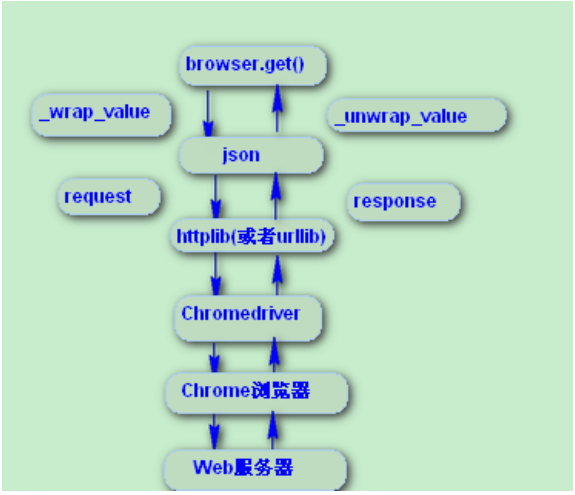
python Selenium原理分析

```
1  #coding=utf-8
2  #!/bin/env python
3
4  #selenium工作原理
5  from selenium import webdriver
6  browser = webdriver.Chrome()
7  browser.get("http://www.baidu.com")
8  browser.find_element_by_id("kw").send_keys("selenium")
9  browser.find_element_by_id("su").click()
10 browser.quit()
```

从命令到底层驱动的数据交互如下:

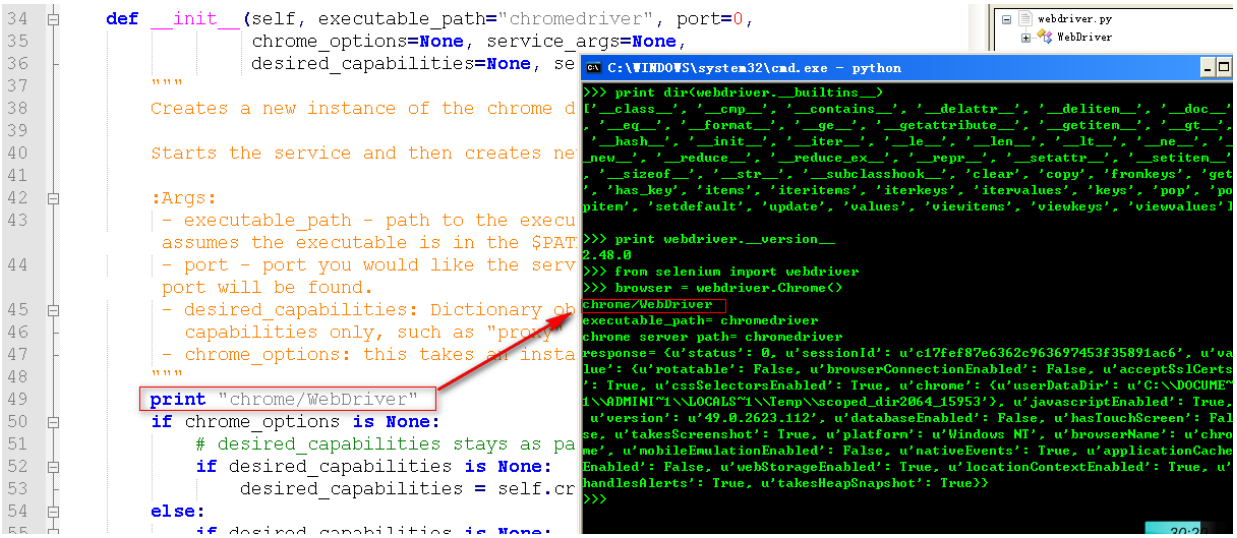


#代码第一行和第二行表示使用utf-8编码以及使用脚本为python
#代码第五行表示从selenium模块中导入webdriver模块
#获取webdriver类的文件路径
>>> print webdriver.__file__
C:\Python27\lib\site-packages\selenium\webdriver__init__.pyc

代码行from selenium import webdriver从selenium模块中导入
webdriver__init__.py 中再导入webdriver中的各个目录中的文件

```
18 from .firefox.webdriver import WebDriver as Firefox
19 from .firefox.firefox_profile import FirefoxProfile
20 from .chrome.webdriver import WebDriver as Chrome
21 from .chrome.options import Options as ChromeOptions
22 from .ie.webdriver import WebDriver as Ie
23 from .edge.webdriver import WebDriver as Edge
24 from .opera.webdriver import WebDriver as Opera
25 from .safari.webdriver import WebDriver as Safari
26 from .blackberry.webdriver import WebDriver as BlackBerry
27 from .phantomjs.webdriver import WebDriver as PhantomJS
28 from .android.webdriver import WebDriver as Android
29 from .remote.webdriver import WebDriver as Remote
30 from .common.desired_capabilities import DesiredCapabilities
31 from .common.action_chains import ActionChains
32 from .common.touch_actions import TouchActions
33 from .common.proxy import Proxy
34
35 __version__ = '2.48.0'
```

