Python 接口自动化

作者：上海-悠悠

更多自动化（**selenium、python 接口、appium**）扫码关注

个人微信公众号：yoyoketang (扫二维码关注)

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

目录

第 1 章 Fiddler 9
1.1 抓 firefox 上 https 请求 9
1.1.1 网页不安全 9
1.1.2 fiddler 设置 9
1.1.3 导出证书 10
1.1.4 导入到 firefox 浏览器 11
1.2 证书导出失败问题 12
1.2.1 证书无法导出 12
1.2.2 无法导出问题解决方案 13
1.2.3 删除证书 15
1.3 抓 app 请求 17
1.3.1 设置 17
1.3.2 查看电脑 IP 18
1.3.3 设置代理 19
1.3.4 抓 APP 上的 HTTPS 请求 20
1.3.5 设置过滤 21
1.4 查看 get 和 post 请求 22
1.4.1 get 请求 22
1.4.2 post 请求 23
1.4.3 如何找出需要的请求 24
1.4.4 get 和 post 请求参数区别 25
1.5 工具介绍 27
1.5.1 工具简介 27
1.5.2 会话框 28
1.5.3 Request 和 Response 29
1.5.4 decode 解码 30
1.6 接口测试 Composer 31
1.6.1 Composer 简介 31
1.6.2 模拟 get 请求 32
1.6.3 Json 数据 34
1.6.4 模拟 post 请求 35
1.7 get 请求 37
1.7.1 url 详解 37
1.7.2 url 解析 39
1.7.3 请求参数（params） 40
1.7.1 UrlEncode 编码 40
1.8 post 请求（body） 41
1.8.1 body 数据类型 41
1.8.2 json 格式 43
1.8.3 x-www-form-urlencoded 44
1.8.4 WebFrom 45
1.8.5 xml 46

1.9 打断点 bpu 46
1.9.1 断点 46
1.9.2 断点的两种方式 47
1.9.3 全局断点 48
1.9.4 单个断点 50
1.9.5 拦截来自某个网站所有请求 51
1.9.6 命令行其它相关指令 52
1.10 会话保存 52
1.10.1 保存为文本 52
1.10.2 几种保存方式 53
1.10.3 乱码问题（decode） 54
1.10.4 保存与导入全部会话 55
1.10.5 Repaly 56
1.11 自定义会话框 57
1.11.1 添加会话框菜单 57
1.11.2 隐藏会话菜单 59
1.11.3 调整会话框菜单顺序 60
1.11.4 会话排序 61
1.12 http 协议简介 62
1.12.1 什么是 http 62
1.12.2 请求报文 62
1.12.3 响应报文 63
1.12.4 完整的 http 内容 64
1.13 请求行 67
1.13.1 8 种请求方法 67
1.13.2 url 详解 69
1.13.3 协议版本 70
第 2 章 requests 70
2.1 发 get 请求 70
2.1.1 环境安装 70
2.1.2 get 请求 71
2.1.3 params 72
2.1.4 content 73
2.1.5 response 74
2.2 发 post 请求 75
2.2.1 查看官方文档 75
2.2.2 发送 post 请求 76
2.2.3 json 77
2.2.4 headers 77
2.2.5 登陆博客园 78
2.2.6 参考代码 79
2.3 发 post 请求(data) 80
2.3.1 登录 jenkins 抓包 80
2.3.2 请求头部 82

2.3.3 实现登录.....83
2.3.4 判断登录是否成功.....84
2.3.5 判断登录成功.....84
2.3.6 参考代码.....85
2.4 data 和 json 傻傻分不清.....86
2.4.1 识别 json 参数.....87
2.4.2 识别 data 参数.....89
2.5 https 请求（SSL）.....91
2.5.1 SSL 问题.....92
2.5.2 verify 参数设置.....93
2.5.3 忽略 Warning.....93
2.5.4 python3 的童鞋.....94
2.6 session 关联接口.....95
2.6.1 session 简介.....95
2.6.2 使用 session 登录.....95
2.6.3 保存编辑.....96
2.6.4 参考代码.....98
2.7 cookie 绕过验证码登录.....100
2.7.1 抓登录 cookie.....100
2.7.2 cookie 组成结构.....101
2.7.3 添加 cookie.....102
2.7.4 参考代码.....103
2.8 Json 数据处理.....104
2.8.1 json 模块简介.....105
2.8.2 Encode（python->json）.....105
2.8.3 decode(json->python).....107
2.8.4 案例分析.....108
2.8.5 参考代码.....109
2.9 重定向（Location）.....110
2.9.1 重定向.....110
2.9.2 禁止重定向（allow_redirects）.....111
2.9.3 获取重定向后地址.....112
2.9.4 参考代码.....113
2.10 参数关联.....113
2.10.1 删除草稿箱.....114
2.10.2 提取参数.....116
2.10.3 传参.....116
2.10.4 参考代码.....117
2.11 token 登录.....118
2.11.1 登录返回 token.....119
2.11.2 请求头带 token.....119
2.11.3 token 关联.....120
2.12 案例分析(csrfToken).....121
2.12.1 登录前后对比.....121

2.12.2 get 请求 123
2.12.3 post 请求遇到的坑....123
2.12.4 重定向....125
2.13 requests 官方文档....126
2.14 常见登录案例（github 源码下载） 127
第 3 章 unitttest....128
3.1 unittest 简介 128
3.1.1 unittest 简介 128
3.1.2 简单用法....129
3.1.3 小试牛刀....130
3.1.4 前置和后置....131
3.2 unittest 执行顺序 132
3.2.1 案例分析....132
3.2.2 执行结果....133
3.2.3 结果分析....133
3.3 unittest 断言 134
3.3.1 简单案例....134
3.3.2 自定义异常....135
3.3.3 unittest 常用的断言方法 136
3.4 案例一：查询快递接口....137
3.4.1 查询 yunda 单号....137
3.4.2 查询 tiantian 单号 138
3.4.3 设计用例....139
3.5 案例二：登录结果....141
3.5.1 抓登录接口....142
3.5.2 参考代码....142
3.6 批量执行用例 discover 144
3.6.1 新建测试项目....144
3.6.2 discover 加载测试用例 146
3.6.3 run 测试用例 147
3.6.4 参考代码....148
3.7 生成 html 报告 149
3.7.1 导入 HTMLTestRunner 149
3.7.2 demo 解析 150
3.7.3 生成 html 报告 151
3.7.4 测试报告详情....152
3.7.5 参考代码....153
3.8 测试报告优化....154
3.8.1 中文乱码....154
3.8.2 修改编码....155
3.8.3 python3 报告问题 156
3.9 搭建项目环境....157
3.9.1 新建工程....157
3.9.2 项目结构....157

3.9.3 run_main 159
3.10 项目源码下载（附案例） 160
3.11 pycharm 运行两种姿势 160
3.11.1 unittest 运行单个用例 160
3.11.2 unittest 运行整个脚本用例 162
3.11.3 python 直接运行模式 162
第 4 章 封装与调用 165
4.1 函数与参数化 165
4.1.1 登录函数 165
4.1.2 保存草稿 165
4.1.3 提取 postid 166
4.1.4 删除草稿 166
4.1.5 参考代码 166
4.2 流程类接口关联 168
4.2.1 接口封装 168
4.2.2 用例导入接口 171
第 5 章 爬虫利器 beautifulsoup4 173
5.1 html 解析器 html.parser 174
5.1.1 安装 174
5.1.2 解析器 175
5.1.3 打印首页博客的时间 176
5.1.4 打印摘要 177
5.1.5 参考代码 178
5.2 bs4 四种对象类 179
5.2.1 读取 HTML 页面 179
5.2.2 解析器：html.parser 180
5.2.3 对象的种类 181
5.2.4 Tag 对象 182
5.2.5 Tag 对象：name 属性 183
5.2.6 Tag 对象：Attributes 184
5.2.7 发福利 184
5.3 爬网站图片 185
5.3.1 目标网站 185
5.3.2 用 find_all 找出所有的标签 185
5.3.3 保存图片 186
5.3.4 参考代码 187
5.4 子节点 188
5.4.1 子节点 188
5.4.2 .contents 189
5.4.3 .children 189
5.4.4 .descendants 190
5.4.5 爬取博客首页的标签内容 191
5.4.6 参考代码 192
5.5 bs4 官方文档 193

第 6 章 数据库读取.....194
6.1 读 excel 数据 xlrd.....194
6.1.1 环境准备.....194
6.1.2 基本操作.....195
6.1.3 excel 存放数据.....196
6.1.4 封装读取方法.....197
6.2 操作 mysql.....198
6.3 操作 oracle.....200
6.4 logger 日志.....202
6.5 发送邮件.....204
6.5.1 163 邮箱.....204
6.5.2 QQ 邮件.....206
6.5.3 兼容 163 和 QQ 邮箱.....208
6.5.4 发送带附件.....209
6.5.5 发给多个收件人.....211
6.5.6 邮件收不到的几种原因.....213
第 7 章 项目实战：git+jenkins 持续集成.....213
7.1 项目结构设计.....213
7.1.1 项目结构.....214
7.1.2 run_main.....214
7.1.3 config 配置.....217
7.1.4 logger.....218
7.1.5 case 放用例.....220
7.1.6 logs 日志查看.....221
7.1.7 生成测试报告.....221
7.1.8 发送报告到邮箱.....222
7.2 项目源码下载.....223
7.3 使用 git 上传到 github.....223
7.3.1 注册 Github 账号.....224
7.3.2 安装 git.....225
7.3.3 本地仓库.....227
7.3.4 遇到问题与解决方案.....229
7.4 更新 github 上代码.....230
7.4.1 clone 代码.....230
7.4.2 git status 查看状态.....231
7.4.3 更新代码.....232
7.5 jenkins 搭建（附赠对应视频）.....233
7.5.1 jenkins.exe 安装.....234
7.5.2 tomcat+Jenkins.war.....236
7.6 git+jenkins 配置 windows 的 slave 节点.....242
7.6.1 管理节点.....243
7.6.2 配置节点.....244
7.6.3 Java web start.....245
7.6.4 启动 Salve.....246

7.6.5 新建 Job 248
7.6.6 源码管理：git 249
7.6.7 执行 dos 指令 251
7.7 jenkins 定时构建 253
7.7.1 定时构建语法 253
7.7.2 Build periodically 254
7.7.3 Poll SCM 254
7.7.4 Job 关联 255
7.7.5 另外两种 256
7.8 jenkins 展示 html 报告 256
7.8.1 构建后操作 257
7.8.2 添加 HTML Publisher plugin 插件 257
7.8.3 添加 reports 259
7.8.4 报告展示 260
7.9 jenkins 上 html 展示问题 262
7.9.1 样式丢失 262
7.9.2 jenkins.msi 解决方案 263
7.9.3 jenkins.war 解决方案 264
7.9.4 重启 jenkins 265
7.10 jenkins 添加 Mac 为 slave-agent 266
7.10.1 新建节点 266
7.10.2 Mac 上设置 269
7.10.3 启动 Slave 270
好书推荐 271
Selenium+python 271
Python 接口自动化 272
Appium+python 273
Python 接口测试开发 274

第 1 章 Fiddler

1.1 抓 firefox 上 https 请求

前言

fiddler 是一个很好的抓包工具，默认是抓 http 请求的，对于 pc 上的 https 请求，会提示网页不安全，这时候需要在浏览器上安装证书。

1.1.1 网页不安全

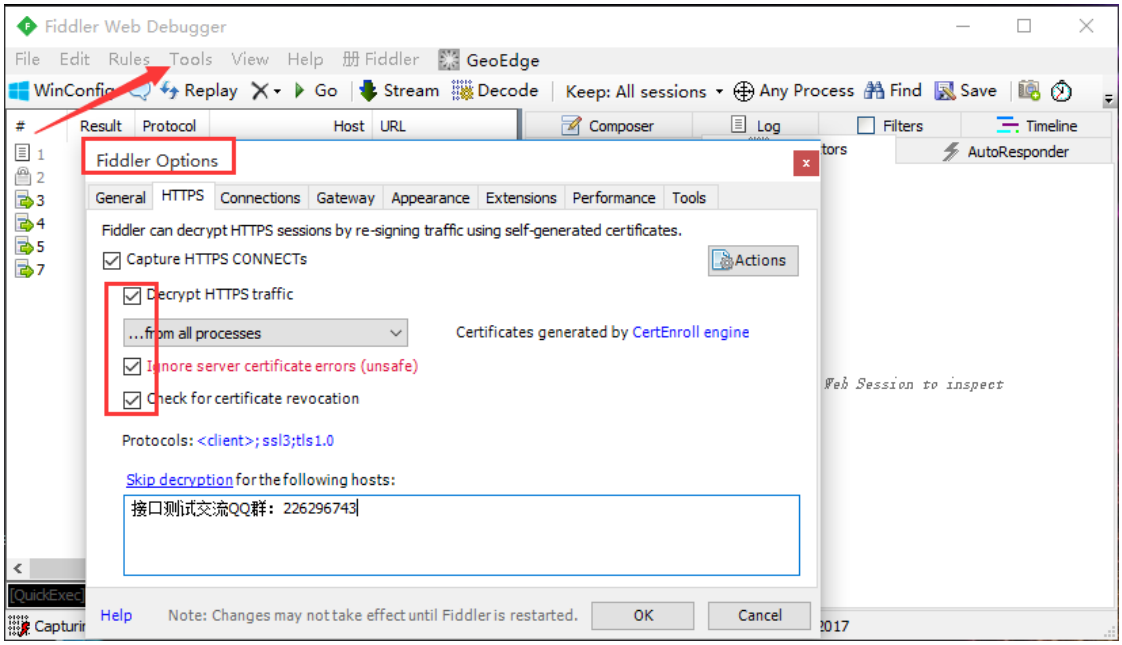
1.用 fiddler 抓包时候，打开百度网页：https://www.baidu.com

2.提示：网页不安全

1.1.2 fiddler 设置

1.打开菜单栏：Tools>Fiddler Options>HTTPS

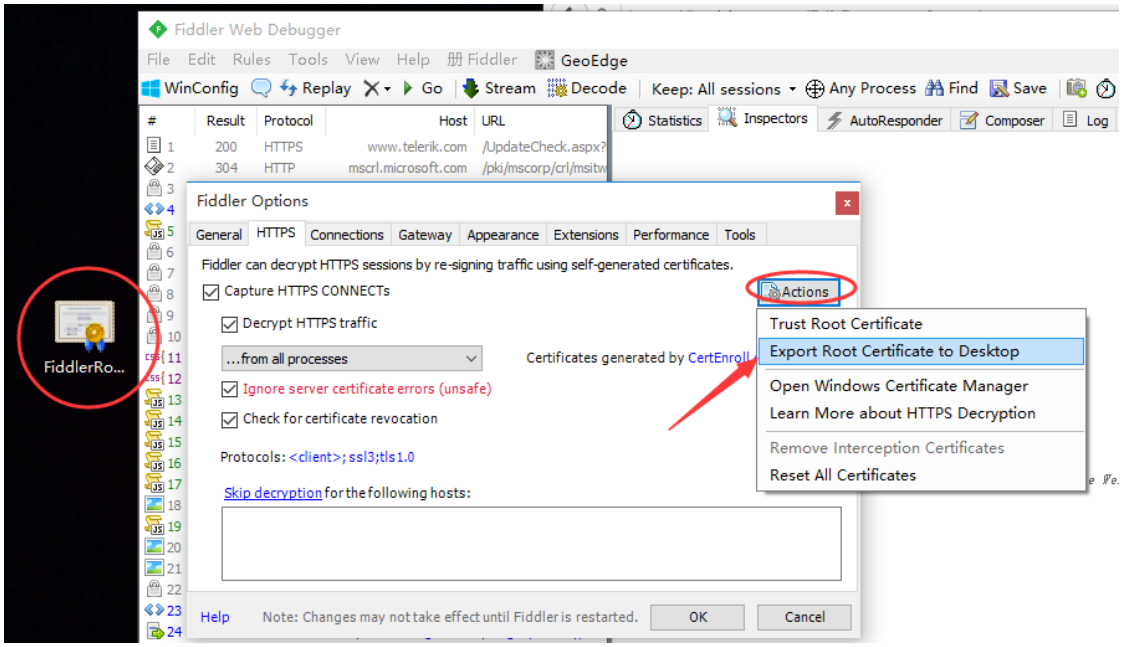
2.勾选 Decrypt HTTPS traffic，里面的两个子菜单也一起勾选了



1.1.3 导出证书

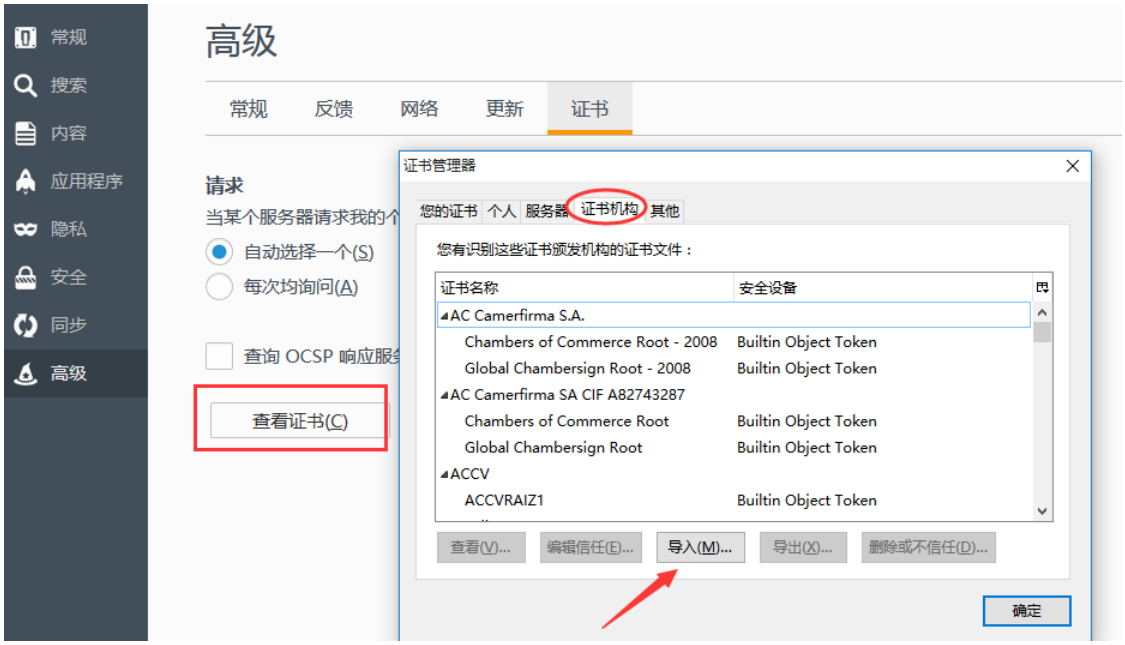
1. 点右上角 Actions 按钮

2. 选第二个选项，导出到桌面，此时桌面上会多一个文件：FiddlerRoot.cer, 如左侧图。

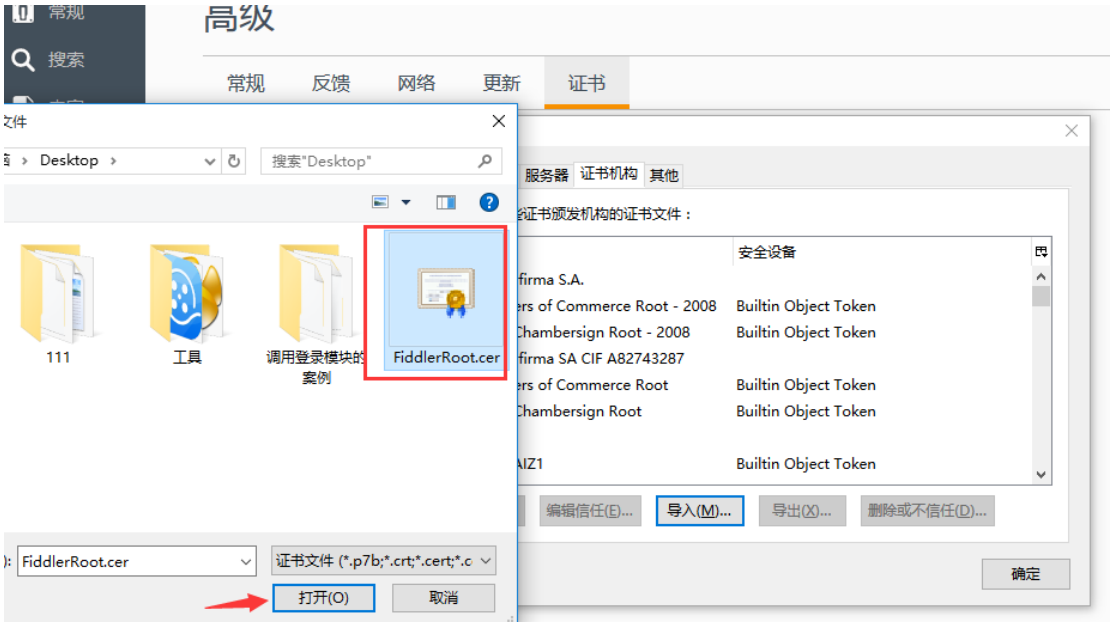


1.1.4 导入到 firefox 浏览器

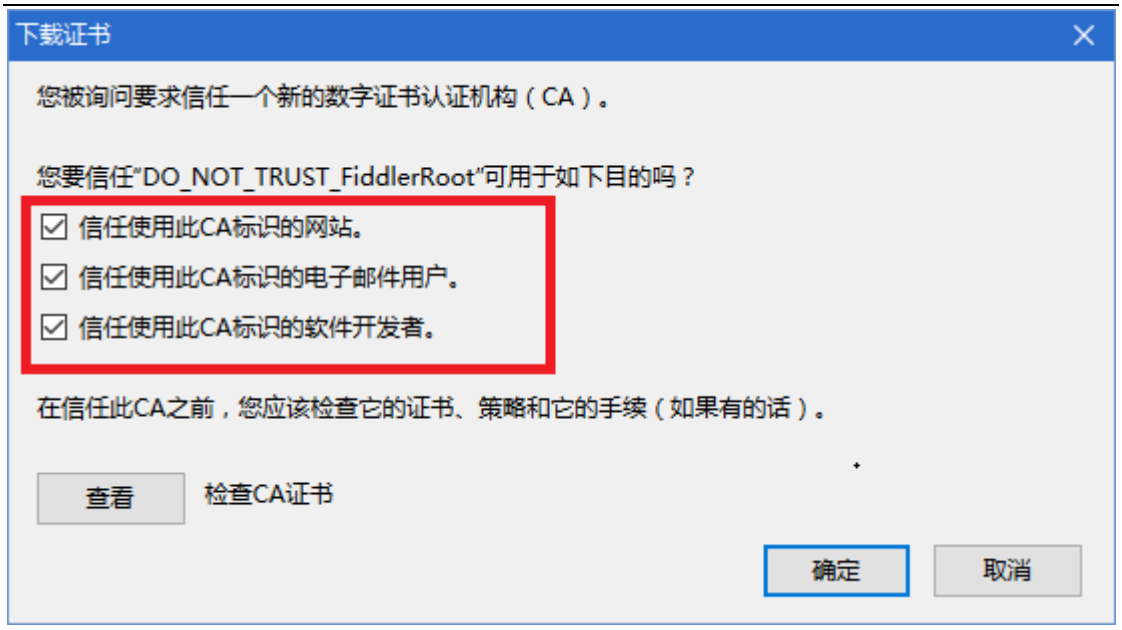
1. 打开右上角浏览器设置》选项》高级》证书》查看证书》证书机构》导入



2. 勾选文件到入



3. 打开文件后，会弹出个框，勾选三个选项就完成操作啦。



如果还不能成功，那就重启浏览器，重启电脑了。

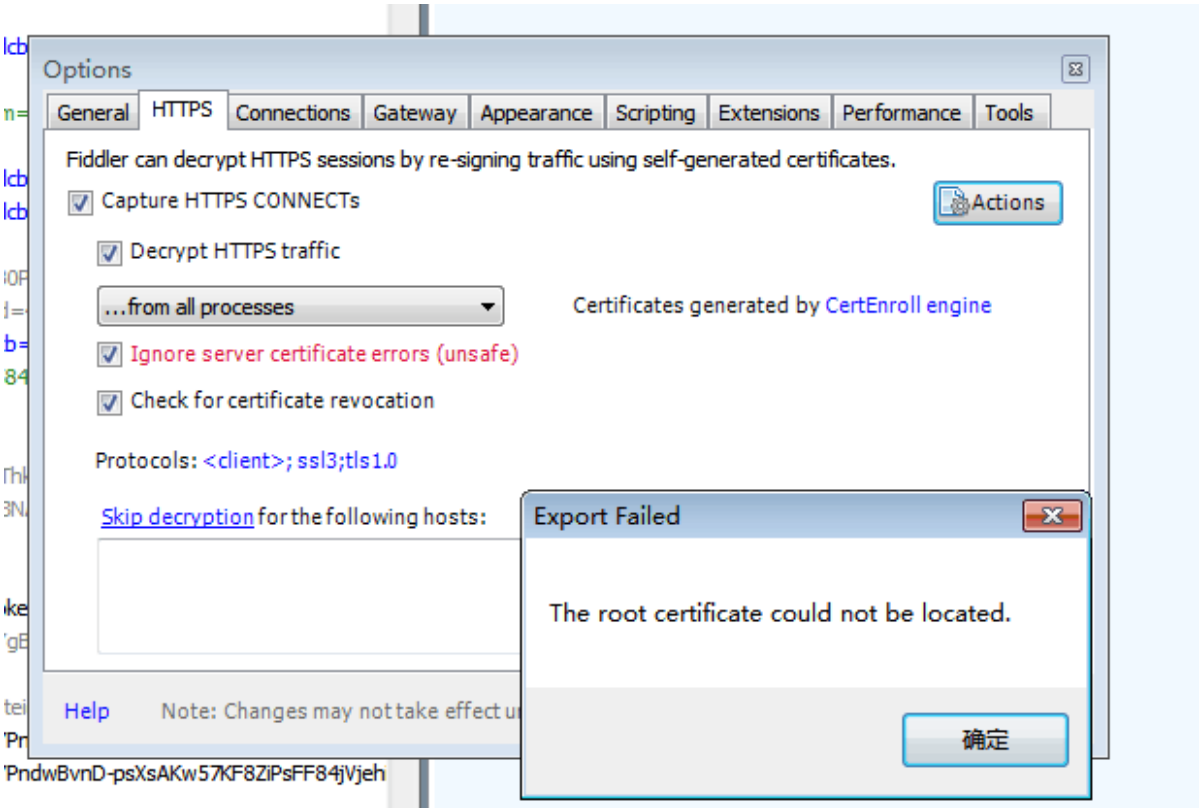
1.2 证书导出失败问题

前言

在点 Actions 时候出现 Export Failed:The root certificate could not be located.最近有很多小伙伴在 fiddler 导出证书的时候，遇到无法导出的问题，收集了几种解决办法，供参考。

1.2.1 证书无法导出

1.在点 Actions 时候出现 Export Failed:The root certificate could not be located.



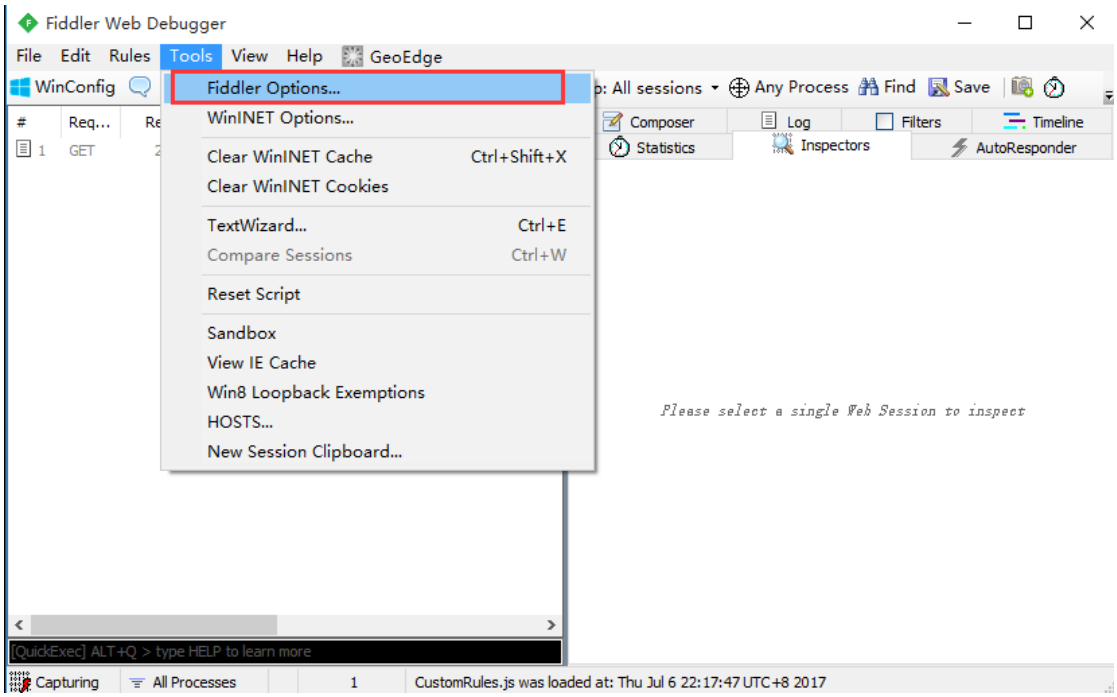
1.2.2 无法导出问题解决方案

1. 首先确保安装的 Fiddler 是较新的版本，先关闭 fiddler

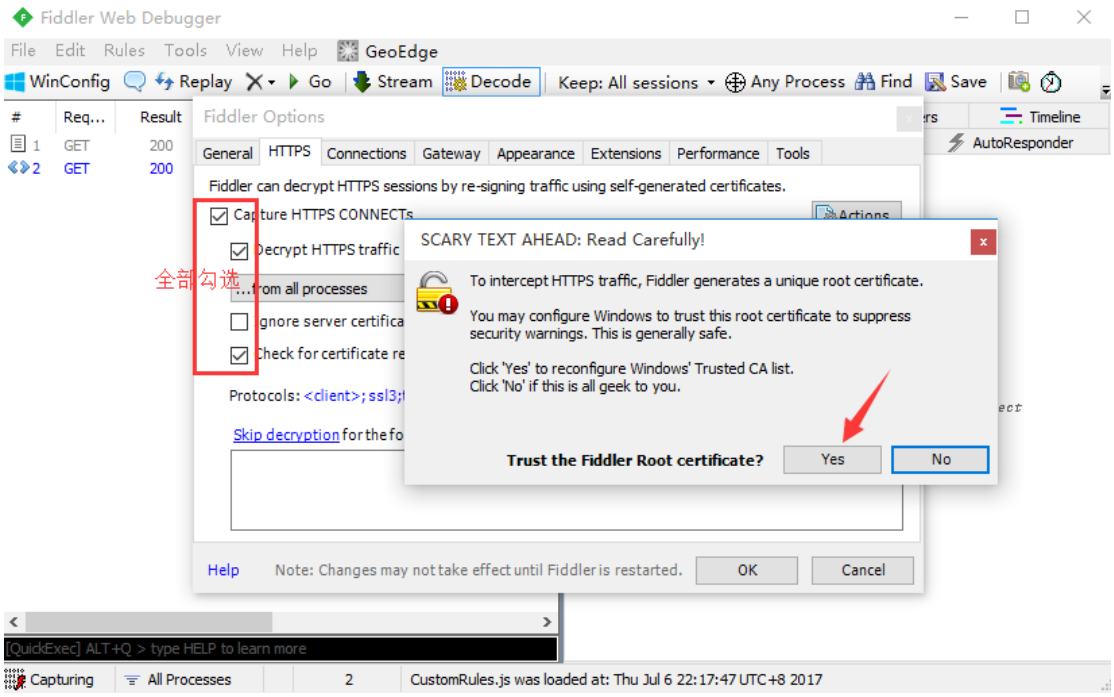
2. 下载并安装 Fiddler 证书生成器

下载地址：
http://www.telerik.com/docs/default-source/fiddler/addons/fiddlercertmaker.exe?sfvrsn=2

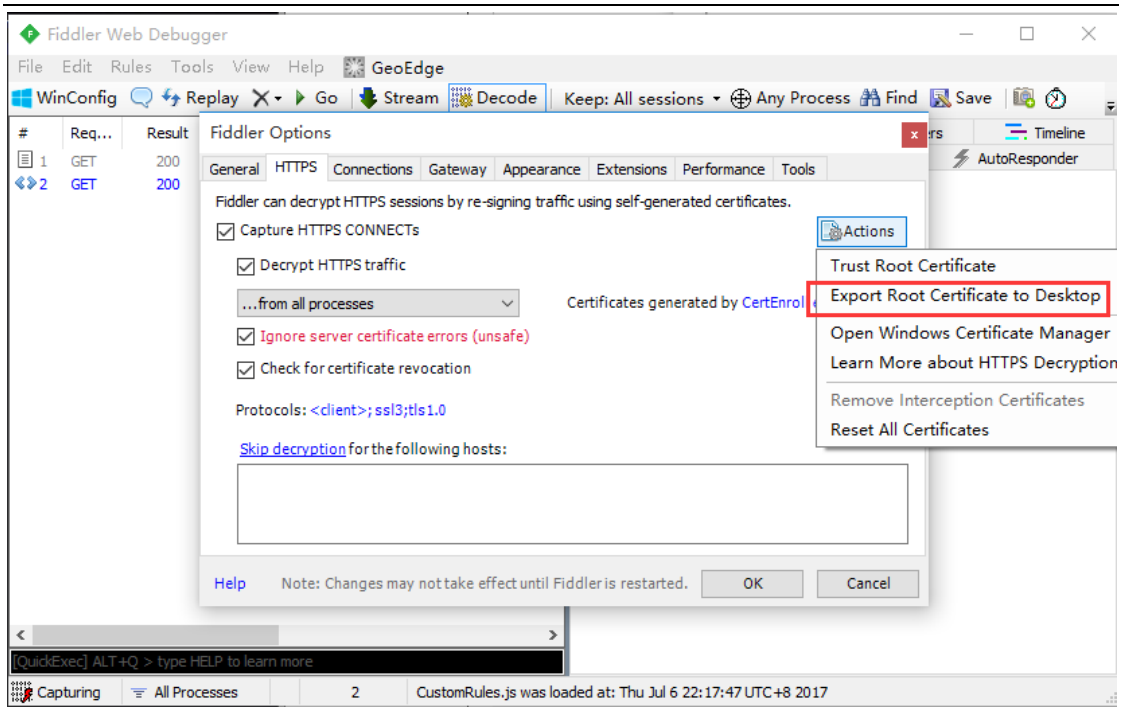
3. 点 Tools>Fiddler Options



4. 勾选 Capture HTTPS traffic



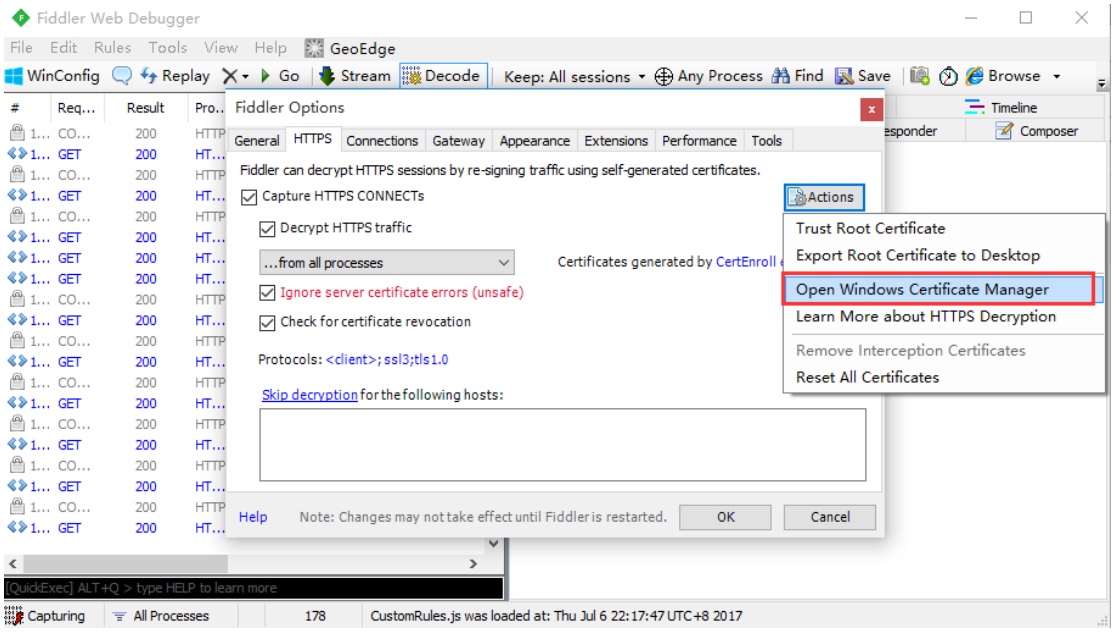
5. 点 Actions 按钮，Export Root Certificate Desktop 按钮导出到桌面



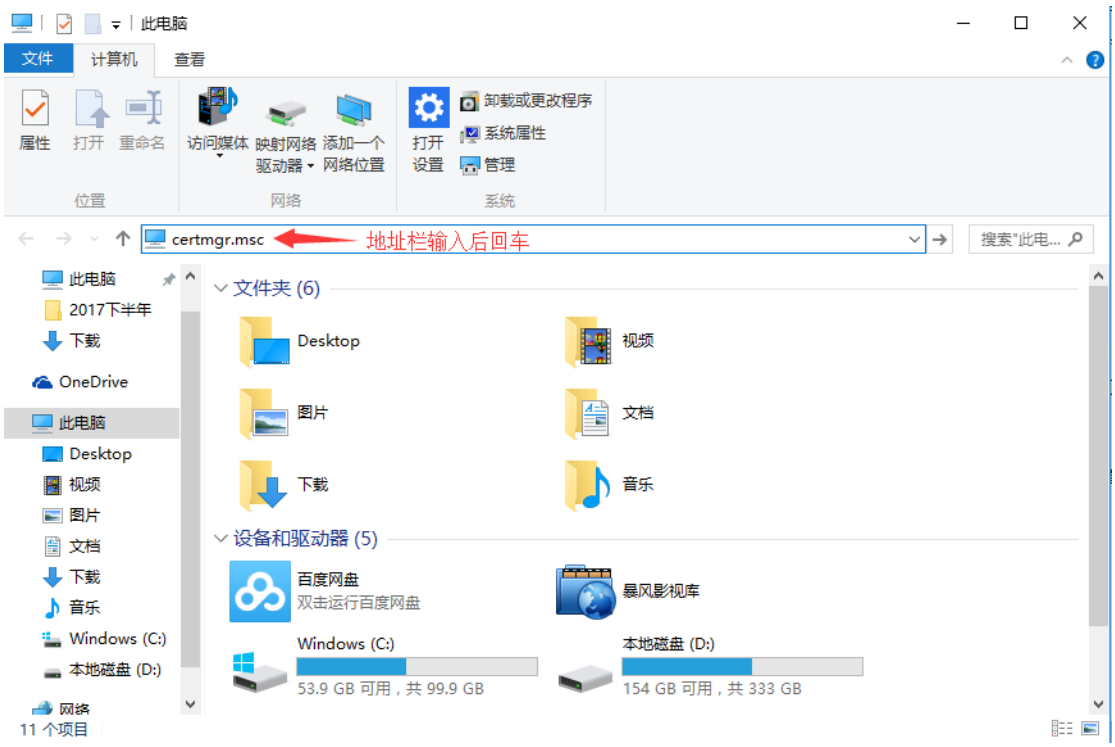
1.2.3 删除证书

1. 有些小伙伴可能之前装过一些 fiddler 证书，安装的姿势不对，导致新的证书不起作用，这时候需要先删掉之前的证书了

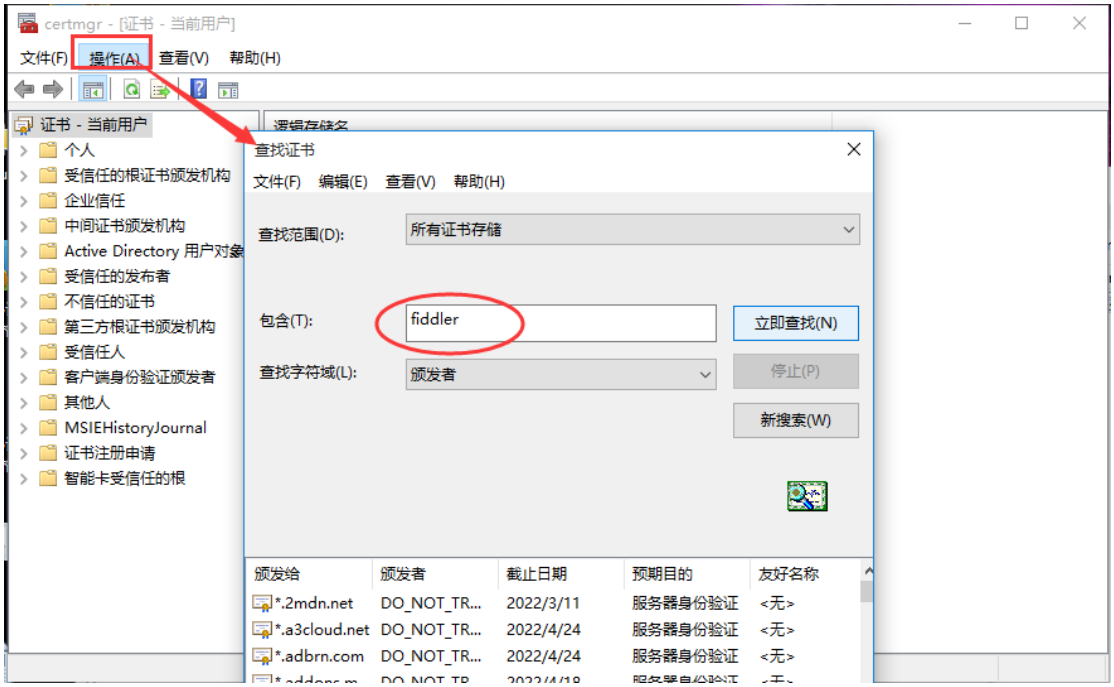
方法一：从 fiddler 里打开证书管理界面



方法二、从文件管理器输入：certmgr.msc 并回车



2. 搜索之前安装的 fiddler 证书，找到之后全部删除



3. 重新下载证书生成器：
http://www.telerik.com/docs/default-source/fiddler/addons/fiddlercertmaker.exe?sfvrsn=2

一路傻瓜式安装，遇到警告什么的直接忽略就行。

4.安装好证书后，安装 1.1 章节 Fiddler 抓包 1-抓 firefox 上 https 请求的浏览器导入证书就行了（弄好之后，重启电脑就 OK 了）

1.3 抓 app 请求

前言

fiddler 抓手机 app 的请求，估计大部分都会，但是如何只抓来自 app 的请求呢？

把来自 pc 的请求过滤掉，因为请求太多，这样会找不到重要的信息了。

环境准备：

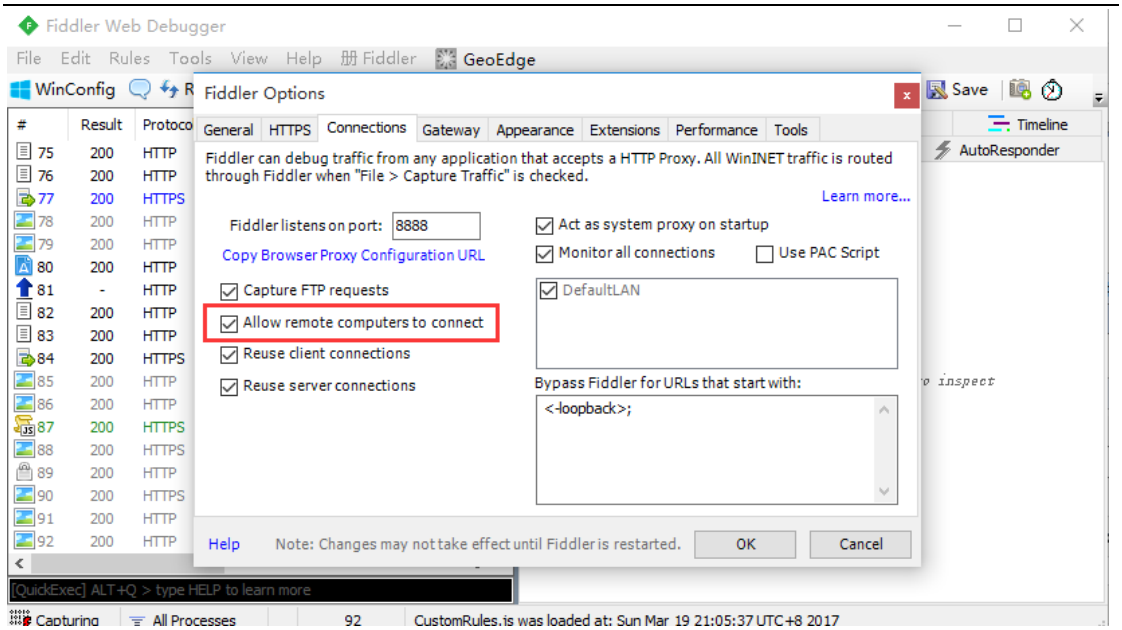
1.电脑上已装 fiddler

2.手机和电脑在同一局域网

1.3.1 设置

1.fiddler>Tools>Fiddler Options>Connections 勾选 Allow remote computers to connect。

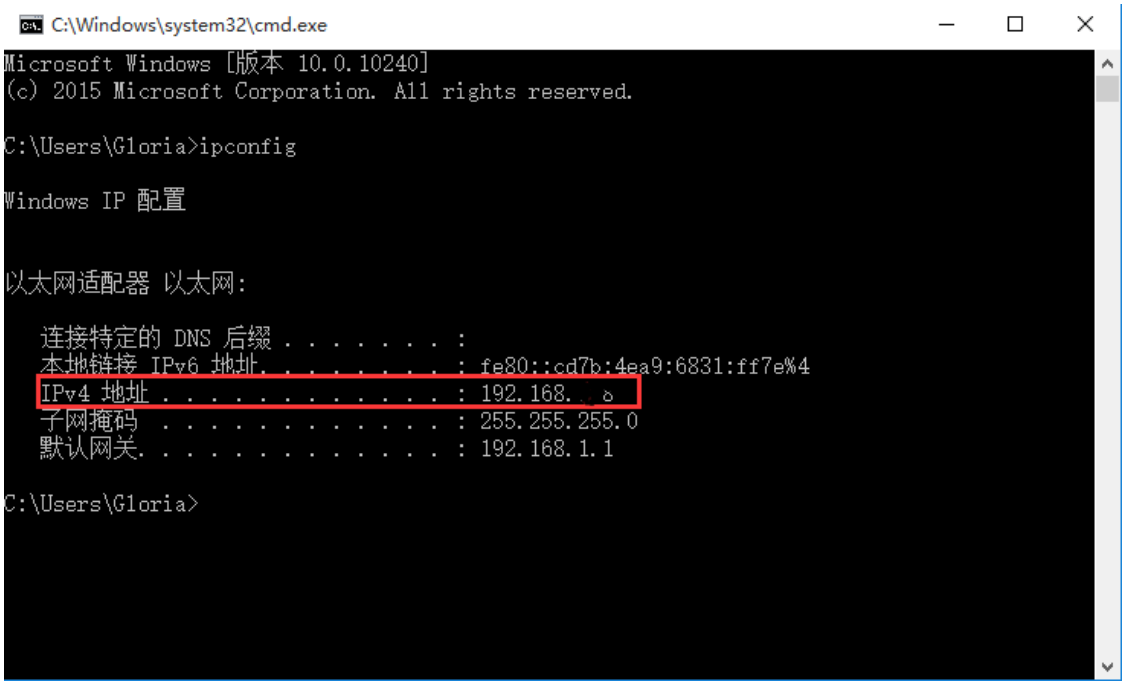
2.记住这里的端口号：8888，后面会用到。



1.3.2 查看电脑 IP

1.打开 cmd，输入：ipconfig,记住这个 IPv4 地址。

>>ipconfig



1.3.3 设置代理

1.手机设置->WLAN 设置->选择该 wifi，点右边的箭头（有的手机是长按弹出选项框）。

2.选择修改网络配置：

配置主机名：与主机电脑 IP 地址保持一致

端口号：8888

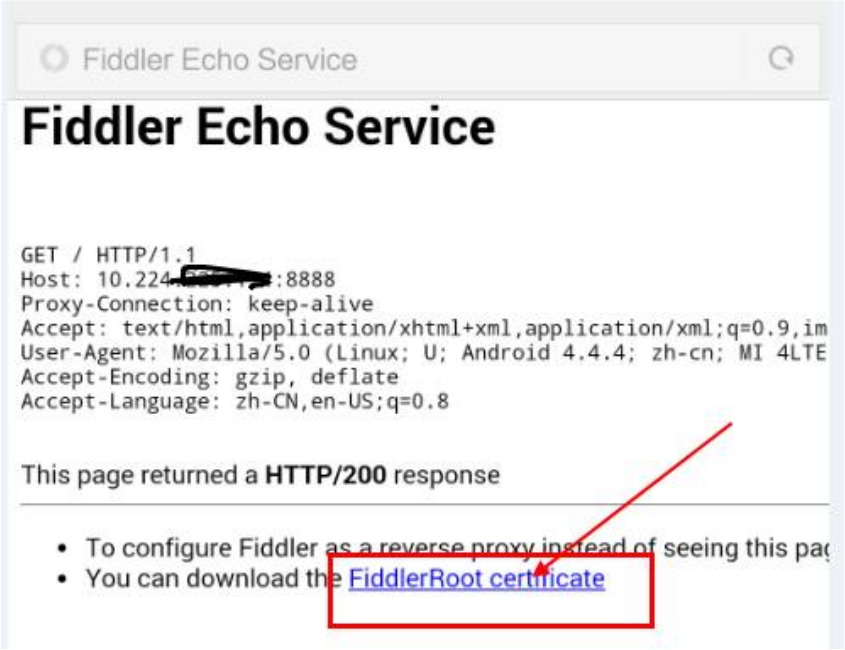
3.保存后就可以抓到来自手机的请求了。

1.3.4 抓 APP 上的 HTTPS 请求

1.如果 app 都是 http 请求，是不需要安装证书，能直接抓到的，如果是 https 请求，这时候手机就需要下载证书了。

2.打开手机浏览器输入：http://10.224.xx.xx:8888 ，这个中间的 host 地址就是前面查到的本机地址。

3.出现如下画面，点箭头所指的位置，点击安装就可以了。



1.3.5 设置过滤

1.手机上设置代理后，这时候 fiddler 上抓到的是 pc 和 app 所有的请求，如果 pc 上打开网址，会很多，这时候就需要开启过滤功能了。

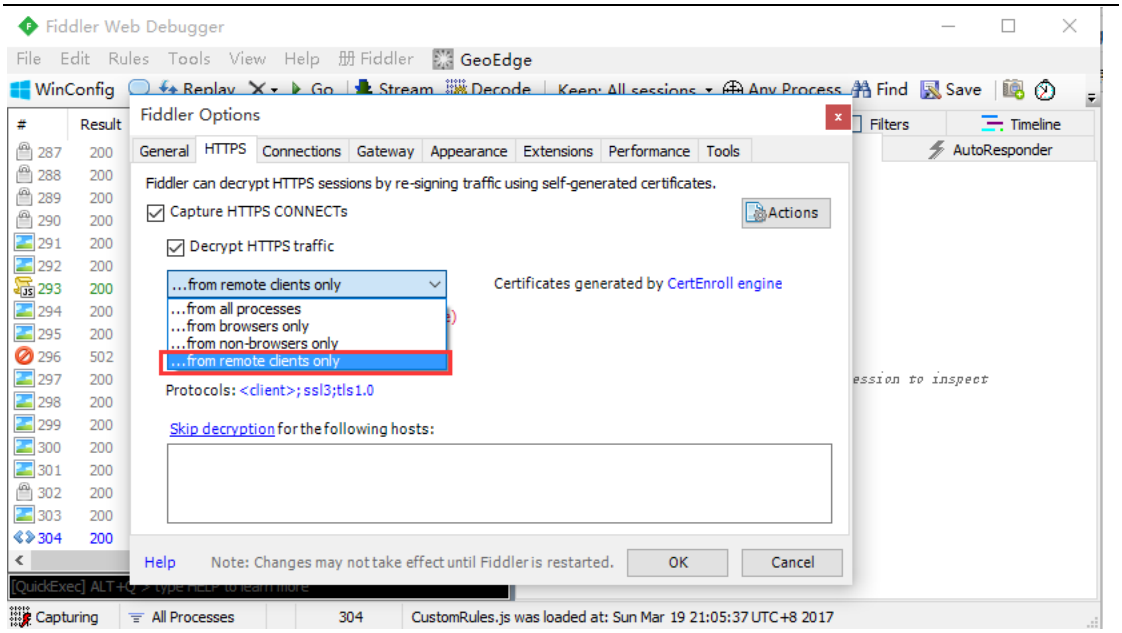
2.打开 fiddler>Tools>Fiddler Options>HTTPS>...from remote clients only,勾选这个选项就可以了

...from all processes :抓所有的请求

...from browsers only ： 只抓浏览器的请求

...from non-browsers only :只抓非浏览器的请求

...from remote clients only:只抓远程客户端请求



（注意：如果手机设置代理后，测玩之后记得恢复原样，要不然手机无法正常上网。）

1.4 查看 get 和 post 请求

前言

前面两篇关于 Fiddler 抓包的一些基本配置，配置完之后就可以抓到我们想要的数据了，接下来就是如何去分析这些数据。

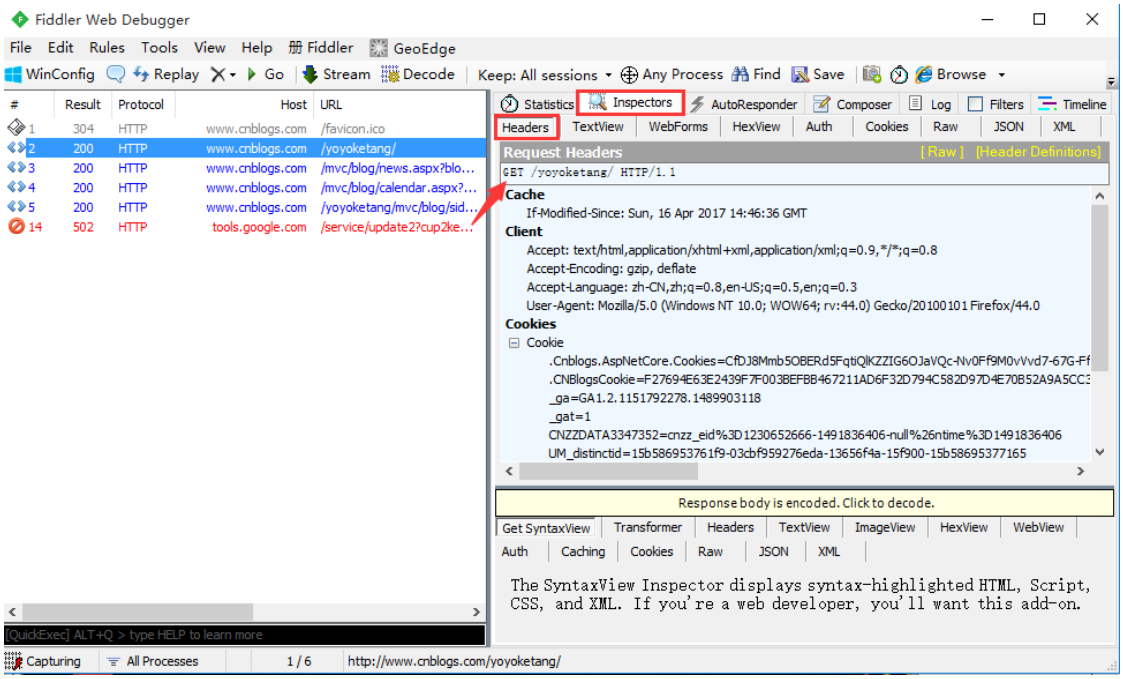
本篇以博客园的请求为例，简单分析 get 与 post 数据有何不一样，以后也能分辨出哪些是 get，哪些是 post 了。

1.4.1 get 请求

1. 打开 fiddler 工具，然后浏览器输入博客首页地址：
http://www.cnblogs.com/yoyoketang/

2. 点开右侧 Inspectors 下的 Headers 区域，查看 Request Headers

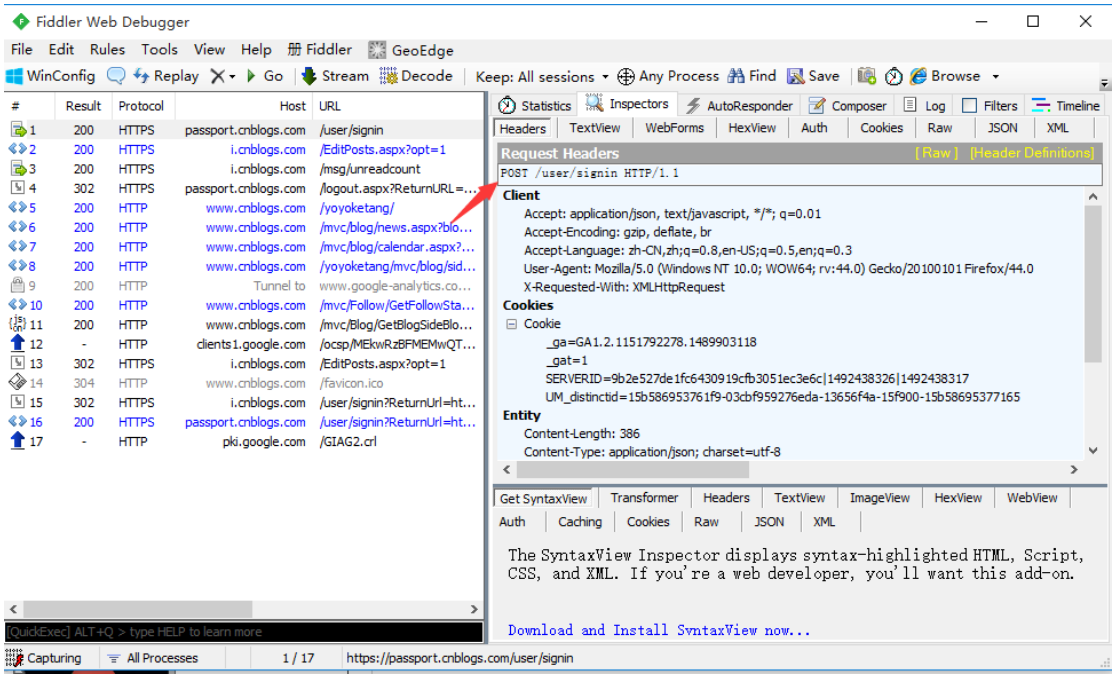
3. Request Headers 区域里面的就是请求头信息，可以看到打开博客园首页的是 get 请求



1.4.2 post 请求

1. 打开登录首页：https://passport.cnblogs.com/user/signin

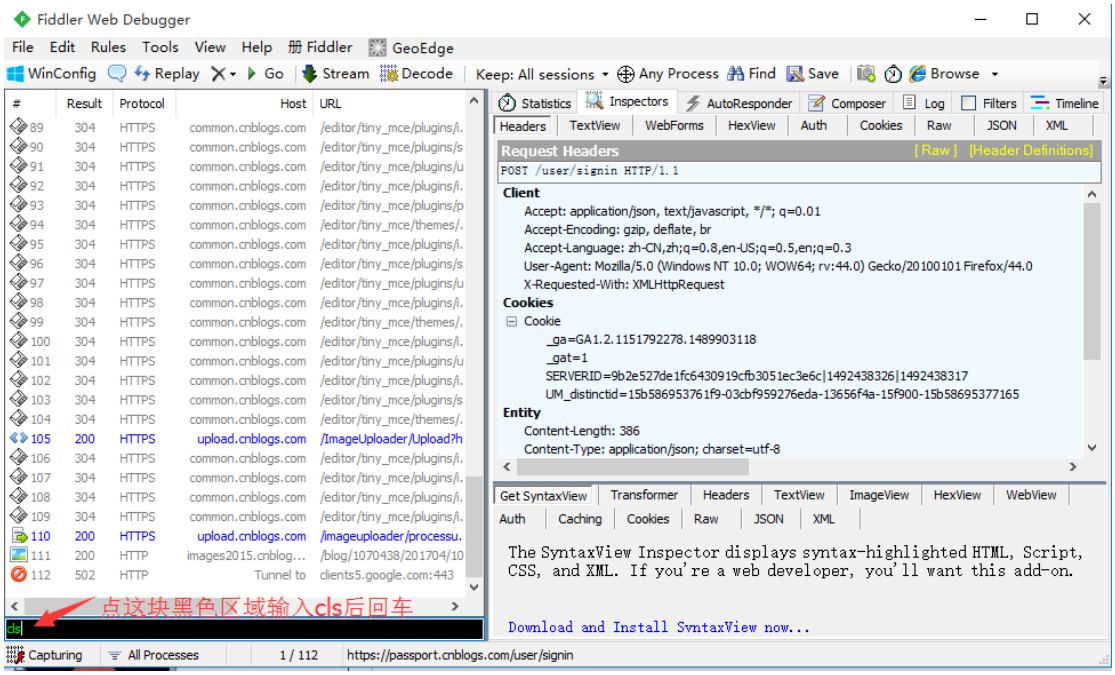
2. 输入账号和密码登录成功后，查看 fiddler 抓包的请求头信息，可以看出是 post 请求



1.4.3 如何找出需要的请求

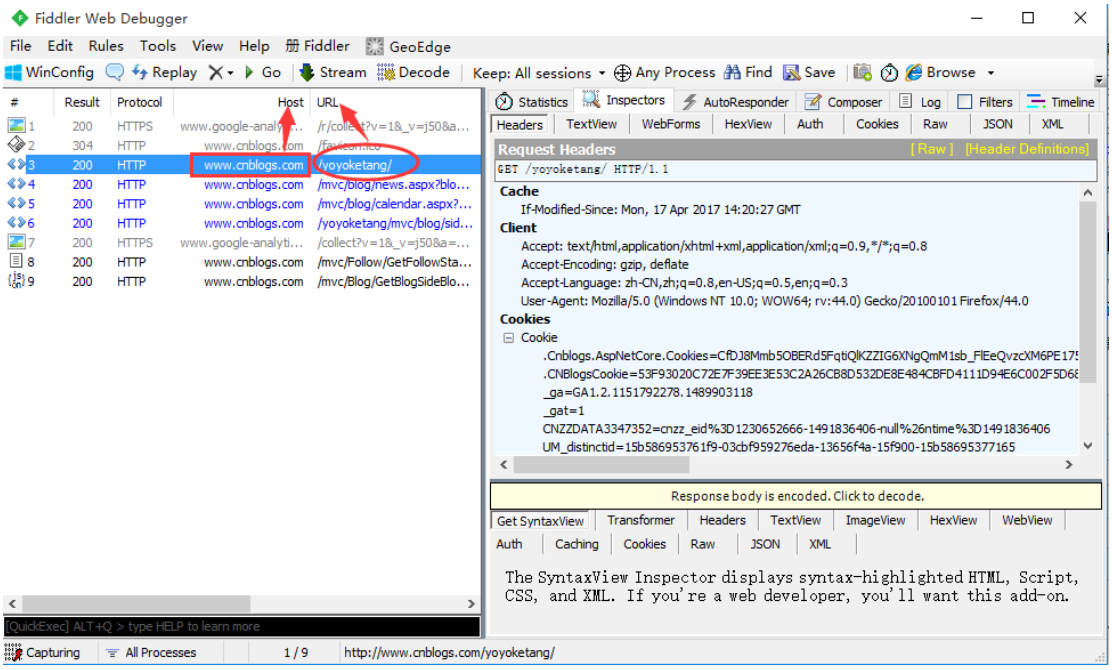
1. 打开 fiddler 后，左边会话框区域刷刷刷的很多请求，那么如何有效的找出自己需要的请求呢？

2. 首先第一步：清屏（cls），在左下角命令行输入 cls，清空屏幕



3. 第二步在浏览器输入 url 地址的时候，记住这个地址，如打开博客首页：http://www.cnblogs.com/yoyoketang/

在点击登录按钮的时候，不要做多与的操作了，然后查看 fiddler 会话框，这时候有好几个请求。



4. 如上图，红色框框这个地方就是 host 地址，红色圈圈地方就是 url 的路径（yoyoketang），也就是博客首页的地址了，那这个请求就是博客首页的请求了。

1.4.4 get 和 post 请求参数区别

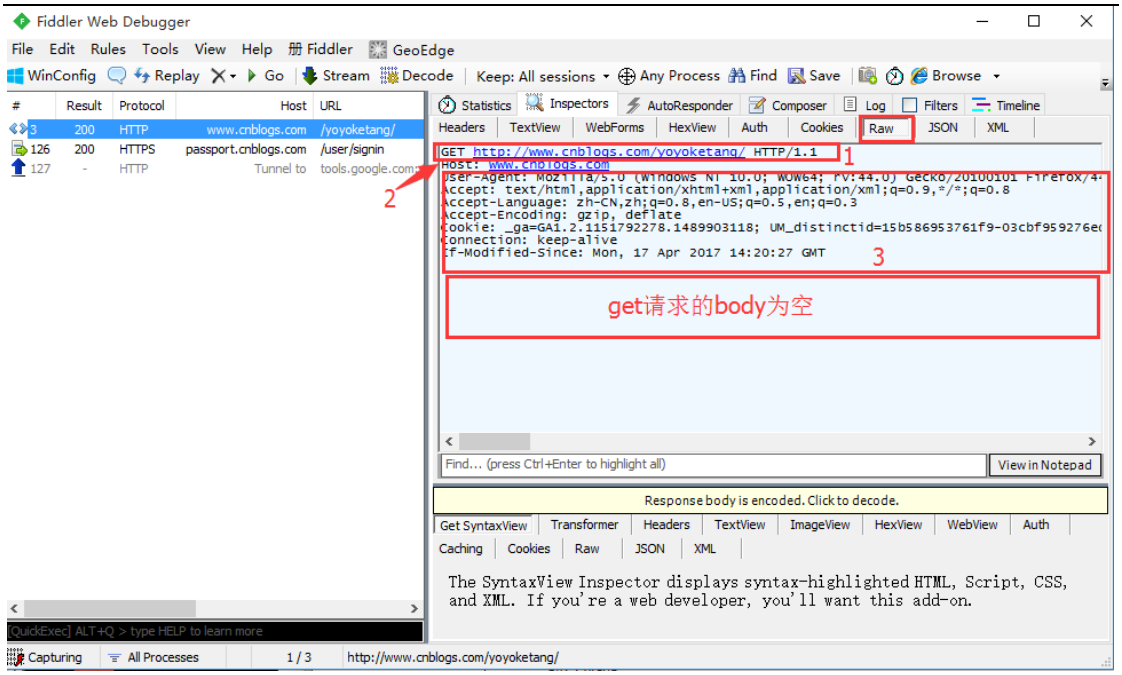
1. 关于 get 和 post 的功能上区别就不说了，大家自己查资料，这里主要从 fiddler 抓包的层面查看请求参数上的区别

2. get 请求的 Raw 参数查看，主要分三部分：

--第 1 部分是请求 url 地址

--第 2 部分是 host 地址

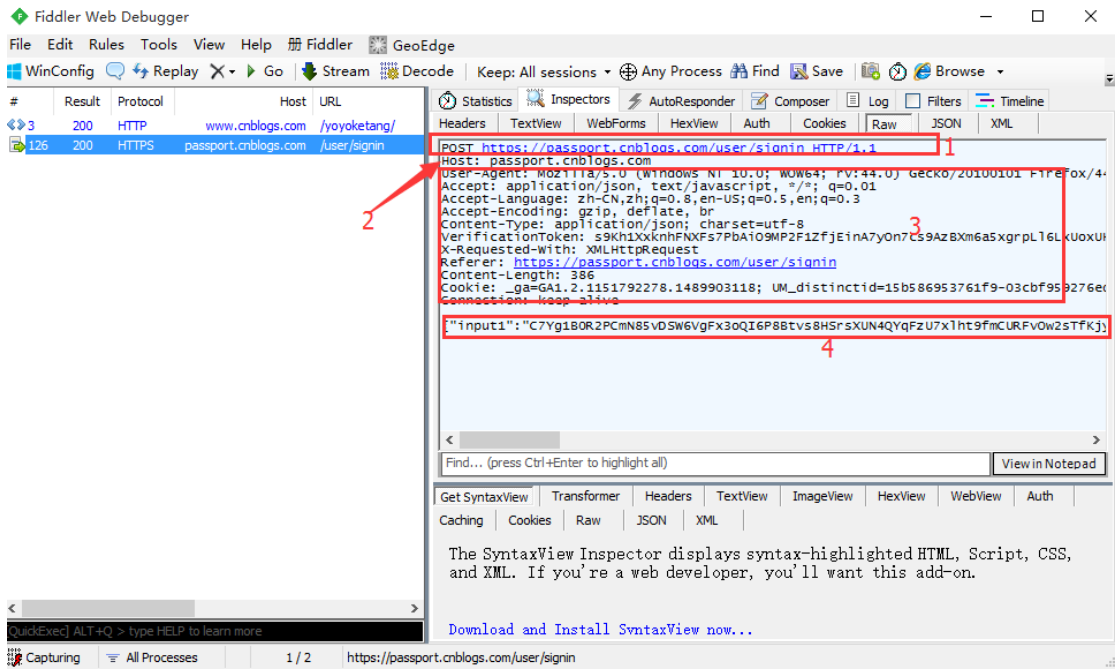
--第 3 部分是请求头部信息 header



3. 再查看博客登录请求的 Raw 信息，post 的信息分四部分。

--前面 3 块内容都一样，第 3 部分和第 4 部分中间会空一行

--第 4 部分内容就是 post 请求的请求 body（get 请求是没 body 的）



1.5 工具介绍

前言

本篇简单的介绍下 fiddler 界面的几块区域，以及各自区域到底是干什么用的，以便于各好的掌握这个工具

1.5.1 工具简介

1. 第一块区域是设置菜单，这个前面 2 篇都有介绍

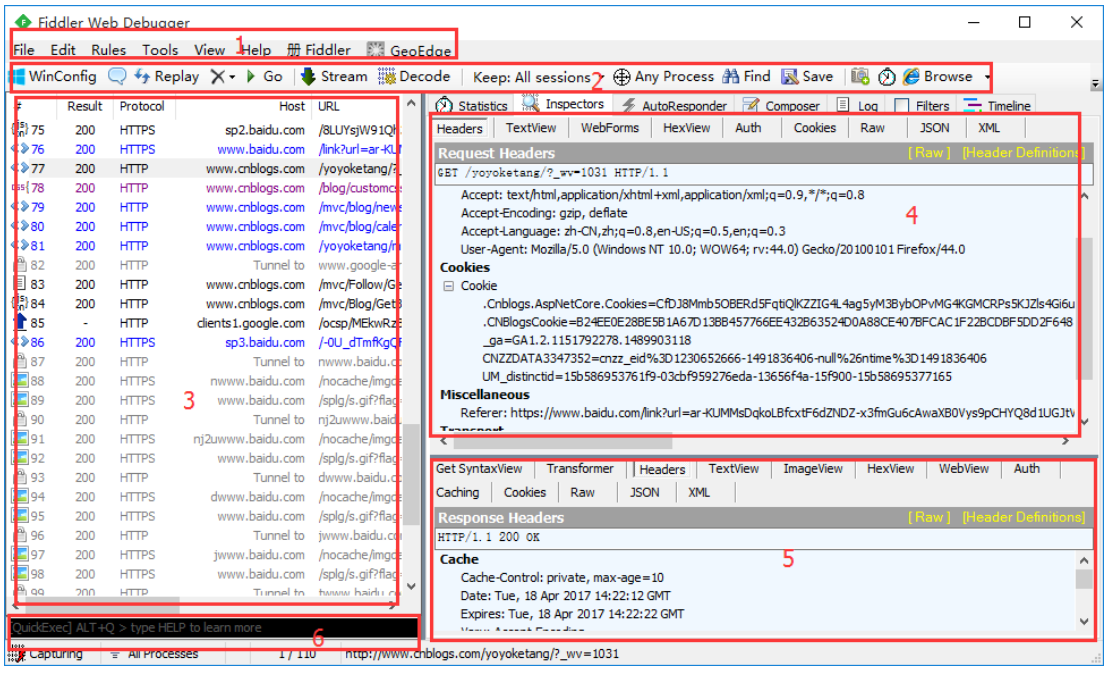
2. 第二块区域是一些快捷菜单，可以点下快捷功能键

3. 第三块左边是抓捕的请求会话列表，每一个请求就是一个会话

4. 第四块右边上方区域是 request 请求的详细信息，可以查看 headerd、cookies、raw、json 等

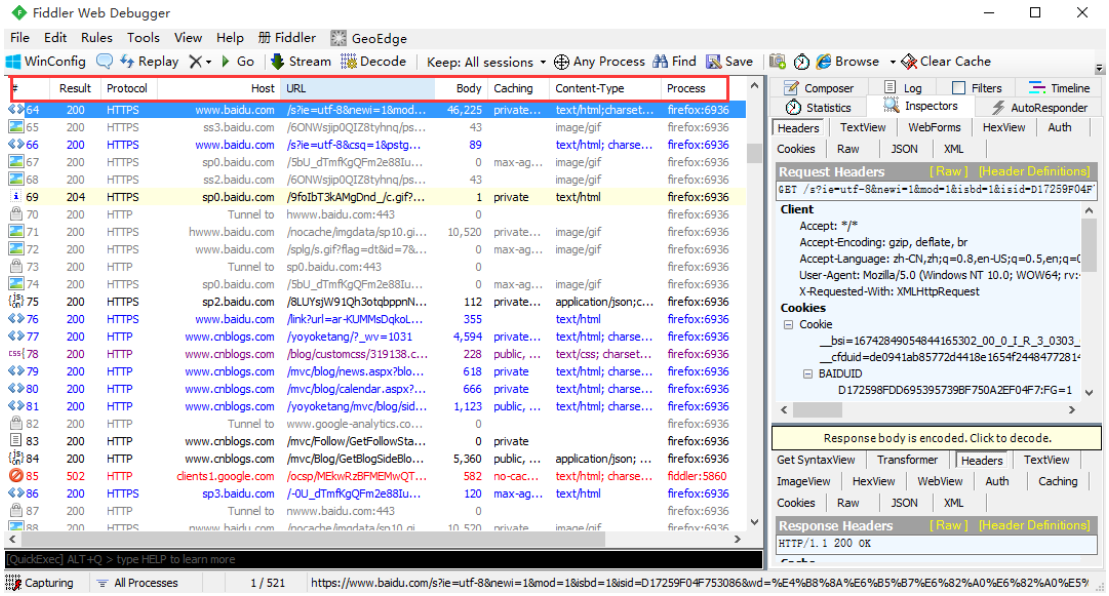
5. 第五块右边下方区域就是 response 信息，可以查看服务端返回的 json 数据或其它信息

6. 第六块区域左下角黑色的那块小地方，虽然很不起眼，容易被忽略掉，这地方是命令行模式，可以输入简单的指令如：cls，执行清屏的作用等



1.5.2 会话框

1.会话框主要查看请求的一些请求的一些基本信息，如#、result、protocol、host、url、body、 caching、content-type、process



2、#：会话框列表最左侧，#号这一栏是代表这个请求大概是什么内容，<>这个符号就是我们一般要测试的请求与响应的类型。

3.result:这里是服务器返回的代码，如

--200，请求 ok;2xx 一般是服务器接受成功了并处理

--3xx，重定向相关

--4xx,404 最常见的的就是找不到服务器,一般是请求地址有问题

--5xx,这个一般是服务器本身的错误

4.protocol：这个是协议类型，如 http、https

5.host：主机地址或域名

6.url:请求的路径

7.body：该条请求产生的数据大小

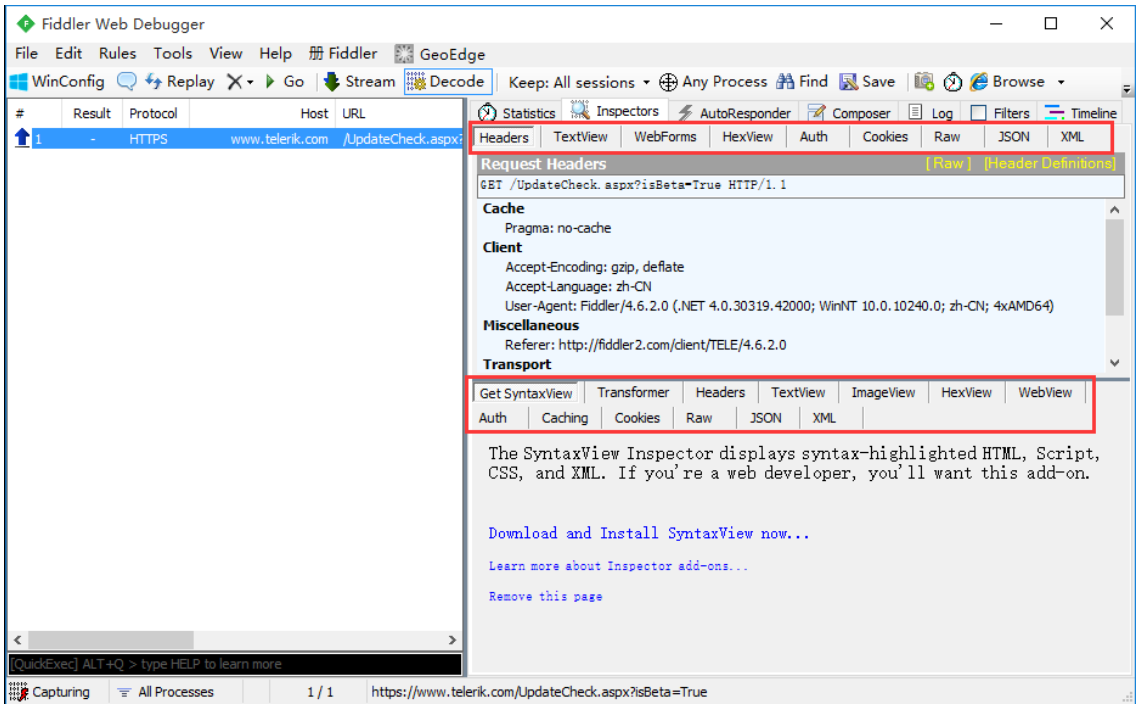
8.caching：缓存相关

9. content-type：连接类型

10. process:客户端类型

1.5.3 Request 和 Response

1. Request 是客户端发出去的数据，Response 是服务端返回过来的数据，这两块区域功能差不多



2. headers:请求头，这里包含 client、cookies、transport 等

3. webfroms：请求参数信息表格展示，更直观。可以直接该区域的参数

4. Auth:授权相关，如果现实如下两行，说明不需要授权

No Proxy-Authorization Header is present.

No Authorization Header is present.

