## 1、os.fork()

从fork()函数原型来看，它也属于一个内建函数。子进程永远返回0，而父进程返回子进程的ID。这样做的理由是，一个父进程可以fork()出很多子进程，所以，父进程要记下每个子进程的ID，而子进程只需要调用getppid()就可以拿到父进程的ID。

只能在POSIX系统上可用。在windows版的python中，os模块没有定义os.fork函数。相反，windows程序员用多线程编程技术来完成并发任务

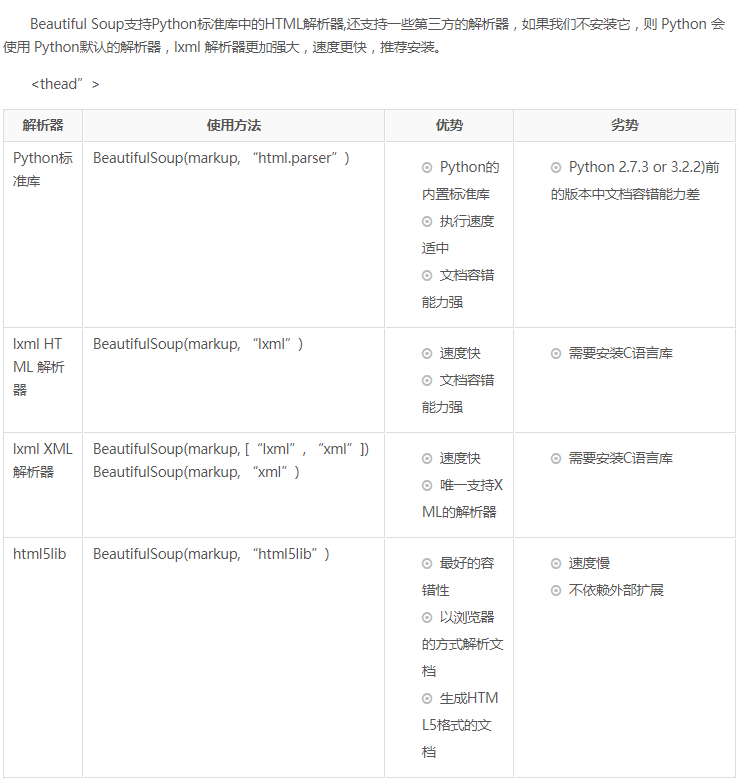
## 2、POSIX系统

POSIX表示可移植操作系统接口（Portable Operating System Interface of UNIX，缩写为 POSIX ），POSIX标准定义了操作系统应该为应用程序提供的接口标准，是IEEE为要在各种UNIX操作系统上运行的软件而定义的一系列API标准的总称，其正式称呼为IEEE 1003，而国际标准名称为ISO/IEC 9945。

## 3、鲁棒性

鲁棒是Robust的音译，也就是健壮和强壮的意思。它是在异常和危险情况下系统生存的关键。比如说，计算机[软件](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6)在输入错误、磁盘故障、网络过载或有意攻击情况下，能否不死机、不崩溃，就是该软件的鲁棒性。所谓“鲁棒性”，是指控制系统在一定（结构，大小）的参数[摄动](https://baike.baidu.com/item/%E6%91%84%E5%8A%A8)下，维持其它某些性能的特性。根据对性能的不同定义，可分为稳定鲁棒性和性能鲁棒性。以[闭环系统](https://baike.baidu.com/item/%E9%97%AD%E7%8E%AF%E7%B3%BB%E7%BB%9F)的鲁棒性作为目标设计得到的固定控制器称为[鲁棒控制](https://baike.baidu.com/item/%E9%B2%81%E6%A3%92%E6%8E%A7%E5%88%B6)器。

## 4、BeautifulSoup

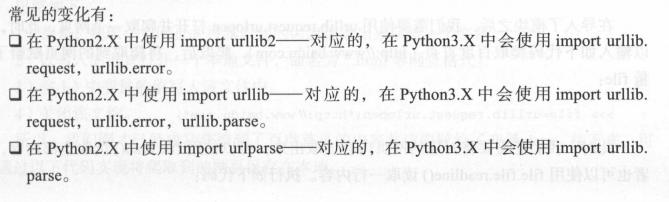


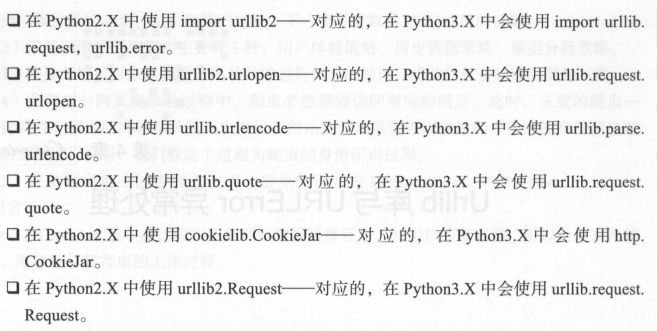
## 5、pip与apt-get的区别

pip用来安装来自PyPI的python所有的依赖包，并且可以选择安装任何在PyPI上已上传的先前版本的依赖包；   
apt-get可以用来安装软件、更新源、也可以用来更新自Ubuntu的典型依赖包，典型安装即意味着它只是安装（最新发布的，或最近一个的）单一版本，并且我们不能决定我们要安装的依赖包的版本或选择它之前的版本。

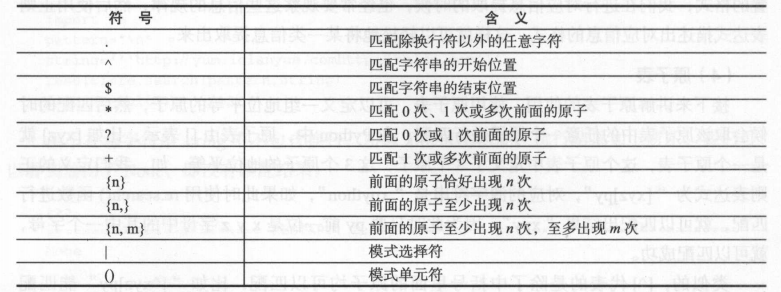


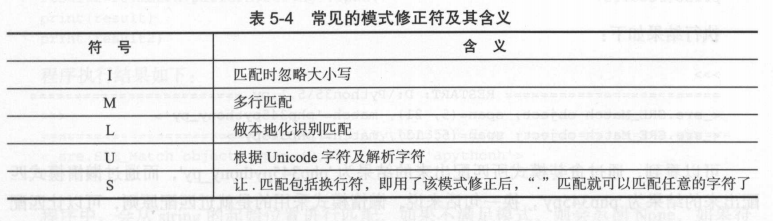
## 6、2.X版本与3.X版本urllib变化





## 7、Regex元字符

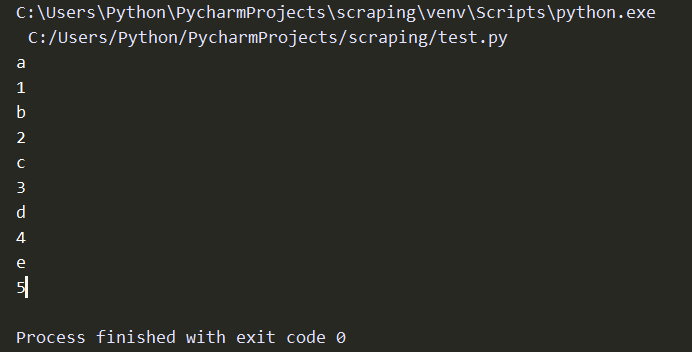




## 8、并列for依次输出

num = ['1','2','3','4','5']  
let = ['a','b','c','d','e']  
a = 1  
for x in num:  
 name = 'x' + str(a)  
 exec(name+'=x')  
 a += 1  
b = 1  
for y in let:  
 name = 'x' + str(b)   
 print(y)  
 exec("print("+name+")")   
 b += 1

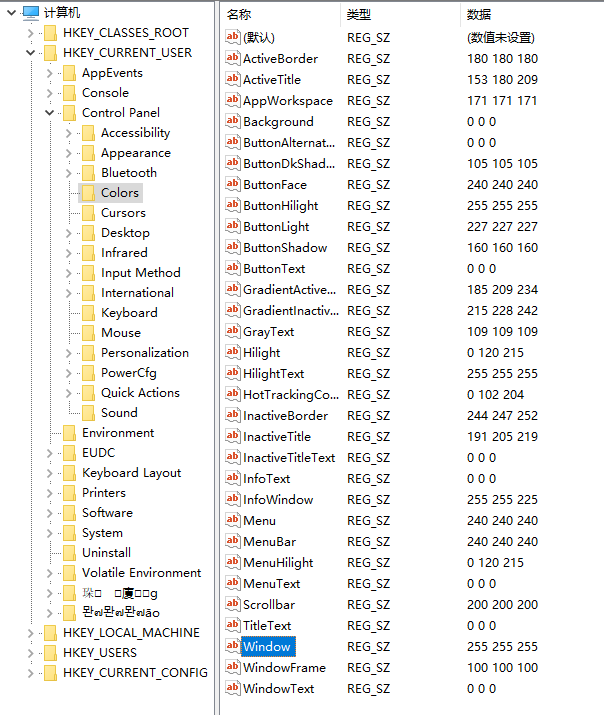
输出



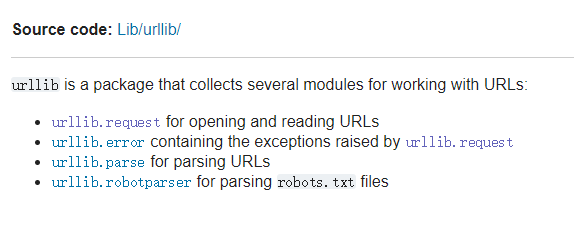
## 9、网站容量查询参考



## 10、Windows调节记事本背景颜色

Win+R->regedit-> 

## 11、urllib的结构：



## 12、进程池pool.map(), pool.join()

1.Join(),主进程阻塞等待子进程的退出，join方法必须在close或terminate之后使用。

## 13、map()函数语法：

### 1.语法

map(function, iterable,…)

### 2.示例

def square(x):#计算平方数

return x \*\* 2

a = map(square,[1,2,3,4,5])

list(a)

>>>[1, 4, 9, 16, 25]

b = map(lambda x: x\*\*2, [1,2,3,4,5])#使用lambda匿名函数

list(b)

>>>[1, 4, 9, 16, 25]

#提供了两个列表，对相同位置的列表数据进行相加

c = map(lambda x, y: x + y, [1,2,3,4,5], [2,4,6,8,10])

list(c)

