目录 | (六十二期·上)

投稿邮箱：editor@51testing.com

测试回顾：艰难与收获并存的一年

◆ 作者：邀明月

摘要：
2020 年是入职单位的第 6 个年头，这一年经历了测试组重组，项目重心从本省转移到外省，身份从测试人员到兼职需求人员，沟通对象从项目成员到客户，常常感慨最多的一句话是“太难了!”。在高压的工作节奏中，利用碎片时间学习，通过了中级软考，获得了软件评测师资格证书。总得来说，2020 是最累的一年，也是收获最多的一年。

2020 年是入职单位的第 6 个年头，这一年经历了测试组重组，项目重心从本省转移到外省，身份从测试人员到兼职需求人员，沟通对象从项目成员到客户，常常感慨最多的一句话是“太难了!”。在高压的工作节奏中，利用碎片时间学习，通过了中级软考，获得了软件评测师资格证书。总得来说，2020 是最累的一年，也是收获最多的一年。

2020 年 4 月末，测试组组长离职了，感觉自己在工作上依靠的大树倒了。测试组剩下自己和另外一个测试人员了。2020 年 5 月末，开始着手外省项目的需求整理工作。同时兼顾本省的测试工作。2020 年 6 月中旬，另外一个测试人员也离职了，测试组只剩下了自己。本省项目的测试任务、外省项目的需求整理全部压到自己身上，身体的劳累和心理上压力积攒到了极致，情绪也受到了影响，对开发人员犯的低级错误会无法忍受，导致自己在工作群输出一些抱怨的话语。事后觉得自己好像太激动了，不该说出那些抱怨的话。有一天下班后在路上遇到了一个关系不错的开发同事，就跟他聊了一会：

我：你们是不是觉得我最近脾气特别不好，其实我也不是针对谁。

开发同事：没事，咱们在一起工作这么多年，你平时是啥样的，我们大家都知道。

我：嗯，主要是最近工作太累了，遇到太过分的问题，实在是受不了了。

开发同事：大家都知道你工作累，不会在意的。

我：瞬间崩溃大哭了起来……

当时特别感谢同事的理解，哭过之后，自己紧绷的神经瞬间放松了，压力和情绪也一下子释放了。没什么大不了了，挺挺会过去的。

紧张的工作有条不紊的进行着，同时也开始了测试人员的招聘面试工作。2020 年 6 月末，终于迎来了一名测试人员的到来。8 月初，另外一名测试人员也加入了我们，测试组重组完成，而我又多了一个新身份——测试负责人。平时除了要完成自己的测试任务及需求确认工作，还要兼顾 2 名新入职的测试小伙伴业务知识培训。每个测试小伙伴经过大概一周的业务知识的熟悉，都可以负责项目的简单测试工作，我的测试工作终于有人分担了。

此时的我开始了另外一个挑战，即与客户进行沟通。客户会向我说明业务知识，最开始跟客户沟通，会头疼，业务知识的接收有些吃力，只能是客户说，我边听边记录。 跟客户接触了一段时间后，对业务知识也熟悉了，在沟通的过程中，终于可以插上几句话了！最后项目快结束时，客户会主动给我打电话，询问和商量一些业务的处理方法，我心想客户已经对我产生了信任感。时间是最好的见证者，我从一个纯聆听者蜕变成了一个可以发表自己见解的人。

工作职责的增多和工作强度的加大，加班时间也会很多。自己的辛苦，领导也是看在眼里。有一次汇报完当天工作，领导暖心的说，最近工作辛苦了，早点回家吧，多陪陪孩子。心里瞬间充满了暖意，工作的疲劳也消散而去。有个理解并看见自己的领导也是工作中的一件幸事。因为自己的时间大部门都用在了工作上，老公会吐槽好久没好好看他一眼了，回家注意力在孩子身上，上班注意力在工作上面。真心感谢老公的体谅和对自己工作的支持。

距 2020 年下半年软考考试还有 1 个月的时候，我开始利用中午午休时间突击软考，每天看大概一小时软考讲解视频。临考试前一周，每天中午做一会历年考试真题，熟悉软考题型和解答方式。打印准考证的时候，同事开玩笑说，别去考试了，来年大家一起考。我说，不得，我要去参加考试，万一过了呢！哈哈！最后还真的考过了，可能是知识和经验的积累帮助了我。

2020 年还提升了自己对问题分析的能力，遇到 bug，会利用自己对业务知识的掌握及测试经验，主动对问题进行分析，帮助开发人员定位问题，用业务逻辑去支撑开发人员解决问题。有效的沟通也促进了跟开发人员良性的合作关系发展。

今年年初，省外的项目终于上线了。在客户交流群中，自己被各方人员同时@，一时间有些不知所措。项目经理安慰我说，没事，别慌，一个一个来。吃了这颗定心丸，开始镇定的验证问题、反馈问题、分析问题、解决问题。经过半个月的时间，新系统稳定了下来。领导夸奖说，我们还是挺牛的，这么大个新系统，这么快就使用起来了。

过去的整整一年，艰难的工作磨炼了自己的抗压能力，同时也学会了碎片时间的利用、与客户的沟通。感谢公司领导的肯定和推荐，最终被评为了公司年度优秀员工。

不经过风雨怎能见彩虹，不经过磨练怎能再成长。是挑战，也是机遇，踏踏实实地前行，就能看到收获的样子。

手机软件测试基本设置

◆作者：胡军英

应用场景：智能手机助推动了越来越多的手机应用程序，测试设备的多模型，多版本使得测试也变得更加复杂化。今天这篇对手机应用程序做自动化测试的基本环境配置文章，就让我们一起来搭建一个 emulator 测试机吧。本篇文章适合初次学习手机软件自动化测试的朋友 。我们从最基本的必要工具的安、配置、虚拟机建立开始，一步步学习搭建测试环境，最后应用一个小的测试用例学习连接 emulator，安装测试程序、定位测试元素、验证期待结果等基础知识，成功实现手机应用程序在 emulator 上的自动测试运行。

温馨提示：如各位阅读者想按着示例操作，请勿必配置以下工具：

- IDE： PyCharm Community Edition
- 语言：Python
- 测试框架： Pytest
- 服务器：Appium
- 测试设备： Android Emulator
- 示例测试程序安装包: alipay

知识重点：

1.Android Studio 安装与配置

2.Android Emulator 创建与应用

3.利用 adb 获取测试程序

4.Appium 安装与配置

5.安装、测试应用程序在 Emulator 上

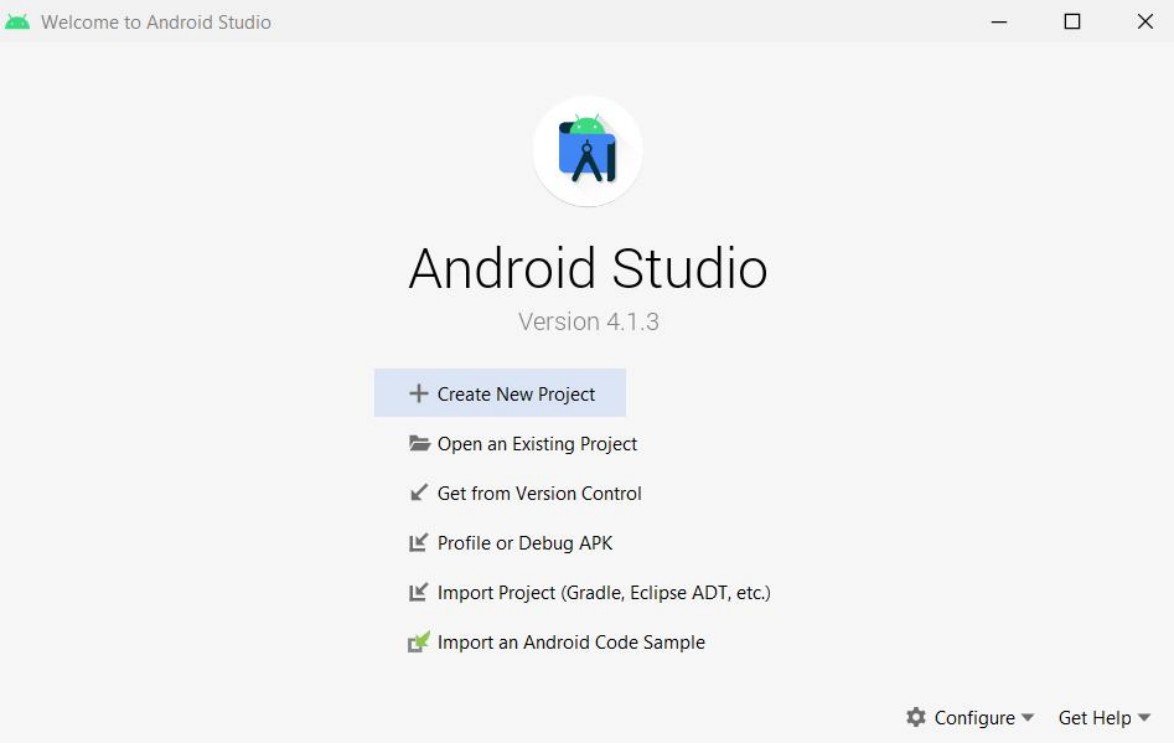
6.利用 Appium 获取测试程序元素

一、Android Studio 安装与配置

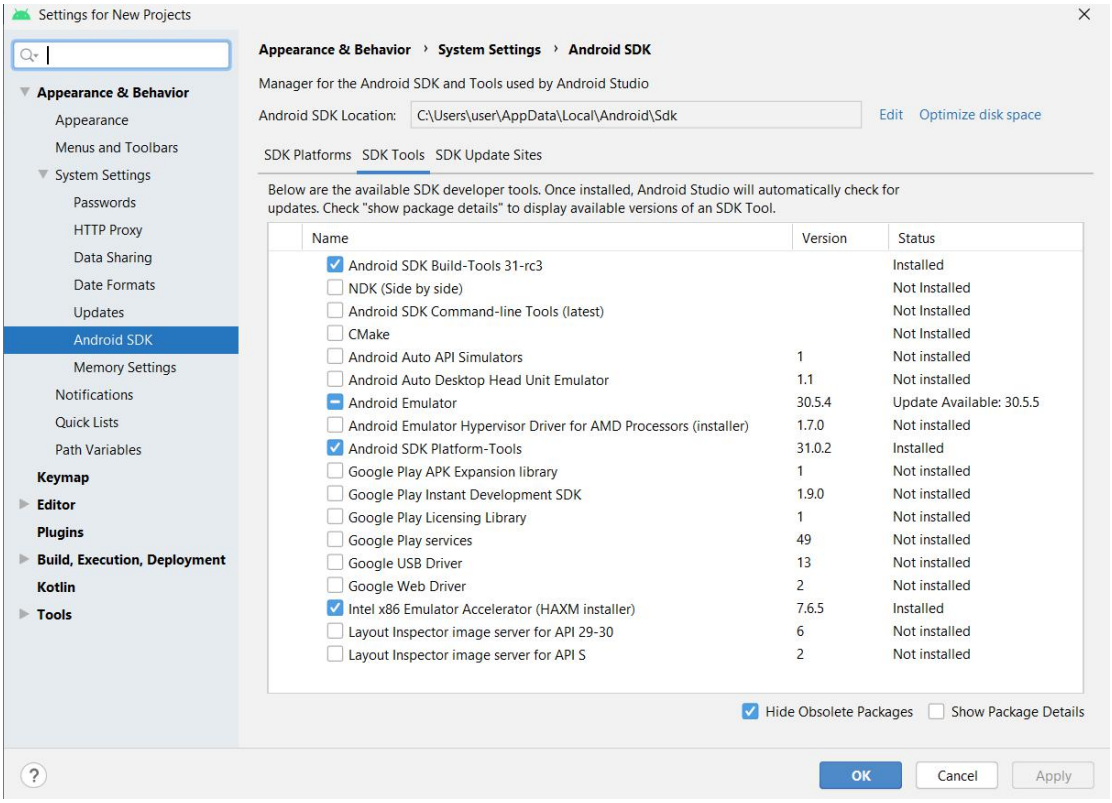
1.下载：打开 Android Studio 的官方网站（https://developer.android.com/studio），下载 Android Stuidio 的安装包。

2.安装：安装包下载以后，直接双击安装包，安装程序会自动启动。您可以选择默认选项安装，也可根据实际情况选择安装目录。

3.启动：完成安装以后，启动 Android Studio，如下面的 Welcome to Android Studio。

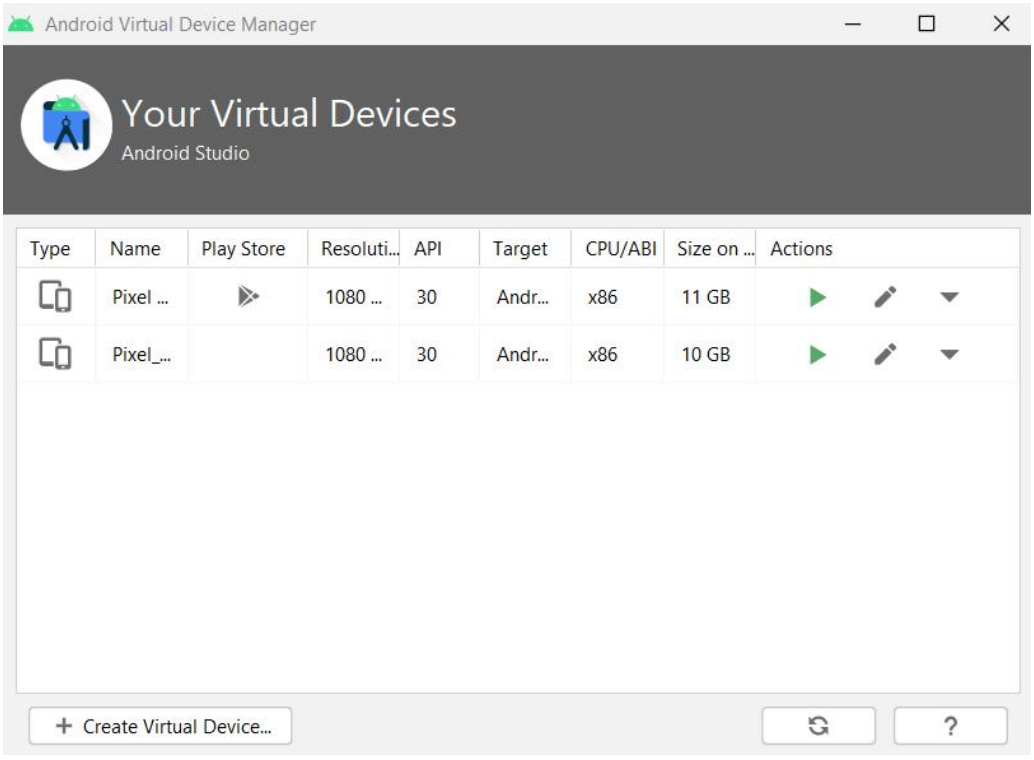


4.依次打开 Configure > SDK Manager，在打开的 Settings for New Projects 窗口下，点击 SDK Tools 确保 Android SDK Platform-Tools 已经安装。记下上面的 Android SDK Locations（C:\Users\user\AppData\Local\Android\Sdk）。确认以后，关闭窗口。

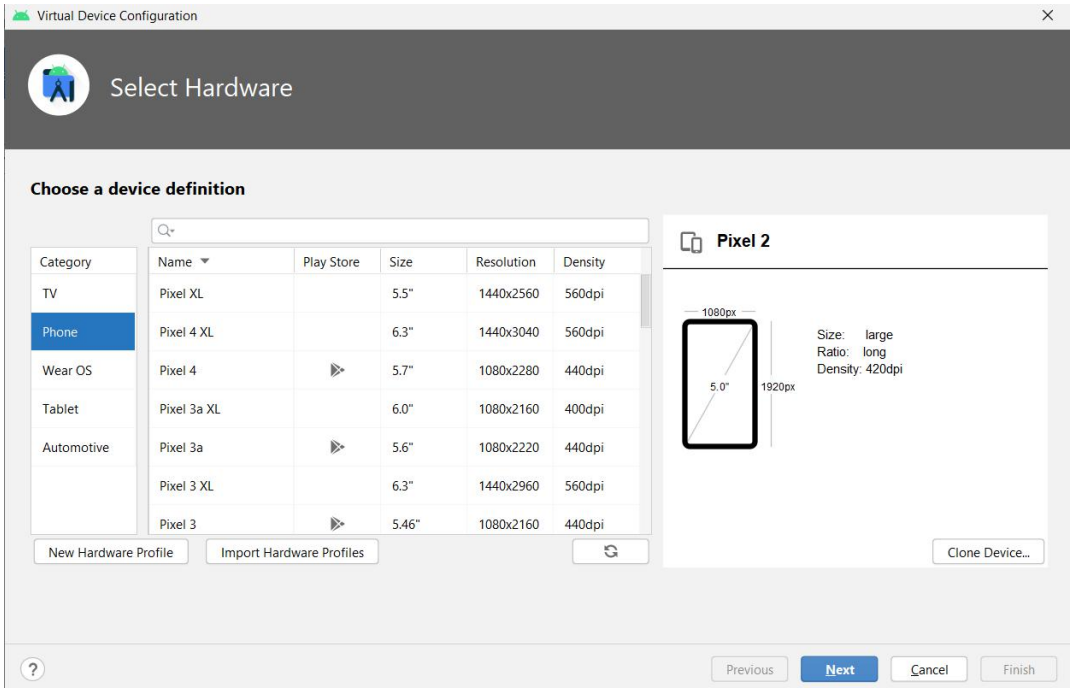


二、Android Emulator 创建与应用

1. 在 Welcome to Android Studio 窗口，依次打开 Configure > AVD Manager

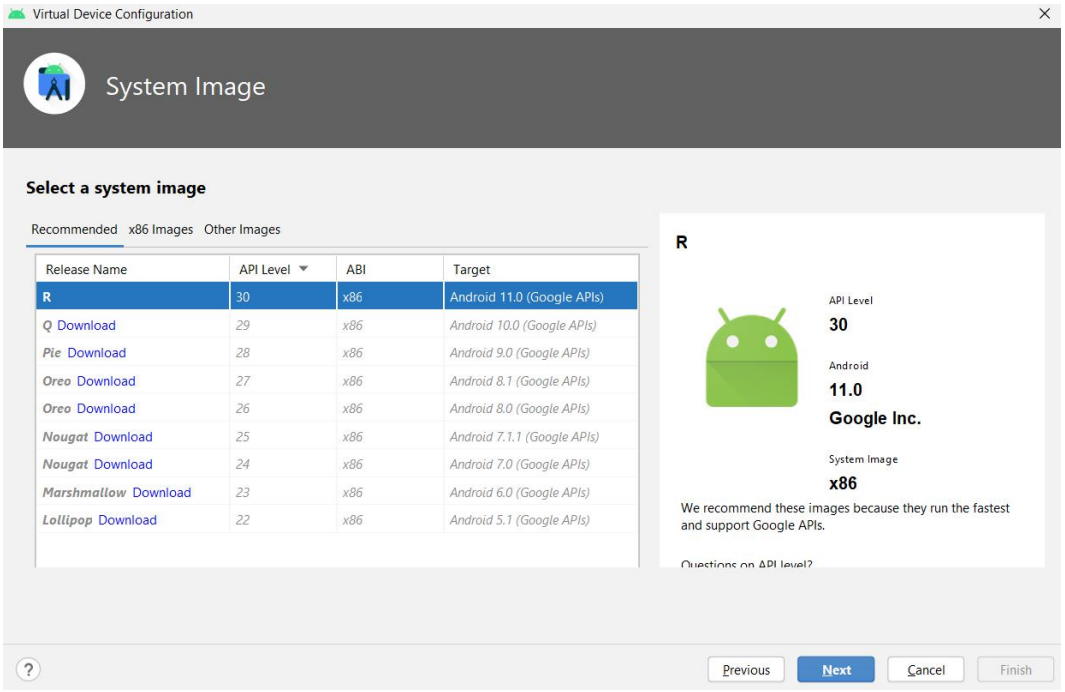


2.在新打开的 Android Virtual Device Manager 窗口，点击 Create Virtual Device 按钮。

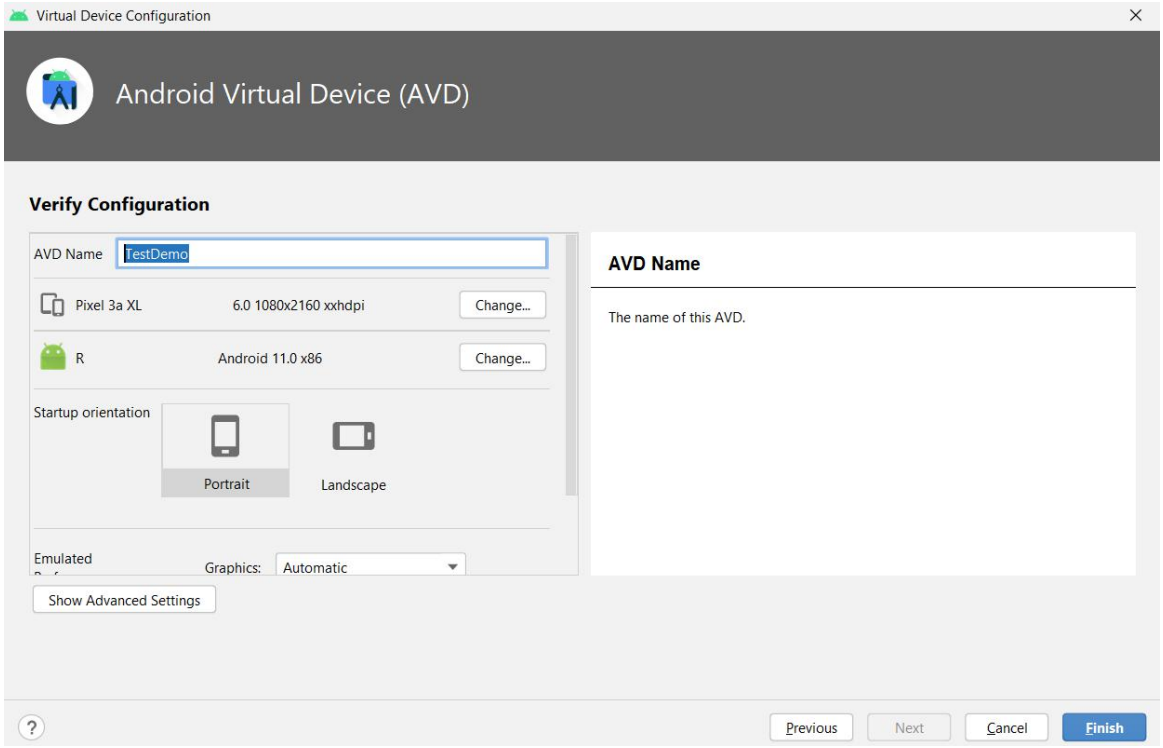


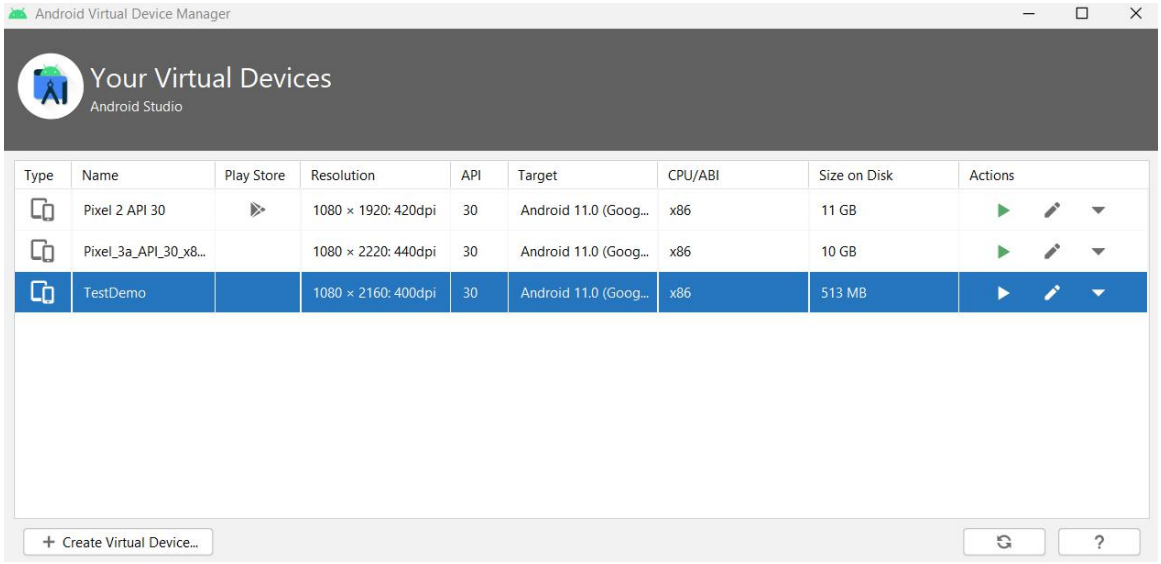
3. 在 Virtual Device Configuration 窗口，选择 Phone，在对应的参数列表里，选择您需要的设备参数。本示例中没有特别的要求。

4.选择好测试设备以后，点击 Next 按钮。在新窗口中，您需要点击 Download 下载对应的 Android Version 。(注意，我已经下载了 Android 11.0，所以这里就没有 Download 显示)。下载 Version 以后，点击 Next 按钮。



5.最后一步，我们给 Emulator 定义一个名称 （AVD Name：TestDemo)。点击 Finish 按钮。 现在，创建好的 Android 设备 （TestDemo） 就显示在了 Android Virtual Device Manager 列表里。





