Python 总结

目录

前言

知乎：路人甲
微博：玩数据的路人甲
微信公众号：一个程序员的日常

在知乎分享已经有一年多了，之前一直有朋友说我的回答能整理成书籍了，一直偷懒没做，最近有空仔细整理了知乎上的回答和文章另外也添加了一些新的内容，完成了几本小小的电子书，这一本是有关于 Python 方面的。

还有另外几本包括我的一些数据分析方面的读书笔记、增长黑客的读书笔记、机器学习十大算法等等内容。将会在我的微信公众号：一个程序员的日常进行更新，同时也可以关注我的知乎账号：路人甲 及时关注我的最新分享用数据讲故事。

（一）如何学习 Python

学习 Python 大致可以分为以下几个阶段：

1. 刚上手的时候肯定是先过一遍 Python 最基本的知识，比如说：变量、数据结构、语法等，基础过的很快，基本上 1~2 周时间就能过完了，我当时是在这儿看的基础：Python 简介 | 菜鸟教程

2. 看完基础后，就是做一些小项目巩固基础，比方说：做一个终端计算器，如果实在找不到什么练手项目，可以在 Codecademy - learn to code, interactively, for free 上面进行练习。

3. 如果时间充裕的话可以买一本讲 Python 基础的书籍比如《Python 编程》，阅读这些书籍，在巩固一遍基础的同时你会发现自己诸多没有学习到的边边角角，这一步是对自己基础知识的补充。

4. Python 库是 Python 的精华所在，可以说 Python 库组成并且造就了 Python，Python 库是 Python 开发者的利器，所以学习 Python 库就显得尤为重要：The Python Standard Library ，Python 库很多，如果你没有时间全部看完，不妨学习一遍常用的 Python 库：Python 常用库整理 - 知乎专栏

5. Python 库是开发者利器，用这些库你可以做很多很多东西，最常见的网络爬虫、自然语言处理、图像识别等等，这些领域都有很强大的 Python 库做支持，所以当你学了 Python 库之后，一定要第一时间进行练习。如何寻找自己需要的 Python 库呢？推荐我之前的一个回答：如何找到适合需求的 Python 库?

6. 学习使用了这些 Python 库，此时的你应该是对 Python 十分满意，也十分激动能遇到这样的语言，就是这个时候不妨开始学习 Python 数据结构与算法，Python 设计模式，这是你进一步学习的一个重要步骤：faif/python-patterns

7. 当度过艰难的第六步，此时选择你要研究的方向，如果你想做后端开发，不妨研究研究 Django，再往后，就是你自己自由发挥了。

（二）一些 Python 免费课程推荐

以下课程都为免费课程

1. python 零基础相关
适用人群：Python 零基础的初学者、Web 开发程序员、运维人员、有志于从事互联网行业以及各领域应用 Python 的人群

- 疯狂的 Python：快速入门精讲
- 零基础入门学习 Python
- 玩转 Python 语言
- Python 语言程序设计
- 程序设计入门
- 可汗学院公开课：计算机科学
- python 入门到精通
- Python 交互式编程入门的课程主页
- Python 交互编程入门（第 2 部分）的课程主页

2. python web 方向
Python Django 快速 Web 应用开发入门

3. python 爬虫
Python 实战：一周学会爬取网页

4. python 数据分析方向
数据分析实战基础课程

（三）Python 爬虫需要哪些知识？

要学会使用 Python 爬取网页信息无外乎以下几点内容：

1、要会 Python

2、知道网页信息如何呈现

3、了解网页信息如何产生

4、学会如何提取网页信息

第一步 Python 是工具，所以你必须熟练掌握它，要掌握到什么程度呢？如果你只想写一写简单的爬虫，不要炫技不考虑爬虫效率，你只需要掌握：

- 数据类型和变量
- 字符串和编码
- 使用 list 和 tuple
- 条件判断、循环
- 使用 dict 和 set

你甚至不需要掌握函数、异步、多线程、多进程，当然如果想要提高自己小爬虫的爬虫效率，提高数据的精确性，那么记住最好的方式是去系统的学习一遍 Python，去哪儿学习？Python 教程

假设已经熟悉了最基础的 Python 知识，那么进入第二步：知道网页信息如何呈现？你首先要知道所需要抓取的数据是怎样的呈现的，就像是你要学做一幅画，在开始之前你要知道这幅画是用什么画出来的，铅笔还是水彩笔...可能种类是多样的，但是放到网页信息来说这儿只有两种呈现方式：

1、HTML （HTML 简介）

2、JSON （JSON 简介）

HTML 是用来描述网页的一种语言

JSON 是一种轻量级的数据交换格式

假设你现在知道了数据是由 HTML 和 JSON 呈现出来的，那么我们紧接着第三步：数据怎么来？数据当然是从服务器反馈给你的，为什么要反馈给你？因为你发出了请求。

