**快代理爬虫开发**

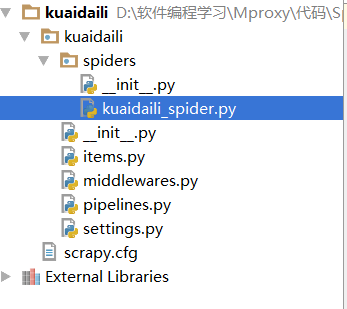
想清楚了怎么用Kafka，接下来我们开发项目中第一个真正实用的部分：快代理网站的爬虫。

**初次尝试**

使用下面的命令创建项目：

scrapy startproject kuaidaili

在spiders目录下面创建kuaidaili\_spider.py。一般来说爬虫的名字以域名或者网站名命名。



先写一个框架代码，类似下面这样：

#-\*- coding: utf-8 -\*-

import scrapy

class KuaidailiSpider(scrapy.spiders.Spider):

name = "kuaidaili"

allowed\_domains = ["kuaidaili.com"]

download\_delay = 1

start\_urls = []

def \_\_init\_\_(self):

for x in range(10):

self.start\_urls.append('http://www.kuaidaili.com/proxylist/%d/' % (x+1))

def parse(self, response):

pass

稍微解释一下。先说几个属性。

name代表爬虫的名字，这个是Scrapy框架指定必须要有的；

allowed\_domains代表spider允许爬取的域名(domain)列表(list)。 当 OffsiteMiddleware 启用时， 域名不在列表中的URL不会被跟进；

download\_delay表示每爬取一个页面等待的时间，避免太频繁的访问被封；

关于start\_urls，官方的说法是这样的：当没有制定特定的URL时，spider将从该列表中开始进行爬取。 因此，第一个被获取到的页面的URL将是该列表之一。 后续的URL将会从获取到的数据中提取。但是这个网站情况比较特殊，有价值的数据就在这10个页面里，所以我们都写上去，让爬虫爬取就是了。

然后接下来写取数据的代码。先试试把页面内容打印出来。代码如下：

#-\*- coding: utf-8 -\*-

import scrapy

class KuaidailiSpider(scrapy.spiders.Spider):

name = "kuaidaili"

allowed\_domains = ["kuaidaili.com"]

download\_delay = 1

start\_urls = []

def \_\_init\_\_(self):

for x in range(1):

self.start\_urls.append('http://www.kuaidaili.com/proxylist/%d/' % (x+1))

def parse(self, response):

print(response.body)

使用下面的命令运行爬虫：

scrapy crawl kuaidaili

并没有输出，查看日志，有如下输出：

2017-02-27 23:52:26 [scrapy.core.engine] DEBUG: Crawled (521) <GET http://www.kuaidaili.com/robots.txt> (referer: None)

2017-02-27 23:52:28 [scrapy.core.engine] DEBUG: Crawled (521) <GET http://www.kuaidaili.com/proxylist/1/> (referer: None)

看起来是被侦测出爬虫了。我们先把之前写过的User-Agent池配进去。

# -\*-coding:utf-8-\*-

from scrapy import log

"""避免被ban策略之一：使用useragent池。

使用注意：需在settings.py中进行相应的设置。

"""

import random

from scrapy.contrib.downloadermiddleware.useragent import UserAgentMiddleware

class RotateUserAgentMiddleware(UserAgentMiddleware):

def \_\_init\_\_(self, user\_agent=''):

self.user\_agent = user\_agent

def process\_request(self, request, spider):

ua = random.choice(self.user\_agent\_list)

if ua:

# Show current useragent

print "\*\*\*\*\*\*\*\*Current UserAgent:%s\*\*\*\*\*\*\*\*\*\*\*\*" %ua

# do the log

# log.msg('Current UserAgent: '+ua, level='INFO')

request.headers.setdefault('User-Agent', ua)

# the default user\_agent\_list composes chrome,I E,firefox,Mozilla,opera,netscape

# for more user agent strings,you can find it in http://www.useragentstring.com/pages/useragentstring.php

user\_agent\_list = [

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 "

"(KHTML, like Gecko) Chrome/22.0.1207.1 Safari/537.1",

"Mozilla/5.0 (X11; CrOS i686 2268.111.0) AppleWebKit/536.11 "

"(KHTML, like Gecko) Chrome/20.0.1132.57 Safari/536.11",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.6 "

"(KHTML, like Gecko) Chrome/20.0.1092.0 Safari/536.6",

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.6 "