6.在列表中点击设备对应的绿色运行小图标，启动你的设备吧。

三、利用 adb 获取测试程序

注：这个章节我们用的是真机，获取测试程序信息。

1. 打开 Command Prompt，去到 platform-tools 路径 ：C:\Users\user\AppData\Local\Android\Sdk\ platform-tools

2.启动你的手机处于开发者模式。(通常当连接手机与电脑时，会弹出一个提示：USB 调试关闭。可以通过点击这个提示查看如何启动开发者模式。如果没有提示，通过网上搜索一下，手机设置 USB 调试，开发者选项)

3.打开 USB 调试以后，执行命令 adb devices （确保你的手机与电脑连接中），如果你启动 USB 调试正确，会得到当前连接的设备列表。

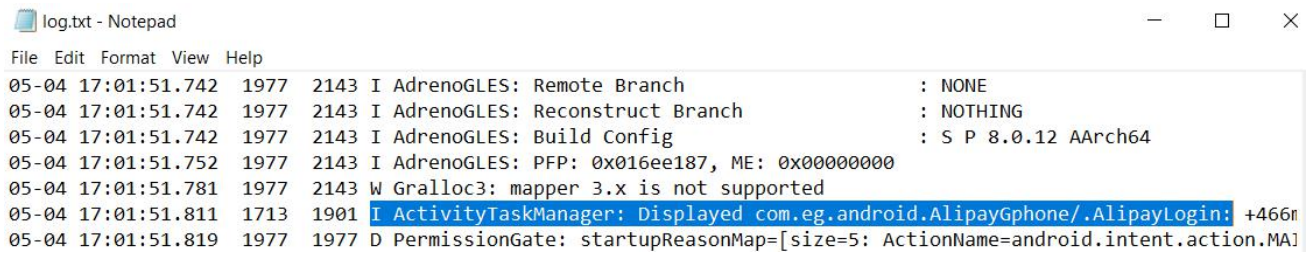
```
C:\Users\user\AppData\Local\Android\Sdk\platform-tools>adb.exe devices
List of devices attached
ee1ed769        device
```

4. 执行命令 ： adb logcat>d:/log.txt （确保一定只有你的手机一个设备连接中，先前启动 Emulator 也关闭。同时最好关闭手机中其它正在运行的程序。此命令是要记录下手机程序运行时的 log，您可以指定任何有效的地址。）

```
C:\Users\user\AppData\Local\Android\Sdk\platform-tools>adb logcat>d:/log.txt
```

5.现在打开测试程序在你的手机上(这里以 Alipay 为例，程序已经安装在了手机上)。好了，回到 Command Prompt， 点按 Ctrl＋C 结束命令执行。

6. 打开文件 d:/log.txt 搜索关键字 Displayed，会找到对应启动的 Alipay 的程序，记录下对应程序信息 com.eg.android.AlipayGphone/.AlipayLogin ， 完成以后可以关闭对应 log， Command prompt 窗口，断开电脑与手机的连接。

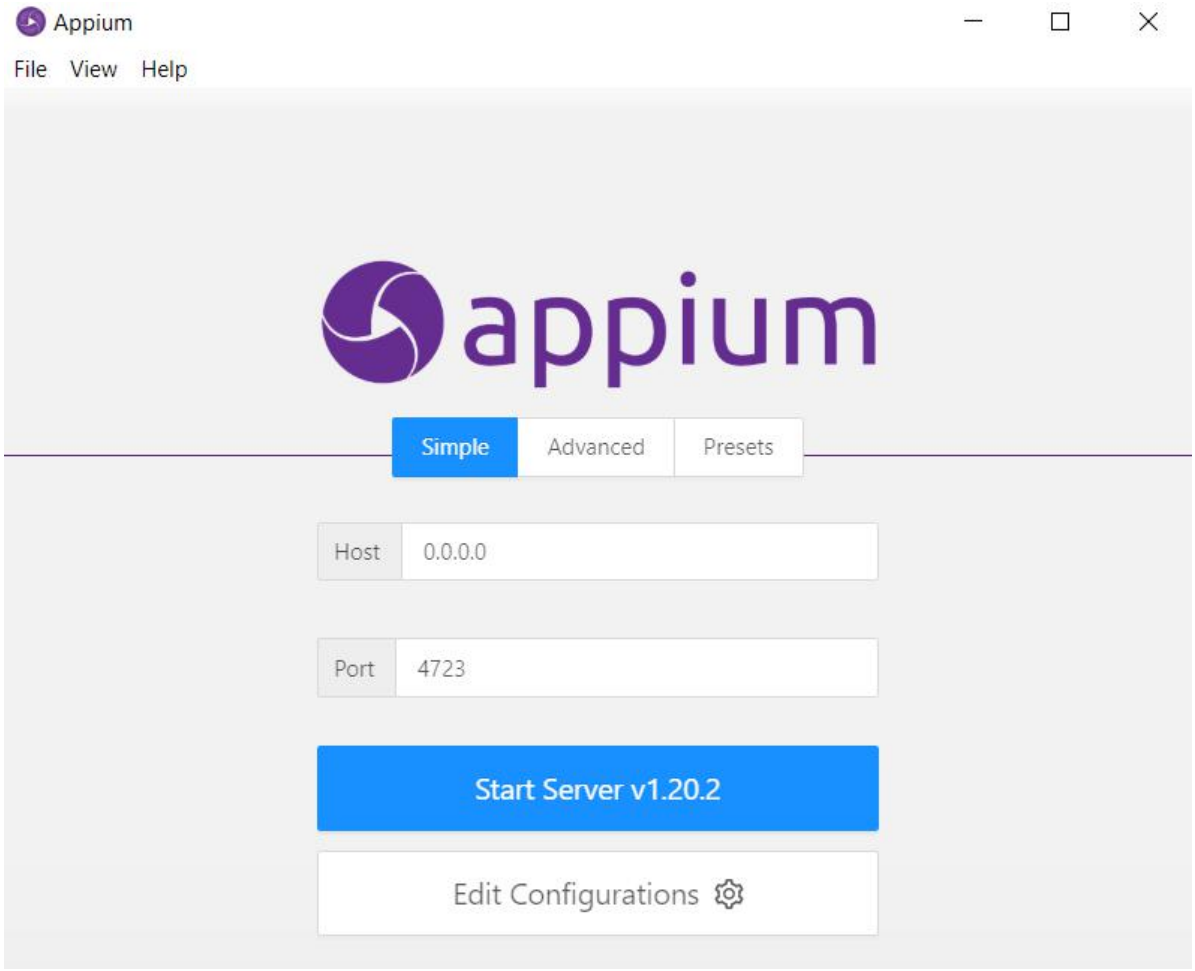


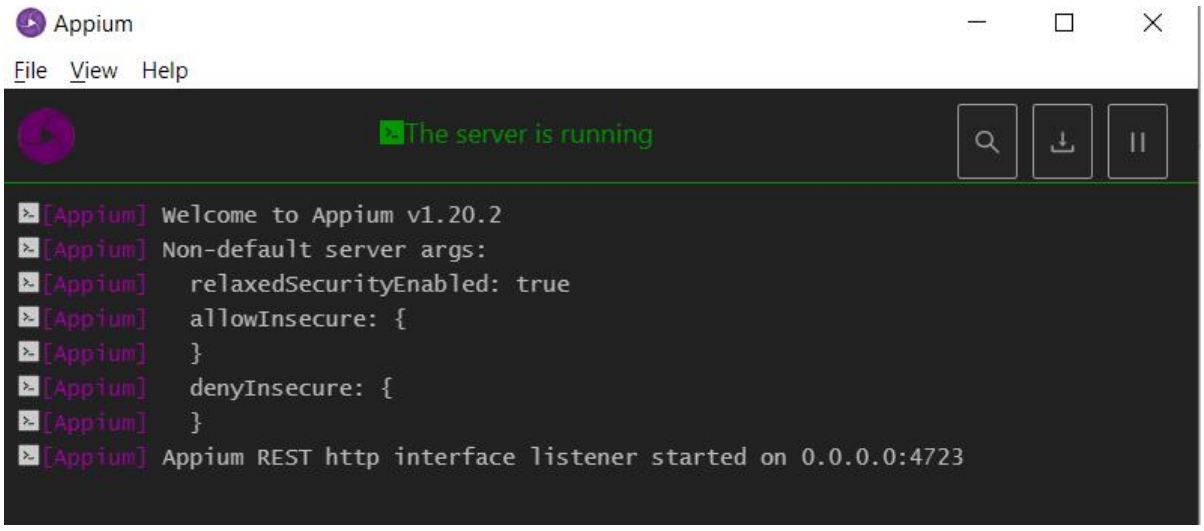
四、Appium 安装与配置

1.下载 Appium ：打开 Appium 的官方网站 https://appium.io/ ， 去到下载的列表，下载 Appium-windows-1.20.2.exe （这里使用的是 Windows OS）

2.安装： 下载完成以后，双击安装包，按照步骤一步一步安装 。

3.启动： 安装完毕以后，启动 Appium ， 在打开的窗口中点击 Start Server v1.20.2 ，如果一切配置正常，现在你可以看到 appium 启动的窗口。



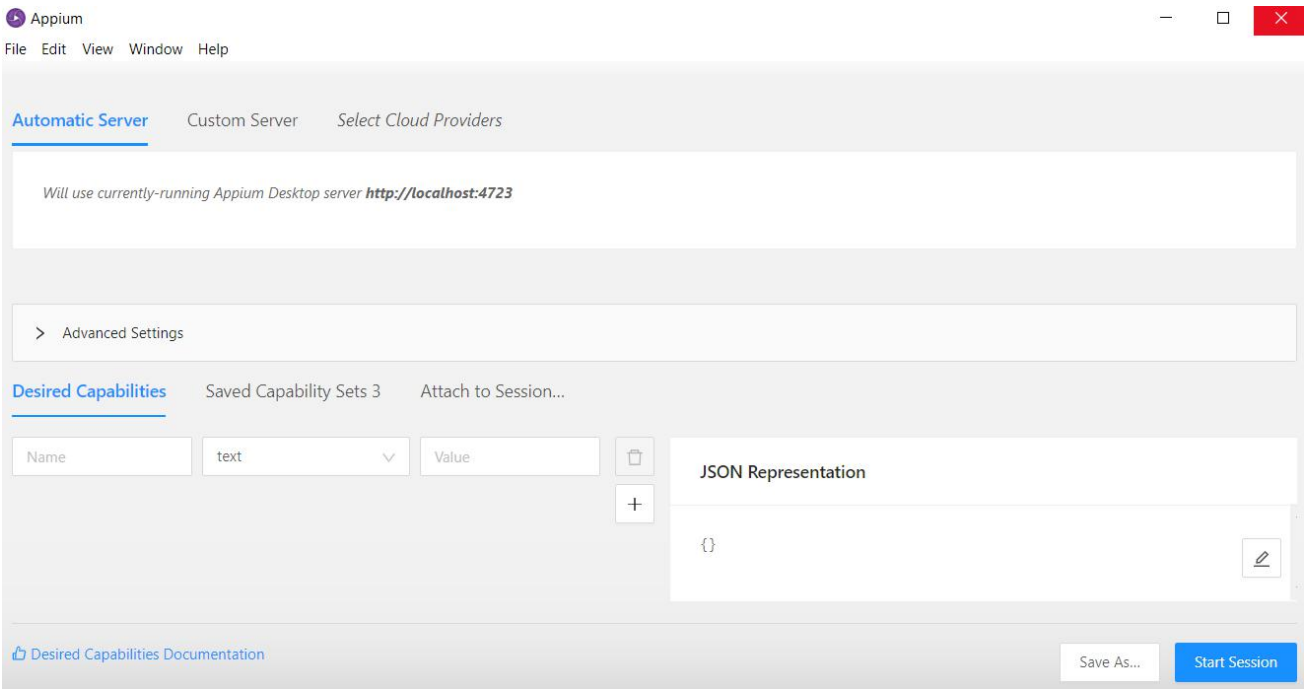


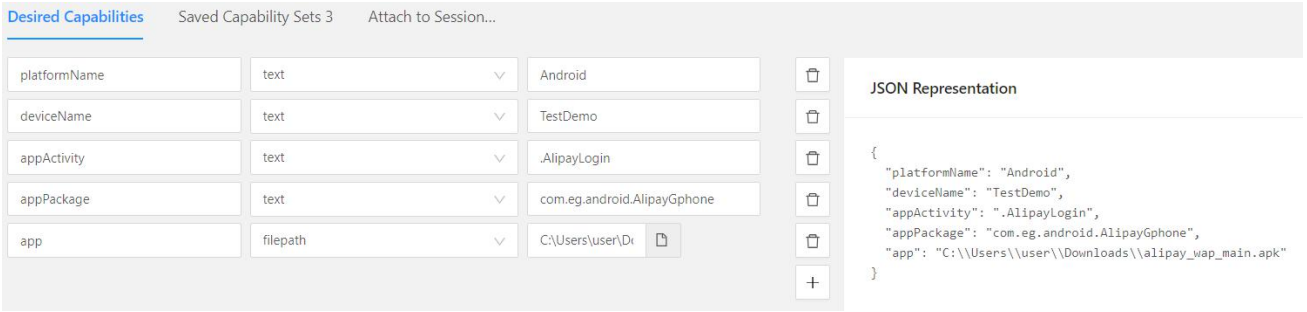
4.配置测试程序 ： 在 Appium 服务启动的窗口，点击 右上角的第一个 “搜索” 符

号 图标，在新打开窗口的 Desired Capabilities 中配置测试程序以及相关的设备信息（注：示例中配置的是使用 Android Emulator，Emulator 是没有安装测试程序的，所以这里要指定测试安装包的位置）。下面解释了每个参数的含义。

- 测试平台 ："platformName": "Android"
- 测试设备名称 ："deviceName": "TestDemo"
- 测试应用程序启动名称 ："appActivity": ".AlipayLogin"
- 测试应用程序包名称 ："appPackage": "com.eg.android.AlipayGphone"
- 测试应用程序安装包: "app": "C:\\Users\\user\\Downloads\\alipay_wap_main.apk"

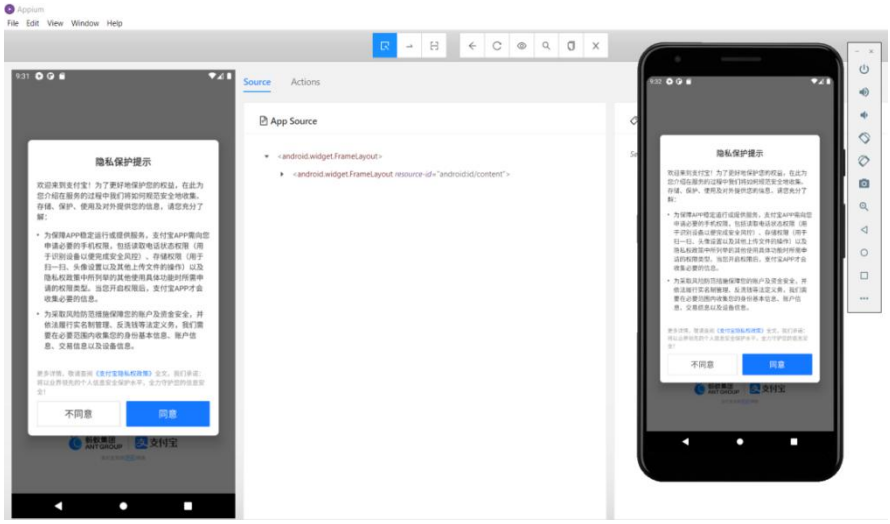
记录下 Appium 服务器地址： http://localhost:4723/wd/hub



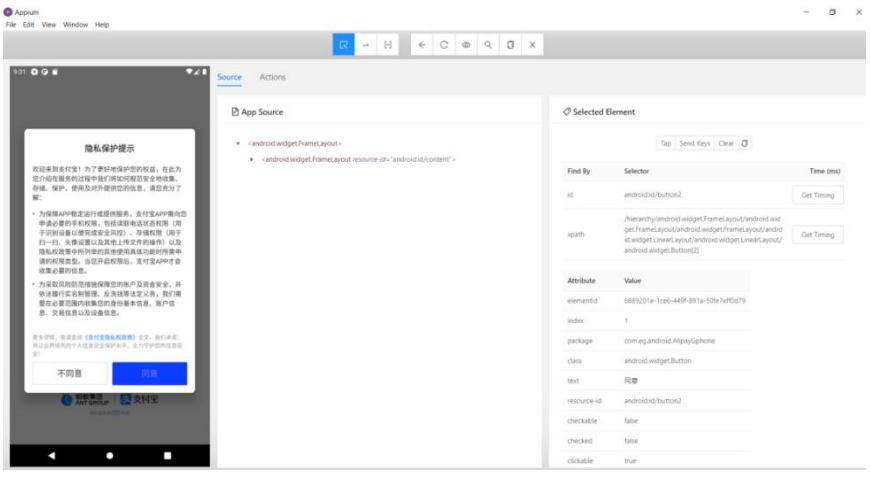


5.启动测试程序：以上配置信息填写正确以后，确保 Emulator (TestDemo)正在运行。现在点击 Start Session。如果一切正常，你会看到 Appium 开始运行程序安装，并且打开

新的程序运行窗口。同时对应的 Emulator (通过 Android Studio 运行)也会出现对应的程序运行界面。 如下图所示，左边是 Appium 程序启动界面，右边是 Emulator 运行的界面（注：这两个界面是不同的窗口，我把它们放在了同一个窗口中是方便贴图）。



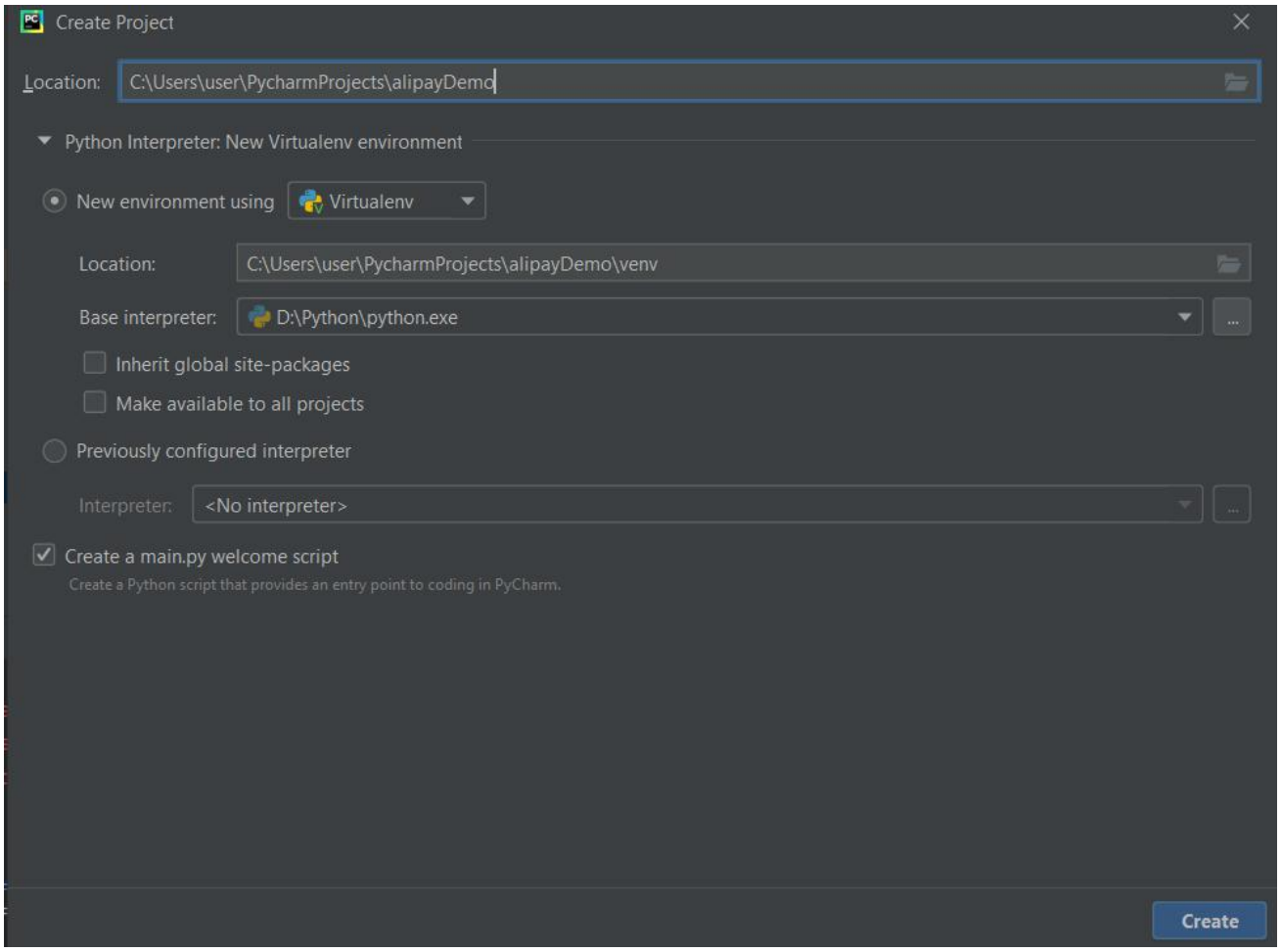
6.获取元素：在 Appium 的程序运行界面，点击上方第一个图标（select element），然后点按“同意”按钮，随即你就会看到右边的 Selected Element 会显示出当前选择的按钮的详细属性。对啦，小伙伴们一定猜到了，这些属性就是我们在写自动化测试代码时需求的元素定位信息，模拟用户操作动作时需要的。依据现有测试的规范，定位元素有好几种方法，示例主要为展示测试环境与配置为目的的。因此就不详细介绍元素的定位方法。 在测试代码演示中，仅用最常用的一种方法通过 ID 来获取要点击的元素。现在您可以按这个方法依次取得测试场景中需要的测试步骤中按钮的 ID。通过点按 Tap 模拟点击动作。(注：示例中，有一个点击 email account 的链接没有 ID, 用了 uiautomator 的定位方式，如果有些不清楚如何使用，这里您可以先忽略这个步骤。把应用程序运行起来，能够看到“同意”按钮就是成功了。本篇文章最要的目的是学习环境的搭建与配置。)



五、自动化测试程序

到现在为止，我们已经可以成功启动应用程序并进行对应的操作在 Emulator 上。接下来需要做的就是把上面启动程序以及对应用程序的操变成自动化。示例中，我们用了 Python 语言和 Pytest 框架。现在确保您安装了 Python 和 IDE。同时，确保 Python 的环境变量设置正确（Python 在安装时有一个选项，自动添加环境变量，您不妨勾选一下，这样就不用再手动去设置了）。

1.创建 Pytest 项目： 在 IDE 中需要创建一个项目。



2.安装 Pytest： 在新建的项目目录下，执行命令 pip install －U pytest （注意：Python 的环境变量一定要配置正确），安装完毕以后，执行命令 pytest --version 如果显示 pytest version 信息，说明您已安装正确。

```
Terminal:  Local  +
(venv) C:\Users\user\PycharmProjects\alipayDemo>pip install -U pytest
Collecting pytest
  Downloading pytest-6.2.4-py3-none-any.whl (280 kB)
     |████████████████████████████████| 280 kB 6.8 MB/s
Collecting colorama
  Using cached colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting attrs>=19.2.0
  Downloading attrs-21.1.0-py2.py3-none-any.whl (55 kB)
     |████████████████████████████████| 55 kB 1.5 MB/s
Collecting atomicwrites>=1.0
  Using cached atomicwrites-1.4.0-py2.py3-none-any.whl (6.8 kB)
Collecting py>=1.8.2
  Using cached py-1.10.0-py2.py3-none-any.whl (97 kB)
Collecting pluggy<1.0.0a1,>=0.12
  Using cached pluggy-0.13.1-py2.py3-none-any.whl (18 kB)
Collecting toml
  Using cached toml-0.10.2-py2.py3-none-any.whl (16 kB)
```

```
(venv) C:\Users\user\PycharmProjects\alipayDemo>pytest --version
pytest 6.2.4
```

3. 测试代码：示例的测试场景描述如下：

1) 安装 alipay 应用程序在 Emulator

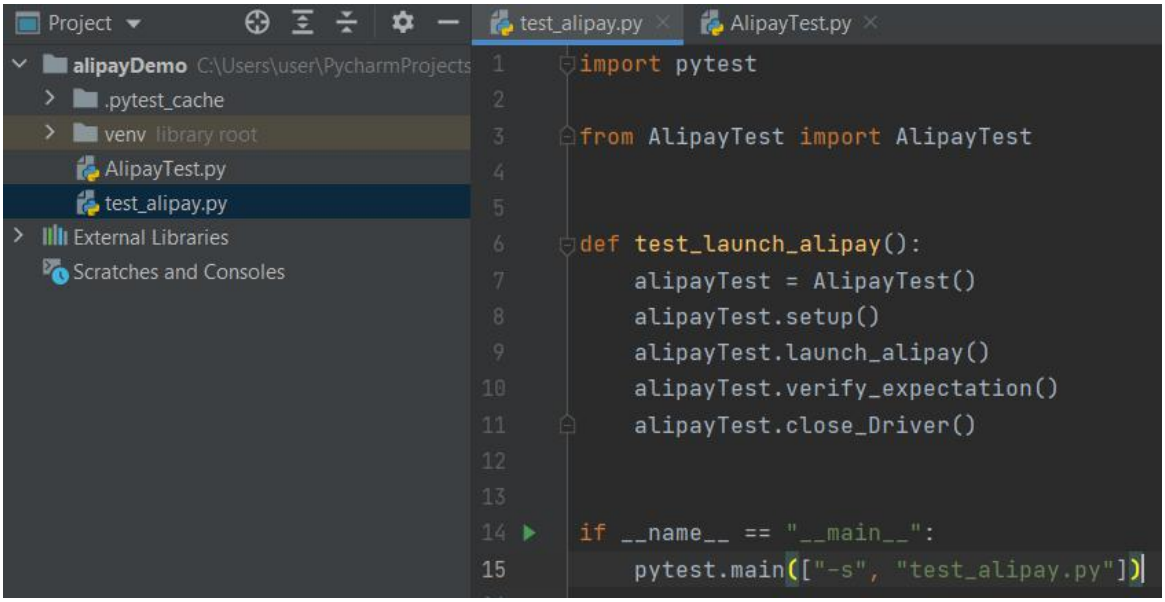
2) 点击 “同意 ”按钮，

3) 点击允许程序访问手机的 camera, photo

4) 点击登陆方式 ： Email Account

5) 验证登陆输入框确认用户在显示登陆页面

创建测试用例文件 test_alipay.py， 如下图代码所示： 这里为了使测试用例看起来简洁，执行步骤写在了另一个类文件里，这也是为了方便调用、易阅读 （测试框架配置这部分并不是本示例涉及的，不过初学的时候保持代码的整洁、易读会更容易调试代码）。如果只是想运行起安装程序，此时您可以只执行 setup()方法即可。