“Hi~，服务器我要这个资源”
“正在传输中...”
“已经收到 HTML 或者 JSON 格式的数据”

这个请求是什么请求？要搞清楚这一点你需要了解一下 http 的基础知识，更加精确来说你需要去了解 GET 和 POST 是什么，区别是什么。也许你可以看看这个：浅谈 HTTP 中 Get 与 Post 的区别 - hyddd - 博客园

很高兴你使用的是 Python，那么你只需要去掌握好快速上手 - Requests 2.10.0 文档，requests 可以帮你模拟发出 GET 和 POST 请求，这真是太棒了。

饭菜已经备好，两菜一汤美味佳肴，下面就是好好享受了。现在我们已经拿到了数据，我们需要在这些错乱的数据中提取我们需要的数据，这时候我们有两个选择。

第一招：万能钥匙
Python 正则表达式指南 ，再大再乱的内容，哪怕是大海捞针，只要告诉我这个针的样子我都能从茫茫大海中捞出来，强大的正则表达式是你提取数据的不二之选。

第二招：笑里藏刀
Beautiful Soup 4.2.0 文档，或许我们有更好的选择，我们把原始数据和我们想要的数据的样子扔个这个 Beautifulsoup，然后让它帮我们去寻找，这也是一个不错的方案，但是论灵活性，第二招还是略逊于第一招。

第三招：双剑合璧
最厉害的招式莫过于结合第一招和第二招了，打破天下无敌手。

基础知识我都会，可是我还是写不了一个爬虫啊！
客观别急，这还没完。

以下这些项目，你拿来学习学习练练手。

一些教学项目你值得拥有：

- 03. 豆瓣电影 TOP250
- 04. 另一种抓取方式

还不够？这儿有很多：

- 知乎--你需要这些：Python3.x 爬虫学习资料整理
- 如何学习 Python 爬虫[入门篇]？ - 知乎专栏
- 知乎--Python 学习路径及练手项目合集

（四）Python 爬虫进阶

爬虫无非分为这几块：分析目标、下载页面、解析页面、存储内容，其中下载页面不提。

1. 分析目标

所谓分析就是首先你要知道你需要抓取的数据来自哪里？怎么来？普通的网站一个简单的 POST 或者 GET 请求，不加密不反爬，几行代码就能模拟出来，这是最基本的，进阶就是学会分析一些复杂的目标，比如说：淘宝、新浪微博登陆以及网易云的评论信息等等。

2. 解析页面

解析页面主要是选择什么库或者那些库结合能使解析速度更快，可能你一开始你通过种种地方了解到了 bs 库，于是你对这个库很痴迷，以后只要写爬虫，总是先写上：

```
import requests

from bs4 import BeautifulSoup
```

当然 bs 已经很优秀了，但是并不代表可以用正则表达式解析的页面还需要使用 bs，也不代表使用 lxml 能解决的还要动用 bs，所以这些解析库的速度是你在进阶时要考虑的问题。

3. 存储内容

刚开始学爬虫，一般爬取的结果只是打印出来，最后把在终端输出的结果复制粘贴保存就好了；后来发现麻烦会用上 xlwt/openpyxl/csv 的把存储内容写入表格，再后来使用数据库 sqlite/mysql/neo4j 只要调用了库都很简单，当然这是入门。

进阶要开始学习如何选择合适的数据库，或者存储方式。当爬取的内容过千万的时候，如何设计使存储速度更快，比如说当既有人物关系又有人物关系的时候，一定会用 neo4j 来存储关系，myslq 用来存储用户信息，这样分开是因为如果信息全部存入 neo4j，后期的存储速度经十分的慢。

当你每个步骤都能做到很优秀的时候，你应该考虑如何组合这四个步骤，使你的爬虫达到效率最高,也就是所谓的爬虫策略问题,爬虫策略学习不是一朝一夕的事情，建议多看看一些比较优秀的爬虫的设计方案，比如说 Scrapy。

除了爬取策略以外，还有几点也是必备的：

1. 代理策略以及多用户策略

代理是爬虫进阶阶段必备的技能，与入门阶段直接套用代理不同，在进阶阶段你需要考虑如何设计使用代理策略，什么时候换代理，代理的作用范围等等，多用户的抓取策略考虑的问题基本上与代理策略相同。

2. 增量式抓取以及数据刷新

比如说你抓取的是一个酒店网站关于酒店价格数据信息的，那么会有这些问题：酒店的房型的价格是每天变动的,酒店网站每天会新增一批酒店,那么如何进行存储、如何进行数据刷新都是应该考虑的问题。

