

本文主要介绍当前框架的PO模式设计与如何进行多系统支持

1-PO模式简介

PO模型是:Page Object Model的简写 页面对象模型

作用：就是把测试页面和测试脚本进行分离，即把页面元素和方法封装，供测试脚本进行调用;

分层机制，让不同层去做不同类型的事情，让代码结构清晰，增加复用性。

PO设计模式UI自动化测试中最佳的设计模式之一，主要体现在对界面交互细节的封装

优点：提高代码的可读性 减少了代码的重复 提高代码的可维护性，特别是针对UI界面频繁的项目

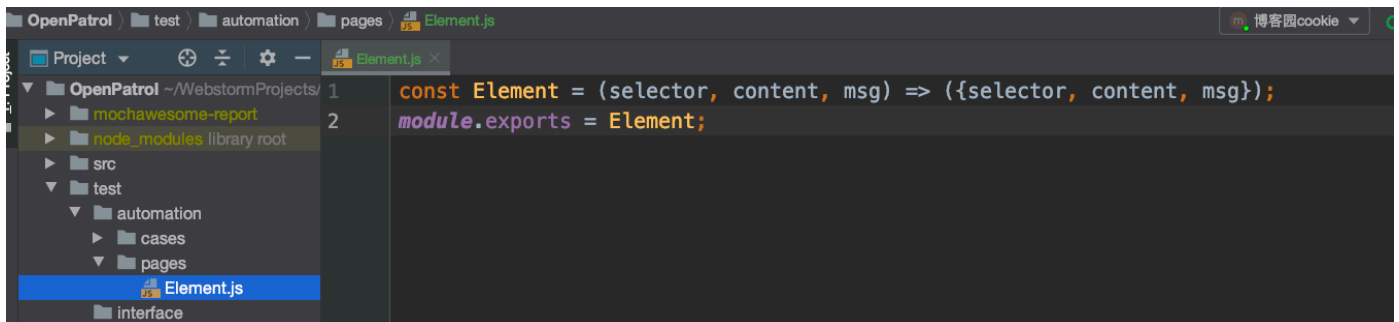
缺点：造成项目结构比较复杂，因为是根据流程进行了模块化处理

2-元素封装

PO模式的核心是基于元素封装，因此基于PO模式的理念，我们进行类似的元素封装定义，我们将页面元素抽象为**Element**，包含：元素选择器，元素内容，自定义信息。

在test/automation/pages/Element.js

```
const Element = (selector, content, msg) => ({selector, content, msg});  
module.exports = Element;
```