4. 创建类文件 AlipayTest.py 这个文件里主要有两个作用：

配置 emulator 和 appium(setup())。如在刚才的 Appium 的参数基本相同。

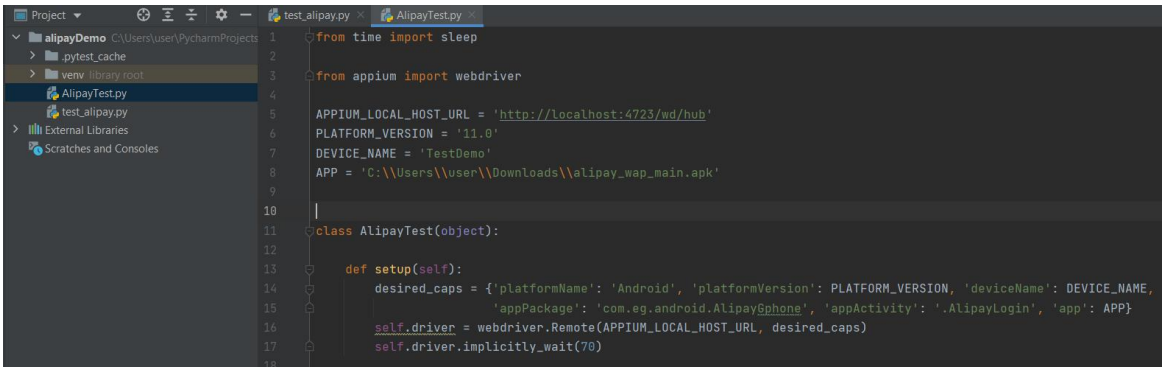
•指定 appium 服务器： APPIUM_LOCAL_HOST_URL = 'http://localhost:4723/wd/hub'

• 指定测试机使用的版本： PLATFORM_VERSION = '11.0'

• 指定测试定名称： DEVICE_NAME = 'TestDemo'

• 测试程序安装包：APP = 'C:\\Users\\user\\Downloads\\alipay_wap_main.apk'

• appPackage 与 appActivity 保持一样。

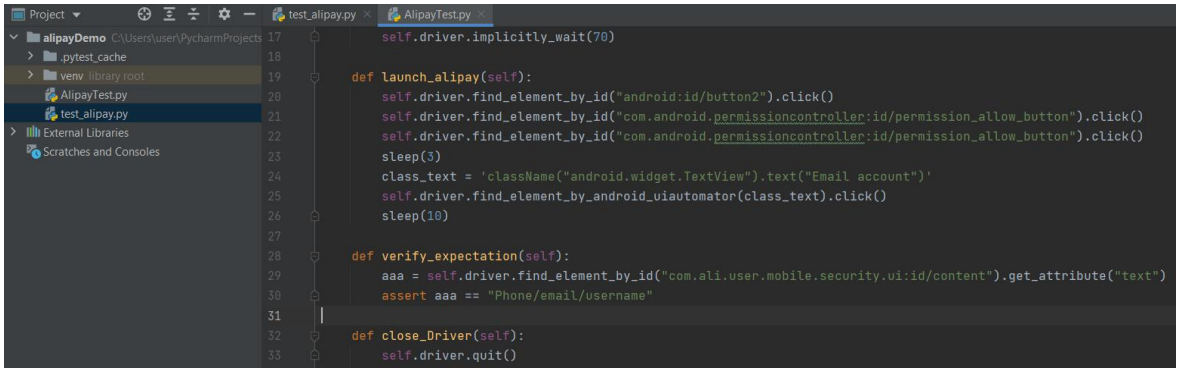


具体的测试执行步骤与验证以及关闭。 示例中写了三个方法：

•launch_alipay () - 这个方法主要执行测试场景中的 2，3，4 步骤 （注意，在 setup 的方法中，我们已经指定了测试应用程序安装包，所以在安装包会在启动时自动安装。）

• verify_expectation() - 是场景中的验证步骤。

• Close_Driver() - 是关闭当场开启的连接。



5.运行： 运行之前需要确保：

• Appium 服务器是启动着的，并且没有运行的测试程序

• Emulator （testDemo) 运行着 ，并且没有被其他程序占用

在对应的项目目录下输入 pytest， 测试用例将会自动运行起来，同时通过 emulator 也可以看到每个执行步骤的界面运行。

```
(venv) C:\Users\user\PycharmProjects\alipayDemo>pytest
============================================================================= test session starts ==
platform win32 -- Python 3.9.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: C:\Users\user\PycharmProjects\alipayDemo
collected 1 item

test_alipay.py .

============================================================================== 1 passed in 41.01s ===
```

六、结语

您的测试程序运行起来了吗？如果没有，也不要着急。再对照上面的步骤一步步检查一下，看是不是哪里没有配置好。同时也可以通过 Appium 服务器 运行的 log 里找寻可能出错的原因。即使是一个高手，这样的事情也是时有发生，这是因为手机程序的测试环境配置太过复杂了，特别又是测试在 emulator 上运行。 另外，报表就不多介绍了，需要的小伙们可以在 51testing 里搜索一下 ”一键出结果，轻松缓解测试数据收集焦虑症!” ， “安装 配置 2 步走 自动化测试报告就生成了 ” 。 这两篇文章我有详细介绍使用 allure report 与 jenkins 集成的操作步骤，以及做为独立运行时的配置和执法方法。好了，今天就介绍到此吧。随时欢迎小伙伴进行交流、提建议。

Maven 学习保姆式教程

◆ 作者：平平无奇的大腻腻

1 什么是 Maven?

官网：https://maven.apache.org/

Maven 是 Apache 下的一个纯 java 开发的开源项目，它是一个项目管理工具，使用 maven 对 java 项目进行构建、依赖管理。

1.1 项目构建

项目构建是一个项目从编写源代码到编译、测试、运行、打包、部署、运行的过程

传统使用 eclipse 构建过程：

1）在 eclipse 中创建一个 javaweb 工程

2）在工程中编写源代码及配置文件等

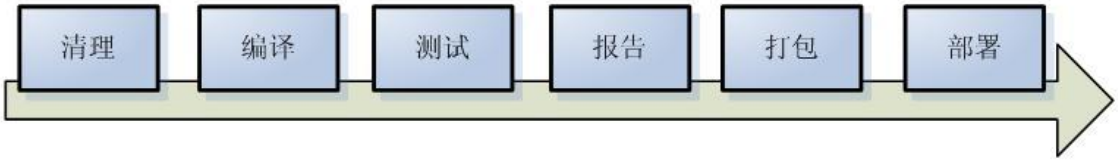
3）对源代码进行编译，java 文件编译成 class 文件

4）执行 Junit 单元测试

5）将工程打成 war 包部署至 tomcat 运行

Maven 项目构建过程：

maven 将项目构建的过程进行标准化，每个阶段使用一个命令完成，下图展示了构建过程的一些阶段，后面章节详细介绍每个阶段，这里先大概了解下：



上图中部分阶段对应命令如下：

清理阶段对应 maven 的命令是 clean，清理输出的 class 文件

编译阶段对应 maven 的命令是 compile，将 java 代码编译成 class 文件。

打包阶段对应 maven 的命令是 package，java 工程可以打成 jar 包，web 包可以打成 war 包

运行一个 maven 工程（web 工程）需要一个命令：jerty:run

1.2 依赖管理

什么是依赖？一个 java 项目可能要使用一些第三方的 jar 包才可以运行，那么我们说这个 java 项目依赖了这些第三方的 jar 包。

什么是依赖管理？就是对项目所有依赖的 jar 包进行规范化管理。

传统的依赖管理：

程序员从网上下载 jar 包添加到项目工程中；存在的缺陷：1、没有对 jar 包的版本统一管理，容易导致版本冲突。2、从网上找 jar 包非常不方便，有些 jar 找不到。3、jar 包添加到工程中导致工程过大。

Maven 项目依赖管理：

maven 项目管理所依赖的 jar 包不需要手动向工程添加 jar 包，只需要在 pom.xml（maven 工程的配置文件）添加 jar 包的坐标，自动从 maven 仓库中下载 jar 包、运行，如下图。且我们不需要去关注 jar 包之间的的依赖，例如：我们的项目依赖 abc 这个 jar 包，而 abc 又依赖 xyz 这个 jar 包，当我们声明了 abc 的依赖时，Maven 自动把 abc 和 xyz 都加入了我们的项目依赖，不需要我们自己去研究 abc 是否需要依赖 xyz。

1.3 总结 Maven 的好处

1、一步构建

maven 对项目构建的过程进行标准化，通过一个命令即可完成构建过程。

2、依赖管理

maven 工程不用手动导 jar 包，通过在 pom.xml 中定义坐标从 maven 仓库自动下载，方便且不易出错。

3、maven 的跨平台，可在 window、linux 上使用。

4、maven 遵循规范开发有利于提高大型团队的开发效率，降低项目的维护成本。

2 Maven 环境的安装

2.1 下载安装

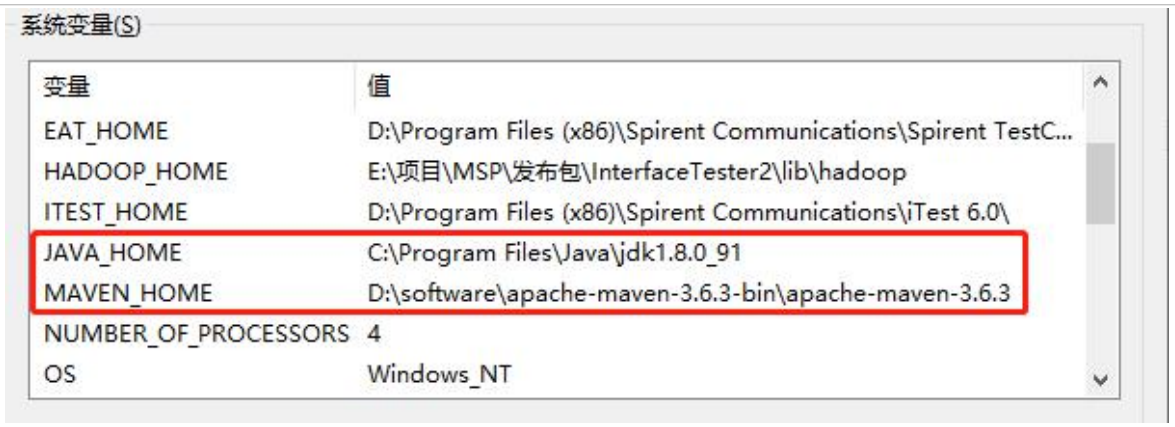
下载（https://maven.apache.org/download.cgi）

解压：

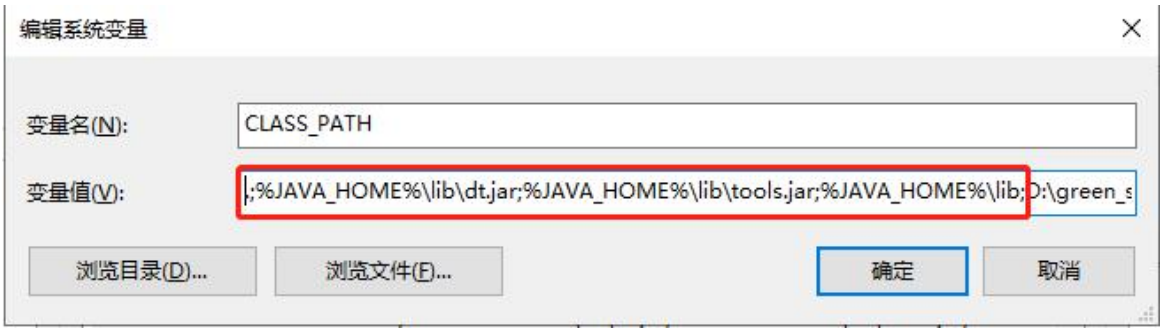
2.2 环境变量配置

1)环境变量配置

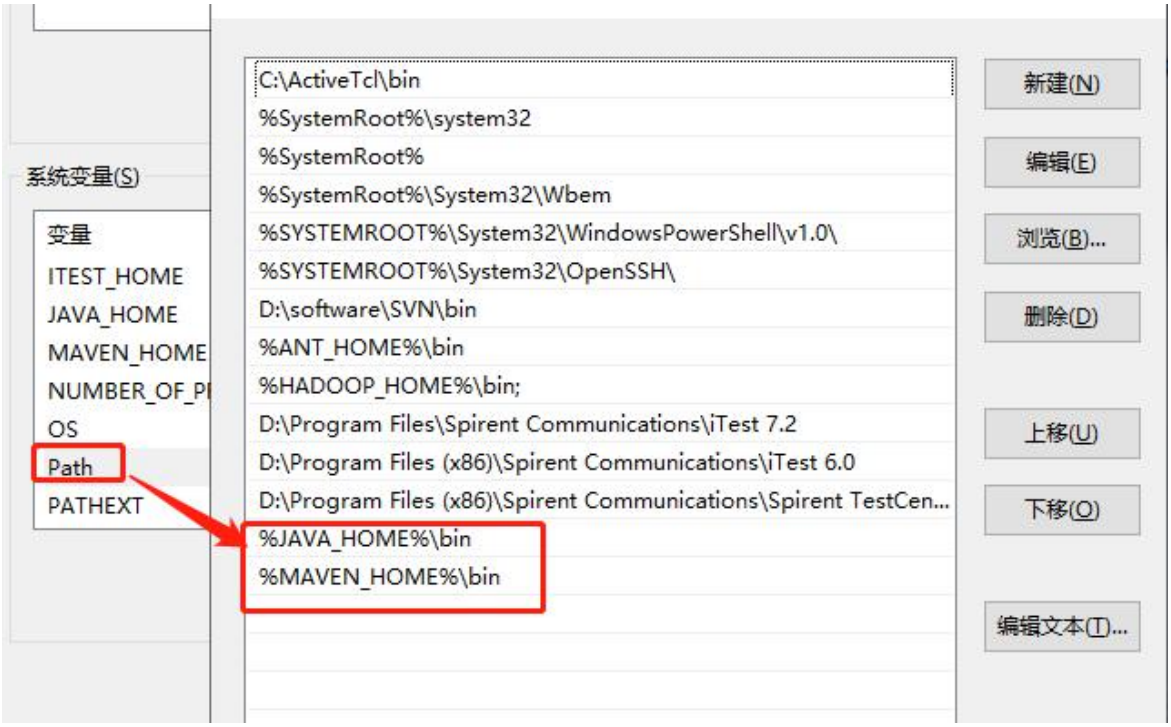
新建 JAVA_HOME、MAVEN_HOME 变量



修改 CLASS_PATH:（.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;）



添加 Path:

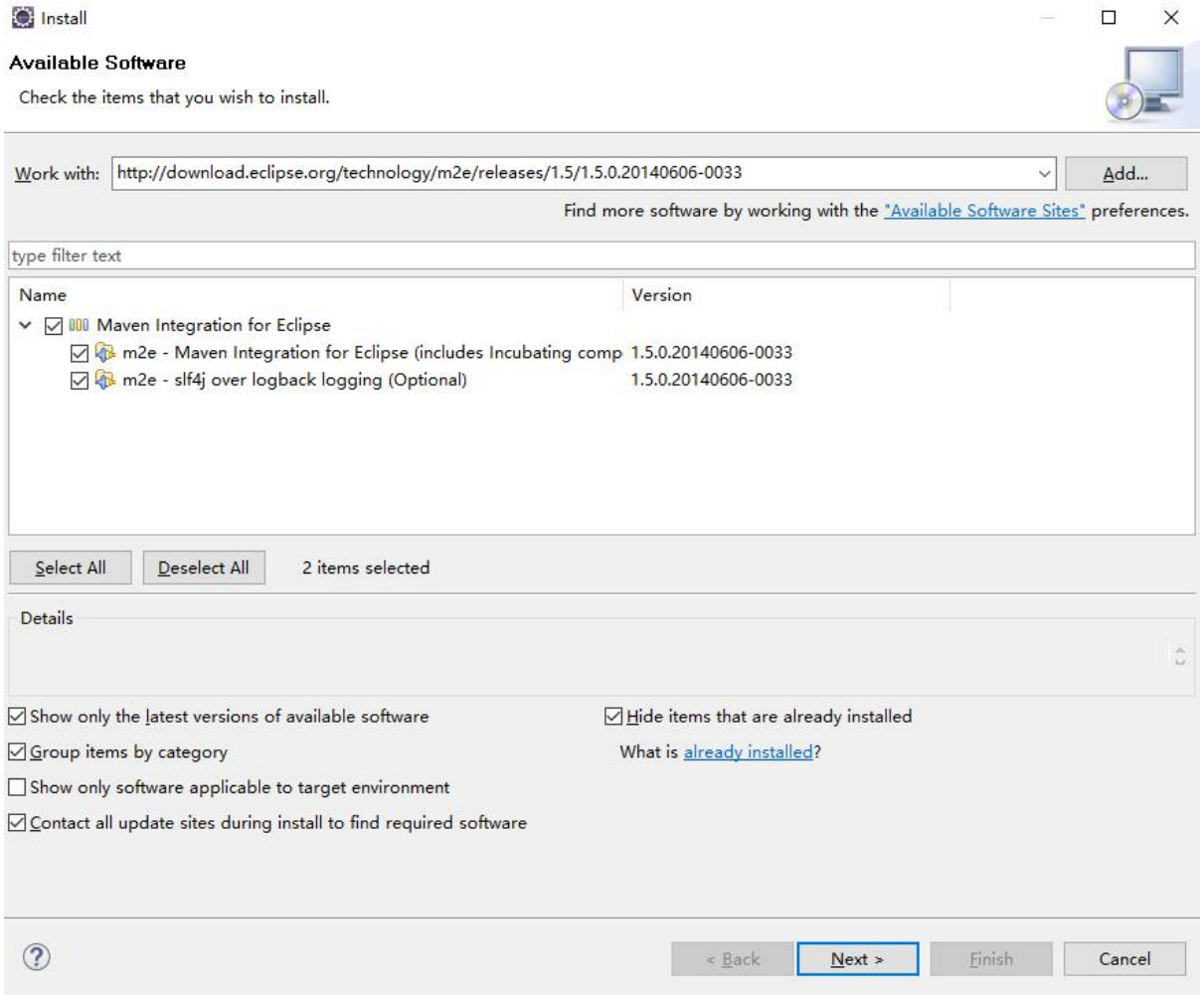


2)检测安装是否成功

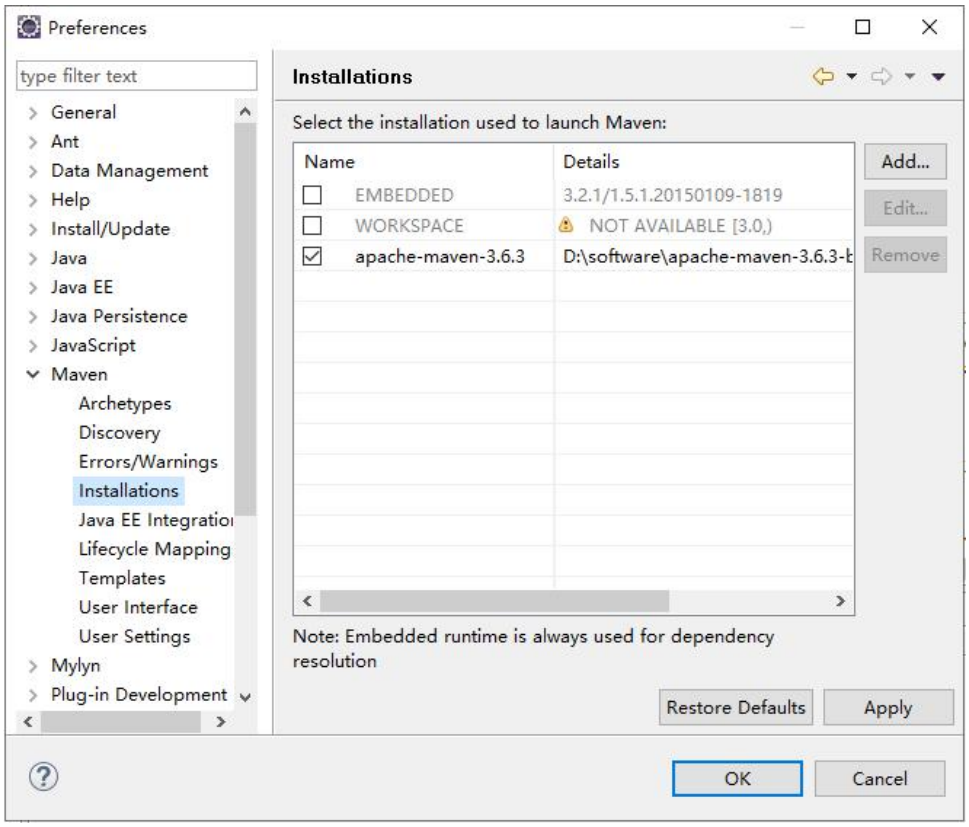
```
C:\Users\wyj>java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b15, mixed mode)

C:\Users\wyj>mvn -v
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: D:\software\apache-maven-3.6.3-bin\apache-maven-3.6.3\bin\..
Java version: 1.8.0_91, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_91\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

2.3 Eclipse 下 maven 插件的安装

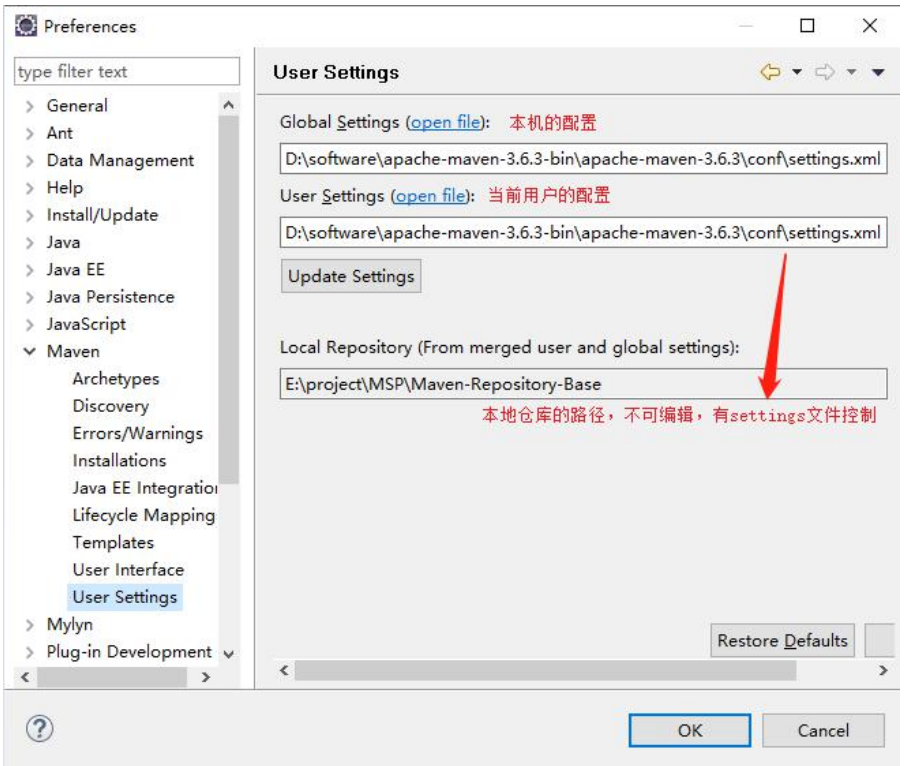
1)在 eclipse 菜单栏中找到 help->InstallNewSoftware，在线下载即可(如下图)



2)Maven 选项中点击 Installations，之后选择自己本地 Maven 的安装位置（如下图）



3)在 UserSettings 中配置（如下图），UserLevel 优先级>GlobalLevel

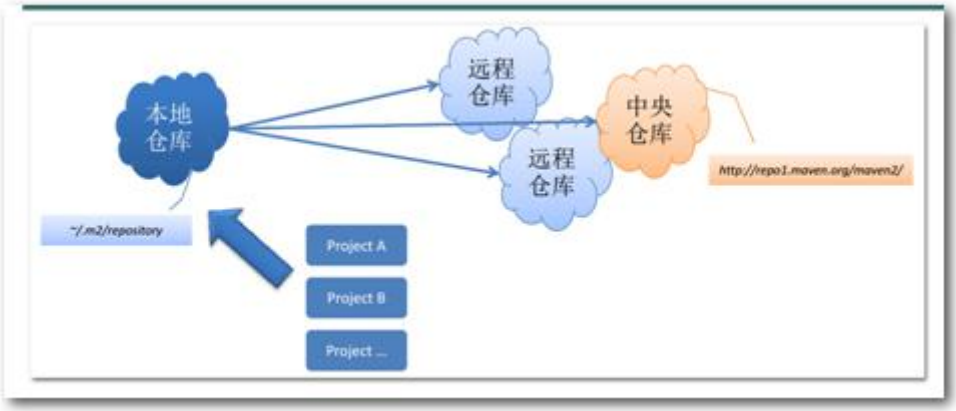


通过以上步骤，eclipse 中的 Maven 配置已经搞定，你就可以创建 Maven 工程了。

3 Maven 仓库

maven 的工作需要从仓库下载一些 jar 包，如下图所示，本地的项目 A、项目 B 等都会通过 maven 软件从远程仓库下载 jar 包并存在本地仓库，本地仓库就是本地文件夹，当第二次需要此 jar 包时则不再从远程仓库下载，因为本地仓库已经存在了，可以将本地仓库理解为缓存，有了本地仓库就不用每次从远程仓库下载了。

下图描述了 maven 中仓库的类型：



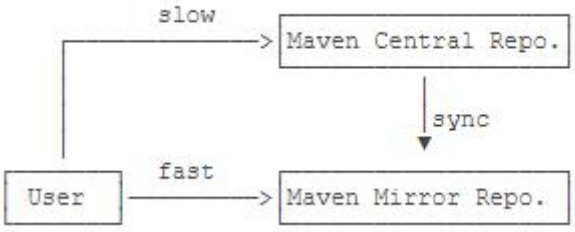
• 本地仓库：用来存储从远程仓库或中央仓库下载的插件和 jar 包，项目使用一些插件或 jar 包，优先从本地仓库查找。

默认本地仓库位置在${user.dir}/.m2/repository，${user.dir}表示 windows 用户目录。

• 远程仓库：如果本地需要插件或者 jar 包，本地仓库没有，默认去远程仓库下载。

远程仓库也就是我们常说的私服。私服还充当一个代理服务器，当私服上没有 jar 包会从互联网中央仓库自动下载，详细参见后文 Maven 私服介绍。

• 中央仓库：在 maven 软件中内置一个远程仓库地址 http://repo1.maven.org/maven2，它是中央仓库，服务于整个互联网，它是由 Maven 团队自己维护，里面存储了非常全的 jar 包，它包含了世界上大部分流行的开源项目构件。除了可以从 Maven 的中央仓库下载外，还可以从 Maven 的镜像仓库下载。如果访问 Maven 的中央仓库非常慢，我们可以选择一个速度较快的 Maven 的镜像仓库。Maven 镜像仓库定期从中央仓库同步：



settings.xml 文件配置（官方文档：https://maven.apache.org/settings.html），settings.xml 是 maven 的全局配置文件；而 pom.xml 文件是所在项目的局部配置对使用到主要配置简单介绍：

