程柳锋　　2017.7.12

PYTHON编写知乎爬虫实践

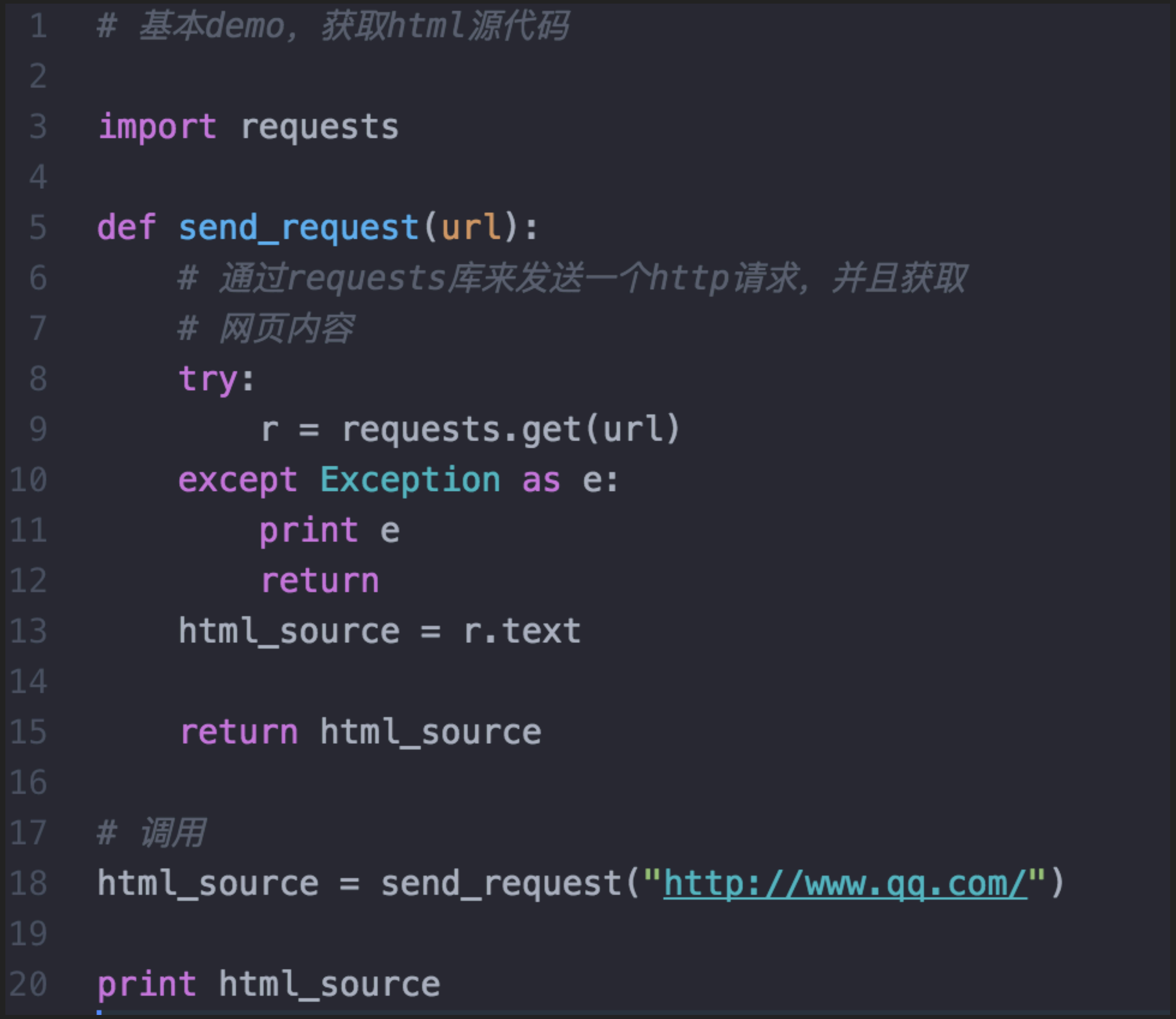
个人介绍

- 2014.12~2017.2 淘宝，闲鱼拍卖
- 2017.2 ～ 至今 腾讯，Now直播
- 博客：http://www.cnblogs.com/cpselvis
- Github: https://github.com/cpselvis

分享大纲

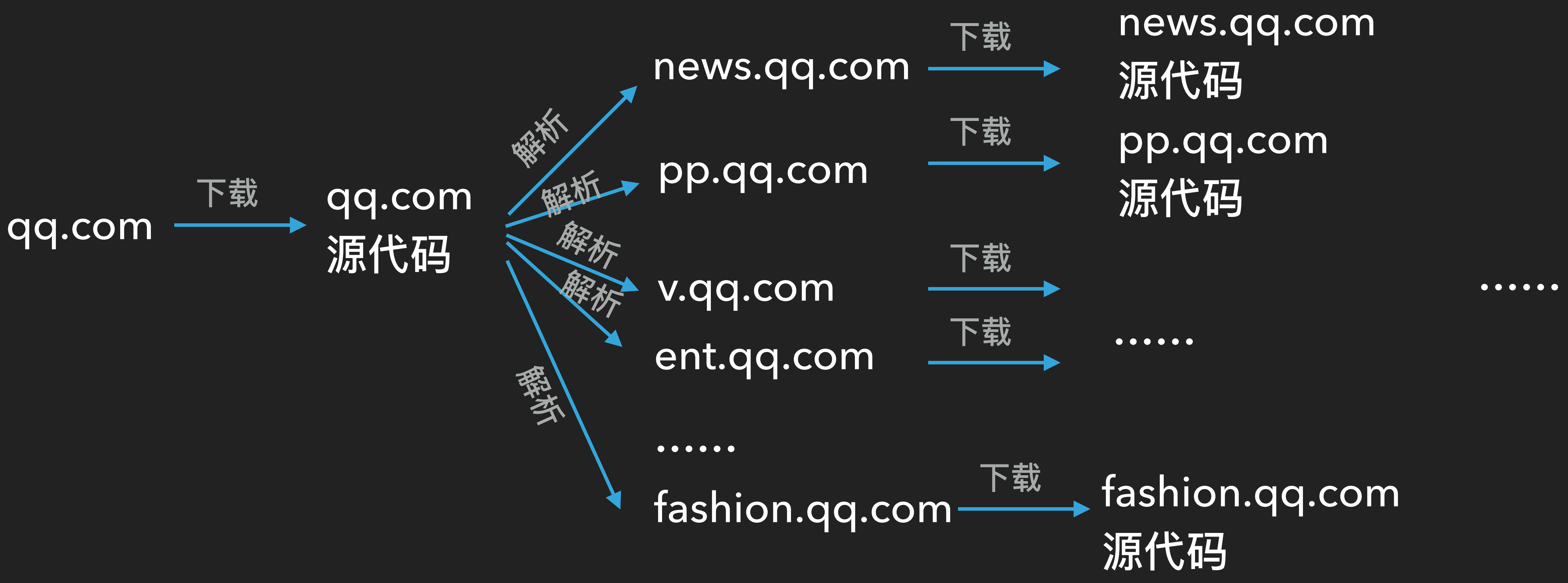
- 爬虫工作原理
- 网页抓取、去重、解析和存储相关技术栈
- Bloom Filter算法介绍
- 使用redis queue实现大规模网页抓取
- 反爬虫策略及应对措施
- QA

从种子URL获取HTML源代码说起



```python
# 基本demo，获取html源代码

import requests

def send_request(url):
    # 通过requests库来发送一个http请求，并且获取
    # 网页内容
    try:
        r = requests.get(url)
    except Exception as e:
        print e
        return
    html_source = r.text

    return html_source

# 调用
html_source = send_request("http://www.qq.com/")

print html_source
```

