6/8/22

6/11/22, 6/13🡪6/19

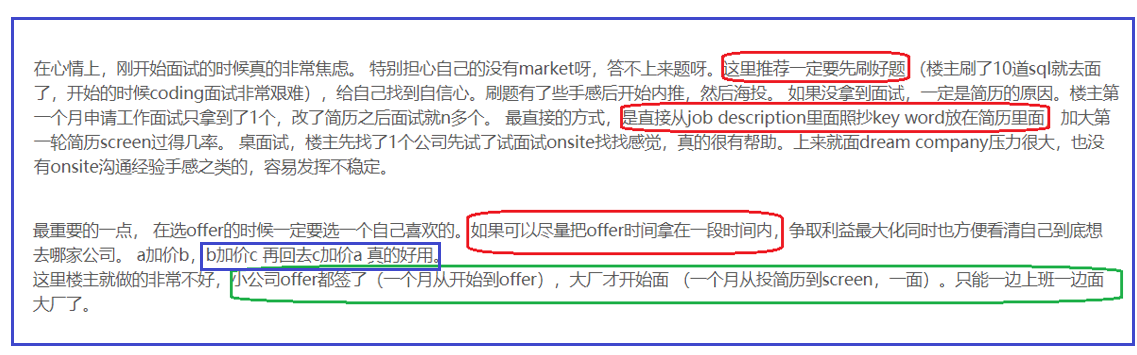
7/12/22, 7/27/22, 7/31, 8/1, 8/8, 9/7

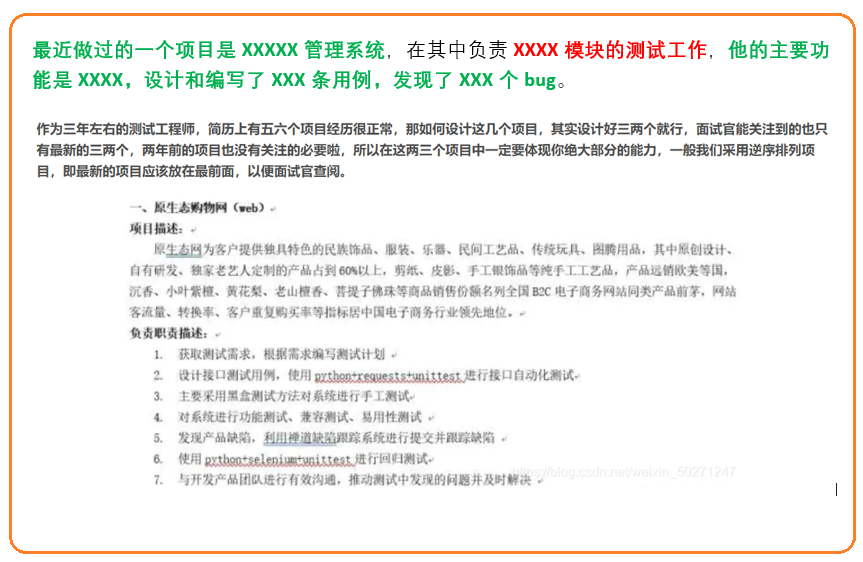
[roblox.com](https://www.roblox.com/), greghefflyakabubby, 6-!5LwUYQHd9L9\*

1. Selenium – python
2. Python re
3. Python request
4. Json, csv file handling
5. Locust performance test
6. Chrome network
7. Resu **keypoints**.
8. Python Mysql
9. Curl
10. Token 管理
11. **API 测试过程中的第三方依赖**
12. **beautifulSoup**
13. **jsonpath**

9/9/22

Focus: summary and specific inv questions.