5. cookies:查看 cookie 详情

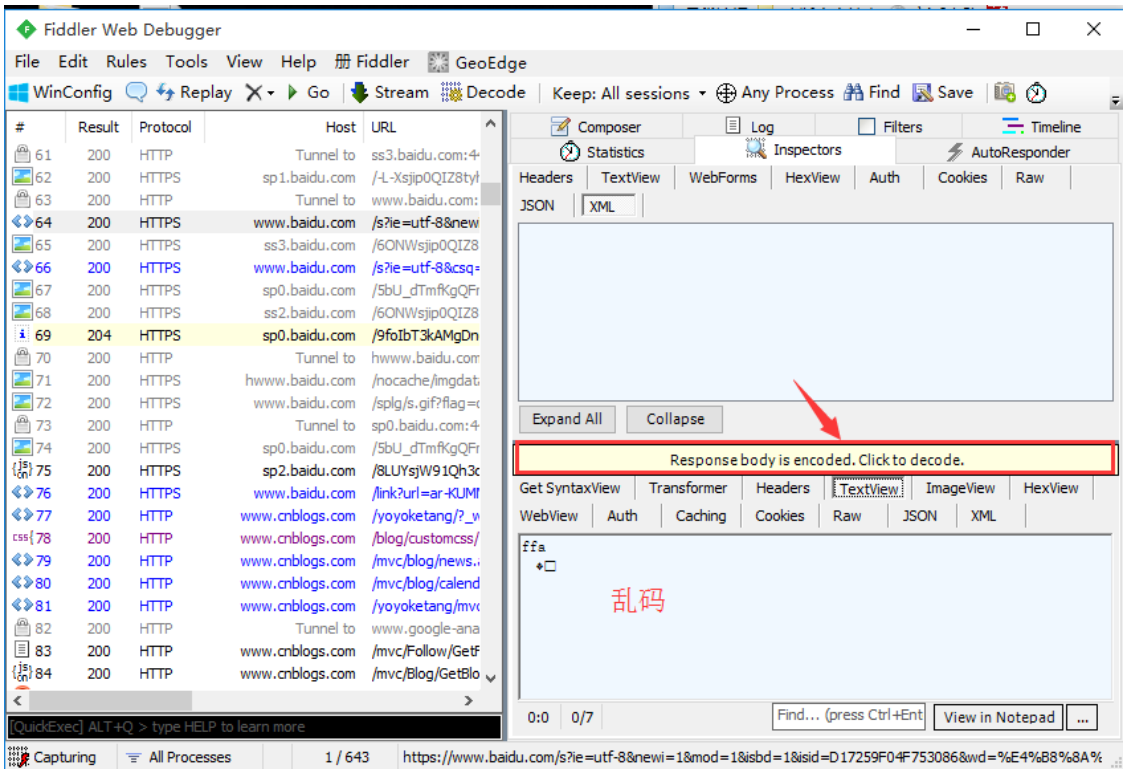
6. raw:查看一个完整请求的内容，可以直接复制

7. json:查看 json 数据

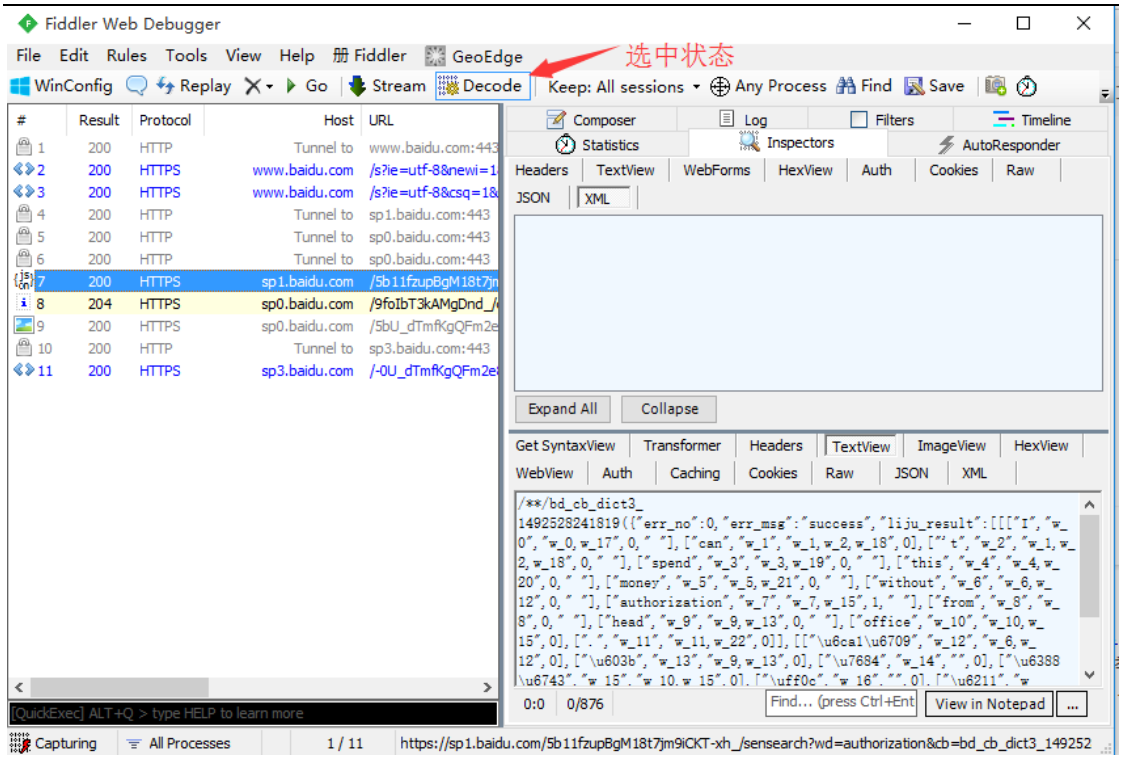
8. xml:查看 xml 文件的信息

1.5.4 decode 解码

1. 如果 response 的 TextView 区域出现乱码情况，可以直接点下方黄色区域解码



2. 也可以选中上方快捷菜单 decode，这样后面的请求都会自动解码了



1.6 接口测试 Composer

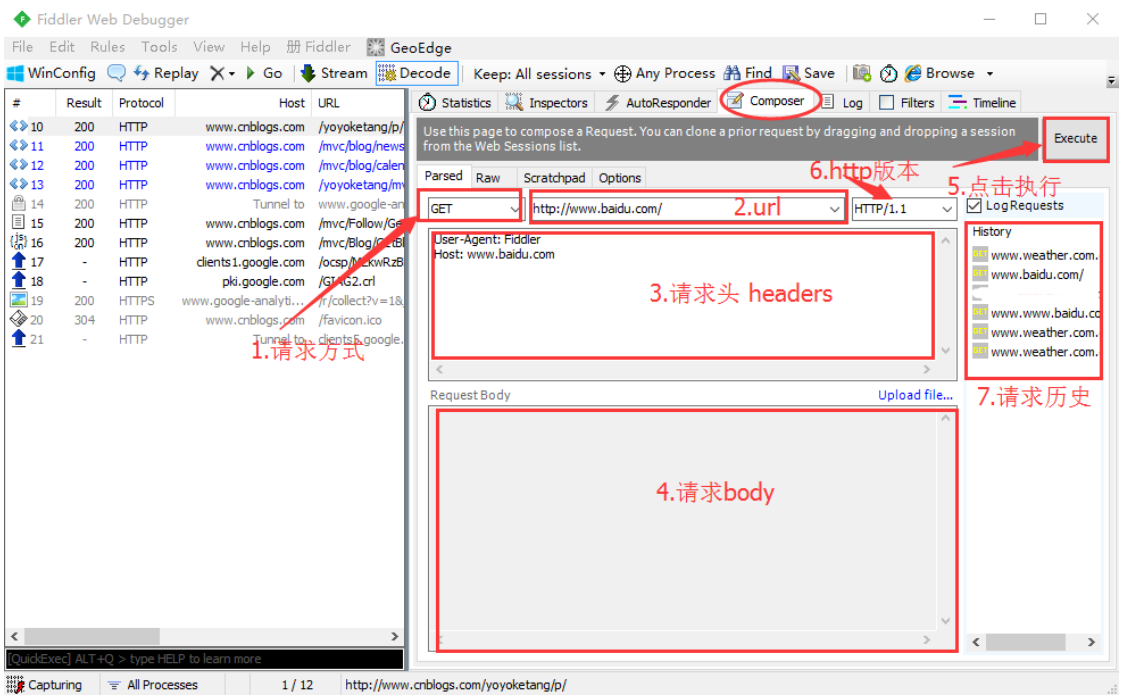
前言

Fiddler 最大的优势在于抓包，我们大部分使用的功能也在抓包的功能上，fiddler 做接口测试也是非常方便的。

对应没有接口测试文档的时候，可以直接抓完包后，copy 请求参数，修改下就可以了。

1.6.1 Composer 简介

点开右侧 Composer 区域，可以看到如下界面，就是测试接口的界面了



1. 请求方式：点开可以勾选请求协议是 get、post 等

2. url 地址栏：输入请求的 url 地址

3. 请求头：第三块区域可以输入请求头信息

4. 请求 body：post 请求在此区域输入 body 信息

5. 执行：Execute 按钮点击后就可以执行请求了

6. http 版本：可以勾选 http 版本

7. 请求历史：执行完成后会在右侧 History 区域生成历史记录

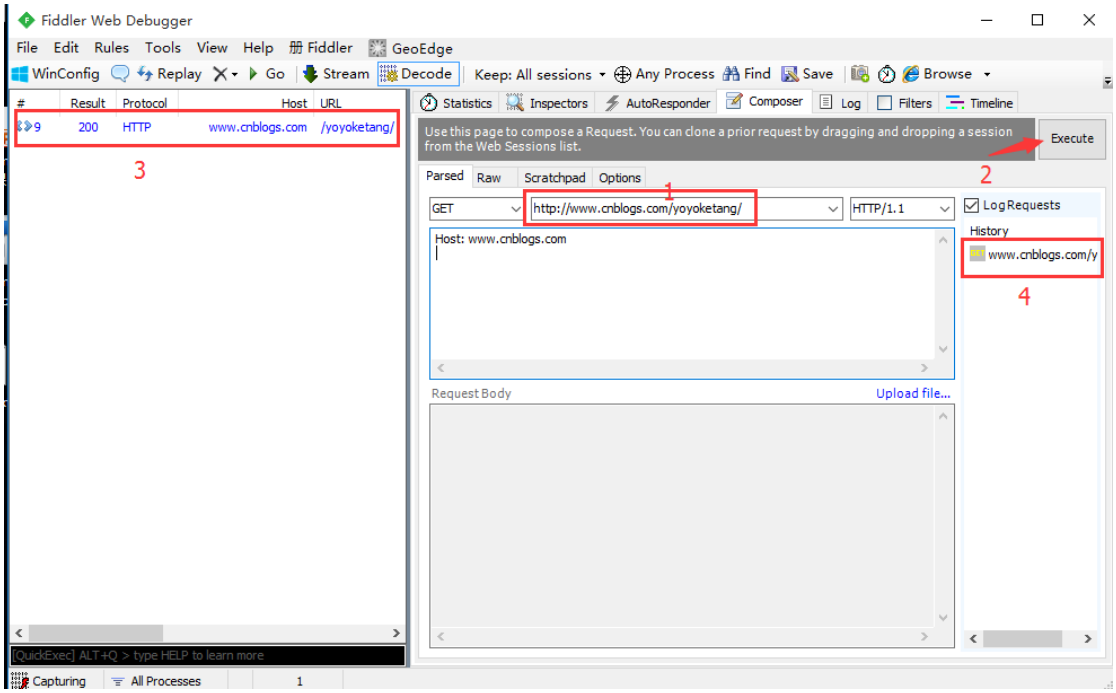
1.6.2 模拟 get 请求

1. 在 Composer 区域地址栏输入博客首页：http://www.cnblogs.com/yoyoketang/

2. 选择 get 请求，点 Execute 执行，请求就可以发送成功啦

3. 请求发送成功后，左边会话框会生成一个会话记录，可以查看抓包详情

4. 右侧 history 区域会多一个历史请求记录



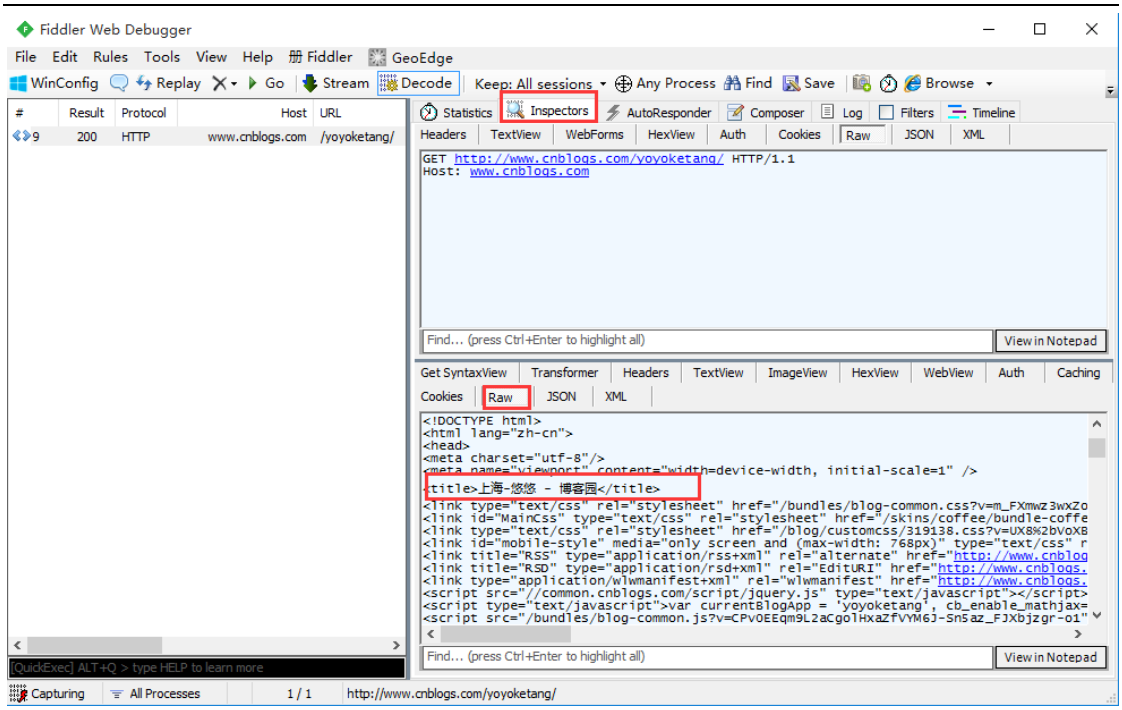
5.会话框选中该记录，查看测试结果：

--选中该会话，点开 Inspectors

--response 区域点开 Raw 区域

--Raw 查看的是 HTML 源码的数据

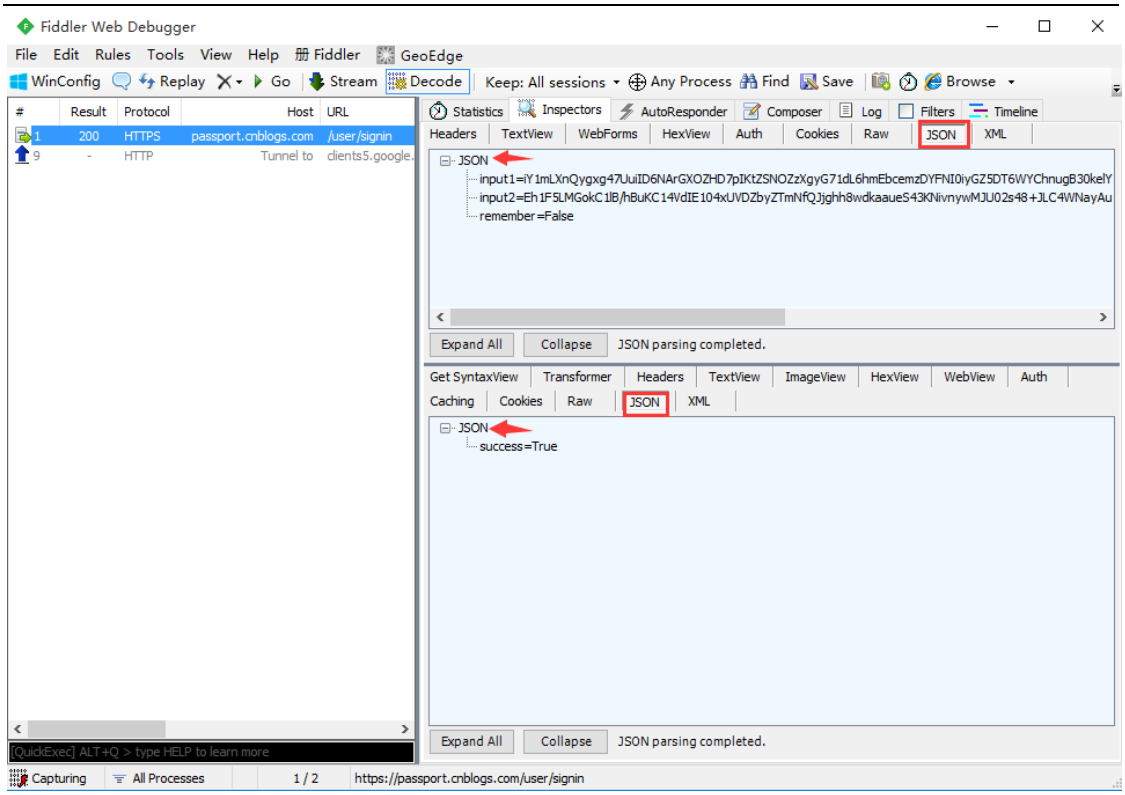
--也可以点 WebView，查看返回的 web 页面数据



1.6.3 Json 数据

1. 有些 post 的请求参数和返回参数是 Json 格式的，如博客园的登录请求：https://passport.cnblogs.com/user/signin

2. 查看 json 数据如下图



1.6.4 模拟 post 请求

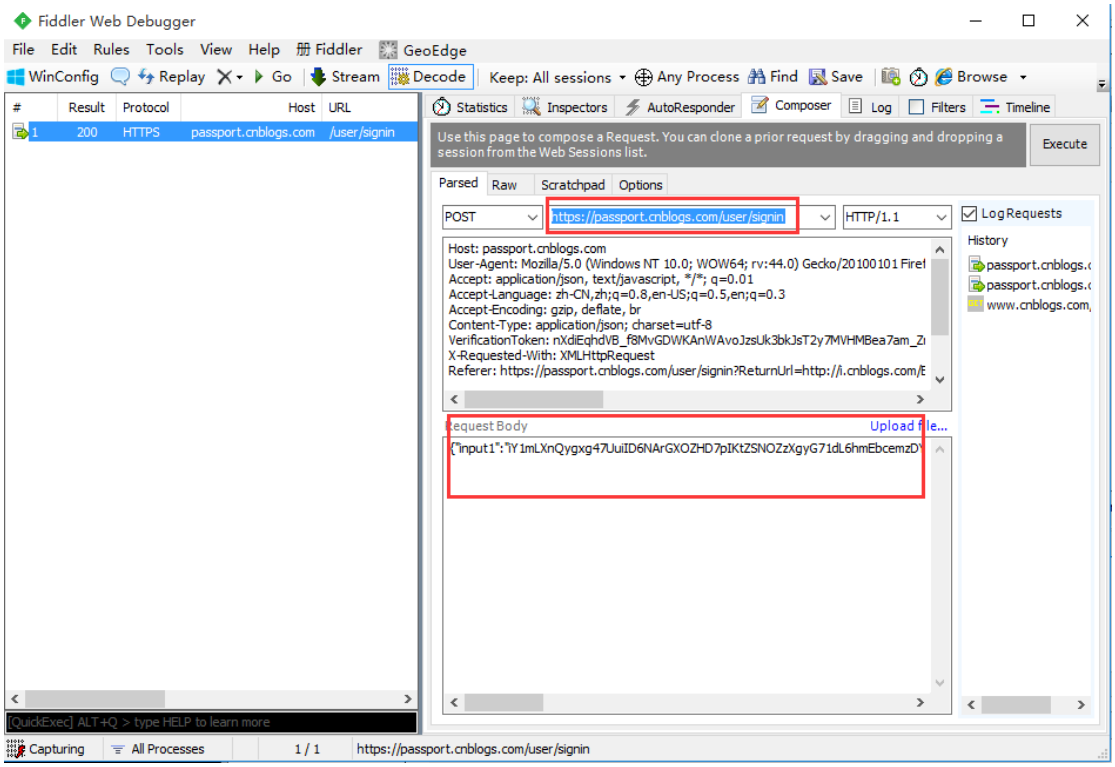
1. 请求类型勾选 post

2. url 地址栏输入对应的请求地址

3. body 区域写登录的 json 参数

4. header 请求头区域，可以把前面抓包的数据 copy 过来

（注意，有些请求如果请求头为空的话，会请求失败的）



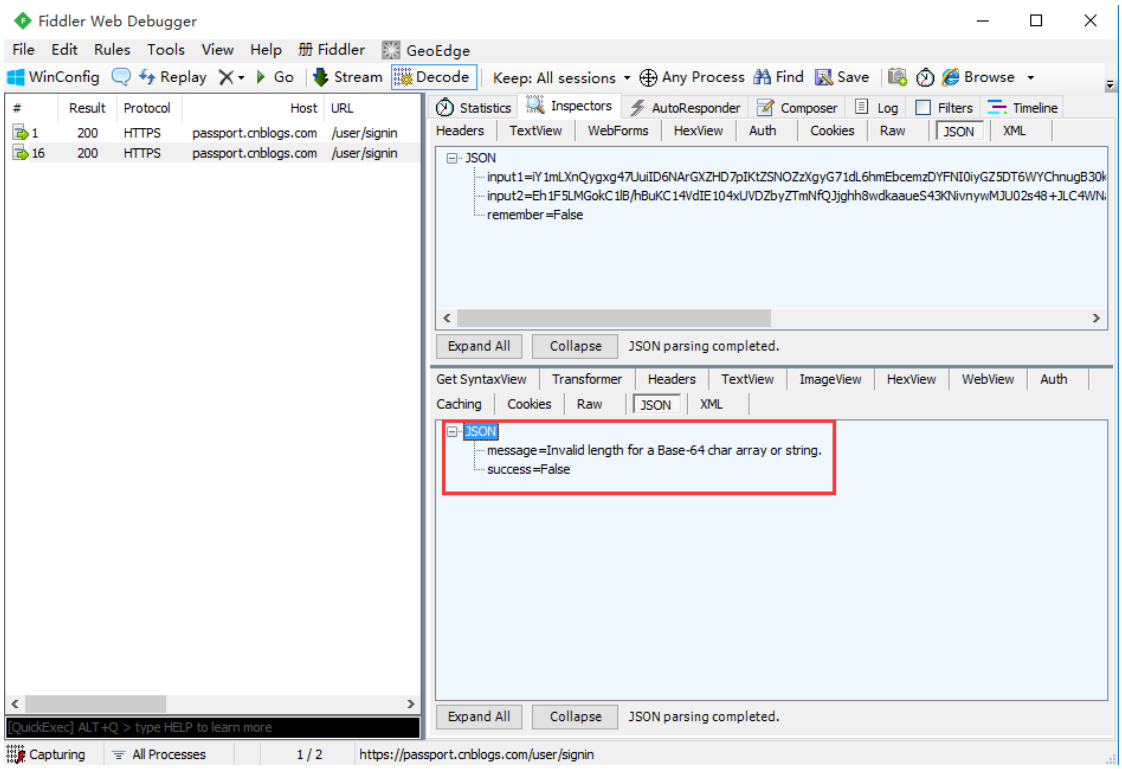
5. 执行成功后查看测试结果：

--执行成功如第三所示的图，显示 success=True

--执行失败如下图所示，显示

message=Invalid length for a Base-64 char array or string.

success=False



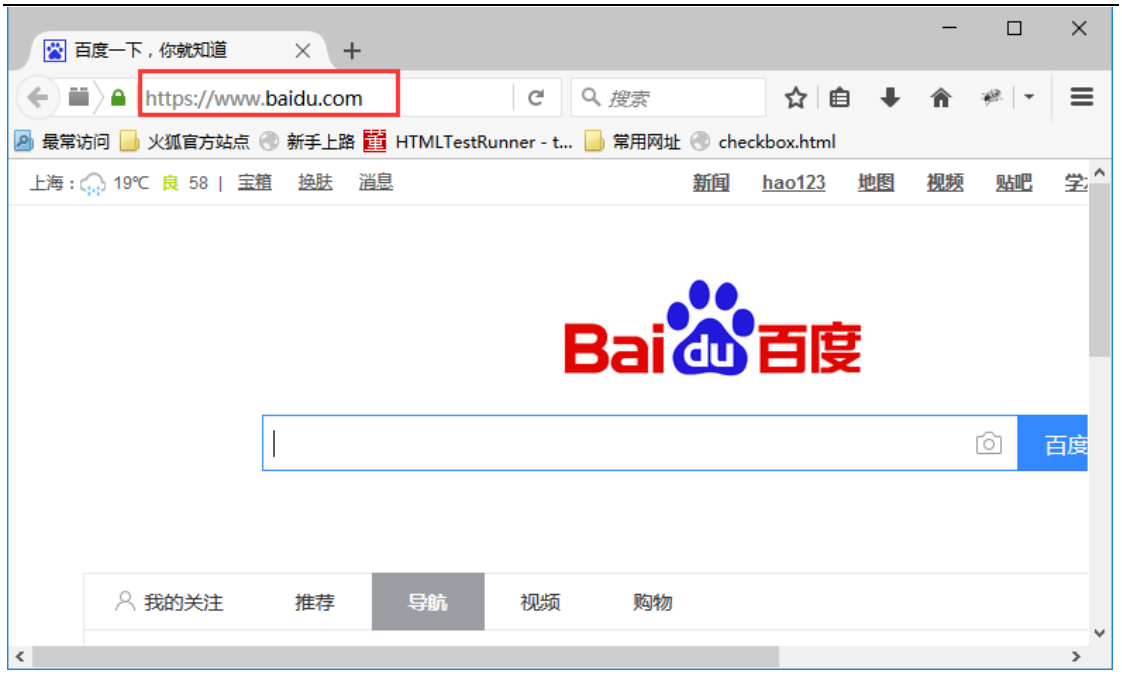
1.7 get 请求

前言

上一篇介绍了 Composer 的功能，可以模拟 get 和 post 请求，get 请求有些是不带参数的，这种比较容易，直接放到 url 地址栏就行。有些 get 请求会带有参数，本篇详细介绍 url 地址格式。

1.7.1 url 详解

1.url 就是我们平常打开百度在地址栏输入的：https:www.baidu.com,如下图，这个是最简单的 url 地址，打开的是百度的主页



2. 再看一个稍微复杂一点的 url，在百度输入框输入：上海悠悠博客园

3. 查看 url 地址栏，对比之前的百度首页 url 地址，后面多了很多参数。当然最主要的参数是:wd=上海悠悠博客园（后面的一大串可以暂时忽略）。

4. 那么问题来了，这些参数有什么作用呢？

可以做个简单的对比，在地址栏分别输入：

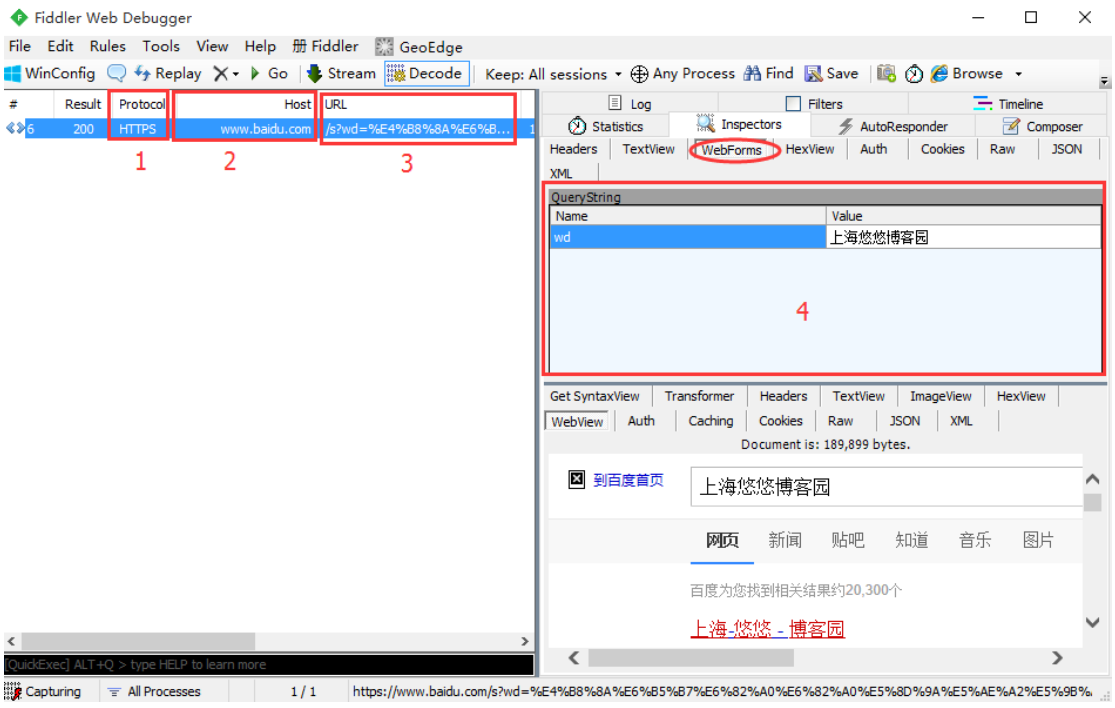
https:www.baidu.com

https://www.baidu.com/s?wd=上海悠悠博客园

对比打开的页面有什么不一样，现在知道作用了吧，也就是说这个多的"/s？wd=上海悠悠博客园"就是搜索的结果页面

1.7.2 url 解析

1.以"https://www.baidu.com/s?wd=上海悠悠博客园"这个 url 请求的抓包为例



2.那么一个完整的 url 地址，基本格式如下：

https://host:port/path?xxx=aaa&ooo=bbb

--http/https：这个是协议类型，如图中 1 所示

--host:服务器的 IP 地址或者域名，如图中 2 所示

--port:HTTP 服务器的默认端口是 80，这种情况下端口号可以省略。

如果使用了别的端口，必须指明，例如：192.168.3.111:8080，这里的 8080 就是端口

--path:访问资源的路径,如图中 3 所示/s（图中 3 是把 path 和请求参数放一起了）

--？:url 里面的？这个符号是个分割线，用来区分问号前面的是 path，问号后面的是参数

--url-params:问号后面的是请求参数，格式：xxx=aaa，如图 4 区域就是请求参数

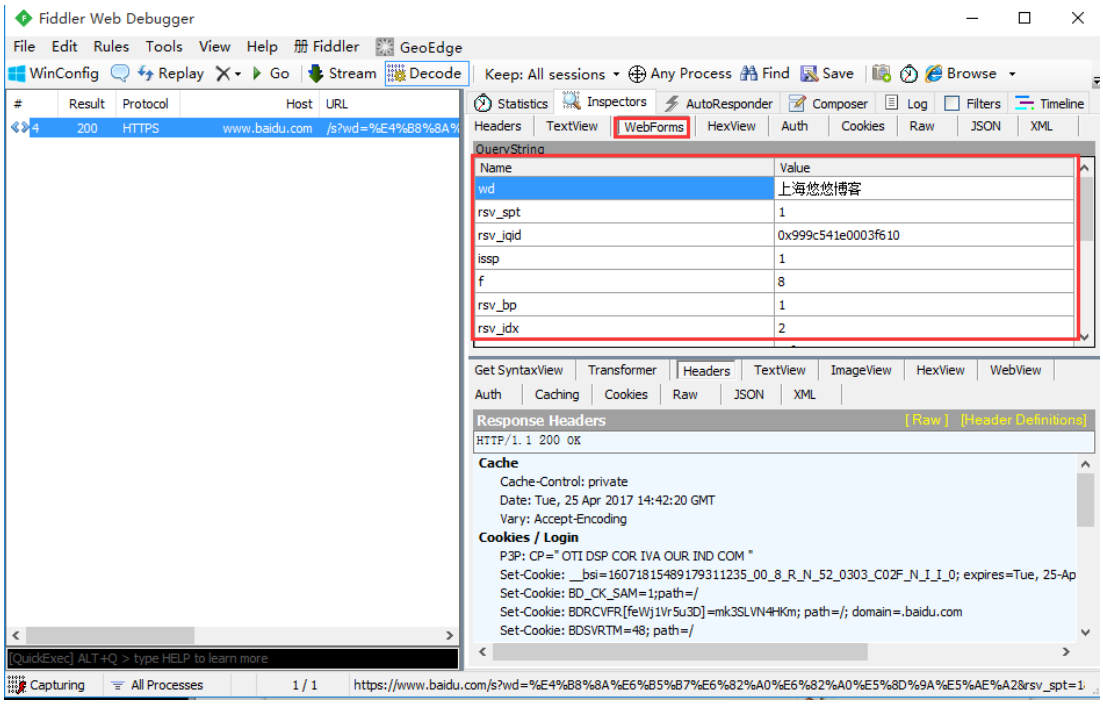
--&：多个参数用&符号连接

1.7.3 请求参数（params）

1.在 url 里面请求参数一般叫 params，但是我们在 fiddler 抓包工具看到的参数是：QueryString

2.QueryString 是像服务端提交的参数，其实跟 params 是一个意思，每个参数对应的都有 name 和 value 值

3.多个参数情况如下

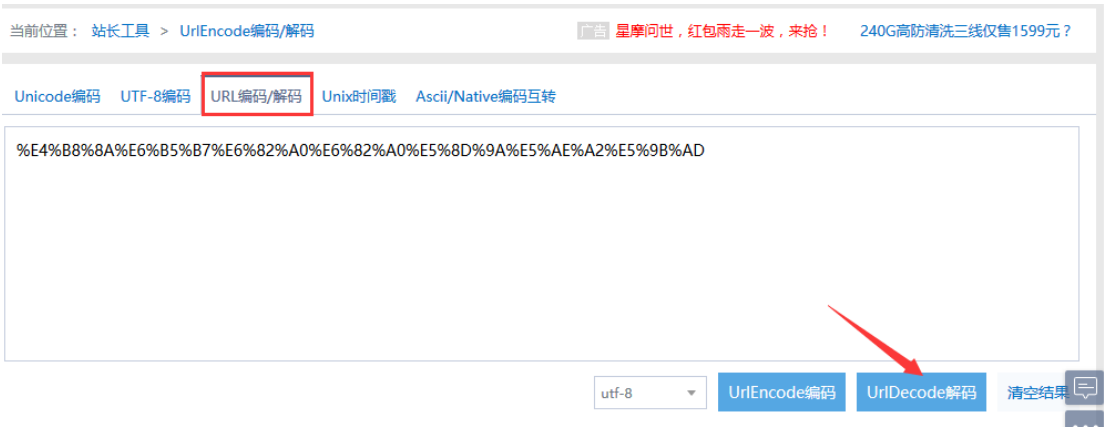


1.7.1 UrlEncode 编码

1.如果 url 地址的参数带有中文的，一般在 url 里面会是这样的，如第二点里的 wd=%E4%B8%8A%E6%B5%B7%E6%...

像看到%E4 这种编码的就是经过 url 编码过的，需要解码就能看到是什么中文了

2.用 urlencode 在线编码/解码工具，地址：
http://tool.chinaz.com/tools/urlencode.aspx



个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

1.8 post 请求（body）

前言上一篇讲过 get 请求的参数都在 url 里，post 的请求相对于 get 请求多了个 body 部分，本篇就详细讲解下 body 部分参数的几种形式。

1.8.1 body 数据类型

常见的 post 提交数据类型有四种：

1.第一种：application/json：这是最常见的 json 格式，也是非常友好的深受小伙伴喜欢的一种，如下

```
{"input1":"xxx","input2":"ooo","remember":false}
```

2.第二种：application/x-www-form-urlencoded：浏览器的原生 form 表单，如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数

input1=xxx&input2=ooo&remember=false

3.第三种：multipart/form-data:这一种是表单格式的，数据类型如下

------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="text"

title

------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="file"; filename="chrome.png"

Content-Type: image/png

PNG ... content of chrome.png ...

------WebKitFormBoundaryrGKCBY7qhFd3TrwA--

4.第四种：text/xml:这种直接传的 xml 格式

<!--?xml version="1.0"?-->

<methodcall>

<methodname>examples.getStateName</methodname>

<params>

<param>

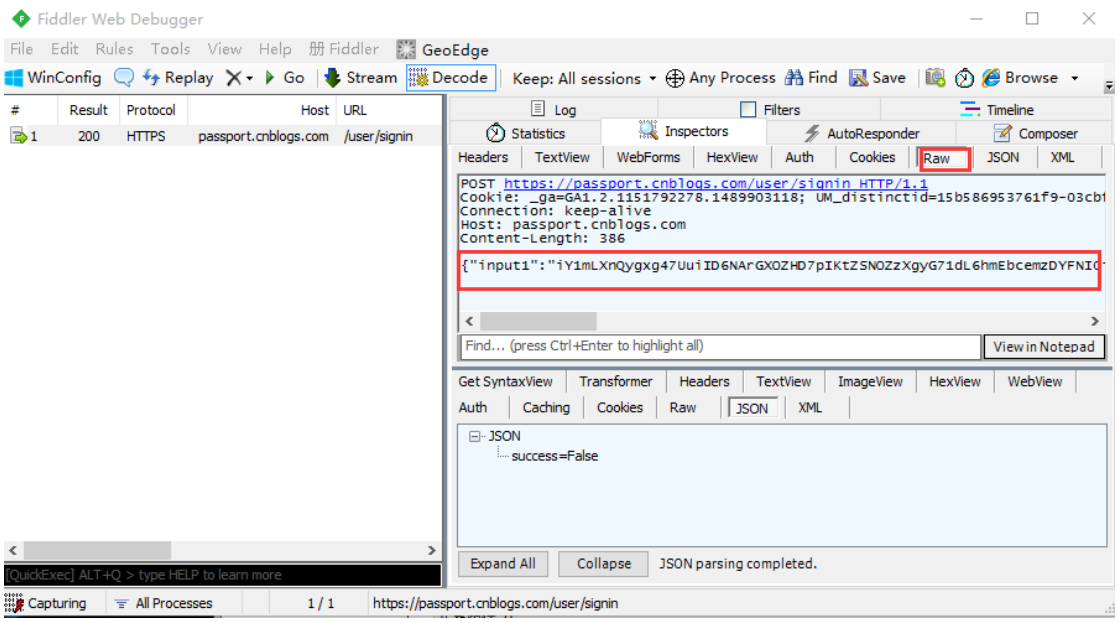
<value><i4>41</i4></value>

</params>

</methodcall>

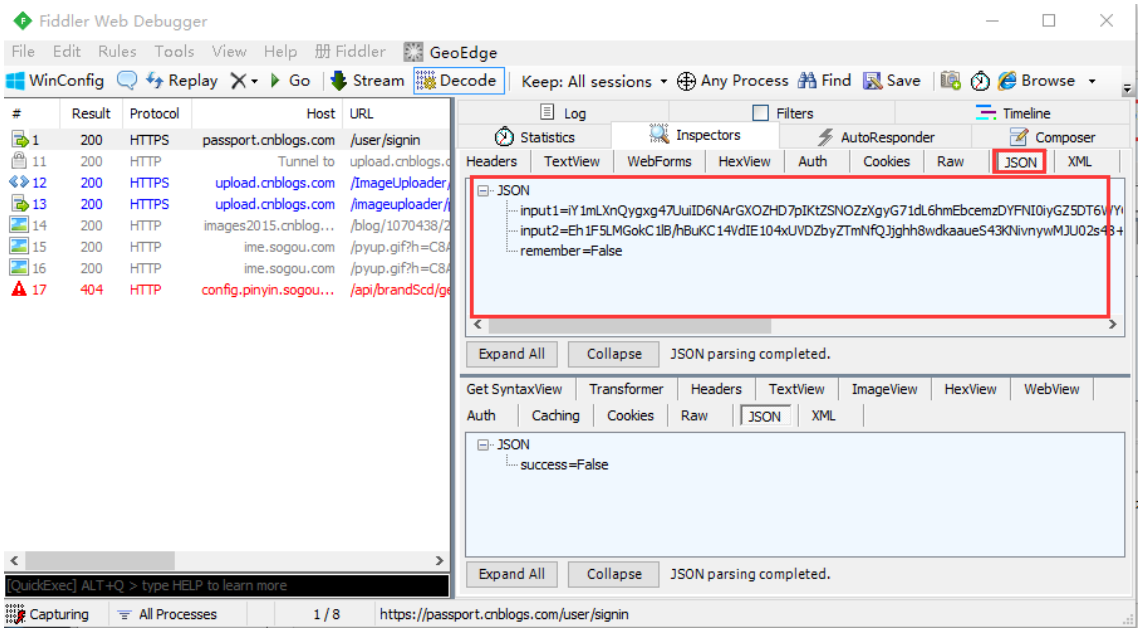
1.8.2 json 格式

1. 打开博客园的登录页面，输入账号密码后抓包，查看 post 提交数据，点开 Raw 查看整个请求的原始数据



2. 前面讲过 post 的请求多一个 body 部分，上图红色区域就是博客园登录接口的 body 部分，很明显这种格式是前面讲到的第一种 json 格式

3. 查看 json 格式的树状结构，更友好，可以点开 JSON 菜单项



4.查看这里的 json 数据，很明显传了三个参数：

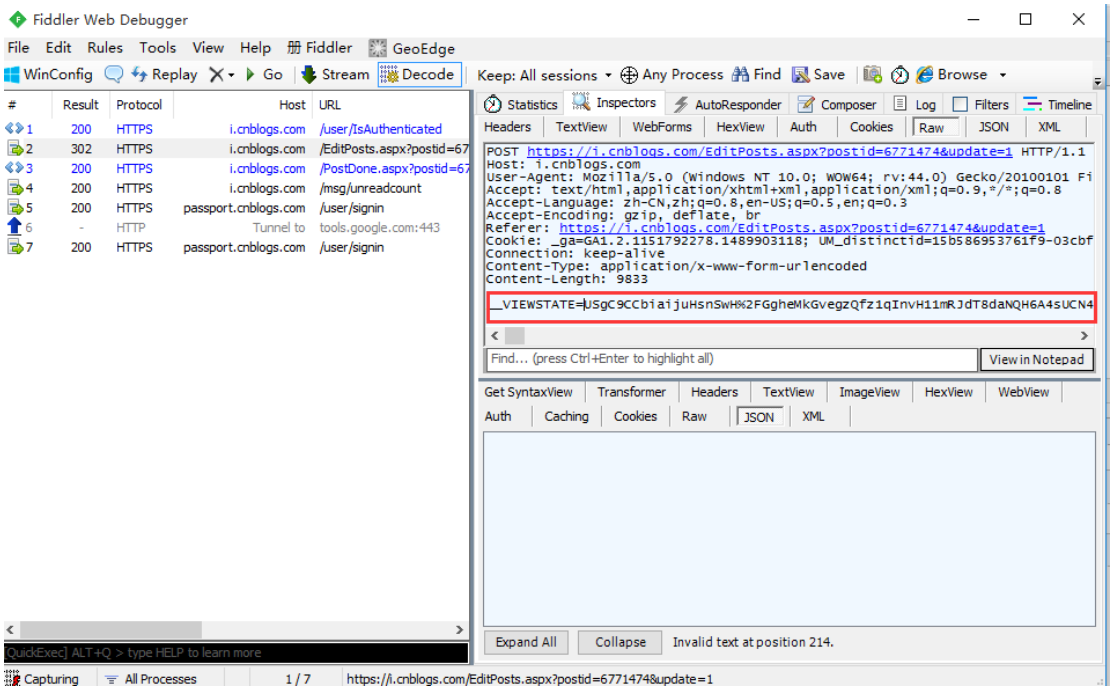
--input1:这个是登录的账号参数（加密过）

--input2:这个是登录的秘密参数（加密过）

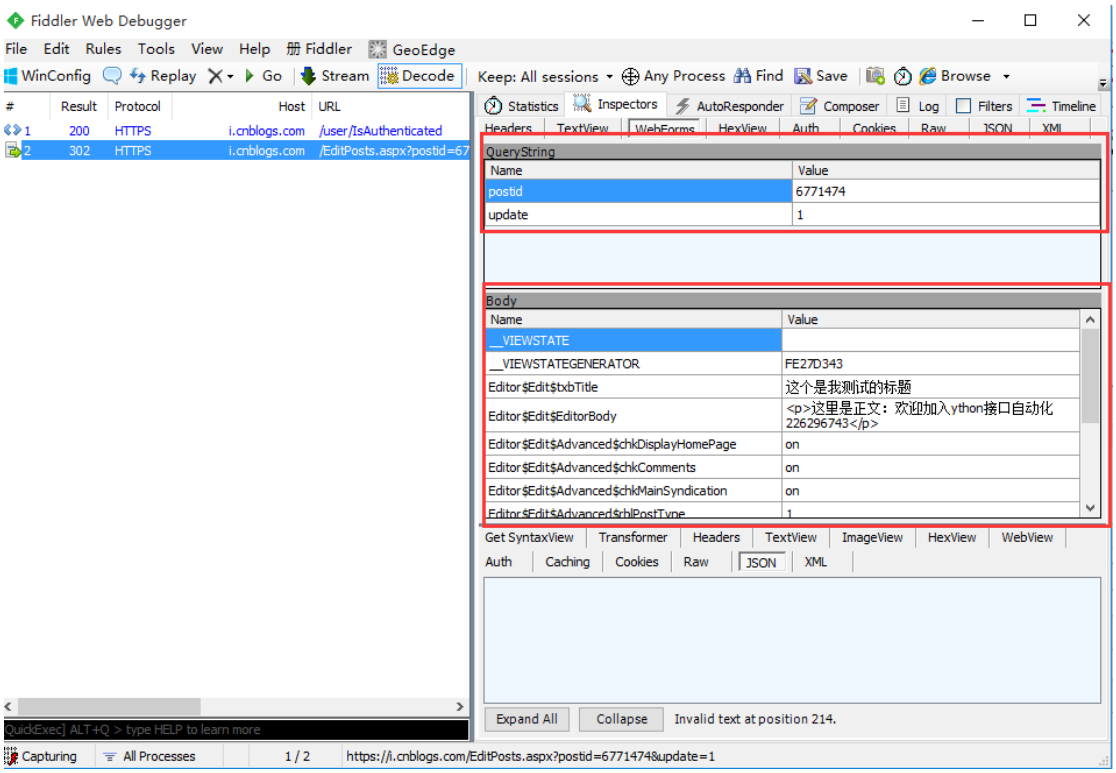
--remember:这个是登录页面的勾选是否记住密码的选项，False 是不记住，True 是记住

1.8.3 x-www-form-urlencoded

1.登录博客园后，打开新随笔，随便写一个标题和一个正文后保存，抓包数据如下



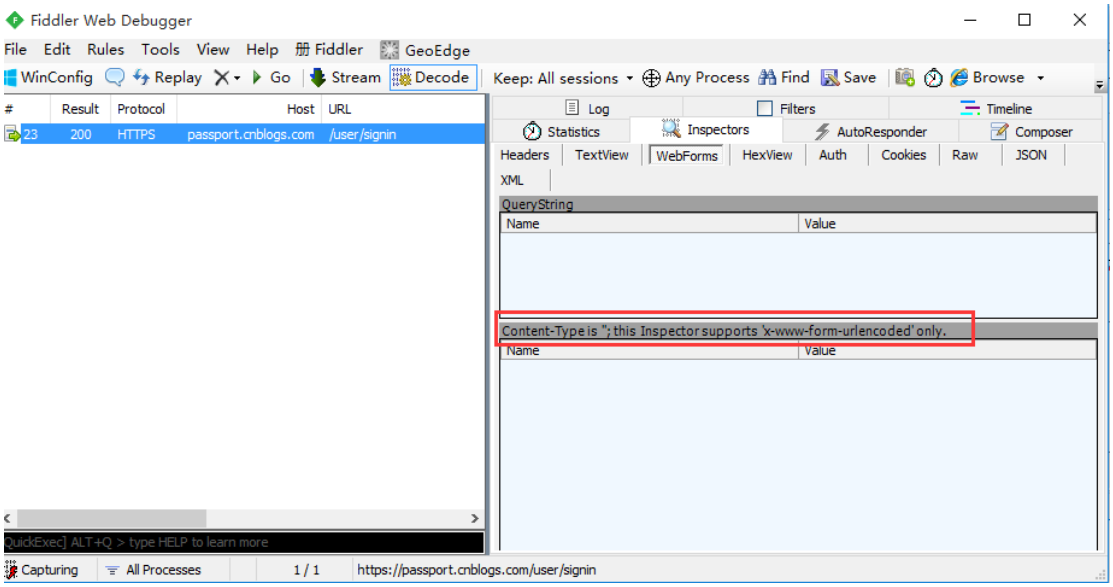
2.如上图的这种格式，很明显就属于第二种了，这种类型的数据查看，在 WebFrom 里面查看了



3. 上面红色框框的 Query String 是 url 里面的参数，下面红色框框的 body 部分就是这次 post 提交的 body 参数部分了

1.8.4 WebFrom

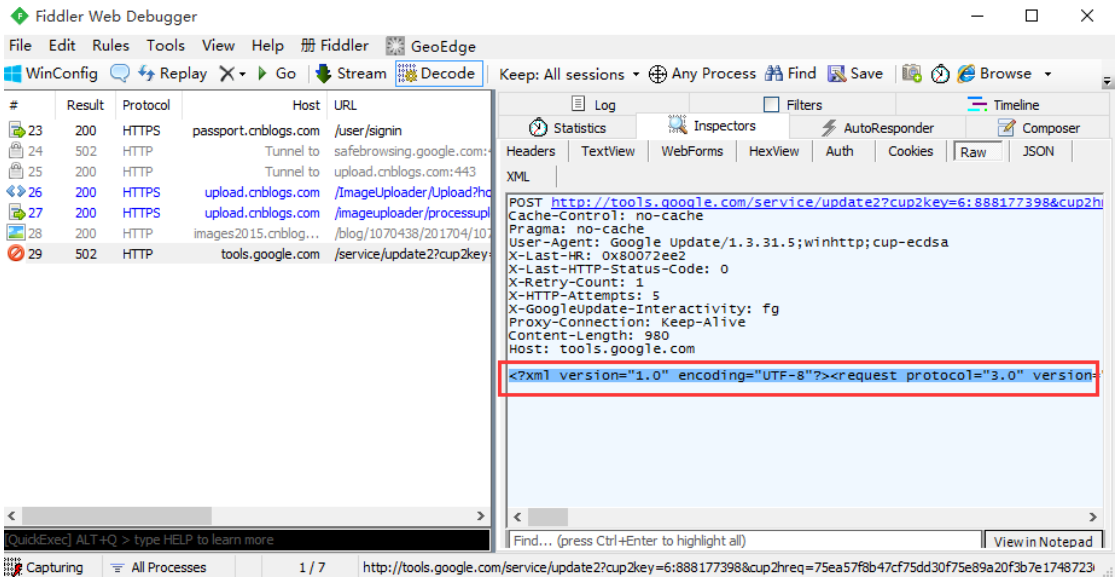
1. 为什么登录请求的 WebFrom 的 body 部分为空呢？



2. 看上图红色框框的显示：这里只支持 application/x-www-form-urlencoded 这种格式的 body 参数，也就是说 json 格式的，需要在 JOSN 这一栏查看了

1.8.5 xml

1. 如果遇到 text/xml 这种格式的 body，那就如下图这样的了



1.9 打断点 bpu

前言

先给大家讲一则小故事，在我们很小的时候是没有手机的，那时候跟女神聊天都靠小纸条。某屌丝 A 男对隔壁小王的隔壁女神 C 倾慕已久，于是天天小纸条骚扰，无奈中间隔着一个小王，这样小王就负责传小纸条了。有一天小王忍不住偷偷打开 A 男表白的纸条，把里面内容改了下，改成了：我的同桌小王喜欢你。最后女神 C 和小王走在了一起。。。这是一个悲伤的故事！

1.9.1 断点

1. 为什么要打断点呢？

比如一个购买的金额输入框，输入框前端做了限制 100-1000，那么我们测试的时候，需要测试小于 100 的情况下。很显然前端只能输入大于 100 的。这是我们可以先抓到接口，修改请求参数，绕过前端，传一个小于 100 的数，检查服务端的功能是否 OK。

也就是说接口测试其实是不需要管前端的，主要测后端的功能。Fiddler 作为代理服务器的作用其实就相当于上面故事里面的小王，传纸条的作用，Fiddler(小王)修改了请求参数（小纸条），是为了验证服务端功能（女神 C）。

2.Fiddler 可以修改以下请求

--Fiddler 设置断点，可以修改 HTTP 请求头信息，如修改 Cookie，User-Agent 等

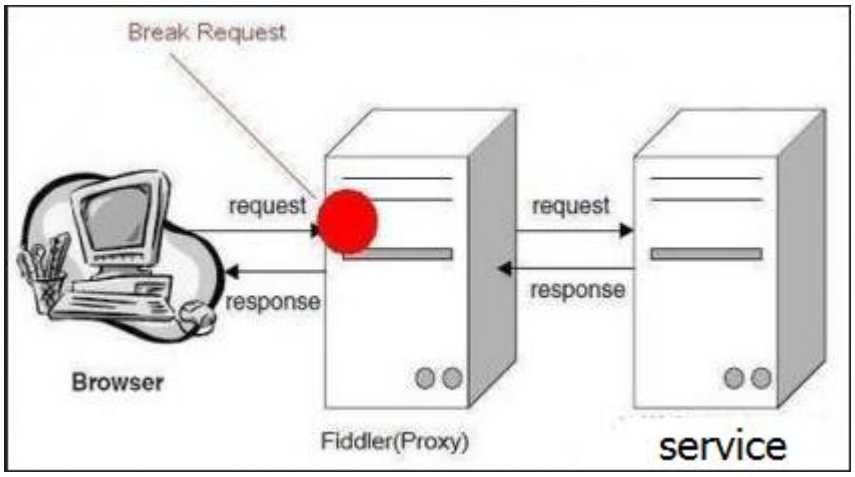
--可以修改请求数据，突破表单限制，提交任意数字，如充值最大 100,可以修改成 10000

--拦截响应数据，修改响应体，如修改服务端返回的页面数据

1.9.2 断点的两种方式

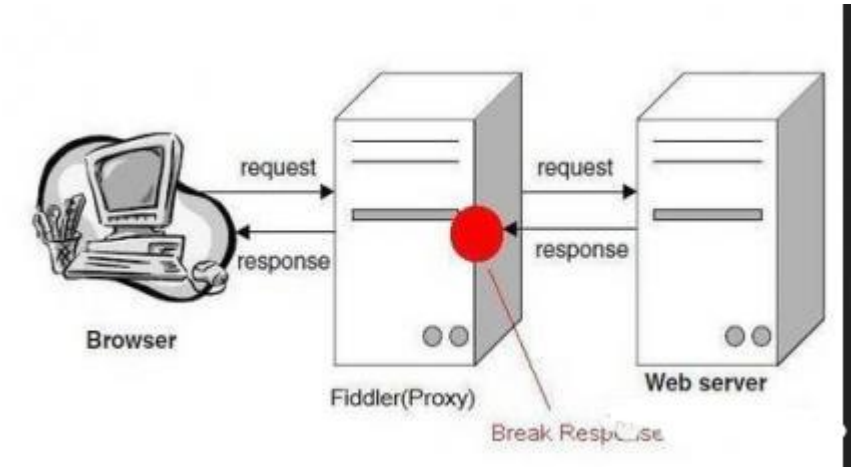
1.before response：这个是打在 request 请求的时候，未到达服务器之前

--屌丝 A 传给小王的时候，小王在这个时候拦截了小纸条，未传给女神 C



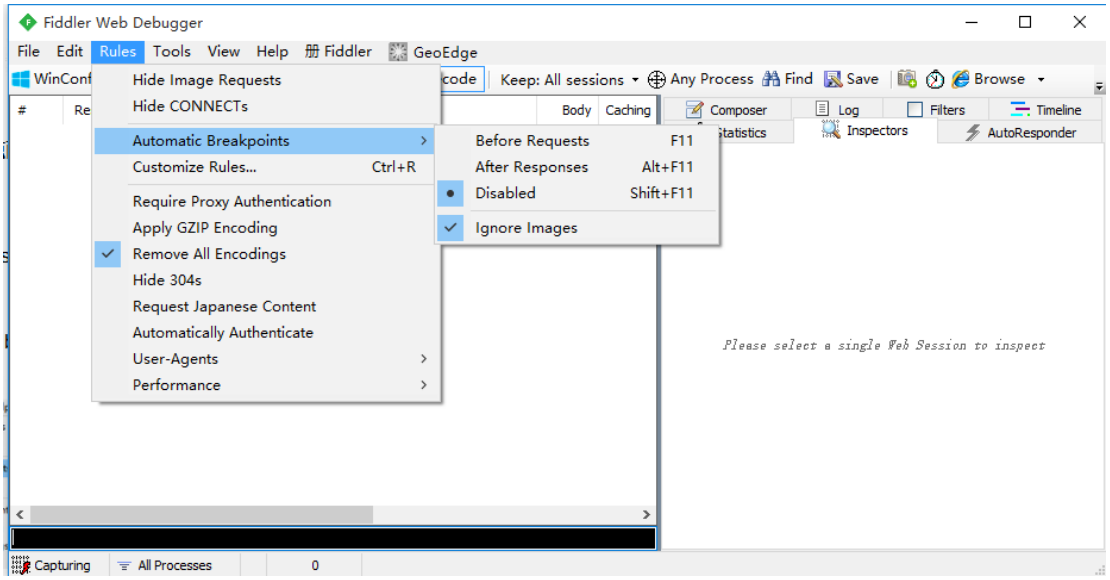
2.after response：也就是服务器响应之后，在 Fiddler 将响应传回给客户端之前。

--女神 C 回了小纸条，小王拿到后拦截了，未传给屌丝 A

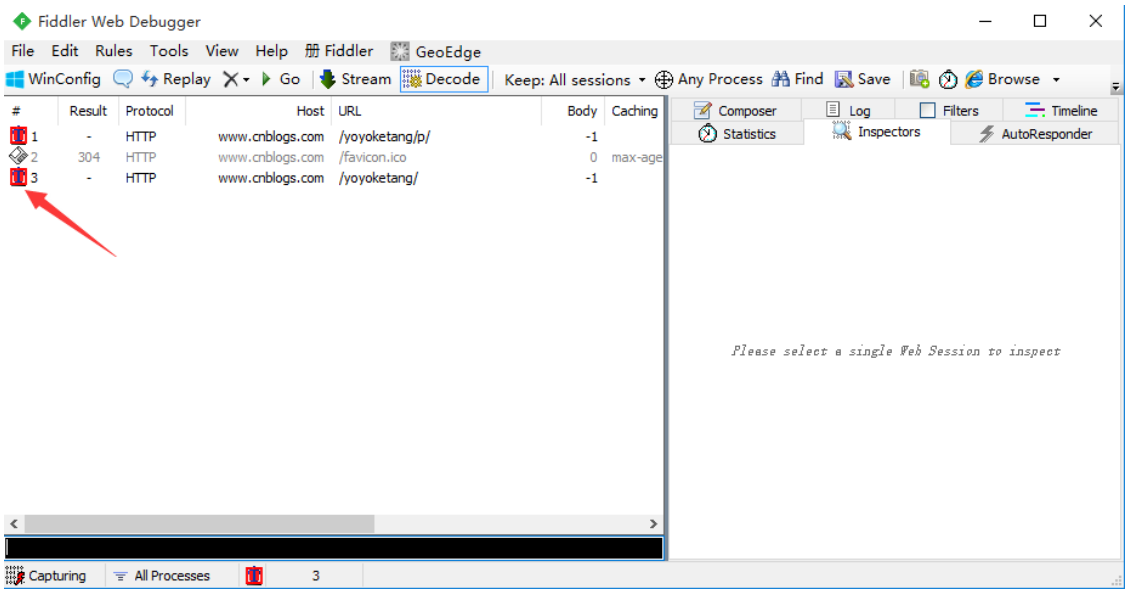


1.9.3 全局断点

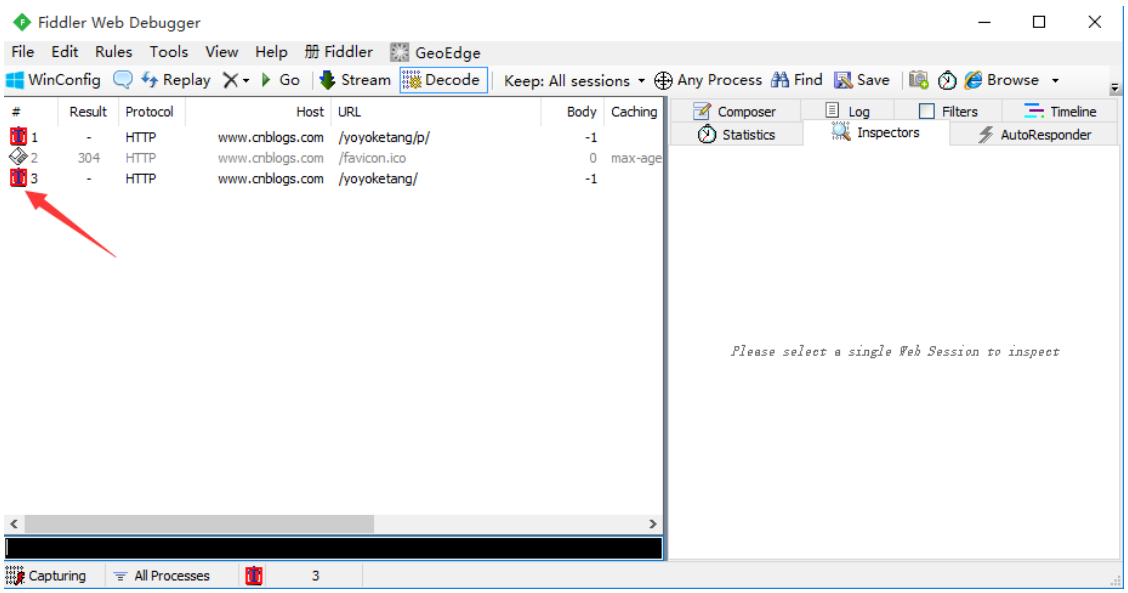
1.全局断点就是中断 fiddler 捕获的所有请求，先设置下，点击 rules-> automatic breakpoint ->before requests



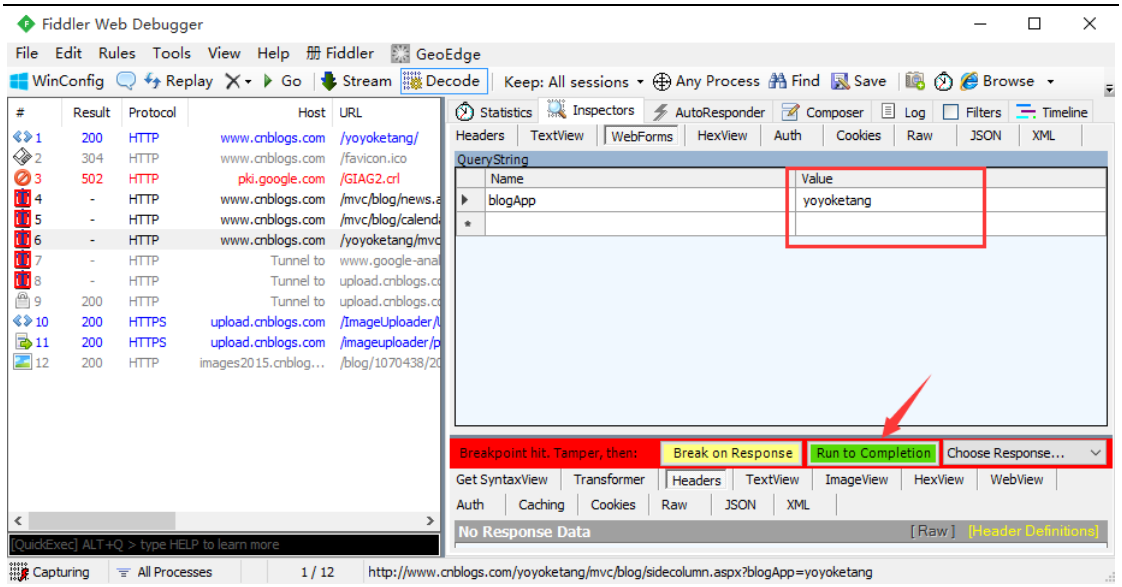
2.选中 before requests 选项后，打开博客园首页：http://www.cnblogs.com/yoyoketang/，看到如下 T 的标识，说明断点成功



3. 打完断点后，会发现所有的请求都无法发出去了，这时候，点下 Go 按钮，就能走下一步了



4. 找到需要修改的请求后，选中该条会话，右侧打开 WebFroms,这时候里面的参数都是可以修改的了



5. 修改之后点 Run to Completion 就能提交了，于是就成功修改了请求参数了

6. 打全局断点的话，是无法正常上网的，需要清除断点：rules-> automatic breakpoint ->disabled

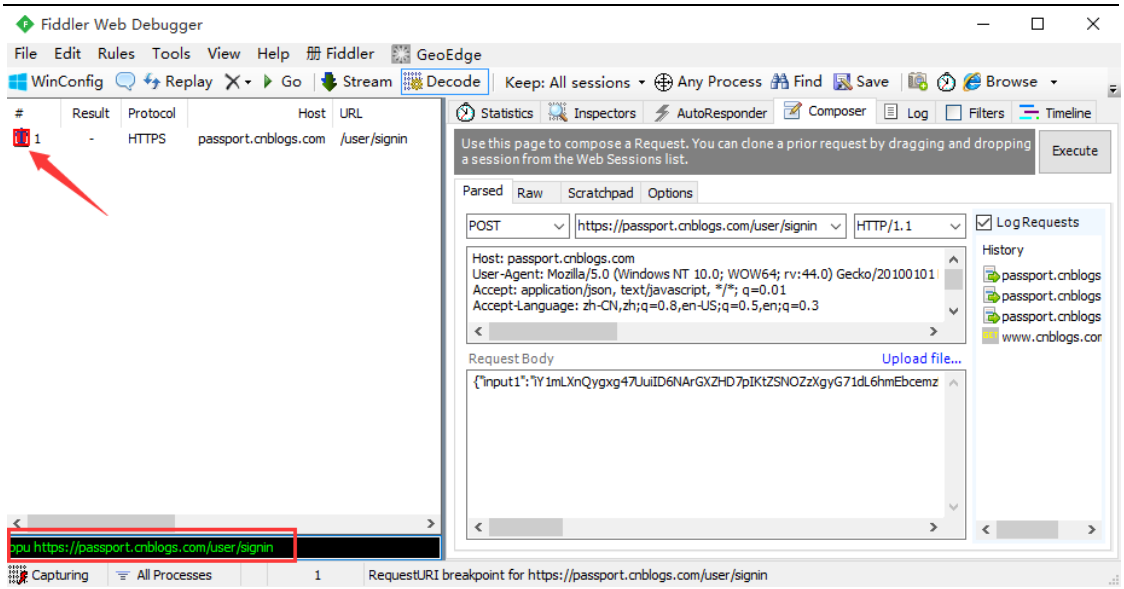
1.9.4 单个断点

已经知道了某个接口的请求地址，这时候只需要针对这一条请求打断点调试，在命令行中输入指令就可以

请求前断点（before response)： **bpu**

1. 论坛登录接口：https://passport.cnblogs.com/user/signin

2. 命令行输入：bpu https://passport.cnblogs.com/user/signin 回车



3. 请求登录接口的时候，就会只拦截登录这个接口了，此时可以修改任意请求参数

4. 取消断点，在命令行输入： **bpu** 回车就可以了

响应后断点（after　requests）： **bpafter**

1. 论坛登录接口：https://passport.cnblogs.com/user/signin

2. 在命令行输入：bpafter https://passport.cnblogs.com/user/signin　回车

3. 登录博客园，会发现已经拦截到登录后服务器返回的数据了，此时可以修改任意返回数据

4. 取消断点，在命令行输入： **bpafter** 回车就可以了

1.9.5 拦截来自某个网站所有请求

1. 在命令行输入：bpu www.cnblogs.com

2. 打开博客园任意网页，发现都被拦截到了

3. 打开博客园其他网站，其它网站可以正常请求

4. 说明只拦截了来自部落论坛（www.cnblogs.com）的请求

5. 清除输入 bpu 回车即可

1.9.6 命令行其它相关指令

Bpafter，　Bps，　bpv，　bpm，　bpu

这几个命令主要用于批量设置断点

Bpafter　xxx:　中断　URL　包含指定字符的全部　session　响应

Bps　xxx:　中断　HTTP　响应状态为指定字符的全部　session　响应

Bpv　xxx:　中断指定请求方式的全部　session　响应

Bpm　xxx:　中断指定请求方式的全部　session　响应 、、同于 bpv　xxx

Bpu　xxx:与 bpafter 类似

当这些命令没有加参数时，会清空所有设置了断点的 HTTP 请求。

更多的其他命令可以参考 Fiddler 官网手册

1.10 会话保存

前言

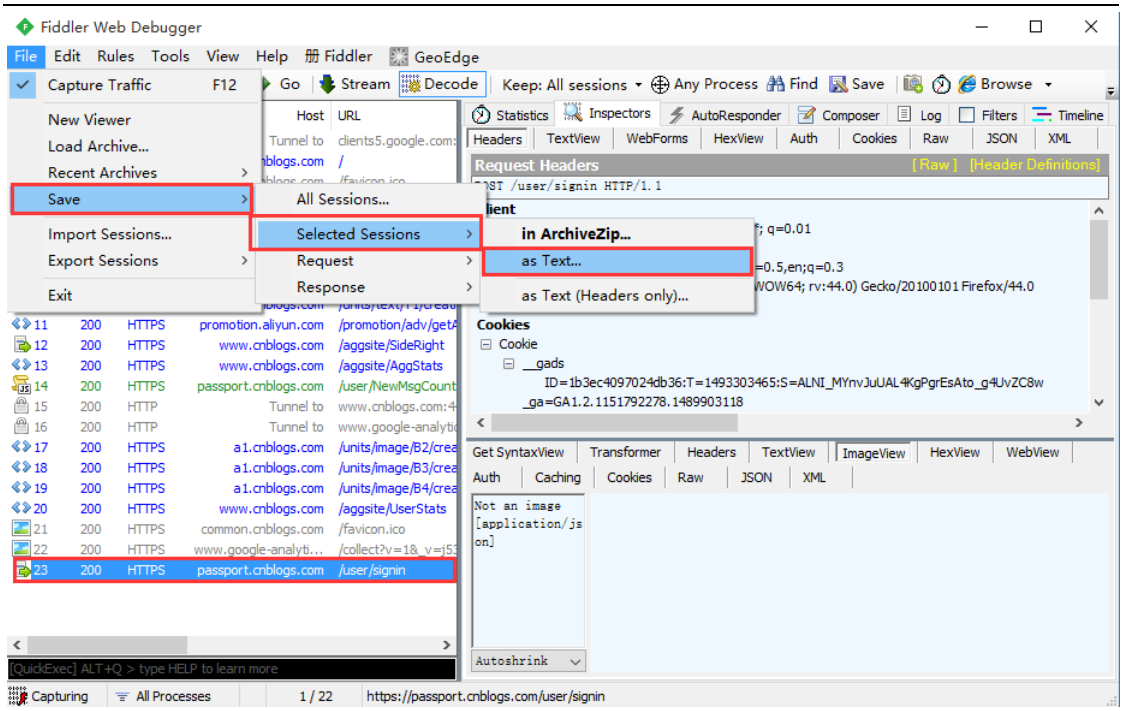
为什么要保存会话呢？举个很简单的场景，你在上海测试某个功能接口的时候，发现了一个 BUG，而开发这个接口的开发人员是北京的一家合作公司。你这时候给对方开发提 bug，

如何显得专业一点,能让对方心服口服的接受这个 BUG 呢？如果只是截图的话,不是很方便，因为要截好几个地方还描述不清楚，不如简单粗暴一点把整个会话保存起来，发给对方。

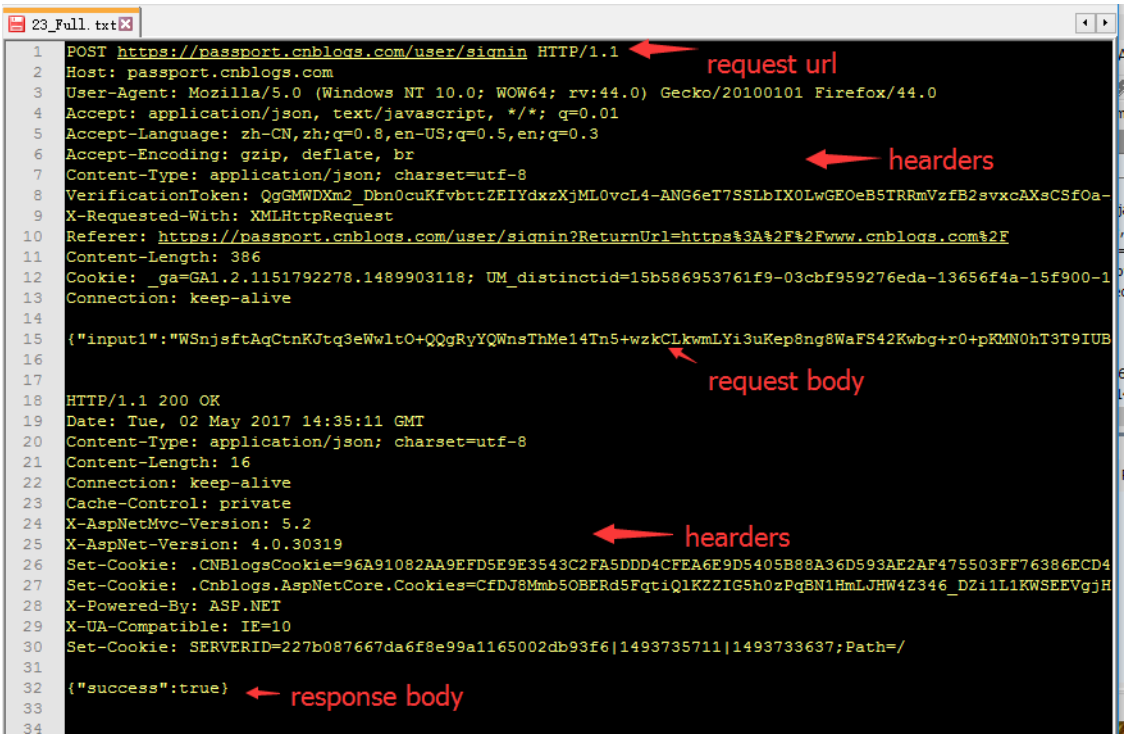
1.10.1 保存为文本

1.以博客园登录为例，抓到登录的请求会话

2.点左上角 File>Save>Selected Sessions>as Text，保存到电脑上就是文本格式的



3. 文本格式的可以直接打开，结果如下图



1.10.2 几种保存方式

1. save-All Sessions ：保存所有的会话，saz 文件

2.save-Selected Session:保存选中的会话

--in ArchiveZIP ：保存为 saz 文件

--as Text ：以 txt 文件形式保存整个会话包括 Request 和 Response

--as Text (Headers only)：仅保存头部

3.Request：保存请求

--Entir Request:保存整个请求信息（headers 和 body）

--Request Body:只保存请求 body 部分

4.Response：保存返回

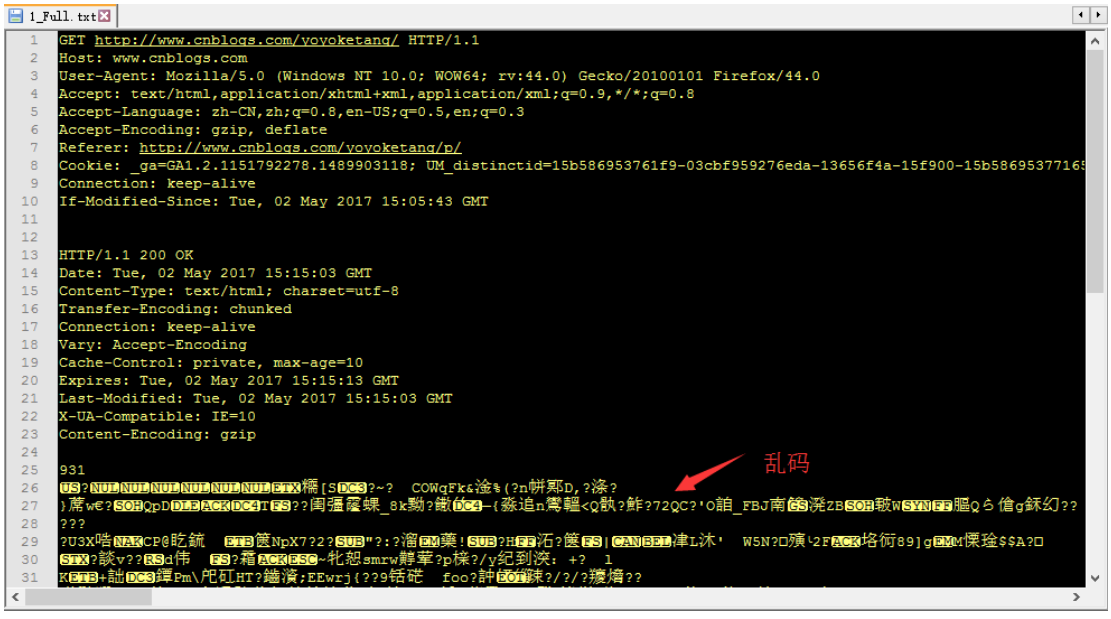
--Entir Response:保存整个返回信息（headers 和 body）

--Response Body:只保存返回 body 部分

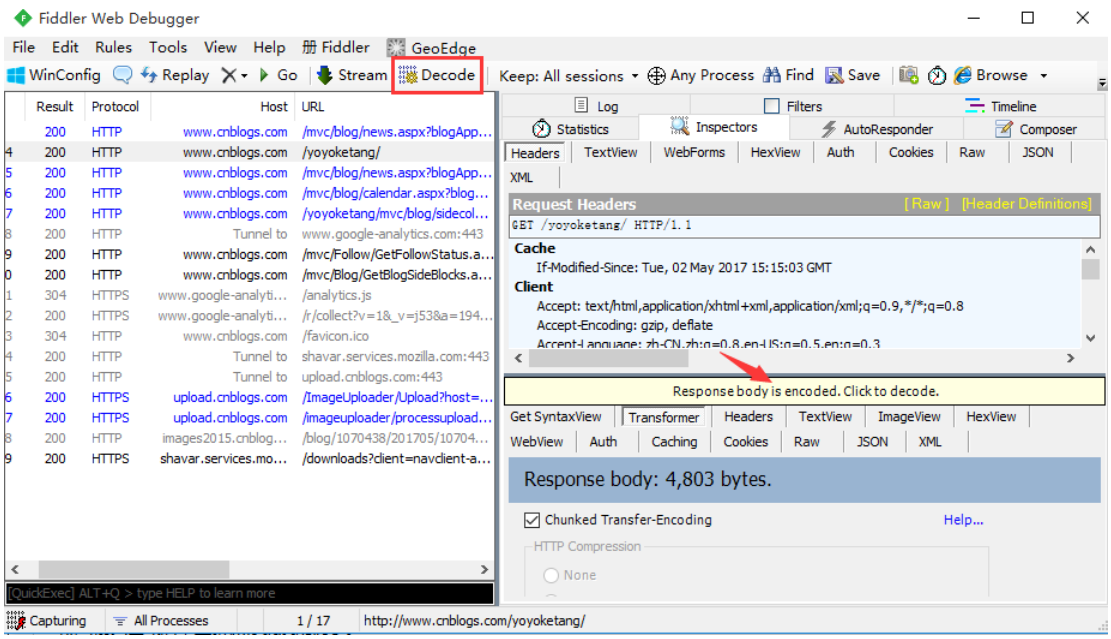
--and Open as Local File：保存 Response 信息，并打开文件

1.10.3 乱码问题（decode）

1.打开博客园首页：http://www.cnblogs.com/yoyoketang/，保存之后查看，会发现返回的是乱码



2. 遇到这种情况，主要是需要解码，用前面学到的 decode 方法



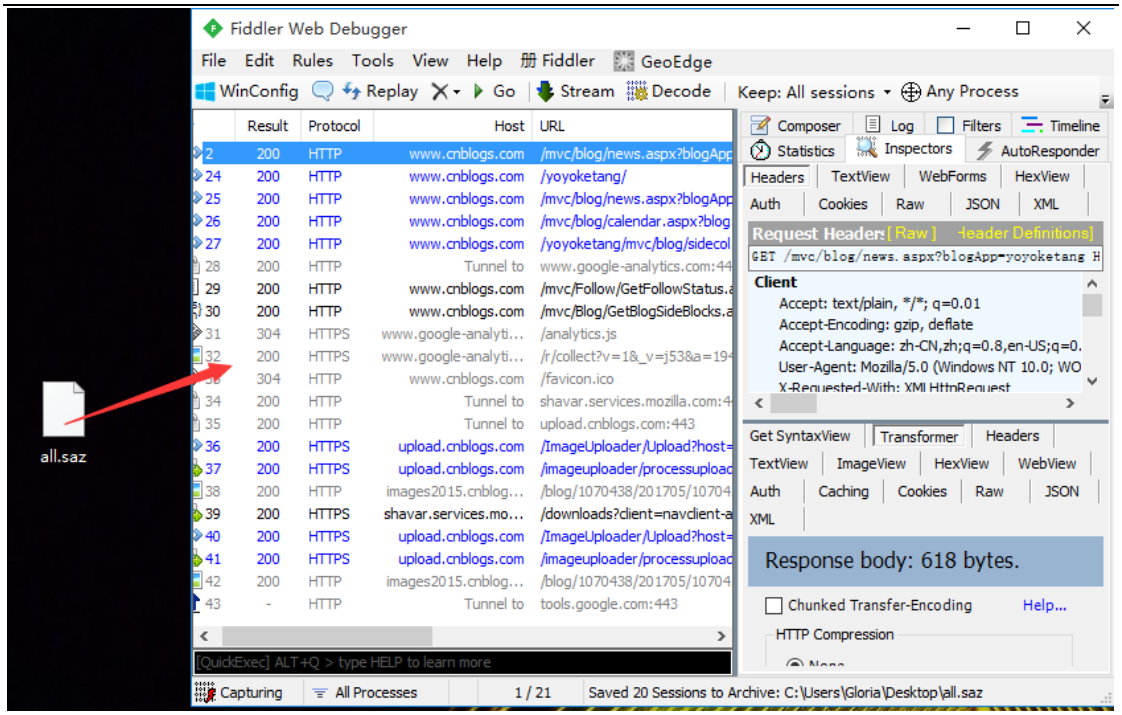
3. 点击箭头区域后，重新保存就没乱码了。

4. 还有一个最简单办法就是选中上图会话框上的 decode 按钮，这样就自动解码了。

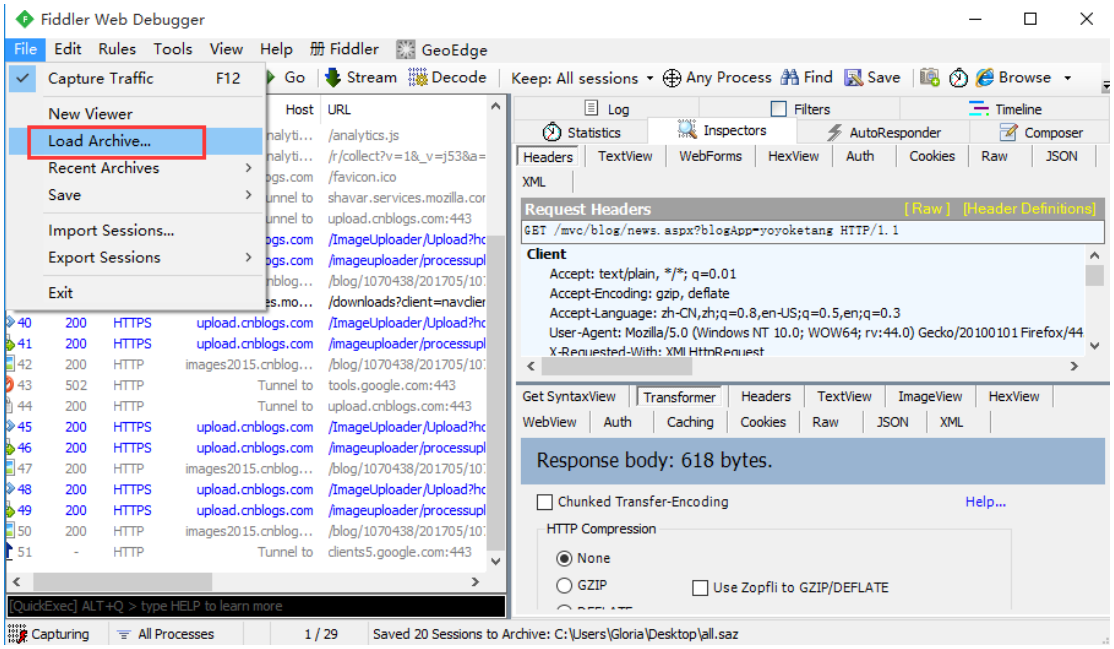
1.10.4 保存与导入全部会话

1. 我们可以打开 fiddler，操作完博客园后，选中 save>All Sessions,保存全部会话

2. 保存后，在 fiddler 打开也很方便，直接把刚才保存的会话按住拽进来就可以了



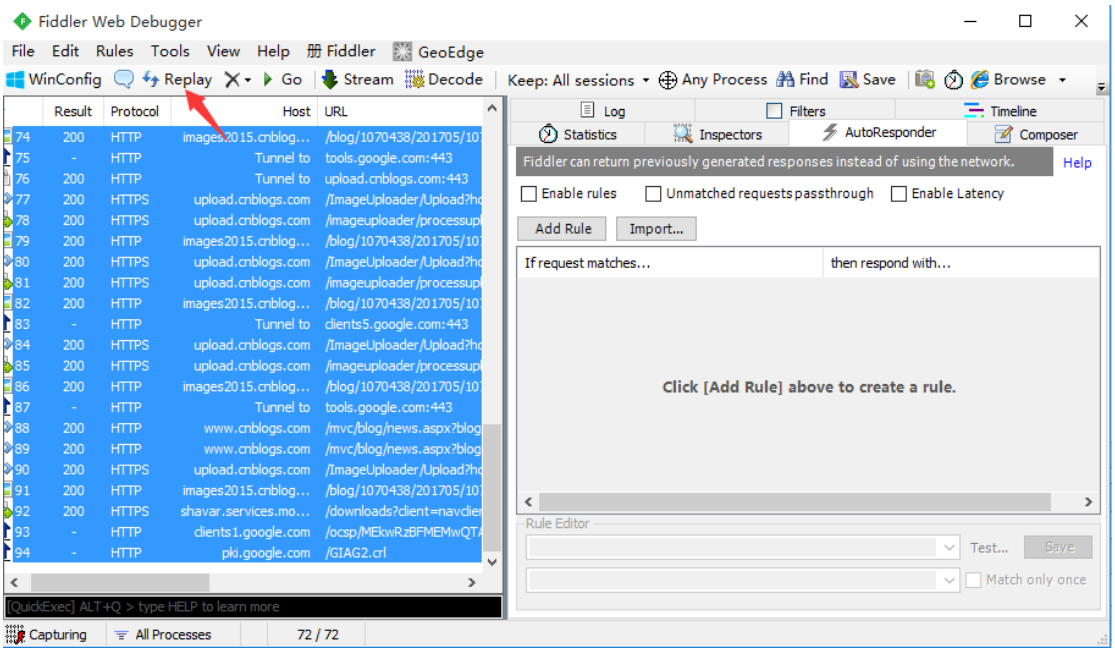
3. 也可以选择 File>Load Archive 导入这个文件



1.10.5 Repaly

1. 导入请求后，可以选中某个请求，点击 Repaly 按钮，重新发请求

2. 也可以 ctrl+all 全部选中后，点 Repaly 按钮，一次性批量请求



这里保存会话和 replay 功能其实就是相当于录制和回放了

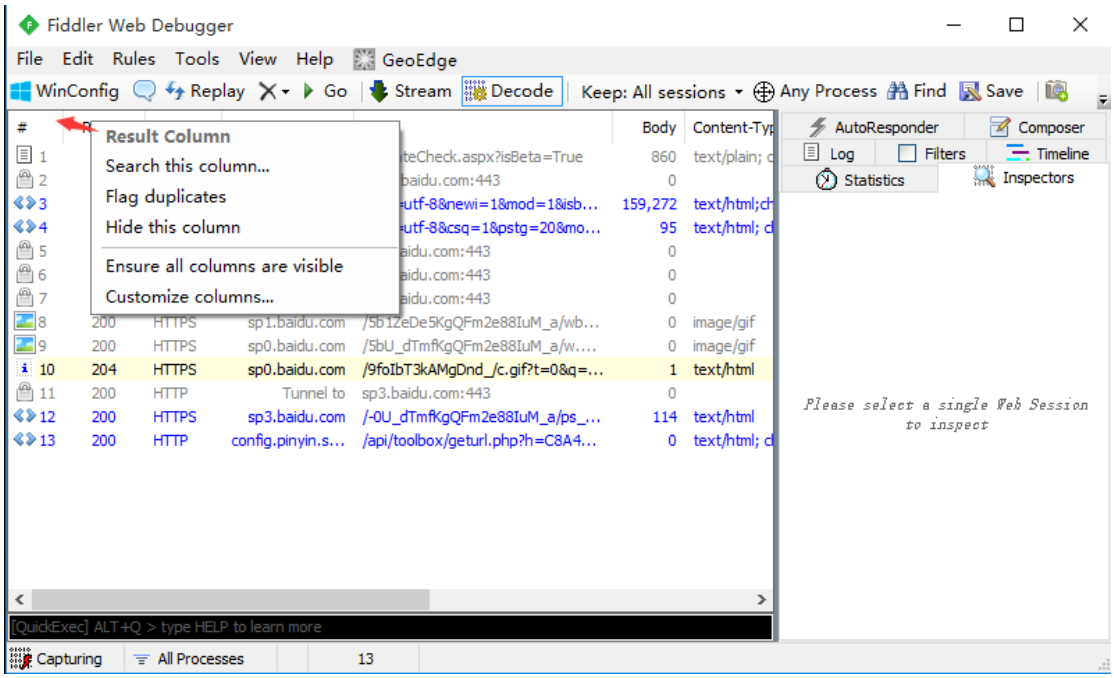
1.11 自定义会话框

前言

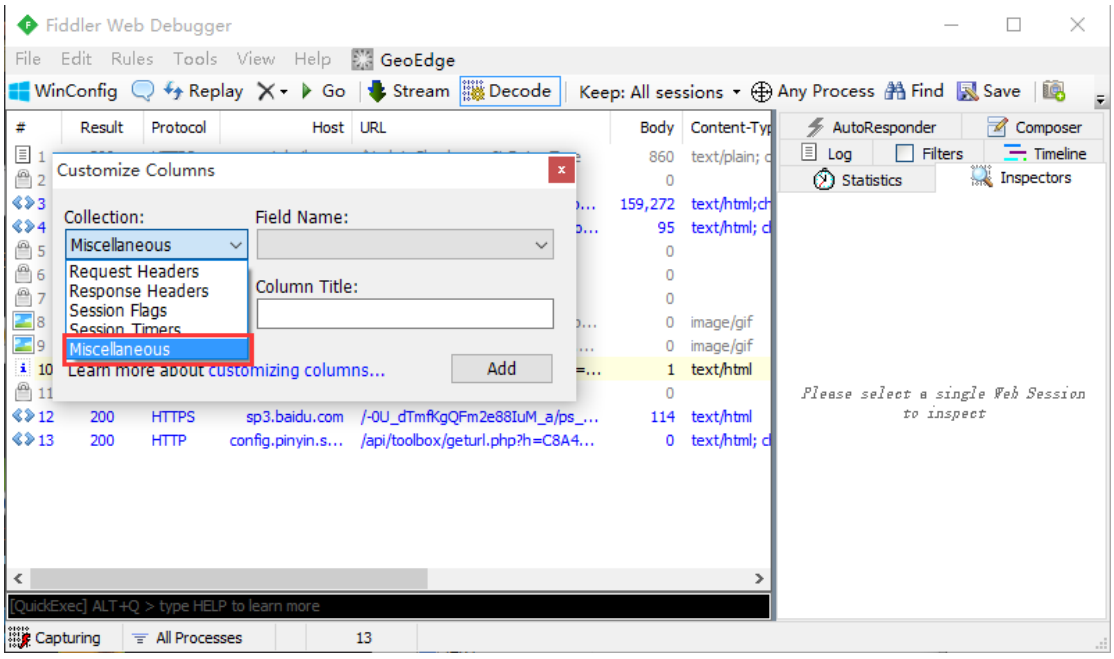
在使用 fiddler 抓包的时候，查看请求类型 get 和 post 每次只有点开该请求，在 Inspectors 才能查看 get 和 post 请求，不太方便。于是可以在会话框直接添加请求方式。

