**实 验 报 告**

**课程名称** \_\_\_\_\_ Python程序设计\_\_\_\_\_\_\_\_\_

**实验项目** \_\_\_爬取百度百科等网站，例如明朝皇帝，并将人物关系以适当形式显示出来。

**实验仪器** \_\_\_\_\_\_\_\_\_\_\_PC\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**学 院**\_\_\_\_\_\_\_\_计算机学院\_\_\_\_\_\_\_

**专 业**\_\_\_\_

**班级/学号\_**

**学生姓名** \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**实验日期** \_\_\_\_\_

**成 绩** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**指导教师** \_\_\_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_\_

目录

[一、 实验目的 2](#_Toc153225242)

[二、 实验内容 2](#_Toc153225243)

[2.1网络爬虫 2](#_Toc153225244)

[2.2网络爬虫基本思路 2](#_Toc153225245)

[2.3Web相关技术简介 3](#_Toc153225246)

[2.3.1 HTML 3](#_Toc153225247)

[2.3.2CSS 3](#_Toc153225248)

[2.3.3Java Script 3](#_Toc153225249)

[2.3.4三者关系 3](#_Toc153225250)

[2.4 环境配置 3](#_Toc153225251)

[三、 实验原理-网络爬虫系统设计 5](#_Toc153225252)

[3.1 整体框架 5](#_Toc153225253)

[3.2 时序图 5](#_Toc153225254)

[3.3 URL管理器 6](#_Toc153225255)

[3.4 网页解析器 6](#_Toc153225256)

[3.4.1 Dom树 6](#_Toc153225257)

[3.5 调度器 7](#_Toc153225258)

[3.6 爬虫 7](#_Toc153225259)

[3.7 项目管道 7](#_Toc153225260)

[3.8 Scrapy框架整体流程 8](#_Toc153225261)

[四、 实验步骤-网络爬虫Scrapy框架模块实现 9](#_Toc153225262)

[4.1 爬虫类（spdiders.king.py） 9](#_Toc153225263)

[4.2 实体类（items.py） 12](#_Toc153225264)

[4.3 中间组件（middlewares.py） 13](#_Toc153225265)

[4.4 管道类（Pipelines.py） 15](#_Toc153225266)

[4.5 数据库（Pipelines.py） 15](#_Toc153225267)

[4.6 启动类(main.py) 17](#_Toc153225268)

[4.7 相关设置（settings.py） 17](#_Toc153225269)

[4.8 数据展示（过程及MySql数据库） 17](#_Toc153225270)

[五、 实验心得与体会: 17](#_Toc153225271)

[附录：程序源代码 17](#_Toc153225272)

[Spider 17](#_Toc153225273)

[Items 22](#_Toc153225274)

[Pipelines 22](#_Toc153225275)

[Settings 24](#_Toc153225276)

[Main 27](#_Toc153225277)

## 实验目的

通过本实验，能够读取数据并完成数据的分析或显示。要求掌握python语言的基本数据类型列表、字典的使用和操作，以及文件的读写操作，并用python语言完成实际任务。

利用Scrapy框架，爬取wiki百科上的从清朝皇帝开始的所有皇帝的人物关系并以适当形式展现出来。

## 实验内容

## 2.1网络爬虫

网络爬虫（也称为网络蜘蛛、网络机器人或网络脚本）是一种自动化程序，设计用于在互联网上浏览并检索信息。网络爬虫通过访问网页并收集数据，然后将这些数据用于各种目的，如建立搜索引擎索引、数据挖掘、或其他自动化任务。

网络爬虫在搜索引擎中起着关键作用，通过定期爬取互联网上的网页并建立索引，使搜索引擎能够快速响应用户的查询。然而，网络爬虫也可能被滥用，例如用于非法目的，如爬取个人信息或大规模抓取网站内容，这可能导致服务器负载过重或侵犯隐私。因此，有些网站在其robots.txt文件中规定了哪些页面可以被爬取，哪些不可以。

## 2.2网络爬虫基本思路

1. **访问页面**: 爬虫按照一定的规则访问网页，可以是深度优先或广度优先，以获取页面上的内容
2. **解析内容:** 爬虫分析所访问页面的内容，提取有用的信息，如文本、链接图像等
3. **存储数据:** 爬虫将提取的数据存储在本地或远程数据库中，以备后续分析或检索
4. **跟踪链接:** 爬虫继续跟踪页面上的链接，不断扩展其访问范围
5. **选择起始点:** 爬虫首先需要一个或多个起始网址，从这些网址开始浏览。
6. **整理数据:** 对搜索到的数据进一步整理，并以适当格式呈现给用户

## 2.3Web相关技术简介

### 2.3.1 HTML

HTML（Hypertext Markup Language）是一种标记语言，用于创建和组织网页的结构和内容。HTML是网页的基础，它由一系列标记（markup）组成，这些标记描述了文档的结构，例如标题、段落、链接、图像等。HTML标记告诉浏览器如何呈现页面的内容和布局。

### 2.3.2CSS

CSS（Cascading Style Sheets）是一种用于描述文档样式（如布局、颜色、字体等）的样式表语言。它与HTML一起被用于设计和格式化网页，使得开发者能够更好地控制页面的外观和布局。CSS通过定义规则，指定如何渲染HTML元素，从而实现页面的样式和设计。

### 2.3.3Java Script

JavaScript是一种高级的、解释型的编程语言，用于在网页上实现动态交互效果。它是一种脚本语言，可以通过嵌入在HTML中，由浏览器解释和执行。JavaScript使开发者能够在用户与网页交互时动态地改变网页的内容、结构和样式，以及响应用户的操作。

