­­­­­­­­­­­­­­­­­­­

未命名

**本科生毕业论文（设计）册**

学 院： 软件学院

专 业： 软件工程

年 级： 2015级

学生姓名： 于淑新

指导教师： 王琪

**河北师范大学本科生毕业论文（设计）任务书**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **姓 名** | **于淑新** | **学院** | **软件学院** | **专业** | **软件工程** | **年级（班）** | **2015级4班** |
| **毕业论文（设计）题目** | | **基于Python的音乐平台爬虫实现与数据抓取** | | | | | |
| **指导教师** | **王琪** | **职称** | **工程师** | **教研室** | **人工智能** | **研究方向** | **人工智能** |
| **论文（设计）基本要求：**  论文的基本内容：本论文主要研究的主题是运用Python这门编程语言对九酷音乐网站进行爬取，主要内容有以下几点：  爬虫的背景、发展概况  爬虫所需的技术（Python，Django，MySQL）以及爬虫的思想等  基本环节及要求：  首先对九酷音乐网站进行细致的模块划分，并分析出每个模块的URL特征  其次根据模块特征编写链接  最后使用Python的Requests库进行相关代码的编写并爬取数据 | | | | | | | |
| **论文（设计）研究目标：**  本论文主要是对九酷音乐网站进行爬取，即是对音乐数据的抓取，旨在实现免费听音乐、免费下载音乐的功能。 | | | | | | | |
| **主要参考文献：**   1. 零一，韩要宾，黄园园.Python 3 爬虫、数据清洗与可视化实战[M].北京：电子工业出版社，2018年9月. 2. 凯瑟琳﹒雅姆尔，理查德﹒劳森.用Python写网络爬虫[M].第二版.北京：人民邮电出版社，2018年12月. | | | | | | | |

指 导 教师： 年 月 日

教研室主任： 年 月 日

**河北师范大学本科生毕业论文（设计）开题报告书**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **姓 名** | | **于淑新** | **学院** | **软件学院** | **专业** | **软件工程** | **年级（班）** | **2015级4班** |
| **论文题目** | | **基于Python的音乐平台爬虫实现与数据抓取** | | | | | **指导教师** | **王琪** |
| **课题论证** | 随着经济的飞速发展，人们的生活压力也越来越大，取而代之的是人们对精神境界的向往也越来越高。在繁忙的工作制约能听到自己喜欢的音乐也成为人们对精神世界的一种向往。但是近几年由于音乐版权的提出，使得各大音乐平台不得不下架没有版权的音乐，这也让大多数的人不得不同时下载多个音乐APP才能听到自己想听的音乐。这样既造成的手机资源的浪费也消耗了人们的时间，得不偿失。而我们研究的主要意图是打造一款兼容各大音乐网站音乐的在线听歌网站。 | | | | | | | |
| **方案设计** | 1. 实用工具：谷歌浏览器、Pycharm编辑器 2. 首先使用谷歌浏览器分析每个模块的URL构成 3. 其次编写URL并提取每首歌曲的ID 4. 最后使用Python进行爬虫代码的编写 | | | | | | | |
| **进度计划** | 1. 2018年12月1日至2019年3月1日学习Python爬虫和Django 2. 2019年3月2日至2019年4月1日进行网站分析、爬虫代码的编写和使用Django连接数据库 3. 2019年4月2日至2019年4月15日进行毕业论文的编写和格式的调整 4. 2019年5月1日至2019年5月10日进行毕业答辩和第二次论文格式调整 | | | | | | | |
| **指**  **导**  **教**  **师**  **意见** | **指导教师：**  **年 月 日** | | | | | | | |
| **教研室意见** | **教研室主任：**  **年 月 日** | | | | | | | |

**河北师范大学本科生毕业论文（设计）文献综述**

|  |
| --- |
| 现如今音乐APP种类繁多，人们的选择也多种多样。但是更让消费者烦忧的是如何选择一款音乐曲目多、原唱歌曲多的APP已经成为让人们困扰不已的事，如何在最短的时间内找出适合自己的音乐APP已经成为消费者最为关心的事，而歌曲的种类也成为一个音乐APP生存发展的关键。  针对上述问题，爬虫技术应运而生。我们可以通过爬虫技术爬取各大音乐APP的音乐，并向用户免费提供这些音乐，然后在平台上加入适量的广告。这样既缩短了用户的寻找时间，也增加了商品的流通性。  网络爬虫（Web crawler），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本，它们被广泛用于互联网搜索引擎或其他类似网站，可以自动采集所有其能够访问到的页面内容，以获取或更新这些网站的内容和检索方式。从功能上来讲，爬虫一般分为数据采集，处理，储存三个部分。  传统爬虫从一个或若干初始网页的URL开始，获得初始网页上的URL，在抓取网页的过程中，不断从当前页面上抽取新的URL放入队列,直到满足系统的一定停止条件。聚焦爬虫的工作流程较为复杂，需要根据一定的网页分析算法过滤与主题无关的链接，保留有用的链接并将其放入等待抓取的URL队列。然后，它将根据一定的搜索策略从队列中选择下一步要抓取的网页URL，并重复上述过程，直到达到系统的某一条件时停止。另外，所有被爬虫抓取的网页将会被系统存贮，进行一定的分析、过滤，并建立索引，以便之后的查询和检索；对于聚焦爬虫来说，这一过程所得到的分析结果还可能对以后的抓取过程给出反馈和指导。  相对于通用网络爬虫，聚焦爬虫还需要解决三个主要问题： （1）对抓取目标的描述或定义； （2）对网页或数据的分析与过滤； （3）对URL的搜索策略。  Web网络爬虫系统的功能是下载网页数据，为搜索引擎系统提供数据来源。很多大型的网络搜索引擎系统都被称为基于 Web数据采集的搜索引擎系统，比如 Google、Baidu。由此可见Web 网络爬虫系统在搜索引擎中的重要性。网页中除了包含供用户阅读的文字信息外，还包含一些超链接信息。Web网络爬虫系统正是通过网页中的超连接信息不断获得网络上的其它网页。正是因为这种采集过程像一个爬虫或者蜘蛛在网络上漫游，所以它才被称为网络爬虫系统或者网络蜘蛛系统，在英文中称为Spider或者Crawler。  Web网络爬虫系统一般会选择一些比较重要的、出度(网页中链出超链接数)较大的网站的URL作为种子URL集合。网络爬虫系统以这些种子集合作为初始URL，开始数据的抓取。因为网页中含有链接信息，通过已有网页的 URL会得到一些新的 URL，可以把网页之间的指向结构视为一个森林，每个种子URL对应的网页是森林中的一棵树的根节点。这样，Web网络爬虫系统就可以根据广度优先算法或者深度优先算法遍历所有的网页。由于深度优先搜索算法可能会使爬虫系统陷入一个网站内部，不利于搜索比较靠近网站首页的网页信息，因此一般采用广度优先搜索算法采集网页。Web网络爬虫系统首先将种子URL放入下载队列，然后简单地从队首取出一个URL下载其对应的网页。得到网页的内容将其存储后，再经过解析网页中的链接信息可以得到一些新的URL，将这些URL加入下载队列。然后再取出一个URL，对其对应的网页进行下载，然后再解析，如此反复进行，直到遍历了整个网络或者满足某种条件后才会停止下来。 |