3.验证码相关的一些问题

有很多人提到验证码，我个人认为验证码不是爬虫主要去解决的问题，验证码不多的情况考虑下载到本地自己输入验证码，在多的情况下考虑接入打码平台。

（五）Python 爬虫面试指南

前段时间快要毕业,而我又不想找自己的老本行 Java 开发了,所以面了很多 Python 爬虫岗位。因为我在南京上学，所以我一开始只是在南京投了简历，我一共面试了十几家企业，其中只有一家没有给我发 offer，其他企业都愿意给到 10K 的薪资，不要拿南京的薪资水平和北上深的薪资水平比较，结合面试常问的问题类型说一说我的心得体会。

第一点：Python

因为面试的是Python爬虫岗位,面试官大多数会考察面试者的基础的Python知识，包括但不限于：

- Python2.x 与 Python3.x 的区别
- Python 的装饰器

- Python 的异步
- Python 的一些常用内置库，比如多线程之类的

第二点：数据结构与算法
数据结构与算法是对面试者尤其是校招生面试的一个很重要的点，当然小公司不会太在意这些，从目前的招聘情况来看对面试者的数据结构与算法的重视程度与企业的好坏成正比，那些从不问你数据结构的你就要当心他们是否把你当码农用的，当然以上情况不绝对，最终解释权归面试官所有。

第三点：Python 爬虫

最重要也是最关键的一点当然是你的 Python 爬虫相关的知识与经验储备，这通常也是面试官考察的重点，包括但不限于：

- 你遇到过的反爬虫的策略有哪些？
- 你常用的反反爬虫的方案有哪些？
- 你用过多线程和异步吗？除此之外你还用过什么方法来提高爬虫效率？
- 有没有做过增量式抓取？
- 对 Python 爬虫框架是否有了解？

第四点：爬虫相关的项目经验
爬虫重在实践，除了理论知识之外，面试官也会十分注重爬虫相关的项目：

- 你做过哪些爬虫项目？如果有 Github 最好
- 你认为你做的最好的爬虫项目是哪个？其中解决了什么难题？有什么特别之处？

以上是我在面试过程中，会碰到的一些技术相关的问题的总结，当然面试中不光是技术这一点，但是对于做技术的，过了技术面基本上就是薪资问题了。

（六）推荐一些不错的 Python 博客

如果是 Python 基础的话，廖雪峰的博客教程会是一个不错的选择：

- Python3 教程
- Python 2.7 教程

当然很多刚接触 Python 的同学反应廖大大的教程中部分跳跃性太大，如果觉得跳跃性太大可以结合菜鸟教程一起看：

- Python3 教程 | 菜鸟教程
- Python 基础教程 | 菜鸟教程

如果你英文稍好的话推荐还是看官方文档：Python 3.6.0 documentation

如果不是为了学习 Python 基础的话，推荐几个其他的博客。

- 董老师的博客：小明明 s à domicile 《Python-Web 开发实战》的作者，
- 知乎某位工程师的博客：分类《Python》 ，具体是哪位大神我不太清楚。
- 依云大大的博客文章值得深读：依云's Blog
- 《从 Python 开始学编程》的作者博客：Python - 标签 - Vamei - 博客园 ，但是此博客的内容也是比较偏向基础知识的。
- pythonware 的创造者，Python 图像库（PIL）的创造者：effbot.org
- 我很喜欢的一位作者，Pyhub 创始人：Yusheng's Tech Blog
- xlzd 杂谈 文章不是很多，有兴趣可以多看看在知乎的他。
- twelfthing - 博客园
- Python | the5fire 的技术博客

（七）Python 如何进阶

很多人在学习编程之初都会碰到这种问题：**学会了基础的语法了，但是还是做不了项目，不知道如何下手。**

当初，我学习 C 的时候是这样、Java 的时候是这样、Python 的时候也是这样，其实不管什么语言、什么知识都是这样:**理论基础知识 - 能动手做项目是有一道鸿沟的**。

那么如何突破这条鸿沟？中间的桥梁是什么？

其实题主自己已经回答出来了：**照抄！**

所谓照抄前提是有样本。

首先找到一些简单易上手的项目，这些项目大多散落在 Python 实践相关的书籍中、Github 上，这些实战项目知乎上都有很多推荐。

1. 一些比较好的适合初学者动手的项目：

- Show-Me-the-Code/show-me-the-code
- aosabook/500lines

另外知乎上这个问题下的一些推荐的项目还是非常适合新手练习的，可以作为参考：Python 的练手项目有哪些值得推荐？

2. 大多数的 Python 书里面（除了纯理论书）都是有小项目的，而且书的一个优点是它会一步一步解释这样做的原因。

先照抄这些项目，实现这些小功能在电脑上能运行确认无误之后，回过头来看代码：

- 有没有你不理解的地方，不理解的地方标记去搜索引擎或者书中找解释。
- 学习作者设计这个项目的思路方法，并运用到接下来的项目，如果时间充裕，建议隔天再重新再不看书的情况下重新自己实现一遍这些小项目。

