爬虫请求的数据来自response,与elements中的不一定相同(因为浏览器会主动获取js，css等内容)

url的形式：scheme://host[:port#]/path/…/[?query-string][#anchor]

协议 资源路径 参数 锚点(删除不影响网页数据)

HTTP请求形式



requests模块

发送请求,获取响应

import requests

# 获取请求后的响应,response为对象

url="http://www.baidu.com"

response = requests.get(url)

# 查看响应解码后的数据

response.content.decode()

response.content.decode("GBK")

response.text

# 请求的常用属性

response.url

response.request.url

response.status\_code #状态码

response.headers

response.request.headers

# 带header的请求

headers = {"User-Agent":"..."}

requests.get(url,headers=headers)

# 带参数的请求

kw = {'wd':'长城'}

requests.get(url,params=kw)

format方法

"hello {}".format("world")

post请求

response = requests.post("http://www.baidu.com/",data=data,headers=headers)

程序中获取命令行的参数

some = sys.argv[1]

python3 test.py "hello"

🡪some = hello

代理服务器

使用代理

proxies = { “协议”:”协议+ip+端口”

"http": "http://12.34.56.79:9527",

"https": "https://12.34.56.79:9527",

}

requests.get("http://www.baidu.com", proxies = proxies)

处理cookie

1.创建session对象

session = requests.session()

response = session.get(url,headers)

2.在请求头中添加cookie

headers = {...,

"Cookie":" ..."}

requests.get(url,headers=headers)

3.添加cookies参数

cookie\_dict = {"cookie的name":"cookie的value"}

requests.get(url,headers=headers,cookies=cookie\_dict}

字典推导式

{i:i+1 for i in range(10)}

分析定位js

1通过Event Listener查找监听事件

2通过search all files来搜索(ctrl+shift+F)

获取响应的cookie

response = requests.get(url)

response.cookies # cookie对象

cookies = requests.utils.dict\_from\_cookiejar(response.cookies)

关闭ssl证书校验verify

response = requests.get(url,verify=False)

超时参数

response = requests.get(url,timeout=3) # 超过3s,会抛出异常

retrying模块,通过装饰器方式使用

from retrying import retry

@retry(stop\_max\_attempt\_number=3) # 最大重试3次，3次全部报错，才会报错

def get\_data(url):

...

数据的分类

1结构化 json,xml

直接转化

2非结构化 html

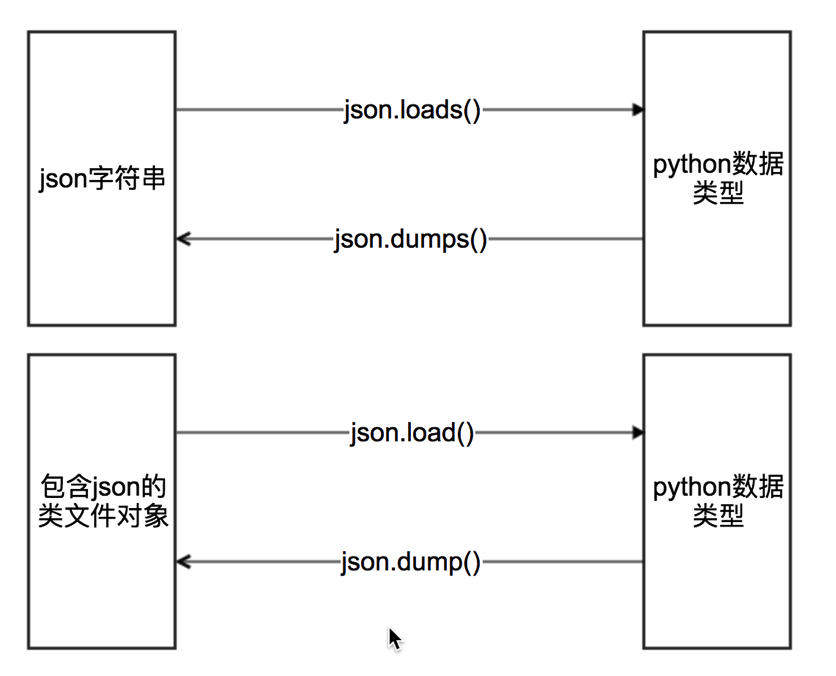
正则,xpath

美化打印

from pprint import pprint

pprint()