"(KHTML, like Gecko) Chrome/20.0.1090.0 Safari/536.6",

"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.1 "

"(KHTML, like Gecko) Chrome/19.77.34.5 Safari/537.1",

"Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/536.5 "

"(KHTML, like Gecko) Chrome/19.0.1084.9 Safari/536.5",

"Mozilla/5.0 (Windows NT 6.0) AppleWebKit/536.5 "

"(KHTML, like Gecko) Chrome/19.0.1084.36 Safari/536.5",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",

"Mozilla/5.0 (Windows NT 5.1) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",

"Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_8\_0) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3",

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",

"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",

"Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 "

"(KHTML, like Gecko) Chrome/19.0.1061.0 Safari/536.3",

"Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/535.24 "

"(KHTML, like Gecko) Chrome/19.0.1055.1 Safari/535.24",

"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/535.24 "

"(KHTML, like Gecko) Chrome/19.0.1055.1 Safari/535.24"

]

在settings.py中进行如下配置：

#取消默认的useragent,使用新的useragent

DOWNLOADER\_MIDDLEWARES = {

'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware' : None,

'kuaidaili.rotate\_useragent.RotateUserAgentMiddleware' :400

}

还是不行。在网上找了找，可以在settings.py里面配置一下，忽略521错误。

HTTPERROR\_ALLOWED\_CODES = [521]

然后再次运行，得到了下面的结果：

我在浏览器里面试了一下，浏览器会自动跳到/?yundun=2dd18b5fbdf705420548这个地址上来。

虽然爬不成了，为什么感觉好奇妙（笑）

**抓取js生成的页面**

搜索了一下，需要用到scrapy-splash结合Scrapy来爬取JS的动态页面。

scrapy-splash基本的安装可以参考[http://www.cnblogs.com/zhonghuasong/p/5976003.html](http://www.cnblogs.com/zhonghuasong/p/5976003.html" \t "_blank)

docker的安装可以参考[http://www.linuxidc.com/Linux/2014-08/105656.htm](http://www.linuxidc.com/Linux/2014-08/105656.htm" \t "_blank)

使用下面的命令启动scrapy-splash：

docker run -d -p 8050:8050 --net=host scrapinghub/splash

可以使用一个简单的命令停止所有运行的容器：

docker stop $(docker ps)

配置splash服务，下面的修改都在settings.py中进行。

1.添加splash服务器地址：

SPLASH\_URL = 'http://localhost:8050'

2.将splash middleware添加到DOWNLOADER\_MIDDLEWARE中：

DOWNLOADER\_MIDDLEWARES = {

'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware' : None,

'kuaidaili.rotate\_useragent.RotateUserAgentMiddleware' :400,

'scrapy\_splash.SplashCookiesMiddleware': 723,

'scrapy\_splash.SplashMiddleware': 725,

'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware': 810,

}

3.Enable SplashDeduplicateArgsMiddleware:

SPIDER\_MIDDLEWARES = {

'scrapy\_splash.SplashDeduplicateArgsMiddleware': 100,

}

4.Set a custom DUPEFILTER\_CLASS:

DUPEFILTER\_CLASS = 'scrapy\_splash.SplashAwareDupeFilter'

5.a custom cache storage backend:

HTTPCACHE\_STORAGE = 'scrapy\_splash.SplashAwareFSCacheStorage'

把爬虫的代码稍微修改一下：

def start\_requests(self):

for i, url in enumerate(self.start\_urls):

