId=480000002230338>

Playwright + Python

Web 自动化测试

上海-悠悠