1.11.1 添加会话框菜单

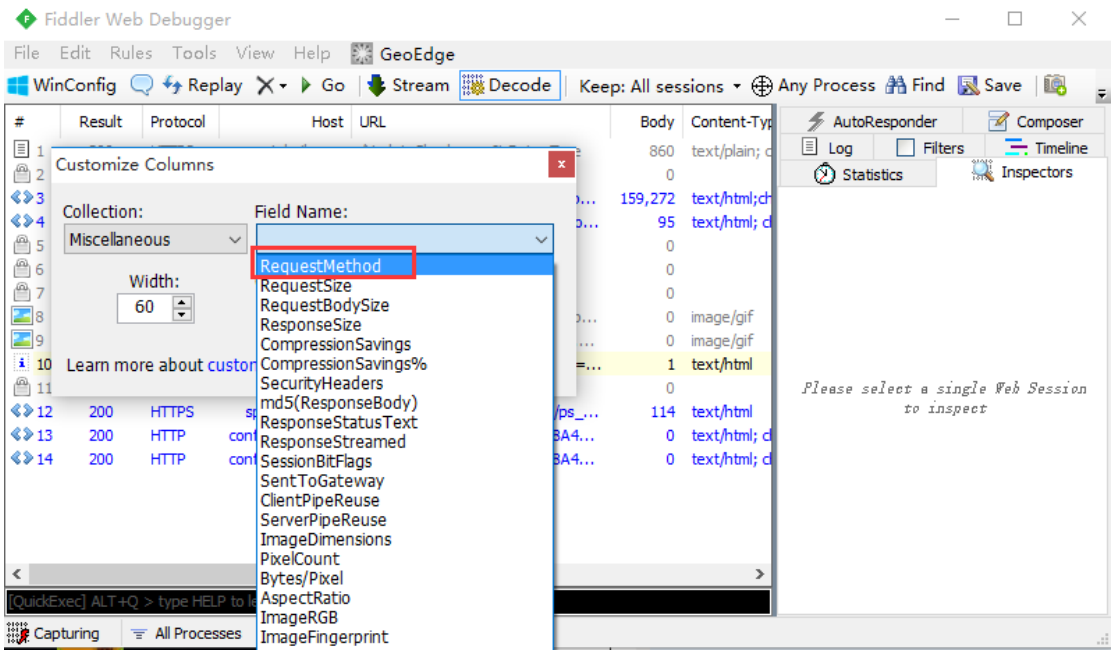
1. 点会话框菜单（箭头位置），右键弹出选项菜单



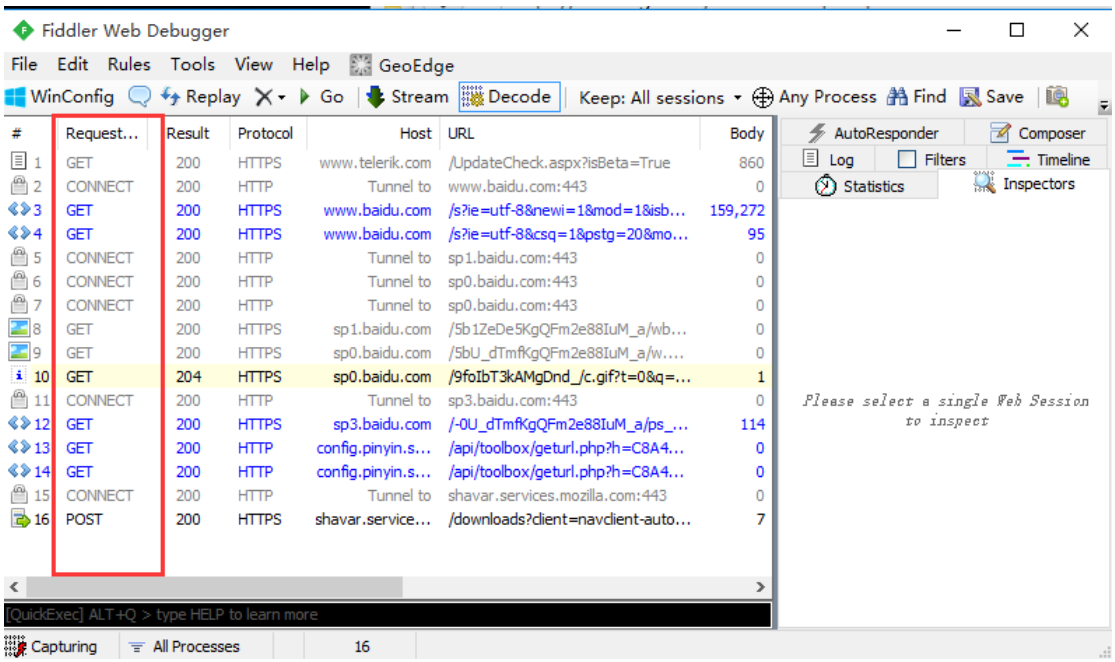
2. 选择 Customize columns 选项，Collection 选项选择 Miscellaneous



3. Field Name 选择：RequestMethod

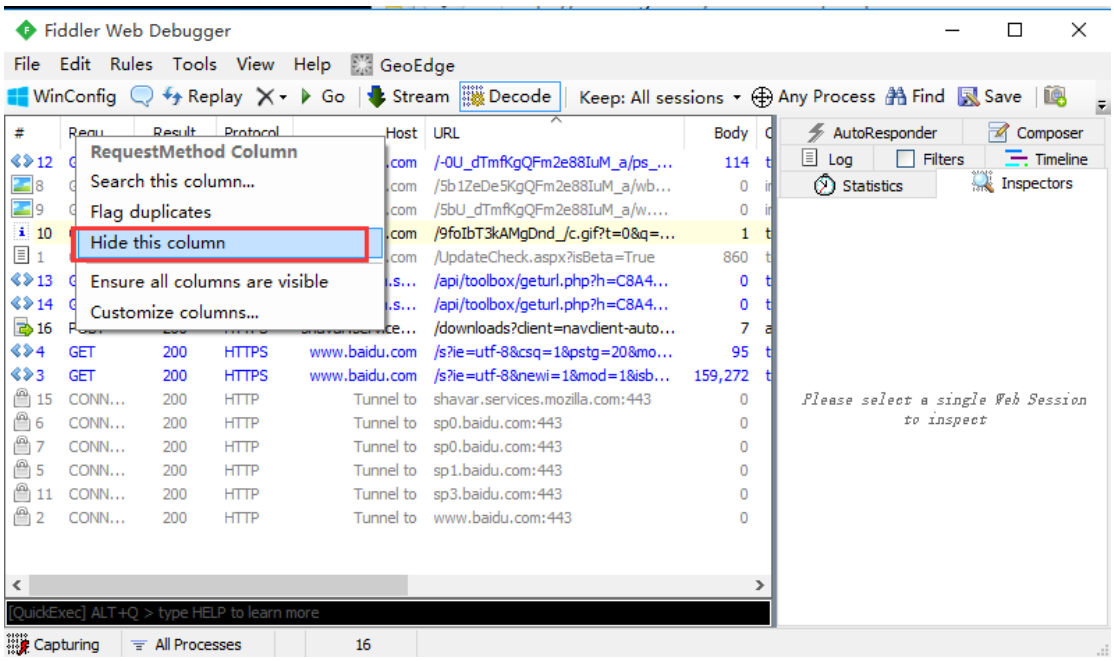


4. 点 Add 按钮即可添加成功

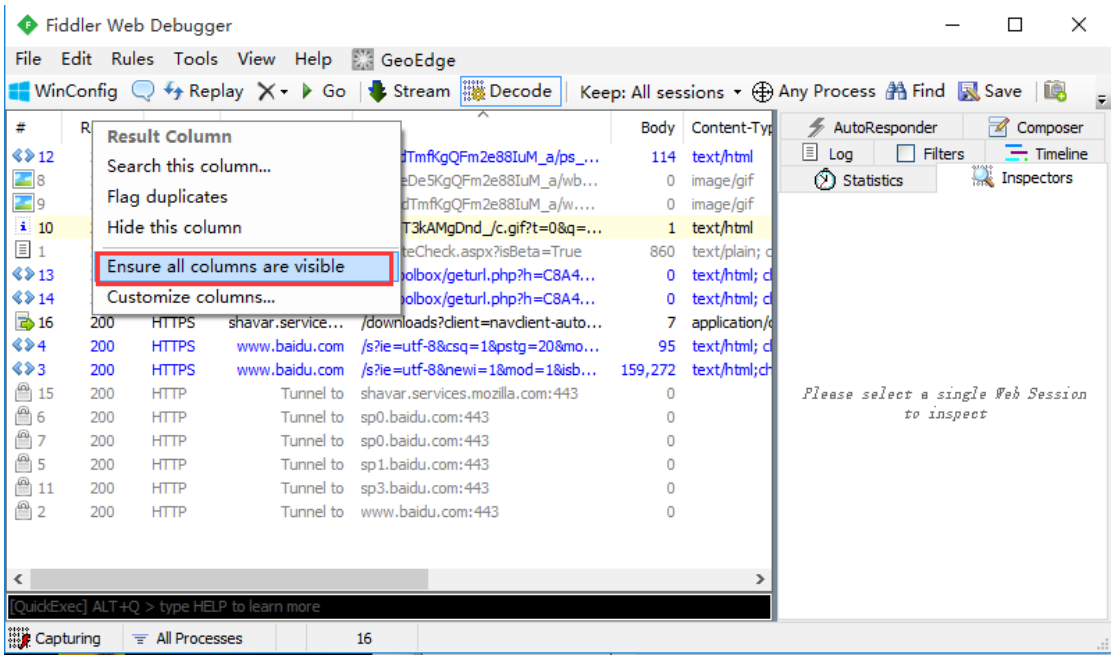


1.11.2 隐藏会话菜单

1. 选择需要隐藏的菜单，右键。选择 Hide this column

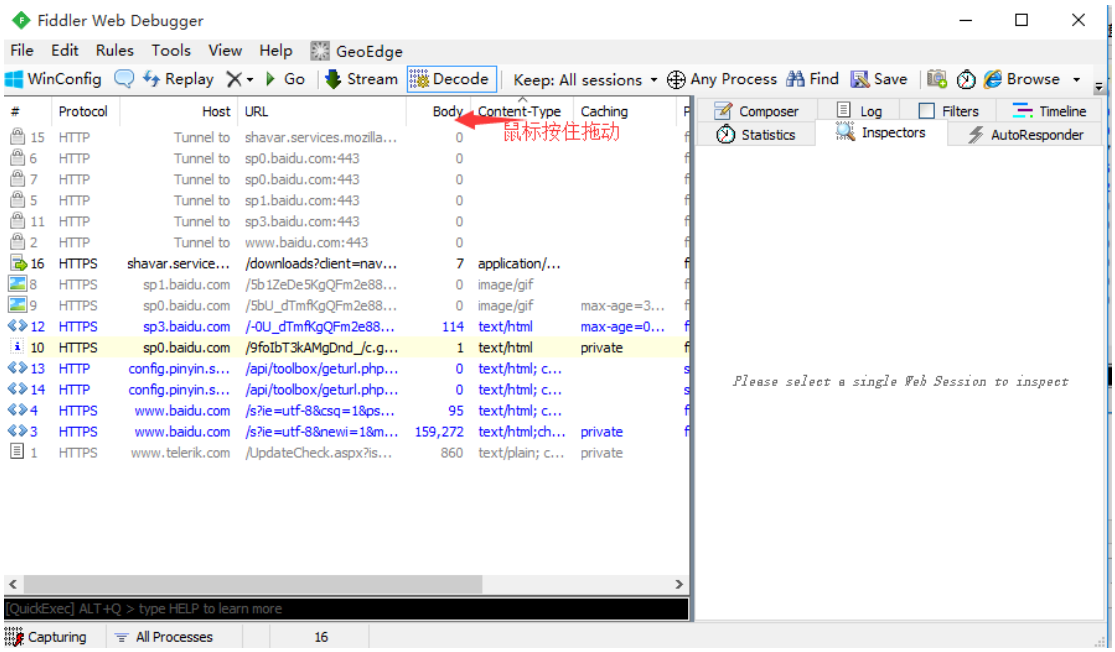


2. 隐藏后也可以让隐藏的菜单显示出来：Ensure all columns are visble



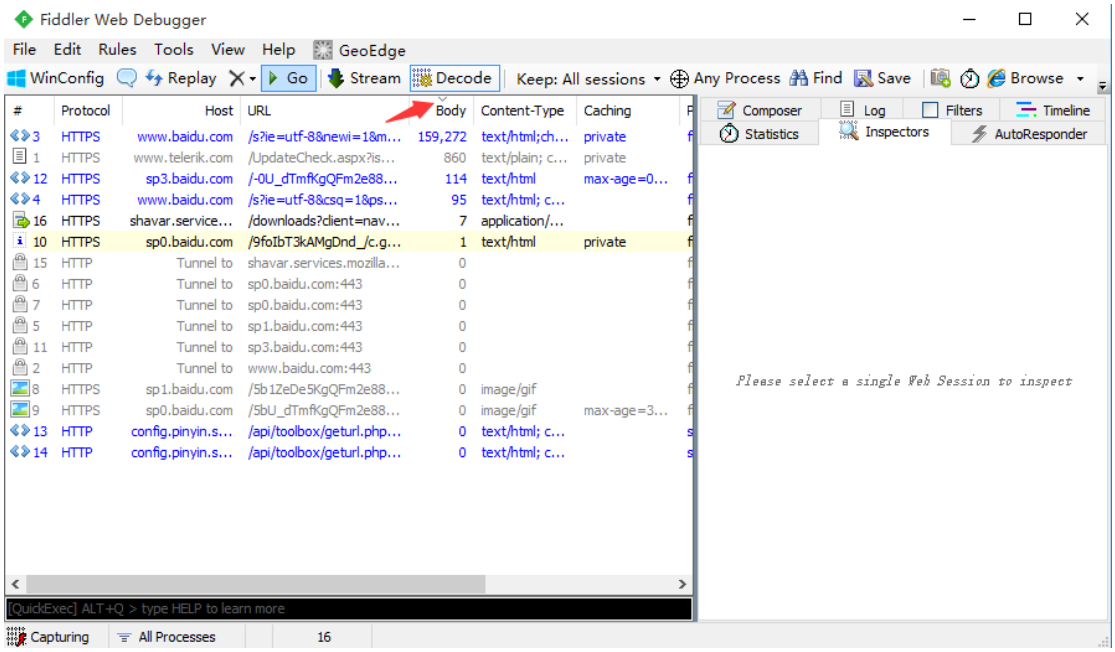
1.11.3 调整会话框菜单顺序

1. 如果需要调整会话框菜单顺序，如：Content-Type 菜单按住后往前移动，就能调整了



1.11.4 会话排序

1. 点击会话框上的菜单，就能对会话列表排序了，如点 body 菜单



2. 点完后上面有个上箭头（正序），或者下箭头（倒叙）。但是不能取消，取消的话关掉 fiddler 后重新打开就行了

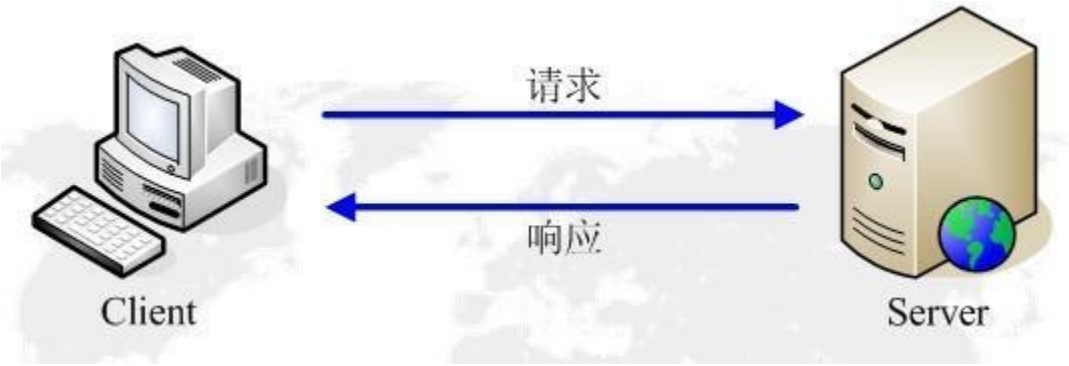
1.12 http 协议简介

1.12.1 什么是 http

1.HTTP 协议是 Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。

2.HTTP（HyperText Transfer Protocol）协议是基于 TCP 的应用层协议，它不关心数据传输的细节，主要是用来规定客户端和服务端的数据传输格式，最初是用来向客户端传输 HTML 页面的内容。默认端口是 80

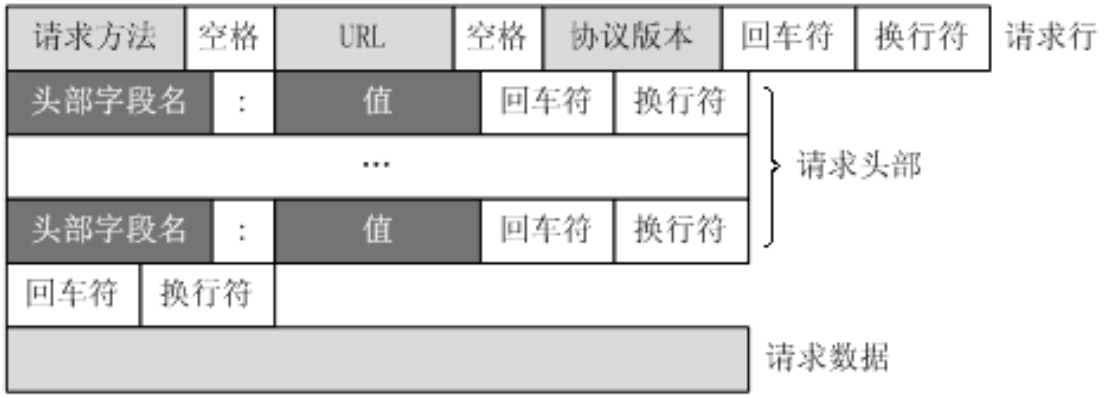
3.http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议



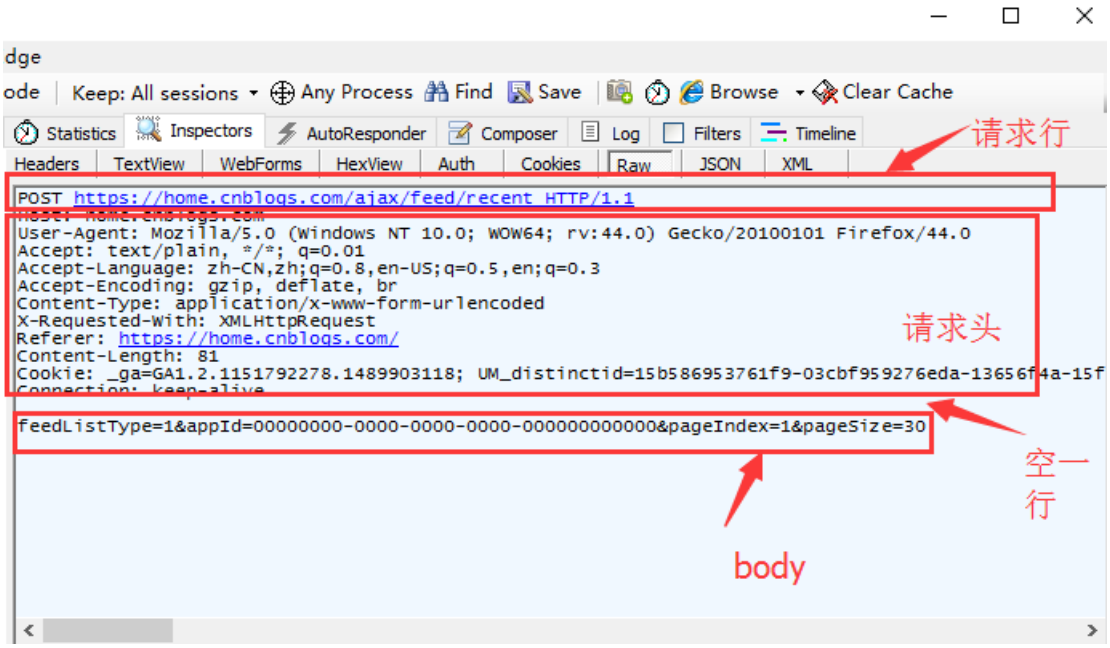
1.12.2 请求报文

1.HTTP 请求报文主要由请求行、请求头部、空一行、请求正文 4 部分组成

（当然，如果不算空的一行，那就是 3 个部分）



2. 下图是 fiddler 工具抓的 post 请求报文（工具使用看 fiddler 篇），可以对照上图，更清楚的理解 http 的请求报文内容。

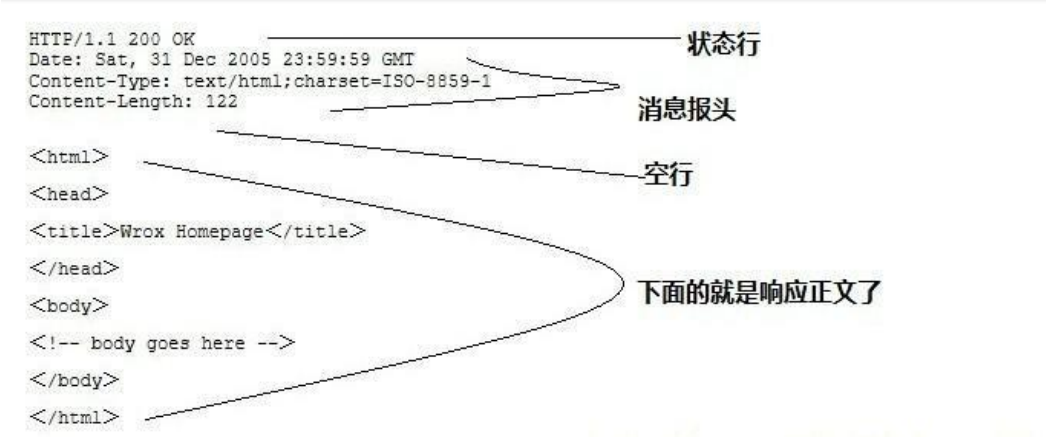


1.12.3 响应报文

1. HTTP 响应报文主要由状态行、消息报头、空一行、响应正文 4 部分组成

（当然，如果不算空的一行，那就是 3 个部分）

2.下图就是一个请求的响应内容，用 fiddler 抓包工具可以查看



1.12.4 完整的 http 内容

1.一个完整的 http 协议其实就两块内容，一个是发的请求，一个服务端给的响应。

2.以下是请求 https://github.com/timeline.json 这个地址后,用 fiddler 抓包导出为文本，查看完整的 http 请求内容。（具体操作查看《fiddler　1.10 会话保存》）

3.内容如下：

（----------以下是请求报文----------）

GET https://github.com/timeline.json HTTP/1.1

Host: github.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate, br

Cookie: xxx(已省略)

Connection: keep-alive

（----------以下是响应报文----------）

```
HTTP/1.1 410 Gone

Date: Sun, 25 Jun 2017 15:41:13 GMT

Content-Type: application/json; charset=utf-8

Server: GitHub.com

Status: 410 Gone

Cache-Control: no-cache

Vary: X-PJAX

X-UA-Compatible: IE=Edge,chrome=1

Set-Cookie:  xxx(已省略)

X-Request-Id: d09e199dc290c6f0dc79fe49007069ab

X-Runtime: 0.004161

Content-Security-Policy:  xxx(已省略)

Strict-Transport-Security:  xxx(已省略)

X-Content-Type-Options: nosniff

X-Frame-Options: deny

X-XSS-Protection: 1; mode=block

X-Runtime-rack: 0.007388

X-GitHub-Request-Id: FE36:2B0A9:177175F:23C092D:594FD998
```

Content-Length: 379

（----------------以下是响应正文（json 格式）--------------------）

{"message":"Hello there, wayfaring stranger. If you’re reading this then you probably didn’t see our blog post a couple of years back announcing that this API would go away: http://git.io/17AROg Fear not, you should be able to get what you need from the shiny new Events API instead.","documentation_url":"https://developer.github.com/v3/activity/events/#list-public-events"}

1.13 请求行

1.13.1 8 种请求方法

1.请求行有三个主要参数：请求方法、url、协议版本。

2.请求方法包含：

1）get

请求指定的页面信息，并返回实体主体。

2） post

向指定资源提交数据进行处理请求(例如提交表单或者上传文件)。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。

3） HEAD

类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头

4） OPTIONS

返回服务器针对特定资源所支持的 HTTP 请求方法，也可以利用向 web 服务器发送‘*’的请求来测试服务器的功能性

5） PUT

向指定资源位置上传其最新内容

6） DELETE

请求服务器删除 Request-URL 所标识的资源

7） TRACE

回显服务器收到的请求，主要用于测试或诊断

8） CONNECT

HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

注意：

1）方法名称是区分大小写的。

2）最常见的的就是通常说的 get 和 post 方法。

1.13.2 url 详解

1.打开百度，在搜索框输入任意文字，搜索后，复制地址栏的 url 地址：

https://www.baidu.com/s?wd=%E4%B8%8A%E6%B5%B7%E6%82%A0%E6%82%A0%E5%8D%9A%E5%AE%A2&rsv_spt=1&rsv_iqid=0x91baaabd00070ba2&issp=1&f=8&rsv_bp=1&rsv_idx=2

2.那么一个完整的 url 地址，基本格式如下：

https://host:port/path?xxx=aaa&ooo=bbb

--http/https：这个是协议类型，如图中 1 所示

--host:服务器的 IP 地址或者域名，如图中 2 所示

--port:HTTP 服务器的默认端口是 80，这种情况下端口号可以省略。

如果使用了别的端口，必须指明，例如：192.168.3.111:8080，这里的 8080 就是端口

--path:访问资源的路径,如图中 3 所示/s（图中 3 是把 path 和请求参数放一起了）

--? :url 里面的？这个符号是个分割线，用来区分问号前面的是 path，问号后面的是参数

--url-params:问号后面的是请求参数，格式：xxx=aaa，如图 4 区域就是请求参数

--&：多个参数用&符号连接

1.13.3 协议版本

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD 方法。

HTTP1.1 新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

第 2 章 requests

2.1 发 get 请求

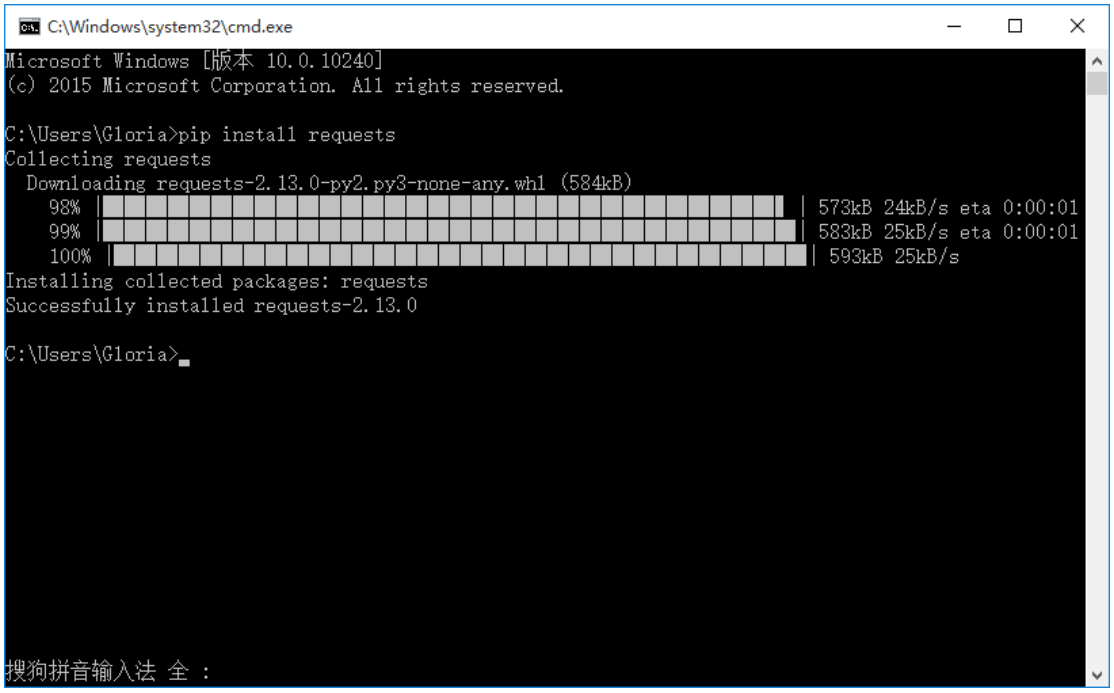
前言

requests 模块，也就是老污龟，为啥叫它老污龟呢，因为这个官网上的 logo 就是这只污龟，接下来就是学习它了。

2.1.1 环境安装

1.用 pip 安装 requests 模块

>>pip install requests



2.1.2 get 请求

1.导入 requests 后，用 get 方法就能直接访问 url 地址，如：
http://www.cnblogs.com/yoyoketang/，看起来是不是很酷

2.这里的 r 也就是 response，请求后的返回值，可以调用 response 里的 status_code 方法查看状态码

3.状态码 200 只能说明这个接口访问的服务器地址是对的，并不能说明功能 OK，一般要查看响应的内容，r.text 是返回文本信息

```
# coding:utf-8
import requests

#请求博客首页
r = requests.get('http://www.cnblogs.com/yoyoketang/')

print r.status_code
print r.text
```

```
est_get
D:\test\python2\python.exe D:/test/yoyotest/req/test_get.py
200

<!DOCTYPE html>
<html lang="zh-cn">
<head>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>上海-悠悠 - 博客园</title>
```

2.1.3 params

1.再发一个带参数的 get 请求，如在博客园搜索：yoyoketang,url 地址为：http://zzk.cnblogs.com/s/blogpost?Keywords=yoyoketang

2.请求参数：Keywords=yoyoketang，可以以字典的形式传参:{"Keywords": "yoyoketang"}

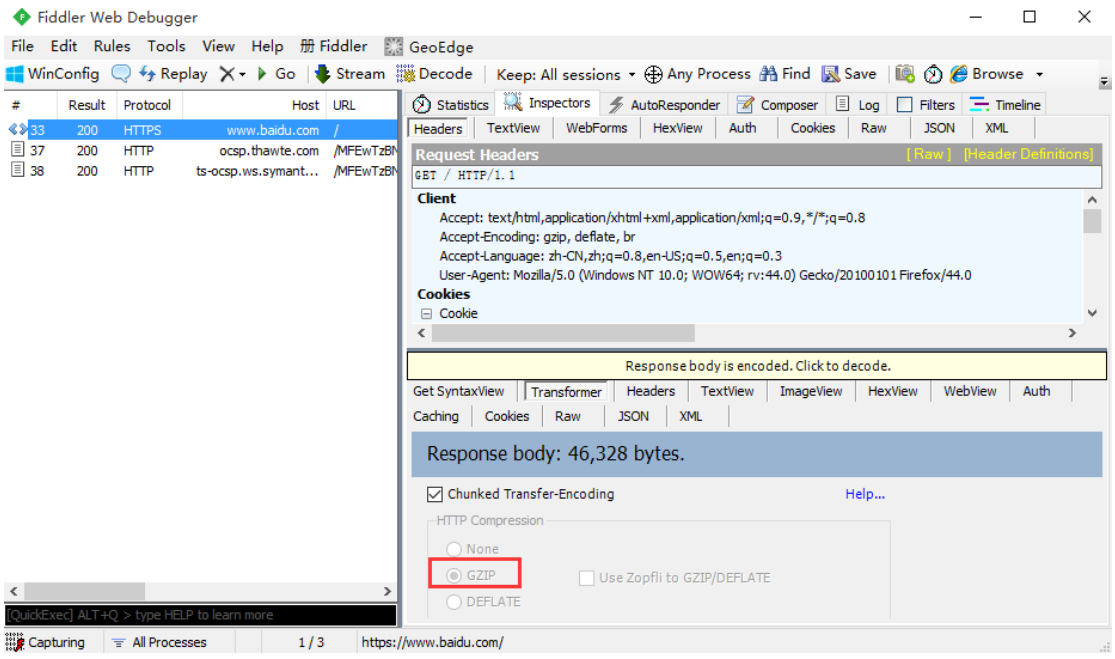
3.多个参数格式：{"key1": "value1", "key2": "value2", "key3": "value3"}

```python
# coding:utf-8
import requests

# 找找看搜索功能
par = {"Keywords": "yoyoketang"}
r = requests.get('http://zzk.cnblogs.com/s/blogpost', params=par)

print r.status_code
print r.text
```

```
test_get | test_get
D:\test\python2\python.exe D:/test/yoyotest/req/test_get.py
200
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title> yoyoketang - 博客园找找看</title>
    <link rel="shortcut icon" href="/Content/Images/favicon.ico" type="image/x-icon" />
    <link rel="stylesheet" href="//at.alicdn.com/t/font_1476446892_1265016.css" />
    <link href="/bundles/vendor/css/search?v=TPZ2g11WKxuQMoDW28Ba86EiiXG-IC754vBdoWpS_cI
```

2.1.4 content

1. 百度首页如果用 r.text 会发现获取到的内容有乱码，因为百度首页响应内容是 gzip 压缩的（非 text 文本）



2. 如果是在 fiddler 工具乱码，是可以点击后解码的，在代码里面可以用 r.content 这个方法，content 会自动解码 gzip 和 deflate 压缩

```
# coding:utf-8
import requests
# 请求百度首页
r = requests.get('https://www.baidu.com/')

print r.url
print r.encoding    # 编码
print r.content  # 获取返回内容（自动解码gzip）
print r.headers
print r.cookies
```

```
est_get
D:\test\python2\python.exe D:/test/yoyotest/req/test_get.py
https://www.baidu.com/
ISO-8859-1
<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-
                </script> <a href=//www.baidu.com/more/ name=tj_briicon class=bri style="

{'Content-Encoding': 'gzip', 'Transfer-Encoding': 'chunked', 'Set-Cookie': 'BDORZ=27315;
<RequestsCookieJar[<Cookie BDORZ=27315 for .baidu.com/>]>
```

2.1.5 response

1. response 的返回内容还有其它更多信息

-- r.status_code　　#响应状态码
-- r.content　　#字节方式的响应体，会自动为你解码 gzip 和 deflate 压缩
-- r.headers　　#以字典对象存储服务器响应头，但是这个字典比较特殊，字典键不区分大小写，若键不存在则返回 None
-- r.json()　　#Requests 中内置的 JSON 解码器
-- r.url　　# 获取 url
-- r.encoding　　# 编码格式
-- r.cookies　　# 获取 cookie
-- r.raw　　#返回原始响应体
-- r.text　　#字符串方式的响应体，会自动根据响应头部的字符编码进行解码
-- r.raise_for_status() #失败请求(非 200 响应)抛出异常

2.2 发 post 请求

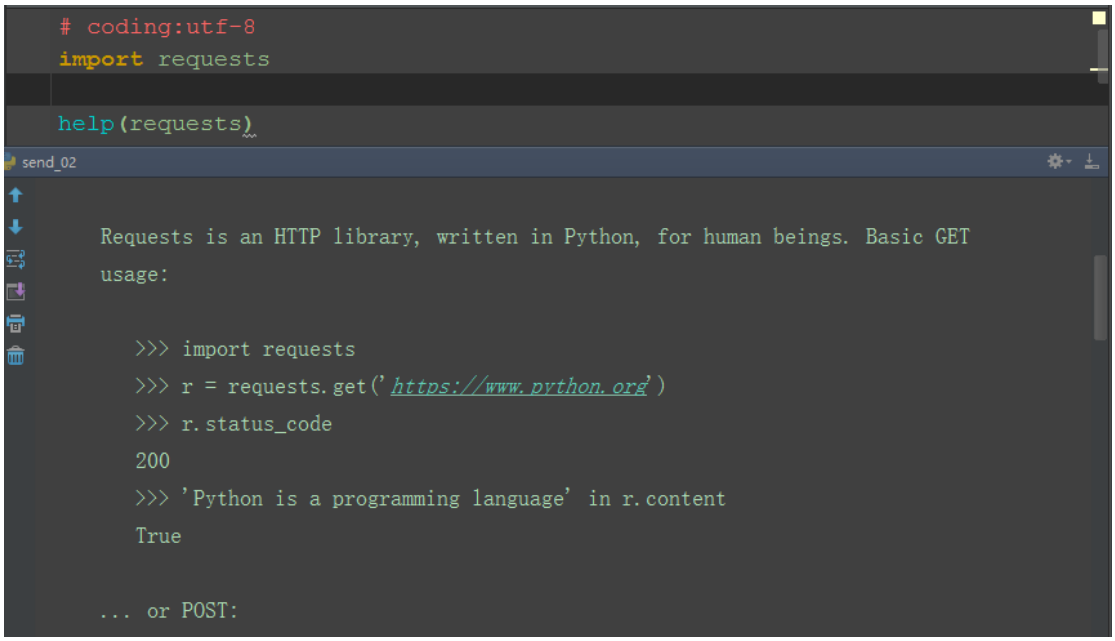
前言

发送 post 的请求参考例子很简单，实际遇到的情况却是很复杂的，首先第一个 post 请求肯定是登录了，但登录是最难处理的。登录问题解决了，后面都简单了。

2.2.1 查看官方文档

1.学习一个新的模块，其实不用去百度什么的，直接用 help 函数就能查看相关注释和案例内容。

```
>>import requests

>>help(requests)
```



2.查看 python 发送 get 和 post 请求的案例

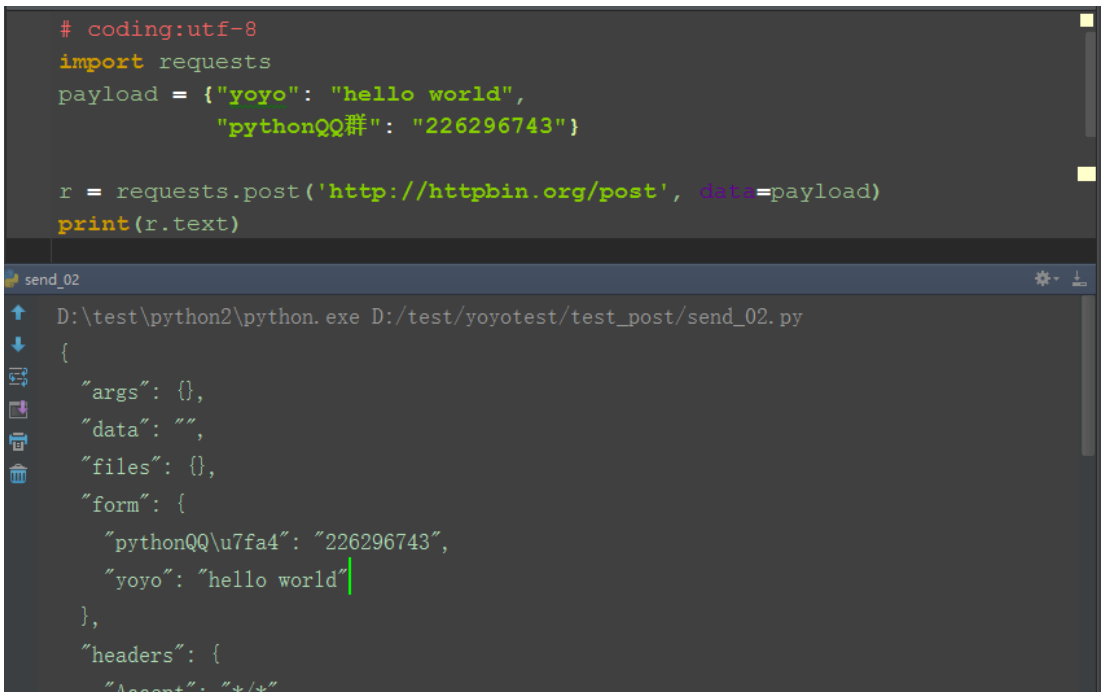
```
>>> import requests
>>> r = requests.get('https://www.python.org')
>>> r.status_code
    200
>>> 'Python is a programming language' in r.content
    True
```

```
... or POST:

>>> payload = dict(key1='value1', key2='value2')
>>> r = requests.post('http://httpbin.org/post', data=payload)
>>> print(r.text)
            {
                ...
                "form": {
                    "key2": "value2",
                    "key1": "value1"
                },
                ...
            }
```

2.2.2 发送 post 请求

1.用上面给的案例，做个简单修改，发个 post 请求

2.payload 参数是字典类型，传到如下图的 form 里

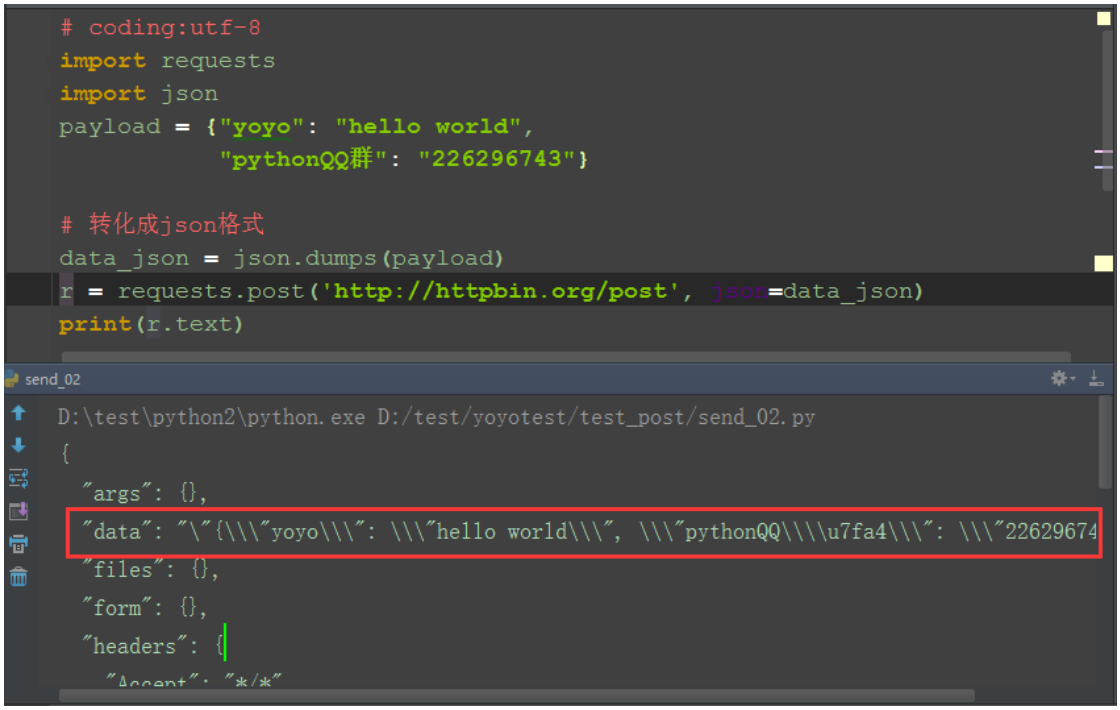


2.2.3 json

1. post 的 body 是 json 类型，也可以用 json 参数传入。

2. 先导入 json 模块，用 dumps 方法转化成 json 格式。

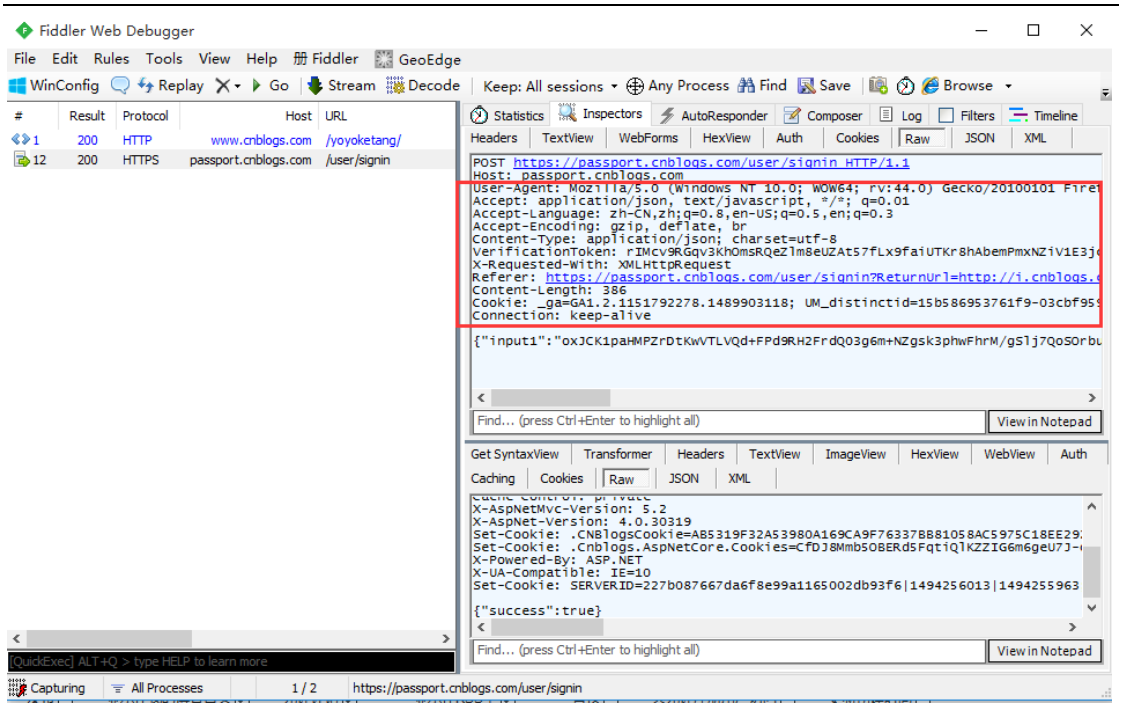
3. 返回结果，传到 data 里



2.2.4 headers

1. 以博客园为例，模拟登陆，实际的情况要比上面讲的几个基本内容要复杂很多，一般登陆涉及安全性方面，登陆会比较复杂

2. 这里需添加请求头 headers, 可以用 fiddler 抓包



3. 讲请求头写成字典格式

```
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101
    Firefox/44.0",
    "Accept": "application/json, text/javascript, */*; q=0.01",
    "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
    "Accept-Encoding": "gzip, deflate, br",
    "Content-Type": "application/json; charset=utf-8",
    "X-Requested-With": "XMLHttpRequest",
    "Cookie": "xxx.............",        # 此处 cookie 省略了
    "Connection": "keep-alive"
    }
```

2.2.5 登陆博客园

（注意：博客园的登录方式变了，所以这里的代码仅供参考，不要复制，肯定登录不成功!!!）

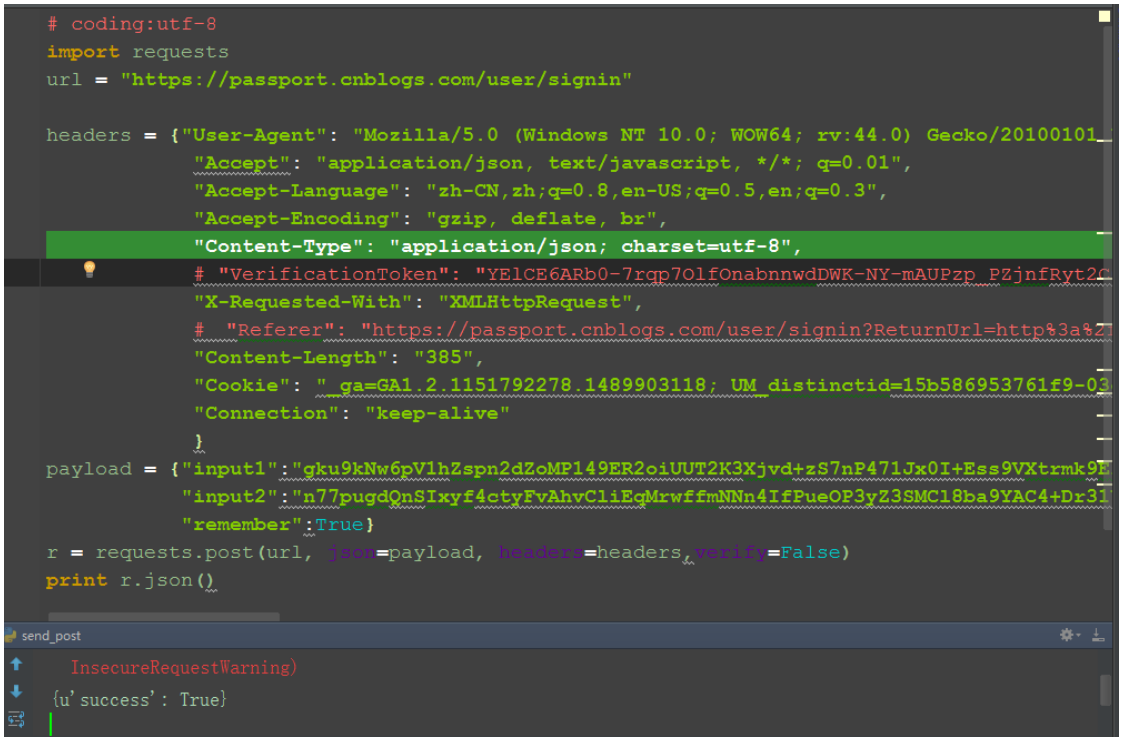
1. 由于这里是 https 请求，直接发送请求会报错误:SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:590)

2. 可以加个参数：verify=False，表示忽略对 SSL 证书的验证

3. 这里请求参数 payload 是 json 格式的，用 json 参数传

4. 红色注释那两行可以不用写

5. 最后结果是 json 格式，可以直接用 r.json 返回 json 数据：{u'success': True}



2.2.6 参考代码

（注意：博客园的登录方式变了，所以这里的代码仅供参考，不要复制，肯定登录不成功!!!）

```
# coding:utf-8
import requests
url = "https://passport.cnblogs.com/user/signin"

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0",
           "Accept": "application/json, text/javascript, */*; q=0.01",
           "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
           "Accept-Encoding": "gzip, deflate, br",
           "Content-Type": "application/json; charset=utf-8",
           # "VerificationToken": "",
           "X-Requested-With": "XMLHttpRequest",
```

```
          #  "Referer": "",
          "Content-Length": "385",
          "Cookie": "xxx...",    # 此处省略
          "Connection": "keep-alive"
          }


payload = {"input1":"xxx",
          "input2":"xxx",
          "remember":True}


r = requests.post(url, json=payload, headers=headers,verify=False)
print r.json()
```

（注意：博客园的登录方式变了，所以这里的代码仅供参考，不要复制，肯定登录不成功!!!）

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

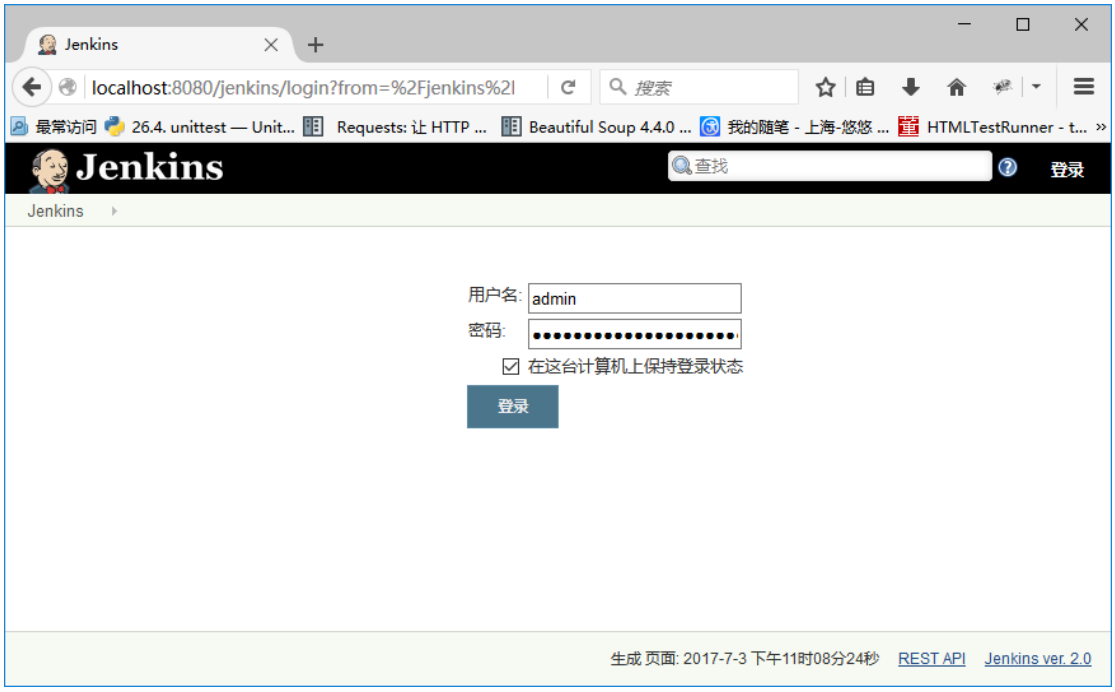
2.3 发 post 请求(data)

前言：

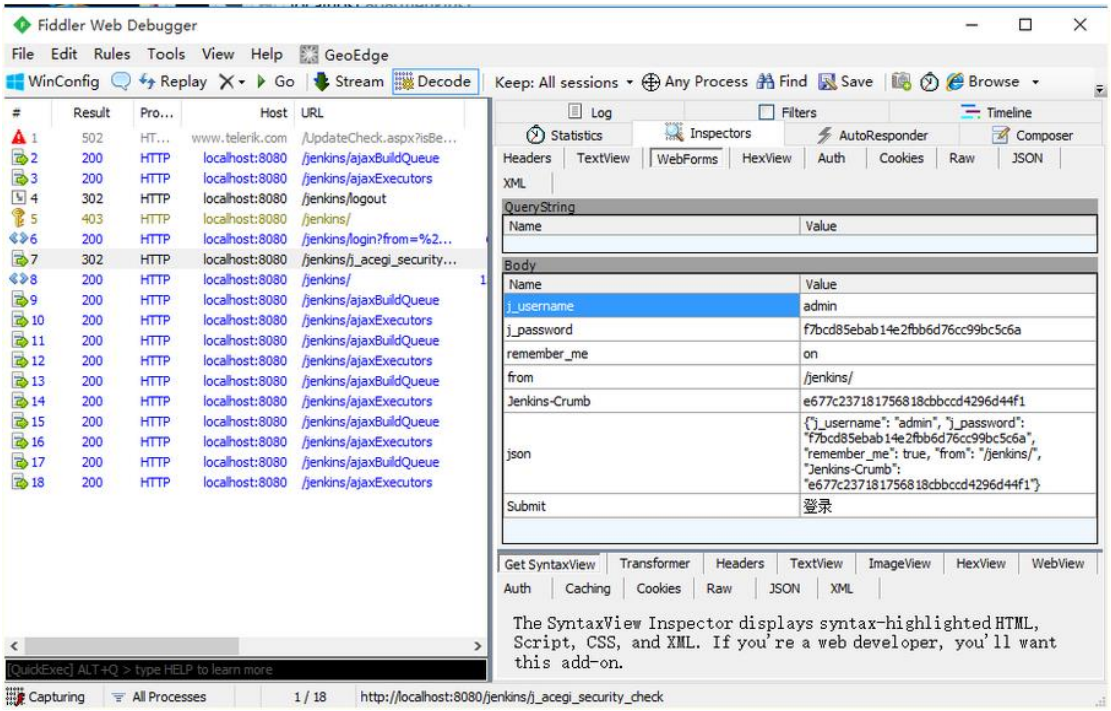
前面登录博客园的是传 json 参数，有些登录不是传 json 的，如 jenkins 的登录，本篇以 jenkins 登录为案例，传 data 参数。

2.3.1 登录 jenkins 抓包

1.登录 jenkins，输入账号和密码



2. fiddler 抓包



3. 这个 body 参数并不是 json 格式，是 key=value 格式，也就是前面介绍 post 请求四种数据类型里面的第二种

一、body数据类型

常见的post提交数据类型有四种：

1.第一种：application/json：这是最常见的json格式，也是非常友好的深受小伙伴喜欢的一种，如下

{"input1":"xxx","input2":"ooo","remember":false}

2.第二种：application/x-www-form-urlencoded：浏览器的原生 form 表单，如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数

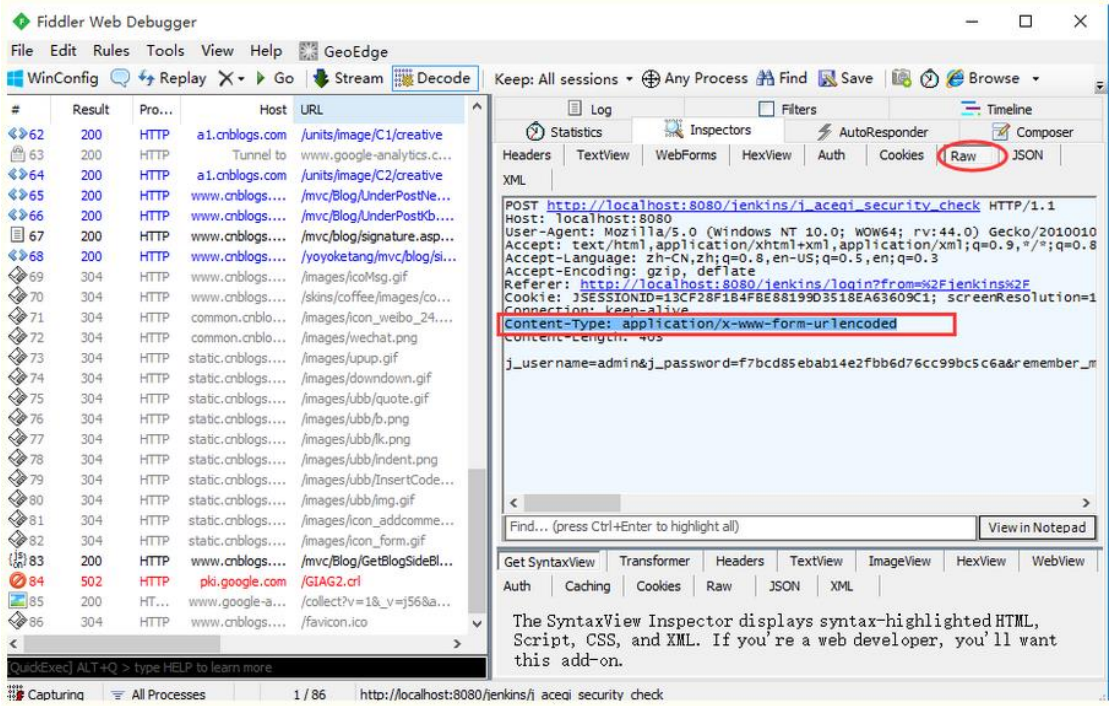
input1=xxx&input2=ooo&remember=false

3.第三种：multipart/form-data:这一种是表单格式的，数据类型如下

```
------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="text"

title

------WebKitFormBoundaryrGKCBY7qhFd3TrwA

Content-Disposition: form-data; name="file"; filename="chrome.png"
```

2.3.2 请求头部

1.上面抓包已经知道 body 的数据类型了，那么头部里面 Content-Type 类型也需要填写对应的参数类型



2.3.3 实现登录

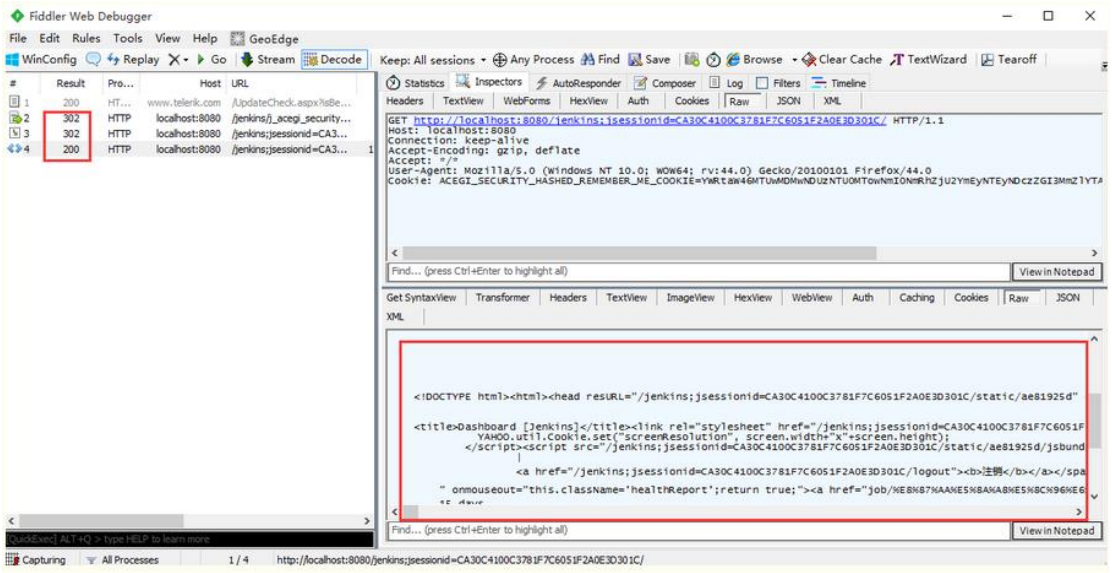
1、登录代码如下：

```
# coding:utf-8
import requests

# 先打开登录首页，获取部分 cookie
url = "http://localhost:8080/jenkins/j_acegi_security_check"
headers = {
                        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0"
                    }   # get 方法其它加个 ser-Agent 就可以了
d = {"from": "",
    "j_password": "f7bcd85ebab14e2fbb6d76cc99bc5c6a",
    "j_username": "admin",
    "Jenkins-Crumb": "e677c237181756818cbbccd4296d44f1",
    "json": {"j_username": "admin",
             "j_password": "f7bcd85ebab14e2fbb6d76cc99bc5c6a",
             "remember_me": True,
             "from": "",
             "Jenkins-Crumb": "e677c237181756818cbbccd4296d44f1"
            },
      "remember_me": "on",
      "Submit": u"登录"
      }
s = requests.session()
r = s.post(url, headers=headers, data=d)
print r.content
```

2.打印结果

2.3.4 判断登录是否成功

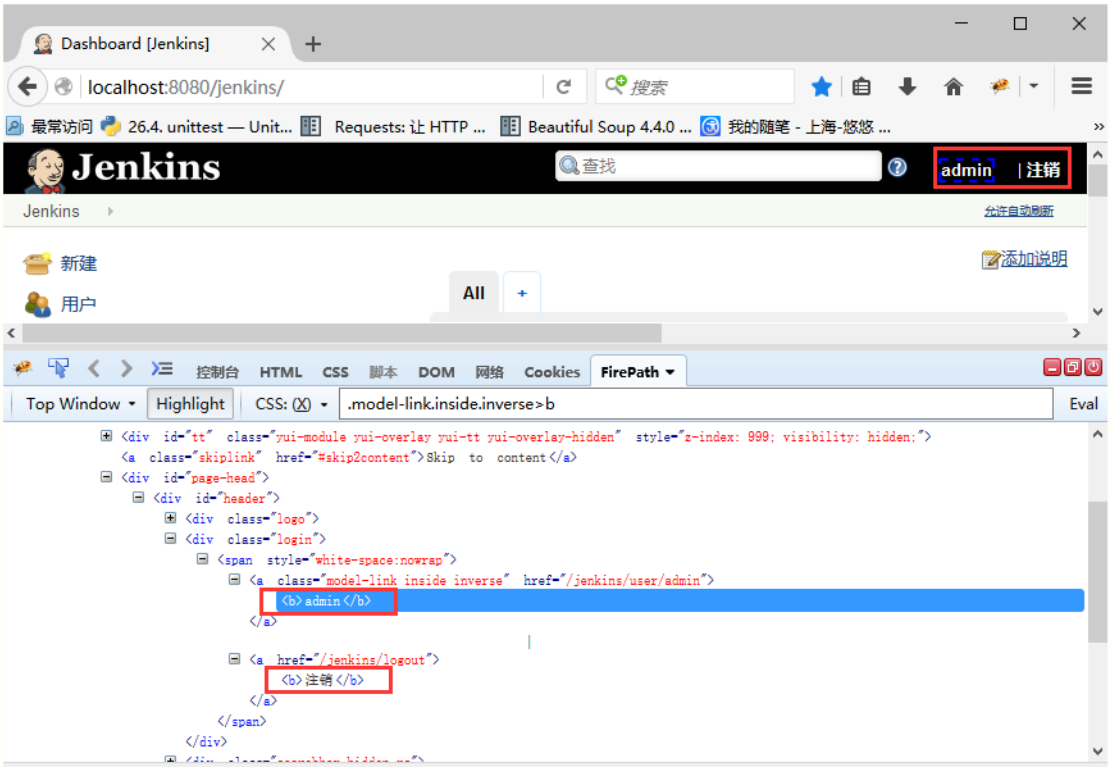
1. 首先这个登录接口有重定向，看左边会话框 302，那登录成功的结果看最后一个 200 就行



2. 返回的结果并不是跟博客园一样的 json 格式，返回的是一个 html 页面

2.3.5 判断登录成功

1. 判断登录成功，可以抓取页面上的关键元素，比如：账号名称 admin，退注销按钮



2. 通过正则表达式提出这 2 个关键字

2.3.6 参考代码

```
# coding:utf-8
import requests

# 先打开登录首页，获取部分 cookie
url = "http://localhost:8080/jenkins/j_acegi_security_check"
headers = {
```

```
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0"
                        }    # get 方法其它加个 ser-Agent 就可以了
d = {"from": "",
     "j_password": "f7bcd85ebab14e2fbb6d76cc99bc5c6a",
     "j_username": "admin",
     "Jenkins-Crumb": "e677c237181756818cbbccd4296d44f1",
     "json": {"j_username": "admin",
              "j_password": "f7bcd85ebab14e2fbb6d76cc99bc5c6a",
              "remember_me": True,
              "from": "",
              "Jenkins-Crumb": "e677c237181756818cbbccd4296d44f1"
               },
     "remember_me": "on",
     "Submit": u"登录"
          }
s = requests.session()
r = s.post(url, headers=headers, data=d)

# 正则表达式提取账号和登录按钮
import re
t = re.findall(r'<b>(.+?)</b>', r.content)
print t[0]
print t[1]
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

2.4 data 和 json 傻傻分不清

前言

在发 post 请求的时候，有时候 body 部分要传 data 参数，有时候 body 部分又要传 json 参数，那么问题来了：到底什么时候该传 json，什么时候该传 data?

2.4.1 识别 json 参数

1.在前面 1.8 章节讲过，post 请求的 body 通常有四种类型，最常见的就是 json 格式的了，这个还是很多识别的

一、body数据类型

常见的post提交数据类型有四种：

1.第一种：application/json：这是最常见的json格式，也是非常友好的深受小伙伴喜欢的一种，如下

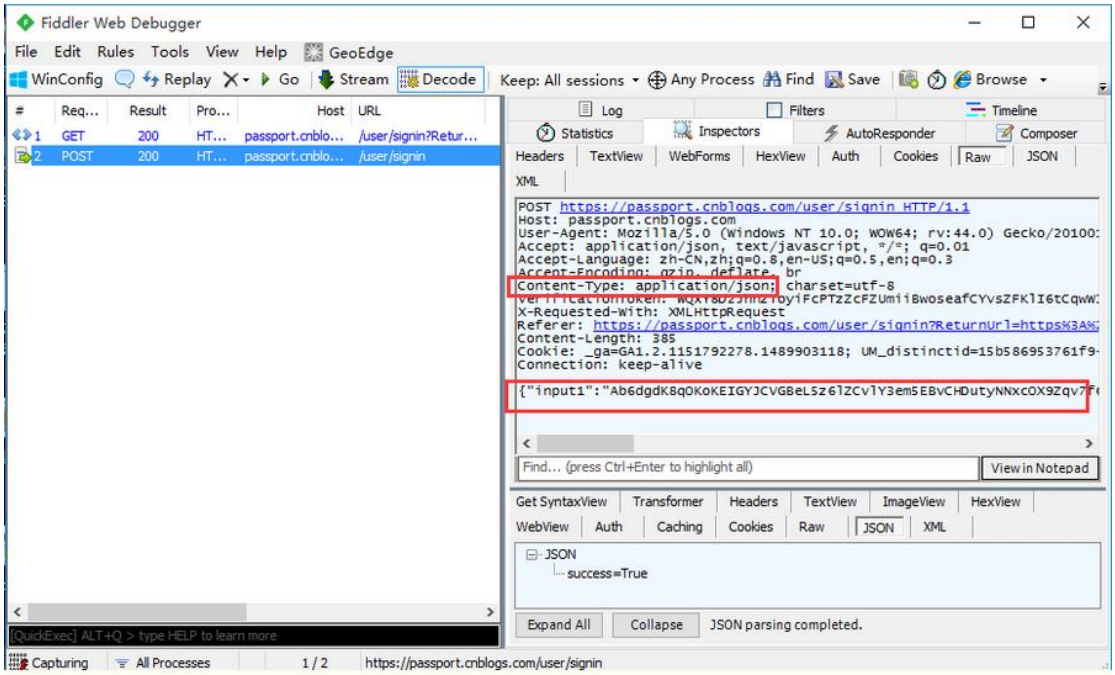
{"input1":"xxx","input2":"ooo","remember":false}

2.第二种：application/x-www-form-urlencoded：浏览器的原生 form 表单，如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数

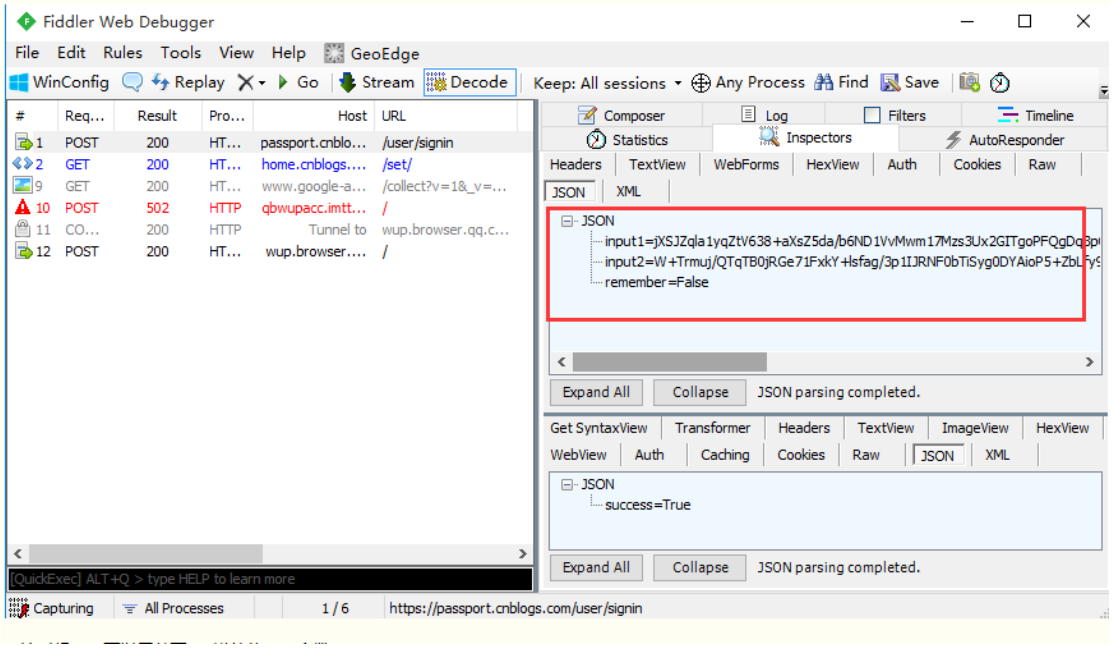
input1=xxx&input2=ooo&remember=false

2.用抓包工具查看，首先点开 Raw 去查看 body 部分，如下图这种，参数最外面是大括号{　　}包起来的，这种已经确诊为 json 格式了。

（也可以查看 Content-Type 参数是 application/json）



3. 再一次确认，可以点开 Json 这一项查看，点开之后可以看到这里的几组参数是 json 解析后的



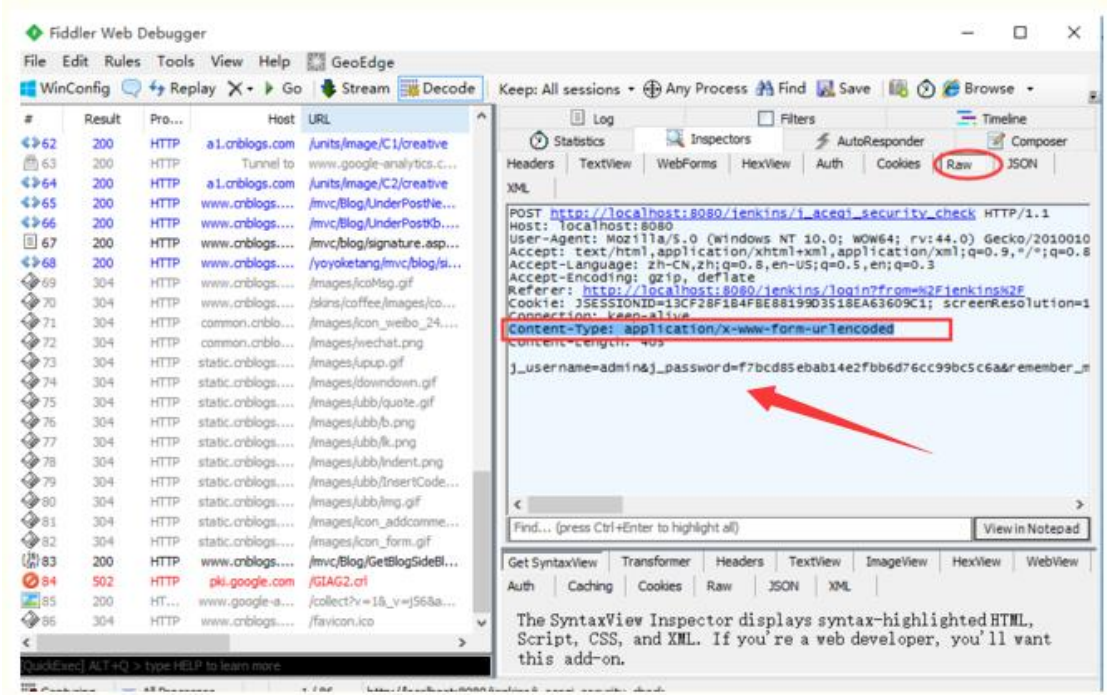
4. 这时候，就可以用前面 2.2 讲的传 json 参数

```
# coding:utf-8
import requests
url = "https://passport.cnblogs.com/user/signin"

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0",
           "Accept": "application/json, text/javascript, */*; q=0.01",
           "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
           "Accept-Encoding": "gzip, deflate, br",
           "Content-Type": "application/json; charset=utf-8",
           # "VerificationToken": "",
           "X-Requested-With": "XMLHttpRequest",
           #  "Referer": "",
           "Content-Length": "385",
           "Cookie": "xxx...",  # 此处省略
           "Connection": "keep-alive"
           }

payload = {"input1":"xxx",
              "input2":"xxx",
              "remember":True}

r = requests.post(url, json=payload, headers=headers,verify=False)
print r.json()
```

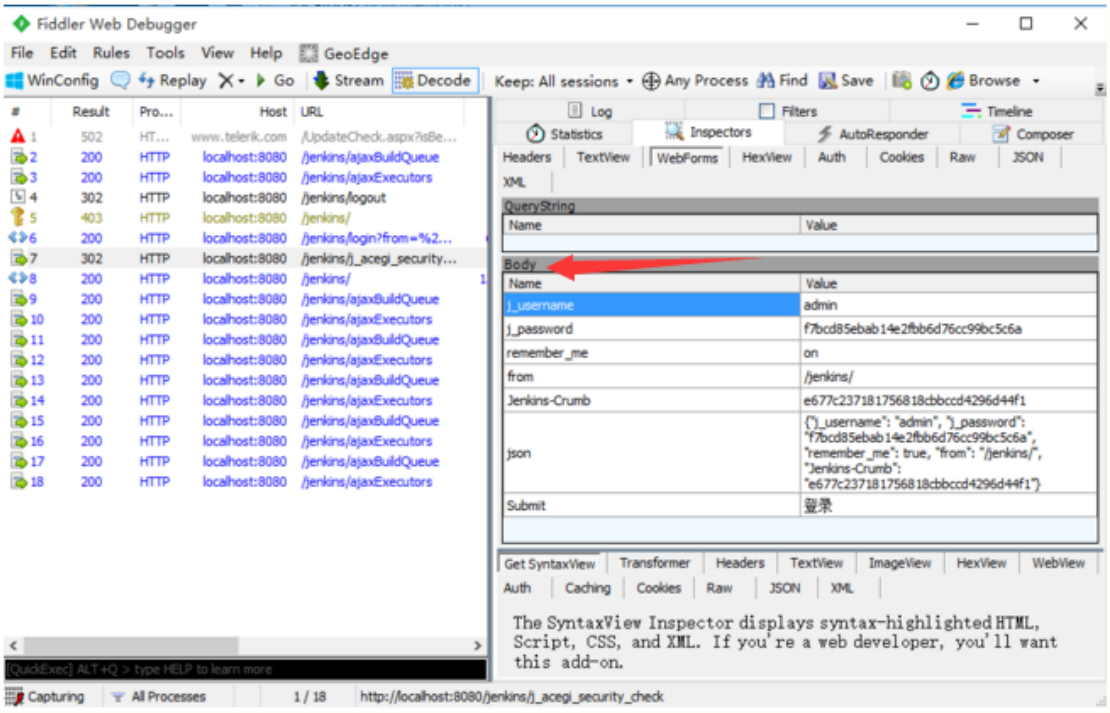
2.4.2 识别 data 参数

1.data 参数也就是这种格式：key1=value1&key2=value2...这种格式很明显没有大括号

点开 Raw 查看，跟上面的 json 区别还是很大的



2.因为这个是非 json 的，所以点开 Json 这个菜单是不会有解析的数据的，这种数据在 WebForms 里面查看



3.可以看到这种参数显示在 Body 部分，左边的 Name 这项就是 key 值，右边的 Value 就是对应的 value 值，像这种参数转化从 python 的字典格式就行了

```
d = {"from": "",
    "j_password": "f7bcd85ebab14e2fbb6d76cc99bc5c6a",
    "j_username": "admin",
    "Jenkins-Crumb": "e677c237181756818cbbccd4296d44f1",
    "json": {"j_username": "admin",
        "j_password": "f7bcd85ebab14e2fbb6d76cc99bc5c6a",
        "remember_me": True,
        "from": "",
        "Jenkins-Crumb": "e677c237181756818cbbccd4296d44f1"},
    "remember_me": "on",
    "Submit": u"登录"
    }
```

```
4.这一种发post时候就传data参数就可以了，格式如下：
s = requests.session()
r = s.post(url, headers=headers, data=d)    # 这里的d就是上一步的字典格式的参数
```

4. 这一种发 post 时候就传 data 参数就可以了，格式如下：

```
s = requests.session()
r = s.post(url, headers=headers, data=d)          # 这里的 d 就是上一步的字典格式的参数
```

现在能分得清 data 参数和 json 参数的不？

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

2.5 https 请求（SSL）

前言

本来最新的 requests 库 V2.13.0 是支持 https 请求的，但是一般写脚本时候，我们会用抓包工具 fiddler，这时候会 报：requests.exceptions.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:590)

小编环境：

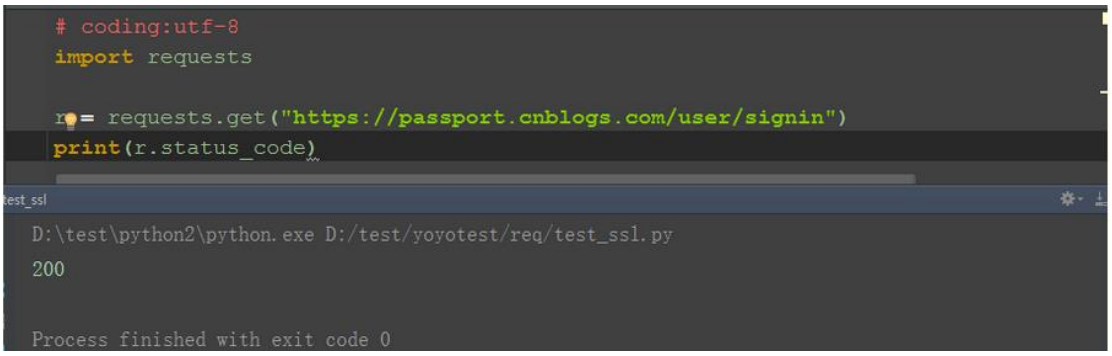
python：2.7.12

requests：2.13.0

fiddler：v4.6.2.0

2.5.1 SSL 问题

1.不启用 fiddler，直接发 https 请求，不会有 SSL 问题（也就是说不想看到 SSL 问题，关掉 fiddler 就行）



2.启动 fiddler 抓包，会出现这个错误：requests.exceptions.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:590)

```
# coding:utf-8
import requests

r = requests.get("https://passport.cnblogs.com/user/signin")
print(r.status_code)
```

```
test_ssl
  resp = self.send(prep, **send_kwargs)
File "D:\test\python2\lib\site-packages\requests\sessions.py", line 609, in send
  r = adapter.send(request, **kwargs)
File "D:\test\python2\lib\site-packages\requests\adapters.py", line 497, in send
  raise SSLError(e, request=request)
requests.exceptions.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:590)

Process finished with exit code 1
```

2.5.2 verify 参数设置

1. Requests 的请求默认 verify=True

2. 如果你将 verify 设置为 False，Requests 也能忽略对 SSL 证书的验证

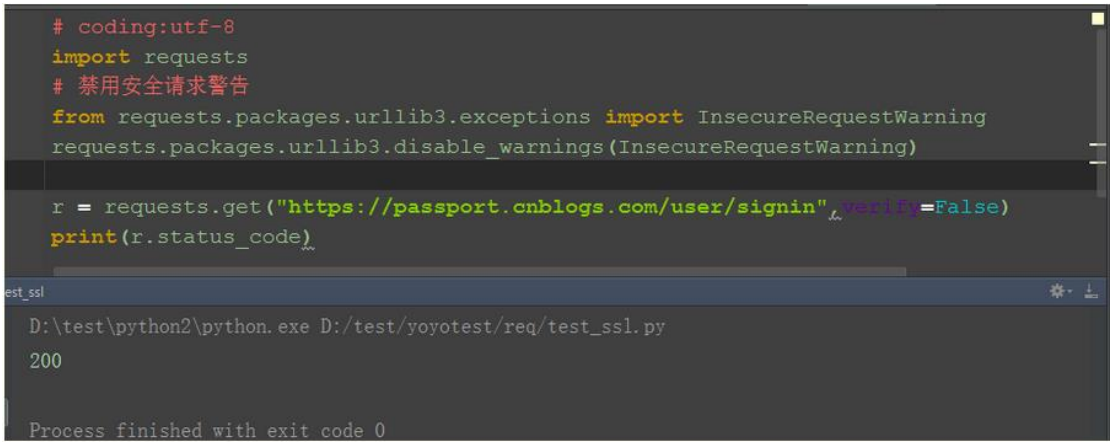
3. 但是依然会出现两行 Warning,可以不用管

```
# coding:utf-8
import requests

r = requests.get("https://passport.cnblogs.com/user/signin",verify=False)
print(r.status_code)
```