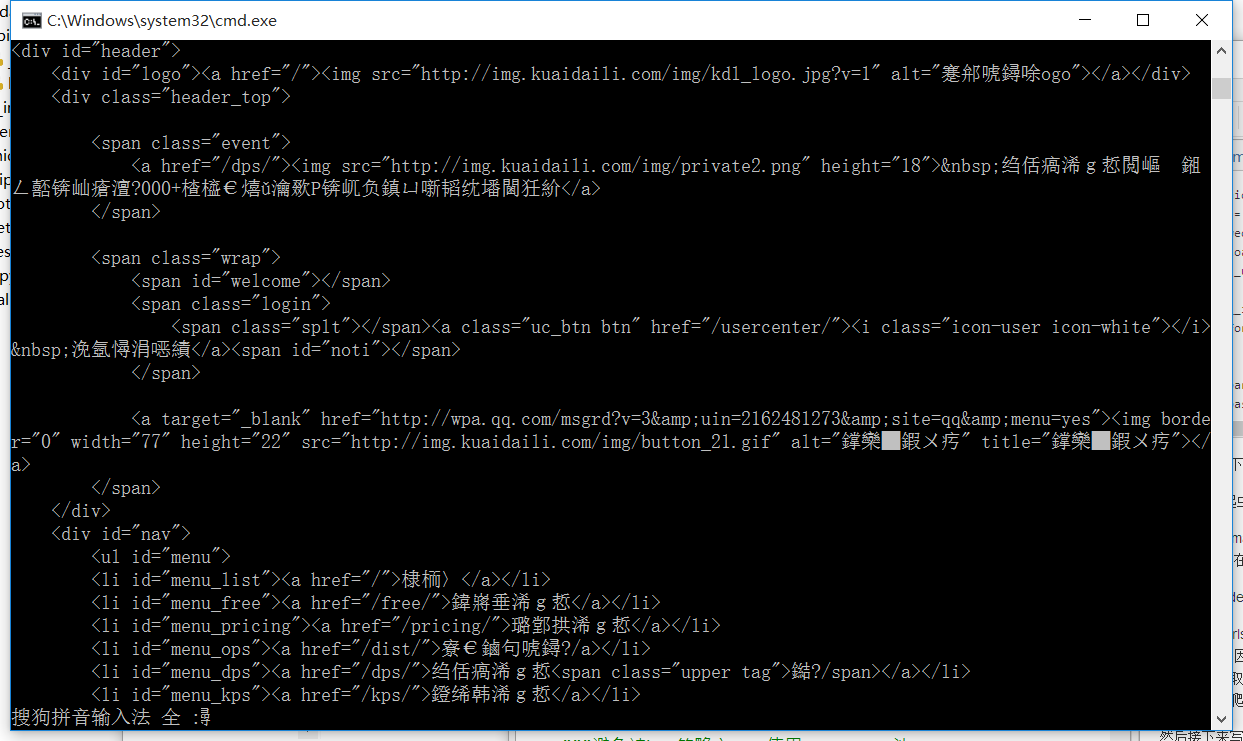
# yield Request(url, meta={'cookiejar': i},

# headers=self.headers,

# callback=self.parse)

yield SplashRequest(url, self.parse, args={'wait': 0.5})