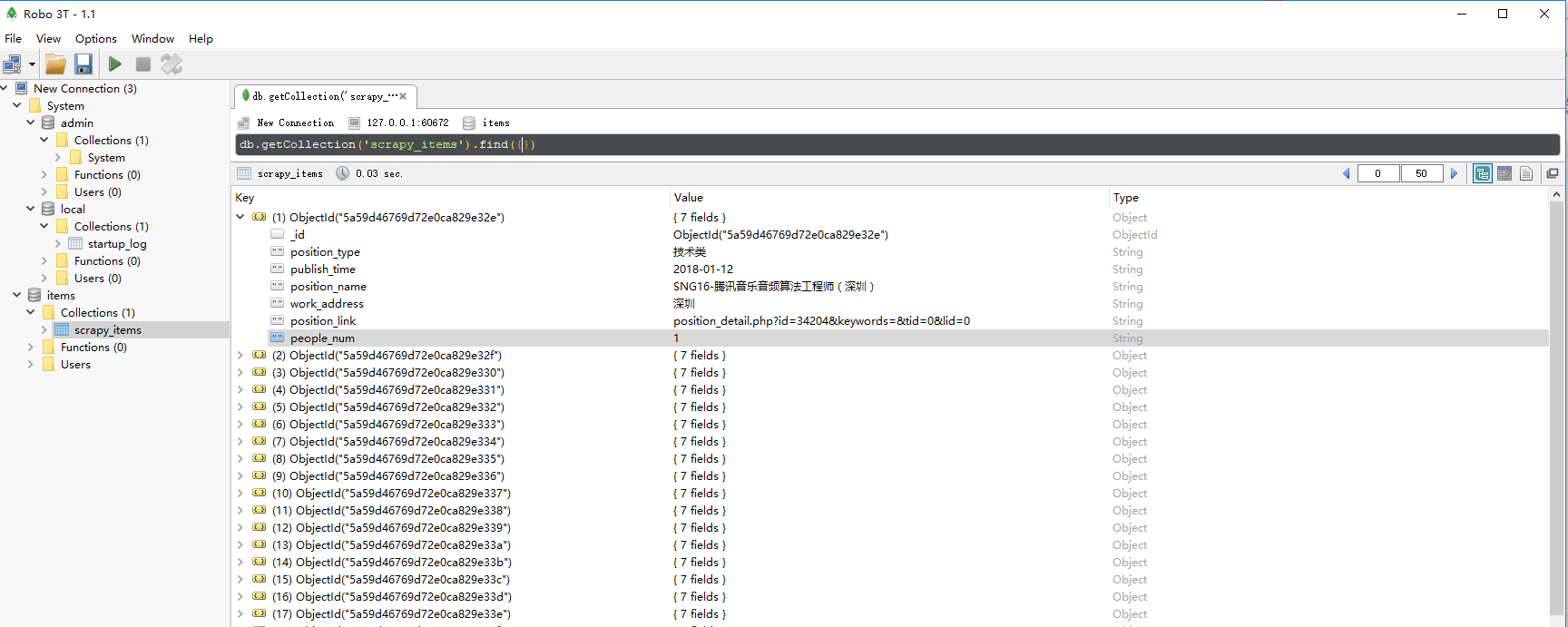
>>>[3, 6, 9, 12, 15]

## 14将mongodb设置为开机启动

1. 新建文件夹db作为数据库数据存储了路径  
 cd ~  
 mkdir db  
 2. 设置mongod为开机启动  
 sudo vim /etc/rc.local   
  
 添加如下一行到文件/etc.rc.local 的"exit 0"行之前：  
 sudo /opt/mongodb-linux-x86\_64-ubuntu1604-3.4.10/bin/mongod --dbpath /home/tarena/db  
 3. 重启  
 reboot  
  
   
 验证：  
tarena@tedu:~$ mongo」  
—————————

## 15、利用Robo 3T，对mongo进行数据可视化

<https://robomongo.org/download>



## 16、值类型 引用类型

值类型：int str tuple(不可改变)

引用类型：list, dict, set(可改变)

## 17、id()

>>>b = ‘hello’

>>>id(b)

>>>55336696 #这是b的内存地址

## 18、chr() ord()

>>>chr(65)

>>>’A’

>>>ord(‘A’)

>>>65

## 19、isinstance()

>>>a = 1

>>>isinstance(a, int)

>>>True

>>>isinstance(a, str)

>>>False

>>>isinstance(a, (int, str, float)) #判断a是否为int，str，float三种元素中的任意一种

>>>True

(推荐使用isinstance做类型判断)

## 21、not 和 and 和 or的优先级

>>> 1 or 2 and 3

>>>1 #and的优先级高于or

not > and > or



## 22、改变系统默认循环次数

>>> import sys

>>>sys.setrecursionlimit(10000) #设定循环次数为10000次，不过计算机本身有最高循环次数，在3000次左右

## 23、序列解包

>>>a, b, c = [1,2,3]

>>>print(a)

>>>1

>>>print(a,b,c)

>>>1 2 3 #同样适用于元祖

## 24、参数赋值

def add(x, y):

result = x + y

return result

c = add(2，3) #或者 c = add(y = 3, x = 2)

## 20、对象的三个特征

id, value, type

## 25、默认参数

def student\_file(name, gender=’男’, age=’18’, collage=’人民大学’):

print(‘我叫’ + name)

….

#如果默认值需要需改，那么直接在调用函数中传入实际需要参数即可。

## 26、可变参数

def demo(\*param):

print(param)

print(type(param))

demo(1,2,3,4,5,6)

>>>(1,2,3,4,5,6)

>>><class ‘tuple’>

a = (1,2,3,4,5,6)

demo(a)

>>>((1,2,3,4,5,6),) #二维元祖

a = (1,2,3,4,5,6)

demo(\*a)

>>>(1,2,3,4,5,6)

参数放置顺序

demo(必须参数，可变参数，默认参数=…)

def demo(param1, \*param, param=2):

print(param1)

print(param)

print(param2)

demo(‘a’, 1,2,3, param2=’param’)

>>>a

>>>(1,2,3)

>>>param

def squsum(\*param):

sum = 0

for i in param:

sum += i\*i

print(sum)

squsum(1,2,3) #or squsum(\*[1,2,3])

>>>14

## 27、关键字可变参数

def city\_temp(\*\*param):

print(param)

print(type(param))

city\_temp(bj=’22’, sh=’25’)

>>>{‘bj’:’22’, ‘sh’:’25’}

>>><class ‘dict’>

(用for in 循环遍历字典，打印出的是字典的key,

用for key, value in dict.items(),打印出的是字典的键和值)

def city\_temp(\*\*param):

print(param)

for key, value in param.items():

print(key, ':', value)

city\_temp(bj='22', sh='25')如果想将字典传入关键字参数

a = {'x':2, 'y':3, 'z':4}

city\_temp(\*\*a)

## 28、全局变量

def demo():

global c

c = 2

demo() #此处需要调用一下demo()

print(c)

>>>2

## 29、三元表达式

条件为真时返回的结果 if 条件判断 else 条件为假时返回的结果

>>> x = 2

>>> y = 3

>>> r = x if x > y else y

>>> print(r)

3

## 30、map函数

>>> list\_x = [1,2,3,4,5,6]

>>> def square(x):

...     return x \* x

...

>>> r = map(square, list\_x)

>>> print(list(r))

[1, 4, 9, 16, 25, 36]

## 31、lambda函数

>>> list\_x = [1,2,3,4,5]

>>> r = map(lambda x:x\*x, list\_x)

>>> print(list(r))

[1, 4, 9, 16, 25]

>>> list\_x = [1,2,3,4,5]

>>> list\_y = [2,4,6,8,10]

>>> r = map(lambda x, y:x\*x + y, list\_x, list\_y)

>>> print(list(r))

[3, 8, 15, 24, 35]

当两个参数数量不同时，取参数较少的为准进行计算

>>> list\_x = [1,2,3,4]

>>> list\_y = [2,4,6,8,10]

>>> r = map(lambda x, y:x\*x + y, list\_x, list\_y)

>>> print(list(r))

[3, 8, 15, 24]

## 32、reduce函数

#下面例子不是连续相加，是连续调用lambda表达式

>>> from functools import reduce

>>> list\_x = [1,2,3,4,5]

>>> r = reduce(lambda x,y:x+y, list\_x)

>>> print(r)

15

>>> r = reduce(lambda x,y:x+y, list\_x, 10)#这里的10是作为初始值，作为调用lambda表达式的第一个x

25

## 33、filter函数

>>> list\_x = [1,0,1,1,0]

>>> r = filter(lambda x:True if x==1 else False, list\_x)

>>> print(list(r))

[1, 1, 1]

## 34、装饰器

import time

def decorator(func):

def wrapper(\*args, \*\*kw): #多个参数

print(time.time())

func(\*args, \*\*kw)

return wrapper

@decorator

def f1(func\_name):

print('this is a function' + func\_name)

@decorator

def f2(func\_name1, func\_name2):

print('this is a function' + func\_name1)

print('this is a function' + func\_name2)

@decorator

def f3(func\_name1, func\_name2, \*\*kw):

print('this is a function' + func\_name1)

print('this is a function' + func\_name2)

print(kw)

f1('test\_func')

f2('test\_func1', 'test\_func2')

f3('test\_func1', 'test\_func2', a=1, b=2, c='123')

## 35、Python中用字典映射代替switch函数

day = 0

def get\_sunday():

return 'Sunday'

def get\_monday():

return 'Monday'

def get\_tuesday():

return 'Tuesday'

def get\_default():

return 'Unknown'

switcher = {

0 : get\_sunday,

1 : get\_monday,

2 : get\_tuesday

}

day\_name = switcher.get(day, get\_default)()

print(day\_name)

## 36、列表推导式（集合、字典、元祖都可被推导）

a = [1,2,3,4,5,6]

b = [i\*i for i in a]

print(b)

a = [1,2,3,4,5,6]

b = [i\*i for i in a if i > 3]

print(b)

[16, 25, 36]

注：b中列表推导式的形式不仅仅限于列表，得到的结果取决于b的形式，下例中为集合，上例中为列表

a = {1,2,3,4,5,6}

b = {i\*i for i in a if i > 3}

print(b)

{16, 25, 36}

## 37、字典推导式

列表

students = {

'Anna' : 18,

'Bob' : 19,

'Cindy' : 22

}

b = [key for key,value in students.items()]

print(b)

字典

students = {

'Anna' : 18,

'Bob' : 19,

'Cindy' : 22

}

b = {value:key for key,value in students.items()}

print(b)

>>> {18: 'Anna', 19: 'Bob', 22: 'Cindy'}

元祖

students = {

'Anna' : 18,

'Bob' : 19,

'Cindy' : 22

}

b = (key for key,value in students.items())

for x in b:

print(x)

## 38、对象存在不一定是True

class Test():

def \_\_len\_\_(self):

return 0

test = Test()

if test:

print('S')

else:

print(‘F’)

## 39、Terminal中的networksetup –listallhardwareports

man networksetup

lsy:~ futeen$ networksetup -listallhardwareports

Hardware Port: Wi-Fi

Device: en0

Ethernet Address: 8c:85:90:41:d8:ab

Hardware Port: Bluetooth PAN

Device: en6

Ethernet Address: 8c:85:90:43:4c:72

Hardware Port: Thunderbolt 1

Device: en3

Ethernet Address: 6a:00:a9:80:d9:01

Hardware Port: Thunderbolt 2

Device: en1

Ethernet Address: 6a:00:a9:80:d9:00

Hardware Port: Thunderbolt 3

Device: en4

Ethernet Address: 6a:00:a9:80:d9:05

Hardware Port: Thunderbolt 4

Device: en2

Ethernet Address: 6a:00:a9:80:d9:04

Hardware Port: Thunderbolt Bridge

Device: bridge0

Ethernet Address: 6a:00:a9:80:d9:00

VLAN Configurations

===================

关闭WiFi

networksetup

-setairportpower

en0

off

## 40、类与对象

class Student():

name = ''

age = 0

def do\_homework(self):

print('homework')

student = Student()

student.do\_homework()

## 41、类的一些总结

class Student():

sum = 0

def \_\_init\_\_(self， name, age): #构造函数，每次会自动被调用，构造类的初始特征。

self.name = name

self.age = age

print(self.name) #这个self.name读取的是变量的name

print(name) #这个name读取的是形参的变量

print(Student.sum) #或者print(self.\_\_class\_\_.sum)