DEMO解释



qq.com

下载

qq.com
源代码

解析

解析

解析

解析

解析

news.qq.com

pp.qq.com

v.qq.com

ent.qq.com

……

fashion.qq.com

下载

下载

下载

下载

下载

news.qq.com
源代码

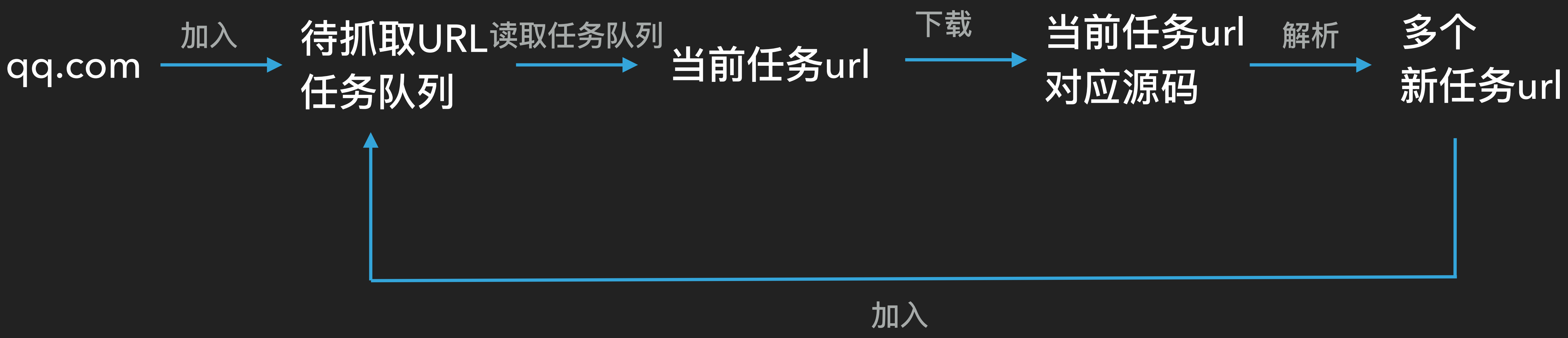
pp.qq.com
源代码

……

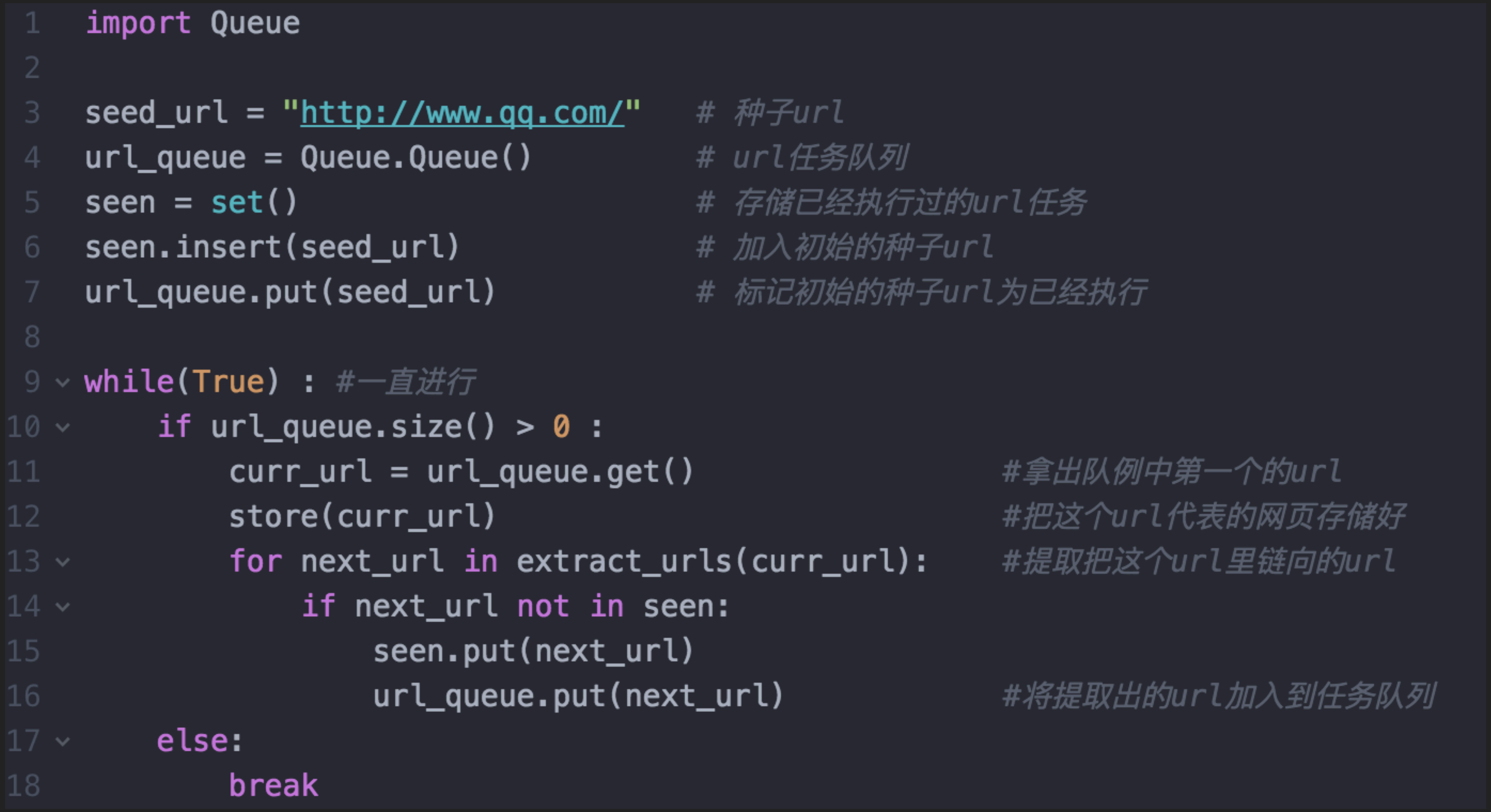
fashion.qq.com
源代码

……

引入URL队列模型

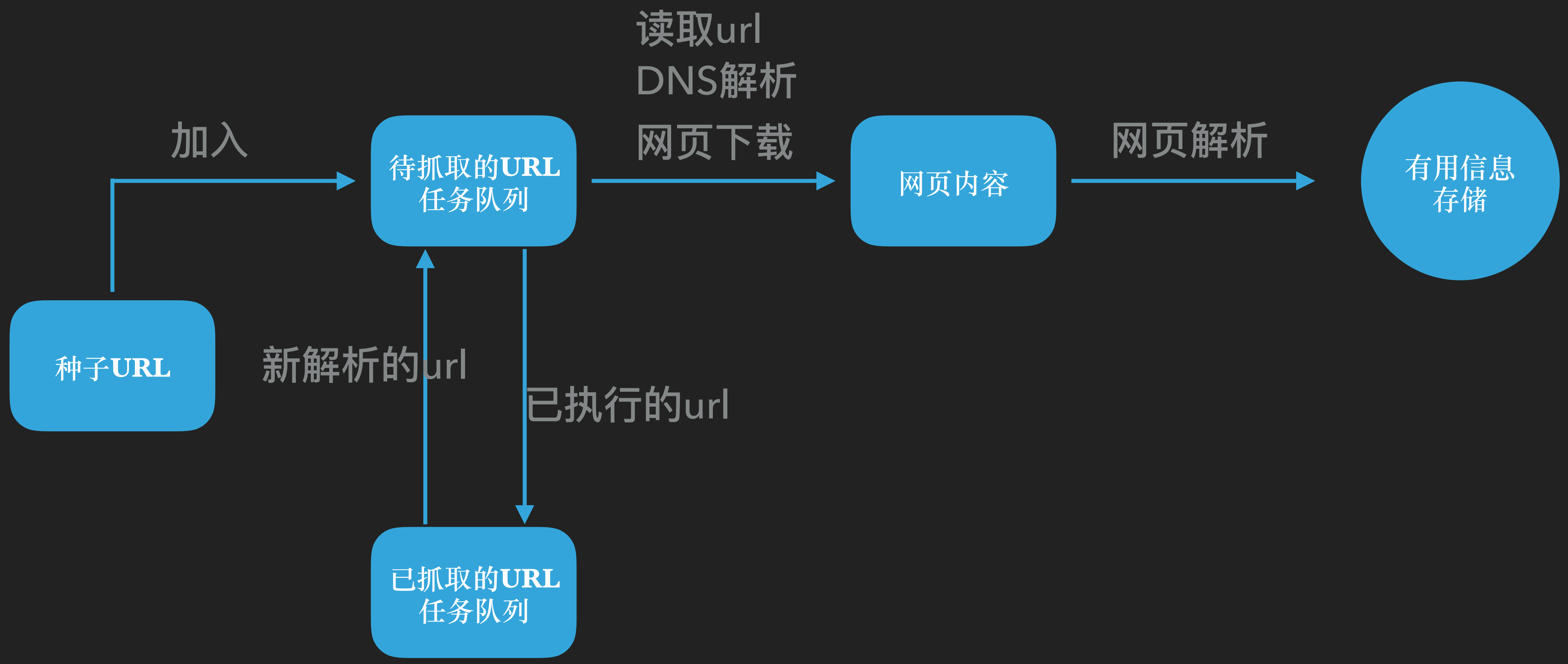

qq.com
加入
待抓取URL
任务队列
读取任务队列
当前任务url
下载
当前任务url
对应源码
解析
多个
新任务url
加入

通俗易懂的伪代码



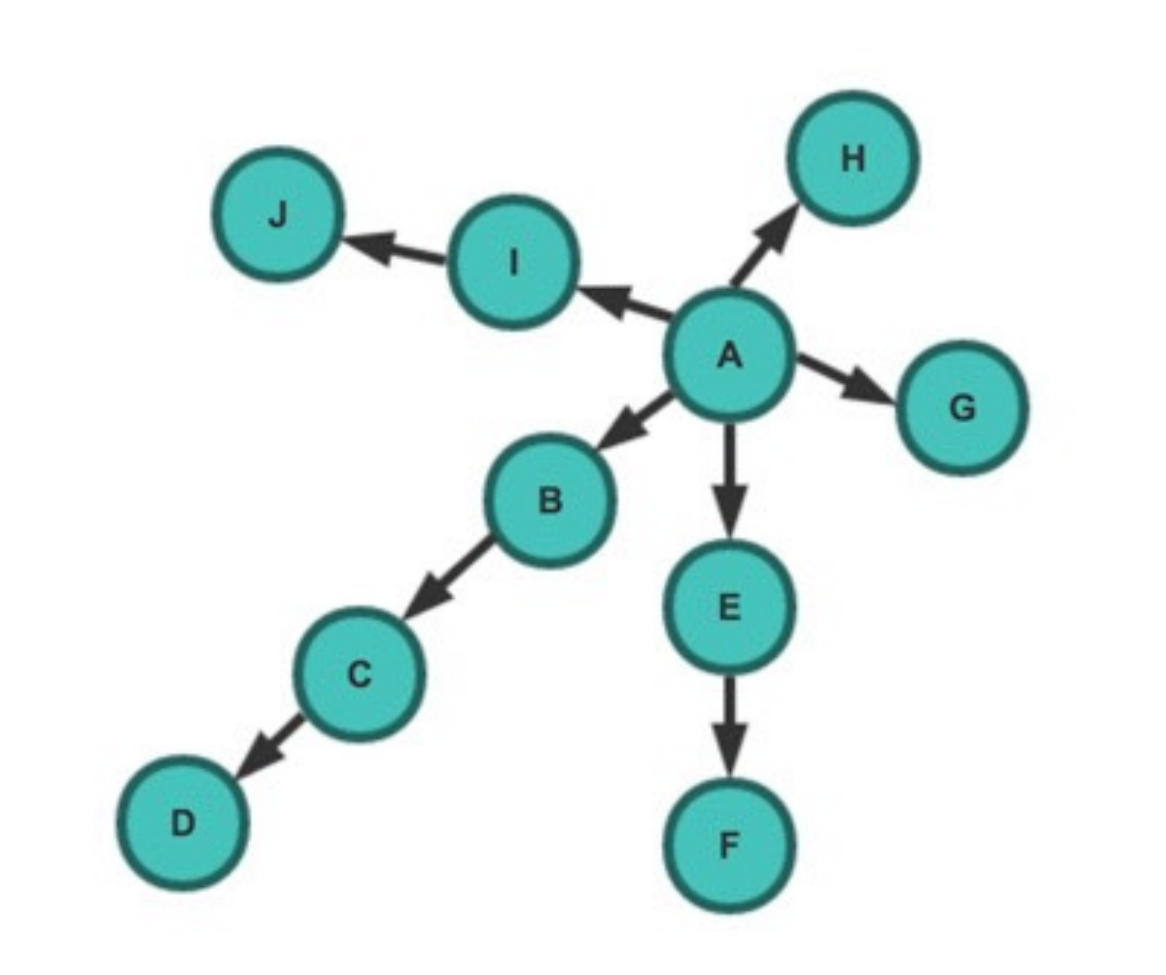
```
import Queue

seed_url = "http://www.qq.com/"    # 种子url
url_queue = Queue.Queue()         # url任务队列
seen = set()                      # 存储已经执行过的url任务
seen.insert(seed_url)             # 加入初始的种子url
url_queue.put(seed_url)           # 标记初始的种子url为已经执行

while(True) : #一直进行
    if url_queue.size() > 0 :
        curr_url = url_queue.get()                 #拿出队例中第一个的url
        store(curr_url)                            #把这个url代表的网页存储好
        for next_url in extract_urls(curr_url):    #提取把这个url里链向的url
            if next_url not in seen:
                seen.put(next_url)
                url_queue.put(next_url)            #将提取出的url加入到任务队列
    else:
        break
```

爬虫的基本流程总结

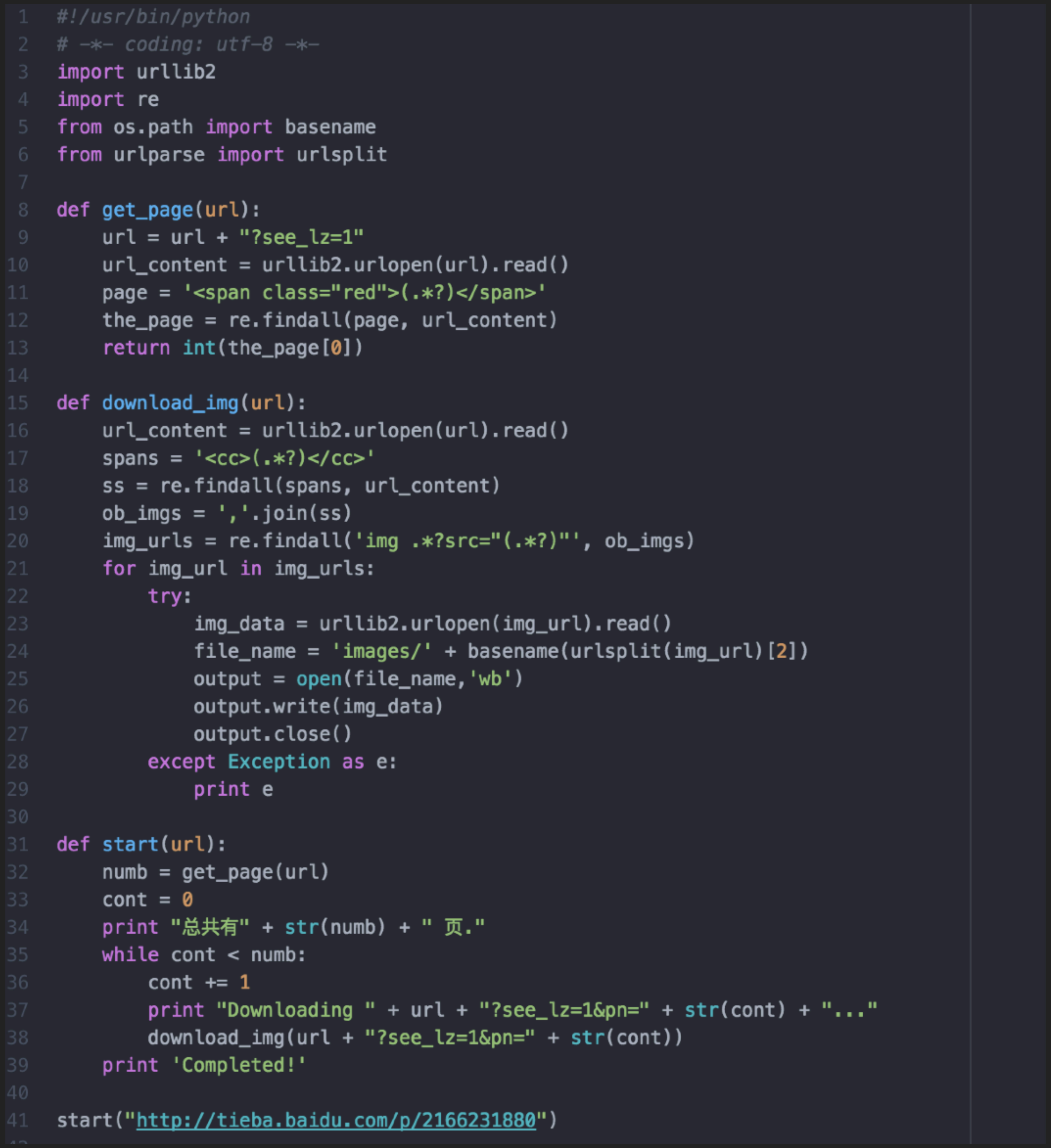

加入
待抓取的URL
任务队列
读取url
DNS解析
网页下载
网页内容
网页解析
有用信息
存储
种子URL
新解析的url
已执行的url
已抓取的URL
任务队列