JSON



json.loads()

# ensure\_ascii显示中文 indent换行缩进字符数

json.dumps(dict, ensure\_ascii=False,indent=2)

#json.dump 实现把python类型写入类文件对象

with open("temp.txt","w") as f:

json.dump(mydict,f,ensure\_ascii=False,indent=2)

# json.load 实现类文件对象中的json字符串转化为python类型

with open("temp.txt","r") as f:

my\_dict = json.load(f)

正则



# re.S/re.DOTALL忽略\n,不匹配\n re.I会略大小写

# re.M把一行视为多行进行匹配输出

re.findall(‘.\*?’,str,re.S)

pattern.findall（找所有）

返回一个列表，没有就是空列表

pattern.sub（替换）

re.sub("\d","\_","chuan1zhi2") >> ["chuan\_zhi\_"]

re.compile（编译）

返回一个模型P，具有和re一样的方法，但是传递的参数不同

匹配模式需要传到compile中

p = re.compile("\d",re.S)

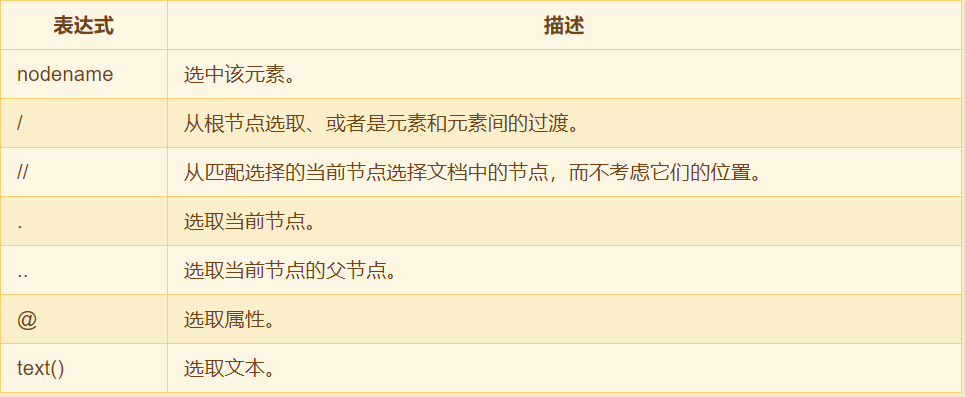
p.findall("chuan1zhi2")

r’’ 忽略转义符号带来的影响

windows中路径

Xpath

定位



查找特定节点



//a[contains(@class,’n’)] 选取class包含n的a节点

/\* //\* 匹配所有元素,子元素

@\* 匹配任何属性

‘|’或语法,同时获取两个xpath节点

lxml使用

from lxml import etree

html = etree.HTML(text)

text = etree.tostring(html).decode()

ret\_list = html.xpath("xpath字符串") # 返回列表

提取数据字典

先分组,再提取

li\_list = html.xpath("//li[@class='item-1']")

for li in li\_list:

item = {}

item["href"] = li.xpath("./a/@href")[0] if len(li.xpath("./a/@href"))>0 else None

item["title"] = li.xpath("./a/text()")[0] if len(li.xpath("./a/text()"))>0 else None

url地址解码方法

requests.utils.unquote()

多线程操作

多线程

t1 = threading.Thread(target=func,args=(,))

t1.setDaemon(True)

t1.start() #此时线程才会启动

队列使用

from queue import Queue

q = Queue(maxsize=100)

item = {}

q.put\_nowait(item) #不等待直接放，队列满的时候会报错

q.put(item) #放入数据，队列满的时候回等待

q.get\_nowait() #不等待直接取，队列空的时候会报错

q.get() #取出数据，队列为空的时候会等待

q.qsize() #获取队列中现存数据的个数

q.join() #队列中维持了一个计数，计数不为0时候让主线程阻塞等待，队列计数为0的时候才会继续往后执行

q.task\_done()

