# 自动化测试框架的类型



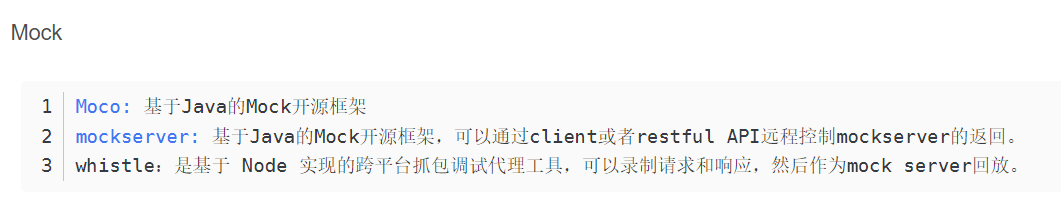












1. **自动化测试框架类型**

**市场上的自动化测试框架可能因支持不同的关键因素（如可重用性、易维护性等）而有所不同。如以下几种类型：**

（1）模块化测试框架

**这种框架将测试的应用程序分为许多逻辑和独立的模块。对每个模块，创建一个独立的测试脚本，互不影响。**

**缺点：每个模块实现测试脚本的同时，测试数据已经嵌入到测试脚本中，这导致使用不同的测试数据进行测试时需要在测试脚本中进行操作。**

1. 测试库架构框架

**库体系结构测试框架是建立在基于模块的测试框架之上，但比后者有一些额外的优势。它没有将测试的应用程序划分为测试脚本，而是划分为函数。因此，为测试中的应用程序创建一个由公共函数组成的公共库，当需要时，可以从测试脚本中调用这些库。**

**比如：一个登录的步骤，可以把它组合成一个函数，并保存到一个库中，登录时可以直接从库中调用这个函数，而不需要重新再编写代码。**

**缺点：像基于模块的框架一样，测试数据嵌入测试脚本中，改变数据需要修改脚本；随着越来越多的库的引入，可能会使框架越来越复杂。**

1. 数据驱动测试框架

**数据驱动测试框架将测试脚本逻辑和测试数据彼此分离。可以把测试数据单独存储起来，存储数据的可以是 xml文件、excel文件、文本文件、csv文件、ODBC数据库等。一般都是以 key-value 格式存储，方便获取使用。**

**缺点：过程比较复杂，可能需要定制测试数据源格式和读取机制**

1. 关键字驱动测试框架

**关键字驱动测试框架是对数据驱动测试框架的扩展，从某种意义上说，它不仅将测试数据从脚本中分离出来，它还将数据测试脚本的特定代码集保存到外部数据文件中。这些代码集被称为关键字，每个关键字都一种操作。关键字和测试数据都是独立于该框架。**

**缺点：需要懂得关键字的创建机制，从而可以自己开发关键字；随着越来越多的关键字引入，可能会使框架逐渐变得复杂；测试用例变得更长且复杂，从而影响测试用例的可维护性**

1. 混合测试框架

**混合测试框架就是上述（模块化，数据驱动和关键字驱动）多种类型框架的组合。**

**缺点：会比其他类型的测试框架更为复杂一些，例如阅读、维护等**

（6）行为驱动开发框架

**行为驱动测试框架可以让开发人员、测试人员等以易于阅读和理解的格式实现功能验证的自动化。可通过自然语言来描述测试功能，测试场景，测试步骤，测试结果等。**

1. **自动化测试框架组成部分**

****采用selenium webdriver+testng自动化测试框架:****

**配置文件**

**persities/ini文件**

**公共方法**

**测试用例**

1. **单元测试框架：管理、创建测试单元（testng/unitest）【testng.xml 、注解、pom.xml文件（引入依赖包）】**
2. **测试驱动：@dataprovider注解/ properties文件**
3. **PO测试设计模式：将一个页面设置为一个类，测试代码直接调用被测页面，元素属性变更后，直接修改被测页面即可**

