浏览器分析

useragent

这里指的是，软件按照一定的格式向远端的服务器提供一个标识自己的字符串。
在HTTP协议中，使用user-agent字段传送这个字符串。

注意：这个值可以被修改

格式

```
现在浏览器的user-agent值格式一般如下：
Mozilla/[version] ([system and browser information]) [platform] ([platform details])
[extensions]

例如：
Chrome
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/57.0.2987.133 Safari/537.36

Firefox
Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0

IE
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
```

信息提取

pyyaml、ua-parser、user-agents模块。

安装

```
$ pip install pyyaml ua-parser user-agents
```

使用

```
from user_agents import parse

useragents = [
    "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \
Chrome/57.0.2987.133 Safari/537.36",
    "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0",
    "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0",
    "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0; SLCC2; \
.NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C;
.NET4.0E)"
```

```
]

for uastring in useragents:
    ua = parse(uastring)
    print(ua.browser, ua.browser.family, ua.browser.version, ua.browser.version_string)

# 运行结果
Browser(family='Chrome', version=(57, 0, 2987), version_string='57.0.2987') Chrome (57, 0, 2987) 57.0.2987
Browser(family='Firefox', version=(56, 0), version_string='56.0') Firefox (56, 0) 56.0
Browser(family='Firefox', version=(52, 0), version_string='52.0') Firefox (52, 0) 52.0
Browser(family='IE', version=(10, 0), version_string='10.0') IE (10, 0) 10.0
```

ua.browser.family和ua.browser.version_string分别返回浏览器名称、版本号。

数据分析

conversion 增加对useragent的处理

```
from user_agents import parse
conversion = {
    'datetime': lambda timestr: datetime.datetime.strptime(timestr, '%d/%b/%Y:%H:%M:%S %z'),
    'status': int,
    'length': int,
    'useragent': lambda ua: parse(ua)
}
```

增加浏览器分析函数

```
# 浏览器分析
def browser_handler(iterable):
    browsers = {}
    for item in iterable:
        ua = item['useragent']

        key = (ua.browser.family, ua.browser.version_string)
        browsers[key] = browsers.get(key, 0) + 1
    return browsers
```

注册handler，注意时间窗口宽度

```
reg(browser_handler, 5, 5)
```

问题
如果想知道所有浏览器的统计，怎么办？

```
# 浏览器分析
allbrowsers = {}

def browser_handler(iterable):
    browsers = {}
    for item in iterable:
        ua = item['useragent']
        key = (ua.browser.family, ua.browser.version_string)

        browsers[key] = browsers.get(key, 0) + 1
        allbrowsers[key] = allbrowsers.get(key, 0) + 1
    #print(sorted(allbrowsers.items(), key=lambda x: x[1], reverse=True)[:10])
    return browsers
```