1.  Jira is Java based, also offer restAPI. Confirmed.
2. Find out frondend scheme
3. Setup of backend and frontend

* Understand relationship of restAPI and java based backend
* <https://developer.atlassian.com/server/jira/platform/architecture-overview/>
* Jira is **a web application written in Java**. It is deployed as a **standard**[**Java WAR**](https://docs.oracle.com/javaee/7/tutorial/packaging003.htm)**file** into a java Servlet Container such as [Tomcat](http://tomcat.apache.org/)
* <https://developer.atlassian.com/server/jira/platform/rest-apis/>
* Welcome to the Jira Server platform REST API reference. You can use this REST API to **build apps for Jira**, develop integrations between Jira and other applications, or **script interactions with Jira**.
* The Jira **REST API enables you to interact with Jira programmatically**. Use this API to [build apps](https://developer.atlassian.com/cloud/jira/platform/integrating-with-jira-cloud/), script interactions with Jira, or develop any other type of integration. This page documents the REST resources available in Jira Cloud, including the HTTP response codes and example requests and responses
* The Jira REST APIs are used to interact with the Jira Server applications remotely, for example, when configuring webhooks. The Jira Server platform provides the REST API for common features, like issues and workflows.
* To get started, read the reference documentation: [Jira Server platform REST API](https://docs.atlassian.com/software/jira/docs/api/REST/latest/)
* Generating a REST client for Jira Cloud
* https://community.atlassian.com/t5/Jira-articles/Generating-a-REST-client-for-Jira-Cloud/ba-p/1307133
* Jira’s frontend implementation
* Tomcat, maven, webserver of Jira
* CI/CD of jira pipeline
* Database: mysql or postsqlgre

**3/19/22**

1. My stack : Java-Application+restAPI + frontend (jquery/react)
2. Web/app server: Apache Tomcat 9.0
3. Java EE:

|  |  |
| --- | --- |
| **Jakarta EE 8** | 2019-09-10 |
| Jakarta EE 10 | 2022-03-31 |

1. CI/CD pipeline or freestyle
2. Performance test
3. **UI** test separated from API test.

2/26/22

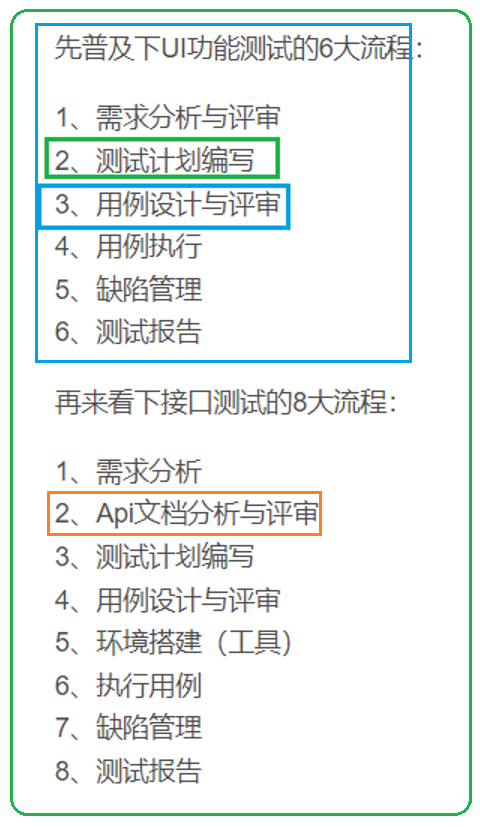
1. Understand Privat vs Public Cloud

Can I consider LP server as private Cloud

**1/27/22**

制定测试计划，确定测试范围和测试策略，一般包括以下几个部分：

1. **功能性测试;**
2. **界面测试**
3. **性能测试**
4. **数据库测试**
5. **安全性测试**
6. **兼容性测试.**
7. **API TEST**

****

5\_26-22

1. Resu relevant
2. Automation platform
3. **My stack** : Java-Application+restAPI + frontend (jquery/react)

* Web/app server: **apache Tomcat 9.0**
* Java EE:

|  |  |
| --- | --- |
| **Jakarta EE 8** | 2019-09-10 |
| Jakarta EE 10 | 2022-03-31 |

