

# 目录：

liupeng678 / Python-100-Days  
forked from jackfrued/Python-100-Days

Python - 100天从新手到大师

Edit

Manage topics

179 commits

1 branch

0 releases

7 contributors

Branch: master ▾

New pull request

Create new file

Upload files

Find File

Clone or download ▾

This branch is even with jackfrued:master.

Pull request Compare



jackfrued Merge pull request jackfrued#55 from wjsvec/patch-1 ...

Latest commit bd39f99 May 8, 2019

Day01-15	Merge pull request jackfrued#55 from wjsvec/patch-1	May 8, 2019
Day16-20	修正了缩进的问题	May 3, 2019
Day21-30	更新了部分代码和文档	Apr 23, 2019
Day31-35	更新了部分代码和文档	Apr 23, 2019
Day36-40	更新了README.md	May 3, 2019
Day41-55	更新了Django部分的文档	May 6, 2019
Day56-60	调整了部分目录结构	Apr 26, 2019
Day61-65	添加了Tornado相关的文档和代码	May 3, 2019
Day66-75	Merge pull request jackfrued#27 from royaso/patch-2	May 4, 2019
Day76-90	更新了文档和目录结构	Feb 12, 2019
Day91-100	更新了部分代码和文档	Apr 23, 2019
res	更新了Django部分的文档	May 2, 2019
.gitignore	修改了.gitignore文件	May 27, 2018
PEP 8风格指南.md	修改了部分文档	Aug 11, 2018
Python参考书籍.md	更新了部分文档	May 24, 2018
Python惯例.md	修改了部分文档	Aug 11, 2018
README.md	更新了第15天的文档	May 4, 2019
玩转PyCharm.md	更新了Django部分的文档	May 2, 2019
用函数还是用复杂的表达式.md	update Day06	Mar 5, 2018
知乎问题回答.md	更新了测试和知乎问答	Mar 24, 2019
那些年我们踩过的那些坑.md	调整了部分目录结构	Apr 26, 2019

README.md



## Python - 100天从新手到大师

作者：骆昊

### Python应用领域和就业形势分析

简单的说，Python是一个“优雅”、“明确”、“简单”的编程语言。

- 学习曲线低，非专业人士也能上手
- 开源系统，拥有强大的生态圈
- 解释型语言，完美的平台可移植性
- 支持面向对象和函数式编程
- 能够通过调用C/C++代码扩展功能

- 代码规范程度高，可读性强

目前几个比较流行的领域，Python都有用武之地。

- 云基础设施 - Python / Java / Go
- DevOps - Python / Shell / Ruby / Go
- 网络爬虫 - Python / PHP / C++
- 数据分析挖掘 - Python / R / Scala / Matlab
- 机器学习 - Python / R / Java / Lisp

作为一名Python开发者，主要的就业领域包括：

- Python服务器后台开发 / 游戏服务器开发 / 数据接口开发工程师
- Python自动化运维工程师
- Python数据分析 / 数据可视化 / 大数据工程师
- Python爬虫工程师
- Python聊天机器人开发 / 图像识别和视觉算法 / 深度学习工程师

下图显示了主要城市Python招聘需求量及薪资待遇排行榜（截止到2018年5月）。

#### 招聘需求量地区排行 Top 10

1 北京	24600个职位
2 上海	14764个职位
3 深圳	10158个职位
4 杭州	6203个职位
5 广州	4934个职位
6 成都	2905个职位
7 南京	1932个职位
8 武汉	1850个职位
9 厦门	1116个职位
10 苏州	1069个职位

#### 招聘薪酬待遇地区排行 Top 10

1 北京	¥19490
2 上海	¥16570
3 深圳	¥15020
4 杭州	¥14330
5 苏州	¥11910
6 广州	¥11210
7 南京	¥11080
8 成都	¥10500
9 长沙	¥10400
10 西安	¥9870

## 北京python • 工资收入水平

北京python平均工资：¥ 19490/月，取自 8233 份样本



# 成都python • 工资收入水平

成都python平均工资：¥ 10500/月，取自 1089 份样本



给初学者的几个建议：

- Make English as your working language.
- Practice makes perfect.
- All experience comes from mistakes.
- Don't be one of the leeches.
- Either stand out or kicked out.

## Day01~15 - Python语言基础

### Day01 - 初识Python

- Python简介 - Python的历史 / Python的优缺点 / Python的应用领域
- 搭建编程环境 - Windows环境 / Linux环境 / MacOS环境
- 从终端运行Python程序 - DOS命令 / Hello, world / print函数 / 运行程序
- 使用IDLE - 交互式环境(REPL) / 编写多行代码 / 运行程序 / 退出IDLE
- 注释 - 注释的作用 / 单行注释 / 多行注释

### Day02 - 语言元素

- 程序和进制 - 指令和程序 / 冯诺依曼机 / 二进制和十进制 / 八进制和十六进制
- 变量和类型 - 变量的命名 / 变量的使用 / input函数 / 检查变量类型 / 类型转换
- 数字和字符串 - 整数 / 浮点数 / 复数 / 字符串 / 字符串基本操作 / 字符编码
- 运算符 - 数学运算符 / 赋值运算符 / 比较运算符 / 逻辑运算符 / 身份运算符 / 运算符的优先级
- 应用案例 - 华氏温度转换成摄氏温度 / 输入圆的半径计算周长和面积 / 输入年份判断是否是闰年

### Day03 - 分支结构

- 分支结构的应用场景 - 条件 / 缩进 / 代码块 / 流程图
- if语句 - 简单的if / if-else结构 / if-elif-else结构 / 嵌套的if
- 应用案例 - 用户身份验证 / 英制单位与公制单位互换 / 掷骰子决定做什么 / 百分制成绩转等级制 / 分段函数求值 / 输入三条边的长度如果能构成三角形就计算周长和面积

### Day04 - 循环结构

- 循环结构的应用场景 - 条件 / 缩进 / 代码块 / 流程图
- while循环 - 基本结构 / break语句 / continue语句
- for循环 - 基本结构 / range类型 / 循环中的分支结构 / 嵌套的循环 / 提前结束程序

- 应用案例 - 1~100求和 / 判断素数 / 猜数字游戏 / 打印九九表 / 打印三角形图案 / 猴子吃桃 / 百钱百鸡

#### Day05 - 总结和练习

- 基础练习 - 水仙花数 / 完美数 / 五人分鱼 / Fibonacci数列 / 回文素数
- 综合练习 - Craps赌博游戏

#### Day06 - 函数和模块的使用

- 函数的作用 - 代码的坏味道 / 用函数封装功能模块
- 定义函数 - def语句 / 函数名 / 参数列表 / return语句 / 调用自定义函数
- 调用函数 - Python内置函数 / 导入模块和函数
- 函数的参数 - 默认参数 / 可变参数 / 关键字参数 / 命名关键字参数
- 函数的返回值 - 没有返回值 / 返回单个值 / 返回多个值
- 作用域问题 - 局部作用域 / 嵌套作用域 / 全局作用域 / 内置作用域 / 和作用域相关的关键字
- 用模块管理函数 - 模块的概念 / 用自定义模块管理函数 / 命名冲突的时候会怎样 ( 同一个模块和不同的模块 )

#### Day07 - 字符串和常用数据结构

- 字符串的使用 - 计算长度 / 下标运算 / 切片 / 常用方法
- 列表基本用法 - 定义列表 / 用下表访问元素 / 下标越界 / 添加元素 / 删除元素 / 修改元素 / 切片 / 循环遍历
- 列表常用操作 - 连接 / 复制(复制元素和复制数组) / 长度 / 排序 / 倒转 / 查找
- 生成列表 - 使用range创建数字列表 / 生成表达式 / 生成器
- 元组的使用 - 定义元组 / 使用元组中的值 / 修改元组变量 / 元组和列表转换
- 集合基本用法 - 集合和列表的区别 / 创建集合 / 添加元素 / 删除元素 / 清空
- 集合常用操作 - 交集 / 并集 / 差集 / 对称差 / 子集 / 超集
- 字典的基本用法 - 字典的特点 / 创建字典 / 添加元素 / 删除元素 / 取值 / 清空
- 字典常用操作 - keys()方法 / values()方法 / items()方法 / setdefault()方法
- 基础练习 - 跑马灯效果 / 列表找最大元素 / 统计考试成绩的平均分 / Fibonacci数列 / 杨辉三角
- 综合案例 - 双色球选号 / 井字棋

#### Day08 - 面向对象编程基础

- 类和对象 - 什么是类 / 什么是对象 / 面向对象其他相关概念
- 定义类 - 基本结构 / 属性和方法 / 构造器 / 析构器 / \_\_str\_\_方法
- 使用对象 - 创建对象 / 给对象发消息
- 面向对象的四大支柱 - 抽象 / 封装 / 继承 / 多态
- 基础练习 - 定义学生类 / 定义时钟类 / 定义图形类 / 定义汽车类

#### Day09 - 面向对象进阶

- 属性 - 类属性 / 实例属性 / 属性访问器 / 属性修改器 / 属性删除器 / 使用\_\_slots\_\_
- 类中的方法 - 实例方法 / 类方法 / 静态方法
- 运算符重载 - \_\_add\_\_ / \_\_sub\_\_ / \_\_or\_\_ / \_\_getitem\_\_ / \_\_setitem\_\_ / \_\_len\_\_ / \_\_repr\_\_ / \_\_gt\_\_ / \_\_lt\_\_ / \_\_le\_\_ / \_\_ge\_\_ / \_\_eq\_\_ / \_\_ne\_\_ / \_\_contains\_\_
- 类(的对象)之间的关系 - 关联 / 继承 / 依赖
- 继承和多态 - 什么是继承 / 继承的语法 / 调用父类方法 / 方法重写 / 类型判定 / 多重继承 / 菱形继承(钻石继承)和C3算法
- 综合案例 - 工资结算系统 / 图书自动折扣系统 / 自定义分数类

#### Day10 - 图形用户界面和游戏开发

- 使用tkinter开发GUI
- 使用pygame三方库开发游戏应用
- “大球吃小球”游戏

#### Day11 - 文件和异常

- 读文件 - 读取整个文件 / 逐行读取 / 文件路径
- 写文件 - 覆盖写入 / 追加写入 / 文本文件 / 二进制文件
- 异常处理 - 异常机制的重要性 / try-except代码块 / else代码块 / finally代码块 / 内置异常类型 / 异常栈 / raise语句

- 数据持久化 - CSV文件概述 / csv模块的应用 / JSON数据格式 / json模块的应用
- 综合案例 - 歌词解析

#### Day12 - 字符串和正则表达式

- 字符串高级操作 - 转义字符 \ 原始字符串 \ 多行字符串 \ in 和 not in 运算符 \ is开头的方法 \ join 和 split 方法 \ strip 相关方法 \ pyperclip 模块 \ 不变字符串 和 可变字符串 \ StringIO 的使用
- 正则表达式入门 - 正则表达式的作用 \ 元字符 \ 转义 \ 量词 \ 分组 \ 零宽断言 \ 贪婪匹配与惰性匹配懒惰 \ 使用 re 模块实现正则表达式操作 ( 匹配、搜索、替换、捕获 )
- 使用正则表达式 - re 模块 \ compile 函数 \ group 和 groups 方法 \ match 方法 \ search 方法 \ findall 和 finditer 方法 \ sub 和 subn 方法 \ split 方法
- 应用案例 - 使用正则表达式验证输入的字符串

#### Day13 - 进程和线程

- 进程和线程的概念 - 什么是进程 / 什么是线程 / 多线程的应用场景
- 使用进程 - fork 函数 / multiprocessing 模块 / 进程池 / 进程间通信
- 使用线程 - thread 模块 / threading 模块 / Thread 类 / Lock 类 / Condition 类 / 线程池

#### Day14-A - 网络编程入门

- 计算机网络基础 - 计算机网络发展史 / "TCP-IP" 模型 / IP 地址 / 端口 / 协议 / 其他相关概念
- 网络应用架构 - "客户端-服务器" 架构 / "浏览器-服务器" 架构
- Python 网络编程 - 套接字的概念 / socket 模块 / socket 函数 / 创建 TCP 服务器 / 创建 TCP 客户端 / 创建 UDP 服务器 / 创建 UDP 客户端 / SocketServer 模块

#### Day14-B - 网络应用开发

- 访问网络 API - 网络 API 概述 / 访问 URL / requests 模块 / 解析 JSON 格式数据
- 文件传输 - FTP 协议 / ftplib 模块 / 交互式 FTP 应用
- 电子邮件 - SMTP 协议 / POP3 协议 / IMAP 协议 / smtplib 模块 / poplib 模块 / imaplib 模块
- 短信服务 - twilio 模块 / 国内的短信服务

#### Day15 - 图像和文档处理

- 用 Pillow 处理图片 - 图片读写 / 图片合成 / 几何变换 / 色彩转换 / 滤镜效果
- 读写 Word 文档 - 文本内容的处理 / 段落 / 页眉和页脚 / 样式的处理
- 读写 Excel 文件 - xlrd 模块 / xlwt 模块
- 生成 PDF 文件 - pypdf2 模块 / reportlab 模块

#### Day16~Day20 - Python 语言进阶

- 常用数据结构
- 函数的高级用法 - "一等公民" / 高阶函数 / Lambda 函数 / 作用域和闭包 / 装饰器
- 面向对象高级知识 - "三大支柱" / 类与类之间的关系 / 垃圾回收 / 魔术属性和方法 / 混入 / 元类 / 面向对象设计原则 / GoF 设计模式
- 迭代器和生成器 - 相关魔术方法 / 创建生成器的两种方式 /
- 并发和异步编程 - 多线程 / 多进程 / 异步 IO / async 和 await

#### Day21~30 - Web 前端入门

- 用 HTML 标签承载页面内容
- 用 CSS 渲染页面
- 用 JavaScript 处理交互式行为
- jQuery 入门和提高
- Vue.js 入门
- Element 的使用
- Bootstrap 的使用

#### Day31~35 - 玩转 Linux 操作系统

- 操作系统发展史和Linux概述
- Linux基础命令
- Linux中的实用程序
- Linux的文件系统
- Vim编辑器的应用
- 环境变量和Shell编程
- 软件的安装和服务的配置
- 网络访问和管理
- 其他相关内容

## Day36~40 - 数据库基础和进阶

- **关系型数据库MySQL**
  - 关系型数据库概述
  - MySQL的安装和使用
  - SQL的使用
    - DDL - 数据定义语言 - create / drop / alter
    - DML - 数据操作语言 - insert / delete / update / select
    - DCL - 数据控制语言 - grant / revoke
  - 相关知识
    - 范式理论 - 设计二维表的指导思想
    - 数据完整性
    - 数据一致性
  - 在Python中操作MySQL
- **NoSQL入门**
  - NoSQL概述
  - Redis概述
  - Mongo概述

## Day41~55 - 实战Django

### Day41 - 快速上手

- Web应用工作原理和HTTP协议
- Django框架概述
- 5分钟快速上手
- 使用视图模板

### Day42 - 深入模型

- 关系型数据库配置
- 管理后台的使用
- 使用ORM完成对模型的CRUD操作
- Django模型最佳实践
- 模型定义参考

### Day43 - 静态资源和Ajax请求

- 加载静态资源
- 用Ajax请求获取数据

### Day44 - 表单的应用

### Day45 - Cookie和Session

### Day46 - 中间件的应用

### Day47 - 日志和缓存

Day48 - [文件上传和富文本编辑](#)

Day49 - [文件下载和报表](#)

Day50 - [RESTful架构和DRF入门](#)

Day51 - [RESTful架构和DRF进阶](#)

Day52 - [使用缓存](#)

Day53 - [短信和邮件](#)

Day54 - [异步任务和定时任务](#)

Day55 - [单元测试和项目上线](#)

- 项目开发流程和相关工具
- 生成非HTML内容
- 项目部署和测试
- 项目性能初步调优
- Web应用安全保护

Day56~60 - [实战Flask](#)

Day56 - [Flask入门](#)

Day57 - [模板的使用](#)

Day58 - [表单的处理](#)

Day59 - [数据库操作](#)

Day60 - [项目实战](#)

Day61~65 - [实战Tornado](#)

Day61 - [预备知识](#)

- 并发编程
- I/O模式和事件驱动

Day62 - [Tornado入门](#)

- Tornado概述
- 5分钟上手Tornado
- 路由解析
- 请求处理器

Day63 - [异步化](#)

- aiomysql和aioredis的使用

Day64 - [WebSocket的应用](#)

- WebSocket简介
- WebSocket服务器端编程
- WebSocket客户端编程
- 项目：Web聊天室

Day65 - [项目实战](#)

- 前后端分离开发和接口文档的撰写
- 使用Vue.js实现前端渲染
- 使用ECharts实现报表功能

- 使用WebSocket实现推送服务

## Day66~75 - 爬虫开发

Day66 - 网络爬虫和相关工具

Day67 - 数据采集和解析

Day68 - 存储数据

Day69 - 并发下载

Day70 - 解析动态内容

Day71 - 表单交互和验证码处理

Day72 - Scrapy入门

Day73 - Scrapy高级应用

Day74 - Scrapy分布式实现

Day75 - 爬虫项目实战

## Day76~90 - 数据处理和机器学习

Day76 - 机器学习基础

Day77 - Pandas的应用

Day78 - NumPy和SciPy的应用

Day79 - Matplotlib和数据可视化

Day80 - k最近邻(KNN)分类

Day81 - 决策树

Day82 - 贝叶斯分类

Day83 - 支持向量机(SVM)

Day84 - K-均值聚类

Day85 - 回归分析

Day86 - 大数据分析入门

Day87 - 大数据分析进阶

Day88 - Tensorflow入门

Day89 - Tensorflow实战

Day90 - 推荐系统

## Day91~100 - 团队项目开发

### 第91天：团队开发和项目选题

#### 1. 软件过程模型

- 经典过程模型（瀑布模型）
  - 可行性分析（研究做还是不做），输出《可行性分析报告》。
  - 需求分析（研究做什么），输出《需求规格说明书》和产品界面原型图。

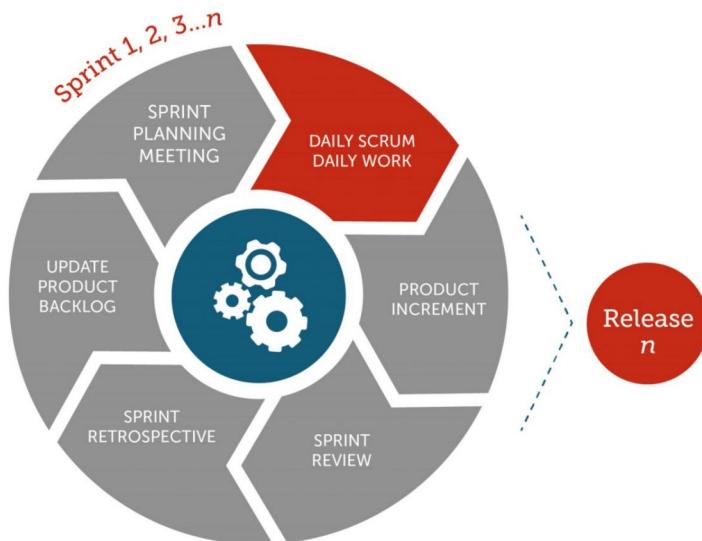
- 概要设计和详细设计，输出概念模型图、物理模型图、类图、时序图等。
  - 编码 / 测试。
  - 上线 / 维护。
- 敏捷开发 (Scrum) - 产品所有者、Scrum Master、研发人员 - Sprint
- 产品的Backlog (用户故事、产品原型)。
  - 计划会议 (评估和预算)。
  - 日常开发 (站立会议、番茄工作法、结对编程、测试先行、代码重构……)。
  - 修复bug (问题描述、重现步骤、测试人员、被指派人)。
  - 评审会议 (Showcase)。
  - 回顾会议 (当前周期做得好和不好的地方)。
- 补充：敏捷软件开发宣言
- **个体和互动** 高于 流程和工具
  - **工作的软件** 高于 详尽的文档
  - **客户合作** 高于 合同谈判
  - **响应变化** 高于 遵循计划

#### Roles:

- Product Owner
- Scrum Master
- Team Members
- Stakeholders
- Users

#### Preparation:

- Business Case & Funding
- Contractual Agreement
- Vision
- Initial Product Backlog
- Initial Release Plan
- Stakeholder Buy-in
- Assemble Team



**Sprint cycle**

角色：产品所有者（决定做什么，能对需求拍板的人）、团队负责人（解决各种问题，专注如何更好的工作，屏蔽外部对开发团队的影响）、开发团队（项目执行人员，具体指开发人员和测试人员）。

准备工作：商业案例和资金、合同、憧憬、初始产品需求、初始发布计划、入股、组建团队。

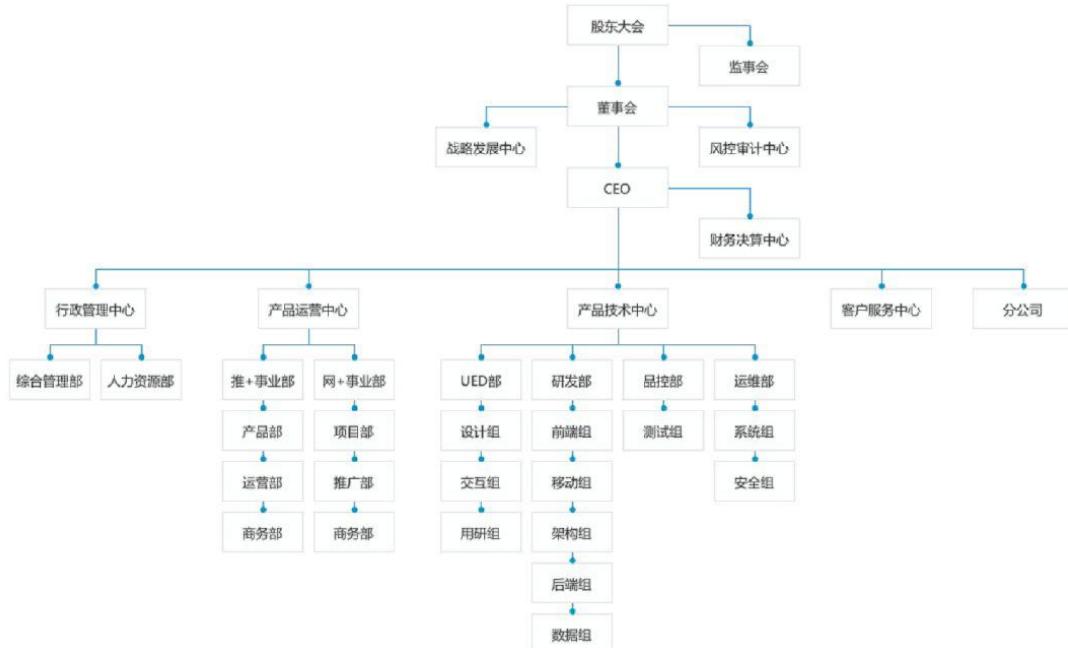
敏捷团队通常人数为8-10人。

工作量估算：将开发任务量化，包括原型、Logo设计、UI设计、前端开发等，尽量把每个工作分解到最小任务量，最小任务量标准为工作时间不能超过两天，然后估算总体项目时间。把每个任务都贴在白板上面，白板上分三部分：to do (待完成)、in progress (进行中) 和done (已完成)。

## 2. 项目团队组建

- 团队的构成和角色

说明：谢谢付祥英女士绘制了下面这张精美的公司组织架构图。



- 编程规范和代码审查 ( flake8、pylint )

```

*****
Module example11
example11.py:8:0: C0111: Missing class docstring (missing-docstring)
example11.py:8:0: R0903: Too few public methods (1/2) (too-few-public-methods)
example11.py:25:0: C0111: Missing function docstring (missing-docstring)
example11.py:42:0: C0102: Black listed name "foo" (blacklisted-name)
example11.py:42:0: C0111: Missing function docstring (missing-docstring)
example11.py:42:18: C0103: Variable name "fn" doesn't conform to snake_case naming style (invalid-name)
example11.py:49:0: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-name)
example11.py:49:0: C0103: Argument name "y" doesn't conform to snake_case naming style (invalid-name)
example11.py:49:0: C0111: Missing function docstring (missing-docstring)
example11.py:53:0: C0111: Missing function docstring (missing-docstring)
example11.py:57:4: C0103: Variable name "fn" doesn't conform to snake_case naming style (invalid-name)

-----
Your code has been rated at 6.94/10 (previous run: 6.67/10, +0.28)

```

- Python中的一些“惯例” (请参考《Python惯例-如何编写Pythonic的代码》)

- 影响代码可读性的原因：

- 代码注释太少或者没有注释
- 代码破坏了语言的最佳实践
- 反模式编程 (意大利面代码、复制-黏贴编程、自负编程、.....)

### 3. 团队开发工具介绍

- 版本控制 : Git、Mercury
- 缺陷管理 : [Gitlab](#)、[Redmine](#)
- 敏捷闭环工具 : [禅道](#)、[JIRA](#)
- 持续集成 : [Jenkins](#)、[Travis-CI](#)

请参考《团队项目开发》。

#### 项目选题和理解业务

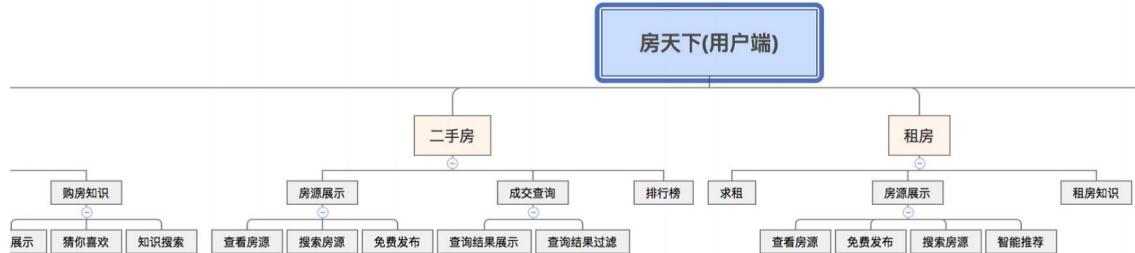
##### 1. 选题范围设定

- CMS ( 用户端 ) : 新闻聚合网站、问答/分享社区、影评/书评网站等。

- MIS ( 用户端+管理端 ) : KMS、KPI考核系统、HRS、CRM系统、供应链系统、仓储管理系统等。
- App后台 ( 管理端+数据接口 ) : 二手交易类、报刊杂志类、小众电商类、新闻资讯类、旅游类、社交类、阅读类等。
- 其他类型 : 自身行业背景和工作经验、业务容易理解和把控。

## 2. 需求理解、模块划分和任务分配

- 需求理解 : 头脑风暴和竞品分析。
- 模块划分 : 画思维导图 ( XMind ) , 每个模块是一个枝节点 , 每个具体的功能是一个叶节点 ( 用动词表述 ) , 需要确保每个叶节点无法再生出新节点 , 确定每个叶子节点的重要性、优先级和工作量。
- 任务分配 : 由项目负责人根据上面的指标为每个团队成员分配任务。



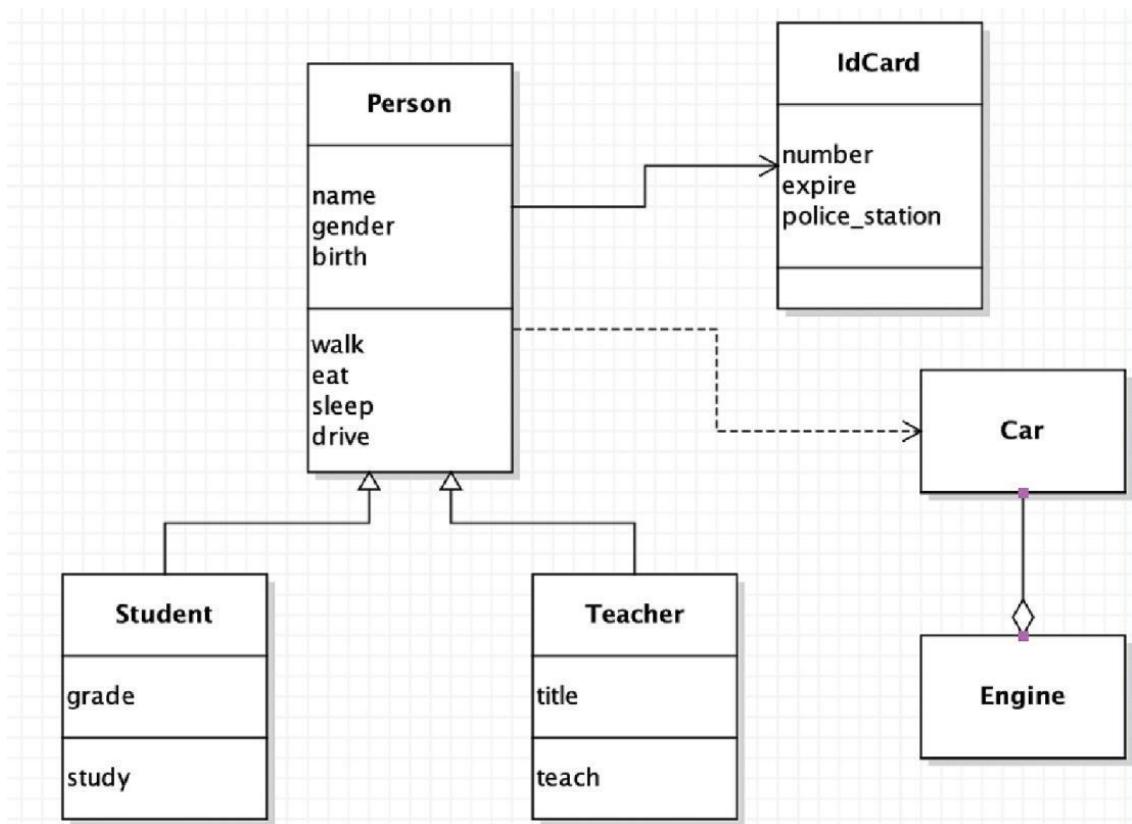
## 3. 制定项目进度表 ( 每日更新 )

模块	功能	人员	状态	完成	工时	计划开始	实际开始	计划结束	实际结束	备注
评论	添加评论	王大锤	正在进行	50%	4	2018/8/7		2018/8/7		
	删除评论	王大锤	等待	0%	2	2018/8/7		2018/8/7		
	查看评论	白元芳	正在进行	20%	4	2018/8/7		2018/8/7		需要进行代码审查
	评论投票	白元芳	等待	0%	4	2018/8/8		2018/8/8		

## 第92天 : 数据库设计和OOAD

### 概念模型和正向工程

#### 1. UML ( 统一建模语言 ) 的类图



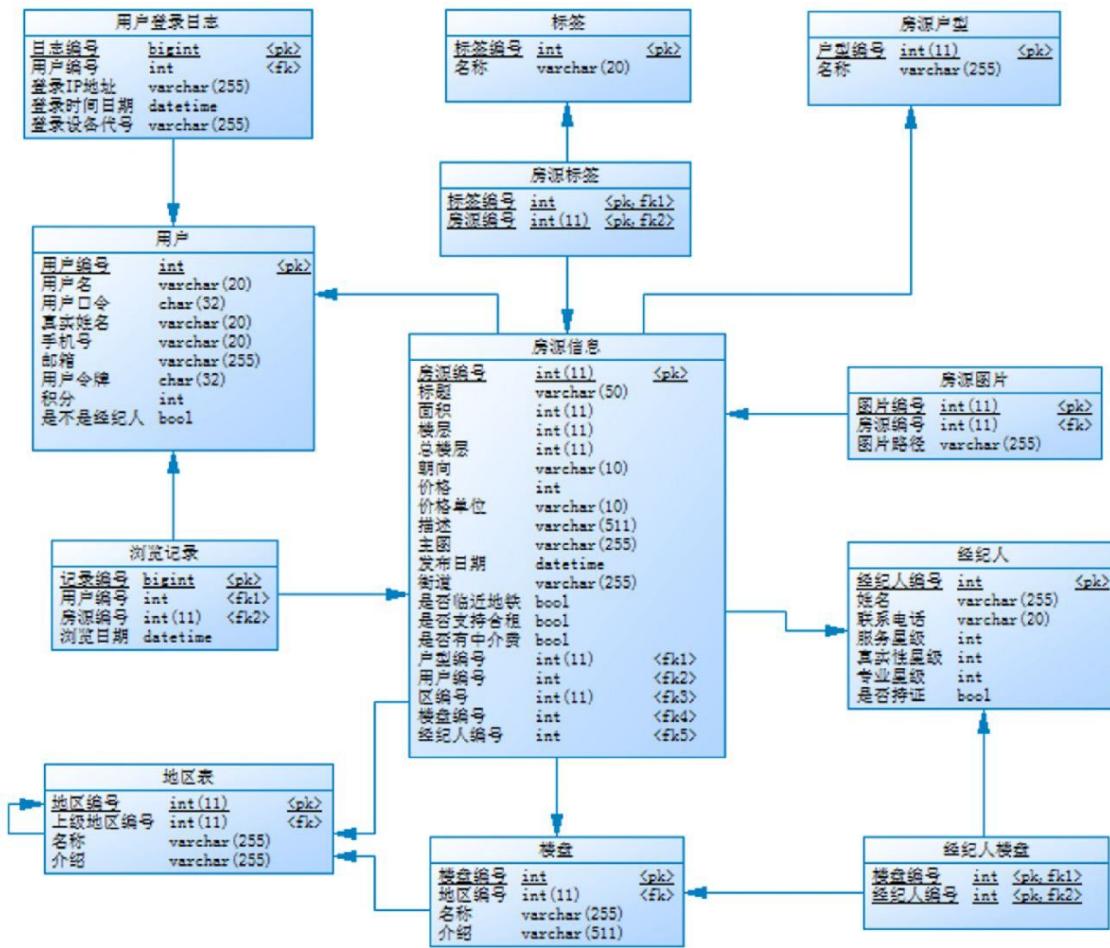
## 2. 通过模型创建表（正向工程）

```

python manage.py makemigrations app
python manage.py migrate
  
```

### 物理模型和反向工程

#### 1. PowerDesigner



## 2. 通过数据表创建模型（反向工程）

```
python manage.py inspectdb > app/models.py
```

### 第93-98天：使用Django开发项目

说明：具体内容请参考《Django知识点概述》

#### 项目开发中的公共问题

- 数据库的配置（多数据库、主从复制、数据库路由）
- 缓存的配置（分区缓存、键设置、超时设置、主从复制、故障恢复（哨兵））
- 日志的配置
- 分析和调试（Django-Debug-ToolBar）
- 好用的Python模块（日期计算、图像处理、数据加密、三方API）

#### REST API设计

- RESTful架构
  - 理解RESTful架构
  - RESTful API设计指南
  - RESTful API最佳实践
- API接口文档的撰写（《网络API接口设计》）
  - RAP2
  - YAPI
- django-REST-framework的应用

#### 项目中的重点难点剖析

- 使用缓存缓解数据库压力 - Redis

2. 使用消息队列做解耦合和削峰 - Celery + RabbitMQ

## 第99-100天：测试和部署

### 单元测试

1. 测试的种类
2. 编写单元测试 ( unittest、pytest、nose2、tox、ddt、..... )
3. 测试覆盖率 ( coverage )

### 项目部署

说明：请参考《[项目部署上线指南](#)》。

1. 部署前的准备工作
  - 关键设置 ( SECRET\_KEY / DEBUG / ALLOWED\_HOSTS / 缓存 / 数据库 )
  - HTTPS / CSRF\_COOKIE\_SECURE / SESSION\_COOKIE\_SECURE
  - 日志相关配置
2. Linux常用命令回顾
3. Linux常用服务的安装和配置
4. uWSGI/Gunicorn和Nginx的使用
  - Gunicorn和uWSGI的比较
    - 对于不需要大量定制化的简单应用程序，Gunicorn是一个不错的选择，uWSGI的学习曲线比Gunicorn要陡峭得多，Gunicorn的默认参数就已经能够适应大多数应用程序。
    - uWSGI支持异构部署。
    - 由于Nginx本身支持uWSGI，在线上一般都将Nginx和uWSGI捆绑在一起部署，而且uWSGI属于功能齐全且高度定制的WSGI中间件。
    - 在性能上，Gunicorn和uWSGI其实表现相当。
5. 虚拟化技术 ( Docker )