爬虫的抓取策略

- 深度优先策略
- 广度优先策略



复杂点的例子：图片抓取程序



```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
import urllib2
import re
from os.path import basename
from urlparse import urlsplit

def get_page(url):
    url = url + "?see_lz=1"
    url_content = urllib2.urlopen(url).read()
    page = '<span class="red">(.*?)</span>'
    the_page = re.findall(page, url_content)
    return int(the_page[0])

def download_img(url):
    url_content = urllib2.urlopen(url).read()
    spans = '<cc>(.*?)</cc>'
    ss = re.findall(spans, url_content)
    ob_imgs = ','.join(ss)
    img_urls = re.findall('img .*?src="(.*?)"', ob_imgs)
    for img_url in img_urls:
        try:
            img_data = urllib2.urlopen(img_url).read()
            file_name = 'images/' + basename(urlsplit(img_url)[2])
            output = open(file_name,'wb')
            output.write(img_data)
            output.close()
        except Exception as e:
            print e

def start(url):
    numb = get_page(url)
    cont = 0
    print "总共有" + str(numb) + " 页."
    while cont < numb:
        cont += 1
        print "Downloading " + url + "?see_lz=1&pn=" + str(cont) + "..."
        download_img(url + "?see_lz=1&pn=" + str(cont))
    print 'Completed!'

start("http://tieba.baidu.com/p/2166231880")
```