* UI: HTML5, CSS, Ajax/js

### Linux Ubuntu Ubuntu 20.04 LTS (Focal Fossa), 2020

* Python 3.8.2

1. **Test scenario**
2. 登录注册
3. Test/
4. Device/
5. testFlow
6. report/gen
7. **功能测试阶段**。主要负责编写测试计划、测试用例、部署禅道BUG管理系统，进行功能测试。
8. 首先，我们将系统分为了test、device、testFlow、report等4个模块。
9. 其次，使用等价类划分和边界值分析法相结合，针对每个模块设计测试用例。
10. **UI层自动化测试。**使用PO设计模式，工具是[Selenium](https://so.csdn.net/so/search?q=Selenium&spm=1001.2101.3001.7020" \t "_blank)+pytest+Jenkins。
11. **UI test**
12. **UI层自动化测试。**使用PO设计模式，工具是[Selenium](https://so.csdn.net/so/search?q=Selenium&spm=1001.2101.3001.7020" \t "_blank)+py test+Jenkins。

#### 目的

这个阶段，是在我们项目需求已经明确，版本已经稳定的情况下开始的，主要**考虑了几个方面**：

* ***UI层在多平台、多浏览器下运行结果存在不同。也就是需要我们在不同平台、浏览器下运行相同的测试案例，大量的重复劳作力***
* Linux Ubuntu 20, windows 10.
* ChromeDriver: chromedriver.exe, [**104.0.5112.20**](https://chromedriver.storage.googleapis.com/index.html?path=104.0.5112.20/)



## FixefoxDriver: geckodriver.exe, 0.31.0 (2022-04-11)

* driver = webdriver.Edge(); Version 103.0.1264.77 (Official build) (64-bit)

<https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>

**msedge**driver.exe

msedgedriver (linux)

* 其次，我们项目因为前期设计不够严谨、版本部署不够规范，会出现BUG重复出现的情况，也就是需要我们每日构建后进行回归测试。

#### 2.2 内容 **PO**

* 在设计UI层[自动化测试](https://so.csdn.net/so/search?q=%E8%87%AA%E5%8A%A8%E5%8C%96%E6%B5%8B%E8%AF%95&spm=1001.2101.3001.7020" \t "_blank)用例的时候，使用的是**PO**设计模式，也就是把**每一个页面所需要操作的元素(element,and its locator)**和步骤(element operation)都封装在一个页面类中。然后 Selenium+Unit test搭建四层框架——实现**test-data**、**test-script**、**业务逻辑分离**(element and its operation)（关键字驱动）

#### 基础层（**BasePage**）

* 设计一个基本的Page类，所有页面皆继承该类。提供了一个页面需要实现的基本功能及公共方法。
* Brower related functions: open\_page,

#### 业务逻辑层（**Pages**）：

* 按照PO设计模式，将每个页面抽象为一个类，放在Pages包里面，每个页面继承Basepage，可调用Data层数据，内容包括：
* 该页面所有的操作对象element-locator
* Element-operations

#### 数据层（**Data**）

* 该层存放相关数据，例如：用户数据和密码。在测试用例可通过调用数据层的数据来进行操作。
* .csv or .json files

#### 测试用例层（Testcases）

* 每一个测试用例testcase都对应Pages里面的一个页面，继承unnitest.TestCase类。
* 通过**调用对应页面类 element-locator and method的方法**，**数据层的数据**、**增加断言**(assert)来验证功能的正确性。

此外通过Jenkins自动执行测试、代码质量检测和部署到测试服务器、部署到生产服务器上

### 2.3 自动化测试执行策略——三个阶段

使用Jenkins持续集成工具来执行测试脚本和部署，主要设置了三个任务：

* **tm\_test：**用于执行自动化测试脚本，检测代码质量
* **tm\_staging\_deploy**：用于在测试服务器上部署代码
* **tm\_deploy**：用于在生产服务器上部署

我们将**测试分为三个阶段**：

1. 开发新的需求时，创建**branch devN**。当在这个分支中，需求开发完成或者Bug修复，

* After unittest, dev check in code and trigger QA test on Jenkins.
* 就配合测试人员利用 pytest框架进行以及功能测试。
* 通过测试后，合并到master上。And run nightly regression.

