Updating Database via Scrapy

Yanqiao Chen(12412115), SUSTech, Shenzhen
Instructor: Dr. Shiqi Yu

December 5, 2025

Contents

1 项目基本信息

1.1 使用工具

工具	版本	说明
Python	3.8.10	编程语言
Scrapy	2.5.1	爬虫框架
PostgesSQL	13.3	数据库管理系统

1.2 项目目标

通过爬虫爬取电影数据库 tmdb 电影和演员数据，更新旧有教学电影数据库中信息（自 2019 以后，根据查询可知电影数据库信息截至 2019 年）。限于 API 爬取速度限制，我只爬取了 2019 年以后的部分电影数据和其部分演员数据，如有需要更新之前的数据，只需要调整爬虫代码中的年份范围以及爬取限制即可，但是这可能需要更长的事件或者使用反爬机制，有可能有法律风险，因此此处并不尝试。

限于 API 速率限制，以及为“教学用”（轻量化、小型化）数据库的考量，我们只爬取每个年份的部分电影数据（数量不等）。如果后续需要增量式更新，本爬虫亦可以通过修改 peopleid 起始以及年份范围来进行增量式更新。

1.3 更新内容

更新了三个表单：movies, credits, people, 并且向 countries 中插入了“??”作为数据库未知国家的占位符。同时注意到，现有数据库中的国家代码”sp” 不符合 ISO 3166-1 标准，但是为了统一修改，将读取到的西班牙国家代码”es” 全部替换为”sp”。同时，我们添加了巴勒斯坦”ps” 作为国家代码。

2 项目过程

整体项目过程如下：

1. 向 tmdb 数据库申请 api 密钥
2. 编写 Scrapy 爬虫，爬取 2019 年以后的电影数据
3. 使用 psycopg2 导入数据库

2.1 申请 API

在 tmdb 官网注册账号后，进入设置页面，申请 API 密钥。申请过程中需要填写一些基本信息以及使用目的等。申请成功后，可以在设置页面查看到 API 密钥。

2.2 编写爬虫

我们本次使用 Scrapy 框架编写爬虫。首先创建一个新的 Scrapy 项目，然后创建一个新的爬虫，命名为 tmdb_spider 并存储在 spiders 文件夹下。然后，我们配置 settings.py 文件和 pipeline.py 文件，以便将爬取的数据存储到数据库中。最后，我们编写 tmdb_spider.py 文件，定义爬虫的行为和数据处理逻辑。

一个典型的爬虫框架如下：

- start_requests: 定义初始请求，通常是从某个 URL 开始爬取数据。
- parse: 处理响应数据，提取所需信息，并生成新的请求。
- item_pipeline: 处理提取的数据，进行清洗、存储等操作。

根据主要的几个表格的关系，我们首先从电影本身的信息开始爬取，然后对于每个电影，爬取其演员和工作人员的信息。最后，将所有数据存储到数据库中。

本次项目中，我进行了如下的操作：

1. 发起请求，获取 2019 年以后的电影列表。
2. 将请求转换为 JSON 格式，便于数据处理。
3. 解析电影数据，提取电影 ID、标题、简介、发布日期、评分等信息。
4. 对于每部电影，发起新的请求，获取其演员和工作人员信息。
5. 解析演员和工作人员数据，提取演员 ID、姓名、角色等信息。
6. 将提取的数据存储到数据库中。同时，为了避免重复存入演员信息（保证唯一的 peopleid），我们使用构造 sql 查询语句检查演员是否已经存在于数据库中。如果已经存在，那么 credit 表中应当使用已有的 peopleid，否则插入新的演员信息并获取新的 peopleid。

在实际爬取过程中，因为一些设置上的问题，我们进行了多次不同年份的爬取。由于单个年份的电影数据较多，且电影网站往往将某一年份的电影持续推送，因此我们在以后的爬取中可能需要设置单个年份的爬取上限以满足各年份均匀分布的要求。否则，爬取到的电影数据可能会集中在某些年份，导致数据库更新不均衡；如果需要更新全部 2019 年以后的数据，则需要更长的时间和更复杂的反爬机制。