**河北师范大学本科生毕业论文（设计）翻译文**

|  |
| --- |
| IRLbot：可扩展到60亿页及以上  Mercator-B和Polybot使用所谓的批量磁盘检查——它们在内存中积累了大量的URL，然后一次将其与已排序的URL文件合并。Mercator-B只在RAM中存储新URL的散列，并将其文本放在磁盘上。为了保留从哈希到文本的映射，每个哈希都附加了一个特殊的指针。内存缓冲区满后，将其排序到位，然后在从磁盘读取时与URLSeen块进行比较。非重复的URL与磁盘上已有的URL合并并写入新版本的URLSeen。然后，指针用于恢复唯一URL的文本并将其附加到磁盘队列。  Polybot将整个URL（即实际字符串）保存在内存中，并将它们组织成二进制搜索树。一旦树大小超过某个阈值，它就会与磁盘文件urlsen合并，后者包含爬虫程序已经看到的压缩URL。除了CPU非常密集（即，根据我们的经验，URL的压缩和二进制字符串树中的搜索相当缓慢），由于RAM的使用效率较低，这种方法必须比Mercator-B更频繁地执行URLSeen扫描。  假设爬虫处于某种稳定状态，其中新URL之间的唯一性概率p保持不变（本文稍后将在实践中验证这一点）。进一步假设urlsen的当前大小为u个条目，分配给u r l检查的RAM大小为r，每个下载页面的平均链接数为l，平均url长度为b，url压缩比为q，爬虫程序希望访问n个页面。接下来，n=ln链接必须通过url检查，其中np是唯一的，bq是压缩url中的平均字节数。最后，用h表示爬虫使用的URL散列的大小，用p表示内存指针的大小。然后我们得到以下结果。  定理1。Urlseen批量磁盘检查的开销是ω（n，r）=α（n，r）bn字节，其中对于Mercator-b：  α(n,R) =2(2UH + pHn)(H + P)/bR+ 2 + p (1)  对于Polybot:  α(n,R) =2(2Ubq + pbqn)(b + 4P)/bR+ p. (2)  这个结果表明，ω（n，r）是两个元素的乘积：所有解析的URL中字节bn的数量以及它们被写入/读取磁盘的次数α（n，r）。如果α（n，r）与n一起增长，则爬虫的开销将超线性地扩展，最终可能会压倒爬虫的极限。当n→∞时，ω（n，r）中的二次项支配其他项，这使得Mercator-B的渐近性能处于：  ω(n,R) =2(H + P)pH/R\*n^2 (3)  和Polybot的  ω(n,R) =2(b + 4P)pbq/R\*n^2. (4)  这两个项的比值是（h+p）h/[bq（b+4p）]，对于h=8字节/哈希的irlbot情况，p=4字节/指针，b=110字节/url，并且使用非常乐观的bq=5字节/url表明mercator-b大约比polybot快7.2倍，因为n→∞。  在对照urlsen（例如mercator-b）检查URL之前，在磁盘上存储URL文本的任何方法的最佳性能都是αmin=2+p，这是将所有bn字节写入磁盘、将其读取以进行处理，然后将bpn字节追加到队列所需的开销。具有内存保持URL的方法（例如polybot）的绝对下限为α0min=p，这是将唯一URL写入磁盘所需的开销。然而，这两种约束在实践中都是不可能实现的。  我们现在描述IRLBOT中使用的URL检查算法，它属于一个更通用的框架，我们称之为具有更新管理的磁盘存储库（drum）。鼓的目的是允许有效存储大量<key，value>对集合，其中key是一些数据的唯一标识符（hash），value是附加到key的任意信息。在这些对上有三个受支持的操作—检查、更新和检查+更新。在第一种情况下，传入的数据集包含的密钥必须与存储在磁盘缓存中的密钥进行检查，并被分类为重复的或唯一的。对于重复的键，可以选择从磁盘中检索与每个键关联的值，并用于某些处理。在第二种情况下，传入列表包含需要合并到现有的磁盘缓存中的<KEY、Value>对。如果存在给定的键，则会更新其值（例如，重写或递增）；如果不存在，则会在磁盘文件中创建一个新条目。最后，第三个操作通过磁盘缓存一次执行检查和更新。另外请注意，鼓可能会提供一个混合列表，其中一些条目只需要检查，而其他条目则需要更新。  转鼓的高级概述如图1所示。在配置中，连续的元组流<key，value，aux>到达鼓，其中aux是与每个键相关联的一些辅助数据。鼓扩散对<key，value>在k磁盘桶之间qh 1，…，qh k基于它们的键（即，同一个桶中的所有键具有相同的位prefix）。这是通过将成对的<key，value>输入到每个大小为m的k内存数组中，然后在缓冲区结束时不断地将它们写入磁盘来完成的。第i个bucket中每个key的aux部分（通常包含url的文本）与qh i中pairs<key，value>以相同的fifo顺序保存在单独的文件qt i中。请注意，为了保持快速的顺序写入/读取，所有存储桶在使用前都预先分配到磁盘上。    图1 汽包运行  一旦最大的存储桶达到一定的大小r<r，以下过程重复，i=1，…，k:1）将存储桶qh i读取到图1所示的存储桶buffer中并进行排序；2）将磁盘缓存z按∆字节的块顺序读取，并与存储桶qh i中的键进行比较，以确定它们的唯一性；3）那些需要更新的qh i与磁盘缓存的内容合并并写入更新版本的z；4）在找到qh i中的所有唯一键后，恢复它们的原始顺序，然后按∆大小块顺序将qt i读取到内存中，并发送每个唯一键的相应辅助部分进行进一步处理（见下文）该算法的一个重要方面是，在一个直通磁盘缓存Z.3中检查所有存储桶。  我们现在解释如何使用鼓来存储爬虫数据。最重要的drum对象是urlsen，它只实现一个操作–check+update。传入元组是<URLHASH、-URLTEX>，其中密钥是每个URL的8字节散列，值为空，辅助数据是URL字符串。找到所有唯一的URL后，它们的文本字符串（辅助数据）将被发送到下一个队列以进行可能的爬行。对于缓存robots.txt，我们有另一个称为robotscache的鼓结构，它支持异步检查和更新操作。对于检查，它接收tuples<hostash-，urlext>和for updates<hostash，hostdata-，其中hostdata包含robots.txt文件、主机的IP地址以及其他与主机相关的可选信息。此部分的最后一个drum对象称为robots requested，用于存储请求robots.txt的站点的散列。与urlseen类似，它只支持同时检查+更新，其传入的元组为<hosthash，-，hostText>。  图2显示了由爬行线程生成的新URL的流向。它们首先使用check+update直接发送到urlseen。重复的URL将被丢弃，并发送唯一的URL以验证其是否符合预算（Star和Beast将在本文稍后讨论）。通过预算的URL将排队使用robotscache对照robots.txt进行检查。具有匹配robots.txt文件的URL会立即被分类为通过或失败。传递的URL在Q中排队，稍后由爬行线程下载。失败的URL将被丢弃。    图2 IRLBOT高层组织  没有匹配robots.txt的URL将发送到队列qr的后面，其主机名将通过使用check+update请求的robotsrequested传递。散列值尚未出现在此文件中的站点通过队列qd被送入执行DNS查找和下载robots.txt的一组特殊线程。随后，他们使用鼓对robotscache进行批量更新。由于在稳定状态下（即不包括初始阶段），下载robots.txt所需的时间远小于qr的平均延迟时间（即1-2天），因此每个URL在该循环中的周期不超过一个。此外，当robotscache检测到某些robots.txt或dns记录已过时时，它将所有相应的URL标记为“无法检查过时的记录”，这将强制robotsrequested提取一组新的排除规则和/或执行另一个dns查找。当robotscache被重新写入时，旧记录会在更新期间自动删除。  应该注意的是，只有当需要立即执行操作时，URL才会保存在内存中，图2中的所有队列都存储在磁盘上。我们还应该注意到，鼓数据结构可以支持磁盘空间允许的任意多的主机名、URL和robots.txt异常规则。  IRLbot：Scaling to 6 Billion Pages and Beyond  Mercator-B and Polybot use a so-called batch disk check – they accumulate a buﬀer of URLs in memory and then merge it with a sorted URLseen ﬁle in one pass. Mercator-B stores only hashes of new URLs in RAM and places their text on disk. In order to retain the mapping from hashes to the text, a special pointer is attached to each hash. After the memory buﬀer is full, it is sorted in place and then compared with blocks of URLseen as they are read from disk. Non-duplicate URLs are merged with those already on disk and written into the new version of URLseen. Pointers are then used to recover the text of unique URLs and append it to the disk queue.  Polybot keeps the entire URLs (i.e., actual strings) in memory and organizes them into a binary search tree. Once the tree size exceeds some threshold, it is merged with the disk ﬁle URLseen, which contains compressed URLs already seen by the crawler. Besides being enormously CPU intensive (i.e., compression of URLs and search in binary string trees are rather slow in our experience), this method has to perform more frequent scans of URLseen than Mercator-B due to the less-eﬃcient usage of RAM.  Assume the crawler is in some steady state where the probability of uniqueness p among new URLs remains constant (we verify that this holds in practice later in the paper). Further assume that the current size of URLseen is U entries, the size of RAM allocated to URL checks is R, the average number of links per downloaded page is l, the average URL length is b, the URL compression ratio is q, and the crawler expects to visit N pages. It then follows that n = lN links must pass through URL check, np of them are unique, and bq is the average number of bytes in a compressed URL. Finally, denote by H the size of URL hashes used by the crawler and P the size of a memory pointer. Then we have the following result.  Theorem 1. The overhead of URLseen batch disk check is ω(n,R) = α(n,R)bn bytes, where for Mercator-B:  α(n,R) =2(2UH + pHn)(H + P)/bR+ 2 + p (1)  and for Polybot:  α(n,R) =2(2Ubq + pbqn)(b + 4P)/bR+ p. (2)  This result shows that ω(n,R) is a product of two elements: the number of bytes bn in all parsed URLs and how many times α(n,R) they are written to/read from disk. If α(n,R) grows with n, the crawler’s overhead will scale superlinearly and may eventually become overwhelming to the point of stalling the crawler. As n → ∞, the quadratic term in ω(n,R) dominates the other terms, which places Mercator-B’s asymptotic performance at  ω(n,R) =2(H + P)pH/R\*n^2 (3)  and that of Polybot at  ω(n,R) =2(b + 4P)pbq/R\*n^2. (4)  The ratio of these two terms is (H + P)H/[bq(b + 4P)], which for the IRLbot case with H = 8 bytes/hash, P = 4 bytes/pointer, b = 110 bytes/URL, and using very optimistic bq = 5 bytes/URL shows that Mercator-B is roughly 7.2 times faster than Polybot as n→∞.  