Python 爬虫

1.1常用请求头信息：

User-Agent：请求载体的身份标识

Connection: 请求完毕后，是断开连接还是保持连接

1.2响应头信息：

Content-Type:服务器相应回客户端的数据类型

1.3 https协议：安全的超文本传输协议（服务器和客户端进行数据交互的一种形式）

1.4 加密方式

对称密钥加密：客户端将密钥和密文同时发送给服务器端

非对称密钥加密：服务器端先将加密方式/公钥发送给客户端，客户端加密完文本后再返回。

但缺点是得确保公钥不被拦截篡改。

证书密钥加密：加入证书认证机构，服务器的公钥被数字签名，然后证书认证机构将密文与证书一并发送给客户端。

1. requests模块：模拟浏览器发请求
   1. 编码流程：

指定url，即网址->发起请求->获取响应数据->持久化存储响应数据

* 1. 第一个源码：爬取b站页面数据

import requests

# 指定url

url = 'https://www.bilibili.com/'

# 发送请求，get方法会返回一个响应对象

respose = requests.get(url)

# 获取响应数据，即页面html源代码

pageText = respose.text

print(pageText)

# 持久化存储

with open('./bilibili.html', 'w', encoding='utf-8') as fp:

fp.write(pageText)

print("爬取结束！！！")

* 1. 简易网页采集器

UA:User-Agent 获取方式：浏览器输入about:version eg:edge浏览器的是：

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 Edg/116.0.1938.81

#UA检测：门户网站的服务器会监测对应请求的载体身份标识，如果检测到请求的载体身份标识为某一款浏览器，则说明该请求为正常请求。否则，为不正常请求，服务器端就很有可能拒绝请求。

#反反爬策略：UA伪装：让爬虫对应的请求身份载体身份标识伪装成某一款浏览器

import requests

if \_\_name\_\_ == '\_\_main\_\_':

# UA伪装:将相应的User-Agent封装到一个字典中

headers ={

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36'

}

# 指定url web后面的?可有可无

url = 'https://www.sogou.com/web'

# 处理url携带的参数：封装到字典中

kw = input('请输入你要查询的关键词：')

param = {

'query': kw

}

# 对指定url发送起的请求对应的url是带参数的，并且请求过程中处理了参数

respose = requests.get(url=url, params=param, headers=headers)

# 获取响应数据，即页面html源代码

pageText = respose.text

print(pageText)

fileName = kw + '.html'

# 持久化存储

with open(fileName, 'w', encoding='utf-8') as fp:

fp.write(pageText)

print(fileName, "保存成功！！！")

* 1. 破解百度翻译

（url = **'https://fanyi.baidu.com/sug'）**

#post请求（携带参数）

#响应的数据是json数据 import json

步骤：

输入对应的postUrl->进行UA伪装->处理post携带参数->请求发送->持久化存储到json数据

#处理post携带参数

word=input(‘enter a word’)

data={

‘kw’:word

}

#请求发送

respose = requests.post(url=postUrl, data=data, headers=headers)

#持久化存储到json数据，在确定响应类型为json的情况下

dic\_obj=response.json()

fileName =word + ‘json’

fp =open(fileName,’w’,encoding=’utf-8’)

json.dump(dic\_obj,fp=fp,ensure\_ascii=False)

