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; .N
ET4.0C; .NET4.0E)"
]

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

# 运行结构
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) 5
6.0
Browser(family='Firefox', version=(52, 0), version_string='52.0') Firefox (52, 0) 5
2.0
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

ops 增加对useragent的处理

```
from user_agents import parse
ops = {
    'datetime': lambda timestr: datetime.datetime.strptime(timestr, '%d/%b/%Y:%H:%M
:%S %z'),
```

```
    'status': int,
    'length': int,
    'request':lambda request:dict(zip(('method','url','protocol'),request.split()))
,
    'useragent':lambda useragent: parse(useragent)
}


from user_agents import parse
ops = {
    'datetime':lambda datestr: datetime.datetime.strptime(datestr, '%d/%b/%Y:%H:%M:
%S %z'),
    'status':int,
    'size':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？

```
allbrowsers = {}

# 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
        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
from pathlib import Path

# 数据源
PATTERN = '''(?P<remote>[\d\.]{7,})\s-\s-\s\[(?P<datetime>[^\[\]]+)\]\s\
"(?P<method>.*)\s(?P<url>.*)\s(?P<protocol>.*)"\s\
(?P<status>\d{3})\s(?P<size>\d+)\s"[^"]+"\s"(?P<useragent>[^"]+)"'''

regex = re.compile(PATTERN) # 编译
from user_agents import parse
ops = {
    'datetime':lambda datestr: datetime.datetime.strptime(datestr, '%d/%b/%Y:%H:%M:
%S %z'),
    'status':int,
    'size':int,
    'useragent':lambda ua: parse(ua)
}

def extract(line:str) -> dict:
    matcher = regex.match(line)
```

```
    if matcher:
        return {name:ops.get(name, lambda x: x)(data) for name, data in matcher.gro
updict().items()}

# 装载文件
def openfile(path:str):
    with open(path) as f:
        for line in f:
            fields = extract(line)
            if fields:
                yield fields
            else:
                continue  # TODO 解析失败则抛弃或者记录日志

def load(*paths):
    for item in paths:
        p = Path(item)
        if not p.exists():
            continue
        if p.is_dir():
            for file in p.iterdir():
                if file.is_file():
                    yield from openfile(str(file))
        elif p.is_file():
            yield from openfile(str(p))


# 数据处理
def source(second=1):
    """生成数据"""
    while True:
        yield {
            'datetime':datetime.datetime.now(datetime.timezone(datetime.timedelta(h
ours=8))),
            'value':random.randint(1,100)
        }
        time.sleep(second)

# 滑动窗口函数
def window(src:Queue, handler, width:int, interval:int):
```

```
    """
    窗口函数

    :param src: 数据源，缓存队列，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
    """

    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:
            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

            # 清除超出width的数据
            buffer = [x for x in buffer if x['datetime'] > current - delta]

# 随机数平均数测试函数
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
```

```python
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:status[k]/total for k,v in status.items()}

allbrowsers = {}

# 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
        allbrowsers[key] = allbrowsers.get(key, 0) + 1

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

# 分发器
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 窗口处理函数

        :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()
```

```
        queues.append(q)

        h = threading.Thread(target=window, args=(q, handler, width, interval))
        handlers.append(h)

    def run():
        for t in handlers:
            t.start() # 启动线程处理数据

        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
```