如果你是跟着实战的书敲代码的，很多时候项目都不会一遍运行成功，那么你就要根据各种报错去寻找原因，这也是一个学习的过程。

总结起来从 Python 入门跳出来的过程分为三步：**照抄、照抄之后的理解、重新自己实现。**

（八）Python 爬虫入门

想写这么一篇文章，但是知乎社区爬虫大神很多，光是整理他们的答案就够我这篇文章的内容了。对于我个人来说我更喜欢那种非常实用的教程，这种教程对于想直接上手爬虫做一些小东西的朋友来说是极好的。
用一个精彩的回答作为开头：如何入门 Python 爬虫？ - 谢科的回答
如果你想学习编程，但是找不到学习路径和资源，欢迎关注专栏：学习编程

第一：Python 爬虫学习系列教程

Python 版本：2.7

整体目录：

一、爬虫入门

- Python 爬虫入门一之综述
- Python 爬虫入门二之爬虫基础了解
- Python 爬虫入门三之 Urllib 库的基本使用
- Python 爬虫入门四之 Urllib 库的高级用法
- Python 爬虫入门五之 URLError 异常处理
- Python 爬虫入门六之 Cookie 的使用
- Python 爬虫入门七之正则表达式

二、爬虫实战

- Python 爬虫实战一之爬取糗事百科段子
- Python 爬虫实战二之爬取百度贴吧帖子
- Python 爬虫实战三之实现山东大学无线网络掉线自动重连
- Python 爬虫实战四之抓取淘宝 MM 照片
- Python 爬虫实战五之模拟登录淘宝并获取所有订单
- Python 爬虫实战六之抓取爱问知识人问题并保存至数据库
- Python 爬虫实战七之计算大学本学期绩点
- Python 爬虫实战八之利用 Selenium 抓取淘宝匿名旺旺

三、爬虫利器

- Python 爬虫利器一之 Requests 库的用法
- Python 爬虫利器二之 Beautiful Soup 的用法
- Python 爬虫利器三之 Xpath 语法与 lxml 库的用法
- Python 爬虫利器四之 PhantomJS 的用法
- Python 爬虫利器五之 Selenium 的用法
- Python 爬虫利器六之 PyQuery 的用法

四、爬虫进阶

- Python 爬虫进阶一之爬虫框架概述
- Python 爬虫进阶二之 PySpider 框架安装配置
- Python 爬虫进阶三之爬虫框架 Scrapy 安装配置
- Python 爬虫进阶四之 PySpider 的用法

第二（第一的姊妹篇）：Python 爬虫入门教程

Python 版本：2.7

教程目录：

- [Python]网络爬虫（一）：抓取网页的含义和 URL 基本构成
- [Python]网络爬虫（二）：利用 urllib2 通过指定的 URL 抓取网页内容

- [Python]网络爬虫（三）：异常的处理和 HTTP 状态码的分类
- [Python]网络爬虫（四）：Opener 与 Handler 的介绍和实例应用
- [Python]网络爬虫（五）：urllib2 的使用细节与抓站技巧
- [Python]网络爬虫（六）：一个简单的百度贴吧的小爬虫
- [Python]网络爬虫（七）：Python 中的正则表达式教程
- [Python]网络爬虫（八）：糗事百科的网络爬虫（v0.3）源码及解析(简化更新)
- [Python]网络爬虫（九）：百度贴吧的网络爬虫（v0.4）源码及解析
- [Python]网络爬虫（十）：一个爬虫的诞生全过程（以山东大学绩点运算为例）
- [Python]网络爬虫（11）：亮剑！爬虫框架小抓抓 Scrapy 闪亮登场！
- [Python]网络爬虫（12）：爬虫框架 Scrapy 的第一个爬虫示例入门教程

第三：你已经看完上面（第一或者第二）的教程：再推荐知乎用户@陈唯源　的实战练习博客

- Python 爬虫学习记录（1）——Xiami 全站播放数
- Python 爬虫学习记录（2）——LDA 处理歌词
- 百度音乐带标签，作曲，演唱者，类别的歌词数据
- Python 爬虫学习记录（4）——传说中的足彩倍投法。。好像也不是那么靠谱
- 2011~2013.5 全球所有足球比赛比分数据以及足彩各公司盘口
- Python 爬虫学习记录（3）——用 Python 获取虾米加心歌曲，并获取 MP3 下载地址
- Python 爬虫学习记录（5）——python mongodb + 爬虫 + web.py 的 acfun 视频排行榜
- Python 爬虫学习记录（0）——Python 爬虫抓站 记录（虾米，百度，豆瓣，新浪微博）