### 性能测试

说明：具体内容请参考《[Django知识点概述](#)》。

1. AB的使用
2. SQLslap的使用
3. sysbench的使用

### 自动化测试

1. 使用Shell和Python进行自动化测试
2. 使用Selenium实现自动化测试
  - Selenium IDE
  - Selenium WebDriver
  - Selenium Remote Control
3. 测试工具Robot Framework介绍

### 项目性能调优

1. 数据库性能调优 - 请参考《[MySQL相关知识](#)》

- 软硬件优化
- SQL优化
- 架构优化
  - 分表分库
  - 主从复制，读写分离
  - 集群架构

2. Web服务器性能优化

- Nginx负载均衡配置

- Keepalived实现高可用

### 3. 代码性能调优

- 多线程

- 异步化

### 4. 静态资源访问优化

- 云存储

- CDN

致谢：感谢我的同事古晔、张旭、肖世荣、王海飞、荣佳伟、路丰坤等在技术上给予的指导和帮助。

0-15天：

Branch: master ▾

[Find file](#) [Copy path](#)

[Python-100-Days / Day01-15 / Day01 / 初识Python.md](#)

 jackfrued Merge pull request [jackfrued#55](#) from wjsvec/patch-1

bd39f99 May 8, 2019

4 contributors



Raw

Blame

History



268 lines (184 sloc) 12.1 KB

## Day01 - 初识Python

### Python简介

#### Python的历史

1. 1989年圣诞节：Guido von Rossum开始写Python语言的编译器。
2. 1991年2月：第一个Python编译器（同时也是解释器）诞生，它是用C语言实现的（后面又出现了Java和C#实现的版本Jython和IronPython，以及PyPy、Brython、Pyston等其他实现），可以调用C语言的库函数。在最早的版本中，Python已经提供了对“类”，“函数”，“异常处理”等构造块的支持，同时提供了“列表”和“字典”等核心数据类型，同时支持以模块为基础的拓展系统。
3. 1994年1月：Python 1.0正式发布。
4. 2000年10月16日：Python 2.0发布，增加了实现完整的垃圾回收，提供了对Unicode的支持。与此同时，Python的整个开发过程更加透明，社区对开发进度的影响逐渐扩大，生态圈开始慢慢形成。
5. 2008年12月3日：Python 3.0发布，它并不完全兼容之前的Python代码，不过因为目前还有不少公司在项目和运维中使用Python 2.x版本，所以Python 3.x的很多新特性后来也被移植到Python 2.6/2.7版本中。

目前我们使用的Python 3.7.x的版本是在2018年发布的，Python的版本号分为三段，形如A.B.C。其中A表示大版本号，一般当整体重写，或出现不向后兼容的改变时，增加A；B表示功能更新，出现新功能时增加B；C表示小的改动（如修复了某个Bug），只要有修改就增加C。如果对Python的历史感兴趣，可以查看一篇名为《[Python简史](#)》的博文。

#### Python的优缺点

Python的优点很多，简单的可以总结为以下几点。

1. 简单和明确，做一件事只有一种方法。
2. 学习曲线低，跟其他很多语言相比，Python更容易上手。
3. 开放源代码，拥有强大的社区和生态圈。
4. 解释型语言，天生具有平台可移植性。
5. 支持两种主流的编程范式（面向对象编程和函数式编程）都提供了支持。
6. 可扩展性和可嵌入性，可以调用C/C++代码，也可以在C/C++中调用Python。
7. 代码规范程度高，可读性强，适合有代码洁癖和强迫症的人群。

Python的缺点主要集中在以下几点。

1. 执行效率稍低，因此计算密集型任务可以由C/C++编写。
2. 代码无法加密，但是现在的公司很多都不是卖软件而是卖服务，这个问题会被淡化。
3. 在开发时可以选择的框架太多（如Web框架就有100多个），有选择的地方就有错误。

## Python的应用领域

目前Python在云基础设施、DevOps、网络爬虫开发、数据分析挖掘、机器学习等领域都有着广泛的应用，因此也产生了Web后端开发、数据接口开发、自动化运维、自动化测试、科学计算和可视化、数据分析、量化交易、机器人开发、图像识别和处理等一系列的职位。

## 搭建编程环境

### Windows环境

可以在[Python官方网站](#)下载到Python的Windows安装程序（exe文件），需要注意的是如果在Windows 7环境下安装需要先安装Service Pack 1补丁包（可以通过一些工具软件自动安装系统补丁的功能来安装），安装过程建议勾选“Add Python 3.6 to PATH”（将Python 3.6添加到PATH环境变量）并选择自定义安装，在设置“Optional Features”界面最好将“pip”、“td/tk”、“Python test suite”等项全部勾选上。强烈建议使用自定义的安装路径并保证路径中没有中文。安装完成会看到“Setup was successful”的提示，但是在启动Python环境时可能会因为缺失一些动态链接库文件而导致Python解释器无法运行，常见的问题是api-ms-win-crt\*.dll缺失以及更新DirectX之后导致某些动态链接库文件缺失，前者可以参照[《api-ms-win-crt\\*.dll缺失原因分析和解决方法》](#)一文讲解的方法进行处理或者直接在[微软官网](#)下载Visual C++ Redistributable for Visual Studio 2015文件进行修复，后者可以下载一个DirectX修复工具进行修复。

### Linux环境

Linux环境自带了Python 2.x版本，但是如果要更新到3.x的版本，可以在[Python的官方网站](#)下载Python的源代码并通过源代码构建安装的方式进行安装，具体的步骤如下所示。

安装依赖库（因为没有这些依赖库可能在源代码构件安装时因为缺失底层依赖库而失败）。

```
yum -y install wget gcc zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel
```



下载Python源代码并解压缩到指定目录。

```
wget https://www.python.org/ftp/python/3.7.3/Python-3.7.3.tgz  
xz -d Python-3.7.3.tar.xz  
tar -xvf Python-3.7.3.tar
```

切换至Python源代码目录并执行下面的命令进行配置和安装。

```
cd Python-3.7.3  
../configure --prefix=/usr/local/python37 --enable-optimizations  
make && make install
```

修改用户主目录下名为.bash\_profile的文件，配置PATH环境变量并使其生效。

```
cd ~  
vim .bash_profile  
  
# ... 此处省略上面的代码 ...  
  
export PATH=$PATH:/usr/local/python37/bin  
  
# ... 此处省略下面的代码 ...  
  
source .bash_profile
```

## MacOS环境

MacOS也是自带了Python 2.x版本的，可以通过[Python的官方网站](#)提供的安装文件（pkg文件）安装3.x的版本。默认安装完成后，可以通过在终端执行python命令来启动2.x版本的Python解释器，可以通过执行python3命令来启动3.x版本的Python解释器。

## 从终端运行Python程序

### 确认Python的版本

在终端或命令行提示符中键入下面的命令。

```
python --version
```

当然也可以先输入python进入交互式环境，再执行以下的代码检查Python的版本。

```
import sys  
  
print(sys.version_info)  
print(sys.version)
```

## 编写Python源代码

可以用文本编辑工具（推荐使用Sublime、Atom、TextMate、VSCode等高级文本编辑工具）编写Python源代码并将其命名为hello.py保存起来，代码内容如下所示。

```
print('hello, world!')
```

## 运行程序

切换到源代码所在的目录并执行下面的命令，看看屏幕上是否输出了"hello, world!"。

```
python hello.py
```

## 代码中的注释

注释是编程语言的一个重要组成部分，用于在源代码中解释代码的作用从而增强程序的可读性和可维护性，当然也可以将源代码中不需要参与运行的代码段通过注释来去掉，这一点在调试程序的时候经常用到。注释在随源代码进入预处理器或编译时会被移除，不会在目标代码中保留也不会影响程序的执行结果。

1. 单行注释 - 以#和空格开头的部分
2. 多行注释 - 三个引号开头，三个引号结尾

```
'''  
第一个Python程序 - hello, world!  
向伟大的Dennis M. Ritchie先生致敬
```

```
Version: 0.1  
Author: 骆昊  
'''
```

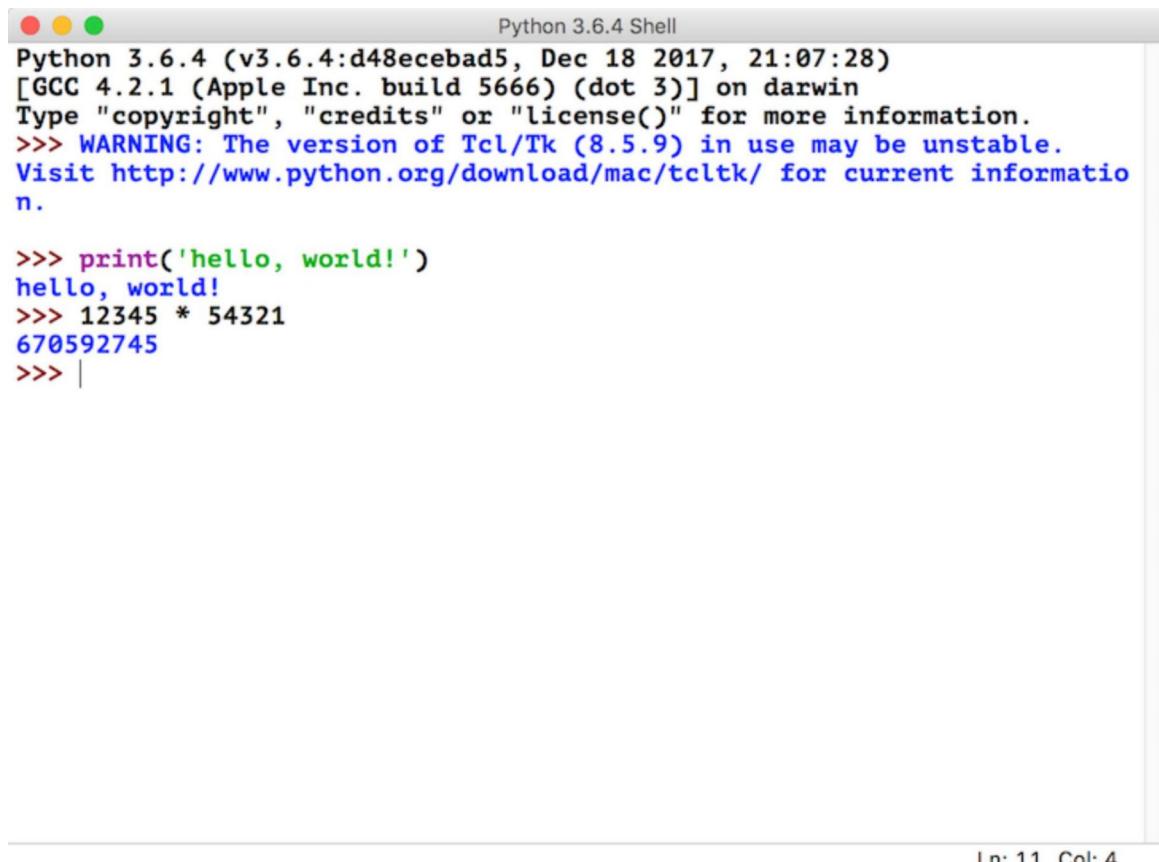
```
print('hello, world!')  
# print("你好,世界!")  
print('你好', '世界')
```

```
print('hello', 'world', sep=', ', end='!')
print('goodbye, world', end='!\n')
```

## 其他工具介绍

### IDLE - 自带的集成开发工具

IDLE是安装Python环境时自带的集成开发工具，如下图所示。但是由于IDLE的用户体验并不是那么好所以很少在实际开发中被采用。



The screenshot shows the Python 3.6.4 Shell window. The title bar reads "Python 3.6.4 Shell". The main area displays the following text:

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> print('hello, world!')
hello, world!
>>> 12345 * 54321
670592745
>>> |
```

In the bottom right corner of the window, there is a status bar with the text "Ln: 11 Col: 4".

### IPython - 更好的交互式编程工具

IPython是一种基于Python的交互式解释器。相较于原生的Python Shell，IPython提供了更为强大的编辑和交互功能。可以通过Python的包管理工具pip安装IPython和Jupyter，具体的操作如下所示。

```
pip install ipython jupyter
```

或者

```
python -m pip install ipython jupyter
```

安装成功后，可以通过下面的ipython命令启动IPython，如下图所示。

```
jackfrued:~ Hao$ ipython
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('hello, world!')
hello, world!

In [2]: ls
Applications/          Downloads/
Applications (Parallels)/ Library/
Desktop/                Movies/
Documents/               Music/
                                         Parallels/
                                         Pictures/
                                         Public/
                                         PycharmProjects/

In [3]: import random

In [4]: random.rand
        randint()  Random      random()    randrange()
```

当然我们也可以通过Jupyter运行名为notebook的项目在浏览器窗口中进行交互式操作。

```
jupyter notebook
```

Python学习  
Day01 - 初始Python  
Python的第一个程序

```
In [1]: print('hello, world!')
hello, world!
```

查看Python版本

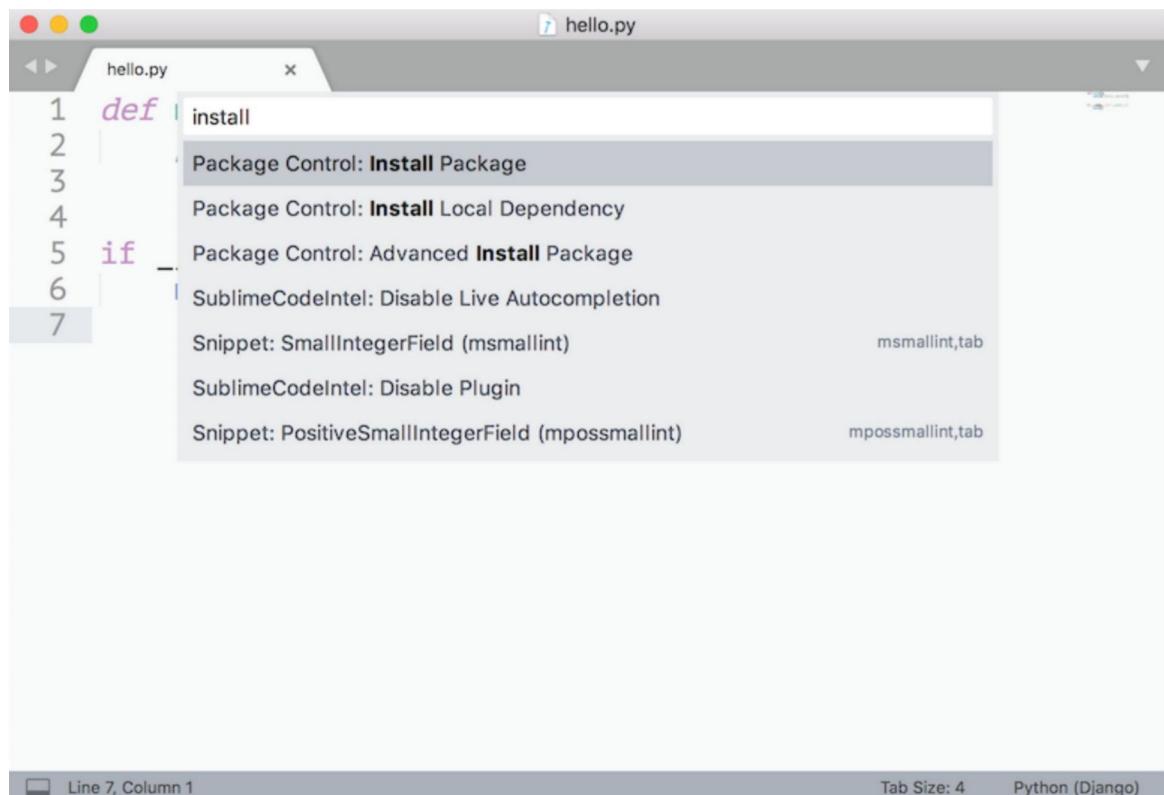
```
In [2]: import sys
print(sys.version_info)
print(sys.version)

sys.version_info(major=3, minor=6, micro=4, releaselevel='final', serial=0)
3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
```

anaconda - 一站式的数据科学神器

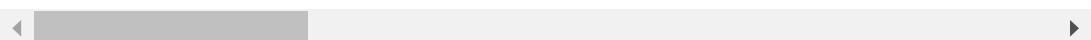
Anaconda指的是一个开源的Python发行版本，其包含了conda、Python等180多个科学包及其依赖项。因为包含了大量的科学包，Anaconda的下载文件比较大（约531MB），如果只需要某些包，或者需要节省带宽或存储空间，也可以使用Miniconda这个较小的发行版（仅包含conda和Python）。对于学习数据科学的人来说，anaconda是绝对的神器，安装简便，而且anaconda支持安装相关软件【例如前文提到的ipython，jupyter notebook，甚至有R等其他数据科学软件】[一个相当有价值的介绍](#)现在唯一的问题在于清华镜像服务已经关闭，跨国下载会比较慢

## Sublime - 文本编辑神器



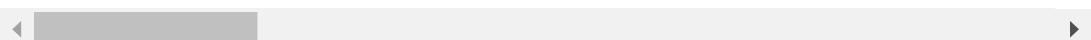
- 首先可以通过[官方网站](#)下载安装程序安装Sublime 3或Sublime 2。
- 安装包管理工具。通过快捷键Ctrl+`或者在View菜单中选择Show Console打开控制台，输入下面的代码。
  - Sublime 3

```
import urllib.request,os;pf='Package Control.sublime-package';ipp=sublime.instal]
```



- Sublime 2

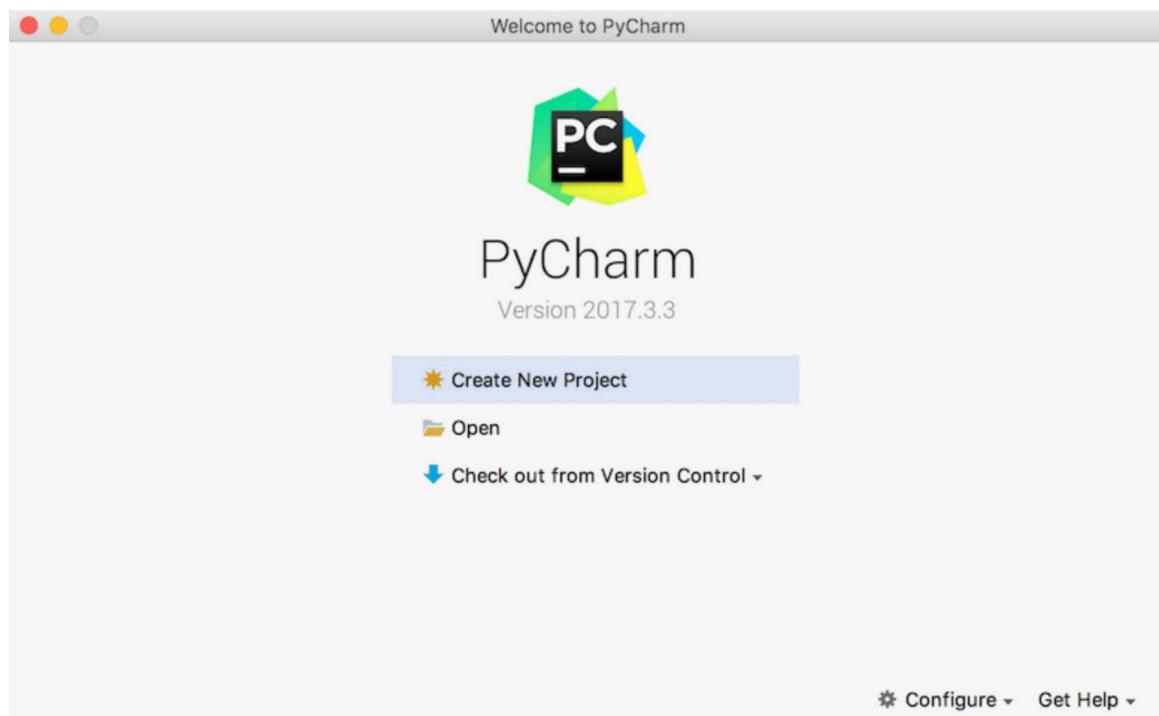
```
import urllib2,os;pf='Package Control.sublime-package';ipp=sublime.installed_pack
```



- 安装插件。通过Preference菜单的Package Control或快捷键Ctrl+Shift+P打开命令面板，在面板中输入Install Package就可以找到安装插件的工具，然后再查找需要的插件。我们推荐大家安装以下几个插件：
  - SublimeCodeIntel - 代码自动补全工具插件。
  - Emmet - 前端开发代码模板插件。
  - Git - 版本控制工具插件。
  - Python PEP8 Autoformat - PEP8规范自动格式化插件。
  - ConvertToUTF8 - 将本地编码转换为UTF-8。

## PyCharm - Python开发神器

PyCharm的安装、配置和使用我们在后面会进行介绍。



## 练习

1. 在Python交互环境中查看下面的代码结果，并将内容翻译成中文。

```
import this

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
```

Errors should never **pass** silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- **and** preferably only one --obvious way to do it.  
Although that way may **not** be obvious at first unless you're Dutch.  
Now **is** better than never.  
Although never **is** often better than \*right\* now.  
If the implementation **is** hard to explain, it's a bad idea.  
If the implementation **is** easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!

## 2. 学习使用turtle在屏幕上绘制图形。

```
import turtle

turtle.pensize(4)
turtle.pencolor('red')
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.mainloop()
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day01-15 / Day02 / 语言元素.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



223 lines (170 sloc) 10.2 KB

## Day02 - 语言元素

### 指令和程序

计算机的硬件系统通常由五大部件构成，包括：运算器、控制器、存储器、输入设备和输出设备。其中，运算器和控制器放在一起就是我们通常所说的中央处理器，它的功能是执行各种运算和控制指令以及处理计算机软件中的数据。我们通常所说的程序实际上就是指令的集合，我们程序就是将一系列的指令按照某种方式组织到一起，然后通过这些指令去控制计算机做我们想让它做的事情。今天我们使用的计算机虽然器件做工越来越精密，处理能力越来越强大，但究其本质来说仍然属于“[冯·诺依曼结构](#)”的计算机。“冯·诺依曼结构”有两个关键点，一是指出要将存储设备与中央处理器分开，二是提出了将数据以二进制方式编码。二进制是一种“逢二进一”的计数法，跟我们人类使用的“逢十进一”的计数法没有实质性的区别，人类因为有十根手指所以使用了十进制（因为在数数时十根手指用完之后就只能进位了，当然凡事都有例外，玛雅人可能是因为长年光着脚的原因把脚趾头也算上了，于是他们使用了二十进制的计数法，在这种计数法的指导下玛雅人的历法就与我们平常使用的历法不一样，而按照玛雅人的历法，2012年是上一个所谓的“太阳纪”的最后一年，而2013年则是新的“太阳纪”的开始，后来这件事情被以讹传讹的方式误传为“2012年是玛雅人预言的世界末日”这种荒诞的说法，今天我们可以大胆的猜测，玛雅文明之所以发展缓慢估计也与使用了二十进制有关）。对于计算机来说，二进制在物理器件上来说是最容易实现的（高电压表示1，低电压表示0），于是在“冯·诺依曼结构”的计算机都使用了二进制。虽然我们并不需要每个程序员都能够使用二进制的思维方式来工作，但是了解二进制以及它与我们生活中的十进制之间的转换关系，以及二进制与八进制和十六进制的转换关系还是有必要的。如果你对这一点不熟悉，可以自行使用[维基百科](#)或者[百度百科](#)科普一下。

### 变量和类型

在程序设计中，变量是一种存储数据的载体。计算机中的变量是实际存在的数据或者说 是存储器中存储数据的一块内存空间，变量的值可以被读取和修改，这是所有计算和控制的基础。计算机能处理的数据有很多中类型，除了数值之外还可以处理文本、图形、音频、视频等各种各样的数据，那么不同的数据就需要定义不同的存储类型。Python 中的数据类型很多，而且也允许我们自定义新的数据类型（这一点在后面会讲到），我们先介绍几种常用的数据类型。

- 整型：Python 中可以处理任意大小的整数（Python 2.x 中有 int 和 long 两种类型的整数，但这种区分对 Python 来说意义不大，因此在 Python 3.x 中整数只有 int 这一种了），而且支持二进制（如 `0b100`，换算成十进制是 4）、八进制（如 `0o100`，换算成十进制是 64）、十进制（`100`）和十六进制（`0x100`，换算成十进制是 256）的表示法。
- 浮点型：浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的，浮点数除了数学写法（如 `123.456`）之外还支持科学计数法（如 `1.23456e2`）。
- 字符串型：字符串是以单引号或双引号括起来的任意文本，比如 `'hello'` 和 `"hello"`，字符串还有原始字符串表示法、字节字符串表示法、Unicode 字符串表示法，而且可以书写成多行的形式（用三个单引号或三个双引号开头，三个单引号或三个双引号结尾）。
- 布尔型：布尔值只有 `True`、`False` 两种值，要么是 `True`，要么是 `False`，在 Python 中，可以直接用 `True`、`False` 表示布尔值（请注意大小写），也可以通过布尔运算计算出来（例如 `3 < 5` 会产生布尔值 `True`，而 `2 == 1` 会产生布尔值 `False`）。
- 复数型：形如 `3+5j`，跟数学上的复数表示一样，唯一不同的是虚部的 `i` 换成了 `j`。

## 变量命名

对于每个变量我们需要给它取一个名字，就如同我们每个人都有属于自己的响亮的名字一样。在 Python 中，变量命名需要遵循以下这些必须遵守硬性规则和强烈建议遵守的非硬性规则。

- 硬性规则：
  - 变量名由字母（广义的 Unicode 字符，不包括特殊字符）、数字和下划线构成，数字不能开头。
  - 大小写敏感（大写的 `a` 和小写的 `A` 是两个不同的变量）。
  - 不要跟关键字（有特殊含义的单词，后面会讲到）和系统保留字（如函数、模块等的名字）冲突。
- PEP 8 要求：
  - 用小写字母拼写，多个单词用下划线连接。
  - 受保护的实例属性用单个下划线开头（后面会讲到）。
  - 私有的实例属性用两个下划线开头（后面会讲到）。

当然，作为一个专业的程序员，给变量（事实上应该是所有的标识符）命名时做到见名知意也是非常重要的。

## 变量的使用

下面通过几个例子来说明变量的类型和变量使用。

....

使用变量保存数据并进行算术运算

Version: 0.1

Author: 骆昊

....

```
a = 321
b = 123
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
print(a ** b)
```

....

使用函数输入

使用int()进行类型转换

用占位符格式化输出的字符串

Version: 0.1

Author: 骆昊

....

```
a = int(input('a = '))
b = int(input('b = '))
print('%d + %d = %d' % (a, b, a + b))
print('%d - %d = %d' % (a, b, a - b))
print('%d * %d = %d' % (a, b, a * b))
print('%d / %d = %f' % (a, b, a / b))
print('%d // %d = %d' % (a, b, a // b))
print('%d %% %d = %d' % (a, b, a % b))
print('%d ** %d = %d' % (a, b, a ** b))
```

....

使用type()检查变量的类型

Version: 0.1

Author: 骆昊

Date: 2018-02-27

....

```
a = 100
b = 12.345
c = 1 + 5j
```

```
d = 'hello, world'  
e = True  
print(type(a))  
print(type(b))  
print(type(c))  
print(type(d))  
print(type(e))
```

在对变量类型进行转换时可以使用Python的内置函数（准确的说下面列出的并不是真正意义上的函数，而是后面我们要讲到的创建对象的构造方法）。

- `int()`：将一个数值或字符串转换成整数，可以指定进制。
- `float()`：将一个字符串转换成浮点数。
- `str()`：将指定的对象转换成字符串形式，可以指定编码。
- `chr()`：将整数转换成该编码对应的字符串（一个字符）。
- `ord()`：将字符串（一个字符）转换成对应的编码（整数）。

## 运算符

Python支持多种运算符，下表大致按照优先级从高到低的顺序列出了所有的运算符，我们会陆续使用到它们。

运算符	描述
<code>[]</code> <code>[:]</code>	下标，切片
<code>**</code>	指数
<code>~</code> <code>+</code> <code>-</code>	按位取反, 正负号
<code>*</code> <code>/</code> <code>%</code> <code>//</code>	乘，除，模，整除
<code>+</code> <code>-</code>	加，减
<code>&gt;&gt;</code> <code>&lt;&lt;</code>	右移，左移
<code>&amp;</code>	按位与
<code>^</code>	按位异或
<code>&lt;=</code> <code>&lt;</code> <code>&gt;</code> <code>&gt;=</code>	小于等于，小于，大于，大于等于
<code>==</code> <code>!=</code>	等于，不等于
<code>is</code> <code>is not</code>	身份运算符
<code>in</code> <code>not in</code>	成员运算符
<code>not</code> <code>or</code> <code>and</code>	逻辑运算符

运算符	描述
= += -= *= /= %= //= **= &= ^=	= ^= >>= <<= `

\*\*说明：\*\*在实际开发中，如果搞不清楚优先级可以使用括号来确保运算的执行顺序。

下面的例子演示了运算符的使用。

```
"""
运算符的使用
```

Version: 0.1

Author: 骆昊

```
"""

a = 5
b = 10
c = 3
d = 4
e = 5
a += b
a -= c
a *= d
a /= e
print("a = ", a)

flag1 = 3 > 2
flag2 = 2 < 1
flag3 = flag1 and flag2
flag4 = flag1 or flag2
flag5 = not flag1
print("flag1 = ", flag1)
print("flag2 = ", flag2)
print("flag3 = ", flag3)
print("flag4 = ", flag4)
print("flag5 = ", flag5)
print(flag1 is True)
print(flag2 is not False)
```

## 练习

练习1：华氏温度转摄氏温度。

```
"""

将华氏温度转换为摄氏温度
F = 1.8C + 32
```

Version: 0.1

Author: 骆昊

```
"""
```

```
f = float(input('请输入华氏温度: '))
c = (f - 32) / 1.8
print('%.1f华氏度 = %.1f摄氏度' % (f, c))
```

## 练习2：输入圆的半径计算计算周长和面积。

```
"""
```

输入半径计算圆的周长和面积

Version: 0.1

Author: 骆昊

```
"""

import math

radius = float(input('请输入圆的半径: '))
perimeter = 2 * math.pi * radius
area = math.pi * radius * radius
print('周长: %.2f' % perimeter)
print('面积: %.2f' % area)
```

## 练习3：输入年份判断是不是闰年。

```
"""
```

输入年份 如果是闰年输出True 否则输出False

Version: 0.1

Author: 骆昊

```
"""

year = int(input('请输入年份: '))
# 如果代码太长写成一行不利于阅读 可以使用\或()折行
is_leap = (year % 4 == 0 and year % 100 != 0 or
           year % 400 == 0)
print(is_leap)
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day01-15 / Day03 / 分支结构.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



231 lines (189 sloc) 6.63 KB

## Day03 - 分支结构

### 分支结构的应用场景

迄今为止，我们写的Python代码都是一条一条语句顺序执行，这种结构的代码我们称之为顺序结构。然而仅有顺序结构并不能解决所有的问题，比如我们设计一个游戏，游戏第一关的通关条件是玩家获得1000分，那么在完成本局游戏后我们要根据玩家得到分数来决定究竟是进入第二关还是告诉玩家“Game Over”，这里就会产生两个分支，而且这两个分支只有一个会被执行，这就是程序中分支结构。类似的场景还有很多，给大家一分钟的时间，你应该可以想到至少5个以上这样的例子，赶紧试一试。

### if语句的使用

在Python中，要构造分支结构可以使用 `if`、`elif` 和 `else` 关键字。所谓关键字就是有特殊含义的单词，像 `if` 和 `else` 就是专门用于构造分支结构的关键字，很显然你不能够使用它作为变量名（事实上，用作其他的标识符也是不可以）。下面的例子中演示了如何构造一个分支结构。

```
"""
用户身份验证
"""

Version: 0.1
Author: 骆昊
"""

username = input('请输入用户名: ')
password = input('请输入口令: ')
# 如果希望输入口令时 终端中没有回显 可以使用getpass模块的getpass函数
# import getpass
# password = getpass.getpass('请输入口令: ')
if username == 'admin' and password == '123456':
    print('身份验证成功!')
```

```
else:  
    print('身份验证失败!')
```

唯一需要说明的是和C/C++、Java等语言不同，Python中没有用花括号来构造代码块而是使用了缩进的方式来设置代码的层次结构，如果 if 条件成立的情况下需要执行多条语句，只要保持多条语句具有相同的缩进就可以了，换句话说连续的代码如果又保持了相同的缩进那么它们属于同一个代码块，相当于是同一个执行的整体。

当然如果要构造出更多的分支，可以使用 if...elif...else... 结构，例如下面的分段函数求值。

```
$$f(x)=\begin{cases} 3x-5 & (x>1) \\ x+2 & (-1 \leq x \leq 1) \\ 5x+3 & (x<-1) \end{cases}$$
```

```
"""  
分段函数求值
```

```
    3x - 5  (x > 1)  
f(x) =  x + 2  (-1 <= x <= 1)  
      5x + 3  (x < -1)
```

```
Version: 0.1  
Author: 骆昊
```

```
"""  
  
x = float(input('x = '))  
if x > 1:  
    y = 3 * x - 5  
elif x >= -1:  
    y = x + 2  
else:  
    y = 5 * x + 3  
print('f(%.2f) = %.2f' % (x, y))
```

当然根据实际开发的需要，分支结构是可以嵌套的，例如判断是否通关以后还要根据你获得的宝物或者道具的数量对你的表现给出等级（比如点亮两颗或三颗星星），那么我们就需要在 if 的内部构造出一个新的分支结构，同理 elif 和 else 中也可以再构造新的分支，我们称之为嵌套的分支结构，也就是说上面的代码也可以写成下面的样子。

```
"""  
分段函数求值  
    3x - 5  (x > 1)  
f(x) =  x + 2  (-1 <= x <= 1)  
      5x + 3  (x < -1)  
  
Version: 0.1  
Author: 骆昊
```

```
x = float(input('x = '))
if x > 1:
    y = 3 * x - 5
else:
    if x >= -1:
        y = x + 2
    else:
        y = 5 * x + 3
print('f(%.2f) = %.2f' % (x, y))
```

\*\*说明：\*\*大家可以自己感受一下这两种写法到底是哪一种更好。在之前我们提到的Python之禅中有这么一句话“Flat is better than nested.”，之所以提出这个观点是因为嵌套结构的嵌套层次多了之后会严重的影响代码的可读性，如果可以使用扁平化的结构就不要去用嵌套，因此之前的写法是更好的做法。

## 练习

### 练习1：英制单位与公制单位互换

```
"""
英制单位英寸和公制单位厘米互换

Version: 0.1
Author: 骆昊
"""

value = float(input('请输入长度: '))
unit = input('请输入单位: ')
if unit == 'in' or unit == '英寸':
    print('%f英寸 = %f厘米' % (value, value * 2.54))
elif unit == 'cm' or unit == '厘米':
    print('%f厘米 = %f英寸' % (value, value / 2.54))
else:
    print('请输入有效的单位')
```

### 练习2：掷骰子决定做什么

```
"""
掷骰子决定做什么事情

Version: 0.1
Author: 骆昊
"""

from random import randint

face = randint(1, 6)
if face == 1:
    result = '唱歌'
```

```
elif face == 2:  
    result = '跳个舞'  
elif face == 3:  
    result = '学狗叫'  
elif face == 4:  
    result = '做俯卧撑'  
elif face == 5:  
    result = '念绕口令'  
else:  
    result = '讲冷笑话'  
print(result)
```

\*\*说明：\*\*上面的代码中使用了random模块的randint函数生成指定范围的随机数来模拟掷骰子。

### 练习3：百分制成绩转等级制

```
"""  
百分制成绩转等级制成绩  
90分以上    --> A  
80分~89分    --> B  
70分~79分    --> C  
60分~69分    --> D  
60分以下    --> E  
  