1. 当master有变动，则触发tm\_test(night regression test) 任务，执行自动化测试脚本和代码质量检测。如果通过则自动触发tm\_staging\_deploy，部署到测试服务器，如果没有通过，自动化测试脚本会将Bug截图发送给测试人员。
2. 登陆生产服务器上，对网站进行功能测试。如果通过测试，则手动触发tm\_deploy，部署到生产服务器。如果没有通过，在禅道管理系统上把bug指派给相应模块的开发人员。
3. **Setup**
4. Client: device and tester

* Install/run **test\_program** as command line tool: IQfactrun\_console
* Install an **agent**
* Monitor device running status of test\_program and fail/pass statistics.
* Set up agent with user credentials
* Agent communicate with both server and IQfactrun\_console test program: get status from test program and updated server.
* Upload/download test\_results/logs, test\_flows to server
* Upload/download csv/json for reporting
* Device and test status is conveyed between **agent and server**.

1. lp server on **premise**.
2. API/UI offers

* **dashboard as a web interface**
* Tester-box and devices are registered and setup through dash-board

1. Manage **user**
2. Manage and create **workflow** for connectivity testing
3. Manage **test**/ (need user to fill up)

* Prev **test status**.
* **test logs** through upload;
* **report**/statistics engine

1. Manage **device status**

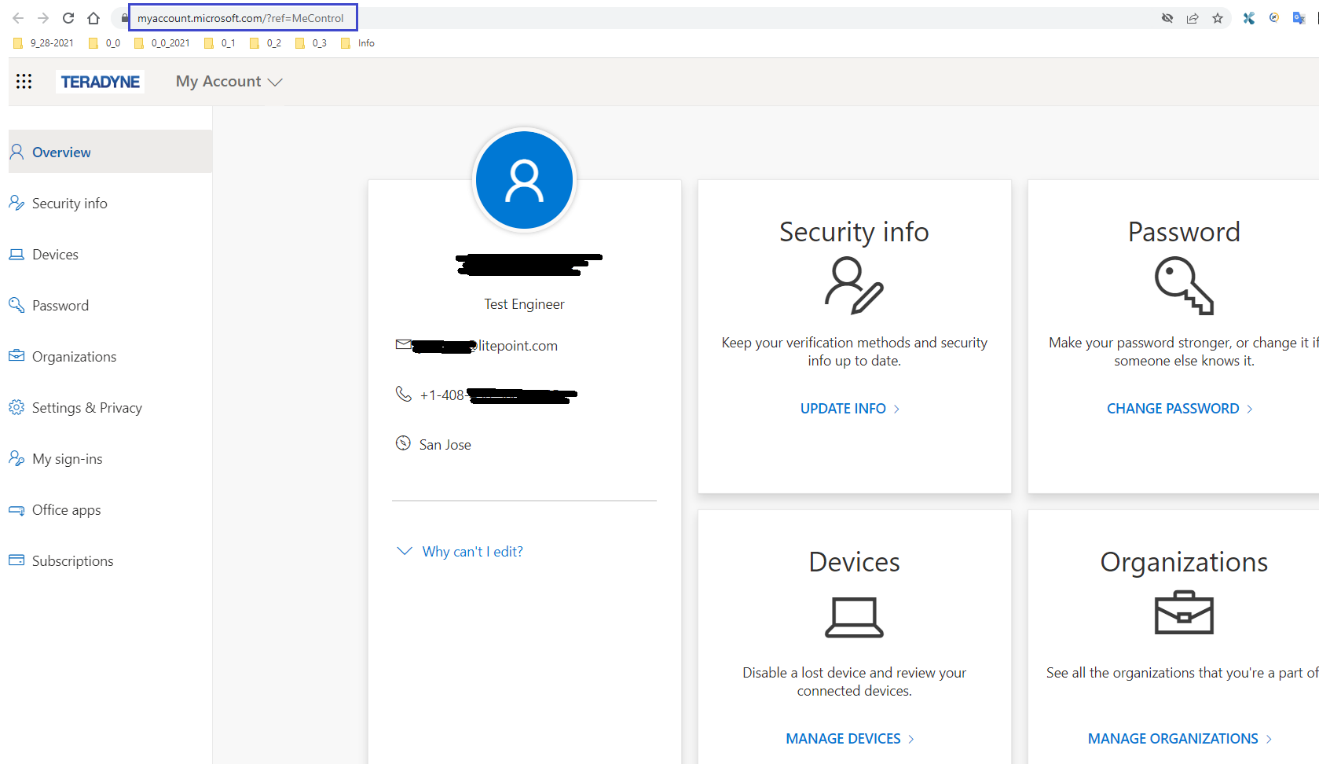
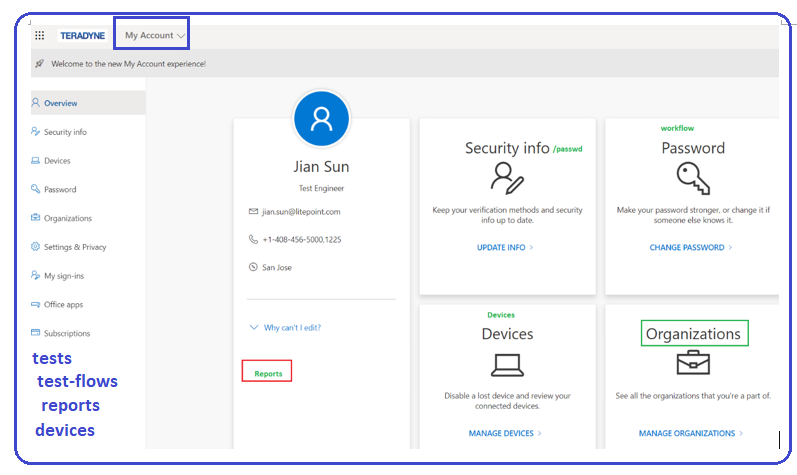
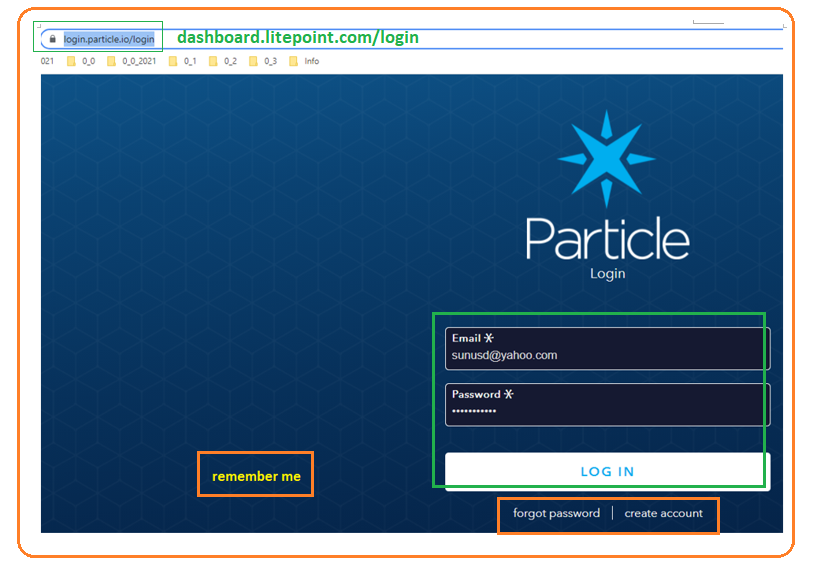
* One – click to start the test; and collect data –NO NEED for it.
* **How device, dashboard and server interact with each other**
* **Install an agent on device PC**
* Agent exchange status related info with server with https in a vpn secured tunnel.
* Agent communicates with server to updated device status.
* **Dashboard manage device info; test related logs, results can be uploaded from dashboard to server;**
* **How about workflow run in device:**
* Download workflow from server through dashboard
* test is run by test program (IQfact+)
* **Dashboard gets workflow from server, download to test PC, run test program.**
* Once test program starts, agent will keep track of status
* UI
* Dashboard

1. **CI/CD** **pipeline** or freestyle

* Daily CI/CD flowchart
* Dev check-in new code to branch.
* Unit test is done before checking in.
* CI/CD build and deploy to webserver
* Kick-off UI/API QA test. (run on CI/CD pc.
* Check out most recently test code.
* Test-code and Dev-code in separate code base.
* Report/summary sent through email
* Issues will be verified and addressed.
* Identified origin of bug; file jira if bug is code related
* Night regression test
* Test is running on Ubuntu env.
* Support windows and run test on weekly regression.
* Cross-browser
* Google Chrome: chromedriver.exe
* Widows edge
* Firefox: geckodriver.exe