第四：最后推荐知乎用户@gaga salamer　的实战练习博客

- 爬虫教程（1）基础入门
- 爬虫教程（2）性能进阶
- 知乎用户信息爬虫（规模化爬取）
- 用 scrapy 爬取豆瓣电影新片榜
- 用 scrapy 对豆瓣 top250 页面爬取（多页面爬取）
- 用 scrapy 自动爬取下载图片
- 用 scrapy 自动下载石原 sama 的豆瓣影人图集(727 张图片，自动下载)

希望以上的教程可以帮助到大家。

（九）Python 开发微信公众号

我的第一个 Python 项目就是做的微信公众号机器人，按照当时我的思路来讲讲如何学习使用 Python 来开发微信公众号：**大家伙收藏顺手点个赞呗。**

微信公众号功能开发分为两大块：需要调用微信内部功能、不需要调用微信内部功能，重点在调用微信内部功能组建。

1、需要调用微信内部功能

需要调用微信内部功能组件的比如：公众号收发消息|图片、页面分享至朋友圈、用户授权提取用户基本信息、微信小店、微信公众号菜单等内部功能组件，这些功能组件在微信公众号开发者文档里面找到：微信公众平台开发者文档

对于这些功能组件，开发者文档都提供了详细的接口文档，告诉你如何调用，而你只需要用 Python 调用这些接口即可。比如一个很简单的消息发送的方法如下：

当然在这所有的调用之前，需要进行一些授权验证，同样开发者文档有一套完整的接入指南：接入指南 - 微信公众平台开发者文档

很遗憾很多初学者在一开始看这份文档的时候并不能看懂，所以这里也推荐一些我以前学习摸索的过程中使用到的一些简单易学的教程。

你可以先做一个简单的微信机器人练练手（零基础十分容易上手）：

- 使用 python 一步一步搭建微信公众平台（一）
- 使用 python 一步一步搭建微信公众平台（二）----搭建一个中英互译的翻译工具
- 使用 python 一步一步搭建微信公众平台（三）----添加用户关注后的欢迎信息与听音乐功能
- 使用 python 一步一步搭建微信公众平台（四）----将小黄鸡引入微信自动回复
- 使用 python 一步一步搭建微信公众平台（五）----使用 mysql 服务来记录用户的反馈

如果你已经能按照以上的教程搭建一个完整的微信机器人了，基本上对于微信收发消息等等简单功能已经没有什么障碍了。下面再继续学习如下教程，开始学习如何调用其他一些相对来说比较复杂的接口。

- 微信公众平台开发入门教程
- 微信公众平台开发—天气预报
- 微信公众平台开发—小黄鸡
- 微信公众平台开发—人脸识别
- 微信公众平台开发—百度地图
- 微信公众平台开发—笑话
- 微信公众平台开发—在线点歌
- 微信公众平台开发—附近查询
- 微信公众平台开发—快递物流
- 微信公众平台开发—一键关注

当你实验了如上的教程之后，相信官方文档的所有接口调用对你来说已经小菜一碟了。

2、不需要调用微信内部功能

不需要调用微信内部功能组件的，就如同正常的 web 页面一样，比如填写表单进行进行注册、点击按钮进行跳转等等，这些都是正常的 web 请求，按照正常的 web 开发方法走即可。

（十）Python 面试概念和代码

```
'''
                    /题目目录/

        (一)、这两个参数是什么意思: *args, **kwargs?
        (二)、谈一谈Python的装饰器(decorator)
        (三)、简要描述Python的垃圾回收机制(garbage collection)
        (四):Python多线程(multi-threading)。这是个好主意吗?
        (五)、 说明os,sys模块不同, 并列举常用的模块方法?
        (六)、什么是lambda表达式?它有什么好处?
        (七)、Python中pass语句的作用是什么?
        (八)、Python是如何进行类型转换的?
        (九)、Python里面如何拷贝一个对象?
        (十)、__new__和__init__的区别。
        (十一)、Python中单下划线和双下划线分别是什么?
        (十二)、说一说Python自省。
        ......
'''
```

（一）、这两个参数是什么意思：*args，**kwargs？我们为什么要使用它们？

答：如果我们不确定往一个函数中传入多少参数，或者我们希望以元组（tuple）或者列表（list）的形式传参数的时候，我们可以使用*args（单星号）。如果我们不知道往函数中传递多少个关键词参数或者想传入字典的值作为关键词参数的时候我们可以使用**kwargs（双星号），args、kwargs 两个标识符是约定俗成的用法。