The best performance of any method that stores the text of URLs on disk before checking them against URLseen (e.g., Mercator-B) is αmin = 2 + p, which is the overhead needed to write all bn bytes to disk, read them back for processing, and then append bpn bytes to the queue. Methods with memory-kept URLs (e.g., Polybot) have an absolute lower bound of α0min = p, which is the overhead needed to write the unique URLs to disk. Neither bound is achievable in practice, however.  We now describe the URL-check algorithm used in IRLbot, which belongs to a more general framework we call Disk Repository with Update Management (DRUM). The purpose of DRUM is to allow for eﬃcient storage of large collections of <key,value> pairs, where key is a unique identiﬁer (hash) of some data and value is arbitrary information attached to the key. There are three supported operations on these pairs – check, update, and check+update. In the ﬁrst case, the incoming set of data contains keys that must be checked against those stored in the disk cache and classiﬁed as being duplicate or unique. For duplicate keys, the value associated with each key can be optionally retrieved from disk and used for some processing. In the second case, the incoming list contains <key,value> pairs that need to be merged into the existing disk cache. If a given key exists, its value is updated (e.g., overridden or incremented); if it does not, a new entry is created in the disk ﬁle. Finally, the third operation performs both check and update in one pass through the disk cache. Also note that DRUM may be supplied with a mixed list where some entries require just a check, while others need an update.  A high-level overview of DRUM is shown in Figure 1. In the ﬁgure, a continuous stream of tuples <key,value,aux> arrives into DRUM, where aux is some auxiliary data associated with each key. DRUM spreads pairs <key,value> between k disk buckets QH 1 ,...,QH k based on their key (i.e., all keys in the same bucket have the same bit-preﬁx). This is accomplished by feeding pairs <key,value> into k memory arrays of size M each and then continuously writing them to disk as the buﬀers ﬁll up. The aux portion of each key (which usually contains the text of URLs) from the i-th bucket is kept in a separate ﬁle QT i in the same FIFO order as pairs <key,value> in QH i . Note that to maintain fast sequential writing/reading, all buckets are pre-allocated on disk before they are used.    Figure 1: Operation of DRUM.  Once the largest bucket reaches a certain size r < R, the following process is repeated for i = 1,...,k: 1) bucket QH i is read into the bucket buﬀer shown in Figure 1 and sorted; 2) the disk cache Z is sequentially read in chunks of ∆ bytes and compared with the keys in bucket QH i to determine their uniqueness; 3) those <key,value> pairs in QH i that require an update are merged with the contents of the disk cache and written to the updated version of Z; 4) after all unique keys in QH i are found, their original order is restored, QT i is sequentially read into memory in blocks of size ∆, and the corresponding aux portion of each unique key is sent for further processing (see below). An important aspect of this algorithm is that all buckets are checked in one pass through disk cache Z.3  We now explain how DRUM is used for storing crawler data. The most important DRUM object is URLseen, which implements only one operation – check+update. Incoming tuples are <URLhash,-,URLtext>, where the key is an 8-byte hash of each URL, the value is empty, and the auxiliary data is the URL string. After all unique URLs are found, their text strings (aux data) are sent to the next queue for possible crawling. For caching robots.txt, we have another DRUM structure called RobotsCache, which supports asynchronous check and update operations. For checks, it receives tuples <HostHash,-,URLtext> and for updates <HostHash, HostData,->, where HostData contains the robots.txt ﬁle, IP address of the host, and optionally other host-related information. The last DRUM object of this section is called RobotsRequested and is used for storing the hashes of sites for which a robots.txt has been requested. Similar to URLseen, it only supports simultaneous check+update and its incoming tuples are <HostHash,-,HostText>.  Figure 2 shows the ﬂow of new URLs produced by the crawling threads. They are ﬁrst sent directly to URLseen using check+update. Duplicate URLs are discarded and unique ones are sent for veriﬁcation of their compliance with the budget (both STAR and BEAST are discussed later in the paper). URLs that pass the budget are queued to be checked against robots.txt using RobotsCache. URLs that have a matching robots.txt ﬁle are classiﬁed immediately as passing or failing. Passing URLs are queued in Q and later downloaded by the crawling threads. Failing URLs are discarded.  URLs that do not have a matching robots.txt are sent to the back of queue QR and their hostnames are passed through RobotsRequested using check+update. Sites whose hash is not already present in this ﬁle are fed through queue QD into a special set of threads that perform DNS lookups and download robots.txt. They subsequently issue a batch update to RobotsCache using DRUM. Since in steady-state (i.e., excluding the initial phase) the time needed to download robots.txt is much smaller than the average delay in QR (i.e., 1-2 days), each URL makes no more than one cycle through this loop. In addition, when RobotsCache detects that certain robots.txt or DNS records have become outdated, it marks all corresponding URLs as “unable to check, outdated records,” which forces RobotsRequested to pull a new set of exclusion rules and/or perform another DNS lookup. Old records are automatically expunged during the update when RobotsCache is re-written.    Figure 2: High level organization of IRLbot  It should be noted that URLs are kept in memory only when they are needed for immediate action and all queues in Figure 2 are stored on disk. We should also note that DRUM data structures can support as many hostnames, URLs, and robots.txt exception rules as disk space allows.   1. Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov∗ Department of Computer Science, Texas A&M University College Station, TX 77843 USA. |

