验证码的处理

在模拟登录中，其实让写爬虫的人疼头就是验证码，只要能破掉验证码，那么登录不是问题.

验证码(Chaptcha)内容从英文字符和数字识别，到数字加减乘除，再到汉字的出现，后面还有 12306 的看图识别，到现在的新型的基于人的行为的谷歌的 reCaptcha，验证码也是经历了很长时间的演变。

思路一：Cookie 登录

当我们在登录的时候遇到了验证码,这时候我们需要人工识别之后才能登录上去,其实这是个非常繁琐的过程，每次登录都要我们手动输入验证码,很不可取,但是大部分的网站当你登录上去之后，**cooke 都会保持较长的一段时间,避免因用户频繁输入账号和密码造成的不便.**我们可以利用这个特性,当我们登录成功一次之后,可以**将 cooke 信息保存到本地，下次登录时直接使用 cooke 登录**

以人人登录为例:

```
# 获取一个有登录信息的 Cookie 模拟登陆
from urllib import request
import chardet

# 1. 构建一个已经登录过的用户的 headers 信息
headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36",
    # 重点：这个 Cookie 是保存了密码无需重复登录的用户的 Cookie，这个 Cookie 里记录了用户名，密码(通常经过 RAS 加密)
    "Cookie": "anonymid=jhsxb2breoia7; _r01_=1; wp=1; ln_hurl=http://head.xiaonei.com/photos/0/0/men_main.gif; depovince=HEN; ln_uact=17752558702; JSESSIONID=abc5Lla996PpxaUC5ELsw; ick_login=9e35fd47-b667-4ea2-ade3-4c6b858f8e5d; jebe_key=c48f502e-c905-49aa-b97b-50aea0dcf6aa%7C077a3e2b1c00096d5c13732ceee74ce5%
```

```
7C1531554565013%7C1%7C1531784703509; first_login_flag=1; wp_fold=0;
jebecookies=f5550ece-d9d5-4ae9-aa4b-3f339c0d1fb4|||||;
_de=32B20555AD3784A6BF2D3D01B72FE013; p=6539369d0cd9183609010b746f65847a2;
t=81991ae9dbb47bfbdbc4acd3ef8248d02;
societyguester=81991ae9dbb47bfbdbc4acd3ef8248d02; id=966924492; xnsid=5a67c8aa;
loginfrom=syshome",
}

# 2. 通过 headers 里的报头信息（主要是 Cookie 信息），构建 Request 对象
https://www.baidu.com/
req = request.Request("http://www.renren.com/966924492/", headers = headers)

# 3. 直接访问 renren 主页，服务器会根据 headers 报头信息（主要是 Cookie 信息），判断这是一个已经登录的用户，并返回相应的页面
response = request.urlopen(req)
# 4. 打印响应内容
html = response.read()
charset = chardet.detect(html)['encoding']
print(charset)
print(html.decode(charset))
```

思路二：传统验证码识别

为什么限定为传统的验证码呢？传统的验证码即传统的输入型验证码，可以是数字、字母和汉字这类验证码不涉及验证码含义的分析,仅仅识别验证码的内容,识别相对简单,进行验证码识别需要使用到 tesseract

ORC 库-Tesseract

文字识别是器视觉的一个分支，将图像翻译成文字一般被称为光学文字识别(Optical Character Recognition, OCR)

Tesseract 是 Python 进行图像处理的库，该引擎最初由惠普公司开发，目前由 Google 主导。

安装

Windows 系统

下载可执行安装文件 https://code.google.com/p/tesseract-ocr/downloads/list 安装。

Linux 系统

可以通过 apt-get 安装: `$sudo apt-get tesseract-ocr`

Mac OS X 系统

用 Homebrew

(http://brew.sh/)等第三方库可以很方便地安装

brew install tesseract

添加环境变量 `TESSDATA_PREFIX`，值：C:\Program Files (x86)\Tesseract-OCR

- 在大多数 Linux 系统和 Mac OS X 系统上,你可以这么设置: `$export TESSDATA_PREFIX=/usr/local/share/Tesseract`

  让 Tesseract 知道训练的数据文件存储的位置。Tesseract 也有自己的已经做好的语言训练集，下载地址：

  https://github.com/tesseract-ocr/tessdata/archive/master.zip

- **安装 pytesseract**

  安装支持 Python 版本的 Tesseract 库：

  ```
  pip install pytesseract
  ```

处理给规范的文字

格式规范的文字具有以下特点:

- 使用一个标准字体(不包含手写体、草书,或者十分“花哨的”字体)• 虽然被复印或拍照,字体还是很清晰,没有多余的痕迹或污点
- 排列整齐,没有歪歪斜斜的字
- 没有超出图片范围,也没有残缺不全,或紧紧贴在图片的边缘

命令运行 Tesseract，读取文件并把结果写到一个文本文件中: `

tesseract test.jpg text

通过 Python 代码实现

```
import pytesseract
from PIL import Image

image = Image.open('./images/tesseracttest.jpg')
text = pytesseract.image_to_string(image)
print(text)
```

运行结果:

```
This is some text, written in Arial, that will be read by
Tesseract. Here are some symbols: !@#$%"&*()
```

常见验证码处理

import pytesseract
from PIL import Image
img =Image.open('./images/recaptcha.png')
img.show()
print(pytesseract.image_to_string(img))

上面的代码在执行后，会返回 - 个空字符串，也就是说 Tesseract 在抽取输入图像中的字符时失败了。这是因为 Tesseract 的设计初衷是抽取更加典型的文本，比如背景统一的图片。

要想更加有效地使用 Tesseract，需要先修改验证码图像，**去除其中的背景噪音，只保留文本部分。**

```
import pytesseract
from PIL import Image

img =Image.open('./images/recaptcha.png')
img.show()
#可以看出，验证码文本一般都是黑色的，背景则会更加明亮，所以我们可以通过检查像素是否为黑色将文本分离出来，该处理过程又被称为阈值化。通过 Pillow 可以很容易地实现该处理过程。
gray = img.convert('L') #灰度化
gray.show()
bw = gray.point(lambda x: 0 if x < 1 else 255,'1')
bw.show()
print(pytesseract.image_to_string(bw))
```

分析：可以看出，验证码文本一般都是黑色的，背景则会更加明亮，所以我们可以通过检查像素是否为黑色将文本分离出来，该处理过程又被称为阈值化。通过 Pillow 可以很容易地实现该处理过程。

进一步改善

· 实验不同阈值

· 腐蚀阈值文本，突出字符形状

· 调整图像大小（有时增大只寸会起到作用）：

· 根据验证码字体训练 OCR 工具：

· 限制结果为字典单词

处理复杂验证码

前面用于测试的验证码系统相对来说比较容易处理，因为文本使用的黑色字体与背景很容易区分，而且文本是水平的，无须旋转就能被 Tesseract 准确解析。一般情况下，网站使用的都是类似这种比较简单的通用验证码系统，此时可以使用 OCR 方法。但是，如果网站使用的是更加复杂的系统，比如 Google 的 reCAPTCHA，OCR 方法则需要花费更多努力，甚至可能无法使用

网页中验证码图片的获取

模拟浏览器，selenium 和 PIL 库的截图功能

```
from selenium import webdriver
from PIL import Image

browser = webdriver.Chrome()
browser.get('https://www.baidu.com')
browser.save_screenshot('./images/baidu.png')
element=browser.find_element_by_xpath('//div[@id="lg"]/img[1]')
#location 办法可能会有偏移，但是每次都会锁定了了验证码的位置，所以稍微修正一下 location 的定位，后面都管用
left   = element.location['x']#验证码图片左上角横坐标
top    = element.location['y']#验证码图片左上角纵坐标
right  = left + element.size['width']#验证码图片右下角横坐标
bottom = top + element.size['height']#验证码图片右下角纵坐标

#rangle=(806,382,913,415)#手工方法有时候可以，但是有时候也会出现第一次可以，但是后面各种偏移定位不准的奇葩现象
im=Image.open('./images/baidu.png')
im_crop=im.crop((left,top,right,bottom))#这个 im_crop 就是从整个页面截图中再截出来的验证码的图片
im_crop.save('./images/logo.png')
browser.quit()
```