另一种答法：当函数的参数前面有一个星号*号的时候表示这是一个可变的位置参数，两个星号**表示这个是一个可变的关键词参数。星号*把序列或者集合解包(unpack)成位置参数，两个星号**把字典解包成关键词参数。

```
tempList = [1,2,3]
tempTuple = (2,3,4)
tempDict = {'s':3,'m':4,'c':5}

def testFunc(*args,**kwargs):
    print args,kwargs

testFunc() #() {}
testFunc(*tempList) #(1, 2, 3) {}
testFunc(*tempTuple) #(2,3,4) {}
testFunc(*tempDict) #('s','m','c'),{}
testFunc(**tempDict) #(),{'s':3,'m':4,'c':5}
testFunc(*tempList,**tempDict) #(1,2,3) {'s':3,'m':4,'c':5}
testFunc(0) #(0,) {}
testFunc(0,*tempList) #(0,1,2,3) {}
testFunc(0,**tempDict) #(0,) {'s': 3, 'm': 4, 'c': 5}
testFunc(0,*tempList,tempName = 'bye',**tempDict)
#(0, 1, 1, 2, 3) {'s': 3, 'm': 4, 'c': 5, 'tempName': 'bye'}
```

（二）、谈一谈 Python 的装饰器（decorator）

装饰器本质上是一个 Python 函数，它可以让其它函数在不作任何变动的情况下增加额外功能，装饰器的返回值也是一个函数对象。它经常用于有切面需求的场景。比如：插入日志、性能测试、事务处理、缓存、权限校验等。有了装饰器我们就可以抽离出大量的与函数功能无关的雷同代码进行重用。

有关于具体的装饰器的用法看这里：装饰器 - 廖雪峰的官方网站

```
import functools
def log(text):
    if isinstance(text, basestring):
        def decorator(func):
            functools.wraps(func)
            def wrapper(*args,**kwargs):
                func(*args,**kwargs)
                print '%s %s'%(text,func.__name__)
            print '%s %s'%('start',func.__name__)
            return wrapper
        return decorator
    else:
        functools.wraps(func)
        def wrapper(*args,**kwargs):
            print 'Call %s:'%(func.__name__)
            return func(*args,**kwargs)
        return wrapper

@log('end')
def now():
    print '2016-11-10'
now()
```

（三）、简要描述 Python 的垃圾回收机制（garbage collection）

Python 中的垃圾回收是以引用计数为主，标记-清除和分代收集为辅。

引用计数：Python 在内存中存储每个对象的引用计数，如果计数变成 0，该对象就会消失，分配给该对象的内存就会释放出来。

标记-清除：一些容器对象，比如 list、dict、tuple，instance 等可能会出现引用循环，对于这些循环，垃圾回收器会定时回收这些循环（对象之间通过引用（指针）

连在一起，构成一个有向图，对象构成这个有向图的节点，而引用关系构成这个有向图的边）。

分代收集：Python 把内存根据对象存活时间划分为三代，对象创建之后，垃圾回收器会分配它们所属的代。每个对象都会被分配一个代，而被分配更年轻的代是被优先处理的，因此越晚创建的对象越容易被回收。

如果你想要深入了解 Python 的 GC 机制，点击这里：[转载]Python 垃圾回收机制--完美讲解!

（四）、Python 多线程（multi-threading）。这是个好主意吗？

Python 并不支持真正意义上的多线程，Python 提供了多线程包。Python 中有一个叫 Global Interpreter Lock（GIL）的东西，它能确保你的代码中永远只有一个线程在执行。经过 GIL 的处理，会增加执行的开销。这就意味着如果你先要提高代码执行效率，使用 threading 不是一个明智的选择，当然如果你的代码是 IO 密集型，多线程可以明显提高效率，相反如果你的代码是 CPU 密集型的这种情况下多线程大部分是鸡肋。

想要深入详细了解多线程，点击这里：详解 Python 中的多线程编程_python

想了解一下 IO 密集和 CPU 密集可以点击这里：CPU-bound(计算密集型) 和 I/O bound(I/O 密集型)