2.3 数据库更新结果

最后结果显示，我们成功更新了 2019 年电影 4426 条，涉及演员和导演近两万人，并且成功将数据存储到数据库中。数据库的完整性和一致性得到了保证，所有数据均符合预期格式和要求。对于那些国家信息缺失的电影，我们使用了“??”作为占位符，确保数据库的完整性。

数据详见附录中的 SQL 文件。

3 课程资源的评估

3.1 Some Comments on the Lecture Notes

Summary Slides 在阅读 Lecture Notes 的时候，我发现在大部分 Slides 中，内容比较分散且不够系统化。在重新阅读复习的过程中，需要花费很大的精力区分例子和关键定义与注意项目。建议在 Lecture Notes 中，根据 Chapter 中不同的概念，增加一页 Summary 进行重点内容的总结和归纳，以此方便在课堂中以及复习过程中方便阅读和理解。

Slides 中的显示错误

- Chapter 2 Slide 13, 14, 15: 文字标题显示重叠；
- Chapter 2 Slide 23: 文字显示重叠；

笔误

- Chapter 2 Slide 38, 34: not null 前面错误的添加了逗号，导致语法错误；

3.2 Some Comments on the Original Filmdb

关于国家编号不符合 ISO-3166-1 的提示 在更新数据库的过程中，我发现现有 filmdb 数据库中的国家编号不符合 ISO-3166-1 标准。例如，西班牙的国家代码应为”es”，但在数据库中使用了”sp”。为了保持一致性，我在爬取数据时将西班牙的国家代码统一替换为”sp”。建议在数据库设计和维护过程中，严格遵守国际标准，以避免混淆和错误。由于 filmdb 中的数据库中原始数据来源不明，因此为了减小对于数据库现有数据的影响，在更新的过程中保持与数据库内部一致。但是建议在后续的数据库设计中，遵循国际标准。

4 一些课程建议

4.1 关于 Lecture

4.2 关于 Lab

4.3 关于 Project

4.4 关于课外阅读

5 总结

A Python Scrapy 爬虫代码

Listing 1: tmdb_spider.py 爬虫代码

```
import scrapy
import re


class TmdbSpider(scrapy.Spider):
    name = "tmdb"
    allowed_domains = ["api.themoviedb.org"]

    API_KEY = "Your API Key"

    def start_requests(self):
        url = "https://api.themoviedb.org/3/discover/movie"
        base_params = {
            'api_key': self.API_KEY,
            'sort_by': 'popularity.desc',
            'page': 1,
            'with_release_type': '2|3',
        }
        years = range(2020, 2021)