未命名

**本科生毕业论文（设计）**

题目：基于Python的音乐平台爬虫实现与数据抓取

学生姓名： 于淑新

指导教师： 王琪

学 院： 软件学院

专 业： 软件工程

年 级： 2015级

完成日期： 年 月 日

学位论文原创性声明

本人所提交的学位论文 ，是在导师的指导下，独立进行研究工作所取得的原创性成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中标明。

本声明的法律后果由本人承担。

论文作者（签名）： 指导教师确认（签名）：

年 月 日 年 月 日

学位论文版权使用授权书

本学位论文作者完全了解河北师范大学有权保留并向国家有关部门或机构送交学位论文的复印件和磁盘，允许论文被查阅和借阅。本人授权河北师范大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其它复制手段保存、汇编学位论文。

论文作者（签名）： 指导教师（签名）：

年 月 日 年 月 日

摘 要

随着人们生活水平的日渐提高和生活节奏的逐步加快，人们不仅仅只满足于对物质层面的需求，对于精神境界的追求也在日渐加深，而听歌也成为了人们追求精神境界的一部分。然而随着社会对版权意识的日益加重，导致越来越多的音乐平台歌曲单调无味，人们获取想要收听的歌曲可能需要同时下载好几个音乐软件。

针对上述问题，本文以Python和Requests环境为基础，以“九酷音乐”为爬取对象，在学习并分析当前爬虫原理、核心模块及运行平台的基础上，探索性地实现一个基于Requests框架的网络爬虫，实现对目标信息的抓取与实现。

本文首先介绍当前爬虫的技术现状及爬虫工作者所面临的问题，其次介绍爬虫的基础知识，然后介绍数据库的存储和使用Django实现前端与MySQL数据库的连接，最后介绍爬虫的成果及展望。

**关键词：** Python 爬虫 Requests Django MySQL

**Abstract**

With the gradual improvement of people's living standards and the gradual acceleration of life rhythm, people are not only satisfied with the material needs, but also the pursuit of spiritual realm is deepening , and listening to songs has become a part of people's pursuit of spiritual realm. However, with the increasing awareness of copyright in society, more and more music platforms become monotonous and tasteless. People may need to download several music software at the same time to get the songs they want to listen to.

In order to solve the above problems, based on Python and Requests environment, taking " Nine Cool Music" as the crawling object, on the basis of learning and analyzing the current crawler principle, core modules and running platform, this paper explores the realization of a web crawler based on Requests framework, and achieves the capture and Realization of target information.

This paper first introduces the current status of crawler technology and the problems faced by crawlers. Secondly, it introduces the basic knowledge of crawlers. Secondly, it introduces the storage of database and the connection between the front end and MySQL database using Django. Finally, it introduces the achievements and prospects of crawlers.

**Key words:** Python Crawler Requests Django MySQL

目 录

[第1章 绪论 1](#_Toc7969927)

[1.1项目的背景及意义 1](#_Toc7969928)

[1.2 爬虫概述及发展历史 1](#_Toc7969929)

[1.3 爬虫的技术现状 1](#_Toc7969930)

[1.4论文的整体框架 2](#_Toc7969931)

[第2章 网络爬虫原理和相关技术介绍 3](#_Toc7969932)

[2.1 python爬虫优势 3](#_Toc7969933)

[2.2 网络爬虫的基本原理 3](#_Toc7969934)

[2.2.1 网页的请求过程 3](#_Toc7969935)

[2.2.2 网页的请求方式 4](#_Toc7969936)

[2.3 Cookie在本爬虫中的作用 4](#_Toc7969937)

[2.3.1 Cookie介绍 4](#_Toc7969938)

[2.3.2 Cookie的缺陷 4](#_Toc7969939)

[2.3.3 Cookie在本项目中的作用 4](#_Toc7969940)

[2.4 本章小结 5](#_Toc7969941)

[第3章 数据转换与数据存储 6](#_Toc7969942)

[3.1 Json数据 6](#_Toc7969943)

[3.2 数据匹配 6](#_Toc7969944)

[3.3 数据转换 6](#_Toc7969945)

[3.4 数据提取 7](#_Toc7969946)

[3.5 数据保存 8](#_Toc7969947)

[3.6 歌曲下载 9](#_Toc7969948)

[3.7 Python连接MySQL数据库 10](#_Toc7969949)

[3.7.1 SQLAlchemy库介绍 10](#_Toc7969950)

[3.7.2 数据库连接 10](#_Toc7969951)

[3.7.3 数据插入和提交 11](#_Toc7969952)

[3.8 本章小结 11](#_Toc7969953)

[第4章 爬虫的总体分析 12](#_Toc7969954)

[4.1 综述 12](#_Toc7969955)

[4.2 网页分析 12](#_Toc7969956)

[4.3 歌曲链接分析 14](#_Toc7969957)

[4.4 歌曲ID分析 14](#_Toc7969958)

[4.5 本章小结 15](#_Toc7969959)

[第5章 Django实现数据库与前端的连接 17](#_Toc7969960)

[5.1 Django介绍 17](#_Toc7969961)

[5.2 创建Django项目 17](#_Toc7969962)

[5.3 创建项目应用 17](#_Toc7969963)

[5.4 music文件配置 17](#_Toc7969964)

[5.4.1 music文件介绍 17](#_Toc7969965)

[5.4.2 seetings.py 文件配置详情 18](#_Toc7969966)

[5.4.3 urls.py文件配置详情 19](#_Toc7969967)

[5.5 index APP文件分析与配置 20](#_Toc7969968)

[5.5.1 models.py的文件编写 20](#_Toc7969969)

[5.5.2 index.py的文件编写 20](#_Toc7969970)

[5.5.3 数据迁移 21](#_Toc7969971)

[参考文献 22](#_Toc7969972)

[致 谢 23](#_Toc7969973)

基于Python的音乐平台爬虫实现与数据抓取

# 第1章 绪论

## 1.1项目的背景及意义

21世纪是互联网飞速发展的世纪，中国自改革开放以来，互联网行业从无到有、再到今天的百花齐放的画面。可以说，互联网行业目前在中国已经成为根深蒂固，不可替代。在互联网飞速发展的这个时代里催生了很多各种大大小小的互联网产品，而音乐平台就是其中的一种。现在网络上充斥着各种大大小小的音乐平台，代表的有QQ音乐、酷狗音乐、酷我音乐、网易云音乐等。但是越来越多的人发现如果只想听某几首歌可能会要同时下载多个音乐软件，这样不仅造成了手机和电脑资源的浪费，更是造成了人们时间上的浪费。

面对以上问题，网络爬虫应运而生，人们可以通过网络爬虫来创建一款集各大音乐平台之所成的音乐软件，它可以通过链接搜索到人们想听的歌曲，也可以通过推荐给出大多数人想听的歌曲信息。使得人们不用再花费大量的资源和时间去寻找自己所要的歌曲信息。