1. **Summary**
2. **UI-Dashboard:** [**https://dashboard.litepoint.com/**](https://dashboard.litepoint.com/)

****

****

1. **/endpoint** Base URL: <https://server:port/api/v1>

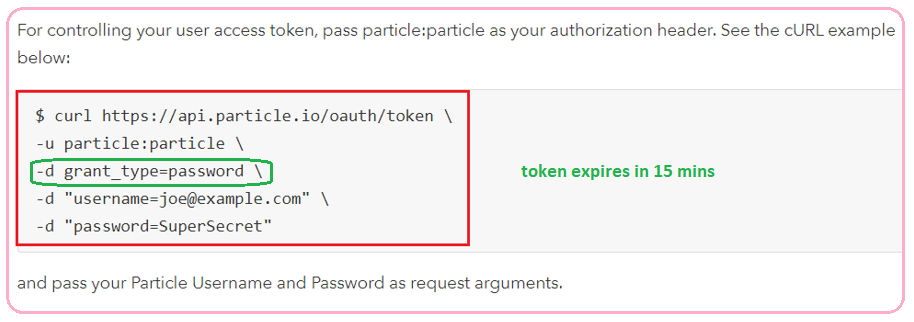
* User profile/my account
* log-in, change passwd, and user info
* user contact: org/org\_id, group/group-id,name, email address, phone,user\_id,
* Base URL: <https://server:port/api/v1>
* Basic path: /organization/{org-id}/
* **User**: /users/{user\_id}
* Organization **sys-admin issue user an organization ID and token**. Token has scope that decides user rights**; Access rights limits to user’s group**.
* User use those two to create a user account with username/email and passwd
* Post-create new user need api key (IT issue)
* Get: show users under a group
* Put/patch: edit user info
* **Device**: devices/{device\_id}
* device general info
* ip, name,id, associated tester ip
* owner-user
* os, FW
* status: running, idle, stopped
* last run test statistics: data-last-run, pass/fail
* add/delete/modify device

**Device status**

* Once iqfactrun is running, agent will update the device’s status on dash-board
* Device status auto-refresh to show real-time test status.
* **tests**: test/{test\_id}
* test/device\_id?num-test=100; pagination
* test\_name, testflow\_name, device\_id/name, user
* pass/fail statistics, test completed date
* **test\_flows**: test\_flow/{test\_flow\_id}
* list of existing testflows, date created/modified
* create/modify/delete test\_flow
* list of example testflows.
* Get: download test\_flow, json or cvs
* Post: create/upload test flow in json or cvs
* **Report**
* List of reports
* Create/delete reports
* Get: download report (pdf)
* **/token**
* **Oauth2** for end users and web/mobile app
* Authorization/oauth2/v2.0
* <https://login.microsoftonline.com/common/oauth2/v2.0>
* API access token or API key

1. How to generate and update token or key

* Get a token through UI
* UI will also show current token being used, and expired token
* Got a token through API call



1. **Summary of lp test soln**

* **Lp offers connectivity test soln for wireless devices.**
* **Our application help customer manage device, testing, analysis and reporting through UI/API.**
* **User can create end-to-end automation testing environment from collection of measurement data to analytics and reporting.**
* We offer web application **test soln** for wireless connectivity test.
* AS SW test engineer, we are focused on UI and rest API testing.

1. **Daily or routine work**
2. REST API testing
3. Front-end UI testing
4. Database query testing.
5. Performance test

* Upload/download test
* Monitor client/server: memory usage, CPU, data written to storage
* Scenario: multiple devices, multi-users.