```
test_ssl
D:\test\python2\python.exe D:/test/yoyotest/req/test_ssl.py
D:\test\python2\lib\site-packages\requests\packages\urllib3\connectionpool.py:852: InsecureRequest\
  InsecureRequestWarning)
D:\test\python2\lib\site-packages\requests\packages\urllib3\connectionpool.py:852: InsecureRequest\
  InsecureRequestWarning)
200

Process finished with exit code 0
```

2.5.3 忽略 Warning

1. 有些小伙伴有强迫症看到红色的心里就发慌，这里加两行代码可以忽略掉警告，眼不见为净！



2.参考代码：

```
# coding:utf-8
import requests
# 禁用安全请求警告
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

url = "https://passport.cnblogs.com/user/signin"
headers = {
         "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0"
                    }
r = requests.get(url, headers=headers, verify=False)
print(r.status_code)
```

2.5.4 python3 的童鞋

1.python3 报错的话，加上这两行代码就行了

```
#导入 urllib3 包
import urllib3
#使用这个方法就 OK 了
urllib3.disable_warnings()
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

2.6 session 关联接口

前言

上一篇模拟登录博客园，但这只是第一步，一般登录后，还会有其它的操作，如发帖，评论等，这时候如何保持会话呢？

（敲黑板！！！由于博客园最近登录机制变了，登录全部走 cookie 登录）

2.6.1 session 简介

1.查看帮助文档,贴了一部分，后面省略了

```
>>import requests
>>help(requests.session())

class Session(SessionRedirectMixin)
 |   A Requests session.
 |
 |   Provides cookie persistence, connection-pooling, and configuration.
 |
 |   Basic Usage::
 |
 |       >>> import requests
 |       >>> s = requests.Session()
 |       >>> s.get('http://httpbin.org/get')
 |       <Response [200]>
 |
 |   Or as a context manager::
 |
 |       >>> with requests.Session() as s:
 |       >>>         s.get('http://httpbin.org/get')
 |       <Response [200]>
```

2.6.2 使用 session 登录

1.使用 session 登录只需在上一篇基础上稍做修改

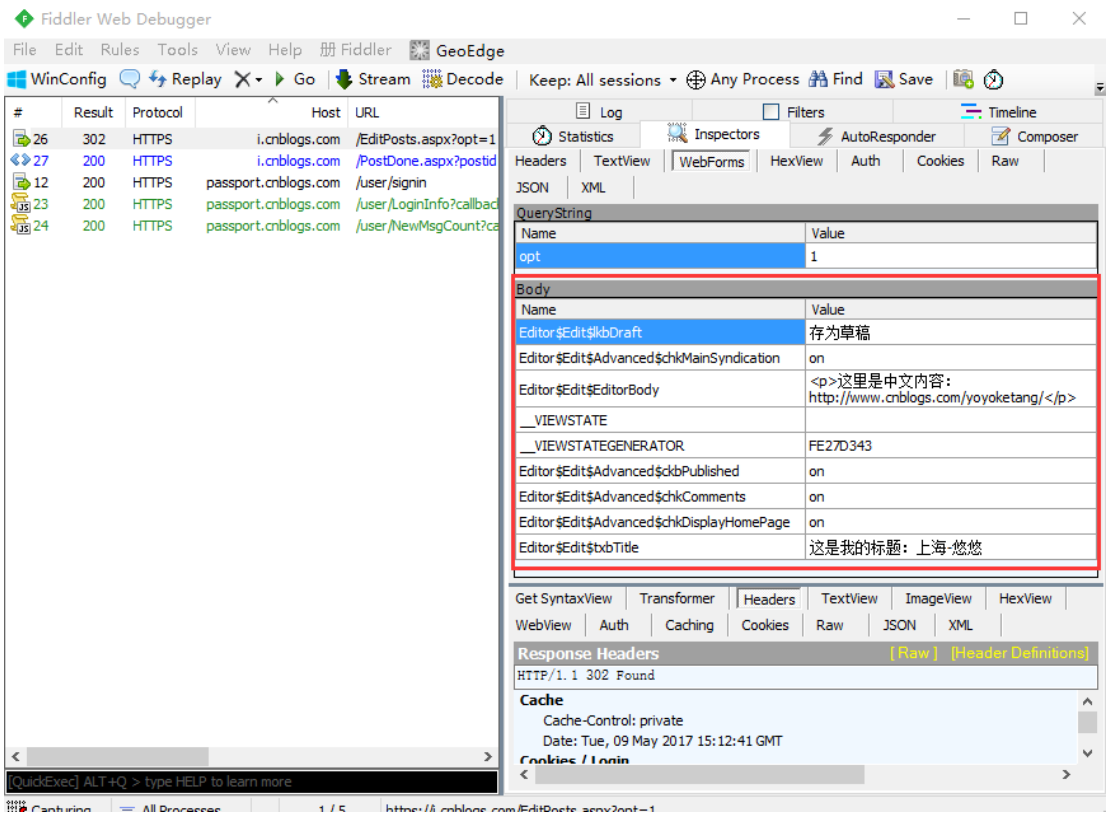
```
# coding:utf-8
import requests
url = "https://passport.cnblogs.com/user/signin"

headers = {
          头部信息已省略

          }
payload = {"input1":"xxx",
          "input2":"xxx",
          "remember":True}
```

（注：由于博客园账号密码不能登录了，所以这里不能 copy）

```
# r = requests.post(url, json=payload, headers=headers,verify=False)

# 修改后如下
s = requests.session()
r = s.post(url, json=payload, headers=headers,verify=False)
print r.json()
```

2.6.3 保存编辑

1.先打开我的随笔，手动输入内容后，打开 fiddler 抓包



2. 把 body 的参数内容写成字典格式，有几个空的参数不是必填的，可以去掉

```
body = {"__VIEWSTATE": "",
        "__VIEWSTATEGENERATOR":"FE27D343",
        "Editor$Edit$txbTitle":"这是我的标题：上海-悠悠",
        "Editor$Edit$EditorBody":"<p>这里是中文内容：
http://www.cnblogs.com/yoyoketang/</p>",
        "Editor$Edit$Advanced$ckbPublished":"on",
        "Editor$Edit$Advanced$chkDisplayHomePage":"on",
        "Editor$Edit$Advanced$chkComments":"on",
        "Editor$Edit$Advanced$chkMainSyndication":"on",
        "Editor$Edit$lkbDraft":"存为草稿",
        }
```

3. 用上面的 session 继续发送 post 请求

```
url2= "https://i.cnblogs.com/EditPosts.aspx?opt=1"
body = {"__VIEWSTATE": "",
        "__VIEWSTATEGENERATOR":"FE27D343",
        "Editor$Edit$txbTitle":"这是我的标题：上海-悠悠",
        "Editor$Edit$EditorBody":"<p>这里是中文内容：http://www.cnblogs.com/yoyoketang/</p>",
        "Editor$Edit$Advanced$ckbPublished":"on",
        "Editor$Edit$Advanced$chkDisplayHomePage":"on",
        "Editor$Edit$Advanced$chkComments":"on",
        "Editor$Edit$Advanced$chkMainSyndication":"on",
        "Editor$Edit$lkbDraft":"存为草稿",
         }
r2 = s.post(url2, data=body, verify=False)
print r.content
```

4.执行后，查看我的草稿箱就多了一条新增的了

2.6.4 参考代码

（以下代码，我把账号密码登录换成 cookie 登录了）

```
# coding:utf-8
import requests

# 先打开登录首页，获取部分 cookie
url = "https://passport.cnblogs.com/user/signin"
headers = {
          "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
```

```
Gecko/20100101 Firefox/44.0"
          }    # get 方法其它加个 ser-Agent 就可以了

s = requests.session()
r = s.get(url, headers=headers,verify=False)
print s.cookies

# 添加登录需要的两个 cookie
c = requests.cookies.RequestsCookieJar()

c.set('.CNBlogsCookie', '这里是抓到的')    # 填上面抓包内容
c.set('.Cnblogs.AspNetCore.Cookies','这里是抓到的')    # 填上面抓包内容
c.set('AlwaysCreateItemsAsActive',"True")
c.set('AdminCookieAlwaysExpandAdvanced',"True")
s.cookies.update(c)
print s.cookies

# 登录成功后保存编辑内容
r1 = s.get("https://i.cnblogs.com/EditPosts.aspx?opt=1", headers=headers,
verify=False)

# 保存草稿箱
url2= "https://i.cnblogs.com/EditPosts.aspx?opt=1"
body = {"__VIEWSTATE": "",
        "__VIEWSTATEGENERATOR":"FE27D343",
        "Editor$Edit$txbTitle":"这是 3111",
        "Editor$Edit$EditorBody":"<p>这里 111：
http://www.cnblogs.com/yoyoketang/</p>",
        "Editor$Edit$Advanced$ckbPublished":"on",
        "Editor$Edit$Advanced$chkDisplayHomePage":"on",
        "Editor$Edit$Advanced$chkComments":"on",
        "Editor$Edit$Advanced$chkMainSyndication":"on",
        "Editor$Edit$Advanced$txbEntryName":"",
        "Editor$Edit$Advanced$txbExcerpt":"",
        "Editor$Edit$Advanced$tbEnryPassword":"",
        "Editor$Edit$lkbDraft":"存为草稿",
                }
r2 = s.post(url2, data=body, verify=False)
print r.content
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

2.7 cookie 绕过验证码登录

前言

有些登录的接口会有验证码：短信验证码，图形验证码等，这种登录的话验证码参数可以从后台获取的（或者查数据库最直接）。

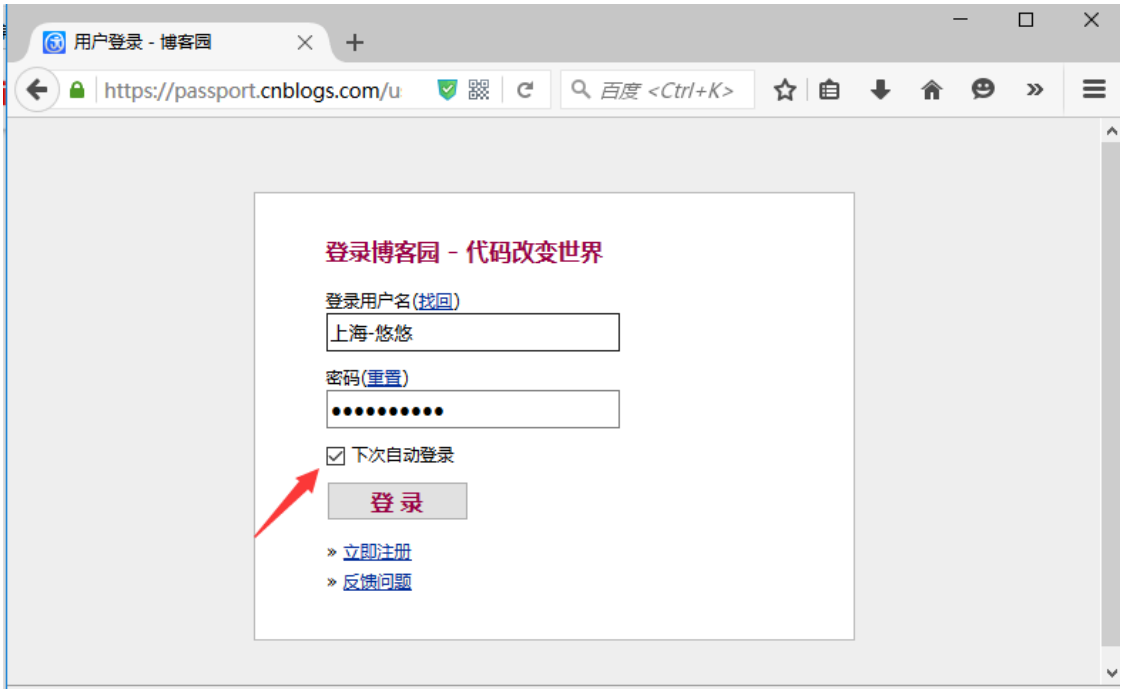
获取不到也没关系，可以通过添加 cookie 的方式绕过验证码。

2.7.1 抓登录 cookie

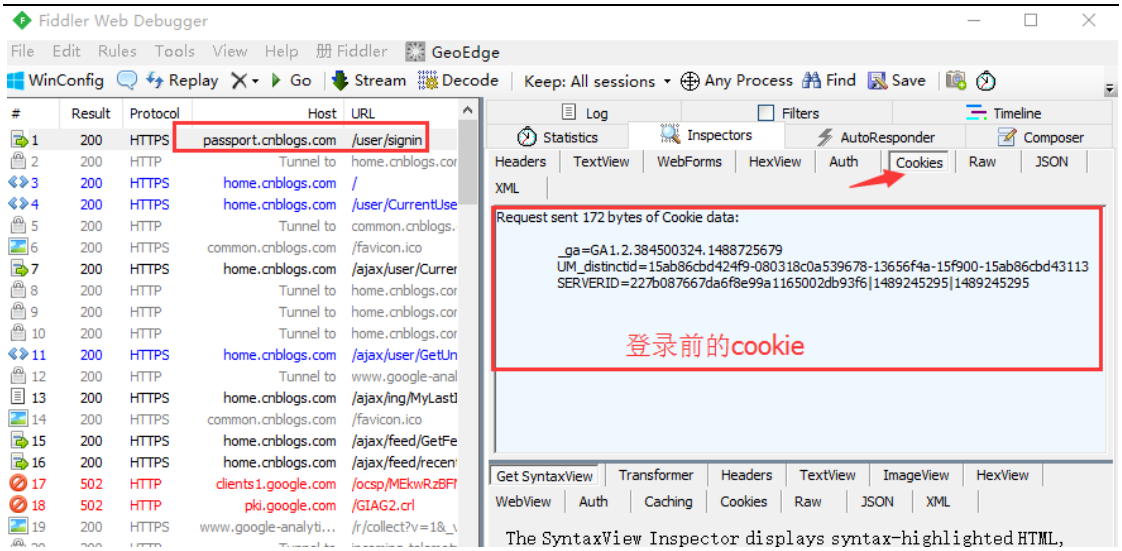
1. 登录后会生成一个已登录状态的 cookie，那么只需要直接把这个值添加到 cookies 里面就可以了。

2. 可以先手动登录一次，然后抓取这个 cookie，这里就需要用抓包工具 fiddler 了

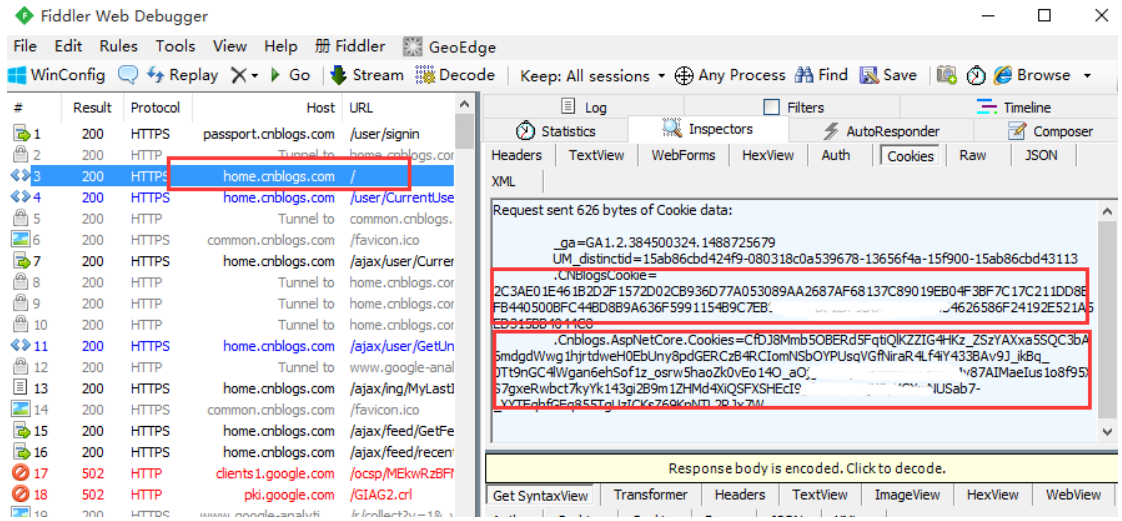
3. 先打开博客园登录界面，手动输入账号和密码（勾选下次自动登录）



4. 打开 fiddler 抓包工具，刷新下登录首页，就是登录前的 cookie 了



5. 登录成功后，再查看 cookie 变化，发现多了两组参数，多的这两组参数就是我们想要的，copy 出来，一会有用



（**最新的博客园这里应该是多出四个 cookie**）

2.7.2 cookie 组成结构

1. 用抓包工具 fidller 只能看到 cookie 的 name 和 value 两个参数，实际上 cookie 还有其它参数

2. 以下是一个完整的 cookie 组成结构

```
cookie ={u'domain': u'.cnblogs.com',
         u'name': u'.CNBlogsCookie',
         u'value': u'xxxx',
```

```
        u'expiry': 1491887887,
        u'path': u'/',
        u'httpOnly': True,
        u'secure': False}
```

name：cookie 的名称

value：cookie 对应的值，动态生成的

domain：服务器域名

expiry：Cookie 有效终止日期

path：Path 属性定义了 Web 服务器上哪些路径下的页面可获取服务器设置的 Cookie

httpOnly：防脚本攻击

secure:在 Cookie 中标记该变量，表明只有当浏览器和 Web Server 之间的通信协议为加密认证协议时，

浏览器才向服务器提交相应的 Cookie。当前这种协议只有一种，即为 HTTPS。

2.7.3 添加 cookie

1.往 session 里面添加 cookie 可以用以下方式

2.set 里面参数按括号里面的参数格式

```
coo =  requests.cookies.RequestsCookieJar()

coo.set('cookie-name',  'cookie-value',  path='/',  domain='.xxx.com')

s.cookies.update(c)
```

3.于是添加登录的 cookie，把第一步 fiddler 抓到的内容填进去就可以了

```
c = requests.cookies.RequestsCookieJar()
c.set('.CNBlogsCookie', 'xxx')
c.set('.Cnblogs.AspNetCore.Cookies','xxx')
s.cookies.update(c)
print(s.cookies)
```

```
# 添加登录需要的两个cookie
c = requests.cookies.RequestsCookieJar()

c.set('.CNBlogsCookie', '290F68DB39BCA779114898C3047A10998EFF8B72B82C3
c.set('.Cnblogs.AspNetCore.Cookies','CfDJ8Mmb5OBERd5FqtiQlKZZIG5_ynkMR
s.cookies.update(c)
print s.cookies
```

4.(敲黑板！！！)由于近期博客园的登录机制变了，这里需要多加 2 个 cookie 参数

```
c.set('AlwaysCreateItemsAsActive',"True")
c.set('AdminCookieAlwaysExpandAdvanced',"True")
```

2.7.4 参考代码

1.由于登录时候是多加 2 个 cookie,我们可以先用 get 方法打开登录首页,获取部分 cookie

2.再把登录需要的 cookie 添加到 session 里

3.添加成功后，随便编辑正文和标题保存到草稿箱

```
# coding:utf-8
import requests

# 先打开登录首页，获取部分 cookie
url = "https://passport.cnblogs.com/user/signin"
headers = {
                        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0"
                    }   # get 方法其它加个 ser-Agent 就可以了

s = requests.session()
r = s.get(url, headers=headers,verify=False)
print s.cookies

# 添加登录需要的两个 cookie
c = requests.cookies.RequestsCookieJar()

c.set('.CNBlogsCookie', '这里是抓到的')   # 填上面抓包内容
c.set('.Cnblogs.AspNetCore.Cookies','这里是抓到的')   # 填上面抓包内容
```

```
c.set('AlwaysCreateItemsAsActive',"True")
c.set('AdminCookieAlwaysExpandAdvanced',"True")
s.cookies.update(c)
print s.cookies

# 登录成功后保存编辑内容
r1 = s.get("https://i.cnblogs.com/EditPosts.aspx?opt=1", headers=headers,
verify=False)

# 保存草稿箱
url2= "https://i.cnblogs.com/EditPosts.aspx?opt=1"
body = {"__VIEWSTATE": "",
                "__VIEWSTATEGENERATOR":"FE27D343",
                "Editor$Edit$txbTitle":"这是 3111",
                "Editor$Edit$EditorBody":"<p>这里 111：
http://www.cnblogs.com/yoyoketang/</p>",
                "Editor$Edit$Advanced$ckbPublished":"on",
                "Editor$Edit$Advanced$chkDisplayHomePage":"on",
                "Editor$Edit$Advanced$chkComments":"on",
                "Editor$Edit$Advanced$chkMainSyndication":"on",
                "Editor$Edit$Advanced$txbEntryName":"",
                "Editor$Edit$Advanced$txbExcerpt":"",
                "Editor$Edit$Advanced$tbEnryPassword":"",
                "Editor$Edit$lkbDraft":"存为草稿",
                  }
r2 = s.post(url2, data=body, verify=False)
print r.content
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

2.8 Json 数据处理

前言

有些 post 的请求参数是 json 格式的，这个前面第二篇 post 请求里面提到过，需要导入 json 模块处理。

一般常见的接口返回数据也是 json 格式的，我们在做判断时候，往往只需要提取其中几个关键的参数就行，这时候就需要 json 来解析返回的数据了。

2.8.1 json 模块简介

1.Json 简介：Json，全名 JavaScript Object Notation，是一种轻量级的数据交换格式，常用于 http 请求中

2.可以用 help(json),查看对应的源码注释内容

Encoding basic Python object hierarchies::

```
>>> import json
>>> json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
'["foo", {"bar": ["baz", null, 1.0, 2]}]'
>>> print json.dumps("\"foo\bar")
"\"foo\bar"
>>> print json.dumps(u'\u1234')
"\u1234"
>>> print json.dumps('\\')
"\\"
>>> print json.dumps({"c": 0, "b": 0, "a": 0}, sort_keys=True)
{"a": 0, "b": 0, "c": 0}
>>> from StringIO import StringIO
>>> io = StringIO()
>>> json.dump(['streaming API'], io)
>>> io.getvalue()
'["streaming API"]'
```

2.8.2 Encode（python->json）

1.首先说下为什么要 encode,python 里面 bool 值是 True 和 False,json 里面 bool 值是 true 和 false,并且区分大小写，这就尴尬了，明明都是 bool 值。

在 python 里面写的代码,传到 json 里,肯定识别不了,所以需要把 python 的代码经过 encode 后成为 json 可识别的数据类型。

2.举个简单例子,下图中 dict 类型经过 json.dumps()后变成 str,True 变成了 true,False 变成了 fasle

```python
# coding:utf-8
import requests
import json

# pytho的字典
payload = {"yoyo": True,
           "json": False,
           "python": "226296743",
           }
print type(payload)
# 转化成json格式
data_json = json.dumps(payload)
print type(data_json)
print data_json
```

```
test_json
D:\test\python2\python.exe D:/test/yoyotest/test_post/test_json.py
<type 'dict'>
<type 'str'>
{"python": "226296743", "json": false, "yoyo": true}

Process finished with exit code 0
```

3. 以下对应关系表是从 json 模块的源码里面爬出来的. python 的数据类，经过 encode 成 json 的数据类型，对应的表如下

Python	JSON
dict	object
list, tuple	array
str, unicode	string
int, long, float	number
True	true
False	false
None	null

2.8.3 decode(json->python)

1. 以第三篇的登录成功结果：{"success":true}为例，我们其实最想知道的是 success 这个字段返回的是 True 还是 False

2. 如果以 content 字节输出，返回的是一个字符串：{"success":true}，这样获取后面那个结果就不方便了

3. 如果经过 json 解码后，返回的就是一个字典：{u'success': True}，这样获取后面那个结果，就用字典的方式去取值：result2["success"]

```
r2 = s.post(url2, data=body, verify=False)
# content是字节数据输出
result1 = r.content
print type(result1)
print result1

# json是经过encode成对应python数据类型的
result2 = r.json()
print type(result2)   # python接口自动化 226296743
print result2
print result2["success"]
```

```
send_session
  InsecureRequestWarning)
{u'success': True}
D:\test\python2\lib\site-packages\requests\packages\urllib3\connectionpool.py:852: InsecureR
  InsecureRequestWarning)
<type 'str'>
{"success":true}
<type 'dict'>
{u'success': True}
True
```

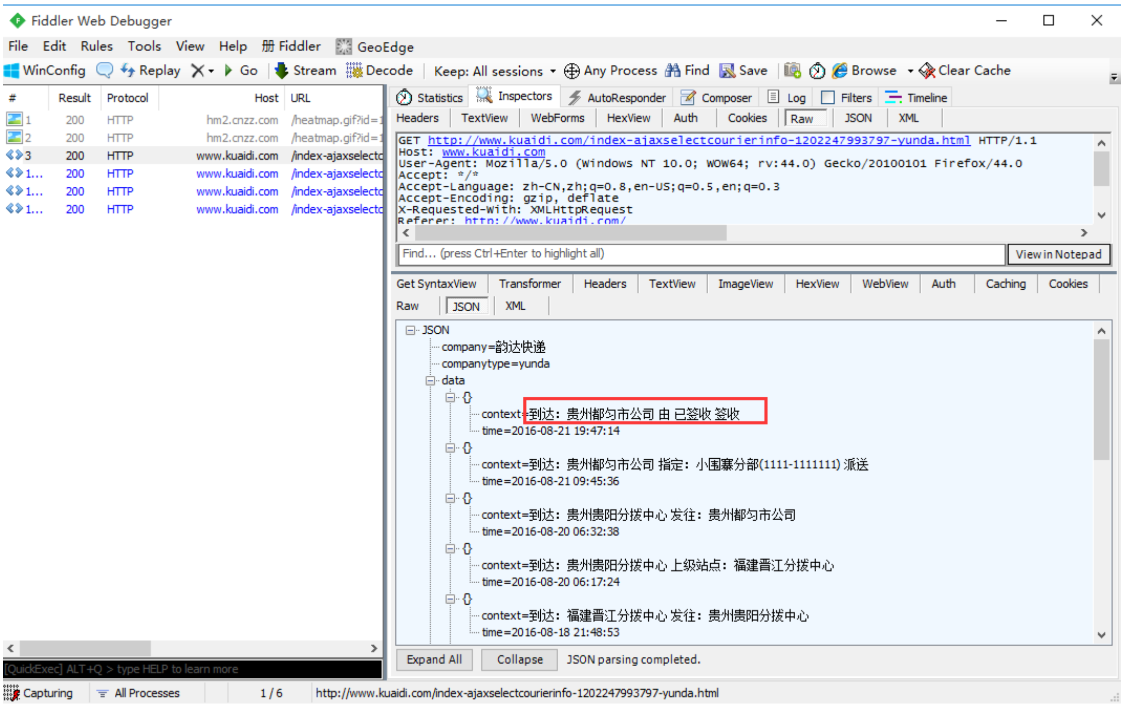
4. 同样 json 数据转化成 python 可识别的数据，对应的表关系如下

JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long

```
          |   | number (real)    | float                        |
          |  +----------------+-------------------+
          |   | true                          |
True                         |
          |  +----------------+-------------------+
          |   | false                     | False                        |
          |  +----------------+-------------------+
          |   | null                        |
None                         |
          |  +----------------+-------------------+
```

2.8.4 案例分析

1.比如打开快递网：http://www.kuaidi.com/，搜索某个单号，判断它的状态是不是已签收



2. 实现代码如下

2.8.5 参考代码

```
# coding:utf-8
import requests

url =
"http://www.kuaidi.com/index-ajaxselectcourierinfo-1202247993797-yunda.html"
headers = {
          "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0"
          }    # get 方法加个 ser-Agent 就可以了

s = requests.session()
r = s.get(url, headers=headers,verify=False)
result = r.json()
data = result["data"]      # 获取 data 里面内容
print data
print data[0]                  # 获取 data 里最上面有个
get_result = data[0]['context']    # 获取已签收状态
print get_result
```

```
if u"已签收" in get_result:
        print "快递单已签收成功"
else:
        print "未签收"
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

2.9 重定向（Location）

前言

某屌丝男 A 鼓起勇气向女神 B 打电话表白，女神 B 是个心机婊觉得屌丝男 A 是好人，不想直接拒绝于是设置呼叫转移给闺蜜 C 了，最终屌丝男 A 和女神闺蜜 C 表白成功了，这种场景其实就是重定向了。

2.9.1 重定向

1. (Redirect)就是通过各种方法将各种网络请求重新定个方向转到其它位置,从地址 A 跳转到地址 B 了。

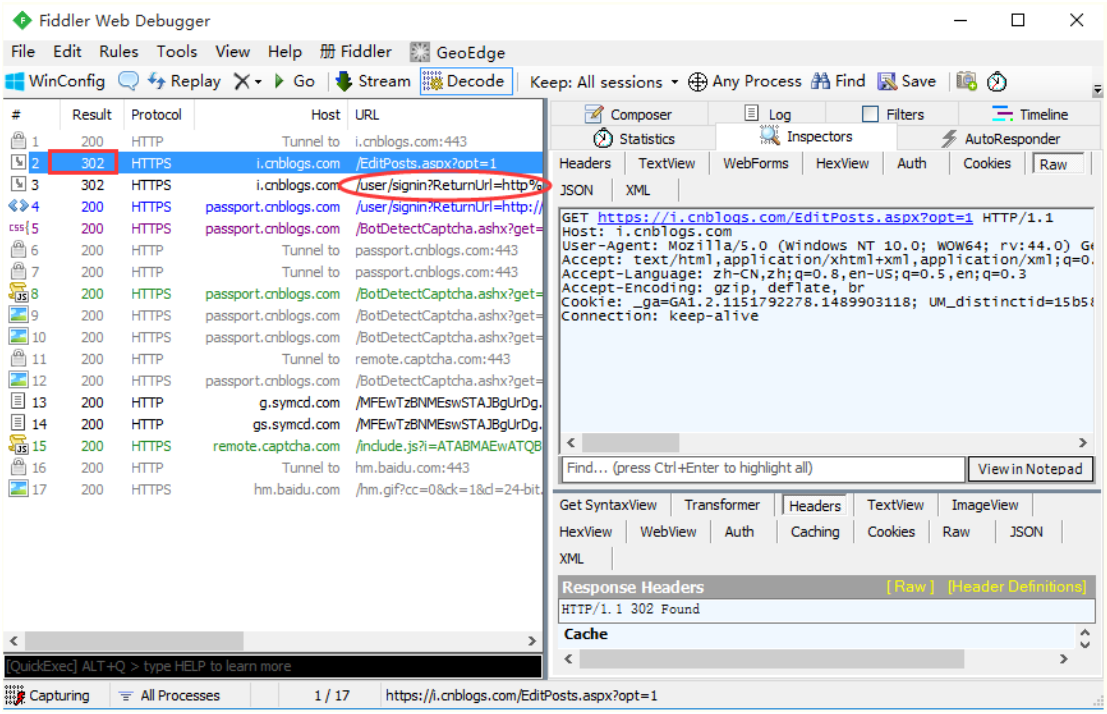
2.重定向状态码:

--301 redirect: 301 代表永久性转移(Permanently Moved)

--302 redirect: 302 代表暂时性转移(Temporarily Moved)

3.举个简单的场景案例，先登录博客园打开我的博客首页，进我的随笔编辑界面，记住这个地址：https://i.cnblogs.com/EditPosts.aspx?opt=1

4.退出博客园登录，把刚才我的随笔这个地址输入浏览器回车，抓包会看到这个请求状态码是 302，浏览器地址栏瞬间刷新跳到登录首页去了



2.9.2 禁止重定向（allow_redirects）

1. 用 get 方法请求：https://i.cnblogs.com/EditPosts.aspx?opt=1

2. 打印状态码是 200，这是因为 requets 库自动处理了重定向请求了

```
# coding:utf-8
import requests

# 请求头
headers = {
     "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefo
          }
s = requests.session()
# 打开我的随笔
r = s.get('https://i.cnblogs.com/EditPosts.aspx?opt=1',
          headers=headers,
          verify=False)
# 打印状态码，自动处理重定向请求
print r.status_code
```

```
test_get
InsecureRequestWarning)
D:\test\python2\lib\site-packages\requests\packages\urllib3\connectionpool.py:852: InsecureRequestWarning:
  InsecureRequestWarning)
200
```

3. 自动处理重定向地址后，我们就获取不到重定向后的 url 了，就无法走下一步，这里我们可以设置一个参数禁止重定向：allow_redirects=False

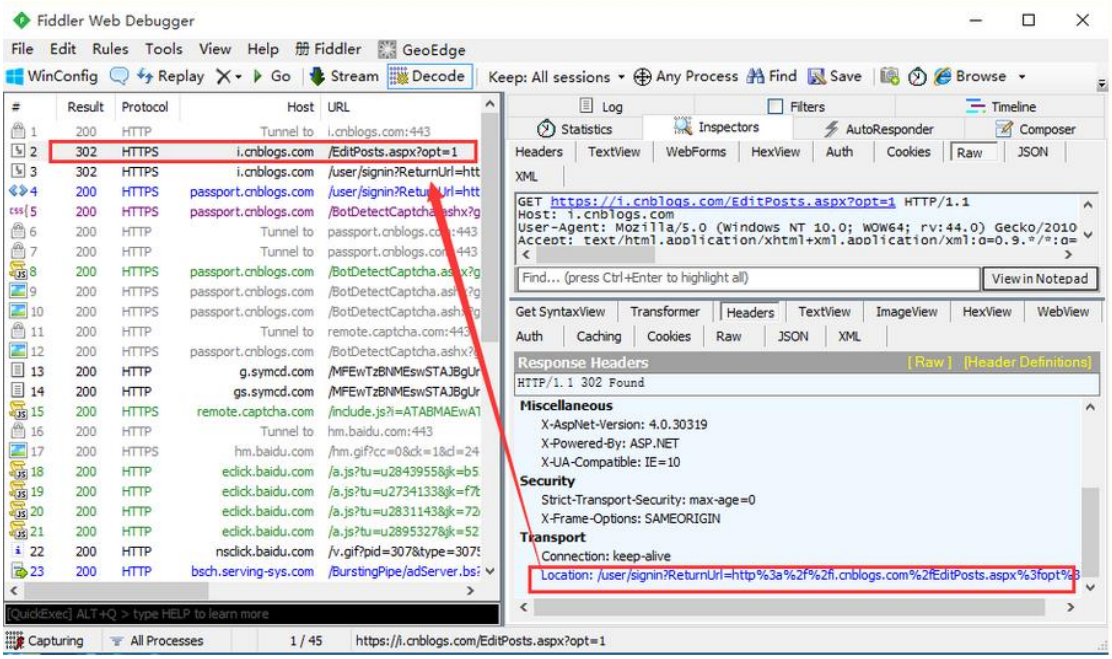
（allow_redirects=True 是启动重定向），然后就可以看到 status_code 是 302 了

```python
# coding:utf-8
import requests

# 请求头
headers = {
     "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firef
          }
s = requests.session()
# 打开我的随笔
r = s.get('https://i.cnblogs.com/EditPosts.aspx?opt=1',
          headers=headers,
          allow_redirects=False,
          verify=False)
# 打印状态码，自动处理重定向请求
print r.status_code
```

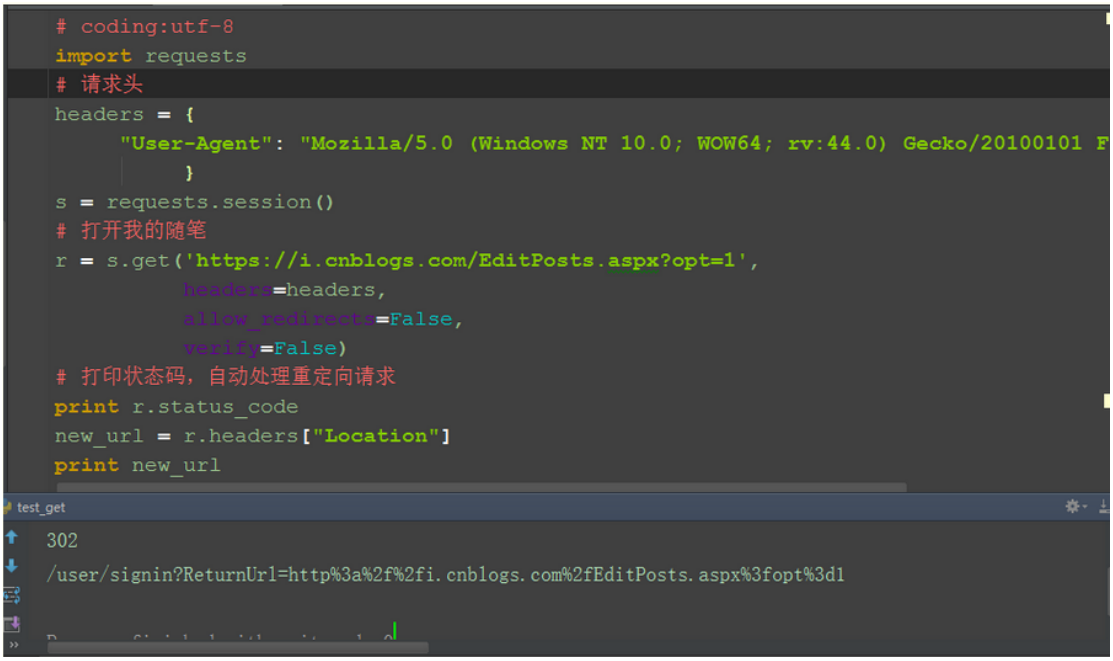
```
test_get
InsecureRequestWarning)
302
```

2.9.3 获取重定向后地址

1.在第一个请求后，服务器会下发一个新的请求链接，在 response 的 headers 里，如下抓包：Location



2.用脚本去获取 Location 地址



2.9.4 参考代码

```
# coding:utf-8
import requests
# 请求头
headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0"
                    }
s = requests.session()
# 打开我的随笔
r = s.get('https://i.cnblogs.com/EditPosts.aspx?opt=1',
                    headers=headers,
                    allow_redirects=True,
                    verify=False)
# 打印状态码，自动处理重定向请求
print r.status_code
new_url = r.headers["Location"]
print new_url
```

2.10 参数关联

前言

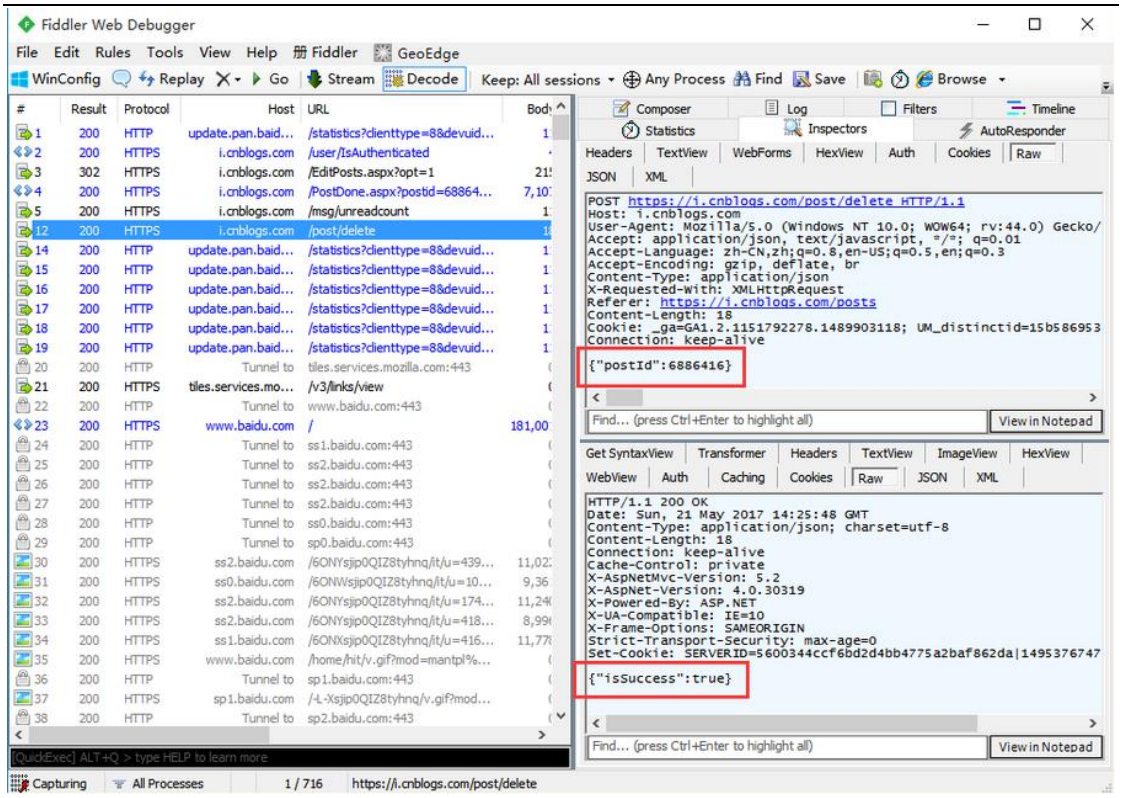
我们用自动化发帖之后，要想接着对这篇帖子操作，那就需要用参数关联了，发帖之后会有一个帖子的 id，获取到这个 id，继续操作传这个帖子 id 就可以了

（博客园的登录机制已经变了，不能用账号和密码登录了，换个网站，或者用 cookie 登录吧）

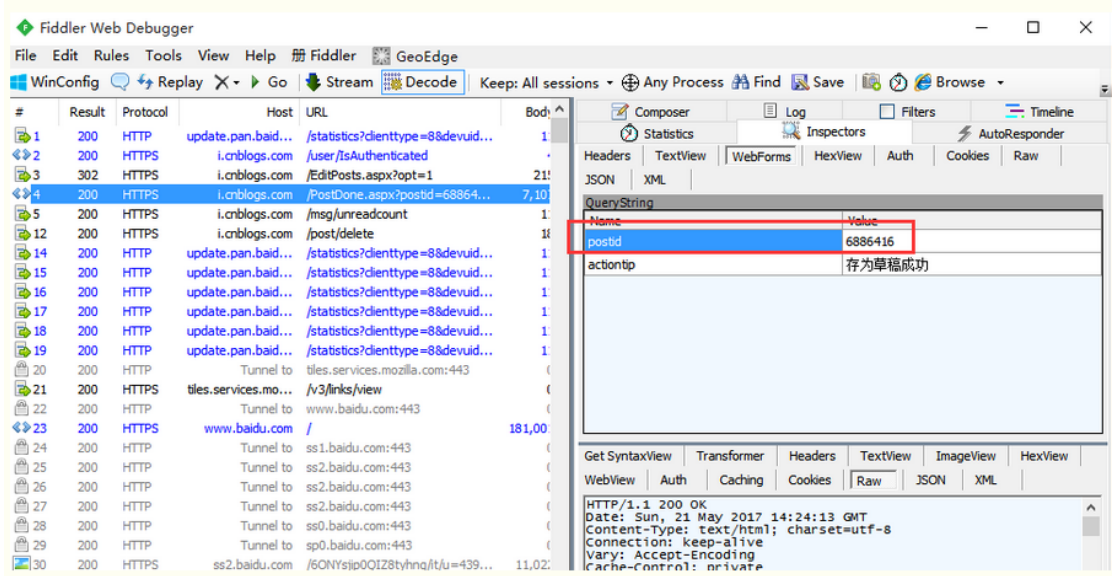
2.10.1 删除草稿箱

1.我们前面讲过登录后保存草稿箱，那可以继续接着操作：删除刚才保存的草稿

2.用 fiddler 抓包，抓到删除帖子的请求，从抓包结果可以看出，传的 json 参数是 postId



3. 这个 postId 哪里来的呢？可以看上个请求 url 地址



4. 也就是说保存草稿箱成功之后，重定向一个 url 地址，里面带有 postId 这个参数。那接下来我们提取出来就可以了

2.10.2 提取参数

1.我们需要的参数 postId 是在保存成功后 url 地址，这时候从 url 地址提出对应的参数值就行了，先获取保存成功后 url

2.通过正则提取需要的字符串，这个参数值前面（postid=）和后面（&）字符串都是固定的

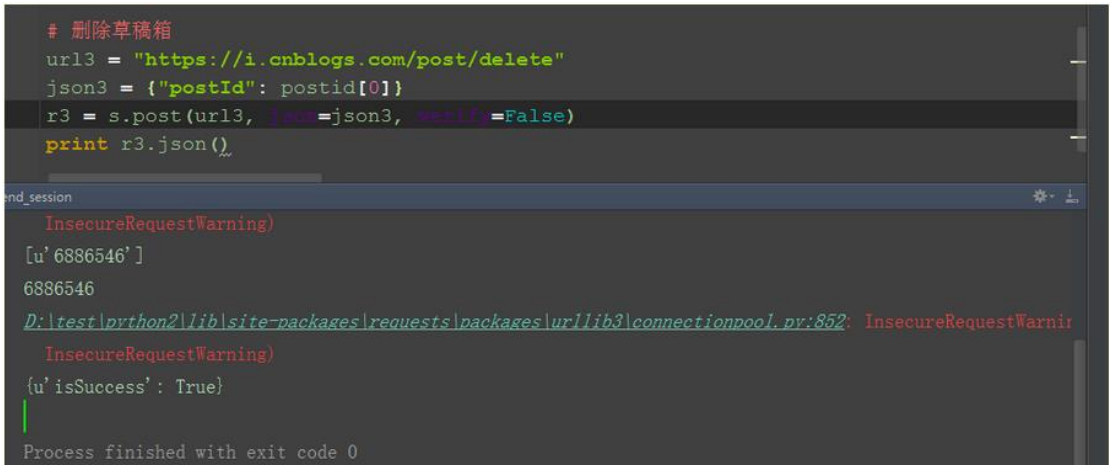
3.这里正则提出来的是 list 类型，取第一个值就可以是字符串了（注意：每次保存需要修改内容，不能重复）

2.10.3 传参

1.删除草稿箱的 json 参数传上面取到的参数：{"postId": postid[0]}

2.json 数据类型 post 里面填 json 就行，会自动转 json

3.接着前面的保存草稿箱操作，就可以删除成功了



2.10.4 参考代码

（敲黑板！！！由于博客园的登录机制变了，后面所有的登录地方全部用 cookie 登录）

```
# coding:utf-8
import requests

# 先打开登录首页，获取部分 cookie
url = "https://passport.cnblogs.com/user/signin"
headers = {
                        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0"
                    }   # get 方法其它加个 ser-Agent 就可以了

s = requests.session()
r = s.get(url, headers=headers,verify=False)
print s.cookies

# 添加登录需要的两个 cookie
c = requests.cookies.RequestsCookieJar()

c.set('.CNBlogsCookie', '这里是抓到的')   # 填上面抓包内容
c.set('.Cnblogs.AspNetCore.Cookies','这里是抓到的')   # 填上面抓包内容
c.set('AlwaysCreateItemsAsActive',"True")
c.set('AdminCookieAlwaysExpandAdvanced',"True")
s.cookies.update(c)
print s.cookies
```

```
# ----------登录全部走 cookie 登录---


# 第二步：保存草稿
url2 = "https://i.cnblogs.com/EditPosts.aspx?opt=1"
body = {"__VIEWSTATE": "",
        "__VIEWSTATEGENERATOR":"FE27D343",
        "Editor$Edit$txbTitle":"这是 3111",
        "Editor$Edit$EditorBody":"<p>这里 111：
http://www.cnblogs.com/yoyoketang/</p>",
        "Editor$Edit$Advanced$ckbPublished":"on",
        "Editor$Edit$Advanced$chkDisplayHomePage":"on",
        "Editor$Edit$Advanced$chkComments":"on",
        "Editor$Edit$Advanced$chkMainSyndication":"on",
        "Editor$Edit$Advanced$txbEntryName":"",
        "Editor$Edit$Advanced$txbExcerpt":"",
        "Editor$Edit$Advanced$tbEnryPassword":"",
        "Editor$Edit$lkbDraft":"存为草稿",
        }

r2 = s.post(url2, data=body, verify=False)
# 获取当前 url 地址
print r2.url

# 第三步：正则提取需要的参数值
import re
postid = re.findall(r"postid=(.+?)&", r2.url)
print postid   # 这里是 list
# 提取为字符串
print postid[0]

# 第四步：删除草稿箱
url3 = "https://i.cnblogs.com/post/delete"
json3 = {"postId": postid[0]}
r3 = s.post(url3, json=json3, verify=False)
print r3.json()
```

2.11 token 登录

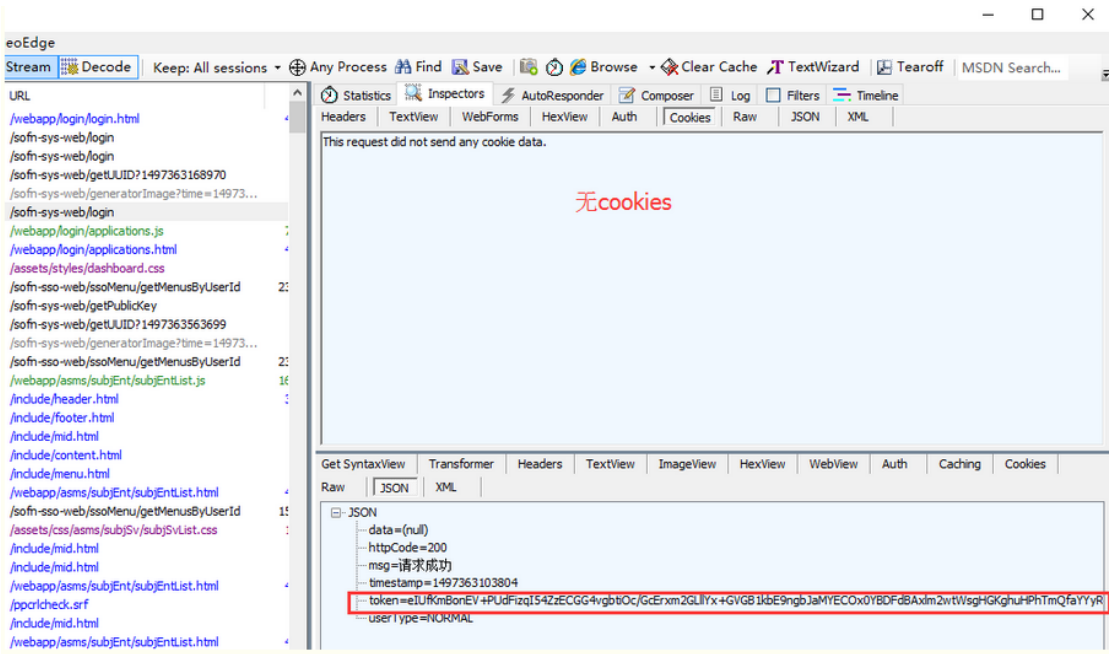
前言

有些登录不是用 cookie 来验证的，是用 token 参数来判断是否登录。

token 传参有两种一种是放在请求头里，本质上是跟 cookie 是一样的，只是换个单词而已；另外一种是在 url 请求参数里，这种更直观。

2.11.1 登录返回 token

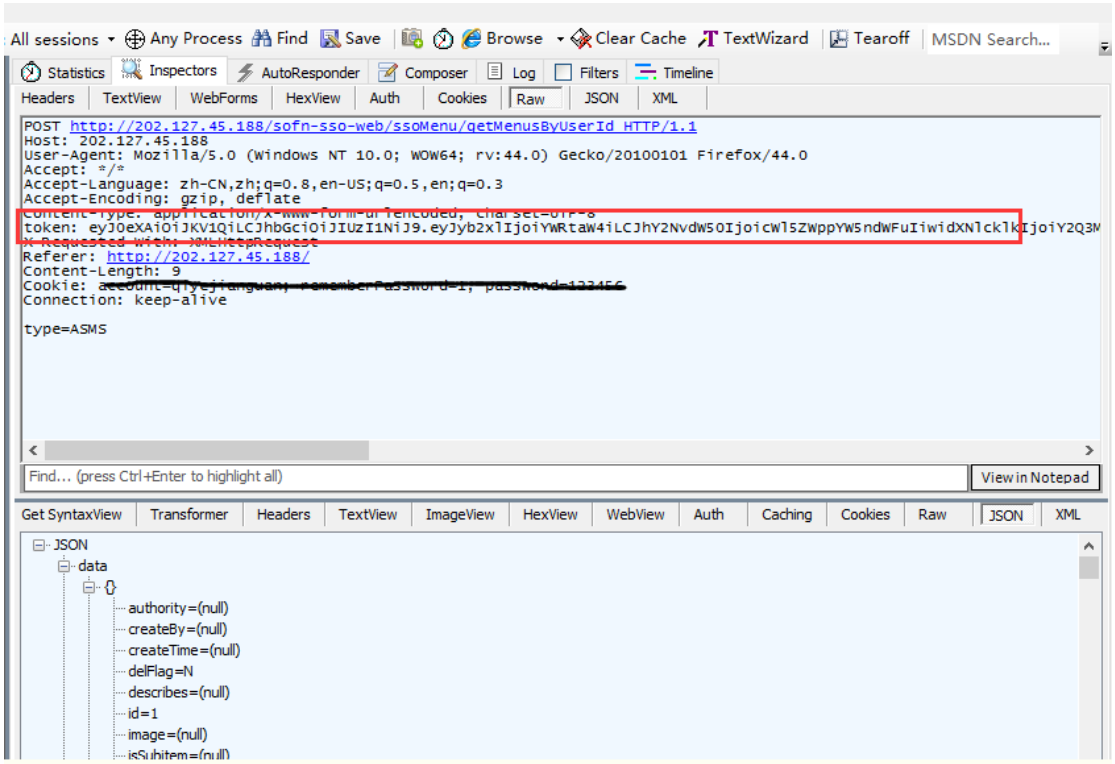
1. 如下图的这个登录，无 cookies



2. 但是登录成功后有返回 token

2.11.2 请求头带 token

1. 登录成功后继续操作其它页面，发现 post 请求的请求头，都会带 token 参数



2. 这种请求其实比 cookie 更简单，直接把登录后的 token 放到头部就行

2.11.3 token 关联

1. 用脚本实现登录，获取 token 参数，获取后传参到请求头就可以了

2. 如果登录有验证码，前面的脚本登录步骤就省略了，自己手动登录后获取 token

```
# coding:utf-8
import requests

header = {# 登录抓包获取的头部
          "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0",
          "Accept": "*/*",
          "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
          "Accept-Encoding": "gzip, deflate",
          "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
          "X-Requested-With": "XMLHttpRequest",
          "Content-Length": "423",
          "Connection": "keep-alive"
```

```
            }

body = {"key1": "value1",
        "key2": "value2"}   # 这里账号密码就是抓包的数据

s = requests.session()
login_url = "http://xxx.login"     #  自己找带 token 网址
login_ret = s.post(login_url, headers=header, data=body)
# 这里 token 在返回的 json 里，可以直接提取
token = login_ret.json()["token"]

# 这是登录后发的一个 post 请求
post_url = "http://xxx"
# 添加 token 到请求头
header["token"] = token
# 如果这个 post 请求的头部其它参数变了，也可以直接更新
header["Content-Length"]="9"
body1 = {
          "key": "value"
        }
post_ret = s.post(post_url, headers=header, data=body1)
print post_ret.content
```

2.12 案例分析(csrfToken)

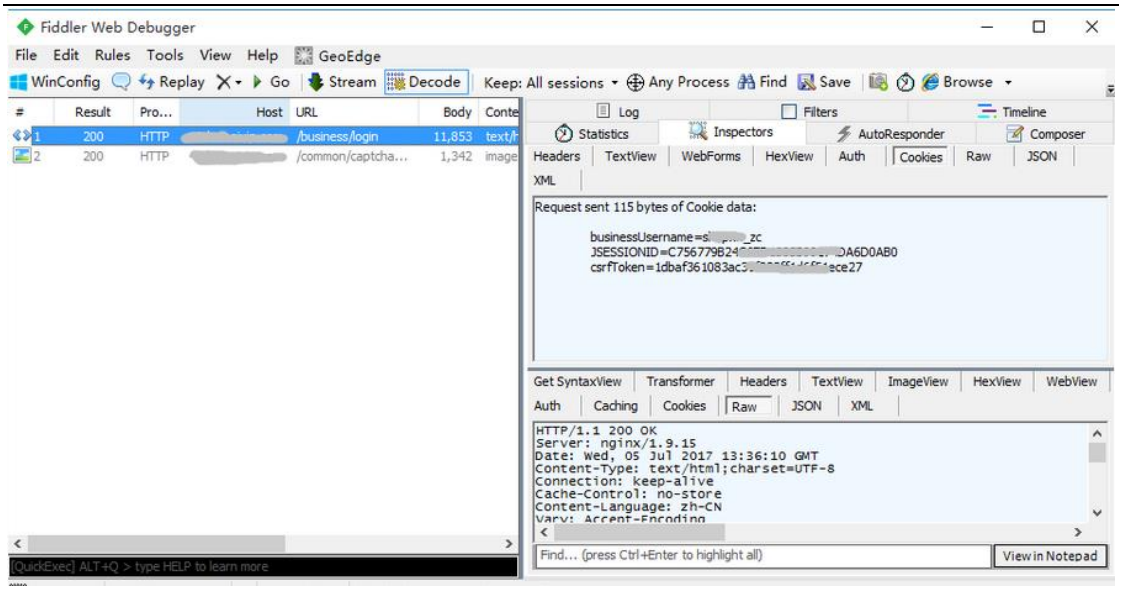
前言：

有些网站的登录方式跟前面讲的博客园和 token 登录会不一样，把 csrfToken 放到 cookie 里，登录前后 cookie 是没有任何变化的，这种情况下如何绕过前端的验证码登录呢？

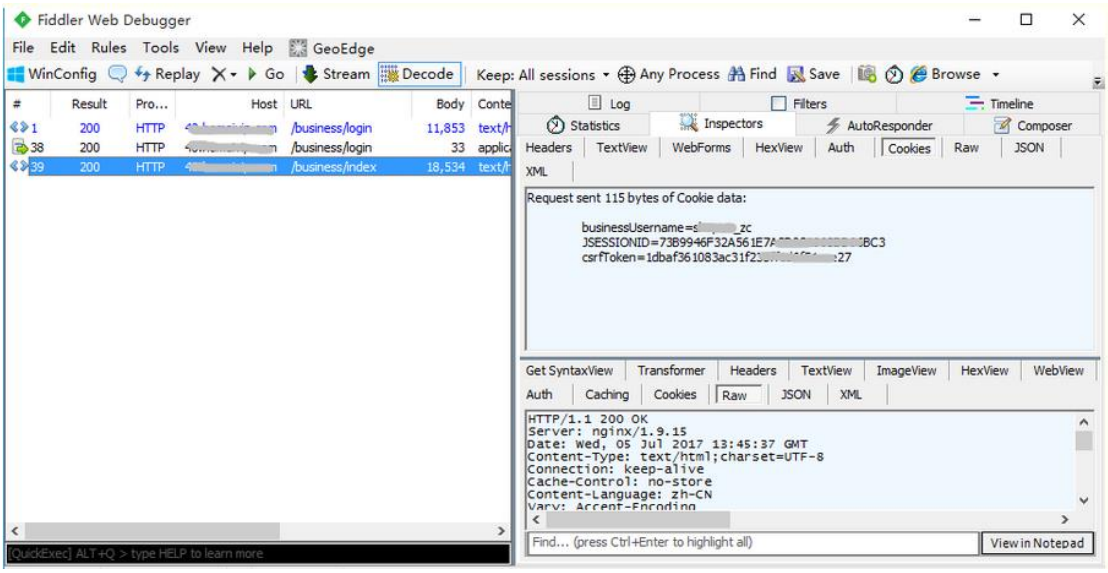
2.12.1 登录前后对比

1.如果登录页面有图形验证码，这种我们一般都是绕过登录的方式，如下图通过抓包分析，首先不输入密码，抓包

（由于这个是别人公司内部网站，所以网址不能公开，仅提供解决问题的思路）



2. 在登录页面输入账号和密码手动登录后，抓包信息如下

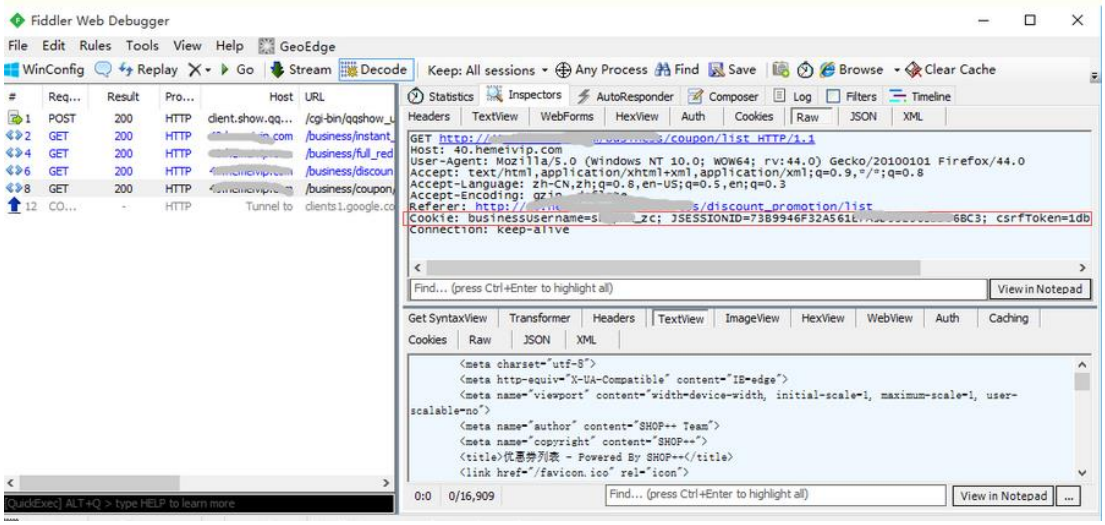


3. 抓包后 cookies 信息在登录前后没任何变化,这里主要有三个参数:

--businessUsername:这个是账号名称
--JSESSIONID: 这个是一串字符串，主要看这个会不会变（一般有有效期）copy 出来就行
--csrfToken: 这个是一串字符串，主要看这个会不会变（一般有有效期）copy 出来就行

2.12.2 get 请求

1.像这种登录方式的 get 请求，请求头部 cookie 没任何变化，这种可以直接忽略登录，不用管登录过程，直接发请求就行

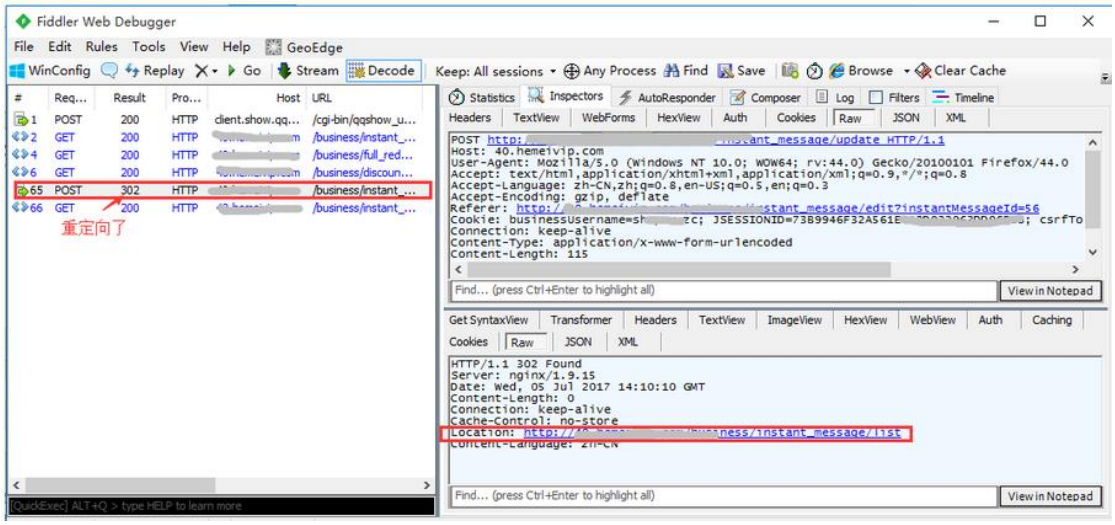


2.代码实现

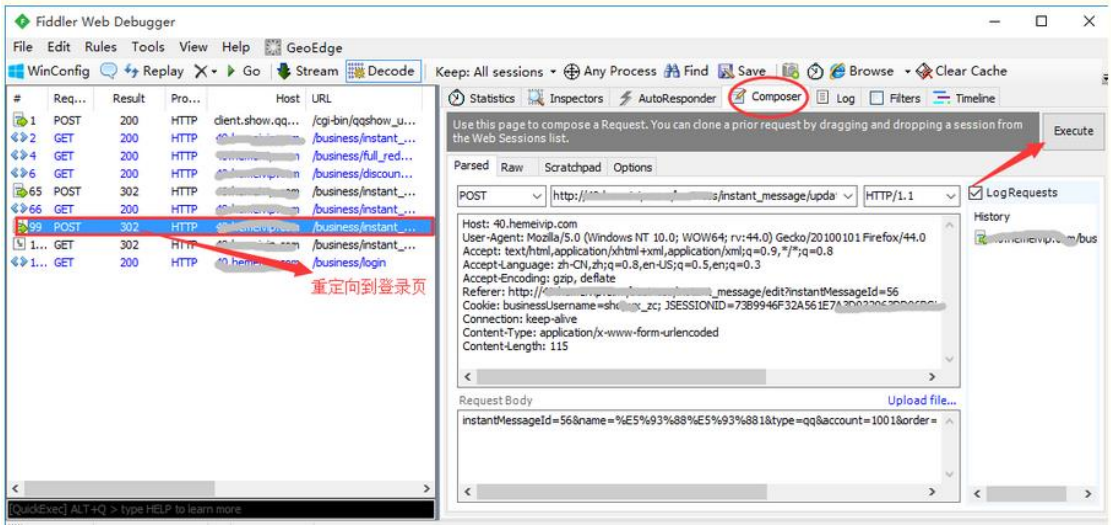
```
# coding:utf-8
import requests
# 优惠券列表
url = 'http://xxx/xxx/coupon/list'
h = {
"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0",
"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
"Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
"Accept-Encoding": "gzip, deflate",
"Cookie": "csrfToken=xxx(复制抓包的信息); JSESSIONID=xxx(复制抓包的信息); businessUsername=（用户名）",
"Connection": "keep-alive"
}
r = requests.get(url, headers=h)
print r.content
```

2.12.3 post 请求遇到的坑

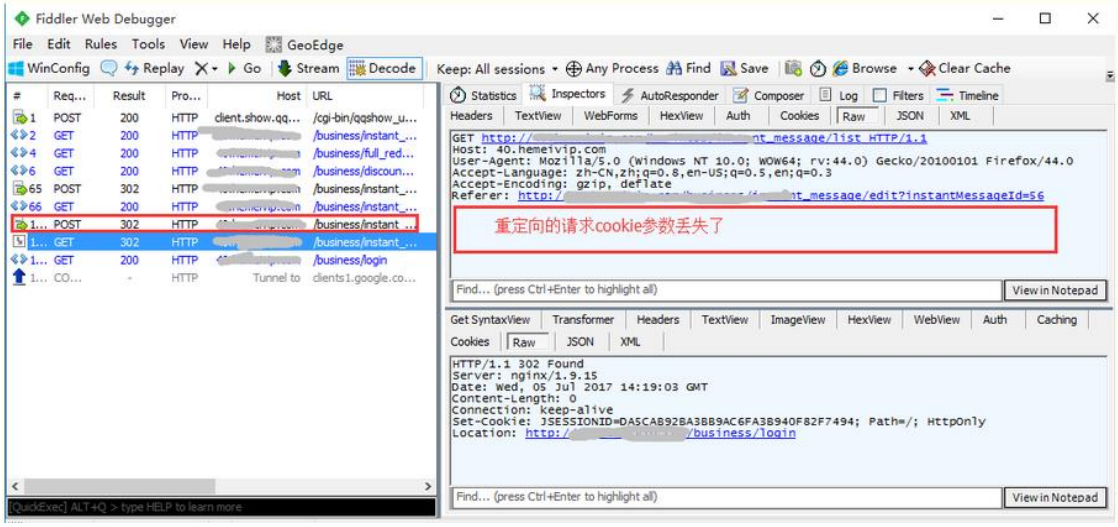
1.post 请求其实也可以忽略登录的过程，直接抓包把 cookie 里的三个参数（businessUsername、JSESSIONID、csrfToken）加到头部也是可以的。



2.但是这里遇到一个坑:用 Composer 发请求，重定向回到登录页了



3.主要原因：重定向的请求，cookie 参数丢失了



2.12.4 重定向

1.解决上面问题，其实很简单，把重定向禁用（具体看 2.8 重定向 Location 这篇）后的链接获取到，重新发个 get 请求，头部带上 cookies 的三个参数就行了

```
# coding:utf-8
import requests
# 主要是 post 请求后重定向，cookie 丢失，所以回到登录页面了
# 解决办法，禁止重定向，获取重定向的 url 后，重新发重定向的 url 地址请求就行了

# 三个主要参数
csrfToken = '获取到的 csrftoken，一般有有效期的'
jsessionId = '获取到的 jsessionid'
userName = '用户名'


url = 'http://xxx/xxxx/update'
h1 = {
        "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:46.0) Gecko/20100101
Firefox/46.0",
        "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
        "Accept-Encoding": "gzip, deflate",
        "Cookie": "csrfToken=%s; JSESSIONID=%s; businessUsername=%s" % (csrfToken,
jsessionId, userName),
        "Connection": "keep-alive",
```

```
        "Content-Type": "application/x-www-form-urlencoded",
        "Content-Length": "115"
        }

body = {"instantMessageId":"56",
         "name": u"哈哈 1",
         "order": "",
         "csrfToken": csrfToken,
         "type": "qq",
         "account": "1001"}

s = requests.session()
r1 = s.post(url, headers=h1, data=body, allow_redirects=False)
print r1.status_code
# 获取重定向的 url 地址
redirect_url = r1.headers["Location"]
print redirect_url

h2 = {
"User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:46.0) Gecko/20100101
Firefox/46.0",
"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
"Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
"Accept-Encoding": "gzip, deflate",
"Cookie": "csrfToken=%s; JSESSIONID=%s; businessUsername=%s" % (csrfToken,
jsessionId, userName),
"Connection": "keep-alive"
}
r2 = s.get(redirect_url, headers=h2)
print r2.content
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

2.13 requests 官方文档

官方文档：http://cn.python-requests.org/zh_CN/latest/

2.14 常见登录案例（github 源码下载）

github 上常见的一些网站登录案例参考：
1.知乎：
https://github.com/xchaoinfo/fuck-login/tree/master/001%20zhihu

2. 126：
https://github.com/xchaoinfo/fuck-login/tree/master/002%20126

3.微博：
https://github.com/xchaoinfo/fuck-login/tree/master/003%20weibo.cn

其它更多：
https://github.com/xchaoinfo/fuck-login

第 3 章 unitttest

3.1 unittest 简介

前言

熟悉 java 的应该都清楚常见的单元测试框架 Junit 和 TestNG，这个招聘的需求上也是经常见到的。python 里面也有单元测试框架-unittest,相当于是一个 python 版的 junit。

python 里面的单元测试框架除了 unittest,还有一个 pytest 框架，这个用的比较少，后面有空再继续分享。

3.1.1 unittest 简介

1.先导入 unittest

2.用 help 函数查看源码解析

3.查看描述：

Python unit testing framework, based on Erich Gamma's JUnit and Kent Beck's Smalltalk testing framework.

翻译：python 的单元测试框架，是基于 java 的 junit 测试框架

```
import unittest
print help(unittest)
```

```
(1)

 Simple usage:

     import unittest

     class IntegerArithmeticTestCase(unittest.TestCase):
         def testAdd(self):  ## test method names begin 'test*'
             self.assertEqual((1 + 2), 3)
             self.assertEqual(0 + 1, 1)
         def testMultiply(self):
             self.assertEqual((0 * 10), 0)
             self.assertEqual((5 * 8), 40)

     if __name__ == '__main__':
         unittest.main()
```

3.1.2 简单用法

1.可以把上图的这段代码 copy 出来，单独运行，看看测试结果。

```
Simple usage:

    import unittest

    class IntegerArithmeticTestCase(unittest.TestCase):
          def testAdd(self):   ## test method names begin 'test*'
                self.assertEqual((1 + 2), 3)
                self.assertEqual(0 + 1, 1)
          def testMultiply(self):
                self.assertEqual((0 * 10), 0)
                self.assertEqual((5 * 8), 40)

    if __name__ == '__main__':
          unittest.main()
```

2.第一行是导入 unittest 这个模块

3.class 这一行是定义一个测试的类，并继承 unittest.TestCase 这个类

4.接下来是定义了两个测试 case 名称:testAdd 和 testMultiply

5.注释里面有句话很重要，这个要敲下黑板记笔记了：**## test method names begin’test*’**

--翻译：测试用例的名称要以 test 开头

6.然后是断言 assert，这里的断言方法是 assertEqual-判断两个是否相等，**这个断言可以是一个也可以是多个**

7.if 下面的这个 unittest.main()是运行主函数，运行后会看到测试结果（跑了两个用例耗时 0.000 秒,两个用例都通过）：

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

3.1.3 小试牛刀

1.上面的两个案例是加法和乘法，我们可以写个 case 试下减法和除法。

2.有很多小伙伴不知道断言怎么写，断言其实就是拿实际结果和期望结果去对比，对比的方法很多，这里只是举的最简单的一个判断相等的方法。

```python
# coding:utf-8
import unittest

class Test(unittest.TestCase):

    def testMinus(self):    # test method names begin 'test*'
        u'''这里是测试减法'''
        result = 6-5  # 实际结果
        hope = 1      # 期望结果
        self.assertEqual(result, hope)

    def testDivide(self):
        u'''这里是测试除法'''
        result = 7/2    # 实际结果
        hope = 3.5      # 期望结果
        self.assertEqual(result, hope)

if __name__ == '__main__':
    unittest.main()  # 交流QQ群：232607095
```

3.最后运行结果，第二个是失败的，失败原因：AssertionError: 3 != 3.5

```
F.
======================================================================
FAIL: testDivide (__main__.Test)
这里是测试除法
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:/test/web-project/p.py", line 14, in testDivide
    self.assertEqual(result, hope)
AssertionError: 3 != 3.5

----------------------------------------------------------------------
Ran 2 tests in 0.000s

FAILED (failures=1)
```

3.1.4 前置和后置

1.setUp:在写测试用例的时候,每次操作其实都是基于打开浏览器输入对应网址这些操作,这个就是执行用例的前置条件。

2.tearDown：执行完用例后，为了不影响下一次用例的执行，一般有个数据还原的过程，这就是执行用例的后置条件。

3.很多人执行完用例，都不去做数据还原，以致于下一个用例执行失败，这就是不喜欢擦屁股的事情，习惯不好。

4.前置和后置都是非必要的条件，如果没有也可以写 pass

```python
# coding:utf-8
import unittest

class Test(unittest.TestCase):

    def setUp(self):
        pass  # 如果没有可以不写，或者pass代替

    def tearDown(self):
        pass

    def testMinus(self):    # test method names begin 'test*'
        u'''这里是测试减法'''
        result = 6-5  # 实际结果
        hope = 1      # 期望结果
        self.assertEqual(result, hope)
```

3.2 unittest 执行顺序

前言

很多初学者在使用 unittest 框架时候，不清楚用例的执行顺序到底是怎样的。对测试类里面的类和方法分不清楚，不知道什么时候执行，什么时候不执行。

本篇通过最简单案例详细讲解 unittest 执行顺序。

3.2.1 案例分析

1.先定义一个测试类，里面写几个简单的 case

```
# coding:utf-8
import unittest
import time
class Test(unittest.TestCase):
	def setUp(self):
		print "start!"

	def tearDown(self):
		time.sleep(1)
		print "end!"

	def test01(self):
		print "执行测试用例 01"

	def test03(self):
		print "执行测试用例 03"

	def test02(self):
		print "执行测试用例 02"

	def addtest(self):
		print "add 方法"

if __name__ == "__main__":
	unittest.main()
```

3.2.2 执行结果

```
D:\test\python2\python.exe D:/test/test01.py
start!
执行测试用例 01
.end!
start!
执行测试用例 02
end!
.start!
执行测试用例 03
end!
.
----------------------------------------------------------------------
Ran 3 tests in 3.001s

OK
```

3.2.3 结果分析

1.执行顺序：

start!-执行测试用例 01-end!

start!-执行测试用例 02-end!

start!-执行测试用例 03-end!