## 1.2 爬虫概述及发展历史

网络爬虫作为互联网技术的一部分，自上个世纪90年来开始，到现在，可以说网络爬虫技术日趋完善[1]。很多音乐网站的核心技术就是网络爬虫，在网络爬虫的开发过程中，Python凭借其简单易学，可扩展可嵌入，丰富而完善的库在众多爬虫语言中脱颖而出，并成为爬虫工作者最喜欢的语言之一，也成为了爬虫界的主流。

## 1.3 爬虫的技术现状

爬虫技术发展到今天，已经出现了很多的功能丰富而强大的框架，比如Python的Scrapy就是一个功能强大的爬虫库，但是相对于Python的Requests库而言，其灵活性相对来说比较低，对于爬虫的初学者而言我建议使用Python的Requests库来进行爬虫，而本文也使用Requests库来进行网页数据的抓取。

其次，在当今社会随着版权意识的不断加深，各大网站平台对于自己的数据都会采用保密措施，这无异于加重了爬虫工作者的难度。对于本文而言，在爬虫的过程中所面对的最大的难度无异于歌曲URL的获取。因为歌曲的每个ID是固定且不重复的，所以本文不在考虑URL了去重的问题。所以在做音乐网站爬虫的时候我们只需要考虑如何获取每首歌曲的URL，然后分析出URL的编写规则即可。

最后，因为考虑到内存大小及容量的问题，我们只是爬取了“九酷”音乐网站的部分音乐信息，所以不涉及爬取规则的制订，因此不需要考虑深度优先和广度优先的问题，只考虑多线程和多进程的问题即可。

## 1.4论文的整体框架

第一章主要介绍爬虫的背景、本论文爬虫的原因和目的、本项目所涉及到的爬虫的问题。

第二章主要介绍python相对于其它语言的优势以及网络爬虫的原理和Cookie在我们此次项目中的应用等。

第三章主要介绍网页数据的转换，换句话说就是当我们利用Python自带的Requests库爬取到网页的初始信息后如何转换成我们能看得懂的信息和转换后信息的提取，最后介绍了如何使用Python这门语言连接MySQL数据库和向数据库中插入提交数据等。

第四章是本项目的核心篇章，本章主要介绍了爬取九酷音乐网站的整体思想，以及对于一个音乐网站进行爬取的总体概括，指出了如何进行网页链接分析和歌曲链接分析。

第五章主要介绍Django关于数据库的连接操作，以及如何使用Django连接数据库和前端。

# 第2章 网络爬虫原理和相关技术介绍

## 2.1 python爬虫优势

在大多数情况下语言的选择将会是爬虫工作者所面临的巨大问题，如何选择一门好的爬虫语言不仅仅是对一个程序员经验的考验，更是对一个程序员能力的考验。经常爬虫的工作者一定会面临这样的问题，那就是网站更新的问题，每次网站更新的时候我们之前编写的爬虫机制可能将不再适用于改网站的爬虫，所以我们必须要对之前的爬虫机制进行修改，这也就是为什么有的公司会设立爬虫维护这一职位的由来了。

试想，对于一个网页的更新问题而言，网页肯定不会是全部更新，二十部分更新，大体机制不会改变，所以对于以前编写的爬虫的修改应该不是很多的。假如我们如果选取C或者C++这两门语言的话，我们可以想象得出我们代码的修改量将是巨大的，因为我们在进行源代码编写的时候，我们的代码量也将是巨大的。也有人会说，Java编写的代码量也不是很多啊，但是我们必须要清楚的是，Java它是一门有严格的语法规则的语言，我们也可以在考虑一下，当我们进行爬虫的时候，你好不容易分析出网页新的编码规则，你还要根据语言的规则来编写相关的代码，你觉得你还会有心情去写代码吗？所以这也就是我们为什么会选取python的原因，python没有严格的规则限制，简洁易懂，所以我们本项目全部采用python编程语言。

## 2.2 网络爬虫的基本原理

### 2.2.1 网页的请求过程

Request（请求）：用户对网站进行操作，即向网站服务器发送请求[2]。

Response（响应）：服务器获取到用户的请求，或验证请求是否有效，然后在根据请求的内容返回给用户相应的数据[2]。

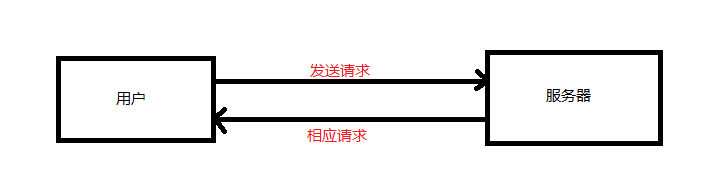


图 2-1 网页请求图

### 2.2.2 网页的请求方式

因为我们的爬虫对象，相应方式为GET方式，所以我们只在此处简单说明GET方式。

GET：是最常见的方式，一般用于获取或者查询信息，它也是大多数网站采用的方式，其优点是响应速度快[3]。

在爬虫之前确定请求方式是爬虫是最基础也是最关键的步骤。

## 2.3 Cookie在本爬虫中的作用

### 2.3.1 Cookie介绍

Cookie是绑定在用户Request请求和Web浏览器之间的一小段文本信息，主要记录的是用户在网页上的操作。这就好像我们写日记一样，是对于我们操作过程的一个记录。

我们在操作大多数的网页的时候，第一次我们会输入用户名和密码，但当我们勾选上永久记住用户名和密码之后，我们后续操作便不用在输入用户名和密码，这是因为服务器会将我们的用户名和密码发送到我们页面的Cookie，然后保存到我们的本地硬盘中。当我们再次调用的时候，服务器会直接通过Cookie调用我们上一次输入的用户名和密码，这样实现了永久的本地保存，我们便不用在输入用户名和密码了。

### 2.3.2 Cookie的缺陷

上面我们提到了Cookie会将用户名和密码保存到们的本地，但是这是一种不安全的行为，假如我们的Cookie在进行本地存储的时候不小心泄露了我们的用户名和密码，这将会造成我们的信息泄露，而跨站点脚本攻击这一技术正好可以达到此目的。跨站点脚本攻击是指利用网页的漏洞在我们的网页中植入非法的代码，以此达到获取我们全部Cookie的信息。试想假如这些信息落入黑客的手中，他们就会模拟正常的用户行为，来达到对我们的网站进行攻击的目的。

### 2.3.3 Cookie在本项目中的作用

我们在前面的两节之中介绍了Cookie的好处以及什么是Cookie。我们本项目是针对九酷音乐网站进行爬取的操作，所以不涉及Cookie的相关操作，而该网站没有反爬虫的机制，所以我们不涉及模拟正常用户的行为，直接进行相关的爬取工作即可。

## 2.4 本章小结

本章节主要介绍为什么我们会采取Python编程语言来编写我们的爬虫，还介绍了网页爬虫的原理，最后介绍了Cookie在我们爬虫项目中的应用。

# 第3章 数据转换与数据存储

## 3.1 Json数据

我们在进行爬虫工作的时候遇到的最多的问题往往是我们使用requests.get方式获取到网页的源代码之后，我们最需要解决的就是网页的数据转换问题，如何将爬取到的源代码变成我们看得懂的和容易操作的数据，这是一个非常关键的问题。

所以，我们在进行大多数数据转换的时候往往转换成Json数据，因为Json数据具有支持多种编程语言和简单易操作的优点，所以在大多数情况下我们往往把爬取到的数据转变成Json数据。