Version: 0.1  
Author: 骆昊  
"""  
  
score = float(input('请输入成绩: '))  
if score >= 90:  
    grade = 'A'  
elif score >= 80:  
    grade = 'B'  
elif score >= 70:  
    grade = 'C'  
elif score >= 60:  
    grade = 'D'  
else:  
    grade = 'E'  
print('对应的等级是:', grade)
```

### 练习4：输入三条边长如果能构成三角形就计算周长和面积

```
"""  
判断输入的边长能否构成三角形  
如果能则计算出三角形的周长和面积
```

```
Version: 0.1  
Author: 骆昊  
"""
```

```

import math

a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))
if a + b > c and a + c > b and b + c > a:
    print('周长: %f' % (a + b + c))
    p = (a + b + c) / 2
    area = math.sqrt(p * (p - a) * (p - b) * (p - c))
    print('面积: %f' % (area))
else:
    print('不能构成三角形')

```

\*\*说明：\*\*上面的代码中使用了 `math` 模块的 `sqrt` 函数来计算平方根。用边长计算三角形面积的公式叫做海伦公式。

### 练习5：个人所得税计算器。

```

"""
输入月收入和五险一金计算个人所得税

Version: 0.1
Author: 骆昊
"""

salary = float(input('本月收入: '))
insurance = float(input('五险一金: '))
diff = salary - insurance - 3500
if diff <= 0:
    rate = 0
    deduction = 0
elif diff < 1500:
    rate = 0.03
    deduction = 0
elif diff < 4500:
    rate = 0.1
    deduction = 105
elif diff < 9000:
    rate = 0.2
    deduction = 555
elif diff < 35000:
    rate = 0.25
    deduction = 1005
elif diff < 55000:
    rate = 0.3
    deduction = 2755
elif diff < 80000:
    rate = 0.35
    deduction = 5505
else:
    rate = 0.45

```

```
deduction = 13505
tax = abs(diff * rate - deduction)
print('个人所得税: ￥%.2f元' % tax)
print('实际到手收入: ￥%.2f元' % (diff + 3500 - tax))
```

\*\*说明:\*\*上面的代码中使用了Python内置的 `abs()` 函数取绝对值来处理 -0 的问题。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day01-15 / Day04 / 循环结构.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



214 lines (166 sloc) 5.7 KB

## Day04 - 循环结构

### 循环结构的应用场景

如果在程序中我们需要重复的执行某条或某些指令，例如用程序控制机器人踢足球，如果机器人持球而且还没有进入射门范围，那么我们就要一直发出让机器人向球门方向奔跑的指令。当然你可能已经注意到了，刚才的描述中其实不仅仅有需要重复的动作，还有我们上一个章节讲到的分支结构。再举一个简单的例子，比如在我们的程序中要实现每隔1秒中在屏幕上打印一个"hello, world"这样的字符串并持续一个小时，我们肯定不能够将 `print('hello, world')` 这句代码写上3600遍，如果真的需要这样做那么编程的工作就太无聊了。因此，我们需要了解一下循环结构，有了循环结构我们就可以轻松的控制某件事或者某些事重复、重复、再重复的发生。在Python中构造循环结构有两种做法，一种是 `for-in` 循环，一种是 `while` 循环。

### for-in循环

如果明确的知道循环执行的次数或者是对一个容器进行迭代（后面会讲到），那么我们推荐使用 `for-in` 循环，例如下面代码中计算 $\sum_{n=1}^{100} n$ 。

用 `for` 循环实现1~100求和

Version: 0.1

Author: 骆昊

"""

```
sum = 0
for x in range(101):
    sum += x
print(sum)
```

需要说明的是上面代码中的 `range` 类型，`range` 可以用来产生一个不变的数值序列，而且这个序列通常都是用在循环中的，例如：

- `range(101)` 可以产生一个0到100的整数序列。
- `range(1, 100)` 可以产生一个1到99的整数序列。
- `range(1, 100, 2)` 可以产生一个1到99的奇数序列，其中的2是步长，即数值序列的增量。

知道了这一点，我们可以用下面的代码来实现1~100之间的偶数求和。

```
"""
用for循环实现1~100之间的偶数求和
```

```
Version: 0.1
Author: 骆昊
"""
```

```
sum = 0
for x in range(2, 101, 2):
    sum += x
print(sum)
```

也可以通过在循环中使用分支结构的方式来实现相同的功能，代码如下所示。

```
"""
用for循环实现1~100之间的偶数求和
```

```
Version: 0.1
Author: 骆昊
"""
```

```
sum = 0
for x in range(1, 101):
    if x % 2 == 0:
        sum += x
print(sum)
```

## while循环

如果要构造不知道具体循环次数的循环结构，我们推荐使用 `while` 循环，`while` 循环通过一个能够产生或转换出 `bool` 值的表达式来控制循环，表达式的值为 `True` 循环继续，表达式的值为 `False` 循环结束。下面我们通过一个“猜数字”的小游戏（计算机出一个1~100之间的随机数，人输入自己猜的数字，计算机给出对应的提示信息，直到人猜出计算机出的数字）来看看如何使用 `while` 循环。

```
"""
猜数字游戏
```

计算机出一个1~100之间的随机数由人来猜  
计算机根据人猜的数字分别给出提示大一点/小一点/猜对了

```
Version: 0.1
Author: 骆昊
"""

import random

answer = random.randint(1, 100)
counter = 0
while True:
    counter += 1
    number = int(input('请输入: '))
    if number < answer:
        print('大一点')
    elif number > answer:
        print('小一点')
    else:
        print('恭喜你猜对了!')
        break
print('你总共猜了%d次' % counter)
if counter > 7:
    print('你的智商余额明显不足')
```

\*\*说明：\*\*上面的代码中使用了 `break` 关键字来提前终止循环，需要注意的是 `break` 只能终止它所在的那个循环，这一点在使用嵌套的循环结构（下面会讲到）需要引起注意。除了 `break` 之外，还有另一个关键字是 `continue`，它可以用来放弃本次循环后续的代码直接让循环进入下一轮。

和分支结构一样，循环结构也是可以嵌套的，也就是说在循环中还可以构造循环结构。  
下面的例子演示了如何通过嵌套的循环来输出一个九九乘法表。

```
"""

输出乘法口诀表(九九表)

Version: 0.1
Author: 骆昊
"""

for i in range(1, 10):
    for j in range(1, i + 1):
        print('%d*%d=%d' % (i, j, i * j), end='\t')
    print()
```

## 练习

练习1：输入一个数判断是不是素数。

```
"""
输入一个正整数判断它是不是素数

Version: 0.1
Author: 骆昊
Date: 2018-03-01
"""

from math import sqrt

num = int(input('请输入一个正整数: '))
end = int(sqrt(num))
is_prime = True
for x in range(2, end + 1):
    if num % x == 0:
        is_prime = False
        break
if is_prime and num != 1:
    print('%d是素数' % num)
else:
    print('%d不是素数' % num)
```

## 练习2：输入两个正整数，计算最大公约数和最小公倍数。

```
"""
输入两个正整数计算最大公约数和最小公倍数

Version: 0.1
Author: 骆昊
Date: 2018-03-01
"""

x = int(input('x = '))
y = int(input('y = '))
if x > y:
    x, y = y, x
for factor in range(x, 0, -1):
    if x % factor == 0 and y % factor == 0:
        print('%d和%d的最大公约数是%d' % (x, y, factor))
        print('%d和%d的最小公倍数是%d' % (x, y, x * y // factor))
        break
```

## 练习3：打印三角形图案。

```
"""
打印各种三角形图案

*
**
***
****
```

```
*****  
  
*  
**  
***  
****  
*****  
  
*  
***  
****  
*****  
*****  
  
Version: 0.1  
Author: 骆昊  
***  
  
row = int(input('请输入行数: '))  
for i in range(row):  
    for _ in range(i + 1):  
        print('*', end='')  
    print()  
  
    for i in range(row):  
        for j in range(row):  
            if j < row - i - 1:  
                print(' ', end='')  
            else:  
                print('*', end='')  
        print()  
  
    for i in range(row):  
        for _ in range(row - i - 1):  
            print(' ', end='')  
        for _ in range(2 * i + 1):  
            print('*', end='')  
        print()
```

Branch: master ▾

[Find file](#) [Copy path](#)

## [Python-100-Days](#) / [Day01-15](#) / [Day05](#) / 总结和练习.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



9 lines (7 sloc) 485 Bytes

## 练习

### 练习清单

1. 寻找“[水仙花数](#)”。
2. 寻找“[完美数](#)”。
3. “[百钱百鸡](#)”问题。
4. 生成“[斐波拉切数列](#)”。
5. Craps赌博游戏。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day01-15 / Day06 / 函数和模块的使用.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



353 lines (247 sloc) 13.4 KB

## 函数和模块的使用

在讲解本章节的内容之前，我们先来研究一道数学题，请说出下面的方程有多少组正整数解。

$\$ \$ x_1 + x_2 + x_3 + x_4 = 8 \$ \$$

事实上，上面的问题等同于将8个苹果分成四组每组至少一个苹果有多少种方案。想到这一点问题的答案就呼之欲出了。

$\$ \$ C_{M^N} = \frac{M!}{N!(M-N)!}, \text{text}\{(M=7, N=3)\} \$ \$$

可以用Python的程序来计算出这个值，代码如下所示。

```
"""
输入M和N计算C(M,N)
"""

m = int(input('m = '))
n = int(input('n = '))
fm = 1
for num in range(1, m + 1):
    fm *= num
fn = 1
for num in range(1, n + 1):
    fn *= num
fmn = 1
for num in range(1, m - n + 1):
    fmn *= num
print(fm // fn // fmn)
```

## 函数的作用

不知道大家是否注意到，在上面的代码中，我们做了3次求阶乘，这样的代码实际上就是重复代码。编程大师Martin Fowler先生曾经说过：“代码有很多种坏味道，重复是最坏的一种！”，要写出高质量的代码首先要解决的就是重复代码的问题。对于上面的代码来说，我们可以将计算阶乘的功能封装到一个称之为“函数”的功能模块中，在需要计算阶乘的地方，我们只需要“调用”这个“函数”就可以了。

## 定义函数

在Python中可以使用 `def` 关键字来定义函数，和变量一样每个函数也有一个响亮的名字，而且命名规则跟变量的命名规则是一致的。在函数名后面的圆括号中可以放置传递给函数的参数，这一点和数学上的函数非常相似，程序中函数的参数就相当于是数学上说的函数的自变量，而函数执行完成后我们可以通过 `return` 关键字来返回一个值，这相当于数学上说的函数的因变量。

在了解了如何定义函数后，我们可以对上面的代码进行重构，所谓重构就是在不影响代码执行结果的前提下对代码的结构进行调整，重构之后的代码如下所示。

```
def factorial(num):
    """
    求阶乘

    :param num: 非负整数
    :return: num的阶乘
    """

    result = 1
    for n in range(1, num + 1):
        result *= n
    return result

m = int(input('m = '))
n = int(input('n = '))
# 当需要计算阶乘的时候不用再写循环求阶乘而是直接调用已经定义好的函数
print(factorial(m) // factorial(n) // factorial(m - n))
```

\*\*说明：\*\*Python的math模块中其实已经有一个`factorial`函数了，事实上要计算阶乘可以直接使用这个现成的函数而不用自己定义。下面例子中的某些函数其实Python中也是内置了，我们这里是为了讲解函数的定义和使用才把它们又实现了一遍，实际开发中不建议做这种低级的重复性的工作。

## 函数的参数

函数是绝大多数编程语言中都支持的一个代码的“构建块”，但是Python中的函数与其他语言中的函数还是有很多不太相同的地方，其中一个显著的区别就是Python对函数参数的处理。在Python中，函数的参数可以有默认值，也支持使用可变参数，所以Python并不需要像其他语言一样支持[函数的重载](#)，因为我们在定义一个函数的时候可以让它有多种不同的使用方式，下面是两个小例子。

```

from random import randint

def roll_dice(n=2):
    """
    摆色子

    :param n: 色子的个数
    :return: n颗色子点数之和
    """
    total = 0
    for _ in range(n):
        total += randint(1, 6)
    return total

def add(a=0, b=0, c=0):
    return a + b + c

# 如果没有指定参数那么使用默认值摇两颗色子
print(roll_dice())
# 摆三颗色子
print(roll_dice(3))
print(add())
print(add(1))
print(add(1, 2))
print(add(1, 2, 3))
# 传递参数时可以不按照设定的顺序进行传递
print(add(c=50, a=100, b=200))

```

我们给上面两个函数的参数都设定了默认值，这也就意味着如果在调用函数的时候如果没有传入对应参数的值时将使用该参数的默认值，所以在上面的代码中我们可以用各种不同的方式去调用 `add` 函数，这跟其他很多语言中函数重载的效果是一致的。

其实上面的 `add` 函数还有更好的实现方案，因为我们可能会对0个或多个参数进行加法运算，而具体有多少个参数是由调用者来决定，我们作为函数的设计者对这一点是一无所知的，因此在不确定参数个数的时候，我们可以使用可变参数，代码如下所示。

```

# 在参数名前面的*表示args是一个可变参数
# 即在调用add函数时可以传入0个或多个参数
def add(*args):
    total = 0
    for val in args:
        total += val
    return total

print(add())
print(add(1))
print(add(1, 2))

```

```
print(add(1, 2, 3))
print(add(1, 3, 5, 7, 9))
```

## 用模块管理函数

对于任何一种编程语言来说，给变量、函数这样的标识符起名字都是一个让人头疼的问题，因为我们会遇到命名冲突这种尴尬的情况。最简单的场景就是在同一个.py文件中定义了两个同名函数，由于Python没有函数重载的概念，那么后面的定义会覆盖之前的定义，也就意味着两个函数同名函数实际上只有一个存在的。

```
def foo():
    print('hello, world!')

def foo():
    print('goodbye, world!')

# 下面的代码会输出什么呢?
foo()
```

当然上面的这种情况我们很容易就能避免，但是如果项目是由多人协作进行团队开发的时候，团队中可能有多个程序员都定义了名为 `foo` 的函数，那么怎么解决这种命名冲突呢？答案其实很简单，Python中每个文件就代表了一个模块（`module`），我们在不同的模块中可以有同名的函数，在使用函数的时候我们通过 `import` 关键字导入指定的模块就可以区分到底要使用的是哪个模块中的 `foo` 函数，代码如下所示。

module1.py

```
def foo():
    print('hello, world!')
```

module2.py

```
def foo():
    print('goodbye, world!')
```

test.py

```
from module1 import foo

# 输出hello, world!
foo()

from module2 import foo
```

```
# 输出goodbye, world!
foo()
```

也可以按照如下所示的方式来区分到底要使用哪一个 `foo` 函数。

test.py

```
import module1 as m1
import module2 as m2

m1.foo()
m2.foo()
```

但是如果将代码写成了下面的样子，那么程序中调用的是最后导入的那个 `foo`，因为后导入的 `foo` 覆盖了之前导入的 `foo`。

test.py

```
from module1 import foo
from module2 import foo

# 输出goodbye, world!
foo()
```

test.py

```
from module2 import foo
from module1 import foo

# 输出hello, world!
foo()
```

需要说明的是，如果我们导入的模块除了定义函数之外还有可以执行代码，那么 Python 解释器在导入这个模块时就会执行这些代码，事实上我们可能并不希望如此，因此如果我们在模块中编写了执行代码，最好是将这些执行代码放入如下所示的条件中，这样的话除非直接运行该模块，`if` 条件下的这些代码是不会执行的，因为只有直接执行的模块的名字才是“`__main__`”。

module3.py

```
def foo():
    pass

def bar():
    pass
```

```
# __name__是Python中一个隐含的变量它代表了模块的名字  
# 只有被Python解释器直接执行的模块的名字才是__main__  
if __name__ == '__main__':  
    print('call foo()')  
    foo()  
    print('call bar()')  
    bar()
```

test.py

```
import module3  
  
# 导入module3时 不会执行模块中if条件成立时的代码 因为模块的名字是module3而不是__main__
```



## 练习

**练习1：实现计算求最大公约数和最小公倍数的函数。**

```
def gcd(x, y):  
    (x, y) = (y, x) if x > y else (x, y)  
    for factor in range(x, 0, -1):  
        if x % factor == 0 and y % factor == 0:  
            return factor  
  
def lcm(x, y):  
    return x * y // gcd(x, y)
```

**练习2：实现判断一个数是不是回文数的函数。**

```
def is_palindrome(num):  
    temp = num  
    total = 0  
    while temp > 0:  
        total = total * 10 + temp % 10  
        temp //= 10  
    return total == num
```

**练习3：实现判断一个数是不是素数的函数。**

```
def is_prime(num):  
    for factor in range(2, num):  
        if num % factor == 0:
```

```
        return False
return True if num != 1 else False
```

#### 练习4：写一个程序判断输入的正整数是不是回文素数。

```
if __name__ == '__main__':
    num = int(input('请输入正整数: '))
    if is_palindrome(num) and is_prime(num):
        print('%d是回文素数' % num)
```

通过上面的程序可以看出，当我们将代码中重复出现的和相对独立的功能抽取成函数后，我们可以组合使用这些函数来解决更为复杂的问题，这也是我们为什么要定义和使用函数的一个非常重要的原因。

最后，我们来讨论一下Python中有关变量作用域的问题。

```
def foo():
    b = 'hello'

    def bar(): # Python中可以在函数内部再定义函数
        c = True
        print(a)
        print(b)
        print(c)

    bar()
    # print(c) # NameError: name 'c' is not defined

if __name__ == '__main__':
    a = 100
    # print(b) # NameError: name 'b' is not defined
    foo()
```

上面的代码能够顺利的执行并且打印出100和“hello”，但我们注意到了，在 `bar` 函数的内部并没有定义 `a` 和 `b` 两个变量，那么 `a` 和 `b` 是从哪里来的。我们在上面代码的 `if` 分支中定义了一个变量 `a`，这是一个全局变量（global variable），属于全局作用域，因为它没有定义在任何一个函数中。在上面的 `foo` 函数中我们定义了变量 `b`，这是一个定义在函数中的局部变量（local variable），属于局部作用域，在 `foo` 函数的外部并不能访问到它；但对于 `foo` 函数内部的 `bar` 函数来说，变量 `b` 属于嵌套作用域，在 `bar` 函数中我们是可以访问到它的。`bar` 函数中的变量 `c` 属于局部作用域，在 `bar` 函数之外是无法访问的。事实上，Python查找一个变量时会按照“局部作用域”、“嵌套作用域”、“全局作用域”和“内置作用域”的顺序进行搜索，前三者我们在上面的代码中已经看到了，所谓的“内置作用域”就是Python内置的那些隐含标识符 `min`、`len` 等都属于内置作用域）。

再看看下面这段代码，我们希望通过函数调用修改全局变量 `a` 的值，但实际上下面的代码是做不到的。

```
def foo():
    a = 200
    print(a) # 200

if __name__ == '__main__':
    a = 100
    foo()
    print(a) # 100
```

在调用 `foo` 函数后，我们发现 `a` 的值仍然是 100，这是因为当我们在函数 `foo` 中写 `a = 200` 的时候，是重新定义了一个名字为 `a` 的局部变量，它跟全局作用域的 `a` 并不是同一个变量，因为局部作用域中有了自己的变量 `a`，因此 `foo` 函数不再搜索全局作用域中的 `a`。如果我们希望在 `foo` 函数中修改全局作用域中的 `a`，代码如下所示。

```
def foo():
    global a
    a = 200
    print(a) # 200

if __name__ == '__main__':
    a = 100
    foo()
    print(a) # 200
```

我们可以使用 `global` 关键字来指示 `foo` 函数中的变量 `a` 来自于全局作用域，如果全局作用域中没有 `a`，那么下面一行的代码就会定义变量 `a` 并将其置于全局作用域。同理，如果我们希望函数内部的函数能够修改嵌套作用域中的变量，可以使用 `nonlocal` 关键字来指示变量来自于嵌套作用域，请大家自行试验。

在实际开发中，我们应该尽量减少对全局变量的使用，因为全局变量的作用域和影响过于广泛，可能会发生意料之外的修改和使用，除此之外全局变量比局部变量拥有更长的生命周期，可能导致对象占用的内存长时间无法被[垃圾回收](#)。事实上，减少对全局变量的使用，也是降低代码之间耦合度的一个重要举措，同时也是对[迪米特法则](#)的践行。减少全局变量的使用就意味着我们应该尽量让变量的作用域在函数的内部，但是如果我们将一个局部变量的生命周期延长，使其在函数调用结束后依然可以访问，这时候就需要使用[闭包](#)，这个我们在后续的内容中进行讲解。

**说明：**很多人经常会将“闭包”一词和“[匿名函数](#)”混为一谈，但实际上它们是不同的概念，如果想提前了解这个概念，推荐看看[维基百科](#)或者[知乎](#)上对这个概念的讨论。

说了那么多，其实结论很简单，从现在开始我们可以将Python代码按照下面的格式进行书写，这一点点的改进其实就是在我们理解了函数和作用域的基础上跨出的巨大的一步。

```
def main():
    # Todo: Add your code here
    pass
```

```
if __name__ == '__main__':
    main()
```

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Day01-15 / Day07 / 字符串和常用数据结构.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

612 lines (496 sloc) 19.3 KB

# 字符串和常用数据结构

## 使用字符串

第二次世界大战促使了现代电子计算机的诞生，当初的想法很简单，就是用计算机来计算导弹的弹道，因此在计算机刚刚诞生的那个年代，计算机处理的信息主要是数值，而世界上的第一台电子计算机ENIAC每秒钟能够完成约5000次浮点运算。随着时间的推移，虽然对数值运算仍然是计算机日常工作中最为重要的事情之一，但是今天的计算机处理得更多的数据都是以文本信息的方式存在的，而Python表示文本信息的方式我们在很早以前就说过了，那就是字符串类型。所谓**字符串**，就是由零个或多个字符组成的有限序列，一般记为 $\text{str} = a_1a_2\ldots a_n (0 \leq n \leq \infty)$ 。

我们可以通过下面的代码来了解字符串的使用。

```
def main():
    str1 = 'hello, world!'
    # 通过len函数计算字符串的长度
    print(len(str1)) # 13
    # 获得字符串首字母大写的拷贝
    print(str1.capitalize()) # Hello, world!
    # 获得字符串变大写后的拷贝
    print(str1.upper()) # HELLO, WORLD!
    # 从字符串中查找子串所在位置
    print(str1.find('or')) # 8
    print(str1.find('shit')) # -1
    # 与find类似但找不到子串时会引发异常
    # print(str1.index('or'))
    # print(str1.index('shit'))
    # 检查字符串是否以指定的字符串开头
    print(str1.startswith('He')) # False
    print(str1.startswith('hel')) # True
    # 检查字符串是否以指定的字符串结尾
    print(str1.endswith('!')) # True
    # 将字符串以指定的宽度居中并在两侧填充指定的字符
```

```

print(str1.center(50, '*'))
# 将字符串以指定的宽度靠右放置左侧填充指定的字符
print(str1.rjust(50, ' '))
str2 = 'abc123456'
# 从字符串中取出指定位置的字符(下标运算)
print(str2[2]) # c
# 字符串切片(从指定的开始索引到指定的结束索引)
print(str2[2:5]) # c12
print(str2[2:]) # c123456
print(str2[2::2]) # c246
print(str2[::-2]) # ac246
print(str2[::-1]) # 654321cba
print(str2[-3:-1]) # 45
# 检查字符串是否由数字构成
print(str2.isdigit()) # False
# 检查字符串是否以字母构成
print(str2.isalpha()) # False
# 检查字符串是否以数字和字母构成
print(str2.isalnum()) # True
str3 = ' jackfrued@126.com '
print(str3)
# 获得字符串修剪左右两侧空格的拷贝
print(str3.strip())

```

```

if __name__ == '__main__':
    main()

```

除了字符串，Python还内置了多种类型的数据结构，如果要在程序中保存和操作数据，绝大多数时候可以利用现有的数据结构来实现，最常用的包括列表、元组、集合和字典。

## 使用列表

下面的代码演示了如何定义列表、使用下标访问列表元素以及添加和删除元素的操作。

```

def main():
    list1 = [1, 3, 5, 7, 100]
    print(list1)
    list2 = ['hello'] * 5
    print(list2)
    # 计算列表长度(元素个数)
    print(len(list1))
    # 下标(索引)运算
    print(list1[0])
    print(list1[4])
    # print(list1[5]) # IndexError: list index out of range
    print(list1[-1])
    print(list1[-3])
    list1[2] = 300
    print(list1)
    # 添加元素

```

```

list1.append(200)
list1.insert(1, 400)
list1 += [1000, 2000]
print(list1)
print(len(list1))
# 删除元素
list1.remove(3)
if 1234 in list1:
    list1.remove(1234)
del list1[0]
print(list1)
# 清空列表元素
list1.clear()
print(list1)

if __name__ == '__main__':
    main()

```

和字符串一样，列表也可以做切片操作，通过切片操作我们可以实现对列表的复制或者将列表中的一部分取出来创建出新的列表，代码如下所示。

```

def main():
    fruits = ['grape', 'apple', 'strawberry', 'waxberry']
    fruits += ['pitaya', 'pear', 'mango']
    # 循环遍历列表元素
    for fruit in fruits:
        print(fruit.title(), end=' ')
    print()
    # 列表切片
    fruits2 = fruits[1:4]
    print(fruits2)
    # fruit3 = fruits # 没有复制列表只创建了新的引用
    # 可以通过完整切片操作来复制列表
    fruits3 = fruits[:]
    print(fruits3)
    fruits4 = fruits[-3:-1]
    print(fruits4)
    # 可以通过反向切片操作来获得倒转后的列表的拷贝
    fruits5 = fruits[::-1]
    print(fruits5)

if __name__ == '__main__':
    main()

```

下面的代码实现了对列表的排序操作。

```

def main():
    list1 = ['orange', 'apple', 'zoo', 'internationalization', 'blueberry']
    list2 = sorted(list1)

```

```

# sorted函数返回列表排序后的拷贝不会修改传入的列表
# 函数的设计就应该像sorted函数一样尽可能不产生副作用
list3 = sorted(list1, reverse=True)
# 通过key关键字参数指定根据字符串长度进行排序而不是默认的字母表顺序
list4 = sorted(list1, key=len)
print(list1)
print(list2)
print(list3)
print(list4)
# 给列表对象发出排序消息直接在列表对象上进行排序
list1.sort(reverse=True)
print(list1)

```

```

if __name__ == '__main__':
    main()

```

我们还可以使用列表的生成式语法来创建列表，代码如下所示。

```

import sys

def main():
    f = [x for x in range(1, 10)]
    print(f)
    f = [x + y for x in 'ABCDE' for y in '1234567']
    print(f)
    # 用列表的生成表达式语法创建列表容器
    # 用这种语法创建列表之后元素已经准备就绪所以需要耗费较多的内存空间
    f = [x ** 2 for x in range(1, 1000)]
    print(sys.getsizeof(f)) # 查看对象占用内存的字节数
    print(f)
    # 请注意下面的代码创建的不是一个列表而是一个生成器对象
    # 通过生成器可以获取到数据但它不占用额外的空间存储数据
    # 每次需要数据的时候就通过内部的运算得到数据(需要花费额外的时间)
    f = (x ** 2 for x in range(1, 1000))
    print(sys.getsizeof(f)) # 相比生成式生成器不占用存储数据的空间
    print(f)
    for val in f:
        print(val)

if __name__ == '__main__':
    main()

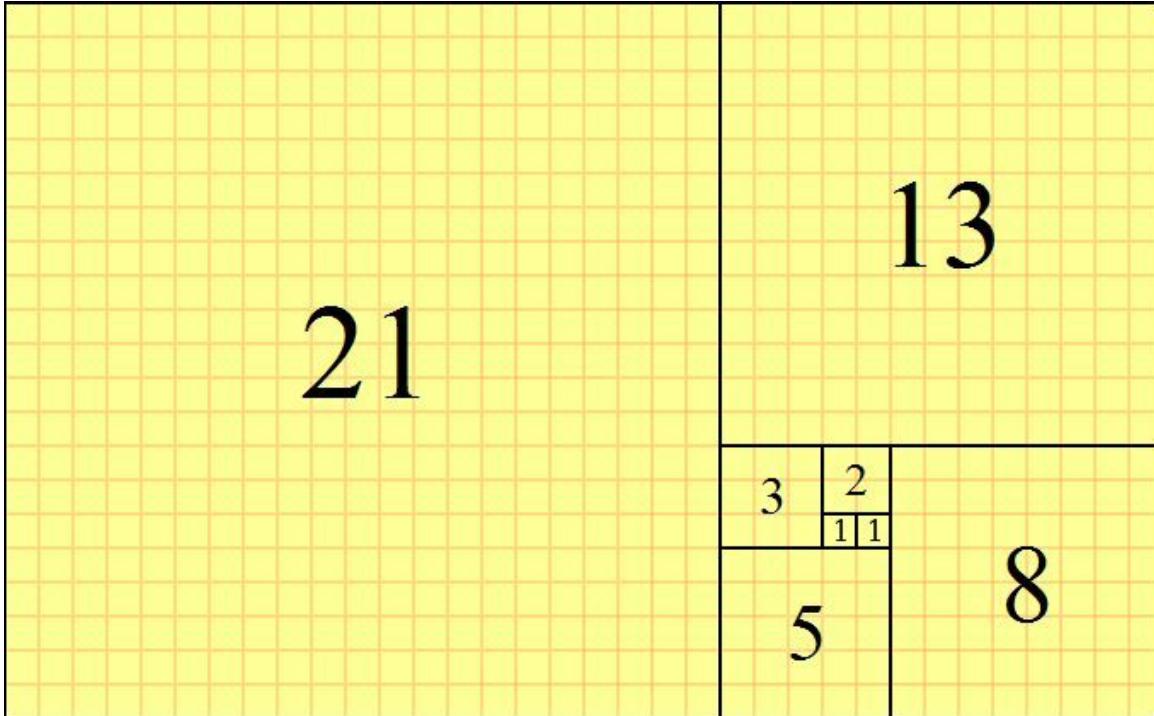
```

除了上面提到的生成器语法，Python中还有另外一种定义生成器的方式，就是通过 `yield` 关键字将一个普通函数改造成生成器函数。下面的代码演示了如何实现一个生成 `斐波拉切数列` 的生成器。所谓斐波拉切数列可以通过下面 `递归` 的方法来进行定义：

$$F_0=0$$

$\$ \$ \{\backslash displaystyle F_{\{1\}}=1\} \$ \$$

$\$ \$ \{\backslash displaystyle F_{\{n\}}=F_{\{n-1\}}+F_{\{n-2\}}\} \{n\} \geq \{2\} \$ \$$



```
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    yield a

def main():
    for val in fib(20):
        print(val)

if __name__ == '__main__':
    main()
```

## 使用元组

Python 的元组与列表类似，不同之处在于元组的元素不能修改，在前面的代码中我们已经不止一次使用过元组了。顾名思义，我们把多个元素组合到一起就形成了一个元组，所以它和列表一样可以保存多条数据。下面的代码演示了如何定义和使用元组。

```
def main():
    # 定义元组
    t = ('骆昊', 38, True, '四川成都')
    print(t)
```

```

# 获取元组中的元素
print(t[0])
print(t[3])
# 遍历元组中的值
for member in t:
    print(member)
# 重新给元组赋值
# t[0] = '王大锤' # TypeError
# 变量t重新引用了新的元组原来的元组将被垃圾回收
t = ('王大锤', 20, True, '云南昆明')
print(t)
# 将元组转换成列表
person = list(t)
print(person)
# 列表是可以修改它的元素的
person[0] = '李小龙'
person[1] = 25
print(person)
# 将列表转换成元组
fruits_list = ['apple', 'banana', 'orange']
fruits_tuple = tuple(fruits_list)
print(fruits_tuple)

if __name__ == '__main__':
    main()

```

这里有一个非常值得探讨的问题，我们已经有了列表这种数据结构，为什么还需要元组这样的类型呢？

1. 元组中的元素是无法修改的，事实上我们在项目中尤其是多线程环境（后面会讲到）中可能更喜欢使用的是那些不变对象（一方面因为对象状态不能修改，所以可以避免由此引起的不必要的程序错误，简单的说就是一个不变的对象要比可变的对象更加容易维护；另一方面因为没有任何一个线程能够修改不变对象的内部状态，一个不变对象自动就是线程安全的，这样就可以省掉处理同步化的开销。一个不变对象可以方便的被共享访问）。所以结论就是：如果不需要对元素进行添加、删除、修改的时候，可以考虑使用元组，当然如果一个方法要返回多个值，使用元组也是不错的选择。
2. 元组在创建时间和占用的空间上面都优于列表。我们可以使用sys模块的getsizeof函数来检查存储同样的元素的元组和列表各自占用了多少内存空间，这个很容易做到。我们也可以在ipython中使用魔法指令%timeit来分析创建同样内容的元组和列表所花费的时间，下图是我的macOS系统上测试的结果。

```
Hao — IPython: Users/Hao — ipython — 66x21
Last login: Thu Mar  8 22:17:13 on console
jackfrued:~ Hao$ ipython
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help
.

