自动化测试框架

理解框架

目标：

1. 框架衡量指标
2. 流行的框架设计思路
3. 框架设计中的分层思想
4. 现有框架存在的问题

一、为什么要设计测试框架

软件产品开发和自动化测试工具

- 区别：用户不同；自动化测试工具针对测试人员；框架是一个半成品
- 产品或工具需要考虑开发效率和成本
- 需要高效的框架
- 框架让产品开发效率更高、降低开发成本，成本更低，所以需要框架
- 框架是介于编程语言原始代码和最终产品之间的一个半成品

如何评价测试框架的好坏（标准）

- 具有独立性，能够独立于其他测试工具
- 重用性
    - 测试步骤尽可能重用
    - 测试脚本、测试数据、测试环境
- 易用性
    - 测试数据好维护，易于定制
    - 支持不同来源数据的读取
    - 测试脚本易于开发
    - 适合于团队开发
- 可维护性
    - 对错误的相应处理机制
    - 方便查找
    - 应该有良好的异常处理机制
    - 测试脚本易于维护
    - 更稳定
- 稳定性
    - 脚本尽可能的稳定
    - 具备无人干预执行能力
- 可移植性
    - 代码具备一定的可移植性
    - 兼容不同的平台

当前框架设计思路

- **数据驱动测试**（Data Driven Testing） **DDT**

    - 由数据决定
- **关键字驱动测试**（Keyword Driven Testing）**KDT**
    - 具体的动作
    - 由步骤决定
- **行为驱动开发/测试**（Behavior Driven Development）**BDD**
- **业务流程测试**（Business Process Testing）**BPT**
- **页面对象模式**（Page Object Model）**POM**
- **基于组件的测试**（Component Base Testing）**CBT**

二、分层思想

1. 概念

   核心就是不同的操作，应该放在不同的类和不同的方法中，层与层之间互相依赖，互相调用，共同完成功能的实现

   广义：单元——函数，类

   集成——类组合起来的衔接

   系统——整合，衔接是否存在问题

   测试行为分层，不同的层，关注这一层执行的内容，即测试阶段。

   狭义（较多）：

    - 代码设计代码分层；
    - 规划代码的功能；
    - 要实现的功能进行拆解，层与层之间相互关联，不要有跨层的行为
2. 指导原则
    - 上层总是依赖于下层，不要有跨层访问
    - 一切从系统需要提供的功能区进行分析
    - 每一层接口需要有明确的职责范围
    - 只有接口定义规范不发生变化，每一层（接口）的实现就互相不受影响

三、当前测试框架存在的问题

1. 测试框架只针对某些特定的测试领域
2. 测试框架为了让测试行为变得简单
3. 为了让工作简单化，试图减少代码量，这样就导致各种的限制比较多，也有可能吸纳之比较死，但是这样就会让一下深度开发定制需求难以或无法实现，从而丧失框架的灵活性和扩展性
4. 测试框架期望自己通用性强，试图能够适应各种测试类型，但是这样可能在深度开发方面的能力会大打折扣

四、图像识别技术

目标：

1. 图像识别匹配算法
2. 利用python开发模板匹配算法
3. 基于图像识别的自动化开发
4. 基于OpenCV的图像识别技术
5. 在移动端的应用

图像识别的原因：

- 界面元素定位的难题
- SikuliX没有专门的python接口，只能利用JPype进行跨语言调用
- 能够快速定位，解决问题
- UI自动化定位，辅助手段

图像识别的概念：

- 核心本质就是对图像轮廓的描述，变形后的容错，像素信息的匹配处理等

图像识别的做法：

- 在一个大的画面中，查找一个匹配某个小区域的画面元素，然后定位到该元素上在进行相应的操作，从而实现测试的目的

图像识别的技术（应用）：

插引法、投影法

图像识别的算法：

- 模板匹配（算法）
    - 对于要操作的画面元素进行单独的截图，这个截图称之为模板匹配
    - 利用这个模板在整个当前屏幕或者当前窗口进行搜索匹配，找到完全符合条件的区域的中心位置坐标的过程
    - 缺点：模板匹配也具有一些局限，它只能进行平行移动 ，若原画面中的匹配模板发生旋转或大小变化时，模板匹配算法无效
- 滑动比对
    - 需要知道图片是如何显示的
    - RGBA（ ）
        - 图片>像素点>四个值的比对
- 匹配度比对（Similarity）
    - 也可以成为相似度，主要用于解决图像匹配过程中的容错问题
    - 考虑角度：
        - 像素占比
        - RGBA值占比
        - 此外，还可以采用预先转换为灰度图的方式

算法思路：

1. 依据测试场景对需要操作的元素截图。注意：截图时最好确保图片的中心位置与元素的中心位置一致
2. 依据需要对当前屏幕截图（被测应用的屏幕）
3. 获取模板图片和屏幕截图的每一个像素点的颜色值并保存到一个列表中，供后续滑动和遍历对比

   （利用模板图片在屏幕截图上进行滑动比对的过程，在比对中进行逐点【像素点】的比较）
4. 通过循环在X和Y轴两个方向上进行逐点判断，如果没有匹配就滑动到下一个点，直到找到匹配为止