技术栈

- 网页抓取：urllib2、urllib3、requests
- 去重：set、Bloom Filter
- 解析：re、XPath、BeautifulSoup
- 存储：MySQL、Mongodb、Redis
- 并发抓取：rq、scrapy、celery、thread、multiprocess

几个比较常见的例子

- 字处理软件中，需要检查一个英语单词是否拼写正确
- 在 FBI，一个嫌疑人的名字是否已经在嫌疑名单上
- 在网络爬虫里，一个网址是否被访问过
- yahoo, gmail等邮箱垃圾邮件过滤功能

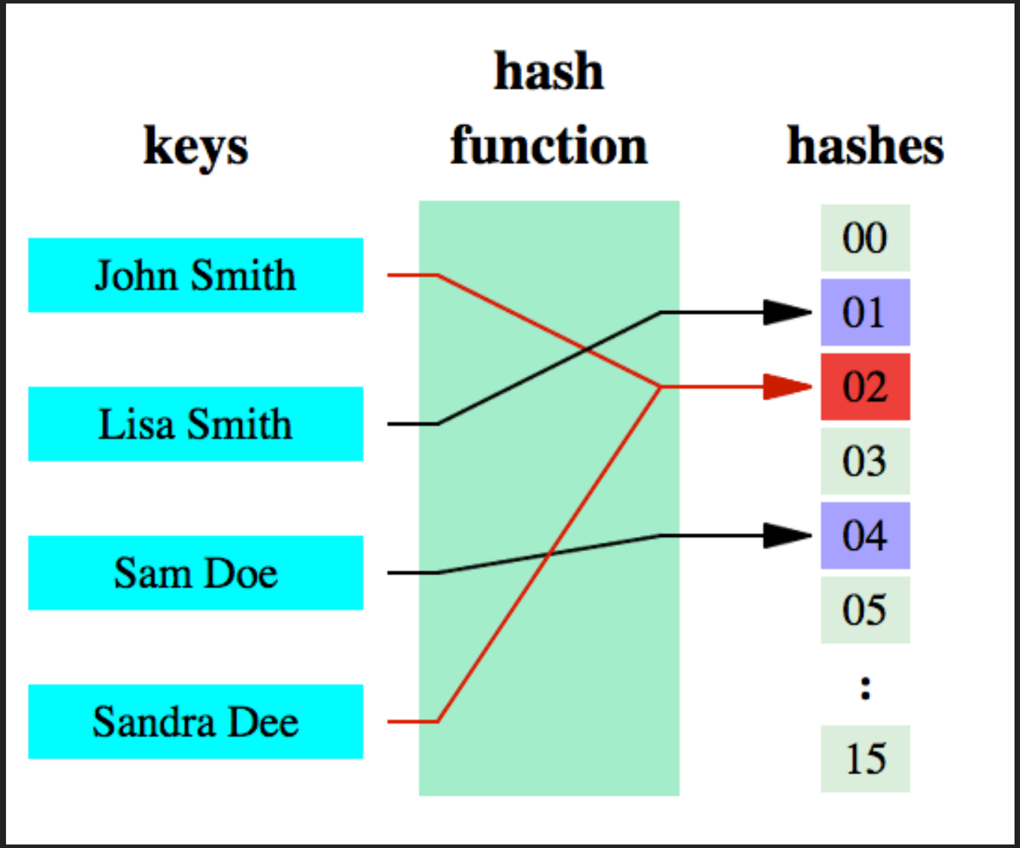
如何判断一个元素是否存在一个集合中?

- 数组
- 链表
- 树、平衡二叉树
- Trie
- 哈希表

哈希函数

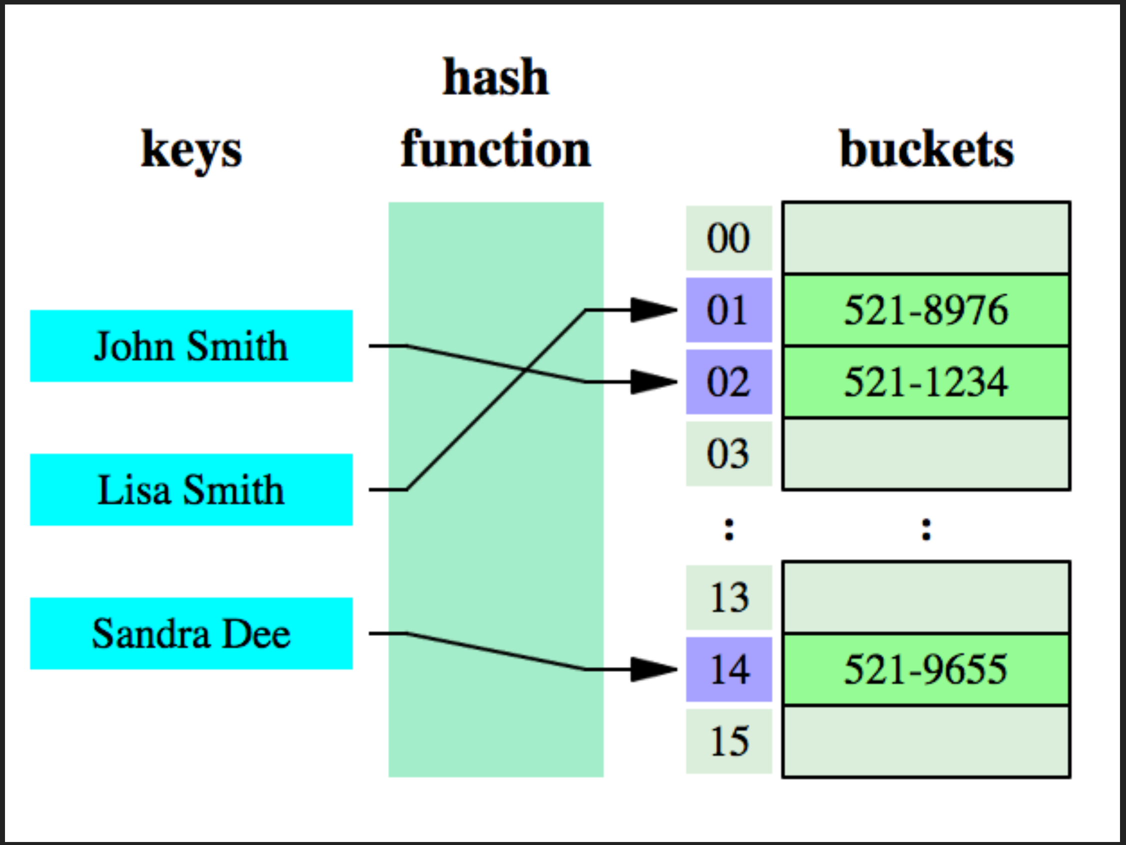
- 将任意大小的数据转换成特定大小的数据的函数
- 转换后的数据称为哈希值或哈希编码
- 特点：
  - 如果两个散列值是不相同的（根据同一函数），

    那么这两个散列值的原始输入也是不相同的
  - 不同的输入可能对应相同的输出（哈希碰撞）



哈希表

- 根据关键码值(Key value)而直接进行访问的数据结构
- 拉链法解决哈希冲突



使用哈希表存储一亿 个垃圾 EMAIL 地址的消耗?

