目录 | (六十四期·下)

混沌工程-为未知做好准备吧

◆ 作者：方正

在最简单的 C/S 模型当中，一次网络通讯会发生这样的过程：

1.客户端将消息发送到网络上。

2.网络向服务器发送消息。

3.服务器验证消息。

4.如果需要，服务器更新其状态。

5.服务器将回复发送到网络上。

6.网络向客户端发送回复。

7.客户端验证回复。

8.如果需要，客户端更新其状态。

在这个过程中可能发生的所有故障排列是令人难以置信的。例如，网络随时可能出现故障，数据存储可能出现故障或过载，或者验证逻辑可能失败导致应用程序崩溃。应对这一挑战的传统方法是测试。但是测试解决的问题常常是在有一个期望结果的前提下，通过断言来验证已知条件，例如结果是否是我们期待的，函数结果返回是否正确。

但随着分布式系统变得越来越复杂，测试已经达到极限。它通常无法解决生产环境的复杂性，例如发生在网络上的随机故障和其他奇怪的间歇性错误；配置限制和配置漂移；未知的呢？你怎么能测试你不知道的东西？

例如，在实例动态启动和停止的系统中，如果任何实例磁盘空间不足会发生什么情

况？如果一个实例没有剩余空间来写入日志，它在大多数情况下会快速失败，这是一个真正的问题，因为快速失败会产生一个黑洞。因为它失败得很快，通常会欺骗负载均衡器，认为该实例可以自由地接收更多请求，而这反过来也会导致更多的失败。

如果我们能更早地发现这一点，我们就可以事先进行改进并避免中断。例如：

1、使用日志轮换。

2、使用集中式日志服务。

3、监控磁盘空间。

4、在每周运营会议上审查指标。

5、当设备上只剩下 10% 的磁盘空间时发出一些警报。

幸运的是，有一种实践可以帮助解决这种未知数——正如你可能猜到的，混沌工程。

混沌工程是一个过程：

1、通过创建破坏性事件（例如服务器中断或 API 限制）来对测试或生产环境中的应用程序施加压力。

2、观察系统如何响应。

3、实施改进。

我们这样做是为了证明或反驳我们对系统处理这些破坏性事件的能力的假设。与其让这些破坏性事件发生在凌晨 1 点、周末和生产环境中，我们不如在受控环境中在工作时间创建它们。混沌工程不是无目的地随机注入故障而是通过精心计划的实验在受控环境中注入故障，以建立对应用程序和工具的信心，以承受动荡的条件。

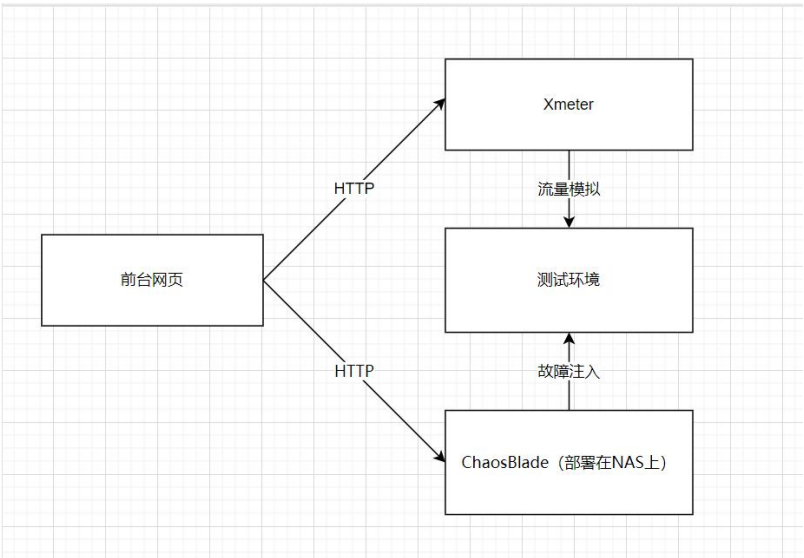
虽然混沌工程是一个很好的选择，但事实是它很难开始，因为在大多数情况下，你必须将不同的工具、脚本和库拼接在一起，以涵盖可以在系统中注入的所有故障。基础设施、网络、应用程序——每个目标都有不同的工作工具，无论是库、代理还是要安装的容器。客户通常不想在他们的应用程序中安装任何额外的东西——要安装的东西越多，就越复杂。或者，如果必须安装东西，它们需要集中管理和统一。

为此我们选择将 ChaosBlade 封装成一个统一的网页工具，选择 ChaosBlade 的原因

是它支持目前主流的故障注入场景，相对其他几个故障注入工具，ChaosBlade 故障注入覆盖的场景最全，支持四大类、几十种常用的故障注入能力。

该网页工具的运行方式是提供一个对于用户非常友好的界面，注入故障时只需要点击相应故障的按钮，该故障会以 HTTP 请求的方式发送到测试环境的 ChasosBlade 中，ChaosBlade 以命令行的方式来执行具体的故障注入请求，同时将返回执行的结果。

与此同时因为在实验环境下，没有真正的用户，无法预测故障会对实际用户产生怎样的影响，因此我们将利用 Xmeter（该工具提供与 Jmeter 类似的发压功能不同的是其有网页管理端，且有部署好的发压服务器）的 API 接口，将其也进一步封装为前端网页按钮，将发压，暂停，监控结果从其原有网页端，与故障注入工具统一放在同一个网页端，这样从故障注入，流量模拟，监控结果均可以在同一个网页端实现。



通过我们的工具，对于测试系统而言无需再部署其他复杂的应用，插件或是容器，只需要将包含 ChasoBlade 的 NAS 挂载上去，对于测试人员而言，无需了解混沌测试的复杂性，例如 Chaosblade 该如何使用 CLI 注入不同的故障，如何使用 Xmeter 工具进行发压，只需通过网页端将录制好的 Jmeter 脚本上传至网页端，点击开始发压即可完成对于测试系统的流量模拟，将一个复杂的概念封装成了简单易用的工具。

在我们进行混沌工程的实践中，也发现了一部分工具本身的问题，例如对于 Pass 上 20 核的服务器，故障注入工具无法真正的注入 CPU 利用率达到 80%的故障，只有部分核心达到了 80%的利用率；在注入 pod 异常关闭时，发现故障工具注入工具无法对 pod 进行关闭；因此，单一的故障注入工具也许无法完全解决我们的问题，集合多个故障注入工具，结合业务环境自主开发，最终封装为统一的工具接口应当是我们的方向。

混沌工程的意义在让我们对未知做好了准备：

1.提高了系统弹性

2.暴露监控、可观察性和警报盲点

3.提高恢复时间和操作技能

最后我想引用《道德经》的一句话，“合抱之木，生于毫末；九层之台，起于垒土；千里之行，始于足下”，所以，我的建议是尝试一下，敢于开始，踏上混沌工程之旅。

局部探索性测试

◆作者：刘晓佳

一、什么是局部探索性测试

局部探索性测试是 James A. Whittaker 提出的探索性测试方法中的一种（包括：局部探索性测试，全局弹缩性测试），通过该方法测试人员可以不必了解许多的测试信息（比如：项目环境信息，产品元素等等）就可以完成测试任务。局部探索性测试的重点是指导测试人员结合自身经验、专业知识，以及软件在具体操作下的响应情况进行测试设计和执行测试。

二、局部探索性测试的关注点

局部探索性测试主要关注 5 个方面：用户输入、状态、代码路径、用户数据和执行环境。为什么要关注这 5 个方面呢？因为所有软件几乎都可以简化为四个任务：输入数据、执行运算、存储数据、输出结果。

1.用户输入

用户输入简而言之就是输入是程序外部引发的，并且使得软件执行了某些代码。设想一下：假如存在一个简单的输入框，可以接受字母和 0-10000 范围内的数字，那么可以存在多少种输入？至少是 26*10000 种（其中还不包括区分大小写字母）。由此可见，对于软件来说，仅仅是一个小功能，都存在太多的输入或输入的组合，而测试人员想要穷尽所有的输入进行测试几乎是不可能的。

1)如何测试用户输入

James A. Whittaker 建议我们可以参考以下几个方面测试用户输入。

- 合法输入和异常输入

对于开发人员而言，合法输入即是能够使程序正常运行的值，而非法输入即是输入值进入程序后会触发错误代码处理。开发人员通常通过输入筛选器、输入检查和使用异常三种方式定义错误处理程序。从测试人员角度，对于这三种输入方法的检查，可以考虑以下方面：

合法输入和异常输入检查	检查点	举例
输入筛选器	开发人员是否正确地实现了该功能? 是否可以绕过屏蔽器?	如:设计一个下拉选择框，可选择数字 1-10，但是用户可以在下拉框种输入字母，这就是绕过输入屏蔽器的异常行为。
输入检查	验证代码 IF…ELSES 分支中对输入值的处理; 验证代码中对输入类型的处理; 验证代码中对输入越界或溢出硬件计算范围的处理;	如：代码中对 x,y 为{0,1}组合的处理，只判定了 x=0 and y=0，x=0 and y=1，x=1 and y=0 三种情况，试试 x=1 and y=1 的输入值; 如：硬件处理范围为 0-65535，而软件输入值范围为-65535-65535，试试-1 的输入值。
异常代码处理	异常反复触发可能引起程序崩溃; 不要忽略输出日志; 查看出现异常时，调用该程序的其他模块;	如：程序运算溢出导致内存增加，溢出异常频发出现可能导致内存溢出; 如：有的程序间调用异常不会在前端显示，但是可能输出到日志，注意日志的使用。

• 常规输入和非常规输入

常规输入指的是程序定义的、开发人员计划中的输入，也是用户经常使用的输入；非常规输入可以是那些使用极少的输入，或在开发人员计划之外的输入。

例如：常见的输入测试来说，非常规输入可以试试特殊字符@￥%&等。

• 默认输入和用户自定义输入

开发人员必须设定一些默认输入（可以是定义的参数或者 NULL 参数），这些输入测试人员在测试时可以不做什么操作，选择使用默认值；用户自定义输入是用户忽略开发人员预置的输入，输入自己想要的内容。从测试人员角度，可以从以下方面考虑：

默认输入和用户自定义输入	检查点	举例
默认输入	不输入任何内容; 传递 NULL 或 NONE 参数;	如：启动软件时，使用默认配置;
用户自定义输入	删除默认值，保留空白字段; 输入其他非空字符;	如：启动软件时，默认配置为 true，改为 false;

• 使用输出来指导输入

列出程序可能的输出结果，选择感兴趣的结果逆推得到可能的输入组合。把输入和输出配对是常用的一种手段，可以保证所有有趣的场景都被测试覆盖到。

2)小结

局部探索性测试关于用户输入的测试点建议，总结如下图所示：

2.状态

设想一个场景：安装一个软件，初次安装和再次安装是不是能完全等同？！答案肯定是不能。因为初次安装和再次安装，环境配置、资源初始情况是不同的。初次安装可能需要设定一些环境变量，而再次安装时这些环境变量可能因为卸载时未被清理而保留下来，因此不需要设定。再者，初次安装硬件资源（内存、磁盘容量）和再次安装肯定是有所差异的，再次安装有可能会因为初次安装卸载后的残留文件占用硬件资源，导致再次安装失败（假设硬件资源紧张情况下）。

而这些，由于输入造成的累积效应或程序内部变量数值发生的变化，称作状态的改变。

1)如何测试软件状态

可以从以下方面考虑。

• 使用状态信息来帮助寻找相关输入

如果两个或多个输入存在某种关联，那么他们应该放在一起测试。而在测试过程中，我们需要观察状态信息对结果的影响，以此来确定需要输入的组合。例如：安装某个软件，需要设定所需磁盘空间大小，但需要大于 128MB，而 硬件空间只剩余 500MB。此时，设计软件安装测试时，超过 500MB 由于磁盘空间不足安装失败，则应该考虑到硬件软件配置范围应在 128MB 到 500MB 之间。

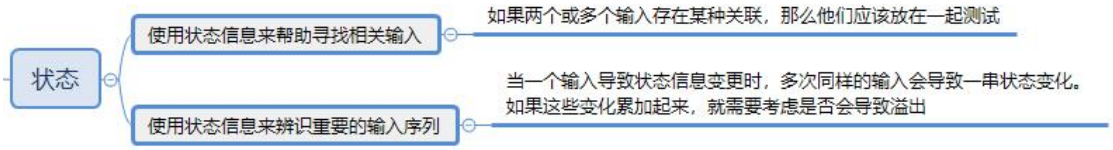
• 使用状态信息来辨识重要的输入序列

当一个输入导致状态信息变更时，多次同样的输入会导致一串状态变化。如果这些变化累加起来，就需要考虑是否会导致溢出。

例如：对软件接口进行压力测试时，内存等资源消耗会逐渐上升，试着观察加压持续一段时间后，软件内存释放是否及时，是否导致了内存溢出。

2)小结

局部探索性测试关于状态的测试点建议，总结如下图所示：



3.代码路径

一个程序有很多条代码路径，测试人员需要了解不同的输入在程序内部运行构成的路径。

如何测试代码路径

例如：测试人员可以使用 UML 流程图工具，如 visustin 绘制出代码路径。在测试过程中，画出代码路径图，使用最短路径法、长路径法等进行测试，观察程序在运行过程中的状态或输入数据的变化。

4.用户数据

测试人员可能时有面临这样一种情况：为什么我们测试时没有发现的故障，使用真实数据测试时故障就出现了。这是因为真实数据通常包含了测试人员不了解的相互关系和结构，而且测试人员也无法凭空推测出真实数据的相互关系和结构。那么应该如何测试用户数据呢？

1)如何测试用户数据

测试用户数据时，需要考虑到真实数据的几个特点：量大、隐私性、安全性。

随着用户数据库不断被增删修改，长年累月积累下来的数据量会非常大。而量大时就会引发存储问题。

例如：我们测试环境模拟数据量大小为 10GB，而真实数据量大小为 100GB，这就会引发大体量问题下的存储、运算、处理等问题，而这些问题可能在我们测试环境中不容易被触发。

此外，用户数据的隐私和安全问题也是测试时必须考虑的。当我们测试使用真实数据时，需要保护真实数据不被篡改、增删，且真实数据中的隐私（例如用户账号、密码等）不被暴露或恶意窃取。

2)小结

局部探索性测试关于用户数据的测试点建议，总结如下图所示：

5.运行环境

当我们把软件安装到一个崭新且它从来没见过的环境中时，可能会失败。为什么呢？因为环境本身也是一种输入源。所以测试人员在产品发布前应尽量尝试各种各样的环境。

1)如何测试运行环境

运行环境包括什么呢？总体来说，就是使用的操作系统和当前配置，以及同一操作系统上可能会和软件交互的其他程序，还包括网络连接情况、带宽、性能等因素。

运行环境测试可以从兼容性方面考虑，如：软件兼容性、硬件兼容性、系统兼容性、网络兼容性、接口兼容性等。

• 软件兼容性

指的是当前软件与其他竞品和非竞品软件的兼容性，是否可以同时运行。

例如：曾经著名的“3Q 大战”，要求用户只能在 360 杀毒软件和 QQ 软件中二选一使用，其实就是非兼容性的体现。

• 硬件兼容性

指的是指多个独立的硬件设备能否在操作系统的统一调度下协调工作发挥性能互不

排斥。

例如：不同的服务器生产厂商对内存、硬盘等插槽都有设定，要求符合响应规格的内存条或硬盘才能在其服务器上使用。这就是硬件兼容性的要求和体现。

• 系统兼容性

系统兼容性大家并不陌生，指的是对操作系统特定的要求。系统兼容性又可以分为：相同操作系统不同版本兼容性，以及不同操作系统兼容性。

系统兼容性	检查点	举例
相同操作系统不同版本兼容性	检查操作系统版本向后兼容性; 检查操作系统版本向前兼容性;	如：使用低版本的 windows xp 操作系统测试软件安装;
不同操作系统兼容性	检查同 Linux 操作系统兼容性; 检查同 Windows 操作系统兼容性; 检查同 IOS 操作系统兼容性; 检查同 Android 操作系统兼容性;	如：分别选择 linux 和 windows 操作系统安装、运行软件, 查看是否正常。