```
<!—本地仓库路径，默认值: ${user.home}/.m2/repository -->
<localRepository>E:\project\MSP\Maven-Repository-Base</localRepository>

<!—为仓库列表配置的下载镜像列表-->
</mirrors>
<mirror>
        <id>alimaven</id>
        <name>aliyun maven</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        <mirrorOf>central</mirrorOf>
</mirror>
<mirror>
        <id>repo1</id>
        <mirrorOf>central</mirrorOf>
        <name>Human Readable Name for this Mirror.</name>
        <url>http://repo1.maven.org/maven2/</url>
  </mirror>
</mirrors>

<!-- 进行远程服务器访问时所需的授权配置信息 -->
</servers>
<server>
        <!-- 这是 server 的 id，该 id 与 distributionManagement 中 repository 元素的 id 相匹配
-->
```

```
            <id>releases</id>
            <username>admin</username>
            <password>admin123</password>
        </server>
    <server>
            <id>snapshots</id>
            <username>admin</username>
            <password>admin123</password>
        </server>
     </servers>

<!--根据环境参数来调整构建配置的列表-->
</profiles>
<profile>
        <!—配置 Maven 默认使用 jdk1.8 编译项目-->
         <id>JDK-1.8</id>
         <activation>
        <activeByDefault>true</activeByDefault>
            <jdk>1.8</jdk>
         </activation>
      <properties>
            <maven.compiler.source>1.8</maven.compiler.source>
            <maven.compiler.target>1.8</maven.compiler.target>
            <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
      </properties>
</profile>
    <profile>
        <id>dev</id>
        <repositories>
          <!—远程仓库列表-->
          <repository>
            <id>nexus</id>
            <url>http://172.16.61.248:8081/nexus/content/groups/public/</url>
```

```
            <releases>
              <enabled>true</enabled>
            </releases>
            <snapshots>
              <enabled>true</enabled>
            </snapshots>
          </repository>
        </repositories>
        <pluginRepositories>
            <!-- 插件仓库，maven 的运行依赖插件，也需要从私服下载插件 -->
            <pluginRepository>
                <id>public</id>
                <name>Public Repositories</name>
                <url>http://172.16.61.248:8081/nexus/content/groups/public/</url>
            </pluginRepository>
        </pluginRepositories>
    </profile>
    </profiles>
  <!-- 让增加的 profile 生效 -->
    <activeProfiles>
      <activeProfile>JDK-1.8</activeProfile>
    <activeProfile>dev</activeProfile>
    </activeProfiles>
```

4 Maven 工程

Maven 包含了一个项目对象模型(ProjectObjectModel)，一组标准集合，一个项目生命周期(ProjectLifecycle)，一个依赖管理系统(DependencyManagementSystem)，和用来运行定义在生命周期阶段(phase)中插件(plugin)目标(goal)的逻辑。

使用 maven 创建的工程我们称它为 maven 工程。

4.1 maven 工程的目录规范

如下：

```
Project
  |-src
  |    |-main
  |    |  |-java          —— 存放项目的.java 文件
  |    |  |-resources     —— 存放项目资源文件，如数据库配置文件
  |    |-test
  |         |-java         ——存放所有测试.java 文件，如 JUnit 测试类
  |         |-resources    —— 测试资源文件
  |-target                 —— 目标文件输出位置例如.class、.jar、.war 文件
  |-pom.xml                ——maven 项目核心配置文件
```

4.2 模块化管理

Maven 多模块项目，通过合理的模块拆分，实现代码的复用，便于维护和管理。注意到 parent 的<packaging>是 pom 而不是 jar，因为 parent 本身不含任何 Java 代码。编写 parent 的 pom.xml 只是为了在各个模块中减少重复的配置。

现在我们的整个工程结构如下：

```
TBase-parent        (父级)
      ---pom.xml
           ---MavenTest-moudle              (moudle 级)
                ----pom.xml
           ---testCase-moudle               (moudle 级)
                ----pom.xml
```

运行：在父工程中配置 module 聚合运行，如下命令行。

4.3 maven 工程的 pom

（标签大全详解，参：https://www.runoob.com/maven/maven-pom.html）

4.3.1 ParentPOM:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.autotest</groupId>
  <artifactId>TBase</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>TBase</name>

  <properties><!—属性 -->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies><!--依赖管理 -->
    <dependency>
        <groupId>com.rabbitmq</groupId>
        <artifactId>amqp-client</artifactId>
        <version>5.1.2</version>
    </dependency>
        ……
  </dependencies>
  <distributionManagement><!--mvn deploy 用来将项目生成的构建分发到私服仓库 -->
    <repository>
        <id>releases</id>
    <url>http://172.16.61.248:8081/nexus/content/repositories/releases/</url>
    </repository>
    <snapshotRepository>
        <id>snapshots</id>
    <url>http://172.16.61.248:8081/nexus/content/repositories/snapshots/</url>
    </snapshotRepository>
  </distributionManagement>

  <build>
    <pluginManagement> <!--使用的插件列表 -->
      <plugins>
        ……
      </plugins>
    </pluginManagement>
  </build>
  <packaging>pom</packaging>

  <modules><!--模块 被构建成项目的一部分-->
    <module>MavenTest</module>
  </modules>
```

4.3.2 MoudlePOM:

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
        <artifactId>TBase</artifactId>
        <groupId>com.autotest</groupId>
        <version>0.0.1-SNAPSHOT</version>
  </parent>
  <groupId>com.autotest</groupId>
  <artifactId>MavenTest</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>MavenTest</name>
  <url>http://www.example.com</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <pluginManagement>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.0.2</version>
            <configuration>
                <archive>
                    <manifest>
                        <addClasspath>true</addClasspath>
                        <mainClass>org.MavenTest.App</mainClass> <!--入口-->
                    </manifest>
                </archive>
            </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

4.4 maven 依赖管理

1)定义依赖坐标

对于某个依赖，Maven 只需要 3 个变量即可唯一确定某个 jar 包：

groupId：属于组织的名称，类似 Java 的包名；

artifactId：该 jar 包自身的名称，类似 Java 的类名；

version：该 jar 包的版本。

2)定义依赖关系

Maven 定义了几种依赖关系，分别是 compile、test、runtime 和 provided：

scope	说明	示例
compile	编译时需要用到该 jar 包（默认）	commons-logging
test	编译 Test 时需要用到该 jar 包	junit
runtime	编译时不需要，但运行时需要用到	mysql
provided	编译时需要用到，但运行时由 JDK 或某个服务器提供	servlet-api

3)搜索第三方组件（https://search.maven.org/），如下截图所示为 json 对应的坐标信息：

4)当前项目依赖情况

第三方组件：参见工程

4.5 maven 构建

maven 对项目构建过程分为三套相互独立的生命周期，如下图所示。

default Lifecycle

```
<phases>
  <phase>validate</phase>
  <phase>initialize</phase>
  <phase>generate-sources</phase>
  <phase>process-sources</phase>
  <phase>generate-resources</phase>
  <phase>process-resources</phase>
  <phase>compile</phase>
  <phase>process-classes</phase>
  <phase>generate-test-sources</phase>
  <phase>process-test-sources</phase>
  <phase>generate-test-resources</phase>
  <phase>process-test-resources</phase>
  <phase>test-compile</phase>
  <phase>process-test-classes</phase>
  <phase>test</phase>
  <phase>prepare-package</phase>
  <phase>package</phase>
  <phase>pre-integration-test</phase>
  <phase>integration-test</phase>
  <phase>post-integration-test</phase>
  <phase>verify</phase>
  <phase>install</phase>
  <phase>deploy</phase>
</phases>
```

clean Lifecycle

```
<phases>
  <phase>pre-clean</phase>
  <phase>clean</phase>
  <phase>post-clean</phase>
</phases>
<default-phases>
  <clean>
    org.apache.maven.plugins:maven-clean-plugin:2.5:clean
  </clean>
</default-phases>
```

site Lifecycle

```
<phases>
  <phase>pre-site</phase>
  <phase>site</phase>
  <phase>post-site</phase>
  <phase>site-deploy</phase>
</phases>
<default-phases>
  <site>
    org.apache.maven.plugins:maven-site-plugin:3.3:site
  </site>
  <site-deploy>
    org.apache.maven.plugins:maven-site-plugin:3.3:deploy
  </site-deploy>
</default-phases>
```

使用 Maven 构建项目就是执行 lifecycle，执行到指定的 phase 为止。每个 phase 会执行自己默认的一个或多个 goal。goal 是最小任务单元。

以 compile 这个 phase 为例，如果执行：mvncompile，Maven 将执行 compile 这个 phase，但是 Maven 本身其实并不知道如何执行 compile，它只是负责找到对应的 compiler 插件，然后执行默认的 compiler:compile 这个 goal 来完成编译。

那如何知道有哪些插件有对应执行了哪些 goal 命令呢？

1、详细参加官网：https://maven.apache.org/plugins/index.html

2、快捷方式：mvnhelp:describe-Dplugin=groupId:artifactId:version，如下：

mvnhelp:describe-Dplugin=compiler

Name:ApacheMavenCompilerPlugin

Description:TheCompilerPluginisusedtocompilethesourcesofyour

project.

GroupId:org.apache.maven.plugins

ArtifactId:maven-compiler-plugin

Version:3.8.0

GoalPrefix:compiler

Thispluginhas3goals:

compiler:compile

Description:Compilesapplicationsources

compiler:help

Description:Displayhelpinformationonmaven-compiler-plugin.

Callmvncompiler:help-Ddetail=true-Dgoal=<goal-name>todisplay

parameterdetails.

compiler:testCompile

Description:Compilesapplicationtestsources.

扩展

了解完 maven 构建的原理，我们便可以自定义插件，其实我们就明白了大概自动化构建从哪里入手，举个例子，我们定义 Jetty 插件并运行：

1.添加 jetty 插件

```
<plugin>
        <groupId>org.mortbay.jetty</groupId>
        <artifactId>maven-jetty-plugin</artifactId>
        <version>6.1.10</version>
        <configuration>
            <httpConnector>
                <port>8080</port>
                <host>localhost</host>
            </httpConnector>
            <webApp>
                <contextPath>/mavenTest</contextPath>
            </webApp>
            <scanIntervalSeconds>1</scanIntervalSeconds>
        </configuration>
    </plugin>
```

2.在项目目录下执行 mvn?jetty:run

```
E:\project\MSP\interface2019\TBase\MavenTest>mvn jetty:run
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.autotest:MavenTest >----------------------
[INFO] Building MavenTest 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> maven-jetty-plugin:6.1.10:run (default-cli) > test-compile @ MavenTest >>>
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b06-SNAPSHOT is missing, no dependency information available
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b07-SNAPSHOT is missing, no dependency information available
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b08-SNAPSHOT is missing, no dependency information available
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b11-SNAPSHOT is missing, no dependency information available
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ MavenTest ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\project\MSP\interface2019\TBase\MavenTest\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ MavenTest ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:testResources (default-testResources) @ MavenTest ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\project\MSP\interface2019\TBase\MavenTest\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ MavenTest ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< maven-jetty-plugin:6.1.10:run (default-cli) < test-compile @ MavenTest <<<
[INFO]
[INFO]
[INFO] --- maven-jetty-plugin:6.1.10:run (default-cli) @ MavenTest ---
[INFO] Configuring Jetty for project: MavenTest
[INFO] Webapp source directory = E:\project\MSP\interface2019\TBase\MavenTest\src\main\webapp
[INFO] web.xml file = E:\project\MSP\interface2019\TBase\MavenTest\src\main\webapp\WEB-INF\web.xml
[INFO] Classes = E:\project\MSP\interface2019\TBase\MavenTest\target\classes
[INFO] Logging to org.slf4j.impl.MavenSimpleLogger(org.mortbay.log) via org.mortbay.log.Slf4jLog
[INFO] Context path = /MavenTest
[INFO] Tmp directory =  determined at runtime
[INFO] Web defaults = org/mortbay/jetty/webapp/webdefault.xml
[INFO] Web overrides =  none
[INFO] Webapp directory = E:\project\MSP\interface2019\TBase\MavenTest\src\main\webapp
[INFO] Starting jetty 6.1.10 ...
[INFO] jetty-6.1.10
[INFO] No Transaction manager found - if your webapp requires one, please configure one.
[INFO] Started SelectChannelConnector@0.0.0.0:8080
```

4.6 maven 工程常用的命令

格式：mvn[options][<goal(s)>][<phase(s)>]

clean：maven 工程的清理命令，执行 clean 会删除 target 目录的内容

compile：maven 工程的编译命令，作用是将 src/main/java 下的文件编译为 class 文件输出到 target 目录下。

test：执行 src/test/java 下单元测试类

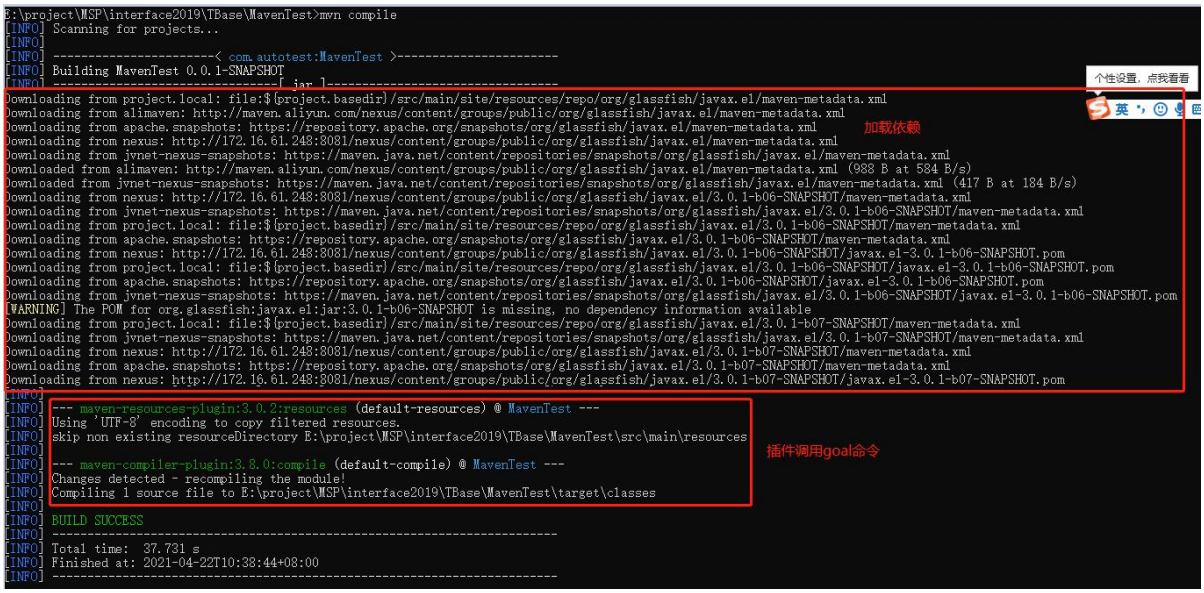
package：maven 工程的打包命令

install：将 maven 打成 jar 包或 war 包发布到本地仓库。

deploy：部署到私服

mvndependency:tree 直接打印出依赖树

module 编译：



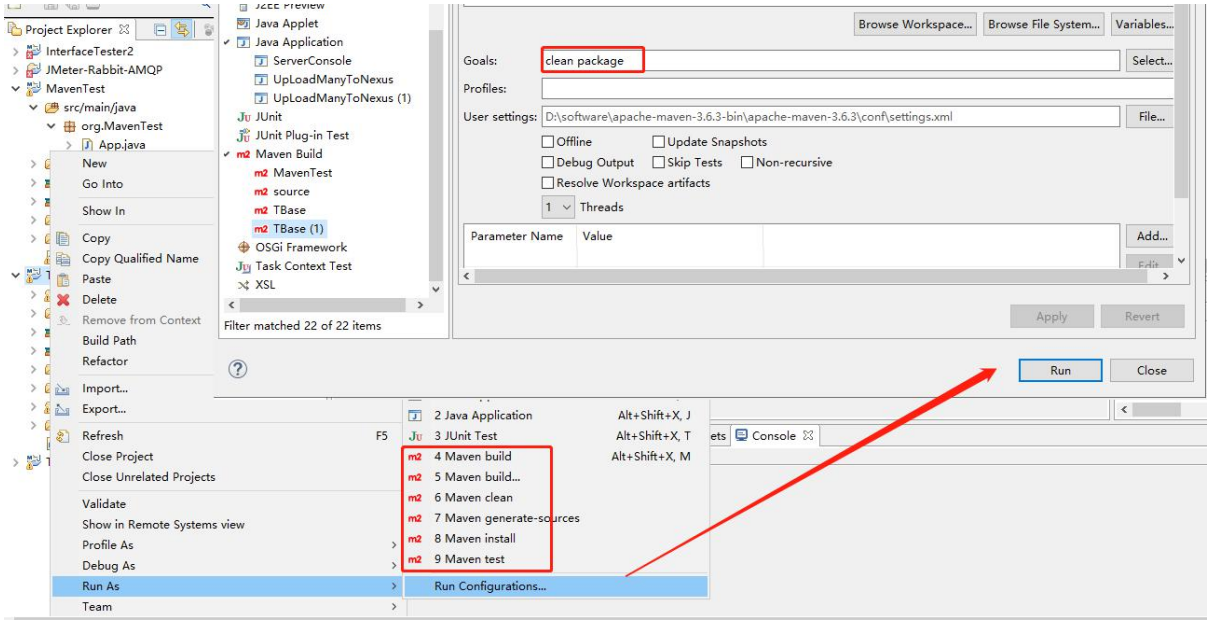
module 打包：

```
E:\project\MSP\interface2019\TBase\MavenTest>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.autotest:MavenTest >----------------------
[INFO] Building MavenTest 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b06-SNAPSHOT is missing, no dependency information available
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b07-SNAPSHOT is missing, no dependency information available
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b08-SNAPSHOT is missing, no dependency information available
[WARNING] The POM for org.glassfish:javax.el:jar:3.0.1-b11-SNAPSHOT is missing, no dependency information available
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ MavenTest ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\project\MSP\interface2019\TBase\MavenTest\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ MavenTest ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:testResources (default-testResources) @ MavenTest ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\project\MSP\interface2019\TBase\MavenTest\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ MavenTest ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to E:\project\MSP\interface2019\TBase\MavenTest\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ MavenTest ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.MavenTest.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.511 s - in org.MavenTest.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0                    打包
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ MavenTest ---
[INFO] Building jar: E:\project\MSP\interface2019\TBase\MavenTest\target\MavenTest-0.0.1-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  25.577 s
[INFO] Finished at: 2021-04-27T17:18:06+08:00
[INFO] ------------------------------------------------------------------------
```

在父工程中配置 module 聚合运行：

```
E:\project\MSP\interface2019\TBase>mvn compile
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] TBase                                                              [pom]
[INFO] MavenTest                                                          [jar]
[INFO]
[INFO] ----------------------< com.autotest:TBase >----------------------
[INFO] Building TBase 0.0.1-SNAPSHOT                                      [1/2]
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] ----------------------< com.autotest:MavenTest >----------------------
[INFO] Building MavenTest 0.0.1-SNAPSHOT                                  [2/2]
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from project.local: file:${project.basedir}/src/main/site/resources/repo/org/glassfish/javax.el/maven-metadata.xml
Downloading from nexus: http://172.16.61.248:8081/nexus/content/groups/public/org/glassfish/javax.el/maven-metadata.xml
Downloading from jvnet-nexus-snapshots: https://maven.java.net/content/repositories/snapshots/org/glassfish/javax.el/maven-metadata.xml
Downloading from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/glassfish/javax.el/maven-metadata.xml
Downloading from apache.snapshots: https://repository.apache.org/snapshots/org/glassfish/javax.el/maven-metadata.xml
Downloaded from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/glassfish/javax.el/maven-metadata.xml (988 B at 367 B/s)
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ MavenTest ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\project\MSP\interface2019\TBase\MavenTest\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ MavenTest ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to E:\project\MSP\interface2019\TBase\MavenTest\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for TBase 0.0.1-SNAPSHOT:
[INFO]
[INFO] TBase ..............................................  SUCCESS [  0.103 s]
[INFO] MavenTest ..........................................  SUCCESS [01:06 min]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:06 min
[INFO] Finished at: 2021-04-09T22:16:06+08:00
[INFO] ------------------------------------------------------------------------
```

4.7 maven 项目在 Eclipse 下构建



5 Maven 私服管理

5.1 需求

在进行接口测试框架进行 Maven 项目整改时，由于前期没有进行统一管理，遇到以下问题：有些 jar 包找不到，有些 jar 包版本较低和中央仓库不匹配，有些 jar 包存在多个版本，还有自定义 jar 包……基于此，我们需要搭建自己的私服来管理以上这些 jar 包，私服服务器即是公司内部的 maven 远程仓库，电脑上安装 maven 软件并且连接私服服务器，可以将自己开发的项目打成 jar 并发布到私服服务器，其它项目组从私服服务器下载所依赖的构件（jar）。

5.2 私服搭建

Nexus 是 Maven 仓库管理器，通过 nexus 可以搭建 maven 仓库，同时 nexus 还提供强大的仓库管理功能，构件搜索功能等。

下载 Nexus，下载地址：http://www.sonatype.org/nexus/archived/

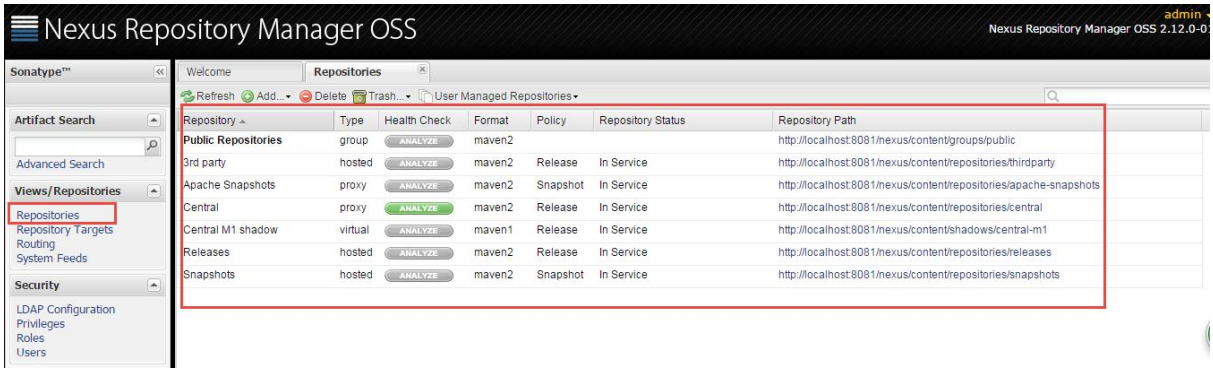
安装 Nexus，解压后 cmd 进入 bin 目录，执行 nexus.batinstall

