13 | 效率为王：脚本与数据的解耦 + Page Object模型

2018-07-27 茹炳晟

讲述：茹炳晟
时长 13:06 大小 6.01M

在上一篇文章中，我用 Selenium 2.0 实现了我们的第一个 GUI 自动化测试用例，在你感觉神奇的同时，是否也隐隐感到一丝丝的担忧呢？比如，测试脚本中既有测试数据又有测试操作，所有操作都集中在一个脚本中等等。

那么，今天我就通过介绍 GUI 测试中两个非常重要的概念：测试脚本和数据的解耦，以及页面对象（Page Object）模型，带你看看如何优化这个测试用例。

测试脚本和数据的解耦

我在前面的文章中，和你分享过 GUI 自动化测试适用的场景，它尤其适用于需要回归测试页面功能的场景。那么，你现在已经掌握了一些基本的 GUI 自动化测试用例的实现方法，

是不是正摩拳擦掌准备批量开发 GUI 自动化脚本，把自己从简单、重复的 GUI 界面操作中解放出来呢？

但是，你很快就会发现，如果在测试脚本中硬编码（hardcode）测试数据的话，测试脚本灵活性会非常低。而且，对于那些具有相同页面操作，而只是测试输入数据不同的用例来说，就会存在大量重复的代码。

举个最简单的例子，上一篇文章中实现的百度搜索的测试用例，当时用例中搜索的关键词是“极客时间”，假设我们还需要测试搜索关键词是“极客邦”和“InfoQ”的场景，如果不做任何处理，那我们就可能需要将之前的代码复制 3 份，每份代码的主体完全一致，只是其中的搜索关键词和断言（Assert）的预期结果不同。

显然，这样的做法是低效的。

更糟糕的是，界面有任何的变更需要修改自动化脚本时，你之前复制出来的三个脚本都需要做相应的修改。比如，搜索输入框的名字发生了变化，你就需要修改所有脚本中 findElement 方法的 by.name 属性。

而这里只有三个脚本还好，如果有 30 个或者更多的脚本呢，你会发现脚本的维护成本实在是太高了。那么，这种情况应该怎么处理呢？

相信你现在已经想到了，把测试数据和测试脚本分离。也就是说测试脚本只有一份，其中需要输入数据的地方会用变量来代替，然后把测试输入数据单独放在一个文件中。这个存放测试输入数据的文件，通常是表格的形式，也就是最常见的 CSV 文件。

然后，在测试脚本中通过 data provider 去 CSV 文件中读取一行数据，赋值给相应的变量，执行测试用例。接着再去 CSV 文件中读取下一行数据，读取完所有的数据后，测试结束。CSV 文件中有几行数据，测试用例就会被执行几次。具体流程如图 1 所示。