### 2.3.4三者关系

HTML提供网页的基本结构和内容，是网页骨架。CSS负责网页的外观的布局。JavaScript负责处理交互和行为，以实现一定的逻辑和行为

## 2.4 环境配置

以Windows系统为例，

1. 安装并配置Python，这里我安装的是Anaconda集成环境
2. 配置环境变量。

在windows的控制面板\系统和安全\系统中，点击“高级系统设置”，然后点击“环境变量”，如图2.1所示，编辑“Path”变量。

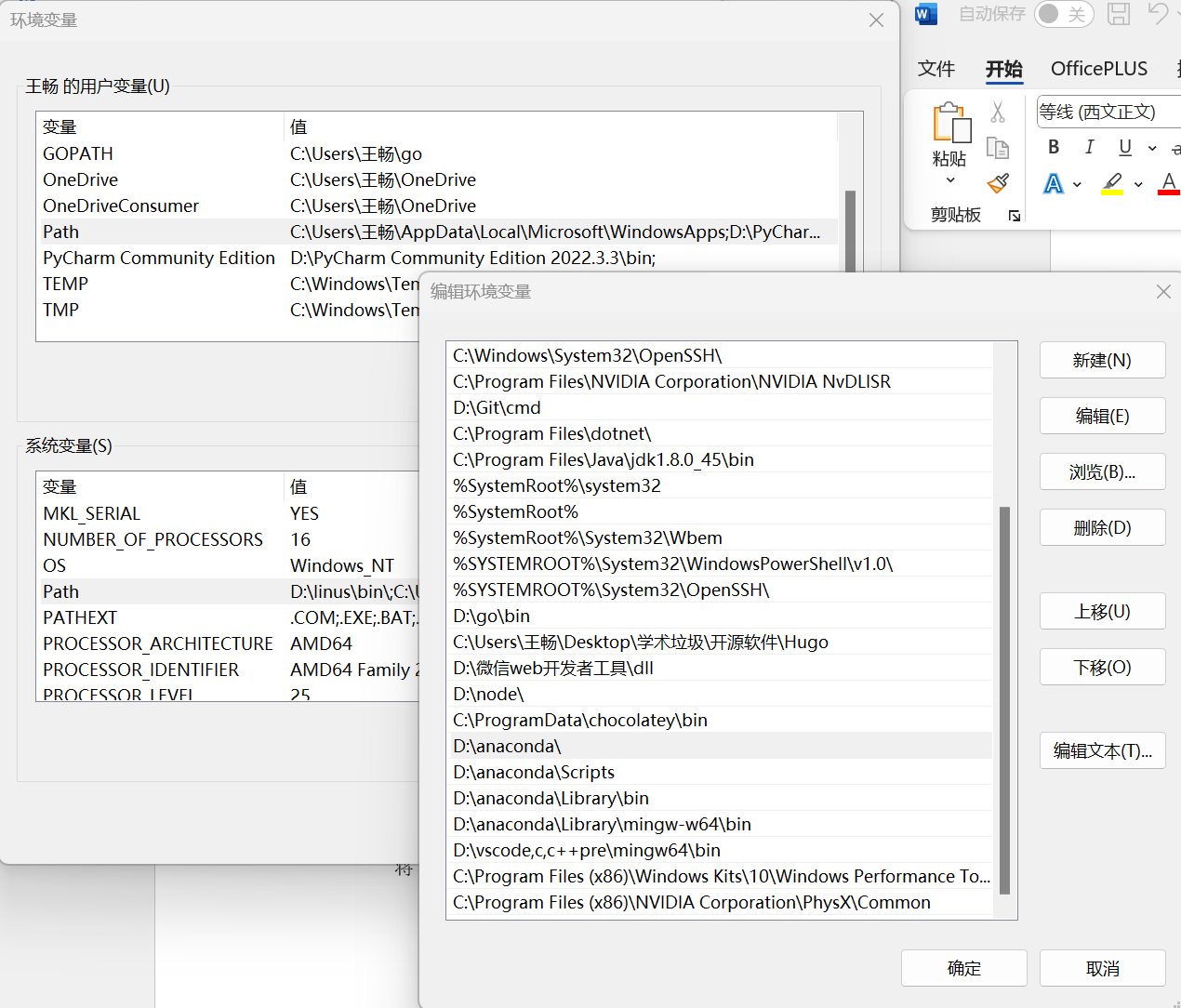


图2.1 环境变量对话框

将anaconda相关目录添加到Path变量中，如图2.1所示

1. 安装其他组件，如Beautiful Soup 4 等库，集成环境Anaconda已安装好，如图2.2所示。

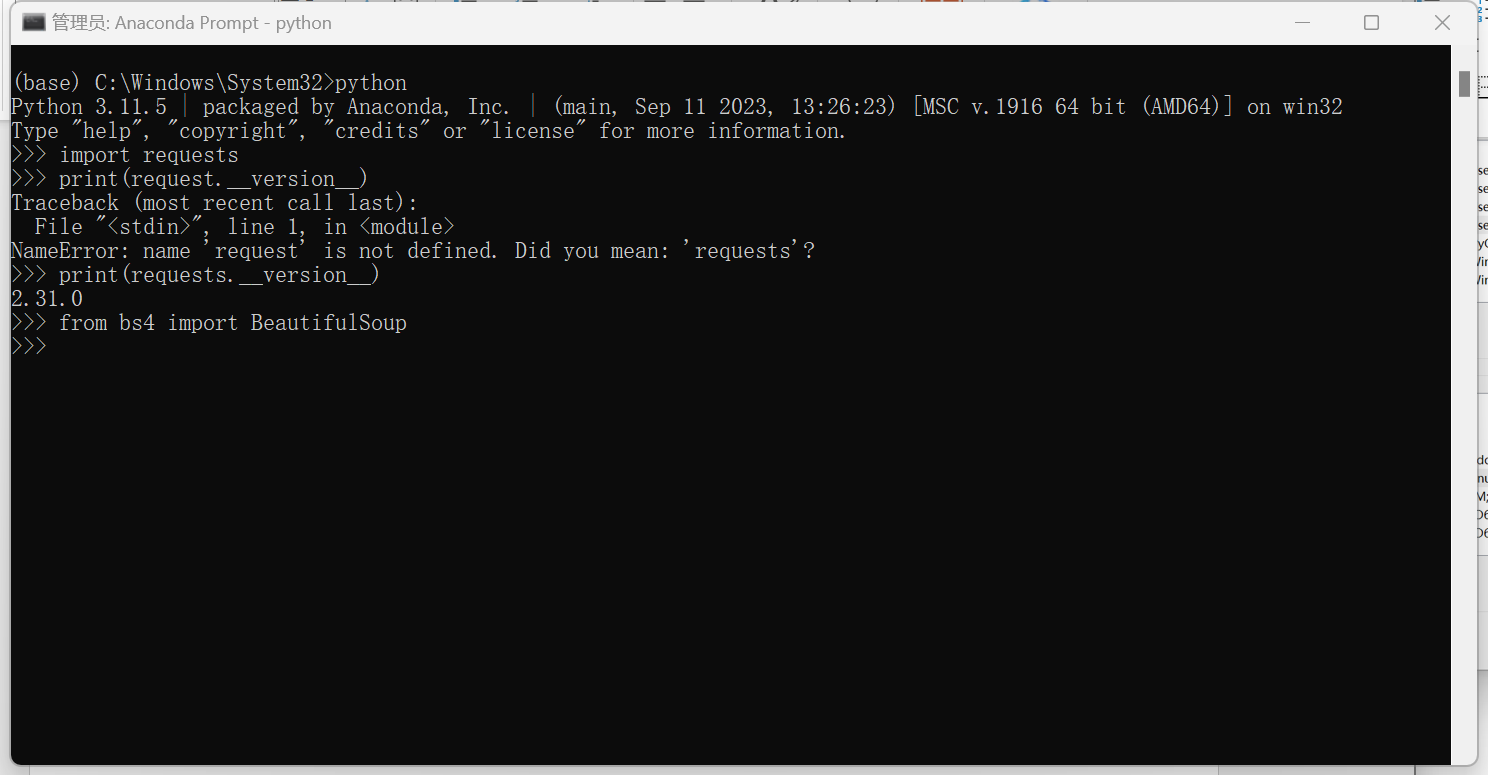


图2.2 检验组件安装

## 实验原理-网络爬虫系统设计

## 3.1 整体框架

整体框架如图3.1所示。



图3.1 简单网络爬虫整体框架

* 调度器负责整体系统的调度运行。
* URL管理器负责处理将要搜索的网页URL。
* 网页下载器实现对网页的下载。
* 网页解析器对下载的网页内容进行解析，取出我们感兴趣的数据和网页URL，将数据送往数据库，将网页URL送往URL管理器。
* 数据库存放从网页中解析出来的数据。这些数据具有对用户来说存在价值。
* 数据展示部分向用户以适当的格式展示搜索出来的数据，包括列表、图表等，例如Google、百度等以列表形式展示包含用户提供的关键字网页URL等。

## 3.2 时序图

简单网络爬虫时序图如图3.2所示。各个部分在此不再赘述，详见课程PPT讲解。



图3.2 简单网络爬虫时序图

## URL管理器

URL管理器负责维护网页URL，这里需要解决几个问题：

1. 如何维护将要下载的网页URL？
2. 如何防止重复和循环下载？如果某个网页已经被下载过，则不需要重新下载。

为解决以上问题，这里设计了两个列表，即未下载URL列表和已下载URL列表。



图3.3 URL管理器数据结构