因此一个通用的页面模型包含：N*Element+页面方法

3-登陆优化

基于上面的分析，博客园的登陆页面抽象：多个元素+登陆方法

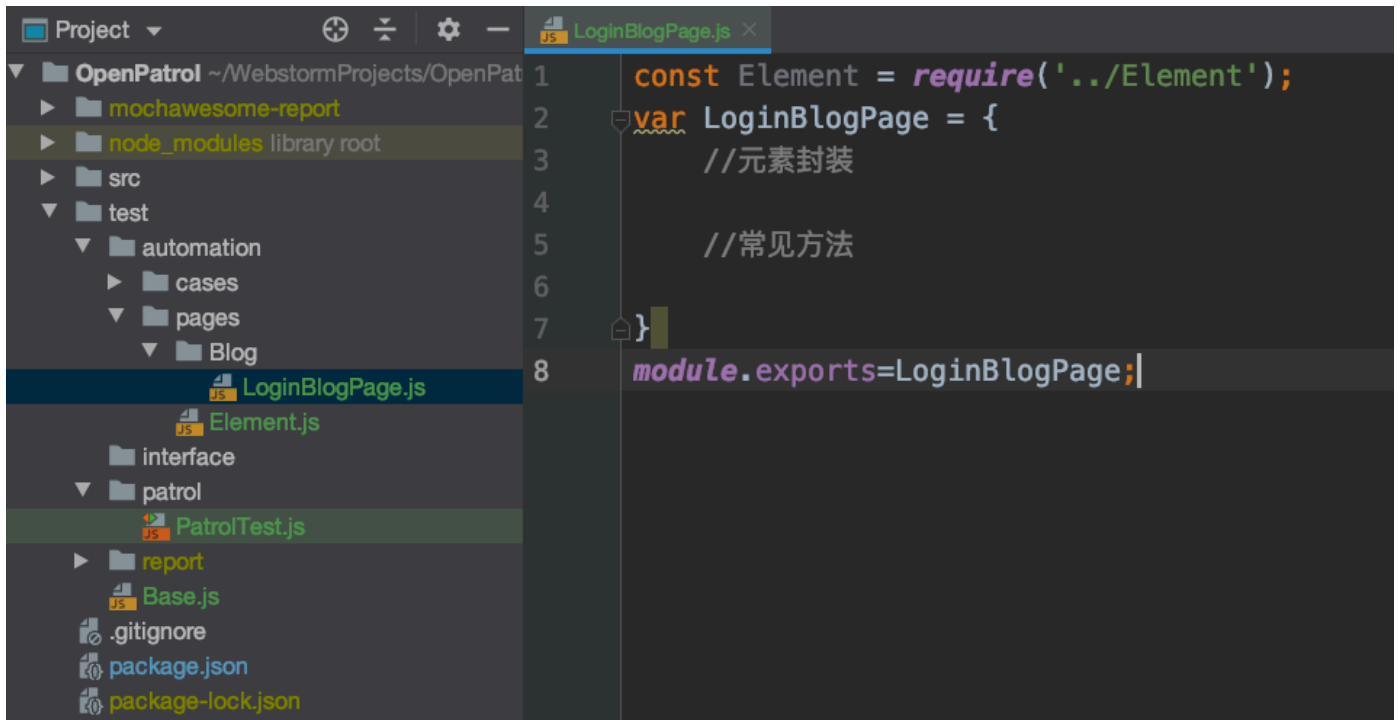
新建：test/automation/pages/Blog/LoginBlogPage.js

每一个页面都包含：元素封装+常见方法

```
const Element = require('../Element');
const LoginBlogPage = {
  //元素封装

  //常见方法

}
module.exports=LoginBlogPage;
```



1-登陆元素封装

cookie登陆过程中，我们针对cookie校验不通过时，需要进行重新的登陆，登陆过程就是针对元素的处理；

```

loginBlogCheck: async function (url, loginName, loginPwd, cookie_File) {
  let login_name = (loginName == undefined ? global.config.login_name : loginName);
  let login_pwd = (loginPwd == undefined ? global.config.login_pwd : loginPwd);
  let cookieFile = (cookie_File === undefined ? global.config.cookies : cookie_File);
  //1-有正确cookie, 直接跳转targetURL
  const goodCookies = await this.checkCookie(login_name, cookieFile, url);
  //2-无正确cookie, 先登陆, 写入cookie信息, 后跳转targetURL
  if (!goodCookies) {
    console.log("cookies校验错误, 开始登陆并写入cookie");
    if (global.cookies === undefined) {
      //1. 登陆: 博客园
      await page.goto("https://account.cnblogs.com/signin");
      //2. 点击: 密码登陆
      await page.waitForSelector('#mat-tab-label-0-0 > div', {timeout: 2*1000}).then(ele=>{ele
      //3. 输入: 用户名
      await page.focus('#mat-input-0');
      await page.type('#mat-input-0', login_name);
      //4. 输入: 密码
      await page.focus('#mat-input-1');
      await page.type('#mat-input-1', login_pwd);
      //5. 勾选: 记住我 (可选)
      // await page.click('#mat-checkbox-1 > label > span.mat-checkbox-inner-container');
      //6. 点击: 登陆
      await page.waitForSelector('body > app-root > app-sign-in-layout > div > div > app-sig
      //等待响应返回200
      await page.waitForResponse(urlOrPredicate: response => response.status() === 200);
      await page.waitForTimeout( milliseconds: 2000);
    }
  }
}

