# Bilibili登陆系统验证码破译

在使用网络爬虫时，经常遇到的问题是浏览器网页会设置各种各样的验证码，来判断你是人为操作还是电脑操作，验证不通过的情况下则无法继续获得数据，本次报告将讲述如何通过程序实现验证码的破译。

验证码有很多种，最早的有填写图片中的内容（一般都为数字或字母或二者的组合），高端的有12306的选择符合特征的图片，目前市面上的主流是拖动滑块到缺口处完成拼图，本次示例选择的bilibili弹幕网的登陆系统采用的就是这种验证码，该验证码由现在市场的验证码服务大咖—极验（geetest）提供。

**工作流程：**

1. **前期准备**

采用的编程语言为python，要想开始工作，首先需要一个工具人，上网浏览物色工具人嘉宾候选，找到一个极为合适的候选——Selenium。Selenium是一个用于web应用程序测试的工具，Selenium测试直接运行在浏览器中，就像用户真正在操作一样，支持许多浏览器。

接下来便是将这个工具人请到我们的本地嘉宾座上去，打开命令行窗口，输入pip install selenium，稍等片刻，下载成功。

打开VS，新建python项目，新建一个python文件，命名为bilibili\_login.py（这是我的命名，事实上可以任意命名）至此准备工作全部完成，可以正式开始我们的工作。

1. **打开浏览器，进入网页，输入账号密码，点击登陆。**

此操作任何人都会手动操作，然而我们本次操作完全由程序来实行。

首先要引入selenium包，在bilibili\_login.py中输入：

from selenium import webdriver

from selenium.webdriver import support

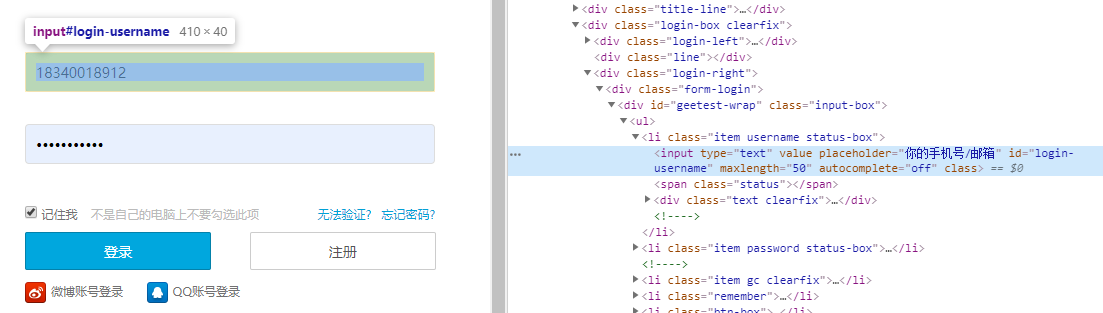
from selenium.webdriver import ActionChains

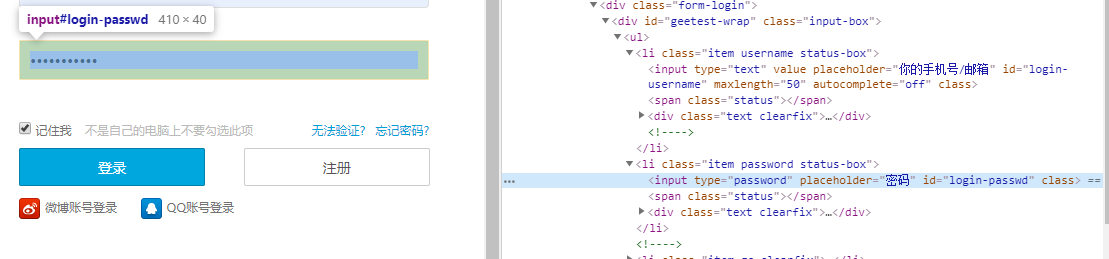
然后输入如下代码，以下代码的作用是使用Chrome浏览器打开bilibili登陆网页：

driver = webdriver.Chrome()

driver.get("https://passport.bilibili.com/login")