@classmethod

def plus\_sum(cls):

cls.sum += 1

print(cls.sum) #类变量

## 42、新建virtualenv指明Python版本

cmd: mkvirtualenv –python=<路径> <新建环境名称>

例:mkvirtualenv –python=C:\Users\Adminstrator\AppData\Local\Programs\Python\Python35-32\Python.exe article\_spider

## 43、Mac terminal光标快速移动办法

* **将光标移动到行首**：control + a
* **将光标移动到行尾**：control + e
* **清除屏幕**：control + l
* 搜索以前使用命令：control + r
* **清除当前行**：control + u
* 清除至当前行尾：control + k
* **单词为单位移动**：option + 方向键

## 44、extract和extract\_first()区别

extract\_first()返回字符串 extract()返回数组

## 45、md5处理URL作用

使URL便成唯一的且长度固定的一个值

## 46、mac进入chrome driver文件夹

cd ‘chrome driver’ #类似连续单词，用引号将其确定为一组

## 47、装饰器

import time

def f1():

print('hello')

def f2():

print('hello')

def print\_current\_time(func):

print(time.time())

func()

print\_current\_time(f1)

print\_current\_time(f2)

def decorator(func):

def wrapper(\*args, \*\*kw):

print(time.time())

func(\*args, \*\*kw)

return wrapper

@decorator

def f1(func\_name):

print('hello' + func\_name)

@decorator

def f2(func\_name1, func\_name2):

print('hello' + func\_name1)

print('hello' + func\_name2)

@decorator

def f3(func\_name1, func\_name2, \*\*kw):

print('hello' + func\_name1)

print('hello' + func\_name2)

print(kw)

f1('test\_func')

f2('test\_func1', 'test\_func2')

f3('test\_func1', 'test\_func2', a=1, b=2, c='123')

## 48、python 大小写转换

capitalize() 首字母大写，其余全部小写

upper() 全转换成大写

lower() 全转换成小写

title()  标题首字大写，如"i love python".title()  "I love python"

## 49、网站状态码

根据响应结果的类型，大致分为以下几类：

2.1 1XX（信息类）

该类型状态码表示接收到请求并且继续处理。

* 100，客户端必须继续发出请求。
* 101，客户端要求服务器根据请求转换HTTP协议版本。

2.2 2XX（响应成功）

该类型状态码表示动作被成功接收、理解和接受。

* 200，表明该请求被成功地完成，所请求的资源发送到客户端。
* 201，提示知道新文件的URL。
* 202，接受并处理，但处理未完成。
* 203，返回信息不确定或不完整。
* 204，收到请求，但返回信息为空。
* 205，服务器完成了请求，用户必须复位当前已经浏览过的文件。
* 206，服务器已经完成了部分用户的GET请求。

2.3 3XX（重定向类）

该类型状态码表示为了完成指定的动作，必须接受进一步处理。

* 300，请求的资源可在多处获得。
* 301，本网页被永久性转移到另一个URL。
* 302，请求的网页被重定向到新的地址。
* 303，建议用户访问其他URL或访问方式。
* 304，自从上次请求后，请求的网页未修改过。
* 305，请求的资源必须从服务器指定的地址获得。
* 306，前一版本HTTP中使用的代码，现已不再使用。
* 307，声明请求的资源临时性删除。

2.4 4XX（客户端错误类）

该类型状态码表示请求包含错误语法或不能正确执行。

* 400，客户端请求有语法错误。
* 401，请求未经授权。
* 402，保留有效ChargeTo头响应。
* 403，禁止访问，服务器收到请求，但拒绝提供服务。
* 404，可连接服务器，但服务器无法取得所请求的网页，请求资源不存在。
* 405，用户在Request-Line字段定义的方法不被允许。
* 406，根据用户发送的Accept，请求资源不可访问。
* 407，类似401，用户必须首先在代理服务器上取得授权。
* 408，客户端没有在用户指定的时间内完成请求。
* 409，对当前资源状态，请求不能完成。
* 410，服务器上不再有此资源。
* 411，服务器拒绝用户定义的Content-Length属性请求。
* 412，一个或多个请求头字段在当前请求中错误。
* 413，请求的资源大于服务器允许的大小。
* 414，请求的资源URL长于服务器允许的长度。
* 415，请求资源不支持请求项目格式。
* 416，请求中包含Range请求头字段，在当前请求资源范围内没有range指示值。
* 417，服务器不满足请求Expect头字段指定的期望值。

2.5 5XX（服务器错误类）

该类型状态码表示服务器或网关错误。

* 500，服务器错误。
* 501，服务器不支持请求的功能。
* 502，网关错误。
* 503，无法获得服务。
* 504，网关超时。
* 505，不支持的http版本。

## 50、scrapy shell 添加user-agent等方法

scrapy shell 命令

1.scrapy shell url　　#url指你所需要爬的网址

2.有些网址数据的爬取需要user-agent，scrapy shell中可以直接添加头文件，

第①种方法

　　 scrapy shell -s USER\_AGENT="Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.86 Safari/537.36" url　　#url指你所需要爬的网址

第②种方法

　　scrapy shell

　　from scrapy import Request

　　req=Request("url",headers={‘User-Agent‘:"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.86 Safari/537.36"})

　　resp=fetch(req)

3.对于爬取到的网址的数据，希望将其保存到本地文件中，

　　with open("e:/python/text.txt",‘w‘) as f:

　　　　f.write(response.body.decode(‘utf-8‘))

4.scrapy shell对于检验正则表达式和css以及xpath是很方便的用法，

scrapy shell -s USER\_AGENT="Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.86 Safari/537.36" url　　#该步骤之后将得到一个页面请求的回答信息response

response.css(".job-name::attr(title)")　　#使用css提取到具体数据

## 51、with open使用方法

1. with open（’文件名‘，‘读写方式’）as f:
2. f.read()  #读取是整个文件
3. f.readline()  #读取第一行
4. f.readlines()    #读取每一行，可以结合for使用(参考我上述完整代码),记得都要带方法都要带括号，不然返回的是内存地址

## 52、virtualenv 新建项目

>>> virtualenv project\_name

>>> cd project\_name

>>> cd bin

>>> source activate #进入虚拟环境

>>> deactivate #退出虚拟环境

>>> virtualenv -p python2/3路径 project\_name #选择创建Python版本的虚拟环境

使用virtualenvwrapper之后

>>> mkvirtualenv project\_name #创建项目

>>> workon project\_name #进入项目文件夹

>>> mkvirtualenv –python=Python版本路径 project\_name #创建对应Python版本虚拟环境

进入虚拟环境后，通过scrapy 新建项目

>>> scrapy genspider jobbole blog.jobbole.com # 格式如示例

## 53、scrapy 查看模板命令

~ scrapy genspider –list #查看模板类型命令

(article\_spider) bogon:~ futeen$ scrapy genspider --list

Available templates:

basic

crawl

csvfeed

xmlfeed

~ scrapy genspider -t crawl lagou [www.lagou.com](http://www.lagou.com) #指定生成的模板类型

## 54、os.path

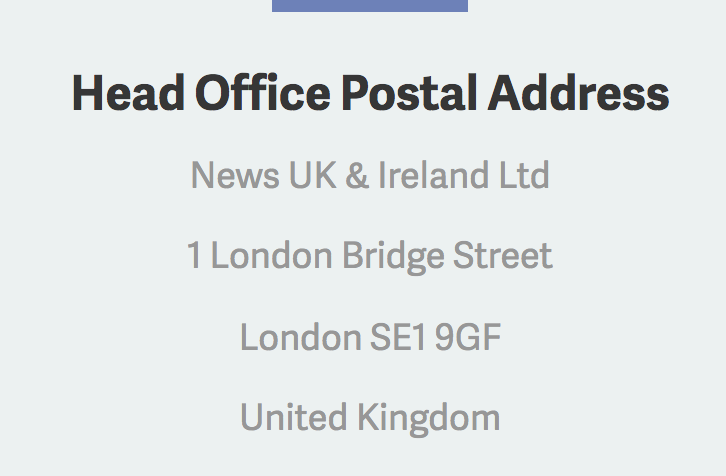
os.path.abspath

os.path.dirname()

os.path.join()

sys.path.insert()

## 55、postal address format



## 56、elasticsearch启动几配合项

Elasticsearch 🡪 文件夹下 ./elasticsearch

Elastcisearch-head 🡪 npm run start

Kibana 🡪 文件夹下 ./kibana

## 57、super()函数

**super()** 函数是用于调用父类(超类)的一个方法。

class A:

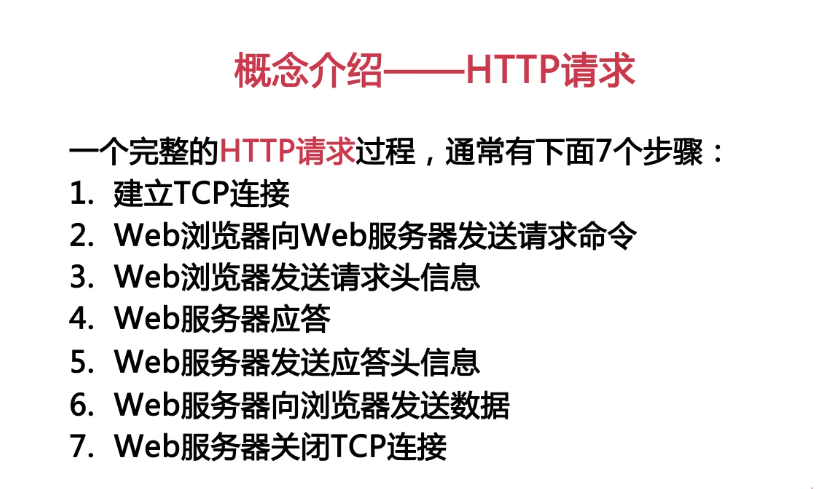
pass

class B(A):

def add(self, x):

super().add(x)

## 58、一个完整的HTTP请求过程



## 59、XMLHttpRequest发送请求



## 60、open()中参数代表含义

Character Meaning  
--------- ---------------------------------------------------------------  
'r' open for reading (default)  
'w' open for writing, truncating the file first  
'x' create a new file and open it for writing  
'a' open for writing, appending to the end of the file if it exists  
'b' binary mode  
't' text mode (default)  
'+' open a disk file for updating (reading and writing)  
'U' universal newline mode (deprecated)