* 1. 豆瓣电影排行get，param需要参数
  2. 药监总局获得许可证信息

动态加载的数据不可直接获得

3.1 数据解析分类

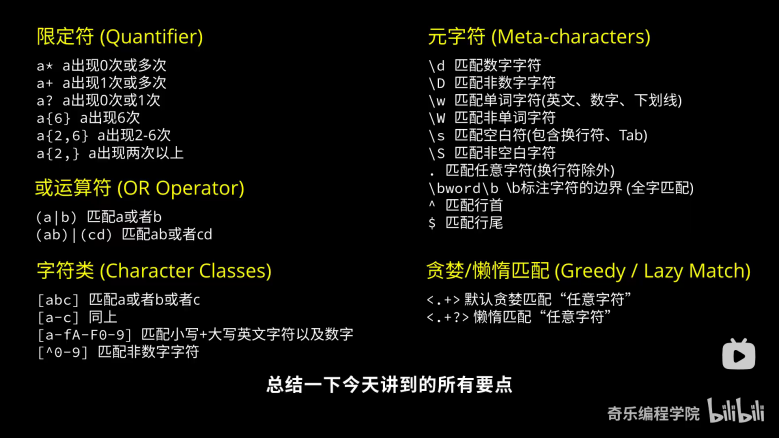
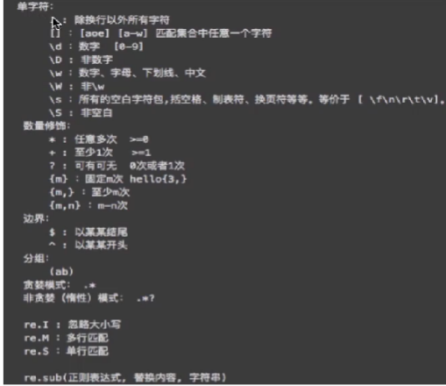
原理：解析的局部文本都会在标签之间或者标签对应的属性之中

步骤：进行指定标签定位->标签或标签对应的属性中的数据值进行提取（解析）

1. 正则

Eg.匹配十六进制RGB值 #[a-fA-F0-9]{6}\b

Eg.IP地址的匹配 \b((25[0-5]|2[0-4]\d|[0-1]?\d\d?)+\.){3}(25[0-5]|2[0-4]\d|[0-1]?\d\d?)\b



\B 匹配不是单词开头或结束的位置

<a[^>]+>匹配用尖括号括起来的以a开头的字符串。

后向引用：\n,n的范围是1-9，匹配被子表达式 n 匹配的相同字符串，下表为分组语法：

|  |  |  |
| --- | --- | --- |
| 分类 | 代码/语法 | 说明 |
| **捕获** | （exp) | 匹配exp,并捕获文本到自动命名的组里 |
| (?<name>exp) | 匹配exp,并捕获文本到名称为name的组里，也可以写成(?'name'exp) |
| (?:exp) | 匹配exp,不捕获匹配的文本，也不给此分组分配组号,占位 |
| **零宽断言** | (?=exp)  **零宽度正预测先行断言** | 匹配exp前面的位置 |
| (?<=exp)  **零宽度正回顾后发断言** | 匹配exp后面的位置 |
| (?!exp)  **零宽度负预测先行断言** | 匹配后面跟的不是exp的位置 |
| (?<!exp)  **零宽度负回顾后发断言** | 匹配前面不是exp的位置 |
| **注释** | (?#comment) | 这种类型的分组不对正则表达式的处理产生任何影响，用于提供注释让人阅读 |

\b(\w+)\b\s+\1\b可以用来匹配重复的单词，像go go, 或者kitty kitty。这个表达式首先是一个单词，也就是单词开始处和结束处之间的多于一个的字母或数字(\b(\w+)\b)，这个单词会被捕获到编号为1的分组中，然后是1个或几个空白符(\s+)，最后是分组1中捕获的内容（也就是前面匹配的那个单词）(\1)。

(?<=\s)\d+(?=\s)匹配以空白符间隔的数字(再次强调，不包括这些空白符)。

(?<=<(\w+)>).\*(?=<\/\1>)匹配不包含属性的简单HTML标签内里的内容

3.2 爬取图片

#url为图片对应地址

# content获取的是二进制形式的图片数据，text（字符串），json是对象

img\_data=requests.get(url=url).content

with open(‘./a.jpg’, ’wb’)as fp:

fp.write(img.data)

3.3正则案例:爬取糗图

import re

import os

#创建文件夹保存图片

if not os.path.exists('./qiutuLibs'):

os.mkdir('./qiutuLibs')