• 网络兼容性

查看软硬件在不同网络类型（如热点、WIFI、2G/3G/4G/5G）下的运行情况。

例如：5G 推广后，某些早前的手机不支持 5G 网络使用。

• 接口兼容性

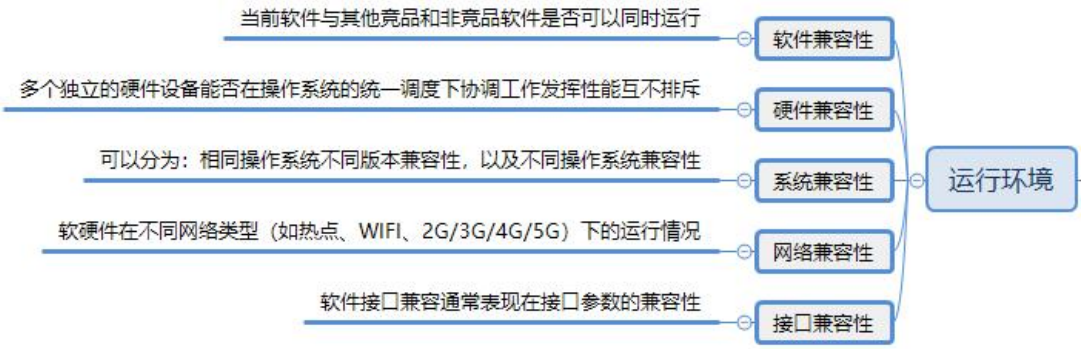
接口兼容性在软件测试中也时常触及，软件接口兼容通常表现在接口参数的兼容性。

例如：某个程序对外查询接口 V1 版本需要传入 2 个参数，而 V2 版本需要传入 3 个参数，而这种改变会影响其他调用该程序接口的应用。因此该程序在涉及或处理时需要

考虑到接口兼容性。如其他应用不传入第 3 个参数时，该程序提供默认值。

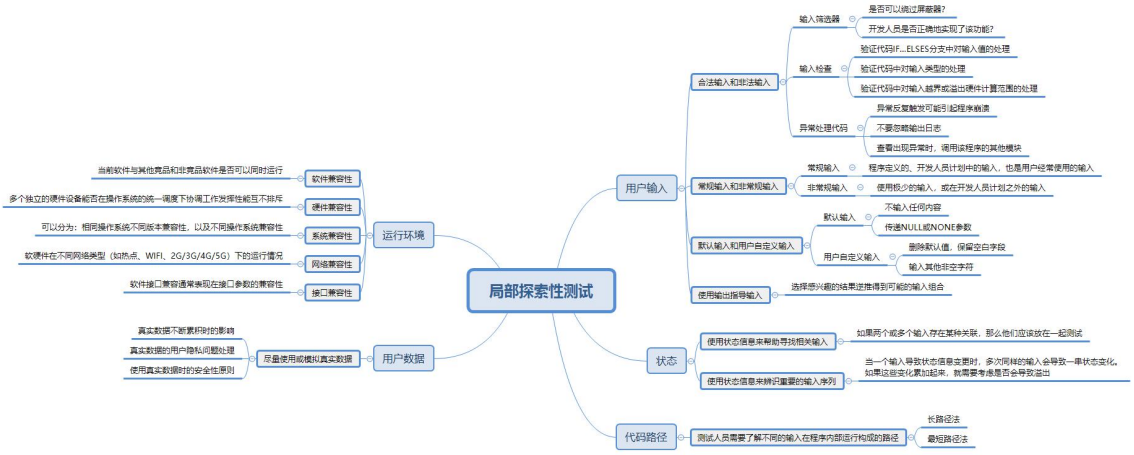
2)小结

局部探索性测试关于运行环境的测试点建议，总结如下图所示：



三、总结

测试本身是个很复杂的工作，需要测试人员结合自身经验、知识、技能等方面，才能有效地进行。本文讲述了 James A. Whittaker《探索式软件测试》一书中关于局部探索性测试地 5 个方面，及相应的某些启发点。最后总结为下图：



良好的开端是自动化测试成功的一半

◆ 作者：枫叶

俗语说得好：良好的开端是成功的一半，在 web 自动化测试中，首页登录是每个测试用例执行的前提。首页登录有验证码，也有为了安全而设置的验证码，对这些验证码进行自动识别，提取文字进行登录，是自动化测试良好的开端，否则一切都无从谈起。至于有些博客提到开发设置万能码等，如果开发团队支持也是可行，不在本文讨论范围之内。“自己动手，丰衣足食”！在即将迎来元旦之际，将此文分享给读者，希望读者们也有一个良好的自动化测试开篇。

首页登录——无验证码

用 Chrome 浏览器测试，运行首页，首页无验证码字段。


```
import time
from selenium import webdriver
# 要测试的网站
weburl = 'XXX'
```

```
driver = webdriver.Chrome()
driver.maximize_window()
driver.get(weburl)
driver.implicitly_wait(5)
# 账号、密码
login = 'XXX'
password = 'XXX'
driver.find_element_by_id('loginName').send_keys(login)
driver.find_element_by_id('loginPassword').send_keys(password)
driver.find_element_by_id('submit').click()
time.sleep(5)
driver.quit()
```

上述代码实例中，用实际的测试数据代替“XXX”。

登录——验证码

四种登录验证码的思路

输入式验证码：这种是最简单的一种，只要识别出里面的内容，然后填入到输入框中即可，也是企业内部网站很常用的一种。只要不是电商网站，一般用这种就足够了。这也是本文着重介绍的技术。这种识别技术叫 OCR，这里我们推荐使用 Python 的第三方库，tesserocr。对于没有什么背影影响的验证码，直接通过这个库来识别就可以。但是对于有嘈杂的背景的验证码这种，直接识别识别率会很低，遇到这种我们就得需要先处理一下图片，先对图片进行灰度化，然后再进行二值化，再去识别，这样识别率会大大提高。

滑动式验证码：模拟人去拖动验证码的行为，点击按钮，然后看到了缺口的位置，最后把拼图拖到缺口位置处完成验证。第一步：点击按钮。第二步：拖到缺口位置。点击式的图文验证和图标选择，图文验证：通过文字提醒用户点击图中相同字的位置进行验证。图标选择：给出一组图片，按要求点击其中一张或者多张。借用万物识别的难度阻挡机器。

这两种原理相似，只不过是一个是给出文字，点击图片中的文字，一个是给出图片，

点出内容相同的图片。这两种没有特别好的方法，只能借助第三方识别接口来识别出相同的内容，然后再使用 selenium 模拟点击即可。

宫格验证码：但是我们发现不一样的验证码个数是有限的，这里采用模版匹配的方法。我觉得就好像暴力枚举，把所有出现的验证码保存下来，然后挑出不一样的验证码，按照拖动顺序命名，我们从左到右上下到下，设为 1,2,3,4。上图的滑动顺序为 4,3,2,1 所以我们命名 4_3_2_1.png，这里得手动。当验证码出现的时候，用我们保存的图片一一枚举，与出现这种比较像素，方法见上面。如果匹配上了，拖动顺序就为 4,3,2,1，然后使用 selenium 模拟即可。

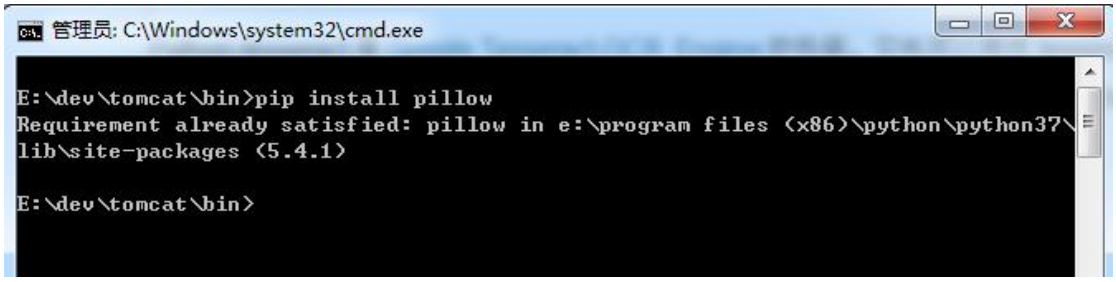
环境准备

识别的依赖关系：

Tesseract-ocr 可进行光学字符识别，它在 python 上的 package 是 pytesseract，要使用 pytesseract，就需要安装 tesseract-ocr。安装 python-tesseract，Python-tesseract 是 Google Tesseract-OCR 的 python 包装器。

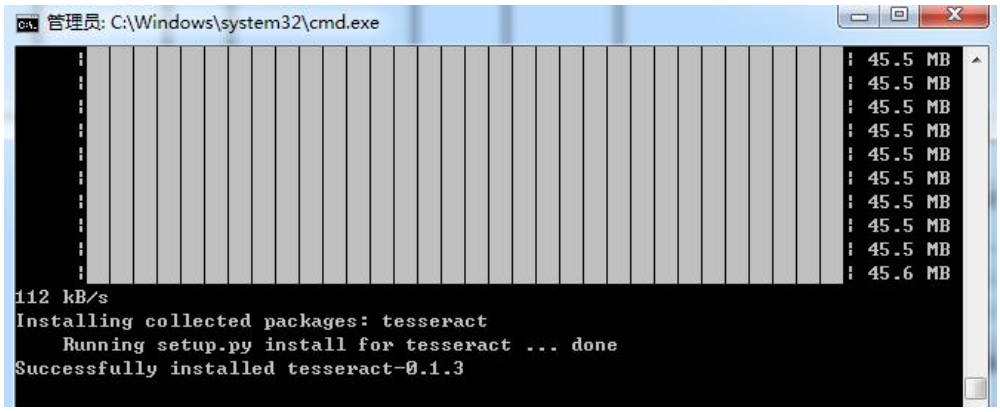
Python-tesseract 是用于 python 的光学字符识别（OCR）工具。也就是说，它将识别并“读取”图像中嵌入的文本。它是 Google Tesseract-OCR Engine 的包装。它也可以用作 tesseract 的独立调用脚本，因为它可以读取 Pillow 和 Leptonica 图像库支持的所有图像类型，包括 jpeg，png，gif，bmp，tiff 等。此外，如果用作脚本，Python-tesseract 将打印识别的文本，而不是将其写入文件。

pip install pillow

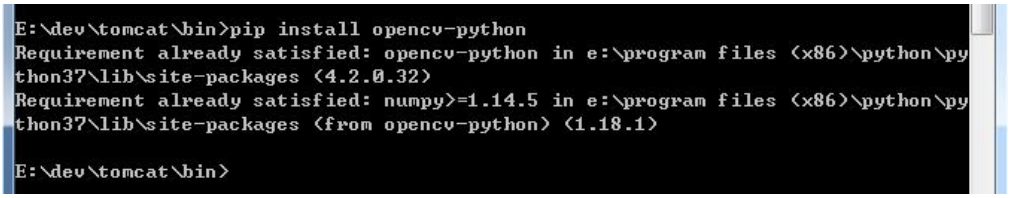


pip install pytesseract

pip install tesseract （有时需要执行多次）



pip install opencv-python



无背景的输入式验证码

编写 **ocr.py** 得到识别的图片

```
#! usr/bin/env python
#-*- coding:utf-8 -*-
# created on Mar26th
```

```
import os
import time
import selenium
from selenium import webdriver
import pytest
from PIL import Image
import pytesseract
import argparse
import cv2
# construct the argument parse and parse the arguments
ap=argparse.ArgumentParser()
ap.add_argument("-i","--picture",required=True,help="path to input picture to be OCR'd")
ap.add_argument("-p","--preprocess",type=str,default="thresh",help="type of preprocessing to be done")
args=vars(ap.parse_args())
# load the example picture and convert it to grayscale
image=cv2.imread(args["picture"])
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
# check to see if we should apply thresholding to preprocess the picture
if args["preprocess"]=="thresh":
    gray=cv2.threshold(gray,0,255,cv2.THRESH_BINARY|cv2.THRESH_OTSU)[1]
# make a check to see if median blurring should be done to remove noise
elif args["preprocess"]=="blur":
    gray=cv2.medianBlur(gray,3)
# write the grayscale picture to disk as a temporary file so we can apply OCR to it
filename="{}.png".format(os.getpid())
cv2.imwrite(filename,gray)
# load the picture as a PIL/Pillow images,apply OCR,and then delete the temporary file
def image_ocr(image_path, output_txt_file_name):
    image_text = pytesseract.image_to_string(image_path)
    os.remove(filename)
    with open(output_txt_file_name, 'w+', encoding='utf-8') as f:
        f.write(image_text)
```

```
            print(image_text)
            f.close()
    # show the output images
    cv2.imshow("Image",image)
    cv2.imshow("Output",gray)
    cv2.waitKey(0)
    执行脚本，即在 ocr.py 当前路径下，执行命令
    python ocr.py --image images/verify.png
```

编写 ocr2.py 得到识别的文本

先安装 tesseract-ocr-setup-4.00.00dev.exe，记住安装路径，例如 E:\Program Files (x86)\Tesseract-OCR，再编写代码：

```
import urllib3
import pytesseract
from PIL import Image
# 创建网络请求
http = urllib3.PoolManager()
# XXX 代替测试网址
res = http.request('get','XXX')
# 将下载的验证码，存储到指定路径
f = open('E:/pic/verify.png','wb+')
f.write(res.data)
f.close()
pytesseract.pytesseract.tesseract_cmd = 'E:/Program Files (x86)/Tesseract-OCR/tesseract.exe'
tessdata_dir_config = '--tessdata-dir "E:/Program Files (x86)/Tesseract-OCR/tessdata"'
image = Image.open("E:/pic/verify.png")
code = pytesseract.image_to_string(image)
print(code)
username='XXX'
password='XXX'
driver.find_element_by_id('username').send_keys(username)
driver.find_element_by_id('password').send_keys(password)
```

```
driver.find_element_by_id('verify').send_keys(code)
driver.find_element_by_id('login').click()
```

可以看到输出 code 的文本值：

```
"E:\Program Files (x86)\Python\Python37\python.exe" E:/ATS/testweb/ocr2.py
0 4 91
```