# put的时候计数+1，get不会-1，get需要和task\_done 一起使用才会-1

多进程

from multiprocessing import Process

t1 = Process(targe=func,args=(,))

t1.daemon = True #设置为守护进程

t1.start() #此时线程才会启动

多进程中使用普通的队列模块Queue会发生阻塞，对应的需要使用multiprocessing提供的JoinableQueue模块

JoinableQueue使用类似Queue

线程池

# 实例化对象

from multiprocessing.dummy import Pool

pool = Pool(process=5) #默认大小是cup的个数

def exetute\_requests\_item\_save(self):

url = self.queue.get()

html\_str = self.parse\_url(url)

content\_list = self.get\_content\_list(html\_str)

self.save\_content\_list(content\_list)

self.total\_response\_num +=1

def \_callback(self,temp):

if self.is\_running: pool.apply\_async(self.exetute\_requests\_item\_save,callback=self.\_callback)

pool.apply\_async(self.exetute\_requests\_item\_save,callback=self.\_callback)

while True: #防止主线程结束

time.sleep(0.0001) #避免cpu空转，浪费资源

if self.total\_response\_num>=self.total\_requests\_num:

self.is\_running= False

break

self.pool.close() #关闭线程池，防止新的线程开启

# self.pool.join() #等待所有的子线程结束

协程池

import gevent.monky

gevent.monky.path\_all()

from gevent.pool import Pool

# from multiprocessing.dummy import Pool # 替换Pool把线程池改为协程池

import requests

from lxml import etree

from queue import Queue

import time

class QiubaiSpider:

def \_\_init\_\_(self):

self.url\_temp = "https://www.qiushibaike.com/8hr/page/{}/"

self.headers = {"User-Agent": "..."}

self.queue = Queue()

self.pool = Pool(5)

self.is\_running = True

self.total\_requests\_num = 0

self.total\_response\_num = 0

def get\_url\_list(self): # 获取url列表

for i in range(1, 14):

self.queue.put(self.url\_temp.format(i))

self.total\_requests\_num += 1

def parse\_url(self, url): # 发送请求，获取响应

return requests.get(url, headers=self.headers).content.decode()

def get\_content\_list(self, html\_str): # 提取段子

html = etree.HTML(html\_str)

div\_list = html.xpath("//div[@id='content-left']/div")

content\_list = []

for div in div\_list:

content = {}

content["content"] = div.xpath(".//div[@class='content']/span/text()")

print(content)

content\_list.append(content)

return content\_list

def save\_content\_list(self, content\_list): # 保存数据

pass

def exetute\_requests\_item\_save(self):

url = self.queue.get()

html\_str = self.parse\_url(url)

content\_list = self.get\_content\_list(html\_str)

self.save\_content\_list(content\_list)

self.total\_response\_num += 1

def \_callback(self, temp):

if self.is\_running:

self.pool.apply\_async(self.exetute\_requests\_item\_save, callback=self.\_callback)

def run(self):

self.get\_url\_list()

for i in range(2): # 控制并发

self.pool.apply\_async(self.exetute\_requests\_item\_save, callback=self.\_callback)

while True: # 防止主线程结束

time.sleep(0.0001) # 避免cpu空转，浪费资源

if self.total\_response\_num >= self.total\_requests\_num:

self.is\_running = False

break

self.pool.close() # 关闭线程池，防止新的线程开启

# self.pool.join() #等待所有的子线程结束

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

qiubai = QiubaiSpider()

qiubai.run()

selenium

from selenium import webdriver

driver = webdriver.Chrome() #创建对象

browser.maximize\_window() #配置浏览器大小

browser.set\_window\_size(1920,1080) #手动配置浏览器大小

driver.get("http://www.baidu.com/") #请求网址

driver.save\_screenshot("./a.png") #截屏保存

# 定位操作

driver.find\_element\_by\_id(“kw”).send\_keys(“长城”) #输入

driver.find\_element\_by\_id("su").click() #点击

driver.page\_source #获取elements源码

driver.get\_cookies() #获取cookies

driver.current\_url #获取当前url

driver.close() #退出当前页面

driver.quit() #退出浏览器

元素定位语法

find\_element\_by\_id (返回一个元素)

**find\_elements\_by\_xpath** （返回一个包含元素的列表）

find\_elements\_by\_link\_text （根据连接文本获取元素列表）

find\_elements\_by\_partial\_link\_text （根据连接包含的文本获取元素列表）

find\_elements\_by\_tag\_name (根据标签名获取元素列表)

find\_elements\_by\_class\_name （根据类名获取元素列表）

find\_elements\_by\_xpath后获取相应数据

获取文本：element.**text**

获取属性值：element.**get\_attribute**("href")