#通用url模板

url='https://www.qiushibaike.com/pic/page/%d/?s=5184961'

for pageNum in range(1,36):

new\_url=format(url%pageNum)

page\_text=requests.get(url=new\_url,headers=headers).text

#使用聚焦爬虫对该页面里的所有图片进行解析

**ex='<div class="thumb">.\*?<img src="(.\*?)" alt.\*?</div>'**

img\_src\_list=re.findall(ex,page\_text,re.S)

#图片地址缺少协议头

for src in img\_src:

src=’https:’+src

img\_data=requests.get(url=src,headers=headers).content

#生成图片名称

img\_name=src.split('/')[-1]

imgPath='./qiutuLibs'+img\_name

with open(imgPath,’wb’) as fp:

fp.write(img\_data)

3.4 bs4是python专用数据解析

步骤：实例化BeatifulSoup对象，将页面源码加载到该对象中->

调用BeatifulSoup对象中相关属性或者方法进行标签定位和数据提取

需pip install bs4和lxml（解析器）

1. 实例化BeatifulSoup对象

#将本地中的html文档中的数据加载到该对象中

fp=open('./text.html','lxml',encoding='utf-8')

soup=BeatifulSoup(fp,'lxml')

#将互联网上获取的页面源码加载到该对象中（常用）

page\_text=response.text

soup=BeatifulSoup(page\_text,'lxml')

1. 提供的用于数据解析的方法和属性

soup.tagName=soup.find('tagName'):返回html中第一次出现的tagName标签

#标签定位：

class要做属性值得加下划线，否则为关键字

soup.find('tagName',class\_/id/其他属性='song')#输出第一个符合条件

soup.find\_all('tagName'):以列表形式返回soup中所有符合条件的标签

#select

select('某种选择器（id/class/标签...选择器')),返回**列表**

Eg.<div class="tange">原文

soup.select('.tang')#类选择器

#层级选择器：

soup.select('.tang > ul >li >a')[0]#使用>作为层级封装符，ul,li,a为标签,第1个数据

soup.select('.tang > ul a')[0]#空格表示多个层级

#获取标签之间的文本数据（字符串）

soup.tagName.text/get\_text()#可获取某个标签下所有文本内容，包括非直系/string#仅获取该标签直系文本内容

#获取标签中的属性值

soup.tagName['属性名称']

3）爬取三国演义各章节标题以及原文

li\_list=soup.select('.book-mulu>ul>li')

fp=open('./sanguo.txt','w',encoding='utf-8')

for li in li\_list:

title=li.a.string

detail\_url='http://www.shicimingju.com'+li.a['href']

#对详情页发起请求，解析出章节内容

detail\_page\_text=requests.get(url=detail\_url,headers=headers).text

#解析出详情页中相关的章节内容

detail\_soup=BeautifulSoup(detail\_page\_text,'lxml')

div\_tag=detail\_soup.find('div',class\_='chapter\_content')

content=div\_tag.text

fp.write(title+':'+content+'\n')

3.5 xpath（包是lxml）

原理：实例化etree对象，且需要将被解析的页面源码数据加载到对应对象->

调用etree对象中的xpath方法结合xpath表达式实现标签的定位和内容捕获

1. 实例化etree对象

**from lxml import etree**

#将本地中的html文档中的数据加载到该对象中

tree=etree.parse(filePath)

#将互联网上获取的页面源码加载到该对象中（常用）

tree=etree.HTML('page\_text')

1. xpath表达式

(1)/第一个表示从根结点开始定位，其他位置表示一个层级

# 返回的是一个存储Element对象的列表

r=tree.xpath('/html/body/div')

1. //表示多个层级，或从任意位置定义

r=tree.xpath('//div')

1. 属性定位[@属性名称=属性值]

r=tree.xpath('//div[@class=“song”]')

1. 索引定位，从1开始

r=tree.xpath('//div[@class="song"]/p[3]')