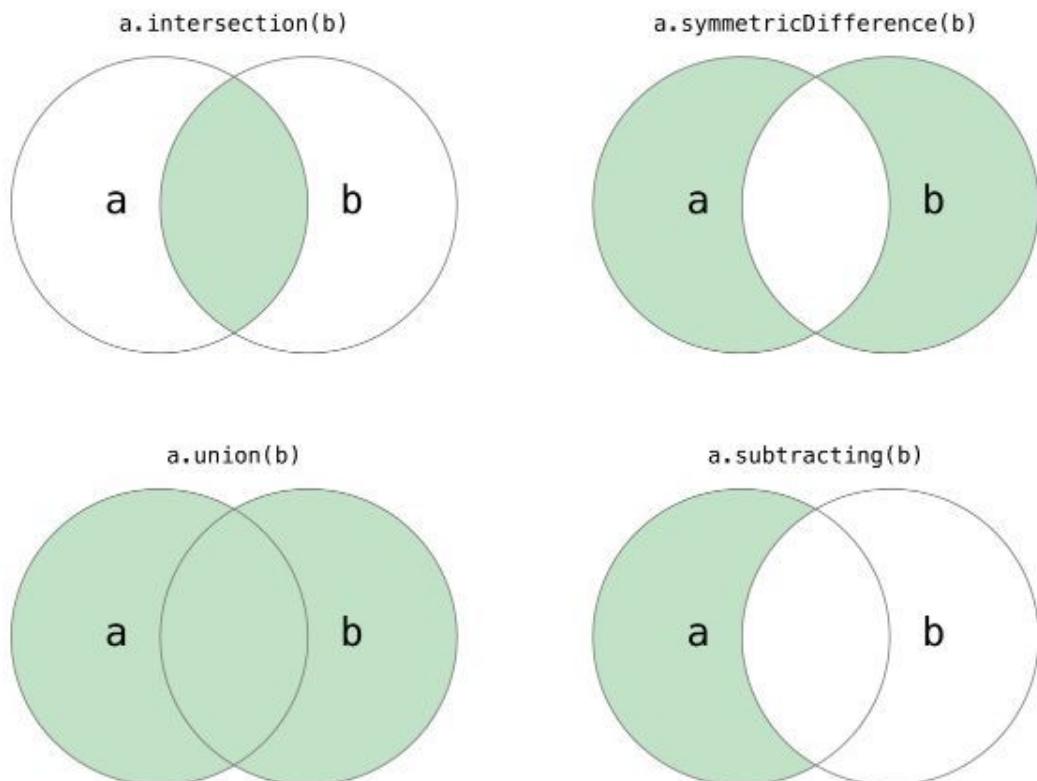
In [1]: %timeit [1, 2, 3, 4, 5]
73.3 ns ± 0.445 ns per loop (mean ± std. dev. of 7 runs, 100000000
loops each)

In [2]: %timeit (1, 2, 3, 4, 5)
16.6 ns ± 0.491 ns per loop (mean ± std. dev. of 7 runs, 1000000000
loops each)

In [3]: []
```

## 使用集合

Python中的集合跟数学上的集合是一致的，不允许有重复元素，而且可以进行交集、并集、差集等运算。



```
def main():
    set1 = {1, 2, 3, 3, 3, 2}
```

```

print(set1)
print('Length =', len(set1))
set2 = set(range(1, 10))
print(set2)
set1.add(4)
set1.add(5)
set2.update([11, 12])
print(set1)
print(set2)
set2.discard(5)
# remove的元素如果不存在会引发KeyError
if 4 in set2:
    set2.remove(4)
print(set2)
# 遍历集合容器
for elem in set2:
    print(elem ** 2, end=' ')
print()
# 将元组转换成集合
set3 = set((1, 2, 3, 3, 2, 1))
print(set3.pop())
print(set3)
# 集合的交集、并集、差集、对称差运算
print(set1 & set2)
# print(set1.intersection(set2))
print(set1 | set2)
# print(set1.union(set2))
print(set1 - set2)
# print(set1.difference(set2))
print(set1 ^ set2)
# print(set1.symmetric_difference(set2))
# 判断子集和超集
print(set2 <= set1)
# print(set2.issubset(set1))
print(set3 <= set1)
# print(set3.issubset(set1))
print(set1 >= set2)
# print(set1.issuperset(set2))
print(set1 >= set3)
# print(set1.issuperset(set3))

if __name__ == '__main__':
    main()

```

**说明**：Python中允许通过一些特殊的方法来为某种类型或数据结构自定义运算符（后面的章节中会讲到），上面的代码中我们对集合进行运算的时候可以调用集合对象的方法，也可以直接使用对应的运算符，例如 & 运算符跟intersection方法的作用就是一样的，但是使用运算符让代码更加直观。

## 使用字典

字典是另一种可变容器模型，类似于我们生活中使用的字典，它可以存储任意类型对象，与列表、集合不同的是，字典的每个元素都是由一个键和一个值组成的“键值对”，键和值通过冒号分开。下面的代码演示了如何定义和使用字典。

```
def main():
    scores = {'骆昊': 95, '白元芳': 78, '狄仁杰': 82}
    # 通过键可以获取字典中对应的值
    print(scores['骆昊'])
    print(scores['狄仁杰'])
    # 对字典进行遍历(遍历的其实是键再通过键取对应的值)
    for elem in scores:
        print('%s\t-->\t%d' % (elem, scores[elem]))
    # 更新字典中的元素
    scores['白元芳'] = 65
    scores['诸葛王朗'] = 71
    scores.update(冷面=67, 方启鹤=85)
    print(scores)
    if '武则天' in scores:
        print(scores['武则天'])
    print(scores.get('武则天'))
    # get方法也是通过键获取对应的值但是可以设置默认值
    print(scores.get('武则天', 60))
    # 删除字典中的元素
    print(scores.popitem())
    print(scores.popitem())
    print(scores.pop('骆昊', 100))
    # 清空字典
    scores.clear()
    print(scores)

if __name__ == '__main__':
    main()
```

## 练习

### 练习1：在屏幕上显示跑马灯文字

```
import os
import time

def main():
    content = '北京欢迎你为你开天辟地.....'
    while True:
        # 清理屏幕上的输出
        os.system('cls') # os.system('clear')
        print(content)
        # 休眠200毫秒
        time.sleep(0.2)
        content = content[1:] + content[0]
```

```
if __name__ == '__main__':
    main()
```

## 练习2：设计一个函数产生指定长度的验证码，验证码由大小写字母和数字构成。

```
import random

def generate_code(code_len=4):
    """
    生成指定长度的验证码

    :param code_len: 验证码的长度(默认4个字符)

    :return: 由大小写英文字母和数字构成的随机验证码
    """
    all_chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    last_pos = len(all_chars) - 1
    code = ''
    for _ in range(code_len):
        index = random.randint(0, last_pos)
        code += all_chars[index]
    return code
```

## 练习3：设计一个函数返回给定文件名的后缀名。

```
def get_suffix(filename, has_dot=False):
    """
    获取文件名的后缀名

    :param filename: 文件名
    :param has_dot: 返回的后缀名是否需要带点
    :return: 文件的后缀名
    """

    pos = filename.rfind('.')
    if 0 < pos < len(filename) - 1:
        index = pos if has_dot else pos + 1
        return filename[index:]
    else:
        return ''
```

## 练习4：设计一个函数返回传入的列表中最大和第二大的元素的值。

```
def max2(x):
    m1, m2 = (x[0], x[1]) if x[0] > x[1] else (x[1], x[0])
    for index in range(2, len(x)):
        if x[index] > m1:
```

```

    m2 = m1
    m1 = x[index]
    elif x[index] > m2:
        m2 = x[index]
return m1, m2

```

## 练习5：计算指定的年月日是这一年的第几天

```

def is_leap_year(year):
"""
判断指定的年份是不是闰年

:param year: 年份
:return: 闰年返回True平年返回False
"""

return year % 4 == 0 and year % 100 != 0 or year % 400 == 0


def which_day(year, month, date):
"""
计算传入的日期是这一年的第几天

:param year: 年
:param month: 月
:param date: 日
:return: 第几天
"""

days_of_month = [
    [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31],
    [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
][is_leap_year(year)]
total = 0
for index in range(month - 1):
    total += days_of_month[index]
return total + date


def main():
print(which_day(1980, 11, 28))
print(which_day(1981, 12, 31))
print(which_day(2018, 1, 1))
print(which_day(2016, 3, 1))

if __name__ == '__main__':
    main()

```

## 练习6：打印杨辉三角。

```

def main():
    num = int(input('Number of rows: '))

```

```

yh = [[]] * num
for row in range(len(yh)):
    yh[row] = [None] * (row + 1)
    for col in range(len(yh[row])):
        if col == 0 or col == row:
            yh[row][col] = 1
        else:
            yh[row][col] = yh[row - 1][col] + yh[row - 1][col - 1]
    print(yh[row][col], end='\t')
print()

```

```

if __name__ == '__main__':
    main()

```

## 综合案例

### 案例1：双色球选号

```

from random import randrange, randint, sample

def display(balls):
    """
    输出列表中的双色球号码
    """
    for index, ball in enumerate(balls):
        if index == len(balls) - 1:
            print('|', end=' ')
        print('%02d' % ball, end=' ')
    print()

def random_select():
    """
    随机选择一组号码
    """
    red_balls = [x for x in range(1, 34)]
    selected_balls = []
    selected_balls = sample(red_balls, 6)
    selected_balls.sort()
    selected_balls.append(randint(1, 16))
    return selected_balls


def main():
    n = int(input('机选几注： '))
    for _ in range(n):
        display(random_select())

```

```
if __name__ == '__main__':
    main()
```

**说明**：上面使用random模块的sample函数来实现从列表中选择不重复的n个元素。

## 综合案例2：约瑟夫环问题

```
"""
《幸运的基督徒》
有15个基督徒和15个非基督徒在海上遇险，为了能让一部分人活下来不得不将其中15个人扔到海里
"""

def main():
    persons = [True] * 30
    counter, index, number = 0, 0, 0
    while counter < 15:
        if persons[index]:
            number += 1
            if number == 9:
                persons[index] = False
                counter += 1
                number = 0
        index += 1
        index %= 30
    for person in persons:
        print('基' if person else '非', end='')
```

```
if __name__ == '__main__':
    main()
```

## 综合案例3：井字棋游戏

```
import os

def print_board(board):
    print(board['TL'] + ' | ' + board['TM'] + ' | ' + board['TR'])
    print('---+---+---')
    print(board['ML'] + ' | ' + board['MM'] + ' | ' + board['MR'])
    print('---+---+---')
    print(board['BL'] + ' | ' + board['BM'] + ' | ' + board['BR'])

def main():
    init_board = {
        'TL': ' ', 'TM': ' ', 'TR': ' ',
        'ML': ' ', 'MM': ' ', 'MR': ' ',
        'BL': ' ', 'BM': ' ', 'BR': ' '
    }
```

```
}

begin = True
while begin:
    curr_board = init_board.copy()
    begin = False
    turn = 'x'
    counter = 0
    os.system('clear')
    print_board(curr_board)
    while counter < 9:
        move = input('轮到%s走棋, 请输入位置: ' % turn)
        if curr_board[move] == ' ':
            counter += 1
            curr_board[move] = turn
            if turn == 'x':
                turn = 'o'
            else:
                turn = 'x'
        os.system('clear')
        print_board(curr_board)
    choice = input('再玩一局?(yes|no)')
    begin = choice == 'yes'

if __name__ == '__main__':
    main()
```

**说明**：最后这个案例来自《Python编程快速上手:让繁琐工作自动化》一书（这本书对有编程基础想迅速使用Python将日常工作自动化的人来说还是不错的选择），对代码做了一点点的调整。

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Day01-15 / Day08 / 面向对象编程基础.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



240 lines (168 sloc) 11 KB

# 面向对象编程基础

活在当下的程序员应该都听过“面向对象编程”一词，也经常有人问能不能用一句话解释下什么是“面向对象编程”，我们先来看看比较正式的说法。

把一组数据结构和处理它们的方法组成对象（object），把相同行为的对象归纳为类（class），通过类的封装（encapsulation）隐藏内部细节，通过继承（inheritance）实现类的特化（specialization）和泛化（generalization），通过多态（polymorphism）实现基于对象类型的动态分派。

这样一是是不是更不明白了。所以我们还是看看更通俗易懂的说法，下面这段内容来自[知乎](#)。



成心文

不会拍电影的程序员不是好机长

51 人赞同了该回答

一句话说明什么是面向对象？你个土鳖，你们全家都是土鳖！

好像有人说过这样的话，当头棒喝的方式虽然情感上不易接受，但记忆效果十分显著。

好吧，如果你觉得“土鳖”实在难听也不能准确定位你的档次，你可以自行将其替换为“土豪”，whatever。

面向对象思想有三大要素：封装、继承和多态。

- 封装：不管你是土鳖还是土豪，不管你中午吃的是窝头还是鲍鱼，你的下水都在你肚皮里，别人看不到你中午吃了啥，除非你自己说给他们听（或者画给他们看，whatever）；
- 继承：刚说了，你个土鳖/豪，你们全家都是土鳖/豪。冰冻三尺非一日之寒，你有今天，必定可以从你爸爸妈妈那里追根溯源。正所谓虎父无犬子，正恩同学那么狠，他爹正日就不是什么善茬，更甭说他爷爷日成，明白了吗？
- 多态：哲学家说过，世上不会有两个一模一样的双胞胎。即使你从你父亲那里继承来的土鳖/豪气质，也不可能完全是从一个模子里刻出来的，总会有些差别。比如你爸喜欢蹲在门前吃面，你喜欢骑在村口的歪脖子树上吃，或者反过来。当然，也可能令尊爱吃小龙虾时旁边有几个

**说明：**以上的内容来自于网络，不代表作者本人的观点和看法，与作者本人立场无关，相关责任不由作者承担。

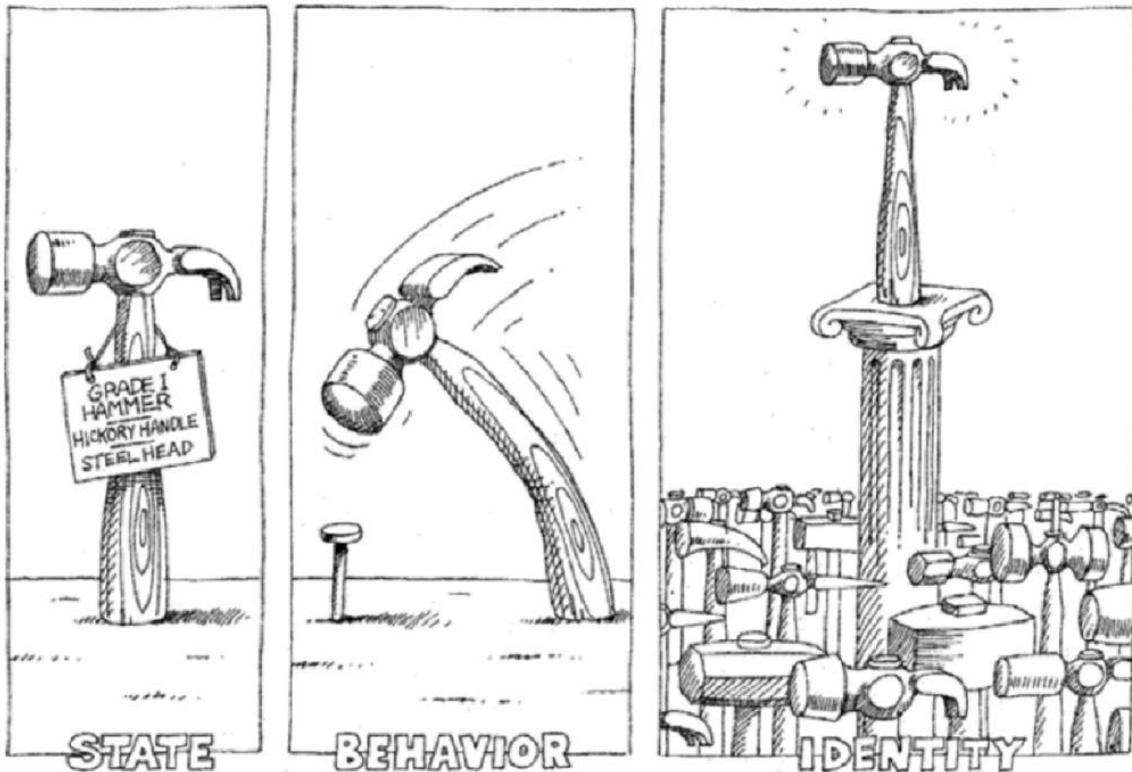
之前我们说过“程序是指令的集合”，我们在程序中书写的语句在执行时会变成一条或多条指令然后由CPU去执行。当然为了简化程序的设计，我们引入了函数的概念，把相对独立且经常重复使用的代码放置到函数中，在需要使用这些功能的时候只要调用函数即可；如果一个函数的功能过于复杂和臃肿，我们又可以进一步将函数继续切分为子函数来降低系统的复杂性。但是说了这么多，不知道大家是否发现，所谓编程就是程序员按照计算机的工作方式控制计算机完成各种任务。但是，计算机的工作方式与正常人类的思维模式是不同的，如果编程就必须得抛弃人类正常的思维方式去迎合计算机，编程的乐趣就少了很多，“每个人都应该学习编程”这样的豪言壮语就只能说说而已。当然，这些还不是最重要的，最重要的是当我们需要开发一个复杂的系统时，代码的复杂性会让开发和维护工作都变得举步维艰，所以在上世纪60年代末期，“[软件危机](#)”、“[软件工程](#)”等一系列的概念开始在行业中出现。

当然，程序员圈子内的人都知道，现实中并没有解决上面所说的这些问题的“[银弹](#)”，真正让软件开发者看到希望的是上世纪70年代诞生的[Smalltalk](#)编程语言中引入的面向对象的编程思想（面向对象编程的雏形可以追溯到更早期的[Simula](#)语言）。按照这种编程理念，程序中的数据和操作数据的函数是一个逻辑上的整体，我们称之为“对象”，而我们解决问题的方式就是创建出需要的对象并向对象发出各种各样的消息，多个对象的协同工作最终可以让我们构造出复杂的系统来解决现实中的问题。

**说明：**当然面向对象也不是解决软件开发中所有问题的最后的“银弹”，所以今天的高级程序设计语言几乎都提供了对多种编程范式的支持，Python也不例外。

## 类和对象

简单的说，类是对象的蓝图和模板，而对象是类的实例。这个解释虽然有点像用概念在解释概念，但是从这句话我们至少可以看出，类是抽象的概念，而对象是具体的东西。在面向对象编程的世界中，一切皆为对象，对象都有属性和行为，每个对象都是独一无二的，而且对象一定属于某个类（型）。当我们把一大堆拥有共同特征的对象的静态特征（属性）和动态特征（行为）都抽取出来后，就可以定义出一个叫做“类”的东西。



## 定义类

在Python中可以使用 `class` 关键字定义类，然后在类中通过之前学习过的函数来定义方法，这样就可以将对象的动态特征描述出来，代码如下所示。

```
class Student(object):

    # __init__是一个特殊方法用于在创建对象时进行初始化操作
    # 通过这个方法我们可以为学生对象绑定name和age两个属性
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def study(self, course_name):
        print('%s正在学习%s.' % (self.name, course_name))

    # PEP 8要求标识符的名字用全小写多个单词用下划线连接
    # 但是很多程序员和公司更倾向于使用驼峰命名法(驼峰标识)
    def watch_av(self):
        if self.age < 18:
```

```
    print('%s只能观看《熊出没》.' % self.name)
else:
    print('%s正在观看岛国爱情动作片.' % self.name)
```

**说明**：写在类中的函数，我们通常称之为（对象的）方法，这些方法就是对象可以接收的消息。

## 创建和使用对象

当我们定义好一个类之后，可以通过下面的方式来创建对象并给对象发消息。

```
def main():
    # 创建学生对象并指定姓名和年龄
    stu1 = Student('骆昊', 38)
    # 给对象发study消息
    stu1.study('Python程序设计')
    # 给对象发watch_av消息
    stu1.watch_av()
    stu2 = Student('王大锤', 15)
    stu2.study('思想品德')
    stu2.watch_av()

if __name__ == '__main__':
    main()
```

## 访问可见性问题

对于上面的代码，有C++、Java、C#等编程经验的程序员可能会问，我们给 `Student` 对象绑定的 `name` 和 `age` 属性到底具有怎样的访问权限（也称为可见性）。因为在很多面向对象编程语言中，我们通常会将对象的属性设置为私有的（`private`）或受保护的（`protected`），简单的说就是不允许外界访问，而对象的方法通常都是公开的（`public`），因为公开的方法就是对象能够接受的消息。在Python中，属性和方法的访问权限只有两种，也就是公开的和私有的，如果希望属性是私有的，在给属性命名时可以用两个下划线作为开头，下面的代码可以验证这一点。

```
class Test:

    def __init__(self, foo):
        self.__foo = foo

    def __bar(self):
        print(self.__foo)
        print('__bar')

def main():
    test = Test('hello')
```

```

# AttributeError: 'Test' object has no attribute '__bar'
test.__bar()
# AttributeError: 'Test' object has no attribute '__foo'
print(test.__foo)

if __name__ == "__main__":
    main()

```

但是，Python并没有从语法上严格保证私有属性或方法的私密性，它只是给私有的属性和方法换了一个名字来“妨碍”对它们的访问，事实上如果你知道更换名字的规则仍然可以访问到它们，下面的代码就可以验证这一点。之所以这样设定，可以用这样一句名言加以解释，就是“We are all consenting adults here”。因为绝大多数程序员都认为开放比封闭要好，而且程序员要自己为自己的行为负责。

```

class Test:

    def __init__(self, foo):
        self.__foo = foo

    def __bar(self):
        print(self.__foo)
        print('__bar')

def main():
    test = Test('hello')
    test.__Test__bar()
    print(test.__Test__foo)

if __name__ == "__main__":
    main()

```

在实际开发中，我们并不建议将属性设置为私有的，因为这会导致子类无法访问（后面会讲到）。所以大多数Python程序员会遵循一种命名惯例就是让属性名以单下划线开头来表示属性是受保护的，本类之外的代码在访问这样的属性时应该要保持慎重。这种做法并不是语法上的规则，单下划线开头的属性和方法外界仍然是可以访问的，所以更多的时候它是一种暗示或隐喻，关于这一点可以看看我的[《Python - 那些年我们踩过的那些坑》](#)文章中的讲解。

## 面向对象的支柱

面向对象有三大支柱：封装、继承和多态。后面两个概念在下一个章节中进行详细的说明，这里我们先说一下什么是封装。我自己对封装的理解是“隐藏一切可以隐藏的实现细节，只向外界暴露（提供）简单的编程接口”。我们在类中定义的方法其实就是把数据和对数据的操作封装起来了，在我们创建了对象之后，只需要给对象发送一个消息（调用方法）就可以执行方法中的代码，也就是说我们只需要知道方法的名字和传入的参数（方法的外部视图），而不需要知道方法内部的实现细节（方法的内部视图）。

## 练习

### 练习1：定义一个类描述数字时钟

```
class Clock(object):
    """数字时钟"""

    def __init__(self, hour=0, minute=0, second=0):
        """初始化方法

        :param hour: 时
        :param minute: 分
        :param second: 秒
        """

        self._hour = hour
        self._minute = minute
        self._second = second

    def run(self):
        """走字"""
        self._second += 1
        if self._second == 60:
            self._second = 0
            self._minute += 1
            if self._minute == 60:
                self._minute = 0
                self._hour += 1
                if self._hour == 24:
                    self._hour = 0

    def show(self):
        """显示时间"""
        return '%02d:%02d:%02d' % \
            (self._hour, self._minute, self._second)

def main():
    clock = Clock(23, 59, 58)
    while True:
        print(clock.show())
        sleep(1)
        clock.run()
```

```
if __name__ == '__main__':
    main()
```

## 练习2：定义一个类描述平面上的点并提供移动点和计算到另一个点距离的方法。

```
from math import sqrt

class Point(object):

    def __init__(self, x=0, y=0):
        """初始化方法

        :param x: 横坐标
        :param y: 纵坐标
        """
        self.x = x
        self.y = y

    def move_to(self, x, y):
        """移动到指定位置

        :param x: 新的横坐标
        :param y: 新的纵坐标
        """
        self.x = x
        self.y = y

    def move_by(self, dx, dy):
        """移动指定的增量

        :param dx: 横坐标的增量
        :param dy: 纵坐标的增量
        """
        self.x += dx
        self.y += dy

    def distance_to(self, other):
        """计算与另一个点的距离

        :param other: 另一个点
        """
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx ** 2 + dy ** 2)

    def __str__(self):
        return '(%s, %s)' % (str(self.x), str(self.y))

def main():
    p1 = Point(3, 5)
    p2 = Point()
```

```
print(p1)
print(p2)
p2.move_by(-1, 2)
print(p2)
print(p1.distance_to(p2))

if __name__ == '__main__':
    main()
```

**说明**：本章中的插图来自于Grady Booch等著作的《面向对象分析与设计》一书，该书是讲解面向对象编程的经典著作，有兴趣的读者可以购买和阅读这本书来了解更多的面向对象的相关知识。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day01-15 / Day09 / 面向对象进阶.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



745 lines (544 sloc) 21.9 KB

# 面向对象进阶

在前面的章节我们已经了解了面向对象的入门知识，知道了如何定义类，如何创建对象以及如何给对象发消息。为了能够更好的使用面向对象编程思想进行程序开发，我们还需要对Python中的面向对象编程进行更为深入的了解。

## @property装饰器

之前我们讨论过Python中属性和方法访问权限的问题，虽然我们不建议将属性设置为私有的，但是如果直接将属性暴露给外界也是有问题的，比如我们没有办法检查赋给属性的值是否有效。我们之前的建议是将属性命名以单下划线开头，通过这种方式来暗示属性是受保护的，不建议外界直接访问，那么如果想访问属性可以通过属性的getter（访问器）和setter（修改器）方法进行对应的操作。如果要做到这点，就可以考虑使用@property包装器来包装getter和setter方法，使得对属性的访问既安全又方便，代码如下所示。

```
class Person(object):

    def __init__(self, name, age):
        self._name = name
        self._age = age

    # 访问器 - getter方法
    @property
    def name(self):
        return self._name

    # 访问器 - getter方法
    @property
    def age(self):
        return self._age

    # 修改器 - setter方法
    @age.setter
```

```

def age(self, age):
    self._age = age

def play(self):
    if self._age <= 16:
        print('%s正在玩飞行棋.' % self._name)
    else:
        print('%s正在玩斗地主.' % self._name)

def main():
    person = Person('王大锤', 12)
    person.play()
    person.age = 22
    person.play()
    # person.name = '白元芳' # AttributeError: can't set attribute

if __name__ == '__main__':
    main()

```

## slots魔法

我们讲到这里，不知道大家是否已经意识到，Python是一门动态语言。通常，动态语言允许我们在程序运行时给对象绑定新的属性或方法，当然也可以对已经绑定的属性和方法进行解绑定。但是如果我们需要限定自定义类型的对象只能绑定某些属性，可以通过在类中定义`__slots__`变量来进行限定。需要注意的是`__slots__`的限定只对当前类的对象生效，对子类并不起任何作用。

```

class Person(object):

    # 限定Person对象只能绑定_name, _age和_gender属性
    __slots__ = ('_name', '_age', '_gender')

    def __init__(self, name, age):
        self._name = name
        self._age = age

    @property
    def name(self):
        return self._name

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, age):
        self._age = age

    def play(self):

```

```

if self._age <= 16:
    print('%s正在玩飞行棋.' % self._name)
else:
    print('%s正在玩斗地主.' % self._name)

def main():
    person = Person('王大锤', 22)
    person.play()
    person._gender = '男'
    # AttributeError: 'Person' object has no attribute '_is_gay'
    # person._is_gay = True

```

## 静态方法和类方法

之前，我们在类中定义的方法都是对象方法，也就是说这些方法都是发送给对象的消息。实际上，我们写在类中的方法并不需要都是对象方法，例如我们定义一个“三角形”类，通过传入三条边长来构造三角形，并提供计算周长和面积的方法，但是传入的三条边长未必能构造出三角形对象，因此我们可以先写一个方法来验证三条边长是否可以构成三角形，这个方法很显然就不是对象方法，因为在调用这个方法时三角形对象尚未创建出来（因为都不知道三条边能不能构成三角形），所以这个方法是属于三角形类而并不属于三角形对象的。我们可以使用静态方法来解决这类问题，代码如下所示。

```

from math import sqrt

class Triangle(object):

    def __init__(self, a, b, c):
        self._a = a
        self._b = b
        self._c = c

    @staticmethod
    def is_valid(a, b, c):
        return a + b > c and b + c > a and a + c > b

    def perimeter(self):
        return self._a + self._b + self._c

    def area(self):
        half = self.perimeter() / 2
        return sqrt(half * (half - self._a) *
                   (half - self._b) * (half - self._c))

def main():
    a, b, c = 3, 4, 5
    # 静态方法和类方法都是通过给类发消息来调用的
    if Triangle.is_valid(a, b, c):
        t = Triangle(a, b, c)

```

```

print(t.perimeter())
# 也可以通过给类发消息来调用对象方法但是要传入接收消息的对象作为参数
# print(Triangle.perimeter(t))
print(t.area())
# print(Triangle.area(t))
else:
    print('无法构成三角形。')

if __name__ == '__main__':
    main()

```

和静态方法比较类似，Python还可以在类中定义类方法，类方法的第一个参数约定名为 `cls`，它代表的是当前类相关的信息的对象（类本身也是一个对象，有的地方也称之为类的元数据对象），通过这个参数我们可以获取和类相关的信息并且可以创建出类的对象，代码如下所示。

```

from time import time, localtime, sleep

class Clock(object):
    """数字时钟"""

    def __init__(self, hour=0, minute=0, second=0):
        self._hour = hour
        self._minute = minute
        self._second = second

    @classmethod
    def now(cls):
        ctime = localtime(time())
        return cls(ctime.tm_hour, ctime.tm_min, ctime.tm_sec)

    def run(self):
        """走字"""
        self._second += 1
        if self._second == 60:
            self._second = 0
            self._minute += 1
            if self._minute == 60:
                self._minute = 0
                self._hour += 1
                if self._hour == 24:
                    self._hour = 0

    def show(self):
        """显示时间"""
        return '%02d:%02d:%02d' % \
               (self._hour, self._minute, self._second)

def main():

```

```
# 通过类方法创建对象并获取系统时间
clock = Clock.now()
while True:
    print(clock.show())
    sleep(1)
    clock.run()

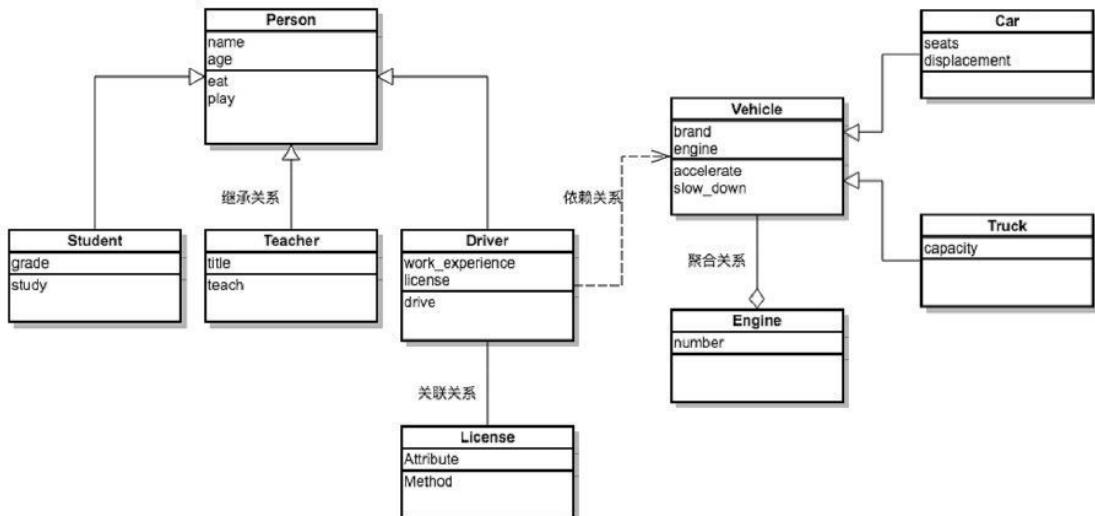
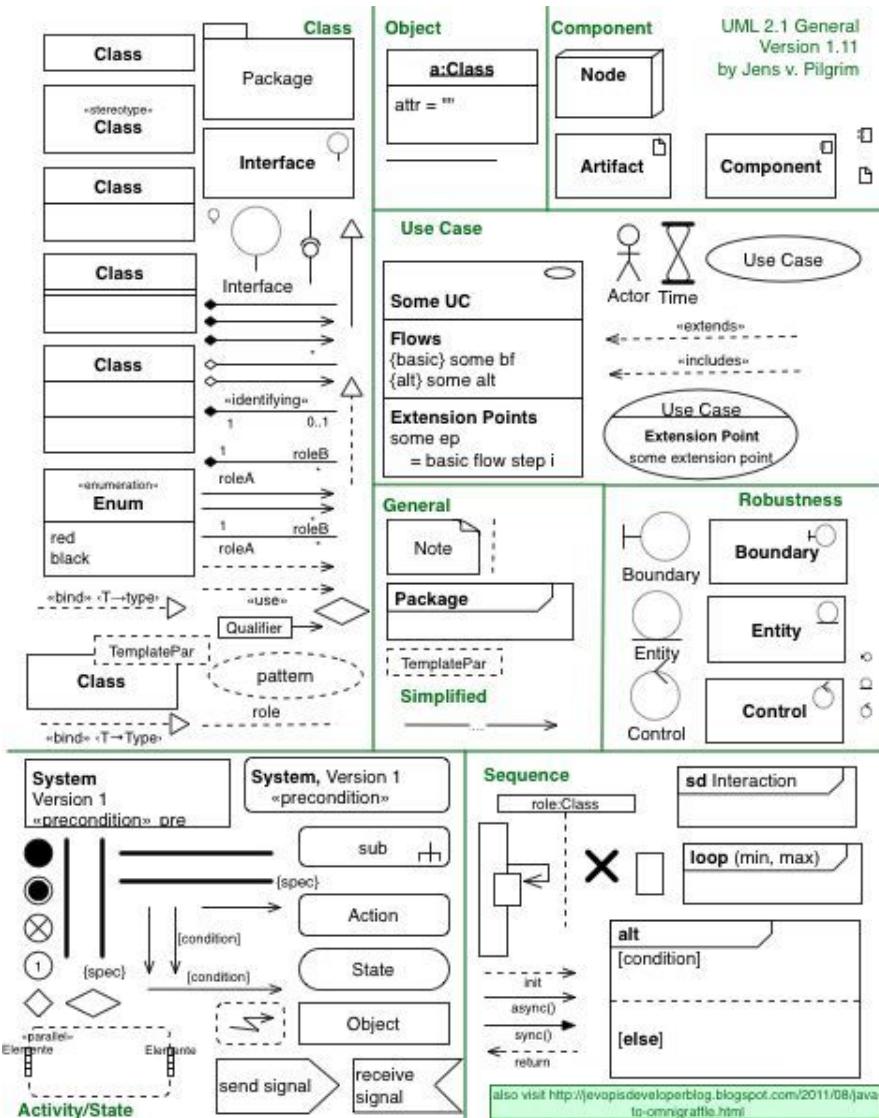
if __name__ == '__main__':
    main()
```

## 类之间的关系

简单的说，类和类之间的关系有三种：is-a、has-a和use-a关系。

- is-a关系也叫继承或泛化，比如学生和人的关系、手机和电子产品的关系都属于继承关系。
- has-a关系通常称之为关联，比如部门和员工的关系，汽车和引擎的关系都属于关联关系；关联关系如果是整体和部分的关联，那么我们称之为聚合关系；如果整体进一步负责了部分的生命周期（整体和部分是不可分割的，同时同在也同时消亡），那么这种就是最强的关联关系，我们称之为合成关系。
- use-a关系通常称之为依赖，比如司机有一个驾驶的行为（方法），其中（的参数）使用到了汽车，那么司机和汽车的关系就是依赖关系。

我们可以使用一种叫做UML（统一建模语言）的东西来进行面向对象建模，其中一项重要的工作就是把类和类之间的关系用标准化的图形符号描述出来。关于UML我们在这里不做详细的介绍，有兴趣的读者可以自行阅读《UML面向对象设计基础》一书。



利用类之间的这些关系，我们可以在已有类的基础上来完成某些操作，也可以在已有类的基础上创建新的类，这些都是实现代码复用的重要手段。复用现有的代码不仅可以减少开发的工作量，也有利于代码的管理和维护，这是我们在日常工作中都会使用到的技术手段。

## 继承和多态

刚才我们提到了，可以在已有类的基础上创建新类，这其中的一种做法就是让一个类从另一个类那里将属性和方法直接继承下来，从而减少重复代码的编写。提供继承信息的我们称之为父类，也叫超类或基类；得到继承信息的我们称之为子类，也叫派生类或衍生类。子类除了继承父类提供的属性和方法，还可以定义自己特有的属性和方法，所以子类比父类拥有的更多的能力，在实际开发中，我们经常会用子类对象去替换掉一个父类对象，这是面向对象编程中一个常见的行为，对应的原则称之为[里氏替换原则](#)。下面我们先看一个继承的例子。

```
class Person(object):
    """人"""

    def __init__(self, name, age):
        self._name = name
        self._age = age

    @property
    def name(self):
        return self._name

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, age):
        self._age = age

    def play(self):
        print('%s正在愉快的玩耍.' % self._name)

    def watch_av(self):
        if self._age >= 18:
            print('%s正在观看爱情动作片.' % self._name)
        else:
            print('%s只能观看《熊出没》.' % self._name)

class Student(Person):
    """学生"""

    def __init__(self, name, age, grade):
        super().__init__(name, age)
        self._grade = grade

    @property
    def grade(self):
        return self._grade

    @grade.setter
    def grade(self, grade):
```

```

        self._grade = grade

    def study(self, course):
        print('%s的%s正在学习%s.' % (self._grade, self._name, course))

class Teacher(Person):
    """老师"""

    def __init__(self, name, age, title):
        super().__init__(name, age)
        self._title = title

    @property
    def title(self):
        return self._title

    @title.setter
    def title(self, title):
        self._title = title

    def teach(self, course):
        print('%s%s正在讲%s.' % (self._name, self._title, course))

def main():
    stu = Student('王大锤', 15, '初三')
    stu.study('数学')
    stu.watch_av()
    t = Teacher('骆昊', 38, '老叫兽')
    t.teach('Python程序设计')
    t.watch_av()

if __name__ == '__main__':
    main()

```

子类在继承了父类的方法后，可以对父类已有的方法给出新的实现版本，这个动作称之为方法重写（override）。通过方法重写我们可以让父类的同一个行为在子类中拥有不同的实现版本，当我们调用这个经过子类重写的方法时，不同的子类对象会表现出不同的行为，这个就是多态（poly-morphism）。