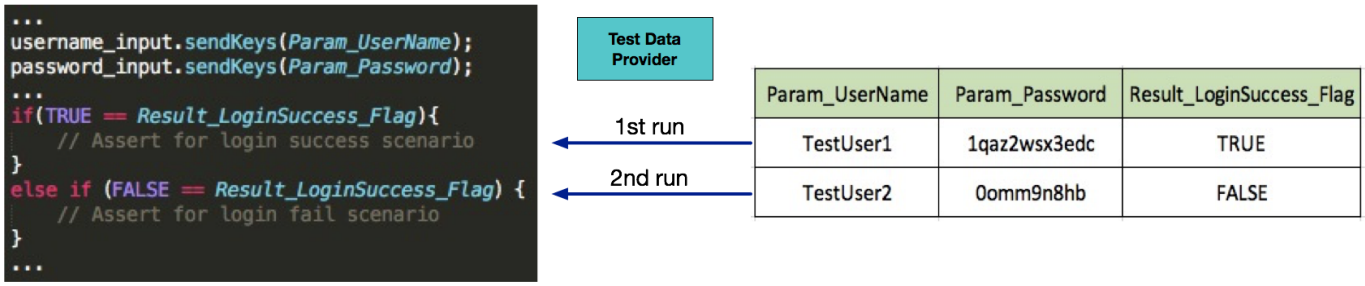

图 1 数据驱动测试的基本概念

这也就是典型的数据驱动（Data-driven）测试了。

1. **数据驱动很好地解决了大量重复脚本的问题，实现了“测试脚本和数据的解耦”。** 目前几乎所有成熟的自动化测试工具和框架，都支持数据驱动的测试，而且除了支持 CSV 这种最常见的数据源外，还支持 xls 文件、JSON 文件，YAML 文件，甚至还有直接以数据库中的表作为数据源的，比如 QTP 就支持以数据库中的表作为数据驱动的数据源。
2. **数据驱动测试的数据文件中不仅可以包含测试输入数据，还可以包含测试验证结果数据，甚至可以包含测试逻辑分支的控制变量。** 图 1 中的“Result_LoginSuccess_Flag”变量其实就是用户分支控制变量。
3. **数据驱动测试的思想不仅适用于 GUI 测试，还可以用于 API 测试、接口测试、单元测试等。** 所以，很多 API 测试工具（比如 SoapUI），以及单元测试框架都支持数据驱动测试，它们往往都是通过 Test Data Provider 模块将外部测试数据源逐条“喂”给测试脚本。

页面对象（Page Object）模型

为了让你了解“页面对象（Page Object）模型”这个概念的来龙去脉，并能够深入理解这个概念的核心思想，我会先从早期的 GUI 自动化测试开始讲起。

早期的 GUI 自动化测试脚本，无论是用开源的 Selenium 开发，还是用商用的 QTP（Quick Test Professional，现在已经改名为 Unified Functional Testing）开发，脚本通常是由一系列的页面控件的顺序操作组成的，如图 2 所示的伪代码展示了一个典型的早期 GUI 测试脚本的结构。

```
1  findElementByName("username").input("testuser001");
2  findElementByName("password").input("password");
3  findElementByName("login_ok_button").click();
4  wait(3000);
5  findElementByName("book_homepage").click();
6  wait(3000);
7  findElementByName("bookname_search_field").input("book name");
8  findElementByName("search_button").click();
9  wait(3000);
10 assert(...);
11 findElementByName("logout_button").click();
12 findElementByName("logout_ok_button").click();
```

图 2 早期的 GUI 测试脚本伪代码示例

我先来简单介绍一下这个脚本实现的功能。

第 1-4 行，输入用户名和密码并点击“登录”按钮，登录完成后页面将跳转至新页面；

第 5 行，在新页面找到“图书”链接，然后点击链接跳转至图书的页面；

第 7-10 行，在图书搜索框输入需要查找的书名，点击“搜索”按钮，然后通过 assert 验证搜索结果；

第 11-12 行，用户登出。

看完这段伪代码，你是不是觉得脚本有点像操作级别的“流水账”，而且可读性也比较差，这主要体现在以下几个方面：

脚本逻辑层次不够清晰，属于 All-in-one 的风格，既有页面元素的定位查找，又有对元素的操作。

脚本的可读性差。 为了方便你理解，示例中的代码用了比较直观的 findElementByName，你可以很方便地从 name 的取值，比如“username”和“password”，猜出脚本所执行的操作。
但在实际代码中，很多元素的定位都会采用 Xpath、ID 等方法，此时你就很难从代码中直观看出到底脚本在操作哪个控件了。也就是说代码的可读性会更差，带来的直接后果就是后期脚本的维护难度增大。
有些公司自动化测试脚本的开发和维护是两拨人，脚本开发并调试完以后，开发人员就会把脚本移交给自动化测试执行团队使用并维护，这种情况下脚本的可读性就至关重要了。但即使是同一拨人维护，一段时间后，当时的开发人员也会遗忘某些甚至是大部分的开发步骤。

由于脚本的每一行都直接描述各个页面上的元素操作，你很难一眼看出脚本更高层的业务测试流程。 比如图 2 的业务测试流程其实就三大步：用户登录、搜索书籍和用户登出，但是通过阅读代码很难一下看出来。

通用步骤会在大量测试脚本中重复出现。 脚本中的某些操作集合在业务上是属于通用步骤，比如上面伪代码的第 1-4 行完成的是用户登录操作，第 11-12 行完成的是用户的登出操作。