启动 Nexus，直接启动 nexus 服务

访问：http://localhost:8081/nexus/（内置账户 admin/admin123）

（详细的配置可查看 nexus 的配置文件 conf/nexus.properties）

查看 nexus 的仓库：



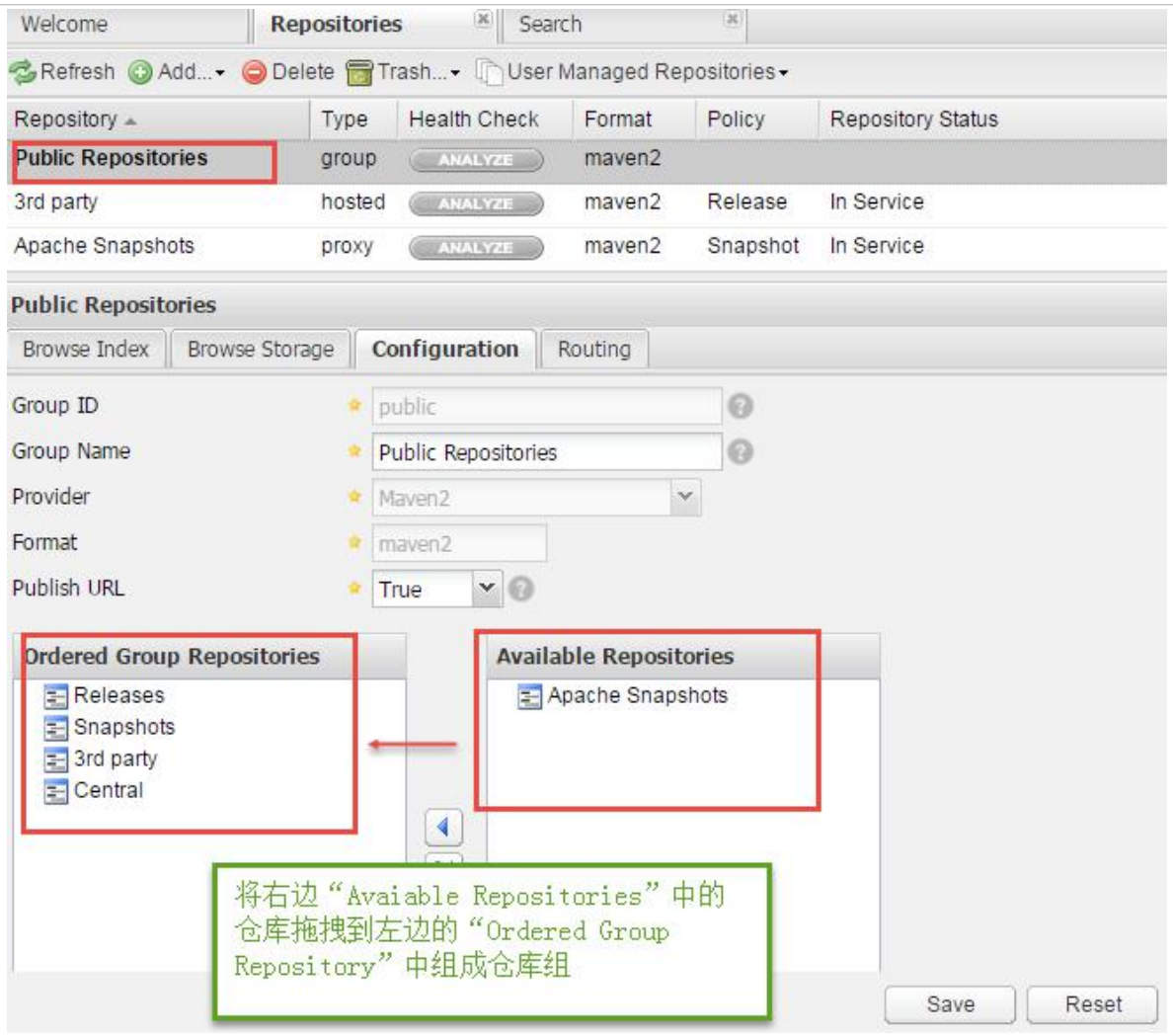
nexus 的仓库有 4 种类型：

Repository	Type
Public Repositories	group
3rd party	hosted
Apache Snapshots	proxy
Central	proxy
Central M1 shadow	virtual
Releases	hosted
Snapshots	hosted

1.hosted，宿主仓库，部署自己的 jar 到这个类型的仓库，包括 releases 和 snapshot 两部分，Releases 公司内部发布版本仓库、Snapshots 公司内部测试版本仓库

2.proxy，代理仓库，用于代理远程的公共仓库，如 maven 中央仓库，用户连接私服，私服自动去中央仓库下载 jar 包或者插件。

3.group，仓库组，用来合并多个 hosted/proxy 仓库，通常我们配置自己的 maven 连接仓库组。



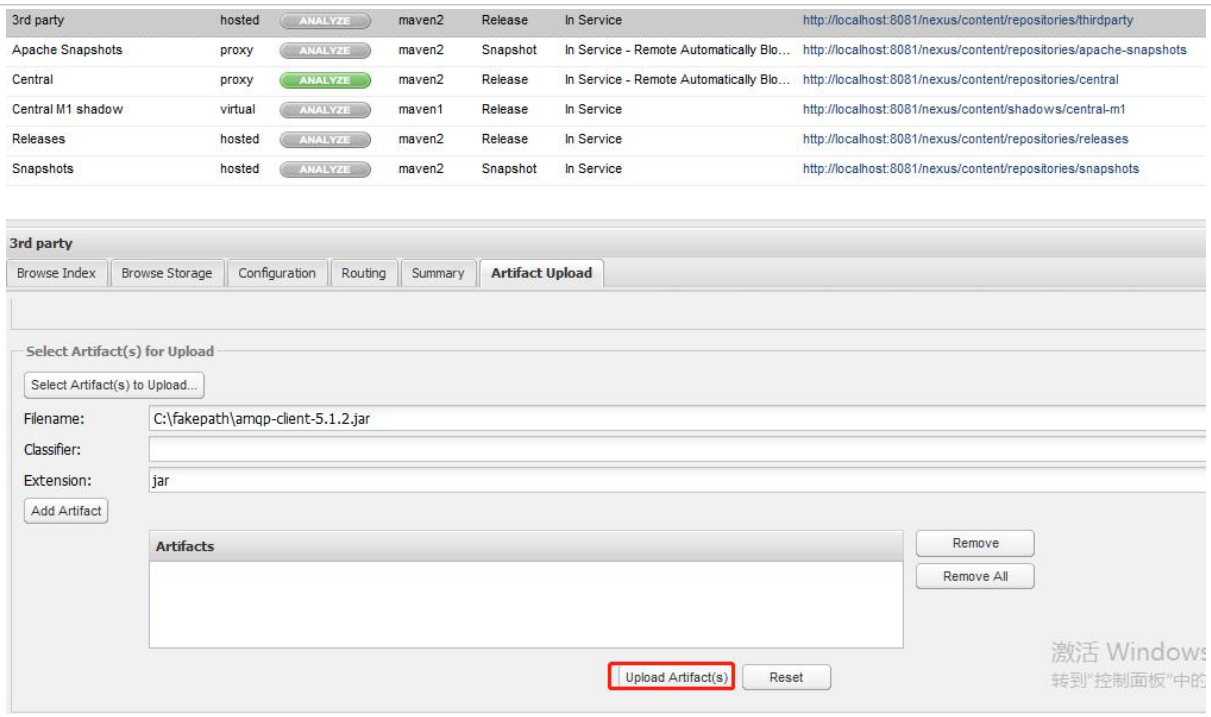
4.virtual(虚拟)：兼容 Maven1 版本的 jar 或者插件

5.3 将项目发布到私服

1、通过 mvndeploy 上传

需要配置 setting（server），pom（distributionManagement），如前文所示

2、Nexus 页面进行上传

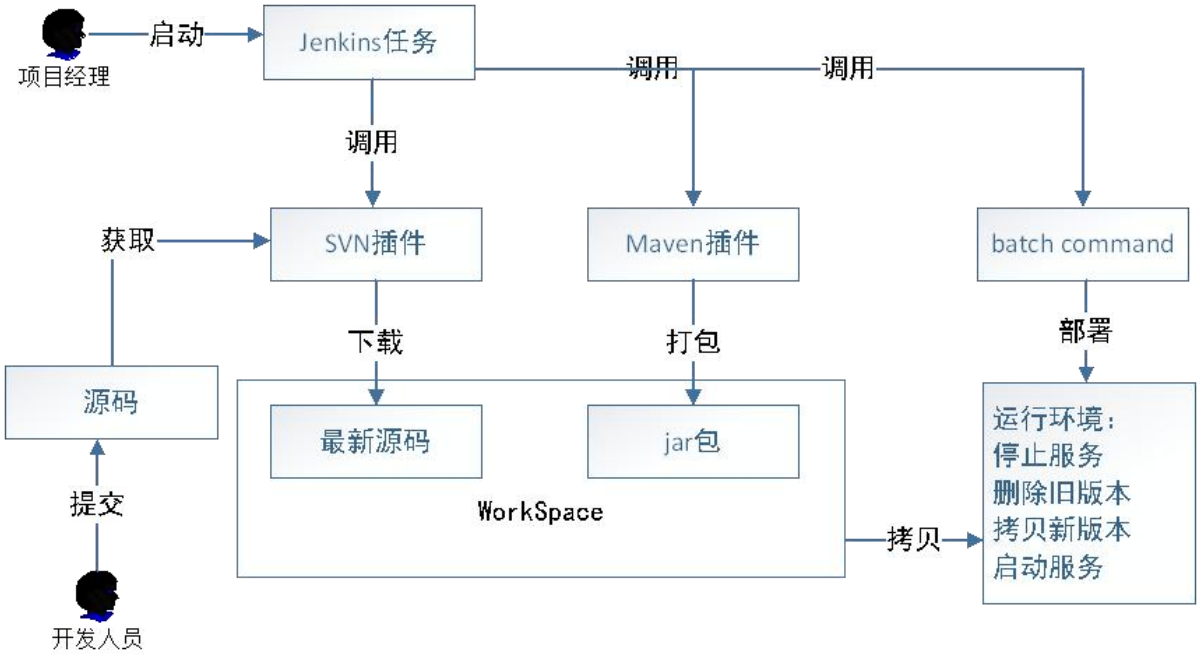


5.4 从私服下载 jar 包

setting.xml 中没有 repositories 的配置标签需要使用 profile 定义仓库，如前文所示。

6 Jenkins+maven 自动集成构建部署

思路：（Devops 思想）



至此，Maven 工具的基础知识就介绍完了。其实 Maven 工具就是我们的 java 开发更加规范化和自动化，Maven 的使用也远远不止这些需要我们在后续工作中继续摸索学习，但是掌握了这些基础知识，再去学习 Maven 的高级特性一定会更加轻松。最后感谢您阅读本文，希望对你有所帮助。

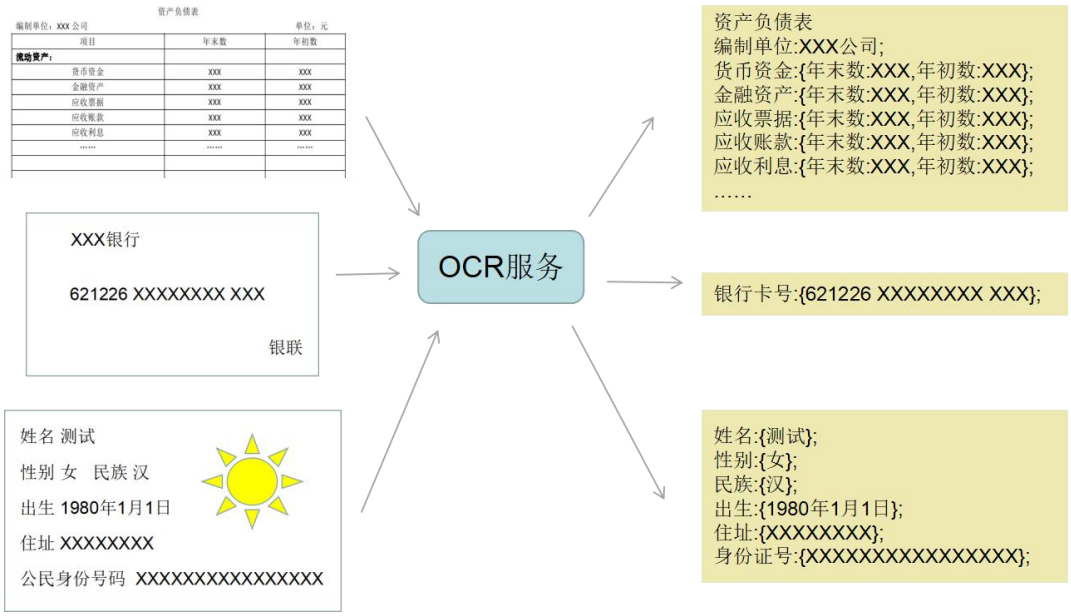
OCR 识别测试探索

◆作者：雷陈芳

题记：
随着人工智能技术的蓬勃发展，越来越多成熟的人工智能技术被应用于软件系统中，OCR 就是被广泛应用的技术之一。本文结合实际工作中的总结，探索如何针对 OCR 开展测试。

一、什么是 OCR

OCR（Optical Character Recognition 光学字符识别）是指通过对图像的分析处理将图像中的文字信息识别、提取并转化为计算机文字，简单来说是识别图片中的字符转化为可编辑文档，例如识别书籍扫描照片中的文字、识别身份证照片中的关键信息、识别银行卡照片中的银行卡号等。OCR 的一般过程为图像预处理、特征提取、文字区域检测、文本识别与输出。OCR 技术从文字类型的角度可分为印刷文字识别与手写文字识别，从识别目的的角度可分为通用 OCR 与专用 OCR，通用 OCR 是指对图片中所有文字和字符进行识别，并可返回文字对应位置信息；专用 OCR 是指对图片中的特定位置、特定信息进行识别提取，通常是针对票据、证件、牌照等制式票证图片。



二、OCR 的应用场景

随着 OCR 技术不断发展，其应用领域也得到相应的扩展，OCR 产品丰富多样，涵盖金融、教育、交通等诸多行业。

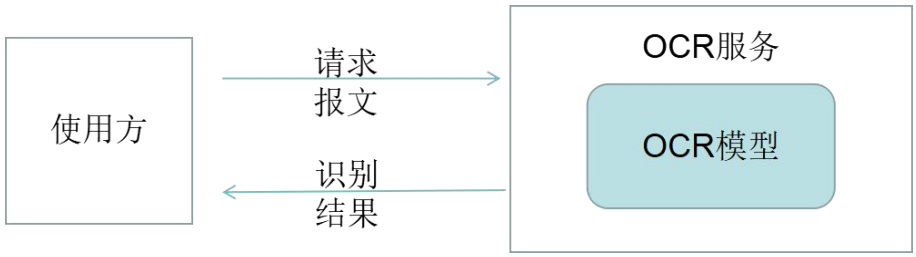
金融行业在柜面、运营、手机银行等业务领域中存在大量凭证识别需求，例如存折、存单、银行卡、营业执照、各类票据、报表等，通过 OCR 识别替代手工输入，可大幅提高业务办理效率，同时 OCR+人工复核提供了双重保障。

教育行业，通过 OCR 技术可实现自动化试卷分析与识别，提高试卷录入效率，并可辅助教师阅卷；题目图片、公式图片 OCR 结合搜索引擎实现快捷智能搜题；笔记图片 OCR 识别实现笔记电子化，方便存储、编辑、传输与查找；OCR 与自然语言处理技术结合，可实现写作辅导与纠错。

行政领域，身份证识别、税票识别、车牌识别等场景都离不开 OCR 技术的应用。

三、OCR 测试

OCR 模型是 OCR 功能的核心，经过大量图片及标注数据学习训练而成，在此基础上封装为 OCR 服务对外提供服务，通常以联机接口形式实现供其他系统调用。OCR 测试可分为两部分，一是对 OCR 模型识别的效果进行测试，判断其是否满足业务应用场景的要求，二是对 OCR 服务进行测试，包括功能测试、性能测试以及安全测试等。



1.OCR 模型准确率测试

与其他人工智能技术相同，OCR 模型无法保证 100%识别正确，模型达到预期准确率标准即测试通过。准确率标准需根据实际业务场景确定，例如客户使用 OCR 识别银行卡号进行绑卡操作的场景，直接面向客户，使用频率高，为保证用户体验，要求 OCR 识别的准确率较高；企业员工使用 OCR 识别辅助录入财税凭证的场景，由于财税数据通常经

多人审核校对，允许 OCR 识别准确率有所下降。此外准确率标准的确定还需结合当前技术现状，例如对与身份证、银行卡等标准证件的识别技术较为成熟，准确率可达到 99%以上，对于财税凭证等含有手写文字的票证，识别难度较大，当前技术仍在不断优化中，准确率较低。

OCR 模型准确率在实际测试场景中需定义具体的计算公式，对于专用 OCR 模型通常分别计算各个待识别字段的准确率，以身份证 OCR 为例，正面待识别字段包括姓名、性别、民族、出生日期、地址与身份证号，准确率=，识别正确是指无一个字符错漏。可为每个字段分别设置准确率标准，也可设置一个统一的标准，要求每个字段都达到这一标准。

待识别字段	测试集样本总数	识别正确样本数	准确率
姓名	1000	956	95.6%
性别	1000	998	99.8%
民族	1000	978	97.8%
出生日期	1000	986	98.6%
地址	1000	912	91.2%
身份证号	1000	996	99.6%

通用 OCR 模型准确率的计算有多种指标，针对字符、句子以及整张图片等不同粒度进行计算，常用指标包括字准率、字召回率、句准率、图片准确率、平均编辑距离等，具体计算方式如下表所示，测试时根据实际业务需求进行选择和定义。

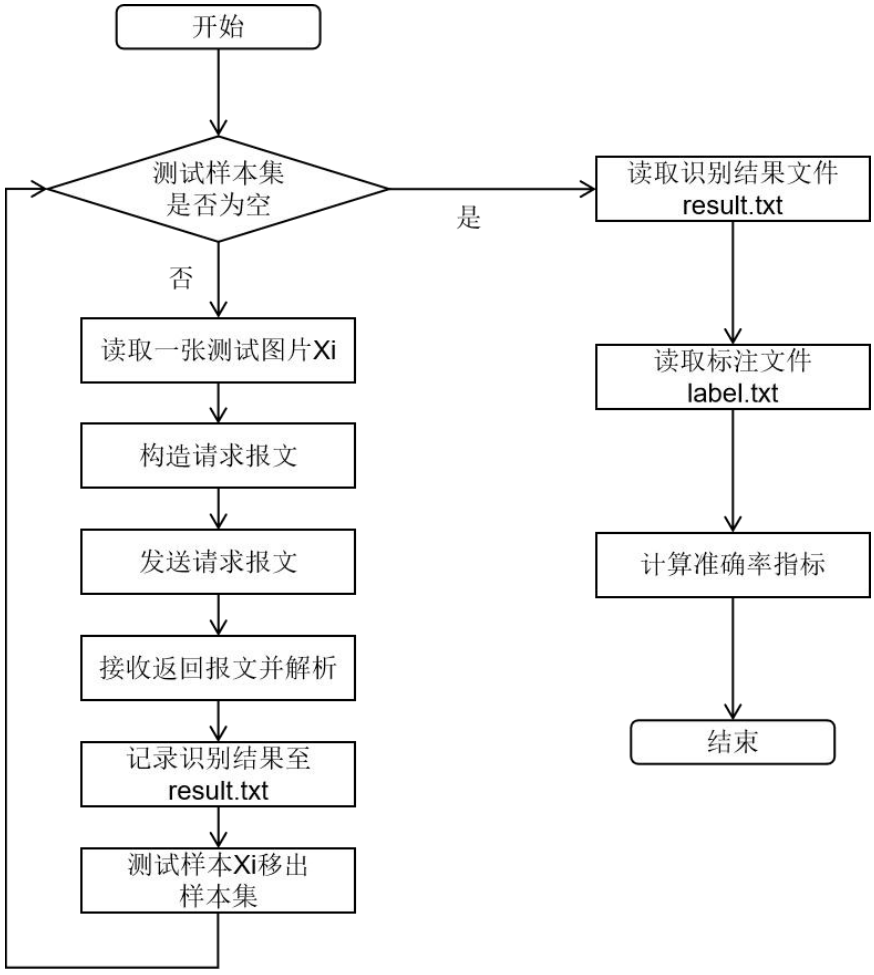
指标	计算公式
字准率	$\frac{\text{识别正确的字符数}}{\text{识别出的字符总数}}\times 100\%$
字召回率	$\frac{\text{识别正确的字符数}}{\text{测试集所有图片总字符数}}\times 100\%$
句准率	$\frac{\text{识别正确的句子数}}{\text{测试集所有图片句子总数}}\times 100\%$
图片准确率	$\frac{\text{完整识别正确的图片数}}{\text{测试集中图片总数}}\times 100\%$
平均编辑距离	编辑距离为字符串A到字符串B最少需要的操作次数，操作包括插入、修改、删除、增加一个字符。例如：test和tst的编辑距离为1，test和tast的编辑距离为1。计算句子平均编辑距离或图片平均编辑距离。

测试数据集对模型准确率测试十分重要，为了避免偶然性，测试样本数量需充足，样本图片的准备尽量贴近实际应用场景，考虑手写印刷、字体、光线、背景、拍照角度、

拍摄设备等多种因素，丰富样本集。此外可根据影响因素细分子样本集，分别计算子样本集准确率，对模型进行深度测试，例如可根据拍摄设备分为手机拍照、高拍仪拍照、扫描仪扫描分别计算不同场景准确率。

光线强弱	背景是否含干扰文字
拍照角度	是否多种语言文字参杂
是否旋转	文字排版
是否含手写	文字是否重叠
字体与背景颜色	字体
拍摄设备	图片格式
图片压缩、像素大小	其他

OCR 模型准确率测试可在模型训练完成后立刻开展，通过准确率测试后再进行模型服务封装，也可在服务封装完成后，在系统测试阶段进行测试，封装后开展的准确率测试可编写自动化测试脚本进行，流程如下图所示。



2.OCR 服务测试

OCR 服务测试包括功能测试、性能测试以及安全测试。功能测试内容可分为两部分，一部分是依据系统接口设计文档，对接口连通性、参数校验、返回交易状态码与提示等进行测试，常用 Postman、Jeter 等接口测试工具，构造请求报文，检查返回结果；另一部分是对业务需求中要求 OCR 服务提供的特定功能进行测试，例如图片格式转换、图片存储、请求限制、日志记录、权限控制等。

性能测试与传统功能相似，进行单交易基准测试、单交易负载测试、阶梯测试、疲劳测试等，监控 OCR 服务的响应时间、TPS、服务器资源使用情况，综合判断服务在性能方面是否能满足业务需要。需要注意的是目前 OCR 技术大多应用了 GPU，在性能测试环境的准备及资源监控中需注意 GPU 的配置与使用情况。

由于 OCR 服务多以接口方式调用，并且传输的图片及识别内容涉及身份证信息、银行卡信息、财务信息、税务信息、合同等敏感数据，使用方通常对服务安全方面的要求较高，安全测试内容主要包括传输加密、签名、越权、SQL 注入等。

最后以身份证 OCR 测试为例，若接口请求方式为 post/json，报文设计如下：

请求报文

```
{
  serviceId: idcard,
  image:图片base64,
  signature: 报文MD5值,
  encrypt:true/false
}
```

返回报文

```
{
  code:1/0[1为成功，0为错误],
  content:{
    姓名:XXX
    性别:XX
    民族:XX
    出生日期:XXXXXX
    地址:XXXXX
    身份证号:XXXXXXXXXXXXX
  }
}
```

则对应测试内容如下表所示：

准确性测试	OCR识别准确率计算
功能测试	1.正向连通性
	2.字段缺失、为空、为空格，eg:serviceId字段缺失、取值为"",取值为"",取值为idcad等
	3.图片base64非法（长度不对、篡改、特殊字符）
	4.图片格式png\jpg\gif……
	5.图片过大、过小、分辨率过高
性能测试	测试场景：负载测试、阶梯测试、疲劳测试 性能指标：TPS、响应时间 资源情况：CPU使用率、内存使用率
安全测试	1.signature值格式非法（长度不对、特殊字符）
	2.signature值错误（篡改）
	3.signature计算的key不对
	4.encrypt取值为true，报文加密
	5.encrypt取值为true，报文加密时key错误

以上是 OCR 通用的测试内容与方法，实际测试过程中仍需根据业务需求与应用场景进行针对性设计，保证测试充分性。

Testes Fixtures 从入门到应用就这么简单

◆ 作者：罗狮小钉

前言：在自动化测试中通常需要在正式执行自动化测试脚本前做一些初始化工作，例如在接口测试前做一些前置参数赋值，数据库操作，在 web 自动化中需要初始化浏览器驱动，在移动端测试中需要初始化模拟器参数配置等；而当自动化业务脚本执行完毕后，同样需要对测试后的环境做清理工作，例如参数还原或销毁，文件关闭，数据库还原恢复等扫尾工作。

1.Testes Fixtures 的优势

如果你熟悉 python unit test，那么一定对 setup()和 downhearted()这两个方法不陌生，这两个方法分别用于处理自动化脚本中初始化及后续清除的工作。今天我们要聊的话题不是 unit test，而是 python 另一个异常强大的测试框架 Testes 中的 fixtures 功能，它除了能够完美替代 unit test 中的 setup()和 downhearted()， 还具有以下这些优势：