可以简单地将其理解为一个未下载网页URL队列和一个已下载网页URL集合。

* 未下载网页URL队列满足先进先出，里面存放还没有下载的网页URL列表。每个要搜索的网页URL依次被加入到队尾，取值则从队头取出。
* 已下载网页集合则保存已经下载的网页URL。

系统运行时，首先从未下载网页URL队头取出一个URL，送入到网页下载器，将网页的内容下载下来，然后将该URL添加到已下载URL集合中，即记录该URL为已下载。如果有新来的网页URL，则首先检查是否在已下载URL集合中存在，如果存在，则说明已经下载过，不需要重新下载；否则将其加入到未下载网页URL队列队尾。

## 3.4 网页解析器

网页解析器的作用是解析网页内容，取得两方面的数据：

1. 价值数据，即网页中用户感兴趣的数据
2. URL列表，即网页中包含的超链接列表，这些URL是其它网页

### 3.4.1 Dom树

网页是用html编写的，浏览器接收到网页后，对其进行解析，表示成文档对象模型（Document Object Model，简称DOM）格式。DOM实际上是以面向对象方式来描述文档的模型，该模型通常表示成一棵树。

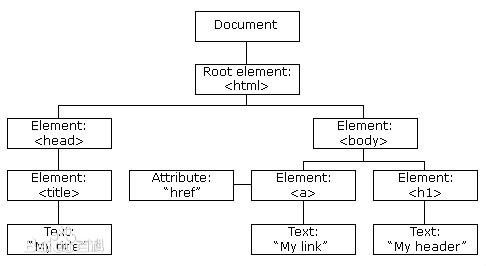


图3.4 DOM树

## 3.5 调度器

调度器的作用是在适当的时候调用各模块。下面给出调度器算法：

S1:URL管理器中是否存在未下载URL，如果不存在，转到S4；

S2:从URL管理器中取得一个未下载URL，调用网页下载器下载相应网页内容，同时通知URL管理器将该URL标记为已经下载；

S3:调用解析器分析下载的内容，获取价值数据和该网页包含的URL列表，然后分别将价值数据送往数据库，将URL列表送往URL管理器。然后转到S1；

S4:根据用户用户要求，得到相关数据，并以适当格式返回给用户端。

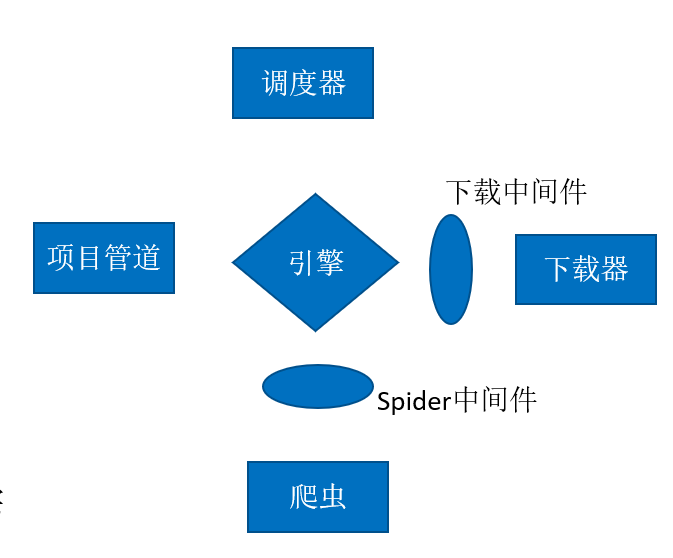
## 爬虫

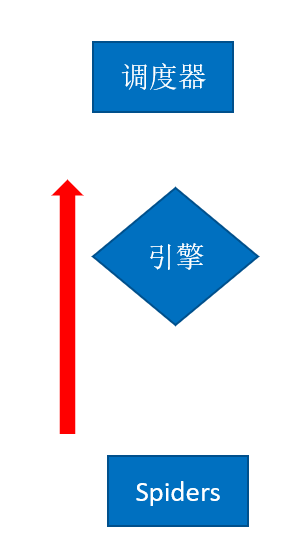
用于从特定的网页中提取自己需要的信息, 即所谓的实体(Item)；用户也可以从中提取出链接，让Scrapy继续抓取下一个页面