#代码行第六行 初始化一个浏览器对象browser=webdriver.Chrome()
#实际调用的是webdriver\chrome\webdriver.py中的__init__函数
#在__init__()函数开始添加打印信息print "chrome/WebDriver",如下图



#chrome\webdriver.py参数详解

```
def __init__(self, executable_path="chromedriver", port=0,
        chrome_options=None, service_args=None,
        desired_capabilities=None, service_log_path=None):
```

#executable_path=chromedriver 就是下载的浏览器驱动chrome浏览器使用的是chromedriver,由于这里没有使用绝对目录,因此需要将chromedriver.exe文件所在的路径加入到环境变量PATH中,即你在cmd输入chromedriver能显示以下信息表示已经正确添加

```
C:\Documents and Settings\Administrator>chromedriver
Starting ChromeDriver 2.18.343845 (73dd713ba7fbfb73cbb514e62641d8c96a94682a) on
port 9515
Only local connections are allowed.
```

chromedriver下载地址： https://sites.google.com/a/chromium.org/chromedriver/downloads

#port=0表示chromedriver开启时监听的接口为空闲接口(即在本PC上没有套接字使用的端口),

#查看本机哪些端口被占用使用

```
C:\Documents and Settings\Administrator>
C:\Documents and Settings\Administrator>chromedriver --port=135
Starting ChromeDriver 2.18.343845 (73dd713ba7fbfb73cbb514e62641d8c96a94682a) o
port 135
Only local connections are allowed.
[0.000][SEVERE]: bind() returned an error: 以一种访问权限不允许的方式做了一个试
问套接字的尝试。 (0x271D)
[0.000][WARNING]: Unknown error 10038 mapped to net::ERR_FAILED
Port not available. Exiting...

C:\Documents and Settings\Administrator>netstat -ano

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       1896
  TCP    0.0.0.0:443            0.0.0.0:0              LISTENING       2484
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:902            0.0.0.0:0              LISTENING       2776
  TCP    0.0.0.0:912            0.0.0.0:0              LISTENING       2776
  TCP    0.0.0.0:1039           0.0.0.0:0              LISTENING       1516
  TCP    0.0.0.0:1048           0.0.0.0:0              LISTENING       3356
```

使用135端口启动失败 因为该端口已经被占用了

#chrome_options=None: this takes an instance of ChromeOptions(表示调用的是ChromeOptions的一个对象实例)

#根据最初__init__.py导入from .chrome.options import Options as ChromeOptions可知

#实际上使用的就是webdriver\chrome\options 中class Options对象实例

#扩展学习 http://note.youdao.com/yws/public/redirect/share?id=5c2bcb548993daf49d743a97dab0c2fb&type=false

#chrome_options一般常用的是设置下载目录和修改浏览器代理,具体见下面的连接

#http://note.youdao.com/yws/public/redirect/share?id=99c94988d9f493c4aecd3f1dab01d5a8&type=false

#desired_capabilities=None

通过参数可以知道这个参数决定了调用的浏览器为chrome,而不是其他浏览器

与chrome_options类似 ,

```
CHROME = {
    "browserName": "chrome",
    "version": "",
    "platform": "ANY",
    "javascriptEnabled": True,
}
```

最后通过chrome = DesiredCapabilities.CHROME.copy()把里面的值拷贝到chrome_options中
#从后面的代码可以看出desired_capabilities 中包含更多的内容，而chrome_options只是其中的一个参数而已

```
desired_capabilities= {'platform': 'ANY', 'browserName': 'chrome', 'version': ''
, 'chromeOptions': {'args': [], 'extensions': []}, 'javascriptEnabled': True}
response= {u'status': 0, u'sessionId': u'f0e0b244d5676ef8ad4c627f35fd54a6', u'va
lue': {u'rotatable': False, u'browserConnectionEnabled': False, u'acceptSslCerts
': True, u'cssSelectorsEnabled': True, u'chrome': {u'userDataDir': u'C:\\DOCUME~
1\\ADMINI~1\\LOCALS~1\\Temp\\scoped_dir5788_13720'}, u'javascriptEnabled': True,
 u'version': u'49.0.2623.112', u'databaseEnabled': False, u'hasTouchScreen': Fal
se, u'takesScreenshot': True, u'platform': u'Windows NT', u'browserName': u'chro
me', u'mobileEmulationEnabled': False, u'nativeEvents': True, u'applicationCache
Enabled': False, u'webStorageEnabled': True, u'locationContextEnabled': True, u'
handlesAlerts': True, u'takesHeapSnapshot': True}}
```

