**作业 Yaoyu Pan**

**请完成三个抓取目标 100分**

1. 版面列表抓取
2. 主题列表抓取
3. 主题内容发帖回帖详情抓取

**提交方式：**

**请按照以下7个步骤，提交对应的代码压缩包文件，以及每个步骤对应的描述文档。**

**以下是评分标准：**

**1.建表语句 15分**

1. 版面列表             3分

tables['board'] =(

"CREATE TABLE `board` (" #创建一个名为board的表

"`name` varchar(64) NOT NULL," #版面名称

"`url` varchar(1024) NOT NULL," #版面url

"`mgr\_id` varchar(32) NOT NULL," #版主id

"`mgr\_url` varchar(512) NOT NULL," #版主url

"`topics\_cnt` int(11) NOT NULL," #主题数量

"`posts\_cnt` int(11) NOT NULL," #发帖数量

" PRIMARY KEY (`name`)," #将board名称设为主键

" UNIQUE (`url`)" #board的url设置为唯一约束

") ENGINE=InnoDB" #数据库存储引擎

)

1. 主题列表             4分

tables['topic'] = (

"CREATE TABLE `topic` (" #创建topic表

" `id` int AUTO\_INCREMENT," #序列号

" `title` varchar(128) NOT NULL," #文章标题

" `url` varchar(1024) NOT NULL," #文章的url

" `author\_id` varchar(32) NOT NULL," #作者的名称

" `author\_url` varchar(32) NOT NULL," #作者的url

" `status` varchar(32) NOT NULL DEFAULT 'new'," #状态

" `rating` int NOT NULL DEFAULT 0," #评分

" `like\_cnt` int NOT NULL DEFAULT 0," #点赞数量

" `reply\_cnt` int NOT NULL DEFAULT 0," #回帖数量

" `publish\_time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP," #发布的时间

" `queue\_time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP," #开始存储的时间

" `done\_time` timestamp NOT NULL DEFAULT '1970-01-02 00:00:00' ON UPDATE CURRENT\_TIMESTAMP,"#爬取并存储完成的时间

"PRIMARY KEY (`id`)," #将topic序列号作为主键

"UNIQUE (`url`)" #将topic的url设置为唯一约束

") ENGINE = InnoDB" #数据库存储引擎

1. 主题内容发帖回帖详情         3分

tables['post'] = (

"CREATE TABLE `post`(" #创建post表

"`topic\_id` varchar(16) NOT NULL," #所要存储的主题的序列号

"`content` varchar(10240) NOT NULL DEFAULT ''," #回帖的内容

"`post\_index` int NOT NULL default 0," #回帖的索引号

"UNIQUE(`topic\_id`, `post\_index`)" #将主题序列号和帖子的索引号设置为主键

") ENGINE=InnoDB" #数据库存储引擎

)

1. 请描述表的作用以及每个字段的含义       5分

版面列表：根据BBS上面的版面信息以及对应的网址，提取相应的版块信息，包括版面的名称，网址，版主信息，网址，主题数和发帖的数目。

主题列表：根据前面爬取的版面列表，进行下一步提取，得到当前版块下发表的文章列表，包括文章的题目，网址，发表的时间，作者的信息，网址，评分，点赞的数目和回帖的数目。

主题内容发帖列表：根据前面爬取的文章列表，提取出当前文章的回帖详情，包括每一楼的回复信息。

**2.数据库和表的创建及数据的插入，查询，更新等封装方法和类。25分**

1. 请先用一段话描述代码编写思路5分

使用MySQL Server 数据库创建表储存爬取的数据，需要先建立远程连接，然后使用mysql-connector-python 中的cursor操作，将指令传输到远端的服务器上，将每个列表爬取的信息及时的插入到新建的表中。首先创建一个smth数据库和board，topic，post三个列表，然后先插入board爬取的信息，然后可以根据爬取的结果进行topic的爬取，此时将topic列表信息也存储到服务器上，在进行post内容爬取时需要进行目标锁定，即先将要爬取的topic状态由‘new’改为一个唯一值，然后调用程序进行爬取，完成之后再讲状态更新为‘done’。这样可以避免重复爬取同一个topic。对于数据的查询和更新，当爬取的贴子有新的回复的时候，需要进行更新。此时可以先将之前爬取的数据备份，然后将对应的topic删除重新进行爬取。