## 项目管道

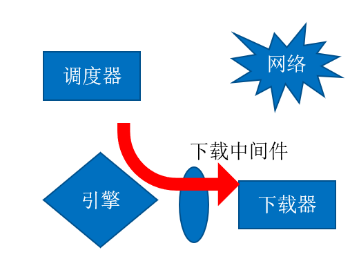
负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息

## Scrapy框架整体流程

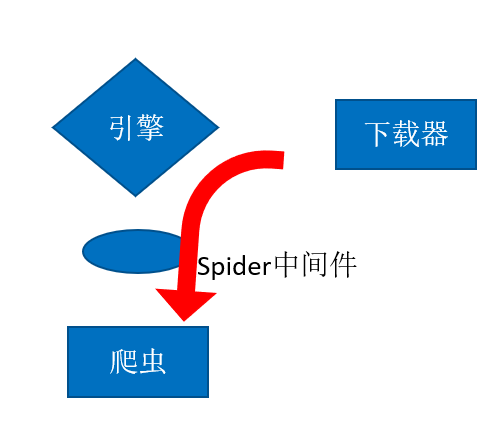


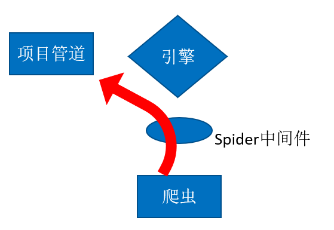
1. 引擎：（打开网站，找到处理该网站的Spider）Hi！Spider,

你第一个爬取的是哪个URL？

1. Spider：第一个URL是xxxx.com；
2. 引擎：调度器，我这有request请求，帮我排序入队；
3. 调度器：好的
4. 引擎：Hi！调度器，把你处理好的request请求给我。
5. 调度器：给你，这是我处理好的request
6. 引擎：Hi！下载器，你按照下载中间件的设置帮我下载一下这个request请求；
7. 下载器：好的！我现在下载；
8. 下载器： Hi！引擎，给你下载好的东西。

（如果失败：sorry，这个request下载失败了。然后引擎告诉调度器，这个request下载失败了，你记录一下，我们待会儿再下载）



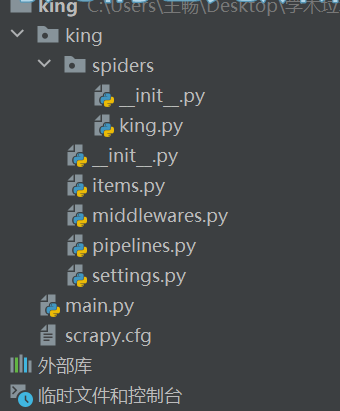
1. 引擎：Hi！Spider，这是下载好的东西，并且已经按照Spider中间件处理过了，你自己处理一下
2. Spider：Hi！引擎，我这里有两个结果，一个是需要跟进的新的URL，另一个是获取到的Item数据。
3. 引擎：
   1. Hi ！管道，我这儿有个item你帮我处理一下；
   2. Hi！调度器，这是需要跟进URL你帮我处理下；
   3. 从第3步开始循环，直到调度器中没有更多的request，引擎关闭网站
4. 管道调度器：好的，现在就做！保存到数据库或文件

## 实验步骤-网络爬虫Scrapy框架模块实现

网络爬虫Scrapy框架包括了引擎（Scrapy）,调度器（Scheduler）,下载器（Downloader）,爬虫（Spiders）,项目管道（Pipeline）。

**注：算法，流程，知识点，核心代码分析在代码中以注释的形式出现为主**

框架一览：



## 4.1 爬虫类（spdiders.king.py）

初始化爬虫类的基本属性，如命名，允许域名，起始地址



然后对页面内容进行解析并获取相应数据：







## 4.2 实体类（items.py）

用来封装爬取的对象属性，并保存下来：



## 4.3 中间组件（middlewares.py）

中间组件是在Scrapy引擎与下载器之间的处理组件，用于自定义、干预请求和响应的处理流程。中间件能够在请求被发送到下载器和响应被返回给引擎之前进行一系列的操作。以下是本次实验可能所需要的中间组件。



## 4.4 管道类（Pipelines.py）

管道是用于处理爬取到的数据的组件。管道负责处理由Spider爬取到的数据，可以进行数据清洗和加工，数据的存储和去重验证等操作，在这次实验中，我们主要用数据清洗和加工并把数据保存到MySql数据库的操作，以下为该部分实现简单管道类.