这些通用的操作，会在其他测试用例的脚本中被多次重复。无论操作发生变动，还是页面控件的定位发生变化时，都需要同时修改大量的脚本。

其实，我上面说到的这四点正是早期 GUI 自动化测试的主要问题，这也是我一直说“开发几个 GUI 自动化测试玩玩会觉得很高效，但是当你开发成百上千个 GUI 自动化测试的时

候，你会很痛苦”的本质含义。

那怎么解决这个问题呢？你可能已经想到了软件设计中模块化设计的思想。

没错，就是利用模块化思想，把一些通用的操作集合打包成一个个名字有意义的函数，然后 GUI 自动化脚本直接去调用这些操作函数来构成整个测试用例，这样 GUI 自动化测试脚本就从原本的“流水账”过渡到了“可重用脚本片段”。

如图 3 所示，就是利用了模块化思想的伪代码。

```
1  testcase_001()
2  {
3      login("testuser001", "password");
4      search("bookname");
5      logout();
6  }
7
8  login(username,password)
9  {
10     findElementByName("username").input(username);
11     findElementByName("password").input(password);
12     findElementByName("login_ok_button").click();
13     wait(3000);
14 }
15
16 search(bookname)
17 {
18     findElementByName("book_homepage").click();
19     wait(3000);
20     findElementByName("bookname_search_field").input(bookname);
21     findElementByName("search_button").click();
22     wait(3000);
23     assert(...);
24 }
25
26 logout()
27 {
28     findElementByName("logout_button").click();
29     findElementByName("logout_ok_button").click();
30 }
```

图 3 基于模块化的 GUI 测试用例伪代码示例

第 1-6 行就是测试用例，非常简单直接，一眼就可以看出测试用例具体在执行什么操作，而各个操作函数的具体内部实现还是之前那些“流水账”。当然这里对于测试输入数据完全可以采用测试驱动方法，这里为了直观我就直接硬编码了测试示例数据。

实际工程应用中，第 1-6 行的测试用例和第 8-30 行的操作函数通常不会放在一个文件中，因为操作函数往往会被很多测试用例共享。这种模块化的设计思想，带来的好处包括：

1. **解决了脚本可读性差的问题，脚本的逻辑层次也更清晰了；**

2. **解决了通用步骤会在大量测试脚本中重复出现的问题，** 现在操作函数可以被多个测试用例共享，当某个步骤的操作或者界面控件发生变化时，只要一次性修改相关的操作函数就可以了，而不需要去每个测试用例中逐个修改。

但是，这样的设计并没有完全解决早期 GUI 自动化测试的主要问题，比如每个操作函数内部的脚本可读性问题依然存在，而且还引入了新的问题，即如何把控操作函数的粒度，以及如何衔接两个操作函数之间的页面。

关于这两个新引入的问题，我会在后面的文章中为你详细阐述。我先来跟你聊聊，怎么解决早期 GUI 自动化测试的“可读性差、难以维护”问题。

现在，操作函数的内部实现还只是停留在“既有页面元素的定位查找，又有对元素的操作”的阶段，当业务操作本身比较复杂或者需要跨多个页面时，“可读性差、难以维护”的问题就会暴露得更加明显了。

那么，有什么更好的办法来解决这个问题吗？答案就是，我要分享的 GUI 自动化测试的第二个概念：页面对象（Page Object）模型。

页面对象模型的核心理念是，以页面（Web Page 或者 Native App Page）为单位来封装页面上的控件以及控件的部分操作。而测试用例，更确切地说是操作函数，基于页面封装对象来完成具体的界面操作，最典型的模式
是“XXXPage.YYYComponent.ZZZOperation”。

基于这个思想，上述用例的伪代码可以进化成如图 4 所示的结构。这里，我只给出了 login 函数的伪代码，建议你按照这种思路，自己去实现一下 search 和 logout 的代码，这样可以帮你更好的体会页面对象模型带来的变化。

```
Class loginPage{
    username_input = findElementByName("username");
    password_input = findElementByName("password");
    login_ok_button = findElementByName("login_ok_button");
    login_cancel_button = findElementByName("login_cancel_button");
}


login(username,password)
{
    loginPage.username_input.input(username);
    loginPage.password_input.input(password);
    loginPage.login_ok_button.click();
}
```