接下来便是找到账号密码的输入框，以及登录按钮，手动打开哔哩哔哩登陆网页，按F12查询网站源代码，找到账号输入框，密码输入框，登陆按钮的元素如下：







Selenium提供了多种定位网页元素的方法，观察源码可知，账号密码输入框可通过id确定，登录按钮可以通过类名确定，故在文件中写入如下代码(为了保护本人自己的账号密码安全，故其中没有写本人的账号密码，请在响应位置输入您自己的账号密码进行测试)：

m\_username = driver.find\_element\_by\_id("login-username")

m\_username.clear()

m\_username.send\_keys("your\_user\_name")

sleep(0.5)

m\_password = driver.find\_element\_by\_id("login-passwd")

m\_password.clear()

m\_password.send\_keys("your\_password")

sleep(0.5)

m\_login = driver.find\_element\_by\_class\_name("btn-login")

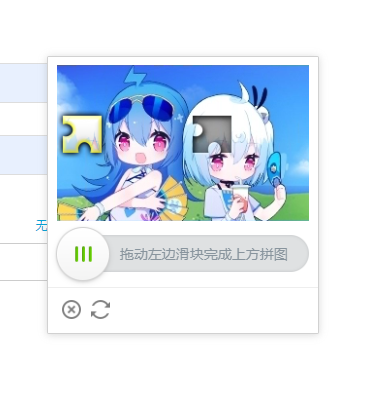
m\_login.click()

sleep(1.5)

该代码作用是输入账号密码，并点击登陆键。

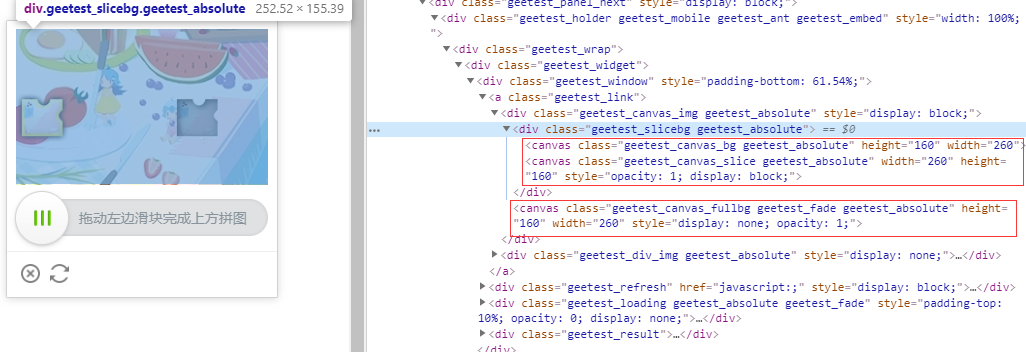
1. **分析验证码**

完成上述步骤后，会弹出验证码，如图所示：



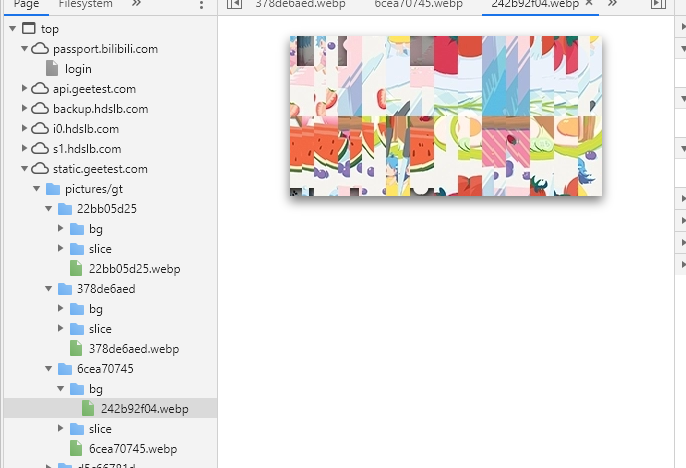
想要破译，一种方法是分析它的js加密方法，通过大量抓包分析找到它的返回参数，直接自动生成需要的参数即可，这种方法工程量大一些，并且官方js脚本一升级，就得重新分析，耗时耗力。