1. 取文本,/text()原来返回仍为字符串列表，且获取标签中直系文本

//text(),可获取标签中非直系文本（所有）

r=tree.xpath('//div[@class="tang"]//li[5]/a**/text()**')**[0]**

r=tree.xpath('//div[@class="tang"]**//text()'**)

1. 取属性/@属性名

r=tree.xpath('//div[@class="tang"]/img/**@src**')

(7)|表示或

1. 例子

li\_list=tree.xpath('//ul[@class="house-list-wrap"/li]')

for li in li\_list:

# .代表li标签

li.xpath('./div[2]/h2/a/text()')

#通用处理中文乱码的解决方案

img\_name=img\_name.encode('iso-8859-1').decode('gbk')

或者将响应数据的编码整体设置为utf-8

response.encoding='utf-8'

r.encoding = r.apparent\_encoding 解决乱码

3.6 反爬机制：验证码，识别验证码图片中的数据，用于模拟登录操作

1）人工肉眼识别（不推荐）

2）第三方自动识别

超级鹰/云码：[www.yunma.com](http://www.yunma.com)

3.7 ajax：异步的 JavaScript 和 XML

原始的页面最初不会包含某些数据，原始页面加载完后，会再向服务器请求某个接口获取数据，然后数据才被处理从而呈现到网页上，这其实就是发送了一个 Ajax 请求

4.1 selenium 模拟浏览器运行库

4.11版本不需要下载一个浏览器的驱动程序（得与当前浏览器版本映射关系相匹配）

其中，webdriver\_manager为自动安装驱动装置的库

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from webdriver\_manager.chrome import **ChromeDriverManager**

browser = webdriver.Chrome(**ChromeDriverManager().install()**)

browser.get('http://www.baidu.com')

search = browser.find\_element\_by\_id('kw')

search.send\_keys('python')

search.send\_keys(Keys.ENTER)

# 关闭浏览器

browser.close()

1）初始化浏览器对象

driver = webdriver.Chrome()

driver.get("https://baidu.com/")

2）获取浏览器当前页面源码数据

page\_text=driver.page\_source

《解析企业名称（使用xpath）

tree =etree.HTML(page\_text)》

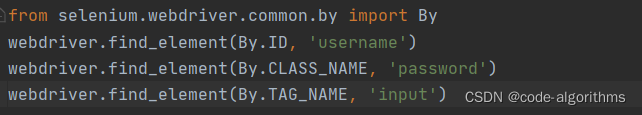
1. 属性定位

#通过webdriver对象的find\_element("属性名","属性值")

find\_element("class name",'btn-search')

#通过webdriver模块中的By，以指定方式定位元素

from selenium.webdriver.common.by import By



find\_element()和find\_elements()的区别

1. find\_element()的返回结果是一个WebElement对象，如果符合条件的有多个，默认返回找到的第一个，如果没有找到则抛出NoSuchElementException异常。

(2）find\_elements()的返回结果是一个包含所有符合条件的WebElement对象的列表，如果未找到，则返回一个空列表。

4) # 获取浏览器当前页面源码数据

page\_text=driver.page\_source

tree =etree.HTML(page\_text)

5)搜索淘宝Iphone

from time import sleep

# 标签定位

search\_input = driver.find\_element(By.ID,'q')

# 标签交互

search\_input.send\_keys('Iphone')

# 进行一组js程序：上滑一个屏幕高度

driver.execute\_script('window.scrollTo(0,document.body.scrollHeight)')

# 点击搜索按钮

btn = driver.find\_element(By.CSS\_SELECTOR, '.btn-search')

btn.click()

sleep(500)

driver.quit()# 关闭浏览器

前进，后退：forward(),back()

1. 实现拖动操作

#如果定位的标签是存在于iframe标签中，则必须通过以下操作再进行定位

driver.switch\_to.frame('id名')# 切换浏览器标签作用域

# 选中要拖动的对象

div = driver.find\_element(By.ID, 'draggable')

#导入动作链模块

from selenium.webdriver import ActionChains

# 实例化动作链对象

action = ActionChains(driver)