图 4 基于页面对象模型的伪代码示例

通过这样的代码结构，你可以清楚地看到是在什么页面执行什么操作，代码的可读性以及可维护性大幅度提高，也可以更容易地将具体的测试步骤转换成测试脚本。

总结

今天我给你讲了什么是数据驱动的测试，让你明白了“测试脚本和数据解耦”的实现方式以及应用场景。接着从 GUI 自动化测试历史发展演变的角度引出了 GUI 测试中的“页面对象模型”的概念。

“测试脚本和数据解耦”的本质是实现了数据驱动的测试，让操作相同但是数据不同的测试可以通过同一套自动化测试脚本来实现，只是在每次测试执行时提供不同的测试输入数据。

“页面对象模型”的核心理念是，以页面为单位来封装页面上的控件以及控件的部分操作。而测试用例使用页面对象来完成具体的界面操作。

希望这篇文章，可以让你更清楚地认识 GUI 自动化测试用例的逻辑以及结构。同时，你可能已经发现，这篇文章的内容并不是局限在某个 GUI 自动化测试框架上，你可以把这些设计思想灵活地运用其他 GUI 自动化测试项目中，这也是我希望达到的“授人以鱼，不如授人以渔”。

思考题

我在文中有这样一段描述：页面对象模型的核心理念是，以页面为单位来封装页面上的控件以及控件的部分操作。但是，现在业界对“是否应该在页面对象模型中封装控件的操作”一

直有不同的看法。

有些观点认为，可以在页面对象模型中封装页面控件的操作；而有些观点则认为，页面对象模型只封装控件，而操作应该再做一层额外的封装。

你更认同哪种观点呢，说说你的理由吧。

欢迎你给我留言。

上一篇　12 | 从0到1：你的第一个GUI自动化测试

下一篇　14 | 更接近业务的抽象：让自动化测试脚本更好地描述业务

精选留言 (34)

写留言

康美之心 ... 置顶

2018-07-29

8

觉得data provide只是从行为操作上分离了数据的提供方式，没有从根本上解决自动化测试中测试数据本身的稳定性、快速响应变化、数据丢失、数据被修改的这些难点和blocker，比如生产数据库里的数据没半边会导入并refresh测试环境数据库，之前cases里使用的数据都没有了；比如几个小组在一个系统里使用同一个测试环境数据库，A组正在用的测试数据B组也正在用，B组还要把数据改变一下再用，或者B组用完后测试数据已经发...

展开

作者回复: 哈哈，你说的完全在点子上，数据驱动不是用来解决你说的这些问题的，你遇到的这一系列问题我后面会介绍一个比较完美的方案，基本可以统一解决所有这些问题，我会在后买专门讲测试数据的时候详细展开，一定让你有豁然开朗的我感觉，因为我们也是一路走过来的

红娟 置顶

2018-07-27

2

首先 周三的例子通过python+seleniun3实现了，并且还开始在项目中开始了脚本编写。有点小小的成就感

其次，今天课程的关键词 数据解偶，模块化，页面对象

数据解偶和模块化我是深有体会的，之前测试项目中有用到自动化，因为是嵌入式设备，所以大多数都是自己写脚本，并没有现成的平台可以用。所以经历很多摸索。一开...

展开

作者回复: 非常感谢高质量的留言👍，页面对象你可以看文章中最后那张截图，页面对象就是把页面元素的定位集中放在一起。

Cynthia🌸...

2018-07-28

6

我们之前采用过这样一种设计：

输入数据和要操作的页面元素，都作为数据在外部进行存储，根据不同业务，以用例的形式进行组织并按行存入数据库，一条用例对应有一行记录页面操作，另一行记录输入数据。

然后脚本中封装了一些方法，使得操作页面元素的写法更加清晰可读。这样在数据中就...

展开

李真真

2018-07-27

5

我一直认为api测试就是接口测试，但是看老师写的貌似有不一样，请问具体有什么区别呢?