```

from abc import ABCMeta, abstractmethod

class Pet(object, metaclass=ABCMeta):
    """宠物"""

    def __init__(self, nickname):
        self._nickname = nickname

    @abstractmethod

```

```

def make_voice(self):
    """发出声音"""
    pass

class Dog(Pet):
    """狗"""

    def make_voice(self):
        print('%s: 汪汪汪...' % self._nickname)

class Cat(Pet):
    """猫"""

    def make_voice(self):
        print('%s: 喵...喵...' % self._nickname)

def main():
    pets = [Dog('旺财'), Cat('凯蒂'), Dog('大黄')]
    for pet in pets:
        pet.make_voice()

if __name__ == '__main__':
    main()

```

在上面的代码中，我们将 Pet 类处理成了一个抽象类，所谓抽象类就是不能够创建对象的类，这种类的存在就是专门为了让其他类去继承它。Python 从语法层面并没有像 Java 或 C# 那样提供对抽象类的支持，但是我们可以通过 abc 模块的 ABCMeta 元类和 abstractmethod 包装器来达到抽象类的效果，如果一个类中存在抽象方法那么这个类就不能够实例化（创建对象）。上面的代码中，Dog 和 Cat 两个子类分别对 Pet 类中的 make\_voice 抽象方法进行了重写并给出了不同的实现版本，当我们在 main 函数中调用该方法时，这个方法就表现出了多态行为（同样的方法做了不同的事情）。

## 综合案例

### 案例1：奥特曼打小怪兽

```

from abc import ABCMeta, abstractmethod
from random import randint, randrange

class Fighter(object, metaclass=ABCMeta):
    """战斗者"""

    # 通过__slots__魔法限定对象可以绑定的成员变量
    __slots__ = ('_name', '_hp')

    @abstractmethod
    def attack(self, target):
        """对指定对象`target`进行一次攻击。
        攻击会使得`target`失去`self`一定量的生命值。
        """
        pass

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def hp(self):
        return self._hp

    @hp.setter
    def hp(self, value):
        self._hp = value

    @property
    def alive(self):
        return self._hp > 0

```

```
def __init__(self, name, hp):
    """初始化方法

    :param name: 名字
    :param hp: 生命值
    """
    self._name = name
    self._hp = hp

    @property
    def name(self):
        return self._name

    @property
    def hp(self):
        return self._hp

    @hp.setter
    def hp(self, hp):
        self._hp = hp if hp >= 0 else 0

    @property
    def alive(self):
        return self._hp > 0

    @abstractmethod
    def attack(self, other):
        """攻击

        :param other: 被攻击的对象
        """
        pass

class Ultraman(Fighter):
    """奥特曼"""

    __slots__ = ('_name', '_hp', '_mp')

    def __init__(self, name, hp, mp):
        """初始化方法

        :param name: 名字
        :param hp: 生命值
        :param mp: 魔法值
        """
        super().__init__(name, hp)
        self._mp = mp

    def attack(self, other):
        other.hp -= randint(15, 25)

    def huge_attack(self, other):
        """究极必杀技(打掉对方至少50点或四分之三的血)
```

```

:param other: 被攻击的对象

:return: 使用成功返回True否则返回False
"""
if self._mp >= 50:
    self._mp -= 50
    injury = other.hp * 3 // 4
    injury = injury if injury >= 50 else 50
    other.hp -= injury
    return True
else:
    self.attack(other)
    return False

def magic_attack(self, others):
    """魔法攻击

    :param others: 被攻击的群体

    :return: 使用魔法成功返回True否则返回False
"""
    if self._mp >= 20:
        self._mp -= 20
        for temp in others:
            if temp.alive:
                temp.hp -= randint(10, 15)
        return True
    else:
        return False

def resume(self):
    """恢复魔法值"""
    incr_point = randint(1, 10)
    self._mp += incr_point
    return incr_point

def __str__(self):
    return '%s奥特曼\n' % self._name + \
           '生命值: %d\n' % self._hp + \
           '魔法值: %d\n' % self._mp

class Monster(Fighter):
    """小怪兽"""

    __slots__ = ('_name', '_hp')

    def attack(self, other):
        other.hp -= randint(10, 20)

    def __str__(self):
        return '%s小怪兽\n' % self._name + \
               '生命值: %d\n' % self._hp

```

```

def is_any_alive(monsters):
    """判断有没有小怪兽是活着的"""
    for monster in monsters:
        if monster.alive > 0:
            return True
    return False

def select_alive_one(monsters):
    """选中一只活着的小怪兽"""
    monsters_len = len(monsters)
    while True:
        index = randrange(monsters_len)
        monster = monsters[index]
        if monster.alive > 0:
            return monster

def display_info(ultraman, monsters):
    """显示奥特曼和小怪兽的信息"""
    print(ultraman)
    for monster in monsters:
        print(monster, end='')

def main():
    u = Ultraman('路飞', 1000, 120)
    m1 = Monster('索隆', 250)
    m2 = Monster('白元芳', 500)
    m3 = Monster('王大锤', 750)
    ms = [m1, m2, m3]
    fight_round = 1
    while u.alive and is_any_alive(ms):
        print('=====第%02d回合===== % fight_round)
        m = select_alive_one(ms) # 选中一只小怪兽
        skill = randint(1, 10) # 通过随机数选择使用哪种技能
        if skill <= 6: # 60%的概率使用普通攻击
            print('%s使用普通攻击打了%s.' % (u.name, m.name))
            u.attack(m)
            print('%s的魔法值恢复了%d点.' % (u.name, u.resume()))
        elif skill <= 9: # 30%的概率使用魔法攻击(可能因魔法值不足而失败)
            if u.magic_attack(ms):
                print('%s使用了魔法攻击.' % u.name)
            else:
                print('%s使用魔法失败.' % u.name)
        else: # 10%的概率使用究极必杀技(如果魔法值不足则使用普通攻击)
            if u.huge_attack(m):
                print('%s使用究极必杀技虐了%s.' % (u.name, m.name))
            else:
                print('%s使用普通攻击打了%s.' % (u.name, m.name))
                print('%s的魔法值恢复了%d点.' % (u.name, u.resume()))
        if m.alive > 0: # 如果选中的小怪兽没有死就回击奥特曼
            print('%s回击了%s.' % (m.name, u.name))
            m.attack(u)
    display_info(u, ms) # 每个回合结束后显示奥特曼和小怪兽的信息

```

```

    fight_round += 1
    print('\n=====战斗结束!=====\\n')
    if u.alive > 0:
        print('%s奥特曼胜利!' % u.name)
    else:
        print('小怪兽胜利!')

```

```

if __name__ == '__main__':
    main()

```

## 案例2：扑克游戏

```

import random

class Card(object):
    """一张牌"""

    def __init__(self, suite, face):
        self._suite = suite
        self._face = face

    @property
    def face(self):
        return self._face

    @property
    def suite(self):
        return self._suite

    def __str__(self):
        if self._face == 1:
            face_str = 'A'
        elif self._face == 11:
            face_str = 'J'
        elif self._face == 12:
            face_str = 'Q'
        elif self._face == 13:
            face_str = 'K'
        else:
            face_str = str(self._face)
        return '%s%s' % (self._suite, face_str)

    def __repr__(self):
        return self.__str__()

class Poker(object):
    """一副牌"""

    def __init__(self):
        self._cards = [Card(suite, face)
                      for suite in range(4)
                      for face in range(1, 14)]

```

```
        for suite in '♠♥♦♣'
        for face in range(1, 14)]
    self._current = 0

@property
def cards(self):
    return self._cards

def shuffle(self):
    """洗牌(随机乱序)"""
    self._current = 0
    random.shuffle(self._cards)

@property
def next(self):
    """发牌"""
    card = self._cards[self._current]
    self._current += 1
    return card

@property
def has_next(self):
    """还有没有牌"""
    return self._current < len(self._cards)

class Player(object):
    """玩家"""

    def __init__(self, name):
        self._name = name
        self._cards_on_hand = []

    @property
    def name(self):
        return self._name

    @property
    def cards_on_hand(self):
        return self._cards_on_hand

    def get(self, card):
        """摸牌"""
        self._cards_on_hand.append(card)

    def arrange(self, card_key):
        """玩家整理手上的牌"""
        self._cards_on_hand.sort(key=card_key)

# 排序规则-先根据花色再根据点数排序
def get_key(card):
    return (card.suite, card.face)
```

```

def main():
    p = Poker()
    p.shuffle()
    players = [Player('东邪'), Player('西毒'), Player('南帝'), Player('北丐')]
    for _ in range(13):
        for player in players:
            player.get(p.next)
    for player in players:
        print(player.name + ':', end=' ')
        player.arrange(get_key)
        print(player.cards_on_hand)

if __name__ == '__main__':
    main()

```

**说明**：大家可以自己尝试在上面代码的基础上写一个简单的扑克游戏，例如21点（Black Jack），游戏的规则可以自己在网上找一找。

### 案例3：工资结算系统

```

"""
某公司有三种类型的员工 分别是部门经理、程序员和销售员
需要设计一个工资结算系统 根据提供的员工信息来计算月薪
部门经理的月薪是每月固定15000元
程序员的月薪按本月工作时间计算 每小时150元
销售员的月薪是1200元的底薪加上销售额5%的提成
"""

from abc import ABCMeta, abstractmethod

```

```

class Employee(object, metaclass=ABCMeta):
    """员工"""

    def __init__(self, name):
        """
        初始化方法

        :param name: 姓名
        """
        self._name = name

    @property
    def name(self):
        return self._name

    @abstractmethod
    def get_salary(self):
        """
        获得月薪

        :return: 月薪
        """

```

```
"""

pass


class Manager(Employee):
    """部门经理"""

    def get_salary(self):
        return 15000.0


class Programmer(Employee):
    """程序员"""

    def __init__(self, name, working_hour=0):
        super().__init__(name)
        self._working_hour = working_hour

    @property
    def working_hour(self):
        return self._working_hour

    @working_hour.setter
    def working_hour(self, working_hour):
        self._working_hour = working_hour if working_hour > 0 else 0

    def get_salary(self):
        return 150.0 * self._working_hour


class Salesman(Employee):
    """销售员"""

    def __init__(self, name, sales=0):
        super().__init__(name)
        self._sales = sales

    @property
    def sales(self):
        return self._sales

    @sales.setter
    def sales(self, sales):
        self._sales = sales if sales > 0 else 0

    def get_salary(self):
        return 1200.0 + self._sales * 0.05


def main():
    emps = [
        Manager('刘备'), Programmer('诸葛亮'),
        Manager('曹操'), Salesman('荀彧'),
        Salesman('吕布'), Programmer('张辽'),
        Programmer('赵云')
    ]
```

```
]

for emp in emps:
    if isinstance(emp, Programmer):
        emp.working_hour = int(input('请输入%s本月工作时间: ' % emp.name))
    elif isinstance(emp, Salesman):
        emp.sales = float(input('请输入%s本月销售额: ' % emp.name))
# 同样是接收get_salary这个消息但是不同的员工表现出了不同的行为(多态)
print('%s本月工资为: ￥%s元' %
      (emp.name, emp.get_salary()))

if __name__ == '__main__':
    main()
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day01-15 / Day10 / 图形用户界面和游戏开发.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



330 lines (264 sloc) 14.5 KB

# 图形用户界面和游戏开发

## 基于tkinter模块的GUI

GUI是图形用户界面的缩写，图形化的用户界面对于使用过计算机的人来说应该都不陌生，在此也无需进行赘述。Python默认的GUI开发模块是tkinter（在Python 3以前的版本中名为Tkinter），从这个名字就可以看出它是基于Tk的，Tk是一个工具包，最初是为Tcl设计的，后来被移植到很多其他的脚本语言中，它提供了跨平台的GUI控件。当然Tk并不是最新和最好的选择，也没有功能特别强大的GUI控件，事实上，开发GUI应用并不是Python最擅长的工作，如果真的需要使用Python开发GUI应用，wxPython、PyQt、PyGTK等模块都是不错的选择。

基本上使用tkinter来开发GUI应用需要以下5个步骤：

1. 导入tkinter模块中我们需要的东西。
2. 创建一个顶层窗口对象并用它来承载整个GUI应用。
3. 在顶层窗口对象上添加GUI组件。
4. 通过代码将这些GUI组件的功能组织起来。
5. 进入主事件循环(main loop)。

下面的代码演示了如何使用tkinter做一个简单的GUI应用。

```
import tkinter
import tkinter.messagebox

def main():
    flag = True

    # 修改标签上的文字
    def change_label_text():
        nonlocal flag
```

```

flag = not flag
color, msg = ('red', 'Hello, world!')\
    if flag else ('blue', 'Goodbye, world!')
label.config(text=msg, fg=color)

# 确认退出
def confirm_to_quit():
    if tkinter.messagebox.askokcancel('温馨提示', '确定要退出吗?'):
        top.quit()

# 创建顶层窗口
top = tkinter.Tk()
# 设置窗口大小
top.geometry('240x160')
# 设置窗口标题
top.title('小游戏')
# 创建标签对象并添加到顶层窗口
label = tkinter.Label(top, text='Hello, world!', font='Arial -32', fg='red')
label.pack(expand=1)
# 创建一个装按钮的容器
panel = tkinter.Frame(top)
# 创建按钮对象 指定添加到哪个容器中 通过command参数绑定事件回调函数
button1 = tkinter.Button(panel, text='修改', command=change_label_text)
button1.pack(side='left')
button2 = tkinter.Button(panel, text='退出', command=confirm_to_quit)
button2.pack(side='right')
panel.pack(side='bottom')
# 开启主事件循环
tkinter.mainloop()

if __name__ == '__main__':
    main()

```

需要说明的是，GUI应用通常是事件驱动式的，之所以要进入主事件循环就是要监听鼠标、键盘等各种事件的发生并执行对应的代码对事件进行处理，因为事件会持续的发生，所以需要这样的一个循环一直运行着等待下一个事件的发生。另一方面，Tk为控件的摆放提供了三种布局管理器，通过布局管理器可以对控件进行定位，这三种布局管理器分别是：Placer（开发者提供控件的大小和摆放位置）、Packer（自动将控件填充到合适的位置）和Grid（基于网格坐标来摆放控件），此处不进行赘述。

## 使用Pygame进行游戏开发

Pygame是一个开源的Python模块，专门用于多媒体应用（如电子游戏）的开发，其中包括对图像、声音、视频、事件、碰撞等的支持。Pygame建立在SDL的基础上，SDL是一套跨平台的多媒体开发库，用C语言实现，被广泛的应用于游戏、模拟器、播放器等的开发。而Pygame让游戏开发者不再被底层语言束缚，可以更多的关注游戏的功能和逻辑。

下面我们来完成一个简单的小游戏，游戏的名字叫“大球吃小球”，当然完成这个游戏并不是重点，学会使用Pygame也不是重点，最重要的我们要在这个过程中体会如何使用前面讲解的面向对象程序设计，学会用这种编程思想去解决现实中的问题。

## 制作游戏窗口

```
import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

    if __name__ == '__main__':
        main()
```

## 在窗口中绘图

可以通过pygame中draw模块的函数在窗口上绘图，可以绘制的图形包括：线条、矩形、多边形、圆、椭圆、圆弧等。需要说明的是，屏幕坐标系是将屏幕左上角设置为坐标原点(0, 0)，向右是x轴的正向，向下是y轴的正向，在表示位置或者设置尺寸的时候，我们默认的单位都是像素。所谓像素就是屏幕上的一个点，你可以用浏览图片的软件试着将一张图片放大若干倍，就可以看到这些点。pygame中表示颜色用的是色光三原色表示法，即通过一个元组或列表来指定颜色的RGB值，每个值都在0~255之间，因为是每种原色都用一个8位(bit)的值来表示，三种颜色相当于一共由24位构成，这也就是常说的“24位颜色表示法”。

```
import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
```

```

# 设置窗口的背景色(颜色是由红绿蓝三原色构成的元组)
screen.fill((242, 242, 242))
# 绘制一个圆(参数分别是: 屏幕, 颜色, 圆心位置, 半径, 0表示填充圆)
pygame.draw.circle(screen, (255, 0, 0,), (100, 100), 30, 0)
# 刷新当前窗口(渲染窗口将绘制的图像呈现出来)
pygame.display.flip()
running = True
# 开启一个事件循环处理发生的事件
while running:
    # 从消息队列中获取事件并对事件进行处理
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

if __name__ == '__main__':
    main()

```

#### ####加载图像

如果需要直接加载图像到窗口上，可以使用pygame中image模块的函数来加载图像，再通过之前获得的窗口对象的 `blit` 方法渲染图像，代码如下所示。

```

import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 设置窗口的背景色(颜色是由红绿蓝三原色构成的元组)
    screen.fill((255, 255, 255))
    # 通过指定的文件名加载图像
    ball_image = pygame.image.load('./res/ball.png')
    # 在窗口上渲染图像
    screen.blit(ball_image, (50, 50))
    # 刷新当前窗口(渲染窗口将绘制的图像呈现出来)
    pygame.display.flip()
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

if __name__ == '__main__':
    main()

```

#### ####实现动画效果

说到[动画](#)这个词大家都不会陌生，事实上要实现动画效果，本身的原理也非常简单，就是将不连续的图片连续的播放，只要每秒钟达到了一定的帧数，那么就可以做出比较流畅的动画效果。如果要让上面代码中的小球动起来，可以将小球的位置用变量来表示，并在循环中修改小球的位置再刷新整个窗口即可。

```
import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 定义变量来表示小球在屏幕上的位置
    x, y = 50, 50
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
        screen.fill((255, 255, 255))
        pygame.draw.circle(screen, (255, 0, 0), (x, y), 30, 0)
        pygame.display.flip()
        # 每隔50毫秒就改变小球的位置再刷新窗口
        pygame.time.delay(50)
        x, y = x + 5, y + 5

    if __name__ == '__main__':
        main()
```

#### 碰撞检测

通常一个游戏中会有很多对象出现，而这些对象之间的“碰撞”在所难免，比如炮弹击中了飞机、箱子撞到了地面等。碰撞检测在绝大多数的游戏中都是一个必须得处理的至关重要的问题，pygame的sprite（动画精灵）模块就提供了对碰撞检测的支持，这里我们暂时不介绍sprite模块提供的功能，因为要检测两个小球有没有碰撞其实非常简单，只需要检查球心的距离有没有小于两个球的半径之和。为了制造出更多的小球，我们可以通过对鼠标事件的处理，在点击鼠标的位置创建颜色、大小和移动速度都随机的小球，当然要做到这一点，我们可以把之前学习到的面向对象的知识应用起来。

```
from enum import Enum, unique
from math import sqrt
```

```
from random import randint

import pygame

@unique
class Color(Enum):
    """颜色"""

    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)
    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    GRAY = (242, 242, 242)

    @staticmethod
    def random_color():
        """获得随机颜色"""
        r = randint(0, 255)
        g = randint(0, 255)
        b = randint(0, 255)
        return (r, g, b)

class Ball(object):
    """球"""

    def __init__(self, x, y, radius, sx, sy, color=Color.RED):
        """初始化方法"""
        self.x = x
        self.y = y
        self.radius = radius
        self.sx = sx
        self.sy = sy
        self.color = color
        self.alive = True

    def move(self, screen):
        """移动"""
        self.x += self.sx
        self.y += self.sy
        if self.x - self.radius <= 0 or \
            self.x + self.radius >= screen.get_width():
            self.sx = -self.sx
        if self.y - self.radius <= 0 or \
            self.y + self.radius >= screen.get_height():
            self.sy = -self.sy

    def eat(self, other):
        """吃其他球"""
        if self.alive and other.alive and self != other:
            dx, dy = self.x - other.x, self.y - other.y
            distance = sqrt(dx ** 2 + dy ** 2)
            if distance < self.radius + other.radius \
```

```

        and self.radius > other.radius:
    other.alive = False
    self.radius = self.radius + int(other.radius * 0.146)

def draw(self, screen):
    """在窗口上绘制球"""
    pygame.draw.circle(screen, self.color,
                       (self.x, self.y), self.radius, 0)

```

## 事件处理

可以在事件循环中对鼠标事件进行处理，通过事件对象的 type 属性可以判定事件类型，再通过 pos 属性就可以获得鼠标点击的位置。如果要处理键盘事件也是在这个地方，做法与处理鼠标事件类似。

```

def main():
    # 定义用来装所有球的容器
    balls = []
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            # 处理鼠标事件的代码
            if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
                # 获得点击鼠标的位置
                x, y = event.pos
                radius = randint(10, 100)
                sx, sy = randint(-10, 10), randint(-10, 10)
                color = Color.random_color()
                # 在点击鼠标的位置创建一个球(大小、速度和颜色随机)
                ball = Ball(x, y, radius, sx, sy, color)
                # 将球添加到列表容器中
                balls.append(ball)
        screen.fill((255, 255, 255))
        # 取出容器中的球 如果没被吃掉就绘制 被吃掉了就移除
        for ball in balls:
            if ball.alive:
                ball.draw(screen)
            else:
                balls.remove(ball)
        pygame.display.flip()
        # 每隔50毫秒就改变球的位置再刷新窗口
        pygame.time.delay(50)

```

```
for ball in balls:  
    ball.move(screen)  
    # 检查球有没有吃到其他的球  
    for other in balls:  
        ball.eat(other)  
  
if __name__ == '__main__':  
    main()
```

上面的两段代码合在一起，我们就完成了“大球吃小球”的游戏（如下图所示），准确的说它算不上一个游戏，但是做一个小游戏的基本知识我们已经通过这个例子告诉大家了，有了这些知识已经可以开始你的小游戏开发之旅了。其实上面的代码中还有很多值得改进的地方，比如刷新窗口以及让球移动起来的代码并不应该放在事件循环中，等学习了多线程的知识后，用一个后台线程来处理这些事可能是更好的选择。如果希望获得更好的用户体验，我们还可以在游戏中加入背景音乐以及在球与球发生碰撞时播放音效，利用pygame的mixer和music模块，我们可以很容易的做到这一点，大家可以自行了解这方面的知识。事实上，想了解更多的关于pygame的知识，最好的教程是[pygame的官方网站](#)，如果英语没毛病就可以赶紧去看看啦。如果想开发[3D游戏](#)，pygame就显得力不从心了，对3D游戏开发如果有兴趣的读者不妨看看[Panda3D](#)。

Branch: master ▾

[Find file](#)[Copy path](#)

## Python-100-Days / Day01-15 / Day11 / 文件和异常.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

266 lines (205 sloc) 12.9 KB

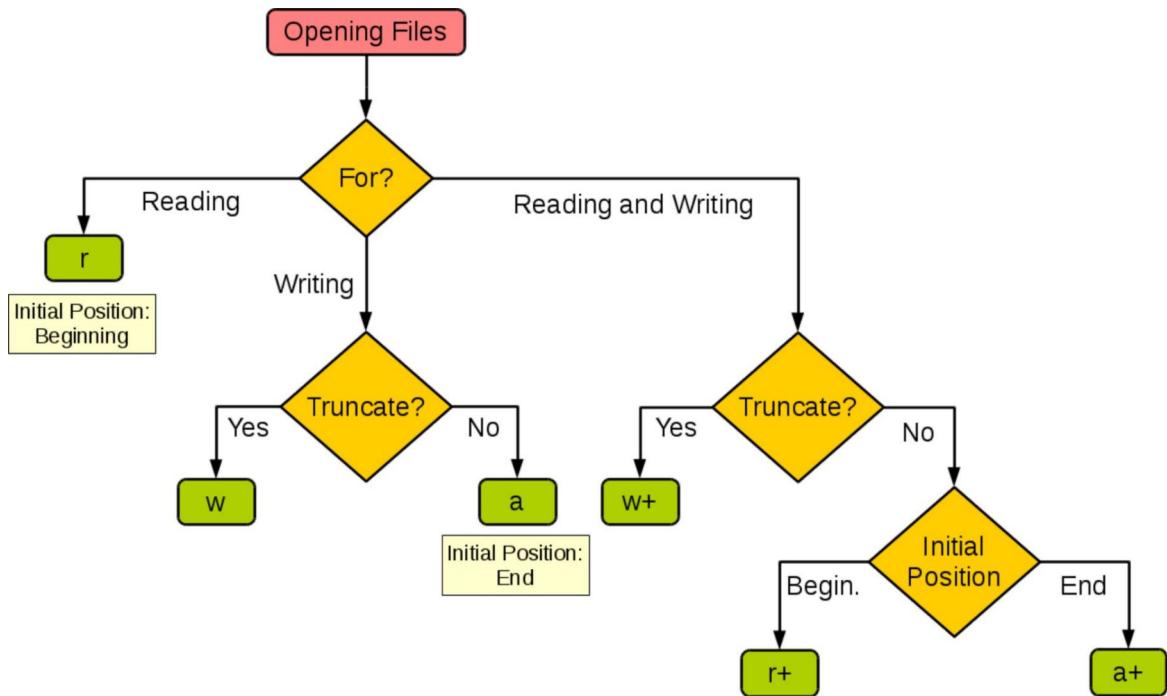
## 文件和异常

在实际开发中，常常需要对程序中的数据进行持久化操作，而实现数据持久化最直接简单的方式就是将数据保存到文件中。说到“文件”这个词，可能需要先科普一下关于[文件系统](#)的知识，对于这个概念，维基百科上给出了很好的诠释，这里不再浪费笔墨。

在Python中实现文件的读写操作其实非常简单，通过Python内置的 `open` 函数，我们可以指定文件名、操作模式、编码信息等来获得操作文件的对象，接下来就可以对文件进行读写操作了。这里所说的操作模式是指要打开什么样的文件（字符文件还是二进制文件）以及做什么样的操作（读、写还是追加），具体的如下表所示。

操作模式	具体含义
'r'	读取（默认）
'w'	写入（会先截断之前的内容）
'x'	写入，如果文件已经存在会产生异常
'a'	追加，将内容写入到已有文件的末尾
'b'	二进制模式
't'	文本模式（默认）
'+'	更新（既可以读又可以写）

下面这张图来自于[菜鸟教程](#)网站，它展示了如果根据应用程序的需要来设置操作模式。



## 读写文本文件

读取文本文件时，需要在使用 `open` 函数时指定好带路径的文件名（可以使用相对路径或绝对路径）并将文件模式设置为 '`r`'（如果不指定，默认值也是 '`r`'），然后通过 `encoding` 参数指定编码（如果不指定，默认值是 `None`，那么在读取文件时使用的是操作系统默认的编码），如果不能保证保存文件时使用的编码方式与 `encoding` 参数指定的编码方式是一致的，那么就可能因无法解码字符而导致读取失败。下面的例子演示了如何读取一个纯文本文件。

```

def main():
    f = open('致橡树.txt', 'r', encoding='utf-8')
    print(f.read())
    f.close()

if __name__ == '__main__':
    main()
  
```

请注意上面的代码，如果 `open` 函数指定的文件并不存在或者无法打开，那么将引发异常状况导致程序崩溃。为了让代码有一定的健壮性和容错性，我们可以使用Python的异常机制对可能在运行时发生状况的代码进行适当的处理，如下所示。

```

def main():
    f = None
    try:
        f = open('致橡树.txt', 'r', encoding='utf-8')
        print(f.read())
    except FileNotFoundError:
        print('无法打开指定的文件!')
  
```

```

except LookupError:
    print('指定了未知的编码!')
except UnicodeDecodeError:
    print('读取文件时解码错误!')
finally:
    if f:
        f.close()

if __name__ == '__main__':
    main()

```

在Python中，我们可以将那些在运行时可能出现状况的代码放在 try 代码块中，在 try 代码块的后面可以跟上一个或多个 except 来捕获可能出现的异常状况。例如在上面读取文件的过程中，文件找不到会引发 FileNotFoundError，指定了未知的编码会引发 LookupError，而如果读取文件时无法按指定方式解码会引发 UnicodeDecodeError，我们在 try 后面跟上了三个 except 分别处理这三种不同的异常状况。最后我们使用 finally 代码块来关闭打开的文件，释放掉程序中获取的外部资源，由于 finally 块的代码不论程序正常还是异常都会执行到（甚至是调用了 sys 模块的 exit 函数退出Python环境，finally 块都会被执行，因为 exit 函数实质上是引发了 SystemExit 异常），因此我们通常把 finally 块称为“总是执行代码块”，它最适合用来做释放外部资源的操作。如果不愿意在 finally 代码块中关闭文件对象释放资源，也可以使用上下文语法，通过 with 关键字指定文件对象的上下文环境并在离开上下文环境时自动释放文件资源，代码如下所示。

```

def main():
    try:
        with open('致橡树.txt', 'r', encoding='utf-8') as f:
            print(f.read())
    except FileNotFoundError:
        print('无法打开指定的文件!')
    except LookupError:
        print('指定了未知的编码!')
    except UnicodeDecodeError:
        print('读取文件时解码错误!')

if __name__ == '__main__':
    main()

```

除了使用文件对象的 read 方法读取文件之外，还可以使用 for-in 循环逐行读取或者用 readlines 方法将文件按行读取到一个列表容器中，代码如下所示。

```

import time

def main():
    # 一次性读取整个文件内容

```

```

with open('致橡树.txt', 'r', encoding='utf-8') as f:
    print(f.read())

# 通过for-in循环逐行读取
with open('致橡树.txt', mode='r') as f:
    for line in f:
        print(line, end='')
        time.sleep(0.5)
    print()

# 读取文件按行读取到列表中
with open('致橡树.txt') as f:
    lines = f.readlines()
print(lines)

if __name__ == '__main__':
    main()

```

要将文本信息写入文件文件也非常简单，在使用 `open` 函数时指定好文件名并将文件模式设置为 '`w`' 即可。注意如果需要对文件内容进行追加式写入，应该将模式设置为 '`a`'。如果要写入的文件不存在会自动创建文件而不是引发异常。下面的例子演示了如何将1-9999直接的素数分别写入三个文件中（1-99之间的素数保存在`a.txt`中，100-999之间的素数保存在`b.txt`中，1000-9999之间的素数保存在`c.txt`中）。

```

from math import sqrt

def is_prime(n):
    """判断素数的函数"""
    assert n > 0
    for factor in range(2, int(sqrt(n)) + 1):
        if n % factor == 0:
            return False
    return True if n != 1 else False

def main():
    filenames = ('a.txt', 'b.txt', 'c.txt')
    fs_list = []
    try:
        for filename in filenames:
            fs_list.append(open(filename, 'w', encoding='utf-8'))
        for number in range(1, 10000):
            if is_prime(number):
                if number < 100:
                    fs_list[0].write(str(number) + '\n')
                elif number < 1000:
                    fs_list[1].write(str(number) + '\n')
                else:
                    fs_list[2].write(str(number) + '\n')
    except IOError as ex:

```

```
    print(ex)
    print('写文件时发生错误!')
finally:
    for fs in fs_list:
        fs.close()
print('操作完成!')


if __name__ == '__main__':
    main()
```

## 读写二进制文件

知道了如何读写文本文件要读写二进制文件也就很简单了，下面的代码实现了复制图片文件的功能。

```
def main():
    try:
        with open('guido.jpg', 'rb') as fs1:
            data = fs1.read()
            print(type(data)) # <class 'bytes'>
        with open('吉多.jpg', 'wb') as fs2:
            fs2.write(data)
    except FileNotFoundError as e:
        print('指定的文件无法打开。')
    except IOError as e:
        print('读写文件时出现错误。')
    print('程序执行结束。')


if __name__ == '__main__':
    main()
```

## 读写JSON文件

通过上面的讲解，我们已经知道如何将文本数据和二进制数据保存到文件中，那么这里还有一个问题，如果希望把一个列表或者一个字典中的数据保存到文件中又该怎么做呢？答案是将数据以JSON格式进行保存。JSON是“JavaScript Object Notation”的缩写，它本来是JavaScript语言中创建对象的一种字面量语法，现在已经被广泛的应用于跨平台跨语言的数据交换，原因很简单，因为JSON也是纯文本，任何系统任何编程语言处理纯文本都是没有问题的。目前JSON基本上已经取代了XML作为异构系统间交换数据的事实标准。关于JSON的知识，更多的可以参考[JSON的官方网站](#)，从这个网站也可以了解到每种语言处理JSON数据格式可以使用的工具或三方库，下面是一个JSON的简单例子。

```
{
    'name': '骆昊',
    'age': 38,
    'qq': 957658,
```

```

    'friends': ['王大锤', '白元芳'],
    'cars': [
        {'brand': 'BYD', 'max_speed': 180},
        {'brand': 'Audi', 'max_speed': 280},
        {'brand': 'Benz', 'max_speed': 320}
    ]
}

```

可能大家已经注意到了，上面的JSON跟Python中的字典其实是一样的，事实上JSON的数据类型和Python的数据类型是很容易找到对应关系的，如下面两张表所示。

JSON	Python
object	dict
array	list
string	str
number (int / real)	int / float
true / false	True / False
null	None

Python	JSON
dict	object
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True / False	true / false
None	null

我们使用Python中的json模块就可以将字典或列表以JSON格式保存到文件中，代码如下所示。

```

import json

def main():
    mydict = {
        'name': '骆昊',
        'age': 38,
        'qq': 957658,
        'friends': ['王大锤', '白元芳'],
    }

```

```

'cars': [
    {'brand': 'BYD', 'max_speed': 180},
    {'brand': 'Audi', 'max_speed': 280},
    {'brand': 'Benz', 'max_speed': 320}
]
}
try:
    with open('data.json', 'w', encoding='utf-8') as fs:
        json.dump(mydict, fs)
except IOError as e:
    print(e)
print('保存数据完成!')

if __name__ == '__main__':
    main()

```

json模块主要有四个比较重要的函数，分别是：

- dump - 将Python对象按照JSON格式序列化到文件中
- dumps - 将Python对象处理成JSON格式的字符串
- load - 将文件中的JSON数据反序列化成对象
- loads - 将字符串的内容反序列化成Python对象

这里出现了两个概念，一个叫序列化，一个叫反序列化。自由的百科全书[维基百科](#)上对这两个概念是这样解释的：“序列化（serialization）在计算机科学的数据处理中，是指将数据结构或对象状态转换为可以存储或传输的形式，这样在需要的时候能够恢复到原先的状态，而且通过序列化的数据重新获取字节时，可以利用这些字节来产生原始对象的副本（拷贝）。与这个过程相反的动作，即从一系列字节中提取数据结构的操作，就是反序列化（deserialization）”。

目前绝大多数网络数据服务（或称之为网络API）都是基于[HTTP协议](#)提供JSON格式的数据，关于HTTP协议的相关知识，可以看看阮一峰老师的[《HTTP协议入门》](#)，如果想了解国内的网络数据服务，可以看看[聚合数据](#)和[阿凡达数据](#)等网站，国外的可以看看[{API}Search](#)网站。下面的例子演示了如何使用[requests](#)模块（封装得足够好的第三方网络访问模块）访问网络API获取国内新闻，如何通过[json](#)模块解析JSON数据并显示新闻标题，这个例子使用了[天行数据](#)提供的国内新闻数据接口，其中的APIKey需要自己到该网站申请。

```

import requests
import json

def main():
    resp = requests.get('http://api.tianapi.com/guonei/?key=APIKey&num=10')
    data_model = json.loads(resp.text)
    for news in data_model['newslist']:
        print(news['title'])

```

```
if __name__ == '__main__':
    main()
```

在Python中要实现序列化和反序列化除了使用json模块之外，还可以使用pickle和shelve模块，但是这两个模块是使用特有的序列化协议来序列化数据，因此序列化后的数据只能被Python识别。关于这两个模块的相关知识可以自己看看网络上的资料。另外，如果要了解更多的关于Python异常机制的知识，可以看看segmentfault上面的文章[《总结：Python中的异常处理》](#)，这篇文章不仅介绍了Python中异常机制的使用，还总结了一系列的最佳实践，很值得一读。

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Day01-15 / Day12 / 字符串和正则表达式.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