   （如果不能匹配，向下一个点滑动，滑动防线显示X轴，当X轴全不对，就在Y轴方向滑动一下，继续对比，直到全部对比完成）
5. 依据匹配的点的位置获取中心点的位置并返回
6. 利用获取到的坐标，进行相应的鼠标和键盘操作即可

制作模板；截取比对画面；根据像素点进行逐一比对；知道滑动到最后位置

X+模板宽度/2；Y+模板高度/2

顶点：大图宽度-小图宽度-1

做模板注意匹配数量，是否是当前画面唯一匹配结果

存在的问题：

1. **匹配效率问题**
    - 解决方案：
    - 对5个特征点进行优先比对，若特征点匹配不成功，就直接滑动到下一个位置
    - 若特征点匹配成功，再继续对当前位置进行全像素必对
    - 全像素不对不成功，也是滑动到下一个位置，在进行比对
    - 全像素比对成功，我们就计算当前位置的中心坐标
2. **匹配度的问题**
    - 解决方案：
    - 制作模板的时候需要注意，尽可能的让模板的匹配度是一个唯一的结果

五、利用python开发

安装python库Pillow

```
#导包
from PIL import Image,ImageGrab#截图
import os
import time
```

代码思路：

1. 定义图像识别的类ImageMatch
2. 定义初始化方法
    - 定义截取的大屏画面（大图）的图像对象screen,定义模板图片（小图）的图片对象template,并对其进行初始化为None
    - 定义大图的图像数据对象（RGBA）screen_data,定义小图的图像数据对象template_data,并初始化为None
3. 定义像素点比对方法compare,这个方法有两个参数p1，p2，分别代表两个像素点#对于像素点p1和p2，对应位置的元素分别进行比较，
    - 如果他们全部相等就返回代表像素点相等的逻辑值，True；如果有任何一组不相等，那就返回代表两个像素点不相等的逻辑值False
4. 定义一个查找模板图片位置坐标的方法find_image，这个方法有一个参数target，就是模板图片的名字
    - 定义一个模板图片路径变量image_path并赋值
    - 利用ImageGrab模块的grab方法获取大图对象，并且对其调用convert方法，参数传入RGBA。从而获取相应格式的大图对象，最后赋值screen
    - 利用Image模块的open方法，获取小图对象，并且对其调用convert方法，参数传入RGBA，从而获取相应格式的小图对象，最后赋值template
    - 利用图像对象的属性size来分别获取大图和小图的宽高screen_width,screen_height,template_width,template_height
    - 利用图像对象的方法load，来分别获取大图和小图的rgba数据对象，并将其赋值给screen_data,template_data

- 利用双层for循环来进行模拟滑动的过程，这里外层循环用来表示Y轴的滑动，内存for循环用来表示X轴的滑动
- 在内层循环体中进行5个特征点的比对

```
#左顶点
self.compare(self.screen_data[0,0],self.template_data[0,0,])
#右顶点
self.compare(self.screen_data[x+template_width-
1,y],self.template_data[template_width-1,0])
#左下角
self.compare(self.screen_data[x,y+template_height-
1],self.template_data[0,template_height-1])
#右下角
self.compare(self.screen_data[x+template_width-1,y+template-
1],self.template_data[template_width-1,template_height-1])
#中心点
self.compare(self.screen_data[x+int(template_width/2),y+int(templat
e_heigh/2)],self.template_data[int(template_width/2),int(template_h
eight/2)])
```

- 当5个特征点比对全部为True时，我们去做全像素匹配，调用check_match方法
- 当5个特征点比对为False时，什么都不做
- 当全像素匹配成功时我们就去计算中心点坐标，并return这个坐标结果
- 当全像素匹配不成功时，什么都不做，这就意味着继续滑动到下一个位置
- 如果双层for循环成功走出来，此时意味着每一找到指定模板图片，所以我们是最后需要返回-1，-1表示没找到

模拟滑动

- 0，0进行比较
- 双层for循环（外层：Y，i=Y轴；内层:X）

```
for y in rang(大图的高度-小图的高度):
    for x in rang(大图的宽度-小图的宽度):
    #大图的（0，0）
    screen_data[0,0]
    #任意坐标
    screen_data[x，y]
    if 条件判断
        全像素匹配
     if 真:
        计算中心点
        return
```

5. 定义一个全像素匹配的方法check_match,这个方法有2个参数，分别表示X，Y，也就是当前左顶点坐标
   - 利用template对象的size属性来获取小图的宽高数据template_width,template_height
   - 利用双层for循环来模拟像素点滑动的过程，外层for循环代表小图的Y轴方向，内存的for循环代表小图的X轴方向
   - 在内层for循环中，检查大图和小图对应像素点的匹配情况，如果匹配就继续滑动，如果不匹配就直接返回False
   - 像素点匹配时，点的比较，self.compare(self.screen_data[x+small_x,y+small_y],self.template_data[small_x,small_y])

    - 如果双层for循环顺利走出来，此时意味全像素匹配成功，应返货True
6. 定义一个方法，用于断言的方法check_exists,主要用于检查模板图片是否存在，这个方法有一个参数target，就是模板图片的名字
    - 首先调用self.find_image方法来获取指定模板图片的坐标位置
    - 检查坐标位置是否为-1，-1，是则返货False，否则返回True