有背景的输入验证码

第一部分 图像识别（验证码）

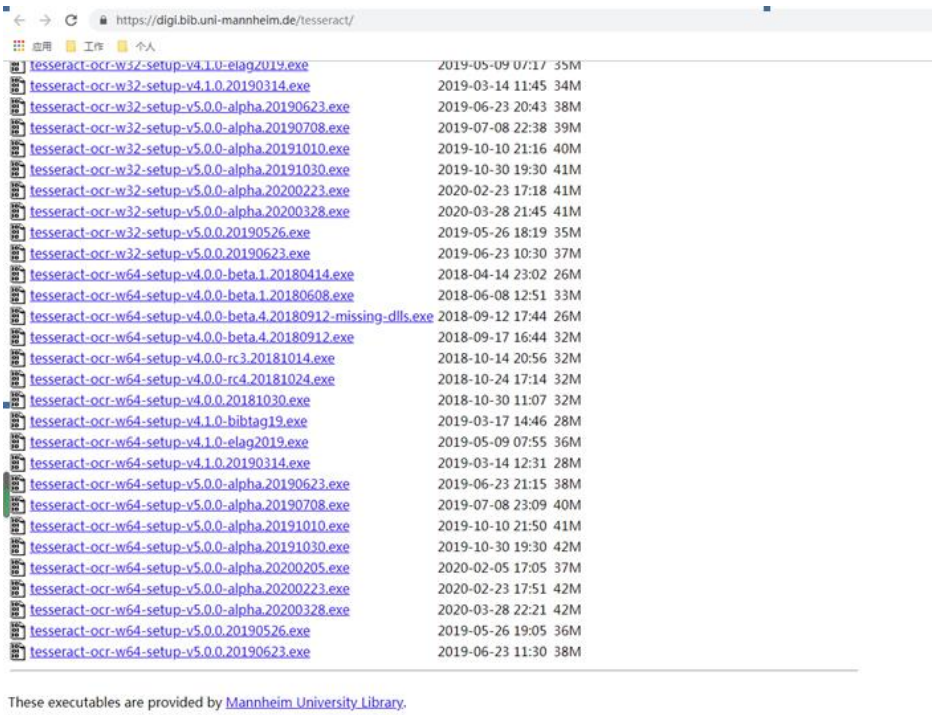
Tesserocr 是一个简单，对 pillow 友好的包装，围绕 tesseract-ocr API 进行光学字符识别（OCR）。在安装 tesserocr 之前要先安装 tesseract。tesseract 下载地址:http://digi.bib.uni-mannheim.de/tesseract。

因为 tesseract 与 tesserocr 有对应关系，注意选择匹配的版本下载：

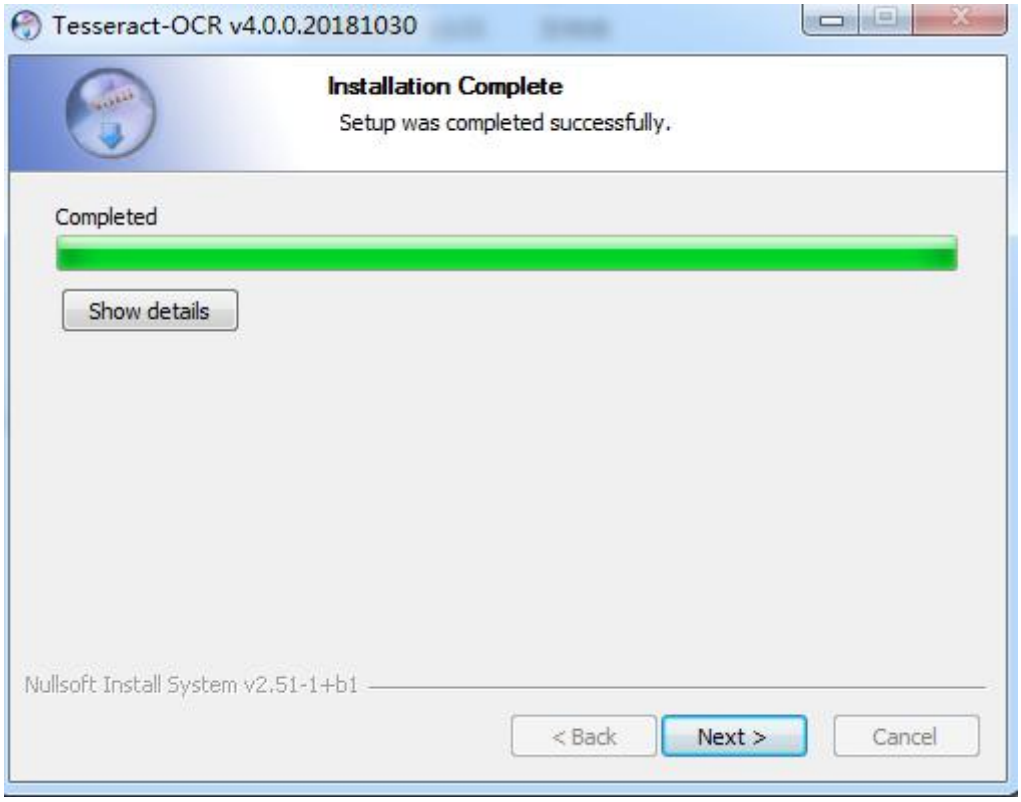
tesserocr v2.4.0 (tesseract 4.0.0)

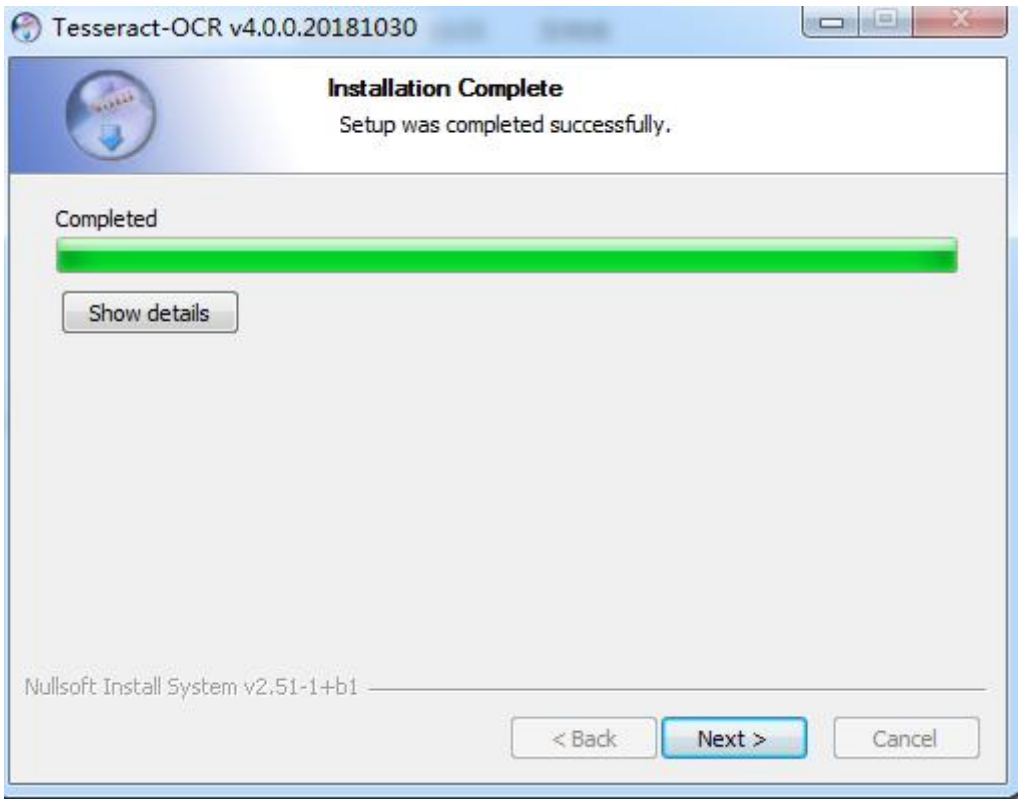
下载 whl：https://github.com/simonflueckiger/tesserocr-windows_build/releases

Windows 下安装 tesseract：

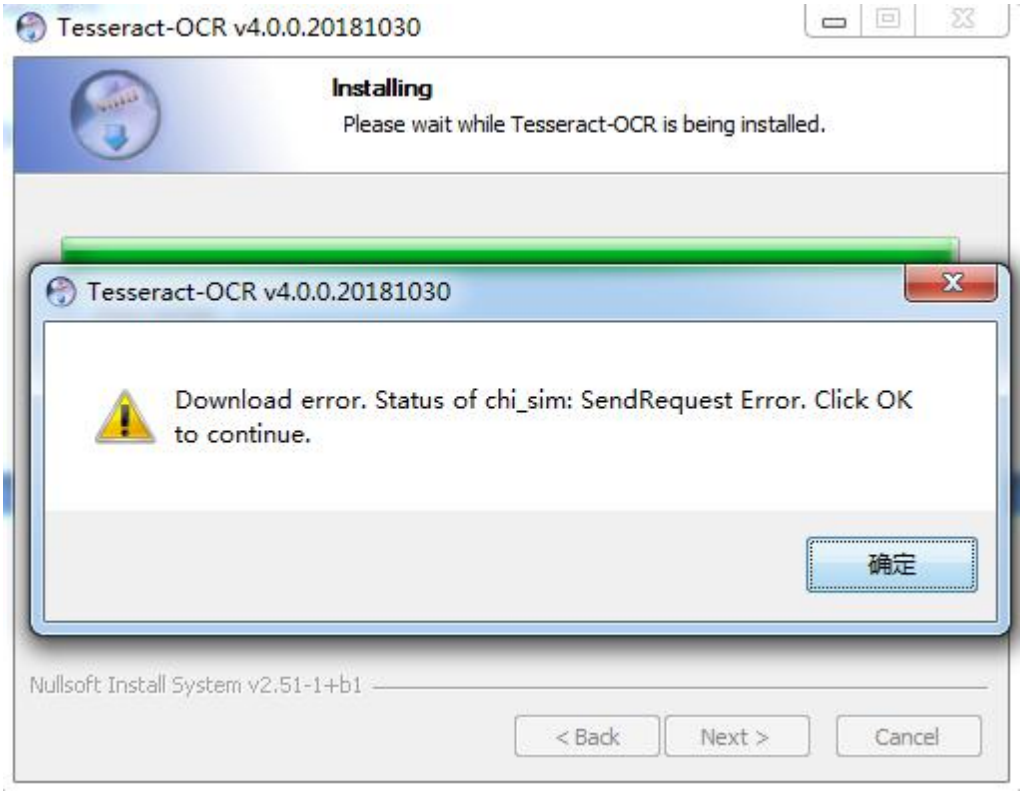


双击运行安装文件，勾选 Additional language data(download）选项来安装 OCR 识别支持的语言包，其他都按默认选项安装。





下载语言包的过程比较曲折，改为在 git 上下载需要的语言包，放在 tessdata 文件夹中。勾选这个选项的话， OCR 便可以识别多国语言。



下载语言包 https://github.com/tesseract-ocr/tessdata，简体中文，放到\tessdata 下。

bel.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
ben.traineddata	remove legacy model from indic and arabic script languages	2 years ago
bod.traineddata	remove legacy model from indic and arabic script languages	2 years ago
bos.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
bre.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
bul.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
cat.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
ceb.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
ces.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
chi_sim.traineddata	Update traineddata LSTM model with best model converted to integer	2 years ago
chi_sim_vert.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
chi_tra.traineddata	Update traineddata LSTM model with best model converted to integer	2 years ago
chi_tra_vert.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
chr.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago
configs	Add tessconfigs submodule and links for required tessdata files	7 months ago
cos.traineddata	Update LSTM Models to integerized tessdata_best for files < 25mb	2 years ago

Tesseract 设置环境变量

将安装路径添加到 path 环境变量中，例如 E:\Program Files (x86)\Tesseract-OCR;

并且为语言包添加到环境变量中，在环境变量中新建一个系统变量，变量名称为 TESSDATA_PREFIX，tessdata 是放置语言包的文件夹，一般在你安装 tesseract 的目录下，即 tesseract 的安装目录就是 tessdata 的父目录，把 TESSDATA_PREFIX 的值设置为 E:\Program Files (x86)\Tesseract-OCR \tessdata; C:\Python\Python37\tessdata 即可。

接下来，再安装 tesserocr。

（1）pip install tesserocr pillow 一般会安装失败；

（2）安装失败，就在下边的 2 个地址下载一个 whl 文件：

Tesserocr

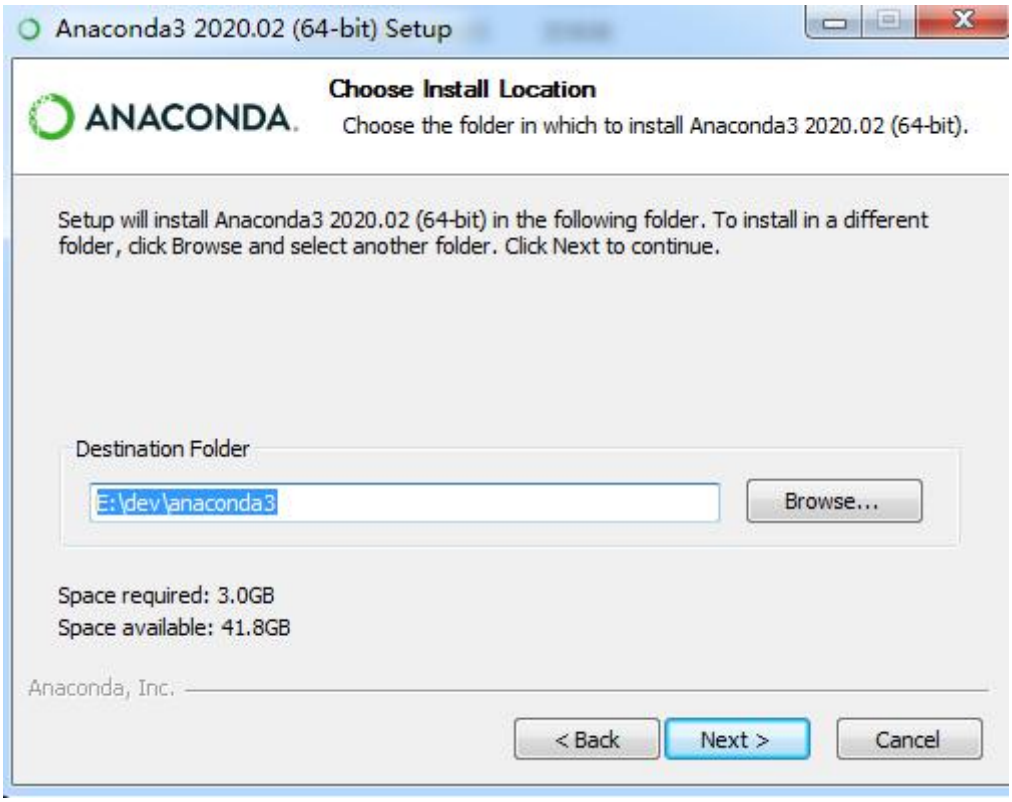
GitHub:?https://github.com/simonflueckiger/tesserocr-windows_build/releases

tesserocr PyPI:?https://pypi.python.org/pypi/tesserocr

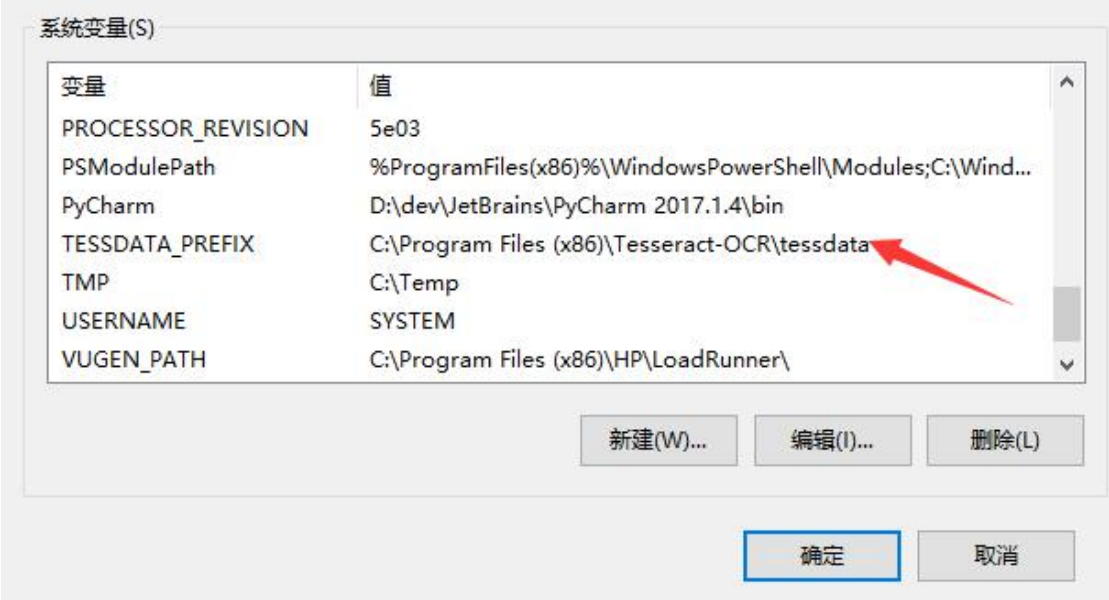
下载的是 tesserocr-2.4.0-cp37-cp37m-win_amd64.whl；

tesserocr-2.4.0-cp37-cp37m-win_amd64.whl

（3）安装 anaconda，https://www.anaconda.com/distribution/，下载安装包，双击运行安装文件，选择目标安装路径 E:\dev\anaconda3，其他按默认安装。