180 lines (137 sloc) 15 KB

# 使用正则表达式

## 正则表达式相关知识

在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要，正则表达式就是用于描述这些规则的工具，换句话说正则表达式是一种工具，它定义了字符串的匹配模式（如何检查一个字符串是否有跟某种模式匹配的部分或者从一个字符串中将与模式匹配的部分提取出来或者替换掉）。如果你在Windows操作系统中使用过文件查找并且在指定文件名时使用过通配符（\*和?），那么正则表达式也是与之类似的用来进行文本匹配的工具，只不过比起通配符正则表达式更强大，它能更精确地描述你的需求（当然你付出的代价是书写一个正则表达式比打出一个通配符要复杂得多，要知道任何给你带来好处的东西都是有代价的，就如同学习一门编程语言一样），比如你可以编写一个正则表达式，用来查找所有以0开头，后面跟着2-3个数字，然后是一个连字号“-”，最后是7或8位数字的字符串（像028-12345678或0813-7654321），这不就是国内的座机号码吗。最初计算机是为了做数学运算而诞生的，处理的信息基本上都是数值，而今天我们在日常工作中处理的信息基本上都是文本数据，我们希望计算机能够识别和处理符合某些模式的文本，正则表达式就显得非常重要的。今天几乎所有的编程语言都提供了对正则表达式操作的支持，Python通过标准库中的re模块来支持正则表达式操作。

我们可以考虑下面一个问题：我们从某个地方（可能是一个文本文件，也可能是网络上的一则新闻）获得了一个字符串，希望在字符串中找出手机号和座机号。当然我们可以设定手机号是11位的数字（注意并不是随机的11位数字，因为你没有见过“25012345678”这样的手机号吧）而座机号跟上一段中描述的模式相同，如果不使用正则表达式要完成这个任务就会很麻烦。

关于正则表达式的相关知识，大家可以阅读一篇非常有名的博客叫[《正则表达式30分钟入门教程》](#)，读完这篇文章后你就可以看懂下面的表格，这是我们对正则表达式中的一些基本符号进行的扼要总结。

符号	解释	示例	说明
----	----	----	----

.	匹配任意字符	b.t	可以匹配bat / but / b#t / b1t等
\w	匹配字母/数字/下划线	b\wt	可以匹配bat / b1t / b_t等 但不能匹配b#t
\s	匹配空白字符 (包括\r、\n、\t等)	love\syou	可以匹配love you
\d	匹配数字	\d\d	可以匹配01 / 23 / 99等
\b	匹配单词的边界	\bThe\b	
^	匹配字符串的开始	^The	可以匹配The开头的字符串
\$	匹配字符串的结束	.exe\$	可以匹配.exe结尾的字符串
\W	匹配非字母/数字/下划线	b\Wt	可以匹配b#t / b@t等 但不能匹配but / b1t / b_t等
\S	匹配非空白字符	love\Syou	可以匹配love#you等 但不能匹配love you
\D	匹配非数字	\d\D	可以匹配9a / 3# / 0F等
\B	匹配非单词边界	\Bio\B	
[]	匹配来自字符集的任意单一字符	[aeiou]	可以匹配任一元音字母字符
[^]	匹配不在字符集中的任意单一字符	[^aeiou]	可以匹配任一非元音字母字符
*	匹配0次或多次	\w*	
+	匹配1次或多次	\w+	
?	匹配0次或1次	\w?	
{N}	匹配N次	\w{3}	

{M,}	匹配至少M次	\w{3,}	
{M,N}	匹配至少M次至多N次	\w{3,6}	
	分支	foo bar	可以匹配foo或者bar
(?#)	注释		
(exp)	匹配exp并捕获到自动命名的组中		
(? <name>exp)</name>	匹配exp并捕获到名为name的组中		
(?:exp)	匹配exp但是不捕获匹配的文本		
(?=exp)	匹配exp前面的位置	\b\w+(?=ing)	可以匹配I'm dancing中的danc
(?<=exp)	匹配exp后面的位置	(? <name>=\bdanc)\w+\b</name>	可以匹配I love dancing and reading中的第一个ing
(?!exp)	匹配后面不是exp的位置		
(?<!exp)	匹配前面不是exp的位置		
*?	重复任意次，但尽可能少重复	a.*b a.*?b	将正则表达式应用于aabab，前者会匹配整个字符串aabab，后者会匹配aab和ab两个字符串
+?	重复1次或多次，但尽可能少重复		
??	重复0次或1次，但尽可能少重复		
{M,N}?	重复M到N次，但尽可能少重复		

{M,}?	重复M次以上，但尽可能少重复		
-------	----------------	--	--

\*\*说明：\*\*如果需要匹配的字符是正则表达式中的特殊字符，那么可以使用\进行转义处理，例如想匹配小数点可以写成\\.就可以了，因为直接写.会匹配任意字符；同理，想匹配圆括号必须写成\\(和\\)，否则圆括号被视为正则表达式中的分组。

## Python对正则表达式的支持

Python提供了re模块来支持正则表达式相关操作，下面是re模块中的核心函数。

函数	说明
compile(pattern, flags=0)	编译正则表达式返回正则表达式对象
match(pattern, string, flags=0)	用正则表达式匹配字符串 成功返回匹配对象 否则返回None
search(pattern, string, flags=0)	搜索字符串中第一次出现正则表达式的模式 成功返回匹配对象 否则返回None
split(pattern, string, maxsplit=0, flags=0)	用正则表达式指定的模式分隔符拆分字符串 返回列表
sub(pattern, repl, string, count=0, flags=0)	用指定的字符串替换原字符串中与正则表达式匹配的模式 可以用count指定替换的次数
fullmatch(pattern, string, flags=0)	match函数的完全匹配（从字符串开头到结尾）版本
findall(pattern, string, flags=0)	查找字符串所有与正则表达式匹配的模式 返回字符串的列表
finditer(pattern, string, flags=0)	查找字符串所有与正则表达式匹配的模式 返回一个迭代器
purge()	清除隐式编译的正则表达式的缓存
re.I / re.IGNORECASE	忽略大小写匹配标记
re.M / re.MULTILINE	多行匹配标记

\*\*说明：\*\*上面提到的re模块中的这些函数，实际开发中也可以用正则表达式对象的方法替代对这些函数的使用，如果一个正则表达式需要重复的使用，那么先通过compile函数编译正则表达式并创建出正则表达式对象无疑是更为明智的选择。

下面我们通过一系列的例子来告诉大家在Python中如何使用正则表达式。

## 例子1：验证输入用户名和QQ号是否有效并给出对应的提示信息。

```
"""
验证输入用户名和QQ号是否有效并给出对应的提示信息

要求：用户名必须由字母、数字或下划线构成且长度在6~20个字符之间，QQ号是5~12的数字且首位
"""

import re

def main():
    username = input('请输入用户名：')
    qq = input('请输入QQ号：')
    # match函数的第一个参数是正则表达式字符串或正则表达式对象
    # 第二个参数是要跟正则表达式做匹配的字符串对象
    m1 = re.match(r'^[0-9a-zA-Z]{6,20}$', username)
    if not m1:
        print('请输入有效的用户名。')
    m2 = re.match(r'^[1-9]\d{4,11}$', qq)
    if not m2:
        print('请输入有效的QQ号。')
    if m1 and m2:
        print('你输入的信息是有效的！')

if __name__ == '__main__':
    main()
```

**提示：**上面在书写正则表达式时使用了“原始字符串”的写法（在字符串前面加上了r），所谓“原始字符串”就是字符串中的每个字符都是它原始的意义，说得更直接一点就是字符串中没有所谓的转义字符啦。因为正则表达式中有很多元字符和需要进行转义的地方，如果不使用原始字符串就需要将反斜杠写作\\，例如表示数字的\d得书写成\\d，这样不仅写起来不方便，阅读的时候也会很吃力。

## 例子2：从一段文字中提取出国内手机号码。

下面这张图是截止到2017年底，国内三家运营商推出的手机号段。

国内三家运营商的号段分别如下：

电信号段:133/153/180/181/189/177；
联通号段:130/131/132/155/156/185/186/145/176；
移动号段：
134/135/136/137/138/139/150/151/152/157/158/159/182/183/184/187/188/147/178

```
import re
```

```

def main():
    # 创建正则表达式对象 使用了前瞻和回顾来保证手机号前后不应该出现数字
    pattern = re.compile(r'(?<=\D)1[34578]\d{9}(?=\D)')
    sentence = '''
        重要的事情说8130123456789遍，我的手机号是13512346789这个靓号，
        不是15600998765，也是110或119，王大锤的手机号才是15600998765。
        '''

    # 查找所有匹配并保存到一个列表中
    mylist = re.findall(pattern, sentence)
    print(mylist)
    print('-----华丽的分隔线-----')
    # 通过迭代器取出匹配对象并获得匹配的内容
    for temp in pattern.finditer(sentence):
        print(temp.group())
    print('-----华丽的分隔线-----')
    # 通过search函数指定搜索位置找出所有匹配
    m = pattern.search(sentence)
    while m:
        print(m.group())
        m = pattern.search(sentence, m.end())

if __name__ == '__main__':
    main()

```

**说明**：上面匹配国内手机号的正则表达式并不够好，因为像14开头的号码只有145或147，而上面的正则表达式并没有考虑这种情况，要匹配国内手机号，更好的正则表达式的写法是：`(?<=\D)(1[38]\d{9}|14[57]\d{8}|15[0-35-9]\d{8}|17[678]\d{8})(?=\D)`，国内最近好像有19和16开头的手机号了，但是这个暂时不在我们考虑之列。

### 例子3：替换字符串中的不良内容

```

import re

def main():
    sentence = '你丫是傻叉吗？我操你大爷的。Fuck you.'
    purified = re.sub('[操肏艹]|fuck|shit|傻[比屄逼义缺吊屐]|煞笔',
                      '*', sentence, flags=re.IGNORECASE)
    print(purified) # 你丫是*吗？我*你大爷的。 * you.

if __name__ == '__main__':
    main()

```

**说明**：re模块的正则表达式相关函数中都有一个flags参数，它代表了正则表达式的匹配标记，可以通过该标记来指定匹配时是否忽略大小写、是否进行多行匹配、是否显示调试信息等。如果需要为flags参数指定多个值，可以使用按位或运算符进行叠加，如 flags=re.I | re.M。

#### 例子4：拆分长字符串

```
import re

def main():
    poem = '窗前明月光，疑是地上霜。举头望明月，低头思故乡。'
    sentence_list = re.split(r'[，。。]', poem)
    while '' in sentence_list:
        sentence_list.remove('')
    print(sentence_list) # ['窗前明月光', '疑是地上霜', '举头望明月', '低头思故乡']

if __name__ == '__main__':
    main()
```

#### 后话

如果要从事爬虫类应用的开发，那么正则表达式一定是一个非常好的助手，因为它可以帮助我们迅速的从网页代码中发现某种我们指定的模式并提取出我们需要的信息，当然对于初学者来说，要编写一个正确的适当的正则表达式可能并不是一件容易的事情（当然有些常用的正则表达式可以直接在网上找找），所以实际开发爬虫应用的时候，有很多人会选择Beautiful Soup或Lxml来进行匹配和信息的提取，前者简单方便但是性能较差，后者既好用性能也好，但是安装稍嫌麻烦，这些内容我们会在后期的爬虫专题中为大家介绍。



Branch: master ▾

[Find file](#) [Copy path](#)

## [Python-100-Days / Day01-15 / Day13 / 进程和线程.md](#)

 jackfrued 修正了文档上的部分bug

bd20a91 May 3, 2019

2 contributors 

[Raw](#) [Blame](#) [History](#)



490 lines (346 sloc) 23.9 KB

# 进程和线程

今天我们使用的计算机早已进入多CPU或多核时代，而我们使用的操作系统都是支持“多任务”的操作系统，这使得我们可以同时运行多个程序，也可以将一个程序分解为若干个相对独立的子任务，让多个子任务并发的执行，从而缩短程序的执行时间，同时也让用户获得更好的体验。因此在当下不管是用什么编程语言进行开发，实现让程序同时执行多个任务也就是常说的“并发编程”，应该是程序员必备技能之一。为此，我们需要先讨论两个概念，一个叫进程，一个叫线程。

## 概念

进程就是操作系统中执行的一个程序，操作系统以进程为单位分配存储空间，每个进程都有自己的地址空间、数据栈以及其他用于跟踪进程执行的辅助数据，操作系统管理所有进程的执行，为它们合理的分配资源。进程可以通过fork或spawn的方式来创建新的进程来执行其他的任务，不过新的进程也有自己独立的内存空间，因此必须通过进程间通信机制（IPC，Inter-Process Communication）来实现数据共享，具体的方式包括管道、信号、套接字、共享内存区等。

一个进程还可以拥有多个并发的执行线索，简单的说就是拥有多个可以获得CPU调度的执行单元，这就是所谓的线程。由于线程在同一个进程中，它们可以共享相同的上下文，因此相对于进程而言，线程间的信息共享和通信更加容易。当然在单核CPU系统中，真正的并发是不可能的，因为在某个时刻能够获得CPU的只有唯一的一个线程，多个线程共享了CPU的执行时间。使用多线程实现并发编程为程序带来的好处是不言而喻的，最主要的体现在提升程序的性能和改善用户体验，今天我们使用的软件几乎都用到了多线程技术，这一点可以利用系统自带的进程监控工具（如macOS中的“活动监视器”、Windows中的“任务管理器”）来证实，如下图所示。



当然多线程也并不是没有坏处，站在其他进程的角度，多线程的程序对其他程序并不友好，因为它占用了更多的CPU执行时间，导致其他程序无法获得足够的CPU执行时间；另一方面，站在开发者的角度，编写和调试多线程的程序都对开发者有较高的要求，对于初学者来说更加困难。

Python既支持多进程又支持多线程，因此使用Python实现并发编程主要有3种方式：多进程、多线程、多进程+多线程。

## Python中的多进程

Unix和Linux操作系统上提供了 `fork()` 系统调用来创建进程，调用 `fork()` 函数的是父进程，创建出的是子进程，子进程是父进程的一个拷贝，但是子进程拥有自己的PID。

`fork()` 函数非常特殊它会返回两次，父进程中可以通过 `fork()` 函数的返回值得到子进程的PID，而子进程中的返回值永远都是0。Python的os模块提供了 `fork()` 函数。由于Windows系统没有 `fork()` 调用，因此要实现跨平台的多进程编程，可以使用multiprocessing模块的 `Process` 类来创建子进程，而且该模块还提供了更高级的封装，例如批量启动进程的进程池（`Pool`）、用于进程间通信的队列（`Queue`）和管道（`Pipe`）等。

下面用一个下载文件的例子来说明使用多进程和不使用多进程到底有什么差别，先看看下面的代码。

```
from random import randint
from time import time, sleep
```

```

def download_task(filename):
    print('开始下载%s...' % filename)
    time_to_download = randint(5, 10)
    sleep(time_to_download)
    print('%s下载完成！耗费了%d秒' % (filename, time_to_download))

def main():
    start = time()
    download_task('Python从入门到住院.pdf')
    download_task('Peking Hot.avi')
    end = time()
    print('总共耗费了%.2f秒.' % (end - start))

if __name__ == '__main__':
    main()

```

下面是运行程序得到的一次运行结果。

```

开始下载Python从入门到住院.pdf...
Python从入门到住院.pdf下载完成！耗费了6秒
开始下载Peking Hot.avi...
Peking Hot.avi下载完成！耗费了7秒
总共耗费了13.01秒。

```

从上面的例子可以看出，如果程序中的代码只能按顺序一点点的往下执行，那么即使执行两个毫不相关的下载任务，也需要先等待一个文件下载完成后才能开始下一个下载任务，很显然这并不合理也没有效率。接下来我们使用多进程的方式将两个下载任务放到不同的进程中，代码如下所示。

```

from multiprocessing import Process
from os import getpid
from random import randint
from time import time, sleep

def download_task(filename):
    print('启动下载进程，进程号[%d].' % getpid())
    print('开始下载%s...' % filename)
    time_to_download = randint(5, 10)
    sleep(time_to_download)
    print('%s下载完成！耗费了%d秒' % (filename, time_to_download))

def main():
    start = time()
    p1 = Process(target=download_task, args=('Python从入门到住院.pdf', ))
    p1.start()
    p2 = Process(target=download_task, args=('Peking Hot.avi', ))
    p2.start()

```

```

    p1.join()
    p2.join()
    end = time()
    print('总共耗费了%.2f秒.' % (end - start))

if __name__ == '__main__':
    main()

```

在上面的代码中，我们通过 `Process` 类创建了进程对象，通过 `target` 参数我们传入一个函数来表示进程启动后要执行的代码，后面的 `args` 是一个元组，它代表了传递给函数的参数。`Process` 对象的 `start` 方法用来启动进程，而 `join` 方法表示等待进程执行结束。运行上面的代码可以明显发现两个下载任务“同时”启动了，而且程序的执行时间将大大缩短，不再是两个任务的时间总和。下面是程序的一次执行结果。

```

启动下载进程，进程号[1530].
开始下载Python从入门到住院.pdf...
启动下载进程，进程号[1531].
开始下载Peking Hot.avi...
Peking Hot.avi下载完成！耗费了7秒
Python从入门到住院.pdf下载完成！耗费了10秒
总共耗费了10.01秒。

```

我们也可以使用`subprocess`模块中的类和函数来创建和启动子进程，然后通过管道来和子进程通信，这些内容我们不在此进行讲解，有兴趣的读者可以自己了解这些知识。接下来我们将重点放在如何实现两个进程间的通信。我们启动两个进程，一个输出Ping，一个输出Pong，两个进程输出的Ping和Pong加起来一共10个。听起来很简单吧，但是如果这样写可是错的哦。

```

from multiprocessing import Process
from time import sleep

counter = 0

def sub_task(string):
    global counter
    while counter < 10:
        print(string, end='', flush=True)
        counter += 1
        sleep(0.01)

def main():
    Process(target=sub_task, args=('Ping', )).start()
    Process(target=sub_task, args=('Pong', )).start()

```

```
if __name__ == '__main__':
    main()
```

看起来没毛病，但是最后的结果是Ping和Pong各输出了10个，Why？当我们在程序中创建进程的时候，子进程复制了父进程及其所有的数据结构，每个子进程都有自己独立的内存空间，这也就意味着两个子进程中各有一个 counter 变量，所以结果也就可想而知了。要解决这个问题比较简单办法是使用multiprocessing模块中的 Queue 类，它是可以被多个进程共享的队列，底层是通过管道和信号量（semaphore）机制来实现的，有兴趣的读者可以自己尝试一下。

## Python中的多线程

在Python早期的版本中就引入了thread模块（现在名为\_thread）来实现多线程编程，然而该模块过于底层，而且很多功能都没有提供，因此目前的多线程开发我们推荐使用threading模块，该模块对多线程编程提供了更好的面向对象的封装。我们把刚才下载文件的例子用多线程的方式来实现一遍。

```
from random import randint
from threading import Thread
from time import time, sleep

def download(filename):
    print('开始下载%s...' % filename)
    time_to_download = randint(5, 10)
    sleep(time_to_download)
    print('%s下载完成！耗费了%d秒' % (filename, time_to_download))

def main():
    start = time()
    t1 = Thread(target=download, args=('Python从入门到住院.pdf',))
    t1.start()
    t2 = Thread(target=download, args=('Peking Hot.avi',))
    t2.start()
    t1.join()
    t2.join()
    end = time()
    print('总共耗费了%.3f秒' % (end - start))

if __name__ == '__main__':
    main()
```

我们可以直接使用threading模块的 Thread 类来创建线程，但是我们之前讲过一个非常重要的概念叫“继承”，我们可以从已有的类创建新类，因此也可以通过继承 Thread 类的方式来创建自定义的线程类，然后再创建线程对象并启动线程。代码如下所示。

```

from random import randint
from threading import Thread
from time import time, sleep

class DownloadTask(Thread):

    def __init__(self, filename):
        super().__init__()
        self._filename = filename

    def run(self):
        print('开始下载%s...' % self._filename)
        time_to_download = randint(5, 10)
        sleep(time_to_download)
        print('%s下载完成！耗费了%d秒' % (self._filename, time_to_download))

def main():
    start = time()
    t1 = DownloadTask('Python从入门到住院.pdf')
    t1.start()
    t2 = DownloadTask('Peking Hot.avi')
    t2.start()
    t1.join()
    t2.join()
    end = time()
    print('总共耗费了%.2f秒.' % (end - start))

if __name__ == '__main__':
    main()

```

因为多个线程可以共享进程的内存空间，因此要实现多个线程间的通信相对简单，大家能想到的最直接的办法就是设置一个全局变量，多个线程共享这个全局变量即可。但是当多个线程共享同一个变量（我们通常称之为“资源”）的时候，很有可能产生不可控的结果从而导致程序失效甚至崩溃。如果一个资源被多个线程竞争使用，那么我们通常称之为“临界资源”，对“临界资源”的访问需要加上保护，否则资源会处于“混乱”的状态。下面的例子演示了100个线程向同一个银行账户转账（转入1元钱）的场景，在这个例子中，银行账户就是一个临界资源，在没有保护的情况下我们很有可能会得到错误的结果。

```

from time import sleep
from threading import Thread

class Account(object):

    def __init__(self):
        self._balance = 0

```

```

def deposit(self, money):
    # 计算存款后的余额
    new_balance = self._balance + money
    # 模拟受理存款业务需要0.01秒的时间
    sleep(0.01)
    # 修改账户余额
    self._balance = new_balance

@property
def balance(self):
    return self._balance

class AddMoneyThread(Thread):

    def __init__(self, account, money):
        super().__init__()
        self._account = account
        self._money = money

    def run(self):
        self._account.deposit(self._money)

def main():
    account = Account()
    threads = []
    # 创建100个存款的线程向同一个账户中存钱
    for _ in range(100):
        t = AddMoneyThread(account, 1)
        threads.append(t)
        t.start()
    # 等所有存款的线程都执行完毕
    for t in threads:
        t.join()
    print('账户余额为: ￥%d元' % account.balance)

if __name__ == '__main__':
    main()

```

运行上面的程序，结果让人大跌眼镜，100个线程分别向账户中转入1元钱，结果居然远远小于100元。之所以出现这种情况是因为我们没有对银行账户这个“临界资源”加以保护，多个线程同时向账户中存钱时，会一起执行到 `new_balance = self._balance + money` 这行代码，多个线程得到的账户余额都是初始状态下的 0，所以都是 0 上面做了+1的操作，因此得到了错误的结果。在这种情况下，“锁”就可以派上用场了。我们可以通过“锁”来保护“临界资源”，只有获得“锁”的线程才能访问“临界资源”，而其他没有得到“锁”的线程只能被阻塞起来，直到获得“锁”的线程释放了“锁”，其他线程才有机会获得“锁”，进而访问被保护的“临界资源”。下面的代码演示了如何使用“锁”来保护对银行账户的操作，从而获得正确的结果。

```
from time import sleep
from threading import Thread, Lock


class Account(object):

    def __init__(self):
        self._balance = 0
        self._lock = Lock()

    def deposit(self, money):
        # 先获取锁才能执行后续的代码
        self._lock.acquire()
        try:
            new_balance = self._balance + money
            sleep(0.01)
            self._balance = new_balance
        finally:
            # 在finally中执行释放锁的操作保证正常异常锁都能释放
            self._lock.release()

    @property
    def balance(self):
        return self._balance


class AddMoneyThread(Thread):

    def __init__(self, account, money):
        super().__init__()
        self._account = account
        self._money = money

    def run(self):
        self._account.deposit(self._money)


def main():
    account = Account()
    threads = []
    for _ in range(100):
        t = AddMoneyThread(account, 1)
        threads.append(t)
        t.start()
    for t in threads:
        t.join()
    print('账户余额为: ￥%d元' % account.balance)

if __name__ == '__main__':
    main()
```

比较遗憾的一件事情是Python的多线程并不能发挥CPU的多核特性，这一点只要启动几个执行死循环的线程就可以得到证实了。之所以如此，是因为Python的解释器有一个“全局解释器锁”（GIL）的东西，任何线程执行前必须先获得GIL锁，然后每执行100条字节码，解释器就自动释放GIL锁，让别的线程有机会执行，这是一个历史遗留问题，但是即便如此，就如我们之前举的例子，使用多线程在提升执行效率和改善用户体验方面仍然是有积极意义的。

## 多进程还是多线程

无论是多进程还是多线程，只要数量一多，效率肯定上不去，为什么呢？我们打个比方，假设你不幸正在准备中考，每天晚上需要做语文、数学、英语、物理、化学这5科的作业，每项作业耗时1小时。如果你先花1小时做语文作业，做完了，再花1小时做数学作业，这样，依次全部做完，一共花5小时，这种方式称为单任务模型。如果你打算切换到多任务模型，可以先做1分钟语文，再切换到数学作业，做1分钟，再切换到英语，以此类推，只要切换速度足够快，这种方式就和单核CPU执行多任务是一样的了，以旁观者的角度来看，你就正在同时写5科作业。

但是，切换作业是有代价的，比如从语文切到数学，要先收拾桌子上的语文书本、钢笔（这叫保存现场），然后，打开数学课本、找出圆规直尺（这叫准备新环境），才能开始做数学作业。操作系统在切换进程或者线程时也是一样的，它需要先保存当前执行的现场环境（CPU寄存器状态、内存页等），然后，把新任务的执行环境准备好（恢复上次的寄存器状态，切换内存页等），才能开始执行。这个切换过程虽然很快，但是也需要耗费时间。如果有几千个任务同时进行，操作系统可能就主要忙着切换任务，根本没多少时间去执行任务了，这种情况最常见的就是硬盘狂响，点窗口无反应，系统处于假死状态。所以，多任务一旦多到一个限度，反而会使得系统性能急剧下降，最终导致所有任务都做不好。

是否采用多任务的第二个考虑是任务的类型，可以把任务分为计算密集型和I/O密集型。计算密集型任务的特点是要进行大量的计算，消耗CPU资源，比如对视频进行编码解码或者格式转换等等，这种任务全靠CPU的运算能力，虽然也可以用多任务完成，但是任务越多，花在任务切换的时间就越多，CPU执行任务的效率就越低。计算密集型任务由于主要消耗CPU资源，这类任务用Python这样的脚本语言去执行效率通常很低，最能胜任这类任务的是C语言，我们之前提到了Python中有嵌入C/C++代码的机制。

除了计算密集型任务，其他的涉及到网络、存储介质I/O的任务都可以视为I/O密集型任务，这类任务的特点是CPU消耗很少，任务的大部分时间都在等待I/O操作完成（因为I/O的速度远远低于CPU和内存的速度）。对于I/O密集型任务，如果启动多任务，就可以减少I/O等待时间从而让CPU高效率的运转。有一大类的任务都属于I/O密集型任务，这其中包括了我们很快会涉及到的网络应用和Web应用。

\*\*说明：\*\*上面的内容和例子来自于[廖雪峰官方网站的《Python教程》](#)，因为对作者文中的某些观点持有不同的看法，对原文的文字描述做了适当的调整。

## 单线程 + 异步I/O

现代操作系统对I/O操作的改进中最为重要的就是支持异步I/O。如果充分利用操作系统提供的异步I/O支持，就可以用单进程单线程模型来执行多任务，这种全新的模型称为事件驱动模型。Nginx就是支持异步I/O的Web服务器，它在单核CPU上采用单进程模型就可以高效地支持多任务。在多核CPU上，可以运行多个进程（数量与CPU核心数相同），充分利用多核CPU。用Node.js开发的服务器端程序也使用了这种工作模式，这也是当下实现多任务编程的一种趋势。

在Python语言中，单线程+异步I/O的编程模型称为协程，有了协程的支持，就可以基于事件驱动编写高效的多任务程序。协程最大的优势就是极高的执行效率，因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销。协程的第二个优势就是不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不用加锁，只需要判断状态就好了，所以执行效率比多线程高很多。如果想要充分利用CPU的多核特性，最简单的方法是多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。关于这方面的内容，我稍后会做一个专题来进行讲解。

## 应用案例

### 例子1：将耗时的任务放到线程中以获得更好的用户体验。

如下所示的界面中，有“下载”和“关于”两个按钮，用休眠的方式模拟点击“下载”按钮会联网下载文件需要耗费10秒的时间，如果不使用“多线程”，我们会发现，当点击“下载”按钮后整个程序的其他部分都被这个耗时的任务阻塞而无法执行了，这显然是非常糟糕的用户体验，代码如下所示。

```
import time
import tkinter
import tkinter.messagebox

def download():
    # 模拟下载任务需要花费10秒钟时间
    time.sleep(10)
    tkinter.messagebox.showinfo('提示', '下载完成!')

def show_about():
    tkinter.messagebox.showinfo('关于', '作者: 骆昊(v1.0)')

def main():
    top = tkinter.Tk()
    top.title('单线程')
    top.geometry('200x150')
    top.wm_attributes('-topmost', True)

    panel = tkinter.Frame(top)
    button1 = tkinter.Button(panel, text='下载', command=download)
    button1.pack(side='left')
```

```

button2 = tkinter.Button(panel, text='关于', command=show_about)
button2.pack(side='right')
panel.pack(side='bottom')

tkinter.mainloop()

if __name__ == '__main__':
    main()

```

如果使用多线程将耗时间的任务放到一个独立的线程中执行，这样就不会因为执行耗时间的任务而阻塞了主线程，修改后的代码如下所示。

```

import time
import tkinter
import tkinter.messagebox
from threading import Thread


def main():

    class DownloadTaskHandler(Thread):

        def run(self):
            time.sleep(10)
            tkinter.messagebox.showinfo('提示', '下载完成!')
            # 启用下载按钮
            button1.config(state=tkinter.NORMAL)

        def download():
            # 禁用下载按钮
            button1.config(state=tkinter.DISABLED)
            # 通过daemon参数将线程设置为守护线程(主程序退出就不再保留执行)
            # 在线程中处理耗时间的下载任务
            DownloadTaskHandler(daemon=True).start()

        def show_about():
            tkinter.messagebox.showinfo('关于', '作者: 骆昊(v1.0)')


    top = tkinter.Tk()
    top.title('单线程')
    top.geometry('200x150')
    top.wm_attributes('-topmost', 1)

    panel = tkinter.Frame(top)
    button1 = tkinter.Button(panel, text='下载', command=download)
    button1.pack(side='left')
    button2 = tkinter.Button(panel, text='关于', command=show_about)
    button2.pack(side='right')
    panel.pack(side='bottom')

    tkinter.mainloop()

```

```
if __name__ == '__main__':
    main()
```

## 例子2：使用多进程对复杂任务进行“分而治之”。

我们来完成1~100000000求和的计算密集型任务，这个问题本身非常简单，有点循环的知识就能解决，代码如下所示。

```
from time import time

def main():
    total = 0
    number_list = [x for x in range(1, 100000001)]
    start = time()
    for number in number_list:
        total += number
    print(total)
    end = time()
    print('Execution time: %.3fs' % (end - start))

if __name__ == '__main__':
    main()
```

在上面的代码中，我故意先去创建了一个列表容器然后填入了100000000个数，这一步其实是比较耗时间的，所以为了公平起见，当我们将这个任务分解到8个进程中去执行的时候，我们暂时也不考虑列表切片操作花费的时间，只是把做运算和合并运算结果的时间统计出来，代码如下所示。

```
from multiprocessing import Process, Queue
from random import randint
from time import time

def task_handler(curr_list, result_queue):
    total = 0
    for number in curr_list:
        total += number
    result_queue.put(total)

def main():
    processes = []
    number_list = [x for x in range(1, 100000001)]
    result_queue = Queue()
    index = 0
    # 启动8个进程将数据切片后进行运算
    for _ in range(8):
        p = Process(target=task_handler, args=(number_list[index:index+1250000], result_queue))
        p.start()
        processes.append(p)
        index += 1250000
    for p in processes:
        p.join()
    result_list = [result_queue.get() for _ in range(8)]
    print(result_list)
    print(sum(result_list))
```

```
p = Process(target=task_handler,
            args=(number_list[index:index + 12500000], result_queue))
index += 12500000
processes.append(p)
p.start()
# 开始记录所有进程执行完成花费的时间
start = time()
for p in processes:
    p.join()
# 合并执行结果
total = 0
while not result_queue.empty():
    total += result_queue.get()
print(total)
end = time()
print('Execution time: ', (end - start), 's', sep='')

if __name__ == '__main__':
    main()
```

比较两段代码的执行结果（在我目前使用的MacBook上，上面的代码需要大概6秒左右的时间，而下面的代码只需要不到1秒的时间，再强调一次我们只是比较了运算的时间，不考虑列表创建及切片操作花费的时间），使用多进程后由于获得了更多的CPU执行时间以及更好的利用了CPU的多核特性，明显的减少了程序的执行时间，而且计算量越大效果越明显。当然，如果愿意还可以将多个进程部署在不同的计算机上，做成分布式进程，具体的做法就是通过multiprocessing.managers模块中提供的管理器将 Queue 对象通过网络共享出来（注册到网络上让其他计算机可以访问），这部分内容也留到爬虫的专题再进行讲解。

Branch: master ▾

[Find file](#) [Copy path](#)

[Python-100-Days](#) / [Day01-15](#) / [Day14-A](#) / 网络编程入门.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



300 lines (212 sloc) 17.1 KB

# 网络编程入门

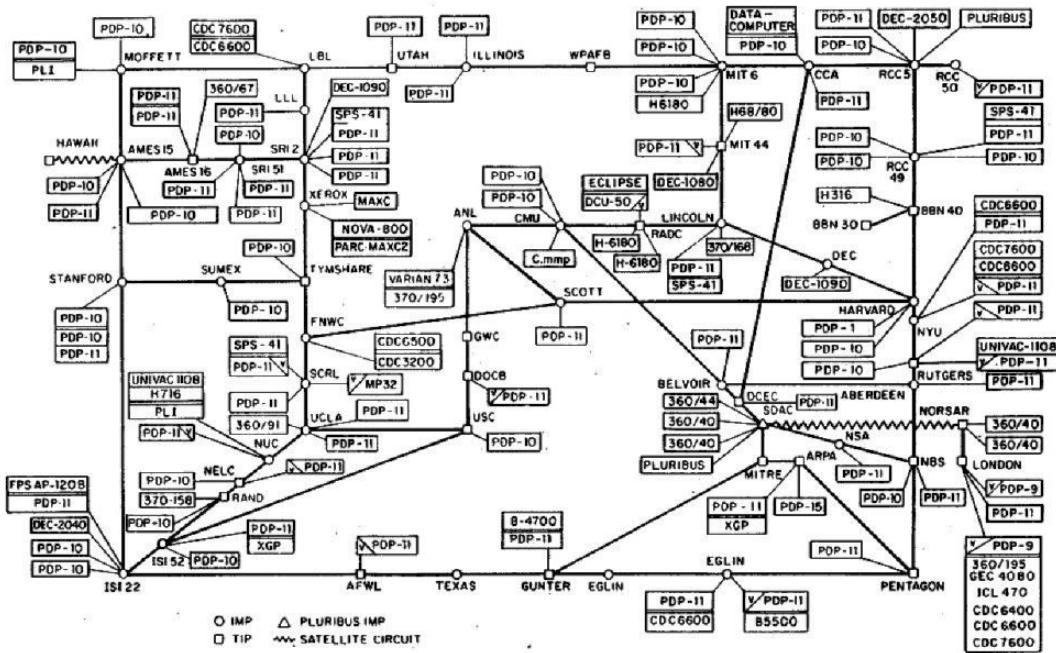
## 计算机网络基础

计算机网络是独立自主的计算机互联而成的系统的总称，组建计算机网络最主要的目的  
是实现多台计算机之间的通信和资源共享。今天计算机网络中的设备和计算机网络的用  
户已经多得不可计数，而计算机网络也可以称得上是一个“复杂巨系统”，对于这样的系  
统，我们不可能用一两篇文章把它讲清楚，有兴趣的读者可以自行阅读Andrew  
S.Tanenbaum老师的经典之作《计算机网络》或Kurose和Ross老师合著的《计算机网络：  
自顶向下方法》来了解计算机网络的相关知识。