- 哈希函数将一个email地址映射成8字节信息指纹
- 哈希表存储效率通常小于50%（哈希冲突）
- 消耗的内存：8 * 2 * 1亿 字节 = 1.6G 内存

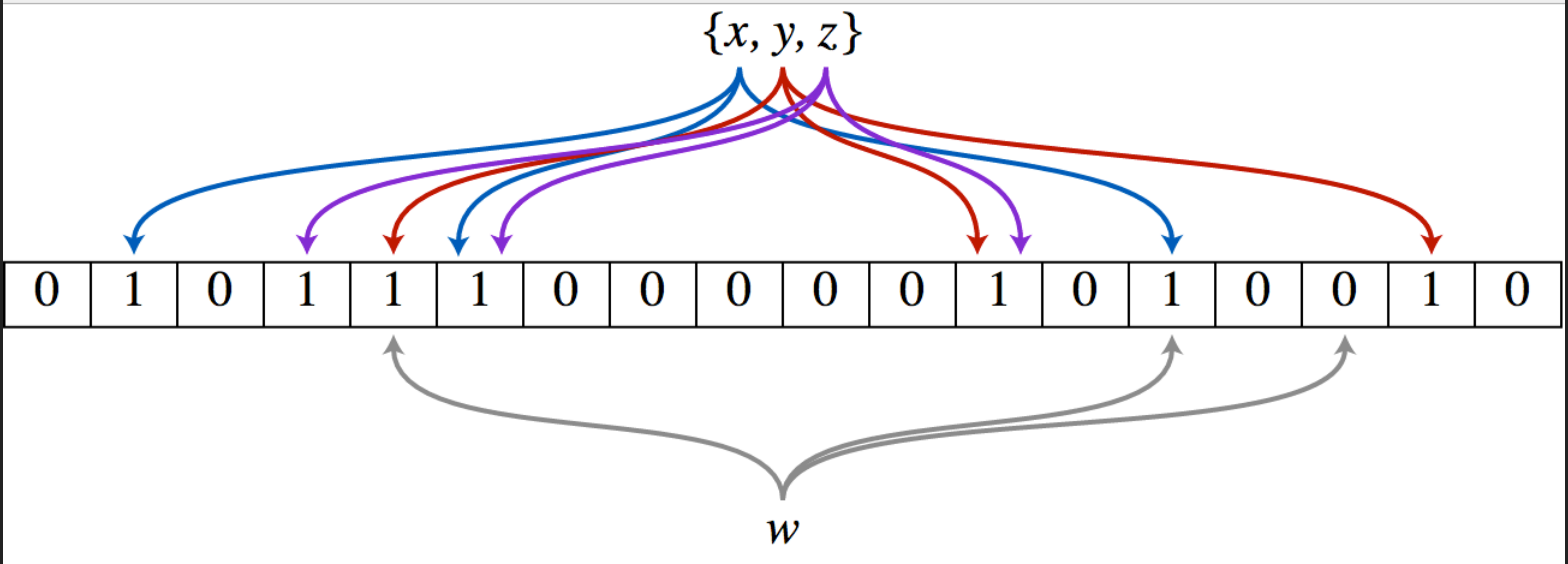
布隆过滤器（BLOOM FILTER）

- 巴顿.布隆于一九七零年提出
- 一个很长的二进制向量 （位数组）
- 一系列随机函数 (哈希)
- 空间效率和查询效率高
- 有一定的误判率（哈希表是精确匹配）

布隆过滤器的实现要求

- 长度为m的位数组
- k个独立的良好的哈希函数
  - 均匀的随机分布
  - 哈希冲突低
- k远小于m
- m和容纳的元素之间存在比例
- m 和 k的值取决于误判率

直观的描述



0	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	1	0

布隆过滤器添加元素

- 将要添加的元素给k个哈希函数
- 得到对应于位数组上的k个位置
- 将这k个位置设为1

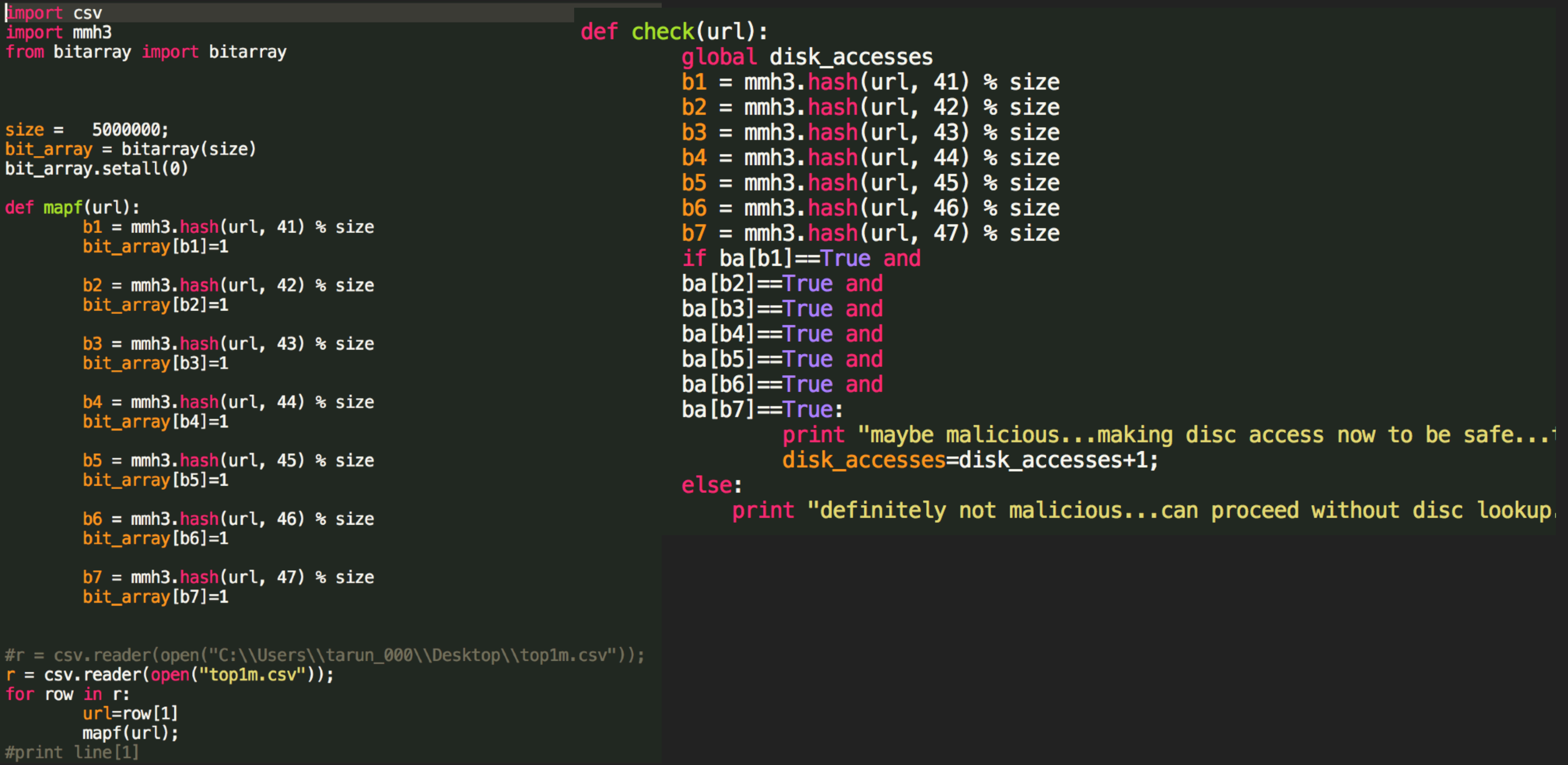
布隆过滤器查询元素

- 将要查询的元素给k个哈希函数
- 得到对应于位数组上的k个位置
- 如果k个位置有一个为0，则肯定不在集合中
- 如果k个位置全部为1，则可能在集合中 误判率存在的原因

布隆过滤器高效的原因

- 普通数据结构都存取元素
- Bloom filter只需存取要一个位数组
- 查询效率O(k)