```

上述流程可以抽象为:

1个登陆URL: <https://account.cnblogs.com/signin>

5个元素: 密码登陆按钮, 用户名输入, 密码输入, 记住我单选框, 登陆按钮

1个登陆方法:

博客园用户登录



代码改变世界

密码登录

1

短信登录

登录用户名 / 邮箱

2

请输入登录用户名或邮箱

密码

3

请输入密码

☒ 记住我

4

登录

5

第三方登录/注册



没有账户, [立即注册](#)

```
const Element = require('../Element');
const LoginLoginPage = {
  'login_url' : "https://account.cnblogs.com/signin",
  //元素封装
  'password_login':Element('#mat-tab-label-0-0 > div', '密码登录'),
  'name':Element('#mat-input-0', '登陆用户名/邮箱'),
  'password':Element('#mat-input-1', '密码'),
  'remember':Element('#mat-checkbox-1 > label > span.mat-checkbox-inner-container',
    '记住我'),
  'login_btn':Element('body > app-root > app-sign-in-layout > div > div > app-sign-in
    > app-content-container > div > div > div > form > div > button', '登录'),
  //常见方法
}
module.exports=LoginLoginPage;
```

2-登陆方法封装

封装密码登陆方法：实现和之前完全一样

```
passwordLogin: async function (login_name, login_pwd) {
    //1. 登陆：博客园
    await page.goto(LoginBlogPage.login_url);
    //2. 点击：密码登陆
    await page.waitForSelector(LoginBlogPage.password_login.selector, {timeout: 2 *
1000}).then(ele => {
        ele.click()
    }).catch(e => {
        console.log(e)
    })
    //3. 输入：用户名
    await page.focus(LoginBlogPage.name.selector);
    await page.type(LoginBlogPage.name.selector, login_name);
    //4. 输入：密码
    await page.focus(LoginBlogPage.password.selector);
    await page.type(LoginBlogPage.password.selector, login_pwd);
    //5. 勾选：记住我（可选）
    // await page.click(LoginBlogPage.remember.selector);
    //6. 点击：登陆
    await page.waitForSelector(LoginBlogPage.login_btn.selector, {timeout: 2 *
1000}).then(ele => {
        ele.click()
    }).catch(e => {
        console.log(e)
    })
    //等待响应返回200
    await page.waitForResponse(response => response.status() === 200);
    await page.waitForTimeout(2000);
}
```