## 计算机网络发展史

1. 1960s - 美国国防部ARPANET项目问世，奠定了分组交换网络的基础。

ARPANET LOGICAL MAP, MARCH 1977

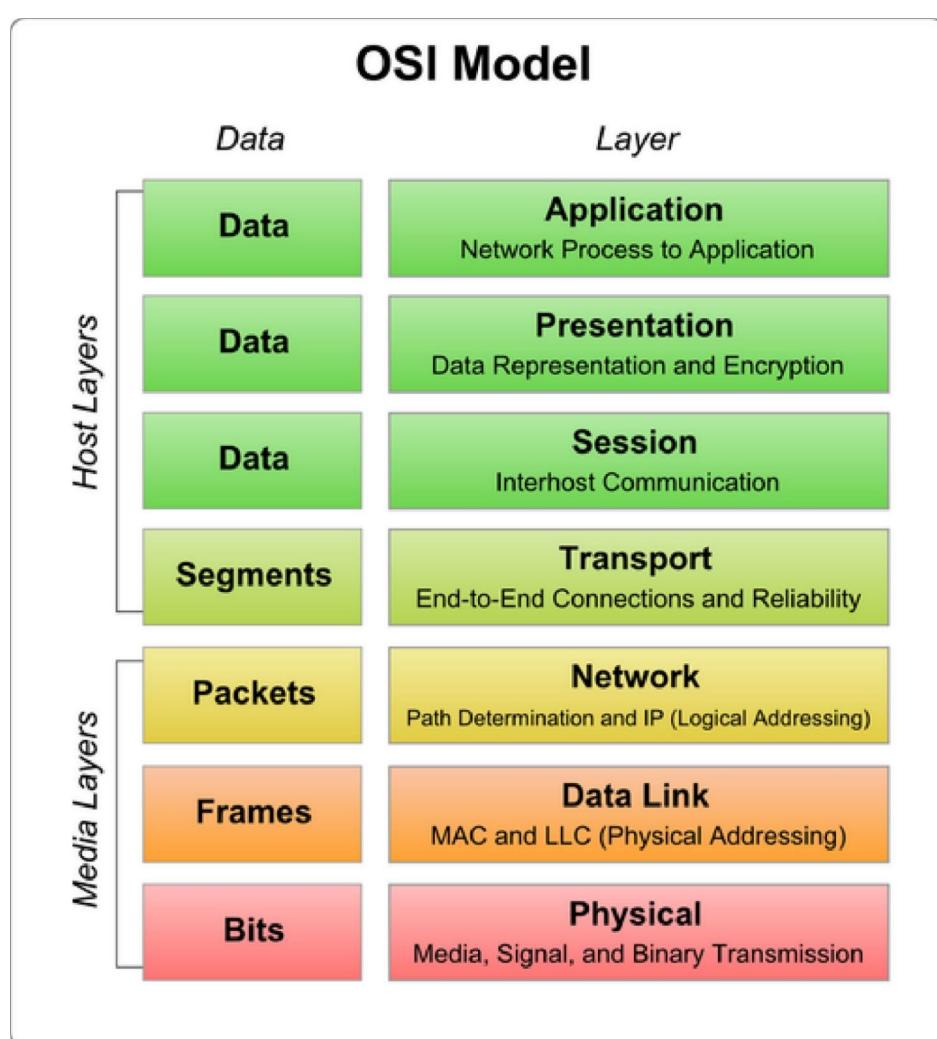


(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

NAMES SHOWN ARE IMP NAMES, NOT NECESSARILY HOST NAMES

2. 1980s - 国际标准化组织 (ISO) 发布OSI/RM，奠定了网络技术标准化的基础。

## OSI Model

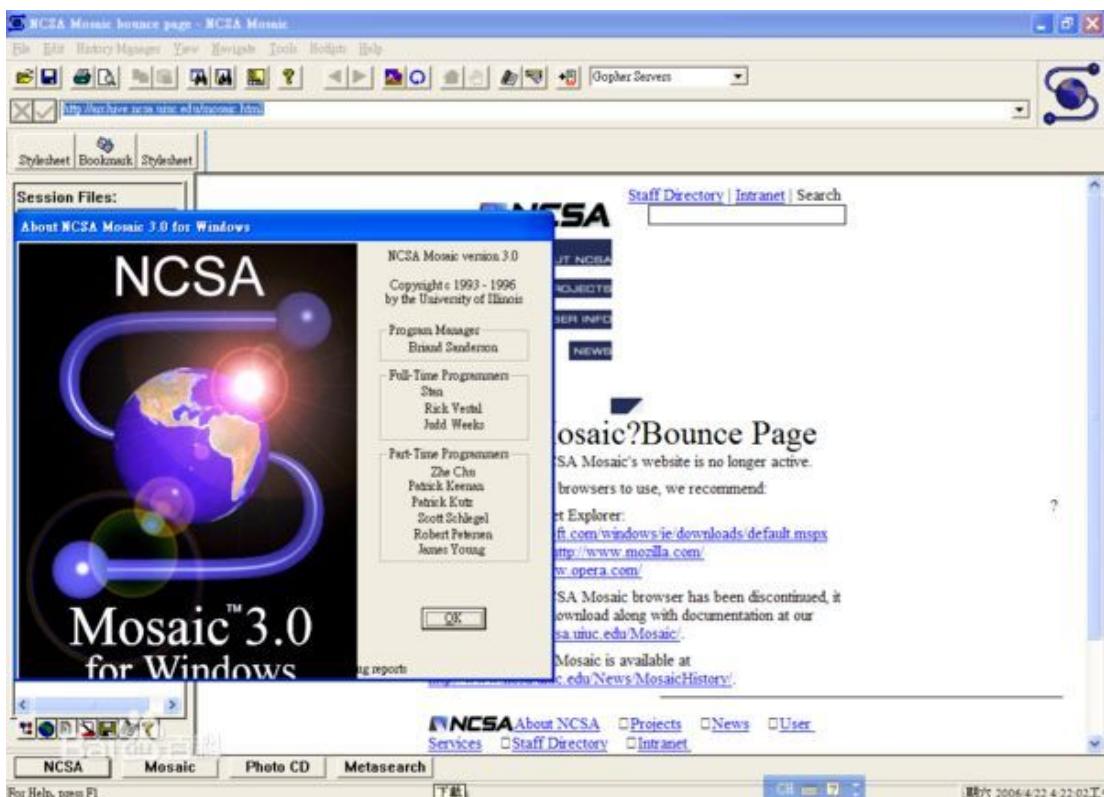


3. 1990s - 英国人蒂姆·伯纳斯-李发明了图形化的浏览器，浏览器的简单易用性使得计算机网络迅速被普及。

在没有浏览器的年代，上网是这样的。

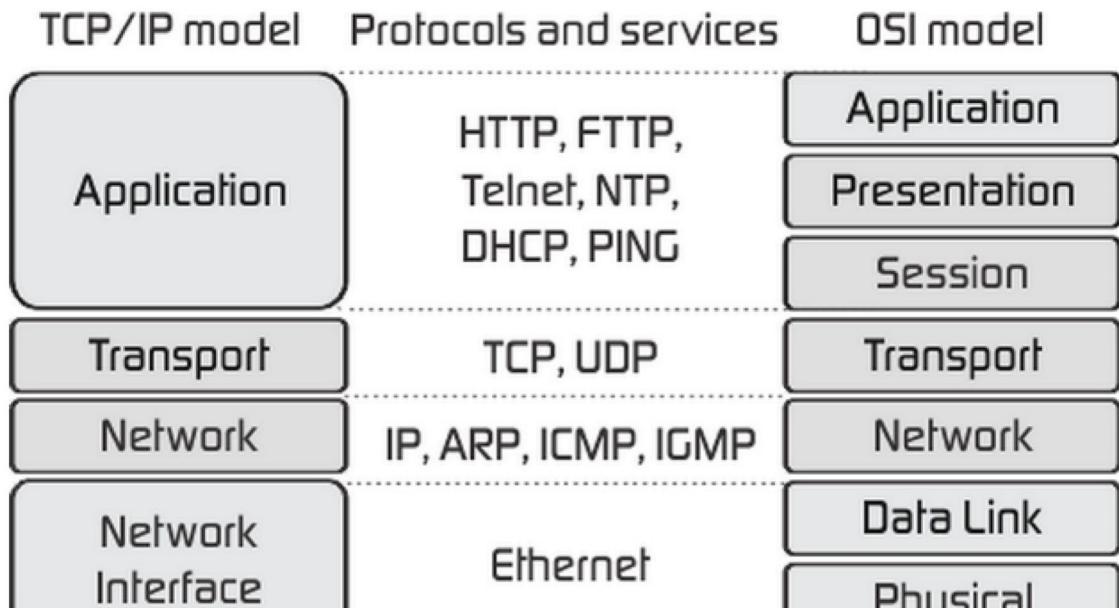


有了浏览器以后，上网是这样的。



TCP/IP模型

实现网络通信的基础是网络通信协议，这些协议通常是由[互联网工程任务组（ IETF ）](#)制定的。所谓“协议”就是通信计算机双方必须共同遵从的一组约定，例如怎样建立连接、怎样互相识别等，网络协议的三要素是：语法、语义和时序。构成我们今天使用的Internet的基础的是TCP/IP协议族，所谓协议族就是一系列的协议及其构成的通信模型，我们通常也把这套东西称为TCP/IP模型。与国际标准化组织发布的OSI/RM这个七层模型不同，TCP/IP是一个四层模型，也就是说，该模型将我们使用的网络从逻辑上分解为四个层次，自底向上依次是：网络接口层、网络层、传输层和应用层，如下图所示。



IP通常被翻译为网际协议，它服务于网络层，主要实现了寻址和路由的功能。接入网络的每一台主机都需要有自己的IP地址，IP地址就是主机在计算机网络上的身份标识。当然由于IPv4地址的匮乏，我们平常在家里、办公室以及其他可以接入网络的公共区域上网时获得的IP地址并不是全球唯一的IP地址，而是一个[局域网（ LAN ）](#)中的内部IP地址，通过[网络地址转换（ NAT ）服务](#)我们也可以实现对网络的访问。计算机网络上有大量的被我们称为“[路由器](#)”的网络中继设备，它们会存储转发我们发送到网络上的数据分组，让从源头发出的数据最终能够找到传送到目的地通路，这项功能就是所谓的路由。

TCP全称传输控制协议，它是基于IP提供的寻址和路由服务而建立起来的负责实现端到端可靠传输的协议，之所以将TCP称为可靠的传输协议是因为TCP向调用者承诺了三件事情：

1. 数据不传丢不传错（利用握手、校验和重传机制可以实现）。
2. 流量控制（通过滑动窗口匹配数据发送者和接收者之间的传输速度）。
3. 拥塞控制（通过RTT时间以及对滑动窗口的控制缓解网络拥堵）。

## 网络应用模式

1. C/S模式和B/S模式。这里的C指的是Client（客户端），通常是一个需要安装到某个宿主操作系统上的应用程序；而B指的是Browser（浏览器），它几乎是所有图形化操作系统都默认安装了一个应用软件；通过C或B都可以实现对S（服务器）的访

问。关于二者的比较和讨论在网络上有一大堆的文章，在此我们就不再浪费笔墨了。

2. 去中心化的网络应用模式。不管是B/S还是C/S都需要服务器的存在，服务器就是整个应用模式的中心，而去中心化的网络应用通常没有固定的服务器或者固定的客户端，所有应用的使用者既可以作为资源的提供者也可以作为资源的访问者。

## 基于HTTP协议的网络资源访问

### HTTP ( 超文本传输协议 )

HTTP是超文本传输协议 ( Hyper-Text Transfer Protocol ) 的简称，维基百科上对HTTP的解释是：超文本传输协议是一种用于分布式、协作式和超媒体信息系统的应用层协议，它是万维网数据通信的基础，设计HTTP最初的目的为了提供一种发布和接收HTML页面的方法，通过HTTP或者HTTPS ( 超文本传输安全协议 ) 请求的资源由URI ( 统一资源标识符 ) 来标识。关于HTTP的更多内容，我们推荐阅读阮一峰老师的[《HTTP 协议入门》](#)，简单的说，通过HTTP我们可以获取网络上的 ( 基于字符的 ) 资源，开发中经常会用到的网络API ( 有的地方也称之为网络数据接口 ) 就是基于HTTP来实现数据传输的。

### JSON格式

JSON ( JavaScript Object Notation ) 是一种轻量级的数据交换语言，该语言以易于让人阅读的文字 ( 纯文本 ) 为基础，用来传输由属性值或者序列性的值组成的数据对象。尽管JSON是最初只是Javascript中一种创建对象的字面量语法，但它在当下更是一种独立于语言的数据格式，很多编程语言都支持JSON格式数据的生成和解析，Python内置的json模块也提供了这方面的功能。由于JSON是纯文本，它和XML一样都适用于异构系统之间的数据交换，而相较于XML，JSON显得更加的轻便和优雅。下面是表达同样信息的XML和JSON，而JSON的优势是相当直观的。

XML的例子：

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
    <from>Alice</from>
    <to>Bob</to>
    <content>Will you marry me?</content>
</message>
```

JSON的例子：

```
{
    'from': 'Alice',
    'to': 'Bob',
    'content': 'Will you marry me?'
}
```

## requests库

requests是一个基于HTTP协议来使用网络的第三库，其[官方网站](#)有这样的一句介绍它的话：“Requests是唯一的一个**非转基因**的Python HTTP库，人类可以安全享用。”简单的说，使用requests库可以非常方便的使用HTTP，避免安全缺陷、冗余代码以及“重复发明轮子”（行业黑话，通常用在软件工程领域表示重新创造一个已有的或是早已被优化过的根本方法）。前面的文章中我们已经使用过这个库，下面我们还是通过requests来实现一个访问网络数据接口并从中获取美女图片下载链接然后下载美女图片到本地的例子程序，程序中使用了[天行数据](#)提供的网络API。

我们可以先通过pip安装requests及其依赖库。

```
pip install requests
```

如果使用PyCharm作为开发工具，可以直接在代码中书写`import requests`，然后通过代码修复功能来自动下载安装requests。

```
from time import time
from threading import Thread

import requests

# 继承Thread类创建自定义的线程类
class DownloadHanlder(Thread):

    def __init__(self, url):
        super().__init__()
        self.url = url

    def run(self):
        filename = self.url[self.url.rfind('/') + 1:]
        resp = requests.get(self.url)
        with open('/Users/Hao/' + filename, 'wb') as f:
            f.write(resp.content)

def main():
    # 通过requests模块的get函数获取网络资源
    # 下面的代码中使用了天行数据接口提供的网络API
    # 要使用该数据接口需要在天行数据的网站上注册
    # 然后用自己的Key替换掉下面代码中的APIKey即可
    resp = requests.get(
        'http://api.tianapi.com/meinv/?key=APIKey&num=10')
    # 将服务器返回的JSON格式的数据解析为字典
    data_model = resp.json()
    for mm_dict in data_model['newslist']:
        url = mm_dict['picUrl']
        # 通过多线程的方式实现图片下载
        DownloadHanlder(url).start()
```

```
if __name__ == '__main__':
    main()
```

## 基于传输层协议的套接字编程

套接字这个词对很多不了解网络编程的人来说显得非常晦涩和陌生，其实说得通俗点，套接字就是一套用C语言写成的应用程序开发库，主要用于实现进程间通信和网络编程，在网络应用开发中被广泛使用。在Python中也可以基于套接字来使用传输层提供的传输服务，并基于此开发自己的网络应用。实际开发中使用的套接字可以分为三类：流套接字（TCP套接字）、数据报套接字和原始套接字。

### TCP套接字

所谓TCP套接字就是使用TCP协议提供的传输服务来实现网络通信的编程接口。在Python中可以通过创建socket对象并指定type属性为SOCK\_STREAM来使用TCP套接字。由于一台主机可能拥有多个IP地址，而且很有可能会配置多个不同的服务，所以作为服务器端的程序，需要在创建套接字对象后将其绑定到指定的IP地址和端口上。这里的端口并不是物理设备而是对IP地址的扩展，用于区分不同的服务，例如我们通常将HTTP服务跟80端口绑定，而MySQL数据库服务默认绑定在3306端口，这样当服务器收到用户请求时就可以根据端口号来确定到底用户请求的是HTTP服务器还是数据库服务器提供的服务。端口的取值范围是0~65535，而1024以下的端口我们通常称之为“著名端口”（留给像FTP、HTTP、SMTP等“著名服务”使用的端口，有的地方也称之为“熟知端口”），自定义的服务通常不使用这些端口，除非自定义的是HTTP或FTP这样的著名服务。

下面的代码实现了一个提供时间日期的服务器。

```
from socket import socket, SOCK_STREAM, AF_INET
from datetime import datetime

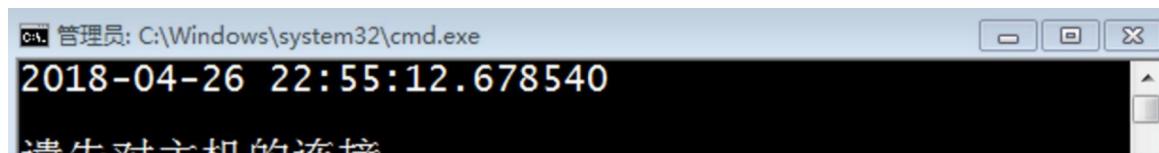
def main():
    # 1. 创建套接字对象并指定使用哪种传输服务
    # family=AF_INET - IPv4地址
    # family=AF_INET6 - IPv6地址
    # type=SOCK_STREAM - TCP套接字
    # type=SOCK_DGRAM - UDP套接字
    # type=SOCK_RAW - 原始套接字
    server = socket(family=AF_INET, type=SOCK_STREAM)
    # 2. 绑定IP地址和端口(端口用于区分不同的服务)
    # 同一时间在同一个端口上只能绑定一个服务否则报错
    server.bind(('192.168.1.2', 6789))
    # 3. 开启监听 - 监听客户端连接到服务器
    # 参数512可以理解为连接队列的大小
    server.listen(512)
    print('服务器启动开始监听...')
    while True:
```

```
# 4.通过循环接收客户端的连接并作出相应的处理(提供服务)
# accept方法是一个阻塞方法如果没有客户端连接到服务器代码不会向下执行
# accept方法返回一个元组其中的第一个元素是客户端对象
# 第二个元素是连接到服务器的客户端的地址(由IP和端口两部分构成)
client, addr = server.accept()
print(str(addr) + '连接到了服务器。')
# 5.发送数据
client.send(str(datetime.now()).encode('utf-8'))
# 6.断开连接
client.close()

if __name__ == '__main__':
    main()
```

运行服务器程序后我们可以通过Windows系统的telnet来访问该服务器，结果如下图所示。

```
telnet 192.168.1.2 6789
```



当然我们也可以通过Python的程序来实现TCP客户端的功能，相较于实现服务器程序，实现客户端程序就简单多了，代码如下所示。

```
from socket import socket

def main():
    # 1.创建套接字对象默认使用IPv4和TCP协议
    client = socket()
    # 2.连接到服务器(需要指定IP地址和端口)
```

```

client.connect(('192.168.1.2', 6789))
# 3.从服务器接收数据
print(client.recv(1024).decode('utf-8'))
client.close()

if __name__ == '__main__':
    main()

```

需要注意的是，上面的服务器并没有使用多线程或者异步I/O的处理方式，这也就意味着当服务器与一个客户端处于通信状态时，其他的客户端只能排队等待。很显然，这样的服务器并不能满足我们的需求，我们需要的服务器是能够同时接纳和处理多个用户请求的。下面我们来设计一个使用多线程技术处理多个用户请求的服务器，该服务器会向连接到服务器的客户端发送一张图片。

服务器端代码：

```

from socket import socket, SOCK_STREAM, AF_INET
from base64 import b64encode
from json import dumps
from threading import Thread

def main():

    # 自定义线程类
    class FileTransferHandler(Thread):

        def __init__(self, cclient):
            super().__init__()
            self.cclient = cclient

        def run(self):
            my_dict = {}
            my_dict['filename'] = 'guido.jpg'
            # JSON是纯文本不能携带二进制数据
            # 所以图片的二进制数据要处理成base64编码
            my_dict['filedata'] = data
            # 通过dumps函数将字典处理成JSON字符串
            json_str = dumps(my_dict)
            # 发送JSON字符串
            self.cclient.send(json_str.encode('utf-8'))
            self.cclient.close()

    # 1.创建套接字对象并指定使用哪种传输服务
    server = socket()
    # 2.绑定IP地址和端口(区分不同的服务)
    server.bind(('192.168.1.2', 5566))
    # 3.开启监听 - 监听客户端连接到服务器
    server.listen(512)
    print('服务器启动开始监听...')
    with open('guido.jpg', 'rb') as f:

```

```

# 将二进制数据处理成base64再解码成字符串
data = b64encode(f.read()).decode('utf-8')
while True:
    client, addr = server.accept()
    # 启动一个线程来处理客户端的请求
    FileTransferHandler(client).start()

if __name__ == '__main__':
    main()

```

客户端代码：

```

from socket import socket
from json import loads
from base64 import b64decode

def main():
    client = socket()
    client.connect(('192.168.1.2', 5566))
    # 定义一个保存二进制数据的对象
    in_data = bytes()
    # 由于不知道服务器发送的数据有多大每次接收1024字节
    data = client.recv(1024)
    while data:
        # 将收到的数据拼接起来
        in_data += data
        data = client.recv(1024)
    # 将收到的二进制数据解码成JSON字符串并转换成字典
    # loads函数的作用就是将JSON字符串转成字典对象
    my_dict = loads(in_data.decode('utf-8'))
    filename = my_dict['filename']
    filedatal = my_dict['filedata'].encode('utf-8')
    with open('/Users/Hao/' + filename, 'wb') as f:
        # 将base64格式的数据解码成二进制数据并写入文件
        f.write(b64decode(filedatal))
    print('图片已保存。')

if __name__ == '__main__':
    main()

```

在这个案例中，我们使用了JSON作为数据传输的格式（通过JSON格式对传输的数据进行了序列化和反序列化的操作），但是JSON并不能携带二进制数据，因此对图片的二进制数据进行了Base64编码的处理。Base64是一种用64个字符表示所有二进制数据的编码方式，通过将二进制数据每6位一组的方式重新组织，刚好可以使用0~9的数字、大小写字母以及“+”和“/”总共64个字符表示从 000000 到 111111 的64种状态。[维基百科](#)上有关于Base64编码的详细讲解，不熟悉Base64的读者可以自行阅读。

**说明**：上面的代码主要为了讲解网络编程的相关内容因此并没有对异常状况进行处理，请读者自行添加异常处理代码来增强程序的健壮性。

### UDP套接字

传输层除了有可靠的传输协议TCP之外，还有一种非常轻便的传输协议叫做用户数据报协议，简称UDP。TCP和UDP都是提供端到端传输服务的协议，二者的差别就如同打电话和发短信的区别，后者不对传输的可靠性和可达性做出任何承诺从而避免了TCP中握手和重传的开销，所以在强调性能和而不是数据完整性的场景中（例如传输网络音视频数据），UDP可能是更好的选择。可能大家会注意到一个现象，就是在观看网络视频时，有时会出现卡顿，有时会出现花屏，这无非就是部分数据传丢或传错造成的。在Python中也可以使用UDP套接字来创建网络应用，对此我们不进行赘述，有兴趣的读者可以自行研究。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day01-15 / Day14-B / 网络应用开发.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



122 lines (94 sloc) 5.1 KB

# 网络应用开发

## 发送电子邮件

在即时通信软件如此发达的今天，电子邮件仍然是互联网上使用最为广泛的应用之一，公司向应聘者发出录用通知、网站向用户发送一个激活账号的链接、银行向客户推广它们的理财产品等几乎都是通过电子邮件来完成的，而这些任务应该都是由程序自动完成的。

就像我们可以用HTTP（超文本传输协议）来访问一个网站一样，发送邮件要使用SMTP（简单邮件传输协议），SMTP也是一个建立在TCP（传输控制协议）提供的可靠数据传输服务的基础上的应用级协议，它规定了邮件的发送者如何跟发送邮件的服务器进行通信的细节，而Python中的smtplib模块将这些操作简化成了几个简单的函数。

下面的代码演示了如何在Python发送邮件。

```
from smtplib import SMTP
from email.header import Header
from email.mime.text import MIMEText

def main():
    # 请自行修改下面的邮件发送者和接收者
    sender = 'abcdefg@126.com'
    receivers = ['uvwxyz@qq.com', 'uvwxyz@126.com']
    message = MIMEText('用Python发送邮件的示例代码。', 'plain', 'utf-8')
    message['From'] = Header('王大锤', 'utf-8')
    message['To'] = Header('骆昊', 'utf-8')
    message['Subject'] = Header('示例代码实验邮件', 'utf-8')
    smtpser = SMTP('smtp.126.com')
    # 请自行修改下面的登录口令
    smtpser.login(sender, 'secretpass')
    smtpser.sendmail(sender, receivers, message.as_string())
    print('邮件发送完成!')
```

```
if __name__ == '__main__':
    main()
```

如果要发送带有附件的邮件，那么可以按照下面的方式进行操作。

```
from smtplib import SMTP
from email.header import Header
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart

import urllib

def main():
    # 创建一个带附件的邮件消息对象
    message = MIMEMultipart()

    # 创建文本内容
    text_content = MIMEText('附件中有本月数据请查收', 'plain', 'utf-8')
    message['Subject'] = Header('本月数据', 'utf-8')
    # 将文本内容添加到邮件消息对象中
    message.attach(text_content)

    # 读取文件并将文件作为附件添加到邮件消息对象中
    with open('/Users/Hao/Desktop/hello.txt', 'rb') as f:
        txt = MIMEText(f.read(), 'base64', 'utf-8')
        txt['Content-Type'] = 'text/plain'
        txt['Content-Disposition'] = 'attachment; filename=hello.txt'
        message.attach(txt)
    # 读取文件并将文件作为附件添加到邮件消息对象中
    with open('/Users/Hao/Desktop/汇总数据.xlsx', 'rb') as f:
        xls = MIMEText(f.read(), 'base64', 'utf-8')
        xls['Content-Type'] = 'application/vnd.ms-excel'
        xls['Content-Disposition'] = 'attachment; filename=month-data.xlsx'
        message.attach(xls)

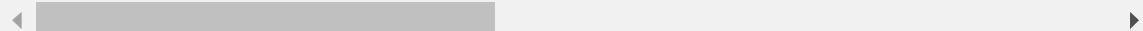
    # 创建SMTP对象
    smtper = SMTP('smtp.126.com')
    # 开启安全连接
    # smtper.starttls()
    sender = 'abcdefg@126.com'
    receivers = ['uvwxyz@qq.com']
    # 登录到SMTP服务器
    # 请注意此处不是使用密码而是邮件客户端授权码进行登录
    # 对此有疑问的读者可以联系自己使用的邮件服务器客服
    smtper.login(sender, 'secretpass')
    # 发送邮件
    smtper.sendmail(sender, receivers, message.as_string())
    # 与邮件服务器断开连接
    smtper.quit()
```

```
print('发送完成!')\n\nif __name__ == '__main__':\n    main()
```

## 发送短信

发送短信也是项目中常见的功能，网站的注册码、验证码、营销信息基本上都是通过短信来发送给用户的。在下面的代码中我们使用了[互亿无线](#)短信平台（该平台为注册用户提供了50条免费短信以及常用开发语言发送短信的demo，可以登录该网站并在用户自服务页面中对短信进行配置）提供的API接口实现了发送短信的服务，当然国内的短信平台很多，读者可以根据自己的需要进行选择（通常会考虑费用预算、短信到达率、使用的难易程度等指标），如果需要在商业项目中使用短信服务建议购买短信平台提供的套餐服务。

```
import urllib.parse\nimport http.client\nimport json\n\n\ndef main():\n    host = "106.ihuyi.com"\n    sms_send_uri = "/webservice/sms.php?method=Submit"\n    # 下面的参数需要填入自己注册的账号和对应的密码\n    params = urllib.parse.urlencode({'account': '你自己的账号', 'password' : '你自己的\n    print(params)\n    headers = {'Content-type': 'application/x-www-form-urlencoded', 'Accept': 'text/pl\n    conn = http.client.HTTPConnection(host, port=80, timeout=30)\n    conn.request('POST', sms_send_uri, params, headers)\n    response = conn.getresponse()\n    response_str = response.read()\n    jsonstr = response_str.decode('utf-8')\n    print(json.loads(jsonstr))\n    conn.close()\n\nif __name__ == '__main__':\n    main()
```



Branch: master ▾

[Find file](#)[Copy path](#)

## Python-100-Days / Day01-15 / Day15 / 图像和办公文档处理.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

122 lines (84 sloc) 4.57 KB

# 图像和办公文档处理

用程序来处理图像和办公文档经常出现在实际开发中，Python的标准库中虽然没有直接支持这些操作的模块，但我们可以借助Python生态圈中的第三方模块来完成这些操作。

## 操作图像

### 计算机图像相关知识

1. 颜色。如果你有使用颜料画画的经历，那么一定知道混合红、黄、蓝三种颜料可以得到其他颜色，事实上这三种颜色就是我们称为美术三原色的东西，它们是不能再分解的基本颜色。在计算机中，我们可以将红、绿、蓝三种色光以不同的比例叠加来组合成其他颜色，因此这三种颜色就是色光三原色，所以我们通常会将一个颜色表示为一个RGB值或RGBA值（其中的A表示Alpha通道，它决定了透过这个图像的像素，也就是透明度）。

名称	RGB值	名称	RGB值
White	(255, 255, 255, 255)	Red	(255, 0, 0, 255)
Green	(0, 255, 0, 255)	Blue	(0, 0, 255, 255)
Gray	(128, 128, 128, 255)	Yellow	(255, 255, 0, 255)
Black	(0, 0, 0, 255)	Purple	(128, 0, 128, 255)

2. 像素。对于一个由数字序列表示的图像来说，最小的单位就是图像上单一颜色的小方格，这些小方块都有一个明确的位置和被分配的色彩数值，而这些一小方格的颜色和位置决定了该图像最终呈现出来的样子，它们是不可分割的单位，我们通常称之为像素（pixel）。每一个图像都包含了一定量的像素，这些像素决定图像在屏幕上所呈现的大小。

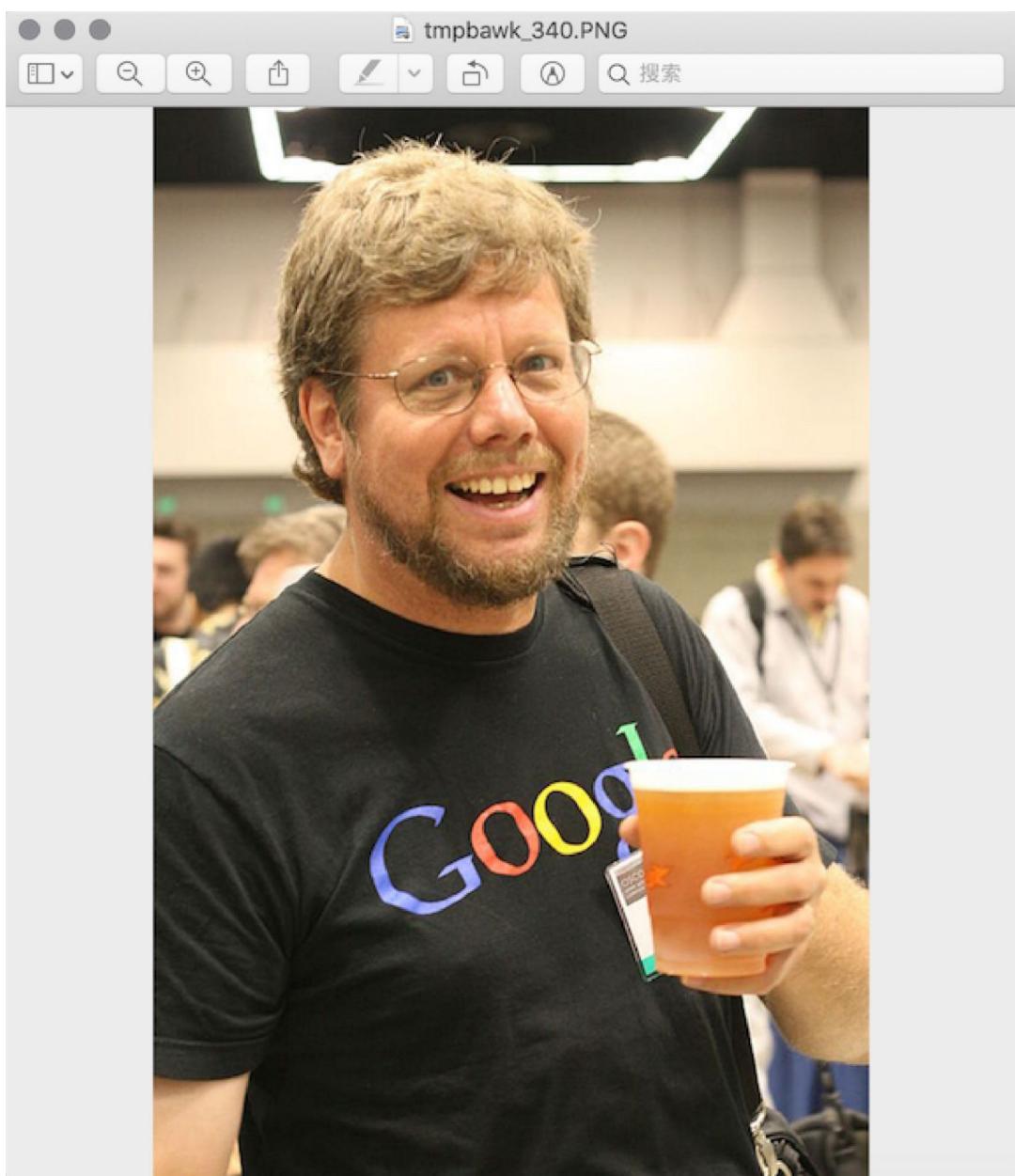
### 用Pillow操作图像

Pillow是由从著名的Python图像处理库PIL发展出来的一个分支，通过Pillow可以实现图像压缩和图像处理等各种操作。可以使用下面的命令来安装Pillow。

```
pip install pillow
```

Pillow中最为重要的是Image类，读取和处理图像都要通过这个类来完成。

```
>>> from PIL import Image  
>>>  
>>> image = Image.open('./res/guido.jpg')  
>>> image.format, image.size, image.mode  
('JPEG', (500, 750), 'RGB')  
>>> image.show()
```



## 1. 剪裁图像

```
>>> image = Image.open('./res/guido.jpg')
>>> rect = 80, 20, 310, 360
>>> image.crop(rect).show()
```



## 2. 生成缩略图

