网站爬取URL去重方法

大家好啊，昨天我们讨论了网站爬取的两种策略——深度优先和广度优先。我们无论采取这两个策略的哪一个，都不可避免地会碰到一个相对比较严重的问题，那就是URL去重问题。

无论是什么网站，都会存在大量的URL重复的问题，如果不处理好这个问题，最严重的情况是可能会陷入死循环中。

例如大多数网站的第一个URL都是链接到首页的，如果采用深度优先策略，而不对URL进行去重，就会陷入死循环中。

还有可能出现什么问题呢，那就是虽然能够向整个网站的所有URL发起请求，但是将会耗费巨大的内存，甚至还没等爬取完整个网站的URL，内存就已经不够用了。

所以URL去重是我们在爬取大量数据的时候不可避免要碰见的问题，接下来，我们来讨论下几种URL去重的常用方法。

第一种：将访问过的URL保存到数据库中，当爬取到一个新的URL时，就去数据库中查询是否这个URL是否已经爬取过

优点：应用起来最简单

缺点：效率非常低、耗内存

第二种：将访问过的URL保存在set（集合）中，那么只需要O(1)的时间复杂度就能查询到URL

优点：效率很高，查询快，应用起来简单

缺点：耗内存

举例：假设一个网站有1亿个URL，每个URL假设有50个字符，按照python的编码格式Unicode，按每个字符2个字节算，那么需要多大的内存来保存这些URL呢？

100000000 \* 2 byte \* 50个字符 / 1024 / 1024 / 1024 ≈ 9G

那么2亿条就需要18G的内存，3亿条就需要27G的内存，这是多么恐怖的内存数字。

第三种：URL经过md5等方法哈希之后保存到set中（Scrapy框架采取的URL去重方法就是类似这种方法）

优点：效率很高，查询快

缺点：耗内存（但相对第二种方法，已经缩小了几倍的内存）

举例：md5编码能够将一个字符缩减到固定的长度，md5的一般编码长度是128bit，那么就是16byte，还是按照1亿个URL计算，需要耗费多大的内存来保存这些URL呢？

100000000 \* 16 byte / 1024 / 1024 / 1024 ≈ 1.5G

# 实现功能：将URL通过md5哈希之后，得到一个固定长度的字符串

improt hashlib

**def** **get\_md5**(url):

**if** isinstance(url, str):

url = url.encode('utf-8')

m = hashlib.md5(url)

**return** m.digest()

result = get\_md5('https://www.baidu.com')

print(result)

print(len(result))

第四种：用bitmap方法，将访问过的URL通过hash函数映射到某一位上，也就是某一个bit上

优点：进一步压缩了保存URL需耗费的内存

缺点：哈希冲突很高，不太适用

举例：一个byte有8个bit，也就是8个位，bitmap就是将一个URL通过hash函数，将它映射到8个位上的某一个位上，这样就进一步压缩了保存URL需耗费的内存，但极有可能将多个URL映射到了同一个位上，也就造成了哈希冲突，造成哈希冲突后，就需要向下寻址，有兴趣的童鞋可以网上搜索哈希冲突的解决方法。

第五种：bloomfilter方法对bitmap进行改进，多重hash函数降低哈希冲突

优点：既保留了bitmap的内存压缩优点，又良好解决了哈希冲突

缺点：难以理解

举例：还是按照1亿个URL来计算，采用这种方法需要占用多大的内存呢？

100000000 \* 1 bit / 8 / 1024 / 1024/ 1024  ≈  12M

当然这只是理想状况，尽管bloomfilter对bitmap进行了优化，但不可避免地还是会有哈希冲突的发生，导致内存是12M只是一种理想状况下的数字，实际上肯定不止12M，但无论如何，和之前的几种方法比较，内存还是成倍地进行了压缩

好了，到这里，几种URL去重的方法，就已经结束了，大家加油。

二十五  阅