# 点击且长按指定标签

action.click\_and\_hold(div)

for i in range(5):

# perform()立即执行动作链操作

# move\_by\_offset(x,y) x水平方向像素值，y竖直方向像素值

action.move\_by\_offset(17, 0).perform()

sleep(0.3)

# 释放动作链

action.release()

1. 无头浏览器（无可视化）+规避检测 二选一

#实例化Options对象

#法一：option = webdriver.ChromeOptions()

option.add\_argument('headless')

#法二：实现无可视化界面

from selenium.webdriver.chrome.options import Options

options = Options()

options.add\_argument('--headless')#不提供可视化界面

options.add\_argument('--disable-gpu')#用于规避bug

driver= webdriver.Chrome(options=options)

#规避selenium被网站检测

from selenium.webdriver import Chrome

options = webdriver.ChromeOptions()

options.add\_experimental\_option("excludeSwitches", ["enable-automation"])

options.add\_experimental\_option('useAutomationExtension', False)

# 实例化一个浏览器对象，打开浏览器

driver = webdriver.Chrome(options=options)

driver.execute\_cdp\_cmd("Page.addScriptToEvaluateOnNewDocument", {

"source": """

Object.defineProperty(navigator, 'webdriver', {

get: () => undefined

})

""

})

5.1 数据库MySQL

1）mysql的卸载

进入mysql的安装目录，获取my.ini中的

datadir=D:/learn/IDE/MySQL/MySQL Server 8.0\Data

进入控制面板卸载mysql后，在D盘中搜索以上路径，删除文件夹。

1. 服务启动

法一：进入计算机-管理中的服务与应用程序->服务/cmd->services.msc->找到MySQL服务->右键停止或者启动，可以更改它的属性使得它变成手动或自动模式。

法二：cmd->以管理员身份运行->net stop mysql 停止服务/net start mysql 启动mysql

1. mysql -u root -p进入我的数据库，先输入密码再进入

访问别人的mysql：mysql -h127.0.0.1(别人的ip地址) -u root -p

mysql --host=ip --user=用户名 --password=链接目标密码

exit/quit:退出

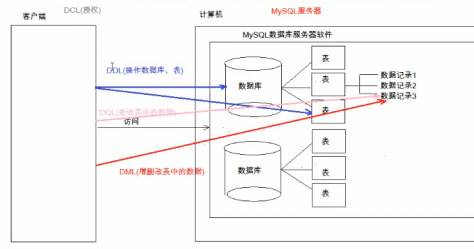
1. 配置文件my.ini 数据库：文件夹/表：文件/数据

5.2 SQL（结构化查询语言）操作关系型数据库

数据类型：https://www.runoob.com/sql/sql-datatypes.html

1） 单行/多行书写，分号结尾；不区分大小写，关键字使用大写；单行注释：-- （空格)注释内容或者#注释内容 多行注释：/\* \*/

2）分类：



2.1）DDL：

操作数据库：CRUD/

1）创建create：**create database 数据库名**;

#如果不存在db1则创建db1，并且将指定字符集为gbk

**create database if not exists db1 character set gbk**;

2）查询retrieve：查询所有数据库的名称：**show databases**；

show create database 数据库名称;# 查看某个数据库的字符集utf8/创建语句



3）修改update：**alter database**数据库名称**character set**字符生名称

4）删除delete: **DROP DATABASE 数据库名； drop database if exists db3;**

5）使用数据库：

**查询当前正在使用的数据库名称 select database();**

使用数据库 **use 数据库名；**

2.2）操作表

**use** game;#使用该数据库

1）创建表：**create table 表名称（**

#字段名称 数据类型

id INT,

name VARCHAR(100),

gold DECIMAL(10，2）

**)**

**设置默认值：https://www.cnblogs.com/mysqljs/p/14462211.html**

**数据类型：int整数 double(5,2)小数,共5位，保留到小数点后两位，**

**date 日期（yyyy-MM-dd),datetime(yyyy-MM-dd HH:mm:ss)**

**timestamp时间戳（年月日时分秒) ，不赋值或者为null则使用当前系统时间**

**varchar(20)可变字符串,最大二十个字符**

1. 查询表结构：**DESC 表名**；

查询某个数据库中所有表名称：**show tables**；

3）修改列表数据结构：**ALTER table** 表名**MODIFY COLUMN** **字段名**(如:name) **新数据类型**（如：VARCHAR(200))；