1. Desktop

* Download and install litepoint SDK, test\_program
* Include testProgram-Iqfactrun,

1. Automation testing and framework.

* Test script automation
* Improve framework for performance and scalability.
* Develop/create new tools

1. 在哪里工作，工作内容和职位是什么，主要负责什么
2. 项目的challenge是什么，
3. 现如今感兴趣的方向
4. A couple of typical bug fixes, and new features verifications will do the intv questions.
5. **Projects**
6. **Tableau**

* <https://github.com/tableau?page=1>
* <https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api.htm>

1. **Challenge work**
2. When define common functions: fixture function or util function

* Need to customize
* If too general, function tends to be complex and error – prone.
* If too specific, ends up defining too many functions with similar purpose.
* Solution: need to consider overall structure and features, make necessary / relevant trade off.
* Need to give a solid example

1. Conftest

* Too many mixture in one conftest
* Soln: make conftest under specific folder, corresponding specific feature
* Create fixture\_test folder to save fixture files and then add plug-in to conftest.py.
* Make one overall conftest for util – like mixture, like connect db.

1. Common scenario:

* Dev says I can’t see on my setup.

Soln: use dedicated debug test station for debugging purpose. Easy to reproduce and show results to dev.

1. **你对测试最大的兴趣在哪里?为什么?**
2. **你自认为测试的优势在哪里**
3. Identify bug comes from UI or backend

# **有哪些让你印象深刻的bug**?

* Updated certain tests with a new feature to existing workflow through UI.
* After updating through UI, noticed all the tests contains new functionality.
* Initially suspect, UI doesn’t display workflow properly.
* Use API to update, status\_code shows successful with 200, with updates included. Patch didn’t include all content of workflow
* Running get, noticed all the tests have been updated, which is wrong.
* 前段时间遇到一个工单，客户反馈，只要进入订单列表界面 1~2 秒，客户端就会 Crash，订单列表界面示意如下
* 由于是客户投诉的 Bug，没有 Debug 信息，先猜测各种情况，[数组](https://so.csdn.net/so/search?q=%E6%95%B0%E7%BB%84&spm=1001.2101.3001.7020" \t "_blank)越界/后台传 nil 值/内存泄露/ KVO 赋未定义值等等；然而经过仔细分析模拟逐个排除了上述可能，仍查找不到 Crash 原因，百思不得其解。

排除了代码的问题，只有可能是数据问题了

* bug发生在服务端的算法模型层，算法的边界遗漏会导致部分用户的广告永远无法获得曝光或推荐，这就会造成这部分广告用户付了钱却没有在移动端的界面上展示他们的广告(影响会很差)，通过模拟一系列的数据测试，最终发现规律，问题出在算法模型的临界上……

### ***在使用Selenium中遇到的最大的问题****？如何解决？——测试用例的可靠性*****

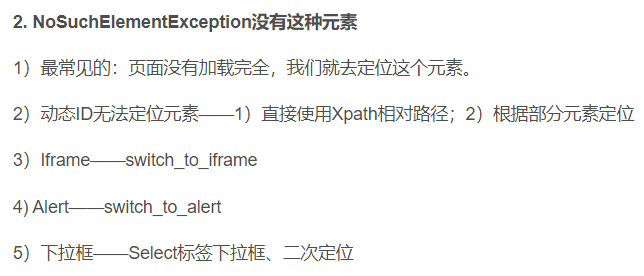
**误报**通常是我们在使用selenium的最头疼的问题，这使得很难把selenium测试用例加入到自动构建中。有些构建是必须要成功的，如果失败将会阻塞整个发布流程。

**解决方法**——重试我们的解决方案是在测试步骤和测试集中都加入重试机制。

产生误报**最大原因是**selenium在页面加载完成之前就开始请求页面资源。

**重试机制：**

利用递归封装了一个等待元素的方法。其中，设置最大等待时间为1s，轮询时间为50ms，这个方法会不断轮询，直到方法执行成功或者超过设置的最大等待时间。在我们最好的一次实践中，我们把一个测试用例的误报率从10%降低到0，并且执行时间从原先的45秒降低到33秒。

* **NoSuchElementException**
* 
* 找不到元素，脚本报“NoSuchElementException:Unable to find element”，或者"定位到了，不能操作，点击无效"

1) 首先查看自己的“属性值”是否写正确

2) 元素的标签不唯一，默认找到第一个

3)向上查看，元素是否在[frame](https://so.csdn.net/so/search?q=frame&spm=1001.2101.3001.7020" \t "_blank)或iframe框架中