参数准备好后就是打开chromedriver和chrome进程了

```
60          self.service = Service(executable_path, port=port,
61              service_args=service_args, log_path=service_log_path)
62          self.service.start()
63          # time.sleep()
64          try:
65              RemoteWebDriver.__init__(self,
66                  command_executor=ChromeRemoteConnection(
67                      remote_server_addr=self.service.service_url),
68                  desired_capabilities=desired_capabilities)
69          except:
70              self.quit()
71              raise
72          self._is_remote = False
```

代码60-61行表示打开赋值需要用到的参数给chromedriver.exe
代码62行表示运行chromedriver.exe(注意这里开启的不是chrome.exe 即不是打开的浏览器进程)
#打开chrome浏览器函数RemoteWebDriver.__init__()
#from selenium.webdriver.remote.webdriver import WebDriver as RemoteWebDriver
RemoteWebDriver.__init__

chromedriver打开后监听127.0.0.1:40228端口,当此套件字获取到webdriver发送的命令后,使用新的127.0.0.1:40235套接字作为客户端把数据发送而给127.0.0.1:12305(即chrome浏览器监听的客户端),chrome浏览器收到数据后会建立新的套接字发送request给Web服务器进行交互,最后获得从Web服务器的response后把数据发给chromedriver客户端(127.0.0.1:40235),chromedriver获取到数据后将其转发给webdriver,从而我们获取到了WEB服务器传回来的信息，至此完成数据交互

```
C:\Documents and Settings\Administrator>tasklist | findstr "chrome"
chromedriver.exe              4732 Console                    0      6,500 K
chrome.exe                    5744 Console                    0     75,060 K
chrome.exe                    5080 Console                    0      3,608 K
chrome.exe                    5780 Console                    0     17,624 K
chrome.exe                    4724 Console                    0     56,284 K

C:\Documents and Settings\Administrator>netstat -ano| findstr "4732 5744 5080 47
32"
  TCP    127.0.0.1:12305        0.0.0.0:0              LISTENING       5744
  TCP    127.0.0.1:12305        127.0.0.1:40235        ESTABLISHED     5744
  TCP    127.0.0.1:40228        0.0.0.0:0              LISTENING       4732
  TCP    127.0.0.1:40235        127.0.0.1:12305        ESTABLISHED     4732
... browser = webdriver.Chrome()
executable_path= chromedriver
self.port= 40228
command_executor= <selenium.webdriver.chrome.remote_connection.ChromeRemoteConne
ction object at 0x0101B1B0>
```

```
7   browser.get("http://www.baidu.com")
8   browser.find_element_by_id("kw").send_keys("selenium")
9   browser.find_element_by_id("su").click()
10  browser.quit()
```

代码第7-10行主要作用是通过webdriver接口发送命令给chromedriver 然后chromedriver将数据转给chrome浏览器，chrome浏览器发送请求给WEB服务器,WEB服务器响应后把信息发送给chrome浏览器,chrome浏览器收到数据后转发给chromedriver,chromedriver最后将数据传给webdriver,
browser.get("http://www.baidu.com")
#调用C:\Python27\Lib\site-packages\selenium\webdriver\remote\webdriver.py 中class WebDriver的get函数

```
    def get(self, url):
        """
        Loads a web page in the current browser session.
        """
        self.execute(Command.GET, {'url': url})
```

get函数又调用自己的execute函数,其中命令为driver_command=Command.GET params={'url' : url}
driver_command=Command.GET(Command.GET值为字符串"get",见C:\Python27\Lib\site-packages\selenium\webdriver\remote\command.py文件)，如下图

```
STATUS = "status"
NEW_SESSION = "newSession"
GET_ALL_SESSIONS = "getAllSessions"
DELETE_SESSION = "deleteSession"
CLOSE = "close"
QUIT = "quit"
GET = "get"
```

execute函数如下:

```
    def execute(self, driver_command, params=None):
        """
        Sends a command to be executed by a command.CommandExecutor.

        :Args:
         - driver_command: The name of the command to execute as a string.
         - params: A dictionary of named parameters to send with the command.

        :Returns:
          The command's JSON response loaded into a dictionary object.
        """
        if self.session_id is not None:
            if not params:
                params = {'sessionId': self.session_id}
            elif 'sessionId' not in params:
                params['sessionId'] = self.session_id

        params = self._wrap_value(params)
        response = self.command_executor.execute(driver_command, params)
        # print "response=",response
        if response:
            self.error_handler.check_response(response)
            response['value'] = self._unwrap_value(
                response.get('value', None))
            return response
        # If the server doesn't send a response, assume the command was
        # a success
        return {'success': 0, 'value': None, 'sessionId': self.session_id}
```