处理cookie

# 把cookie转化为字典

{cookie[‘name’]: cookie[‘value’] for cookie in driver.get\_cookies()}

#删除一条cookie

driver.delete\_cookie("CookieName")

# 删除所有的cookie

driver.delete\_all\_cookies()

使用selenium切换frame

driver.switch\_to.frame(id/name/num)

无头模式(不打开浏览器)

options = webdriver.ChromeOptions()

options.add\_argument(“--headless”) # 开启无头模式

options.add\_argument(“--disable-gpu”) # 关闭gpu

options.add\_argument(“--user-agent=Mo...”) # 添加请求头

options.add\_argument(“--proxy-server=http://12.34.56.78:8001”) # 设置代理

# options.set\_headless() 类似的无头模式

driver = webdriver.Chrome(chrome\_options=options) # 使用配置

driver.get(“http://www.baidu.com”)

driver.save\_screenshot(‘a.png’)

driver.quit()

新的查找方法

from selenium import webdriver  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC

# 设置等待条件

wait = WebDriverWait(driver, 20)

name=wait.until(EC.presence\_of\_element\_located((By.ID, 'loginName')))  
name.send\_keys(...)

wait.until(EC.title\_is('我的首页'))

常用的判断条件：  
title\_is 标题是某内容  
title\_contains 标题包含某内容  
presence\_of\_element\_located 元素加载出，传入定位元组，如(By.ID, 'p')  
visibility\_of\_element\_located 元素可见，传入定位元组  
visibility\_of 可见，传入元素对象  
presence\_of\_all\_elements\_located 所有元素加载出  
text\_to\_be\_present\_in\_element 某个元素文本包含某文字  
text\_to\_be\_present\_in\_element\_value 某个元素值包含某文字  
frame\_to\_be\_available\_and\_switch\_to\_it frame加载并切换  
invisibility\_of\_element\_located 元素不可见  
element\_to\_be\_clickable 元素可点击  
staleness\_of 判断一个元素是否仍在DOM，可判断页面是否已经刷新  
element\_to\_be\_selected 元素可选择，传元素对象  
element\_located\_to\_be\_selected 元素可选择，传入定位元组  
element\_selection\_state\_to\_be 传入元素对象以及状态，相等返回True，否则返回False  
element\_located\_selection\_state\_to\_be 传入定位元组以及状态，相等返回True，否则返回False  
alert\_is\_present 是否出现Alert

mongodb

启动：sudo service mongod start

停止：sudo service mongod stop

默认端⼝：27017

服务端mongodb无法启动的解决方法

sudo mongod --config /etc/mongod.conf &

查看当前的数据库：db

查看所有的数据库：show dbs /show databases

切换数据库：use db\_name

删除当前的数据库：db.dropDatabase()

手动创建集合：

db.createCollection(name,options)

db.createCollection("stu")

db.createCollection("sub", { capped : true, size : 10 } )

参数capped： 默认值为false表示不设置上限,值为true表示设置上限

参数size：当capped值为true时，需要指定此参数，表示上限⼤⼩,当⽂档达到上限时，会将之前的数据覆盖，单位为字节

查看集合：show collections

删除集合：db.集合名称.drop()

查看集合内数据：db.集合名称.find()

数据类型:

Object ID： ⽂档ID

Object： ⽤于嵌⼊式的⽂档， 即⼀个值为⼀个⽂档

...

Date： 存储当前⽇期或时间的UNIX时间格式(参数的格式为YYYY-MM-DD new Date('2017-12-20'))

mongodb的增删改查

插入

db.集合名称.insert() \_id存在报错

eg:db.stu.insert({\_id:"20170101",name:'gj',gender:1})

保存

db.集合名称.save(document) \_id存在为更新

查询

db.集合名称.find()

更新

db.集合名称.update(<query> ,<update>,{multi: <boolean>})

参数query:查询条件

参数update:更新操作符

参数multi:可选，默认是false只更新1条数据，true表示更新全部数据,需要{$set

db.stu.update({name:'hr'},{name:'mnc'}) 更新一条(全覆盖)

db.stu.update({name:'hr'},{$set:{name:'hys'}}) 更新一条

db.stu.update({},{$set:{gender:0}},{multi:true}) 更新全部

删除

db.集合名称.remove(<query>,{justOne:<boolean>})

justOne:true只删1条,默认false

高级操作

⽅法find()：查询

db.集合名称.find({条件⽂档})