然后再次运行爬虫。



可以看出虽然有乱码，不过已经拿到数据了。Great！

**关于docker中的splash的配置**

其实我上面的代码在运行时出现过多次time-out错误。在http://amaster:8050/调试的时候连百度都访问不了。

为此我做了几件事，不知道是哪个起作用了。

1.修改了/etc/NetworkManager/NetworkManager.conf文件，将dns=dnsmasq注释掉。

#dns=dnsmasq

restart network-manager

2.重装了docker，重新下载了splash镜像

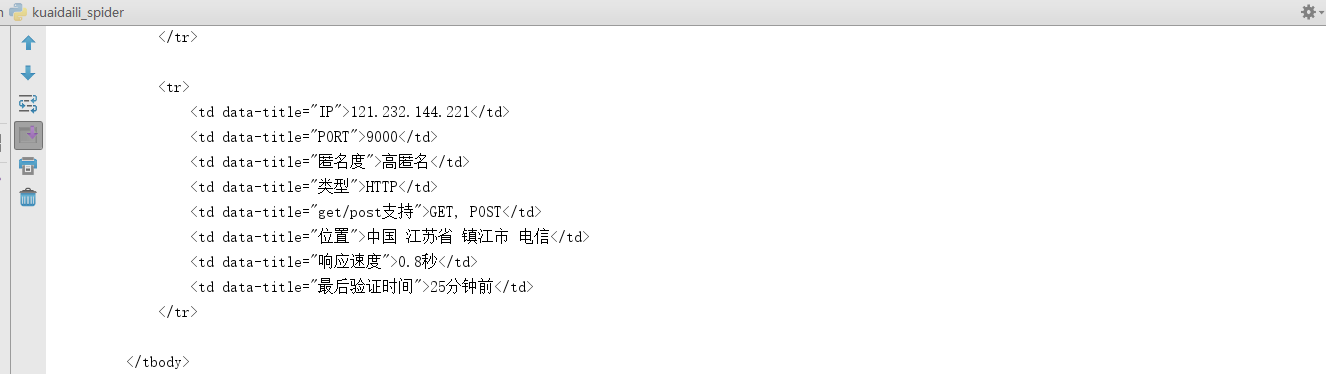
3.调整了启动splash的命令（上文已经修改了）：

docker run -d -p 8050:8050 --net=host scrapinghub/splash

**解决乱码问题**

调试了半天，觉得不应该乱码，因为各种编码都是utf-8都是匹配的啊。

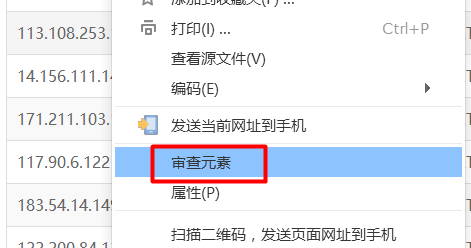
最后把运行环境配到PyCharm来了（参考[https://my.oschina.net/xueba/blog/492948](https://my.oschina.net/xueba/blog/492948" \t "_blank)），果然输出不是乱码。



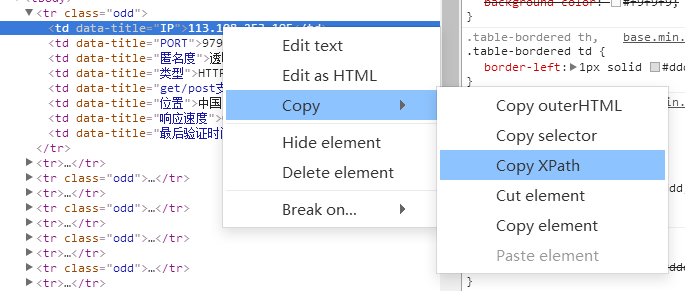
**提取数据**

忙了好久，终于可以提取数据了。

在Chrome的调试工具里又发现神器了。居然可以直接把xpath复制出来。。。



第一步，在要提取的元素上面右键->审查元素



第二步，在选中的元素上面右键，复制xpath即可

第三步，提取数据

def parse(self, response):

ip1 = response.xpath('''//\*[@id="index\_free\_list"]/table/tbody/tr[1]/td[1]/text()''').extract\_first()

print(ip1)

提取结果为：

'113.108.253.195'

不过我们还是稍作修改再使用。

def parse(self, response):

trs = response.xpath('''//\*[@id="index\_free\_list"]/table/tbody/tr''')

for tr\_selector in trs:

ip = tr\_selector.xpath('''./td[1]/text()''').extract\_first()

port = tr\_selector.xpath('''./td[2]/text()''').extract\_first()

anonymity = tr\_selector.xpath('''./td[3]/text()''').extract\_first()

type = tr\_selector.xpath('''./td[4]/text()''').extract\_first()

location = tr\_selector.xpath('''./td[6]/text()''').extract\_first()

item = [ip,port,anonymity,type,location]

print('\t'.join(item))

输出结果如下：

117.90.4.140 9000 高匿名 HTTP 中国 江苏省 镇江市 电信

117.90.1.87 9000 高匿名 HTTP 中国 江苏省 镇江市 电信

218.17.252.34 3128 匿名 HTTP 中国 广东省 深圳市 电信

112.74.72.1 8080 高匿名 HTTP, HTTPS 中国 广东省 深圳市 阿里云

124.230.150.214 8998 高匿名 HTTP, HTTPS 中国 湖南省 邵阳市 电信

121.232.148.129 9000 高匿名 HTTP 中国 江苏省 镇江市 电信

222.220.35.82 3128 透明 HTTP 云南省西双版纳傣族自治州 电信

183.54.244.204 9000 透明 HTTP, HTTPS 中国 广东省 广州市 电信

113.116.118.44 8998 高匿名 HTTP, HTTPS 中国 广东省 深圳市 电信

113.108.253.195 9797 透明 HTTP, HTTPS 中国 广东省 东莞市 电信

好，终于拿到了数据。

**输出结果到日志文件**

之前我们在需求里面提到过，需要将爬取过程记录到日志文件中以备今后的分析使用。接下来我们来实现这个需求。

这部分我们需要用到Python的logging模块，关于logging模块的使用，可以参考[http://www.cnblogs.com/dkblog/archive/2011/08/26/2155018.html](http://www.cnblogs.com/dkblog/archive/2011/08/26/2155018.html" \t "_blank)。

接下来上代码。

#-\*- coding: utf-8 -\*-

import logging

import uuid

from logging.handlers import RotatingFileHandler

import sys

reload(sys)

sys.setdefaultencoding('gbk')

import scrapy

from scrapy\_splash import SplashRequest

from kuaidaili.items import KuaidailiItem

class KuaidailiSpider(scrapy.spiders.Spider):

name = "kuaidaili"

allowed\_domains = ["kuaidaili.com"]

download\_delay = 1

start\_urls = []

def \_\_init\_\_(self):

for x in range(1):

self.start\_urls.append('http://www.kuaidaili.com/proxylist/%d/' % (x+1))