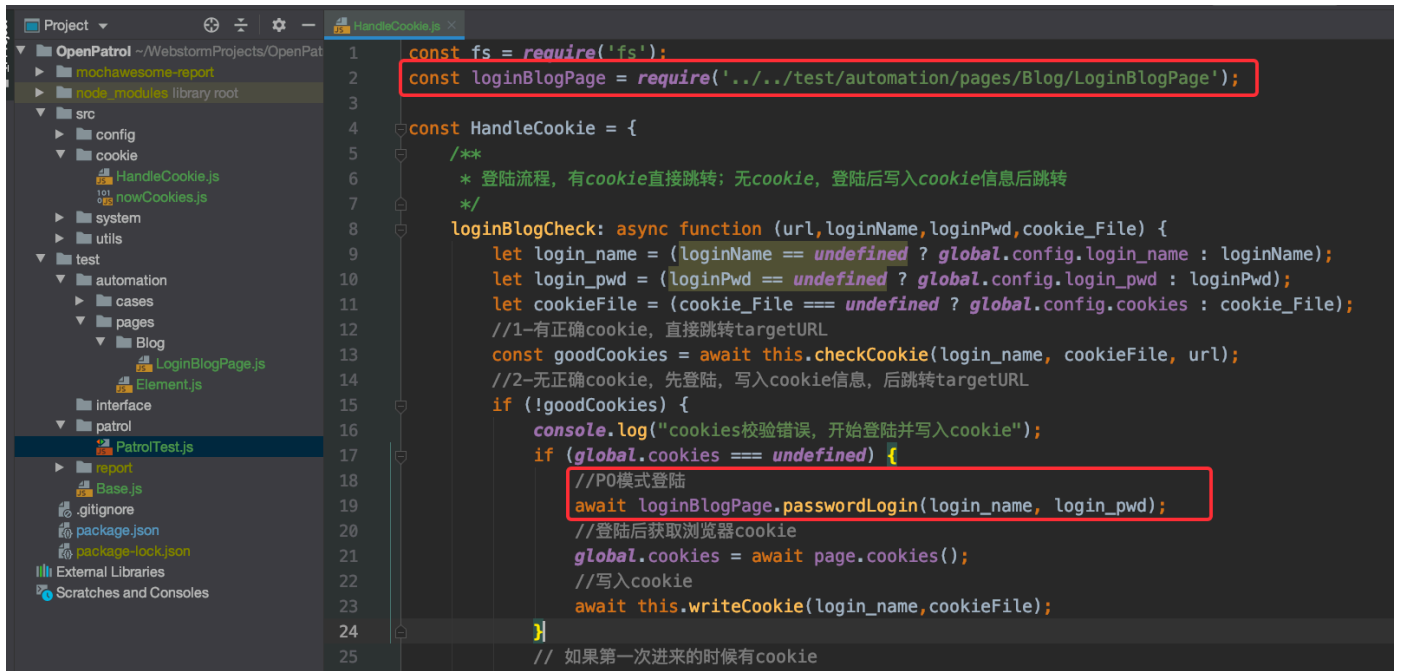
3-cookie登陆优化

修改：src/cookie/HandleCookie.js

登陆过程改为登陆页面操作

```
const loginBlogPage = require('../test/automation/pages/Blog/LoginBlogPage');

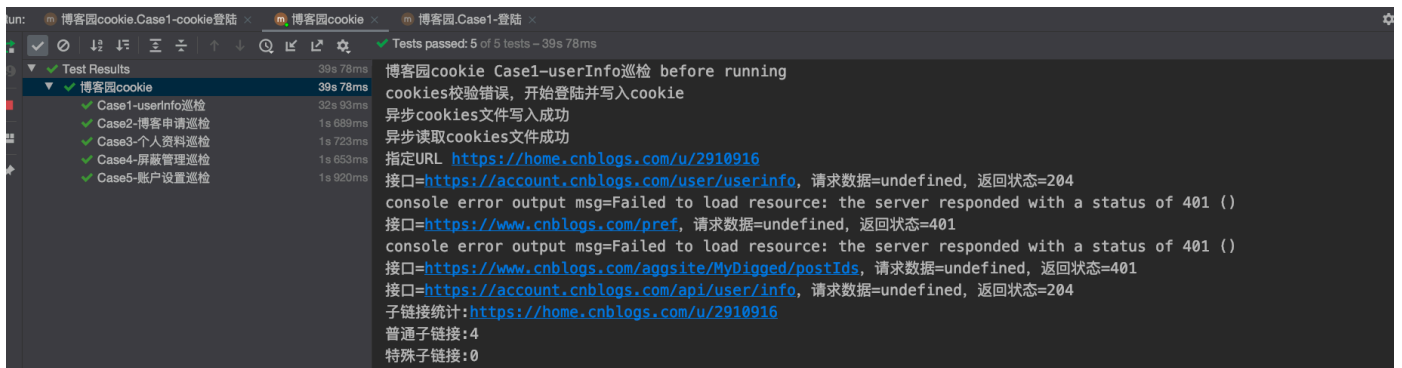
//PO模式登陆
await loginBlogPage.passwordLogin(login_name, login_pwd);
```



4-巡检测试

运行巡检：test/patrol/PatrolTest.js

登陆过程正常



4-拓展

拓展1:登陆页面

test/automation/pages/Blog/LoginBlogPage.js

可以封装更多的页面元素（例如：注册，忘记密码等），方法（短信登陆）

博客园用户登录



代码改变世界

密码登录

短信登录

登录用户名 / 邮箱

密码

忘记登录用户名

忘记密码

☒ 记住我

登录

第三方登录/注册



没有账户，[立即注册](#)