**测试报告**

**测试试报告输出（一般是html格式）**

**日志文件**

**Log4j模块，打印指定的日志级别**

**持续集成**

**Jenkins**

# 常见的自动化测试框架

一、web自动化测试框架

1. **Robot Framework**

**如果要使用python自动化测试框架进行测试自动化，则Robot Framework是最佳选择。**

* **Robot Framework基于Python**
* **Robot Framework使用关键字驱动的方法使测试用例易于创建**
* **是用于[验收测试](https://so.csdn.net/so/search?q=%E9%AA%8C%E6%94%B6%E6%B5%8B%E8%AF%95&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/MXB1220/article/details/_blank)和验收测试驱动开发的自动化测试框架**

1. **Selenium**

**Selenium是一个用于Web应用程序测试的工具。**

**selenium三大部分：**

* **Selenium IDE：Firefox的一个扩展，它可以进行录制回放，并把录制的操作以多种语言（例如java、python等）的形式导出成测试用例。**
* **Selenium WebDriver：提供Web自动化所需的API，主要用作浏览器控制、页面元素选择和调试。不同的浏览器需要不同的WebDriver。**

**webdriver工作原理：**按照c/s架构原理设计

简单来说就是我们写的代码，以http请求的方式发送给服务器端（浏览器），服务器端接收请求，执行相应操作，并返回给客户端。

使用的协议：

JSON Wire protocol（作用是可以让同一个浏览器驱动处理不同编程语言的脚本）

* **Selenium Grid：分布式运行，在不同机器的不同浏览器上运行。**

**主要功能包括：测试与浏览器的兼容：支持多浏览器 (ie，firefox，chrome)**

二、移动端自动化测试框架

1. **Appium**

**Appium是一个开源的、跨平台的自动化测试工具，支持IOS、Android和FirefoxOS平台**

# 单元测试框架

**测试用例使用单元测试框架：testng、Junit、unitest、pytest**

* Junit适合隔离性强的单元测试，testng单元、集成测试都可以，也能做数据驱动
* unitest是python自带的，pytest需要安装比unitest更加简洁、效率更高

# 自动化三层架构

1. 页面元素层
2. 业务逻辑层

（3）测试用例层

测试用例层调用业务逻辑层；业务逻辑层调用页面元素层。

为什么要分层，意义在哪里？

实现页面元素，公共方法，公共数据，业务逻辑，测试用例集中式管理； 增加脚本的重复利用率；增加脚本的可维护性。

# 自动化测试框架

利用selenium工具

**1.web UI 自动化框架**

java+selenium+testng+ddt （数据驱动用perperties文件）

pyhon+selenium+unitest+ddt （数据驱动用yaml文件）

**2.接口自动化框架**

java+httpclient+testng+ddt（导入依赖包httpclient）

pthyon+request+unitest+ddt（导入request库）

利用robotframwork框架

**1.robotframwork+RIDE**

pip安装robotframwork

pip 安装robotframwork-RIDE

如果python版本是3.9，安装RIDE会失败，python版本太高——考虑RIDE与python版本兼容问题,可放弃RIDE方法

1. **robotframwork+pycharm——无版本兼容的问题**

KDT：pthyon+robotframwork+unitest+ddt

关键字分为：系统关键字（第三方库）；用户关键字（自定义）

# 驱动模式

1. 关键字驱动模式：将一些逻辑封装成关键字（一个函数名）
2. 数据驱动模式：测试数据参数化，业务逻辑与测试数据分离

# 测试设计模式

1. PO模式

Page Object模式是[Selenium](https://so.csdn.net/so/search?q=Selenium&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/jdhfjb/article/details/_blank)中的一种测试设计模式，主要是将每一个页面设计为一个Class（封装在一个class类中），其中包含页面中需要测试的所有元素的属性和操作，这样在Selenium测试页面中可以通过调用页面类来获取页面元素，这样巧妙的避免了当页面元素id或者位置变化时，需要改测试页面代码的情况。当页面元素id变化时，只需要更改测试页Class中页面的属性即可。

目的: 测试代码与被测页面对象代码分离

# 控制浏览器、定位元素的方法

## 元素定位方法

**id/xpath/classname/css/name**

## 浏览器窗口操作

**浏览器窗口最大化、窗口前进【forword()方法】、窗口后退【 back()方法】、窗口关闭【close(关闭当前窗口)、quit(关闭所有窗口)】、浏览器刷新【refresh】、多窗口跳转【获取所有窗口句柄，switch\_to】**