# self.start\_urls.append('http://www.baidu.com')

self.headers = {

"Host": "www.kuaidaili.com",

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

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

"Accept-Encoding": "gzip, deflate",

"Referer": "http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1521-3773",

"Cookie": "\_gat=1; channelid=0; sid=1488211261856538; \_ga=GA1.2.167063512.1488083088; Hm\_lvt\_7ed65b1cc4b810e9fd37959c9bb51b31=1488083088,1488174082,1488205850,1488211463; Hm\_lpvt\_7ed65b1cc4b810e9fd37959c9bb51b31=1488211463",

"Connection": "keep-alive",

"Upgrade - Insecure - Requests": "1"

}

self.init\_log()

def init\_log(self):

# add log ratate

Rthandler = RotatingFileHandler('kuaidaili\_spider.log', maxBytes=10 \* 1024 \* 1024, backupCount=100,encoding = "gbk")

Rthandler.setLevel(logging.INFO)

formatter = logging.Formatter('%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s')

Rthandler.setFormatter(formatter)

logging.getLogger().addHandler(Rthandler)

def start\_requests(self):

for i, url in enumerate(self.start\_urls):

yield SplashRequest(url, self.parse, args={'wait': 1})

def parse(self, response):

try:

trs = response.xpath('''//\*[@id="index\_free\_list"]/table/tbody/tr''')

for tr\_selector in trs:

item = KuaidailiItem()

item['ip'] = tr\_selector.xpath('''./td[1]/text()''').extract\_first()

item['port'] = tr\_selector.xpath('''./td[2]/text()''').extract\_first()

item['anonymity'] = tr\_selector.xpath('''./td[3]/text()''').extract\_first()

item['type'] = tr\_selector.xpath('''./td[4]/text()''').extract\_first()

item['location'] = tr\_selector.xpath('''./td[6]/text()''').extract\_first()

item['task\_id'] = str(uuid.uuid4())

yield item

except Exception as e:

logging.exception("An Error Happend")

虽然有点重复，不过为了看清楚，把所有的代码贴上来。重点是关注init\_log这个方法。在这里面我们添加了一个带滚动的日志文件记录器。然后在pause中使用了它。如果解析过程出了任何错误，它将会把exception的traceback信息记录到日志文件中，然后继续执行。一方面方便排错，另一方面也做到之前提过的运行过程中无论出什么问题都不能崩溃的需求。

还有一个重点是在import部分。

import sys

reload(sys)

sys.setdefaultencoding('gbk')

如果不加上这几句，因为系统默认的编码是ascii，如果traceback信息中含有中文(很不幸，我的路径有中文)就会报错，导致tranceback信息无法写入到日志文件。

然后再来看pipelines中的使用。

class KuaidailiKafkaPipeline(object):

def process\_item(self, item, spider):

log\_msg = "Get a proxy[%(ip)s\t%(port)s\t%(anonymity)s\t%(type)s\t%(location)s]. Task Id is:%(task\_id)s" % item

logging.info(log\_msg)

return item

很简单，只是记录了获取到了一个代理服务器。

**提交数据到Kafka集群**

历经千辛万苦，我们终于拿到了数据，接下来需要把数据提交到Kafka集群中，然后就会有验证器负责校验代理服务器的有效性。

class KuaidailiKafkaPipeline(object):

def \_\_init\_\_(self):

self.producer = KafkaProducer(value\_serializer=lambda m: json.dumps(m).encode('utf-8'),

bootstrap\_servers=['amaster:9092', 'anode1:9092', 'anode2:9092'])

def \_\_del\_\_(self):

if self.producer is not None:

self.producer.close()

def process\_item(self, item, spider):

try:

log\_msg = "Get a proxy[%(ip)s\t%(port)s\t%(anonymity)s\t%(type)s\t%(location)s]. Task Id is:%(task\_id)s" % item

logging.info(log\_msg)

self.producer.send('unchecked-servers', item.\_\_dict\_\_) # Makes the item could be JSON serializable

except Exception as e:

logging.exception("An Error Happens")

return item

经过修改后的pipelines的代码如上所示。有一点需要注意一下self.producer.send('unchecked-servers', item.\_\_dict\_\_)中间的那个\_\_dict\_\_是为了把对象转成json。可以参考[http://stackoverflow.com/questions/10252010/serializing-python-object-instance-to-json](http://stackoverflow.com/questions/10252010/serializing-python-object-instance-to-json" \t "_blank)