拓展2:其他方式登陆

其他登陆校验，可以参考LoginBlogPage.js的封装，在HandleCookie.js登陆时进行替换就可以完成其他方式的登陆校验（例如：短信登陆流程，忘记密码流程等）

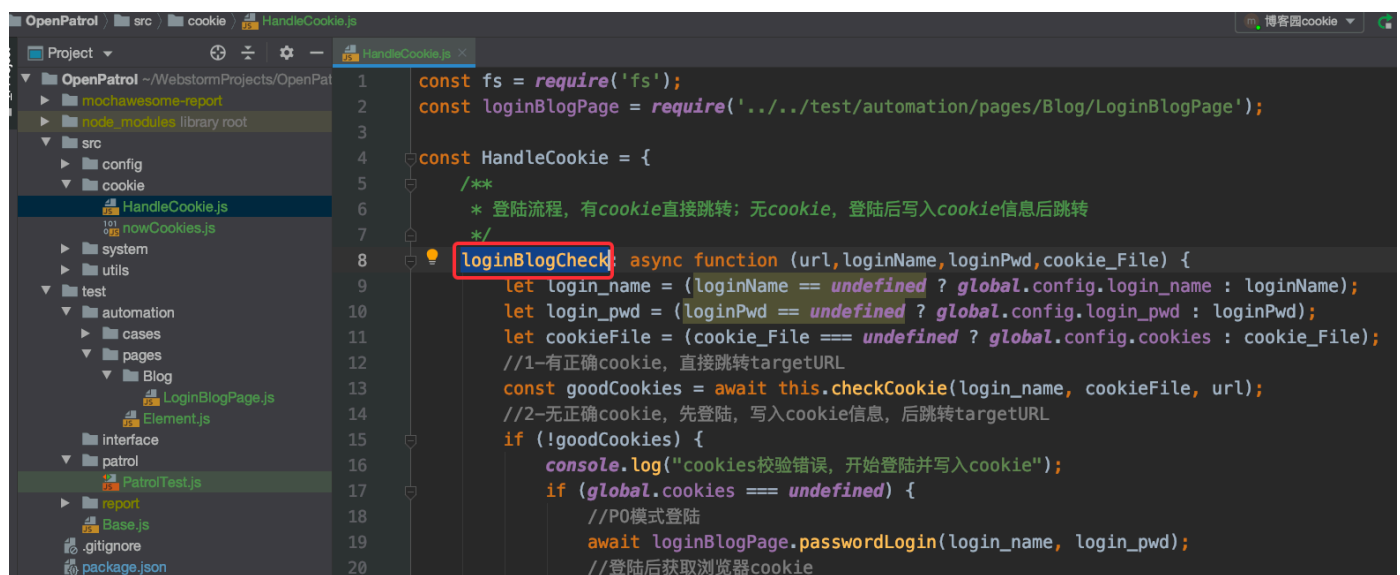
```

loginBlogCheck: async function (url,loginName,loginPwd,cookie_File) {
    let login_name = (loginName == undefined ? global.config.login_name : loginName);
    let login_pwd = (loginPwd == undefined ? global.config.login_pwd : loginPwd);
    let cookieFile = (cookie_File === undefined ? global.config.cookies : cookie_File)
    //1-有正确cookie, 直接跳转targetURL
    const goodCookies = await this.checkCookie(login_name, cookieFile, url);
    //2-无正确cookie, 先登陆, 写入cookie信息, 后跳转targetURL
    if (!goodCookies) {
        console.log("cookies校验错误, 开始登陆并写入cookie"); 替换其他方式登陆
        if (global.cookies === undefined) {
            //P0模式登陆
            await loginBlogPage.passwordLogin(login_name, login_pwd);
            //登陆后获取浏览器cookie
            global.cookies = await page.cookies();
            //写入cookie
            await this.writeCookie(login_name,cookieFile);
        }
    }
}