1. 数据连接池编写 5分

def \_\_init\_\_(self, size\_pool):

try:

cnx = mysql.connector.connect(user=self.dbconfig['user'], password=self.dbconfig['password'], auth\_plugin='mysql\_native\_password')

except mysql.connector.Error as err:

print(err)

return

cursor = cnx.cursor()

try:

cnx.database = self.dbconfig['database']

except mysql.connector.Error as err:

print(err)

#if err.errno == errorcode.ER\_BAD\_DB\_ERROR:

self.create\_database(cursor)

cnx.database = self.dbconfig['database']

self.create\_tables(cursor)

finally:

cursor.close()

cnx.close()

self.cnxpool = mysql.connector.pooling.MySQLConnectionPool(

pool\_name='smth\_pool',

\*\*self.dbconfig

)

1. Python实现插入代码 5分

def insert\_topic(self, topic):

con = self.cnxpool.get\_connection()

cursor = con.cursor()

try:

if re.match(r'\d\d:\d\d:\d\d', topic['publish\_time']):

topic['publish\_time'] = (time.strftime("%Y-%m-%d ", time.localtime()) + topic['publish\_time']).strip()

sql = ("INSERT INTO topic(title, url, author\_id, author\_url, publish\_time,"

"rating, like\_cnt, reply\_cnt) "

"VALUES ('{}', '{}', '{}', '{}', '{}', '{}', '{}', '{}' )").format(

topic['title'], topic['url'], topic['author\_id'],

topic['author\_url'], topic['publish\_time'], topic['rating'],

topic['num\_likes'], topic['num\_replies'])

# print(sql)

cursor.execute((sql))

con.commit()

return True

except mysql.connector.Error as err:

# print('insert\_topic() ' + err.msg)

# print("Already exist!")

return False

finally:

cursor.close()

con.close()

def insert\_board(self, board):

try:

con = self.cnxpool.get\_connection()

cursor = con.cursor()

sql = ("INSERT INTO board(url, name, mgr\_id, mgr\_url, topics\_cnt, posts\_cnt)"

"VALUES( '{}', '{}', '{}', '{}', '{}', '{}' )").format(board['board\_url'], board['board\_title'], board['manager\_id'],

board['manager\_url'], board['num\_topics'], board['num\_posts'])

cursor.execute(sql)

con.commit()

return True

except mysql.connector.Error as err:

print('insert\_board', err)

return False

finally:

cursor.close()

con.close()

def insert\_post(self, post):

con = self.cnxpool.get\_connection()

cursor = con.cursor()

try:

sql = "INSERT INTO post(topic\_id, content, post\_index) VALUES ('{}', '{}', '{}' )".format(

post['topic\_id'], post['content'], post['post\_index'])

#print(sql)

cursor.execute((sql))

con.commit()

return True

except mysql.connector.Error as err:

print('insert\_post()' + err.msg)

#print("Already exist!")

return False

finally:

cursor.close()

con.close()

1. Python实现出队业务逻辑代码 5分

def dequeue\_topic(self):

con = self.cnxpool.get\_connection()

cursor = con.cursor()

try:

con.start\_transaction()

const\_id = ('%.9f' % time.time())[4:]

update\_sql = ("UPDATE topic SET status='{}'"

"WHERE status='new' LIMIT 1").format(const\_id)

cursor.execute(update\_sql)

con.commit()

select\_sql = ("SELECT url, id FROM topic WHERE status='{}'").format(const\_id)

cursor.execute(select\_sql)

row = cursor.fetchone()

if row is None:

return None

return row

except mysql.connector.Error as err:

print('dequeue\_topic() ' + err.msg)

return False

finally:

cursor.close()

con.close()