添加到环境变量-系统变量 path 中，TESSDATA_PREFIX 的值=C:\Program Files (x86)\Tesseract-OCR\tessdata



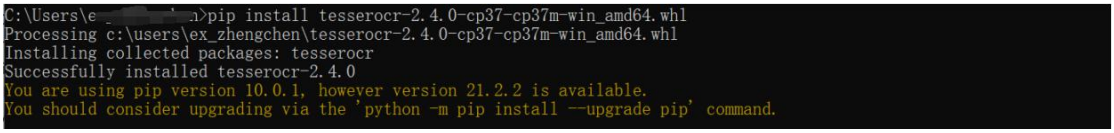
将 tesserocr-2.4.0-cp37-cp37m-win_amd64.whl 拷贝到用户的家目录下，比如 C:\Users\

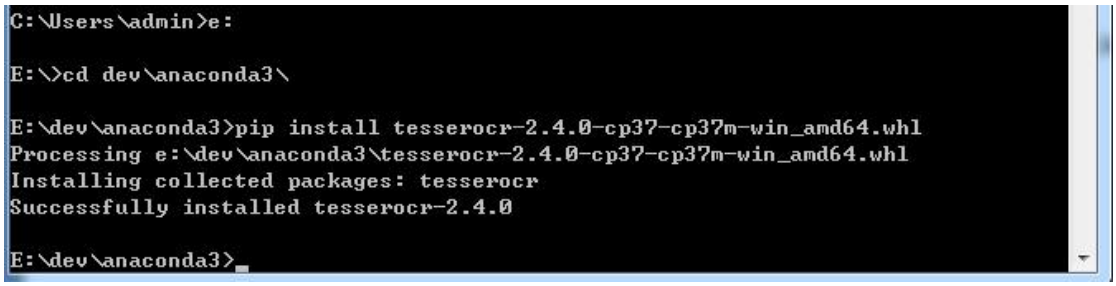
用户，再用 pip 安装这个 whl 文件。

pip install tesserocr-2.4.0-cp37-cp37m-win_amd64.whl

或者把 tesserocr-2.4.0-cp37-cp37m-win_amd64.whl 文件拷贝到 C:\Python37\Lib\site-packages 下，再执行 pip install tesserocr-2.4.0-cp37-cp37m-win_amd64.whl 命令。否则会提示找不到该文件路径。

安装完成 tesserocr-2.4.0





TESSDATA_PREFIX 的值注意这里不要加“;”，否则可能会遇到 RuntimeError: Failed to init API, possibly an invalid tessdata path: E:\Program Files (x86)\Tesseract-OCR \tessdata/ 的报错。

运行 **py** 文件

```
from PIL import Image
import tesserocr
image = Image.open('14252.png')
result = tesserocr.image_to_text(image)
print(result)
```

结果为 0278

```
E:\ATP\testwebfirst\testcase>python test_cr5.py
0278
```

注意：

截图的文件类型应为 png，若是 jpg，截图函数可以跑通，但是后面切割识别会出错！

第二部分 图像切割（验证码）


```
def jietu():
    elementcode=driver.find_element_by_id('verify_img')
    # 获取左上角坐标
    left=elementcode.location['x']
    top=elementcode.location['y']
    # 获取右下角坐标
    right=elementcode.size['width']+left
    bottom=elementcode.size['height']+top
    # print(left,top,right,bottom)
    # 切割
    img=Image.open(screen_save_path)
    codeimg=img.crop((left,top,right,bottom))
codeimg.save("E:/pic/verifycode.png")
```

```
    # 第三方接口，后面会具体介绍
    def coderegconize():
        try:
            r = ShowapiRequest("https://route.showapi.com/932-2", "168625",
"566bb29c35da4556953c03ed41dfedd8")
            # 接口的参数
            r.addBodyPara("typeId", "14")
            r.addBodyPara("convert_to_jpg", "0")
            r.addFilePara("image", "E:/pic/verifycode.png")
            picture = Image.open("E:/pic/verifycode.png")
            # code = pytesseract.image_to_string(picture)
            regconizecode = tesserocr.image_to_text(picture)
            # print(type(regconizecode))
            print("识别结果为：",regconizecode)
            return regconizecode
        except Exception as e:
            print("未识别到验证码")

    jietu()
    coderegconize()
```

```
"E:\Program Files (x86)\Python\Python37\python.exe" E:/ATS/testweb/test_cr3.py
web login test start...
E:\ATS\testweb\ShowapiRequest.py:22: ResourceWarning: unclosed file <_io.BufferedReader name='E:/pic/verifycode.png'>
  files[key] = open(r"%s" % (value_url), 'rb')
识别结果为： 3869

识别结果为： 3869
```

第三方接口

对于有背景的验证码，推荐第三方接口——易源，网址 https://www.showapi.com/console#/dashboard，具体参数说明请参考易源网站，这里就不引用了。

```
        rq = ShowapiRequest("https://route.showapi.com/184-4",
"193284","991cd8e44cbb45bc92e2a83353888fdd")
        # r.addBodyPara("img_base64", "")
        rq.addBodyPara("typeId", "35")
        rq.addBodyPara("convert_to_jpg", "1")
        rq.addBodyPara("needMorePrecise", "1")
        rq.addFilePara("image", "./images/verifyimg.png")
        result = rq.post().json()
        # print(result)
        recognizecode = result['showapi_res_body']['Result']
        print("识别结果为：", recognizecode)
```

```
"E:\Program Files (x86)\Python\Python37\python.exe" E:/ATS/Test_webnew  /testcase/testdir/test_  .py
web login...
识别结果为： emta4
```

附上完整的代码：

```
def setUp(self):
        # XXX 代替要测试的网站
        weburl = 'XXX'
        self.driver = webdriver.Chrome()
        self.driver.maximize_window()
        self.driver.get(weburl)
        time.sleep(5)
    print("web login...")
        number = 3
        for i in range(number):
            try:
                file = os.path.abspath(os.path.join(os.path.dirname(__file__), ".")) + '/picture/'
                now = 'nowpic'
                screenfile = file + now + '.png'
                self.driver.save_screenshot(screenfile)
                def codeimage():
                    codelement = self.driver.find_element_by_id('verify_img')
```

```
                left = codelement.location['x']
                top = codelement.location['y']
                right = codelement.size['width'] + left
                bottom = codelement.size['height'] + top
                # print(left,top,right,bottom)
                img = Image.open(screenfile)
                codeimg = img.crop((left, top, right, bottom))
                codeimg.save("./images/verifyimg.png")
    # 验证码图片函数调用
            codeimage()
            def coderecognize():
                try:
                    rq = ShowapiRequest("https://route.showapi.com/184-4",
"193284","991cd8e44cbb45bc92e2a83353888fdd")
                    # r.addBodyPara("img_base64", "")
                    rq.addBodyPara("typeId", "35")
                    rq.addBodyPara("convert_to_jpg", "1")
                    rq.addBodyPara("needMorePrecise", "1")
                    rq.addFilePara("image", "./images/verifyimg.png")
                    result = rq.post().json()
                    # print(result)
                    recognizecode = result['showapi_res_body']['Result']
                    print("识别结果为：", recognizecode)
                    return recognizecode
                except Exception as e:
                    print("未识别到验证码")

            # XXX 代替账号、密码
            user = 'XXX'
            pwd = 'XXX'
            codevalue = coderecognize()
            self.driver.find_element_by_name('username').clear()
            self.driver.find_element_by_name('username').send_keys(user)
```

```
                self.driver.find_element_by_name('password').clear()
                self.driver.find_element_by_name('password').send_keys(pwd)
                self.driver.find_element_by_name('verify').clear()
                self.driver.find_element_by_name('verify').send_keys(codevalue)
                self.driver.find_element_by_class_name('beg-pull-right').click()
                self.driver.implicitly_wait(20)
                assert  '首页' in self.driver.find_element_by_id('left_side_10001').text
                print("登录的用例执行通过")
                break
            except Exception as e:
                print("login failed.")
        time.sleep(2)
    def tearDown(self):
        driver = self.driver
        element = driver.find_elements_by_class_name('layui-nav-item')[2]
        ActionChains(driver).move_to_element(element).perform()
        time.sleep(2)
        driver.find_element_by_class_name('confirm-rst-url-btn-logout').click()
        time.sleep(2)
        driver.find_element_by_class_name('layui-layer-btn0').click()
        time.sleep(1)
        assert driver.find_element_by_name('username')
        print("web logout.")
    driver.quit()
```

参考

官网:https://pypi.org/project/pytesseract/

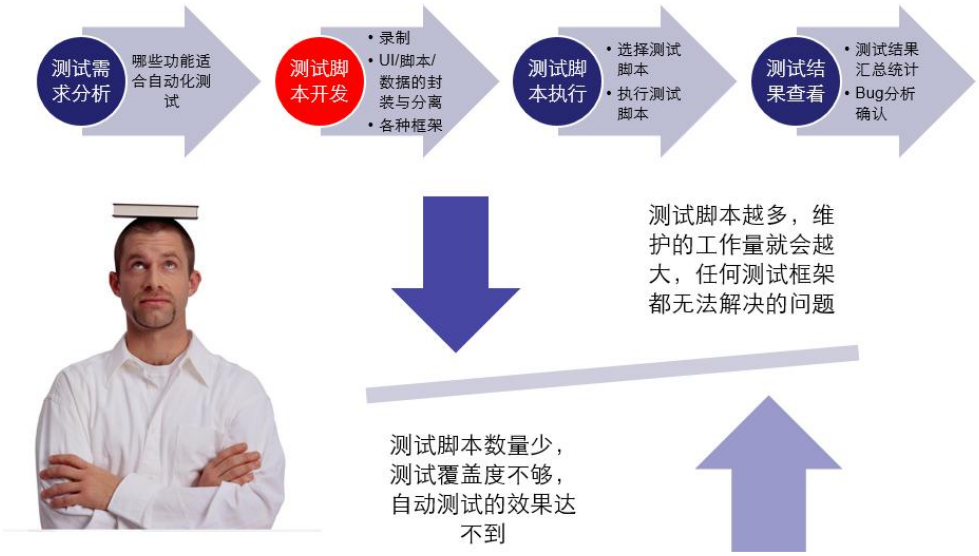
以及各大网站博客。致谢所有朋友!

没有脚本的自动化测试

◆作者：云竹

一．自动化测试的痛点：

说到功能自动化测试，我们可能首先会想到 QTP、Selenium 等诸多的自动化测试工具和解决方案，如何使脚本开发变得简单、如何使脚本维护变得容易、如何提高脚本开发的效率等话题，一直以来都是自动化测试领域关注的焦点问题，然而无论是哪种工具哪种解决方案，都离不开测试脚本的开发，有过脚本开发经历的人应该有体会，虽然设计良好的自动化测试架构可以简化脚本开发，但是面对成百上千上万的功能，开发和维护脚本始终不是一件容易的事，其复杂度绝不亚于一个开发项目。



脚本开发和维护工作量会与被测系统的复杂程度和变更有关，下面是一些举例：

• 页面输入项越多，脚本编写工作量越大。

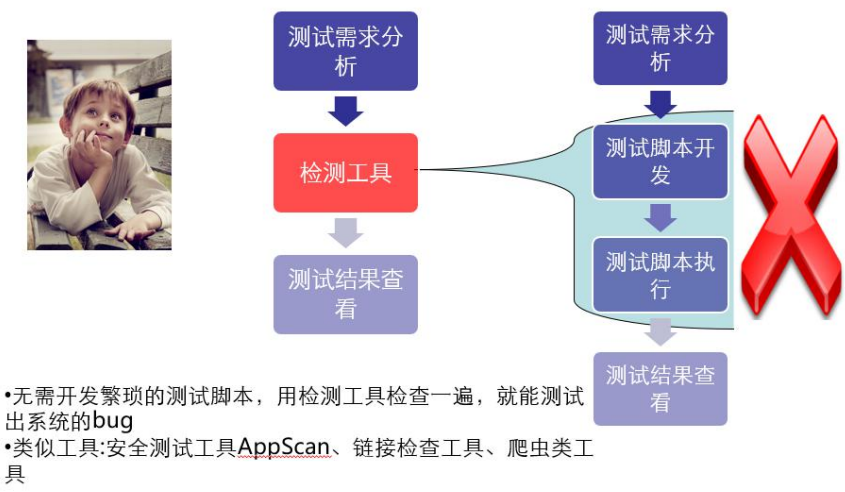
• 页面输入项名称的改变，脚本也需要调整修改，例如将“移动电话”改成了“手机号码”。

• 页面必填输入项的增减，会使得脚本不得不跟着调整。

• 页面输入项的类型改变，脚本也必然要做修改，例如“学历字段”原来是文本输入框，后来改成了下拉选择框。

• 菜单名称修改，测试菜单的脚本需要调整。

自动化测试之所以实施起来困难重重，究其根源还是测试脚本导致，那么有没有一种自动化测试是不需要编写一个个测试脚本的呢？



做过安全测试的应该知道 AppSan 这个工具，只要告诉 AppSan 要检测的系统地址，进行一些简单的用户登录的配置，AppScan 就可以自动对被测系统进行检测，找出可能的安全性问题并生成报告，不需要去写任何的测试用例和测试脚本(手动探测还是需要录制脚本的)。如果在做功能自动化测试时，也有一个像 AppScan 之类的工具，不需要写那些繁琐的测试用例和测试脚本，只要扫描一下，就能测试出系统的功能性问题，那将是多么令人兴奋的事情！然而，市场上并没有也不可能有这样的通用工具，毕竟功能测试与非功能测试不同，是与被测试系统紧密相关的，即使有这样一个工具也一定是专门针对某一个被测系统的。

幸运的是，笔者的被测系统是典型的信息处理类的 web 系统，模块多、菜单多、功能点多、页面风格及交互方式统一规范，正式由于这些特点，使得这种无脚本自动化测试成为可能。