（五）、 说明 os, sys 模块不同，并列举常用的模块方法？

官方文档：

os 模板提供了一种方便的使用操作系统函数的方法

sys 模板可供访问由解释器使用或维护的变量和与解释器交互的函数

另一种回答：

os 模块负责程序与操作系统的交互，提供了访问操作系统底层的接口。sys 模块负责程序与 Python 解释器的交互，提供了一系列的函数和变量用户操作 Python 运行时的环境。一些常用的方法：

```
'''
    os.remove()             删除文件
    os.rename()             重命名文件
    os.walk()               生成目录树下的所有文件名
    os.chdir()              改变目录
    os.mkdir/makedirs       创建目录/多层目录
    os.rmdir/removedirs     删除目录/多层目录
    os.listdir()            列出指定目录的文件
    os.getcwd()             取得当前工作目录
    os.chmod()              改变目录权限
    os.path.basename()      去掉目录路径，返回文件名
    os.path.dirname()       去掉文件名，返回目录路径
    os.path.join()          将分离的各部分组合成一个路径名
    os.path.split()         返回（dirname(),basename())元组
    os.path.splitext()      (返回filename,extension)元组
    os.path.getatime\ctime\mtime分别返回最近访问、创建、修改时间
    os.path.getsize()       返回文件大小
    os.path.exists()        是否存在
    os.path.isabs()         是否为绝对路径
    os.path.isdir()         是否为目录
    os.path.isfile()        是否为文件
'''

'''
    sys.argv            命令行参数List，第一个元素是程序本身路径
    sys.modules.keys()  返回所有已经导入的模块列表
    sys.exc_info()      获取当前正在处理的异常类,exc_type、exc_value、exc_traceback当前处理的异常详细信息
    sys.exit(n)         退出程序，正常退出时exit(0)
    sys.hexversion      获取Python解释程序的版本值，16进制格式如：0x020403F0
    sys.version         获取Python解释程序的版本信息
    sys.maxint          最大的Int值
    sys.maxunicode      最大的Unicode值
    sys.modules         返回系统导入的模块字段，key是模块名，value是模块
    sys.path            返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值
    sys.platform        返回操作系统平台名称
    sys.stdout          标准输出
    sys.stdin           标准输入
    sys.stderr          错误输出
    sys.exc_clear()     用来清除当前线程所出现的当前的或最近的错误信息
    sys.exec_prefix     返回平台独立的python文件安装的位置
    sys.byteorder       本地字节规则的指示器，big-endian平台的值是'big',little-endian平台的值是'little'
    sys.copyright       记录python版权相关的东西
    sys.api_version     解释器的C的API版本
    sys.version_info
'''
```

一些常用的用法示例：

```
import os
filePath = '/root/Desktop/temp/'
if os.path.isdir(filePath):
    if os.path.exists(os.path.join(filePath,'tempTest.txt')):
        print 'tempTest.txt Exists.'
    else:
        print 'tempTest.txt Not Exists.'
    for everyChild in os.listdir(filePath):
        everyFilePath = os.path.join(filePath,everyChild)
        if os.path.isfile(everyFilePath):
            print os.path.basename(everyFilePath),
            print os.path.getsize(everyFilePath),
            print os.path.getatime(everyFilePath)
        elif os.path.isdir(everyFilePath):
            print '%s is a DIR'%everyChild
else:
    print 'Not dir.'
```

想要了解更详细的使用请访问：os 和 sys 模块 - 君醉

（六）、什么是 lambda 表达式？它有什么好处？
简单来说，lambda 表达式通常是当你需要使用一个函数，但是又不想费脑袋去命名一个函数的时候使用，也就是通常所说的匿名函数。
lambda 表达式一般的形式是：关键词 lambda 后面紧接一个或多个参数，紧接一个冒号“：”，紧接一个表达式。lambda 表达式是一个表达式不是一个语句。

```
f = lambda x,y,z : z + y + z
print f(4,2,6)

L = {'f1':(lambda x,y:x**2+y**2),
     'f2':(lambda x,y:x**3+y**3),
     'f3':(lambda x,y:x**4+y**3)}
print L['f2'](3,2)
```

想更加详细的了解 Python 中的 Lamdba 表达式可以点击这里：Lambda 表达式有何用处？如何使用？ - Python

（七）、Python 中 pass 语句的作用是什么？
pass 语句不会执行任何操作，一般作为占位符或者创建占位程序
（八）、Python 是如何进行类型转换的？
Python 提供了将变量或值从一种类型转换为另一种类型的内置方法。

```
119  int(x [,base])          将x转换为一个整数
120  long(x [,base])         将x转换为一个长整数
121  float(x)                将x转换到一个浮点数
122  complex(real [,imag])   创建一个复数
123  str(x)                  将对象 x 转换为字符串
124  repr(x)                 将对象 x 转换为表达式字符串
125  eval(str)               用来计算在字符串中的有效Python表达式,并返回一个对象
126  tuple(s)                将序列 s 转换为一个元组
127  list(s)                 将序列 s 转换为一个列表
128  chr(x)                  将一个整数转换为一个字符
129  unichr(x)               将一个整数转换为Unicode字符
130  ord(x)                  将一个字符转换为它的整数值
131  hex(x)                  将一个整数转换为一个十六进制字符串
132  oct(x)                  将一个整数转换为一个八进制字符串
133  '''
134  print int(0),int('1')
135  print long(0),long('123456')
136  print float(0),float('12')
137  print str(0),str('12'),str([1,2]),str({'1':1,'2':2})
138  print list((1,2,3,4,5))
```

```
0 1
0 123456
0.0 12.0
0 12 [1, 2] {'1': 1, '2': 2}
[1, 2, 3, 4, 5]
```