## 4.5 数据库（Pipelines.py）







## 4.6 启动类(main.py)

用于启动scrapy框架下的指定爬虫类

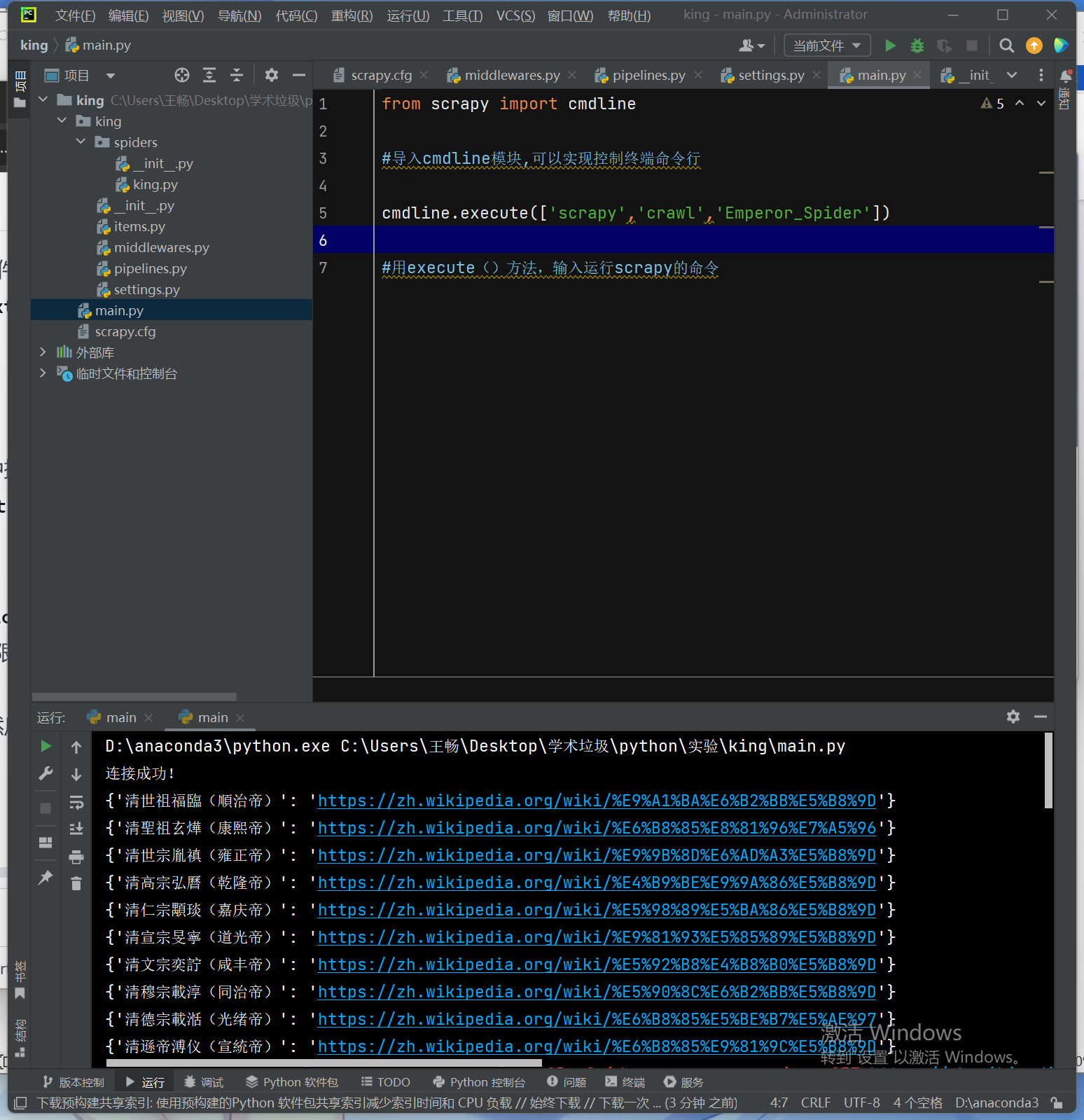


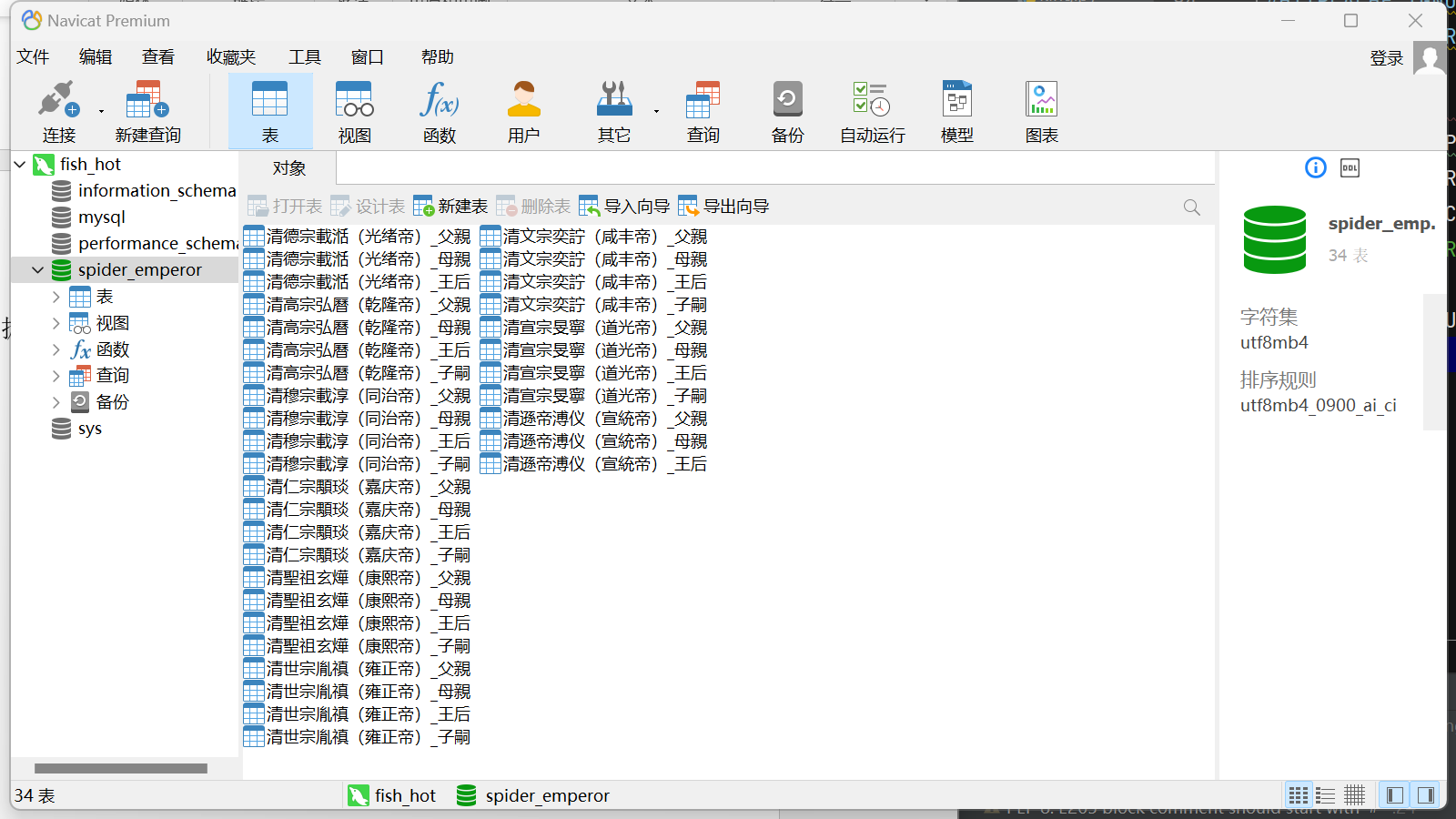
## 4.7 相关设置（settings.py）

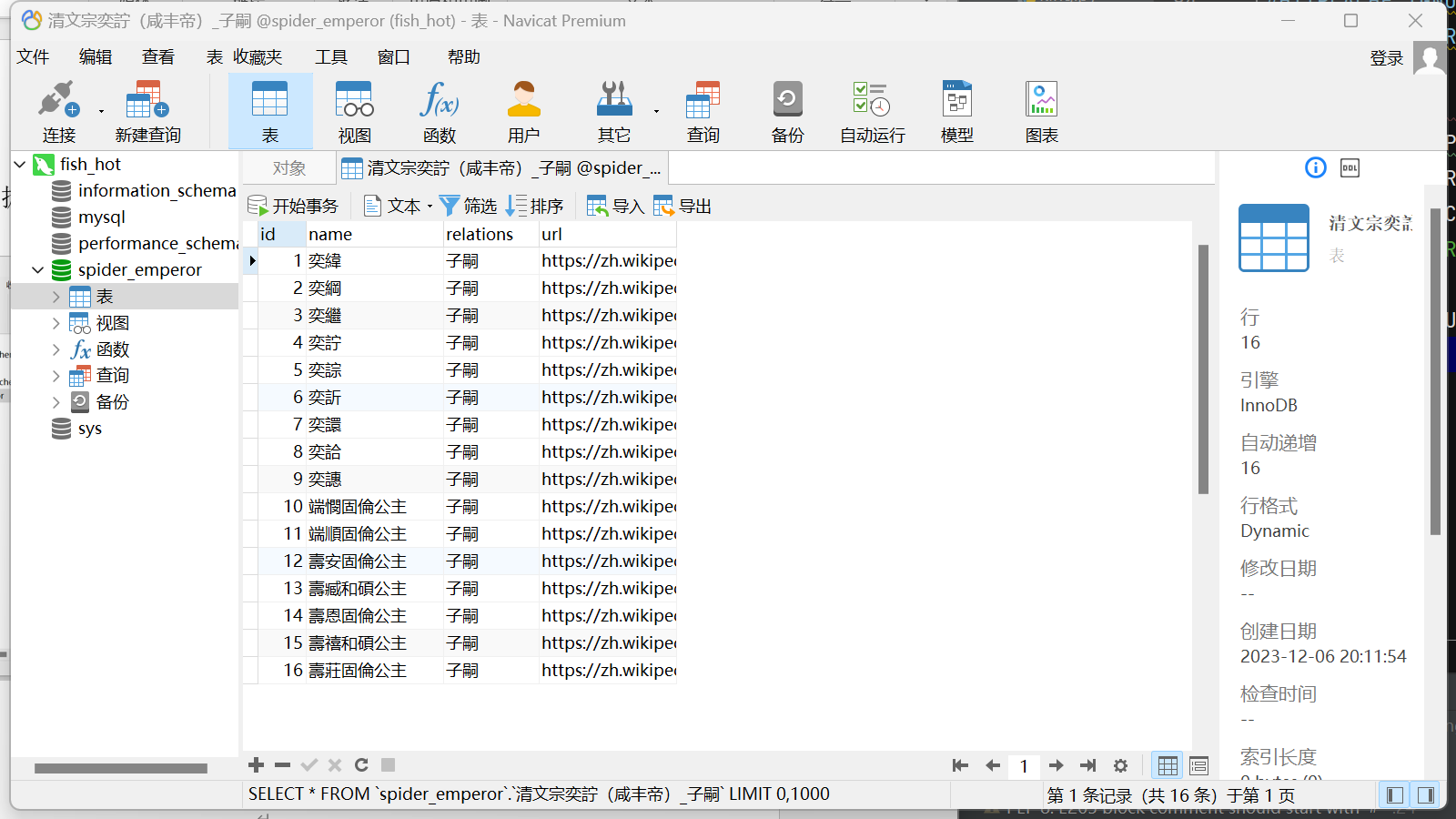
在设置里用来设置一些初始属性和外部要求或命令，这里仅展现该实验所需要的设置内容，其余无关的因内容过长，省略。



## 4.8 数据展示（过程及MySql数据库）







## 实验心得与体会:

这次实验，我选择了Scrapy爬虫框架，并构建了基本的爬取流程。通过分析清朝皇帝的wiki百科页面HTML结构，定位到了目标信息的位置，使用BeautifulSoup进行解析和提取。

在提取人物关系信息时，我使用了正则表达式和CSS选择器等方法，逐一获取了父母、配偶、子嗣等关系的数据。并且获取了历代皇帝的数据，并把每个皇帝的跳转页面传到下一个程序，通过在不同页面之间进行跳转，获取了清朝不同时期的皇帝关系，获得了相应的数据。

获取到数据后，我进行了一系列的数据处理，包括清理不规则的文本、处理缺失数据、等，以确保最后得到的数据集是需要的有用的。最后，把数据按照一定的结构存到MySql数据库中。

通过这次实验，我对Scrapy框架、BeautifulSoup库、正则表达式等工具有了更深入的了解，同时也培养了我处理数据的能力。

## 附录：程序源代码

## Spider

import scrapy  
import bs4  
import re  
from ..items import KingItem  
  
class EmperorSpider(scrapy.Spider):  
 # 定义一个爬虫类EmperorSpider  
 name = 'Emperor\_Spider'  
 # name是定义爬虫的名字，这个名字是爬虫的唯一标识。  
  
 allowed\_domains = ['zh.wikipedia.org']  