## 3.2 数据匹配

当我们获得网页源代码之后，需要对网页的源代码进行提取以获得我们需要的数据，这是我们需要用到正则表达式，对整个网页中不需要的数据进行丢弃，需要的数据予以保留。

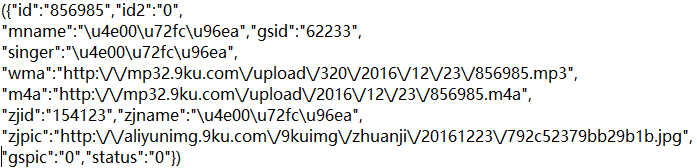


图 3-1 音乐信息截图

这是我们所获得的每首歌曲的网页源代码，我们需要获得的是歌曲的全部信息，即字典里面的内容。所以此时我们需要用到正则表达式对网页的内容进行匹配。详细代码如下：

str\_json = re.findall(‘{.\*}’, response.text)[0]

## 3.3 数据转换

str\_json即是我们需要获得的数据，在正则表达式中‘.’是匹配任意一个字符，‘\*’是匹配零个或多个字符，‘{.\*}’即是我们要匹配的是字典中的所有字符，而后面我们加上[0]，是因为，如果不加上[0]我们将会看到打印的结果是以列表的形式展现出来的，而我们加上[0]说明的是我们需要的是列表里面的内容，即为：



图 3-2 音乐截图

通过打印的结果我们可以看出，打印的是一个字符串的Json数据，这样我们是没有办法用Json语法去提取我们需要的数据的，所以我们需要把它转换成我们的data\_json，就是转换成一个真正的Json格式，具体代码如下：

data\_json = json.loads(str\_json)

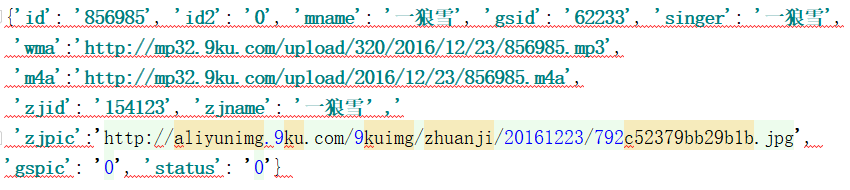


图 3-3 音乐字典

这才是我们真正要获得的数据，此时我们就可以对我们需要的信息进行提取了。即用Python字典key、value的方式提取字典里面的内容。

## 3.4 数据提取

在上一节中我们打印出了一首歌曲的详细信息，也已经对歌曲的信息进行了编码并转换成了我们能看得懂的信息，这节我们主要是根据Python字典进项详细的信息提取。

对于一首歌而言，我们需要保存的信息有以下方面：

1. id：歌曲ID相当于人的身份证一样，是唯一的确定的标识，我们获得并存取歌曲ID以后，就可以在数据库中根据这个唯一特定的标识去查找到我们所需要的任何歌曲信息。
2. mname：即为歌曲的名称，是除了歌曲ID以外，歌曲的另一个身份特征的标志。
3. singer：即歌手的名称。在大多数情况下，对于歌曲名一样的歌，歌手是不同的，而同一个歌手也可以有不同的歌。所以我们在除了ID之外，歌曲名加歌手名也是对一首歌唯一特定的标识。
4. wma，m4a：在本项目中wma，m4a这两个字典key值共同存储了我们歌曲的链接信息，在一般情况下我们应该是选取哪个关键值作为存储条件都是可以的，但是在本项目中，我们必须要存取两个字段并进行去重和异常值去除操作，因为这两个字段都是不完整的字段。

## 3.5 数据保存

在上面的章节中我们介绍了如何提取数据以及每段数据值所代表的含义，接下来我们将介绍如何对提取的数据进行保存以及为什么我们在众多结构中选取字典作为数据存储的对象。

学过数据结构的人都知道，不同的存储结构它的时间复杂度是不一样的，即对计算机的性能消耗也是不一样的，如何选取高效、快速的存储结构，这往往是一个程序员需要考虑的问题。在本项目中，我们使用字典来对我们的数据进行存储。选取字典的唯一好处就是字典key，value对应，我们在只需要在查找的时候输入字典的键就可以快速准确的找到字典的值。相对于列表这种存储结构而言，字典可以说是异常高效了。

存储详细代码如下：

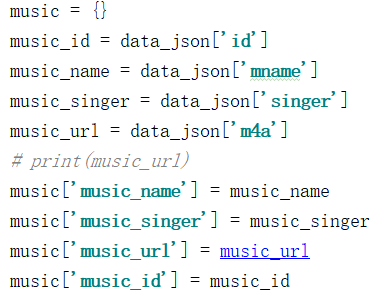


图 3-4 信息存储

但是在实际运行的过程中我们会看到歌曲的链接不全都是存入到m4a中的，有时候会在wma中，这时我们就需要对URL进行判断，看一下我们到底是要m4a的数据还是wma的数据。还有通过分析我们可以看到m4a的内容只有两种：第一种为歌曲的URL是以字符串的形式存在的，第二种为None，而m4a的存在形式有多种。所以我们在进行URL判断的时候是需要对if条件中的m4a进行判断即可。具体代码如下的：

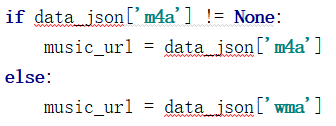


图 3-5 存储判断

所以对歌曲实现歌曲信息存储的总代码为：

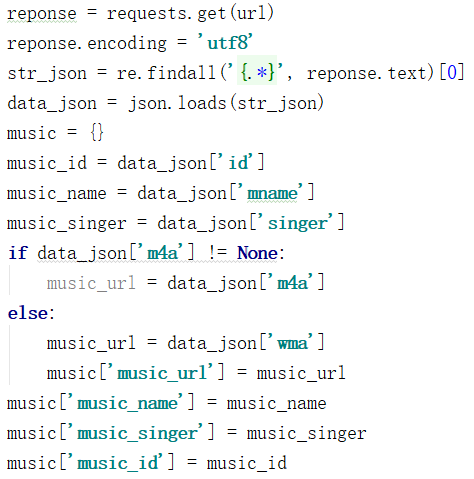


图 3-6 音乐存储

## 3.6 歌曲下载

当我们获取到歌曲的基本信息以后，为了能实现稳定的播放，我们需要将歌曲下载到本地，然后在数据库中存入本地的地址，为考虑到资源配置的问题，我们实现部分歌曲的下载，具体代码如下：

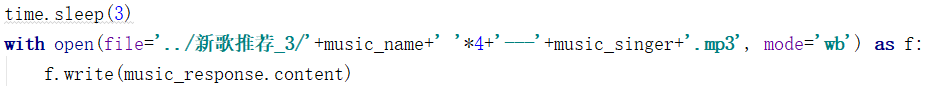


图 3-7 存储位置

在上面的代码中，我们设置了time的时间，这样做是为了减轻服务器的压力，模拟正常用户的行为。我们还在open函数中写了file='../新歌推荐'这是利用相对路径的写法把歌曲下载到指定的文件夹，值得注意的是在“新歌推荐”后面一定不要加上“/”，mode='wb'代表我们只是向文件夹中写入文件。如果想获取更多的方式请百度，我们这里不做详细的解释。