2.从执行结果可以看出几点

--先执行的前置 setUp，然后执行的用例(test*)，最后执行的后置 tearDown

--测试用例（test*）的执行顺序是根据 01-02-03 执行的，也就是说根据用例名称来顺序执行的

--addtest（self）这个方法没执行,说明只执行 test 开头的用例

3.3 unittest 断言

前言

在测试用例中，执行完测试用例后，最后一步是判断测试结果是 pass 还是 fail，自动化测试脚本里面一般把这种生成测试结果的方法称为断言（assert）。

用 unittest 组件测试用例的时候,断言的方法还是很多的,下面介绍几种常用的断言方法：assertEqual、assertIn、assertTrue

3.3.1 简单案例

1.下面写了 4 个 case，其中第四个是执行失败的

```
# coding:utf-8
import unittest
class Test(unittest.TestCase):
	def test01(self):
		'''判断 a == b '''
		a = 1
		b = 1
		self.assertEqual(a, b)

	def test02(self):
		'''判断 a in b '''
		a = "hello"
		b = "hello world!"
		self.assertIn(a, b)

	def test03(self):
		'''判断 a is True '''
		a = True
		self.assertTrue(a)

	def test04(self):
		'''失败案例'''
		a = "上海-悠悠"
		b = "yoyo"
```

```
            self.assertEqual(a, b)

if __name__ == "__main__":
        unittest.main()
```

2.执行结果如下

```
Failure
Expected :'\xe4\xb8\x8a\xe6\xb5\xb7-\xe6\x82\xa0\xe6\x82\xa0'
Actual     :'yoyo'
  <Click to see difference>

Traceback (most recent call last):
   File "D:\test\yoyotest\kecheng\test12.py", line 27, in test04
       self.assertEqual(a, b)
AssertionError: '\xe4\xb8\x8a\xe6\xb5\xb7-\xe6\x82\xa0\xe6\x82\xa0' != 'yoyo'
```

3.执行的结果，中文编码不对，没正常显示中文，遇到这种情况，可以自定义异常输出

3.3.2 自定义异常

1.以 assertEqual 为例分析：

```
assertEqual(self, first, second, msg=None)
        Fail if the two objects are unequal as determined by the '=='
        operator.
```

2.翻译：如果两个对象不能相等，就返回失败，相当于 return: first==second

3.这里除了相比较的两个参数 first 和 second,还有第三个参数 msg=None,这个 msg 参数就是遇到异常后自定义输出信息

```
    def test04(self):
        '''失败案例'''
        a = "上海-悠悠"
        b = "yoyo"    # QQ群交流:232607095
        self.assertEqual(a, b, msg="失败原因：%s ！= %s"%(a,b))

if __name__ == "__main__":
    unittest.main()
```

```
4 tests done: 1 failed – 2ms
D:\test\python2\python.exe "D:\test\pycharm\PyCharm 5.0.4\helpers\pycharm\utrunner.p
Testing started at 22:09 ...

Failure
Traceback (most recent call last):
  File "D:\test\yoyotest\kecheng\test12.py", line 25, in test04
    self.assertEqual(a, b, msg="失败原因：%s ！= %s"%(a,b))
AssertionError: 失败原因：上海-悠悠 ！= yoyo
```

3.3.3 unittest 常用的断言方法

1.assertEqual(self, first, second, msg=None)

--判断两个参数相等：first == second

2.assertNotEqual(self, first, second, msg=None)

--判断两个参数不相等：first ！= second

3.assertIn(self, member, container, msg=None)

--判断是字符串是否包含：member in container

4.assertNotIn(self, member, container, msg=None)

--判断是字符串是否不包含：member not in container

5.assertTrue(self, expr, msg=None)

--判断是否为真：expr is True

6.assertFalse(self, expr, msg=None)

--判断是否为假：expr is False

7.assertIsNone(self, obj, msg=None)

--判断是否为 None：obj is None

8.assertIsNotNone(self, obj, msg=None)
--判断是否不为 None：obj is not None

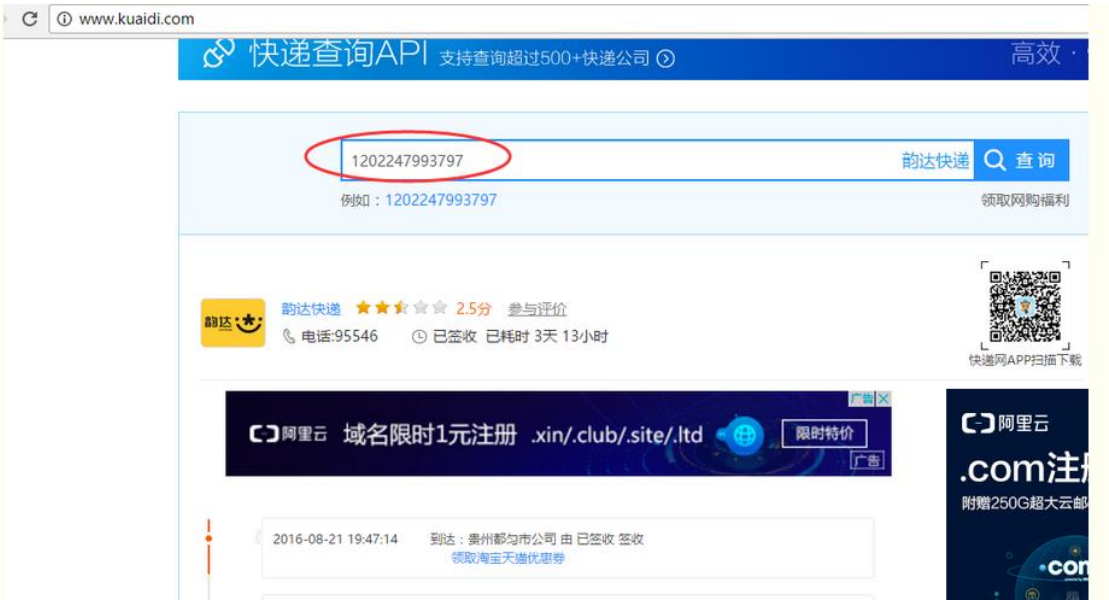
3.4 案例一：查询快递接口

前言：

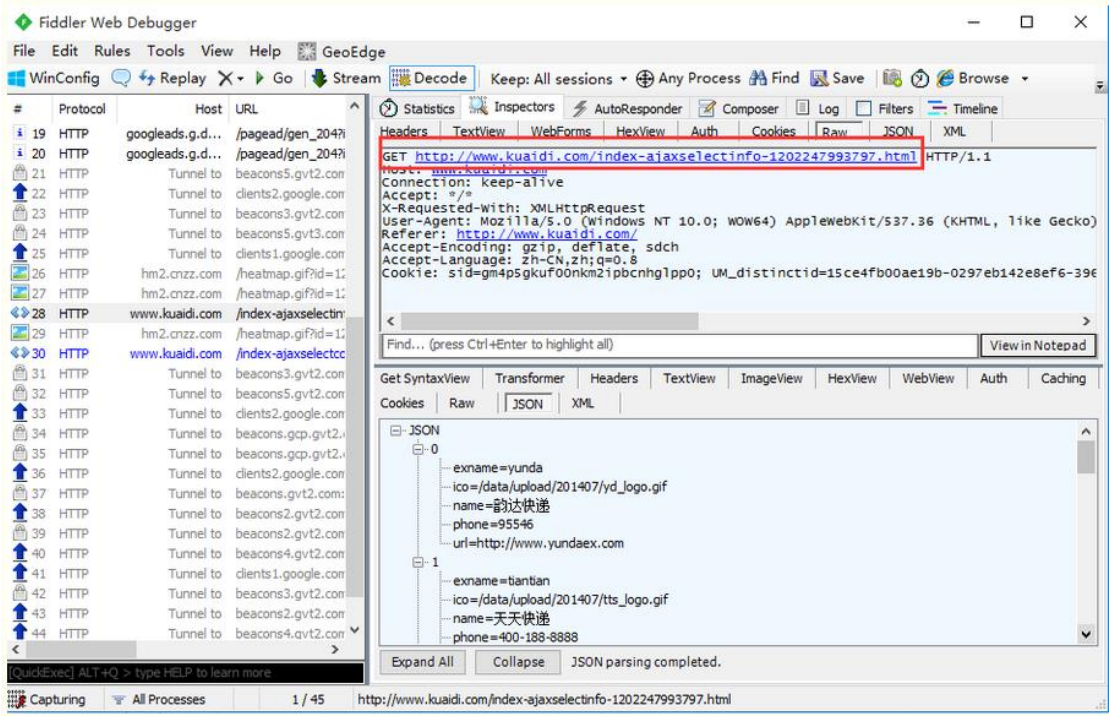
这里以快递查询的接口为例，设计了 2 个用例，一个是查询 yunda 快递单号，一个是查询 tiantian 快递的单号。

3.4.1 查询 yunda 单号

1.查询 yunda 快递单号：1202247993797 ，测试地址：http://www.kuaidi.com/



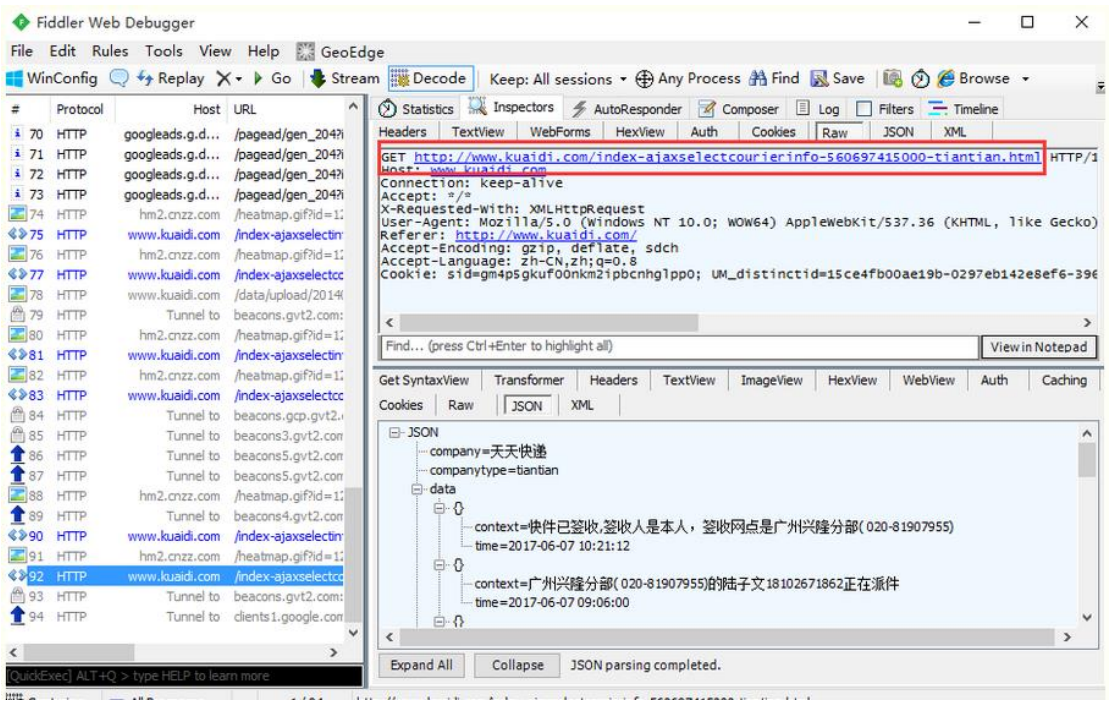
2.用 fiddler 抓包，获取单号查询请求的接口地址



3. 抓包后得到请求接口：
http://www.kuaidi.com/index-ajaxselectinfo-1202247993797.html

3.4.2 查询 tiantian 单号

1. 查询 tiantian 快递单号：560697415000



2.获取接口 url 地址：
http://www.kuaidi.com/index-ajaxselectcourierinfo-560697415000-tiantian.html

3.4.3 设计用例

1.针对上面两个接口请求，设计两个测试用例，参考代码：

```
# coding:utf-8

import unittest

import requests

class Test_Kuaidi(unittest.TestCase):

    def setUp(self):

        self.headers = {

                    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0"

                }  # get 方法其它加个 ser-Agent 就可以了

        self.s = requests.session()
```

```
    def test_yunda(self):
        danhao = '1202247993797'
        kd = 'yunda'
        # 这里对 url 的单号参数了
        self.url =
"http://www.kuaidi.com/index-ajaxselectcourierinfo-%s-%s.html" %(danhao, kd)
        print(self.url)
        # 第一步发请求
        r = self.s.get(self.url, headers=self.headers, verify=False)
        result = r.json()
        # 第二步获取结果
        print(result['company'])  # 获取公司名称
        data = result["data"]   # 获取 data 里面内容
        print(data[0])            # 获取 data 里最上面有个
        get_result = data[0]['context']  # 获取已签收状态
        print(get_result)
        # 断言：测试结果与期望结果对比
        self.assertEqual(u"韵达快递", result['company'])
        self.assertIn(u"已签收", get_result)
    def test_tiantian(self):
        danhao = '560697415000'
        kd = 'tiantian'
        # 这里对 url 的单号参数了
```

```
            self.url =
"http://www.kuaidi.com/index-ajaxselectcourierinfo-%s-%s.html" %(danhao,kd)

            print(self.url)

            # 第一步发请求

            r = self.s.get(self.url, headers=self.headers, verify=False)

            result = r.json()

            # 第二步获取结果

            print(result['company'])   # 获取公司名称

            data = result["data"]    # 获取 data 里面内容

            print(data[0])             # 获取 data 里最上面有个

            get_result = data[0]['context']   # 获取已签收状态

            print(get_result)

            # 断言：测试结果与期望结果对比

            self.assertEqual(u"天天快递", result['company'])

            self.assertIn(u"已签收", get_result)

if __name__ == "__main__":

    unittest.main()
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

3.5 案例二：登录结果

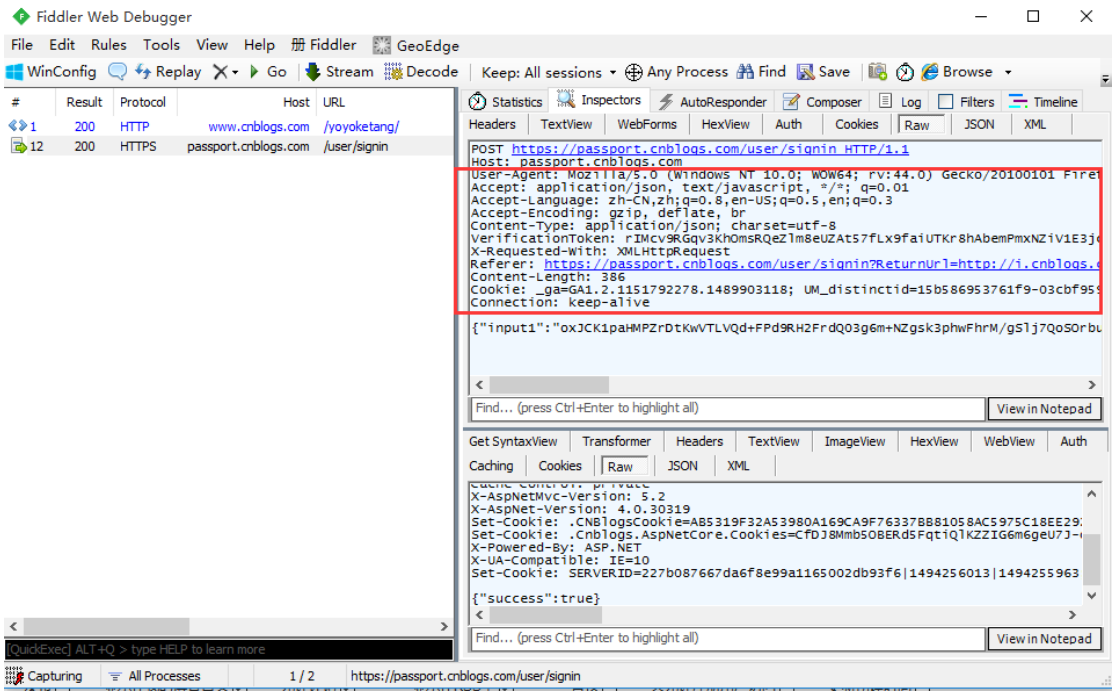
前言：

本篇以博客园登录为案例，结合 unittest 框架写 2 个用例

3.5.1 抓登录接口

1. 以博客园为例，模拟登陆，实际的情况要比上面讲的几个基本内容要复杂很多，一般登陆涉及安全性方面，登陆会比较复杂

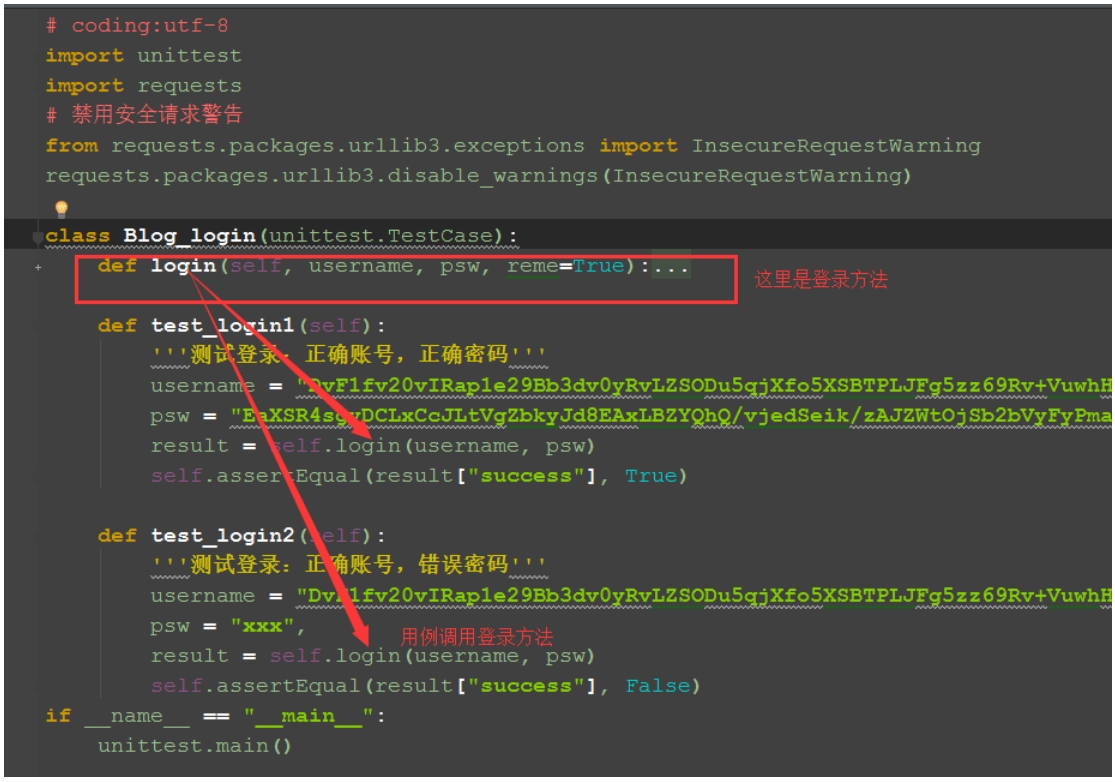
2. 这里需添加请求头 headers, 可以用 fiddler 抓包，账号和密码是加密后的字串



（具体操作参考：2.2 发 post 请求）

3.5.2 参考代码

（博客园登录相关的，不能用账号和密码登录了，仅供学习思路）




```
# coding:utf-8
import unittest
import requests
# 禁用安全请求警告
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
class Blog_login(unittest.TestCase):
    def login(self, username, psw, reme=True):
        '''三个参数：
        账号：username，密码：psw,记住登录：reme=True'''
        url = "https://passport.cnblogs.com/user/signin"
        header = {
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0",
            "Cookie": "xxx 已省略",
            "X-Requested-With": "XMLHttpRequest",
            "Connection": "keep-alive",
            "Content-Length": "385"
                }
        json_data = {"input1": username,
                "input2": psw,
                "remember": reme}
        res = requests.post(url, headers=header, json=json_data, verify=False)
        result1 = res.content  # 字节输出
```

```
        print result1
        return res.json()       # 返回 json
    def test_login1(self):
        '''测试登录：正确账号，正确密码'''
        username = "正确账号，抓包获得的加密字符串",
        psw = "正确密码，抓包获得的加密字符串",
        result = self.login(username, psw)
        self.assertEqual(result["success"], True)
    def test_login2(self):
        '''测试登录：正确账号，错误密码'''
        username = "正确账号",
        psw = "xxx 错误密码",
        result = self.login(username, psw)
        self.assertEqual(result["success"], False)
if __name__ == "__main__":
    unittest.main()
```

3.6 批量执行用例 discover

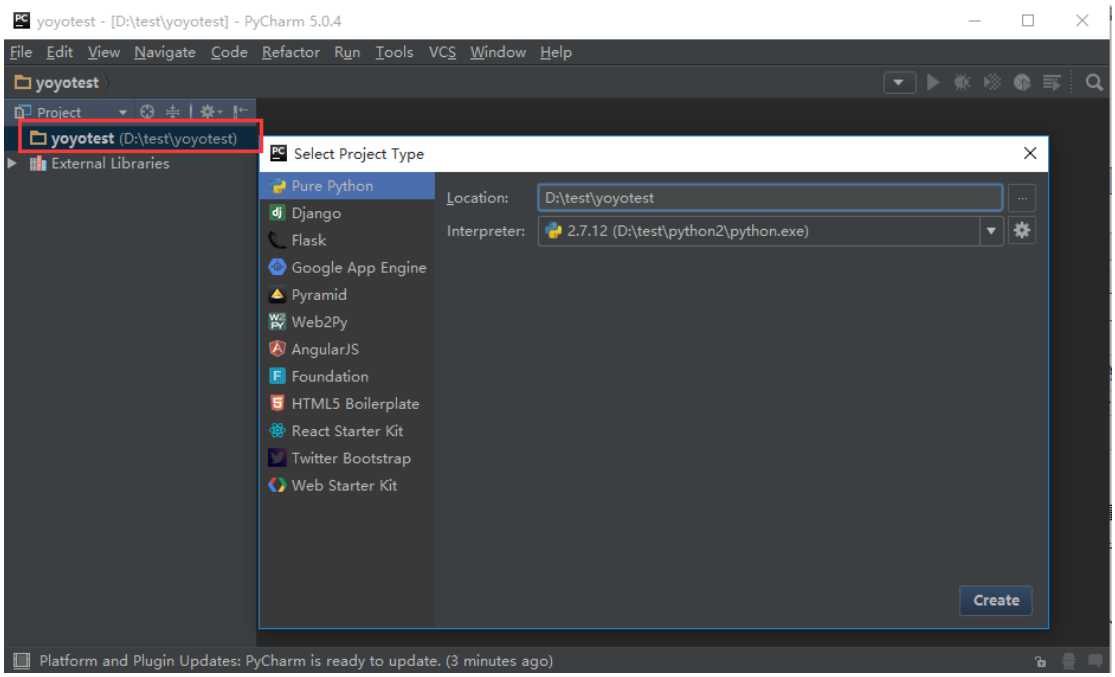
前言

我们在写用例的时候，单个脚本的用例好执行，那么多个脚本的时候，如何批量执行呢？这时候就需要用到 unittest 里面的 discover 方法来加载用例了。

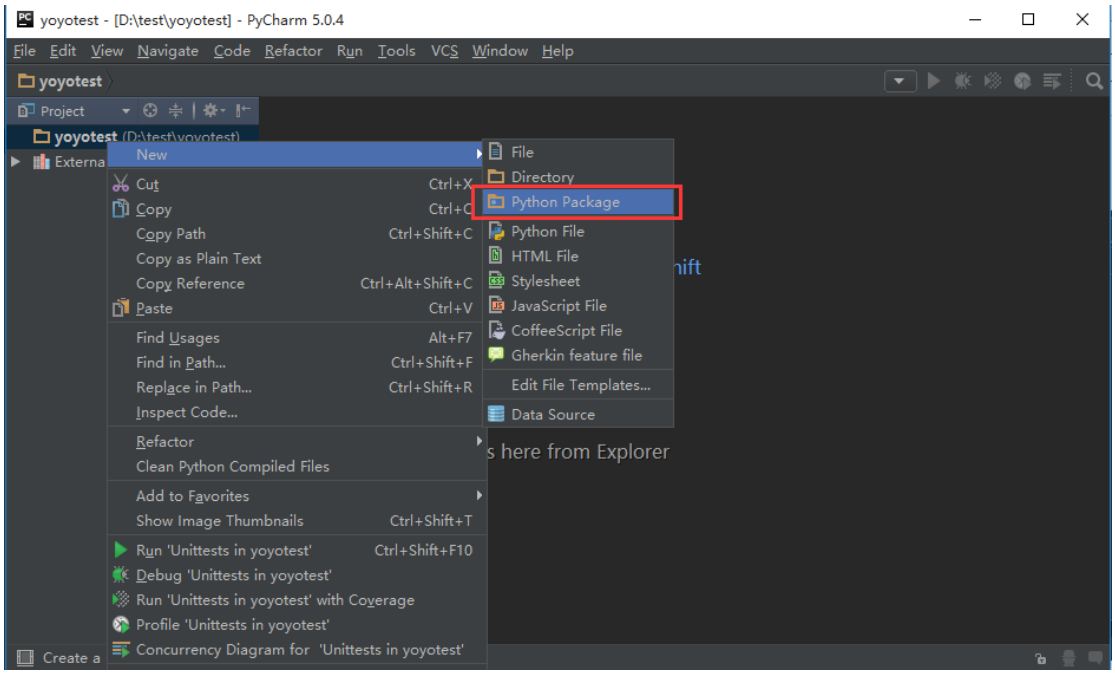
加载用例后,用 unittest 里面的 TextTestRunner 这里类的 run 方法去一次执行多个脚本的用例。

3.6.1 新建测试项目

1.pycharm 左上角 File>New Projetc>Pure Python,在 location 位置命名一个测试工程的名称：yoyotest,然后保存

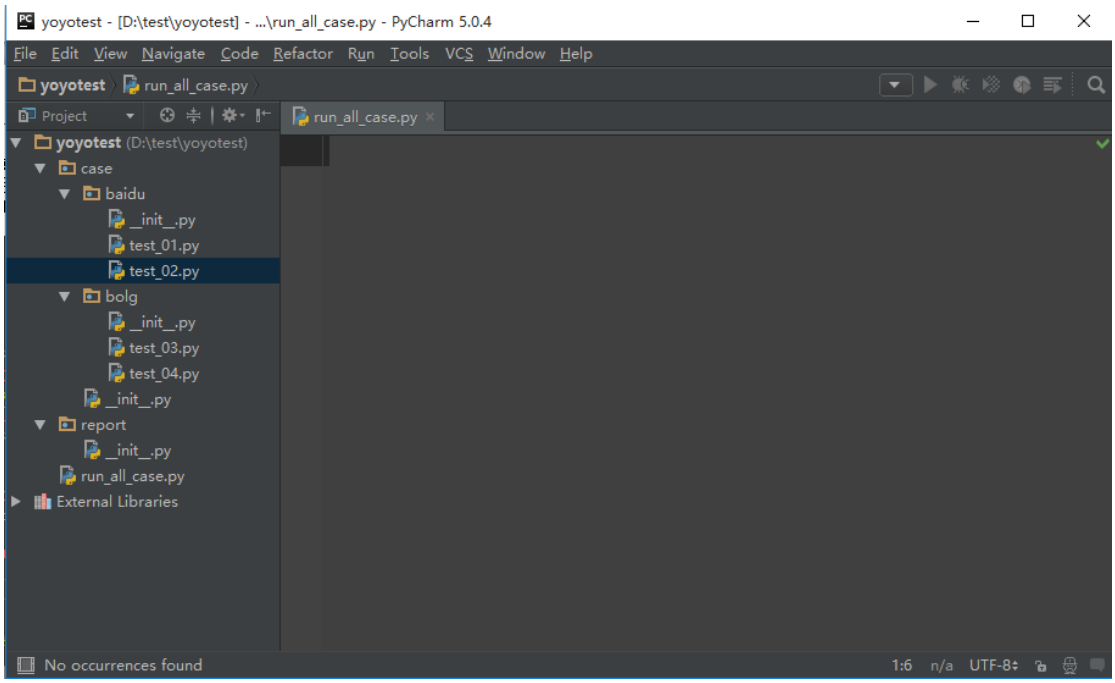


2. 选中刚才新建的工程右键>New>Python Package>新建一个 case 文件夹



3. 重复第 2 步的操作，新建一个 case 的文件夹，在里面添加一个 baidu 和一个 blog 的文件夹，里面分别有两个用例的脚本，如下图所示。

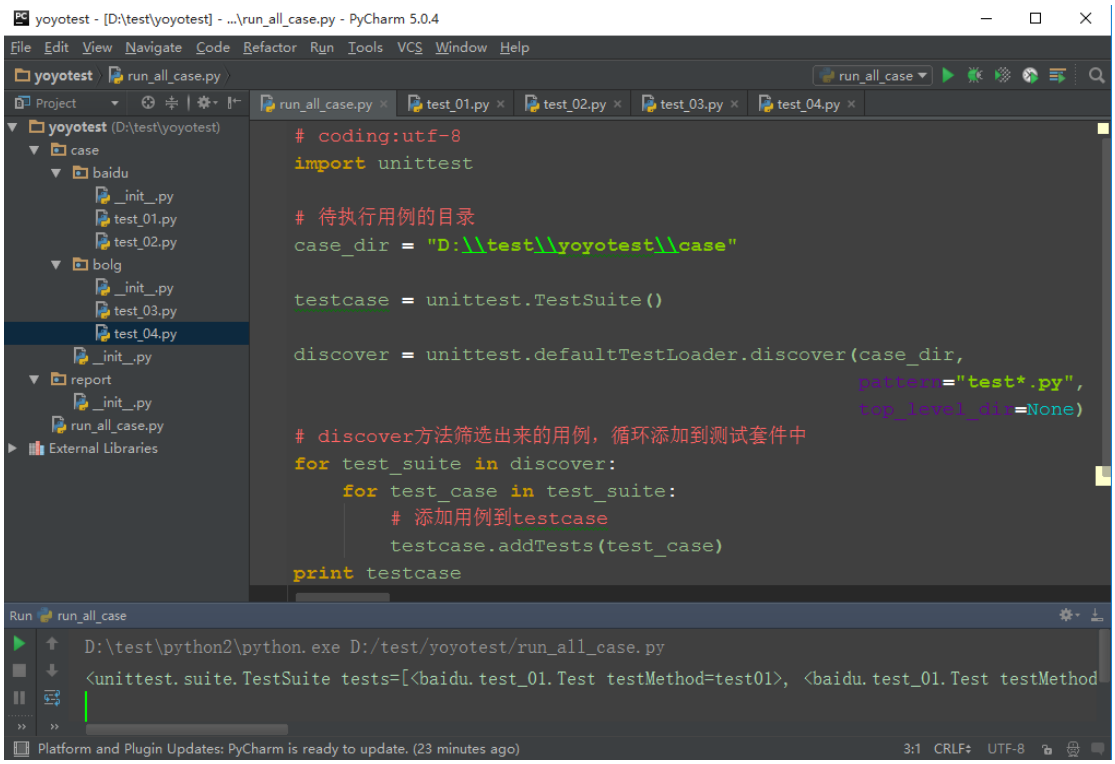
test_01, test_02, test_03, test_04 是我们写用例的脚本



4. 在 yoyotest 这个项目下面创建一个脚本 run_all_case.py，接下来用这个脚本去批量执行所有的用例。

3.6.2 discover 加载测试用例

1. discover 方法里面有三个参数：
-case_dir:这个是待执行用例的目录。
-pattern：这个是匹配脚本名称的规则，test*.py 意思是匹配 test 开头的所有脚本。
-top_level_dir：这个是顶层目录的名称，一般默认等于 None 就行了。
2. discover 加载到的用例是一个 list 集合，需要重新写入到一个 list 对象 testcase 里，这样就可以用 unittest 里面的 TextTestRunner 这里类的 run 方法去执行。

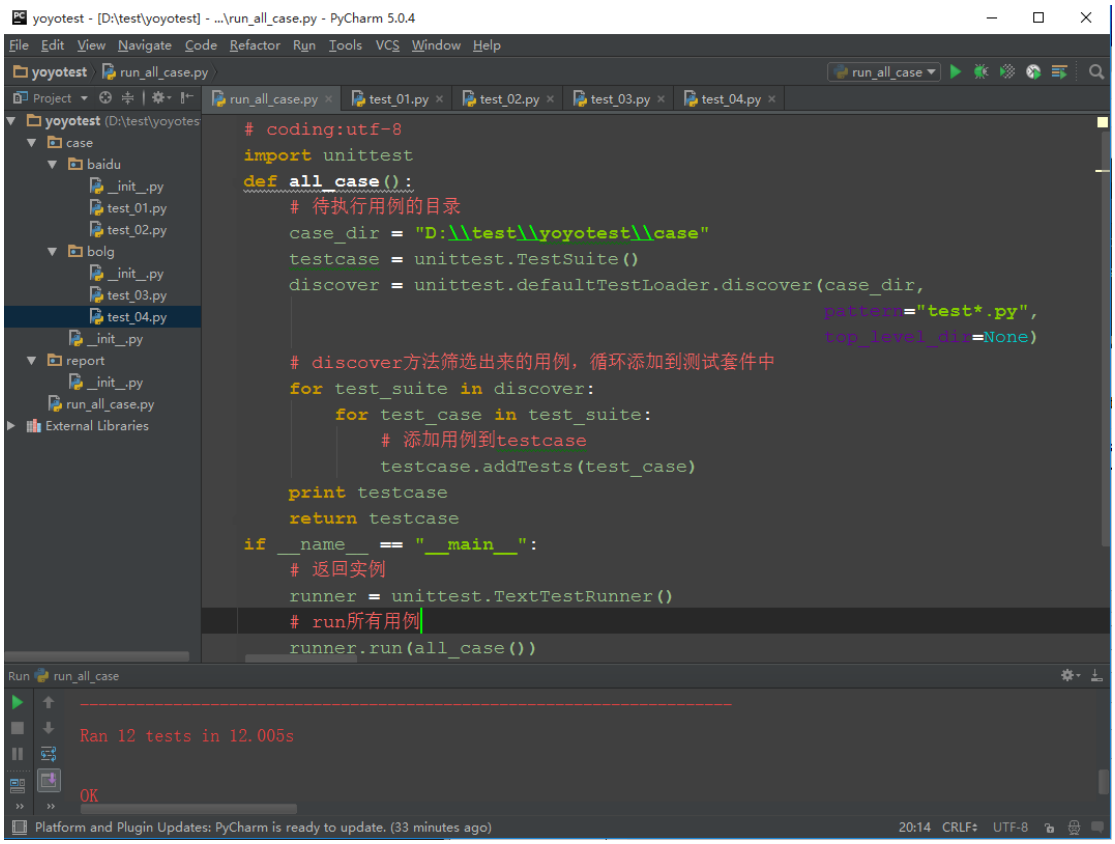


3.运行后结果入下，就是加载到的所有测试用例了：

<unittest.suite.TestSuite tests=[<baidu.test_01.Test testMethod=test01>, <baidu.test_01.Test testMethod=test02>, <baidu.test_01.Test testMethod=test03>, <baidu.test_02.Test testMethod=test01>, <baidu.test_02.Test testMethod=test02>, <baidu.test_02.Test testMethod=test03>, <bolg.test_03.Test testMethod=test01>, <bolg.test_03.Test testMethod=test02>, <bolg.test_03.Test testMethod=test03>, <bolg.test_04.Test testMethod=test01>, <bolg.test_04.Test testMethod=test02>, <bolg.test_04.Test testMethod=test03>]>

3.6.3 run 测试用例

1.为了更方便的理解，可以把上面 discover 加载用例的方法封装下，写成一个函数
2.先返回 TextTestRunner()类的实例
3.调用 run 方法去执行 all_case()这个函数



3.6.4 参考代码

```
    '''加载所有的测试用例'''
    testunit = unittest.TestSuite()
    # 定义discover方法的参数
    discover = unittest.defaultTestLoader.discover(case_path,
                                                   pattern=rule,
                                                   top_level_dir=None)
    # discover方法筛选出来的用例，循环添加到测试套件中
    # for test_suite in discover:
    #     for test_case in test_suite:
    #         testunit.addTests(test_case)
    #         print testunit
    testunit.addTests(discover)   # 直接加载discover
    print(testunit)
    return testunit
```

这个写法可以兼容python2和3

```
# coding:utf-8
import unittest
def all_case():
    # 待执行用例的目录
    case_dir = "D:\\test\\yoyotest\\case"
    testcase = unittest.TestSuite()
```

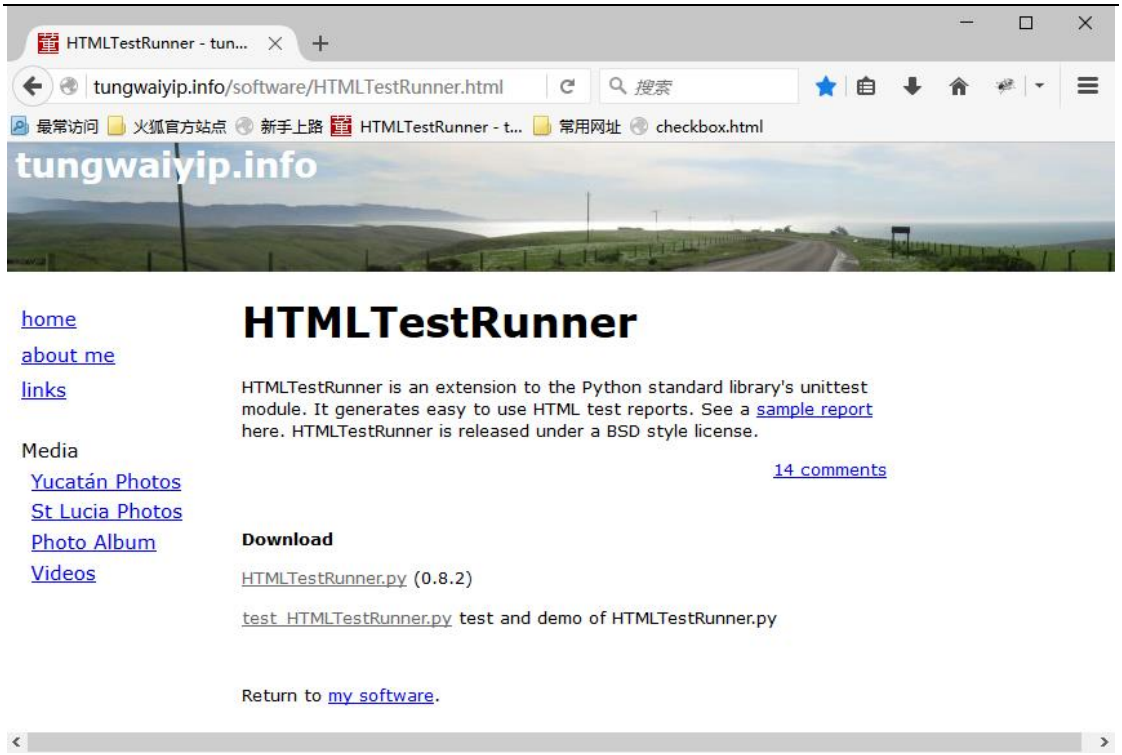
```
    discover = unittest.defaultTestLoader.discover(case_dir,
                                                   pattern="test*.py",
                                                   top_level_dir=None)
    testcase.addTests(discover)  # 直接加载 discover
    print(testcase)
    return testcase
if __name__ == "__main__":
    # 返回实例
    runner = unittest.TextTestRunner()
    # run 所有用例
    runner.run(all_case())
```

3.7 生成 html 报告

前言
批量执行完用例后，生成的测试报告是文本形式的，不够直观，为了更好的展示测试报告，最好是生成 HTML 格式的。
unittest 里面是不能生成 html 格式报告的，需要导入一个第三方的模块：HTMLTestRunner

3.7.1 导入 HTMLTestRunner

1.这个模块下载不能通过 pip 安装了，只能下载后手动导入，下载地址：http://tungwaiyip.info/software/HTMLTestRunner.html



2. Download 下 HTMLTestRunner.py 文件就是我们需要下载的包。

3. 下载后手动拖到 python 安装文件的 Lib 目录下

3.7.2 demo 解析

1. 下载 Download 下的第二个文件 test_HTMLTestRunner.py,这个就是官方给的一个测试 demo 了，从这个文件可以找到该模块的用法。

2. 找到下图这段，就是官方给的一个 demo 了，test_main()里上半部分就是加载测试 case,我们不需要搞这么复杂。

3. 最核心的代码是下面的红色区域，这个就是本篇的重点啦。

test_HTMLTestRunner.py

```
# This is the main test on HTMLTestRunner

class Test_HTMLTestRunner(unittest.TestCase):

    def test0(self):
        self.suite = unittest.TestSuite()
        buf = StringIO.StringIO()
        runner = HTMLTestRunner.HTMLTestRunner(buf)
        runner.run(self.suite)
        # didn't blow up? ok.
        self.assert_('</html>' in buf.getvalue())

    def test_main(self):
        # Run HTMLTestRunner. Verify the HTML report.

        # suite of TestCases
        self.suite = unittest.TestSuite()
        self.suite.addTests([
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTest0),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTest1),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestBasic),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestHTML),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestLatin1),
            unittest.defaultTestLoader.loadTestsFromTestCase(SampleTestUnicode),
            ])

        # Invoke TestRunner
        buf = StringIO.StringIO()
        #runner = unittest.TextTestRunner(buf)       #DEBUG: this is the unittest baseline
        runner = HTMLTestRunner.HTMLTestRunner(
                    stream=buf,
                    title='<Demo Test>',
                    description='This demonstrates the report output by HTMLTestRunner.'
                    )
        runner.run(self.suite)

        # Define the expected output sequence. This is imperfect but should
        # give a good sense of the well being of the test.
        EXPECTED = u"""
```

3.7.3 生成 html 报告

1.我们只需把上面红色区域代码 copy 到上一篇的基础上稍做修改就可以了，这里主要有三个参数:

--stream:测试报告写入文件的存储区域

--title:测试报告的主题

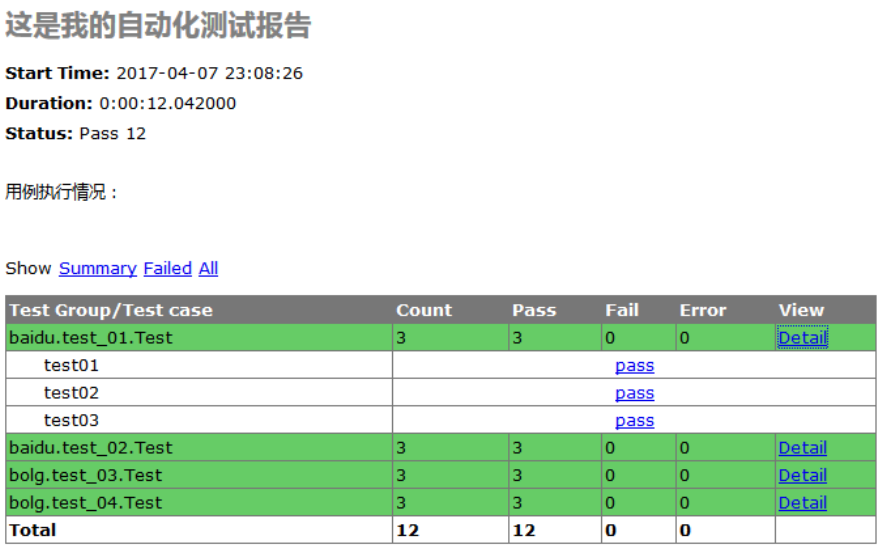
--description：测试报告的描述

2.report_path 是存放测试报告的地址

```
if __name__ == "__main__":
    # 返回实例
    # runner = unittest.TextTestRunner()
    import HTMLTestRunner
    report_path = "D:\\test\\yoyotest\\report\\result.html"

    fp = open(report_path, "wb")
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp,
                                           title=u'这是我的自动化测试报告',
                                           description=u'用例执行情况：')

    # run所有用例   交流QQ群：232607095
    runner.run(all_case())
    fp.close()
```

3.7.4 测试报告详情

1.找到测试报告文件，用浏览器打开，点开 View 里的 Detail 可以查看详情描述。



这是我的自动化测试报告

Start Time: 2017-04-07 23:08:26
Duration: 0:00:12.042000
Status: Pass 12

用例执行情况：

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
baidu.test_01.Test	3	3	0	0	Detail
test01			pass		
test02			pass		
test03			pass		
baidu.test_02.Test	3	3	0	0	Detail
bolg.test_03.Test	3	3	0	0	Detail
bolg.test_04.Test	3	3	0	0	Detail
Total	**12**	**12**	**0**	**0**	

2.为了生成带中文描述的测试用例，可以在 case 中添加注释，如在 test_01 的脚本添加如下注释：

```
class Test(unittest.TestCase):
    def setUp(self):
        print "start!"
    def tearDown(self):
        time.sleep(1)
        print "end!"
    def test01(self):
        u'''测试登录用例，账号：xx 密码 xx'''
```

```
        print "执行测试用例 01"
    def test03(self):
        u'''测试登搜索用例，关键词：xxx'''
        print "执行测试用例 03"
```

3. 重新运行后查看测试报告

这是我的自动化测试报告

Start Time: 2017-04-07 23:24:12
Duration: 0:00:12.013000
Status: Pass 12

用例执行情况：

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
baidu.test_01.Test	3	3	0	0	Detail
test01: 测试登录用例，账号：xx 密码xx			pass		
test02			pass		
test03: 测试登搜索用例，关键词：xxx			pass		
baidu.test_02.Test	3	3	0	0	Detail
bolg.test_03.Test	3	3	0	0	Detail
bolg.test_04.Test	3	3	0	0	Detail
Total	**12**	**12**	**0**	**0**	

3.7.5 参考代码

```
# coding:utf-8
import unittest
import HTMLTestRunner
def all_case():
    # 待执行用例的目录
    case_dir = "D:\\test\\yoyotest\\case"
    testcase = unittest.TestSuite()
    discover = unittest.defaultTestLoader.discover(case_dir,
                                                   pattern="test*.py",
                                                   top_level_dir=None)
    # discover 方法筛选出来的用例，循环添加到测试套件中
    for test_suite in discover:
        for test_case in test_suite:
            # 添加用例到 testcase
            testcase.addTests(test_case)
    print testcase
    return testcase
```

```
if __name__ == "__main__":
    # 返回实例
    # runner = unittest.TextTestRunner()
    report_path = "D:\\test\\yoyotest\\report\\result.html"
    fp = open(report_path, "wb")
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp,
                                           title=u'这是我的自动化测试报告',
                                           description=u'用例执行情况：')
    runner.run(all_case())
    fp.close()
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

3.8 测试报告优化

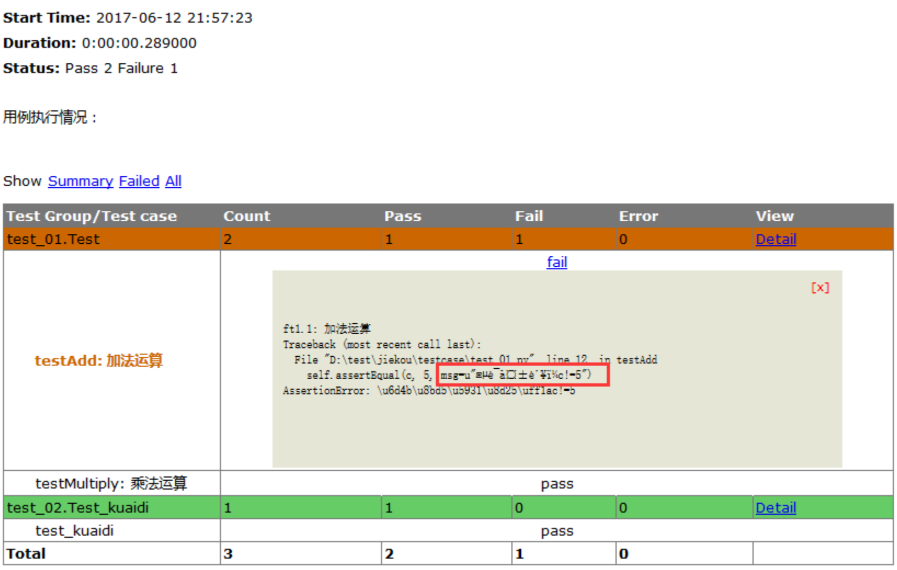
前言

python2 用 HTMLTestRunner 生成测试报告时，有中文输出情况会出现乱码,这个主要是编码格式不统一，改下编码格式就行。

下载地址：http://tungwaiyip.info/software/HTMLTestRunner.html

3.8.1 中文乱码

1.测试报告中，msg 自定义异常内容有中文情况会出现乱码，如下图所示



3.8.2 修改编码

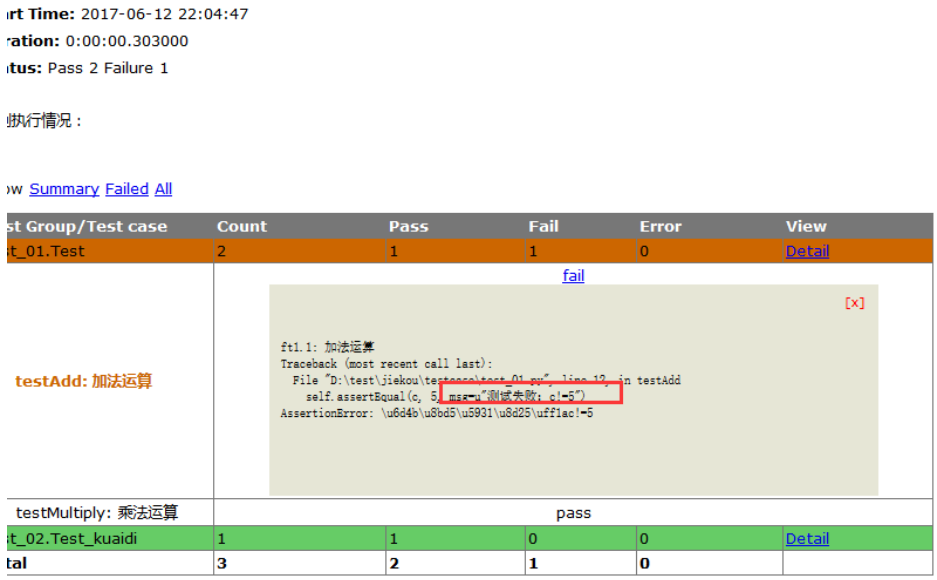
1. 找到 HTMLTestRunner.py 文件，搜索:uo =
2. 找到红色区域设置编码的两个地方

```
def _generate_report_test(self, rows, cid, tid, n, t, o, e):
    # e.g. 'pt1.1', 'ft1.1', etc
    has_output = bool(o or e)
    tid = (n == 0 and 'p' or 'f') + 't%s.%s' % (cid+1,tid+1)
    name = t.id().split('.')[-1]
    doc = t.shortDescription() or ""
    desc = doc and ('%s: %s' % (name, doc)) or name
    tmpl = has_output and self.REPORT_TEST_WITH_OUT

    # o and e should be byte string because they ar
    if isinstance(o,str):
        # TODO: some problem with 'string_escape':
        # uo = unicode(o.encode('string_escape'))
        uo = o.decode('latin-1')
    else:
        uo = o
    if isinstance(e,str):
        # TODO: some problem with 'string_escape':
        # ue = unicode(e.encode('string_escape'))
        ue = e.decode('latin-1')
    else:
        ue = e

    script = self.REPORT_TEST_OUTPUT_TMPL % dict(
        id = tid,
        output = saxutils.escape(uo+ue),
    )
```

3. 注释掉红色区域这两个设置，重新添加编码格式为：uo = o.decode('utf-8')　ue = e.decode('utf-8')

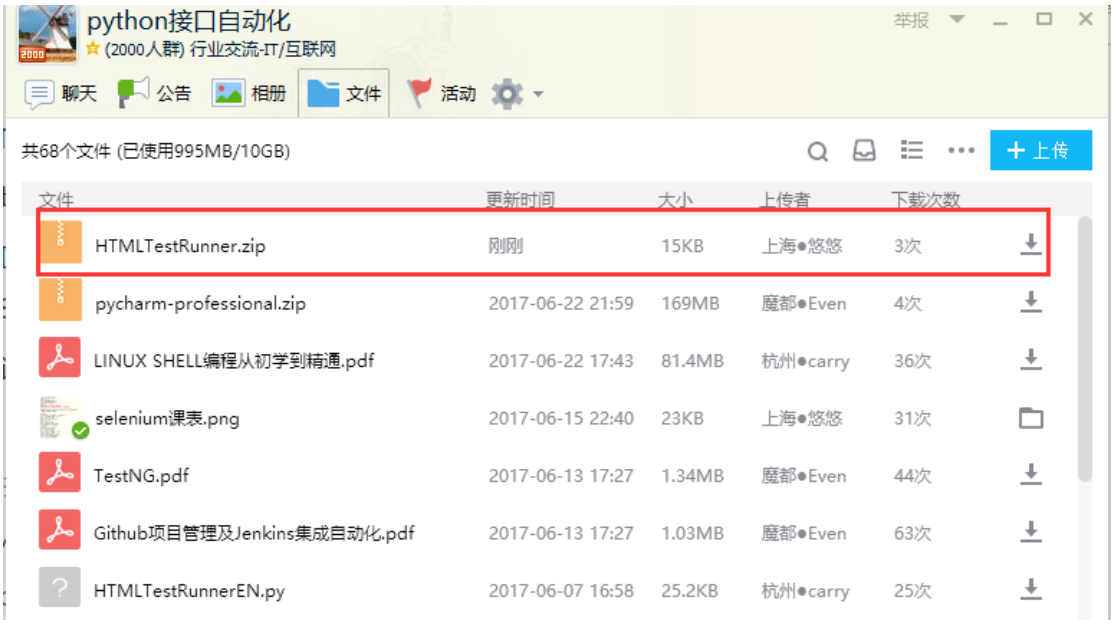
```
    def _generate_report_test(self, rows, cid, tid, n, t, o, e):
        # e.g. 'pt1.1', 'ft1.1', etc
        has_output = bool(o or e)
        tid = (n == 0 and 'p' or 'f') + 't%s.%s' % (cid+1,tid+1)
        name = t.id().split('.')[-1]
        doc = t.shortDescription() or ""
        desc = doc and ('%s: %s' % (name, doc)) or name
        tmpl = has_output and self.REPORT_TEST_WITH_OUTPUT_TMPL or self.REPORT_TEST_NO_OUTPUT_TMPL

        # o and e should be byte string because they are collected from stdout and stderr?
        if isinstance(o,str):
            # TODO: some problem with 'string_escape': it escape \n and mess up formating
            # uo = unicode(o.encode('string_escape'))
            # uo = o.decode('latin-1')
            uo = o.decode('utf-8')
        else:
            uo = o
        if isinstance(e,str):
            # TODO: some problem with 'string_escape': it escape \n and mess up formating
            # ue = unicode(e.encode('string_escape'))
            # ue = e.decode('latin-1')
            ue = e.decode('utf-8')
        else:
            ue = e
```

4. 修改好之后记得保存，重新运行，乱码问题就解决了



3.8.3 python3 报告问题

1. python3 的小伙伴直接用这个下载地址：
http://tungwaiyip.info/software/HTMLTestRunner.html
的文件，是不能直接生成报告的，需要稍做修改
2. 修改后的源文件已经打包放到 QQ 群文件：226296743



个人博客地址：http://www.cnblogs.com/yoyoketang/

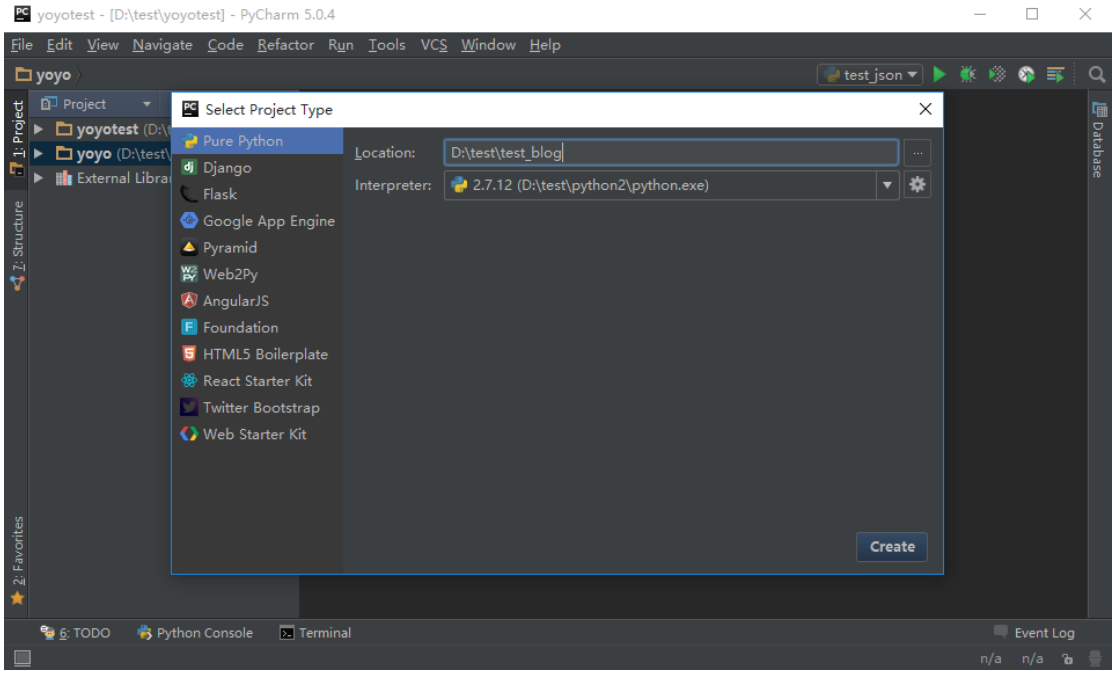
QQ 交流群：226296743

3.9 搭建项目环境

前言
到 unittest 这里基本上可以搭建一个简易的项目框架了，我们可以用一条 run_main.py 脚本去控制执行所有的用例，并生成报告，发送邮件一系列的动作

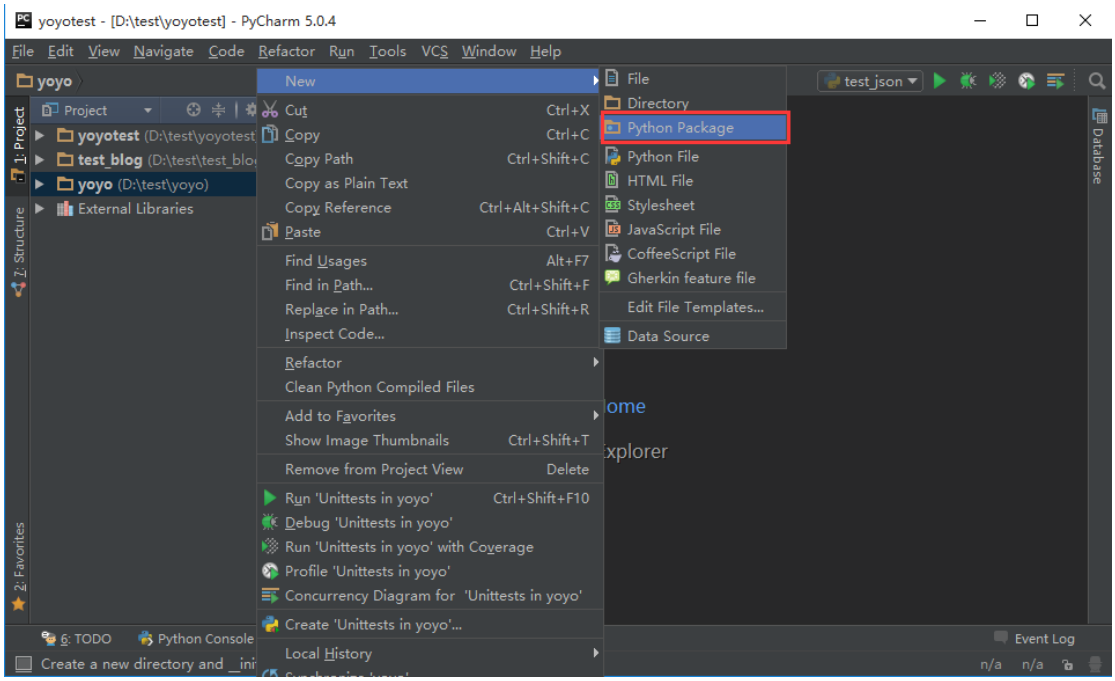
3.9.1 新建工程

1. 打开 pycharm 左上角 File>New Project，在 Location 位置输入项目名称：D:\test\test_blog
2. 创建之后，选择 Open in current window 就可以了



3.9.2 项目结构

1. 在测试工程下，创建文件夹，一定要选 Python Package 的方式创建，要不然后面导入自己写的模块会出现各种问题



2. 在工程下创建以下几个文件

--test_case　　这个文件夹放所有测试用例

----blog_home　　可以按功能用例模块划分

---------test_home

---------test_home_1　　测试用例以 test 开头命名

----blog_login

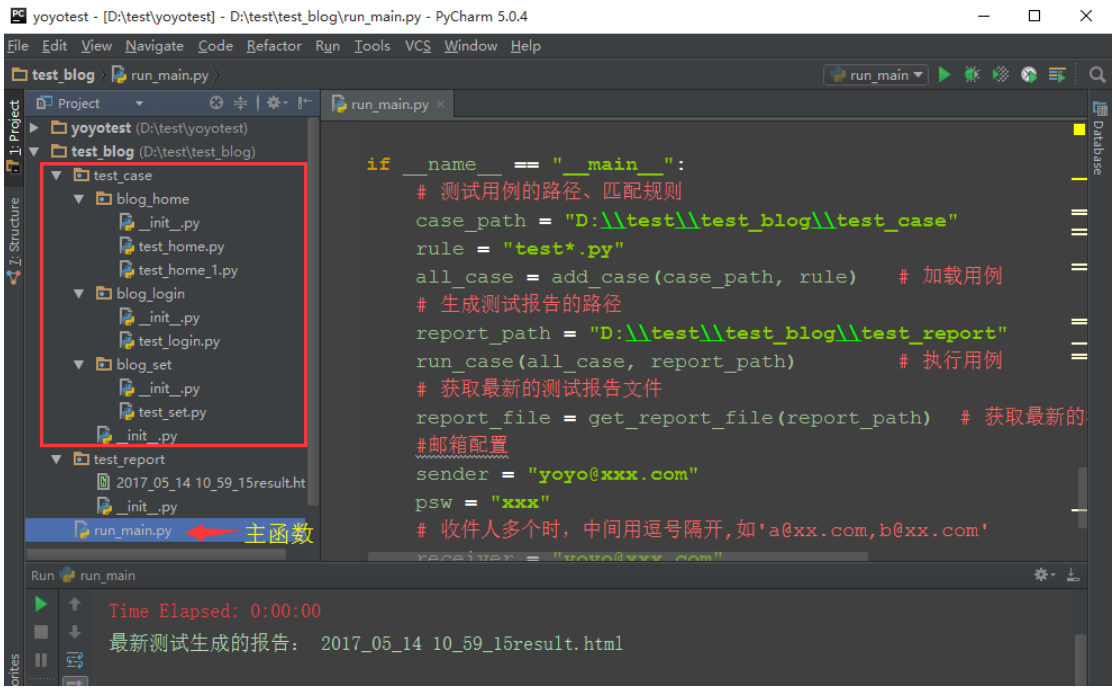
---------test_login

----blog_set

---------test_set

--test_report

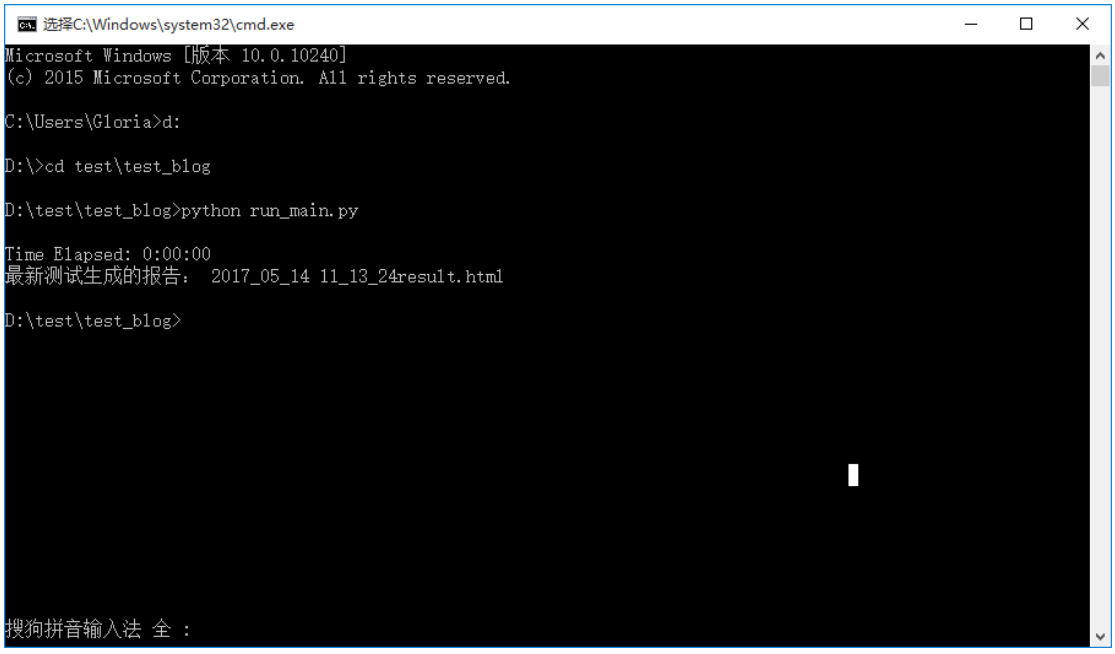
--run_main.py



3.9.3 run_main

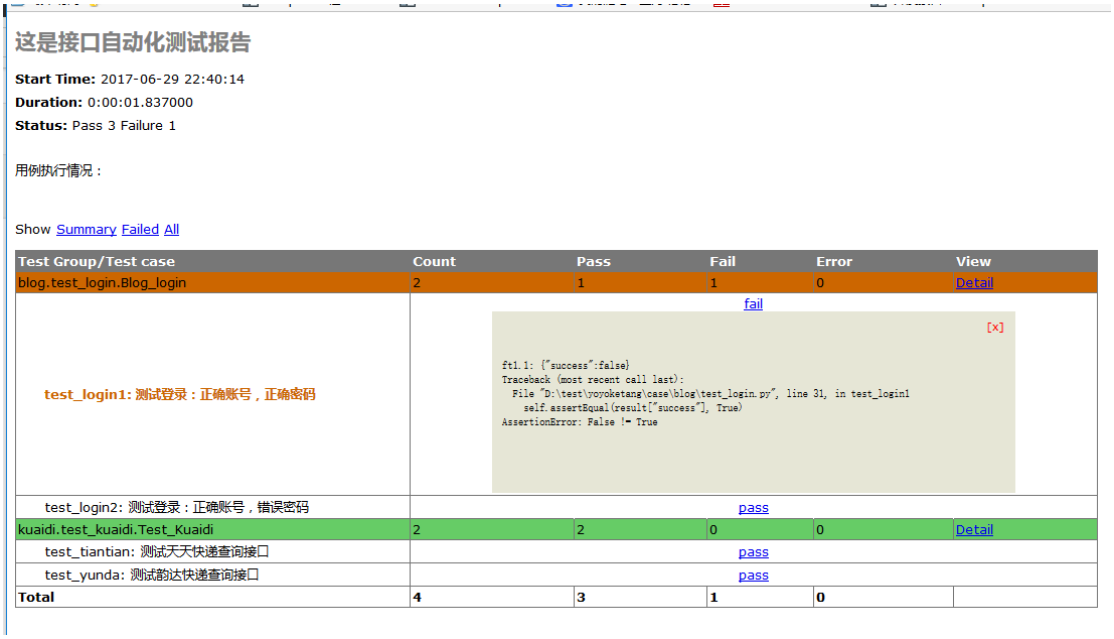
1. run_main.py 这个脚本里面写主函数，控制执行所有的用例，最终我们只需要运行这个脚本就可以了
2. 我们也可以在 cmd 里执行这个脚本文件，这样就不用依赖 pycharm 去执行了（后续用 jenkins 执行，也是同样道理，启动 cmd 执行脚本）

```
>>d:
>>cd test\test_blog
>>python run_main.py
```



3.10 项目源码下载（附案例）

百度云盘地址：
链接：http://pan.baidu.com/s/1jI3L1o6 密码：jadd



这是接口自动化测试报告

Start Time: 2017-06-29 22:40:14
Duration: 0:00:01.837000
Status: Pass 3 Failure 1

用例执行情况：

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
blog.test_login.Blog_login	2	1	1	0	Detail
test_login1: 测试登录：正确账号，正确密码	fail [x] ft1.1: {"success":false} Traceback (most recent call last): File "D:\test\yoyoketang\case\blog\test_login.py", line 31, in test_login1 self.assertEqual(result["success"], True) AssertionError: False != True				
test_login2: 测试登录：正确账号，错误密码	pass				
kuaidi.test_kuaidi.Test_Kuaidi	2	2	0	0	Detail
test_tiantian: 测试天天快递查询接口	pass				
test_yunda: 测试韵达快递查询接口	pass				
Total	**4**	**3**	**1**	**0**	

3.11 pycharm 运行两种姿势

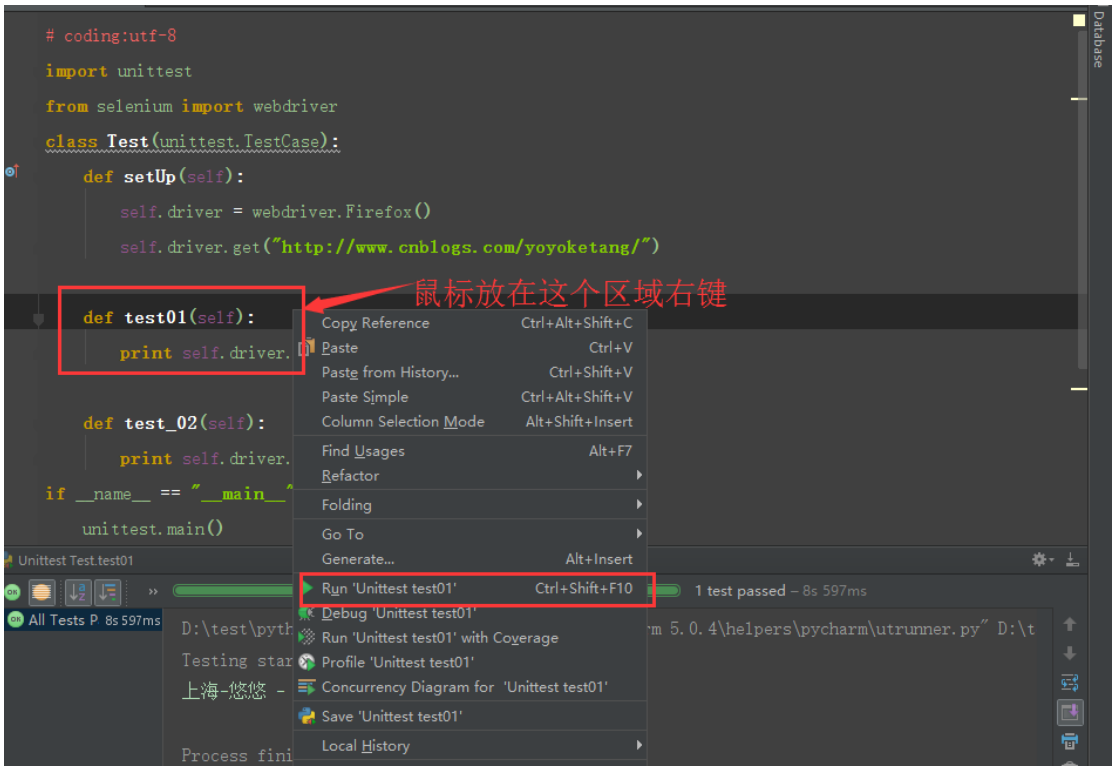
前言
经常有人在群里反馈，明明代码一样的啊，为什么别人的能出报告，我的出不了报告；为什么别人运行结果跟我的不一样啊。。。
这种问题先检查代码，确定是一样的，那就是运行姿势不对了，一旦导入 unittest 模块，pycharm 会自动识别以 unittest 的姿势去运行了。

3.11.1 unittest 运行单个用例

1. 如下代码，如果我只想运行其中的一个，如 test1, 如何运行呢？

```python
# coding:utf-8
import unittest
from selenium import webdriver
class Test(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.get("http://www.cnblogs.com/yoyoketang/")

    def test01(self):
        print self.driver.title

    def test_02(self):
        print self.driver.title
if __name__ == "__main__":
    unittest.main()
```

2. 如果想运行哪个用例，鼠标放到对应的区域，右键就能直接运行单个用例了

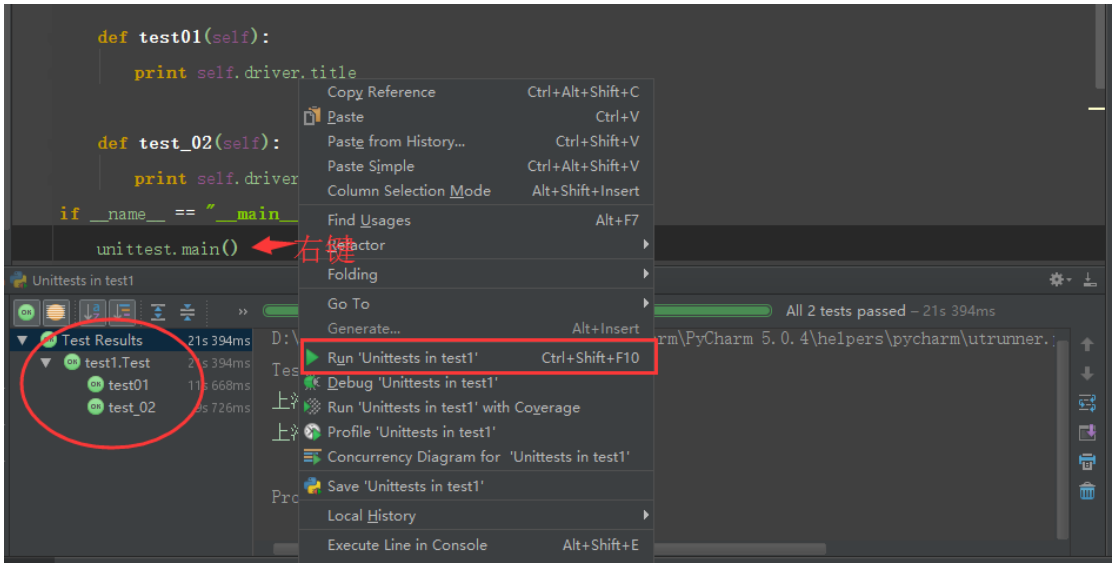


3. 注意上图红色框框，显示的是 Run 'Unittest test01'，这个意思很明显，就是说以 unittest 的方式只运行 test01

如果想运行 test2，鼠标放到 test2 区域右键，就会显示：Run 'Unittest test02'

3.11.2 unittest 运行整个脚本用例

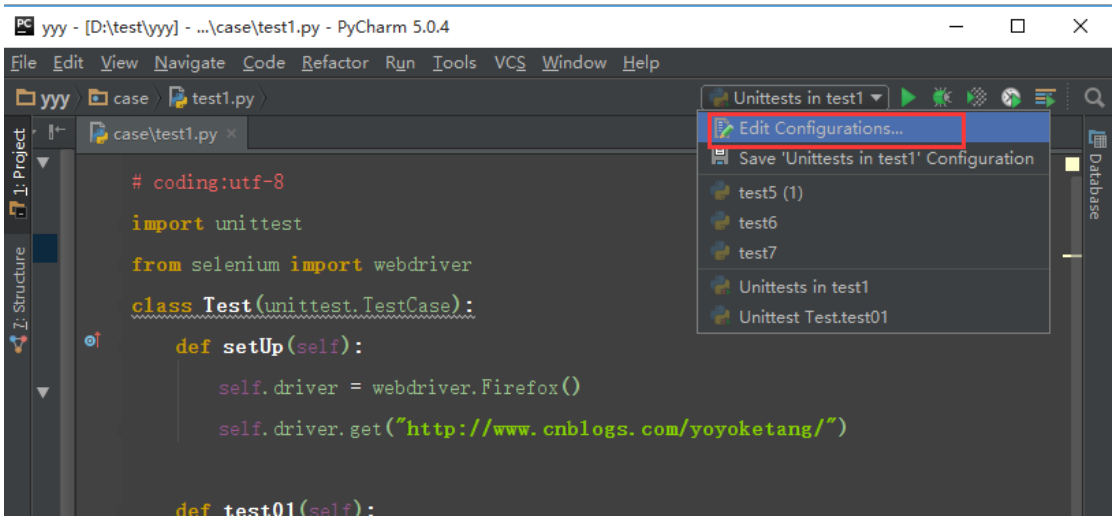
1. 如果想当前的脚本上所有的用例一起执行，只需把鼠标放到 if __name__ == "__main__": 这句话的后面或者下方就行了



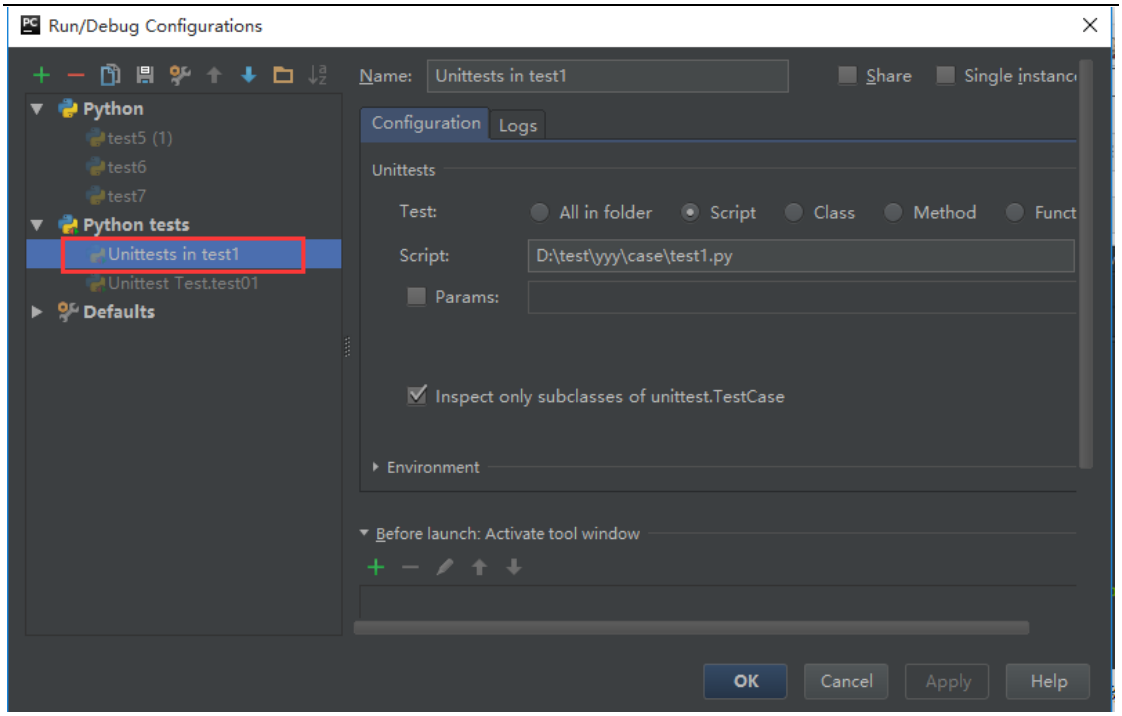
2. 上图显示的是 Run Unittest in test1 运行的是整个脚本

3.11.3 python 直接运行模式

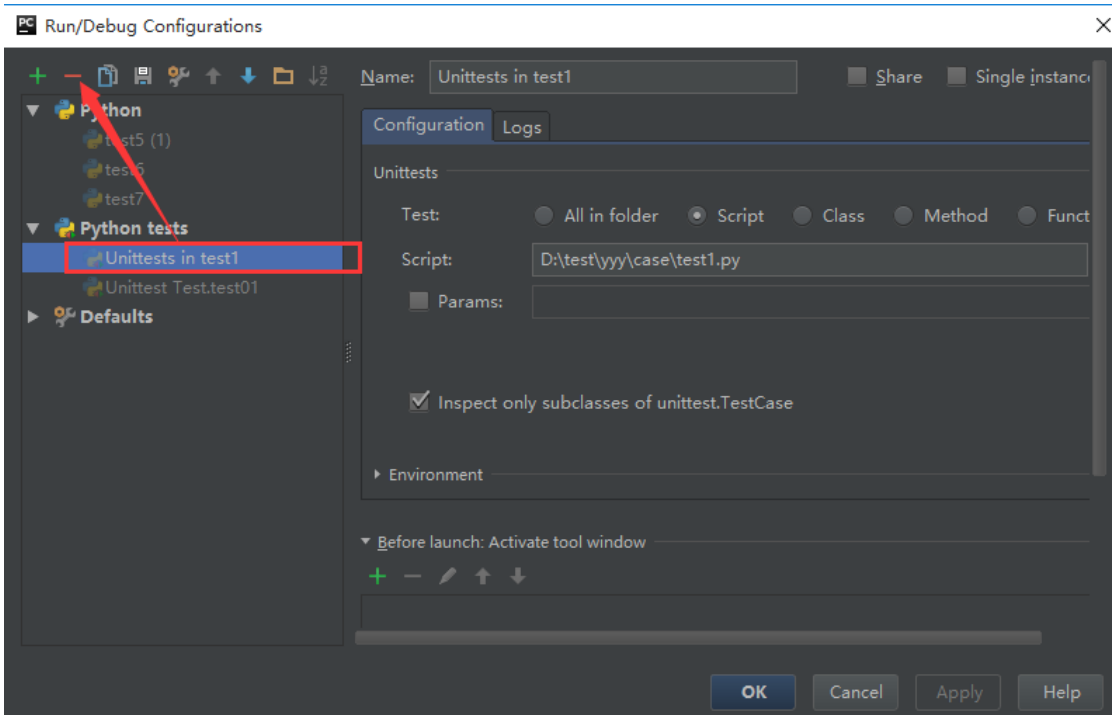
1. 以上方式只能执行单元测试框架里面的内容，是没有测试报告的输出的，如果想输出测试报告，就得以非 unittest 模式运行，也就是用 python 直接运行
2. 点右上角，点开：Edit Configurations



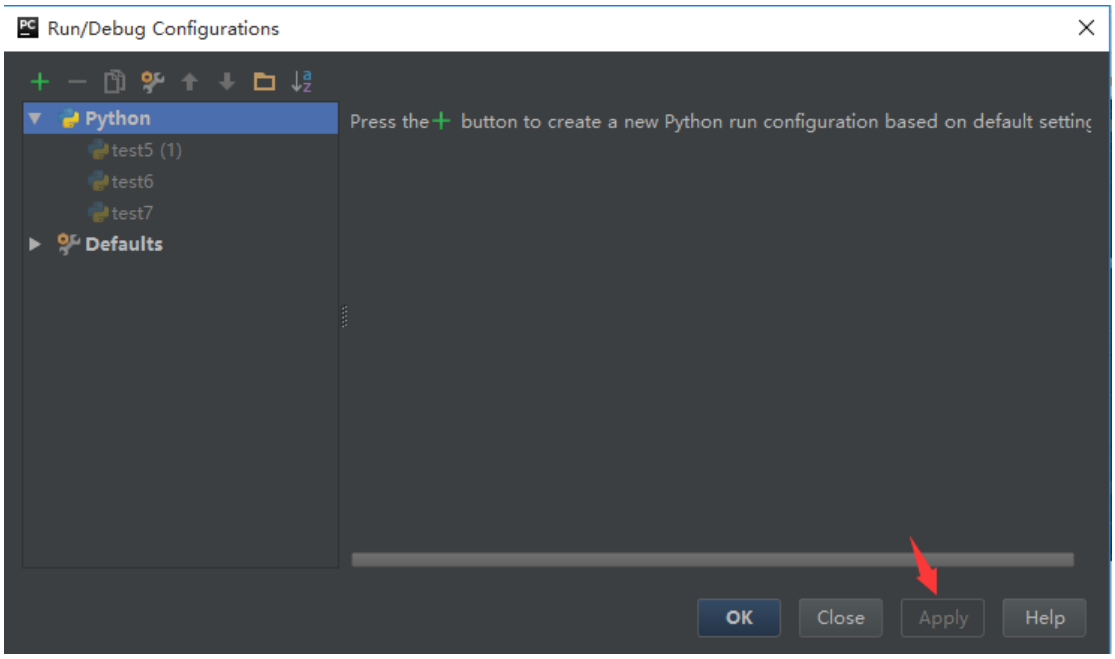
3. 展开 Python 和 Python tests 这两个选项，会发现刚才运行的脚本名称在 Python tests 这个选项下，也就是 pycharm 这个工具自动默认为 unittest 姿势运行了。



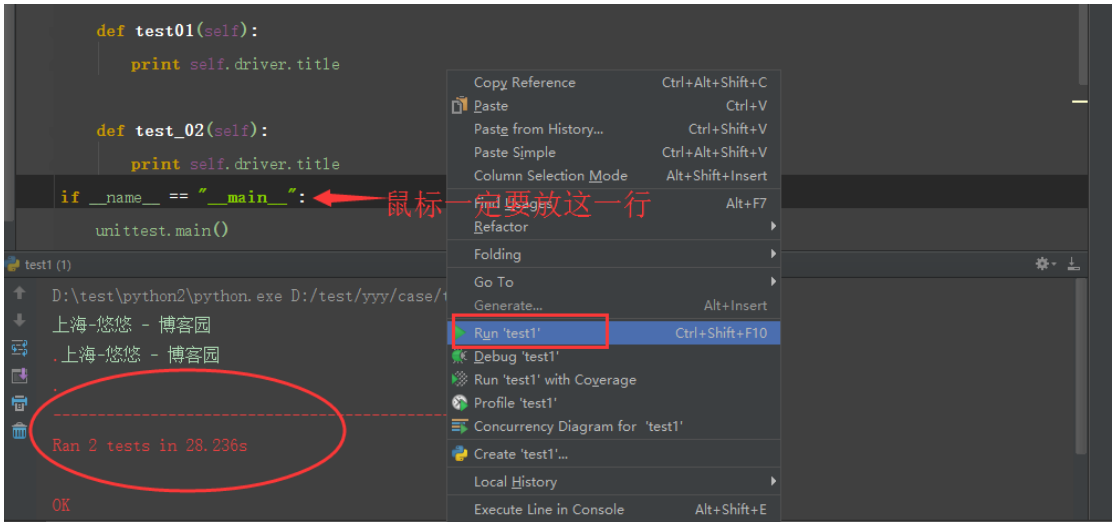
4.解决办法，只需从这里移除掉就行，选中这个脚本，点左上角的减号：一



5 全部移除后，点 Apply 应用，然后点 OK 保存



6. 再回到脚本界面，右键运行，就能出测试报告了
（敲黑板了!!! 鼠标一定要放在 if 这一行后边右键，才能看到正常显示 Run “test1”）



个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

第 4 章 封装与调用

4.1 函数与参数化

前言

前面实现了参数的关联，那种只是记流水账的完成功能，不便于维护，也没什么可读性，接下来这篇可以把每一个动作写成一个函数，这样更方便了。

参数化的思维只需记住一点：不要写死

4.1.1 登录函数

1. s 参数是 session 的一个实例类，先放这里，方便写后面代码

2. 登录函数传三个参数，s 是需要调用前面的 session 类，所以必传，可以传个登录的 url，然后 payload 是账号和密码

```
s = requests.session()      # s参数作为全局的
def login(s, url, payload):
    '''登录'''
    headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firef
               "Accept": "application/json, text/javascript, */*; q=0.01",
               "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
               "Accept-Encoding": "gzip, deflate, br",
               "Content-Type": "application/json; charset=utf-8",
               "X-Requested-With": "XMLHttpRequest",
               "Content-Length": "385",
               "Cookie": "_ga=GA1.2.1151792278.1489903118; UM_distinctid=15b586953761f9-03cbf959
               "Connection": "keep-alive"
               }
    r = s.post(url, json=payload, headers=headers, verify=False)
    result = r.json()
    print result
    return result['success']    # 返回True或False
```

4.1.2 保存草稿

1. 编辑内容的标题 title 和正文 body_data 参数化了，这样后面可以方便传不同值

2. 这里返回了获取到新的 url 地址，因为后面的 postid 参数需要在这里提取

```
def save_box(s, url2, title, body_data):
    '''# 获取报存之后url地址'''
    body = {"__VIEWSTATE": "",
        "__VIEWSTATEGENERATOR": "FE27D343",
        "Editor$Edit$txbTitle": title,
        "Editor$Edit$EditorBody": "<p>"+body_data+"</p>",
        "Editor$Edit$Advanced$ckbPublished": "on",
        "Editor$Edit$Advanced$chkDisplayHomePage": "on",
        "Editor$Edit$Advanced$chkComments": "on",
        "Editor$Edit$Advanced$chkMainSyndication": "on",
        "Editor$Edit$lkbDraft": "存为草稿",
        }
    r2 = s.post(url2, data=body, verify=False)
    print r2.url
    return r2.url
```

4.1.3 提取 postid

1. 这里用正则表达式提取 url 里面的 postid

```
def get_postid(u):
    '''正则提取postid'''
    import re
    postid = re.findall(r"postid=(.+?)&", u)
    print postid  # 这里是list
    if len(postid) < 1:
        return ''
    else:
        return postid[0]
```

4.1.4 删除草稿

1. 传个 url 和 postid 就可以了

```
def delete_box(s,url3, postid):
    '''删除草稿箱'''
    json3 = {"postId": postid}
    r3 = s.post(url3, json=json3, verify=False)
    print r3.json()
```

4.1.5 参考代码

```
# coding:utf-8
import requests
```

```
def login(s, url, payload):
'''登录'''
    headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0)
Gecko/20100101 Firefox/44.0",
               "Accept": "application/json, text/javascript, */*; q=0.01",
               "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
               "Accept-Encoding": "gzip, deflate, br",
               "Content-Type": "application/json; charset=utf-8",
               "X-Requested-With": "XMLHttpRequest",
               "Content-Length": "385",
               "Cookie": "xxx 已省略",
               "Connection": "keep-alive"
               }
    r = s.post(url, json=payload, headers=headers, verify=False)
    result = r.json()
  print result
    return result['success']    # 返回 True 或 False
def save_box(s, url2, title, body_data):
    '''# 获取报存之后 url 地址'''
    body = {"__VIEWSTATE": "",
        "__VIEWSTATEGENERATOR": "FE27D343",
        "Editor$Edit$txbTitle": title,
        "Editor$Edit$EditorBody": "<p>"+body_data+"</p>",
        "Editor$Edit$Advanced$ckbPublished": "on",
        "Editor$Edit$Advanced$chkDisplayHomePage": "on",
        "Editor$Edit$Advanced$chkComments": "on",
        "Editor$Edit$Advanced$chkMainSyndication": "on",
        "Editor$Edit$lkbDraft": "存为草稿",
        }
 r2 = s.post(url2, data=body, verify=False)
    print r2.url
    return r2.url
def get_postid(u):
    '''正则提取 postid'''
    import re
    postid = re.findall(r"postid=(.+?)&", u)
    print postid  # 这里是 list
    if len(postid) < 1:
        return ''
    else:
        return postid[0]
def delete_box(s,url3, postid):
    '''删除草稿箱'''
json3 = {"postId": postid}
```

```
    r3 = s.post(url3, json=json3, verify=False)
    print r3.json()
if __name__ == "__main__":
    url = "https://passport.cnblogs.com/user/signin"
    payload = {
                "input1": "xxx",
                "input2": "xxx",
                "remember": True
               }
    s = requests.session()
    login(s, url, payload)
    url2 = "https://i.cnblogs.com/EditPosts.aspx?opt=1"
    u = save_box(s, url2, "标题", "正文内容")
 postid = get_postid(u)
    url3 = "https://i.cnblogs.com/post/delete"
    delete_box(s, url3, postid)
```

4.2 流程类接口关联

前言：

流程相关的接口，主要用 session 关联，如果写成函数（如上篇），s 参数每个函数都要带，每个函数多个参数，这时候封装成类会更方便。

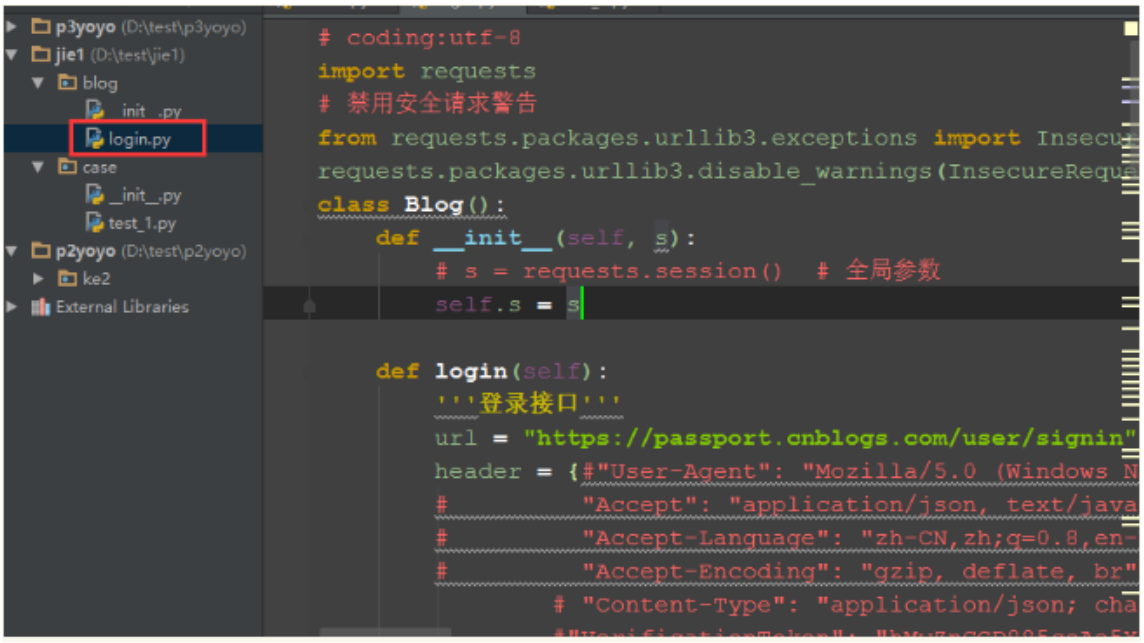
4.2.1 接口封装

1.有些接口经常会用到比如登录的接口，这时候我们可以每个接口都封装成一个方法，如：登录、保存草稿、发帖、删帖，这四个接口就可以写成四个方法

2.接口封装好了后，后面我们写用例那就直接调用封装好的接口就行了，有些参数，可以参数化，如保存草稿的 title 和 body 两个参数是动态的。

3.像这种流程类的接口，后面的会依赖前面的，就可以用 session 关联起来

4.保存以下脚本 login.py



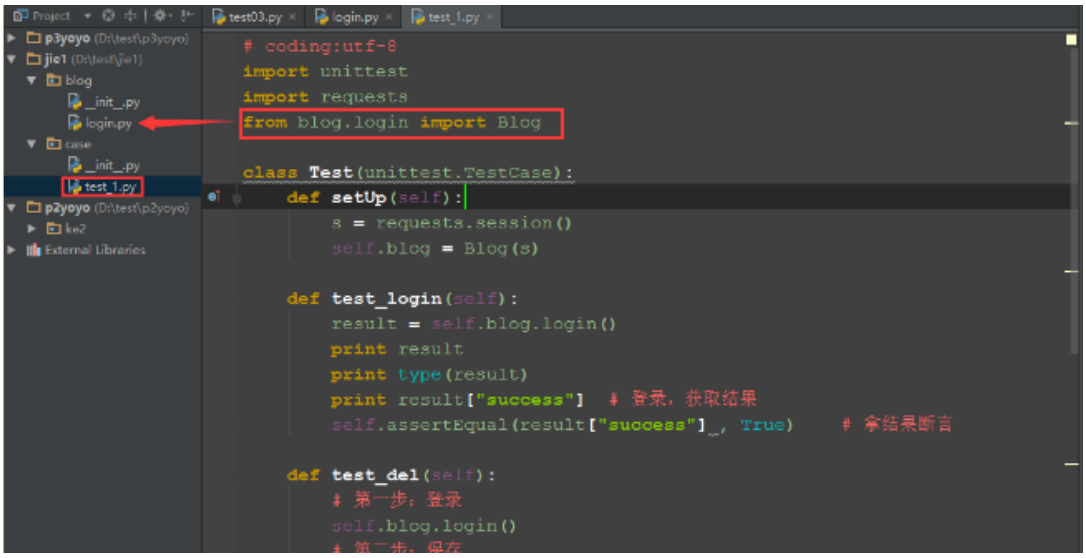

```
# coding:utf-8
import requests
# 禁用安全请求警告
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
class Blog():
    def __init__(self, s):
# s = requests.session()  # 全局参数
        self.s = s
    def login(self):
        '''登录接口'''
        url = "https://passport.cnblogs.com/user/signin"
        header = {#"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0",
        #         "Accept": "application/json, text/javascript, */*; q=0.01",
        #         "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
        #         "Accept-Encoding": "gzip, deflate, br",
                # "Content-Type": "application/json; charset=utf-8",
                #"VerificationToken": "xxx 已省略",
```

```
                "Cookie":"xxx 已省略",
                "X-Requested-With":"XMLHttpRequest",
# "Connection":"keep-alive",
                # "Content-Length":"385"
        }
        json_data = {"input1":"账号",
                "input2":"密码",
                "remember": False}
        res = self.s.post(url, headers=header, json=json_data, verify=False)
        result1 = res.content  # 字节输出
        print result1
        return res.json()
    def save(self, title, body):
        '''保存草稿箱：
        参数 1：title  # 标题
        参数 2：body   # 中文'''
        url2 = "https://i.cnblogs.com/EditPosts.aspx?opt=1"
        d = {"__VIEWSTATE": "",
            "__VIEWSTATEGENERATOR":"FE27D343",
            "Editor$Edit$txbTitle":title,
            "Editor$Edit$EditorBody":"<p>%s</p>"%body,
            "Editor$Edit$Advanced$ckbPublished":"on",
            "Editor$Edit$Advanced$chkDisplayHomePage":"on",
            "Editor$Edit$Advanced$chkComments":"on",
            "Editor$Edit$Advanced$chkMainSyndication":"on",
            "Editor$Edit$lkbDraft":"存为草稿",
             }
        r2 = self.s.post(url2, data=d, verify=False)  #保存草稿箱
        print r2.url
```

```
return r2.url
    def get_postid(self, r2_url):
        '''正则表达式提取'''
        import re
        postid = re.findall(r"postid=(.+?)&", r2_url)
        print postid  # 这里是 list []
        #  提取为字符串
        print postid[0]
        return postid[0]
    def del_tie(self, postid):
        '''删除帖子'''
        del_json = {"postId": postid}
        del_url = "https://i.cnblogs.com/post/delete"
        r3 = self.s.post(del_url, json=del_json, verify=False)
 print r3.json()["isSuccess"]
        return r3.json()
if __name__ == "__main__":
    s = requests.session()
    Blog(s).login()
```

4.2.2 用例导入接口

1.导入上面 login.py 模块写的接口



2.用例自己随便设计，参考如下：

```
# coding:utf-8
import unittest
import requests
from blog.login import Blog
class Test(unittest.TestCase):
    def setUp(self):
        s = requests.session()
        self.blog = Blog(s)
    def test_login(self):
        result = self.blog.login()
        print result
        print type(result)
        print result["success"]  # 登录，获取结果
        self.assertEqual(result["success"] , True)     # 拿结果断言
def test_del(self):
        # 第一步：登录
        self.blog.login()
        # 第二步：保存
```

```
        r2_url = self.blog.save(title="12121", body="WQASDAS")
        pid = self.blog.get_postid(r2_url)
        # 第三步：删除
        result = self.blog.del_tie(pid)
        self.assertEqual(result["isSuccess"], True)
if __name__ == "__main__":
  unittest.main()
```

3.运行结果：

第 5 章　爬虫利器 beautifulsoup4

官方文档：http://beautifulsoup.readthedocs.io/zh_CN/latest/

内容目录

Beautiful Soup 4.4.0 文档
- 寻求帮助

快速开始

安装 Beautiful Soup
- 安装完成后的问题
- 安装解析器

如何使用

对象的种类
- Tag
  - Name
  - Attributes
    - 多值属性
- 可以遍历的字符串
- BeautifulSoup
- 注释及特殊字符串

遍历文档树
- 子节点
  - tag的名字
  - .contents 和 .children
  - .descendants
  - .string
  - .strings 和 stripped_strings
- 父节点
  - .parent
  - .parents
- 兄弟节点
  - .next_sibling 和 .previous_sibling

Beautiful Soup 4.4.0 文档

Beautiful Soup 是一个可以从HTML或XML文件中提取数据的Python库.它能够通过你喜欢的转换器实现惯用的文档导航,查找,修改文档的方式.Beautiful Soup会帮你节省数小时甚至数天的工作时间.