⽅法findOne()：查询，只返回第⼀个

db.集合名称.findOne({条件⽂档})

⽅法pretty()： 将结果格式化

db.集合名称.find({条件⽂档}).pretty()

比较运算符

等于： 默认是等于判断， 没有运算符

⼩于：$lt （less than）

⼩于等于：$lte （less than equal）

⼤于：$gt （greater than）

⼤于等于：$gte

不等于：$ne

eg: db.stu.find({age:{$gte:18}})

逻辑运算符

and：在json中写多个条件即可,用 ”,” 隔开

or:使⽤$or， 值为数组， 数组中每个元素为json

eg: db.stu.find({$or:[{age:{$gte:18}},{gender:true}],name:'gj'})

范围内$in:

eg:db.stu.find({age:{$in:[18,28,38]}})

正则

使⽤//或$regex编写正则表达式

eg:

查询sku以abc开头的数据

db.products.find({sku:/^abc/})

查询sku以789结尾的数据

db.products.find({sku:{$regex:'789$'}})

skip与limit

⽅法limit()： ⽤于读取指定数量的⽂档

db.集合名称.find().limit(NUMBER)

查询2条学⽣信息

db.stu.find().limit(2)

⽅法skip()： ⽤于跳过指定数量的⽂档

db.集合名称.find().skip(NUMBER)

db.stu.find().skip(2)

同时使用

db.stu.find().limit(4).skip(5)

或

db.stu.find().skip(5).limit(4)

自定义查询$where

查询年龄⼤于30的学⽣

db.stu.find({

$where:function() {

return this.age>30;}

})

投影

在查询到的返回结果中， 只选择必要的字段, 值为1表示显示， 值为0不显

命令：db.集合名称.find({},{字段名称:1,...})

eg: db.stu.find({},{\_id:0,name:1,gender:1})

排序

⽅法sort()， ⽤于对 集进⾏排序

命令：db.集合名称.find().sort({字段:1,...})

参数1为升序排列 参数-1为降序排列

eg:根据性别降序， 再根据年龄升序

db.stu.find().sort({gender:-1,age:1})

统计

命令1：db.集合名称.find({条件}).count()

命令2：db.集合名称.count({条件})

eg: db.stu.find({gender:true}).count()

db.stu.count({age:{$gt:20},gender:true})