修改表名：ALTER TABLE 表名 **RENAME COLUNMN** **字段名**(如:name) **to 新字段名**；

修改表某字段默认值：ALTER table 表名**MODIFY 字段名 数据类型 DEFAULT 1；//默认为1**

添加新字段名：ALTER TABLE 表名 **ADD COLUMN 新字段名**(如:last) **新数据结构**(DATETIME)；

删除字段：ALTER TABLE 表名 **DROP COLUMN 字段名**；

删除整个表：**DROP TABLE 表名**；

2.3）操作数据

**INSERT INTO** 表名 （字段名，隔开）VLALUES （数值，隔开）（数值，隔开）#即可插入多条数据

#若字段顺序与表中数据结构定义相同则可省略，若只设置部分列数值，则使用默认值

**SELECT** \* from 表名；#从该表查询所有数据

**UPDATE** 表名 set level =1 where name = ‘李四’;#修改李四的等级为1级

UPDATE 表名 set level =1 ，gold=0；#修改全部

**DELETE FROM 表名** where gold=0;#删除表中全部gold为0的数据

2.4)数据的导入导出

使用命令行导出数据到具体文件：

mysqldump -u root -p 数据库名称 表名称（省略则到处所有数据库数据）> game.sql(文件名）

导入数据：mysql -u root -p 数据库名< 文件名

2.5）常用语句

**NOT>AND>OR**

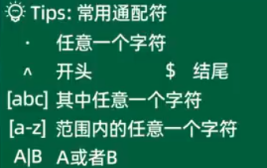
where lever **IN**(1,3,5)#在1，3，5之间

where lever **between** 1 **AND** 10;

NOT 取反 LIKE 模糊查询：%表示任意个字符，\_任意一个字符

where name **LIKE** ‘王%’#王姓玩家‘%王%’#名字中包含王 ’王\_’

正则表达式 where name REGEXP ‘^王.$’#查找姓王并且名字只有两个字的玩家

’王’#名字中包含王字

‘[王张]’或者’王|张’#名字中包含王字或张字

查找空结果：where email **is null** 或者where email <=> null(mysql专用）

查找非空结果：where email **is NOT** null

查找空字符串：where email = ’‘

排序：ORDER BY

SELECT \* FROM 表名 OEDER BY 字段名；#升序排列

SELECT \* FROM 表名 OEDER BY 字段名 DESC；#降序排列

Eg. SELECT \* FROM player OEDER BY 字level DESC,exp ASC;#按等级降序，经验升序

SELECT \* FROM 表名 OEDER BY 5 DESC；#按第5列降序排名

聚合函数

AVG（）集合中的平均值；COUNT（）项目数；MAX（）；MIN（）；SUM（）；

SELECT COUNT（\*）FROM 表名；SELECT AVG（level) FROM 表名；

分组：SELECT sex,count(\*) FROM 表名 **GROUP BY** sex；

SELECT level，count(level) from 表名 group by level；#每个等级的玩家数

HAVING: **筛选**分组后的数据

SELECT level，count(level) from 表名 group by level **HAVING** count(level)>4；#玩家数量大于4的等级

SELECT level，count(level) from 表名 group by level **HAVING** count(level)>4 order by count(level) DESC;