布隆过滤器的简单实现

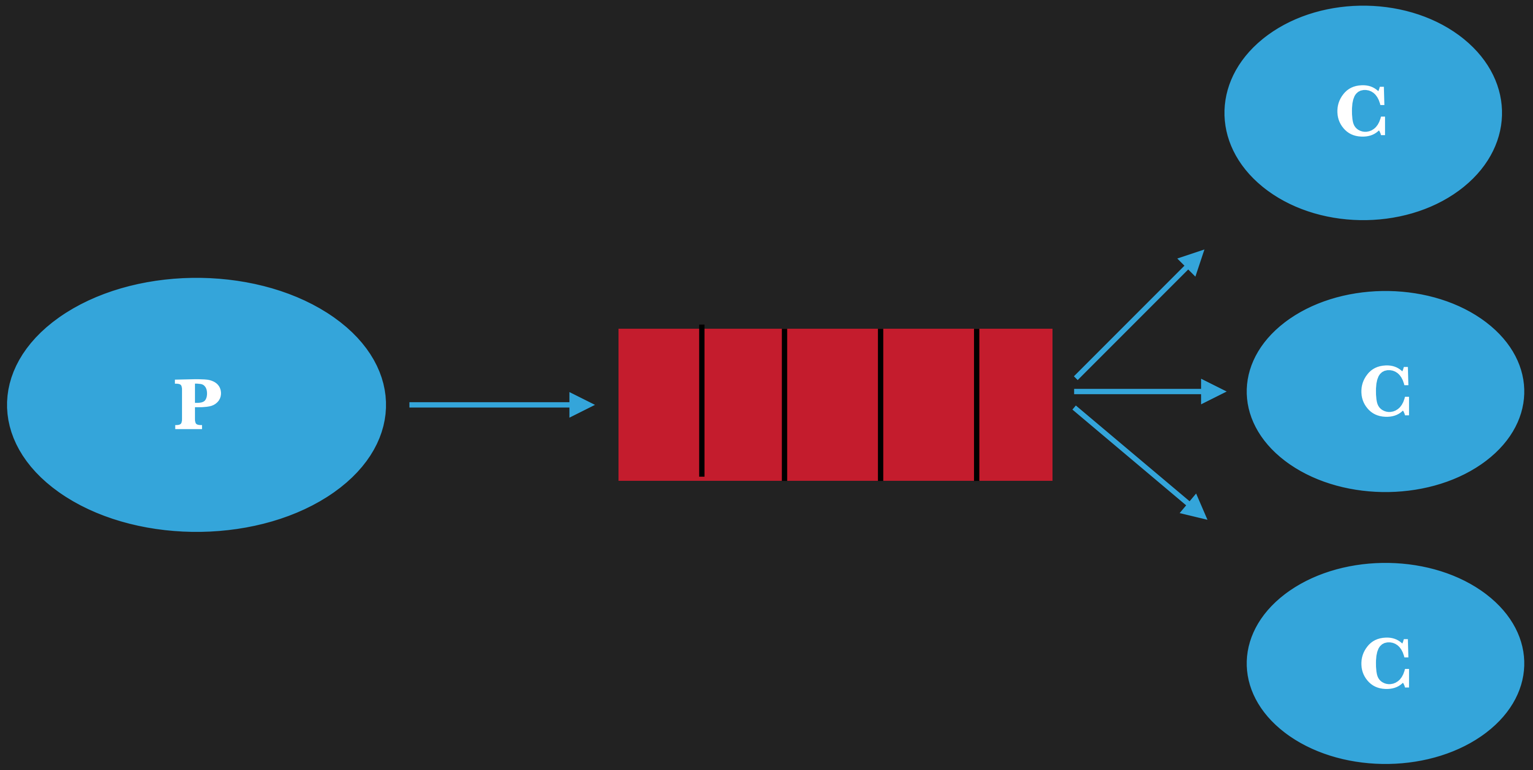


```
import csv
import mmh3
from bitarray import bitarray



size =   5000000;
bit_array = bitarray(size)
bit_array.setall(0)

def mapf(url):
        b1 = mmh3.hash(url, 41) % size
        bit_array[b1]=1

        b2 = mmh3.hash(url, 42) % size
        bit_array[b2]=1

        b3 = mmh3.hash(url, 43) % size
        bit_array[b3]=1

        b4 = mmh3.hash(url, 44) % size
        bit_array[b4]=1

        b5 = mmh3.hash(url, 45) % size
        bit_array[b5]=1

        b6 = mmh3.hash(url, 46) % size
        bit_array[b6]=1

        b7 = mmh3.hash(url, 47) % size
        bit_array[b7]=1


#r = csv.reader(open("C:\\Users\\tarun_000\\Desktop\\top1m.csv"));
r = csv.reader(open("top1m.csv"));
for row in r:
        url=row[1]
        mapf(url);
#print line[1]
```

```
def check(url):
    global disk_accesses
    b1 = mmh3.hash(url, 41) % size
    b2 = mmh3.hash(url, 42) % size
    b3 = mmh3.hash(url, 43) % size
    b4 = mmh3.hash(url, 44) % size
    b5 = mmh3.hash(url, 45) % size
    b6 = mmh3.hash(url, 46) % size
    b7 = mmh3.hash(url, 47) % size
    if ba[b1]==True and
    ba[b2]==True and
    ba[b3]==True and
    ba[b4]==True and
    ba[b5]==True and
    ba[b6]==True and
    ba[b7]==True:
        print "maybe malicious...making disc access now to be safe...
        disk_accesses=disk_accesses+1;
    else:
      print "definitely not malicious...can proceed without disc lookup
```

消息队列-生产者消费者模式介绍

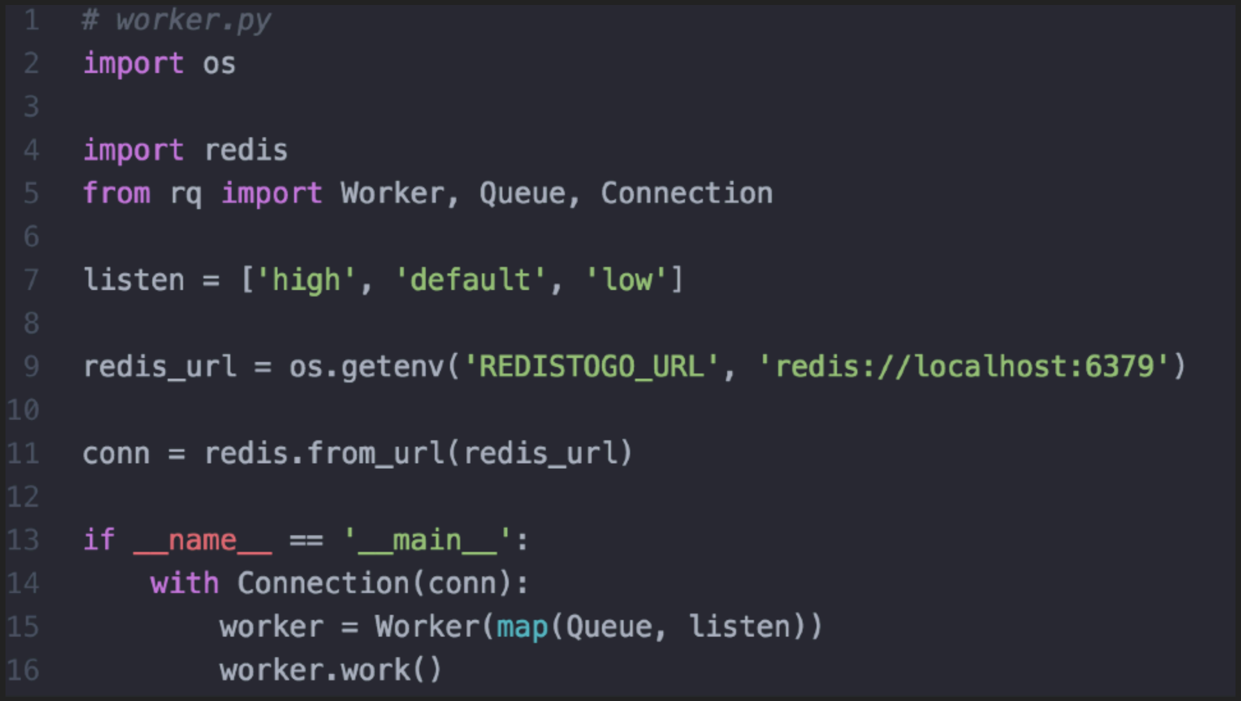
- 元素：生产者、消费者、任务队列
- 调度：round-robin、权重调度
- 消息ack确认