思路三：打码平台

当传统验证码识别难度加大,识别程序很难保证较高的准确率,例如： 

,验证码粘连组曲非常严重,识别起来比较困难,这时候人工打码就产生了

人工打码采用自动识别+人工识别的组合方式

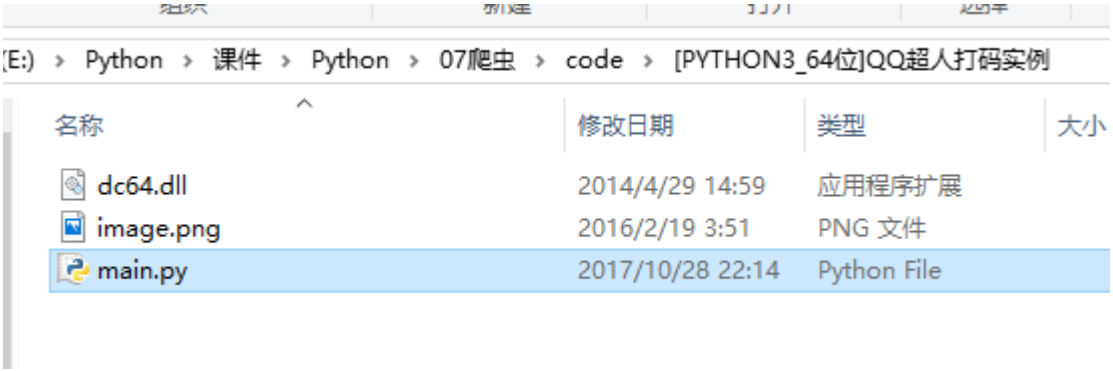
主要人工打码的平台有**打码兔和 QQ 超人打码**等

都提供了各种编程语言的接入方式,包括 Python,当然人工打码是需要收费的

以 QQ 超人打码为例,首先要去注册开发者账号,在识别程序中需要填写个人帐号进行认证计费,如图

http://www.chaorendama.com/reg.aspx

注册完成后,官方提供了各种编程语言接人方式的示例,其中就有 Pyfhon 的，可以根据提供的 API 示例，开发自己的识别程序



main.py 源码分析：

```
import sys
#QQ 超人打码支持类库
import ctypes
from os.path import join, dirname, abspath, exists
dll = ctypes.windll.LoadLibrary(join(dirname(__file__),'dc64.dll'))
dll.GetUserInfo.restype = ctypes.c_uint64
```

```
dll.RecYZM_A.restype = ctypes.c_uint64
dll.RecByte_A.restype = ctypes.c_uint64

class dcVerCode:
    #user QQ 超人打码账号
    #pwd QQ 超人打码密码
    #softId 软件 ID 缺省为 0,作者务必提交 softId,已保证分成
    def __init__(self,user,pwd,softId="0"):
        self.user = user.encode(encoding="utf-8")
        self.pwd = pwd.encode(encoding="utf-8")
        self.softId = softId.encode(encoding="utf-8")

    #获取账号剩余点数
    #成功返回剩余点数
    #返回"-1"----网络错误
    #返回"-5"----账户密码错误

    def getUserInfo(self):
        p = dll.GetUserInfo(self.user,self.pwd)
        if p:
                return ctypes.string_at(p,-1).decode()
        return ''

    #解析返回结果,成功返回(验证码,验证码 ID),失败返回错误信息
    #点数不足:Error:No Money!
    #账户密码错误:Error:No Reg!
    #上传失败，参数错误或者网络错误:Error:Put Fail!
    #识别超时:Error:TimeOut!
    #上传无效验证码:Error:empty picture!
    #账户或 IP 被冻结:Error:Account or Software Bind!
    #软件被冻结:Error:Software Frozen!
    def parseResult(self,result):
        list = result.split('|')
        if len(list)==3:
            return (list[0],list[2])
        return (result,'')

    #recByte 根据图片二进制数据识别验证码,返回验证码,验证码 ID
    #buffer 图片二进制数据

    def recByte(self,buffer):
        p = dll.RecByte_A(buffer,len(buffer),self.user,self.pwd,self.softId)
        if p:
            str = ctypes.string_at(p,-1).decode()
```

```
            return self.parseResult(str)
        return ''

    #recYZM 根据验证码路径识别,返回验证码,验证码 ID
    #path 图片路径
    def recYZM(self,path):
        p = dll.RecYZM_A(path.encode(encoding="utf-8"),self.user,self.pwd,self.softId)
        if p:
            str = ctypes.string_at(p,-1).decode()
            return self.parseResult(str)
        return ''

    #reportErr 提交识别错误验证码
    #imageId 验证码 ID
    def reportErr(self,imageId):
        dll.ReportError(self.user,imageId)

if __name__ == '__main__':
    client = dcVerCode('chaorenuser','chaorenpass','0'); #超人打码帐号,超人打码密码,软件 ID
    img = open('image.png','rb')
    buffer = img.read()
    img.close()

    #查询帐号余额
    print ('帐号余额:' + client.getUserInfo())

    #按图片字节数据识别
    yzm,imageId = client.recByte(buffer)
    print('识别结果：'+yzm,'验证码 ID：' + imageId)

    #按图片本地路径识别
    yzm,imageId = client.recYZM("image.png")
    print('识别结果：'+yzm,'验证码 ID：' + imageId)
    #client.reportErr(imageId) 只有在验证码识别错误时才运行这个方法,恶意提交将会受到惩罚
```