## iframe框架

**存在frame框架，而处在frame框架中的元素，不能直接定位，需要先跳进当前的frame中，再进行定位。【driver.switch\_to.frame(“id属性”)】**

## 鼠标键盘操作

**1.生成鼠标对象：ActionChains**

**2.做对应的操作：右击【context\_click】,双击【double\_click】, 鼠标悬停【move\_to\_element】,点击【click】**

**3.执行相应的操作：perform**

**例如： 悬停在注册上边**

**element = driver.find\_element(By.CSS\_SELECTOR, "button")**

**调用鼠标悬停事件方法**

**action.move\_to\_element(element)**

**调用鼠标执行的方法**

**action.perform()**

**键盘输入文本**

**element.send\_keys('admin1')**

## 显示等待、隐式等待、强制性等待

**隐式等待是全局的，针对所有的元素查找方法；显式等待值针对单一元素或者一组元素，可自定义等待条件**

**显式等待：等待某个元素加载完成**

**隐式等待：等待整个网页加载完成**

**强制性等待：sleep(10) 必须等到10s**

## 窗口截图

**driver.get\_screenshot\_as\_file(file\_path)  将截图放在指定路径**

**可用于：保留失败用例的截图，作为失败证据**

## 验证码处理

1. **测试环境下，开发人员去掉验证码——最简单的方法**
2. **记录cookie：通过向浏览器中添加cookie，可以绕过登录的验证码**
3. **测试环境下，开发人员设置万能验证码**
4. **用第三方技术识别验证码，但识别效果往往不太好**

## JavaScript的应用

**JavaScript相对于selenium原生api的执行速度更快，如果有遇到selenium难以处理的操作时，可以采用执行js代码去完成。**

1. **滚动条：滚动至最底部、最顶端、指定位置、指定元素**

滚动条是由js代码编写的

直接模拟运行JavaScript，使用execute\_script()方法

1. **selenium原生api无法操作控件，用js移除readonly属性**
2. **基本元素操作**

## 日志处理

关注对象：（1）日志级别（2）输出地方

为了打印重要信息，方便问题定位：

日志处理比打印在控制台的优势是：设置不同的日志等级，可以只输出重要信息，而不必显示大量的调试信息。

1. java——Log4j模块

Log4j有三个主要的组件：

Loggers(记录器)：日志级别：DEBUG < INFO < WARN < ERROR < FATAL

Appenders (输出源)：控制台/文件

Layouts(布局)：日志输出样式

1. python——log模块



## 参数化(数据驱动)

1. java : property、excel等文件

testng：@DataProvider注解

public class DataProviderTest {

@Test(dataProvider = "stuData") //这个名称需要和@DataProvider(name='xx')是一样的才能找到

public void studentTest(String name,int age){

System.out.println("name="+name+","+"age="+age);

}

/\*\*

\* 通过一个方法获取到参数数，并且return。使用@DataProvider()进行注释

\* @return

\*/

@DataProvider(name="stuData")

public Object[][] dataProvider(){

Object[][] o =new Object[][]{ //注意我们这个类型必须是Object

{"zhangsan",20},

{"lisi",30},

{"wangwu",40}

};

return o;

}

}

输出：

name=zhangsan,age=20

name=lisi,age=30

name=wangwu,age=40

1. python: yml、excel等文件

ddt注解：测试类class前加@ddt 类内的测试方法前加@data( 参数类型)

1. 数据类型是数组



1. 数据类型是列表——多参数

数据解析时需要解包需要使用@unpack



## 测试报告

基于html的测试报告

1.testng自带默认测试报告

# 如何提高selenium脚本的执行速度

1. 减少使用sleep强制等待，使用隐式等待或者显示等待
2. 元素定位id定位速度>xpath>classname,
3. 实现多线程并行执行用例
4. 测试用例减少不必要的操作和重复的操作，可实现直接跳转到某页或者通过数据库插入数据；

# 元素找不到的原因？

1. 动态id定位不到元素
2. Xpath路径描述错误
3. 元素在frame里，要swtich to才可以
4. 跳转到另一个新开的网页窗口，要swtich to才可以