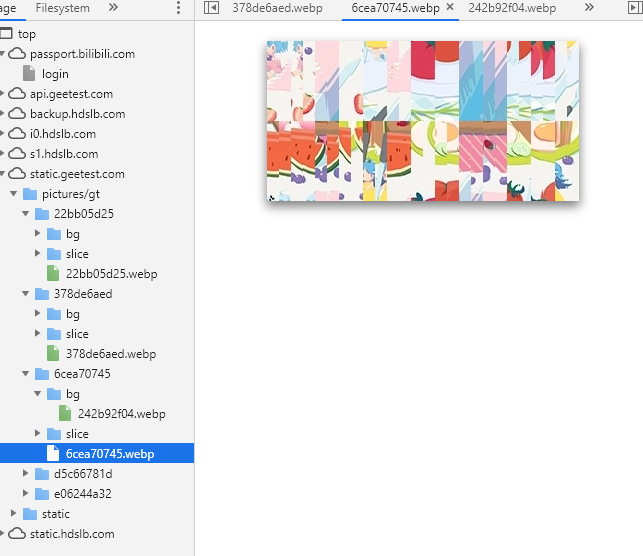
而本次实验采用的是另一种办法，模拟用户拖动，我们人为操作时，需将滑块拖动到缺口位置完成拼图，那么我们在模拟时，首先要知道的便是缺口的位置。我们在此来分析网页源码。



可以看到，很遗憾，只是给出了几个canvas元素，并没有给出canvas上的图片的内容或者链接，第一是为了安全，第二每次弹出的验证码图片都是不一样的，缺口也不一样。不过这很正常，这种用户量多的大型动态网站，采用这些措施，是很必要的，也没有想到能那么容易就获取到其验证码图片，我们再想想其他方法。

在Elements界面没有找到所需要的，我们去Sources界面找找。在Sources界面，找到了这样的一个文件：





可以发现，源图片是剪开的，显示的完整图片是由源图片通过一定的排列方式重组得到的，就像是在玩拼图游戏，而拼图的步骤就隐藏在源码的js文件里，于是又去查看关于验证码的js文件。将其复制到本地编辑器后查看，如下：

js文件本身没有换行，本身可读性就极差，且该文件进行了加密处理，所有变量的名称都进行了更改，毫无可读性，在看了一段时间后便放弃了，不仅感叹这些网页工作者真是煞费苦心，为了自己的保密，啥招都使上了。这下子得想其他的办法了。

1. **再次分析源码**

再次在源码中寻求解决办法，发现了显示验证码的canvas元素中有如下属性：



只对html略知皮毛的我也知道这表示的是显示方式，那么将其修改后会发生什么，进行了尝试后，我有了一些发现。

将滑块的display值设置为none，会显示带缺口图片。如图：



再将full\_bg的display值设置为inline，会显示不带缺口的完整图片。



1. **截取带缺口的图和完整的图片：**

现在灵感有了，只要获得带缺口的图和不带缺口的图，比较两者之间的差异，就可以得到缺口的位置。

这下子又要请出我们的工具人了，上网查询后，selenium中提供了运行js脚本的功能，可以利用js脚本来改变网页元素的属性值(本例中由于要截取图片，需要对style的display值进行更改)，还提供了截取窗口的函数，还发现了可以得到元素位置和尺寸的函数，这下子所需要的数据都齐全了，只要根据画布的尺寸和位置，再窗口截图中截取那一部分就行了。当然，为了精度，再这些步骤之前要先最大化浏览器页面。代码如下：

#截取背景图片

slice = driver.find\_element\_by\_class\_name("geetest\_canvas\_slice")

js = "document.getElementsByClassName(\"geetest\_canvas\_slice\")[0].style.display=\"none\";"

driver.execute\_script(js)

driver.switch\_to.default\_content()

screenshot = driver.get\_screenshot\_as\_png()

screenshot = Image.open(BytesIO(screenshot))

screenshot.save("D:\\screen1.png")

#截取完整图片

full\_bg = driver.find\_element\_by\_class\_name("geetest\_canvas\_fullbg")

js = "document.getElementsByClassName(\"geetest\_canvas\_fullbg\")[0].style.display=\"inline\";"

driver.execute\_script(js)

driver.switch\_to.default\_content()

screenshot = driver.get\_screenshot\_as\_png()

screenshot = Image.open(BytesIO(screenshot))

screenshot.save("D:\\screen2.png")

print(full\_bg.location)

x = full\_bg.location['x']

y = full\_bg.location['y']

w = full\_bg.size['width']

h = full\_bg.size['height']

cropImage = screenshot.crop((x,y,x+w,y+h))