下面就来介绍一下这种扫描式的自动化测试是如何进行的。

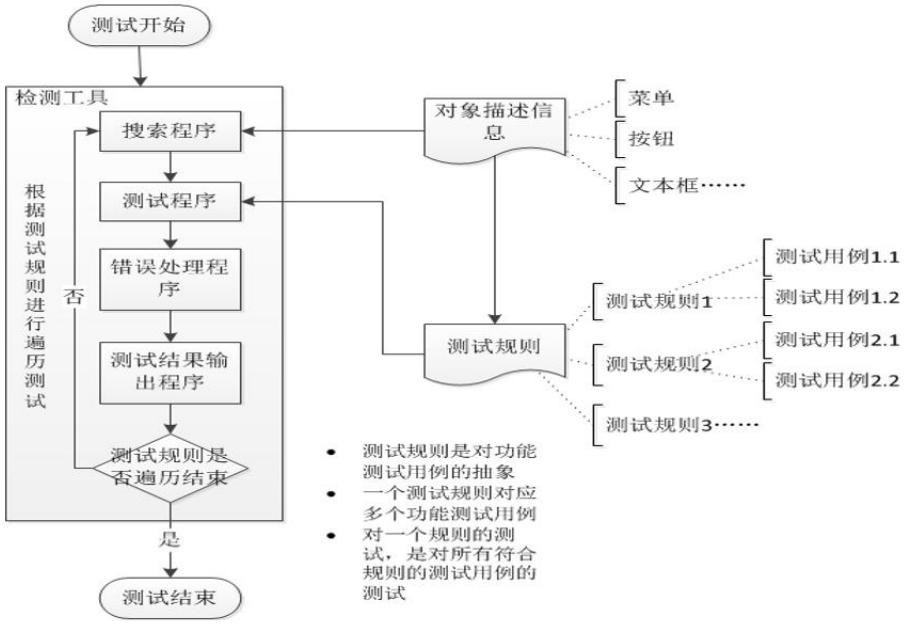
二．扫描式自动化测试

针对被测系统的特点，采用的自动化测试策略是：只针对基本的增删改查功能进行自动化测试。

按照粗略的统计，被测系统中功能模块近 30 个，菜单近 800 个，按照平均一个菜单内有 5 个功能点计算，总的功能个数将达到 4000 个(实际功能数应该远高于此)，假设在这 4000 个功能中，基本的增删改查功能占到 25%，将会有 1000 个功能需要进行自动化测试。如果每个功能对应一个测试脚本，将会有至少 1000 个测试脚本需要开发和维护。

针对以上需求和特点，结合过往自动化测试的实践经历，笔者提出了“扫描式自动化测试”的方案，其核心思想就是将操作步骤类似的测试用例或者测试脚本，抽取出通用的测试规则，针对测试规则去编写扫描测试程序，只要是符合测试规则的功能，都可以自动被扫描发现并被自动测试，无需编写一个一个的测试脚本。

扫描测试程序的大概流程如下：



下面是测试用例和测试规则的举例：

用例编号	测试步骤	预期结果	测试规则	测试对象
1	点击"用户管理"菜单	菜单正常显示没有异常	点击菜单，菜单显示没有异常	菜单
2	点击"管理员管理"菜单	菜单正常显示没有异常		
3	点击"用户角色管理"菜单	菜单正常显示没有异常		
4	点击"管理角色管理"菜单	菜单正常显示没有异常		

用例编号	测试步骤	预期结果	测试规则	测试对象
1	1. 点击"用户管理"菜单 2. 点击"新建"按钮 3. 输入"用户名"、"用户密码"信息 4. 点击"保存"按钮	新建操作成功，出现操作成功提示信息	点击菜单，点击新建类型的按钮，在输入页面给各类输入项赋值，点保存按钮，新建操作成功	菜单 新建类按钮 保存类按钮 各种输入对象(文本框、下拉框、日期框、选择控件等)
2	1. 点击"管理员管理"菜单 2. 点击"新建"按钮 3. 输入"管理员用户名"、"管理员密码"信息 4. 点击"保存"按钮	新建操作成功，出现操作成功提示信息		
3	1. 点击“用户角色管理”菜单 2. 点击“添加”按钮 3. 输入“用户角色编码”、“用户角色名称”信息 4. 点击“确定”按钮	新建操作成功，出现操作成功提示信息		
4	1. 点击“管理角色管理”菜单 2. 点击“添加”按钮 3. 输入“管理角色编码”、“管理角色名称”信息 4. 点击“确定”按钮	新建操作成功，出现操作成功提示信息		

新建用户、新建管理员、新建用户角色、新建管理员角色 4 个测试用例，正常是要写 4 个测试脚本，现在都不需要了，只需要运行一下扫描程序，就可以自动被发现和测试。

下面是对扫描式自动化测试的定义和特点总结。

• 什么是扫描式自动化测试：能够对被测系统的功能进行搜索并自动进行测试的过程。

• 适用场景：适合基本功能的测试，如增删改查等，不适合业务流程场景的测试。

• 主要特点：

- 无需编写特定功能的测试脚本，只有一个扫描程序即可完成所有增删改查的测试。

- 无脚本维护工作，例如页面增加了必填输入项，输入项名称变了，类型变了等，都不需要调整扫描程序，只有页面元素的定位属性变了，才需做相应调整。

- 测试的范围不固定，扫描到 10 个新建功能，就测试 10 个，扫描到 100 个，就测试 100 个，如果新开发了 100 个新建功能，只要是符合测试规则的，都可以被测试到，无需额外编写脚本。

- 测试数据不固定，系统中有什么数据就用什么数据，不会特意准备测试数据。比如修改功能的测试，默认选择列表中第一条数据，然后修改保存。随着系统的不断运行，扫描测试所使用的数据也会发生变化，可能会发现更多的问题。

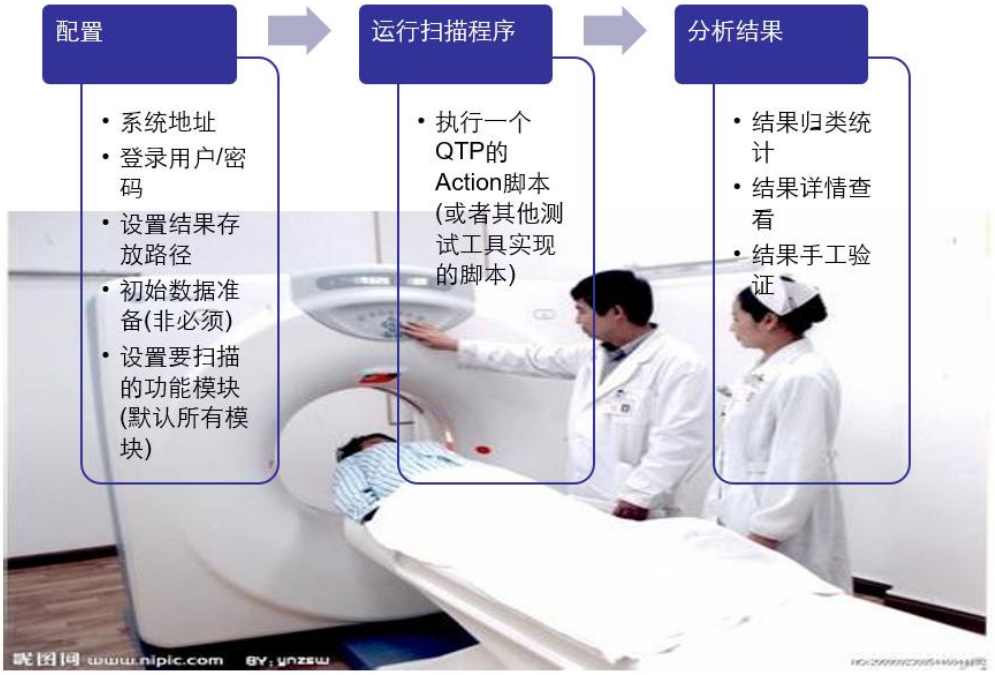
- 覆盖度高。针对一个个功能的测试脚本，通常会选择重要度高、使用频繁的功能来做脚本(毕竟是要投入脚本开发成本的)，覆盖度有限，采用扫描测试后，所有功能都会测试到，覆盖度大幅提升。

• 与常规自动化测试的对比

	本质区别	脚本维护	学习成本	适应变化程度	实施难度	投入产出
常规自动化测试	有脚本	有	有	低	复杂	低投入低产出 高投入低产出 高投入高产出
扫描式自动化测试	无脚本	无	几乎没有	高	简单	低投入中产出

	测试用例	测试数据准备	测试完整性	容错性	运行效率	测试管理支撑
常规自动化测试	自定义	必须	高	低	高	需要
扫描式自动化测试	受限于测试规则	不是必须	低	高	低	不需要

下面是运行测试的过程：



下面是扫描结果示例：

扫描项	菜单		新建				修改			
	800		476		378		359		246	
明细	点击结果	执行次数	点击结果	执行次数	操作结果	执行次数	点击结果	执行次数	操作结果	执行次数
	System Error	36	无新建类按钮	281	无保存按钮	29	无修改图标	394	无保存按钮	53
	您访问的功能不存在或未正确部署，请联系产品技术支持。	5	System Error	15	System Error	12	System Error	30	System Error	3
	成功	757	您访问的功能不存在或未正确部署，请联系产品技术支持。	20	出现脚本错误	4	您访问的功能不存在或未正确部署，请联系产品技术支持。	2	出现操作成功提示	230
	[illegible]未设置阶段目标值信息，不能收集	2	Incorrect result size: expected 1, actual 2	8	Incorrect result size: expected 1, actual 2	1	出现脚本错误	4	必填项不能为空	8
			新建按钮点击成功	407	出现操作成功提示	245	Incorrect result size: expected 1, actual 2	8	未出现弹出页面	3
			必填项不能为空	2	必填项不能为空	29	修改页面进入成功	299	SQL语句存在语法错误	1

三．扫描式自动化的实现

具体实现不依赖于某种工具，qtp、selenium、robotframework 等都可以实现，笔者用 qtp 和 robotframework 均做了实现，实现过程大同小异，一些关键点总结如下：

1.与常规自动化测试相同，页面对象同样需要做封装，菜单、文本输入框、下拉框、

日期选择框、列表选择、树型选择、新建类按钮、编辑类按钮、保存类按钮等等。

2.菜单遍历逻辑实现，扫描程序中最核心的循环是菜单遍历，登录系统后，根据菜单的元素定位属性，找到所有菜单对象，然后循环点击，每点一个菜单后，对当前页面做增删改查测试，然后进入下一个菜单继续测试。要根据具体被测系统的菜单导航方式来实现。

3.页面自动输入实现，当新建页面打开后，要能完成所有输入项(至少是必填项)的自动数据录入，例如文本框输入唯一字符串，下拉框选择第一条记录，日期框输入当前日期等等，如有特殊业务需求再做特殊处理。

4.批量定位元素的应用，扫描程序中大量使用了批量元素定位，例如找到一批菜单，循环点菜单，页面中找到多个文本输入框、多个下拉框，循环自动赋值。

5.错误检查处理程序，当点击某个菜单、某个新建、某个保存时，需要做断言检查，成功则继续，失败则跳过，扫描程序不会因为断言失败而结束。

四．小结

从事自动化测试已有多个年头，从最早的QTP录制脚本，到QTP描述性编程、selenium，再到robotframework，都有做过尝试，但大多都因为投入少，时断时续，见效慢，耗时耗力而无功而返，可谓是此起彼伏，一波三折，最终还是采用了这种扫描式的自动化测试方案，一直沿用至今。还是那句话：只有适合的才是最好的，希望通过本文，可以给大家在自动化测试领域多一些启示。

浅谈智能驾驶中的持续集成(CI)测试

◆ 作者：狼图腾

序

不知不觉，2021 年已经进入尾声，作为智能驾驶领域的一枚软件测试人员，在部门做自动化有小一年时间了，从早前的懵懂无知，到现在现在稍稍清晰，也还在谨慎的前进着，很庆幸能够从事自动驾驶相关的测试工作，于是想着结合自己的历程精简出一些经验吧，其实也谈不上经验，就算是一种经历吧，如果有更深认识的朋友，欢迎互相讨论，谢谢。

近年来电动汽车发展的如火如荼，各大企业争相进入造车领域，这使得自动驾驶有了一个很好的载体，智慧出行，必然是接下来智慧生活乃至人工智能领域最为重要的发展方向之一，并且由于方兴未艾，各方力量纷纷涌入，造手机、无人机的，做房地产的，以及传统车企也被迫向新能源汽车转型，如：极狐阿尔法 S(北汽)、极氪 001(吉利)、智己(上汽/阿里)、奔驰 EQS……一款比一款吸引人眼球，甚至福特的图腾车型野马 Mustang 也开始电动化、智能化……大有逐鹿中原之势。下面就聊一聊智能驾驶。

目前市面上做智能驾驶的主要部分为两大类，一类是以华为/百度/大疆/滴滴等为代表的智能驾驶技术供应商，一类是以特斯拉/小鹏等车企为代表的自研智能驾驶技术。像华为、百度费那么大劲研发自动驾驶功能，并不是为了单独给一家车企使用的，因为他们不造车，所以对于他们来说，最理想的情况就是这个市场上所有的品牌都用他们的技术，这样不但收入最高，研发的成本也可以分摊得很低。而像特斯拉/小鹏等车企为代表的自研智能驾驶技术芯片的公司的优势在于后期将不存在“受制于人”这种短板(就像当前手机上的的“芯片荒”)。

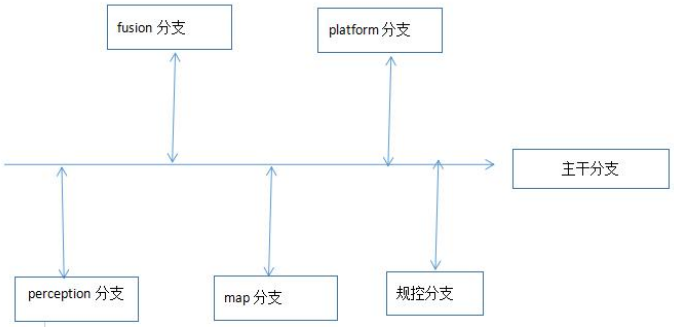
今天的文章中会向大家介绍一下小编在项目中运用到的持续集成(CI)思想。

什么是 CI?