 # allowed\_domains是定义允许爬虫爬取的网址域名（不需要加https://）。  
 # 如果网址的域名不在这个列表里，就会被过滤掉。  
  
 start\_urls = ['https://zh.wikipedia.org/wiki/%E7%9A%87%E5%A4%AA%E6%A5%B5']  
 # start\_urls是定义起始网址，就是爬虫从哪个网址开始抓取。  
 # 其中的乱码为16进制的朱元璋  
 # 并且allowed\_domains的设定对start\_urls里的网址不会有影响。  
 '''  
 def parse(self, response):  
 # 使用BeautifulSoup解析HTML响应  
 bs = bs4.BeautifulSoup(response.text, 'html.parser')  
  
 # 定义皇帝字典{名字，网址}  
 emperor\_list = {}  
 emperor\_name = ''  
  
 # 使用正则表达式匹配妃后和公主的模式  
 pattern = re.compile('.\*(妃|后)$')  
 pattern1 = re.compile('.\*公主$')  
  
 # 爬取历任皇帝  
 emperor\_next = bs.find('table', {'class': ['infobox', 'vcard']}).find('tbody').findAll('tr')  
 for tr in emperor\_next:  
 # 查找继任的<th>元素  
 emp = tr.find('th', {'scope': 'row', 'style': 'text-align:left;white-space:nowrap;;'}, text='繼任')  
 if emp:  
 # 获取皇帝的名字和网址  
 emp\_name = tr.find('td').text.strip()  
 emp\_url = 'https://zh.wikipedia.org' + tr.find('td').find('a')['href']  
  