练习：统计每个姓氏玩家的数量，并将结果按照数量来降序排序,只显示大于等于5的姓氏

**substr**(字段名，开始位置,截取长度）

SELECT **SUBSTR（name,1,1),COUNT(SUBSTR（name,1,1))** from player

**group by** **SUBSTR（name,1,1)** HAVING (**SUBSTR（name,1,1)**)>=5

ORDER BY COUNT(**SUBSTR（name,1,1)**) DESC

LINIT 3#只返回前三名的姓氏

LINIT 3,3#第一个参数是偏移量，第二个参数是返回的数量。此为4~6

去重：DISTINCT

SELECT DISTINCT sex from player；# 返回去重后的结果

并集UNION （默认去除重复记录）ALL

INTERSECT 交集

EXCEPT 差集

2.6）子查询：

select \* from 表名 where level > (select AVG(level) from 表名）

select level，**ROUND** （select AVG(level) from 表名） as average(别名）from 表名

创建新表 create table new\_player select \* from player where lever<6

select **exists** (select \* from player where level >10)#返回值只有1或者0

2.7）表关联（相同字段）:笛卡尔积+条件过滤

内连接（两表中都有的数据）INNER JOIN

select \* from player **inner join** equip **on** player.id=equip.player\_id(关联玩家表与装备表）

左连接（左表中的所有数据+右表匹配数据，右表无用NULL填充）left JOIN

右连接 right join

select \* from player ,equip where player.id=equip.player\_id

2.8）索引

创建索引

create [unique(唯一索引)|fulltext(全文索引)|special(空间索引)] index index\_name on table\_name(字段名，隔开）

查看索引：show index from table\_name

删除索引：drop index 索引名 on table\_name

修改表结构时创建索引：alter table table\_name **add index** on 索引名（字段）

2.9）视图（数据动态）

创建视图:**create view** view\_name **as** select \* from player order by level desc limit 10;

查询视图：select \* from view\_name；

修改视图:alter view view\_name as select \* from player order by level limit 10;

删除视图：drop view view\_name;

5.3 pymysql

from pymysql import Connection

5.3.1# 创建数据库链接

con = None# 全局变量

try:

con = Connection(

host="localhost", # 主机名

port=3306, # 端口

user="root", # 用户名

password="Lucy0311@" # 密码

**database="db1" ,**# 指定操作的数据库

**autocommit=True**# 设置自动提交

)

print(type(con))

print(con.get\_host\_info())# 端口

print(con.get\_server\_info())#版本信息

except Exception as e:

print("异常：",e)

finally:

if con:

# 关闭连接

con.close()

5.3.2执行ddl语言：适应cursor（）创建一个游标对象，执行MySQL语句并返回结果

# 创建游标对象

**cursor = con.cursor()**

# 定义一个建表sql语句

# auto\_increment每次插入新记录自主创建主键字段，上一个加1

# engine为存储引擎,

# primary key规范存储主键和唯一性

sql = """

CREATE TABLE `t\_student`(

`id` int(11) NOT NULL **AUTO\_INCREMENT**,

`name` varchar(10) DEFAULT NULL,

`age` int(11) DEFAULT NULL,

PRIMARY KEY(`id`)

) **ENGINE=InnoDB** DEFAULT CHARSET=utf8

"""

# 选择要操作的数据库，

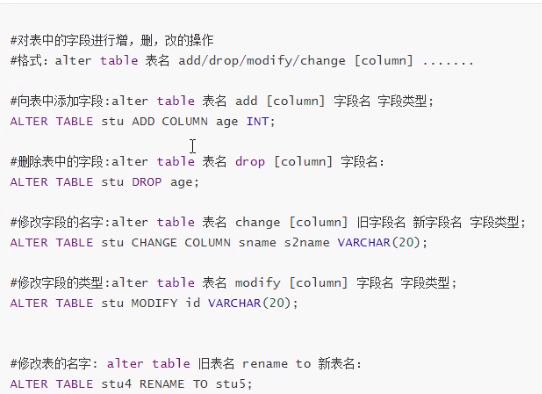
# con.select\_db("db1")可在数据库连接时确定该数据库

# 使用游标对象执行sql

**cursor.execute(sql)**

# 关闭游标对象

# cursor.close()可省略



sql="ALTER TABLE t\_student ADD COLUMN sex VARCHAR(2)"