```

```
21          for year in years:
22              year_params = base_params.copy()
23              year_params['primary_release_year'] = year
24
25              query_string = '&'.join([f"{k}={v}" for k, v in year_params.items
                    ()])
26              full_url = f"{url}?{query_string}"
27
28              self.logger.info(f"Starting␣crawl␣for␣year:␣{year}")
29              yield scrapy.Request(url=full_url, callback=self.parse_discover)
30
31      def parse_discover(self, response):
32          try:
33              data = response.json()
34          except Exception as e:
35              self.logger.error(f"Failed␣to␣parse␣JSON␣from␣Discover:␣{e}")
36              return
37
38          for movie in data.get('results', []):
39              movie_id = movie.get('id')
40              detail_url = f"https://api.themoviedb.org/3/movie/{movie_id}?
                    api_key={self.API_KEY}"
41              yield scrapy.Request(
42                  url=detail_url,
43                  callback=self.parse_details,
44                  meta={'initial_data': movie}
45              )
46
47          current_page = data.get('page')
48          total_pages = data.get('total_pages')
49
50          if current_page is not None and current_page < total_pages:
51              if current_page >= 500:
52                  self.logger.warning("Reached␣TMDB␣API␣limit␣(Page␣500).␣
```

```
                Stopping pagination for this query.")
53          else:
54              next_page = current_page + 1
55              next_url = re.sub(r'page=\d+', f'page={next_page}', response.
                    url)
56              self.logger.info(f"Paging: Requesting page {next_page} of {
                    total_pages}")
57              yield scrapy.Request(url=next_url, callback=self.
                    parse_discover)
58
59      def parse_details(self, response):
60          initial_data = response.meta['initial_data']
61          try:
62              detail_data = response.json()
63          except Exception as e:
64              self.logger.error(f"Failed to parse JSON from Details: {e}")
65              return
66
67          item_data = {
68              'movieid': detail_data.get('id'),
69              'title': initial_data.get('title'),
70              'release_date': initial_data.get('release_date'),
71              'runtime': detail_data.get('runtime', 0) or 0
72          }
73
74          countries = detail_data.get('production_countries', [])
75          if countries:
76              code = countries[0].get('iso_3166_1')
77              if code == 'ES' or code == 'es':
78                  code = 'sp'
79              if code == 'KN' or code == 'kn':
80                  code = 'ke'
81              item_data['country'] = code.lower() if code else '??'
82          else:
```

```
83              item_data['country'] = '??'
84
85          credits_url = f"https://api.themoviedb.org/3/movie/{item_data['
                movieid']}/credits?api_key={self.API_KEY}"
86          yield scrapy.Request(
87              url=credits_url,
88              callback=self.parse_credits,
89              meta={'item_data': item_data}
90          )
91
92      def parse_credits(self, response):
93          from ..items import TmdbMovieItem
94
95          item_data = response.meta['item_data']
96          try:
97              credits_data = response.json()
98          except Exception as e:
99              self.logger.error(f"Failed to parse JSON from Credits: {e}")
100             return
101
102         item = TmdbMovieItem(**item_data)
103         processed_people = []
104
105         for p in credits_data.get('cast', [])[:5]:
106             processed_people.append({
107                 'tmdb_id': p.get('id'),
108                 'name': p.get('name'),
109                 'job': 'A',
110                 'gender': None,
111                 'born': None,
112                 'died': None
113             })
114
115         for p in credits_data.get('crew', []):
```

```
116             if p.get('job') == 'Director':
117                 processed_people.append({
118                     'tmdb_id': p.get('id'),
119                     'name': p.get('name'),
120                     'job': 'D',
121                     'gender': None,
122                     'born': None,
123                     'died': None
124                 })
125 
126         if processed_people:
127             item['cast_crew'] = processed_people
128 
129             first_person = processed_people[0]
130             person_url = f"https://api.themoviedb.org/3/person/{first_person
                ['tmdb_id']}?api_key={self.API_KEY}"
131 
132             yield scrapy.Request(
133                 url=person_url,
134                 callback=self.parse_person_details,
135                 meta={
136                     'item': item,
137                     'people_list': processed_people,
138                     'current_index': 0
139                 }
140             )
141         else:
142             item['cast_crew'] = []
143             yield item
144 
145     def parse_person_details(self, response):
146         meta = response.meta
147         try:
148             person_data = response.json()
```

```
149             except Exception as e:
150                 self.logger.error(f"Failed to parse JSON from Person: {e}")
151                 return
152 
153             item = meta['item']
154             people_list = meta['people_list']
155             current_index = meta['current_index']
156 
157             person = people_list[current_index]
158 
159             gender_code = person_data.get('gender')
160             if gender_code == 1:
161                 person['gender'] = 'F'
162             elif gender_code == 2:
163                 person['gender'] = 'M'
164             else:
165                 person['gender'] = '?'
166 
167             birthday = person_data.get('birthday')
168             if birthday:
169                 person['born'] = int(birthday[:4])
170             else:
171                 person['born'] = 0
172 
173             deathday = person_data.get('deathday')
174             if deathday:
175                 person['died'] = int(deathday[:4])
176             else:
177                 person['died'] = None
178 
179             next_index = current_index + 1
180 
181             if next_index < len(people_list):
182                 next_person = people_list[next_index]
```

```
            person_url = f"https://api.themoviedb.org/3/person/{next_person
                ['tmdb_id']}?api_key={self.API_KEY}"

            yield scrapy.Request(
                url=person_url,
                callback=self.parse_person_details,
                meta={
                    'item': item,
                    'people_list': people_list,
                    'current_index': next_index
                }
            )
        else:
            yield item
```