首先要明确两个概念：何为持续？何为集成？持续就是每完成一个完整的部分，就向下个环节交付，发现问题马上调整，不会放大到其他部分和后面的环节；集成就是软件个人研发的部分向软件整体部分交付，以便尽早发现个人开发部分的问题。总而言之，对于测试团队来讲，CI 是一种将被测代码频繁的集成到项目稳定分支的做法，包括构建、测试等，以保证新代码和原有代码能正确地集成在一起，此处所说的“被测代码”指的是单独分支上开发的功能，这部分代码会被测试然后集成到主线上。

一般的开发团队都拥有自己的源代码版本控制库来存储其项目（一般为 git 或者 SVN）。稳定的分支（master）通常是作为主线分支，新功能的开发一般很少会在主线上完成。开发人员在开发自己模块的新功能时，会创建自己的分支。当代码开发完毕，开发人员 push 了自己的代码到自己的 branch 后，会执行 pull 请求，此时一些基本的自动化测试会针对这个分支执行。

以本人所在的智能驾驶项目为例，自动驾驶的原理首先是通过摄像机、激光雷达、毫米波雷达、超声波等车载传感器来感知周围的环境信息(相当于人类眼睛的作用)，比如车辆轮廓、地面上的车道线(道路结构认知)等信息，然后依据所获取的信息由适当的工作模型来制定相应的策略，如预测自车与其他车辆、行人等在未来一段时间内的运动状态，并进行避免碰撞路径规划。在规划好路径之后，由控制系统控制车辆沿着期望的轨迹行驶。自动驾驶涉及到传感器环境感知，高精地图/GPS 定位系统，多种数据融合，决策与规划算法运算，运算结果的电子控制与执行等过程。基于以上的多个模块，便产生了不同的分支模块，不同分支测试过后的代码就会不断集成到主干上。



为什么要做持续集成？

在软件工程里，有一个指标叫做质量成本，即项目出现质量问题的修复成本。相信所有刚接触软件测试学习的人应该都听说过这个理论：随着时间的推移，质量成本以指数级趋势增加。所以我们做持续集成的根本目的就是为了尽可能早地发现、解决质量问题。

我所在的项目是如何做持续集成的？

持续集成是为了让整个项目运转更加高效，所以必须保证工程的构建和测试运行效率足够高。

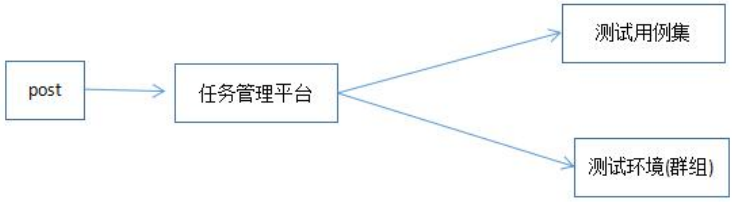
首先是工程的构建，相信很多人应该听说过 jenkins，jenkins 在主要的功能就是版本的构建，因为项目中每个分支每天会在凌晨进行版本构建并触发冒烟测试任务，所以小编运用 jenkins 的情况较少，只是在手动构建版本中偶尔使用过，由于目前掌握的不多，也在慢慢的学习中，后续或许会专门出一篇进行实例讲解，感兴趣的读者也可以了解一下这方面的知识，毕竟 jenkins 在业内也是一个很常用工具，在此只讲一下每日构建流水线的基本流程:拉取代码——>编译——>构建——>代码扫描——>打包部署——>自动化测试——>测试报告——邮件发送。

其次就是精准的测试，为了保证工程的高效运行，采用的是冒烟测试，如果每次只跑与这次代码相关的测试用例是最好不过的，不过这种做法很难实现，因为相关性不好评估，所以通常是筛选出 N 条测试用例(通常为三四百，具体根据版本特性进行增删)，每次持续集成就定期去跑全量的这些测试用例，一般会在凌晨对指定的分支调度冒烟测试用例集，第二天来公司查看测试报告就行了。这样每天冒烟测试都能保证基本功能可以被验证，对于智能驾驶，这部分用例主要就是一些诸如红绿灯、环岛、并线等不同的道路场景。

我所在的 ADS(Autonomous Driving Solution)自动驾驶解决方案是由华为自研的自动驾驶系统，测试产品是 MDC(Mobile Data Center)，作为一种移动数据中心，可以把它看成计算机中的 CPU。每天的基本任务就是上文所说的 daily 版本冒烟测试结果分析，保证各分支版本无阻塞问题以及业务问题的定位跟踪等。

前文有提到每天凌晨会自动构建 daily 版本并启动冒烟测试任务，当然也可以使用

postman 来手动调度任务，请求体中包含所需创建的任务名称，目标版本号，版本构建(buid)号，自动化用例集名称，以及自动化脚本所需的环境集群等，post 请求发送成功后便可在任务管理平台上查看当前所创建的自动化任务。



自动化用例结果的分析及问题修复

最后就是测试用例结果的分析，如果有问题，及时推进修复。这也是整个过程中最主要的工作内容。

自动化用例其实就是拿实际路测的数据包跑 MDC 最新的感知等模块,完成体验分析，如碰撞风险/S 型轨迹等。

任务执行过程中每跑完一条脚本都会将测试结果保存在本地工作站中，通过 Linux 命令去查看失败用例相关信息，如用例名称，失败原因，数据连接等，根据失败原因以及数据链接中的日志等信息去定位具体导致失败的原因,将失败用例结果反馈给开发推动解决，问题解决后会由分支同步推送主干，并验证主干是否解决。

ps:可能在很多人的认知里，自动化脚本是由测试人员编写的，其实不然，本项目就是个例外，自动化脚本是由开发用 python 写的，脚本内容基本相同，最大的区别就是存放的数据片段不同，每条用例都是由 test_script.py 格式的脚本及 config.json 配置文件组成，测试脚本中主要包含以下内容：①环境准备；②把包 copy 到相应目录；③订阅输出的 topic；④播包；⑤结果判断。json 格式的配置文件里存放的是需要播的包，如数据片段名称等内容。测试脚本基于 pytest 框架。（关于 python 自带的 unitest 框架和第三方 pytest 框架区别简单的做了一个总结放在文末附表中，各位可以对照参考）

附表：unnitest 与 pytest 区别：

	unittest	pytest
总体	用例格式复杂，兼容性差，插件少，方便二次开发。	pytest 用例格式简单，方便快捷，可以执行 unittest 风格的测试用例，有较好的兼容性。插件丰富。
用例编写规范	①测试文件必须先 import unittest ②测试类必须继承 unittest.TestCase，要有 unittest.main()方法 ③测试方法必须以“test_”开头	①测试文件必须先 import pytest ②测试文件名必须以“test_”开头或者"_test" 结尾（如：test_script.py），import unittest ③测试类命名以"Test"开头 ④测试方法必须以“test_”开头
用例执行	默认执行全部用例，也可以通过加载 testsuit，执行部分用例，提供了 setUp/tearDown，只能针对所有用例	pytest 提供了@pytest.fixture()这个装饰器，可以自定义方法函数，以及可以更加灵活的调用类和方法。
断言	很多断言格式(assertEqual、assertIn、assertTrue、assertFalse)	只有 assert 一个表达式，用起来比较方便
报告	HTMLTestRunner 包可生成 HTML 格式的测试报告	有 pytest-HTML、allure 插件
失败重跑	不支持	支持用例执行失败重跑，pytest-rerunfailures 插件

笑谈 3 年测试生涯

◆作者：节节高

题记：
从事测试工作已 3 年有余了，今天想聊一下自己刚入门时和现在的今昔对比，虽然现在也没什么成就，只能说笑谈一下自己的测试生涯，各位看官就当是茶余饭后的吐槽吧，另外也想写一写自己的职场感想，希望对刚开始工作的小伙伴能够有些帮助或启发。

选择测试的原因

我大学学的是计算机专业，对于 IT 互联网行业，那也算是正统科班出身吧，大四那年就进了一家还挺大的软件公司实习，开发公司的自主产品，一个线上管理软件。所在的团队各个都是猫头鹰界的扛把子，动不动就干到半夜，我一个实习生，早走显得不够努力，只能也跟着硬熬，好在当时带我的组长照顾，让我早点下班，但其实在实习的近半年里，我也没有准时下班过。

写到这里，大家也明白了我进测试行业的原因了，因为开发界，我扛不住啊！“打铁还需自身硬”是不是？我自身条件不行，缺觉，熬不住，尤其发版前的熬夜，那惊心动魄的场面（其实后来发现，测试也一样），恕我只能知难而退。

当然后来也才知道并不是所有开发都是这样的，我那时候所在的团队，他们都非常敬业，照他们自己所言，一旦开始码代码，就进入了人码合一的境界，熬夜只是无心插柳而为之。

入门测试

进入测试界，其实也是因为之前提到的这家公司，里面一位资历比较老的同事，跳槽了，去了家规模几十号人的公司，让我毕业后去那里跟着他继续干，我说不干开发了，太累人，没想到他说那就去他那儿做测试吧，测试简单。于是我就这么同意了。

于是我毕业后的第一份工作，就是一个黑盒测试员。每天只知道机械地点点点，以

至于脑子日渐白痴化，在我眼中，世界上只有两种 BUG：大 BUG 和小 BUG。除了导致宕机、闪退、死循环的 BUG 是大 BUG，其他都是小 BUG，后来在和程序员对接中，知道了 fatal error、warning 等等，我才恍然大悟！我怎么也是计算机专业正统科班出身，怎么能把 error 等级都忘记了！真是点点点给点傻了！

以前会觉得测出一个致命错误非常开心，感觉自己干了大事似的，耀武扬威去和开发炫耀，现在不这么认为了，因为每次出现这种问题，开发都要调好久，而后还得我在旁边绞尽脑汁复现问题，调试完了还得无止尽地测试，现在，我只希望少出 BUG 为妙。

时间久了，每天重复的点点，让我觉得自己就像是个只会机械化操作的行尸走肉，我突然意识到，这项工作毫无意义，而且太过单调，我需要进步和新的契机。

当时促使自己进步的还有一个原因，就是薪资的问题，当时我就拿着 5K 多块钱的工资，相比班里的从事开发的那些，确实低了点，这让我挺不好受的，感觉落人一截，会被瞧不起，就像那句歌词：生活不止眼前的苟且，还有诗和远方的白眼。（不好意思，忍不住改编了下。）

开始进阶

于是我开始寻求进步的方法，开始研究各个工具，postman、fiddle、jmeter、selenium、appium，反正网上能下的，都下了个遍，发现这些工具，用起来还都挺简单实用的，入门肯定是不在话下，同时也开始自己写代码，至少别把大学学的那些给落下，当时 java 试着写了写（所在团队的项目用 java 写的），前端的样式也试着改了改，这个很容易，浏览器 F12，就可以在上面改。以前提 bug 只会说这个盒子的高度不对，现在我会说：overflow 没有写、盒子层级 z-index 不对等等，总之就是，我能指出你的不足，我还能提供修改建议，这对我来说是十分有成就感的，从开发哑口无言的态度也能看的出来，我的地位一下子崇高了起来！

当然我没有急着跳槽，我开始用各种工具对我们的项目做测试，还给部门经理提了很多建议，当时经理也应该是对测试不甚了解，被我说得也是一愣一愣的，以至于后来我提出离职的时候，他毅然决然地留住了我，给我的工资翻了个倍。

目前现状

是的，我还是在这家公司。

三年的时间，公司也从几十人发展成了几百号人，技术人才跳来跳去很常见，大多留不久，看多了来来去去的人，我竟然都成了老员工了。现在也是有两名手下的测试小组长啦！

经理听了我之前的建议，开始重视软件测试，这有好有坏，好的是地位高了，工资涨了，软件也更趋于完美了，不好的是责任更大了，任务也重了，尤其是发版前也要跟着熬夜了。

万把块的工资在测势界可能也不算高吧，但我自己已经很知足了，主要还是看个人的选择吧，这里有我熟悉的同事，有我熟悉的业务，如果可以的话，我希望能一直呆下去与公司共成长。不由得感叹一句，我可真是个长情的人呐！

职场的建议

无论什么工作，第一重要的肯定是扎实的技术及理论基础，自己有过硬的本事，无论到哪儿都能过得风生水起。

第二重要的就是个人业务能力了，或者说沟通能力，拥有良好的人际关系就等于拥有一个良好的工作氛围，基本上所有的工作就是团队作业，避免不了人与人相处，如果没有一个良好的氛围，那待着多难受，氛围好，工作效率也会提高不少，就像网上说的：如何区别一个团队是好是坏，好的团队就是领导和下属都觉得对方都很厉害，不好的团队就是领导和下属都觉得对方是 SB。

第三重要的是领悟能力，谁都喜欢举一反三的人，反之，没人会乐于和一个要说很多遍才能听懂话的人一起共事，除非你俩真是关系好得出奇。领悟能力好的人能顺利了解整个团队项目的业务逻辑，甚至运作方式，哪怕团队大换血，也不用怕自己或者新团队理不清业务逻辑线，这样的人往往是公司需要把握住的人才。

以上是我的个人经验~希望各位看官赏脸给个好评，哈哈，下回再见。

一文读懂契约测试

◆ 作者： 咖啡猫

前言：最近，公司正在推动整个质量体系的改革工作，其中一项改革议题就是新增契约测试这一专项测试内容。为了搞懂契约测试的来龙去脉进而可以用相应的工具将契约测试落地，笔者很是花了一番工夫。因而想动笔将这一探索的结果记录下来，分享给更多的小伙伴们。

一、契约测试的由来

提起契约测试，就不得不说近两三年大行其道的微服务。“微服务”是一种架构概念，他的目的是通过将功能分解到各个离散的服务中以实现对解决方案的解耦，从而降低系统的耦合性，并提供更加灵活的服务支持。

什么？听不懂.....

好吧，来张图直观一点：

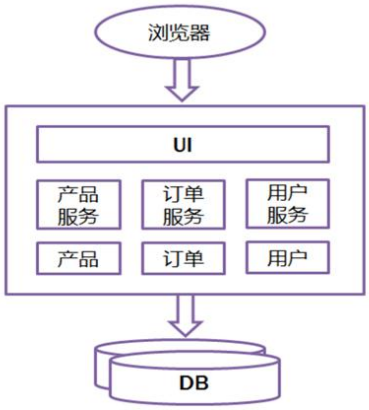


图 1　单体式开发

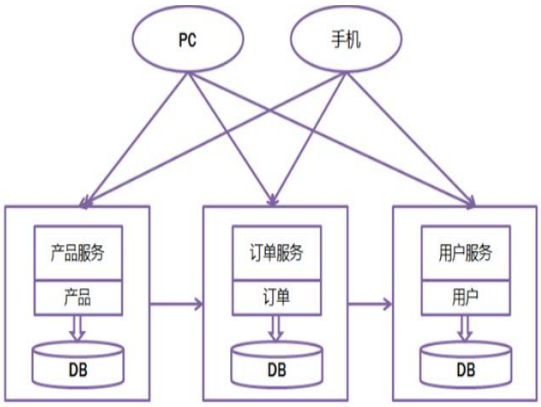


图 2　基于微服务架构的开发

图 1 是传统的 WEB 开发模式--单体式开发，它把所有的功能都打包在一个 WAR 包里，然后部署在一个诸如 TOMCAT 之类的容器内，它基本上没有任何外部依赖，便于集中式管理。但是它也有自己的缺点：比如开发效率低，因为各个模块之间可能相互等待；比如维护难，因为所有功能代码都耦合在一起，不方便 BUG 的定位和代码调试；又比如稳定性不好，一个微小的 BUG 都可能牵一发而动全身。

图 2 就说基于微服务架构的开发模式，它把一个复杂的大型系统拆分成一个个独立的服务，每一个微服务看成是用一组 API 提供业务功能的组件，每个微服务由一个团队来开发维护，各个微服务之间使用 HTTP API 进行通讯，服务可以采用不同的编程语言与数据库。这样的架构不仅可以有效提高团队的整体工作效率，还可以通过这种松耦合的关系（不同服务的代码分离开来），方便 BUG 的定位和代码的调试，提高系统的复用性和稳定性。

二、契约测试的定义

接下来的一个问题就是：为什么采用微服务就需要契约测试呢？

事实就是：当把一个复杂系统拆解成一个个微服务后，会出现服务之间的相互调用关系变得错综复杂。比如同一个微服务 A 同时被微服务 B、微服务 C、微服务 D、微服务 E.....所调用。当作为服务提供者的微服务 A 发生改变时，怎么保证这种更改对其它所有使用者造成的影响都被感知到了呢？这时契约测试就派上用场了。

契约测试 ，又称之为 消费者驱动的契约测试(Consumer-Driven Contracts，简称 CDC)，根据 消费者驱动契约 ，我们可以将服务分为消费者端和生产者端，而消费者驱动的契约测试的核心思想在于是从消费者业务实现的角度出发，由消费者自己会定义需要的数据格式以及交互细节，并驱动生成一份契约文件。然后生产者根据契约文件来实现自己的逻辑，并在持续集成环境中持续验证。

以消费者 B 为例，它制定与生产者 A 之间数据交互的协议、格式等方面的内容，并生成契约文件。此时 A 按照契约文件来提供服务就好。这样 A、B 都按契约行事，只关注自己的部分就可以达成整个系统对它们的要求了。