## 3.7 Python连接MySQL数据库

### 3.7.1 SQLAlchemy库介绍

我们使用Python的SQLAlchemy库，该库提供了SQL工具及对象关系映射工具。

### 3.7.2 数据库连接

导入SQLAlchemy，并初始化DBSession，具体代码如下：



图 3-8 数据库代码

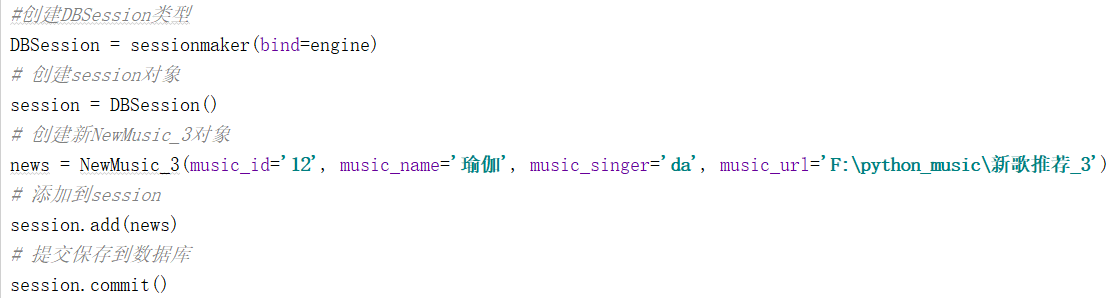


图 3-9 数据提交

### 3.7.3 数据插入和提交

下面我们将要向数据表中插入数据并向数据库提交，具体代码如下：

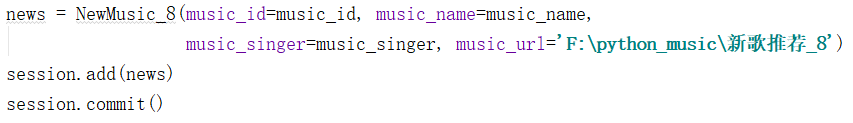


图 3-10 数据插入

通过循环的方式向新创建的对象中插入数据，然后增加到数据表中，提交到数据库中。

## 3.8 本章小结

本章主要介绍了Json数据的好处以及对于GET请求下网页数据的处理。当我们请求到网页的源代码之后需要对网页代码进行分析，来获取我们需要的数据，即数据匹配。

第一：当我们逐个排查找到了自己的目标数据之后，利用正则表达式提取出目标数据，因为正则提取后数据是以列表的形式存在的，所以我们要利用列表的下表进行数据的提取，这是数据的初步提取。

第二：因为之前提取到的数据只是str\_json数据，即不成熟的Json数据，所以接下来我们要转换成我们所熟悉的数据，即data\_json数据，利用json.loads函数进行数据的转换。

第三：当我们转换成我们熟悉的数据之后，在利用字典键的方式提取字典键所对应的值，然后存入到我们所创建的空字典中，进而完成数据的存储。

第四：使用Python的SQLAlchemy库，连接MySQL数据库，并向数据库中插入数据，将数据存储到数据库中，歌曲的地址链接为本地的歌曲地址。

# 第4章 爬虫的总体分析

## 4.1 综述

在前面三个章节中我们详细的介绍了如何爬取“九酷”音乐网站，也列出了详细的代码，但是对于我们拿到一个网站从哪几方面分析，还有链接是如何抓取的，我们并没有做出详细的解释，所以在此章节中我们将会教大家如何分析一个网站。我们还是用“九酷”音乐网站为例。

## 4.2 网页分析

我们在做爬虫的时候，基本上都用的是谷歌浏览器，至于谷歌浏览器的优点我们在这不详加解释。

第一：在谷歌浏览器中输入<http://www.9ku.com/>。

第二：点击新歌推荐，然后按F12，打开开发者界面，然后定位到Network。

第三：勾选上Preserve log，然后向下滑动鼠标滚轮，就会看到如下图所示：

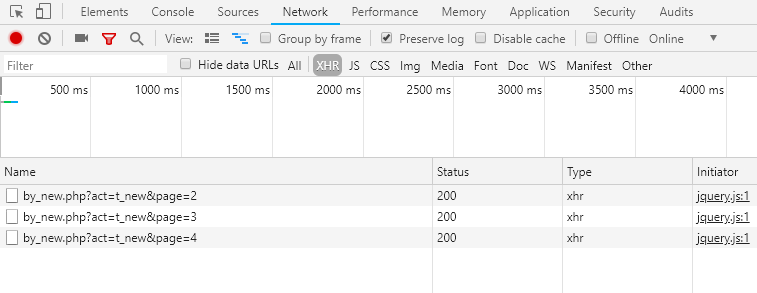


图 4-1 主题信息展示

第四：选择Name中的任何一个链接，在浏览器中打开，显示界面如下图所示：



图 4-2 歌曲界面

第五：我们改变网页链接的最后一个数字，可跳转到不同的歌曲界面，也这是我们要爬取的整个的网站页面。

第六：我们选择任意一首歌曲，还是打开开发者界面，选择Media，然后多选择几首歌曲进行播放，如下图所示：

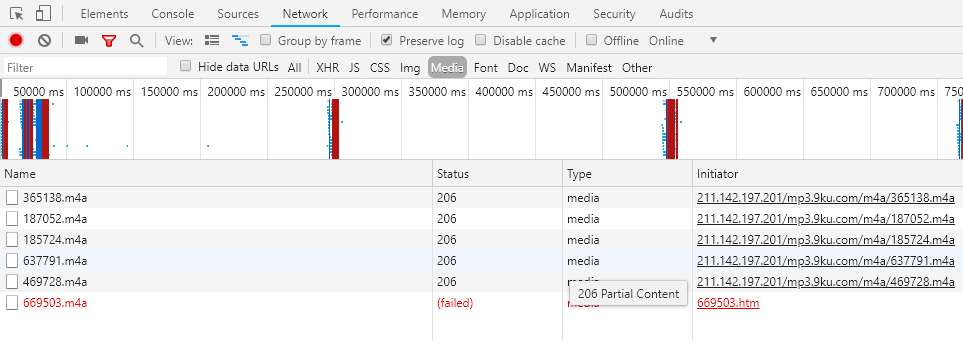


图 4-3 歌曲链接

第七：我们选择XHR，点击Response，会看到每条歌曲的详细信息和下载链接，在Preview模式下，文字会以我们看得懂的方式展现出来，如下图所示：



图 4-4 歌曲网页信息

## 4.3 歌曲链接分析

在上一章节中我们对爬取网页进行了全面的分析，现在我们分析网页歌曲页面详细信息的链接，如下面所示：

<http://www.9ku.com/html/playjs/507/506703.js>

<http://www.9ku.com/html/playjs/184/183203.js>

<http://www.9ku.com/html/playjs/362/361892.js>

我们在此处只列出三个链接，通过菲尼发现，歌曲详细页面的链接只有后面几位是不同的，即：507/506703.js、184/183203.js、362/361892.js。而前面的三位数字是后面数字的前三位加一生成的，而后面的数字是歌曲的ID，所以通过分析我们可以得出下一步的信息，就是我们要获得歌曲的ID一切就可以迎刃而解了。

## 4.4 歌曲ID分析

我们打开4.2章节中得到的链接，然后打开开发者工具如下图所示：



图 4-5 ID图

通过分析我们可以发现，歌曲的ID存在于a标签的href属性中，接下来我们实现获取即可，如下所示：

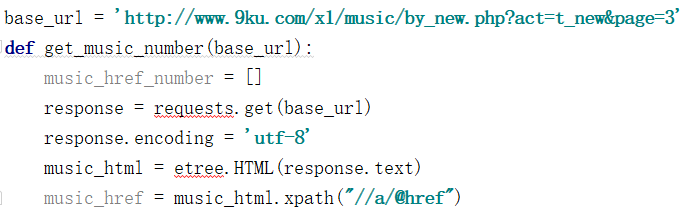


图 4-6 URL获取图

获取到href的属性值后，它是以字符串的形式存在的，我们可以直接使用切片的形式来获取每首歌曲的ID，然后存入列表即可，如下面代码所示：

for i in range(len(music\_href)):

music\_href\_number.append(music\_href[i].split('/')[-1].split('.')[0])