这篇文档介绍了BeautifulSoup4中所有主要特性,并且有小例子,让我来向你展示它适合做什么,如何工作,怎样使用,如何达到你想要的效果,和处理异常情况.

文档中出现的例子在Python2.7和Python3.2中的执行结果相同

你可能在寻找 Beautiful Soup3 的文档,Beautiful Soup 3 目前已经停止开发,我们推荐在现在的项目中使用Beautiful Soup 4, 移植到BS4

这篇帮助文档已经被翻译成了其它语言:

- 这篇文档当然还有中文版.
- このページは日本語で利用できます(外部リンク)
- 이 문서는 한국어 번역도 가능합니다. (외부 링크)

寻求帮助

如果你有关于BeautifulSoup的问题,可以发送邮件到 讨论组 .如果你的问题包含了一段需要转换的HTML代码,那么确保你提的问题描述中附带这段HTML文档的 代码诊断 [1]

快速开始

5.1 html 解析器 html.parser

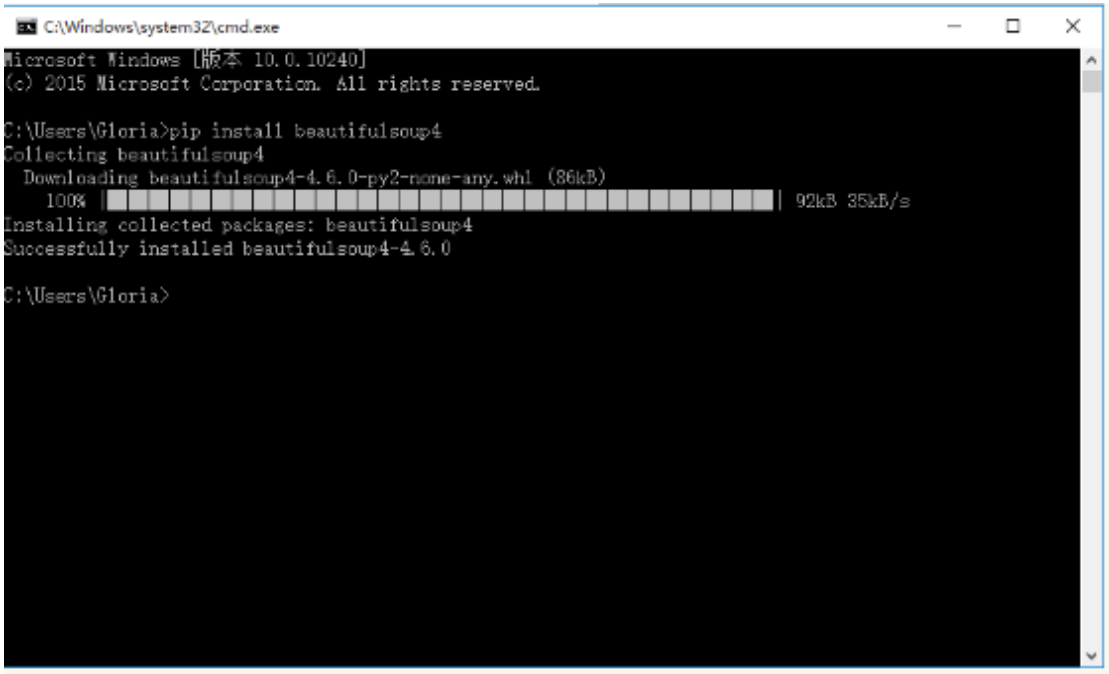
前言

以博客园为例，爬取我的博客上首页的发布时间、标题、摘要，本篇先小试牛刀，先了解下它的强大之处，后面讲 beautifulsoup4 的详细功能。

5.1.1 安装

1.打开 cmd 用 pip 在线安装 beautifulsoup4

```
>pip install beautifulsoup4
```



5.1.2 解析器

1.我们主要用第一个 html.parser，这个是 python 的标准库，可以直接用。其它几个需要安装对应解析器，下表列出了主要的解析器,以及它们的优缺点:

解析器	使用方法	优势	劣势
Python标准库	`BeautifulSoup(markup, "html.parser")`	• Python的内置标准库 • 执行速度适中 • 文档容错能力强	• Python 2.7.3 or 3.2.2)前 的版本中文档容错能力差
lxml HTML 解析器	`BeautifulSoup(markup, "lxml")`	• 速度快 • 文档容错能力强	• 需要安装C语言库
lxml XML 解析器	`BeautifulSoup(markup, ["lxml-xml"])` `BeautifulSoup(markup, "xml")`	• 速度快 • 唯一支持XML的解析器	• 需要安装C语言库
html5lib	`BeautifulSoup(markup, "html5lib")`	• 最好的容错性 • 以浏览器的方式解析文档 • 生成HTML5格式的文档	• 速度慢 • 不依赖外部扩展

5.1.3 打印首页博客的时间

1.这里直接定位不好定位到，可以先定位它的父元素：class="dayTitle"

2.用 requests 里的 get 方法打开博客首页，r.content 返回整个 html 内容，返回类型为 string

3. 查找所有的 class 属性为 dayTitle 的 Tag 类

4. 获取当前 Tag 的标签为 a 的 string 值

```
# coding:utf-8
from bs4 import BeautifulSoup
import requests

r = requests.get("http://www.cnblogs.com/yoyoketang/")
# 请求首页后获取整个html界面
blog = r.content
# print blog
# 用html.parser解析html
soup = BeautifulSoup(blog, "html.parser")
# 获取所有的class属性为dayTitle，返回Tag类
times = soup.find_all(class_="dayTitle")
for i in times:
    a = i
    print i.a.string  # 获取a标签的文本
```

```
D:\test\python2\python.exe D:/test/yoyotest/testbs/test_01.py
2017年5月21日
2017年5月16日
2017年5月14日
2017年5月13日
```

5.1.4 打印摘要

1. 获取标题方法跟上面一样，获取摘要的话，这里不太一样，这个父类 下多了一个子类 a

2. 先获取 div 这个 Tag 类，tag 的 .contents 属性可以将 tag 的子节点以列表的方式输出

3. 因为摘要可以看成是第一个子元素，取下标[0]就可以读出来

5.1.5 参考代码

```
# coding:utf-8
from bs4 import BeautifulSoup
import requests

r = requests.get("http://www.cnblogs.com/yoyoketang/")
# 请求首页后获取整个 html 界面
blog = r.content
# print blog

# 用 html.parser 解析 html
soup = BeautifulSoup(blog, "html.parser")
# 获取所有的 class 属性为 dayTitle，返回 Tag 类
times = soup.find_all(class_="dayTitle")

# for i in times:
#     print i.a.string  # 获取 a 标签的文本
title = soup.find_all(class_="postTitle")
# for i in title:
#     print i.a.string
# 读取摘要内容

descs = soup.find_all(class_="postCon")

# for i in descs:
#     # tag 的 .contents 属性可以将 tag 的子节点以列表的方式输出
#     c = i.div.contents[0]  # 取第一个
```

```
#     print c

for i, j, k in zip(times,title,descs):
    print i.a.string
    print j.a.string
    print k.div.contents[0]
    print ""
```

5.2 bs4 四种对象类

前言

本篇详细介绍 beautifulsoup4 的功能，从最基础的开始讲起，让小伙伴们都能入门

5.2.1 读取 HTML 页面

1.先写一个简单的 html 页面，把以下内容 copy 出来，保存为 html 格式文件

```
<meta charset="UTF-8"> <!-- for HTML5 -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<html><head><title>yoyo ketang</title></head>
<body>
<b><!--Hey, this in comment!--></b>
<p class="title"><b>yoyoketang</b></p>
<p class="yoyo">这里是我的微信公众号：yoyoketang
<a     href="http://www.cnblogs.com/yoyoketang/tag/fiddler/"     class="sister"
id="link1">fiddler</a>,
<a     href="http://www.cnblogs.com/yoyoketang/tag/python/"     class="sister"
id="link2">python</a>,
<a     href="http://www.cnblogs.com/yoyoketang/tag/selenium/"     class="sister"
id="link3">selenium</a>;
快来关注吧！</p>
<p class="story">...</p>
```

2.为了学习方便，可以把此 html 文件和脚本放同一文件夹下

3.用 python 的 open 函数读取这个 html,如下图能正确打印出来，说明读取成功了

```
# coding:utf-8
from bs4 import BeautifulSoup

yoyo = open("yoyo.html")
print yoyo.read()
```

```
test_03  test_03
D:\test\python2\python.exe D:/test/yoyotest/testbs/test_03.py
<meta charset="UTF-8"> <!-- for HTML5 -->
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<html><head><title>yoyo ketang</title></head>
<body>
<p class="title"><b>yoyoketang</b></p>

<p class="yoyo">这里是我的微信公众号：yoyoketang
<a href="http://www.cnblogs.com/yoyoketang/tag/fiddler/" class="sister" id="link1">fiddler</a>,
<a href="http://www.cnblogs.com/yoyoketang/tag/python/" class="sister" id="link2">python</a>,
<a href="http://www.cnblogs.com/yoyoketang/tag/selenium/" class="sister" id="link3">selenium</a>;
快来关注吧！</p>

<p class="story">...</p>
```

5.2.2 解析器：html.parser

1.用 BeautifulSoup 的构造方法,就能得到一个文档的对象，可以传入一段字符串或一个文件句柄

2.如果我们调用 BeautifulSoup 这个类的时候，不带"html.parser"参数，这时候会有个 waring 的，下图红色框框区域给出了建议。

```
from bs4 import BeautifulSoup

yoyo = open("yoyo.html")

soup = BeautifulSoup(yoyo)
#prettify()可以自动解析成html格式
print soup.prettify()
```

```
test_03  test_03

The code that caused this warning is on line 6 of the file D:/test/yoyotest/testbs/test_03.py. To

 BeautifulSoup(YOUR_MARKUP})

to this:

 BeautifulSoup(YOUR_MARKUP, "html.parser")

  markup_type=markup_type))
```

3. html.parser 这个是 python 库里面自带的解析器，无需安装。

4. prettify()这个方法是把文件解析成 html 格式，用 html 的标准格式输出（有缩进的）

```
# coding:utf-8
from bs4 import BeautifulSoup

yoyo = open("yoyo.html")

soup = BeautifulSoup(yoyo, "html.parser")
print soup.prettify()
```

```
test_03  test_03
D:\test\python2\python.exe D:/test/yoyotest/testbs/test_03.py
<meta charset="utf-8"/>
<!-- for HTML5 -->
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
 <html>
  <head>
   <title>
    yoyo ketang
   </title>
  </head>
  <body>
```

5.2.3 对象的种类

1. Beautiful Soup 将复杂 HTML 文档转换成一个复杂的树形结构,每个节点都是 Python 对象,

所有对象可以归纳为 4 种:

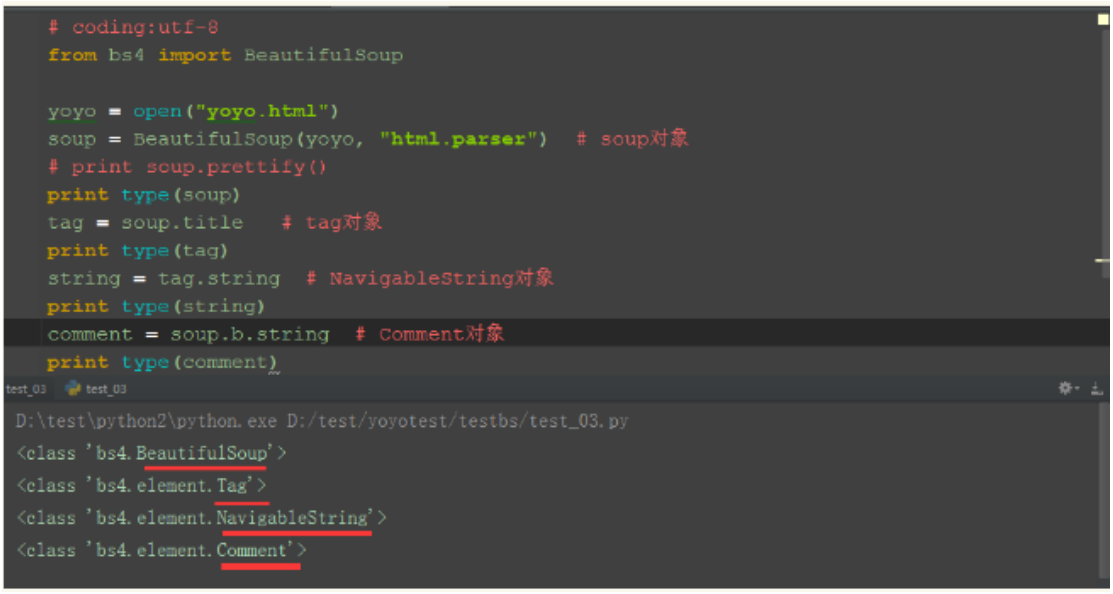
Tag ：　标签对象，如： yoyoketang ，这就是一个标签

NavigableString ：字符对象，如：这里是我的微信公众号：yoyoketang

BeautifulSoup　：就是整个 html 对象

Comment　　：注释对象，如：!-- for HTML5 --，它其实就是一个特殊 NavigableString

2.弄清楚了这四个对象，后面所有的操作但是基于这四个（其实最后面一个 comment 可以忽略，几乎用不到）



5.2.4 Tag 对象

1.html 里面识别一个 tag，其实一般像这种：

yoyoketang
yoyoketang
<head><title>yoyo ketang</title></head>

以上这些都是 tag,很明显都是以<xx 开头</xx> 结尾这种，就是一个完整的 tag

2.通过标签的名称，来获取 tag 对象

3.如果有多个相同的标签名称，返回的是第一个

```
# coding:utf-8
from bs4 import BeautifulSoup

yoyo = open("yoyo.html")
soup = BeautifulSoup(yoyo, "html.parser")  # soup对象
tag1 = soup.head  # <head tag
print type(tag1)  # 返回的是Tag对象
print tag1
tag2 = soup.title  # <title tag
print tag2
# 如果有多个相同的标签，返回的是第一个
tag3 = soup.a      # <a tag
print tag3
```

```
test_03  test_03
D:\test\python2\python.exe D:/test/yoyotest/testbs/test_03.py
<class 'bs4.element.Tag'>
<head><title>yoyo ketang</title></head>
<title>yoyo ketang</title>
<a class="sister" href="http://www.cnblogs.com/yoyoketang/tag/fiddler/" id="link1">fiddler</a>
```

5.2.5 Tag 对象：name 属性

1. tag 的 name 属性，每一个 tag 都有 name 属性，用.name 获取，如上面的三个 name 属性分别为 p, b, head。也就是<xx 开头的，这里的 xx 就是它的 name 属性

2. html 是一个树状结构的，每个 tag 都有属性，整个 html 对象也有 name 属性，它的属性是：[document]

```
# coding:utf-8
from bs4 import BeautifulSoup

yoyo = open("yoyo.html")
soup = BeautifulSoup(yoyo, "html.parser")  # soup对象
tag1 = soup.head  # <head tag
print tag1.name
tag2 = soup.title  # <title tag
print tag2.name
# 多个相同标签的时候，取第一个
tag3 = soup.a      # <a tag
print tag3.name
# 整个html对象的属性
print soup.name
```

```
test_03  test_03
D:\test\python2\python.exe D:/test/yoyotest/testbs/test_03.py
head
title
a
[document]
```

5.2.6 Tag 对象： Attributes

1.tag.attrs 可以打印出所有的属性，可以看出是个字典格式的

2.那么获取其中的某一个属性，就跟操作字典一样，如：tag["href"]

3.由于 class 属性一般可以为多个，中间空格隔开，所以 class 属性获取的是一个 list 类型：[u'sister']

4.tag.string，这里 Tag 对象变成 NavigableString 对象了，呵呵小伙伴们要懵逼了吧。其实打印出来就是一个字符串。

5.2.7 发福利

1.爬糗事百科首页的段子

```
# coding:utf-8
from bs4 import BeautifulSoup
import requests

r = requests.get("https://www.qiushibaike.com/")
qiubai = r.content
soup = BeautifulSoup(qiubai, "html.parser")
duanzi = soup.find_all(class_="content")

for i in duanzi:
    # tag 的 .contents 属性可以将 tag 的子节点以列表的方式输出
    duan = i.span.contents[0]  # 取第一个
    print duan
```

对 python 接口自动化有兴趣的，可以加 python 接口自动化 QQ 群：226296743

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

5.3 爬网站图片

前言

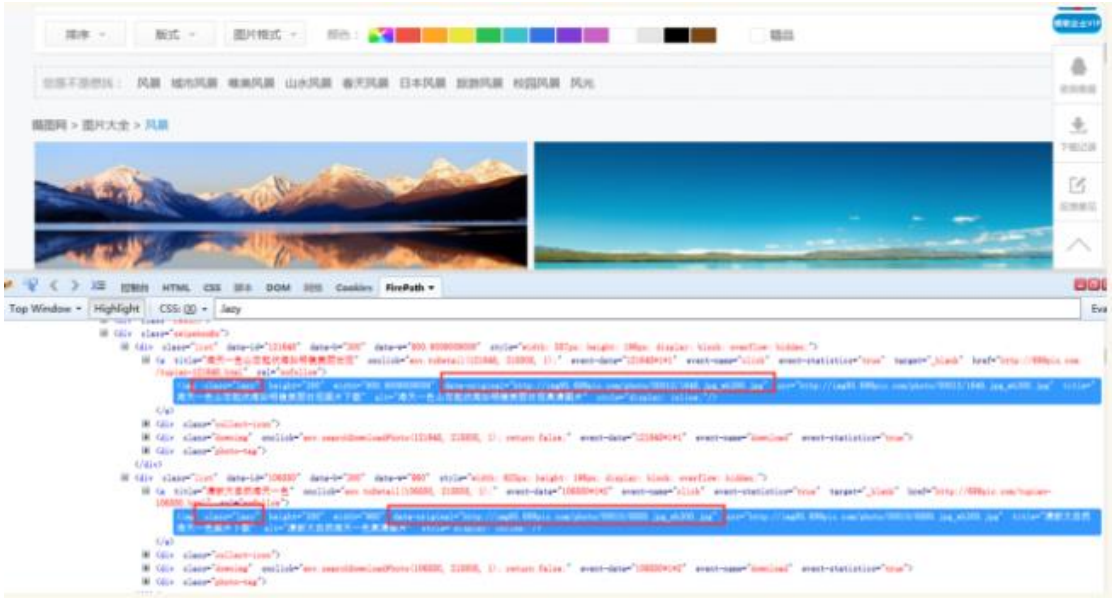
本篇手把手教大家如何爬取网站上的图片，并保存到本地电脑

5.3.1 目标网站

1.随便打开一个风景图的网站：http://699pic.com/sousuo-218808-13-1.html

2.用 firebug 定位，打开 firepath 里 css 定位目标图片

3.从下图可以看出，所有的图片都是 img 标签，class 属性都是 lazy



5.3.2 用 find_all 找出所有的标签

1.find_all(class_="lazy")获取所有的图片对象标签

2.从标签里面提出 jpg 的 url 地址和 title

```
# coding:utf-8
from bs4 import BeautifulSoup
```

```
import requests
import os

r = requests.get("http://699pic.com/sousuo-218808-13-1.html")
fengjing = r.content
soup = BeautifulSoup(fengjing, "html.parser")
# 找出所有的标签
images = soup.find_all(class_="lazy")
# print images # 返回 list 对象

for i in images:
    jpg_rl = i["data-original"]  # 获取 url 地址
    title = i["title"]           # 返回 title 名称
    print title
    print jpg_rl
    print ""
```

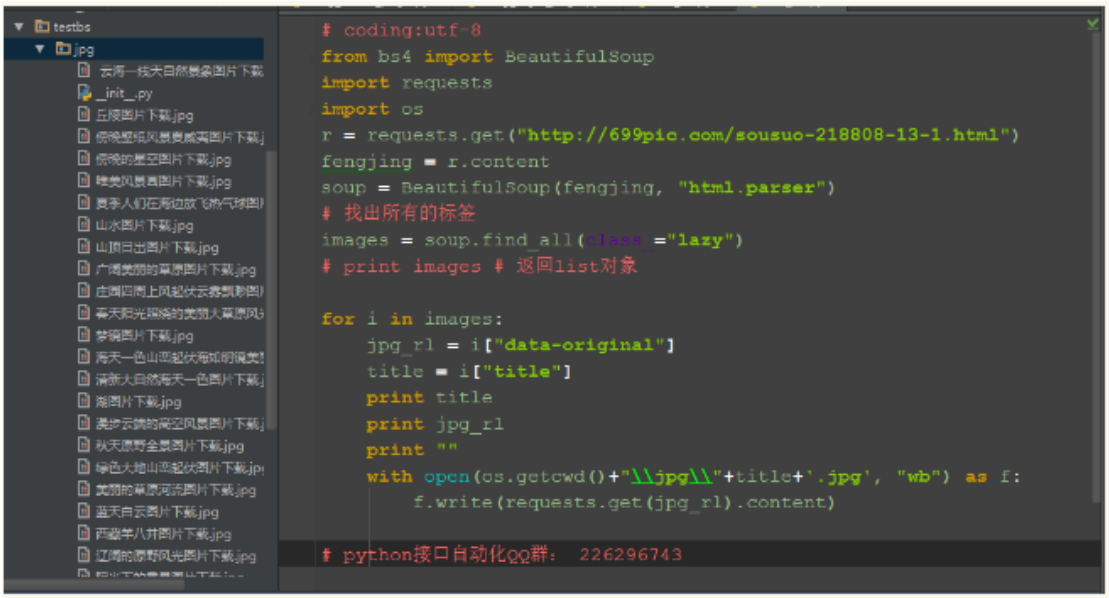
5.3.3 保存图片

1.在当前脚本文件夹下创建一个 jpg 的子文件夹

2.导入 os 模块，os.getcwd()这个方法可以获取当前脚本的路径

3.用 open 打开写入本地电脑的文件路径，命名为：os.getcwd()+"\\jpg\\"+title+'.jpg'（命名重复的话，会被覆盖掉）

4.requests 里 get 打开图片的 url 地址，content 方法返回的是二进制流文件，可以直接写到本地



5.3.4 参考代码

```
# coding:utf-8
from bs4 import BeautifulSoup
import requests
import os

r = requests.get("http://699pic.com/sousuo-218808-13-1.html")
fengjing = r.content
soup = BeautifulSoup(fengjing, "html.parser")

# 找出所有的标签
images = soup.find_all(class_="lazy")
# print images # 返回 list 对象

for i in images:
    try:
        jpg_rl = i["data-original"]
        title = i["title"]
        print title
        print jpg_rl
        print ""
        # 保存图片
        with open(os.getcwd()+"\\jpg\\"+title+'.jpg', "wb") as f:
            f.write(requests.get(jpg_rl).content)
```

```
    except:
        pass
```

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

5.4 子节点

前言

很多时候我们无法直接定位到某个元素，我们可以先定位它的父元素，通过父元素来找子元素就比较容易

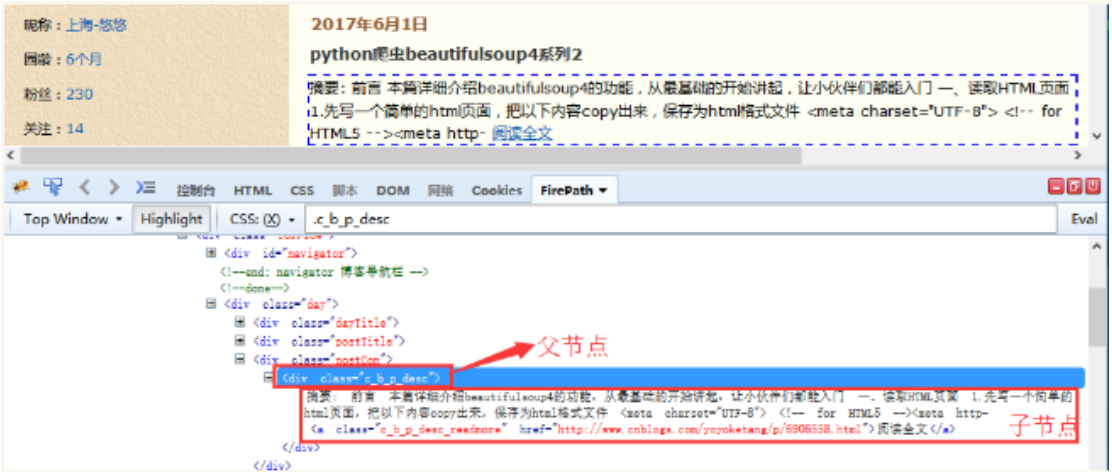
5.4.1 子节点

1.以博客园首页的摘要为例： 这个 tag 为起点

2.那么 div 这个 tag 就是父节点

3."摘要: 前言 本篇详细。。。"这个 string 就是上面 div 的子节点（string 通常看成是一个 tag 的子节点）

4. 阅读全文这个也是 div 的子节点



5.4.2 .contents

1.tag 对象 contents 可以获取所有的子节点，返回的是 list

2.len()函数统计子节点的个数

3.通过下标可以取出对应的子节点

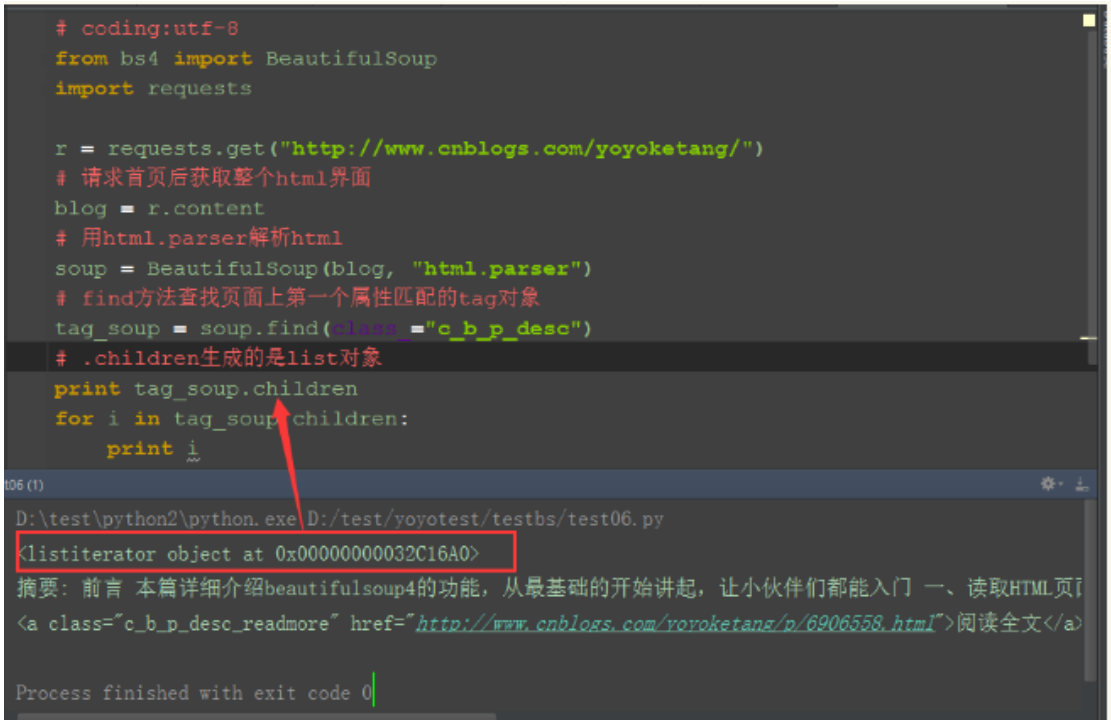
```
# coding:utf-8
from bs4 import BeautifulSoup
import requests

r = requests.get("http://www.cnblogs.com/yoyoketang/")
# 请求首页后获取整个 html 界面
blog = r.content

# 用 html.parser 解析 html
soup = BeautifulSoup(blog, "html.parser")

# find 方法查找页面上第一个属性匹配的 tag 对象
tag_soup = soup.find(class_="c_b_p_desc")

# len 函数获取子节点的个数
print len(tag_soup.contents)

# 循环打印出子节点
for i in tag_soup.contents:
    print i

# 通过下标取出第 1 个 string 子节点
print tag_soup.contents[0]

# 通过下标取出第 2 个 a 子节点
print tag_soup.contents[1]
```

5.4.3 .children

1.点 children 这个生成的是 list 对象，跟上面的点 contents 功能一样

2. 只是这里是 list 对象，就只能 for 循环读出了，不能通过下标获取

（一般上面那个 contents 用的比较多，可能 children 性能更快吧，我猜想的嘿嘿！）



5.4.4 .descendants

1. 上面的 contents 只能获取该元素的直接子节点，如果这个元素的子节点又有子节点（也就是孙节点了），
这时候获取所有的子孙节点就可以用.descendants 方法

2. 获取 div 的子节点有两个，子孙节点有三个，因为 a 标签下还有个“阅读全文”这个 string 子节点

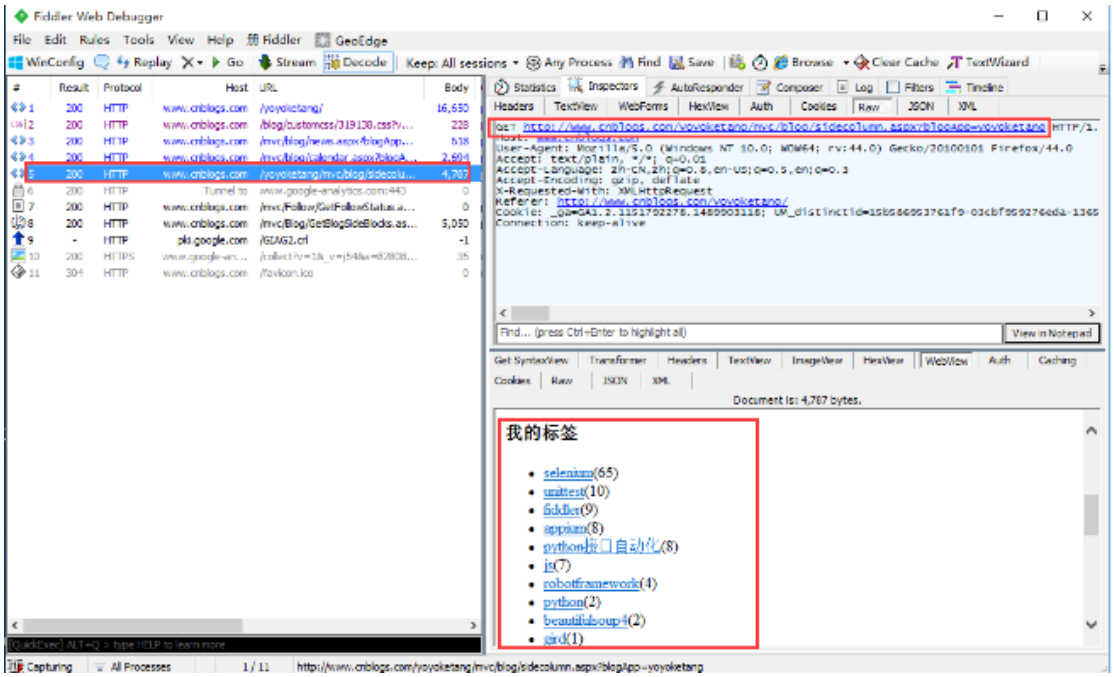
```
# coding:utf-8

from bs4 import BeautifulSoup
import requests

r = requests.get("http://www.cnblogs.com/yoyoketang/")

# 请求首页后获取整个 html 界面
blog = r.content

# 用 html.parser 解析 html
```

```
soup = BeautifulSoup(blog, "html.parser")

# find 方法查找页面上第一个属性匹配的 tag 对象
tag_soup = soup.find(class_="c_b_p_desc")

# len 函数获取子节点的个数
print len(list(tag_soup.children))

# 获取子孙节点的个数
print len(list(tag_soup.descendants))

for i in tag_soup.descendants:
    print i
```

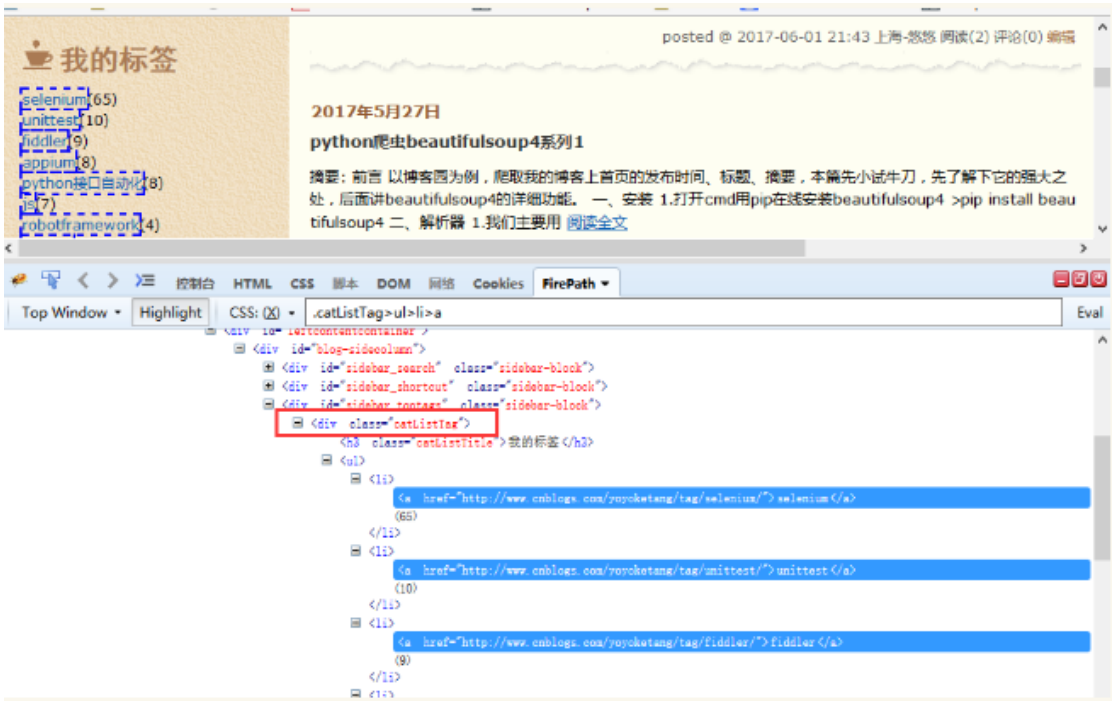
5.4.5 爬取博客首页的标签内容

1. 博客左侧的标签并不是这个链接：http://www.cnblogs.com/yoyoketang/

2. 通过抓包可以看到，这个 url 地址：
   http://www.cnblogs.com/yoyoketang/mvc/blog/sidecolumn.aspx?blogApp=yoyoketang



3. 可以先定位父元素：



5.4.6 参考代码

```
# coding:utf-8
from bs4 import BeautifulSoup
```

```
import requests

r = requests.get("http://www.cnblogs.com/yoyoketang/mvc/blog/sidecolumn.aspx?blogApp=yoyoketang")

# 请求首页后获取整个 html 界面
blog = r.content

# 用 html.parser 解析 html
soup = BeautifulSoup(blog, "html.parser")

tag_soup = soup.find(class_="catListTag")

# print body.prettify()

ul_soup = tag_soup.find_all("a")
print ul_soup

for i in ul_soup:
    print i.string
```

5.5 bs4 官方文档

官方文档：http://beautifulsoup.readthedocs.io/zh_CN/latest/

内容目录

Beautiful Soup 4.4.0 文档
- 寻求帮助

快速开始

安装 Beautiful Soup
- 安装完成后的问题
- 安装解析器

如何使用

对象的种类
- Tag
  - Name
  - Attributes
    - 多值属性
- 可以遍历的字符串
- BeautifulSoup
- 注释及特殊字符串

遍历文档树
- 子节点
  - tag的名字
  - .contents 和 .children
  - .descendants
  - .string
  - .strings 和 stripped_strings
- 父节点
  - .parent
  - .parents
- 兄弟节点
  - .next_sibling 和 .previous_sibling

Beautiful Soup 4.4.0 文档

Beautiful Soup 是一个可以从HTML或XML文件中提取数据的Python库.它能够通过你喜欢的转换器实现惯用的文档导航,查找,修改文档的方式.Beautiful Soup会帮你节省数小时甚至数天的工作时间.

这篇文档介绍了BeautifulSoup4中所有主要特性,并且有小例子,让我来向你展示它适合做什么,如何工作,怎样使用,如何达到你想要的效果,和处理异常情况.

文档中出现的例子在Python2.7和Python3.2中的执行结果相同

你可能在寻找 Beautiful Soup3 的文档,Beautiful Soup 3 目前已经停止开发,我们推荐在现在的项目中使用Beautiful Soup 4, 移植到BS4

这篇帮助文档已经被翻译成了其它语言:

- 这篇文档当然还有中文版.
- このページは日本語で利用できます(外部リンク)
- 이 문서는 한국어 번역도 가능합니다. (외부 링크)

寻求帮助

如果你有关于BeautifulSoup的问题,可以发送邮件到 讨论组.如果你的问题包含了一段需要转换的HTML代码,那么确保你提的问题描述中附带这段HTML文档的 代码诊断 [1]

快速开始

第 6 章 数据库读取

6.1 读 excel 数据 xlrd

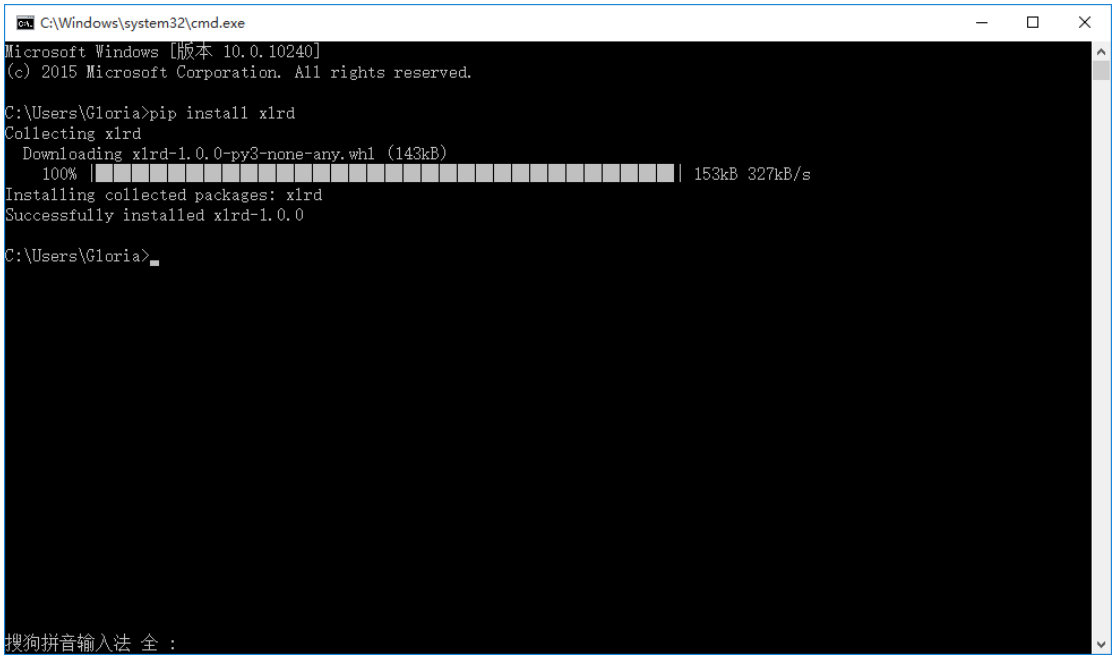
前言

当登录的账号有多个的时候，我们一般用 excel 存放测试数据，本节课介绍，python 读取 excel 方法，并保存为字典格式。

6.1.1 环境准备

1.先安装 xlrd 模块，打开 cmd，输入 pip install xlrd 在线安装

```
>>pip install xlrd
```



6.1.2 基本操作

1.exlce 基本操作方法如下

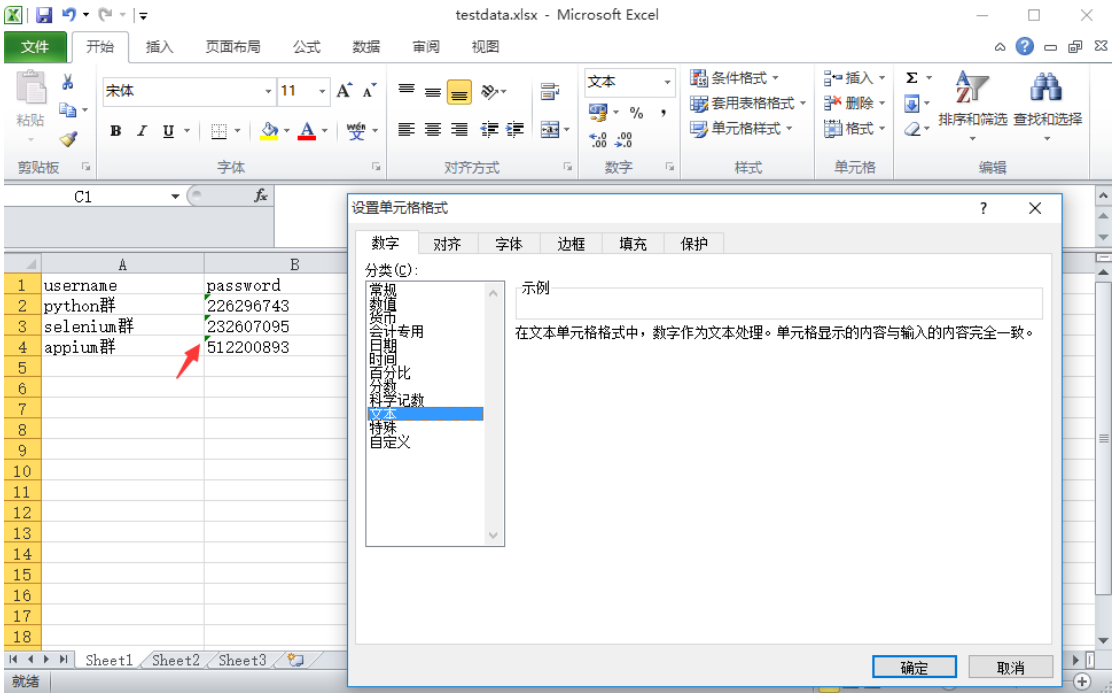
```
# 打开 exlce 表格，参数是文件路径
data = xlrd.open_workbook('test.xlsx')

# table = data.sheets()[0]          #  通过索引顺序获取
# table = data.sheet_by_index(0)    #  通过索引顺序获取

table = data.sheet_by_name(u'Sheet1')  # 通过名称获取

nrows = table.nrows  # 获取总行数
ncols = table.ncols  # 获取总列数

#  获取一行或一列的值，参数是第几行
print table.row_values(0)  # 获取第一行值
print table.col_values(0)  # 获取第一列值
```

```
# coding:utf-8
import xlrd

# 打开exlce表格
data = xlrd.open_workbook('testdata.xlsx')

# table = data.sheets()[0]            #  通过索引顺序获取
# table = data.sheet_by_index(0)      #  通过索引顺序获取
table = data.sheet_by_name(u'Sheet1')  # 通过名称获取

nrows = table.nrows  # 获取总行数
ncols = table.ncols  # 获取总列数

print table.row_values(0)  # 获取第一行值
print table.col_values(0)  # 获取第一列值
```

6.1.3 excel 存放数据

1. 在 excel 中存放数据，第一行为标题，也就是对应字典里面的 key 值，如：username，password

2. 如果 excel 数据中有纯数字的一定要右键》设置单元格格式》文本格式，要不然读取的数据是浮点数
（先设置单元格格式后编辑，编辑成功左上角有个小三角图标）



6.1.4 封装读取方法

1.最终读取的数据是多个字典的 list 类型数据，第一行数据就是字典里的 key 值，从第二行开始一一对应 value 值

2.封装好后的代码如下

```
# coding:utf-8
import xlrd
class ExcelUtil():
    def __init__(self, excelPath, sheetName):
        self.data = xlrd.open_workbook(excelPath)
        self.table = self.data.sheet_by_name(sheetName)
        # 获取第一行作为 key 值
        self.keys = self.table.row_values(0)
        # 获取总行数
        self.rowNum = self.table.nrows
        # 获取总列数
        self.colNum = self.table.ncols
    def dict_data(self):
        if self.rowNum <= 1:
            print("总行数小于 1")
        else:
            r = []
            j=1
            for i in range(self.rowNum-1):
                s = {}
                # 从第二行取对应 values 值
                values = self.table.row_values(j)
                for x in range(self.colNum):
                    s[self.keys[x]] = values[x]
                r.append(s)
                j+=1
            return r
if __name__ == "__main__":
    filepath = "D:\\test\\web-project\\5ke\\testdata.xlsx"
    sheetName = "Sheet1"
    data = ExcelUtil(filepath, sheetName)
    print data.dict_data()
```

运行结果：

```
[{u'username': u'python\u7fa4', u'password': u'226296743'},
{u'username': u'selenium\u7fa4', u'password': u'232607095'},
```

```
{u'username': u'appium\u7fa4', u'password': u'512200893'}]
```

6.2 操作 mysql

封装操作 mysql 代码：

```
# coding:utf-8
import MySQLdb
# import os
# os.environ['NLS_LANG'] = 'SIMPLIFIED CHINESE_CHINA.UTF8'
mysql_info = {"host": '192.168.xx.xx',
              "port": 3306,
              "user": 'root',
              "passwd": '11',
              "db": 'xxx',
              "charset": 'utf8'}
class MysqlUtil():
    '''
    mysql 数据库相关操作
    连接数据库信息：mysql_info
    创建游标：mysql_execute
    查询某个字段对应的字符串：mysql_getstring
    查询一组数据：mysql_getrows
    关闭 mysql 连接：mysql_close
    '''
    def __init__(self):
        self.db_info = mysql_info
        u'''连接池方式'''
        self.conn = MysqlUtil.__getConnect(self.db_info)
    @staticmethod
    def __getConnect(db_info):
        '''静态方法，从连接池中取出连接'''
        try:
            conn = MySQLdb.connect(host=db_info['host'],
                                   port=db_info['port'],
                                   user=db_info['user'],
                                   passwd=db_info['passwd'],
                                   db=db_info['db'],
                                   charset=db_info['charset'])
            return conn
        except Exception as a:
```

```
            print("数据库连接异常：%s"%a)
    def mysql_execute(self, sql):
        '''执行 sql 语句'''
        cur = self.conn.cursor()
        try:
            cur.execute(sql)
        except Exception as a:
            self.conn.rollback()          # sql 执行异常后回滚
            print("执行 SQL 语句出现异常：%s"%a)
        else:
            cur.close()
            self.conn.commit()           # sql 无异常时提交
    def mysql_getrows(self, sql):
        ''' 返回查询结果'''
        cur = self.conn.cursor()
        try:
            cur.execute(sql)
        except Exception as a:
            print("执行 SQL 语句出现异常：%s"%a)
        else:
            rows = cur.fetchall()
            cur.close()
            return rows
    def mysql_getstring(self, sql):
        '''查询某个字段的对应值'''
        rows = self.mysql_getrows(sql)
        if rows != None:
            for row in rows:
                for i in row:
                    return i
    def mysql_close(self):
        ''' 关闭 close mysql'''
        try:
            self.conn.close()
        except Exception as a:
            print("数据库关闭时异常：%s"%a)
# MySQLdb.connect()      建立数据库连接
# cur = conn.cursor()     #通过获取到的数据库连接 conn 下的 cursor()方法来创建游标。
# cur.execute()     #过游标 cur 操作 execute()方法可以写入纯 sql 语句。通过 execute()方法中写如 sql 语句来对数据进行操作。
# cur.close()      # cur.close() 关闭游标
# conn.commit()    # conn.commit()方法在提交事物，在向数据库插入(或
```

```
update)一条数据时必须要有这个方法，否则数据不会被真正的插入。
# conn.rollback() # 发生错误时候回滚
# conn.close()     # Conn.close()关闭数据库连接
if __name__ == "__main__":
    mysql = MysqlUtil()
    sql = "SELECT * FROM xxx  where ID = 'xxx'"
    mysql.mysql_execute(sql)
    print mysql.mysql_getrows(sql)
    print mysql.mysql_getstring(sql)
    mysql.mysql_close()
```

6.3 操作 oracle

cx_Oracle 库的安装还是有点复杂的，这里先贴代码，有空了再整理详细教程出来

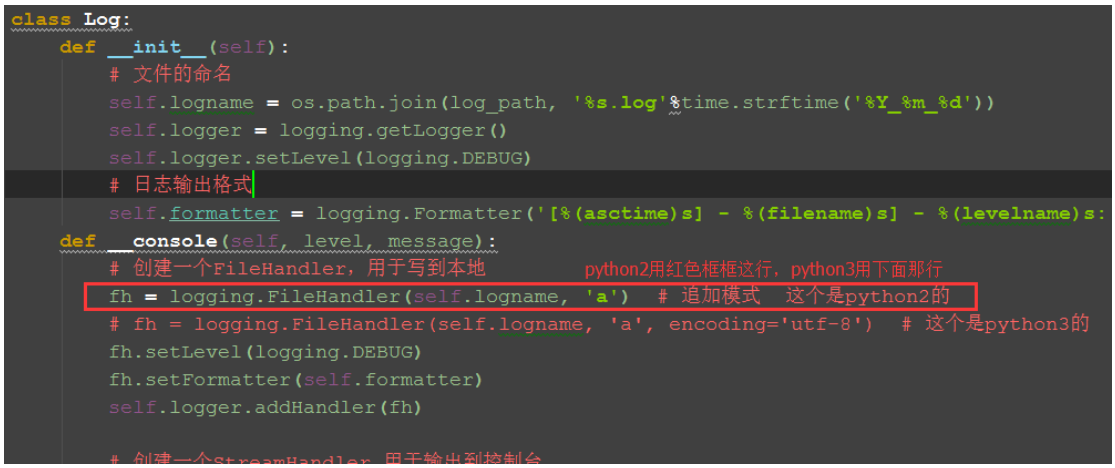
```
# coding:utf-8
import cx_Oracle
import os
os.environ['NLS_LANG'] = 'SIMPLIFIED CHINESE_CHINA.UTF8'
u'''Oracle 数据库相关操作
连接数据库名称如：xxx
查询一组数据：oracle_getrows
查询某个字段对应的字符串：oracle_getstring
执行 sql 语句：oracle_sql
关闭 oracle 连接：oracle_close
'''
dbname = {"user": "xxx",
          "pwd": "password",
          "dsn": "192.168.x.xxx:1521/ORCL"}
class OracleUtil():
    def __init__(self):
        ''' 连接池方式'''
        self.db_info = dbname
        self.conn = OracleUtil.__getConnect(self.db_info)

    @staticmethod
    def __getConnect(db_info):
        ''' 静态方法，从连接池中取出连接'''
        try:
            con = cx_Oracle.connect(db_info['user'], db_info['pwd'],
db_info['dsn'])
```

```
            return con
        except Exception as a:
            print("数据库连接异常：%s"%a)

    def oracle_getrows(self, sql):
        ''' 执行查询 sql'''
        try:
            cursor = self.conn.cursor()
            try:
                cursor.execute(sql)
                rows = cursor.fetchall()
                return rows
            except Exception as a:
                print("执行 sql 出现异常：%s"%a)
            finally:
                cursor.close()
        except Exception as a:
            print("数据库连接异常：%s"%a)

    def oracle_getstring(self, sql):
        ''' 查询某个字段的对应值'''
        rows = self.oracle_getrows(sql)
        if rows != None:
            for row in rows:
                for i in row:
                    return i

    def oracle_sql(self, sql):
        ''' 执行 sql 语句'''
        try:
            cursor = self.conn.cursor()
            try:
                cursor.execute(sql)
            except Exception as a:
                print("执行 sql 出现异常：%s"%a)
            finally:
                cursor.close()
        except Exception as a:
            print("数据库连接异常：%s"%a)

    def orcle_close(self):
        ''' 关闭 orcle 连接'''
        try:
```

```
            self.conn.close()
        except Exception as a:
            print("数据库关闭时异常：%s"%a)

if __name__ == "__main__":
        oracl = OracleUtil()
        sql = "select *from xxxe where id = 'xxx'"
        s = oracl.oracle_getstring(sql)
        print s
        oracl.orcle_close()
```

6.4 logger 日志

Python2 和 python3 的代码有一点点区别，见下图




```
# coding:utf-8
import logging, time, os
# 这个是日志保存本地的路径
log_path = "D:\\test\\newp\\report"
class Log:
    def __init__(self):
        # 文件的命名
        self.logname = os.path.join(log_path,
'%s.log'%time.strftime('%Y_%m_%d'))
        self.logger = logging.getLogger()
        self.logger.setLevel(logging.DEBUG)
        # 日志输出格式
        self.formatter = logging.Formatter('[%(asctime)s]
- %(filename)s] - %(levelname)s: %(message)s')
    def __console(self, level, message):
```

```
        # 创建一个 FileHandler，用于写到本地
        fh = logging.FileHandler(self.logname, 'a')  # 追加模式  这个是 python2 的
        # fh = logging.FileHandler(self.logname, 'a', encoding='utf-8') # 这个是 python3 的
        fh.setLevel(logging.DEBUG)
        fh.setFormatter(self.formatter)
        self.logger.addHandler(fh)
        # 创建一个 StreamHandler,用于输出到控制台
        ch = logging.StreamHandler()
        ch.setLevel(logging.DEBUG)
        ch.setFormatter(self.formatter)
        self.logger.addHandler(ch)
        if level == 'info':
            self.logger.info(message)
        elif level == 'debug':
            self.logger.debug(message)
        elif level == 'warning':
            self.logger.warning(message)
        elif level == 'error':
            self.logger.error(message)
        # 这两行代码是为了避免日志输出重复问题
        self.logger.removeHandler(ch)
        self.logger.removeHandler(fh)
        # 关闭打开的文件
        fh.close()
    def debug(self, message):
        self.__console('debug', message)
    def info(self, message):
        self.__console('info', message)
    def warning(self, message):
        self.__console('warning', message)
    def error(self, message):
        self.__console('error', message)
if __name__ == "__main__":
   log = Log()
   log.info("---测试开始----")
   log.info("输入密码")
   log.warning("----测试结束----")
```

6.5 发送邮件

前言

本篇总结了 QQ 邮箱和 163 邮箱发送邮件，邮件包含 html 中文和附件，可以发给多个收件人，专治各种不行，总之看完这篇麻麻再也不用担心我的邮件收不到了。以下代码兼容 python2 和 python3，运行无异常，放心大胆食用。

6.5.1 163 邮箱

1. 先导入 smtplib 库用来发送邮件，导入 MIMEText 库用来做纯文本的邮件模板

2. 先准备几个跟发邮件相关的参数，每个邮箱的发件服务器都不一样，以 163 为例，百度搜到发件服务器为：smtp.163.com

```
# coding:utf-8
import smtplib
from email.mime.text import MIMEText

# ----------1.跟发件相关的参数------
smtpserver = "smtp.163.com"          # 发件服务器
port = 0                             # 端口
sender = "yoyo@163.com"              # 账号
psw = "***************"              # 密码
receiver = "283340479@qq.com"        # 接收人
```

3. 接下来就是写邮件的主题和正文内容，正文这里用 html 格式的

4. 最后调用发件服务

```
# ----------2.编辑邮件的内容------
subject = "这个是主题163"
body = '<p>这个是发送的163邮件</p>'  # 定义邮件正文为html格式
msg = MIMEText(body, "html", "utf-8")
msg['from'] = sender
msg['to'] = "283340479@qq.com"
msg['subject'] = subject

# ----------3.发送邮件------
smtp = smtplib.SMTP()
smtp.connect(smtpserver)                                    # 连服务器
smtp.login(sender, psw)                                     # 登录
smtp.sendmail(sender, receiver, msg.as_string())  # 发送
smtp.quit()                                                 # 关闭
```

5.参考代码：

```
# coding:utf-8
import smtplib
from email.mime.text import MIMEText

# ----------1.跟发件相关的参数------
smtpserver = "smtp.163.com"              # 发件服务器
port = 0                                                    # 端口
sender = "yoyo@163.com"                  # 账号
psw = "**************"                                  # 密码
receiver = "283340479@qq.com"          # 接收人

# ----------2.编辑邮件的内容------
subject = "这个是主题 163"
body = '<p>这个是发送的 163 邮件</p>'  # 定义邮件正文为 html 格式
msg = MIMEText(body, "html", "utf-8")
msg['from'] = sender
msg['to'] = "283340479@qq.com"
msg['subject'] = subject

# ----------3.发送邮件------
smtp = smtplib.SMTP()
smtp.connect(smtpserver)      # 连服务器
smtp.login(sender, psw)         # 登录
smtp.sendmail(sender, receiver, msg.as_string())  # 发送
smtp.quit()          # 关闭
```

6.5.2 QQ 邮件

1. QQ 邮箱是需要 SSL 认证的，这种邮箱跟上面的就有点不一样了

QQ邮箱POP3 和 SMTP 服务器地址设置如下：

邮箱	POP3服务器（端口110）	SMTP服务器（端口25）
qq.com	pop.qq.com	smtp.qq.com

注：

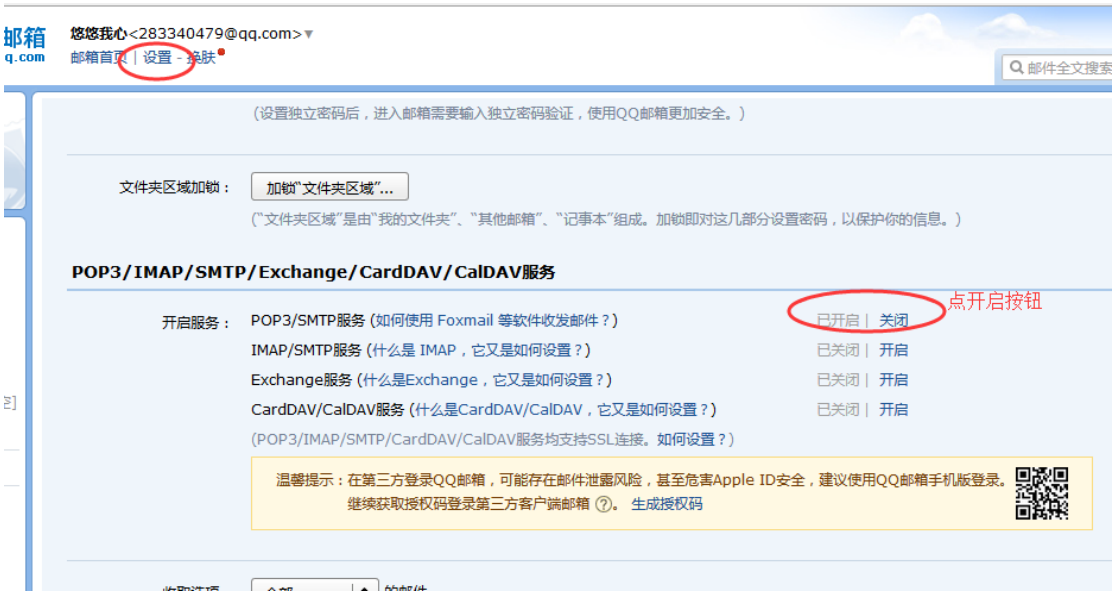
1、SMTP服务器需要身份验证。

2、如果是设置POP3和SMTP的SSL加密方式，则端口如下：

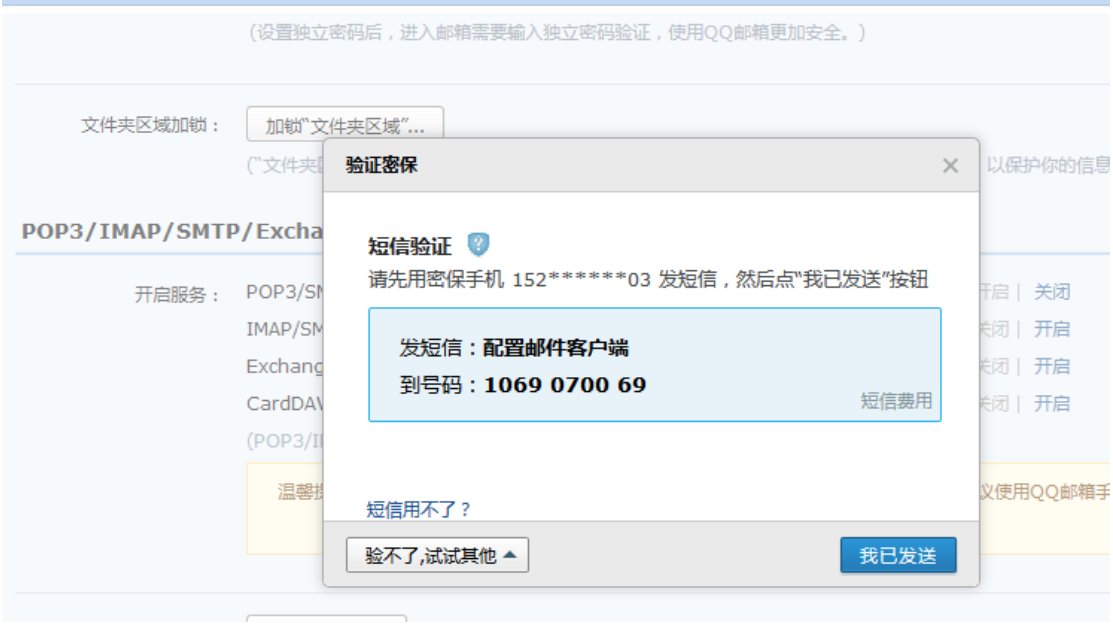
1）POP3服务器（端口995）；

2）SMTP服务器（端口465或587）。

2. 找到 QQ 邮箱授权码，打开 QQ 邮箱-设置-账号-POP3 开启服务-开启
（如果已经开启了，不知道授权码，就点温馨提示里面的‘生成授权码’）



3. 发验证短信获取授权码，照着提示发个短信，如何点我已发送，就会收到授权码了



4.收到授权码后复制，保存下来，这个就可以当 QQ 邮箱的密码了

5.QQ 邮箱发送邮件代码，跟 163 有点不一样，如下图红色框框：

```
# ----------3.发送邮件------
# smtp = smtplib.SMTP()
# smtp.connect(smtpserver)                              # 连服务器
smtp = smtplib.SMTP_SSL(smtpserver, port)
smtp.login(sender, psw)                                 # 登录
smtp.sendmail(sender, receiver, msg.as_string())        # 发送
smtp.quit()                                             # 关闭
```

6.参考代码：

```
# coding:utf-8
import smtplib
from email.mime.text import MIMEText

# ----------1.跟发件相关的参数------
# smtpserver = "smtp.163.com"          # 发件服务器
smtpserver = "smtp.qq.com"
port = 465                             # 端口
sender = "283340479@qq.com"            # 账号
psw = "**************"                  # 密码
receiver = "283340479@qq.com"          # 接收人

# ----------2.编辑邮件的内容------
subject = "这个是主题 QQ"
body = '<p>这个是发送的 QQ 邮件</p>'      # 定义邮件正文为 html 格式
msg = MIMEText(body, "html", "utf-8")
msg['from'] = sender
msg['to'] = "283340479@qq.com"
msg['subject'] = subject

# ----------3.发送邮件------
# smtp = smtplib.SMTP()
# smtp.connect(smtpserver)                              # 连服务器
smtp = smtplib.SMTP_SSL(smtpserver, port)
smtp.login(sender, psw)                                 # 登录
smtp.sendmail(sender, receiver, msg.as_string())        # 发送
smtp.quit()                                             # 关闭
```

6.5.3 兼容 163 和 QQ 邮箱

1.如果想兼容上面两种方式发送邮件，只需把第三块内容稍微改下，如下所示

```
# ----------3.发送邮件------
try:
    smtp = smtplib.SMTP()
    smtp.connect(smtpserver)                          # 连服务器
    smtp.login(sender, psw)
except:
    smtp = smtplib.SMTP_SSL(smtpserver, port)
    smtp.login(sender, psw)                           # 登录
smtp.sendmail(sender, receiver, msg.as_string())      # 发送
smtp.quit()                                           # 关闭
```

6.5.4 发送带附件

1.上面的 MIMEText 只能发送正文，无法带附件，发送带附件的需要导入另外一个模块 MIMEMultipart

2.先读取要发送文件的内容，file_path 是路径的参数名

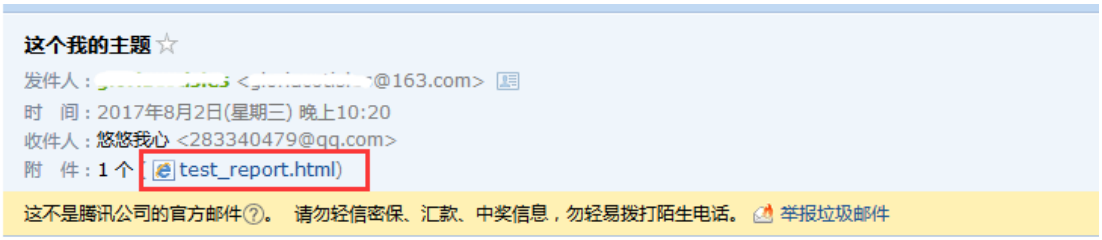
3.下图红色框框 file_name 参数是发送的附件重新命名

4.参考代码：

```
# coding:utf-8
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
# ----------1.跟发件相关的参数------
smtpserver = "smtp.163.com"          # 发件服务器
port = 0                                    # 端口
sender = "yoyo@163.com"              # 账号
psw = "**********"                          # 密码
receiver = "283340479@qq.com"        # 接收人
# ----------2.编辑邮件的内容------
# 读文件
file_path = "result.html"
with open(file_path, "rb") as fp:
    mail_body = fp.read()
msg = MIMEMultipart()
msg["from"] = sender                        # 发件人
msg["to"] = receiver                        # 收件人
msg["subject"] = "这个我的主题"          # 主题
# 正文
body = MIMEText(mail_body, "html", "utf-8")
msg.attach(body)
# 附件
```

```
att = MIMEText(mail_body, "base64", "utf-8")
att["Content-Type"] = "application/octet-stream"
att["Content-Disposition"] = 'attachment; filename="test_report.html"'
msg.attach(att)
# ----------3.发送邮件------
try:
    smtp = smtplib.SMTP()
    smtp.connect(smtpserver)                    # 连服务器
    smtp.login(sender, psw)
except:
    smtp = smtplib.SMTP_SSL(smtpserver, port)
    smtp.login(sender, psw)                     # 登录
smtp.sendmail(sender, receiver, msg.as_string())  # 发送
smtp.quit()
```

5.最后结果，有图有真相



6.5.5 发给多个收件人

1.上面都是发给一个收件人，那么如何一次发给多个收件人呢？只需改两个小地方

2.把 receiver 参数改成 list 对象，单个多个都是可以收到的

3.msg["to"]这个参数不能用 list 了，得先把 receiver 参数转化成字符串，如下图所示

4.参考代码：

```
# coding:utf-8
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# ----------1.跟发件相关的参数------
smtpserver = "smtp.163.com"　　# 发件服务器
port = 0　　# 端口
sender = "yoyo@163.com"　　# 账号
psw = "*********"　　# 密码
# receiver = ["xxxx@qq.com"]　　# 单个接收人也可以是 list
receiver = ["xxxx@qq.com", "yoyo@qq.com"]　　# 多个收件人 list 对象

# ----------2.编辑邮件的内容------
# 读文件
file_path = "result.html"
with open(file_path, "rb") as fp:
    mail_body = fp.read()
msg = MIMEMultipart()
msg["from"] = sender　　# 发件人
msg["to"] = ";".join(receiver)　　# 多个收件人 list 转 str
msg["subject"] = "这个我的主题 999"　　# 主题
# 正文
body = MIMEText(mail_body, "html", "utf-8")
msg.attach(body)
# 附件
att = MIMEText(mail_body, "base64", "utf-8")
```

```
att["Content-Type"] = "application/octet-stream"
att["Content-Disposition"] = 'attachment; filename="test_report.html"'
msg.attach(att)

# ----------3.发送邮件------
try:
    smtp = smtplib.SMTP()
    smtp.connect(smtpserver)                          # 连服务器
    smtp.login(sender, psw)
except:
    smtp = smtplib.SMTP_SSL(smtpserver, port)
    smtp.login(sender, psw)                           # 登录
smtp.sendmail(sender, receiver, msg.as_string())  # 发送
smtp.quit()                                           # 关闭
```

6.5.6 邮件收不到的几种原因

1.Subject 和正文内容不要用 hello、hehe、test 等单词
2.from(发件人)和 to(收件人)不要为空，
（要不然会被认为是垃圾邮件）
3.找不到的话，先看下垃圾信箱，是不是跑到垃圾箱了
4.如果前几次可以收到，后来收不到了，需改下 subject 内容
（因为每次都是一个 subject，系统也会拒收的，把 subject 内容设置为动态的是最好的）
5.部分邮箱是 ssl 加密了的，所以无法发送，如:qq 邮箱
（用授权码去登录）
6.要是按照上面的步骤来报错了，说明**代码抄错了，多检查几次**。
(以上代码在 python2 和 python3 上都测试通过了）

第 7 章 项目实战：git+jenkins 持续集成

7.1 项目结构设计

前言

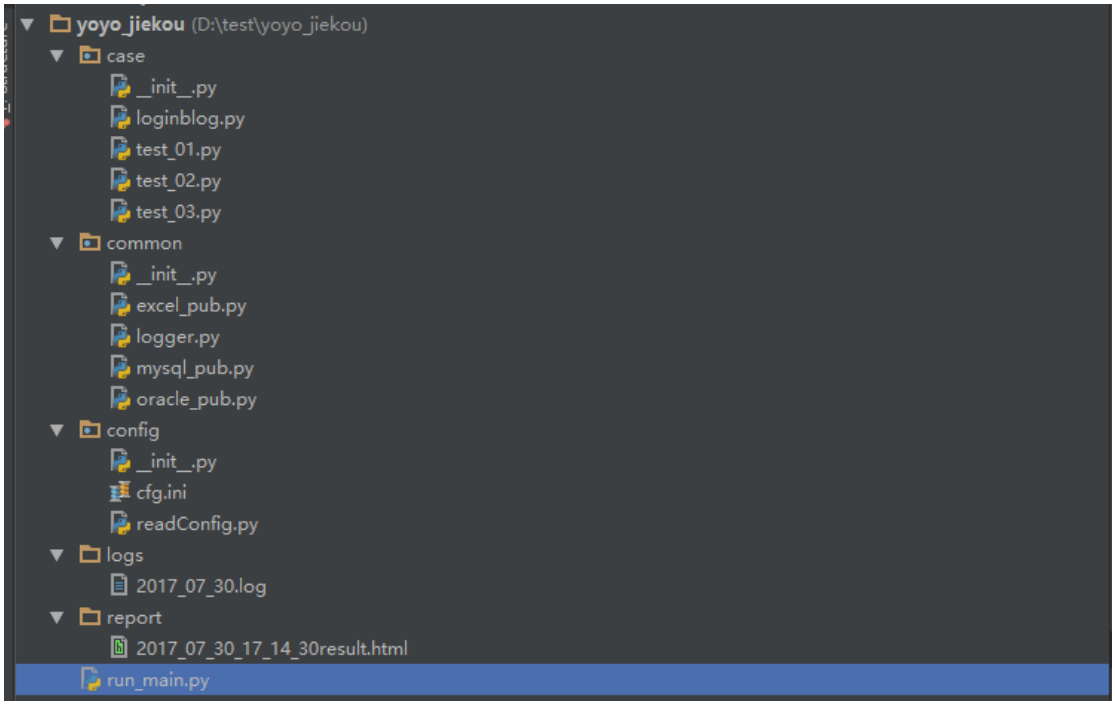
有很多小伙伴吵着要完整的项目源码，完整的项目属于公司内部的代码，这个是没法分享的,违法职业道德了,就算别人分享了,也只适用于本公司内部的业务。所以用例的代码还是得自己去一个个写，我只能分享项目框架代码，添砖加瓦的事情你们得自己亲自去做，只能帮你们到这了。

7.1.1 项目结构

1.新建一个工程（一定要创建工程），工程名称自己定义，如：yoyo_jiekou

2.在工程的根目录新建一个脚本：run_main.py,用来执行全部用例

3.在工程下创建以下几个 pakage:
--case:这个包放 test 开头的测试用例，也可以放一些封装接口的方法，如：loginblog(如果封装的接口比较多，也可以单独放一个包）
--common:这个包放一些公共的方法，如：读取 excel 文件方法，读取 mysql、oracle，logger.py 这个是封装日志的输入
--config:cfg.ini 这里是配置文件，如邮箱的一些参数：收件人，发件人，密码等,readConfig.py 用于读取配置文件
--logs:这里存放日志信息
--report:这里存放测试报告



7.1.2 run_main

第一步：用 discover 方法加载所有的测试用例

1. cur_path 这个参数是读取当前这个脚本的真实路径，也就是 run_main.py 的真实路径
2. caseName="case"这个 case 是存放测试用例的文件夹，如果没有的话，自动创建。如果想运行其它文件夹的用例，就改下 caseName 这个参数值
3. rule="test*.py"这个是匹配用例脚本名称的规则，默认匹配 test 开头的所有用例

```
# 当前脚本所在文件真实路径
cur_path = os.path.dirname(os.path.realpath(__file__))

def add_case(caseName="case", rule="test*.py"):
    '''第一步：加载所有的测试用例'''
    case_path = os.path.join(cur_path, caseName)  # 用例文件夹
    # 如果不存在这个case文件夹，就自动创建一个
    if not os.path.exists(case_path):os.mkdir(case_path)
    print("test case path:%s"%case_path)
    # 定义discover方法的参数
    discover = unittest.defaultTestLoader.discover(case_path,
                                                   pattern=rule,
                                                   top_level_dir=None)
    print(discover)
    return discover
```

第二步：生成 HTML 报告

1. 把上一步加载到用例的参数传入这个函数，测试报告的文件名称默认 report 文件夹：reportName="report

2. 如果没有这个 report 文件夹也没关系，可以自动创建的

```
def run_case(all_case, reportName="report"):
    '''第二步：执行所有的用例，并把结果写入HTML测试报告'''
    now = time.strftime("%Y_%m_%d_%H_%M_%S")
    report_path = os.path.join(cur_path, reportName)  # 用例文件夹
    # 如果不存在这个report文件夹，就自动创建一个
    if not os.path.exists(report_path):os.mkdir(report_path)
    report_abspath = os.path.join(report_path, now+"result.html")
    print("report path:%s"%report_abspath)
    fp = open(report_abspath, "wb")
    runner = HTMLTestRunner.HTMLTestRunner(stream=fp,
                                           title=u'自动化测试报告,测试结果如下：',
                                           description=u'用例执行情况：')

    # 调用add_case函数返回值
    runner.run(all_case)
    fp.close()
```

第三步：获取最新的测试报告

1. 如果第二步生成的测试报告加了时间戳，想找到最新的文件就用第三步

2.如果第二步不加时间戳，只是生成 result.html，那这一步其实没卵用，可以忽略
（个人觉得报告用一个名称 result.html 就行,新的自动覆盖旧的，**加时间戳虽然可以装逼，但是最后集成 jenkins 会不好弄**）

```
def get_report_file(report_path):
    '''第三步：获取最新的测试报告'''
    lists = os.listdir(report_path)
    lists.sort(key=lambda fn: os.path.getmtime(os.path.join(report_path, fn)))
    print (u'最新测试生成的报告：  '+lists[-1])
    # 找到最新生成的报告文件
    report_file = os.path.join(report_path, lists[-1])
    return report_file
```

第四步：发送测试报告到邮箱

1.像 QQ 邮箱这种 ssl 加密的就走 SMTP_SSL，用授权码登录

2.其它邮箱就正常账号密码登录，走 SMTP

```
def send_mail(sender, psw, receiver, smtpserver, report_file, port):
    '''第四步：发送最新的测试报告内容'''
    with open(report_file, "rb") as f:
        mail_body = f.read()
    # 定义邮件内容
    msg = MIMEMultipart()
    body = MIMEText(mail_body, _subtype='html', _charset='utf-8')
    msg['Subject'] = u"自动化测试报告"
    msg["from"] = sender
    msg["to"] = psw
    msg.attach(body)
    # 添加附件
    att = MIMEText(open(report_file, "rb").read(), "base64", "utf-8")
    att["Content-Type"] = "application/octet-stream"
    att["Content-Disposition"] = 'attachment; filename= "report.html"'
    msg.attach(att)
    try:
        smtp = smtplib.SMTP_SSL(smtpserver, port)
    except:
        smtp = smtplib.SMTP()
        smtp.connect(smtpserver, port)
    # 用户名密码
    smtp.login(sender, psw)
    smtp.sendmail(sender, receiver, msg.as_string())
    smtp.quit()
    print('test report email has send out !')
```