定位优先选择ID; name ; css\_selector；xpath；classname

# 提升用例的稳定性

1．页面加载时间不够，判断元素可点击element\_to\_be\_clickable()

2. 页面内容更新

3. 测试环境数据有变动

# 自动化测试用例从哪里来？

-手工用例中抽取

# 你觉得自动化测试最大的缺陷是什么？

-不稳定

-不易维护

-成本与收益

-可靠性

# 分布式压测怎么做

1、Jmeter分布式测试时，选择其中一台作为控制机，其它机器做为代理机。  
2、执行时，控制机会把脚本发送到每台代理机上，代理机拿到脚本后开始执行，它应该是通过命令行模式来执行的。  
3、执行后，代理机会把结果回传给控制机，控制机会收集所有代理机的信息并汇总

TestNg自动化测试框架：@test ;@beforeclass ;@afterclass;等测试注解；适合测试人员写；

JUnit是一个Java语言的单元测试框架：@test ;@beforeclass ;@afterclass;等测试注解 适合开发人员写；

# 数据驱动方法

**1.property文件**

测试行为循环使用文件中存储的测试数据【或者是execl文件】

**2.testng dataprovide**

public class DataProviderTest {

private static WebDriver driver;

@DataProvider(name="searchWords")

public static Object[][] word(){

return new Object[][]{{"蝙蝠侠","主演","迈克尔"},{"超人","导演","唐纳"},{"生化危机","编剧","安德森"}};

}

@Test(dataProvider="searchWords")

public void dataProviderTest(String searchWord1,String searchWord2,String SearchResult) {

driver.get("http://www.sogou.com");

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

//在搜索框中输入参数

driver.findElement(By.id("query")).sendKeys(searchWord1+""+searchWord2);

driver.findElement(By.id("stb")).click();

//单击搜索按钮后，等待3秒显示搜索结果

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

//判断搜索结果的页面是否包括测试数据中期望的关键词

Assert.assertTrue(driver.getPageSource().contains(SearchResult));

driver.quit();

}

@BeforeMethod

public void beforeMethod() {

System.setProperty("webdriver.firefox.bin", "D:/Mozilla Firefox/firefox.exe");

driver = new FirefoxDriver();

}

@AfterMethod

public void afterMethod() {

driver.quit();

}

}

说明；测试方法中，使用(dataProvider="searchWords")来引入数据。dataProviderTest方法的三个入参分别是使用了searchWords测试数据集中的每个一维数组中的数据进行赋值。这个测试方法会被调用三次，分别使用测试数据集合中的三组数据

# 常见的API接口：Http接口、webservice接口

1.http接口走http协议，通过路径来区分调用方法，请求报文一般是key-value形式的，返回报文一般是json串，常用的是get和post方法来请求。

2.webservice接口走的soap协议，通过http传输，请求报文和返回报文都是xml格式的。

# 设置多线程：利用testng.xml文件

1.testng文件设置

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite parallel=*"tests"* Thread-count=*"2"* name=*"Suite"*>

<listeners>

<!--失败重试监听器结果只记录最后一次结果-->

<listener class-name=*"Listener.RetryListener"*></listener>

<listener class-name=*"Listener.ResultListener"*></listener>

</listeners>

<test name=*"TestBaidu1"*>

<classes>

<class name=*"Baidu.BaiduTest"* />

</classes>

</test>

<test name=*"TestBaidu2"*>

<classes>

<class name=*"Baidu.BaiduTest02"* />

</classes>

</test>

</suite>

2.登录设置全局的driver

**public** **class** Login {

**public** **static** ThreadLocal<WebDriver> *ThreadDriver*=**new** ThreadLocal<WebDriver>();

**public** **static** WebDriver ChromeLogin (WebDriver driver ){

**try** {

driver=*ThreadDriver*.get();

System.setProperty("webdriver.chrome.driver", "C:\\My-Files-Dn\\Google\\Chrome\\Application\\chromedriver.exe");

driver = **new** ChromeDriver();

driver.manage().window().maximize();

driver.get("https://hao.360.com/");

*ThreadDriver*.set(driver);

}**catch**(Exception e) {

e.printStackTrace();

}

**return** driver;

}