第四部分：使用REDIS QUEUE实现大规模网页抓取

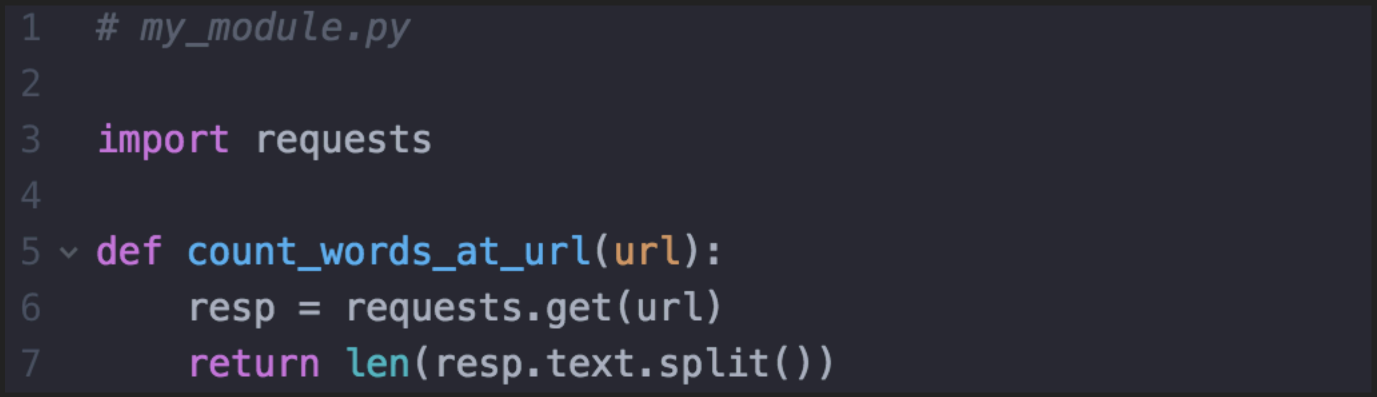
rq示例

worker.py 后台执行进程



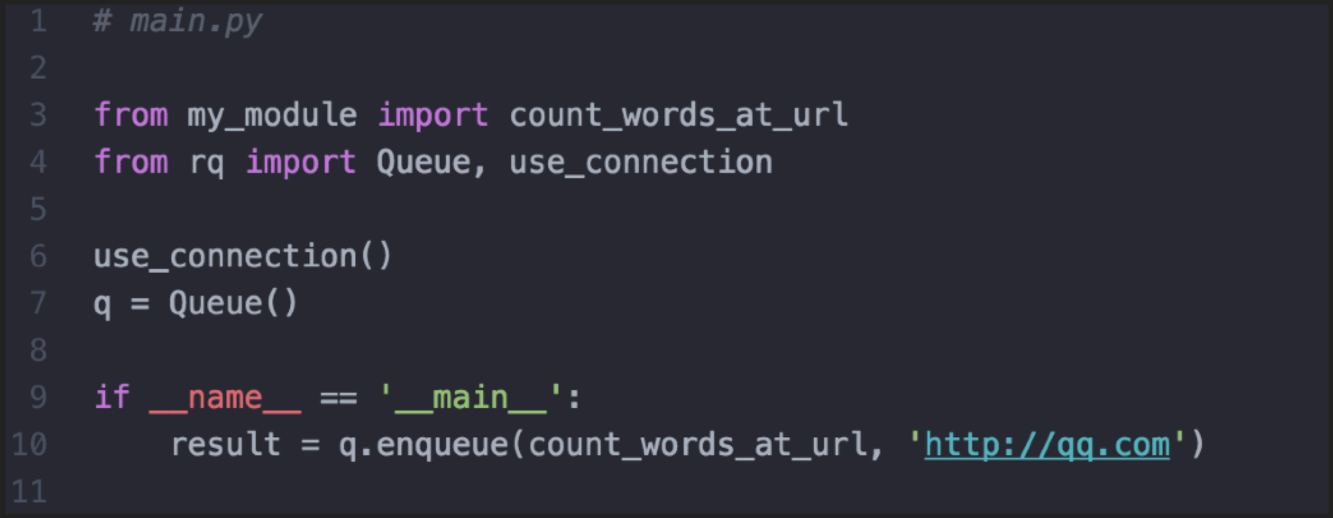
```python
# worker.py
import os

import redis
from rq import Worker, Queue, Connection

listen = ['high', 'default', 'low']

redis_url = os.getenv('REDISTOGO_URL', 'redis://localhost:6379')

conn = redis.from_url(redis_url)

if __name__ == '__main__':
    with Connection(conn):
        worker = Worker(map(Queue, listen))
        worker.work()
```

my_module.py 工具函数



```python
# my_module.py

import requests

def count_words_at_url(url):
    resp = requests.get(url)
    return len(resp.text.split())
```

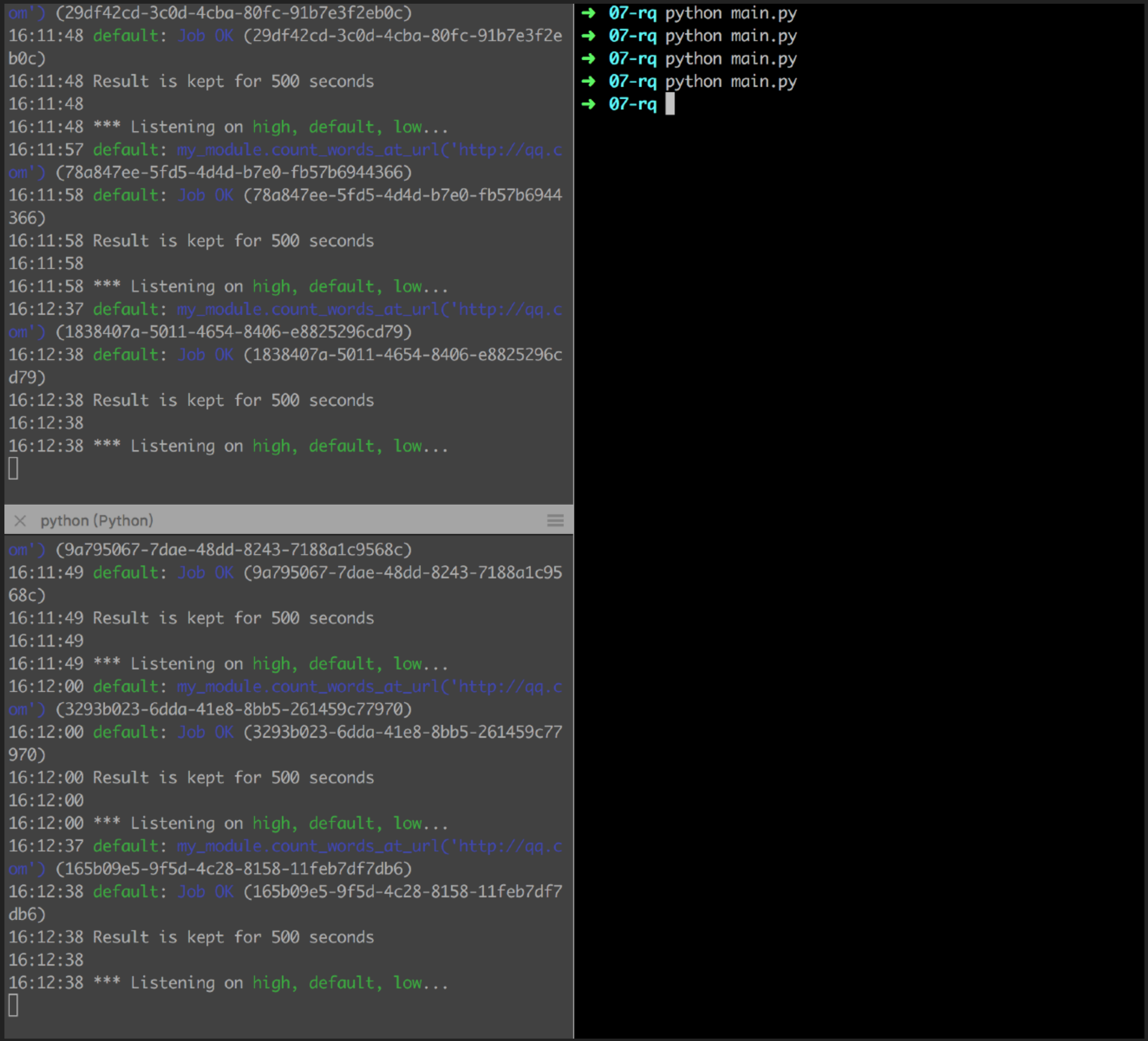
main.py 主入口



```python
# main.py

from my_module import count_words_at_url
from rq import Queue, use_connection

use_connection()
q = Queue()

if __name__ == '__main__':
    result = q.enqueue(count_words_at_url, 'http://qq.com')

```

第四部分：使用REDIS QUEUE实现大规模网页抓取

截图



```
om') (29df42cd-3c0d-4cba-80fc-91b7e3f2eb0c)
16:11:48 default: Job OK (29df42cd-3c0d-4cba-80fc-91b7e3f2e
b0c)
16:11:48 Result is kept for 500 seconds
16:11:48
16:11:48 *** Listening on high, default, low...
16:11:57 default: my_module.count_words_at_url('http://qq.c
om') (78a847ee-5fd5-4d4d-b7e0-fb57b6944366)
16:11:58 default: Job OK (78a847ee-5fd5-4d4d-b7e0-fb57b6944
366)
16:11:58 Result is kept for 500 seconds
16:11:58
16:11:58 *** Listening on high, default, low...
16:12:37 default: my_module.count_words_at_url('http://qq.c
om') (1838407a-5011-4654-8406-e8825296cd79)
16:12:38 default: Job OK (1838407a-5011-4654-8406-e8825296c
d79)
16:12:38 Result is kept for 500 seconds
16:12:38
16:12:38 *** Listening on high, default, low...
```

```
om') (9a795067-7dae-48dd-8243-7188a1c9568c)
16:11:49 default: Job OK (9a795067-7dae-48dd-8243-7188a1c95
68c)
16:11:49 Result is kept for 500 seconds
16:11:49
16:11:49 *** Listening on high, default, low...
16:12:00 default: my_module.count_words_at_url('http://qq.c
om') (3293b023-6dda-41e8-8bb5-261459c77970)
16:12:00 default: Job OK (3293b023-6dda-41e8-8bb5-261459c77
970)
16:12:00 Result is kept for 500 seconds
16:12:00
16:12:00 *** Listening on high, default, low...
16:12:37 default: my_module.count_words_at_url('http://qq.c
om') (165b09e5-9f5d-4c28-8158-11feb7df7db6)
16:12:38 default: Job OK (165b09e5-9f5d-4c28-8158-11feb7df7
db6)
16:12:38 Result is kept for 500 seconds
16:12:38
16:12:38 *** Listening on high, default, low...
```

```
➜ 07-rq python main.py
➜ 07-rq python main.py
➜ 07-rq python main.py
➜ 07-rq python main.py
➜ 07-rq
```