Listing 2: pipeline.py 数据库存储代码

```
import psycopg2
from scrapy.exceptions import DropItem
from .items import TmdbMovieItem


class PostgresPipeline:
    START_PEOPLE_ID = 26319 # 此处根据数据库实际调整
    people_cache = {}

    current_people_id = START_PEOPLE_ID

    @classmethod
    def from_crawler(cls, crawler):
        # 从 settings.py 中加载数据库配置
        db_settings = crawler.settings.getdict('DATABASE')
        return cls(db_settings)

    def __init__(self, db_settings):
```

```
19          self.db_settings = db_settings
20          self.conn = None
21          self.cursor = None
22
23      def open_spider(self, spider):
24          """爬虫开启时连接数据库"""
25          try:
26              self.conn = psycopg2.connect(
27                  host=self.db_settings['host'],
28                  port=self.db_settings['port'],
29                  database=self.db_settings['database'],
30                  user=self.db_settings['username'],
31                  password=self.db_settings['password']
32              )
33              # 关闭自动提交，手动控制事务
34              self.conn.autocommit = False
35              self.cursor = self.conn.cursor()
36              spider.logger.info("Database connection established successfully.
                    ")
37
38              try:
39                  self.cursor.execute(f"SELECT setval('people_peopleid_seq', {
                        self.START_PEOPLE_ID - 1}, true);")
40                  self.conn.commit()
41                  spider.logger.info(f"peopleid sequence set to start at {self.
                        START_PEOPLE_ID}.")
42              except psycopg2.Error as e:
43                  self.conn.rollback()
44                  spider.logger.warning(f"Could not set sequence (might not
                        exist): {e}")
45
46          except psycopg2.Error as e:
47              spider.logger.error(f"Database connection failed: {e}")
48              raise
```

```
49
50      def close_spider(self, spider):
51          """爬虫关闭时关闭连接"""
52          if self.conn:
53              self.conn.close()
54              spider.logger.info("Database connection closed.")
55
56
57      def _execute_sql(self, sql, params=None):
58          """执行 SQL 语句，出错时抛出异常由 process_item 捕获"""
59          self.cursor.execute(sql, params)
60
61      def process_item(self, item, spider):
62          if not isinstance(item, TmdbMovieItem):
63              return item
64
65          try:
66              # 1. 插入或更新 Movies 表 (父表)
67              self._insert_movie(item)
68
69              # 2. 遍历演职员，处理 People 和 Credits
70              for person_data in item.get('cast_crew', []):
71                  # 获取有效的人物 ID (查找现有或插入新人物)
72                  people_id = self._insert_or_lookup_person(person_data)
73
74                  # 3. 插入 Credits 表 (子表)
75                  self._insert_credit(item['movieid'], people_id, person_data['
                        job'])
76
77              # 4. 提交整个事务 (只有当所有步骤都成功时)
78              self.conn.commit()
79              # spider.logger.debug(f"Committed transaction for movie: {item['
                    movieid']}")
80
```

```
81          except psycopg2.Error as e:
82              self.conn.rollback()
83              spider.logger.error(
84                  f"DB␣Error␣processing␣movie␣{item.get('movieid')}:␣{e.pgerror
                        .strip()␣if␣e.pgerror␣else␣e}")
85
86          except Exception as e:
87              self.conn.rollback()
88              spider.logger.error(f"General␣Error␣processing␣movie␣{item.get('
                    movieid')}:␣{e}")
89
90          return item
91
92      def _insert_movie(self, item):
93          """插入或更新␣Movies␣表"""
94
95          country_code = item['country']
96
97          sql = """
98  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣INSERT␣INTO␣movies␣(movieid,␣title,␣country,␣year_released,␣
        runtime)
99  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣VALUES␣(%s,␣%s,␣%s,␣%s,␣%s)␣ON␣CONFLICT␣(movieid)␣DO␣\
100 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣UPDATE␣\
101 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SET␣title␣=␣EXCLUDED.title,␣runtime␣=␣EXCLUDED.runtime;␣\
102 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
103         params = (
104             item['movieid'],
105             item['title'],
106             country_code,
107             int(item['release_date'][:4]) if item['release_date'] else 0,
108             item['runtime']
109         )
110         self._execute_sql(sql, params)
111
```

```
112     def _insert_or_lookup_person(self, person_data):
113         """
114 ␣␣␣␣␣␣␣␣查找或插入人物，并返回␣peopleid。
115 ␣␣␣␣␣␣␣␣逻辑:
116 ␣␣␣␣␣␣␣␣1.␣检查本地缓存。
117 ␣␣␣␣␣␣␣␣2.␣检查数据库␣(SELECT)。
118 ␣␣␣␣␣␣␣␣3.␣如果不存在，插入新记录␣(INSERT)。
119 ␣␣␣␣␣␣␣␣"""
120         full_name = person_data.get('name')
121         parts = full_name.strip().split('␣', 1)
122         first_name = parts[0] if len(parts) > 1 else None
123         surname = parts[-1]
124         if len(parts) == 1:
125             first_name = None
126 
127         people_key = (surname, first_name)
128 
129         if people_key in self.people_cache:
130             return self.people_cache[people_key]
131 
132         lookup_sql = """
133 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣peopleid␣\
134 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣people
135 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣WHERE␣surname␣=␣%s␣\
136 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣AND␣first_name␣IS␣NOT␣DISTINCT␣\
137 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣%s;␣\
138 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
139         self._execute_sql(lookup_sql, (surname, first_name))
140         result = self.cursor.fetchone()
141 
142         if result:
143             people_id = result[0]
144             self.people_cache[people_key] = people_id
145             return people_id
```

```
146
147         assigned_id = self.current_people_id
148
149         born_year = person_data.get('born', 0)
150         died_year = person_data.get('died')
151         gender_char = person_data.get('gender', '?')
152
153         insert_sql = """
154 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣INSERT␣INTO␣people␣(peopleid,␣first_name,␣surname,␣
    born,␣died,␣gender)
155 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣VALUES␣(%s,␣%s,␣%s,␣%s,␣%s,␣%s);␣\
156 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
157         params = (
158             assigned_id,
159             first_name,
160             surname,
161             born_year,
162             died_year,
163             gender_char
164         )
165         self._execute_sql(insert_sql, params)
166
167         self.current_people_id += 1
168         self.people_cache[people_key] = assigned_id
169
170         return assigned_id
171
172     def _insert_credit(self, movieid, peopleid, credited_as):
173         sql = """
174 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣INSERT␣INTO␣credits␣(movieid,␣peopleid,␣credited_as)
175 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣VALUES␣(%s,␣%s,␣%s)␣ON␣CONFLICT␣DO␣NOTHING;␣\
176 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"""
177         params = (movieid, peopleid, credited_as)
178         self._execute_sql(sql, params)
```