 # 更新皇帝字典  
 emperor\_list.update({emp\_name: emp\_url})  
 emperor\_name = next(iter(emperor\_list.keys()))  
 break  
  
 # 打印皇帝字典  
 print(emperor\_list)  
  
 # 发送请求，继续爬取继任皇帝的详细信息  
 yield scrapy.Request(url=emp\_url, callback=self.parse)  
  
 # 定义关系列表  
 relations = ['父親', '母親', '王后', '子嗣']  
  
 # 遍历关系列表  
 for relation in relations:  
 # 遍历历任皇帝的<tr>元素  
 for tr in emperor\_next:  
 # 查找关系对应的<th>元素  
 table = tr.find('th', {'scope': 'row', 'style': 'text-align:left;white-space:nowrap;;'}, text=relation)  
  
 if table:  
 # 获取<td>元素，包含关系对应的人物信息  
 td\_element = tr.findNext('td')  
 # 获取<a>元素列表  
 a\_element = td\_element.findAll('a')  
  
 # 处理不同关系的情况  
 if relation == '子嗣':  
 # 处理子嗣关系，过滤掉不符合条件的人物  
 for i, a in enumerate(a\_element):  
 emp\_name = a.getText()  
 if not pattern1.match(emp\_name):  
 if i % 2 != 0 or pattern.match(emp\_name):  
 continue  
 emp\_name = a.getText()  
 emp\_url = a['href']  
  
 # 创建KingItem对象，表示人物信息  
 people = KingItem(host=emperor\_name, url='https://zh.wikipedia.org' + emp\_url,  
 name=emp\_name,  
 relations=relation)  
 # 返回人物信息  
 yield people  
 else:  
 # 处理其他关系，遍历<a>元素列表  
 for i, a in enumerate(a\_element):  
 # 过滤掉不符合条件的人物  
 if i % 2 != 0:  
 continue  
 emp\_name = a.getText()  
 emp\_url = a['href']  
  
 # 创建KingItem对象，表示人物信息  
 people = KingItem(host=emperor\_name, url='https://zh.wikipedia.org' + emp\_url,  
 name=emp\_name,  
 relations=relation)  
 # 返回人物信息  
 yield people  
 '''  
 def parse(self**,** response):  
 # parse是Scrapy里默认处理response的一个方法。  
 bs = bs4.BeautifulSoup(response.text**,** 'html.parser')  
 # 定义皇帝字典{名字，网址}  
 emperor\_list = {}  
 emperor\_name=''  
 pattern = re.compile('.\*(妃|后)$')  
 pattern1=re.compile('.\*公主$')  
 # 爬取历任皇帝  
 emperor\_next = bs.find('table'**,** {'class': ['infobox'**,** 'vcard']}).find('tbody').findAll('tr')  
 for tr in emperor\_next:  
 emp = tr.find('th'**,** {'scope': 'row'**,** 'style': 'text-align:left;white-space:nowrap;;'}**,** text='繼任')  
 if emp:  
 emp\_name = tr.find('td').text.strip()  
 emp\_url = 'https://zh.wikipedia.org'+tr.find('td').find('a')['href']  
  
 emperor\_list.update({emp\_name: emp\_url})  
 emperor\_name = next(iter(emperor\_list.keys()))  
 break  
 print(emperor\_list)  
 yield scrapy.Request(url=emp\_url**,** callback=self.parse)  
  
 relations=['父親'**,**'母親'**,**'王后'**,**'子嗣']  
  
 for relation in relations:  
  
 for tr in emperor\_next:  
 table=tr.find('th'**,** {'scope': 'row'**,** 'style': 'text-align:left;white-space:nowrap;;'}**,** text=relation)  
  
 if table:  
 td\_element = tr.findNext('td')  
 a\_element=td\_element.findAll('a')  
 if relation=='子嗣':  
 for i**,**a in enumerate(a\_element):  
 emp\_name = a.getText()  
 if not pattern1.match(emp\_name):  
 if i % **2** != **0** or pattern.match(emp\_name):  
 continue  
 emp\_name=a.getText()  
 emp\_url=a['href']  
  
 people=KingItem(host=emperor\_name**,**url='https://zh.wikipedia.org'+emp\_url**,**name=emp\_name**,**relations=relation)  
 yield people  
 else:  
 for i**,**a in enumerate(a\_element):  
 if i % **2** != **0**:  
 continue  
 emp\_name=a.getText()  
 emp\_url=a['href']  
  
 people=KingItem(host=emperor\_name**,**url='https://zh.wikipedia.org'+emp\_url**,**name=emp\_name**,**relations=relation)  
 yield people

## Items

# Define here the models for your scraped items  
#  
# See documentation in:  
# https://docs.scrapy.org/en/latest/topics/items.html  
  
import scrapy  
  
  
class KingItem(scrapy.Item):  
 # define the fields for your item here like:  
 # 定义关系人  
 host = scrapy.Field()  
 # 定义关系  
 relations = scrapy.Field()  
 # 定义名字  
 name = scrapy.Field()  
 # 定义目标网址  
 url = scrapy.Field()

## Pipelines

# Define your item pipelines here  
#  
# Don't forget to add your pipeline to the ITEM\_PIPELINES setting  
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html  
  
  
# useful for handling different item types with a single interface  
from itemadapter import ItemAdapter  
import pymysql  
from scrapy.exceptions import DropItem  
  
class KingPipeline:  
 def process\_item(self**,** item**,** spider):  
 return item  
class MySQLPipeline:  
 def \_\_init\_\_(self):  
 self.connection = pymysql.connect(host='localhost'**,** user='root'**,** password='qrewsdfa'**,** database='spider\_emperor'**,** charset='utf8'**,** use\_unicode=True) # 连接数据库  
 self.cursor = self.connection.cursor() # 创建游标  
 print('连接成功！')  
 def process\_item(self**,**item**,**spider):  
 # 确保Item中的字段不为空  
 if not all(item[field] for field in item.fields):  
 raise DropItem("Item is missing required fields.")  
  
