python 爬虫去重策略

python 爬虫去重策略

1、将访问过的 URL 保存到数据库中

缺点：效率低

2、将访问过的 URL 保存到 set 中

优点：只需要 o(1)的代价就可以查询 URL

缺点:对内存要求高。若有 1 亿网页,则占用内存为:1000000000*2byte*50 个字符/1024/1024/1024 = 9G

3、URL 经过 md5 等方法哈希后保存到 set 中

优点：可以成倍降低内存占用，Scrapy 使用的这种方法

4、用 bitmap 或者 bloomfilter 方法，将访问过的 URL 通过 hash 函数映射到某一位

bitmap 方法优点：一亿 URL 占用约 12M
bitmap 方法：去重没那么精准，存在冲突。
bloomfilter 优点：对 bitmap 进行改进，多重 hash 函数降低冲突

案例：空姐网相册爬虫（去重）

1、将访问过的 URL 保存到数据库中

把照片的 url 保存到 mangodb 数据库中

数据库初始化

```
server = 'localhost'
port = '27017'
dbname = 'admin'
user = 'admin'
pwd = '123'
uri = 'mongodb://' + user + ':' + pwd + '@' + server + ':' + port + '/' + dbname
client = pymongo.MongoClient(uri)    ##与 MongDB 建立连接
db = client['dbkongjie']    ##  选择一个数据库
```

self.kongjie_collection = db['kongjie'] ##在数据库中，选择一个集合

去重操作

```
if self.kongjie_collection.find_one({'img_url': url}):  ##判断 url 是否已经在数据库中，在则忽略。
    print(u'这个页面已经爬取过了')
else:
    #处理操作
```

添加数据

```
self.kongjie_collection.save({'img_url': url})
```

2、将访问过的 URL 保存到 set 中

集合的操作

set 是一个无序且不重复的元素集合。

创建集合 set

```
s = set()
s = {11,22,33,44}
```

集合添加、删除

python 集合的添加有两种常用方法，分别是 add 和 update。
add 方法：是把要传入的元素添加到集合中，例如：

```
a = set('boy')
a.add('python')
```

update 方法：是把要传入的元素拆分，做为个体传入到集合中

```
se = {11, 22, 33}
be = {22,44,55}
se.update(be)
```

判断元素是否在 set 中

```
if x in s x：
    print(‘exist.’)
else：
```

```
    print('not exist.')
```

去重操作

定义集合

self.img_urls = set() ##初始化一个集合 用来保存图片地址

去重操作

if url in self.img_urls: ##判断照片页面的 url 是否已经存在于集合中，不在就运行 else 下的内容，在则忽略。

```
    print(u'这个页面已经爬取过了')
else:
    #处理操作
```

添加数据：

self.img_urls.add(url)

3、URL 经过 md5 等方法哈希后保存到 set 中

MD5 使用

```
from hashlib import md5

data = 'Jet2017'
hash_md5 = md5(data.encode('utf8'))
psw = hash_md5.hexdigest()
print(psw)
```

去重操作

```
from hashlib import md5
```

定义集合

self.img_urls = set() ##初始化一个集合 用来保存图片地址

去重操作

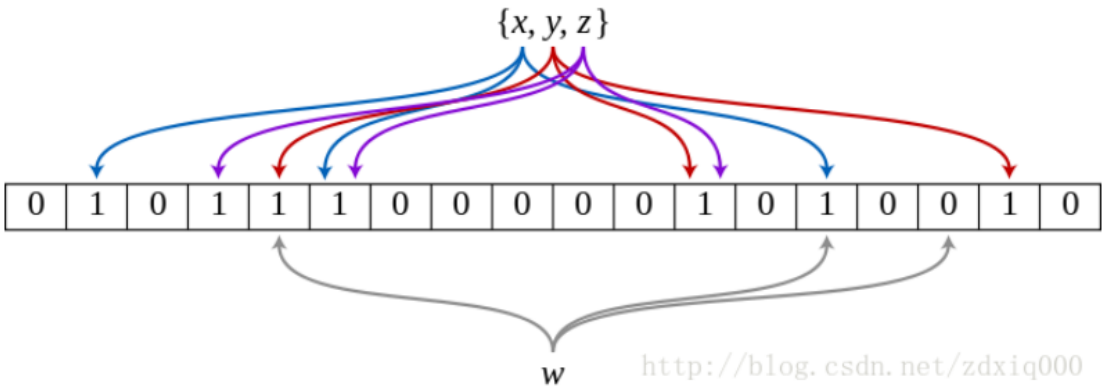
```
hash_md5 = md5(url.encode('utf8'))
hash_str = hash_md5.hexdigest()
#判断照片页面的 url 是否已经存在于集合中，不在就运行 else 下的内容，在则忽略。
if hash_str in self.img_urls:
    print(u'这个页面已经爬取过了')
else:
    #处理操作
```

添加数据：

self.img_urls.add(hash_str)

4、bloomfilter 方法去重

布隆过滤器（Bloom Filter)。如果说 Bitmap 对于每一个可能的整型值，通过直接寻址的方式进行映射，相当于使用了一个哈希函数，那布隆过滤器就是引入了 k(k>1)k(k>1)个相互独立的哈希函数，保证在给定的空间、误判率下，完成元素判重的过程。下图中是 k=3k=3 时的布隆过滤器。



安装

pip install bitarray-0.8.3-cp35-cp35m-win_amd64.whl
pip install pybloom_live

该模块包含两个类实现布隆过滤器功能。

BloomFilter 是定容。

ScalableBloomFilter 可以自动扩容

使用

ScalableBloomFilter

```
from pybloom_live import ScalableBloomFilter

sbf=ScalableBloomFilter(initial_capacity=100, error_rate=0.001,
mode=ScalableBloomFilter.LARGE_SET_GROWTH)

url = "www.baidu.com"
url2 = "www.douban,com"

sbf.add(url)

print(url in sbf)      # True
print(url2 in sbf)    # False
```

BloomFilter

```
from pybloom_live import BloomFilter

bf = BloomFilter(capacity=1000)

bf.add("www.baidu.com")

print("www.baidu.com" in bf)      # True
print("www.douban.com" in bf)    # False
```

去重操作

from pybloom_live import ScalableBloomFilter

定义集合

self.sbf = ScalableBloomFilter(initial_capacity=100, error_rate=0.001, mode=ScalableBloomFilter.LARGE_SET_GROWTH)

去重操作

```
#判断照片页面的 url 是否已经存在于集合中，不在就运行 else 下的内容，在则忽略。
if url in self. sbf:
    print(u'这个页面已经爬取过了')
else:
    #处理操作
```

添加数据：

self. sbf.add(url)