三、契约测试的特点

说起契约测试的特点，首先要介绍下它与单元测试、API 测试、集成测试的区别。

• 单元测试：对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如 C 语言中单元指一个函数，Java 里单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。

• API 测试：对软件中的业务接口进行测试，着重关注接口的请求格式、调用参数、接口返回的内容、接口的性能、接口的安全性等待。比如：测试接口的功能是否按接口文档中定义的那样完全实现了，对异常请求有没有合适的处理方式。接口最大并发数是否满足业务需求。接口的是否有安全漏洞等等。

• 契约测试：为了测试服务之间连接调用的正确性，为了验证服务提供者的功能是不是真正能够满足消费者的需求。它其实体现了测试前移的思想，把本来要通过集成测试才能验证的工作化作单元测试和接口测试，用更轻量的方式快速进行验证。

• 集成测试：站在用户的角度来验证整个系统功能的正确性，它测试的对象是端到端的业务流程。并融入用户故事和真实数据，很显然它的业务价值是最高的。

总之，契约测试体现的是一种测试前移的思想，它测试的对象是服务之间链路的正确性，是对传统单元测试、API 测试、集成测试的有力补充。

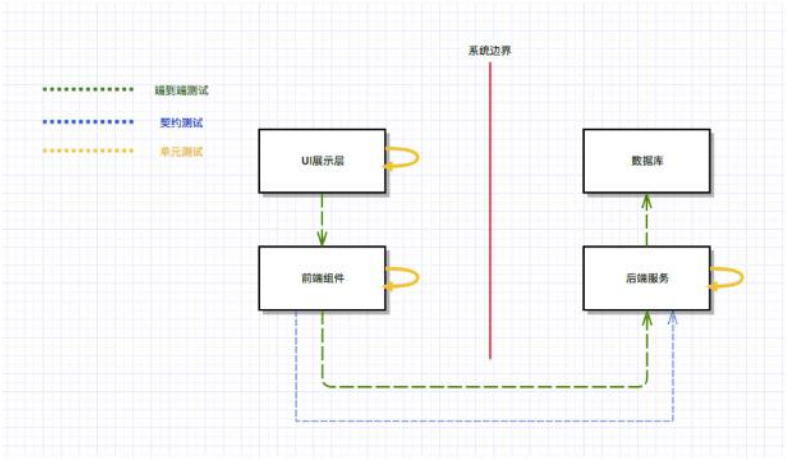


图 3 单元测试、契约测试、端到端测试的区别与联系

四、契约测试的作用

• 可以使得生产端和消费端之间测试解耦，不再需要等到系统联调才能发现问题。尤其再某些大型互联网项目中，联调可能会涉及不同国家和地区的多个团队进行，相当耗时耗力，如果能通过契约测试提前发现问题，可以减少很多不必要的沟通成本。

• 完全由消费者驱动的方式，消费者需要什么数据，服务端就给什么样的数据，数据契约也是由消费者来定的。这说明契约测试更加注重业务需求和消费端体验，以业务为中心促进质量的全面提升。

• 测试前移，越早的发现问题，保证后续测试的完整性。在软件测试行业有这么一条公认规则：BUG 发现的越早，其修复的代价越低。契约测试让测试迁移，就降低了 BUG 的修复成本，保证后续测试的完整性。

• 通过契约测试，团队能以一种离线的方式(不需要消费者、提供者同时在线)，通过契约作为中间的标准，验证提供者提供的内容是否满足消费者的期望。

五、契约测试的实战工具--Pact

Pact 是一个开源框架，最早是由澳洲最大的房地产信息提供商 REA Group 的开发者及咨询师们共同创造。Pact 工具于 2013 年开始开源，发展到今天已然形成了一个小的生态圈，包括各种语言（Ruby/Java/.NET/JavaScript/Go/Scala/Groovy...）下的 Pact 实现，契

约文件共享工具 Pact Broker 等。Pact 的用户已经遍及包括 RedHat、IBM、Accenture 等在内的若干知名公司，Pact 已经是事实上的契约测试方面的业界标准。

Pact 的基本流程：

Step1. 编写测试用例，生成契约文件。在消费者端 Consumer 写一个对接口发送请求的单元测试，在运行这个单元测试的时候，Pact 会将服务提供者自动用一个 MockService 代替，并自动生成契约文件，这个契约文件是 Json 形式的。

Step2. 利用 pact-verifier 命令和契约文件，验证接口提供者是否正确?在服务提供端 Provider 做契约验证测试，将 Provider 服务启动起来以后，通过 pact 插件可以运行一个命令，比如你是用 maven，就是 mvn pact:verify，它会自动按照契约生成接口请求并验证接口响应是否满足契约中的预期。

因此，我们可以看到在这个过程中，在消费者端不用启动 Provider，在服务提供端不用启动 Consumer，却完成了与集成测试类似的验证。

下面给一个 DEMO 方便大家理解这一过程：

1.编写服务提供者 A 的代码：

```
import json
from flask import Flask
test_app = Flask(__name__)
@test_app.route('/')
def get_information():
    information = {
        "size": "M",
        "color": "red"
    }
    return json.dumps(information)
if __name__ == '__main__':
    test_app.run(port=8088)
```

2.编写服务消费者 B 的代码：

```
import json
import requests
from flask import Flask
test_app = Flask(__name__)
@test_app.route('/')
def show_information():
    res = requests.get("http://localhost:8088/").json()
    result = {
        "code":0,
        "msg":"ok",
        "data": res
    }
    return json.dumps(result)
if __name__ == '__main__':
    test_app.run(port=8089)
```

3.编写测试用例的代码

```
import atexit
import requests
import unittest
from pact.consumer import Consumer
from pact.provider import Provider
# 定义一个 pact，消费者是 B，生产者是 A，契约文件存放在 pacts 文件夹下
pact = Consumer('B').has_pact_with(Provider('A'), pact_dir='./pacts')
# 启动 pact 服务
pact.start_service()
atexit.register(pact.stop_service)
# 测试用例
class UserTesting(unittest.TestCase):
    def test_service(self):
        # 消费者定义的期望结果
```

```
            expected = {"size": "M", "color": "red"}
            # 消费者定义的契约的实际内容。包括请求参数、请求方法、请求头、响应值等
            (pact
              .given('test service.')
              .upon_receiving('a request for B')
              .with_request('get', '/')
              .will_respond_with(200, body=expected))
            # pact 自带一个 mock 服务，端口 1234
            # 用 requests 向 mock 接口发送请求，验证 mock 的结果是否正确
            with pact:
                res = requests.get("http://localhost:1234").json()
            self.assertEqual(res, expected)
if __name__ == "__main__":
    Utest = UserTesting()
    Utest.test_service()
```

4.生成实际的契约文件 b-a.json

```
{
  "consumer": {
    "name": "B"
  },
  "provider": {
    "name": "A"
  },
  "interactions": [
    {
      "description": "a request for B",
      "providerState": "test service.",
      "request": {
        "method": "get",
        "path": "/"
      },
```

```
        "response": {
          "status": 200,
          "headers": {
          },
          "body": {
            "size": "M",
            "color": "red"
          }
        }
      }
    ],
     "metadata": {
      "pactSpecification": {
        "version": "1.0"
      }
    }}
```


5、根据契约文件，验证服务 A 是否正确

a. 启动服务 A(提供者)。

b. 执行 pact-verifier --provider-base-url=http://127.0.0.1:8088 --pact-url=./pacts/b-a.json，即可验证接口是否符合契约。

后记：本文从契约测试的由来、契约测试的定义、契约测试的特点、契约测试的作用和契约测试的实战工具 Pact 等五个方面全方位多维度的阐述了契约测试，既有理论原理的描述，也有具体的代码实现，希望能给大家带来一点启发！

双剑合并-拨开内存溢出的真相

◆作者：云竹

一、问题描述：

对于 B/S 架构的 J2EE 应用系统，应用服务器发生内存溢出(OutOfMemoryError)错误是非常棘手的问题，解决问题的关键是要定位哪里出了问题。一般内存溢出的原因有以下几类:

• Jvm 内存参数设置不合理，如-Xmx 设置的比较小，-Xms 设置的过大，sun 的 jvm 没有设置-XX:MaxPermSize 大小等等，由于参数设置问题导致的内存溢出往往容易发现，调整一下参数，问题基本就可以解决。

• 应用服务器缺少必要的补丁，例如 websphere 的某些版本，存在很多性能缺陷，某些情况下也会导致内存溢出，打上补丁后，问题基本可以得到解决。

• 应用程序代码存在缺陷，此类问题是内存溢出最常见的原因，解决起来也最麻烦，本文主要描述这类内存溢出问题如何进行分析和定位。

二、解决方案

工欲善其事，必先利其器，有了好的工具，解决问题往往能起到事半功倍的作用，很多时候没有工具支持，问题很难解决。

本文主要涉及到 2 个工具的运用：

• 分析 javacore 的工具：IBM Thread and Monitor Dump Analyzer for Java

• 分析 heapdump 的工具：Eclipse Memory Analyzer

通过这 2 个工具，一般的内存溢出问题基本可以定位。

在介绍这 2 个工具之前，先对一些相关概念做下解释说明：

• Heapdump，某一时刻 Java 进程内存的快照，记录当时内存中各对象的存储和引用关系等。Heapdump 中不包含对象的调用信息，即无法从 heapdump 中看出某个对象是哪段程序代码生成出来的。

•Javacore，某一时刻 Java 进程中线程的快照，记录当时所有线程的执行堆栈信息等。Javacore 中可以看到各线程都在处理哪些程序代码，可以看到当前 Java 都在“做什么事”，在线程的执行堆栈中，有代码的调用信息，也有对象生成的相关信息等。

• 内存溢出一般分两种：内存使用过度和内存泄露。使用过度指在短时间内急剧耗尽大量内存导致内存溢出；内存泄露指长时间内，内存逐渐耗尽导致内存溢出，时间可能是几个小时或几天(通过开启详细垃圾回收可以清晰地观察到内存的分配情况)。JAVA 系统的内存溢出大部分都是由于内存使用过度导致，内存泄露比较少见。本文后面介绍的也都是关于内存使用过度导致内存溢出的分析方法。

• 对于内存使用过度导致的内存溢出，可以通过 heapdump 和 javacore 结合分析定位问题，对于内存泄露，主要还是分析 heapdump。

当 JAVA 进程出现内存溢出错误时，通常会生成 heapdump 文件和 javacore 文件，例如 Websphere 部署的 web 应用，当出现内存溢出时，会在如下目录——

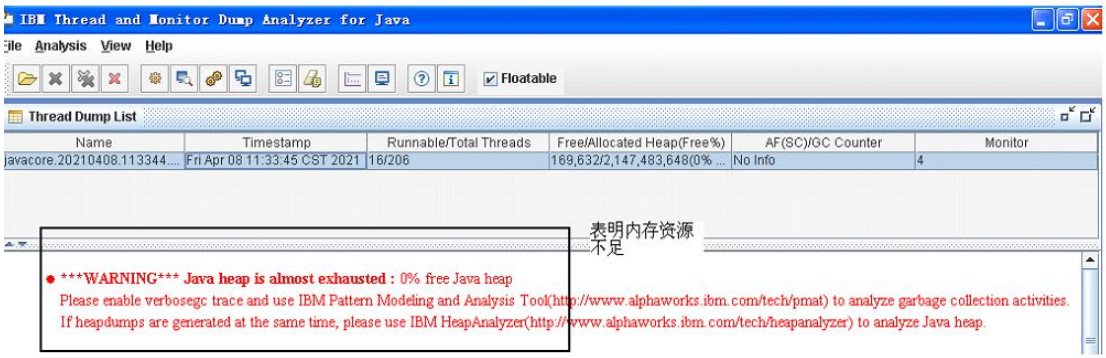
/opt/IBM/WebSphere/AppServer/profiles/server1 看到出现类似这样的文件：

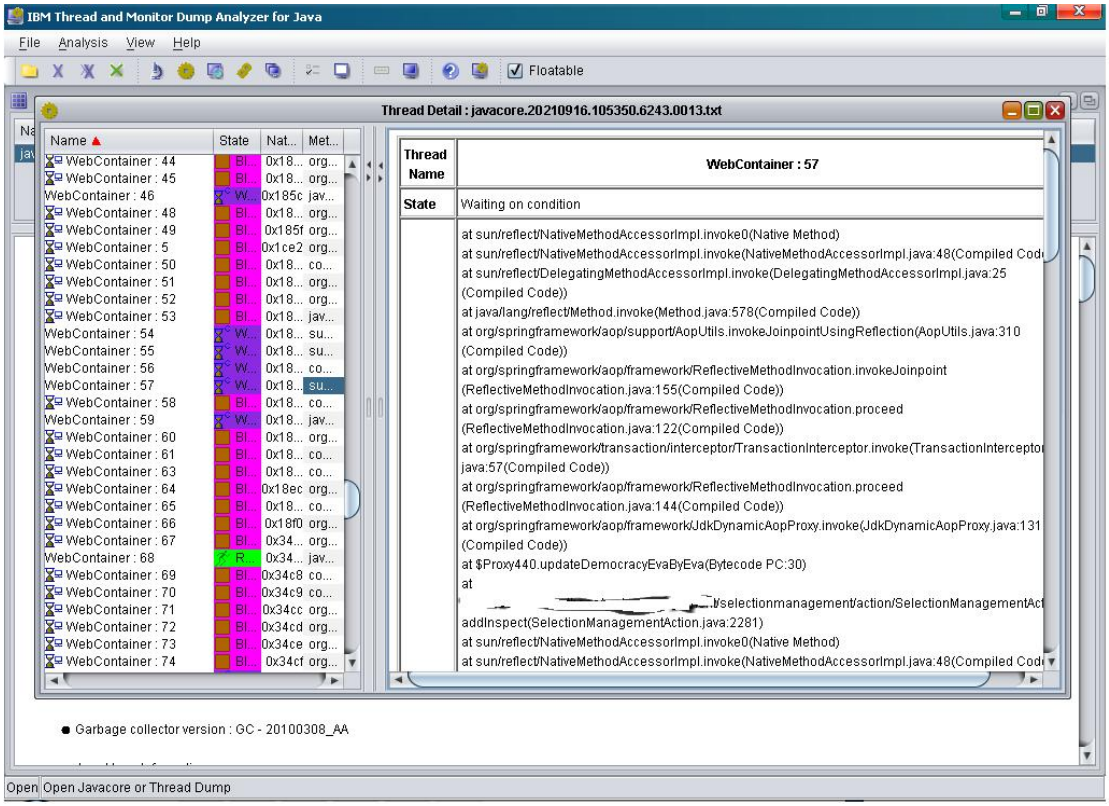
Heapdump.*.phd 文件记录 heapdump 信息，javacore.*.txt 记录 javacore 信息，一般内存溢出时会出现多个 phd 文件和 javacore 文件,分析时用其中一个即可。