• Testes fixtures 以模块化的方式实现，易于使用，无需涉及到该功能内部实现过程的学习，即 0 成本使用；

• Testes fixtures 可以指定作用域范畴，其默认范围是功能级别的，此外还可设置 module，class，session 作用域；

• Testes fixtures 提高测试代码的可读性，一致性，使代码更易于维护；

• Testes fixtures 的高可重用性，可用于单元测试，及复杂业务场景测试；

• Testes fixtures 通过装饰器以依赖注入的方式作用于测试用例，可以在 fixture 中编写前后置操作，通过 yield 进行区别前置操作和后置操作内容；

读到这里是不是有点犯晕乎，没关系，下面我们就一起来从 0-1 结合案例学习 Testes fixtures 在不同场景中的应用。

2.Testes Fixtures 在 Function 级别中的应用

Testes Fixtures 的默认作用范围是 function 级，我们通过下面的测试场景进行展示。

【测试场景】

（1）通过 Chrome 浏览器，打开百度主页，搜索“51testing”词条；

（2）通过 Chrome 浏览器，打开学掌门 at study 主页，搜索“自动化测试”词条。

【分析与设计】

（1）当前场景中共有两个测试任务，所以我们必须创建两个测试用例；

（2）两个测试任务共同点都需要开启 Chrome 浏览器，所以 Chrome Driver 的初始化可以单独拿出来作为 function 级别的 Testes Fixtures；

（3）两个测试用例只需在执行前调用这个 Testes Fixtures 即可。

【Testes 脚本实现】

（1）相关库的导入

```
1  import pytest
2  from selenium import webdriver
3  from selenium.webdriver.common.keys import Keys
4  from selenium.webdriver.common.by import By
5  import time
```

（2）创建 Testes Fixtures

```
8  #Function 级别的 fixture
9  #用于初始化chrome driver
10 @pytest.fixture()
11 def chrome_driver_init():
12     print("\n初始化chrome_driver")
13     chrome_driver = webdriver.Chrome()
14     return chrome_driver
```

（3）创建测试用例，用于测试百度搜索

```
16      # 测试用例1：打开百度主页，搜索“51testing”词条
17      def test_open_baidu(chrome_driver_init):
18          print("执行百度搜索")
19          driver = chrome_driver_init
20          # 打开百度主页
21          driver.get('https://www.baidu.com/')
22          driver.maximize_window()
23          time.sleep(2)
24          # 搜索“51testing”词条
25          ele = driver.find_element(By.XPATH,"//*[@id='kw']")
26          ele.send_keys("51testing")
27          time.sleep(2)
28          ele.send_keys(Keys.RETURN)
29          time.sleep(2)
30          driver.quit()
31          print("完成百度搜索")
```

fixture调用

（4）创建测试用例，用于测试 at study 搜索

```
33      # 测试用例2： 打开学掌门atstudy主页，搜索“自动化测试”词条
34      def test_open_atstudy(chrome_driver_init):
35          print("执行atstudy搜索")
36          driver = chrome_driver_init
37          # 打开atsutdy主页
38          driver.get('https://www.atstudy.com/')
39          driver.maximize_window()
40          time.sleep(2)
41          # 搜索“自动化测试”相关课程
42          ele = driver.find_element(By.XPATH,"//*[@id='__layout']/div/div[1]/div/div[1]/div[2]/div[1]/input")
43          ele.send_keys("自动化测试")
44          time.sleep(2)
45          ele.send_keys(Keys.RETURN)
46          time.sleep(2)
47          driver.quit()
48          print("完成atsutdy搜索")
```

fixture调用

到此，该场景的自动化测试脚本编写完成，是不是特别简单，只需将共同的部分单独做成一个 fixture，在有需要的测试用例中，将 fixture 名字作为该测试用例方法的参数，直接传入即可。

当执行测试用例的时候，Testes 会自行判断当前测试用例中有没有 fixture 的植入，如果有，则先运行 fixture 中的脚本，再运行对应的测试用例。

【执行结果】

如图所示，2 个测试用例都自行调用 fixture，对 chrome driver 进行了初始化。

```
============================= test session starts ==============================
platform win32 -- Python 3.9.1, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: D:\
plugins: Faker-6.4.1, forked-1.3.0, xdist-2.2.1collected 2 items

fixture_00.py
初始化chrome_driver
.执行百度搜索
完成百度搜索

初始化chrome_driver
.执行atstudy搜索
完成atsutdy搜索
                                                                      [100%]

============================== 2 passed in 37.71s ==============================
Process finished with exit code 0
```

测试用例1

测试用例2

3.Testes Fixtures 在 Class 级别中的应用

我们发现在上面应用中，虽然 Function 级别的 Testes Fixtures 完成了自动化脚本的前置步骤——chore driver 初始化，但两个测试用例还存在重复的部分，即关闭浏览器的收尾工作，那么 Testes Fixtures 是不是也能一并解决后置操作呢？答案是显而易见的。下面我们就来看一下 Testes Fixtures 在 Class 级别的场景应用。

【测试场景】

同上

【分析与设计】

（1）自动化测试前置操作：Chrome Driver 初始化；后置操作：关闭浏览器

（2）通过 Class 级别的 Testes Fixtures 统一实现前后置操作；

（3）通过“@Testes.mark.fixtures”注释实现 Testes Fixtures 的调用。

【Testes 脚本实现】

（1）相关库的导入

```
import pytest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time
```

（2）创建 Testes Fixtures

```
#Class级别的 fixture
#用于初始化chrome driver, 关闭浏览器
@pytest.fixture(scope="class")
def chrome_driver_init(request):
    chrome_driver = webdriver.Chrome()
    request.cls.driver = chrome_driver
    yield
    chrome_driver.quit()
```

“request.c ls.driver”可以理解为当前这个?Chrome Cabdriver 的实例；

“yield”之前的代码是前置操作，之后的代码是后置操作，当这个 Fixtures 被调用时，yield 部分会自动识别对应的测试用例来执行。

（3）创建测试类，其中包括两个测试方法（分别用于百度和 at study 搜索）

```
@pytest.mark.usefixtures("chrome_driver_init")
class Test_01:

    def test_open_baidu(self):
        print("执行百度搜索")
        # 打开百度主页
        self.driver.get('https://www.baidu.com/')
        self.driver.maximize_window()
        time.sleep(2)
        # 搜索“51testing”词条
        ele = self.driver.find_element(By.XPATH,"//*[@id='kw']")
        ele.send_keys("51testing")
        time.sleep(2)
        ele.send_keys(Keys.RETURN)
        time.sleep(2)
        print("完成百度搜索")
```

fixtures调用

```
37    def test_open_atstudy(self):
38        print("执行atstudy搜索")
39        # 打开atsutdy主页
40        self.driver.get('https://www.atstudy.com/')
41        self.driver.maximize_window()
42        time.sleep(2)
43        # 搜索“自动化测试”相关课程
44        ele = self.driver.find_element(By.XPATH,"//*[@id='__layout']/div/div[1]/div/div[1]/div[2]/div[1]/input")
45        ele.send_keys("自动化测试")
46        time.sleep(2)
47        ele.send_keys(Keys.RETURN)
48        time.sleep(2)
49        print("完成atsutdy搜索")
```

在测试类前，通过“@Testes.mark.fixtures("chrome_driver_i nit")”调用 Class 级别的 Fixtures。

【执行结果】

如图所示，2 个测试用例都自行调用 fixture，对 chrome driver 进行了初始化，在执行完测试用例后，关闭浏览器进程。

```
============================= test session starts ==============================
platform win32 -- Python 3.9.1, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir:
plugins: Faker-6.4.1, forked-1.3.0, xdist-2.2.1collected 2 items

fixture_01.py                                                          [100%]

============================== 2 passed in 26.09s ==============================.执行百度搜索
完成百度搜索
.执行atstudy搜索
完成atsutdy搜索
```

4.Testes Fixtures 在 Class 级别中的应用（多浏览器）

在 web 自动化测试中，最常见的莫过于同一个业务功能需要兼容多个浏览器，遇到这样的兼容性场景，借助于 Testes Fixtures 又该如何实现呢？

【测试场景】

同上

附加条件：每个业务功能都必须在 Chrome 和 Edge 上执行通过。

【分析与设计】

最简单的实现方式：增加一个 Class 级别的 Testes Fixtures 专门用于 Edge Cabdriver 的初始化和关闭。

【Testes 脚本实现】

（1）相关库的导入

```
import pytest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time
```

（2）分别创建 2 个 Testes Fixtures 用于 Chrome 和 Edge

```
#Class级别的 fixture
#用于初始化chrome driver, 关闭浏览器
@pytest.fixture(scope="class")
def chrome_driver_init(request):
    print("\n chrome driver 初始化")
    chrome_driver = webdriver.Chrome()
    request.cls.driver = chrome_driver
    yield
    chrome_driver.quit()

#Class级别的 fixture
#用于初始化edge driver, 关闭浏览器
@pytest.fixture(scope="class")
def edge_driver_init(request):
    print("\n edge driver 初始化")
    edge_driver = webdriver.Edge()
    request.cls.driver = edge_driver
    yield
    edge_driver.quit()
```

（3）创建运行在 chrome 浏览器上的测试类

```
@pytest.mark.usefixtures("chrome_driver_init")
class Test_01:
    def test_open_baidu(self):
        print("执行百度搜索")
        # 打开百度主页
        self.driver.get('https://www.baidu.com/')
        self.driver.maximize_window()
        time.sleep(2)
        # 搜索“51testing”词条
        ele = self.driver.find_element(By.XPATH,"//*[@id='kw']")
        ele.send_keys("51testing")
        time.sleep(2)
        ele.send_keys(Keys.RETURN)
        time.sleep(2)
        print("完成百度搜索")
    def test_open_atstudy(self):
        print("执行atstudy搜索")
        # 打开atsutdy主页
        self.driver.get('https://www.atstudy.com/')
        self.driver.maximize_window()
        time.sleep(2)
        # 搜索“自动化测试”相关课程
        ele = self.driver.find_element(By.XPATH,"//*[@id='__layout']/div/div[1]/div/div[1]/div[2]/div[1]/input")
        ele.send_keys("自动化测试")
        time.sleep(2)
        ele.send_keys(Keys.RETURN)
        time.sleep(2)
        print("完成atsutdy搜索")
```

调用应用于chrome浏览器的那个fixtures

（4）创建运行在 edge 浏览器上的测试类

```
@pytest.mark.usefixtures("edge_driver_init")
class Test_02:
    def test_open_baidu(self):
        print("执行百度搜索")
        # 打开百度主页
        self.driver.get('https://www.baidu.com/')
        self.driver.maximize_window()
        time.sleep(2)
        # 搜索“51testing”词条
        ele = self.driver.find_element(By.XPATH,"//*[@id='kw']")
        ele.send_keys("51testing")
        time.sleep(2)
        ele.send_keys(Keys.RETURN)
        time.sleep(2)
        print("完成百度搜索")
    def test_open_atstudy(self):
        print("执行atstudy搜索")
        # 打开atsutdy主页
        self.driver.get('https://www.atstudy.com/')
        self.driver.maximize_window()
        time.sleep(2)
        # 搜索“自动化测试”相关课程
        ele = self.driver.find_element(By.XPATH,"//*[@id='__layout']/div/div[1]/div/div[1]/div[2]/div[1]/input")
        ele.send_keys("自动化测试")
        time.sleep(2)
        ele.send_keys(Keys.RETURN)
        time.sleep(2)
        print("完成atsutdy搜索")
```

调用应用于edge浏览器的那个fixtures

我们发现这两个测试类（Test_01, Test_02）除了开头调用不同的 fixtures 以外，其余都是一模一样的，貌似挺繁琐的，是不是该优化一下呢？别急，我们先执行一下上面的脚本，看看是否如我们所愿，百度和 atstudy的搜索功能分别在 chrome 和 edge 两个浏览器上顺利运行。

5.Testes Fixtures 参数化在 Class 级别中的应用（多浏览器）

从上面场景得知，虽然成功实现了需求，但脚本存在大量的冗余，同样的查询脚本由于调用的浏览器驱动不同，需要重复写两次，如果有更多的待测浏览器版本呢？难不成要重复 N 多遍？答案当然是“NO”!

Testes Fixtures 中的参数化提供了很好的解决方案。

【测试场景】

同上

附加条件：实现 Fixtures 参数化。

【分析与设计】

浏览器名称以参数化列表的形式，传入 Fixtures。

【Testes 脚本实现】

（1）相关库的导入

```
import pytest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time
```

（2）创建 1 个 Testes Fixtures，Chrome 和 Edge 以参数列表形式声明

```
@pytest.fixture(params=["chrome", "edge"],scope="class")
def driver_init(request):
    if request.param == "chrome":
        print("\n chrome driver 初始化")
        web_driver = webdriver.Chrome()
    if request.param == "edge":
        print("\n edge driver 初始化")
        web_driver = webdriver.Edge()
    request.cls.driver = web_driver
    yield
    web_driver.quit()
```

根据浏览器名判断使用哪个驱动

（3）创建测试类

```
@pytest.mark.usefixtures("driver_init")
class Test_01:
    def test_open_baidu(self):
        print("执行百度搜索")
        # 打开百度主页
        self.driver.get('https://www.baidu.com/')
        self.driver.maximize_window()
        time.sleep(2)
        # 搜索“51testing”词条
        ele = self.driver.find_element(By.XPATH,"//*[@id='kw']")
        ele.send_keys("51testing")
        time.sleep(2)
        ele.send_keys(Keys.RETURN)
        time.sleep(2)
        print("完成百度搜索")
```

调用fixtures

```
def test_open_atstudy(self):
    print("执行atstudy搜索")
    # 打开atsutdy主页
    self.driver.get('https://www.atstudy.com/')
    self.driver.maximize_window()
    time.sleep(2)
    # 搜索“自动化测试”相关课程
    ele = self.driver.find_element(By.XPATH,"//*[@id='__layout']/div/div[1]/div/div[1]/div[2]/div[1]/input")
    ele.send_keys("自动化测试")
    time.sleep(2)
    ele.send_keys(Keys.RETURN)
    time.sleep(2)
    print("完成atsutdy搜索")
```

这里我们只需要创建一次测试脚本，调用带有参数化的 fixtures，Testes Fixtures 机制就能自动识别其是否包含参数化，按照参数列表逐一将每个参数待入测试脚本进行运行，直到所有的参数都参与执行完毕。

【执行结果】

如图所示，测试脚本分别在 chrome 和 edge 两个浏览器中执行完毕，Testes Fixtures 参数化机制大大提高了脚本编写效率，有效消除代码冗余。

```
============================= test session starts =============================
platform win32 -- Python 3.9.1, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: [redacted]
plugins: Faker-6.4.1, forked-1.3.0, xdist-2.2.1collected 4 items

fixture_03.py                                                        [100%]

============================== 4 passed in 53.10s ==============================
 chrome driver 初始化
.执行百度搜索
完成百度搜索
.执行atstudy搜索
完成atsutdy搜索

 edge driver 初始化
.执行百度搜索
完成百度搜索
.执行atstudy搜索
完成atsutdy搜索

Process finished with exit code 0
```

6.Testes Fixtures 实现多个测试脚本间的共享

Testes Fixtures 参数化有效提高测试脚本编写效率，从测试角度而言，即实现了数据驱动的效果 —— 同一组测试数据集，共享一个测试脚本。那么有没有可能做到同一个 Fixtures， 共享与多个测试文件呢？答案当然是“YES”!

Testes Fixtures 机制中约定了可以将 fixtures 独立出来，放在固定的【contest.oy】中。就可以实现 fixtures 在多个测试文件中共享的效果。

【测试场景】

同上

附加条件：实现 fixtures 独立，且共享与多个测试文件。

【分析与设计】

将上面的 fixtures 从脚本中剥离出去，放入 contest.oy 文件。

【Testes 脚本实现】

（1）创建 contest.oy 文件，将上个场景中的 fixtures 从脚本中删除，放到新建的 contest.oy 文件中，并且这个文件和测试脚本文件在同一目录下。

conftest.py

```
import pytest
from selenium import webdriver


@pytest.fixture(params=["chrome", "edge"],scope="class")
def driver_init(request):
    if request.param == "chrome":
        print("\n chrome driver 初始化")
        web_driver = webdriver.Chrome()
    if request.param == "edge":
        print("\n edge driver 初始化")
        web_driver = webdriver.Edge()
    request.cls.driver = web_driver
    yield
    web_driver.quit()
```

（2）创建测试脚本

```
import pytest
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time
```

```
@pytest.mark.usefixtures("driver_init")
class Test_01:

    def test_open_baidu(self):
        print("执行百度搜索")
        # 打开百度主页
        self.driver.get('https://www.baidu.com/')
        self.driver.maximize_window()
        time.sleep(2)
        # 搜索“51testing”词条
        ele = self.driver.find_element(By.XPATH,"//*[@id='kw']")
        ele.send_keys("51testing")
        time.sleep(2)
        ele.send_keys(Keys.RETURN)
        time.sleep(2)
        print("完成百度搜索")
```

调用fixtures

```
26      def test_open_atstudy(self):
27          print("执行atstudy搜索")
28          # 打开atsutdy主页
29          self.driver.get('https://www.atstudy.com/')
30          self.driver.maximize_window()
31          time.sleep(2)
32          # 搜索“自动化测试”相关课程
33          ele = self.driver.find_element(By.XPATH,"//*[@id='__layout']/div/div[1]/div/div[1]/div[2]/div[1]/input")
34          ele.send_keys("自动化测试")
35          time.sleep(2)
36          ele.send_keys(Keys.RETURN)
37          time.sleep(2)
38          print("完成atsutdy搜索")
```

这里我们只需要把之前的测试脚本（非 fixtures 部分）直接拷贝即可。

【执行结果】

如图所示，和之前的效果一样，所不同的是，这次我们将 fixtures 单独放置在 contest.oy 中，在运行的时候，如果遇到 fixtures 调用，而当前脚本中没有可用的 fixtures，就会自动识别当前脚本同目录下是否有 contest.oy，如果有就自动去这个文件中查找被调用的 fixtures。

独立的 contest.oy 有助于在多个测试文件中实现 fixtures 共享。

```
============================= test session starts =============================
platform win32 -- Python 3.9.1, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: [redacted]
plugins: Faker-6.4.1, forked-1.3.0, xdist-2.2.1collected 4 items

fixture_04.py
 chrome driver 初始化
.执行百度搜索
完成百度搜索
.执行atstudy搜索
完成atsutdy搜索

 edge driver 初始化
.执行百度搜索
完成百度搜索
.执行atstudy搜索
完成atsutdy搜索
                                                                    [100%]
```

7.总结

以上基于 Testes 中 Fixtures 进行了详细介绍，由浅入深探索了 Fixtures 的在测试脚本中的灵活性，结合不同场景帮助大家掌握其常见应用，同时提供了完整的代码示例，希望能够给大家的测试工作带来帮助，感兴趣的读者不妨一试。

Python 脚本实现批量生成百万条 excel 测试数据

◆作者：桃子

背景介绍：最近在测试的时候，比如导入这个功能，想要测试上传大数据量文件是否好用有两个点需要解决，一是怎么生成大量的 excel 文件，二是如何保证生成的文件格式与项目要求的格式一致。有同学可能想到从晚上下载大容量的文件，但是格式这一块无法保证。还有一种笨方法可以复制粘贴，粘贴几十条数据还可以，大容量这块无法保证，这时候脚本的作用就体现了，利用程序的思想解决问题：其实就是 excel 写入数据的过程。

接下来看看实现效果：

部门ID	部门名称	部门顺序	负责人	联系电话	邮箱	状态	上级部门名称	部门编号	部门简称
300	开发	54				正常		26	
301	开发	47				正常		9	
302	测试	29				正常		49	
303	生产	39				正常		51	
304	生产	11				正常		9	
305	开发	44				正常		63	
306	生产	87				正常		64	
307	生产	48				正常		25	
308	开发	85				正常		99	
309	生产	31				正常		42	

实现功能拆分：

可以按照实际的场景想象一下，然后用机器的语言实现。

实际场景： 我们需要创建一个 excel 文件，然后写好标题，之后按照标题填写内容，写好之后保存。

拆分代码功能区：

1.创建一个 excel 对象->创建 sheet

2.添加字段,行 列 名称

3.写入编号字段数据

4.保存

上面稍微复杂一点的时步骤 3，接下来我们按照功能点一行一行填写代码，按功能点填充代码：

1.导入相关包

```
import xlwt #导入 xlwt 函数，专门操作 excel 写文件的函数
import random #导入随机函数
```

2.创建工作簿对象->创建 sheet

```
#创建工作簿对象
book=xlwt.workbook(encoding='utf-8')
#创建 sheet
sheet=book.add_sheet('test',cell_overwrite_ok=True)
```

创建工作使用 workbook()函数；

创建表使用 book.add_sheet()函数,其中一个参数为名称,第二个参数为是否支持重写。

3.添加标题内容：字段,行 列 名称

```
sheet.write(0, 0, '部门 ID')
sheet.write(0, 1, '部门名称')
sheet.write(0, 2, '部门顺序')
sheet.write(0, 3, '负责人')
sheet.write(0, 4, '联系电话')
sheet.write(0, 5, '邮箱')
sheet.write(0, 6, '状态')
sheet.write(0, 7, '上级部门名称')
```

```
sheet.write(0, 8, '部门编号')
sheet.write(0, 9, '部门简称')
```

使用 sheet.write(行，列，内容)函数进行填充，一般标题都是第 0 行。

4.写入编号字段数据

```
for i in range(10):
sheet.write(i+1,0,300+id)
sheet.write(i+1,1,random.choice([测试],[开发],[产品]))
sheet.write(i+1,7,'齐大山铁矿')
sheet.write(id + 1, 6, "正常")
sheet.write(id + 1, 8, random.randint(1, 100))
```

for 循环，逐行写入内容：

random.choice()函数实现随机选择字符串

random.randint(1,100)实现 1,100 随机数

1.保存

```
book.save(r'.\部门信息.xlsx')
```

实现代码：

```
# -*- coding: utf-8 -*-
# @Time       : 2018/12/6 17:10
# @Author     : taozi
# @Disc:      : 生成 10000 条 Excel 数据
# @File       : 1000data.py
# @Software: PyCharm
import xlrd ,xlwt
import random
"""创建一个 excel 对象"""
book = xlwt.Workbook(encoding='utf-8',style_compression=0)
"""创建 sheet"""
sheet = book.add_sheet('test',cell_overwrite_ok=True)
```

```
    """添加字段,行 列 名称 """
    sheet.write(0, 0, '部门 ID')
    sheet.write(0, 1, '部门名称')
    sheet.write(0, 2, '部门顺序')
    sheet.write(0, 3, '负责人')
    sheet.write(0, 4, '联系电话')
    sheet.write(0, 5, '邮箱')
    sheet.write(0, 6, '状态')
    sheet.write(0, 7, '上级部门名称')
    sheet.write(0, 8, '部门编号')
    sheet.write(0, 9, '部门简称')
    """写入编号字段数据"""
    #random.uniform(x,y) 实现在 x，y 之间生成随机数
    #round(x)函数 返回浮点数 x 的四舍五入值
    #random.randint(0x4e00, 0x9fbf) 生成随机中文
    #random.randint(1,100) 生成 1-100 随机正整数
    # 随机选取字符串：
    #random.choice(['剪刀', '石头', '布'])
    for id in range(60000):
        sheet.write(id + 1, 0, id+300)
        sheet.write(id + 1, 1, random.choice(['测试', '开发', '生产']))
        sheet.write(id + 1, 2, random.randint(1, 100))
        sheet.write(id + 1, 6, "正常")
        sheet.write(id + 1, 7,  "1111")
        sheet.write(id + 1, 8, random.randint(1, 100))
    book.save(r'.\部门信息.xls')
```

到上面为止，我们可以新建一个 sheet 页的数据，基本上 16M 左右。但是我想要生成多个 sheet 的数据，造成文件过大改如何改写呢？

我把写数据的代码封装成一个函数。

for 循环生成 sheet 文件，新建文件的同时写入数据，这样你就可以随意自己改写要

生成多少条数据了。改写后的代码如下：

```
# -*- coding: utf-8 -*-
# @Time       : 2018/12/6 17:10
# @Author     : taozi
# @Disc:      : 生成百万条条 Excel 数据
# @File       : 1000data.py
# @Software: PyCharm
import xlrd ,xlwt
import random
"""创建一个 excel 对象"""
book = xlwt.Workbook(encoding='utf-8',style_compression=0)
"""写入编号字段数据"""
def write():
    sheet.write(0, 0, '部门 ID')
    sheet.write(0, 1, '部门名称')
    sheet.write(0, 2, '部门顺序')
    sheet.write(0, 3, '负责人')
    sheet.write(0, 4, '联系电话')
    sheet.write(0, 5, '邮箱')
    sheet.write(0, 6, '状态')
    sheet.write(0, 7, '上级部门名称')
    sheet.write(0, 8, '部门编号')
    sheet.write(0, 9, '部门简称')
    for id in range(60000):
        sheet.write(id + 1, 0, id+300)
        sheet.write(id + 1, 1, random.choice(['测试', '开发', '生产']))
        sheet.write(id + 1, 2, random.randint(1, 100))
        sheet.write(id + 1, 6, "正常")
        sheet.write(id + 1, 7,   "1111")
        sheet.write(id + 1, 8, random.randint(1, 100))
"""创建 sheet 并调用 write 函数写入数据"""
for j in range(3):
```

```
        sheet = book.add_sheet('test'+str(j),cell_overwrite_ok=True)
        write()

book.save(r'.\部门信息.xls')
```

细心的同学可以发现，部门 id 是唯一的字段，我们这样添加内容唯一。那么如何生成唯一的部门 id 值呢？

我们都知道 random 可以随机生成数，那么这个数怎么保证唯一呢？其中有一个函数 sample 就可以实现。把可选的字符串定义一个变量，然后从中选取 x 位字符。代码如下：