Listing 3: items.py Item 定义代码

```python
import scrapy


class TmdbMovieItem(scrapy.Item):
    # Movies 表字段
    movieid = scrapy.Field()
    title = scrapy.Field()
    release_date = scrapy.Field()
    country = scrapy.Field()
    runtime = scrapy.Field()

    cast_crew = scrapy.Field()
```

Listing 4: settings.py 数据库配置代码

```

BOT_NAME = 'tmdb_scraper'

SPIDER_MODULES = ['project2.spiders']
NEWSPIDER_MODULE = 'project2.spiders'

USER_AGENT = 'tmdb_scraper'

ROBOTSTXT_OBEY = True


CONCURRENT_REQUESTS = 4
(4 * 0.25 = 1秒)
DOWNLOAD_DELAY = 0.25

AUTOTHROTTLE_ENABLED = True
AUTOTHROTTLE_START_DELAY = 5.0
AUTOTHROTTLE_MAX_DELAY = 60.0
```

```
19 AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0
20
21 DATABASE = {
22     'drivername': 'postgresql',
23     'host': 'localhost',
24     'port': '5432',
25     'database': 'project2',
26     'username': 'postgres',
27     'password': '1234'
28 }
29
30 ITEM_PIPELINES = {
31     'project2.pipelines.PostgresPipeline': 300,
32 }
33
34 LOG_LEVEL = 'INFO'
35
36 RETRY_TIMES = 3
37
38 COOKIES_ENABLED = False
```

B 更新后的数据库导出的 SQL 文件