下面就介绍如何通过工具分析这 2 个文件，如何通过对这 2 个文件的分析定位出内存溢出(内存使用过度导致)的原因。

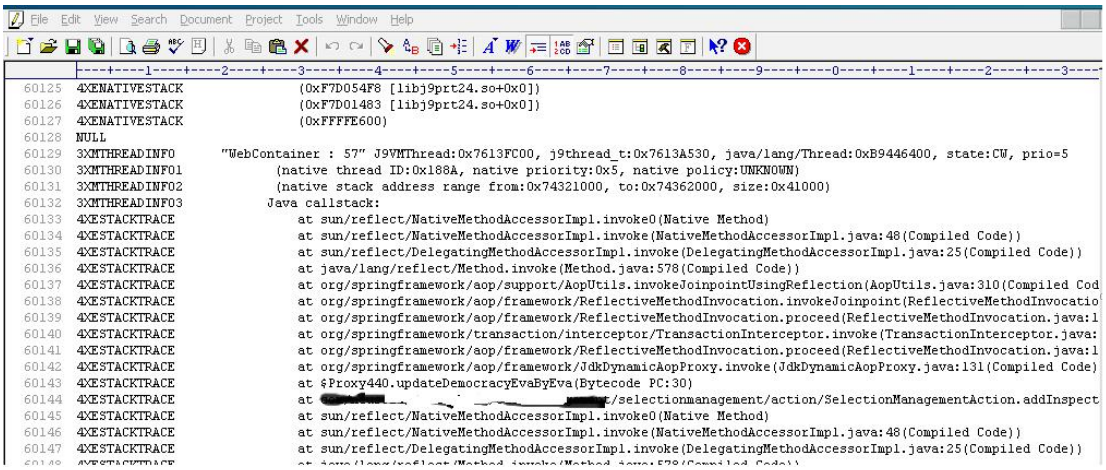
1．分析 javacore

Javacore 的分析工具 IBM Thread and Monitor Dump Analyzer for Java(简称 JCA)，以图形化的方式展现 java 线程的堆栈信息及相互关系等。下面是工具使用的截图：

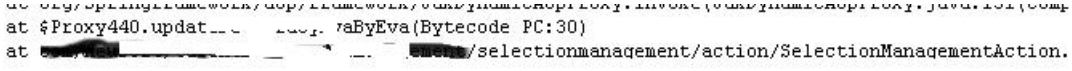




对于 Websphere，主要关注 WebContainer 线程的执行堆栈情况，上图显示了 WebContainer:57 号线程的执行堆栈。Javacore 文件本身就是个文本文件，直接通过文本编辑器如 editplus 也可以打开，下面是用文本编辑器打开 javacore 文件后看到的 WebContainer:57 号线程的执行堆栈：



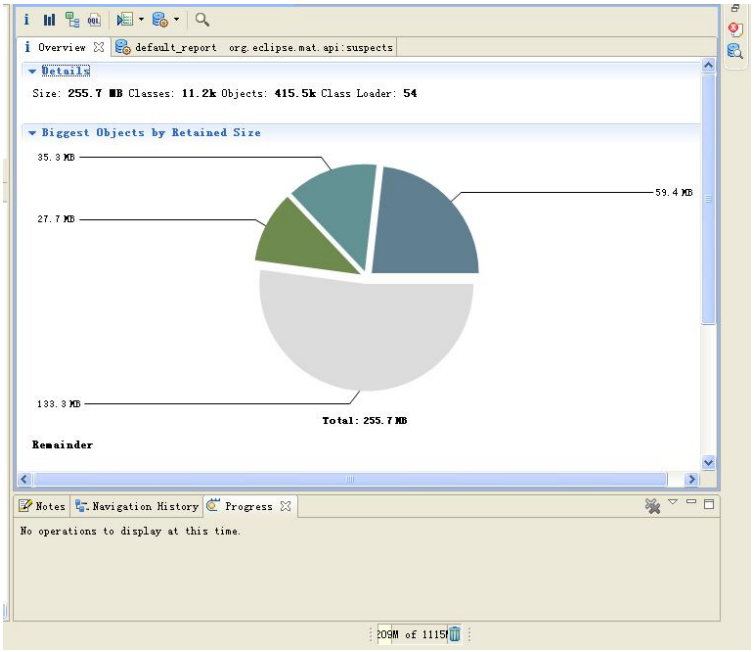
可以看到，通过 JCA 打开 javacore 和通过 editplus 打开 javacore，没有本质的区别，只是展现形式有所区别而已，有关 JCA 工具的具体使用这里不作详细介绍。分析 javacore 的重点是根据每个线程的堆栈信息找出哪个或哪几个线程导致 java 进程的内存溢出，对应线程堆栈中的代码段就是罪魁祸首。例如，如果通过分析得到 WebContainer:57 号线程可能导致内存溢出，那么 57 号线程中的这段代码就是问题所在。

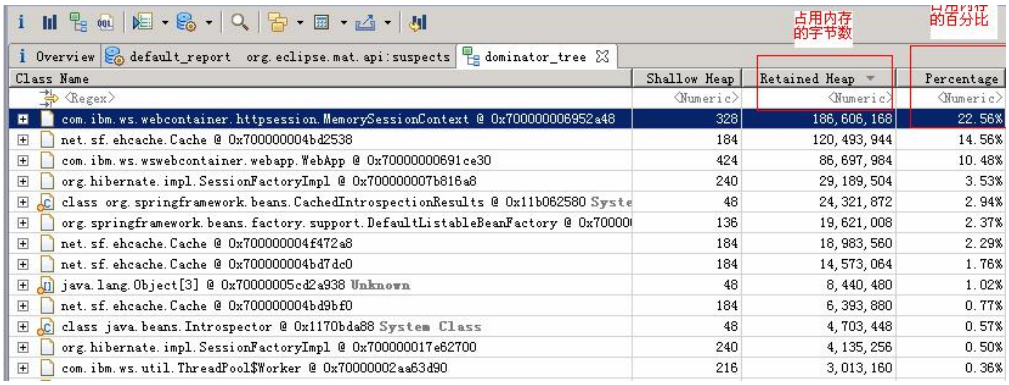


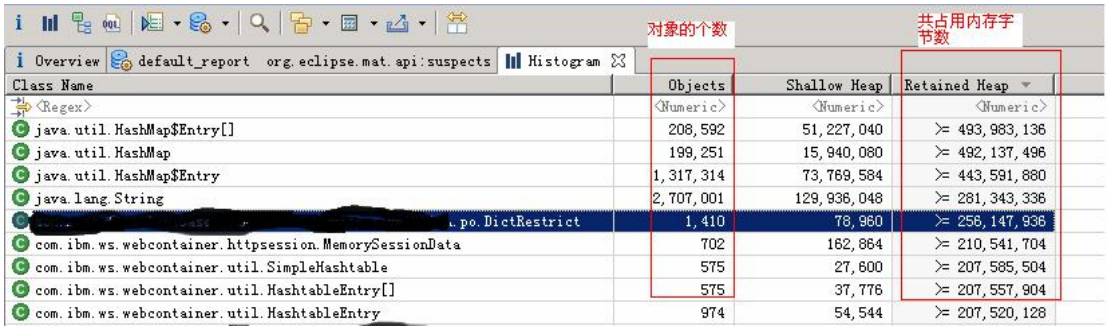
为什么是 WebContainer:57 号线程有问题，而不是其它线程呢？这就需要结合 heapdump 进行综合分析，下面会有详细介绍。

2．分析 Heapdump

Heapdump 的分析工具推荐 Eclipse Memory Analyzer(简称 MAT)，该工具比 IBM 的 HA(Heap Analyzer)工具更强大，更易使用。下面是工具的使用截图：







该工具可以看到对象占用内存的大小、对象的个数等信息，通过分析占用内存比例最大的对象，可以初步判断是哪个对象导致的内存溢出，有关 MAT 工具的具体使用这里不作详细介绍。heapdump 分析的重点是找到占用内存最大的“业务对象”，所谓业务对象，就是和应用业务相关的对象，需要能靠到应用程序上。例如上图所示，虽然 java.lang.String 比 DicRestrict 对象占用的内存要多，但是我们做分析时应该重点关注

DicRestrict 对象，因为 String 对象是通用的业务无关对象，可能很多业务对象都会引用它。Heapdump 分析需要结合业务知识，需要相当的开发和业务经验。

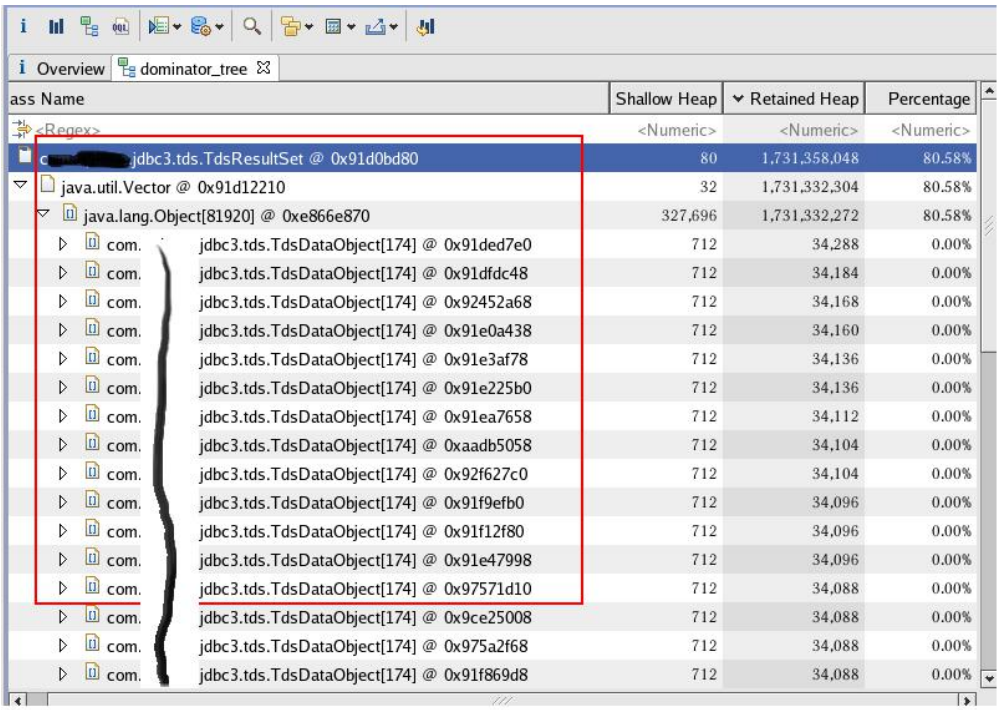
3．双剑合并

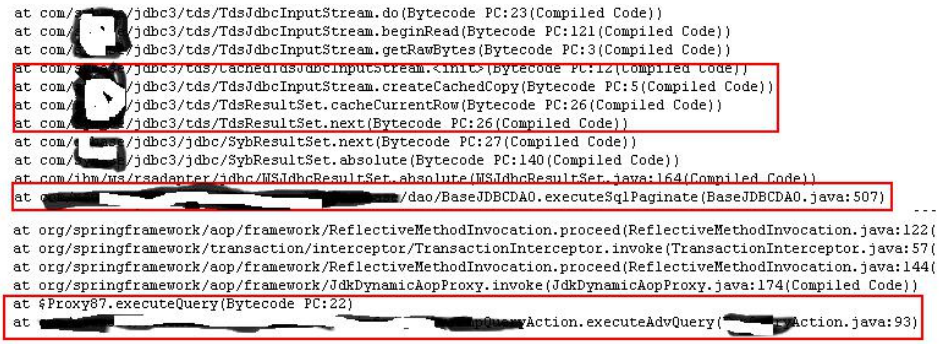
Javacore 和 heapdump 的单独分析，都只是看到了事情的一面，通过一定方法将二者有机的结合起来，问题基本就可以水落石出了。

关联方法：

• 通过对象名称关联

在 heapdump 中占用内存多的对象的名称，可能会出现在 javacore 的线程堆栈中。如下面图示：

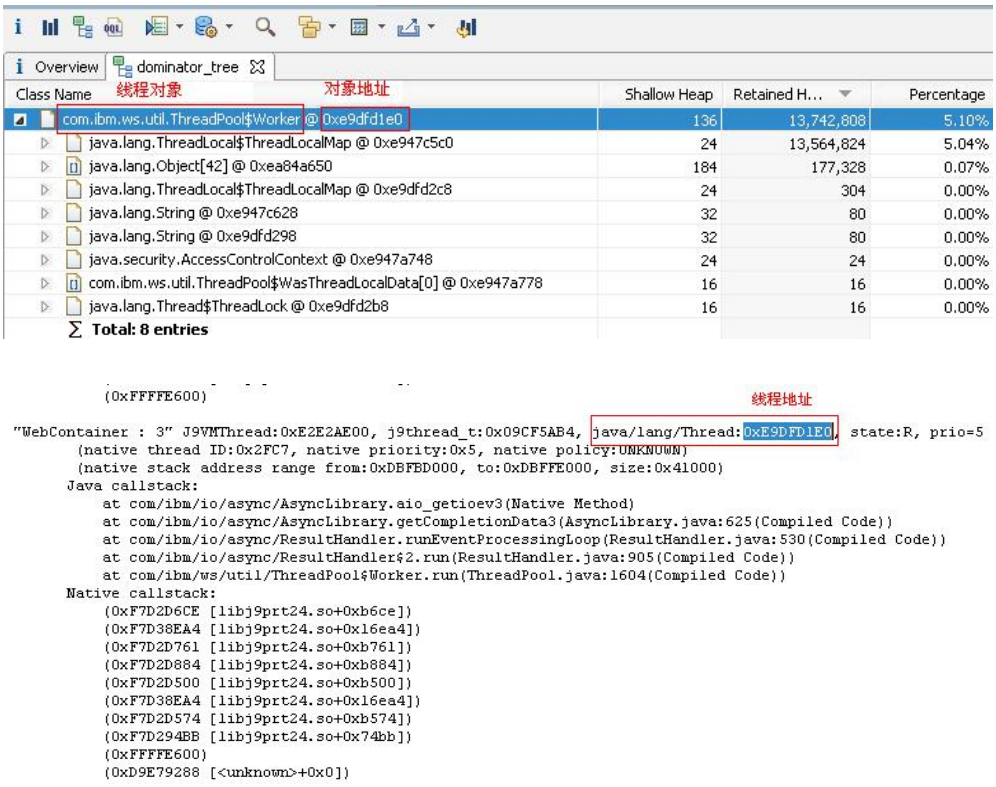




heapdump 中显示 TdsResultSet 占用 80%以上的内存，javacore 中的线程堆栈显示 executeAdvQury 这个业务代码产生了 TdsResultSet 对象，内存和线程两方面的分析相吻合，所以问题精确定位。大部分情况下通过对象名称关联可以找到问题所在。

• 通过线程地址关联

Heapdump 中有一类特殊的对象，即线程对象，线程对象的地址可以关联到 javacore 中的线程地址上。如下图所示：



上面图中显示：Heapdump 中可以看到地址是 0xe9dfd1e0 的线程对象的内存占用信息，javacore 中可以看到地址是 0xe9dfd1e0 的线程的调用执行信息。当 heapdump 中发现有线程对象占用大量内存时，可以通过地址关联方法找到 javacore 中对应执行的程序代码堆栈，从而定位问题所在。地址关联用的比较少，因为很少会出现线程对象占用大量内存的情况，当线程出现死循环或者循环次数过多时可能会出现这种情况。

4．问题定位后的代码检查和测试

通过 heapdump 和 javacore 的综合分析，定位出具体程序代码，接下来就需要开发人员进行进一步的检查和分析，以找出程序的问题，必要时还需要做大量的测试以重现和验证问题。

三、问题结论

本文简要介绍了利用 Jca 和 Mat 工具进行内存溢出分析的方法，对如何关联 javacore 和 heapdump 做了较详细的介绍。内存溢出的分析是一项比较复杂的工作，有些时候准确定位问题是非常困难的，本文介绍的方法适合于大部分情况，有些时候 javacore 和 heapdump 无法完全关联上，此时就需要分析人员具备丰富的经验和足够的能力，以及合理的猜想和推测，这就是内功的修炼了。