完整代码

```
import random
import datetime
import time
from queue import Queue
import threading
import re

# 日志处理正则
pattern = '''(?P<remote>[\d.]{7,}) - - \[(?P<datetime>[\w/: +]+)\] \
"(?P<method>\w+) (?P<url>\S+) (?P<protocol>[\w\d/.]+)" (?P<status>\d+) (?P<length>\d+) \
".+" "(?P<useragent>.+)"'''
# 编译
regex = re.compile(pattern)

from user_agents import parse
conversion = {
    'datetime': lambda timestr: datetime.datetime.strptime(timestr, '%d/%b/%Y:%H:%M:%S %z'),
    'status': int,
    'length': int,
    'useragent': lambda ua: parse(ua)
}

def extract(logline: str) -> dict:
    """返回字段的字典，如果返回None说明匹配失败"""
    m = regex.match(logline)
    if m:
        return {k:conversion.get(k, lambda x:x)(v) for k,v in m.groupdict().items()}
    else:
        return None # 或输出日志记录

# 装载日志数据，数据源
from pathlib import Path

def loadfile(filename:str, encoding='utf-8'):
    """装载日志文件"""
```

```
    with open(filename, encoding=encoding) as f:
        for line in f:
            fields = extract(line)
            if isinstance(fields, dict):
                yield fields
            else:
                continue # TODO 解析失败就抛弃，或者打印日志

def load(*paths, encoding='utf-8', ext="*.log", glob=False):
    """装载日志文件"""
    for p in paths:
        path = Path(p)
        if path.is_dir(): # 只处理目录
            if isinstance(ext, str):
                ext = [ext]
            else:
                ext = list(ext)

            for e in ext:  # 按照扩展名递归
                files = path.rglob(e) if glob else path.glob(e)  # 是否递归
                for file in files:
                    yield from loadfile(str(file.absolute()), encoding=encoding)
        elif path.is_file():
            yield from loadfile(str(path.absolute()), encoding=encoding)



def window(src: Queue, handler, width: int, interval: int):
    """窗口函数

    :param iterator: 数据源，生成器，用来拿数据
    :param handler: 数据处理函数
    :param width: 时间窗口宽度，秒
    :param interval: 处理时间间隔，秒
    """
    if interval > width:  # width < interval不处理
        return

    start = datetime.datetime.strptime('20170101 000000 +0800', '%Y%m%d %H%M%S %z')
    current = datetime.datetime.strptime('20170101 010000 +0800', '%Y%m%d %H%M%S %z')
    buffer = []  # 窗口中的待计算数据
    delta = datetime.timedelta(seconds=width - interval)

    while True:
        # 从数据源获取数据
        data = src.get()
        if data:  # 攒数据
            buffer.append(data)  # 存入临时缓冲等待计算
            current = data['datetime']

        # 每隔interval计算buffer中的数据一次
        if (current - start).total_seconds() >= interval:
            ret = handler(buffer)
```

```
            print('{}'.format(ret))
            start = current

            # 保留buffer中未超出width的数据。如果delta为0，说明width等于interval，buffer直接清空
            buffer = [x for x in buffer if x['datetime'] > current - delta] if delta else []


# 处理函数，送入一批数据计算出一个结果，下为平均值
def handler(iterable):
    return sum(map(lambda x: x['value'], iterable)) / len(iterable)

# 测试函数
def donothing_handler(iterable):
    return iterable

# 状态码占比
def status_handler(iterable):
    # 时间窗口内的一批数据
    status = {}
    for item in iterable:
        key = item['status']
        status[key] = status.get(key, 0) + 1
    #total = sum(status.values())
    total = len(iterable)
    return {k:v/total for k,v in status.items()}

# 浏览器分析
allbrowsers = {}

def browser_handler(iterable):
    browsers = {}
    for item in iterable:
        ua = item['useragent']
        key = (ua.browser.family, ua.browser.version_string)

        browsers[key] = browsers.get(key, 0) + 1
        allbrowsers[key] = allbrowsers.get(key, 0) + 1
    print(sorted(allbrowsers.items(), key=lambda x: x[1], reverse=True)[:10])
    return browsers

def dispatcher(src):
    # 分发器中记录handler，同时保存各自的队列
    handlers = []
    queues = []

    def reg(handler, width: int, interval: int):
        """注册 窗口函数

        :param handler: 注册的数据处理函数
        :param width: 时间窗口宽度
        :param interval: 时间间隔
        """
        q = Queue()  # 每一个handler自己的数据源queue
```

```
        queues.append(q)

        # 每一个handler都运行在单独的线程中
        t = threading.Thread(target=window, args=(q, handler, width, interval))
        handlers.append(t)

    def run():
        for t in handlers:
            t.start()  # 启动线程，运行所有的处理函数

        for item in src:  # 将数据源取到的数据分发到所有队列中
            for q in queues:
                q.put(item)

    return reg, run


if __name__ == '__main__':
    import sys
    #path = sys.argv[1]
    path = 'test.log'

    reg, run = dispatcher(load(path))

    reg(status_handler, 10, 5)  # 注册
    reg(browser_handler, 5, 5)
    run()  # 运行
```