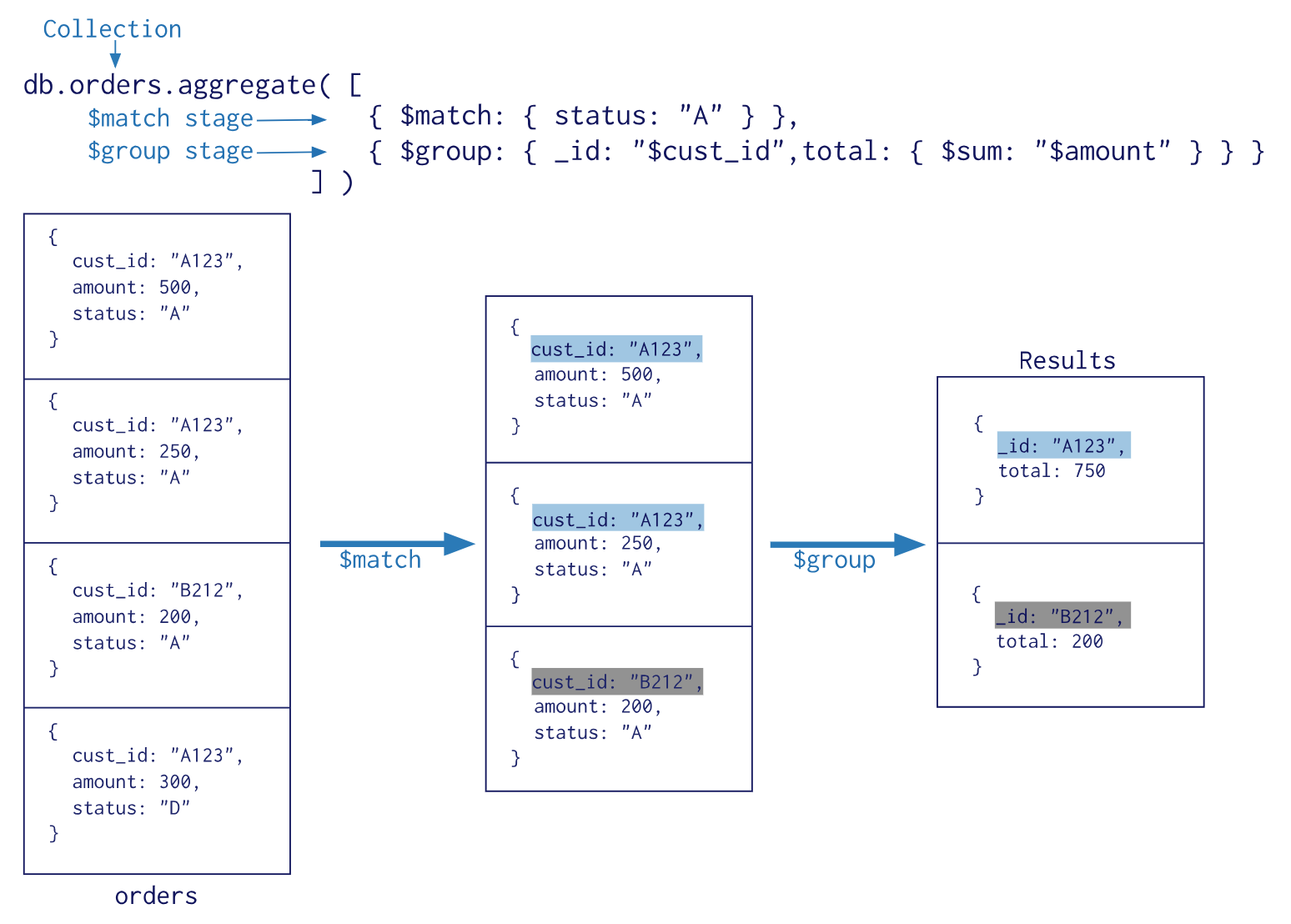
去重

命令：db.集合名称.distinct('去重字段',{条件})

eg: db.stu.distinct('hometown',{age:{$gt:18}})

聚合操作

语法：db.集合名称.aggregate({管道:{表达式}})



常用管道

$group： 将集合中的⽂档分组， 可⽤于统计结果

$match： 过滤数据， 只输出符合条件的⽂档

$project： 修改输⼊⽂档的结构， 如重命名、 增加、 删除字段、 创建计算结果

$sort： 将输⼊⽂档排序后输出

$limit： 限制聚合管道返回的⽂档数

$skip： 跳过指定数量的⽂档， 并返回余下的⽂档

常用表达式

$sum： 计算总和， $sum:1 表示以⼀倍计数

$avg： 计算平均值

$min： 获取最⼩值

$max： 获取最⼤值

$push： 在结果⽂档中插⼊值到⼀个数组中

1. 分组

db.stu.aggregate(

{$group:

{

\_id:"$gender", #分组依据

counter:{$sum:1}, #$sum:1 表示以⼀倍计数

total\_age:{$sum:”$age”}, #求和

avg\_age:{$avg:”$age”}, #求平均

}

}

)

多字段分组,\_id值为json

db.stu.aggregate({$group:{\_id:{gender:"$gender",hometown:"$hometown "}, counter:{$sum:1}}})

2.不分组,统计文档

db.stu.aggregate(

{$group:

{

\_id:null,

counter:{$sum:1}

}

}

)

3.数据透视

eg: 统计不同性别的学生

db.stu.aggregate(

{$group:

{

\_id:null,

name:{$push:"$name"}

}

}

)

eg: 使用$$ROOT可以将整个文档放入数组中

db.stu.aggregate(

{$group:

{

\_id:null,

name:{$push:"$$ROOT"}

}

}

)

$match用于进行数据的过滤

eg: 查询年龄大于20的男女学生的人数

db.stu.aggregate(

{$match:{age:{$gt:20}}

{$group:{\_id:"$gender",counter:{$sum:1}}}

)

$project用于修改文档的输入输出结构

0不显示 , 1显示, 新增字段

eg: 查询男女生人生，输出人数

db.stu.aggregate(

{$group:{\_id:"$gender",counter:{$sum:1}}}

{$project:{\_id:0,counter:1}}

)