**合法的mode有：**

**r、rb、r+、rb+、w、wb、w+、wb+、a、ab、a+、ab+**

## 61、python类操作符



## 62、MacOS MySQL重启服务

个人\磁盘\usr\local\mysql\support-files 目录下执行命令

启动MySQL服务

sudo /usr/local/MySQL/support-files/mysql.server start

停止MySQL服务

sudo /usr/local/mysql/support-files/mysql.server stop

重启MySQL服务

sudo /usr/local/mysql/support-files/mysql.server restart

## 63、rm 命令

-d：直接把欲删除的目录的硬连接数据删除成0，删除该目录；

-f：强制删除文件或目录；

-i：删除已有文件或目录之前先询问用户；

-r或-R：递归处理，将指定目录下的所有文件与子目录一并处理；

--preserve-root：不对根目录进行递归操作；

-v：显示指令的详细执行过程。

## 64、Remove Mysql Completely

Remove MySQL completely  
Open the Terminal  
  
Use mysqldump to backup your databases  
  
Check for MySQL processes with: ps -ax | grep mysql  
  
Stop and kill any MySQL processes  
  
Analyze MySQL on HomeBrew:  
  
brew remove mysql  
brew cleanup  
Remove files:  
  
sudo rm /usr/local/mysql  
sudo rm -rf /usr/local/var/mysql  
sudo rm -rf /usr/local/mysql\*  
sudo rm ~/Library/LaunchAgents/homebrew.mxcl.mysql.plist  
sudo rm -rf /Library/StartupItems/MySQLCOM  
sudo rm -rf /Library/PreferencePanes/My\*  
Unload previous MySQL Auto-Login:  
  
launchctl unload -w ~/Library/LaunchAgents/homebrew.mxcl.mysql.plist  
Remove previous MySQL Configuration:  
  
subl /etc/hostconfig`   
# Remove the line MYSQLCOM=-YES-  
Remove previous MySQL Preferences:  
  
rm -rf ~/Library/PreferencePanes/My\*  
sudo rm -rf /Library/Receipts/mysql\*  
sudo rm -rf /Library/Receipts/MySQL\*  
sudo rm -rf /private/var/db/receipts/\*mysql\*  
Restart your computer just to ensure any MySQL processes are killed  
  
Try to run mysql, it shouldn't work

## 65、Python的 函数 参数为数据结构和表的区别

a = 1

**def** change\_integer(a):

    a = a + 1

**return** a

**print** change\_integer(a)

**print** a

#===(Python中 "#" 后面跟的内容是注释，不执行 )

b = [1,2,3]

**def** change\_list(b):

    b[0] = b[0] + 1

**return** b

**print** change\_list(b)

**print** b

第一个例子，我们将一个整数变量传递给函数，函数对它进行操作，但原整数变量a不发生变化。

第二个例子，我们将一个表传递给函数，函数进行操作，原来的表b发生变化。

对于基本数据类型的变量，变量传递给函数后，函数会在内存中复制一个新的变量，从而不影响原来的变量。（我们称此为值传递）

但是对于表来说，表传递给函数的是一个指针，指针指向序列在内存中的位置，在函数中对表的操作将在原有内存中进行，从而影响原有变量。 （我们称此为指针传递）

## 66、python传递机制

1、**值传递**类型例如以下：  
a = 1

def change\_integer(a):

a = a + 1

return a

print change\_integer(a)

print a  
上面的代码仅仅是将a的值作为参数传递进去。程序运行完之后a的值并未改变  
**2、引用传递**的类型例如以下：  
b = [1, 2, 3]  
def change\_list(b):

b[0]= b[0]+1

return b

print change\_list(b)

print b  
而这里将b的地址传递进去之后，程序运行完之后b的值也随之改变。

**如果你传入的参数对象是可变对象：列表，字典，这个时候就是引用传递，如果参数在函数体内被修改，那么源对象也会被修改。**

**如果你传入的参数对象是不可变的对象：数字，元组，字符串，这个时候就是值传递。那么源对象是不会改变的**

## 67、解决supervisor端口被占用问题：

报错：

Error: Another program is already listening on a port that one of our HTTP servers is configured to use. Shut this program down first before starting supervisord.

For help, use /usr/local/bin/supervisord -h

解决：

**ps -ef | grep supervisord**

501 958 1 0 7:16下午 ?? 0:03.56 /usr/local/Cellar/supervisor/3.3.4/libexec/bin/python /usr/local/opt/supervisor/bin/supervisord -c /usr/local/etc/supervisord.ini --nodaemon

501 12029 877 0 11:06下午 ttys001 0:00.00 grep supervisord

**kill -s SIGTERM 958**

**supervisorctl**

## 68、xpath表达公式：

//tag[@attibute='value']

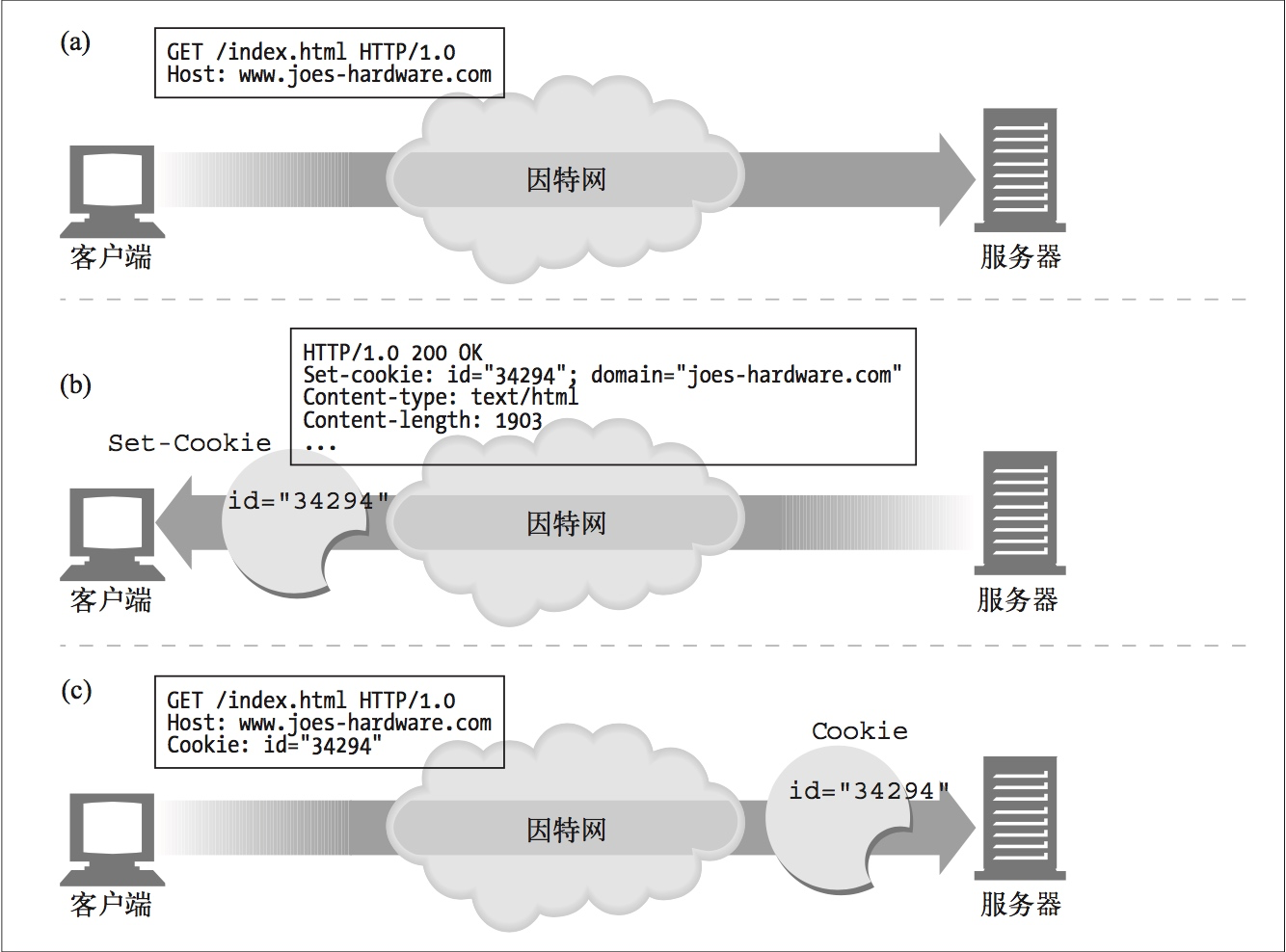
## 69、爬虫之cookie和session

**什么是Cookie**

其实简单的说就是当用户通过http协议访问一个服务器的时候，这个服务器会将一些Name/Value键值对返回给客户端浏览器，并将这些数据加上一些限制条件。在条件符合时，这个用户下次再访问服务器的时候，数据又被完整的带给服务器。

因为http是一种无状态协议，用户首次访问web站点的时候，服务器对用户一无所知。而Cookie就像是服务器给每个来访问的用户贴的标签，而这些标签就是对来访问的客户端的独有的身份的一个标识，这里就如同每个人的身份证一样，带着你的个人信息。而当一个客户端第一次连接过来的时候，服务端就会给他打一个标签，这里就如同给你发了一个身份证，当你下载带着这个身份证来的时候，服务器就知道你是谁了。所以Cookie是存在客户端的，这里其实就是在你的浏览器中。

Cookie中包含了一个由名字=值（name = value）这样的信息构成的任意列表，通过Set-Cookie或Set-Cookie2 HTTP响应（扩展）首部将其贴到客户端身上。如下图例子所示：



其实这里有一个非常典型的应用，就是关于你登录很多网站的账号信息，你让记住密码之后，一段时间内，不需要输入密码，每次都是登录状态

**Cookie的分类**

这里Cookie主要分为两种：  
会话Cookie：不设置过期时间，保存在浏览器的内存中，关闭浏览器，Cookie便被销毁  
普通Cookie:设置了过期时间，保存在硬盘上

**Cookie属性**

因为最开始的cookie是网景公司定义的，后来又有了RFC版本所以当前的Cookie有两个版本：Version 0 Version 1 他们有两种设置响应头的标识，分别是：Set-Cookie和Set-Cookie2,这也造成了一些属性的不同,这里需要注意：常用的为Version 0

**Version 0的属性**

NAME = Value :键值对设置要保存的Name/Value,这里的name不能喝其他属性的名字一样  
Expires：过期时间  
Domain：生成该Cookie的域名  
Path： 该Cookie是在当前的哪个路径下生成  
Secure：如果设置了这个属性，那么只会在SSH连接时才会回传该Cookie

**Version 1的属性**

Name=VALUE:键值对设置要保存的Name/Value,这里的name不能喝其他属性的名字一样  
Comment：主是想，用于说明该Cookie有什么用途  
CommentURL:该服务器为此COokie提供URI注释  
Discard:是否在回话结束丢弃该Cookie，默认为false  
Domain:生成该Cookie的域名  
Max-Age:最大失效时间，与Version 0不同的是这里设置的是在多少秒后失效  
Path:该Cookie是在当前的哪个路径下生成  
Port:该 Cookie 在什么端口下可以回传服务端，如果有多个端口，以逗号隔开  
Secure:如果设置了这个属性，那么只会在SSH连接时才会回传该Cookie

**关于Session**

上面我们知道了Cookie可以让服务器端跟踪每个客户端的访问，但是每次客户端的访问都必须传回这些 Cookie，如果 Cookie 很多，这无形地增加了客户端与服务端的数据传输量，而 Session 的出现正是为了解决这个问题。

同一个客户端每次和服务端交互时，不需要每次都传回所有的Cookie值，而是只要传回一个ID这个ID是客户端第一次访问服务器的时候生成的，而且每个客户端是唯一的。这样每个客户端就有了一个唯一的ID，客户端只要传回这个ID就行了，这个ID通常是NANE为JSESIONID的一个Cookie。所以Session其实是利用Cookie进行信息处理的。

* 1. cookie和session的共同之处在于：cookie和session都是用来跟踪浏览器用户身份的会话方式。
  2. cookie 和session的区别是：cookie数据保存在客户端，session数据保存在服务器端。
  3. cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗,如果主要考虑到安全应当使用session，当然也没有绝对的安全，只是相对cookie，session更加安全
  4. session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，如果主要考虑到减轻服务器性能方面，应当使用COOKIE
  5. cookie和session各有优缺点，所以将登陆信息等重要信息存放为SESSION;其他信息如果需要保留，可以放在COOKIE中