思路四：滑动验证码

滑动验证码是最近比较流行的验证方式,是一种基于行为的验证方式

通用的办法是使用 selenium 进行处理

1、在浏览器上模拟以鼠标拖动的操作

2、计算图片中缺口的偏移量 （用到了 PIL 库）

3、模拟人类拖动鼠标的轨迹

案例：python B 站 滑动验证码破解

```
import random
import time

from selenium.webdriver import ActionChains
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
from urllib.request import urlretrieve
from selenium import webdriver
from bs4 import BeautifulSoup
import PIL.Image as image
import re


class Crack():
    def __init__(self, username, passwd):
        self.url = 'https://passport.bilibili.com/login'
        self.browser = webdriver.Chrome()
        self.wait = WebDriverWait(self.browser, 100)
        self.BORDER = 6
        self.passwd = passwd
        self.username = username

    def open(self):
        """
        打开浏览器,并输入查询内容
        """
        self.browser.get(self.url)
        keyword = self.wait.until(EC.presence_of_element_located((By.ID, 'login-username')))
        keyword.send_keys('')
        keyword = self.wait.until(EC.presence_of_element_located((By.ID, 'login-passwd')))
        keyword.send_keys('')
        # bowton.click()
```

```
    def get_images(self, bg_filename='bg.jpg', fullbg_filename='fullbg.jpg'):
        """
        获取验证码图片
        :return: 图片的 location 信息
        """
        bg = []
        fullgb = []
        while bg == [] and fullgb == []:
            bf = BeautifulSoup(self.browser.page_source, 'lxml')
            bg = bf.find_all('div', class_='gt_cut_bg_slice')
            fullgb = bf.find_all('div', class_='gt_cut_fullbg_slice')
        bg_url = re.findall('url\(\"(.*)\"\);', bg[0].get('style'))[0].replace('webp', 'jpg')
        fullgb_url = re.findall('url\(\"(.*)\"\);', fullgb[0].get('style'))[0].replace('webp', 'jpg')
        bg_location_list = []
        fullbg_location_list = []
        for each_bg in bg:
            location = {}
            location['x'] = int(re.findall('background-position: (.*)px (.*)px;',
each_bg.get('style'))[0][0])
            location['y'] = int(re.findall('background-position: (.*)px (.*)px;',
each_bg.get('style'))[0][1])
            bg_location_list.append(location)
        for each_fullgb in fullgb:
            location = {}
            location['x'] = int(re.findall('background-position: (.*)px (.*)px;',
each_fullgb.get('style'))[0][0])
            location['y'] = int(re.findall('background-position: (.*)px (.*)px;',
each_fullgb.get('style'))[0][1])
            fullbg_location_list.append(location)

        urlretrieve(url=bg_url, filename=bg_filename)
        print('缺口图片下载完成')
        urlretrieve(url=fullgb_url, filename=fullbg_filename)
        print('背景图片下载完成')
        return bg_location_list, fullbg_location_list

    def get_merge_image(self, filename, location_list):
        """
        根据位置对图片进行合并还原
        :filename:图片
        :location_list:图片位置
        """
        im = image.open(filename)
        new_im = image.new('RGB', (260, 116))
```

```
        im_list_upper = []
        im_list_down = []

        for location in location_list:
            if location['y'] == -58:
                im_list_upper.append(im.crop((abs(location['x']), 58, abs(location['x']) + 10,
166)))
            if location['y'] == 0:
                im_list_down.append(im.crop((abs(location['x']), 0, abs(location['x']) + 10,
58)))

        new_im = image.new('RGB', (260, 116))

        x_offset = 0
        for im in im_list_upper:
            new_im.paste(im, (x_offset, 0))
            x_offset += im.size[0]

        x_offset = 0
        for im in im_list_down:
            new_im.paste(im, (x_offset, 58))
            x_offset += im.size[0]

        new_im.save(filename)

        return new_im

    def get_merge_image(self, filename, location_list):
        """
        根据位置对图片进行合并还原
        :filename:图片
        :location_list:图片位置
        """
        im = image.open(filename)
        new_im = image.new('RGB', (260, 116))
        im_list_upper = []
        im_list_down = []

        for location in location_list:
            if location['y'] == -58:
                im_list_upper.append(im.crop((abs(location['x']), 58, abs(location['x']) + 10,
166)))
            if location['y'] == 0:
                im_list_down.append(im.crop((abs(location['x']), 0, abs(location['x']) + 10,
```

```
58)))

        new_im = image.new('RGB', (260, 116))

        x_offset = 0
        for im in im_list_upper:
            new_im.paste(im, (x_offset, 0))
            x_offset += im.size[0]

        x_offset = 0
        for im in im_list_down:
            new_im.paste(im, (x_offset, 58))
            x_offset += im.size[0]

        new_im.save(filename)

        return new_im

    def is_pixel_equal(self, img1, img2, x, y):
        """
        判断两个像素是否相同
        :param image1: 图片 1
        :param image2: 图片 2
        :param x: 位置 x
        :param y: 位置 y
        :return: 像素是否相同
        """
        # 取两个图片的像素点
        pix1 = img1.load()[x, y]
        pix2 = img2.load()[x, y]
        threshold = 60
        if (abs(pix1[0] - pix2[0] < threshold) and abs(pix1[1] - pix2[1] < threshold) and abs(
                                pix1[2] - pix2[2] < threshold)):
            return True
        else:
            return False

    def get_gap(self, img1, img2):
        """
        获取缺口偏移量
        :param img1: 不带缺口图片
        :param img2: 带缺口图片
        :return:
        """
```

```
        left = 43
        for i in range(left, img1.size[0]):
            for j in range(img1.size[1]):
                if not self.is_pixel_equal(img1, img2, i, j):
                    left = i
                    return left
        return left

    def get_track(self, distance):
        """
        根据偏移量获取移动轨迹
        :param distance: 偏移量
        :return: 移动轨迹
        """
        # 移动轨迹
        track = []
        # 当前位移
        current = 0
        # 减速阈值
        mid = distance * 4 / 5
        # 计算间隔
        t = 0.2
        # 初速度
        v = 0

        while current < distance:
            if current < mid:
                # 加速度为正 2
                a = 2
            else:
                # 加速度为负 3
                a = -3
            # 初速度 v0
            v0 = v
            # 当前速度 v = v0 + at
            v = v0 + a * t
            # 移动距离 x = v0t + 1/2 * a * t^2
            move = v0 * t + 1 / 2 * a * t * t
            # 当前位移
            current += move
            # 加入轨迹
            track.append(round(move))
        return track
```

```
    def get_slider(self):
        """
        获取滑块
        :return: 滑块对象
        """
        while True:
            try:
                slider = self.browser.find_element_by_xpath("//div[@class='gt_slider_knob gt_show']")
                break
            except:
                time.sleep(0.5)
        return slider

    def move_to_gap(self, slider, track):
        """
        拖动滑块到缺口处
        :param slider: 滑块
        :param track: 轨迹
        :return:
        """
        ActionChains(self.browser).click_and_hold(slider).perform()
        while track:
            x = random.choice(track)
            ActionChains(self.browser).move_by_offset(xoffset=x, yoffset=0).perform()
            track.remove(x)
        time.sleep(0.5)
        ActionChains(self.browser).release().perform()

    def crack(self):
        # 打开浏览器
        self.open()

        # 保存的图片名字
        bg_filename = 'bg.jpg'
        fullbg_filename = 'fullbg.jpg'

        # 获取图片
        bg_location_list, fullbg_location_list = self.get_images(bg_filename, fullbg_filename)

        # 根据位置对图片进行合并还原
        bg_img = self.get_merge_image(bg_filename, bg_location_list)
        fullbg_img = self.get_merge_image(fullbg_filename, fullbg_location_list)
```

```
            # 获取缺口位置
            gap = self.get_gap(fullbg_img, bg_img)
            print('缺口位置', gap)

            track = self.get_track(gap - self.BORDER)
            print('滑动滑块')
            print(track)

            # 点按呼出缺口
            slider = self.get_slider()
            # 拖动滑块到缺口处
            self.move_to_gap(slider, track)


if __name__ == '__main__':
    crack = Crack('username', 'passwd')
    crack.crack()
    print('验证成功')
```