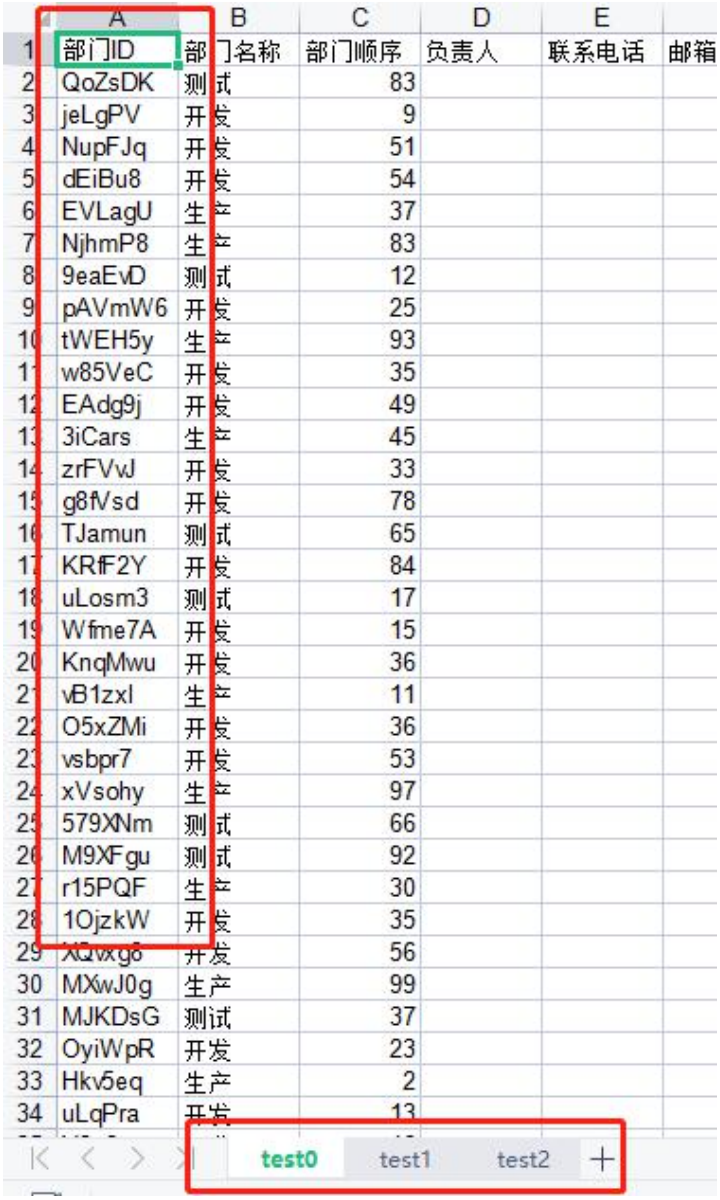
```
strings = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
random_str = random.sample(strings, 6)
```

最终代码：

```
# -*- coding: utf-8 -*-
# @Time       : 2018/12/6 17:10
# @Author     : taozi
# @Disc:      : 生成百万条条 Excel 数据
# @File       : 1000data.py
# @Software: PyCharm
import xlrd ,xlwt
import random
"""创建一个 excel 对象"""
book = xlwt.Workbook(encoding='utf-8',style_compression=0)
"""写入编号字段数据"""
def write():
        sheet.write(0, 0, '部门 ID')
        sheet.write(0, 1, '部门名称')
        sheet.write(0, 2, '部门顺序')
        sheet.write(0, 3, '负责人')
        sheet.write(0, 4, '联系电话')
        sheet.write(0, 5, '邮箱')
```

```
            sheet.write(0, 6, '状态')
            sheet.write(0, 7, '上级部门名称')
            sheet.write(0, 8, '部门编号')
            sheet.write(0, 9, '部门简称')

            for id in range(60000):
                strings =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
                random_str = random.sample(strings, 6)
                sheet.write(id + 1, 0, random_str)
                sheet.write(id + 1, 1, random.choice(['测试', '开发', '生产']))
                sheet.write(id + 1, 2, random.randint(1, 100))
                sheet.write(id + 1, 6, "正常")
                sheet.write(id + 1, 7,   "1111")
                sheet.write(id + 1, 8, random.randint(1, 100))
        """创建 sheet 并调用 write 函数写入数据"""
        for j in range(3):
            sheet = book.add_sheet('test'+str(j),cell_overwrite_ok=True)
            write()

        book.save(r'.\部门信息.xls')
```

实现效果：



	A	B	C	D	E	
1	部门ID	部门名称	部门顺序	负责人	联系电话	邮箱
2	QoZsDK	测试	83			
3	jeLgPV	开发	9			
4	NupFJq	开发	51			
5	dEiBu8	开发	54			
6	EVLagU	生产	37			
7	NjhmP8	生产	83			
8	9eaEvD	测试	12			
9	pAVmW6	开发	25			
10	tWEH5y	生产	93			
11	w85VeC	开发	35			
12	EAdg9j	开发	49			
13	3iCars	生产	45			
14	zrFVvJ	开发	33			
15	g8fVsd	开发	78			
16	TJamun	测试	65			
17	KRfF2Y	开发	84			
18	uLosm3	测试	17			
19	Wfme7A	开发	15			
20	KnqMwu	开发	36			
21	vB1zxl	生产	11			
22	O5xZMi	开发	36			
23	vsbpr7	开发	53			
24	xVsohy	生产	97			
25	579XNm	测试	66			
26	M9XFgu	测试	92			
27	r15PQF	生产	30			
28	1OjzkW	开发	35			
29	XQvxg8	开发	56			
30	MXwJ0g	生产	99			
31	MJKDsG	测试	37			
32	OyiWpR	开发	23			
33	Hkv5eq	生产	2			
34	uLqPra	开发	13			

test0 test1 test2

总结：

回归一些操作 excel 写入文件的步骤：

创建工作簿->创建 sheet->写入字段名称->写入内容，分别用到了 Workbook()，add_sheet(), write()函数

同时过程中使用 for 循环创建表及函数的调用，再有就是利用 random.sample()函数生成唯一字段。

Python 操作数据库完成接口测试

◆ 作者：测试安静

前言：数据库的操作在测试工作中也是经常使用的，通过一些一些工具来操作数据库的方法大家都应该了解，那么 python 操作数据库的大家了解吗？今天安静通过本篇文章介绍下如何通过 python 来操作 mysql 数据库。

pymysql

pymysql 属于 python 的一个第三方库，用例操作 mysql 数据库。

安装：pip install pymysql

源码地址：https://github.com/PyMySQL/PyMySQL/

连接数据库

操作数据库前肯定需要连接数据库了，pymysql 通过 connext 的方法用来连接数据库，其中需要一些数据库的参数内容如：登录账号，登录密码。数据库的地址（这里安静使用的是本地自己安装的数据库），以及需要连接哪一个数据库

```
import pymysql
# 连接数据库
count = pymysql.connect(
                host = 'localhost',       # 数据库地址
                port = 3306,              # 数据库端口号
                user='root',              # 数据库账号
                password='821006052',     # 数据库密码
```

```
                db= 'anjing_test',   # 数据库表名
                charset = 'gbk'       # 中文乱码
)
# 完成 mysql 数据库实例化
db = count.cursor()
```

查找数据

查询内容肯定需要执行 sql 内容 pymysql 这里通过 execute 的方法来执行 sql 命令并返回一共有多少数据，然后在通过 fetchall()来显示 sql 查询结果的所有内容：

```
import pymysql
# 连接数据库
count = pymysql.connect(
                host = 'localhost',       # 数据库地址
                port = 3306,              # 数据库端口号
                user='root',              # 数据库账号
                password='821006052',     # 数据库密码
                db= 'anjing_test',   # 数据库表名
                charset = 'gbk'       # 中文乱码
)
# 完成 mysql 数据库实例化
db = count.cursor()
# sql 语句
sql = 'select * from weather'
# 执行 sql
a = db.execute(sql)
# 查找所以内容
result = db.fetchall()
print(result)
```

通过执行发现已经讲我们这个表中的所有数据全部都查询出来了，安静这里只在数据库中添加了 2 条数据：

```python
import pymysql
# 连接数据库
count = pymysql.connect(
            host='localhost',      # 数据库地址
            port=3306,         # 数据库端口号
            user='root',         # 数据库账号
            password='821006052',      # 数据库密码
            db='anjing_test',  # 数据库表名
            charset='gbk'      # 中文乱码
)
# 完成mysql数据库实例化
db = count.cursor()
# sql语句
sql = 'select * from weather'
# 执行sql
a = db.execute(sql)
# 查找所以内容
result = db.fetchall()
print(result)
```

```
w12 (1)
D:\Python\python.exe E:/web/w12.py
(('上海', '331eab8f3481f37868378fcdc76cb7cd', 1), ('北京', '331eab8f3481f37868378fcdc76cb7cd', 2))
```

pymysql 中也提供了只查询一行数据结果的方法：fetchone()

```
import pymysql
# 连接数据库
count = pymysql.connect(
            host = 'localhost',      # 数据库地址
            port = 3306,          # 数据库端口号
            user='root',          # 数据库账号
            password='821006052',      # 数据库密码
            db= 'anjing_test',  # 数据库表名
            charset = 'gbk'       # 中文乱码
)
# 完成 mysql 数据库实例化
db = count.cursor()
# sql 语句
sql = 'select * from weather'
# 执行 sql
a = db.execute(sql)
# 显示一行查询结果
for i in range(a):
    result = db.fetchone()
    print(result)
```

通过执行可以看出来 fetchone()只能每次查询 1 行数据，然后这里通过 for 循环的方法让其也展示出来了所有的数据内容。

```
import pymysql
# 连接数据库
count = pymysql.connect(
            host='localhost',      # 数据库地址
            port=3306,          # 数据库端口号
            user='root',          # 数据库账号
            password='821006052',      # 数据库密码
            db='anjing_test',   # 数据库表名
            charset='gbk'       # 中文乱码
)
# 完成mysql数据库实例化
db = count.cursor()
# sql语句
sql = 'select * from weather'
# 执行sql
a = db.execute(sql)
# 显示一行查询结果
for i in range(a):
    result = db.fetchone()
    print(result)
```

```
w12 (1)
D:\Python\python.exe E:/web/w12.py
('上海', '331eab8f3481f37868378fcdc76cb7cd', 1)
('北京', '331eab8f3481f37868378fcdc76cb7cd', 2)

Process finished with exit code 0
```

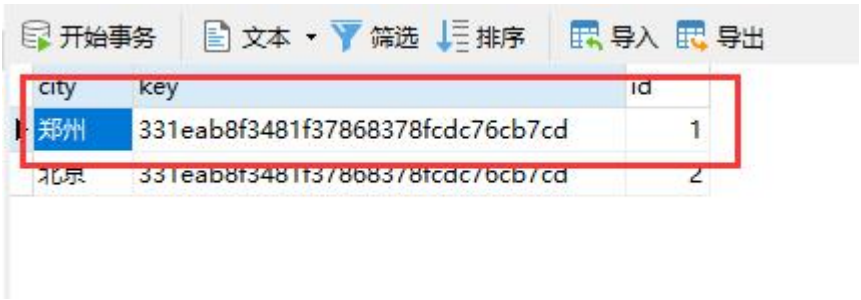
修改数据

当我们在数据库上进行修改数据后，都会需要点击下保存按钮，修改数据才会进行生效，python 操作数据库这里也是一样的。这里通过 commit()的方法来实现的。

```
import pymysql
# 连接数据库
count = pymysql.connect(
            host = 'localhost',      # 数据库地址
            port = 3306,          # 数据库端口号
            user='root',          # 数据库账号
```

```
                password='821006052',      # 数据库密码
                db= 'anjing_test',  # 数据库表名
                charset = 'gbk'       # 中文乱码
)
# 完成 mysql 数据库实例化
db = count.cursor()
# sql 语句
sql = 'update weather set city= "郑州" where id =1 '
# 执行 sql
a = db.execute(sql)
count.commit()
```

通过执行上述代码，查看进行查看数据库数据发现已经将上海更改为了郑州。说明我们的修改操作已经成功了。



删除数据

删除数据操作完成后，也需要通过 commit 来进行保存

```
import pymysql
# 连接数据库
count = pymysql.connect(
                host = 'localhost',       # 数据库地址
                port = 3306,            # 数据库端口号
                user='root',            # 数据库账号
                password='821006052',      # 数据库密码
                db= 'anjing_test',  # 数据库表名
```

```
                charset = 'gbk'        # 中文乱码
)
# 完成 mysql 数据库实例化
db = count.cursor()
# sql 删除语句
sql = 'Delete from weather    where city="郑州" '
# 执行 sql
a = db.execute(sql)
count.commit()
```

通过执行后，查看数据库结果，发现已经将郑州这条数据删除了。

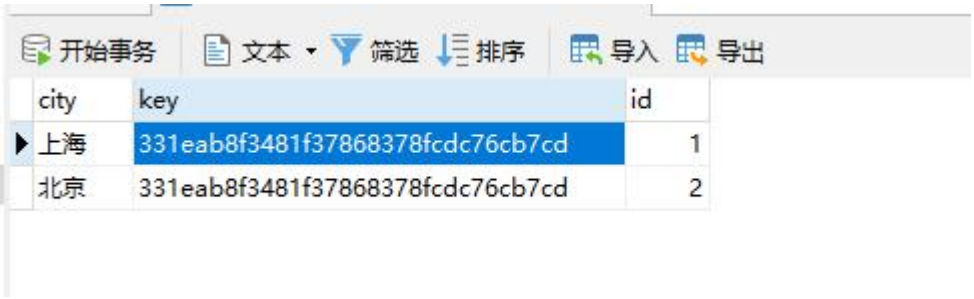
新增数据

增删改查就剩下一个新增数据，新增数据和上述内容基本上都是一直的，我们只需要写 sql 语句，然后在通过 commit 进行保存即刻

```
import pymysql
# 连接数据库
count = pymysql.connect(
                host = 'localhost',        # 数据库地址
                port = 3306,               # 数据库端口号
                user='root',               # 数据库账号
                password='821006052',      # 数据库密码
                db= 'anjing_test',   # 数据库表名
                charset = 'gbk'        # 中文乱码
)
# 完成 mysql 数据库实例化
```

```
db = count.cursor()
# sql 新增语句
sql = 'INSERT INTO weather VALUES ("上海","331eab8f3481f37868378fcdc76cb7cd",1)'
# 执行 sql
a = db.execute(sql)
count.commit()
```

通过执行完成后，可以看到我们的数据库中已经新增一条数据了。



city	key	id
上海	331eab8f3481f37868378fcdc76cb7cd	1
北京	331eab8f3481f37868378fcdc76cb7cd	2

接口实战

上述内容简单的介绍了如何通过 python 连接数据库，并如何进行对数据增删改查。接下来安静通过读取数据库的数据来完成接口实战（本文的接口内容来自聚合数据，需要的可以自行申请。）

接口地址： http://apis.juhe.cn/simpleWeather/query

返回格式： json

请求方式： http get/post

请求示例： http://apis.juhe.cn/simpleWeather/query?city=%E8%8B%8F%E5%B7%9E&key=

接口备注： 通过城市名称或城市ID查询天气预报情况

API测试工具

请求参数说明：

名称	必填	类型	说明
city	是	string	要查询的城市名称/id，城市名称如：温州、上海、北京，需要utf8 urlencode
key	是	string	在个人中心->我的数据,接口名称上方查看

这里通过 requests 库来模拟请求接口，通过读取数据库数据来进行传入接口参数中。

```
import pymysql
import requests
# 连接数据库
count = pymysql.connect(
                host = 'localhost',       # 数据库地址
                port = 3306,              # 数据库端口号
                user='root',              # 数据库账号
                password='821006052',     # 数据库密码
                db= 'anjing_test',   # 数据库表名
                charset = 'gbk'       # 中文乱码
)
# 完成 mysql 数据库实例化
db = count.cursor()
# sql 语句
sql = 'select * from weather'
# 执行 sql
a = db.execute(sql)
# 获取其中一条数据
result = db.fetchone()
# 接口请求参数内容
data = {
      'city':result[0],
      'key':result[1]
}
# url 地址
url = 'http://apis.juhe.cn/simpleWeather/query'
# 模拟 post 请求
r = requests.post(url,data=data)
print(r.text)
```

通过执行后发现，接口的请求结果已经返回出来了。

```python
import pymysql
import requests
# 连接数据库
count = pymysql.connect(
            host='localhost',      # 数据库地址
            port=3306,         # 数据库端口号
            user='root',         # 数据库账号
            password='821006052',      # 数据库密码
            db='anjing_test',  # 数据库表名
            charset='gbk'      # 中文乱码
)
# 完成mysql数据库实例化
db = count.cursor()
# sql语句
sql = 'select * from weather'
# 执行sql
a = db.execute(sql)
# 获取其中一条数据
result = db.fetchone()
# 接口请求参数内容
data = {
    'city':result[0],
    'key':result[1]
}
# url地址
url = 'http://apis.juhe.cn/simpleWeather/query'
# 模拟post请求
r = requests.post(url,data=data)
print(r.text)
```

```
w12 (1)
D:\Python\python.exe E:/web/w12.py
{"reason":"查询成功!","result":{"city":"上海","realtime":{"temperature":"22","humidity":"74","info":"阴","wid":"02","direct":
Process finished with exit code 0
```

总结

上述文章中简单的介绍了如何通过 python 连接数据，以及对数据的增删改查，也通过了一个小小的接口案例来实现了读取数据内容，然后传入接口参数中。这里肯定很多人会想，那么我们的测试数据可以通过数据库的形式进行保存了。当我们接口自动化时候，我们可以通过 sql 语句创建一些测试数据，通过读取数据的方法将数据传入接口中，当接口测试完成后，在进行删除本次测试数据内容。当然安静这是简单的说思路，具体的实践还要根据公司项目进行设计。好了，我是安静，感谢大家的阅读，希望对您有所帮助。

Selenium 处理菜单选项那些事儿你确定都知道吗？

◆作者：罗狮小钉

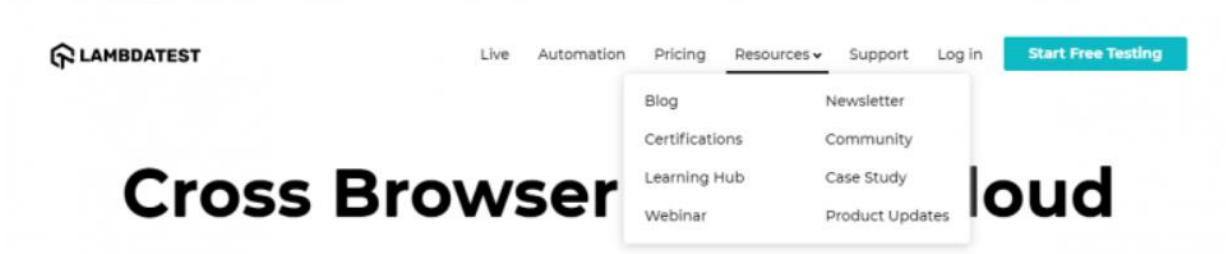
题记：
下拉菜单（Dropdowns）是当今网页中不可或缺的元素，和其他 HTML 元素一样，当我们进行 web 自动化测试时，这些下拉菜单也属于被测对象，是 web 自动化业执行中不可或缺的业务流程。今天就为大家分享 web 站点中常见的四类下拉菜单（Dropdowns）样式，以及 Selenium WebDriver API 中相应的解决方案，从此下拉菜单不再是你 web 测试中的疑难杂症。

1.自动化测试工程师必须知道的几类下拉菜单（Dropdowns）样式

在 HTML 中，我们会遇到 4 类下拉菜单的实现方式:

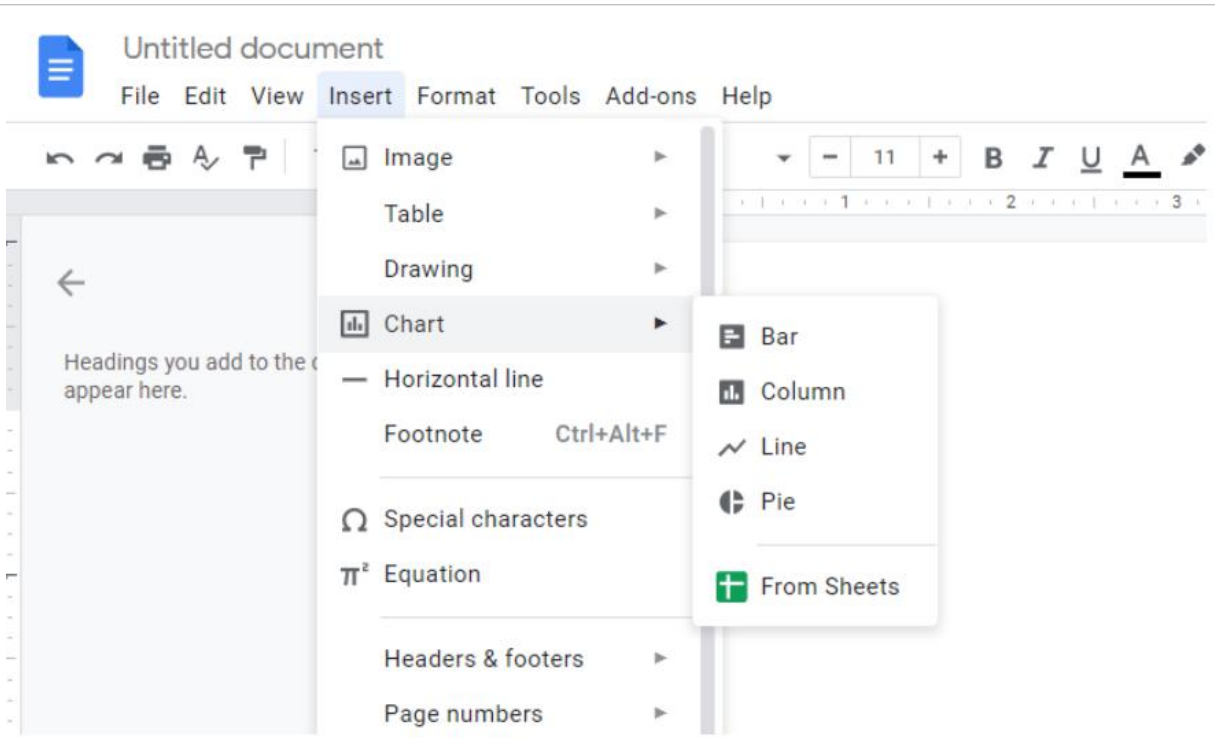
1)下拉菜单导航栏选项(Dropdown Navigation Options)

这类下拉菜单一般以页面导航栏的形式呈现，用于链接到其他页面；



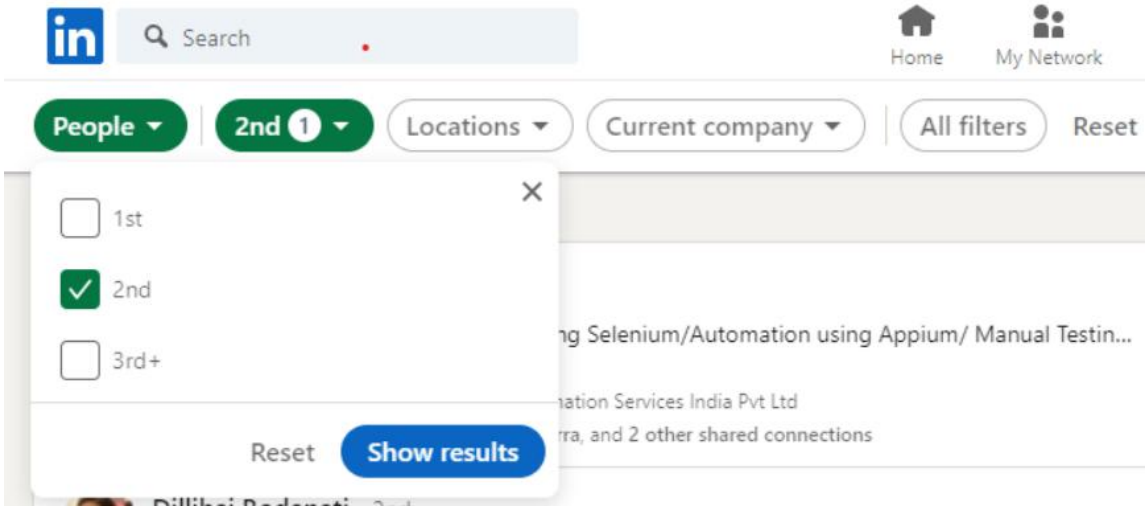
2)下拉菜单命令选项(Dropdown Command Options)

和下拉导航相似，这类下拉选项也置于页面上方，不同的是这些选项用于执行相应的命令操作，而非链接到某个页面，例如 Google 在线文档编辑菜单栏；



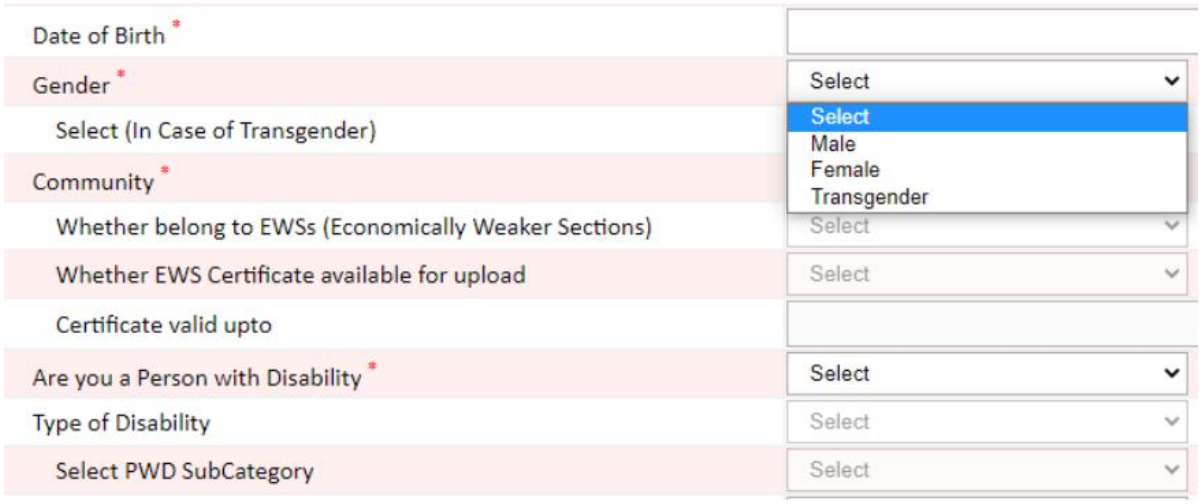
3)属性选择下拉菜单选项(Attribute Selection Dropdown Options)

这类下拉选项通常用于实现搜索过滤功能和自定义选项，例如更改网站的颜色模板或默认语言等；



4)Form 表单中的下拉菜单选项(Form Filling Dropdowns Options)

这类下拉选项主要出现在表单注册表单或产品/服务预定表单中；



2.Selenium Webdriver 中如何处理不同类型的下拉菜单

通常情况下我们可以通过 Selenium Webdriver 中提供的 CSS 或 XPATH 选择器来处理几乎所有类型的下拉元素，但是 Selenium Webdriver API 还给我们提供了一个附加功能，即 SELECT，通过 SELECT 类中封装的 API 来实现下拉选项（<select>标签的下拉选项）自动化交互。

1)Selenium WebDriver 中的 Select 类到底是什么?