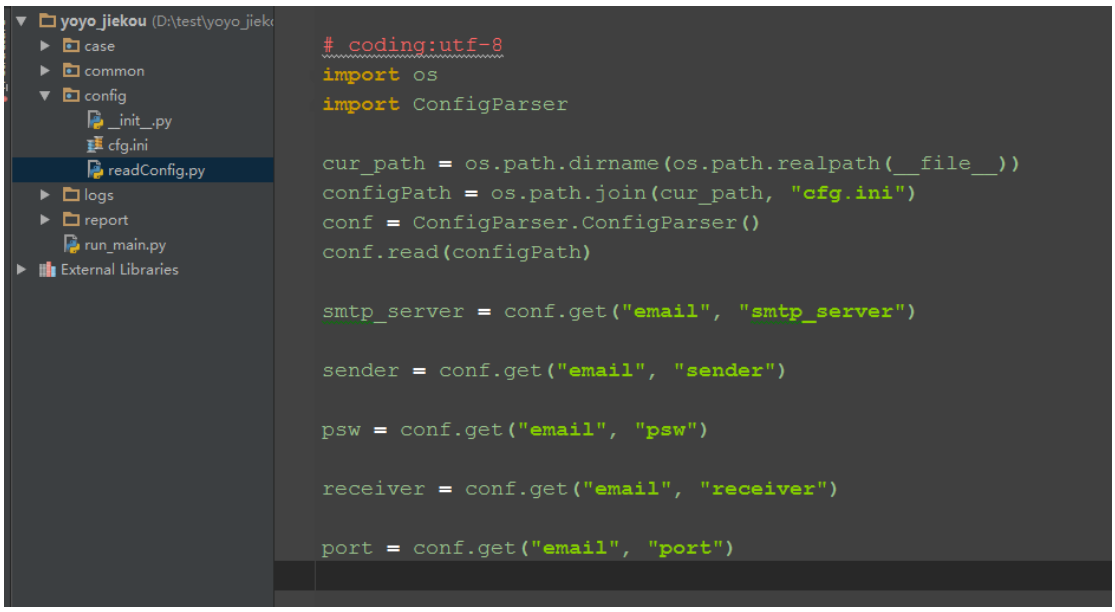
最后执行代码
1.这里邮箱的内容读的配置文件

```python
if __name__ == "__main__":
    all_case = add_case()    # 1加载用例
    # 生成测试报告的路径
    run_case(all_case)         # 2执行用例
    # 获取最新的测试报告文件
    report_path = os.path.join(cur_path, "report")   # 用例文件夹
    report_file = get_report_file(report_path)   # 3获取最新的测试报告
    #邮箱配置
    from config import readConfig
    sender = readConfig.sender
    psw = readConfig.psw
    smtp_server = readConfig.smtp_server
    port = readConfig.port
    receiver = readConfig.receiver
    send_mail(sender, psw, receiver, smtp_server, report_file, port)   # 4最后一步发送报告
```

7.1.3 config 配置

1. cfg. ini 打开，这里写配置文件内容

2. 用 readConfig. py 读取配置文件

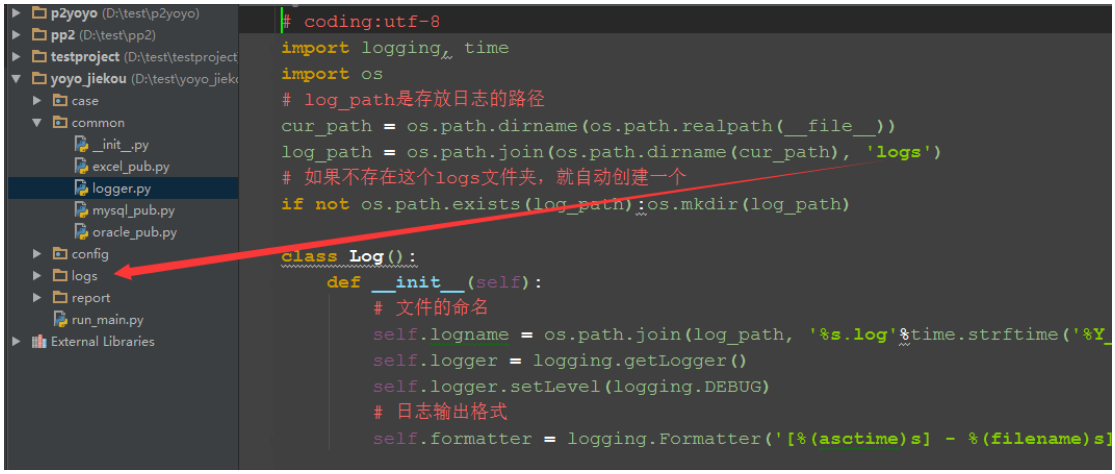


3. 读取的内容就是传入第二步操作里面需要调用邮箱的配置信息

7.1.4 logger

1. logger. py 这个文件放到 common 目录下，封装日志文件的读取

2. 日志保存到 logs 文件夹



3. 封装代码

```
# coding:utf-8
import logging, time
import os
# log_path 是存放日志的路径
cur_path = os.path.dirname(os.path.realpath(__file__))
```

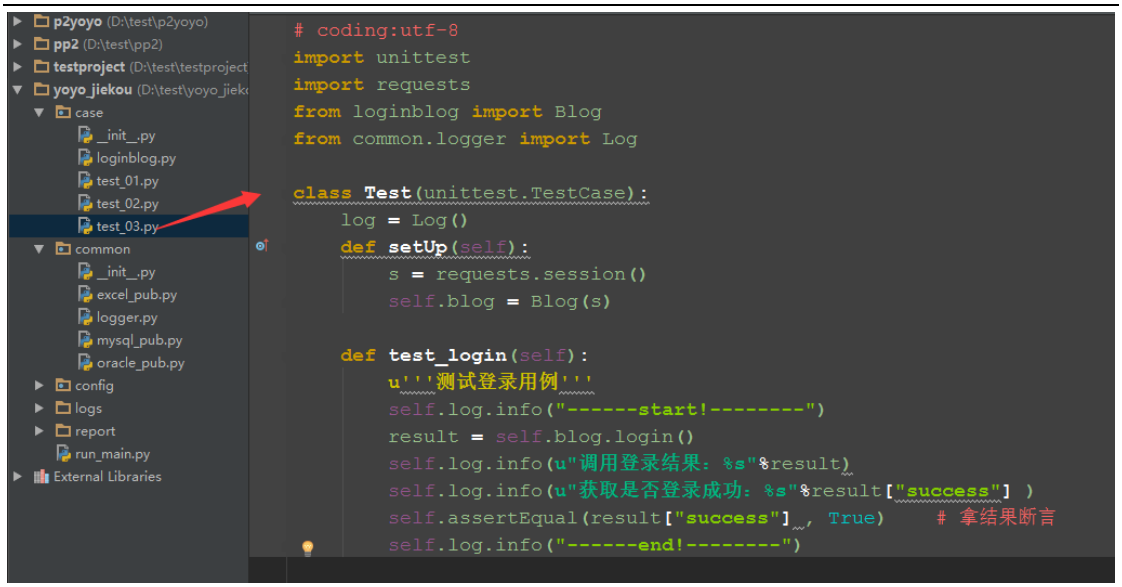
```
log_path = os.path.join(os.path.dirname(cur_path), 'logs')
# 如果不存在这个 logs 文件夹，就自动创建一个
if not os.path.exists(log_path):os.mkdir(log_path)
class Log():
    def __init__(self):
        # 文件的命名
        self.logname             =             os.path.join(log_path,
'%s.log'%time.strftime('%Y_%m_%d'))
        self.logger = logging.getLogger()
        self.logger.setLevel(logging.DEBUG)
        # 日志输出格式
        self.formatter       =       logging.Formatter('[%(asctime)s]
- %(filename)s] - %(levelname)s: %(message)s')
    def __console(self, level, message):
        # 创建一个 FileHandler，用于写到本地
        fh = logging.FileHandler(self.logname, 'a')  # 追加模式  这个
是 python2 的
        # fh = logging.FileHandler(self.logname, 'a', encoding='utf-8')
# 这个是 python3 的
        fh.setLevel(logging.DEBUG)
        fh.setFormatter(self.formatter)
        self.logger.addHandler(fh)
        # 创建一个 StreamHandler,用于输出到控制台
        ch = logging.StreamHandler()
        ch.setLevel(logging.DEBUG)
        ch.setFormatter(self.formatter)
        self.logger.addHandler(ch)
        if level == 'info':
            self.logger.info(message)
        elif level == 'debug':
            self.logger.debug(message)
        elif level == 'warning':
            self.logger.warning(message)
        elif level == 'error':
            self.logger.error(message)
        # 这两行代码是为了避免日志输出重复问题
        self.logger.removeHandler(ch)
        self.logger.removeHandler(fh)
        # 关闭打开的文件
        fh.close()
    def debug(self, message):
        self.__console('debug', message)
    def info(self, message):
```

```
        self.__console('info', message)
    def warning(self, message):
        self.__console('warning', message)
    def error(self, message):
        self.__console('error', message)
if __name__ == "__main__":
    log = Log()
    log.info("---测试开始----")
    log.info("操作步骤 1,2,3")
    log.warning("----测试结束----")
```

7.1.5 case 放用例

1.常用的接口，需要检查被调用的单独封装处理，如登录等

2.test 开头的用例



7.1.6 logs 日志查看

1. 运行完后日志都会收集到 logs 文件夹以日期命名

7.1.7 生成测试报告

1. 这里调用的是 HTMLTestRunner 生成 html 的测试报告

自动化测试报告,测试结果如下：

Start Time: 2017-07-30 22:03:44
Duration: 0:00:03.268000
Status: Pass 3 Failure 3

用例执行情况：

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
test_01.Blog_login	2	1	1	0	Detail
test_login1: 测试登录：正确账号，正确密码	fail [x] ft1.1: [2017-07-30 22:03:44,818] - logger.py] - INFO: -------登录成功用例：start!---------- [2017-07-30 22:03:44,818] - logger.py] - INFO: 输入正确账号：DvF1fv20vIRap1e29Bb3dv0yRvLZSODu5qjXfp5XSBTPLJFg5z [2017-07-30 22:03:44,819] - logger.py] - INFO: 输入正确密码：BaXSR4sgyDCLxCcJLtVgZbkyJd8EAxLBZYQhQ/vjedSeik/zAJ [2017-07-30 22:03:45,155] - logger.py] - INFO: 博客园登录结果：{"success":false} [2017-07-30 22:03:45,157] - logger.py] - INFO: 获取测试结果：{u'success' : False} Traceback (most recent call last): File "D:\test\yoyo_jiekou\case\test_01.py", line 41, in test_login1 self.assertEqual(result["success"], True) AssertionError: False != True				
test_02.Test_Kuaidi: 测试快递查询接口	2	0	2	0	Detail
test_tiantian: 测试天天快递，单号：560697415000	fail				
test_yunda: 测试韵达快递，单号：1202247993797	fail				
test_03.Test	2	2	0	0	Detail
Total	**6**	**3**	**3**	**0**	

7.1.8 发送报告到邮箱

1. 运行 run_main.py 就会自动生成报告，然后发送到邮箱了
（这里 QQ 邮箱的展示是有问题的，一般用企业邮箱的话，是可以正常展示的）

7.2 项目源码下载

源码打包放到百度云盘了

（目前代码是在 2.7 上跑的通的，3.6 的暂时没试过，问题也不大）

由于有时候下载地址会失效，所以这里就不贴在 pdf 了，去百度阅读上找到这本书的 7.2 章节，里面有下载地址
https://yuedu.baidu.com/ebook/585ab168302b3169a45177232f60ddccda38e695

（7.2 章节有下载地址）

7.3 使用 git 上传到 github

前言

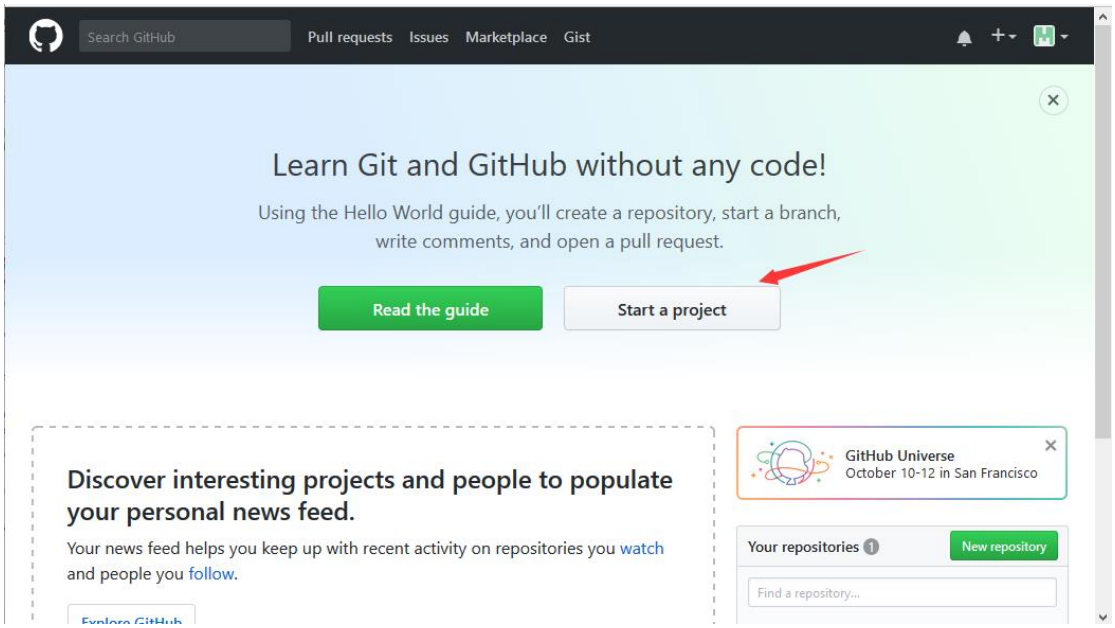
不会使用 github 都不好意思说自己是码农，github 作为一个开源的代码仓库管理平台，对代码的管理确实很方面。

问题来了：为什么越来越多的小伙伴喜欢开源？答：因为我们可以免费下载到大神的源码啊！

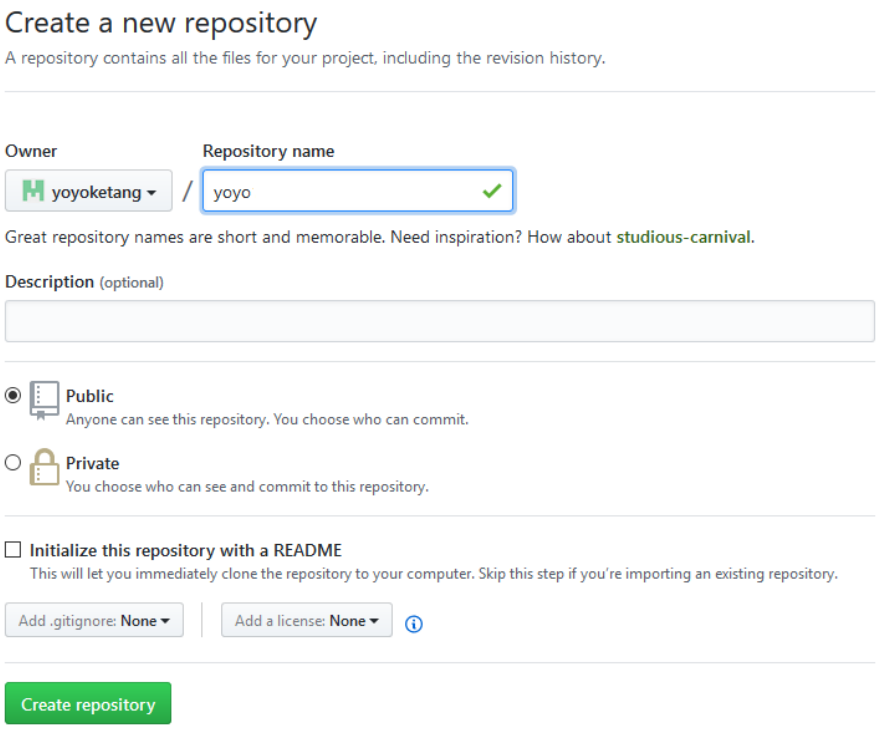
7.3.1 注册 Github 账号

1.先注册一个账号，注册地址：https://github.com/

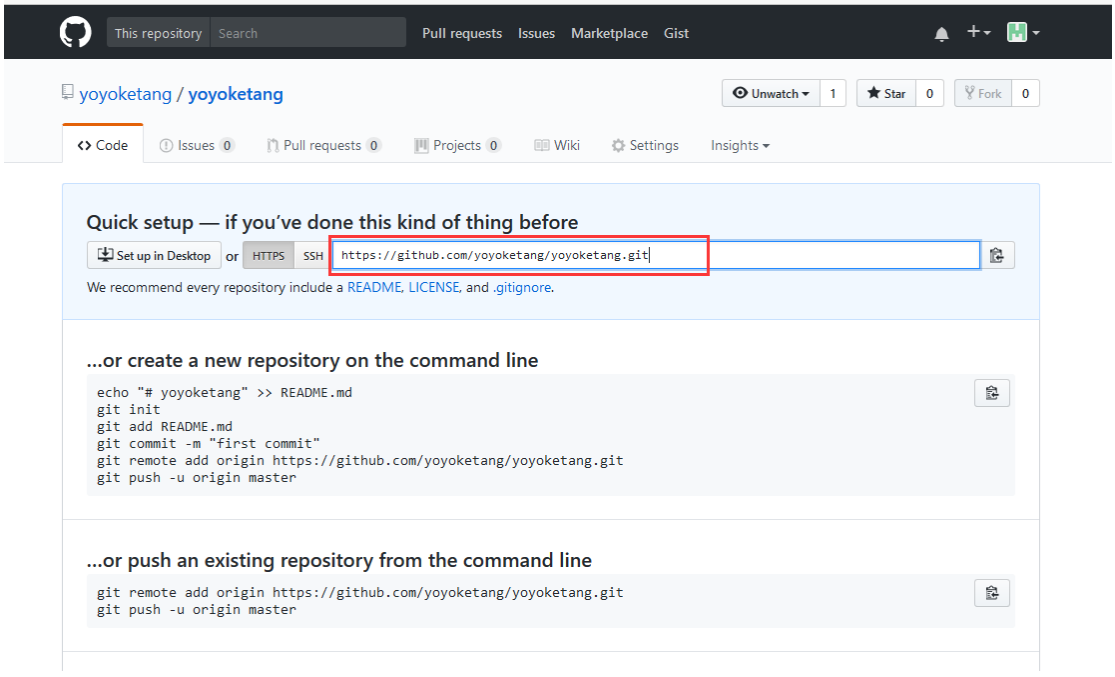
2.登录后，开始你的旅程



3.创建一个 repository name,输入框随便取一个名字（不要用中文!!!）

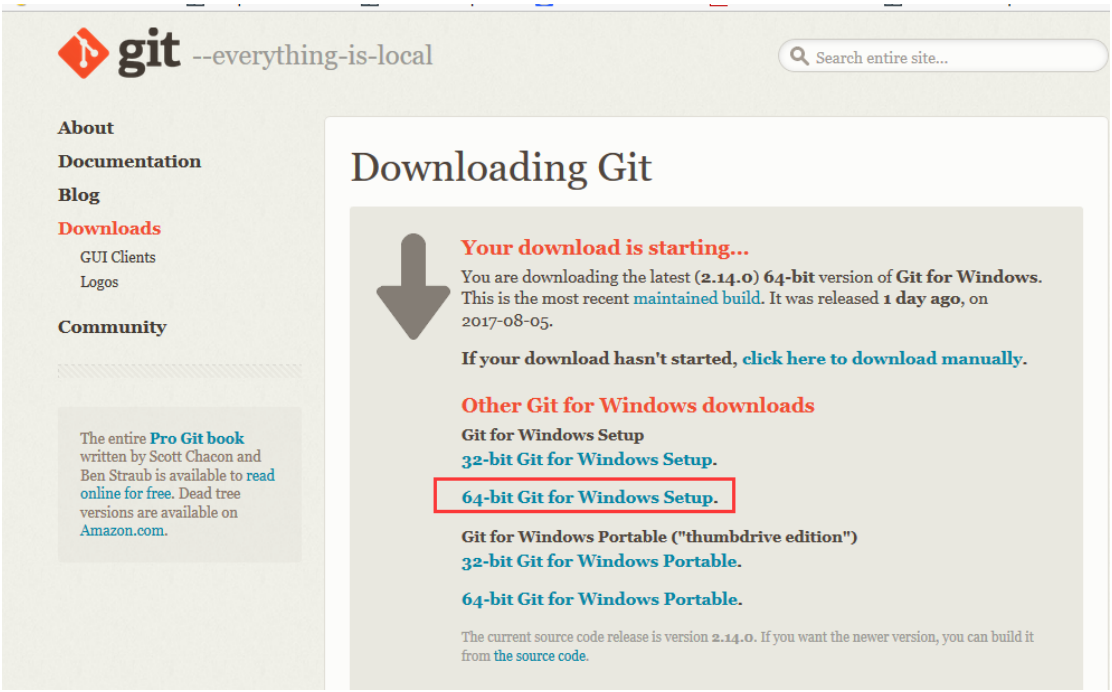


4.点 Create repository 创建成功，界面如下

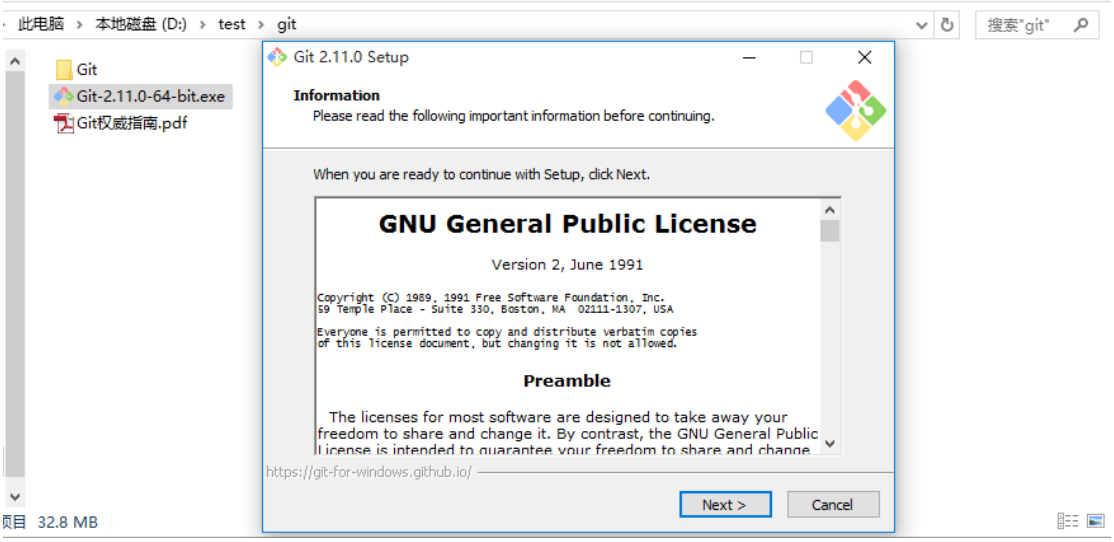


7.3.2 安装 git

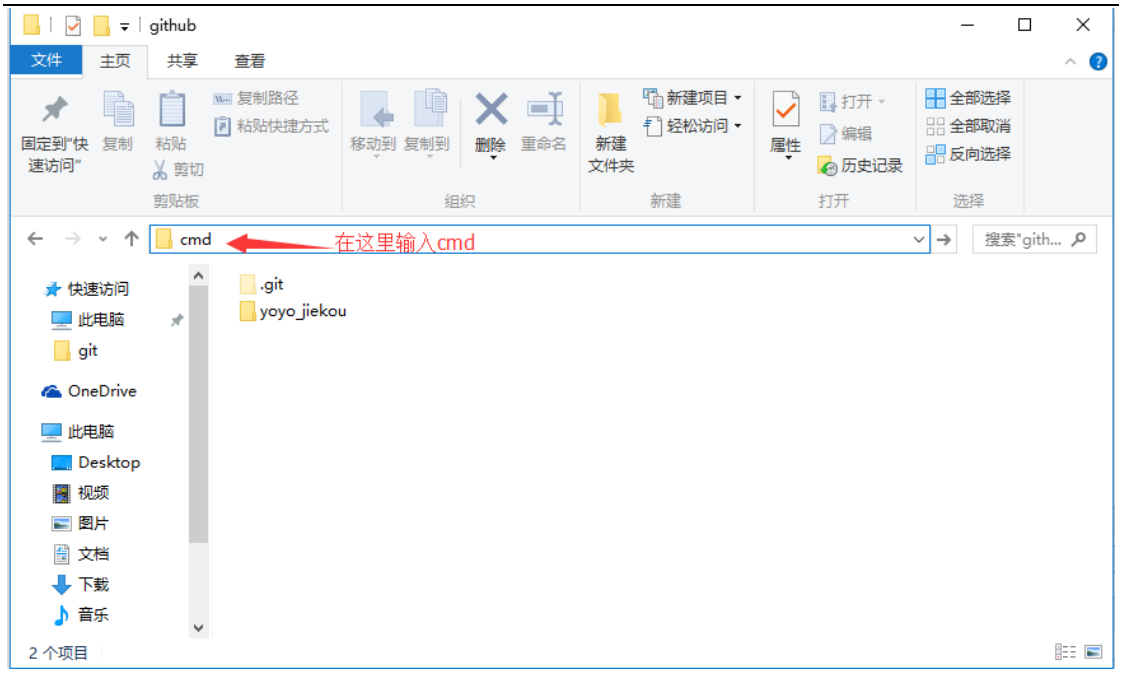
1.git 是本地客户端管理代码的一个工具，下载地址：https://git-scm.com/download/win



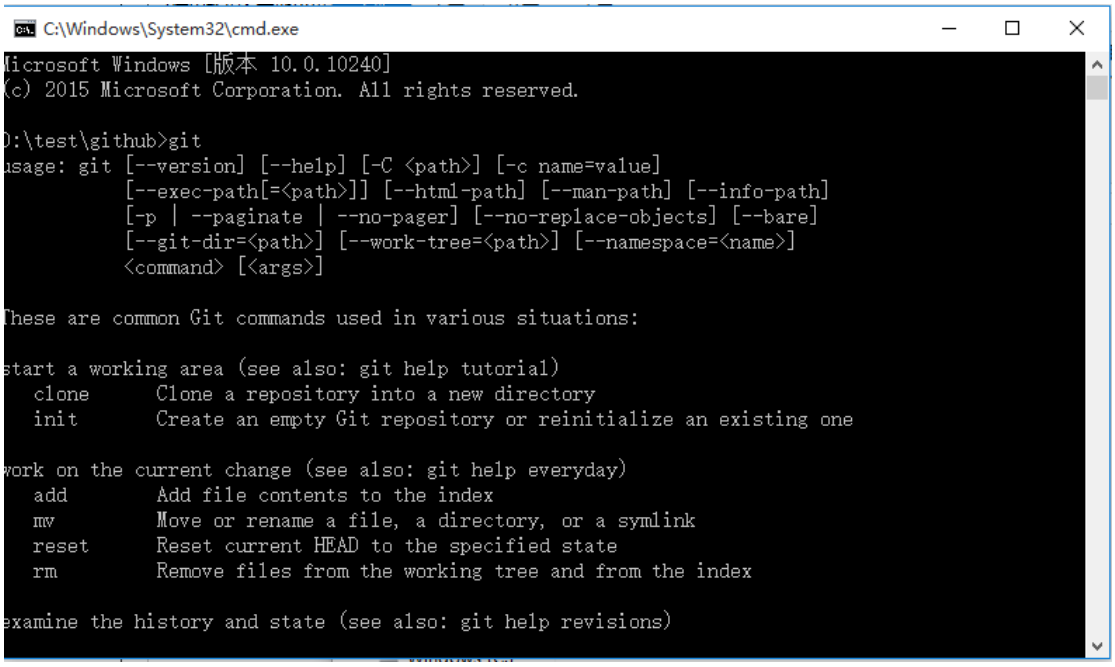
2.下载到本地电脑后，一路傻瓜式安装 Next->Next->-Next



3.安装完成后，本地随便建个文件夹，如 github，在文件地址栏输入 cmd



4.打开 dos 界面后输入 git,看到如下界面说明安装成功



7.3.3 本地仓库

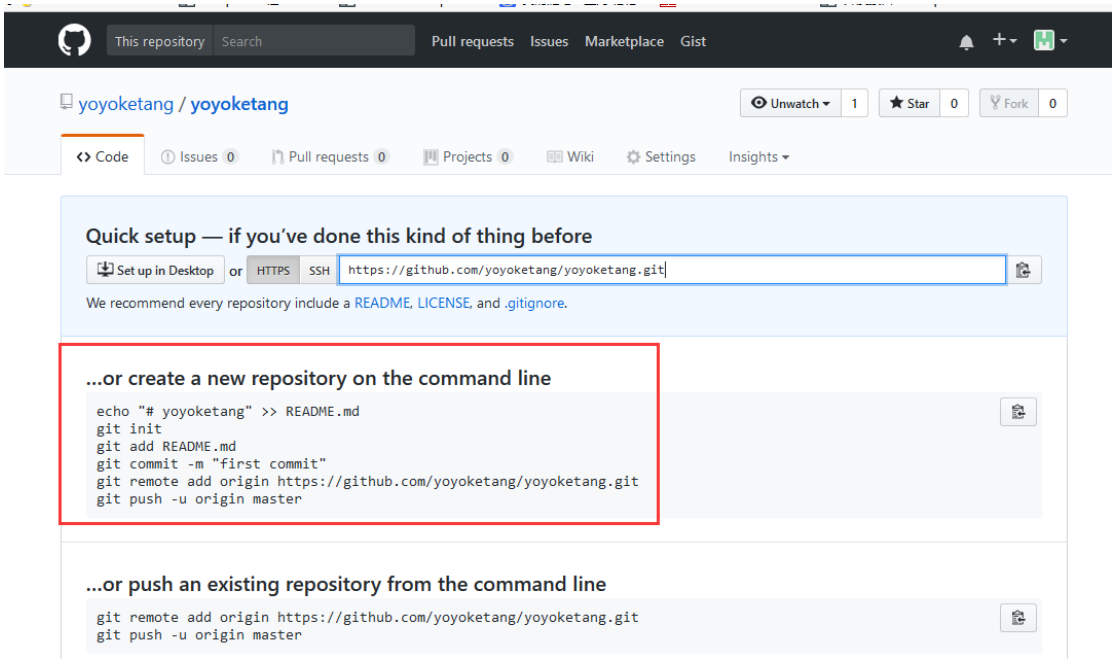
1.回到之前的 github 界面，下面有几个指令告诉我们如何把本地代码上传：

git init

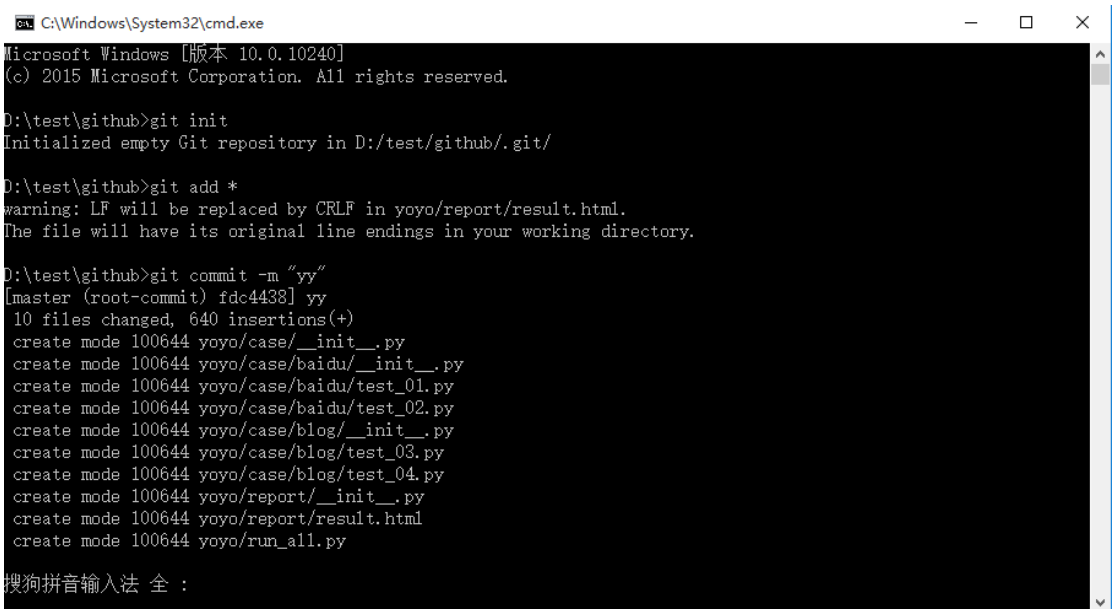
git add README.md

git commit -m "first commit"

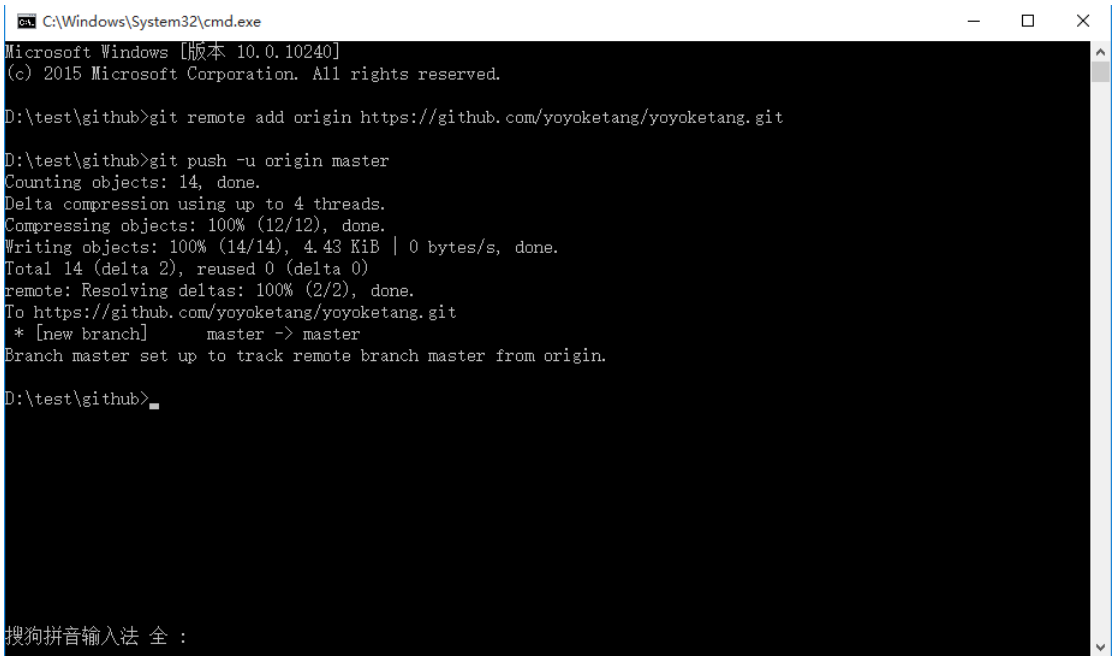
git remote add origin https://github.com/yoyoketang/yoyoketang.git
git push -u origin master



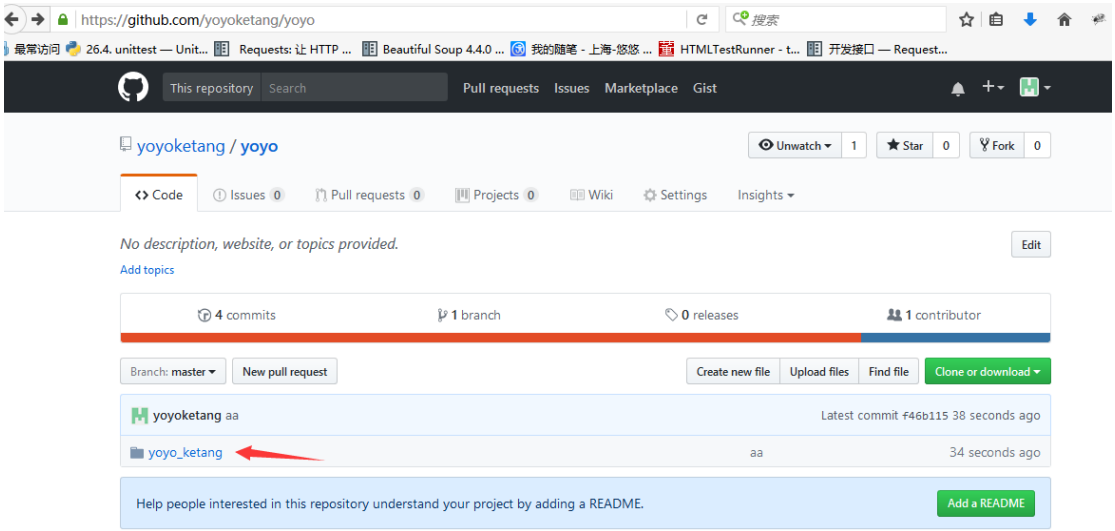
2.在上面创建的 github 文件目录放需要上传的代码，然后安装上面的步骤开始操作
第一步：git init --建仓
第二步：git add　* --添加代码到本地仓库（*是代码添加全部更新的）
第三步：git commit -m "first commit"　--提交到本地缓存（“引号里面是说明提交了什么东西”）



第四步：git remote add origin https://github.com/yoyoketang/yoyoketang.git　--提交到远程 github 上（后面的地址，就是之前配置的 repository 地址）
第五步：git push -u origin master　--push 到 master 分支



3.代码上传成功后，如图所示



7.3.4 遇到问题与解决方案

注意：初次使用的话，在输入上面指令过程中会遇到以下几个问题：

1.要是 cmd 窗口看到提示以下这两个信息

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com
解决办法：按上面的提升，cmd 窗口接着输入
>git config --global user.name "这里是你的 github 用户名"
>git config --global user.email xxx@xxx.com(你的邮箱)
2.提交到远程时候，提示：
fatal: remote origin already exists.
解决办法：删除远程 git 仓库
>git remote rm origin
3.首次操作过程中需要登录就按提示输入账号名和密码

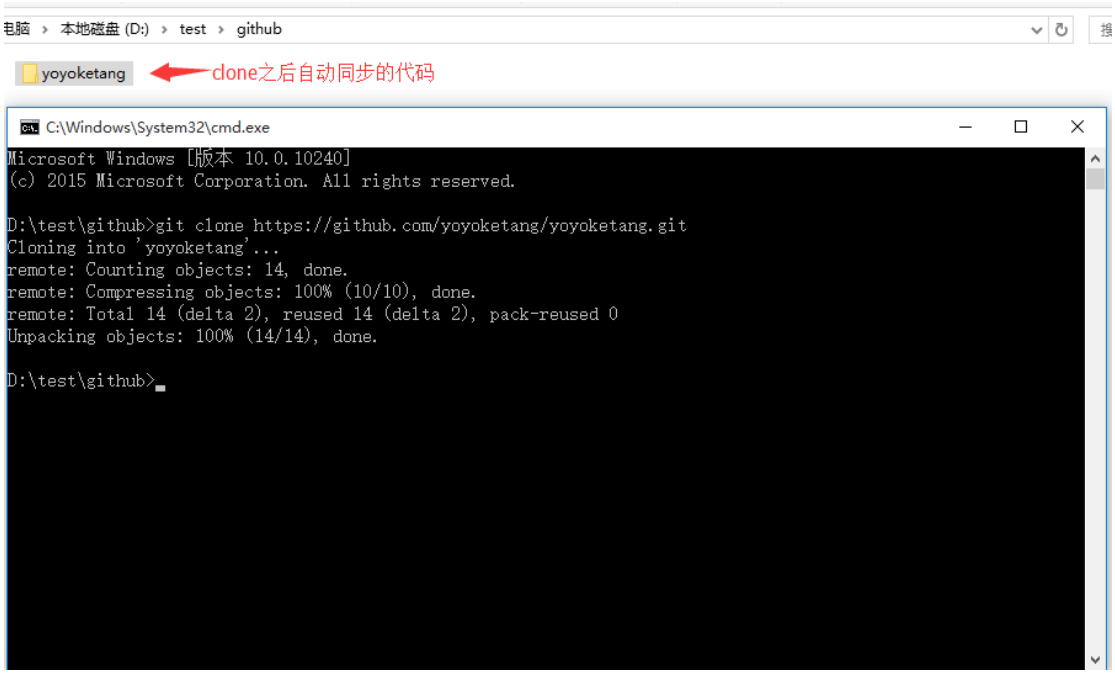
7.4 更新 github 上代码

前言
前面一篇已经实现首次上传代码到 github 了，迈出了装逼第一步，本篇继续讲如何把本地更新的代码同步更新到 github 上

7.4.1 clone 代码

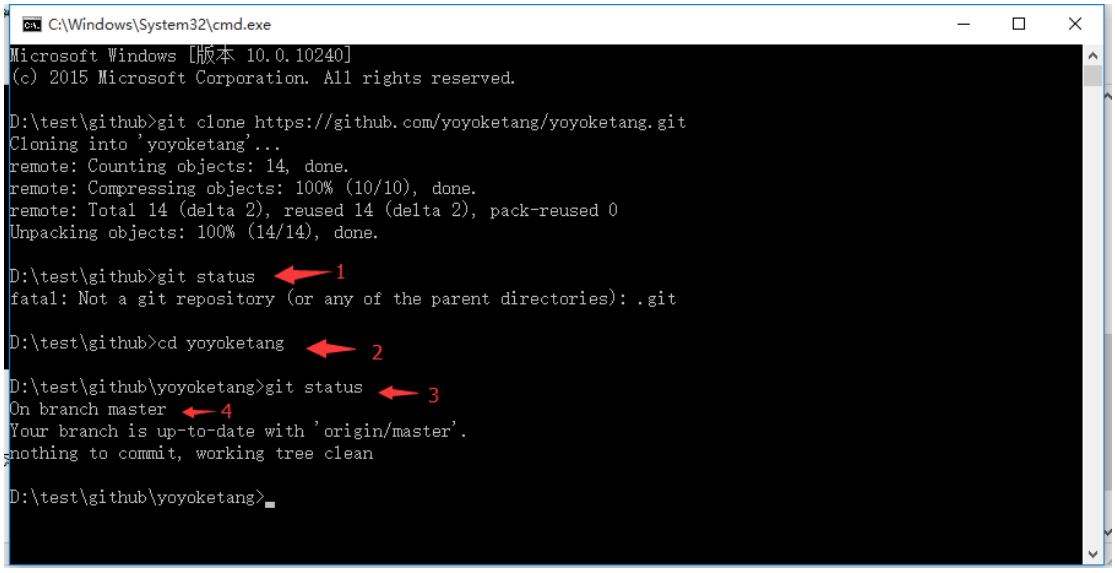
1.把大神的代码 clone 到本地，或者 clone 自己 github 上的代码，使用指令：
>git clone https://github.com/yoyoketang/yoyoketang.git

2.在本地随便建个文件夹，地址栏输入 cmd 打开，然后输入上面指令



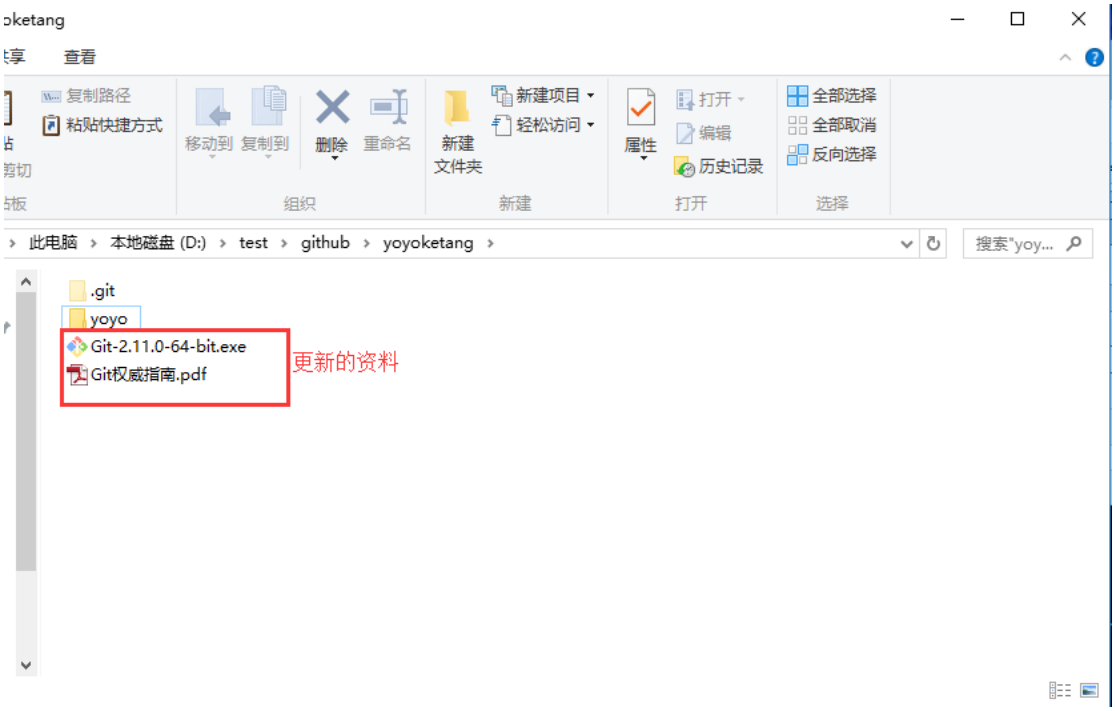
7.4.2 git status 查看状态

1.查看当前的 git 仓库状态，可以使用 git status
>git status
2.如果是在刚才新建的文件夹输入 git status 会出现下图 1 所示，git 目录不对
3.先用 cd 命令切换到 yoyoketang 这个 repository 目录
4.再输入 git status 可以看到 On branch master，这个说明已经在 master 分支上了



7.4.3 更新代码

1.在 yoyoketang 文件夹下更新东西，比如我上传 2 个资料文件

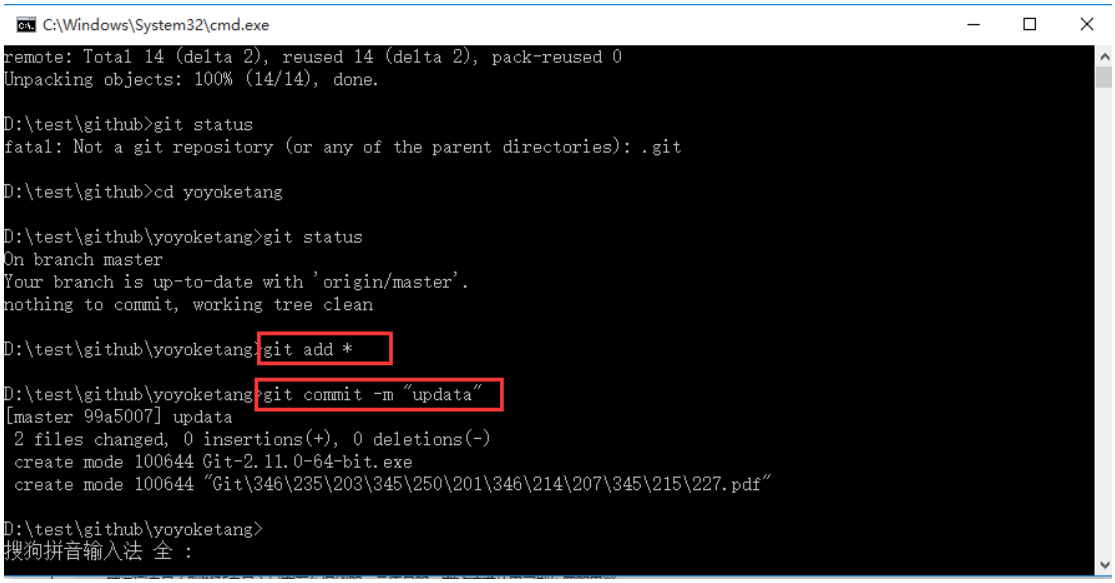


2.更新后使用 git add * (*是更新全部）

>git add *

3.接着输入 git commit -m "更新说明“，commit 只是提交到缓存区域

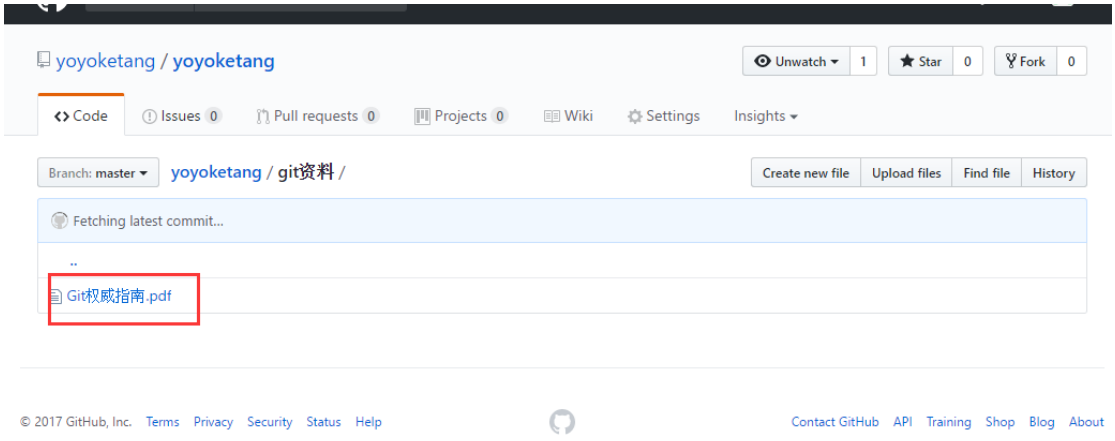
>git commit -m "更新说明 “



4.如果是多人同时开发维护代码，得先 git pull ,拉取当前分支最新代码
>git pull
5.最后 git push origin master,最后一步才是 push 到远程的 master 分支上
(最好不要上传太大文件，要不然太慢了)

```
D:\test\github\yoyoketang>git commit -m "aaa"
[master 7b31d1b] aaa
 2 files changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 Git-2.11.0-64-bit.exe
 rename "Git\346\235\203\345\250\201\346\214\207\345\215\227.pdf" => "git\350\265\204\346\226\231/Git
\346\235\203\345\250\201\346\214\207\345\215\227.pdf" (100%)

D:\test\github\yoyoketang>git pull
Already up-to-date.

D:\test\github\yoyoketang>git push origin master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 81.20 MiB | 261.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/yoyoketang/yoyoketang.git
   fdc4438..7b31d1b  master -> master

D:\test\github\yoyoketang>
```

6.打开 github 界面就能看到同步了



7.5 jenkins 搭建（附赠对应视频）

前言

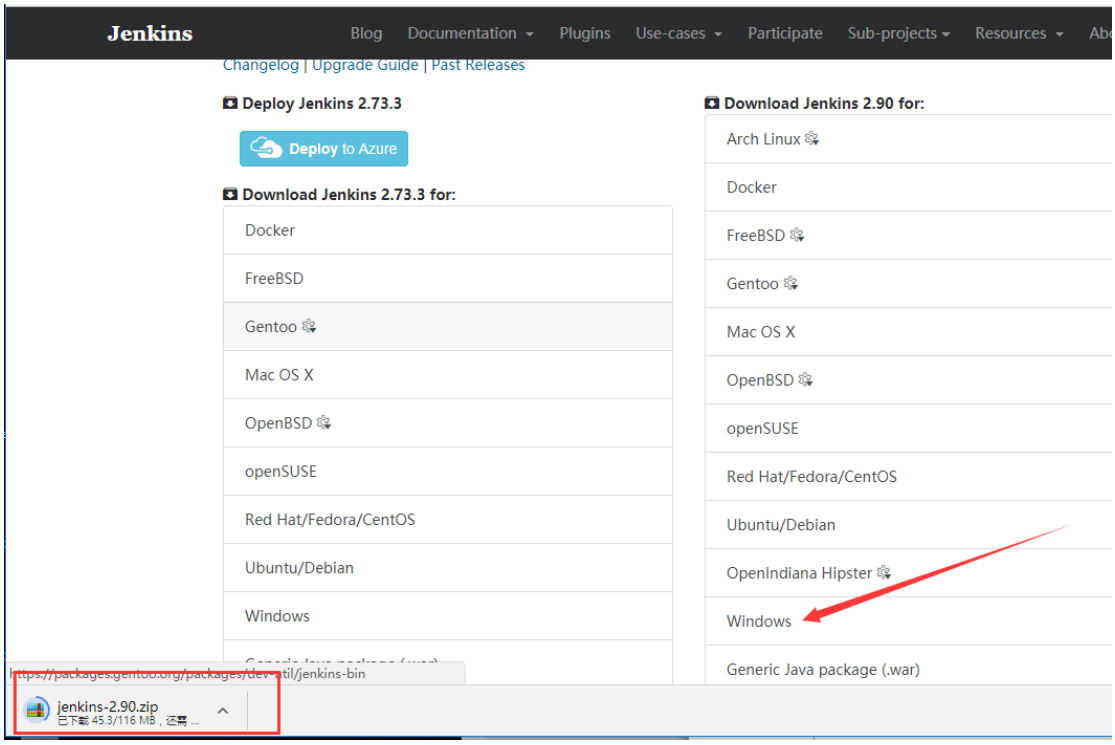
脚本写完之后，一般是集成到 jenkins 环境了，方便一键执行。

Jenkins 的环境安装分两种：

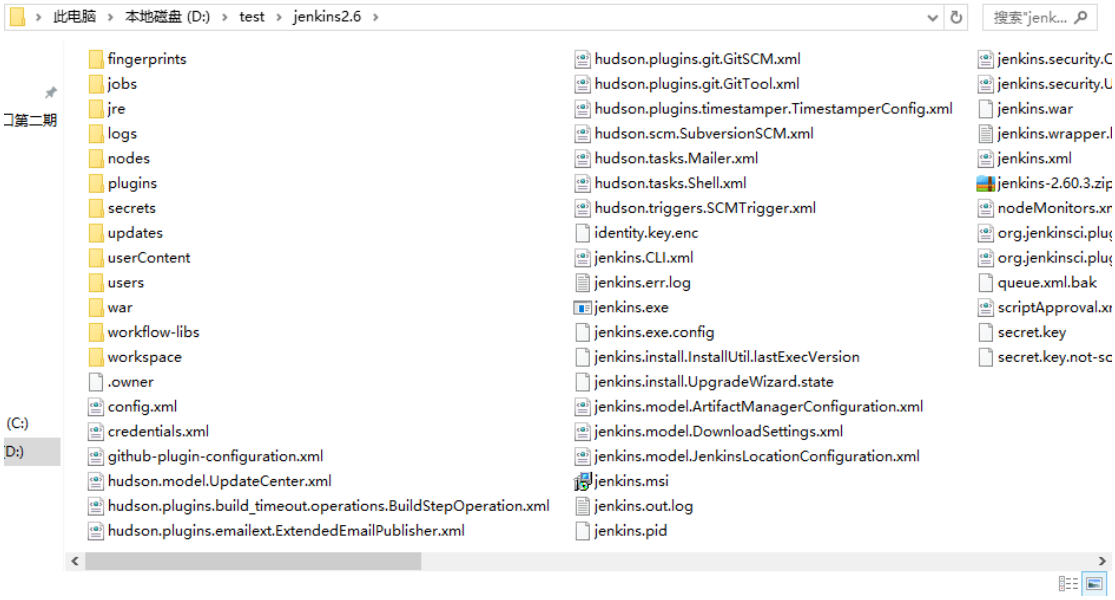
7.5.1 jenkins.exe 安装

第一种：使用 jenkins.exe 的客户端，直接解压傻瓜式安装
（新手推荐第一种）

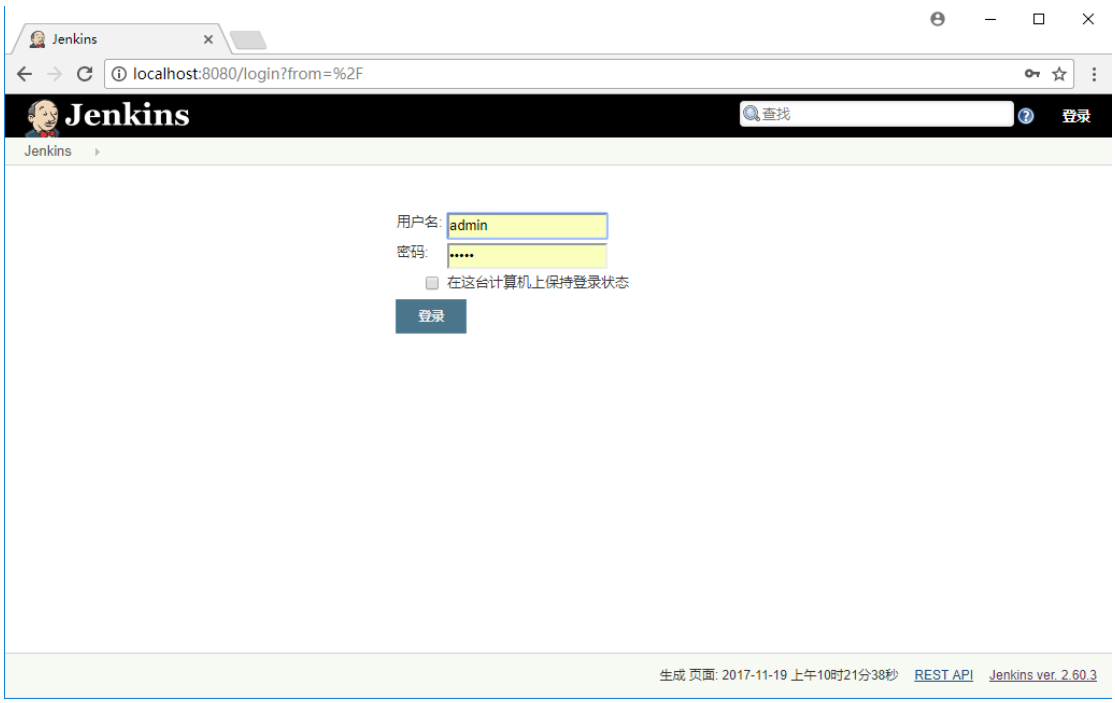
1.下载地址：https://jenkins.io/download/



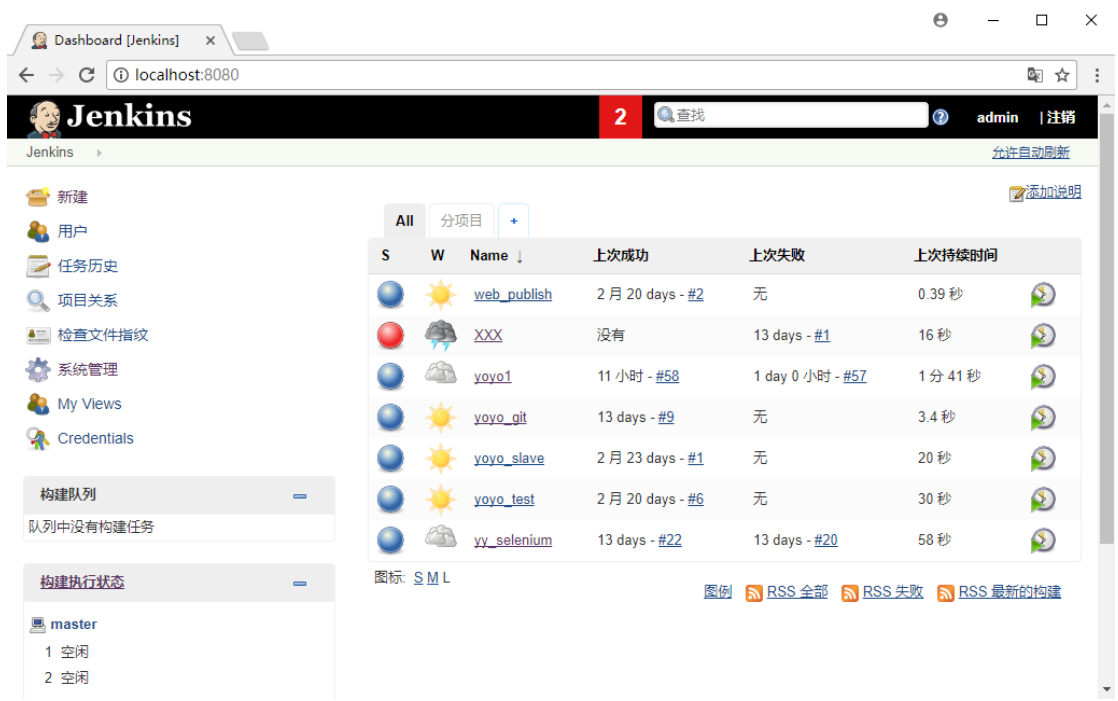
2.下载之后解压出来双击，就能安装好了



3.然后浏览器输入：http://localhost:8080/　即可访问 jenkins 界面



4.用你自己设置的账号密码就能登录了



7.5.2 tomcat+Jenkins.war

第二种：tomcat+Jenkins.war 环境部署包，这种比较复杂

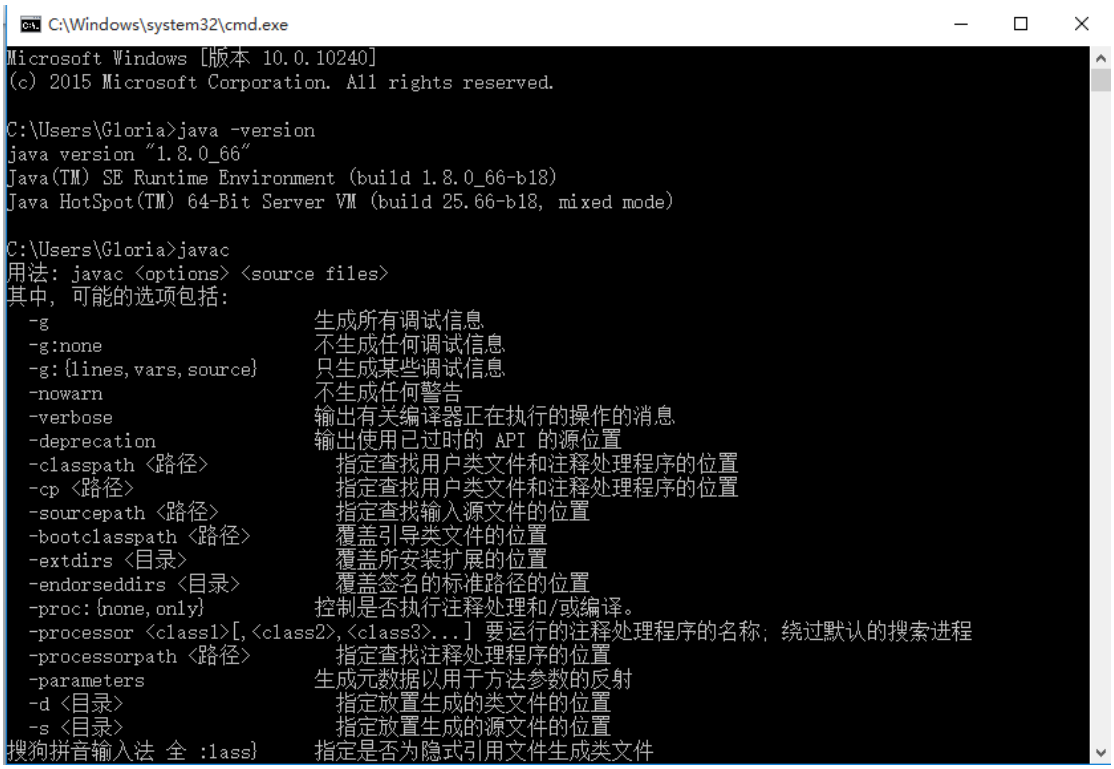
（不会搭建的送一份搭建环境 tomcat+jenkins 的视频：链接 http://pan.baidu.com/s/1qXAHwg0 密码：juy7）

一、环境准备
小编环境：
1.win10 64 位
2.JDK 1.8.0_66
3.tomcat 9.0.0.M4
4.jenkins 2.0

二、安装 JDK
1.安装 JDK 教程很多，这里就不讲了，可以参考这个链接的教程：JDK 安装配置教程_百度经验
http://www.baidu.com/link?url=QeB5aSWWcis5KOpboVcdyUf2CFArSSF7taz4dfHyV4__VJPI3Dgg18x-bYyngqJx1FfH3W79pvDu08fiHwYyvE4z024h7kIzLyCcDC1_sca&wd=&eqid=b54950a6000003d50000000459166d88

在系统环境变量里添加以下变量：
①JAVA_HOME：jdk 的安装路径 C:\Sun\Java\jdk

②CLASSPATH：“.;%JAVA_HOME%\lib;”
③Path：“;%JAVA_HOME%\bin”
2.检查环境是否成功（这一步很重要）
>>java -version
>>javac
(输入 javac 一定要看到中文的这些内容)



二、tomcat 环境
1.打开百度网页，在百度搜索栏里输入关键子：tomcat，在查询结果里点击 tomcat 官网进行下载。(下软件一定要在官网下载)

2.安装过程，网上也有现成的：tomcat 怎样下载安装如何配置环境变量_百度经验
http://www.baidu.com/link?url=Xvw-9ySg2yfyCt_VEcOimHs5gzJ7a2iSX4iFWaEpAaJf-yqVk4m59JVxcZz27Gb3LR8H0K2FQH9qFm1dLYkU0_lMekEFLDEEnajGgzZlyqW

3.tomcat 环境变量
在系统变量中添加以下变量 windows 7 系统安装与配置 Tomcat 服务器环境_百度经验
http://www.baidu.com/link?url=szTb9ln3LomTiovgYlXpH1mInwQesARFSgiW4sTRCadAcozdjrTdV7rCsQUMYLINIABvH4KA8wEuvr6ezvonjjq5IVDN6wNBSJHFRXdVUH3

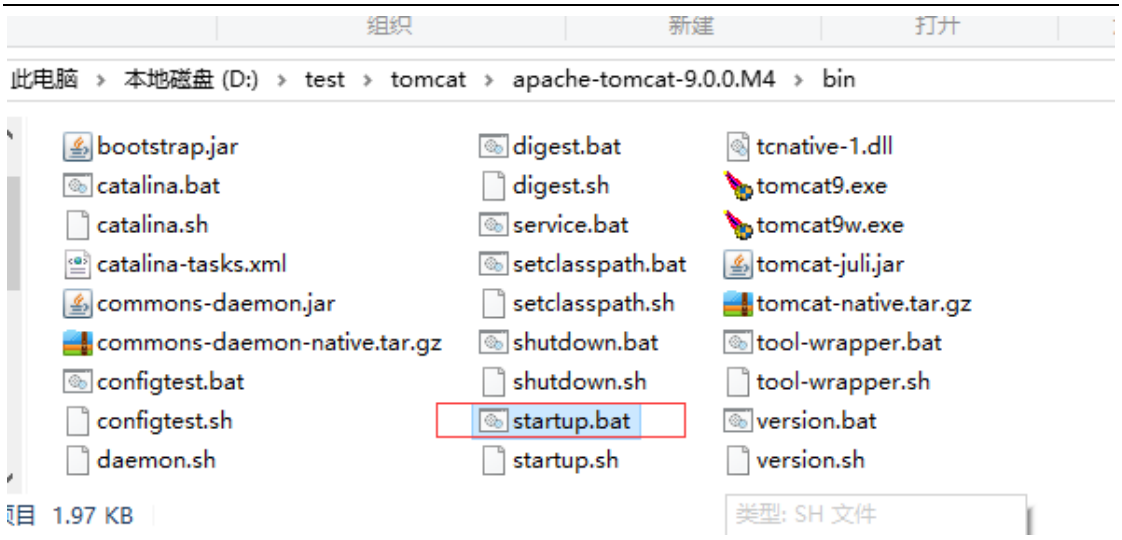
1）新建 TOMCAT_HOME 变量
变量名 TOMCAT_HOME
变量值 d:\tomcat
变量值即为我们下载的 tomcat 解压路径，在这里我的是 d:\tomcat

2)新建 CATALINA_HOME 变量
变量名 CATALINA_HOME
变量值 c:\tomcat
没错，CATALINA_HOME 的变量值与 TOMCAT_HOME 的变量值是一样的。

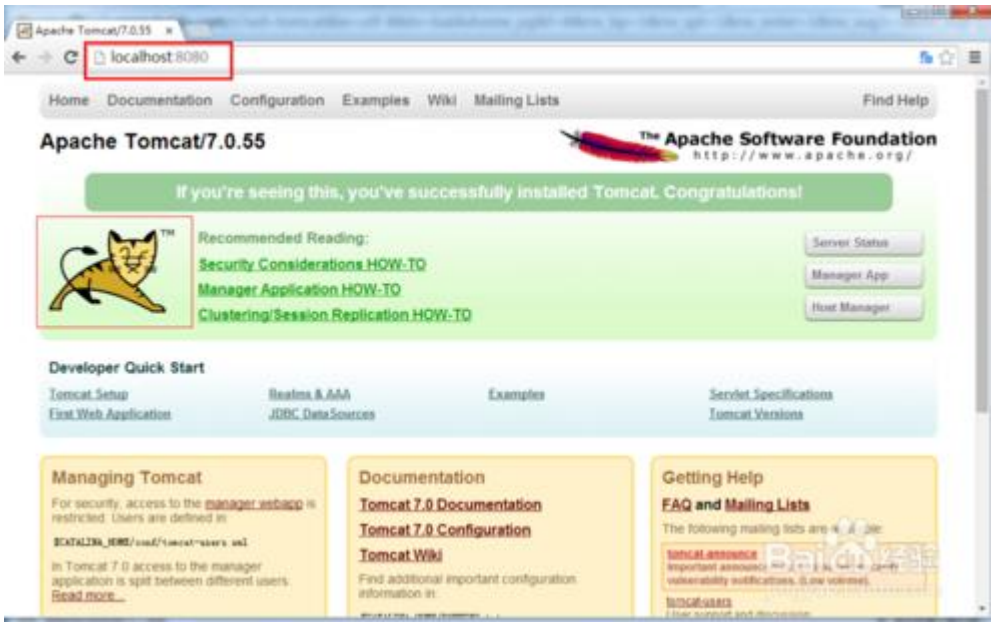
3）修改变量 Path
在系统变量中找到 Path 变量名，双击或点击编辑，在末尾添加如下内容
;%CATALINA_HOME%\bin;%CATALINA_HOME%\lib
这里要注意，各个变量值之间一定要用;分隔。

三、启动 tomcat
1.启动 Tomcat 服务器
在 cmd 命令窗口下输入 startup.bat 回车，运行如下图所示

（装了环境变量就不用切换到目录了）
2.双击启动运行



3.Tomcat 成功启动后，启动浏览器，在浏览器的地址栏里输入：http://localhost:8080/。然后回车。进入如下画面。此时表示 Tomcat 已经正确安装



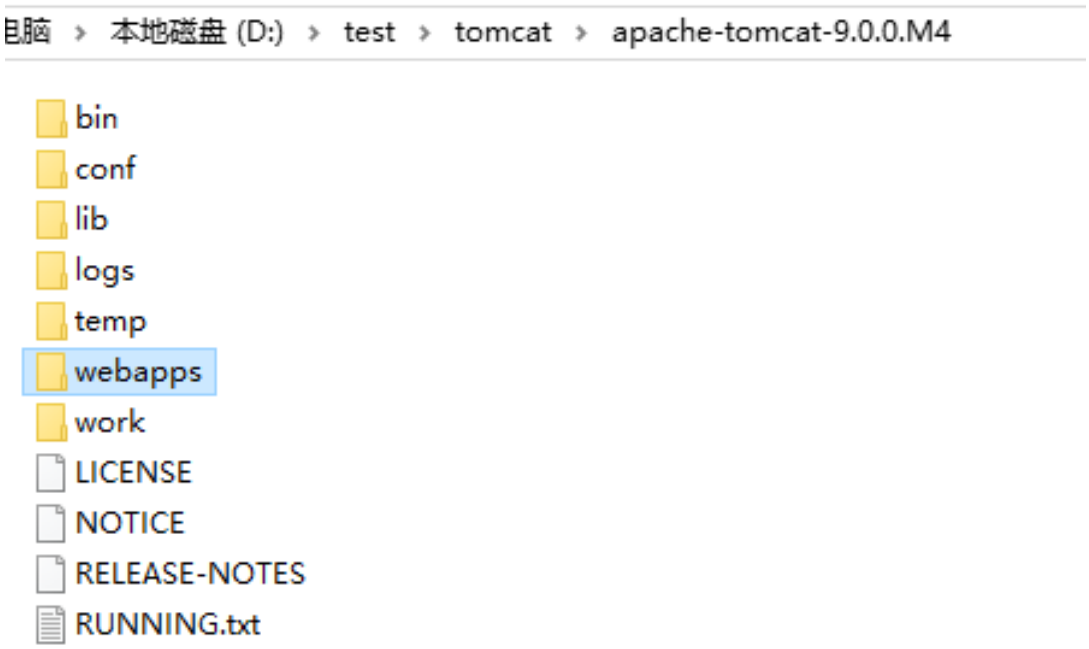
四、jenins 环境

1.Jenkins 包分两种：

一种是 setup.exe 客户端包（这个双击跟装 QQ 一样，就不说了）

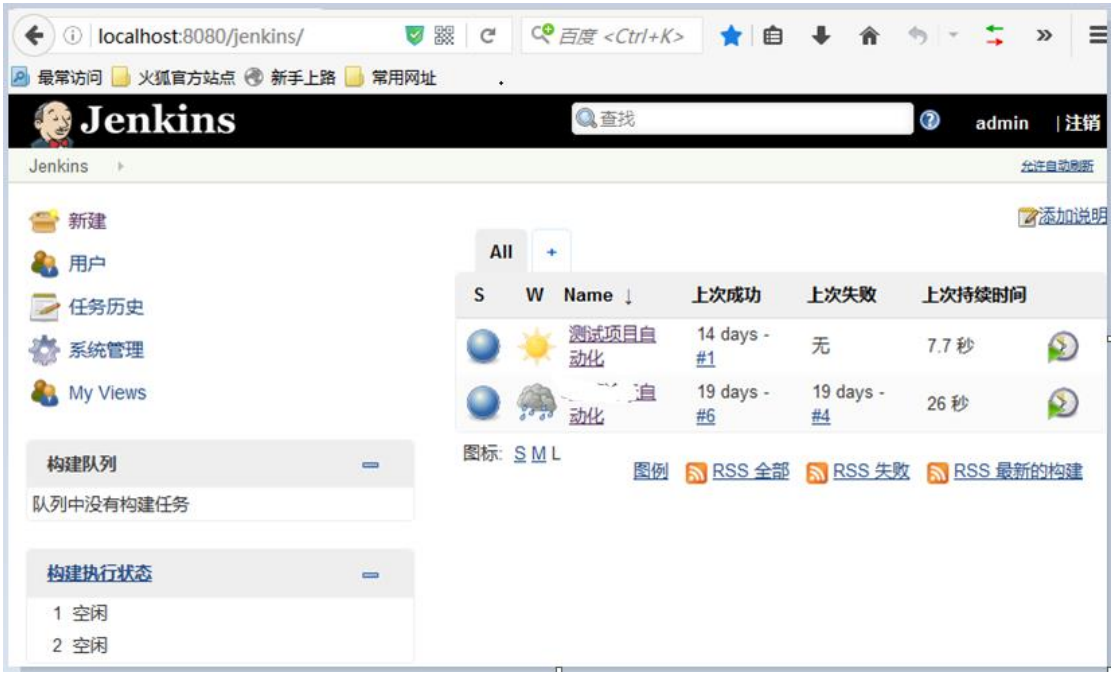
一种是 war 包

2.war 包放到 webapps 下



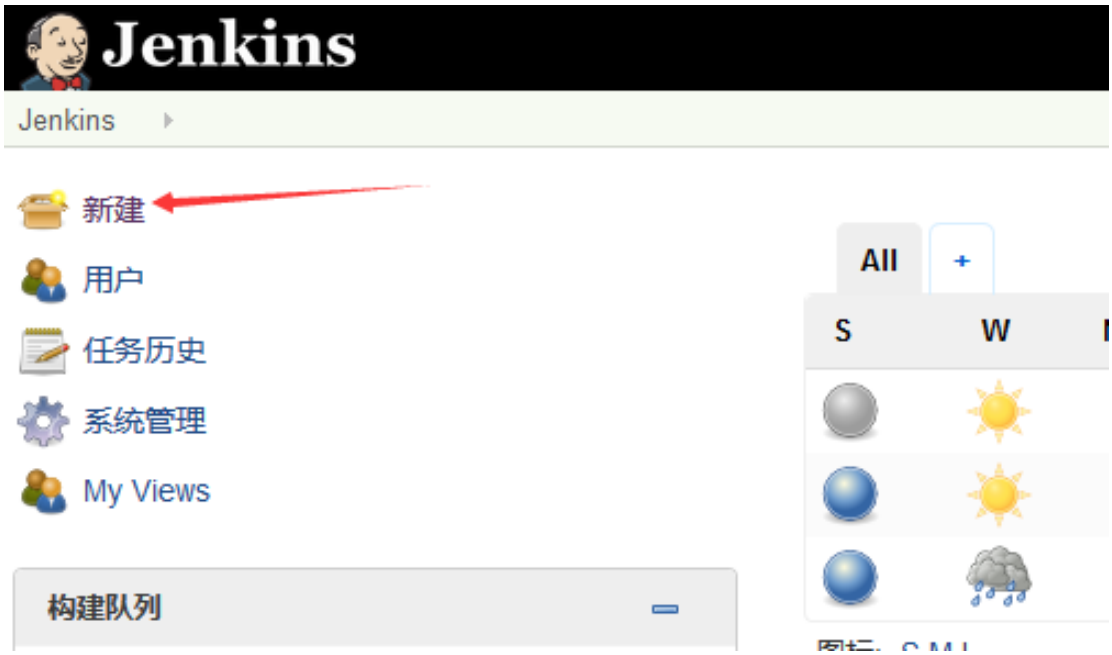
3.启动 jenkins

先重启 tomcat：startup.bat，然后在浏览器输入 http://localhost:8080/jenkins/



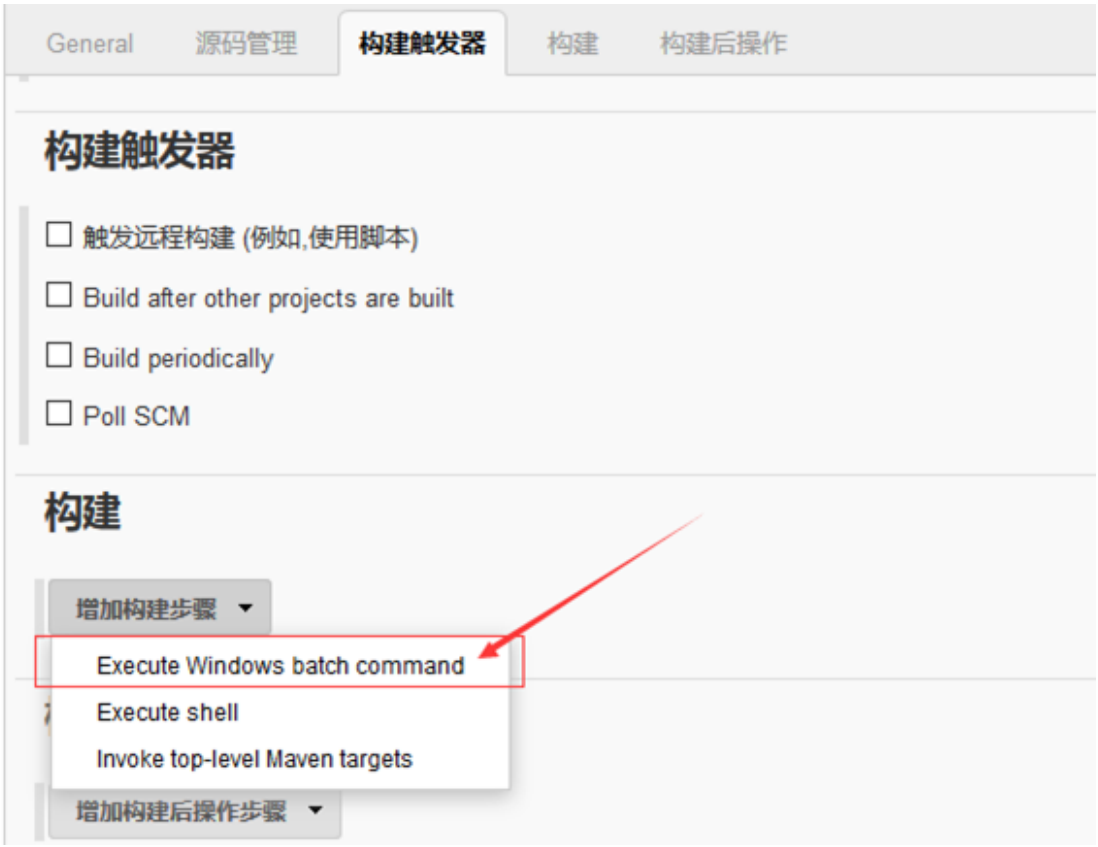
五、配置 jenkins

1.新建一个 job

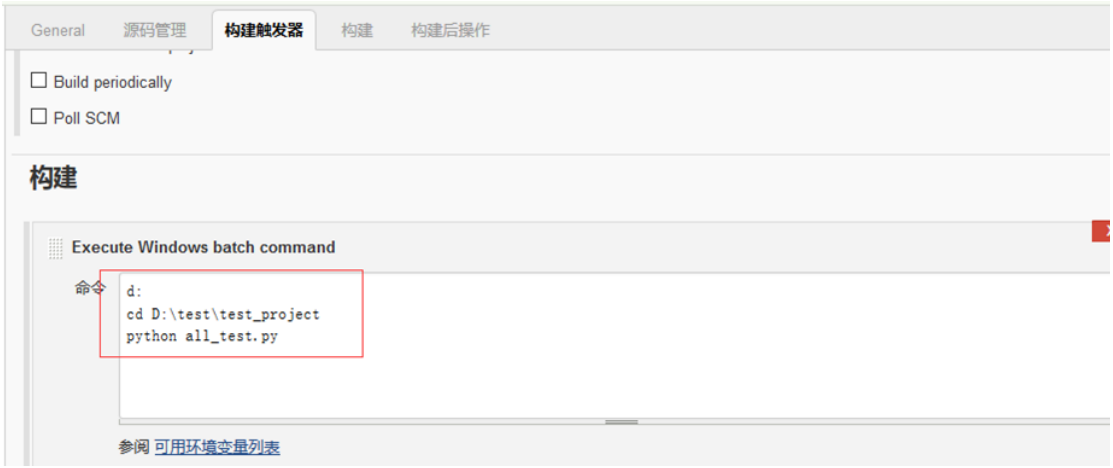


2.输入项目名称

3.构建触发器



4.配置 cmd 指令



5.选择项目一键执行，启动自动化

7.6 git+jenkins 配置 windows 的 slave 节点

前言

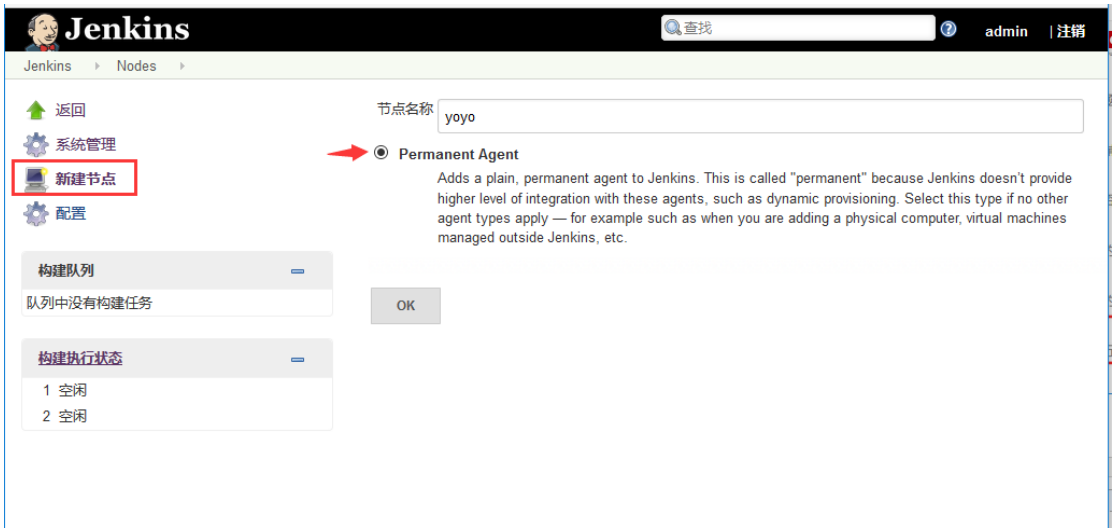
我们在跑自动化项目的时候，希望有单独的测试机能跑自动化项目，并且能集成到 jenkins

上构建任务。如果公司已经有 jenkins 环境了，那无需重新搭建。
只需在现有的平台基础上，新增一个 slave 节点管理就行了。

7.6.1 管理节点

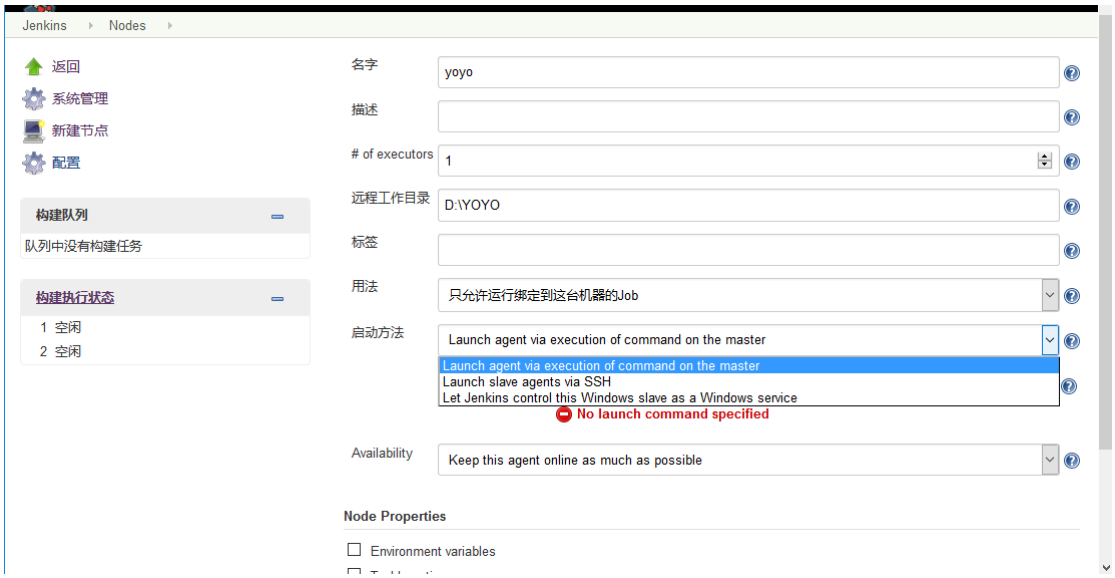
1.打开需要配置的测试电脑（一定要在需要配置的机器上操作），浏览器输入 jenkins 地址
2.系统管理-管理节点