常见站点爬虫攻防

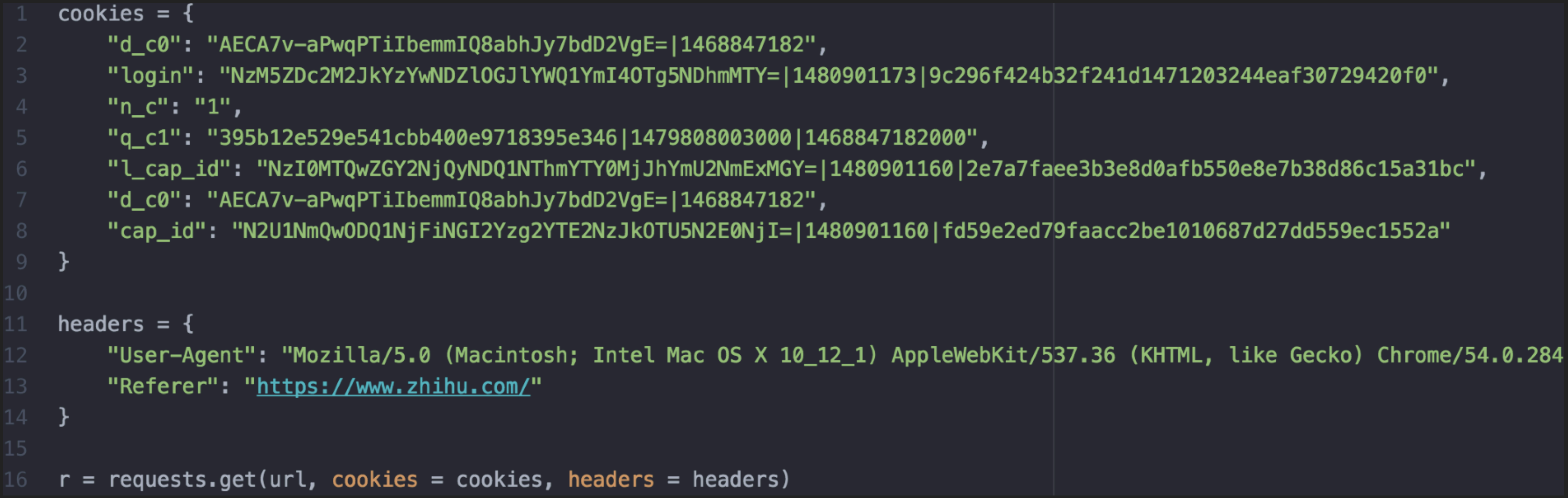
- 用户代理检测： 通过设置User-Agent header
- Referer检测： 通过设置Referer header
- 登陆限制：通过模拟登陆解决
- 访问频率限制：
    - 同一账号的频率限制，则可以使用多个账号轮流发请求
    - 如果针对IP，可通过IP代理池

各种浏览器的user-agent

- IE 8: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0)
- IE 7: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2)
- IE 6: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
- chrome: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36
- safari: ”Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/602.2.14 (KHTML, like Gecko) Version/10.0.1 Safari/602.2.14"

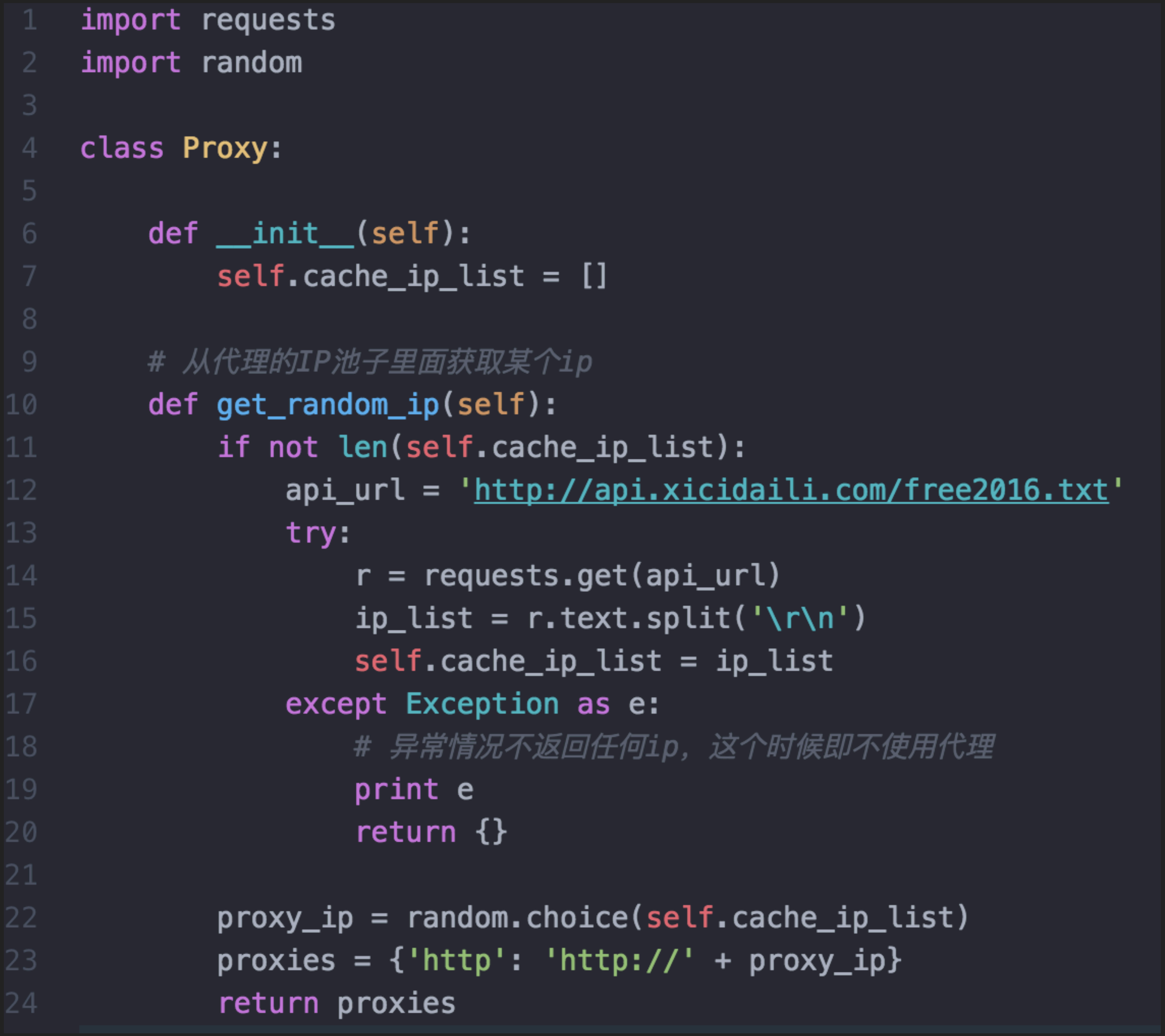
网站反“反盗链”

- referer：用于告知服务器用户的来源页面
- 某些站点会检测是否存在referer
- 资源站点反盗链



```
cookies = {
    "d_c0": "AECA7v-aPwqPTiIbemmIQ8abhJy7bdD2VgE=|1468847182",
    "login": "NzM5ZDc2M2JkYzYwNDZlOGJlYWQ1YmI4OTg5NDhmMTY=|1480901173|9c296f424b32f241d1471203244eaf30729420f0",
    "n_c": "1",
    "q_c1": "395b12e529e541cbb400e9718395e346|1479808003000|1468847182000",
    "l_cap_id": "NzI0MTQwZGY2NjQyNDQ1NThmYTY0MjJhYmU2NmExMGY=|1480901160|2e7a7faee3b3e8d0afb550e8e7b38d86c15a31bc",
    "d_c0": "AECA7v-aPwqPTiIbemmIQ8abhJy7bdD2VgE=|1468847182",
    "cap_id": "N2U1NmQwODQ1NjFiNGI2Yzg2YTE2NzJkOTU5N2E0NjI=|1480901160|fd59e2ed79faacc2be1010687d27dd559ec1552a"
}

headers = {
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.284
    "Referer": "https://www.zhihu.com/"
}

r = requests.get(url, cookies = cookies, headers = headers)
```

代理IP池



```python
import requests
import random

class Proxy:

    def __init__(self):
        self.cache_ip_list = []

    # 从代理的IP池子里面获取某个ip
    def get_random_ip(self):
        if not len(self.cache_ip_list):
            api_url = 'http://api.xicidaili.com/free2016.txt'
            try:
                r = requests.get(api_url)
                ip_list = r.text.split('\r\n')
                self.cache_ip_list = ip_list
            except Exception as e:
                # 异常情况不返回任何ip, 这个时候即不使用代理
                print e
                return {}

        proxy_ip = random.choice(self.cache_ip_list)
        proxies = {'http': 'http://' + proxy_ip}
        return proxies
```

Thanks for attending.