4) 查看元素是否在新打开的页面中，需要切换到新窗口

5) 换其它的定位方式：id/name/class name/tag name/link text/xpath/css selector

6) 检查元素属性是否是会变动的、是否是隐藏的

7)添加等待时间sleep(),implicitly\_wait(),WebDriverWait(driver, 10, 1).until(定位的元素, messages)

8) 查看标签的属性是否有“style=’display:none’->。元素不显示。属性改为block

9) 查看标签的属性是否有‘οnclick=return false’->。取消点击。属性改为false（在百度首页，登陆的属性就是这个）

10) 针对于9和8这两种情况，我在网上搜到的结果是：执行js修改属性(这种方法我没有试过，不知道有没有效果）

js=”document.getElementById(‘title‘).style.display=’block’”

driver.execute\_script(js)

# 你在工作中遇到过印象深刻的困难是什么，你怎么克服的

# 10个前端经常遇到的问题解决方案

### *****如何提高selenium脚本的执行速度？*****

**1）优化测试用例。**

* 尽可能不用sleep、减少使用implicityWait，而使用WebDriverWait/FluentWait，这样可以优化等待时间
* 减少不必要的操作步骤。

**2）使用Selenium grid，通过testNG实现并发执行。**

说到这里，在编写测试用例的时候，一定要实现松耦合，然后再服务器允许的情况下，尽量设置多线程实现并发运行。

**3）设置等待时间、中断页面加载。**如果页面加载内容太多，我们可以查看一下加载缓慢的原因，在不影响测试的情况下，可以设置超时时间，中断页面加载。

### 

### 提高自动化脚本稳定性

首先我们要分析出不稳定的原因，然后有针对的去解决。

1）页面加载内容太多。如果页面加载内容太多，在不影响测试的情况下，我们可以设置超时时间，中断页面加载。

2）网络原因。设置等待时间，如果在响应时间内没有加载成功，则重新执行测试。

3）优化代码，减少容易冲突的函数。

4）多线程运行时，测试用例间相互影响。在并发操作时，如果用例之间的耦合性没有设计好，就会有影响。

综上所述，我们就可以用线程的方式来监控测试进程的WEB加载执行状态。

* 在页面会发生跳转时，启动一个Thread来监控进程的状况。
* 在Thread的run方法中定义一个计时器。
  + 如果计时器超时，则可以刷新页面，计时器清零；
  + 若此时刷新页面再次超时，则关闭当前浏览器进程，fail掉这个测试用例，继续执行其他的测试用例。

1. **TO be TBD**

### *****高质量自动化脚本特点*****

1. 业务和代码分离，封装性好。
2. 用例之间耦合性低，独立性强，易于扩展维护

* Check packets, google chrome

# python篇---在windows下利用pyinstaller打包成exe

1. API 测试项目

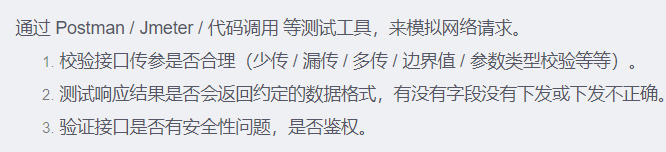
**I want to see specific examples, NOT in general**

API测试用例主要由4个部分组成，分别是：用例的基础信息、前置步骤、请求脚本以及断言

前置步骤用于为执行用例请求做前置准备，分为：前置请求、生成随机数据、前置等待3种类型。

请求脚本中包含了：选择请求方式、维护URL、维护请求头、请求参数或请求体，同时还能从对应的响应结果中提取出变量供后续的用例引用。

断言用于对用例执行后的响应结果做判断，判断请求执行后的响应结果是否满足我们的预期。若满足，则称之为：通过断言；不满足，则为：不满足断言



# **API 测试的具体实现**

<https://blog.csdn.net/weixin_34206899/article/details/93240338?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_utm_term~default-0-93240338-blog-114528699.pc_relevant_downloadblacklistv1&spm=1001.2101.3001.4242.1&utm_relevant_index=3>

1. UI 测试项目
2. Pytest automation tools/utility functions, improvement in Framework