```

拓展3:其他系统

对于其他系统, 可以参考HandleCookie.js的登陆, 创建登陆其他系统的校验方法:



```

1  const fs = require('fs');
2  const loginBlogPage = require('../test/automation/pages/Blog/LoginBlogPage');
3
4  const HandleCookie = {
5      /**
6       * 登陆流程, 有cookie直接跳转; 无cookie, 登陆后写入cookie信息后跳转
7       */
8      loginBlogCheck: async function (url,loginName,loginPwd,cookie_File) {
9          let login_name = (loginName == undefined ? global.config.login_name : loginName);
10         let login_pwd = (loginPwd == undefined ? global.config.login_pwd : loginPwd);
11         let cookieFile = (cookie_File === undefined ? global.config.cookies : cookie_File);
12         //1-有正确cookie, 直接跳转targetURL
13         const goodCookies = await this.checkCookie(login_name, cookieFile, url);
14         //2-无正确cookie, 先登陆, 写入cookie信息, 后跳转targetURL
15         if (!goodCookies) {
16             console.log("cookies校验错误, 开始登陆并写入cookie");
17             if (global.cookies === undefined) {
18                 //P0模式登陆
19                 await loginBlogPage.passwordLogin(login_name, login_pwd);
20                 //登陆后获取浏览器cookie

```

在用例时可以选择新系统的登陆校验:


```

1  const {describe, it, before, after, afterEach} = require('mocha');
2  const Base = require('../Base');
3  const loginCookie = require('../../src/cookie/HandleCookie');
4  const patrol = require('../../src/utils/Patrol');
5
6  describe('博客园cookie', function () {
7    before(Base.before);
8    // after(Base.after);
9    afterEach(Base.afterEach);
10   this.timeout(1000000);
11   it('Case1-userInfo巡检', async function () {
12     const userInfo = "https://home.cnblogs.com/u/2910916";
13     await loginCookie.loginBlogCheck(userInfo, {loginName: null, loginPwd: null});
14     // 巡检
15     await patrol.patrolHref(page, userInfo);
16   });
17 });

```

可以替换其他系统的登陆校验

5-多系统支持

如果支持多系统（例如其他的WEB系统）或者多环境（预发，QA），可以参考上面的流程，只需要创建新的cookie配置+cookie文件+HandleCookie即可：

假设需要新增处理百度系统的相关操作：

1-src/config/config.js

新增cookies_baidu文件配置

```

'cookies': './src/cookie/nowCookies.js',
'cookies_Baidu': './src/cookie/cookiesBaidu.js',

```

2-新建cookie文件

新建：/src/cookie/cookiesBaidu.js,初始化

```
[{"name": "modTime", "time": "1655889493696", "login_name": "UI自动化"}]
```

3-src/cookie/HandleCookie.js

新建登陆方法，只需要替换cookieFile的位置即可（登陆页面封装参考前面的登陆优化）

```
1  const fs = require('fs');
2  const loginBlogPage = require('../../test/automation/pages/Blog/LoginBlogPage');
3
4  const HandleCookie = {
5
6    /**
7     * 登陆流程, 有cookie直接跳转; 无cookie, 登录后写入cookie信息后跳转
8     */
9    loginBlogCheck: async function (url, loginName, loginPwd, cookie_File) {
10      let login_name = (loginName == undefined ? global.config.login_name : loginName);
11      let login_pwd = (loginPwd == undefined ? global.config.login_pwd : loginPwd); 新系统只需要替换cookie文件即可
12      let cookieFile = (cookie_File === undefined ? global.config.cookies | : cookie_File);
13      //1-有正确cookie, 直接跳转targetURL
14      const goodCookies = await this.checkCookie(login_name, cookieFile);
15      //2-无正确cookie, 先登陆, 写入cookie信息, 后跳转targetURL
16      if (!goodCookies) {
17        console.log("cookies校验错误, 开始登陆并写入cookie");
18        if (global.cookies == undefined) {
```

cookies_Baidu (config, config.js) string