Selenium 提供了 Select 类来实现 HTML Select 元素的操作。如何在 Selenium WebDriver 中处理各种下拉菜单，下面我们将看到不同的 SELECT 类函数：

class selenium.webdriver.support.select.Select(webelement)

Selenium Select 类首先会判断 Web 页面上我们与之交互的 HTML Web 元素是否标记为<select>，如果不是<select>元素，则 Selenium WebDriver 会抛出相应的异常(UnexpectedTagNameException)，需要注意的是 Selenium WebDriver 提供的 Select 类，除了与 HTML <select>标记一起使用，不能与其他任何 HTML 标记一起使用。Selenium 类中提供的函数（功能）如下：

- options(self)

—— 查找所有<select>标签中的<options>标签；

- all_selected_options(self)

—— 遍历<select>标签下所有<option>选项，判断其是否被选中，通过 is_selected()方法以列表形式返所有被选中的选项；

• first_selected_option(self)

—— 类似上述功能，遍历<select>标签下所有<option>选项，只返回第一个被选中的选项；

• select_by_value(self, value)

—— 通过 CSS 选择器（CSS selector）定位并返回所有与 value 属性相匹配的<option>选项；

• select_by_index(self, index)

—— 使用<option>选项标签的 index 属性，通过 get_attribute(“index”)返回匹配项；

• select_by_visible_text(self, text)

—— 该方法内部是通过 XPath 和多重 If-else 结构共同实现的，返回<option>选项中文本和给定字符串相匹配的那些选项；

例如： <option>XXXXXXXXXXXX</option>

此外， Select 类还提供了下面这些用来取消选项的功能

• deselect_by_index(self, index)

• deselect_by_value(self, value)

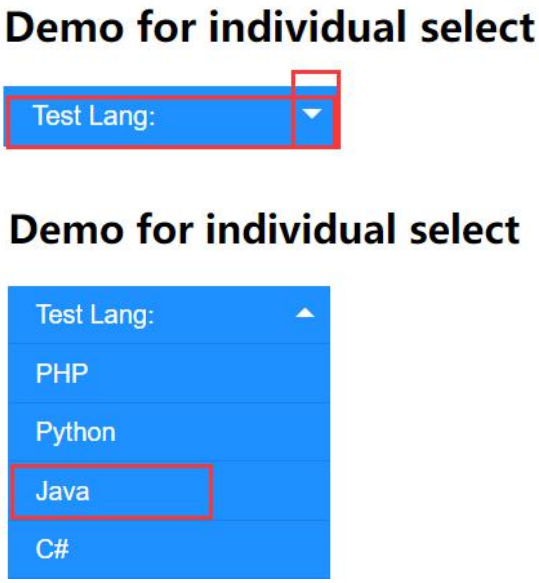
• deselect_all(self)

• deselect_by_visible_text(self, text)

2)Selenium WebDriver 通过文字匹配定位实现下拉选项

【场景 1】

在“Demo for individual select”下有四个选项“Python, Java, C#, PHP”，需要选择“Java”选项；

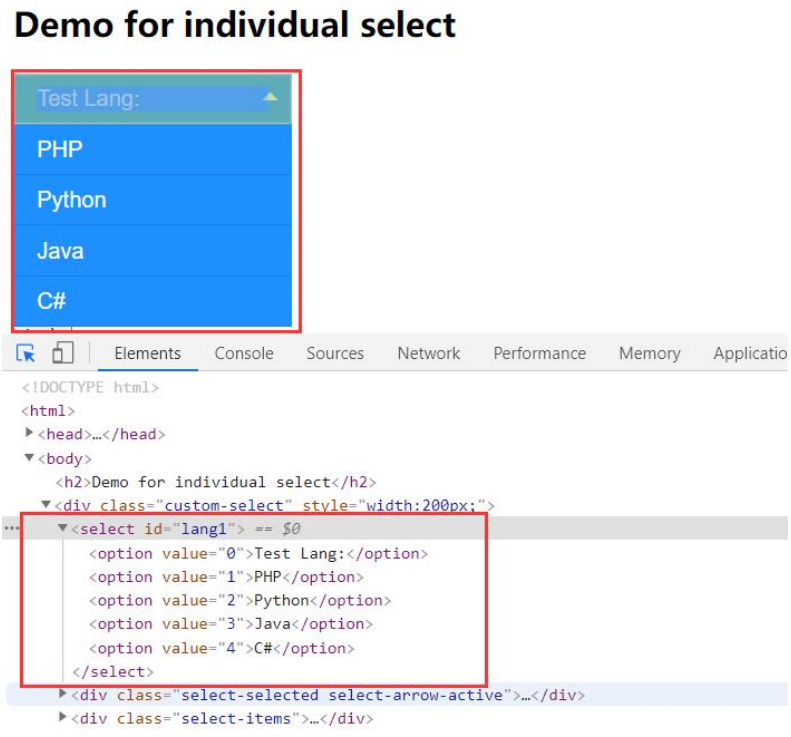


实现方式：

Select 类中的方法 select_by_visible_text(self, text)，这里的 text 是客户端就能看到的文本；

代码片段：

```
dropdown = Select(driver.find_element_by_id('lang1'))
dropdown.select_by_visible_text('Java')
```



【场景 2】

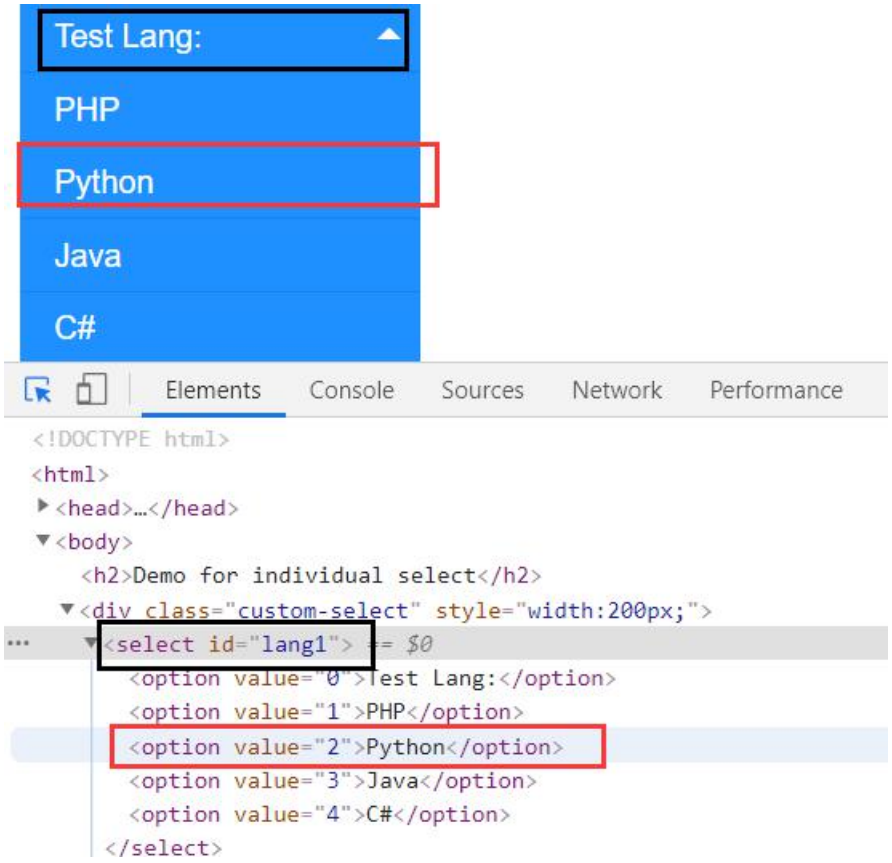
在“Demo for individual select”下有四个选项“Python, Java, C#, PHP”，需要选择“Python”选项；

实现方式：

Select 类中的方法 select_by_value(self, value)，和 select_by_visible_text(self, text)类似，只不过 text 是客户端能看到的文本，实则在选项提交后是作为 value 属性发送至服务端的，所以 Selenium WebDriver API 专门为 Select 类提供了 select_by_value(self, value)方法；

代码片段：

```
dropdown = Select(driver.find_element_by_id('lang1'))
dropdown.select_by_value('python')
```



3)Selenium WebDriver 通过索引定位实现下拉选项

通过 Javascrip 提供的 DOM 属性，我们可以利用 index 选择菜单中的某个<option>选项， 例如 document.getElementById("myCourses").selectedIndex = "3";

同样， Selenium WebDriver 也提供了这样一个方法 select_by_index(self, index)，用来通过 index 自动定位到选项菜单中的某一项。

【场景 3】

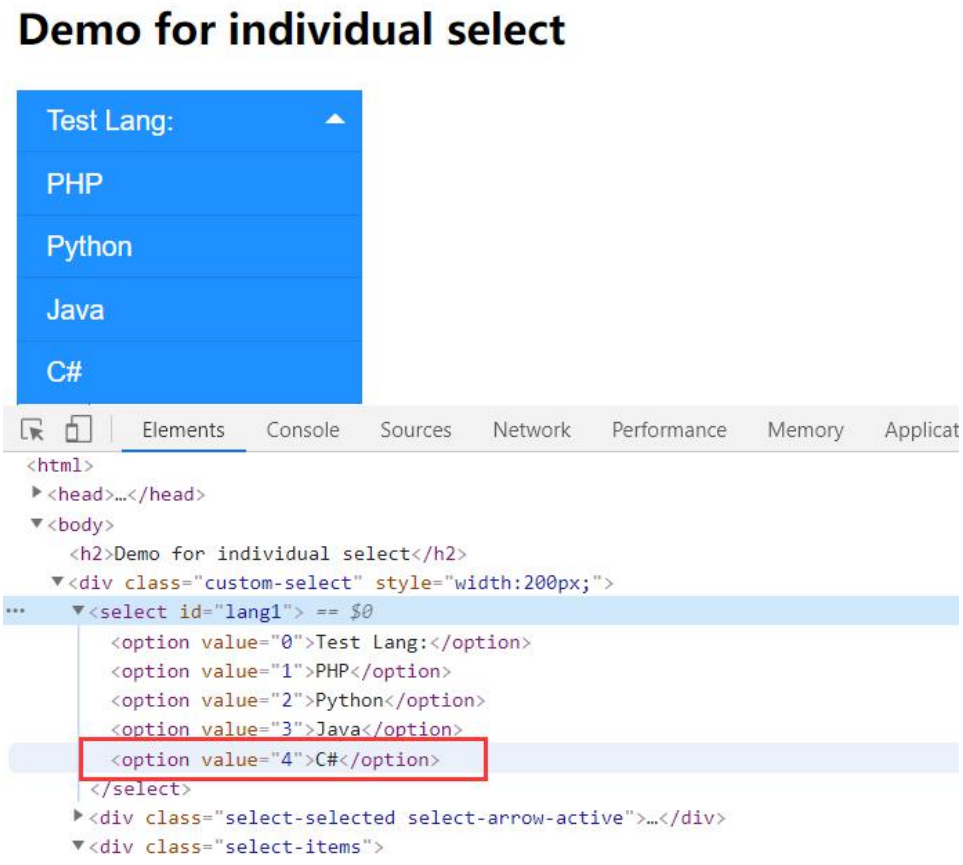
在“Demo for individual select”下有四个选项“Python, Java, C#, PHP”，通过 index 选择“C#”选项；

实现方式：

Select 类中的方法 select_by_index(self, index)，索引数按照选项个数，从 0 开始依次递增；

代码片段：

```
dropdown = Select(driver.find_element_by_id('lang1'))

dropdown.select_by_index(3)
```



4)Selenium WebDriver 实现菜单列表中的多选效果

页面中菜单栏中多选效果是通过<select>元素的 multiple 属性来实现的，理论上我们可以通过 XPATH 来定位该下拉菜单查看其是否有 multiple 属性；Selenium 内部的__init__()初始化方法中就已经帮我们实现了这一功能，这段源码如下，感兴趣的读者可以自行翻阅 Selenium WebDriver 中 Select 类的这段源码详细实现过程；

```
self._el = webelement
multi = self._el.get_attribute("multiple")
self.is_multiple = multi and multi != "false"
```

在实际应用中，我们首先通过 XPATH 或 CSS selector 定位<select>元素，然后判断其是否包含“multiple”属性，例如：

```
dropdown = driver.find_element_by_tag_name('select')
if dropdown.get_attribute("multiple"):
    print("multiple select options can be chosen")
else:
    print("only one select option can be selected")
```

由此可知，当需要在下拉菜单中同时选择多个选项时，有两种方案：可以先便利选项，再通过 Selenium WebDriver 的 Select 类方法进行选择；或者通过 Selenium WebDriver 的 Actionchains 类来选择多个选项，即首选获取<select>元素，然后再按下“ Ctrl”键的情况下对其执行单击操作；下面我们就这两种 Selenium WebDriver API 提供的方案进行演示。

【场景 4】

在“Demo For Multiple Selections”下有四个选项“Python, Java, C#, PHP”，分别选择带有“ PHP”和“ C#”选项；

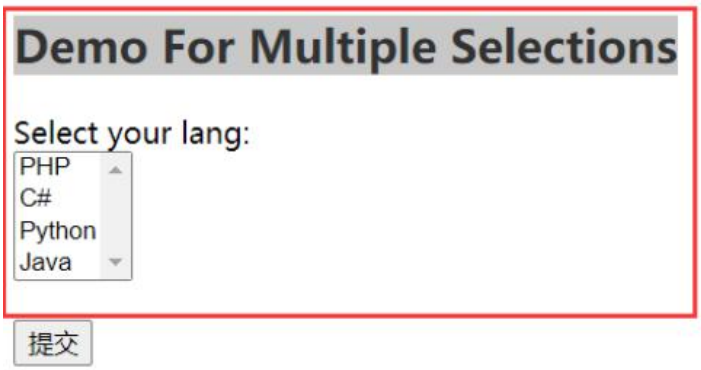
实现方式：

使用 Selenium WebDriver“Actionchains”，首先评估 ID 为“ lang2”的<select>元素

是否具有“multiple”属性。如果是，那么分别选择带有“ PHP”和“ C#”的选项；然后执行链式操作；

代码片段：

```
# 分别选择带有“ PHP”和“ C#”的选项
myOption = driver.find_element_by_xpath("//select[@multiple]/option[contains(text(), 'C#')]")
myOption1 = driver.find_element_by_xpath("//select[@multiple]/option[contains(text(), 'PHP')]")
# 执行链式操作
ActionChains(driver).key_down(Keys.CONTROL).click(myOption).key_up(Keys.CONTROL).perform()
ActionChains(driver).key_down(Keys.CONTROL).click(myOption1).key_up(Keys.CONTROL).perform()
```



【场景 5】

在“Demo For Multiple Selections”下有四个选项“Python, Java, C#, PHP”，分别选择带有“ Java”，“ PHP”和“ Python”选项；

实现方式二：

通过 Selenium WebDriver“Select”类实现；首先找到 ID 为“ lang2”且具有“ multiple”属性的<select>元素。如果是，那么我们分别使用 select_by_index, select_by_value, select_by_visible_text 来选择“ Java”，“ PHP”和“ Python”选项；

代码片段：

```
dropdown = Select(driver.find_element_by_id('lang2'))

        dropdown.select_by_index(3)

        dropdown.select_by_value('php')

        dropdown.select_by_visible_text('Python')
```

5)Selenium WebDriver 实现菜单列表中的全选效果

可以通过遍历列表中所有可用选项，结合 Selenium WebDrivers SELECT API 实现列表选项全选效果。

【场景 6】

在“Demo For Multiple Selections”下有四个选项“Python, Java, C#, PHP”，选中所有选项：

代码片段：

```
dropdown = Select(driver.find_element_by_id('lang2'))
```

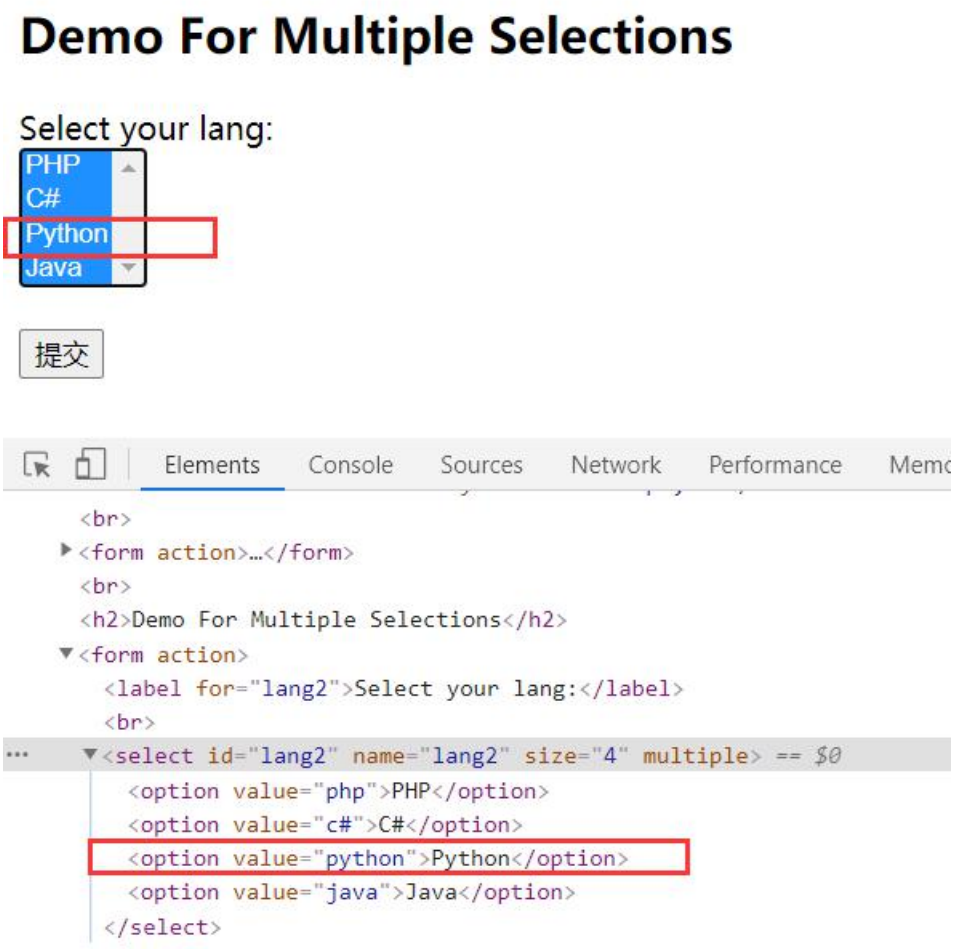
```
for opt in dropdown.options:
    dropdown.select_by_visible_text(opt.get_attribute("innerText"))
```

6)取消选择（取消下拉菜单中的已选项）

可以使用以下任何一种方式来取消菜单中的已选项：

- deselect_by_index(self, index)
- deselect_by_value(self, value)
- deselect_by_visible_text(self, text)

此外，可以使用 deselect_all(self)取消所有选项；



【场景 7】

基于场景 6，取消“python”选项：

代码片段：

```
dropdown = Select(driver.find_element_by_id('lang2'))

dropdown.deselect_by_visible_text('Python')
```

3.完整版 DEMO

结合以上场景 1-7， 整合后的场景如下：

在“Demo For Multiple Selections”下有四个选项“Python, Java, C#, PHP”，分别通过 XPATH, select_by_index, select_by_value, select_by_visible_text 等方法，选择菜单中的选项，并打印输出当前已选项；

完整版代码：

```
from selenium import webdriver
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time

driver = webdriver.Chrome()
driver.get('https://pynishant.github.io/dropdown-selenium-python-select.html')

# 定位ID为'lang2'的下拉菜单，检查其是否有“multiple”属性
dropdown = driver.find_element(By.ID,'lang2')
if dropdown.get_attribute("multiple"):

    print("1. 通过XPATH定位菜单中的选项")
    # 通过XPATH和文本属性，分别定位其中的两个选项"C#" and "PHP"
    myOption = driver.find_element(By.XPATH,"//select[@multiple]/option[contains(text(), 'C#')]")
    myOption1 = driver.find_element(By.XPATH,"//select[@multiple]/option[contains(text(), 'PHP')]")


    # 使用 actionchains 依次选择选项"C#" and "PHP"
    ActionChains(driver).key_down(Keys.CONTROL).click(myOption).key_up(Keys.CONTROL).perform()
    time.sleep(2)

    ActionChains(driver).key_down(Keys.CONTROL).click(myOption1).key_up(Keys.CONTROL).perform()
```

```
    try:
        # 创建指定的Select对象
        dropdown = Select(driver.find_element(By.ID,'lang2'))

        # 通过遍历列表中所有可用选项，打印该Select菜单中所有被选中项
        print("当前菜单中被选中的项 : ")
        for opt in dropdown.all_selected_options:
            print(opt.get_attribute('innerText'))

        # 取消已选项“PHP”
        dropdown.deselect_by_visible_text('PHP')

        # 通过遍历列表中所有可用选项，打印该Select菜单中所有被选中项
        print("取消“PHP”选项后，当前菜单中被选中的项 : ")
        for opt in dropdown.all_selected_options:
            print(opt.get_attribute('innerText'))
        time.sleep(2)

        # 取消所有选项
        dropdown.deselect_all()
        time.sleep(2)

        print("-----------------------------------------------------------------")
        print("2. 分别通过id,value,text定位菜单中的选项")
        # 通过index选择菜单中的选项“Java”
        dropdown.select_by_index(3)
        time.sleep(2)
```

```
        # 通过value选择菜单中的选项“php”
        dropdown.select_by_value('php')
        time.sleep(2)
        # 通过text选择菜单中的选项“Python”
        dropdown.select_by_visible_text('Python')
        # 通过遍历列表中所有可用选项，打印该Select菜单中所有被选中项
        print("当前菜单中被选中的项 :")
        for opt in dropdown.all_selected_options:
            print(opt.get_attribute('innerText'))
        # 取消所有选项
        dropdown.deselect_all()

        print("-----------------------------------------------------------------")
        print("3. 通过遍历Select列表，选择菜单中所有可选项")
        # 通过遍历，选择菜单中所有可选项
        dropdown = Select(driver.find_element(By.ID,'lang2'))
        for opt in dropdown.options:
            dropdown.select_by_visible_text(opt.get_attribute("innerText"))
        # 通过遍历列表中所有可用选项，打印该Select菜单中所有被选中项
        print("当前菜单中被选中的项 :")
        for opt in dropdown.all_selected_options:
            print(opt.get_attribute('innerText'))
        driver.quit()
    except Exception as e:
        print(e)
        print("error")
else:
    print("get_attribute 无效！")
```

运行结果：

```
1. 通过XPATH定位菜单中的选项
当前菜单中被选中的项 :
PHP
C#
取消“PHP”选项后，当前菜单中被选中的项 :
C#
------------------------------------------------------------
2. 分别通过id,value,text定位菜单中的选项
当前菜单中被选中的项 :
PHP
Python
Java
------------------------------------------------------------
3. 通过遍历Select列表，选择菜单中所有可选项
当前菜单中被选中的项 :
PHP
C#
Python
Java

Process finished with exit code 0
```

4.总结

以上基于 Python Selenium WebDriver 中 Select 类提供的 API 进行了详细介绍，深入探索了处理菜单选项的各类方法，以及如何从下拉列表中选择/取消选择选项，其中涉及到的处理方法都分别结合案例进行了有效演示，同时提供了完整的代码示例，感兴趣的读者不妨一试。

《51 测试天地》（六十二）下篇 精彩预览

- Selenium 通过无页面浏览器执行用例
- TestNg 跨浏览器运行测试
- 基于 MySQL Workbench 的性能调优探索与实践
- 启发式策略模型
- 随手写了个【RequestTools_V1】让领导瞧瞧我的能耐性
- 一次破解滑动验证码记录
- 一起来用 GIT 吧
- 自由玩转 Allure 测试报告，你也可以哦

马上阅读