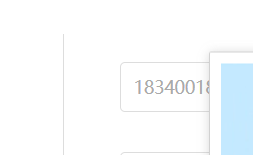
cropImage.save("D:\\fullbg.png")

bg\_image = Image.open("D:\\screen1.png")

bg\_image = bg\_image.crop((x,y,x+w,y+h))

bg\_image.save("D:\\bg.png")

然而点击运行后，发现所截取的部分并不是验证码的部分，有很大的偏差，如图：



在网上查询，发现为如下原因：



由于笔记本屏幕小，所以默认时的显示比例是125%，而函数获得的元素位置信息是没有经过缩放的。解决方法：将缩放与布局改成100%。

改过之后不得不说感觉所有东西都小了许多，眼睛有点不适应，不过程序运行结果总算正确了。



1. **对图片进行处理获得缺口位置：**

从直观感觉上，两幅图片只有在缺口处像素值是不一样的，其他地方的像素值应该是完全相等的，因此我写了一个算法，从左上角点开始，逐列每个像素从上到下遍历，遇到两张图片像素不一样时，返回该像素的列数，然而这样有几个问题。

第一个问题是不知道是截图的原因还是网页上所显示的两张图片的原因，又或是geetest的高端技术，有些非缺口处的像素值并不是完全一样的，只相差了很少，肉眼看不出来，但是会造成我的方法返回错误的列值。解决方法，判断两个像素是否相等时，加入一个阈值，如果两个像素之间的差异小于这个阈值，则算两个像素相等。

第二个问题是，缺口并不是规则的矩形，会有凹槽或者凸起，如图：

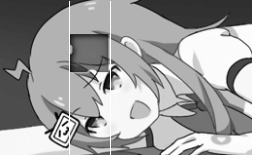
  

解决方法：采用逐列遍历，记录每一列不同的像素数，超过某个阈值后，则返回该列。

第三个问题是，由于图片的保存格式为png，有四个颜色通道，对每个通道值进行阈值判断是否相同，精度不够，而图中缺口处只是图片的亮度不同。解决方法：将图像转化为灰度图，在比较像素时只比较其灰度值。如图：



上述处理获得左边界，同样方式获得又边界，并在图中将边界标示出来：



代码如下：

def is\_pixel\_equal(self, img1, img2, x, y,threshold = 30):

"""

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]

if (abs(pix1 - pix2) < threshold):

#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\_left\_distance\_use\_line(self,fullbg,bg,threshold = 30):

#获得准确的左边界

for i in range ( 1 ,fullbg.size[0]):

diffent\_pixel\_num = 0

for j in range( 1,fullbg.size[1]):

if not self.is\_pixel\_equal(fullbg,bg,i,j,threshold):

diffent\_pixel\_num +=1

if(diffent\_pixel\_num >25):

return i

def nothing(self):

print("nothing")

def get\_right\_distance\_use\_line(self,fullbg,bg,threshold = 30):

#获得准确的右边界

for i in range ( 1 ,fullbg.size[0]):

diffent\_pixel\_num = 0

for j in range( 1,fullbg.size[1]):

if not self.is\_pixel\_equal(fullbg,bg,fullbg.size[0]-i,j,threshold):

diffent\_pixel\_num +=1

if(diffent\_pixel\_num >25):

return fullbg.size[0]-i

def png2gray\_use\_img(self,img):

gray =img.convert('L')

return gray

def mark\_image(self,left\_distance,right\_distance,bg):

#对找到的缺口边界进行白边标记

mark\_bg = bg

for i in range(1,mark\_bg.size[1]):

mark\_bg.putpixel((left\_distance,i),(255))

mark\_bg.putpixel((right\_distance,i),(255))