1. Python实现更新状态为完成代码 5分

def finish\_topic(self, index):

con = self.cnxpool.get\_connection()

cursor = con.cursor()

try:

sql = "UPDATE topic SET status='done' WHERE id={}".format(index)

#print(sql)

cursor.execute((sql))

con.commit()

except mysql.connector.Error as err:

print('insert\_post()' + err.msg)

#print("Already exist!")

finally:

cursor.close()

con.close()

**3.分页关键逻辑处理      15分**

1. 请先用一段话描述代码编写思路 5分

做分页处理时需要找到最大的页数，可以通过标签进行查找。通常是倒数第二个标签，如果是最后几页，就是倒数第一个标签。如果只有一页，返回1，即不需要分页。

1. 关键代码截图展示10分

图片包含 屏幕截图

自动生成的说明

**4.多线程处理逻辑   15分**

1. 请先用一段话描述代码编写思路5分

需要多线程处理时，将每个爬取任务分配给多个线程同时进行，可以提高爬取的效率。首先需要定义最大线程数，然后分配任务，建立一个任务池。如果分配的任务已经超过最大线程数，则需要进行等待。为了降低爬取的频率，还需要计算等待的时间。

1. 关键代码截图展示10分

图片包含 屏幕截图, 文字

自动生成的说明

**5.图片资源处理逻辑      15分**

1. 请先用一段话描述代码编写思路    5分

爬取图片时，可以先找到网站的html源代码里面图片对应的标签信息，然后直接用re方法进行提取并保存获取的图片url地址。再利用request方法提取出url里面的content内容 保存。为了加快爬取的速度，还可以使用多线程。

1. 关键代码截图展示        10分

3图片包含 屏幕截图

自动生成的说明

**6.请你描述你的爬虫项目执行逻辑      5分**

1. 给出脚本启动命令        2分

首先连接MySQL server, mysql -uroot -ppassword ，然后调用mysql\_manager.py，分别运行boards.py， topics.py，posts.py， 提取版面列表，主题列表以及文章的回帖信息，并插入到数据库中保存。

1. 爬虫程序执行顺序        1分

python3 boards.py

python3 topics.py

python3 posts.py

1. 给出关键类和方法        2分

class MysqlManager():

def create\_database(self, cursor):

def create\_tables(self, cursor):

def insert\_topic(self, topic):

def insert\_board(self, board):

def insert\_post(self, post):

def dequeue\_topic(self):

def finish\_topic(self, index):

class BoardsCrawler:

def get\_board\_of\_section(self, section\_idx):

def get\_board\_list(self, etr\_obj = None ):

class TopicsCrawler:

def get\_content(self, board\_url, page):

def get\_max\_page(self):

def get\_topic\_list(self):

class PostsCrawler:

def get\_content(self, topic\_url, page):

def get\_max\_page(self):

def get\_posts(self):

**7.数据存入mysql数据库 10分**

**1）请使用select count(\*) from 表名语句输出条数。**

1. 版面列表  2分（不少于100条数据）

select count(\*) from board;

图片包含 屏幕截图

自动生成的说明

1. 主题列表  2分（不少于100条数据）

select count(\*) from topic;

图片包含 屏幕截图

自动生成的说明

1. 主题内容发帖回帖详情  2分（不少于100条数据）

select count(\*) from post;

图片包含 屏幕截图

自动生成的说明

**2）从你爬取的数据中，使用sql语句查询出以下数据：**

1. 回帖数最多的文章主题      2分

select \* from topic order by reply\_cnt desc limit 1;

图片包含 屏幕截图

自动生成的说明

1. 主题数最多的版面            2分

select \* from board order by topics\_cnt desc limit 1;

图片包含 屏幕截图, 监视器, 草, 户外

自动生成的说明

**请提供sql语句以及sql语句执行结果截图**