3.新建节点，给节点命个名：yoyo
选中 Permanent Agent
点 OK

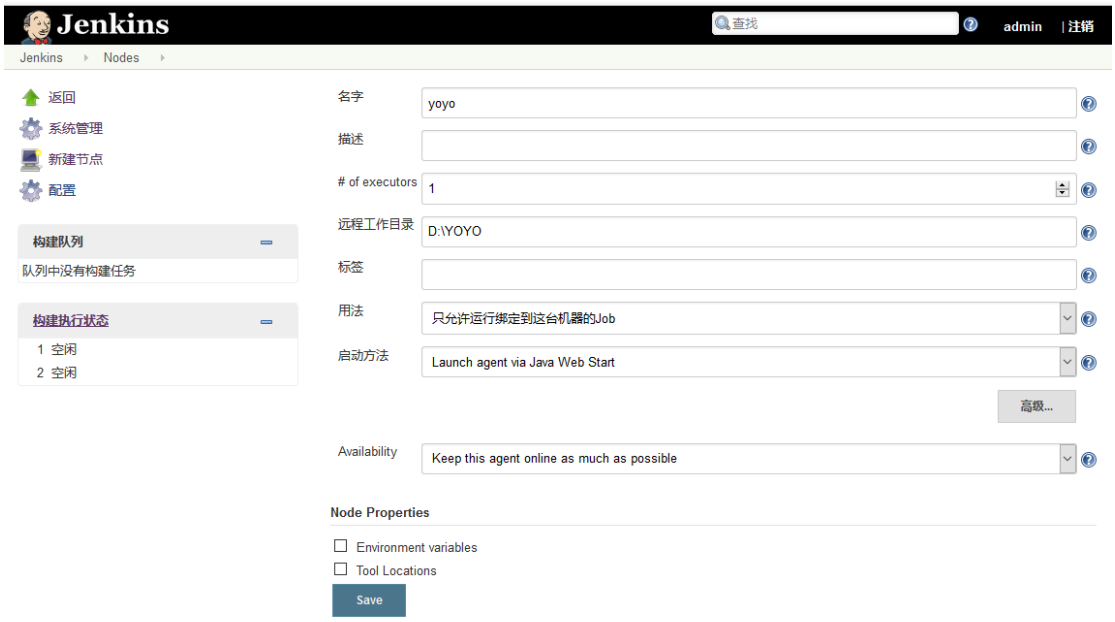


7.6.2 配置节点

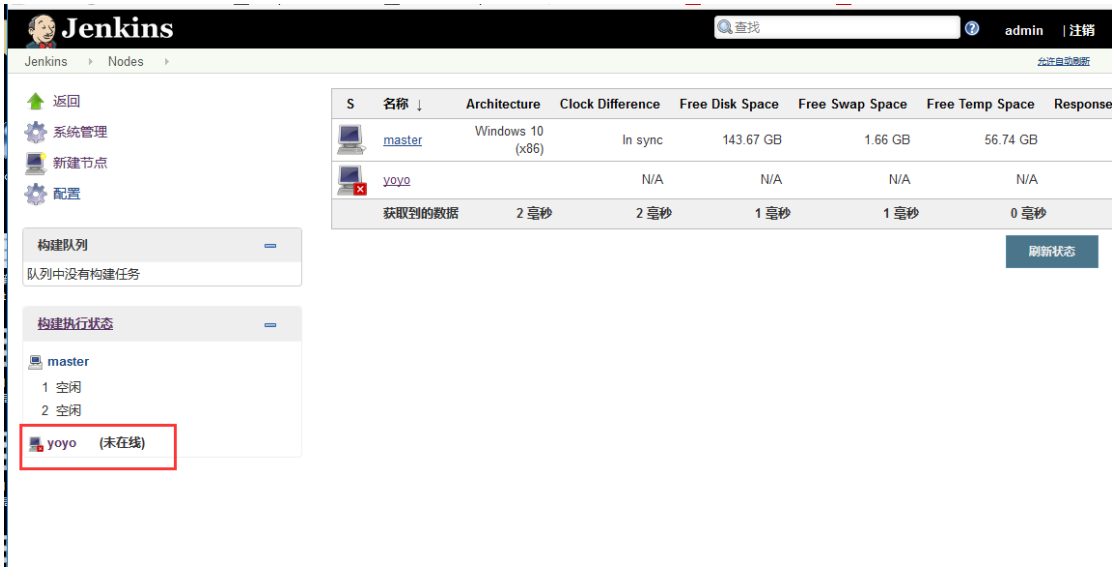
1.远程工作目录：D:\YOYO
--这个地址是测试机的 jenkins 的 workspace 工作目录，自己随便写个本机的路径
2.用法
--尽可能的使用这个节点：其它的 job 也能在这台机器上运行,如果只想运行测试自动化代码，就不选这个
--只允许运行绑定到这台机器的 Job：测试自动化的 Job 绑定这个机器后，就只能在这个机器运行了
3.启动方法：
--windows 上启动方法最好通过 Java web start 来启动 slave
（如果没这个选项就看下一步）
--linux 上启动选：Launch slave agents via SSH



4.如果有的话，启动方法勾选：Launch agents Java Web Start



5.点 Save 保存之后，左边会多一个机器

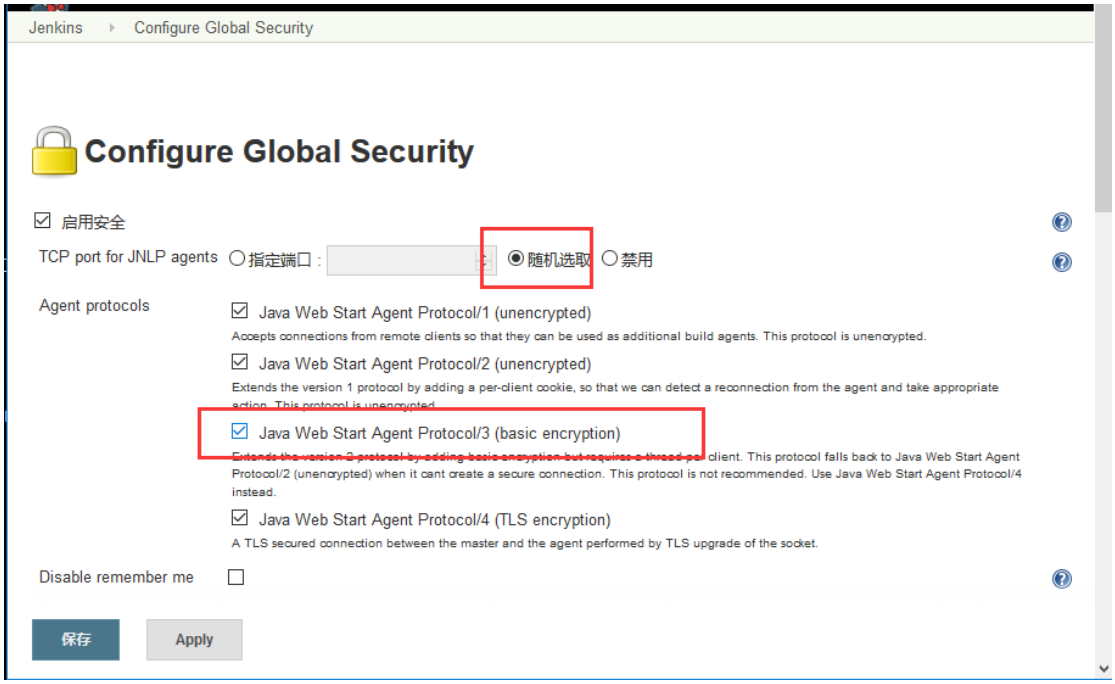


7.6.3 Java web start

1.windows 上最稳的启动方法，选 Java web start 这个选项，如果没有的话，先去系统管理开启

2.系统管理--Configure Global Security

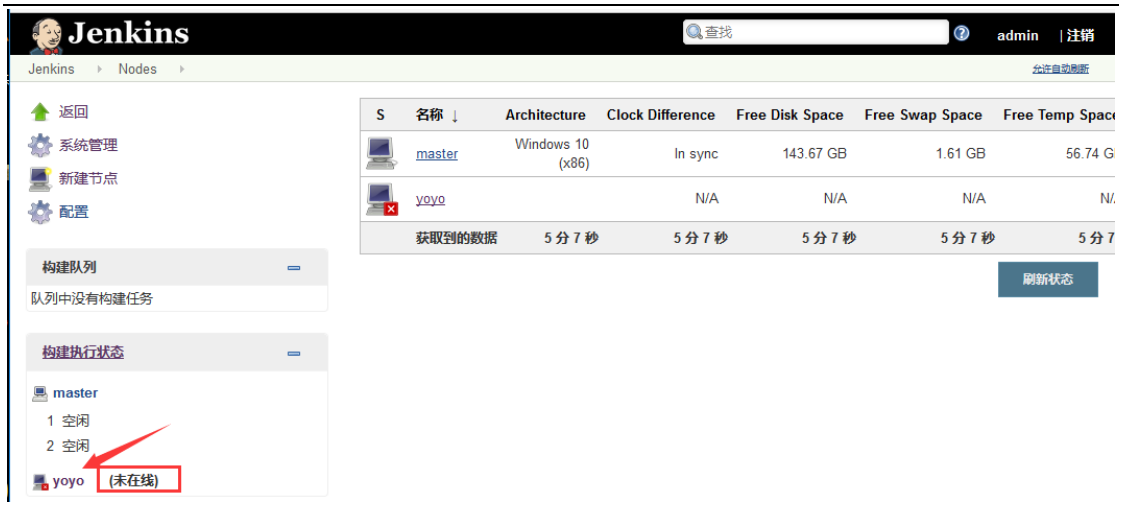
3.点开后，勾选‘随机选取’
点开 Agent Protocols 选项，勾选 Java Web Start Agent



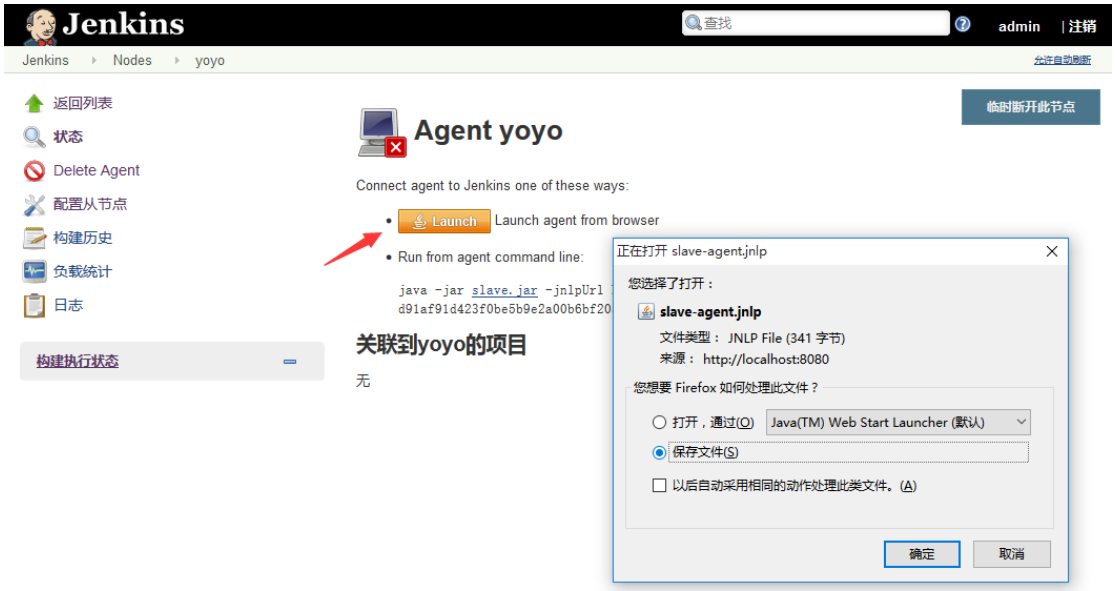
4.点 Apply，再点保存按钮
5.再回到上面的配置节点，启动方法，勾选 Java web start

7.6.4 启动 Salve

1.上面配置的节点是未在线状态，如：yoyo(未在线），这个时候是不能构建任务的

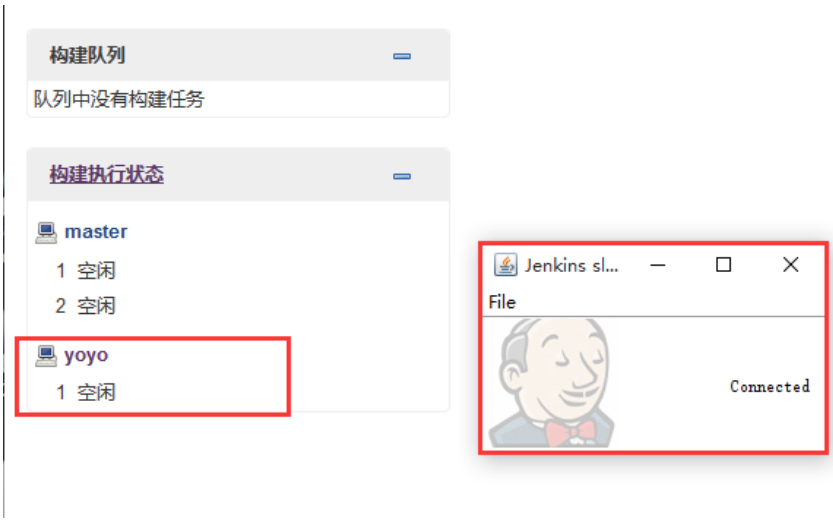


2.点开它，本地安装 slave-agent



3.点上图的 Launch 按钮，会弹出下载提示，可以保存到本地电脑，放到上面新建的 D:\YOYO 目录，方便以后查找

4.双击 slave-agent.jnlp 启动，启动后桌面上会多个图标

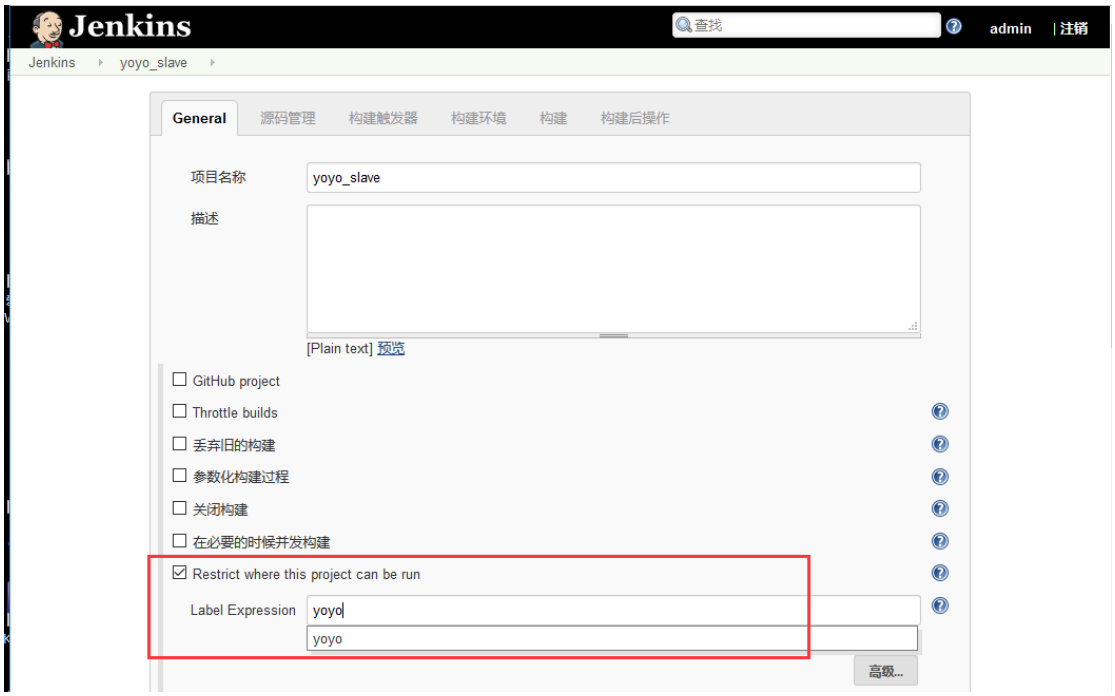


5.回到 jenkins 首页后，这个节点就是在线状态了

7.6.5 新建 Job

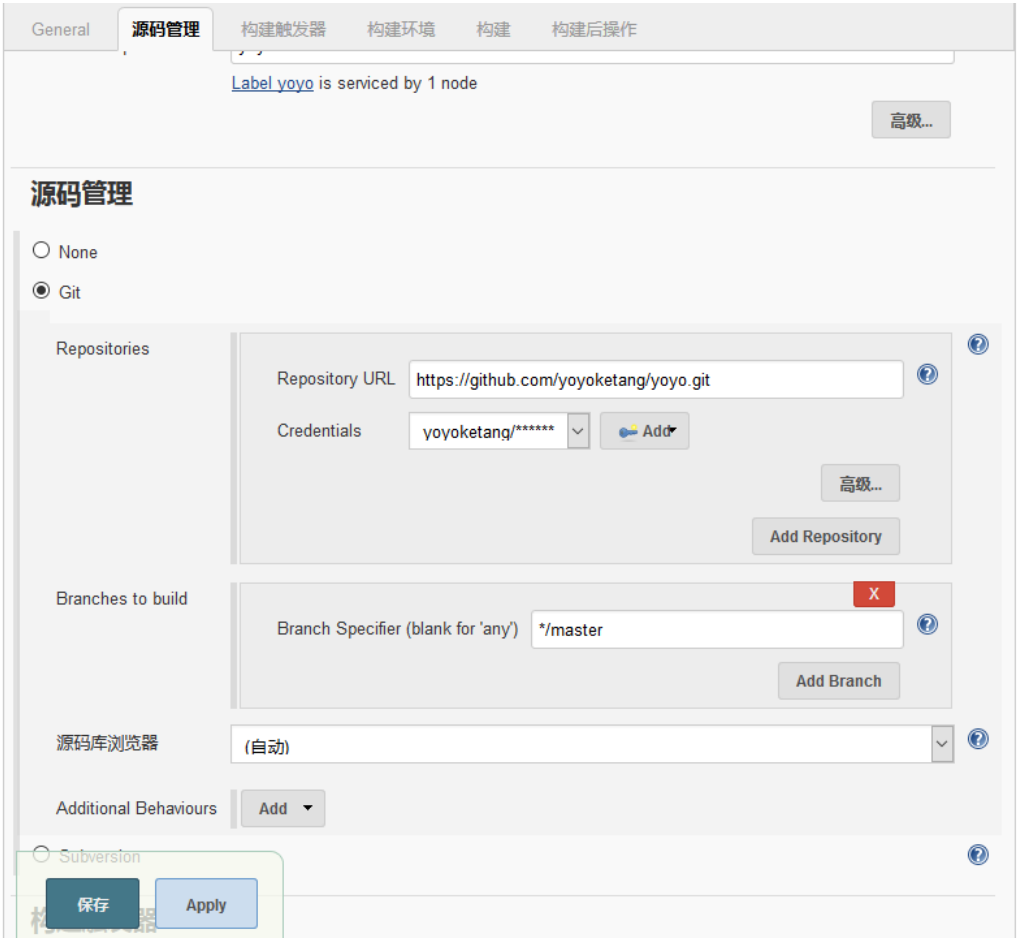
1.新建的 Job，指定到刚才新增的这个节点机器上运行,勾选 Restrict where this project can be run

2.Label Expression 这个地方输入刚才节点机器的名称：yoyo

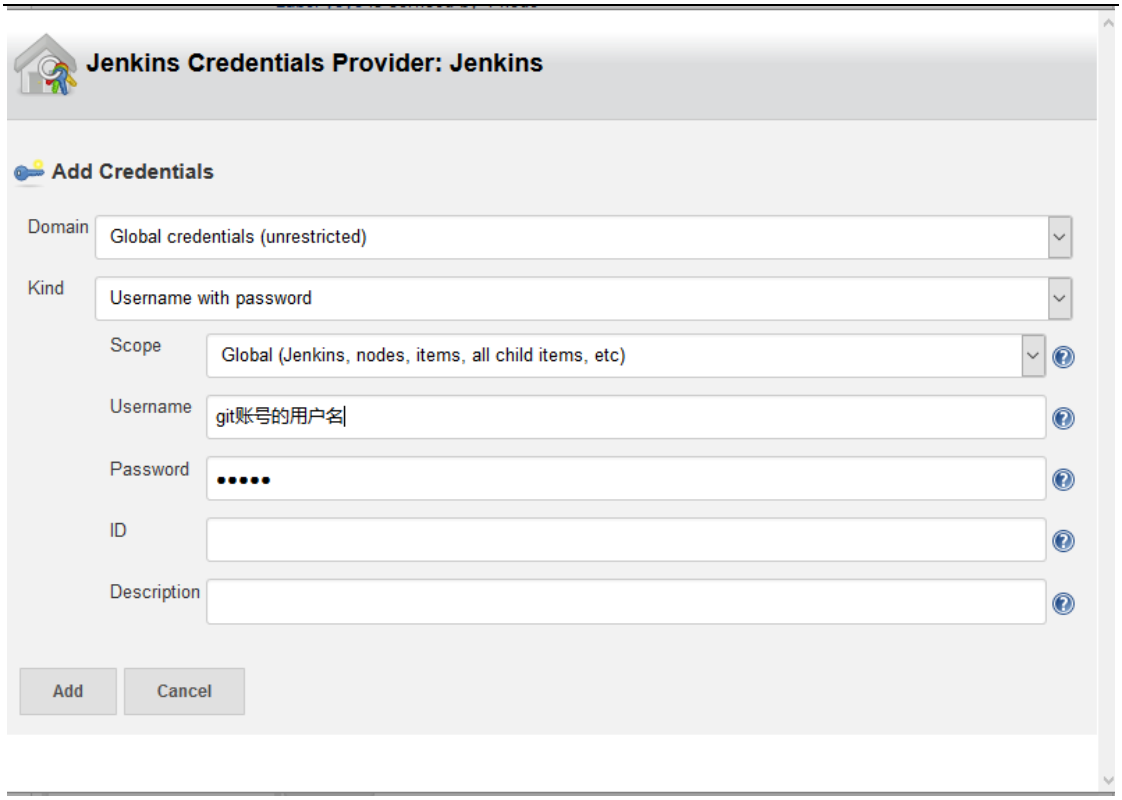


7.6.6 源码管理：git

1.源码管理里面勾选 Git 选项
2.Repository URL 这个地址就是我 git 仓库的地址



3.Credentials 点 add 添加 git 的账号和密码



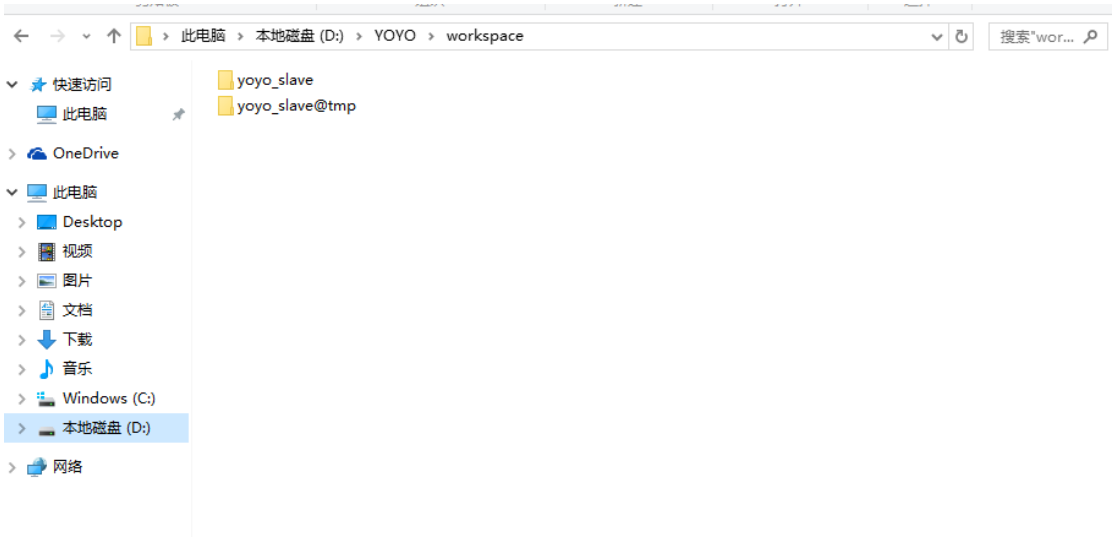
4.Branch Specifier (blank for 'any')这个地方的地址是分支名，如果没指定其它分支，就是默认：*/master

7.6.7 执行 dos 指令

1.添加构建步骤-Execute Windows batch command

2.配置运行 python 脚本的指令

3.最后保存后触发 Job,打开本地电脑的 D:\YOYO\workspace 路径



4.点开 yoyo_slave 这个 Job 名称，里面的代码就是从 git 上自动获取到的

个人博客地址：http://www.cnblogs.com/yoyoketang/

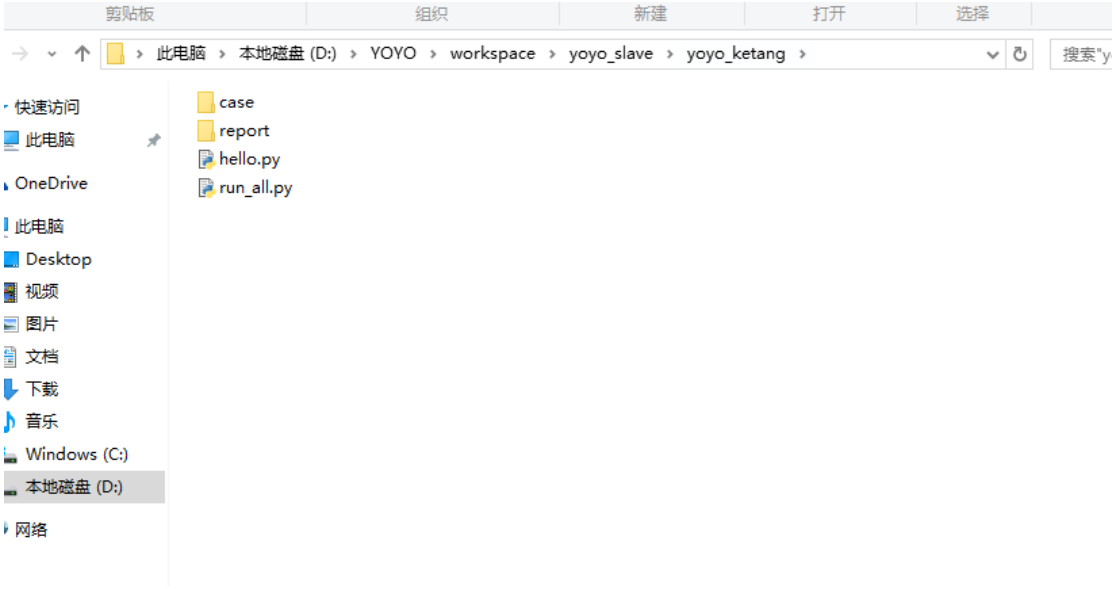
QQ 交流群：226296743

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743



7.7 jenkins 定时构建

前言
跑自动化用例每次用手工点击 jenkins 出发自动化用例太麻烦了，我们希望能每天固定时间跑，这样就不用管了，坐等收测试报告结果就行。

7.7.1 定时构建语法

* * * * *
(五颗星，中间用空格隔开）
第一颗*表示分钟，取值 0~59
第二颗*表示小时，取值 0~23
第三颗*表示一个月的第几天，取值 1~31
第四颗*表示第几月，取值 1~12
第五颗*表示一周中的第几天，取值 0~7，其中 0 和 7 代表的都是周日
1.每 30 分钟构建一次：
H/30 * * * *
2.每 2 个小时构建一次
H H/2 * * *
3.每天早上 8 点构建一次
0 8 * * *
4.每天的 8 点，12 点，22 点，一天构建 3 次
0 8,12,22 * * *
（多个时间点，中间用逗号隔开）

7.7.2 Build periodically

1.Build periodically：周期性进行项目构建，这个是到指定的时间必须触发构建任务

2.比如我想在每天的 9 点，17 点，朝九晚五各构建一次，在 Build periodically 里设置如下

3.上面红色字体：Spread load evenly by using ‘H 9,17 * * *’ rather than ‘0 9,17 * * *’，这句话大概意思就是说，用这个语法会比后后面那个好：H 9,17 * * *

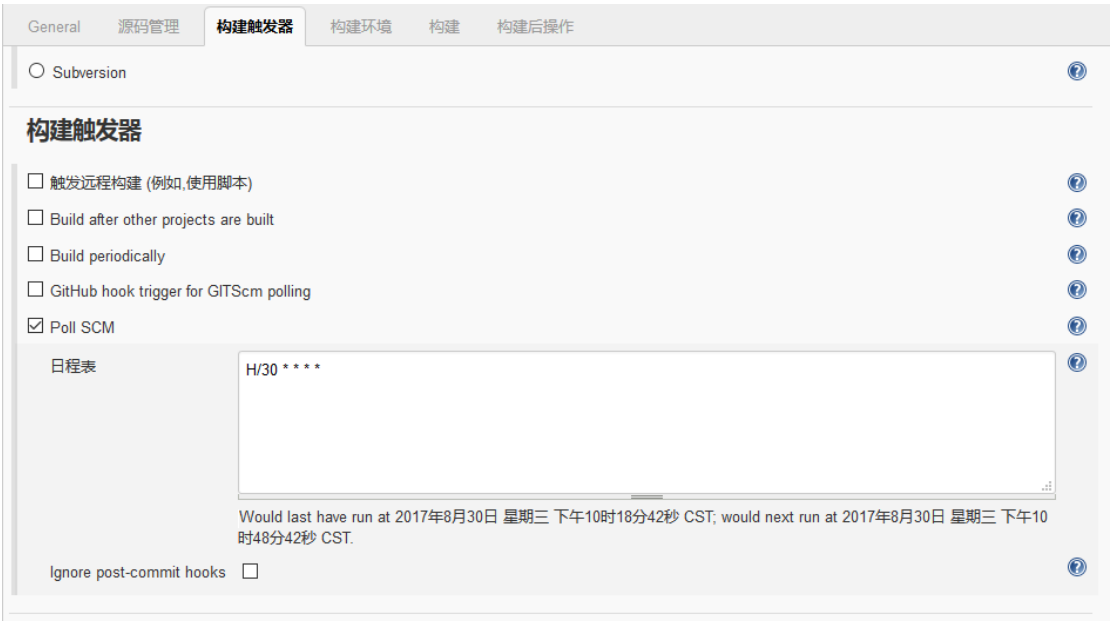
4.下一次构建时间是 05 时 48 分 06 秒，然后再下次是 09 时 48 分 06 秒

7.7.3 Poll SCM

1.Poll SCM:定时检查源码变更（根据 SCM 软件的版本号），如果有更新就 checkout 最新 code

下来，然后执行构建动作

2.如果我想每隔 30 分钟检查一次源码变化，有变化就执行

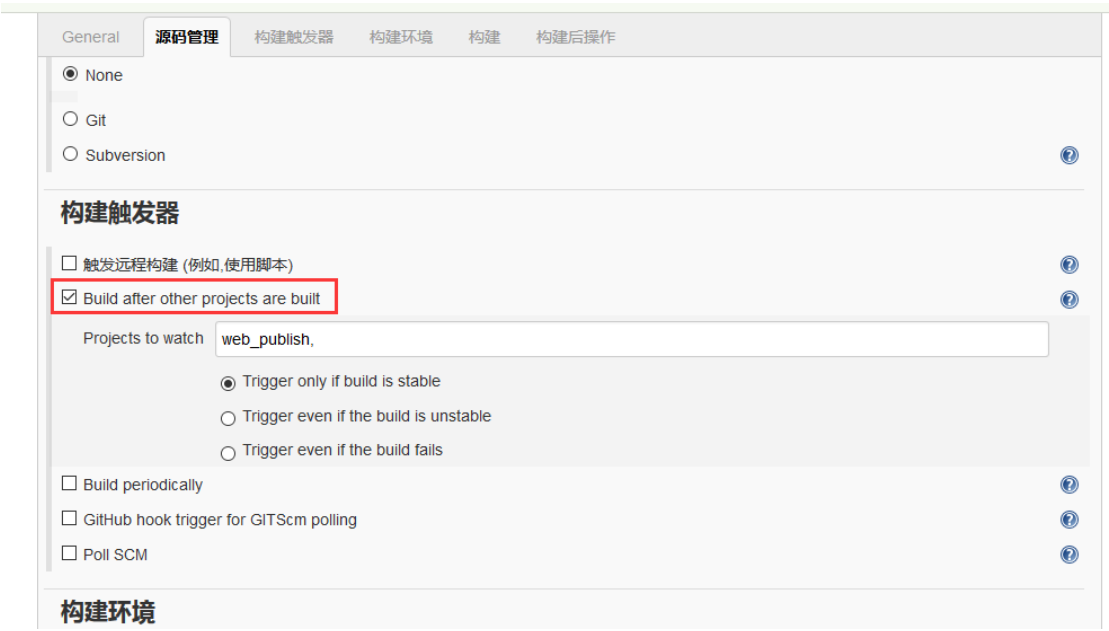


7.7.4 Job 关联

1.举个案例场景，比如我下面 Job1 是 web 项目打包并发布的构建任务，我想每次打完包发布后，然后触发自动化测试 Job2 的构建。

（当然发布后，一般会等几分钟才会完全加载完成，再下一次构建的时候，可以用 python 加个脚本 sleep 几分钟）

2.构建触发器勾选 Build after other projects are built，Projects to watch 输入 Job1 的名称（这里可以输入多个依赖的 jobs，多个 job 中间用逗号隔开）



3.下面有三个选择，一般默认第一个就行
Trigger only if build is stable：构建稳定时触发
Trigger even if the build is unstable ：构建不稳定时触发
Trigger even if the build fails ： 构建失败的时候触发
4.上面设置好后，启动第一个 Job 完成后，就能接着启动第二个 Job 了

7.7.5 另外两种

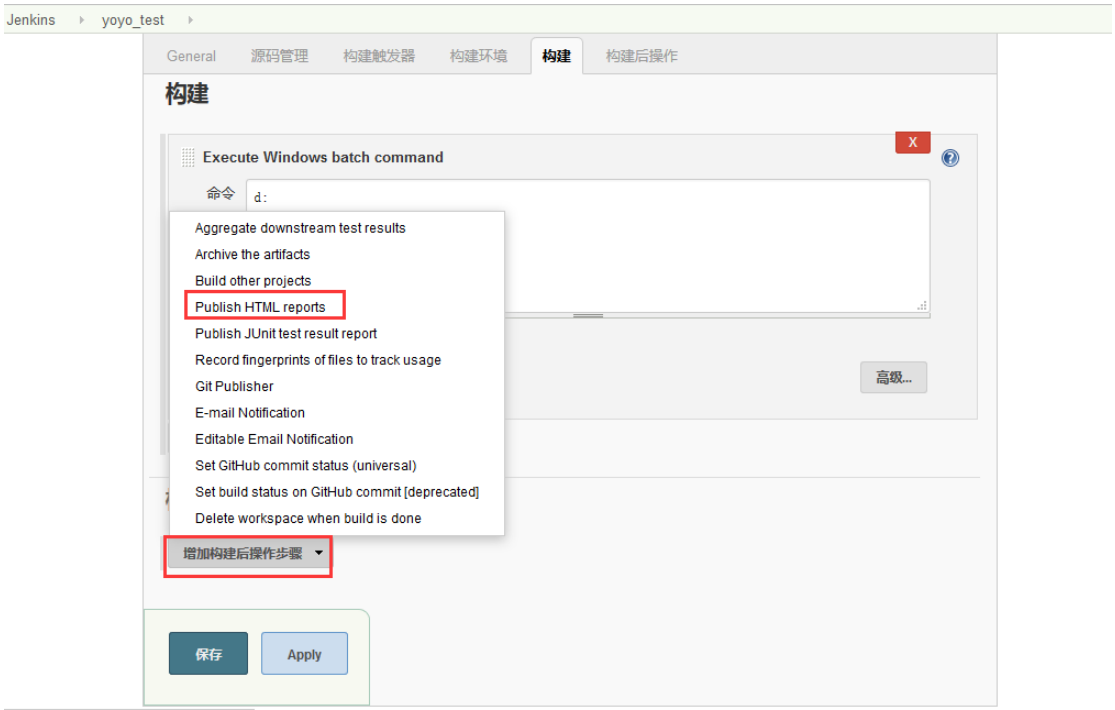
1.触发远程构建 (例如,使用脚本）
2.GitHub hook trigger for GITScm polling： 这个是管理 github 上代码有变动时构建
最后这 2 个一般用的也少，了解下就行

7.8 jenkins 展示 html 报告

前言
在 jenkins 上展示 html 的报告，需要添加一个 HTML Publisher plugin 插件，把生成的 html 报告放到指定文件夹，这样就能用 jenkins 去读出指定文件夹的报告了。

7.8.1 构建后操作

1.前面执行完测试用例后，可以用“添加构建后操作步骤”，读出 html 报告文件

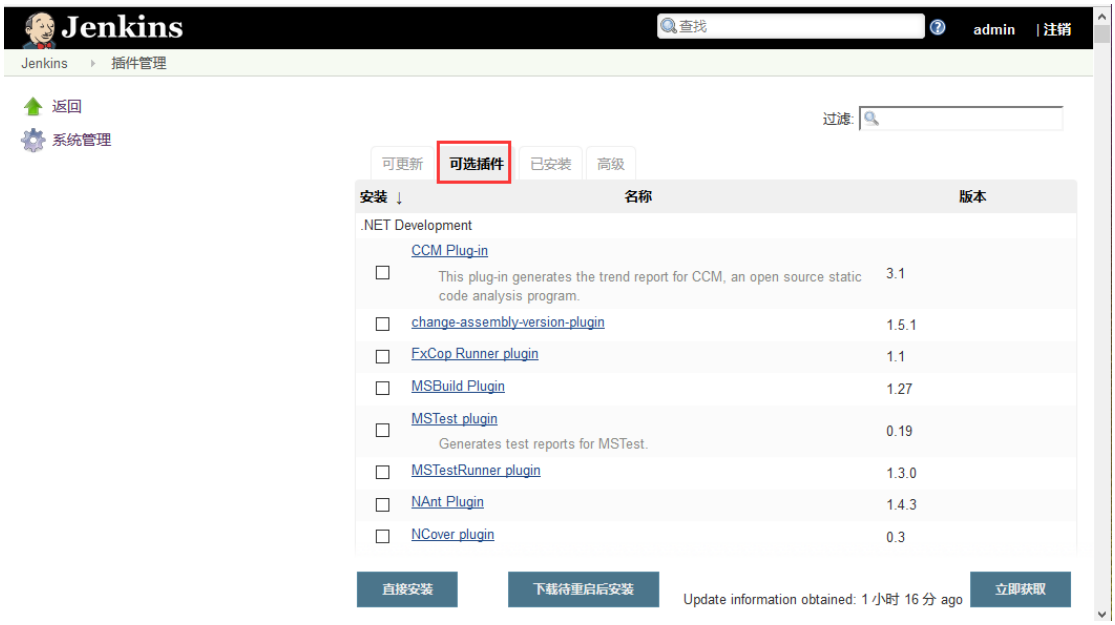


2.如果你的展开后有 Publish THML reports 这个选项就能直接添加了，没有就看下一步。

7.8.2 添加 HTML Publisher plugin 插件

1.打开系统管理-管理插件

2.选中“可选插件”界面

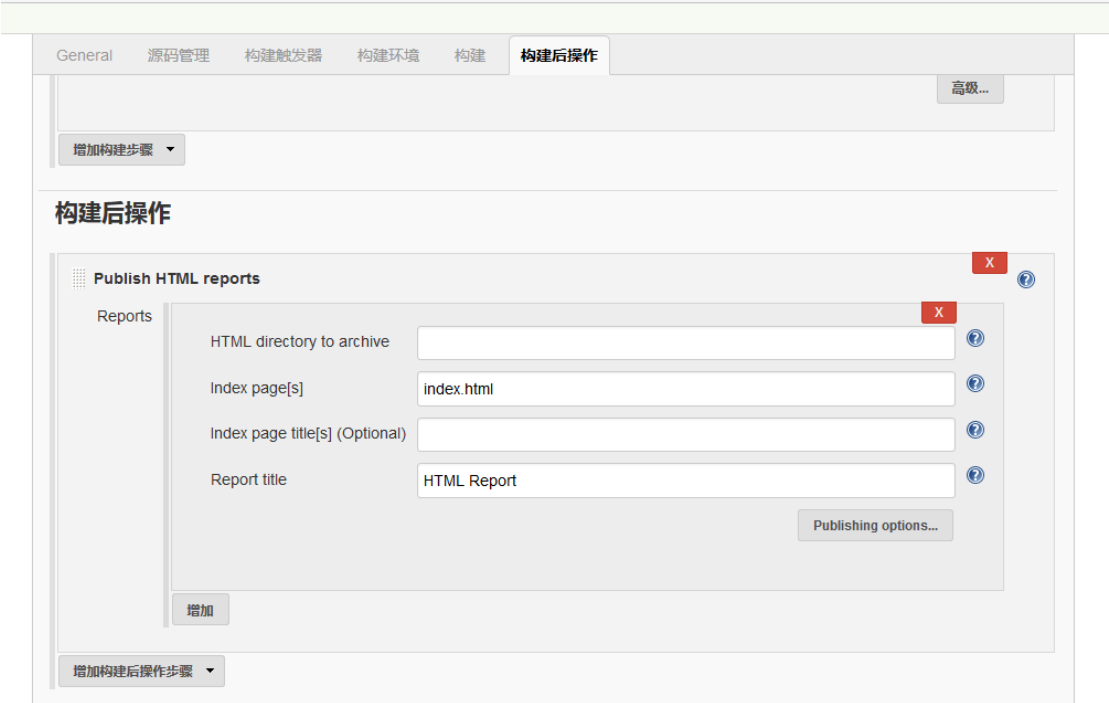


3.在右上角搜索需要安装的插件：HTML Publisher plugin

4.勾选后直接安装，安装完之后可以重新启动 jenkins，再回到第一步的操作就能看到有这个选项了。

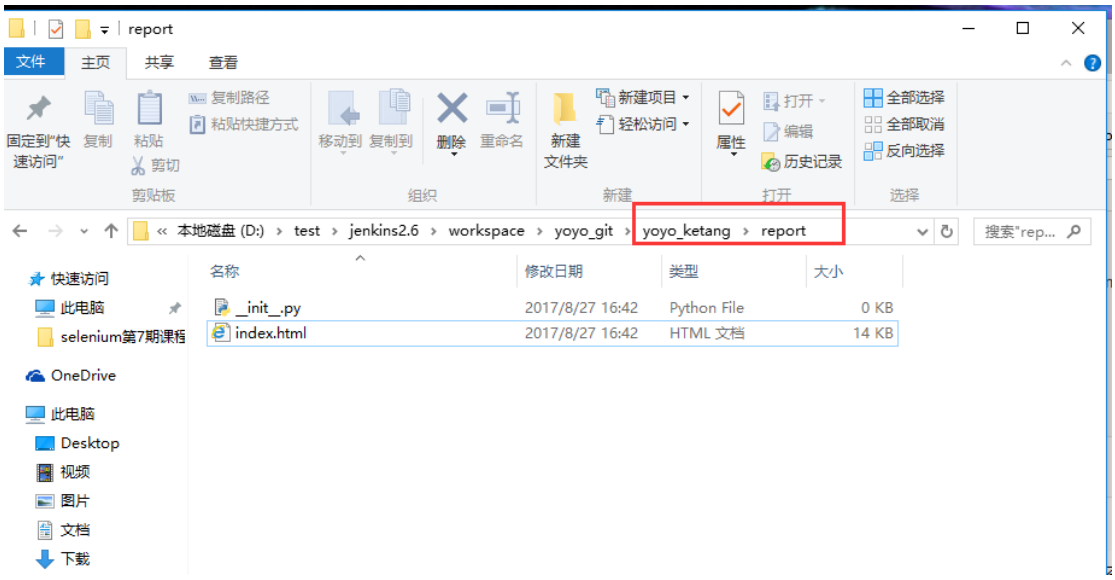
7.8.3 添加 reports

1.点开 Reports 后，界面显示如下



2.HTML directory to archive：这里是在本机上运行完脚本后生成的测试报告路径，这里是相对路径。

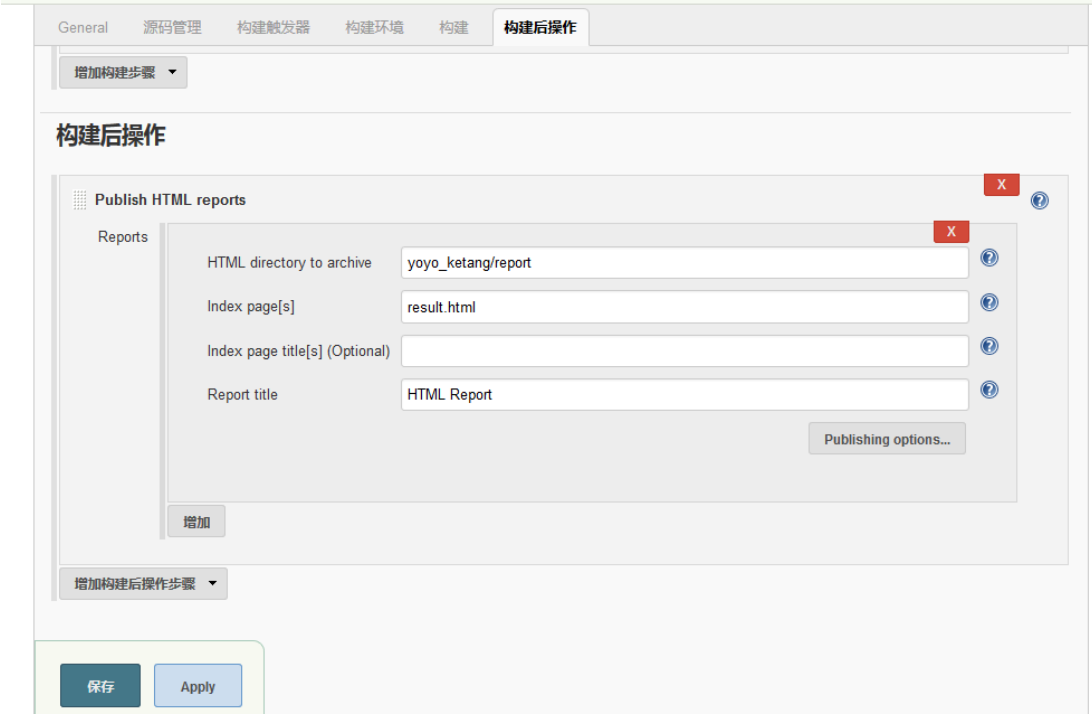
如果前面是用 git 自动拉取的代码，那就在 workspace 找到生成报告的路径，这里工程名称 yoyo_git 可以省略，相对路径填 yoyo_ketang/report



3.报告名称：index.html,这个就是运行完脚本后，在 report 文件夹下生成报告的名称，自己随便取名，保证跟 report 文件夹下名称一致就行

（每次生成一个固定的名称，新的覆盖旧的，不要加时间戳）

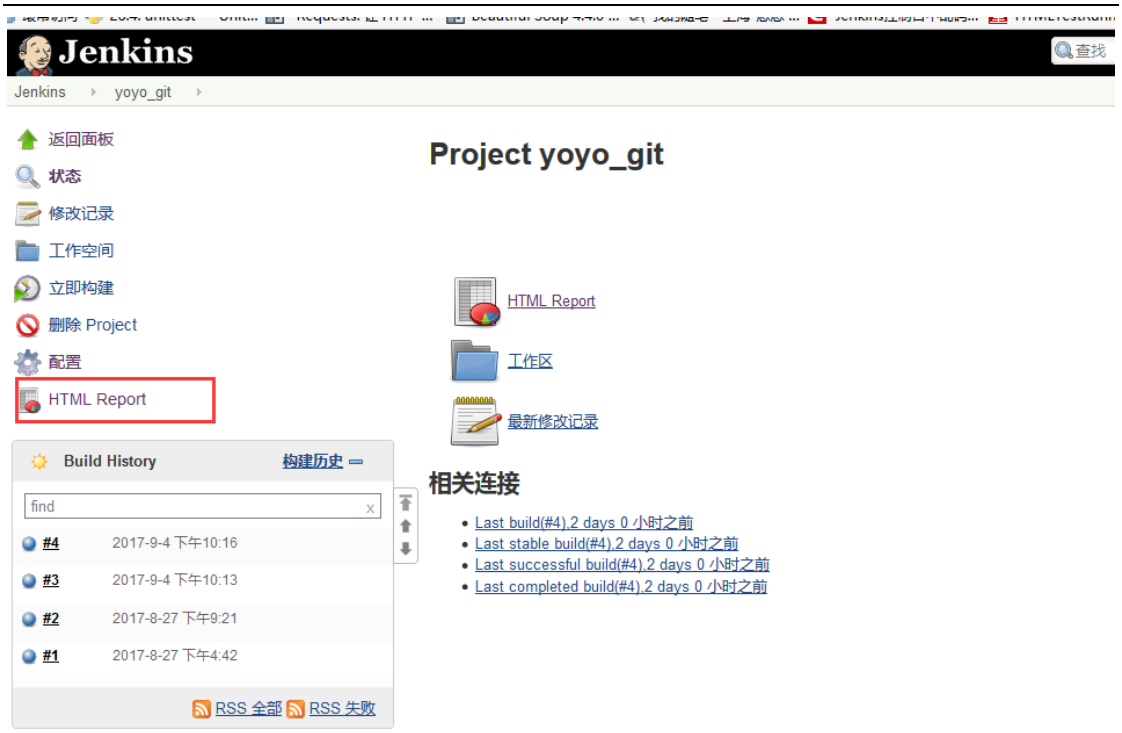
4.显示在 jenkins 上的名称，默认 HTML Report 就行



5.点 apply 运用

7.8.4 报告展示

1.运行完之后，在左侧工程下会生成一个 HTML Report 目录



2.点开查看详情

3.点开后，正常情况应该跟本地 html 浏览器打开是一样的。

这里 html 格式丢失了，因为 jenkins 默认没有加载 css 样式，需要设置下。

7.9 jenkins 上 html 展示问题

前言
jenkins 上使用 HTML Publisher plugin
http://wiki.jenkins-ci.org/display/JENKINS/HTML+Publisher+Plugin
插件生成的 html 报告样式会丢失，需要设置下才能正常显示。

7.9.1 样式丢失

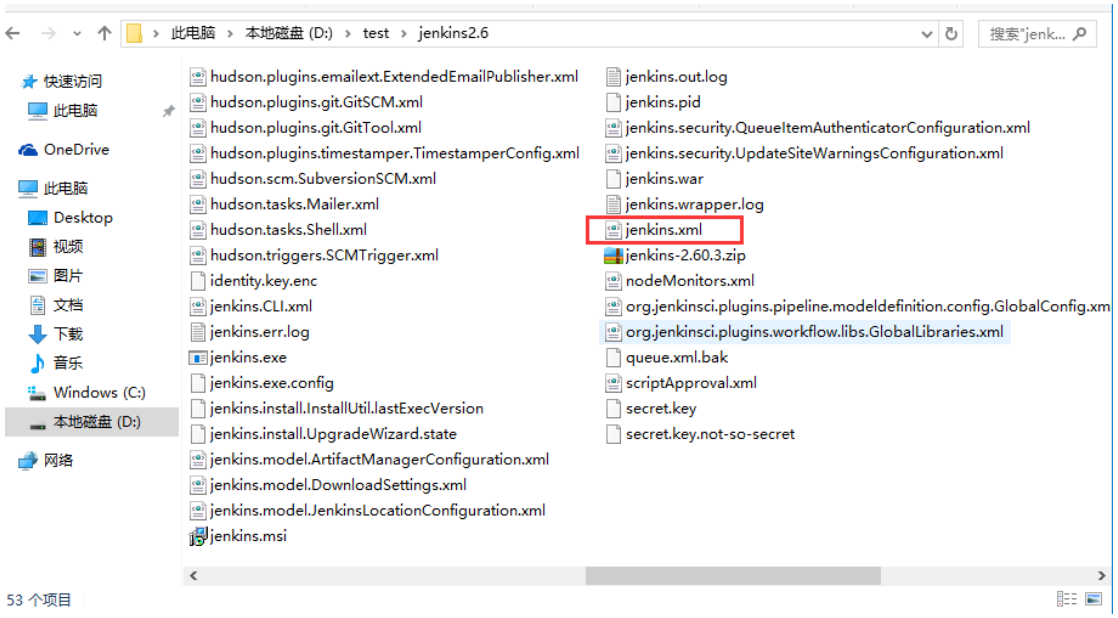
1.官方文档的解释如下，参考地址
https://stackoverflow.com/questions/35783964/jenkins-html-publisher-plugin-no-css-is-displayed-when-report-is-viewed-in-j

从安全性考虑，禁止了 css 和 js 的加载，所以无法展示。
This rule set results in the following:
No JavaScript allowed at all
No plugins (object/embed) allowed
No inline CSS, or CSS from other sites allowed
No images from other sites allowed
No frames allowed
No web fonts allowed
No XHR/AJAX allowed, etc.

2.展示结果如下

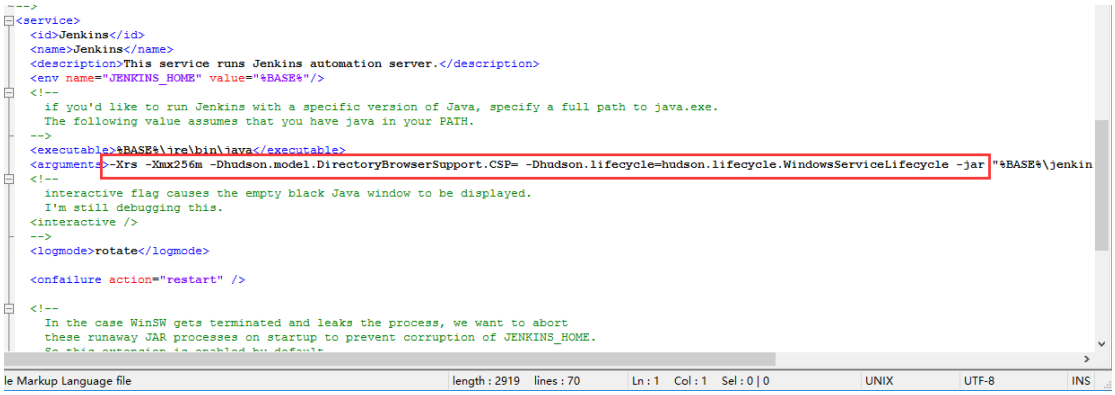
7.9.2 jenkins.msi 解决方案

1.jenkins 的安装分为 jenkins.msi 客户端安装和 jenkins.war 服务端安装，如果是 jenkins.msi 的安装，找到 jenkins.xml 文件



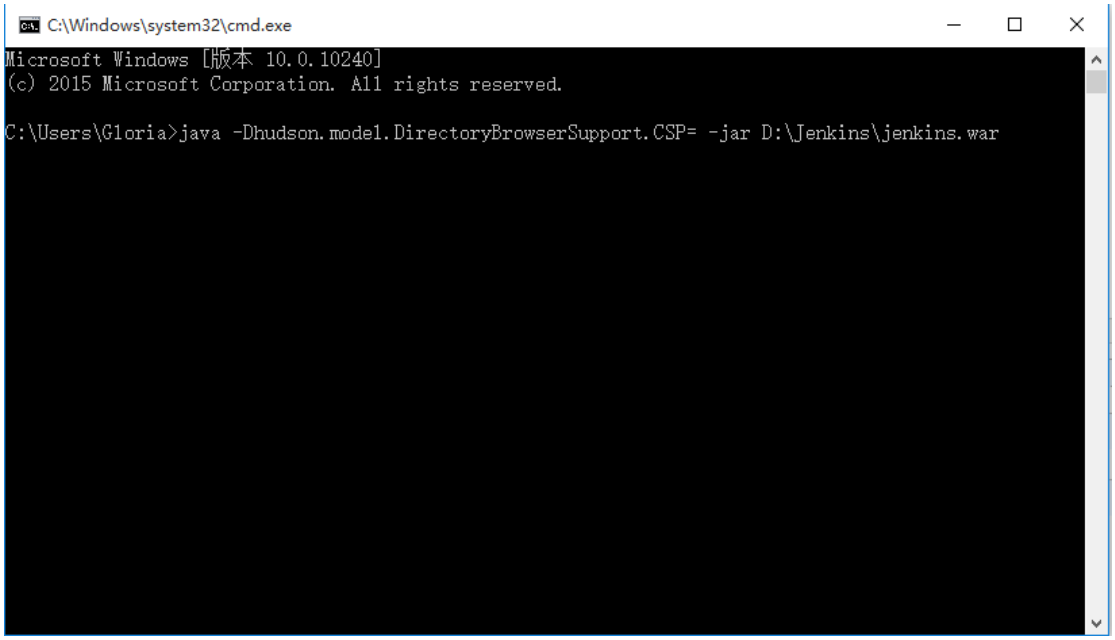
2.用 Notepad++打开这个 jenkins.xml 文件，找到下图的 arguments 这一行，红色区域改成如下，改好后保存：

原来是：-Xrs -Xmx256m -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -jar
修改后：-Xrs -Xmx256m -Dhudson.model.DirectoryBrowserSupport.CSP= -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle –jar



7.9.3 jenkins.war 解决方案

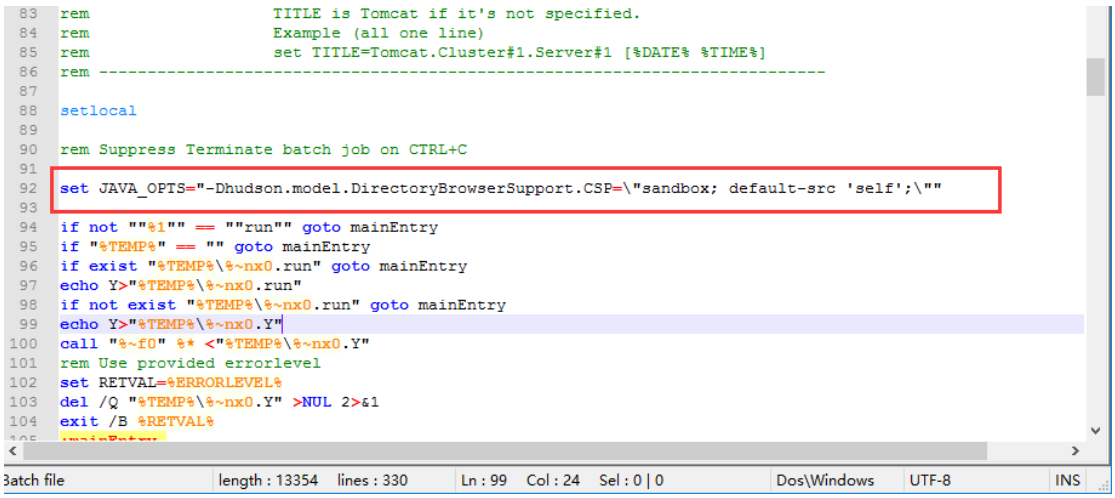
方法一：
1.如果 Jenkins 为 war 包，也就是部署到 tomcat 上这种，可以通过 java 命令运行解决
2. 先关闭 jenkins，打开 cmd 通过如下命令运行 Jenkins:java -Dhudson.model.DirectoryBrowserSupport.CSP= -jar <jenkins.war 的路径>
>java -Dhudson.model.DirectoryBrowserSupport.CSP= -jar D:\Jenkins\jenkins.war



方法二：
1.可以直接改 catalina.bat 文件，找到 tomcat/bin 下的 catalina.bat 文件,用 Notepad++打开，

加上一行代码

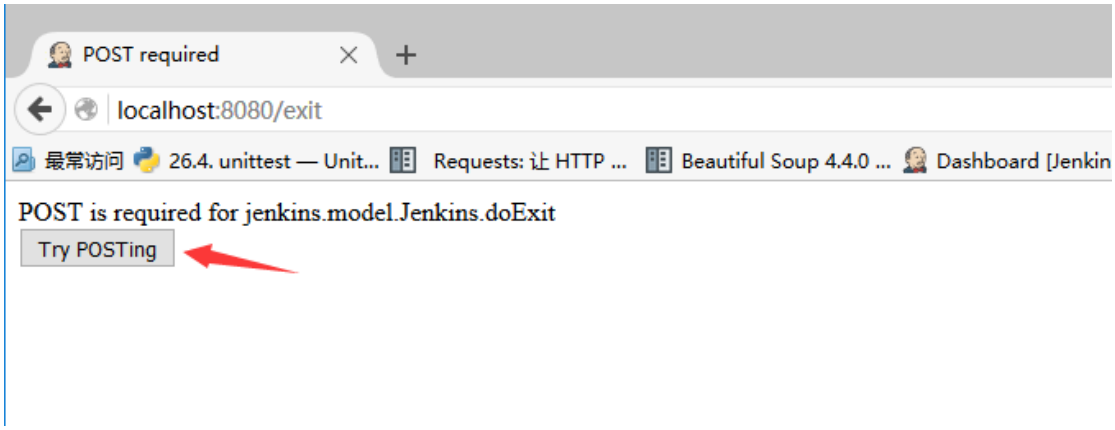
set JAVA_OPTS="-Dhudson.model.DirectoryBrowserSupport.CSP=\"sandbox; default-src ""



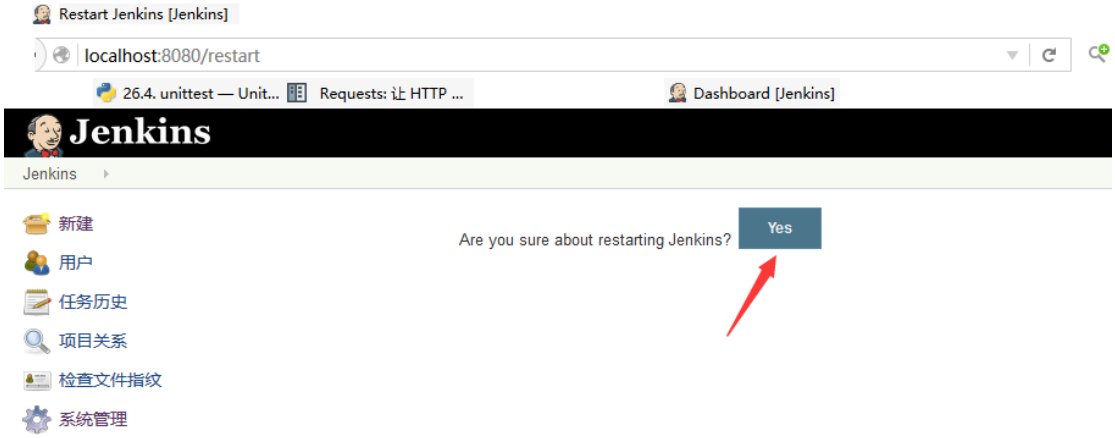
（以上几种方法设置好后，都需要重启 jenkins）

7.9.4 重启 jenkins

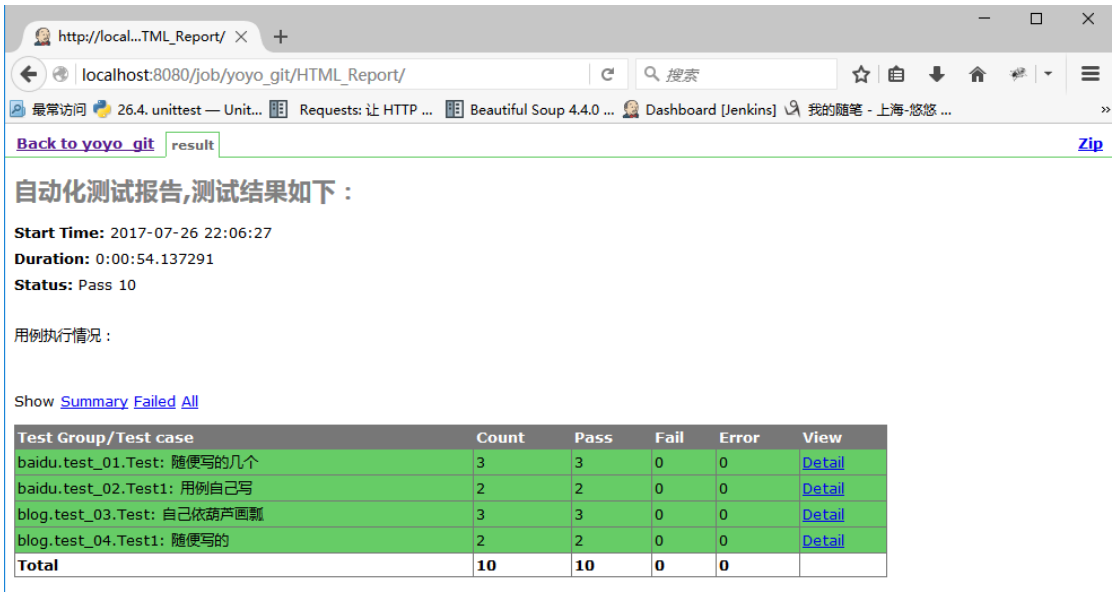
1.先关掉 jenkins,在地址栏 jenkins 的首页后加 exit,如 http://localhost:8080/exit，然后点 Try POSTing 就能关掉 jenkins 了



2.重启 jenkins，在地址栏 jenkins 的首页后加 restart,如 http://localhost:8080/restart ,点 Yes 就能重启了



3.重启后，再重新构建之前生成 html 的 job 任务，就能正常展示 html 报告了



Test Group/Test case	Count	Pass	Fail	Error	View
baidu.test_01.Test: 随便写的几个	3	3	0	0	Detail
baidu.test_02.Test1: 用例自己写	2	2	0	0	Detail
blog.test_03.Test: 自己依葫芦画瓢	3	3	0	0	Detail
blog.test_04.Test1: 随便写的	2	2	0	0	Detail
Total	**10**	**10**	**0**	**0**	

7.10 jenkins 添加 Mac 为 slave-agent

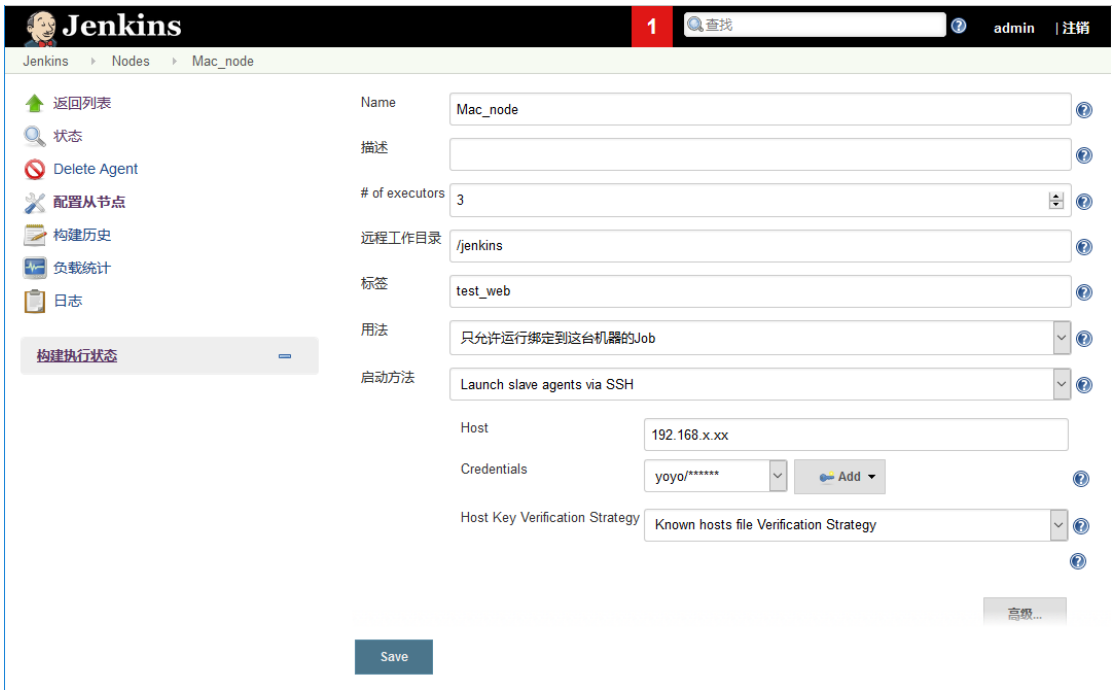
前言

如果有一台 Mac 机作为测试机单独跑自动化，那是极好的，并且很稳，本篇讲解如何把 Mac 机连到 jenkins 上作为一个节点单独跑自动化用例。

7.10.1 新建节点

系统管理>管理节点>新建节点，如下图取名为：Mac_node

2.配置信息



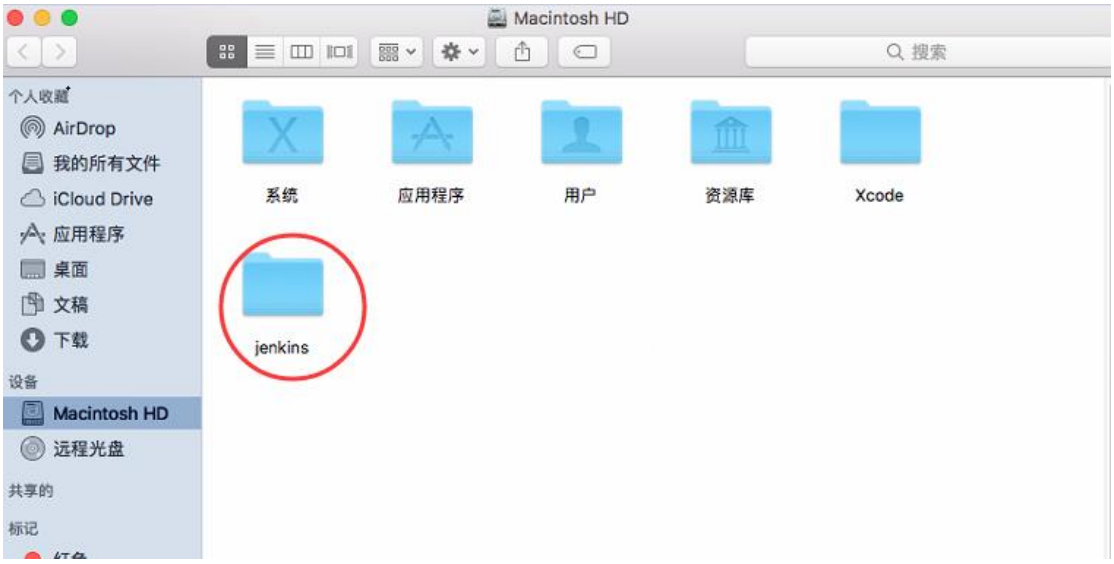
名字：Mac_node

描述：随便写

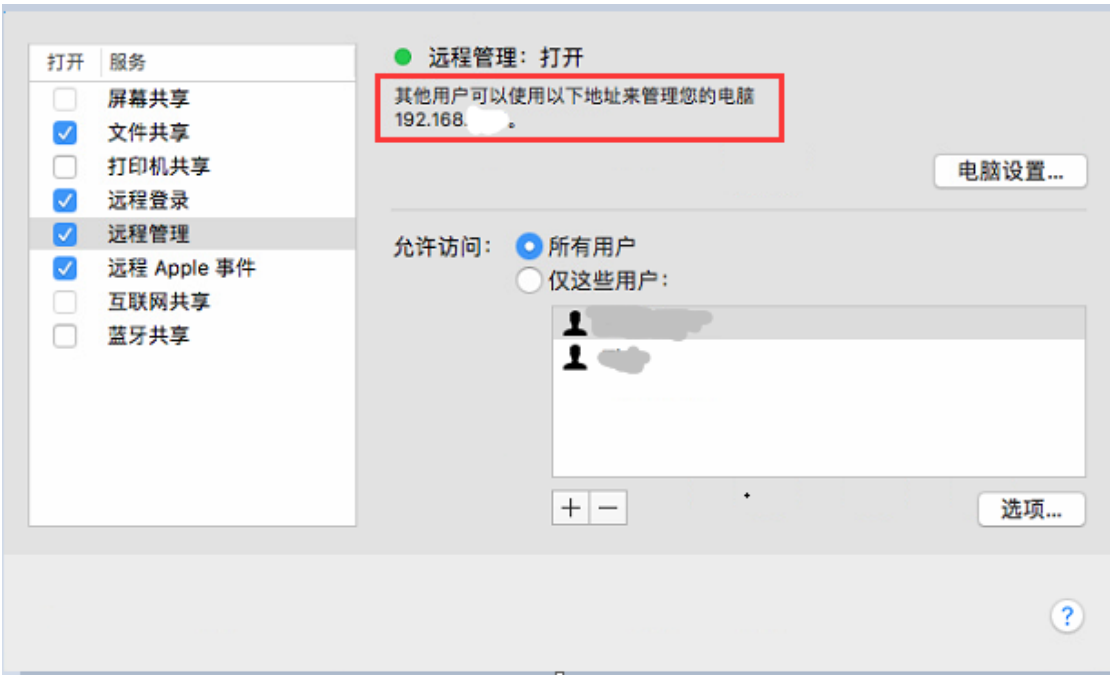
# of executors：3 (默认为 1，表示同该设备同时启动最大的 Job 数，一般不超过 6）

远程工作目录：在 Mac 上运行 jenkins 的工作目录 workspace,自己手动创建一个目录，这样后面 git 拉到的代码会自动放到该目录下

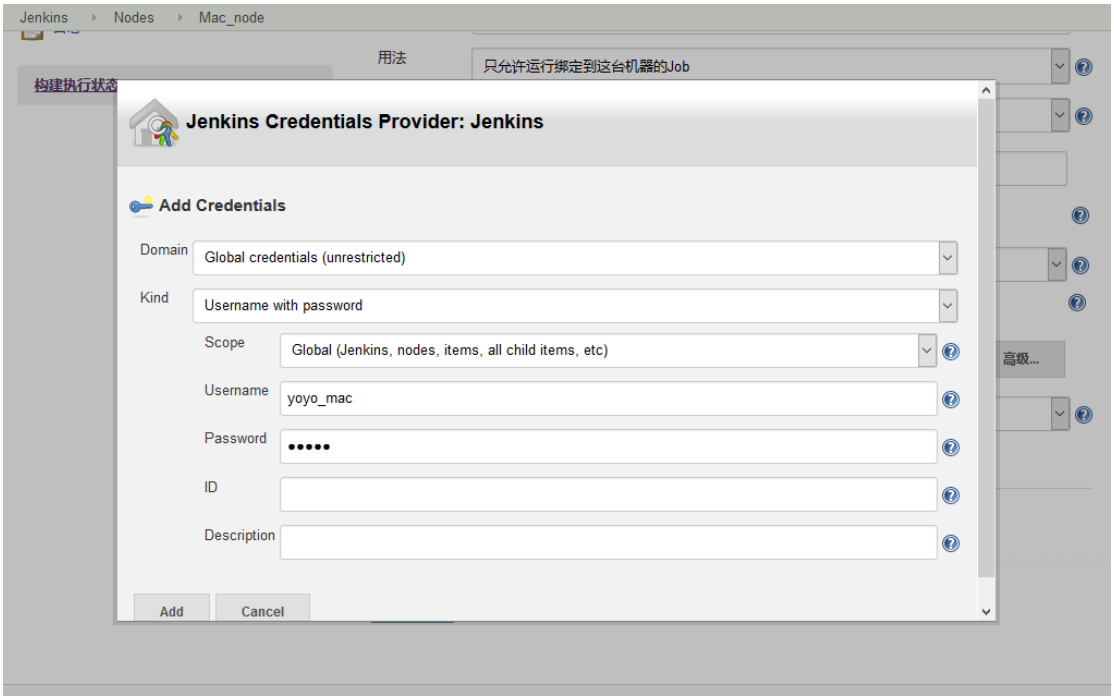
这里一定要注意权限，文件目录设为可读可写，要不然后面 jenkins 启动时候会遇到权限问题



标签 ：可写可不写
用法：只允许运行绑定到这台机器的 Job
启动方法：Launch slave agents via SSH
选 SSH 这一项就行了，有些版本是“Launch slave agents on Unix machine via SSH”
Host：Mac 机器的 Host 地址



Credentials：配置 Mac 机登录时候的用户名和密码

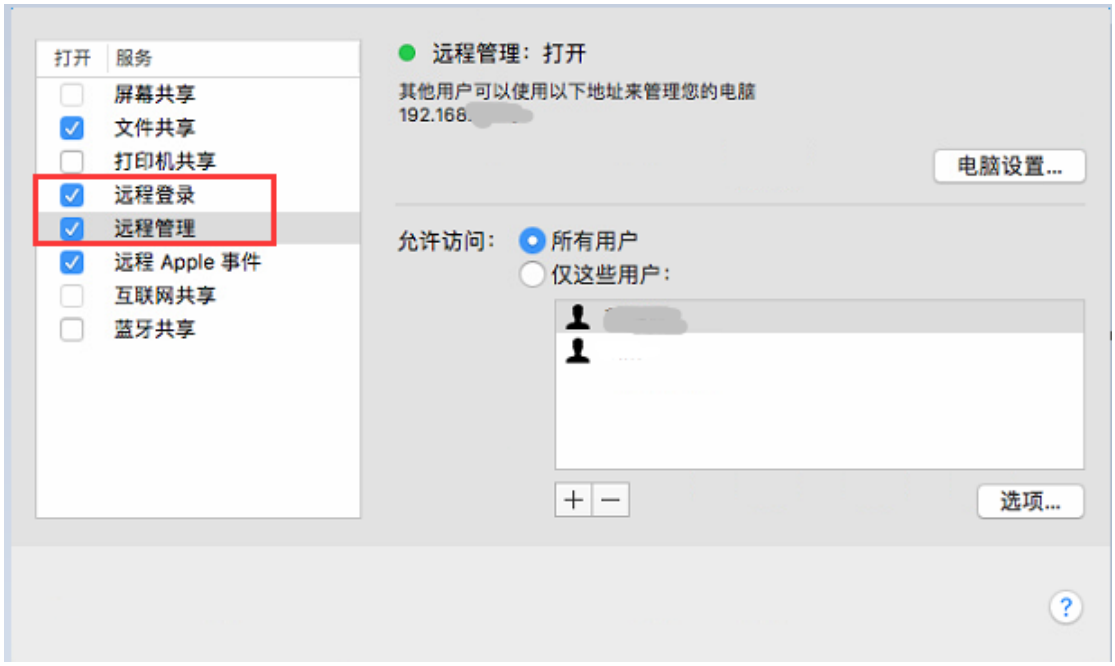


其它：其它的默认选项就行

7.10.2 Mac 上设置

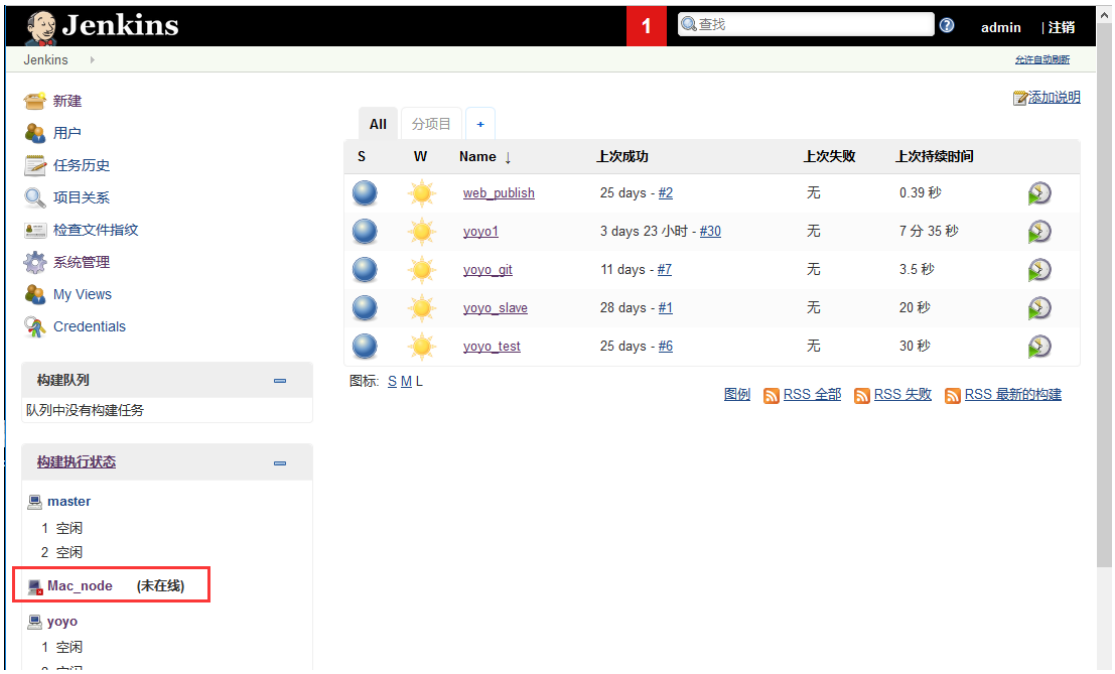
1.系统偏好设置，共享打开

2.把远程登录和远程管理选项勾选，设为所有用户可以访问

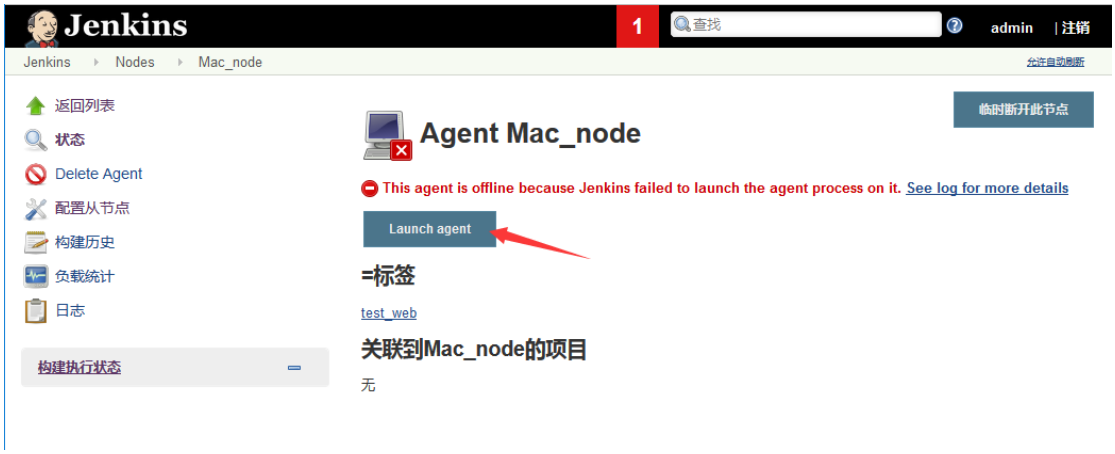


7.10.3 启动 Slave

前面第一步的操作设置好之后，这个节点是“未在线”状态



2.点开它，直接启动：Launch agent



好书推荐

学完此书觉得对你有帮助的，可以关注我的更多作品！

Selenium+python

要想学好 appium，必先学 selenium，这里推荐一本**葵花**和**上海-悠悠**一起写的 selenium+python 的书。

《selenium webdriver 基于 Python 源码案例》本书上架百度阅读以来，获得了非常好的评价，10 分好评，近五万的阅读量，是一本适合初学者的从入门到精通。

Python 接口自动化

第二本是本人写的《python 接口自动化》的书，虽然没 10 分好评，9.8 分也是获得了不错的评价，近 3w 的阅读量

Appium+python

第三本 Appium 的自动化，只能算个半成品，在百度阅读上持续更新，争取在 2018 年前能更新完!!!

《appium 自动化入门（图文教程）-python》

Python 接口测试开发

第四本《基于 Python 进行接口测试开发》，这本是葵花大神写的，这个没基础的同学不建议买，本书是适合 python 基础非常扎实，并且对接口也很熟悉，想往测试开发方向发展的同学，可以看看!!

关注小编！

更多自动化（**selenium、python 接口、appium**）扫码关注

个人微信公众号：yoyoketang (扫二维码关注)

个人博客地址：http://www.cnblogs.com/yoyoketang/

QQ 交流群：226296743