此时我们就已经完成了整个的爬取过程的分析。

## 4.5 本章小结

在此章节中我们分析了整个爬取过程，包括网页分析，歌曲详细链接的分析，最重要的是我们要找到歌曲的响应链接，而不是只有歌曲的链接，否则我们是没有办法找到歌曲链接的编制规则的。

# 第5章 Django实现数据库与前端的连接

## 5.1 Django介绍

Django是一个由Python写成的开源的Web框架，采用的是MTV的框架模式，即模型、模板和视图，三者之间负责不同的职责[4]。

模型：数据存储层，处理与数据相关的所有事物，例如如何存取、如何验证有效性、包含哪些行为及数据之间的关系等。

模板：业务逻辑层，存取模型及调取恰当模板的相关逻辑，模型与模板的桥梁[4]。

视图：表现层，处理与表现相关的决定，例如如何在页面或其他类型文档中进行显示[4]。

## 5.2 创建Django项目

创建Django项目用一行代码即可创建：

django-admin startproject music

我们会看到在music目录下会创建出music的项目，还有manage.py文件。如下图所示：

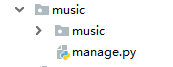


图 5-1 目录图

## 5.3 创建项目应用

在上一章节中我们创建了一个Django项目，紧接着我们需要创建一个我们的项目应用，即项目APP，它相当于网页的功能，每个APP代表一个或多个网页，具体命令为：

python manage.py startapp index

## 5.4 music文件配置

### 5.4.1 music文件介绍

在4.2章节中我们创建了一个music Django项目，我们会发现music文件夹下有如下所示的文件：

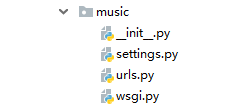


图 5-2 文件图

在这里我们不对每个文件详细进行你介绍，只针对settings.py文件和urls.py文件进行介绍。

settings.py是项目的配置文件，我们主要配置路径、APP和数据库即可。

urls.py是项目的URL设置，即网站的地址信息。

### 5.4.2 seetings.py 文件配置详情

我们已经创建了我们的项目应用，所以应该把项目应用添加到我们的配置文件中，如下图所示：



图 5-3 应用图

接下来是我们的模板文件配置，如下图：



图 5-4 文件配置图

下面是我们的数据库配置，我们不使用Django自带的sqlite3数据库，而是用我们的MySQL数据库，如下面所示：

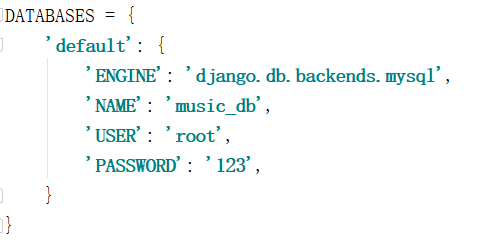


图 5-5 数据库连接图

在数据库配置中非常值得大家注意的是，NAME字段为我们创建的数据库名称，必须要事先在数据库中创建好，还能进行连接。还有就是不要添加HOST和PORT字段，我们在百度网上教程的时候百分之九十九都会添加这两个字段，所以大家在实际应用的时候需要谨慎，不要添加，否则在进行迁移的时候就会出现错误。

接下来是我们的中间件配置，如下面所示：



图 5-6 中文显示

中间件的作用主要是处理用户的请求信息。我们在这里添加的是LocaleMiddleware中间件，目的就是为了让Django内置的功能支持中文显示。

截至到目前为止，我们的setting.py文件已经配置完毕。

### 5.4.3 urls.py文件配置详情

在这里首先我们需要在我们的APP里面添加一个urls.py文件，以存放我们项目的URL，接着我们需要在根目录的urls.py中添加我们项目APP的URL，然后再在项目APP的urls.py中添加具体的URL信息。如下面所示：

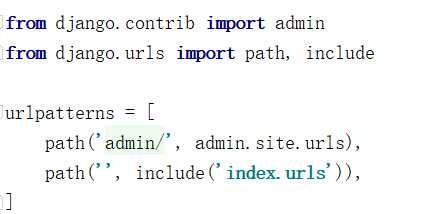


图 5-7 项目连接配置

值得注意的是在根目录的urls.py中最好这么写，有的网上的教程是直接写的导入APP下的views.py文件，这样很容易报错

## 5.5 index APP文件分析与配置

### 5.5.1 models.py的文件编写

在models.py文件中我们主要是定义数据表，我们不需要提前在数据库中创建数据表，这是我们需要注意的。详情代码如下所示：

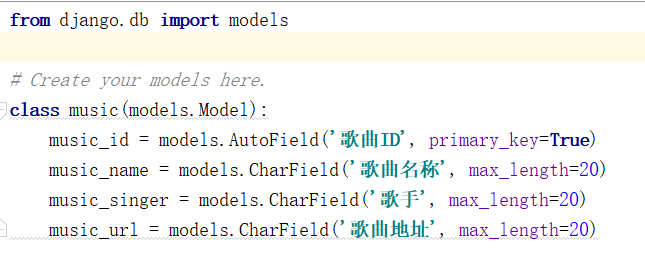


图 5-8 数据表配置

### 5.5.2 index.py的文件编写

详情代码如下所示：

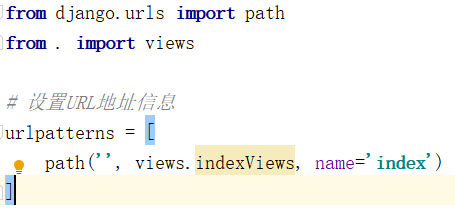


图 5-9 网页显示

### 5.5.3 数据迁移

因前端的需求，对于views.py文件的编写我们将不在这里展示，接下来我们对数据库进行迁移即可，在命令行中输入下面两行代码即可：

python manage.py makemigrations

python manage.py migrate

当迁移完成后数据库中会出现空表，我们利用Navicat数据库的图形化界面向数据表中导入数据即可。

# 参考文献

1. 安子建.基于Scrapy框架的网络爬虫实现与数据抓取分析[D].吉林大学：中国知网，2017年6月1日.
2. 零一，韩要宾，黄园园.Python 3 爬虫、数据清洗与可视化实战[M].北京：电子工业出版社，2018年9月.
3. 凯瑟琳﹒雅姆尔，理查德﹒劳森.用Python写网络爬虫[M].第二版.北京：人民邮电出版社，2018年12月.
4. 黄永祥.玩转Django2.0[M].北京：清华大学出版社，2018年10月.

# 致 谢

四年的读书生活在这个季节即将画上一个句号，而于我的人生却只是一个逗号，我将面对又一次征程的开始。回首往昔岁月，感慨万千。此时，我首先想感谢的是我王琪老师和陶卓老师，可以说，我的论文处处都凝结着这两位老师的心血和汗水。

从我的论文选题、开题、撰写、到论文的定稿，这一路走来，每一步都离不开这两位老师的谆谆教诲和悉心关怀。老师治学严谨、知识渊博、谦虚平和，与老师的每一次谈话都让我受益匪浅，在撰写论文的日子里，每每遇到难题，经过我冥思苦想还是无法解决的，与老师进行探讨，他们的几句话却能让我茅塞顿开!在这两年的时间里，这两位老师教会我的不仅是做学问的态度，更重要的是为人处事的方式，我将带着这些在接下来的工作和生活中，继续努力，不断锤炼自己、超越自己，让自己变得更加优秀。同样地，我也十分感谢软件学院的各位领导和老师们。

感谢河北师范大学软件学院的各位导师，对我的论文给予了悉心指导并提出了许多宝贵意见。

感谢我的家人，是他们一直鼓励我、支持我，使我有了不断克服困难，奋勇前进的动力。