 # 动态选择表名  
 table\_name = f"{item['host']+'\_'+item['relations']}"  
  
 # 创建表（如果不存在）  
 self.create\_table\_if\_not\_exists(table\_name**,**f"{item['host']+'\_'+item['relations']}")  
  
 # 执行插入操作  
 sql = f"INSERT INTO {table\_name} ( name, relations,url) VALUES (%s, %s, %s)"  
 values = ( item["name"]**,** item["relations"]**,**item["url"])  
  
 try:  
 self.cursor.execute(sql**,** values)  
 self.connection.commit()  
 except Exception as e:  
 self.connection.rollback()  
 raise DropItem(f"Failed to insert item into MySQL: {e}")  
  
 return item  
  
 def is\_group\_exists(self**,** group\_name):  
 # 执行 SHOW TABLES 查询  
 show\_tables\_sql = f"SHOW TABLES LIKE '{group\_name}%'"  
 self.cursor.execute(show\_tables\_sql)  
 result = self.cursor.fetchone()  
  
 # 如果查询结果不为空，说明组存在  
 return result is not None  
  
 def create\_table\_if\_not\_exists(self**,** table\_name**,** default\_group="朱元璋"):  
 # 检查表名是否已经包含了默认组名前缀  
 if not table\_name.startswith(default\_group):  
 table\_name = f"{default\_group}\_{table\_name}"  
  
 # 创建表的SQL语句，你需要根据你的需求修改字段类型和约束  
 create\_table\_sql = f"""  
 CREATE TABLE IF NOT EXISTS {table\_name} (  
 id INT AUTO\_INCREMENT PRIMARY KEY,  
 name VARCHAR(255),  
 relations VARCHAR(255),  
 url VARCHAR(255)  
 )  
 """  
  
 # 执行创建表的SQL语句  
 self.cursor.execute(create\_table\_sql)  
 self.connection.commit()  
  
 def open\_spider(self**,** spider):  
 self.connection = pymysql.connect(host='localhost'**,** user='root'**,** password='qrewsdfa'**,** database='spider\_emperor'**,** charset='utf8'**,** use\_unicode=True) # 连接数据库  
 self.cursor = self.connection.cursor()  
  
 def close\_spider(self**,**spider):  
 self.cursor.close() #关闭游标  
 self.connection.close() #关闭数据库

## Settings

# Scrapy settings for king project  
#  
# For simplicity, this file contains only settings considered important or  
# commonly used. You can find more settings consulting the documentation:  
#  
# https://docs.scrapy.org/en/latest/topics/settings.html  
# https://docs.scrapy.org/en/latest/topics/downloader-middleware.html  
# https://docs.scrapy.org/en/latest/topics/spider-middleware.html  
  
BOT\_NAME = "king"  
  
SPIDER\_MODULES = ["king.spiders"]  
NEWSPIDER\_MODULE = "king.spiders"  
  
  
# Crawl responsibly by identifying yourself (and your website) on the user-agent  
#USER\_AGENT = "king (+http://www.yourdomain.com)"  
USER\_AGENT = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'  
  
# Obey robots.txt rules  
ROBOTSTXT\_OBEY = False  
  
# Configure maximum concurrent requests performed by Scrapy (default: 16)  
#CONCURRENT\_REQUESTS = 32  
  
# Configure a delay for requests for the same website (default: 0)  
# See https://docs.scrapy.org/en/latest/topics/settings.html#download-delay  
# See also autothrottle settings and docs  
#DOWNLOAD\_DELAY = 3  
# The download delay setting will honor only one of:  
#CONCURRENT\_REQUESTS\_PER\_DOMAIN = 16  
#CONCURRENT\_REQUESTS\_PER\_IP = 16  
  
# Disable cookies (enabled by default)  
#COOKIES\_ENABLED = False  
  
# Disable Telnet Console (enabled by default)  
#TELNETCONSOLE\_ENABLED = False  
  
# Override the default request headers:  
#DEFAULT\_REQUEST\_HEADERS = {  
# "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8",  
# "Accept-Language": "en",  
#}  
  
# Enable or disable spider middlewares  
# See https://docs.scrapy.org/en/latest/topics/spider-middleware.html  
#SPIDER\_MIDDLEWARES = {  
# "king.middlewares.KingSpiderMiddleware": 543,  
#}  
  
# Enable or disable downloader middlewares  
# See https://docs.scrapy.org/en/latest/topics/downloader-middleware.html  
#DOWNLOADER\_MIDDLEWARES = {  
# "king.middlewares.KingDownloaderMiddleware": 543,  
#}  
  
# Enable or disable extensions  
# See https://docs.scrapy.org/en/latest/topics/extensions.html  
#EXTENSIONS = {  
# "scrapy.extensions.telnet.TelnetConsole": None,  
#}  
  
# Configure item pipelines  
# See https://docs.scrapy.org/en/latest/topics/item-pipeline.html  
ITEM\_PIPELINES = {  
# "king.pipelines.KingPipeline": 300,  
 'king.pipelines.MySQLPipeline':**301,**}  
MYSQL\_HOST = 'localhost'  
MYSQL\_DATABASE = 'spider\_emperor'  
MYSQL\_USER = 'root'  
MYSQL\_PASS = 'qrewsdfa'  
MYSQL\_PORT = **3306**TABLE\_NAME = 'relation'  
# Enable and configure the AutoThrottle extension (disabled by default)  
# See https://docs.scrapy.org/en/latest/topics/autothrottle.html  
#AUTOTHROTTLE\_ENABLED = True  
# The initial download delay  
#AUTOTHROTTLE\_START\_DELAY = 5  
# The maximum download delay to be set in case of high latencies  
#AUTOTHROTTLE\_MAX\_DELAY = 60  
# The average number of requests Scrapy should be sending in parallel to  
# each remote server  
#AUTOTHROTTLE\_TARGET\_CONCURRENCY = 1.0  
# Enable showing throttling stats for every response received:  
#AUTOTHROTTLE\_DEBUG = False  
  
# Enable and configure HTTP caching (disabled by default)  
# See https://docs.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-middleware-settings  
#HTTPCACHE\_ENABLED = True  
#HTTPCACHE\_EXPIRATION\_SECS = 0  
#HTTPCACHE\_DIR = "httpcache"  
#HTTPCACHE\_IGNORE\_HTTP\_CODES = []  
#HTTPCACHE\_STORAGE = "scrapy.extensions.httpcache.FilesystemCacheStorage"  
  
# Set settings whose default value is deprecated to a future-proof value  
REQUEST\_FINGERPRINTER\_IMPLEMENTATION = "2.7"  
TWISTED\_REACTOR = "twisted.internet.asyncioreactor.AsyncioSelectorReactor"  
FEED\_EXPORT\_ENCODING = "utf-8"  
LOG\_LEVEL = 'ERROR'  
# 并发量  
CONCURRENT\_REQUESTS = **1**#下载延迟  
DOWNLOAD\_DELAY = **1**

## Main

from scrapy import cmdline  
  
#导入cmdline模块,可以实现控制终端命令行  
  
cmdline.execute(['scrapy'**,**'crawl'**,**'Emperor\_Spider'])  
  
#用execute（）方法，输入运行scrapy的命令