目录

[Python编程基础 2](#_Toc477184892)

[变量 2](#_Toc477184893)

[字符串 2](#_Toc477184894)

[find 查找字符串是否含有子串 2](#_Toc477184895)

[replace 字符串替换 2](#_Toc477184896)

[列表 2](#_Toc477184897)

[遍历列表时获取每个元素的index 2](#_Toc477184898)

[字典 3](#_Toc477184899)

[元组 3](#_Toc477184900)

[函数式编程 3](#_Toc477184901)

[map 3](#_Toc477184902)

[filter 3](#_Toc477184903)

[reduce 4](#_Toc477184904)

[lambda 4](#_Toc477184905)

[面向对象 4](#_Toc477184906)

[类 4](#_Toc477184907)

[对象 5](#_Toc477184908)

[设计模式 6](#_Toc477184909)

[单例模式 6](#_Toc477184910)

[装饰器实现 6](#_Toc477184911)

[多线程 6](#_Toc477184912)

[threading.Thread 模块 6](#_Toc477184913)

[队列 7](#_Toc477184914)

[task\_done(): 7](#_Toc477184915)

[join(): 7](#_Toc477184916)

[内置库 7](#_Toc477184917)

[sys 7](#_Toc477184918)

[argv 7](#_Toc477184919)

[os 8](#_Toc477184920)

[argparse 8](#_Toc477184921)

[定义一个解析器 8](#_Toc477184922)

[创建定位参数(positional arguments) 8](#_Toc477184923)

[创建可选参数(optional arguments) 8](#_Toc477184924)

[第三方库 10](#_Toc477184925)

[分析工具 10](#_Toc477184926)

[profilehook 10](#_Toc477184927)

[异步队列 11](#_Toc477184928)

[Celery 11](#_Toc477184929)

[安装: 11](#_Toc477184930)

[配置并运行 11](#_Toc477184931)

作者：梁毅强

邮箱：398504533@qq.com

# Python编程基础

## 变量

## 字符串

### find 查找字符串是否含有子串

s = “this is python string”

print s.find(‘this’)

#如果存在则返回子串在字符串的开始位置，这里是0，如果不存在则返回-1。

print ‘this’ in s

#如果存在则返回True,否则返回False

### replace 字符串替换

字符串替换

s = “a.cn,b.com” #将逗号替换为空格

new\_s = s.replace(“,”, “ ”)

## 列表

### 遍历列表时获取每个元素的index

例：

jk = [1,2,3]

sni = ['aa','','bb']

pt = ['cc','dd','']

def get\_total():

totals = []

for idx,val in enumerate(jk):

#print 'idx:',idx, ' val:',val

totals.append({'jk':val, 'sni':sni[idx], 'pt':pt[idx]})

return totals

print get\_total()

[{'sni': 'aa', 'pt': 'cc', 'jk': 1}, {'sni': '', 'pt': 'dd', 'jk': 2}, {'sni': 'bb', 'pt': '', 'jk': 3}]

更为pythonic的写法

def get\_total2():

return [{'jk':val, 'sni':sni[idx], 'pt':pt[idx]} for idx, val in enumerate(jk)]

## 字典

## 元组

## 函数式编程

### map

用一个函数处理列表里的每个元素,这个函数是处理函数

例子：

sq = [1,2,4,5,6]

def add10(ls):

return map(lambda x:x+10, ls)

def add100(ls):

return map(lambda x:x+100, ls)

ls\_10 = add10(sq)

ls\_100 = add100(sq)

[11, 12, 14, 15, 16]

[101, 102, 104, 105, 106]

上例里 lambda x:x+10 和 lambda x:x+100就是处理函数

### filter

用一个函数对列表的元素进行过滤，这个函数只返回True,False

例子：

sq = range(20)

def is\_odd(x): #判断x是否是奇数

return x % 2 != 0

rs = filter(is\_odd, sq)

#[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

比较pythonic的做法：

filter(lambda x:x%2 != 0, range(20)) #一句话搞定

重点是lambda 语法

### reduce

用一个处理函数每次处理序列的前两个元素

例子：

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

reduce(lambda x,y:x+y, ls) #21

处理方式：(((((1+2)+3)+4)+5)+6)

### lambda

函数原型：lambda [arguments]: expression

lambda 可以定义一个匿名简单函数

例子：

取出列表中是负数的数

sq = [1,3,-4,5,66,10,-8]

print filter(lambda x:x < 0, sq)

lambda x:x<0 转换成函数为

def isNegative(x):

if x < 0:return x

lambda 的表达式包含了返回值。

## 面向对象

### 类

新式类：

class MyClass(object):pass

继承自object

旧式类

无继承

class MyClass():pass

新式类与旧式类\_\_dict\_\_上的不同

**class MyOldTest**()**:pass  
  
class MyTest**(object)**:pass  
  
print** MyTest.\_\_dict\_\_  
**print** MyOldTest.\_\_dict\_\_

{'\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'MyTest' objects>, '\_\_module\_\_': '\_\_main\_\_', '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'MyTest' objects>, '\_\_doc\_\_': None}