## 70、爬虫去重方法：



## 71、git删除多于的global config：

git config –global –unset use.name

**删除一个配置项**

命令参数

–unset

格式：git config [–local|–global|–system]

–unset section.key

## 72、MongoDB文章整理：

Python操作MongoDB看这一篇就够了

MongoDB是由C++语言编写的非关系型数据库，是一个基于分布式文件存储的开源数据库系统，其内容存储形式类似JSON对象，它的字段值可以包含其他文档、数组及文档数组，非常灵活。在这一节中，我们就来看看Python 3下MongoDB的存储操作。

1. 准备工作

在开始之前，请确保已经安装好了MongoDB并启动了其服务，并且安装好了Python的PyMongo库。

2. 连接MongoDB

连接MongoDB时，我们需要使用PyMongo库里面的MongoClient。一般来说，传入MongoDB的IP及端口即可，其中第一个参数为地址host，第二个参数为端口port（如果不给它传递参数，默认是27017）：

**import** pymongo  
client = pymongo.MongoClient(host='localhost', port=27017)

这样就可以创建MongoDB的连接对象了。

另外，MongoClient的第一个参数host还可以直接传入MongoDB的连接字符串，它以mongodb开头，例如：

client = MongoClient('mongodb://localhost:27017/')

这也可以达到同样的连接效果。

3. 指定数据库

MongoDB中可以建立多个数据库，接下来我们需要指定操作哪个数据库。这里我们以test数据库为例来说明，下一步需要在程序中指定要使用的数据库：

db = client.test

这里调用client的test属性即可返回test数据库。当然，我们也可以这样指定：

db = client['test']

这两种方式是等价的。

4. 指定集合

MongoDB的每个数据库又包含许多集合（collection），它们类似于关系型数据库中的表。

下一步需要指定要操作的集合，这里指定一个集合名称为students。与指定数据库类似，指定集合也有两种方式：

collection = db.students

collection = db['students']

这样我们便声明了一个Collection对象。

5. 插入数据

接下来，便可以插入数据了。对于students这个集合，新建一条学生数据，这条数据以字典形式表示：

student = {  
    'id': '20170101',  
    'name': 'Jordan',  
    'age': 20,  
    'gender': 'male'  
}

这里指定了学生的学号、姓名、年龄和性别。接下来，直接调用collection的insert()方法即可插入数据，代码如下：

result = collection.insert(student)

print(result)

在MongoDB中，每条数据其实都有一个\_id属性来唯一标识。如果没有显式指明该属性，MongoDB会自动产生一个ObjectId类型的\_id属性。insert()方法会在执行后返回\_id值。

运行结果如下：

5932a68615c2606814c91f3d

当然，我们也可以同时插入多条数据，只需要以列表形式传递即可，示例如下：

student1 = {  
    'id': '20170101',  
    'name': 'Jordan',  
    'age': 20,  
    'gender': 'male'  
}  
  
student2 = {  
    'id': '20170202',  
    'name': 'Mike',  
    'age': 21,  
    'gender': 'male'  
}  
  
result = collection.insert([student1, student2])  
print(result)

返回结果是对应的\_id的集合：

[ObjectId('5932a80115c2606a59e8a048'), ObjectId('5932a80115c2606a59e8a049')]

实际上，在PyMongo 3.x版本中，官方已经不推荐使用insert()方法了。当然，继续使用也没有什么问题。官方推荐使用insert\_one()和insert\_many()方法来分别插入单条记录和多条记录，示例如下：

student = {  
    'id': '20170101',  
    'name': 'Jordan',  
    'age': 20,  
    'gender': 'male'  
}  
  
result = collection.insert\_one(student)  
print(result)  
print(result.inserted\_id)

运行结果如下：

<pymongo.results.InsertOneResult object at 0x10d68b558>  
5932ab0f15c2606f0c1cf6c5

与insert()方法不同，这次返回的是InsertOneResult对象，我们可以调用其inserted\_id属性获取\_id。

对于insert\_many()方法，我们可以将数据以列表形式传递，示例如下：

student1 = {  
    'id': '20170101',  
    'name': 'Jordan',  
    'age': 20,  
    'gender': 'male'  
}  
  
student2 = {  
    'id': '20170202',  
    'name': 'Mike',  
    'age': 21,  
    'gender': 'male'  
}  
  
result = collection.insert\_many([student1, student2])  
print(result)  
print(result.inserted\_ids)

运行结果如下：

<pymongo.results.InsertManyResult object at 0x101dea558>  
[ObjectId('5932abf415c2607083d3b2ac'), ObjectId('5932abf415c2607083d3b2ad')]

该方法返回的类型是InsertManyResult，调用inserted\_ids属性可以获取插入数据的\_id列表。

6. 查询

插入数据后，我们可以利用find\_one()或find()方法进行查询，其中find\_one()查询得到的是单个结果，find()则返回一个生成器对象。示例如下：

result = collection.find\_one({'name': 'Mike'})  
print(type(result))  
print(result)

这里我们查询name为Mike的数据，它的返回结果是字典类型，运行结果如下：

<class 'dict'>  
{'\_id': ObjectId('5932a80115c2606a59e8a049'), 'id': '20170202', 'name': 'Mike', 'age': 21, 'gender': 'male'}

可以发现，它多了\_id属性，这就是MongoDB在插入过程中自动添加的。

此外，我们也可以根据ObjectId来查询，此时需要使用bson库里面的objectid：

**from** bson.objectid **import** ObjectId  
  
result = collection.find\_one({'\_id': ObjectId('593278c115c2602667ec6bae')})  
print(result)

其查询结果依然是字典类型，具体如下：

{'\_id': ObjectId('593278c115c2602667ec6bae'), 'id': '20170101', 'name': 'Jordan', 'age': 20, 'gender': 'male'}

当然，如果查询结果不存在，则会返回None。

对于多条数据的查询，我们可以使用find()方法。例如，这里查找年龄为20的数据，示例如下：

results = collection.find({'age': 20})  
print(results)  
**for** result **in** results:  
    print(result)

运行结果如下：

<pymongo.cursor.Cursor object at 0x1032d5128>  
{'\_id': ObjectId('593278c115c2602667ec6bae'), 'id': '20170101', 'name': 'Jordan', 'age': 20, 'gender': 'male'}  
{'\_id': ObjectId('593278c815c2602678bb2b8d'), 'id': '20170102', 'name': 'Kevin', 'age': 20, 'gender': 'male'}  
{'\_id': ObjectId('593278d815c260269d7645a8'), 'id': '20170103', 'name': 'Harden', 'age': 20, 'gender': 'male'}

返回结果是Cursor类型，它相当于一个生成器，我们需要遍历取到所有的结果，其中每个结果都是字典类型。

如果要查询年龄大于20的数据，则写法如下：

results = collection.find({'age': {'$gt': 20}})

这里查询的条件键值已经不是单纯的数字了，而是一个字典，其键名为比较符号$gt，意思是大于，键值为20。

这里将比较符号归纳为下表。

| **符号** | **含义** | **示例** |
| --- | --- | --- |
| $lt | 小于 | {'age': {'$lt': 20}} |
| $gt | 大于 | {'age': {'$gt': 20}} |
| $lte | 小于等于 | {'age': {'$lte': 20}} |
| $gte | 大于等于 | {'age': {'$gte': 20}} |
| $ne | 不等于 | {'age': {'$ne': 20}} |
| $in | 在范围内 | {'age': {'$in': [20, 23]}} |
| $nin | 不在范围内 | {'age': {'$nin': [20, 23]}} |

另外，还可以进行正则匹配查询。例如，查询名字以M开头的学生数据，示例如下：

results = collection.find({'name': {'$regex': '^M.\*'}})

这里使用$regex来指定正则匹配，^M.\*代表以M开头的正则表达式。

这里将一些功能符号再归类为下表。

| **符号** | **含义** | **示例** | **示例含义** |
| --- | --- | --- | --- |
| $regex | 匹配正则表达式 | {'name': {'$regex': '^M.\*'}} | name以M开头 |
| $exists | 属性是否存在 | {'name': {'$exists': True}} | name属性存在 |
| $type | 类型判断 | {'age': {'$type': 'int'}} | age的类型为int |
| $mod | 数字模操作 | {'age': {'$mod': [5, 0]}} | 年龄模5余0 |
| $text | 文本查询 | {'$text': {'$search': 'Mike'}} | text类型的属性中包含Mike字符串 |
| $where | 高级条件查询 | {'$where': 'obj.fans\_count == obj.follows\_count'} | 自身粉丝数等于关注数 |

关于这些操作的更详细用法，可以在MongoDB官方文档找到：  
https://docs.mongodb.com/manual/reference/operator/query/。

7. 计数

要统计查询结果有多少条数据，可以调用count()方法。比如，统计所有数据条数：

count = collection.find().count()

print(count)

或者统计符合某个条件的数据：

count = collection.find({'age': 20}).count()  
print(count)

运行结果是一个数值，即符合条件的数据条数。

8. 排序

排序时，直接调用sort()方法，并在其中传入排序的字段及升降序标志即可。示例如下：

results = collection.find().sort('name', pymongo.ASCENDING)  
print([result['name'] **for** result **in** results])

运行结果如下：

['Harden', 'Jordan', 'Kevin', 'Mark', 'Mike']

这里我们调用pymongo.ASCENDING指定升序。如果要降序排列，可以传入pymongo.DESCENDING。

9. 偏移

在某些情况下，我们可能想只取某几个元素，这时可以利用skip()方法偏移几个位置，比如偏移2，就忽略前两个元素，得到第三个及以后的元素：

results = collection.find().sort('name', pymongo.ASCENDING).skip(2)  
print([result['name'] **for** result **in** results])

运行结果如下：

['Kevin', 'Mark', 'Mike']

另外，还可以用limit()方法指定要取的结果个数，示例如下：

results = collection.find().sort('name', pymongo.ASCENDING).skip(2).limit(2)  
print([result['name'] **for** result **in** results])

运行结果如下：

['Kevin', 'Mark']

如果不使用limit()方法，原本会返回三个结果，加了限制后，会截取两个结果返回。

值得注意的是，在数据库数量非常庞大的时候，如千万、亿级别，最好不要使用大的偏移量来查询数据，因为这样很可能导致内存溢出。此时可以使用类似如下操作来查询：

**from** bson.objectid **import** ObjectId  
collection.find({'\_id': {'$gt': ObjectId('593278c815c2602678bb2b8d')}})

这时需要记录好上次查询的\_id。

10. 更新

对于数据更新，我们可以使用update()方法，指定更新的条件和更新后的数据即可。例如：

condition = {'name': 'Kevin'}  
student = collection.find\_one(condition)  
student['age'] = 25  
result = collection.update(condition, student)  
print(result)

这里我们要更新name为Kevin的数据的年龄：首先指定查询条件，然后将数据查询出来，修改年龄后调用update()方法将原条件和修改后的数据传入。

运行结果如下：

{'ok': 1, 'nModified': 1, 'n': 1, 'updatedExisting': **True**}

返回结果是字典形式，ok代表执行成功，nModified代表影响的数据条数。

另外，我们也可以使用$set操作符对数据进行更新，代码如下：

result = collection.update(condition, {'$set': student})

这样可以只更新student字典内存在的字段。如果原先还有其他字段，则不会更新，也不会删除。而如果不用$set的话，则会把之前的数据全部用student字典替换；如果原本存在其他字段，则会被删除。