然后使用下面的命令启动一个Kafka消费者用来收消息：

/root/apps/kafka\_2.12-0.10.2.0/bin/kafka-console-consumer.sh --bootstrap-server amaster:9092 --topic unchecked-servers --from-beginning

运行爬虫后，consumer收到如下的消息：

{"\_values": {"task\_id": "e1c73c0e-c1ee-4db7-9596-e328ed6a662c", "ip": "117.90.6.191", "location": "\u4e2d\u56fd \u6c5f\u82cf\u7701 \u9547\u6c5f\u5e02 \u7535\u4fe1", "anonymity": "\u9ad8\u533f\u540d", "type": "HTTP", "port": "9000"}}

{"\_values": {"task\_id": "f6c56876-a382-4b9c-b65d-2e72b8c99795", "ip": "27.46.37.226", "location": "\u4e2d\u56fd \u5e7f\u4e1c\u7701 \u6df1\u5733\u5e02 \u8054\u901a", "anonymity": "\u900f\u660e", "type": "HTTP, HTTPS", "port": "9797"}}

{"\_values": {"task\_id": "86e09244-7804-42e1-8af3-93197ebb1245", "ip": "117.90.6.201", "location": "\u4e2d\u56fd \u6c5f\u82cf\u7701 \u9547\u6c5f\u5e02 \u7535\u4fe1", "anonymity": "\u9ad8\u533f\u540d", "type": "HTTP", "port": "9000"}}

{"\_values": {"task\_id": "e9e973ba-c401-40cd-a3fd-72470cf0e8a9", "ip": "103.198.131.13", "location": "\u4e2d\u56fd \u5e7f\u4e1c\u7701 \u5e7f\u5dde\u5e02 \u9e4f\u535a\u58eb", "anonymity": "\u900f\u660e", "type": "HTTP, HTTPS", "port": "8123"}}

{"\_values": {"task\_id": "8c986947-1db4-4000-b3dd-19a3476bf726", "ip": "117.90.1.244", "location": "\u4e2d\u56fd \u6c5f\u82cf\u7701 \u9547\u6c5f\u5e02 \u7535\u4fe1", "anonymity": "\u9ad8\u533f\u540d", "type": "HTTP", "port": "9000"}}

{"\_values": {"task\_id": "d87e7e03-9cdc-4f84-bc6e-d7e492b923f8", "ip": "183.245.148.131", "location": "\u4e2d\u56fd \u6d59\u6c5f\u7701 \u4e3d\u6c34\u5e02 \u79fb\u52a8", "anonymity": "\u9ad8\u533f\u540d", "type": "HTTP", "port": "80"}}

{"\_values": {"task\_id": "9323a6d6-5f1f-4539-a538-0cdc15e23ab8", "ip": "218.86.60.18", "location": "\u4e2d\u56fd \u798f\u5efa\u7701 \u8386\u7530\u5e02 \u7535\u4fe1", "anonymity": "\u900f\u660e", "type": "HTTP, HTTPS", "port": "808"}}

{"\_values": {"task\_id": "4932f49c-17a6-49dc-9b7d-95a93cf865f3", "ip": "182.88.228.159", "location": "\u5e7f\u897f\u58ee\u65cf\u81ea\u6cbb\u533a\u5357\u5b81\u5e02 \u8054\u901a", "anonymity": "\u9ad8\u533f\u540d", "type": "HTTP, HTTPS", "port": "8123"}}

{"\_values": {"task\_id": "a1fe160d-a566-49fc-bdb0-c1afbed8f15d", "ip": "117.90.7.99", "location": "\u4e2d\u56fd \u6c5f\u82cf\u7701 \u9547\u6c5f\u5e02 \u7535\u4fe1", "anonymity": "\u9ad8\u533f\u540d", "type": "HTTP", "port": "9000"}}

{"\_values": {"task\_id": "0757ecc2-d924-414e-82ac-69ebcc45d53f", "ip": "182.85.254.137", "location": "\u4e2d\u56fd \u6c5f\u897f\u7701 \u5357\u660c\u5e02 \u7535\u4fe1", "anonymity": "\u9ad8\u533f\u540d", "type": "HTTP, HTTPS", "port": "8123"}}