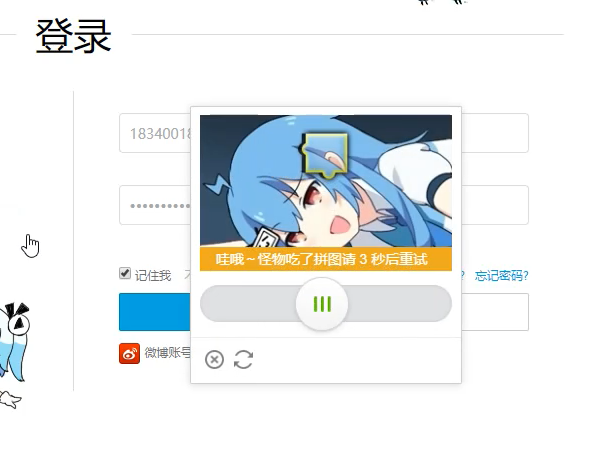
return mark\_bg

1. **根据获得的位置拖动滑块**

到这里，万里长征基本只剩最后一步了，只要把滑块拖到正确的位置完成拼图了就可以了，然而这最后一步还是遇到了许多麻烦。

首先是位置问题，上述你可能会疑惑，已经有左边界的位置了，为什么还要耗费计算量去获得右边界的位置，这是因为在实验过程中发现，缺口处的大小并不是每次都一样的，会差上几个像素，导致精度不够，往往不能移动到正确的位置，故加上了右边界的位置，加上左边界的位置除以二，得到中心的位置，这样成功率会高一些。另外不能只是简单的移动这样的距离，滑块会在缺口位置的偏右侧，故还需加上一个偏移量，才能保证有足够高的正确率。

还有一个问题是如何欺骗浏览器，必须把滑块的移动轨迹模拟的像人为的，一开始我使用匀速移动到目标位置，结果自然可想，被浏览器识别出是机器所为，把验证码给吞了，故我又修改了滑块的移动轨迹，引入随机数，且开始移动幅度较大，移动时还伴随着y轴的上下抖动，大幅度移动超过目标位置后，再以缓慢的速度返回，尽量模拟出是一个人在操作。然而即使这样，还是有一定可能被吞，不过相比较匀速移动的轨迹，至少这个可以成功骗过浏览器几次。



代码：

def super\_move\_slider(self, move\_distance):

#高级拖动

slider = self.driver.find\_element\_by\_class\_name("geetest\_slider\_button")

#print("slidersize:",slider.size)

#slider\_width = slider.size["width"]

#offset = -3

#move\_distance = left\_distance+ slider\_width/2-10

ActionChains(self.driver).click\_and\_hold(slider).perform()

current\_move = 0

while (current\_move < move\_distance):

#if(move\_distance - current\_move < 5):

# each\_step = random.uniform(0.8,1.2)

#else:

each\_step = random.uniform(5,8)

ActionChains(self.driver).move\_by\_offset(each\_step,random.uniform(-2,2)).perform()

current\_move +=each\_step

sleep(0.05)

while(current\_move >move\_distance ):

each\_step = random.uniform(1.0,2.0)

ActionChains(self.driver).move\_by\_offset(-each\_step,random.uniform(-2,2)).perform()

current\_move -=each\_step

sleep(0.1)

#print("current\_move:",current\_move)

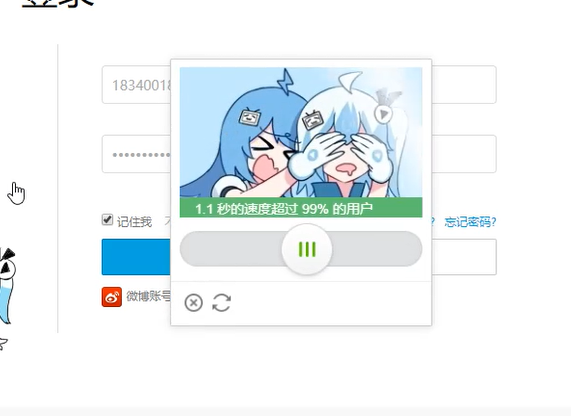
style = slider.get\_attribute("style")

#print("slider属性",style)

ActionChains(self.driver).release().perform()

1. **完善程序循环：**

万里长征还剩最最最后一步，由于我的程序并不是百分之百能成功，故需要写入循环代码，在失败后开始下一次尝试，在这一步骤中发现，本来在移动位置有偏差时，应该点击刷新验证码重新识别，然而geetest提供的刷新按钮只是一个图标，并不是一个按钮，不能像点击登陆一样点击，只能通过刷新浏览器界面的方式重新尝试验证。



源代码：

注：为了保护本人账号安全，没有将本人账号密码放入，请测试者将自己的bilibili账号密码输入main函数的your\_username和your\_password进行测试。



实验录屏：