5.3.3 查询操作

# 获取查询所有结果

cursor.execute("select \* from t\_student")

result = cursor.**fetchall()**

for row in result:

print(row)

5.3.4 插入操作

(需确认提交cursor.commit()或者在创建数据库连接时自动提交autocommit=True)

sql = "Insert into t\_student VALUES (2,'王五',19,'男')"

cursor.execute(sql)

print("主键id=",con.insert\_id())#获取插入的主键

5.3.5 修改操作 update :cursor.execute("**update** t\_student **set** age=19 where id=4")

5.3.6 删除操作 delete ： cursor.execute("delete from t\_student where id=4")

5.3.7 调用存储过程

-- 简单的存储过程在MySQL中：

create procedure test\_add(m INT,n INT,OUT result INT)

BEGIN

SET result=m+n;

END;

-- 测试

SET @s=0;

CALL test\_add(1,2,@s);

SELECT @s;

PyMysql实现：

# 使用游标对象调用存储过程

cursor.execute("CALL test\_add(1,2,@s)")

cursor.execute("select @s")

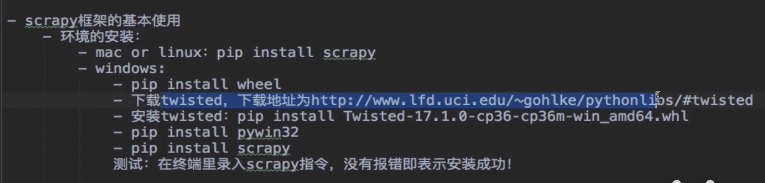
result = cursor.fetchone()

print(result[0])#返回元组的第一个值

6.1 scrapy框架

框架：集成很多功能具有很强通用性的一个项目模板





创建工程：**scrapy startproject** xxxPro

spiders子目录中创建爬虫文件 ：

cd到工程目录xxxPro中->**scrapy genspider** spiderName www.xxx.com(起始url)

执行工程 **scrapy crawl** spiderName --nolog(不要日志)(但错误信息就找不到，不建议使用)

class BaiduSpider(scrapy.Spider):

# 爬虫文件名称，即爬虫源文件的唯一标识

name = "baidu"

# 允许的域名，限定start\_url中哪些url可进行请求发送

# 不过，一般不用。如果是图片，可能在不同的服务器

# allowed\_domains = ["www.baidu.com"]

# 起始url列表：该列表存放的url会被scrapy自动进行请求的发送

start\_urls = ["https://www.baidu.com"]

# 用于数据解析：response对应的是请求成功后对应的响应对象

def parse(self, response):

pass

settings 配置文件:

# Obey robots.txt rules

ROBOTSTXT\_OBEY = **False**

# 显示指定类型的日志信息

**LOG\_LEVEL = 'ERROR'**

**# UA伪装**

USER\_AGENT = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 '

6.2 scrapy数据解析操作

位置：parse函数

#xpath返回的是selector对象列表，

需使用extract提取Selector对象中的data参数，返回字符串或字符串列表，可使用**join**将列表转成字符串：**content = ''.join(context)**

date = [response.xpath('//div[@id="cover"]//text()').](mailto:response.xpath('/div[@id=\"cover\"]/text()').extract())**[extract()](mailto:response.xpath('/div[@id=\"cover\"]/text()').extract())**

**extract\_first()**提取selector对象列表中第一个对象的data，返回字符串



6.3 scrapy的持久化存储

1)基于终端指令

只可以将parse方法的返回值存储到本地的文本文件

将多个数据封装到字典中，将解析数据所在字典添加到列表中，最后返回列表

注意：诸对应的文本文件的类型只可以为: 'json'，'jsonlines'，'jl', ‘csv','xml','marshal','pickle'

指令：scrapy crawl scrapyName **-o filePath**（如：./qiubai.csv)

优点：简洁高效 缺点：局限性强，数据只能存到指定后缀