作者回复: 接口测试的含义范围一般更广，api测试是接口测试的一种，接口测试还包括传统软件集成测试中的模块之间的接口

ll
2018-07-27

5

更倾向于 页面模型只封装控件，这种方法其实也是模块化思想的深一层应用，控件是页面的更小单元，且一个产品或项目中使用的控件种类有限，但是页面却很多，每个页面控件组合方式又不同，在项目中两种方法多次实践后，将控件单独封装后，页面操作又节约大量的开发和维护成本。

作者回复: 很棒的逻辑分析

王征
2018-07-27

2

我同意对操作进行封装的观点，GUI自动化的其中一个问题就是对页面元素的定位操作耗费很多精力去维护，页面元素的多变性也是导致自动化不稳定的原因之一，对操作进行封装能够减少维护脚本的成本，我觉得还是值得投入的

展开

阿廉
2018-07-27

2

个人感觉分开好一些
提高了粒度
增加了灵活的使用性
（入行新手一个月 仅仅是个人感觉）

作者回复: 嗯嗯，这个目前的确没有标准答案，各自都有自己的优缺点

sylan215
2019-02-13

1

1.数据驱动这个概念太重要了，从上层设计的角度看，数据分离是必须的，但有时候项目时间紧急，为了尽快满足实现，很多人就忽略了这些上层设计，只顾眼前利益，结果就影响了长远利益；

2.页面对象模型，这个名词之前就听说了，但是自己在写 selenium 时并没有进行充分的...

展开

Nick
2018-10-11

1

你这里所说的页面对象, 应该就是类似MVVM的ViewModel, ViewModel是应该包含页面的操作

图•美克尔
2018-07-28

1

争论这个也没啥意思，如果代码写出来是给人看给人维护的关键是代码层级要清晰，注释要到位，结构要合理。其他不影响代码性能功能和使用效率的怎么组织都行啊……

展开

潘达
2018-07-27

1

数据驱动测试的思想不适用于 GUI测试
这个是想说“不仅”适用于GUI吧

展开

作者回复: 你完全对，文中有提到，api测试，单元测试其实都是适用的

Geek_c166...
2019-04-28

我们之前写代码，数据放在单独地文件中，元素定位放在yaml文件中，实现数据和代码分离，不管数据还是页面ui变化，直接修改数据文件和yaml文件就可以了，节省维护成本，另外，我们会对共用功能进行封装，对不同页面进行封装，减少重复代码的使用，提高代码的质量，测试用例直接调用就可以了

口水窝

2019-03-25

更倾向于页面封装于控件，而操作在做一次封装，这样页面和操作完全分开，不同的对象去操作，更透明！

小窝子。(...

2019-03-12

个人觉得可以让人工智能介入。利用图像识别。未来这方面是有潜力的。

展开

wyy

2019-03-01

老师，你说的通过web url,自动生成页面上所有控件的定位信息，这个能具体解释下实现原理吗

年轻人的瞎...

2018-12-09

我们主要做后台服务，大部分都是对接口进行自动化，当多个接口一起运行时，有缓存的影响等，容易耦合，请问后台接口自动化如何解耦？

作者回复: 这个取决于你的后台架构以及相应的缓冲机制，没法一概而论

年轻人的瞎...

2018-11-06

现在在处理API自动化，想了解一下如何高效用例，减少复用

展开

_山顶

2018-10-30

针对老师最后提出的问题，我认为可以在页面对象模型中封装页面控件的操作，虽然我刚

接触测试不久，但是对于面向对象编程的概念之前有所了解，所以也就更倾向于在对象模型中同时封装页面控件的操作，我认为这样的好处是我们在获得这个对象模型的同时也获得了该页面控件的操作权，这样对于自动化执行各种控件的操作来说，是比较便捷的，可以提高自动化执行的效率

展开

johnny

2018-10-29

我读完后，一套测试脚本可以抽象出三个类。
测试用例类：定义一个测试用例；
操作函数类：放测试用例调用的操作函数；
页面类（page object）：放置被封装的页面控件；

johnny

2018-10-29

写的太好了。

展开