另外，update()方法其实也是官方不推荐使用的方法。这里也分为update\_one()方法和update\_many()方法，用法更加严格，它们的第二个参数需要使用$类型操作符作为字典的键名，示例如下：

condition = {'name': 'Kevin'}  
student = collection.find\_one(condition)  
student['age'] = 26  
result = collection.update\_one(condition, {'$set': student})  
print(result)  
print(result.matched\_count, result.modified\_count)

这里调用了update\_one()方法，第二个参数不能再直接传入修改后的字典，而是需要使用{'$set': student}这样的形式，其返回结果是UpdateResult类型。然后分别调用matched\_count和modified\_count属性，可以获得匹配的数据条数和影响的数据条数。

运行结果如下：

<pymongo.results.UpdateResult object at 0x10d17b678>  
1 0

我们再看一个例子：

condition = {'age': {'$gt': 20}}  
result = collection.update\_one(condition, {'$inc': {'age': 1}})  
print(result)  
print(result.matched\_count, result.modified\_count)

这里指定查询条件为年龄大于20，然后更新条件为{'$inc': {'age': 1}}，也就是年龄加1，执行之后会将第一条符合条件的数据年龄加1。

运行结果如下：

<pymongo.results.UpdateResult object at 0x10b8874c8>  
1 1

可以看到匹配条数为1条，影响条数也为1条。

如果调用update\_many()方法，则会将所有符合条件的数据都更新，示例如下：

condition = {'age': {'$gt': 20}}  
result = collection.update\_many(condition, {'$inc': {'age': 1}})  
print(result)  
print(result.matched\_count, result.modified\_count)

这时匹配条数就不再为1条了，运行结果如下：

<pymongo.results.UpdateResult object at 0x10c6384c8>  
3 3

可以看到，这时所有匹配到的数据都会被更新。

11. 删除

删除操作比较简单，直接调用remove()方法指定删除的条件即可，此时符合条件的所有数据均会被删除。示例如下：

result = collection.remove({'name': 'Kevin'})  
print(result)

运行结果如下：

{'ok': 1, 'n': 1}

另外，这里依然存在两个新的推荐方法——delete\_one()和delete\_many()。示例如下：

result = collection.delete\_one({'name': 'Kevin'})  
print(result)  
print(result.deleted\_count)  
result = collection.delete\_many({'age': {'$lt': 25}})  
print(result.deleted\_count)

运行结果如下：

<pymongo.results.DeleteResult object at 0x10e6ba4c8>  
1  
4

delete\_one()即删除第一条符合条件的数据，delete\_many()即删除所有符合条件的数据。它们的返回结果都是DeleteResult类型，可以调用deleted\_count属性获取删除的数据条数。

12. 其他操作

另外，PyMongo还提供了一些组合方法，如find\_one\_and\_delete()、find\_one\_and\_replace()和find\_one\_and\_update()，它们是查找后删除、替换和更新操作，其用法与上述方法基本一致。

另外，还可以对索引进行操作，相关方法有create\_index()、create\_indexes()和drop\_index()等。

关于PyMongo的详细用法，可以参见官方文档：http://api.mongodb.com/python/current/api/pymongo/collection.html。

另外，还有对数据库和集合本身等的一些操作，这里不再一一讲解，可以参见官方文档：http://api.mongodb.com/python/current/api/pymongo/。

本节讲解了使用PyMongo操作MongoDB进行数据增删改查的方法。

## 73、xpath基本语句：

//\*[@class=’name’]/a/text()

//\*[@class=’name’]/a/@href

## 74、scrapy的几个命令总结：

以创建tencent项目名称为例：

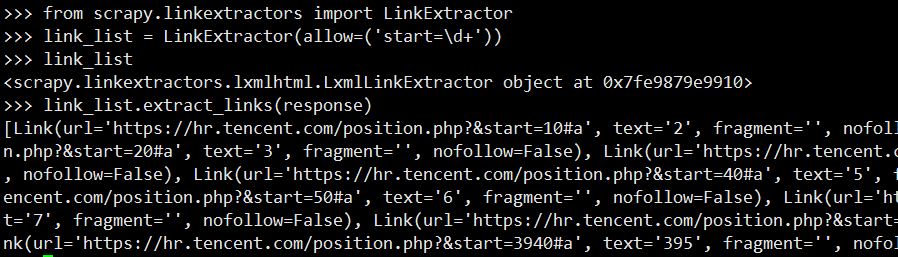
scrapy startproject tencnet

scrapy genspider tencentPosition ‘tencent.com’ #spider名称和项目名称不能相同

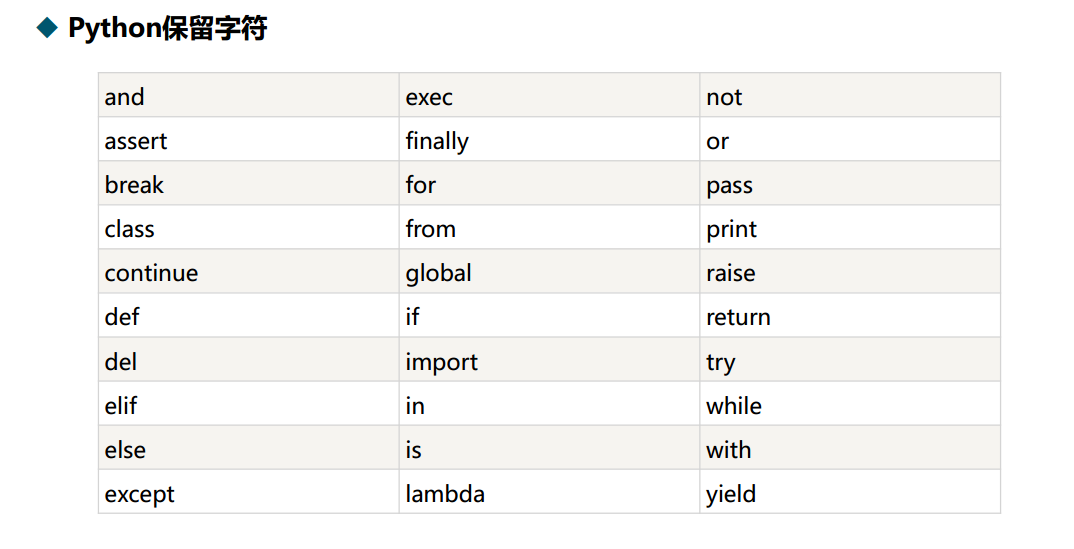
scrapy crawl tencent #启动项目

scrapy genspider -t crawl tencent tencent.com

## 75、scrapy中LinkExtractor的使用：



## 76、python保留字符



## 77、python数据类型转换



## 78、python运算符



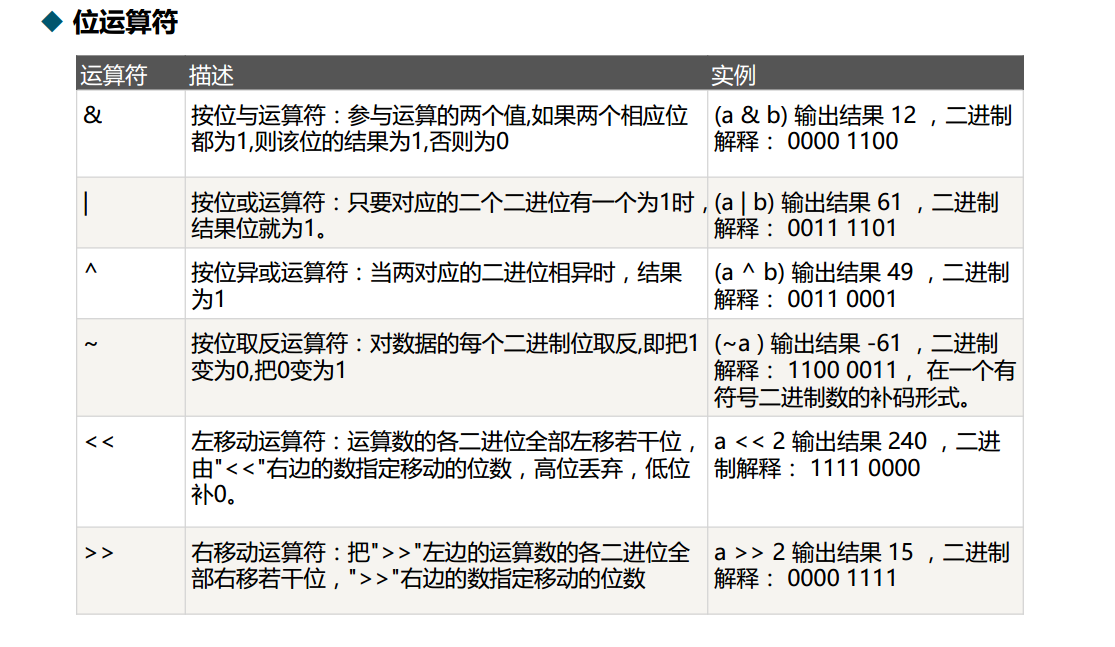
## 79、python比较运算符



## 80、python赋值运算符



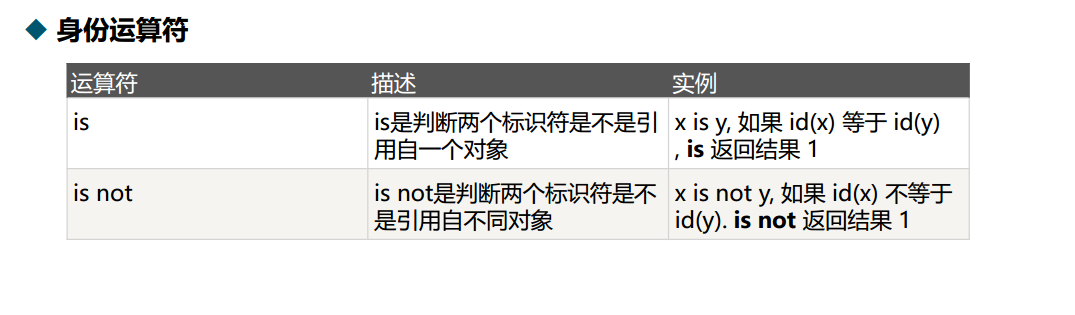
## 81、python位运算符



## 82、python逻辑运算符和成员运算符



## 83、身份运算符



## 84、python运算符优先级



## 85、python循环语句：



## 86、Python嵌套循环：



## 87、Python字符串格式化：





## 88、Python的 open函数：



## 89、Mac更改terminal中ls命令文件夹颜色：

sudo vim .bash\_profile

添加以下命令：

export LS\_OPTIONS='--color=auto'

export CLICOLOR='YES'

export LSCOLORS='**C**' #C代表粗体绿色

退出运行：

source .bash\_profile

颜色字母对照表：

下面是颜色的子母对照：  
    a 黑色  
    b 红色  
    c 绿色  
    d 棕色  
    e 蓝色  
    f 洋红色  
    g 青色  
    h 浅灰色  
    A 黑色粗体  
    B 红色粗体  
    C 绿色粗体  
    D 棕色粗体  
    E 蓝色粗体  
    F 洋红色粗体  
    G 青色粗体  
    H 浅灰色粗体  
    x 系统默认颜色

## 90、数据库conf文件位置：

**ubuntu**

1.conf文件位置：

cd /etc/mongodb.conf

2.db文件位置：

cd /data/db

3.cd /var/lib/mongodb

**Mac**

1. conf文件位置：

cd /usr/local/etc/mongod.conf

1. mongo所在位置：

cd /use/local/bin

3.cd /data/db

redis.conf:

Mac:

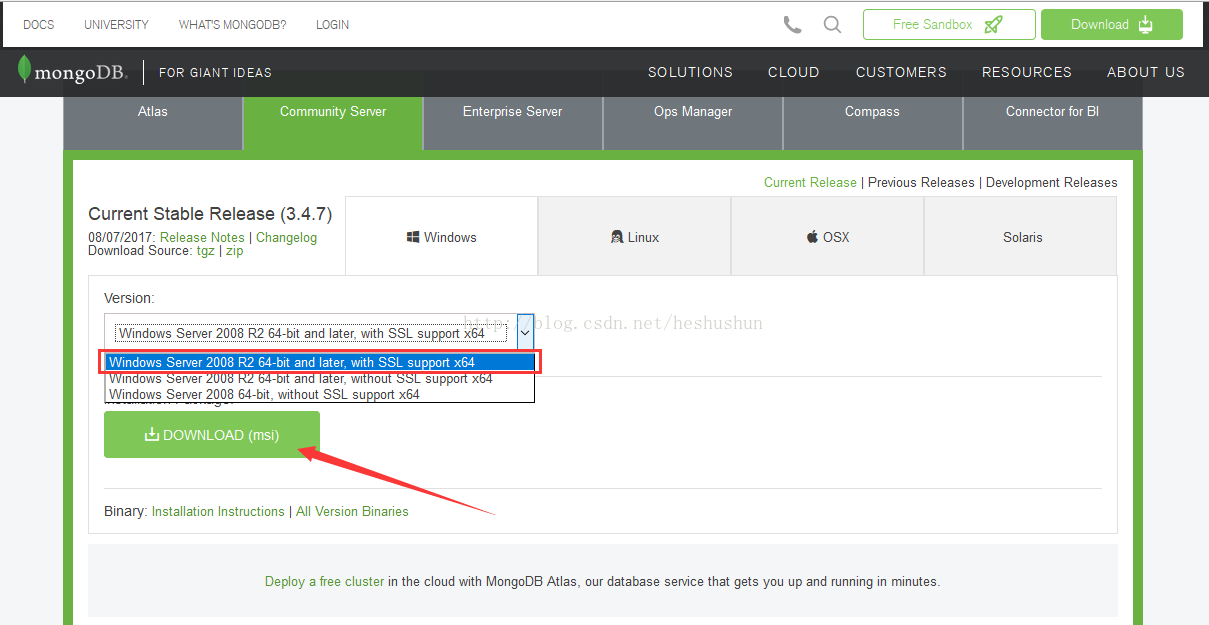
cd /usr/local/etc/redis.conf

ubuntu:

cd /etc/redis/redis.conf

## 91、windows下配置mongodb

**一、先登录Mongodb官网[https://www.mongodb.com/download-center#community](https://www.mongodb.com/download-center" \l "community" \t "_blank) 下载   安装包。32、64位的都行。**

****

**二、安装MongoDB**

**下载后的安装包：https://img-blog.csdn.net/20170901195010442?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvaGVzaHVzaHVu/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center**

**安装比较简单，类似于普通QQ软件，中间主要是选择“Custom”自定义 安装路径修改下：D:\software\MongoDB**

**然后不断“下一步”，安装至结束。**

**安装比较容易。难点在启动Mongodb的服务以及将MongoDB设置成Windows服务，加配置文件在windows的“服务”中找到。**

**三、先创建数据库文件的存放位置**

**在MongoDB下创建data，在data下再创建db：D:\software\MongoDB\data\db**

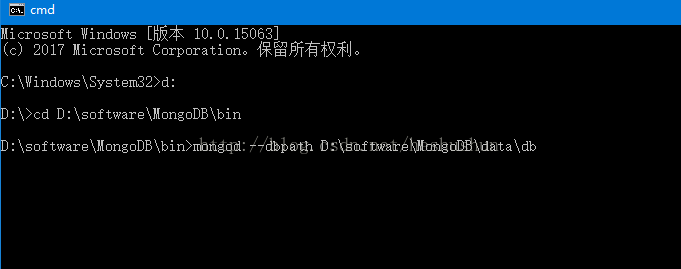
**因为启动mongodb服务之前需要必须创建数据库文件的存放文件夹，否则命令不会自动创建，而且不能启动成功。**

**四、启动MongoDB服务**

**1.打开cmd命令行**

**2.进入D:\software\MongoDB\bin目录（注意：先输入d:进入d盘，然后输入cd D:\software\MongoDB\bin）**

**3.输入如下的命令启动mongodb服务：mongod --dbpath D:\software\MongoDB\data\db**

****

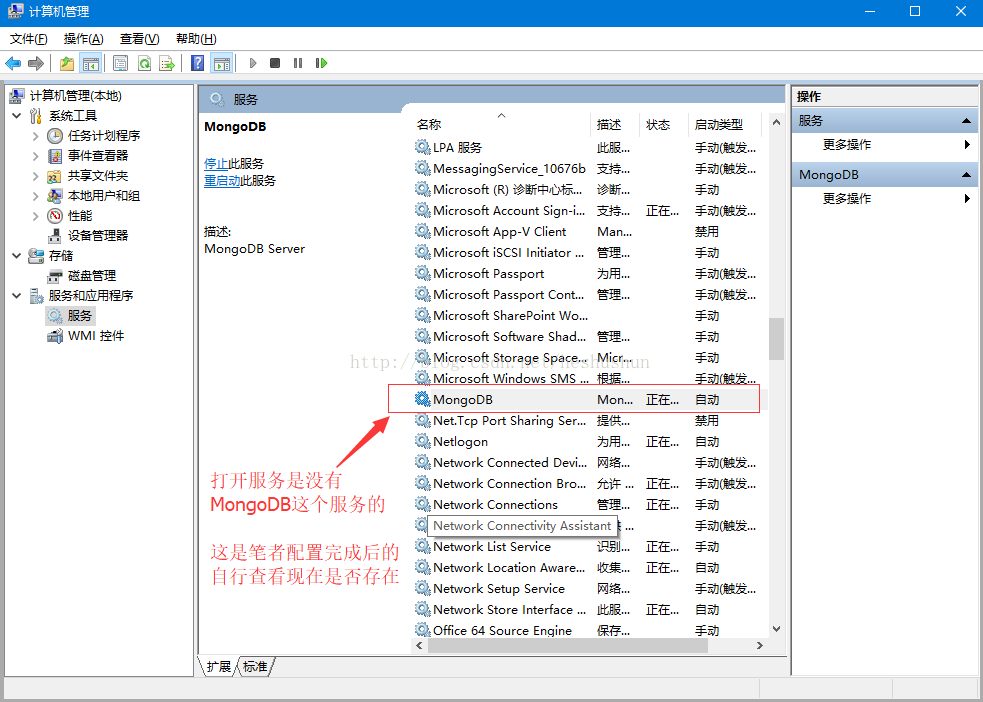
**即是在第三步创建的数据库存放文件路径下启动。**

**4.在浏览器输入http://localhost:27017 （27017是mongodb的端口号）查看，若显示：**

****

**则表示，连接成功。如果不成功，可以查看端口是否被占用。**

**但是在本地windows“服务”中，是没有配置上mongodb 服务的，可以打开“服务”看下**

****

**五、配置本地windows mongodb 服务**

**这样可设置为 开机自启动，可直接手动启动关闭，可通过命令行net start MongoDB 启动。该配置会大大方便。**

**1.先在data文件下创建一个新文件夹log（用来存放日志文件）**

**https://img-blog.csdn.net/20170901202603073?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvaGVzaHVzaHVu/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center**

**2.在Mongodb新建配置文件mongo.config**

****

**可能很多人都不会创建.config配置文件。那给大家介绍下简单的方法：**

**先创建一个mongo.txt文件，再打开，点击”另存为“，将底下的文件类型更改为”全部类型“，并更改文件名称为mongo.config。**

**这样就可以创建一个config的配置文件了。**

**2.用记事本打开mongo.config  ，并输入：**

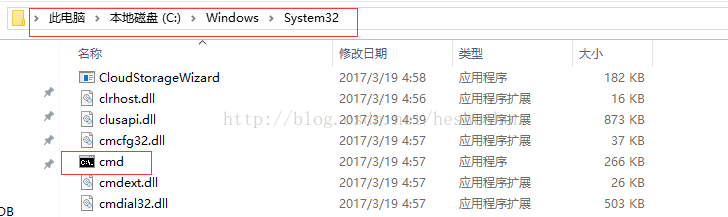
**dbpath=D:\software\MongoDB\data\db**

**logpath=D:\software\MongoDB\data\log\mongo.log**

**3.用管理员身份打开cmd:**

**可能还有很多人不会管理员身份打开cmd。这也介绍下：**

**在下图路径下找到cmd 的运行文件**

****

**然后右键，以管理员身份运行。打开后发现在顶端比普通打开的多了”管理员“三个字**

****

**4.配置windows服务：**

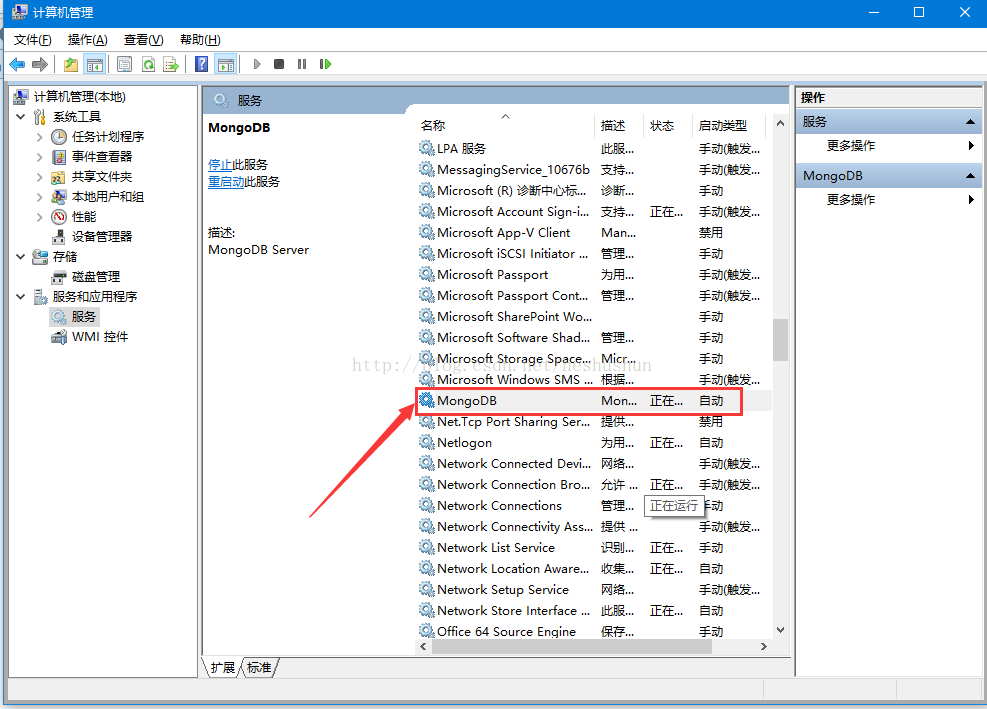
**cmd先跳转到 D:\software\MongoDB\bin目录下。**

**输入：mongod --config D:\software\Mongodb\mongo.config --install --serviceName "MongoDB"**

**即根据刚创建的mongo.config配置文件安装服务，名称为MongoDB。**

****

**完成后，再次查看本地的服务。**

****

**如果成功的话，会发现本地服务多了”MongoDB"服务。**

## 92、git上传代码流程：

以增加readme文件到github.com为例：

$ git init #本地创建仓库

$ touch readme #创建文件

$ git pull origin master #同步远程的master

$ git add readme

$ git commit -m ‘first commit’