重要

execute函数先调用_wrap_value解析参数

```
    def _wrap_value(self, value):
        if isinstance(value, dict):
            converted = {}
            for key, val in value.items():
                converted[key] = self._wrap_value(val)
            return converted
        elif isinstance(value, WebElement):
            return {'ELEMENT': value.id,
            'element-6066-11e4-a52e-4f735466cecf': value.id}
        elif isinstance(value, list):
            return list(self._wrap_value(item) for item in value)
        else:
            return value
```

然后调用self.command_executor对象的execute函数
初始化时定义了self.command_executor对象
self.command_executor = RemoteConnection(command_executor, keep_alive=keep_alive)
self.command_executor.execute函数如下:
#此时的command=Command.GET (即字符串"get") params={"url":url}字典

```
def execute(self, command, params):
    command_info = self._commands[command]
    assert command_info is not None, 'Unrecognised command %s' % command
    data = utils.dump_json(params)
    path = string.Template(command_info[1]).substitute(params)
    url = '%s%s' % (self._url, path)
    print "url:",url
    return self._request(command_info[0], url, body=data)
```

```
>>> browser.get("http://www.baidu.com")
url: http://127.0.0.1:42024/session/3568827579d34bdca86116e962c41913/url
path: /session/3568827579d34bdca86116e962c41913/url
command_info= ('POST', '/session/$sessionId/url')
```

以下是urllib或者urllib2与chromedriver数据交互就不详细介绍了.

self._conn = httplib.HTTPConnection(
str(addr), str(parsed_url.port), timeout=self._timeout)

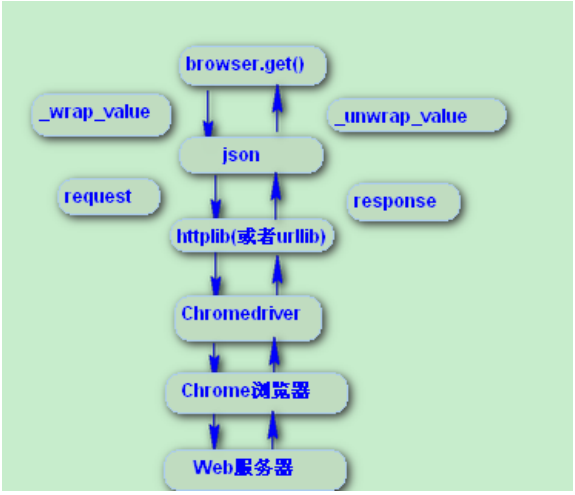
然后调用self._request函数--->调用self._conn.request-->self._conn.request

最后得到chromedriver的响应

大致函数调用如下:

browser.get--->webdriver.remote.Webdriver.get()-->webdriver.remote.Webdriver.command_executor.execute()-->webdriver.remote.remote_connection.execute()-->webdriver.remote.remote_connection._request()-->httplib.HTTPConnection()-->chromedriver-->chrome

最后得出数据交互图如下:



按照上图可以知道，如果调用远程的chrome浏览器,只需要在远程PC上安装chrome浏览器，然后运行chromedriver.exe,在本地端使用webdriver时command_executor命令使用远程的socket套接字就行

现在假如所有的脚本都在PC_A 上，然后要调用PC_B的Chrome浏览器进行测试(PC_A与PC_B数据必须可达)

PC_A (192.168.10.100) ----- PC_B(192.168.10.130)

1、下PC_B上开启chromedriver.exe 端口为9515 (命令行中输入chromedriver --port=9515)

2、在PC_A端代码如下

```
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
from selenium.webdriver.remote.webdriver import WebDriver as RemoteWebDriver
remoteurl='http://192.168.10.130:9515'
browser=
RemoteWebDriver(command_executor=remoteurl,desired_capabilities=DesiredCapabilities.CHROME,keep_alive=True)
driver.get('http://www.baidu.com')
```

by qudeyong