整合代码

load函数就是从日志中提取合格的数据的生成器函数。
它可以作为dispatcher函数的数据源。
原来写的handler函数处理一个字典的'datetime'字段，不能处理日志抽取函数extract返回的字典，提供一个新的函数。

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

# 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

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

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()}


def load(path):
    """装载日志文件"""
    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


# 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

# 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(donothing_handler, 10, 5) # 注册处理函数
    run() # 运行
```

完成分析功能

分析日志很重要，通过海量数据分析就能够知道是否遭受了攻击，是否被爬取及爬取高峰期，是否有盗链等。
百度(Baidu) 爬虫名称(Baiduspider)
谷歌(Google) 爬虫名称(Googlebot)

状态码分析

状态码中包含了很多信息。例如
304，服务器收到客户端提交的请求参数，发现资源未变化，要求浏览器使用静态资源的缓存
404，服务器找不大请求的资源
304占比大，说明静态缓存效果明显。404占比大，说明网站出现了错误链接，或者尝试嗅探网站资源。
如果400、500占比突然开始增大，网站一定出问题了。

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

如果还需要什么分析，增加分析函数handler注册就行了

日志文件的加载

目前实现的代码中，只能接受一个路径，修改为接受一批路径。

可以约定一下路径下文件的存放方式：
如果送来的是一批路径，就迭代其中路径。
如果路径是一个普通文件，就按照行读取内容。
如果路径是一个目录，就遍历路径下所有普通文件，每一个文件按照行处理。不递归处理子目录。

```
from pathlib import Path

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():
                    pass # 和下面处理一样
        elif p.is_file():
            with open(str(p)) 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
```

写的过程中发现重复的地方，把文件处理部分提出来写成函数。

```
from pathlib import Path

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))
```


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

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

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
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()}


# 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
    run() # 运行
```


到这里，一个离线日志分析项目基本完成。

1、可以指定文件或目录，对日志进行数据分析
2、分析函数可以动态注册
3、数据可以分发给不同的分析处理程序处理