$ git push origin master #上传到github

另：测试本地是否和github.com联通：

$ ssh git@github.com

**93、git 删除文件夹及文件夹中文件：**

**以文件夹hello为例：**

**$ git rm -rf hello**

**$ git commit -m ‘delete hello’**

**$ git push**

## 93、linux安装scrapy-redis后路径：

/usr/local/lib/python2.7/dist-packages

## 94、scrapy-redis爬虫启动命令：

$ scrapy runspider [spider\_name].py #spider\_name是爬虫逻辑文件名称

## 95、redis连接指定服务器命令：

如：Mac连接windows，其中windows的ip地址为192.168.1.108，

$ sudo redis-cli -h 192.168.1.108

另：爬虫端无需启动redis-server, Master端启动即可。只要Slave端读取到了Master端的Redis数据库，则表示连接成功，可以实施分布式。

## 96、scrapy管道组件运行顺序：

ITEM\_PIPELINES = {

'youyuan.pipelines.YouyuanPipeline': 300,

'scrapy\_redis.pipelines.RedisPipeline' : 400,

}

# 数字越小，级别越高

## 97、Linux打包、解包命令：

例如：打包youyuan文件夹：

$ tar **-cvf** youyuan.tar youyuan # 仅仅打包，未压缩空间

$ tar **-cxvf** youyuan.tar.gz youyuan #具备压缩功能，使数据变小

解包：

$ tar -xvf youyuan.tar

$ tar -zvf youyuan.tar.gz

## 98、发送包的命令：

**$** sftp [python@192.168.21.84](mailto:python@192.168.21.84) # ip随机

sftp> ls #远程列表

sftp> lls #本地列表

sftp> put youyuan.tar #接上条的youyuan

## 99、scrapy分布式settings设置：

# 使用了scrapy-redis里的去重组件，不使用scrapy默认的去重

**DUPEFILTER\_CLASS = "scrapy\_redis.dupefilter.RFPDupeFilter"**

# 使用了scrapy-redis里的调度器组件，不实用scrapy默认的调度器

**SCHEDULER = "scrapy\_redis.scheduler.Scheduler"**

# 使用队列形式

S**CHEDULER\_QUEUE\_CLASS = "scrapy\_redis.queue.SpiderQueue"**

ITEM\_PIPELINES = {

'youyuan.pipelines.YouyuanPipeline': 300,

'scrapy\_redis.pipelines.RedisPipeline' : 400,

}

## 100、修改文件名称技巧：

$ mv name\_a name\_b # 文件夹移动时可重新命名，移动位置不变即可

## 101、跨站请求伪造（CSRF/XSRF）

　CSRF（Cross-site request forgery跨站请求伪造，也被称为“One Click Attack”或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。尽管听起来像跨站脚本（XSS），但它与XSS非常不同，并且攻击方式几乎相左。XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。

## 102、kali远程ssh开启命令：

$ service ssh start

## 103、查看某个端口是否被监听：

以2132为例：

$ lsof -i: 2131

## 104、kafka命令：

$ bin/zookeeper-server-start.sh config/zookeeper.properties

$ bin/kafka-server-start.sh config/server.properties

#创建话题

$ bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic test1 --partitions 3 --replication-factor 1

$ bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic test1

#创建消费者

$ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test1

$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test1 #另外一个窗口

## 105、jupyter notebook简单操作指南：

Python Jupyter 操作指南

打开Anaconda prompt窗口

资料与代码将存储在一个文件下，需要打开资料所在的文件夹位置

-转换到资料所在的磁盘，例如：D盘

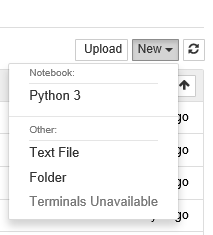
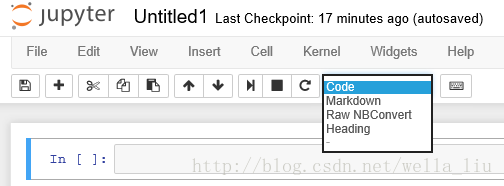
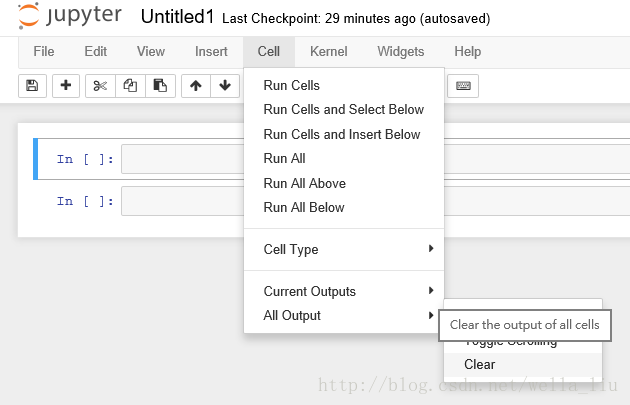
D:

* 打开D盘的文件夹，例如：D:资料（输入cd+空格，按TAB会出现D盘的文件夹目录）

D:>cd 资料

* 输入jupyter notebook ，按enter

D:\资料>jupyter notebook

* 进入jupyter页面，选择Python3   
  
* 进入编辑页面，可通过选择代码行或者文本行   
  
* 程序需要重新跑时，可进行如下操作（将之前命令产生的中间文件取消）   
  
* 按shift+enter可运行该段程式

## 106、消息系统：

为什么要用消息系统

* **解耦**  
  在项目启动之初来预测将来项目会碰到什么需求，是极其困难的。消息队列在处理过程中间插入了一个隐含的、基于数据的接口层，两边的处理过程都要实现这一接口。这允许你独立的扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束
* **冗余**  
  有些情况下，处理数据的过程会失败。除非数据被持久化，否则将造成丢失。消息队列把数据进行持久化直到它们已经被完全处理，通过这一方式规避了数据丢失风险。在被许多消息队列所采用的”插入-获取-删除”范式中，在把一个消息从队列中删除之前，需要你的处理过程明确的指出该消息已经被处理完毕，确保你的数据被安全的保存直到你使用完毕。
* **扩展性**  
  因为消息队列解耦了你的处理过程，所以增大消息入队和处理的频率是很容易的；只要另外增加处理过程即可。不需要改变代码、不需要调节参数。扩展就像调大电力按钮一样简单。
* **灵活性 & 峰值处理能力**  
  在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见；如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，而不会因为突发的超负荷的请求而完全崩溃。
* **可恢复性**  
  当体系的一部分组件失效，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。而这种允许重试或者延后处理请求的能力通常是造就一个略感不便的用户和一个沮丧透顶的用户之间的区别。
* **送达保证**  
  消息队列提供的冗余机制保证了消息能被实际的处理，只要一个进程读取了该队列即可。在此基础上，IronMQ提供了一个”只送达一次”保证。无论有多少进程在从队列中领取数据，每一个消息只能被处理一次。这之所以成为可能，是因为获取一个消息只是”预定”了这个消息，暂时把它移出了队列。除非客户端明确的表示已经处理完了这个消息，否则这个消息会被放回队列中去，在一段可配置的时间之后可再次被处理。
* **顺序保证**  
  在大多使用场景下，数据处理的顺序都很重要。消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理。IronMO保证消息通过FIFO（先进先出）的顺序来处理，因此消息在队列中的位置就是从队列中检索他们的位置。
* **缓冲**  
  在任何重要的系统中，都会有需要不同的处理时间的元素。例如,加载一张图片比应用过滤器花费更少的时间。消息队列通过一个缓冲层来帮助任务最高效率的执行–写入队列的处理会尽可能的快速，而不受从队列读的预备处理的约束。该缓冲有助于控制和优化数据流经过系统的速度。
* **理解数据流**  
  在一个分布式系统里，要得到一个关于用户操作会用多长时间及其原因的总体印象，是个巨大的挑战。消息队列通过消息被处理的频率，来方便的辅助确定那些表现不佳的处理过程或领域，这些地方的数据流都不够优化。
* **异步通信**  
  很多时候，你不想也不需要立即处理消息。消息队列提供了异步处理机制，允许你把一个消息放入队列，但并不立即处理它。你想向队列中放入多少消息就放多少，然后在你乐意的时候再去处理它们。

常用Message Queue对比

* **RabbitMQ**  
  RabbitMQ是使用Erlang编写的一个开源的消息队列，本身支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正因如此，它非常重量级，更适合于企业级的开发。同时实现了Broker构架，这意味着消息在发送给客户端时先在中心队列排队。对路由，负载均衡或者数据持久化都有很好的支持。
* **Redis**  
  Redis是一个基于Key-Value对的NoSQL数据库，开发维护很活跃。虽然它是一个Key-Value数据库存储系统，但它本身支持MQ功能，所以完全可以当做一个轻量级的队列服务来使用。对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。测试数据分为128Bytes、512Bytes、1K和10K四个不同大小的数据。实验表明：入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则慢的无法忍受；出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于Redis。
* **ZeroMQ**  
  ZeroMQ号称最快的消息队列系统，尤其针对大吞吐量的需求场景。ZMQ能够实现RabbitMQ不擅长的高级/复杂的队列，但是开发人员需要自己组合多种技术框架，技术上的复杂度是对这MQ能够应用成功的挑战。ZeroMQ具有一个独特的非中间件的模式，你不需要安装和运行一个消息服务器或中间件，因为你的应用程序将扮演了这个服务角色。你只需要简单的引用ZeroMQ程序库，可以使用NuGet安装，然后你就可以愉快的在应用程序之间发送消息了。但是ZeroMQ仅提供非持久性的队列，也就是说如果宕机，数据将会丢失。其中，Twitter的Storm 0.9.0以前的版本中默认使用ZeroMQ作为数据流的传输（Storm从0.9版本开始同时支持ZeroMQ和Netty作为传输模块）。
* **ActiveMQ**  
  ActiveMQ是Apache下的一个子项目。 类似于ZeroMQ，它能够以代理人和点对点的技术实现队列。同时类似于RabbitMQ，它少量代码就可以高效地实现高级应用场景。
* **Kafka/Jafka**  
  Kafka是Apache下的一个子项目，是一个高性能跨语言分布式发布/订阅消息队列系统，而Jafka是在Kafka之上孵化而来的，即Kafka的一个升级版。具有以下特性：快速持久化，可以在O(1)的系统开销下进行消息持久化；高吞吐，在一台普通的服务器上既可以达到10W/s的吞吐速率；完全的分布式系统，Broker、Producer、Consumer都原生自动支持分布式，自动实现负载均衡；支持Hadoop数据并行加载，对于像Hadoop的一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。Kafka通过Hadoop的并行加载机制来统一了在线和离线的消息处理。Apache Kafka相对于ActiveMQ是一个非常轻量级的消息系统，除了性能非常好之外，还是一个工作良好的分布式系统。

## 107、jupyter notebook进入目标文件夹办法：

首先通过anaconda打开jupyter，

进入浏览器后打开terminal，

在浏览器terminal中进入目标文件夹，

在浏览器terminal在目标文件夹位置输入jupyter notebook。