（九）、Python 里面如何拷贝一个对象？

Python 中对象之间的赋值是按引用传递的，如果要拷贝对象需要使用标准模板中的 copy

copy.copy：浅拷贝，只拷贝父对象，不拷贝父对象的子对象。

copy.deepcopy：深拷贝，拷贝父对象和子对象。

```
142  import copy
143  tempList = [0,1,2,[3,4]]
144  testList = tempList
145  testCopyList = copy.copy(tempList)
146  testDeepCopyList = copy.deepcopy(testList)
147
148  tempList.append('sign')
149  print testList,testCopyList,testDeepCopyList
150
151  testList[3].append('sign')
152  print testList,testCopyList,testDeepCopyList
```

```
[0, 1, 2, [3, 4], 'sign'] [0, 1, 2, [3, 4]] [0, 1, 2, [3, 4]]
[0, 1, 2, [3, 4, 'sign'], 'sign'] [0, 1, 2, [3, 4, 'sign']] [0, 1, 2, [3, 4]]
[Finished in 0.1s]
```

（十）、__new__和__init__的区别。

__init__为初始化方法，__new__方法是真正的构造函数。

__new__是实例创建之前被调用，它的任务是创建并返回该实例，是静态方法

__init__是实例创建之后被调用的，然后设置对象属性的一些初始值。

总结：__new__方法在__init__方法之前被调用，并且__new__方法的返回值将传递给__init__方法作为第一个参数，最后__init__给这个实例设置一些参数。

```
155 class TestClass(object):
156
157     def __new__(cls,*args,**kwargs):
158         print '__new__'
159         return object.__new__(cls,*args,**kwargs)
160
161     def __init__(self,testName):
162         print '__init__'
163         self.testName = testName
164
165 print TestClass('Json').testName
166
```

```
__new__
__init__
Json
[Finished in 0.1s]
```

想要更加详细的了解这两个方法，请点击：Python 中的__new__及其用法

（十一）、Python 中单下划线和双下划线分别是什么？

__name__：一种约定，Python 内部的名字，用来与用户自定义的名字区分开，防止冲突

_name：一种约定，用来指定变量私有

__name：解释器用_classname__name 来代替这个名字用以区别和其他类相同的命名

想要更加详细的了解这两者的区别，请点击：Python 中的下划线（译文）

（十二）、说一说 Python 自省。

自省就是面向对象的语言所写的程序在运行时，所能知道对象的类型。简单一句话就是运行时能够获得对象的类型。比如：type()、dir()、getattr()、hasattr()、isinstance()

```
167 import string
168 print dir(string)
169 print getattr(string, 'strip')
170 print callable(getattr(string,'strip')),callable(getattr(string,'__doc__'))
171
```

```
['Formatter', 'Template', '_TemplateMetaclass', '__builtins__', '__doc__', '__file__', '__name__', '__package__',
'_float', '_idmap', '_idmapL', '_int', '_long', '_multimap', '_re', 'ascii_letters', 'ascii_lowercase', 'ascii_uppercase',
'atof', 'atof_error', 'atoi', 'atoi_error', 'atol', 'atol_error', 'capitalize', 'capwords', 'center', 'count', 'digits',
'expandtabs', 'find', 'hexdigits', 'index', 'index_error', 'join', 'joinfields', 'letters', 'ljust', 'lower', 'lowercase',
'lstrip', 'maketrans', 'octdigits', 'printable', 'punctuation', 'replace', 'rfind', 'rindex', 'rjust', 'rsplit', 'rstrip',
'split', 'splitfields', 'strip', 'swapcase', 'translate', 'upper', 'uppercase', 'whitespace', 'zfill']
<function strip at 0x7fdd29c35cf8>
True False
[Finished in 0.1s]
```

想要完整的理解 Python 自省，请点击：Python 自省（反射）指南

有关于元类以及单例模式会在后面文章中做详细的解释说明。

本文参考文献资料：

- 七、PYTHON 一些基础面试题目总结
- 很全的 Python 面试题

- Python 自省（反射）指南
- Python 学习笔记（十二）：lambda 表达式与函数式编程
- Python 面试必须要看的 15 个问题

（十一）Python 书籍

入门书籍

- 《Python 学习手册》
- 《Head First Python》
- 《Learn Python The Hard Way》
- 《Python 编程：入门到实践》
- 《笨办法学 Python》
- 《简明 Python 教程》

进阶书籍

- 《Python Cookbook（第三版）》
- 《流畅的 Python》
- 《Python 源码剖析》
- 《Python 进阶》
- 《Flask Web 开发:基于 Python 的 Web 应用开发实战》