{'\_\_module\_\_': '\_\_main\_\_', '\_\_doc\_\_': None}

\_\_dict\_\_在这里是类属性，打印的是类变量

### 对象

\_\_dict\_\_ 对象的这个属性可以打印实例变量，显示在\_\_init\_\_里定义的属性和动态定义的属性

**class MyClass**(object)**:** ivar **=** '22'  
 **def \_\_init\_\_**(self)**:** self.name **=** 'cat'  
  
v **=** MyClass()  
**print** v.\_\_dict\_\_  
v.age **=** 20 #动态定义的属性  
**print** v.\_\_dict\_\_

输出

{'name': 'cat'}

{'age': 20, 'name': 'cat'}

#### classmethod VS staticmethod

### 内省

#### inspect

##### getmembers

*# -\*- coding=utf-8 -\*-***class** MyCar(object):  
 **def** \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 **def** get\_name(self):  
 **return** self.name  
  
 **def** get\_age(self):  
 **return** self.age  
  
  
**import** inspect  
  
car = MyCar(**'aaa'**, 22)  
methods = inspect.getmembers(car, predicate=inspect.ismethod)  
**for** method **in** methods:  
 **if** method[0].startswith(**'get'**):  
 **print** method[0], **' '**, method[1]()

get\_age 22

get\_name aaa

## 设计模式

### 单例模式

#### 装饰器实现

**def singleton**(*cls*)**:** instances **=** {}  
  
 **def wrapper**(*\*args*, *\*\*kwargs*)**:  
 if not** instances.has\_key(cls)**:** instances[cls] **=** cls(**\****args*, **\*\****kwargs*)  
 **return** instances[cls]  
  
 **return** wrapper  
  
**@singleton  
class MyClass**(object)**:  
 def \_\_init\_\_**(self, *n*)**:** *#装饰器作用在init之前，执行装饰器的wrapper方法*

self.n **=** *n* **print** 'n:',*n*p **=** MyClass(10)  
  
p2 **=** MyClass(11)

## 多线程

### threading.Thread 模块

start

getName()

setName()

isDaemon()

setDaemon()

join(timeout)

run()

join等待直到线程结束

## 队列

### task\_done():

指示前面get出来队列元素已经处理完成，

### join():

等待队列为空再执行别的动作

# 内置库

## sys

### argv

argv[0] 脚本名称，是否显示全路径要看操作系统，根据验证，win7和centos6.5只显示文件名，不显示全路径，要显示全路径可以用os.path.realpath(argv[0])

argv[1..n] 脚本的参数，因为是通过标准输入，所以所传递的类型全都是str。

## os

## argparse

详细用法参考：<http://www.jianshu.com/p/fef2d215b91d>

命令行选项、参数和子命令的解析器

### 定义一个解析器

parser = argparse.ArgumentParser(description='Example with long option names')

### 创建定位参数(positional arguments)

定位参数的意思是,不用带-的参数

简单示例：

保存文件为job.py

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("echo")

args = parser.parse\_args()

print args.echo #args.echo 为一个变量，即命令行传递过来的第一个参数

如：python job.py foo

则args.echo 的值为foo

### 创建可选参数(optional arguments)

两种方式：

1. 用 – 来指定短参数， 如-h
2. 用 - - 来指定长参数，如 - - help

示例：

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("-v", "--verbosity", help="increase output verbosity")

args = parser.parse\_args()

if args.verbosity:

print "verbosity turned on"

调用例子：

python prog.py -v 1

如果不给-v指定参数，会报错，如果要像-h这样不用指定参数，就要用action功能

例子二：

import argparse

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

parser = argparse.ArgumentParser(prog="uop\_jobs")

parser.add\_argument("-v", "--verbose", help="increate output verbose.")

parser.add\_argument("-g", "--group\_type", help="select dba|other sla group.")

#parser.add\_argument('echo')

args = parser.parse\_args()

#print args.echo

print 'verbose:',args.verbose

if args.group\_type == 'dba':

print 'select dba'

elif args.group\_type == 'other':

print 'select other'

else:

raise Exception("select group error.")

命令行调用：

[root@mytest scripts]# python learn\_arg.py -g other

verbose: None

select other

action 的用法

如果option arguments不需要加参数，像-h这样，则可以用action来指定

示例：

import argparse

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

parser = argparse.ArgumentParser()

parser.add\_argument("-g", "--group", action="store\_true", help="this is group select")

args = parser.parse\_args()

print "select group:",args.group

调用：

[root@mytest scripts]# python learn\_arg2.py

select group: False

[root@mytest scripts]# python learn\_arg2.py -g

select group: True

[root@mytest scripts]#

以上例子用了action=”store\_true”，意思是如果加了-g 则args.group的值为True,否则为False

# 第三方库

## 分析工具

### profilehook

三个主要的装饰器

profile() 对函数进行性能分析

coverage() 计算函数被调用的次数

timecall() 简单统计函数处理时间

使用例子：

@profile  
**def** check\_fun():  
 **for** i **in** range(100000):  
 **pass**

## 异步队列

### Celery

#### 安装:

pip install –U celery flower

flower是一个专门监控celery性能的前端界面应用

#### 配置并运行

默认使用redis作为broker

示例：

tasks.py

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

from celery import Celery

app = Celery("tasks", backend="redis://localhost", broker='redis://localhost//')

@app.task()

def add(x, y):

return x + y

命令行下运行：celery -A tasks worker

计算：

>>> from task import add

>>> res = add.delay(1,2)

>>> res.get()