```
>>> image = Image.open('./res/guido.jpg')
>>> size = 128, 128
>>> image.thumbnail(size)
>>> image.show()
```



## 3. 缩放和黏贴图像

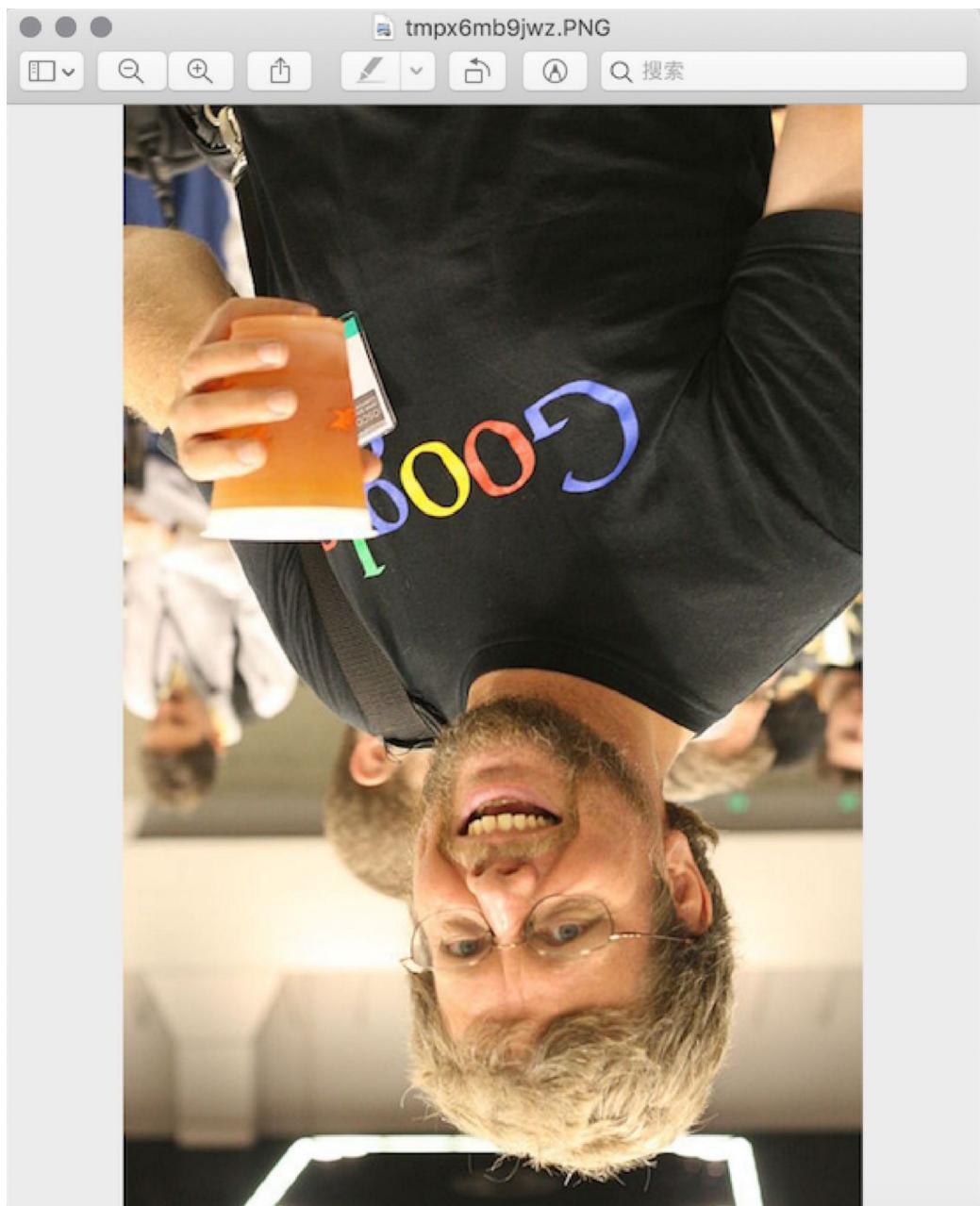
```
>>> image1 = Image.open('./res/luohao.png')
>>> image2 = Image.open('./res/guido.jpg')
```

```
>>> rect = 80, 20, 310, 360
>>> guido_head = image2.crop(rect)
>>> width, height = guido_head.size
>>> image1.paste(guido_head.resize((int(width / 1.5), int(height / 1.5))), (172, 4
```



#### 4. 旋转和翻转

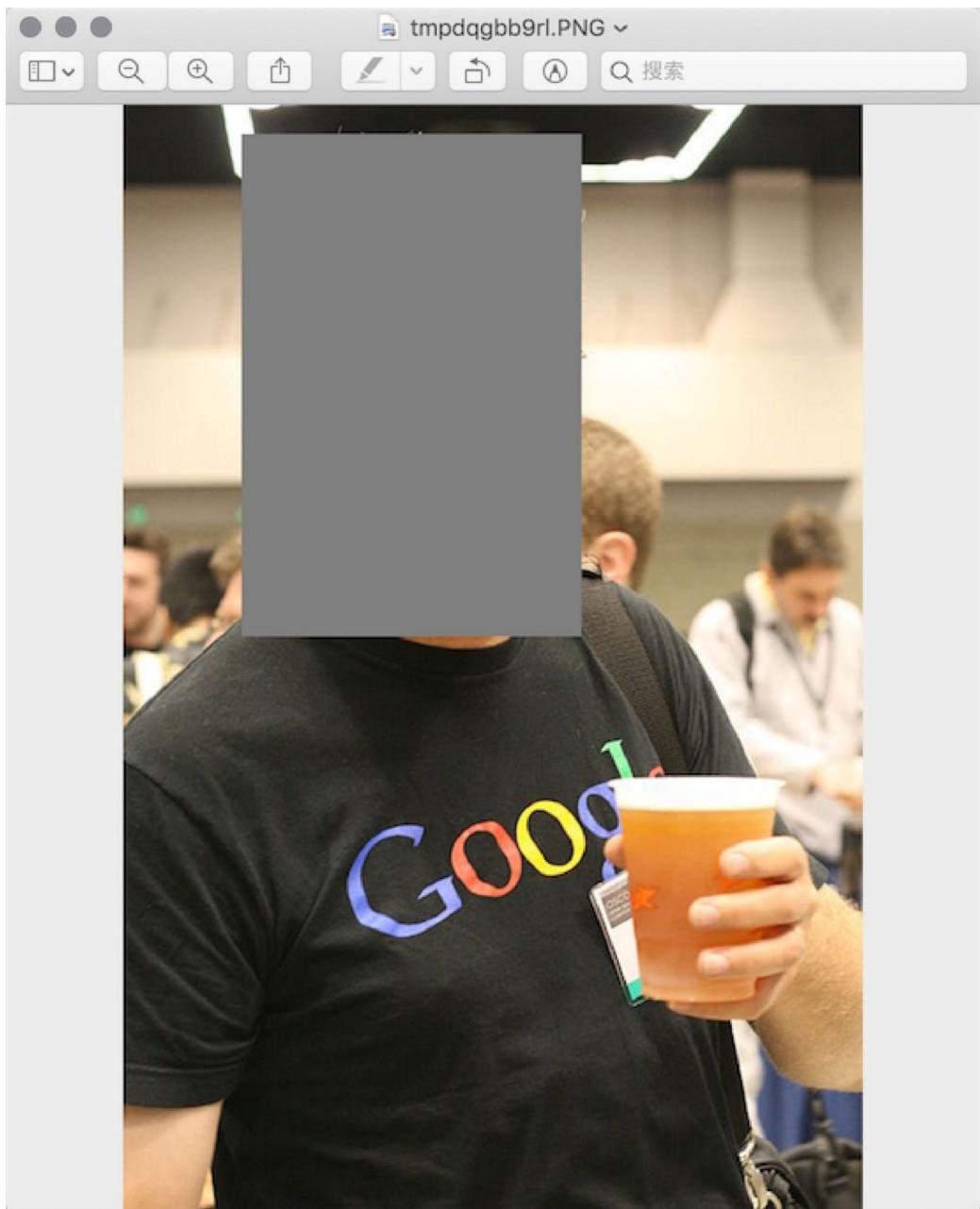
```
>>> image = Image.open('./res/guido.png')
>>> image.rotata(180).show()
>>> image.transpose(Image.FLIP_LEFT_RIGHT).show()
```





## 5. 操作像素

```
>>> image = Image.open('./res/guido.jpg')
>>> for x in range(80, 310):
...     for y in range(20, 360):
...         image.putpixel((x, y), (128, 128, 128))
...
>>> image.show()
```



## 6. 濾鏡效果

```
>>> from PIL import Image, ImageFilter  
>>>  
>>> image = Image.open('./res/guido.jpg')  
>>> image.filter(ImageFilter.CONTOUR).show()
```



## 处理Excel电子表格

Python的openpyxl模块让我们可以在Python程序中读取和修改Excel电子表格，当然实际工作中，我们可能会用LibreOffice Calc和OpenOffice Calc来处理Excel的电子表格文件，这就意味着openpyxl模块也能处理来自这些软件生成的电子表格。关于openpyxl的使用手册和使用文档可以查看它的[官方文档](#)。

## 处理Word文档

利用python-docx模块，Pytho 可以创建和修改Word文档，当然这里的Word文档不仅仅是指通过微软的Office软件创建的扩展名为docx的文档，LibreOffice Writer和OpenOffice Writer都是免费的字处理软件。

## 处理PDF文档

PDF是Portable Document Format的缩写，使用.pdf作为文件扩展名。接下来我们就研究一下如何通过Python实现从PDF读取文本内容和从已有的文档生成新的PDF文件。

# 16-20天：学习python进阶内容

Branch: master ▾

[Find file](#) [Copy path](#)

[Python-100-Days / Day16-20 / Python语言进阶.md](#)

Fetching contributors...

Cannot retrieve contributors at this time.

Raw

Blame

History

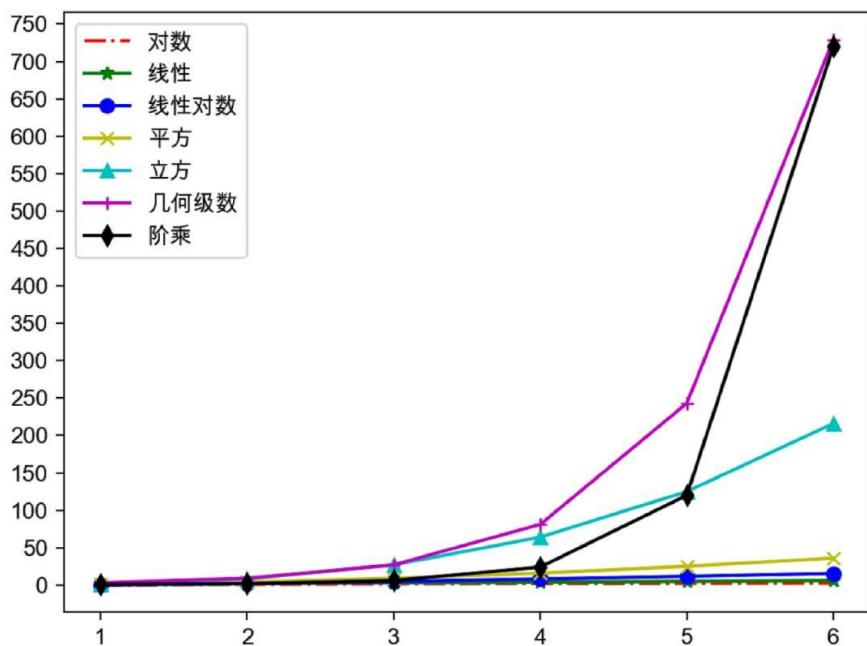
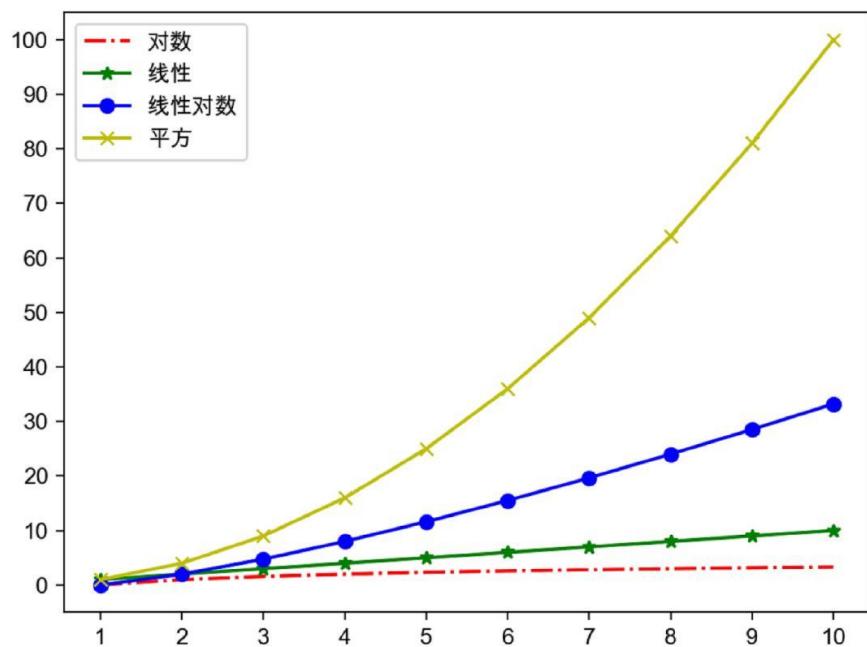


1349 lines (1037 sloc) 45.6 KB

## Python语言进阶

### 1. 数据结构和算法

- 算法：解决问题的方法和步骤
- 评价算法的好坏：渐近时间复杂度和渐近空间复杂度。
- 渐近时间复杂度的大O标记：
  - $O(c)$  - 常量时间复杂度 - 布隆过滤器 / 哈希存储
  - $O(\log_2 n)$  - 对数时间复杂度 - 折半查找 (二分查找)
  - $O(n)$  - 线性时间复杂度 - 顺序查找 / 桶排序
  - $O(n * \log_2 n)$  - 对数线性时间复杂度 - 高级排序算法 (归并排序、快速排序)
  - $O(n^2)$  - 平方时间复杂度 - 简单排序算法 (选择排序、插入排序、冒泡排序)
  - $O(n^3)$  - 立方时间复杂度 - Floyd算法 / 矩阵乘法运算
  - $O(2^n)$  - 几何级数时间复杂度 - 汉诺塔
  - $O(n!)$  - 阶乘时间复杂度 - 旅行经销商问题 - NP



- 排序算法（选择、冒泡和归并）和查找算法（顺序和折半）

```
def select_sort(origin_items, comp=lambda x, y: x < y):
    """简单选择排序"""
    items = origin_items[:]
    for i in range(len(items) - 1):
        min_index = i
        for j in range(i + 1, len(items)):
```

```

        if comp(items[j], items[min_index]):
            min_index = j
        items[i], items[min_index] = items[min_index], items[i]
    return items

def bubble_sort(origin_items, comp=lambda x, y: x > y):
    """高质量冒泡排序(搅拌排序)"""
    items = origin_items[:]
    for i in range(len(items) - 1):
        swapped = False
        for j in range(i, len(items) - 1 - i):
            if comp(items[j], items[j + 1]):
                items[j], items[j + 1] = items[j + 1], items[j]
                swapped = True
        if swapped:
            swapped = False
            for j in range(len(items) - 2 - i, i, -1):
                if comp(items[j - 1], items[j]):
                    items[j], items[j - 1] = items[j - 1], items[j]
                    swapped = True
        if not swapped:
            break
    return items

def merge_sort(items, comp=lambda x, y: x <= y):
    """归并排序(分治法)"""
    if len(items) < 2:
        return items[:]
    mid = len(items) // 2
    left = merge_sort(items[:mid], comp)
    right = merge_sort(items[mid:], comp)
    return merge(left, right, comp)

def merge(items1, items2, comp):
    """合并(将两个有序的列表合并成一个有序的列表)"""
    items = []
    index, index2 = 0, 0
    while index1 < len(items1) and index2 < len(items2):
        if comp(items1[index1], items2[index2]):
            items.append(items1[index1])
            index1 += 1
        else:
            items.append(items2[index2])
            index2 += 1
    items += items1[index1:]
    items += items2[index2:]
    return items

```

```

def seq_search(items, key):
    """顺序查找"""
    for index, item in enumerate(items):
        if item == key:
            return index
    return -1

def bin_search(items, key):
    """折半查找"""
    start, end = 0, len(items) - 1
    while start <= end:
        mid = (start + end) // 2
        if key > items[mid]:
            start = mid + 1
        elif key < items[mid]:
            end = mid - 1
        else:
            return mid
    return -1

```

- 使用生成式（推导式）语法

```

prices = {
    'AAPL': 191.88,
    'GOOG': 1186.96,
    'IBM': 149.24,
    'ORCL': 48.44,
    'ACN': 166.89,
    'FB': 208.09,
    'SYMC': 21.29
}
# 用股票价格大于100元的股票构造一个新的字典
prices2 = {key: value for key, value in prices.items() if value > 100}
print(prices2)

```

说明：生成式（推导式）可以用来生成列表、集合和字典。

- 嵌套的列表

```

names = ['关羽', '张飞', '赵云', '马超', '黄忠']
courses = ['语文', '数学', '英语']
# 录入五个学生三门课程的成绩
# 错误 - 参考http://pythontutor.com/visualize.html#mode=edit
# scores = [[None] * len(courses)] * len(names)
scores = [[None] * len(courses) for _ in range(len(names))]
for row, name in enumerate(names):
    for col, course in enumerate(courses):
        scores[row][col] = float(input(f'请输入{name}的{course}成绩: '))
print(scores)

```

## Python Tutor - VISUALIZE CODE AND GET LIVE HELP

- o heapq、itertools等的用法

```
"""
从列表中找出最大的或最小的N个元素
堆结构(大根堆/小根堆)
"""

import heapq

list1 = [34, 25, 12, 99, 87, 63, 58, 78, 88, 92]
list2 = [
    {'name': 'IBM', 'shares': 100, 'price': 91.1},
    {'name': 'AAPL', 'shares': 50, 'price': 543.22},
    {'name': 'FB', 'shares': 200, 'price': 21.09},
    {'name': 'HPQ', 'shares': 35, 'price': 31.75},
    {'name': 'YHOO', 'shares': 45, 'price': 16.35},
    {'name': 'ACME', 'shares': 75, 'price': 115.65}
]
print(heapq.nlargest(3, list1))
print(heapq.nsmallest(3, list1))
print(heapq.nlargest(2, list2, key=lambda x: x['price']))
print(heapq.nlargest(2, list2, key=lambda x: x['shares']))
```

```
"""
迭代工具 - 排列 / 组合 / 笛卡尔积
"""

import itertools
```

```
itertools.permutations('ABCD')
itertools.combinations('ABCDE', 3)
itertools.product('ABCD', '123')
```

- o collections模块下的工具类

```
"""
找出序列中出现次数最多的元素
"""

from collections import Counter

words = [
    'look', 'into', 'my', 'eyes', 'look', 'into', 'my', 'eyes',
    'the', 'eyes', 'the', 'eyes', 'the', 'eyes', 'not', 'around',
    'the', 'eyes', "don't", 'look', 'around', 'the', 'eyes',
    'look', 'into', 'my', 'eyes', "you're", 'under'
]
counter = Counter(words)
print(counter.most_common(3))
```

○ 常用算法：

- 穷举法 - 又称为暴力破解法，对所有的可能性进行验证，直到找到正确答案。
- 贪婪法 - 在对问题求解时，总是做出在当前看来是最好的选择，不追求最优解，快速找到满意解。
- 分治法 - 把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题，直到可以直接求解的程度，最后将子问题的解进行合并得到原问题的解。
- 回溯法 - 回溯法又称为试探法，按选优条件向前搜索，当搜索到某一步发现原先选择并不优或达不到目标时，就退回一步重新选择。
- 动态规划 - 基本思想也是将待求解问题分解成若干个子问题，先求解并保存这些子问题的解，避免产生大量的重复运算。

穷举法例子：百钱百鸡和五人分鱼。

```
# 公鸡5元一只 母鸡3元一只 小鸡1元三只
# 用100元买100只鸡 问公鸡/母鸡/小鸡各多少只
for x in range(20):
    for y in range(33):
        z = 100 - x - y
        if 5 * x + 3 * y + z // 3 == 100 and z % 3 == 0:
            print(x, y, z)

# A、B、C、D、E五人在某天夜里合伙捕鱼 最后疲惫不堪各自睡觉
# 第二天A第一个醒来 他将鱼分为5份 扔掉多余的1条 拿走自己的一份
# B第二个醒来 也将鱼分为5份 扔掉多余的1条 拿走自己的一份
# 然后C、D、E依次醒来也按同样的方式分鱼 问他们至少捕了多少条鱼
fish = 1
while True:
    total = fish
    enough = True
    for _ in range(5):
        if (total - 1) % 5 == 0:
            total = (total - 1) // 5 * 4
        else:
            enough = False
            break
    if enough:
        print(fish)
        break
    fish += 1
```

贪婪法例子：假设小偷有一个背包，最多能装20公斤赃物，他闯入一户人家，发现如下表所示的物品。很显然，他不能把所有物品都装进背包，所以必须确定拿走哪些物品，留下哪些物品。

名称	价格（美元）	重量（kg）
----	--------	--------

电脑	200	20
收音机	20	4
钟	175	10
花瓶	50	2
书	10	1
油画	90	9

....

贪婪法：在对问题求解时，总是做出在当前看来是最好的选择，不追求最优解，快速找到一个满意解。

输入：  
 20 6  
 电脑 200 20  
 收音机 20 4  
 钟 175 10  
 花瓶 50 2  
 书 10 1  
 油画 90 9  
 ....

```

class Thing(object):
    """物品"""

    def __init__(self, name, price, weight):
        self.name = name
        self.price = price
        self.weight = weight

    @property
    def value(self):
        """价格重量比"""
        return self.price / self.weight

def input_thing():
    """输入物品信息"""
    name_str, price_str, weight_str = input().split()
    return name_str, int(price_str), int(weight_str)

def main():
    """主函数"""
    max_weight, num_of_things = map(int, input().split())
    all_things = []
    for _ in range(num_of_things):
        all_things.append(Thing(*input_thing()))
    all_things.sort(key=lambda x: x.value, reverse=True)
    total_weight = 0
    total_price = 0
    for thing in all_things:
        if total_weight + thing.weight >= max_weight:
            break
        total_weight += thing.weight
        total_price += thing.price
    print(total_price)

```

```

        if total_weight + thing.weight <= max_weight:
            print(f'小偷拿走了{thing.name}')
            total_weight += thing.weight
            total_price += thing.price
        print(f'总价值: {total_price}美元')

if __name__ == '__main__':
    main()

```

## 分治法例子：快速排序。

```

"""
快速排序 - 选择枢轴对元素进行划分，左边都比枢轴小右边都比枢轴大
"""

def quick_sort(origin_items, comp=lambda x, y: x <= y):
    items = origin_items[:]
    _quick_sort(items, 0, len(items) - 1, comp)
    return items


def _quick_sort(items, start, end, comp):
    if start < end:
        pos = _partition(items, start, end, comp)
        _quick_sort(items, start, pos - 1, comp)
        _quick_sort(items, pos + 1, end, comp)

def _partition(items, start, end, comp):
    pivot = items[end]
    i = start - 1
    for j in range(start, end):
        if comp(items[j], pivot):
            i += 1
            items[i], items[j] = items[j], items[i]
    items[i + 1], items[end] = items[end], items[i + 1]
    return i + 1

```

## 回溯法例子：骑士巡逻。

```

"""
递归回溯法：叫称为试探法，按选优条件向前搜索，当搜索到某一步，发现原先选择并不
"""

import sys
import time

SIZE = 5
total = 0

```

```

def print_board(board):
    for row in board:
        for col in row:
            print(str(col).center(4), end=' ')
    print()

def patrol(board, row, col, step=1):
    if row >= 0 and row < SIZE and \
       col >= 0 and col < SIZE and \
       board[row][col] == 0:
        board[row][col] = step
        if step == SIZE * SIZE:
            global total
            total += 1
            print(f'第{total}种走法: ')
            print_board()
            patrol(board, row - 2, col - 1, step + 1)
            patrol(board, row - 1, col - 2, step + 1)
            patrol(board, row + 1, col - 2, step + 1)
            patrol(board, row + 2, col - 1, step + 1)
            patrol(board, row + 2, col + 1, step + 1)
            patrol(board, row + 1, col + 2, step + 1)
            patrol(board, row - 1, col + 2, step + 1)
            patrol(board, row - 2, col + 1, step + 1)
        board[row][col] = 0

def main():
    board = [[0] * SIZE for _ in range(SIZE)]
    patrol(board, SIZE - 1, SIZE - 1)

if __name__ == '__main__':
    main()

```

动态规划例子1：斐波拉切数列。（不使用动态规划将会是几何级数复杂度）

```

"""
动态规划 - 适用于有重叠子问题和最优子结构性质的问题
使用动态规划方法所耗时间往往远少于朴素解法(用空间换取时间)
"""

def fib(num, temp={}):
    """用递归计算Fibonacci数"""
    if num in (1, 2):
        return 1
    try:
        return temp[num]
    except KeyError:
        temp[num] = fib(num - 1) + fib(num - 2)
        return temp[num]

```

动态规划例子2：子列表元素之和的最大值。（使用动态规划可以避免二重循环）

说明：子列表指的是列表中索引（下标）连续的元素构成的列表；列表中的元素是int类型，可能包含正整数、0、负整数；程序输入列表中的元素，输出子列表元素求和的最大值，例如：

输入：1 -2 3 5 -3 2

输出：8

输入：0 -2 3 5 -1 2

输出：9

输入：-9 -2 -3 -5 -3

输出：-2

```
def main():
    items = list(map(int, input().split()))
    size = len(items)
    overall, partial = {}, {}
    overall[size - 1] = partial[size - 1] = items[size - 1]
    for i in range(size - 2, -1, -1):
        partial[i] = max(items[i], partial[i + 1] + items[i])
        overall[i] = max(partial[i], overall[i + 1])
    print(overall[0])

if __name__ == '__main__':
    main()
```

## 2. 函数的使用方式

- 将函数视为“一等公民”
  - 函数可以赋值给变量
  - 函数可以作为函数的参数
  - 函数可以作为函数的返回值
- 高阶函数的用法（filter、map以及它们的替代品）

```
items1 = list(map(lambda x: x ** 2, filter(lambda x: x % 2, range(1, 10))))
items2 = [x ** 2 for x in range(1, 10) if x % 2]
```

- 位置参数、可变参数、关键字参数、命名关键字参数

- 参数的元信息（代码可读性问题）
- 匿名函数和内联函数的用法（lambda 函数）
- 闭包和作用域问题
  - Python 搜索变量的LEGB顺序（Local --> Embedded --> Global --> Built-in）
  - global 和 nonlocal 关键字的作用
 

global : 声明或定义全局变量（要么直接使用现有的全局作用域的变量，要么定义一个变量放到全局作用域）。

nonlocal : 声明使用嵌套作用域的变量（嵌套作用域必须存在该变量，否则报错）。
- 装饰器函数（使用装饰器和取消装饰器）

例子：输出函数执行时间的装饰器。

```
def record_time(func):
    """自定义装饰函数的装饰器"""

    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time()
        result = func(*args, **kwargs)
        print(f'{func.__name__}: {time() - start}秒')
        return result

    return wrapper
```

如果装饰器不希望跟 print 函数耦合，可以编写带参数的装饰器。

```
from functools import wraps
from time import time

def record(output):
    """自定义带参数的装饰器"""

    def decorate(func):

        @wraps(func)
        def wrapper(*args, **kwargs):
            start = time()
            result = func(*args, **kwargs)
            output(func.__name__, time() - start)
            return result
```

```

        return wrapper

    return decorate

from functools import wraps
from time import time

class Record():
    """自定义装饰器类(通过__call__魔术方法使得对象可以当成函数调用)"""

    def __init__(self, output):
        self.output = output

    def __call__(self, func):

        @wraps(func)
        def wrapper(*args, **kwargs):
            start = time()
            result = func(*args, **kwargs)
            self.output(func.__name__, time() - start)
            return result

        return wrapper

```

说明：由于对带装饰功能的函数添加了@wraps装饰器，可以通过func.\_\_wrapped\_\_方式获得被装饰之前的函数或类来取消装饰器的作用。

例子：用装饰器来实现单例模式。

```

from functools import wraps

def singleton(cls):
    """装饰类的装饰器"""
    instances = {}

    @wraps(cls)
    def wrapper(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]

    return wrapper

@singleton
class President():
    """总统(单例类)"""
    pass

```

说明：上面的代码中用到了闭包（closure），不知道你是否已经意识到了。还没有一个小问题就是，上面的代码并没有实现线程安全的单例，如果要实现线程安全的单例应该怎么做呢？

```
from functools import wraps

def singleton(cls):
    """线程安全的单例装饰器"""
    instances = {}
    locker = Lock()

    @wraps(cls)
    def wrapper(*args, **kwargs):
        if cls not in instances:
            with locker:
                if cls not in instances:
                    instances[cls] = cls(*args, **kwargs)
        return instances[cls]

    return wrapper
```

### 3. 面向对象相关知识

- 三大支柱：封装、继承、多态

例子：工资结算系统。

```
"""
月薪结算系统 - 部门经理每月15000 程序员每小时200 销售员1800底薪加销售额5%提成
"""

from abc import ABCMeta, abstractmethod

class Employee(metaclass=ABCMeta):
    """员工(抽象类)"""

    def __init__(self, name):
        self.name = name

    @abstractmethod
    def get_salary(self):
        """结算月薪(抽象方法)"""
        pass

class Manager(Employee):
    """部门经理"""

    def get_salary(self):
```

```
        return 15000.0

class Programmer(Employee):
    """程序员"""

    def __init__(self, name, working_hour=0):
        self.working_hour = working_hour
        super().__init__(name)

    def get_salary(self):
        return 200.0 * self.working_hour

class Salesman(Employee):
    """销售员"""

    def __init__(self, name, sales=0.0):
        self.sales = sales
        super().__init__(name)

    def get_salary(self):
        return 1800.0 + self.sales * 0.05

class EmployeeFactory():
    """创建员工的工厂（工厂模式 - 通过工厂实现对对象使用者和对象之间的解耦合）"""

    @staticmethod
    def create(emp_type, *args, **kwargs):
        """创建员工"""
        emp_type = emp_type.upper()
        emp = None
        if emp_type == 'M':
            emp = Manager(*args, **kwargs)
        elif emp_type == 'P':
            emp = Programmer(*args, **kwargs)
        elif emp_type == 'S':
            emp = Salesman(*args, **kwargs)
        return emp

def main():
    """主函数"""
    emps = [
        EmployeeFactory.create('M', '曹操'),
        EmployeeFactory.create('P', '荀彧', 120),
        EmployeeFactory.create('P', '郭嘉', 85),
        EmployeeFactory.create('S', '典韦', 123000),
    ]
    for emp in emps:
        print('%s: %.2f元' % (emp.name, emp.get_salary()))
```

```
if __name__ == '__main__':
    main()
```



- 类与类之间的关系

- is-a关系：继承
- has-a关系：关联 / 聚合 / 合成
- use-a关系：依赖

例子：扑克游戏。

```
"""
经验：符号常量总是优于字面常量，枚举类型是定义符号常量的最佳选择
"""

from enum import Enum, unique

import random


@unique
class Suite(Enum):
    """花色"""
    SPADE, HEART, CLUB, DIAMOND = range(4)

    def __lt__(self, other):
        return self.value < other.value


class Card():
    """牌"""

    def __init__(self, suite, face):
        """初始化方法"""
        self.suite = suite
        self.face = face

    def show(self):
        """显示牌面"""
        suites = ['♠', '♥', '♦', '♦']
        faces = ['', 'A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J',
                 'Q', 'K']
        return f'{suites[self.suite.value]} {faces[self.face]}'

    def __str__(self):
        return self.show()

    def __repr__(self):
        return self.show()


class Poker():
```

```

"""扑克"""

def __init__(self):
    self.index = 0
    self.cards = [Card(suite, face)
                 for suite in Suite
                 for face in range(1, 14)]

def shuffle(self):
    """洗牌（随机乱序）"""
    random.shuffle(self.cards)
    self.index = 0

def deal(self):
    """发牌"""
    card = self.cards[self.index]
    self.index += 1
    return card

@property
def has_more(self):
    return self.index < len(self.cards)

class Player():
    """玩家"""

    def __init__(self, name):
        self.name = name
        self.cards = []

    def get_one(self, card):
        """摸一张牌"""
        self.cards.append(card)

    def sort(self, comp=lambda card: (card.suite, card.face)):
        """整理手上的牌"""
        self.cards.sort(key=comp)

def main():
    """主函数"""
    poker = Poker()
    poker.shuffle()
    players = [Player('东邪'), Player('西毒'), Player('南帝'), Player('北丐')]
    while poker.has_more:
        for player in players:
            player.get_one(poker.deal())
    for player in players:
        player.sort()
        print(player.name, end=': ')
        print(player.cards)

```

```
if __name__ == '__main__':
    main()
```



- 对象的复制（深复制/深拷贝/深度克隆和浅复制/浅拷贝/影子克隆）
- 垃圾回收、循环引用和弱引用

Python使用了自动化内存管理，这种管理机制以**引用计数**为基础，同时也引入了**标记-清除**和**分代收集**两种机制为辅的策略。

```
typedef struct_object {
    /* 引用计数 */
    int ob_refcnt;
    /* 对象指针 */
    struct_typeobject *ob_type;
} PyObject;

/* 增加引用计数的宏定义 */
#define Py_INCREF(op) ((op)->ob_refcnt++)
/* 减少引用计数的宏定义 */
#define Py_DECREF(op) \
    if (--(op)->ob_refcnt != 0) \
        ; \
    else \
        __Py_Dealloc((PyObject *) (op))
```

导致引用计数+1的情况：

- 对象被创建，例如 `a = 23`
- 对象被引用，例如 `b = a`
- 对象被作为参数，传入到一个函数中，例如 `f(a)`
- 对象作为一个元素，存储在容器中，例如 `list1 = [a, a]`

导致引用计数-1的情况：

- 对象的别名被显式销毁，例如 `del a`
- 对象的别名被赋予新的对象，例如 `a = 24`
- 一个对象离开它的作用域，例如f函数执行完毕时，f函数中的局部变量（全局变量不会）
- 对象所在的容器被销毁，或从容器中删除对象

引用计数可能会导致循环引用问题，而循环引用会导致内存泄露，如下面的代码所示。为了解决这个问题，Python中引入了“标记-清除”和“分代收集”。在创建一个对象的时候，对象被放在第一代中，如果在第一代的垃圾检查中对象存活了下来，该对象就会被放到第二代中，同理在第二代的垃圾检查中对象存活下来，该对象就会被放到第三代中。

```
# 循环引用会导致内存泄露 - Python除了引用技术还引入了标记清理和分代回收
# 在Python 3.6以前如果重写__del__魔术方法会导致循环引用处理失效
# 如果不想造成循环引用可以使用弱引用
list1 = []
list2 = []
list1.append(list2)
list2.append(list1)
```

以下情况会导致垃圾回收：

- 调用 `gc.collect()`
- `gc`模块的计数器达到阈值
- 程序退出

如果循环引用中两个对象都定义了 `__del__` 方法，`gc`模块不会销毁这些不可达对象，因为`gc`模块不知道应该先调用哪个对象的 `__del__` 方法，这个问题在 Python 3.6 中得到了解决。

也可以通过 `weakref` 模块构造弱引用的方式来解决循环引用的问题。

- 魔法属性和方法（请参考《Python魔法方法指南》）

有几个小问题请大家思考：

- 自定义的对象能不能使用运算符做运算？
  - 自定义的对象能不能放到`set`中？能去重吗？
  - 自定义的对象能不能作为`dict`的键？
  - 自定义的对象能不能使用上下文语法？
- 混入（Mixin）

例子：自定义字典限制只有在指定的key不存在时才能在字典中设置键值对。

```
class SetOnceMappingMixin():
    """自定义混入类"""
    __slots__ = ()

    def __setitem__(self, key, value):
        if key in self:
            raise KeyError(str(key) + ' already set')
        return super().__setitem__(key, value)
```

```

class SetOnceDict(SetOnceMappingMixin, dict):
    """自定义字典"""
    pass

my_dict = SetOnceDict()
try:
    my_dict['username'] = 'jackfrued'
    my_dict['username'] = 'hellokitty'
except KeyError:
    pass
print(my_dict)

```

- 元编程和元类

例子：用元类实现单例模式。

```

import threading

class SingletonMeta(type):
    """自定义元类"""

    def __init__(cls, *args, **kwargs):
        cls.__instance = None
        cls.__lock = threading.Lock()
        super().__init__(*args, **kwargs)

    def __call__(cls, *args, **kwargs):
        if cls.__instance is None:
            with cls.__lock:
                if cls.__instance is None:
                    cls.__instance = super().__call__(*args, **kwargs)
        return cls.__instance

class President(metaclass=SingletonMeta):
    """总统(单例类)"""
    pass

```

- 面向对象设计原则

- 单一职责原则 ( SRP ) - 一个类只做该做的事情 ( 类的设计要高内聚 )
- 开闭原则 ( OCP ) - 软件实体应该对扩展开发对修改关闭
- 依赖倒转原则 ( DIP ) - 面向抽象编程 ( 在弱类型语言中已经被弱化 )
- 里氏替换原则 ( LSP ) - 任何时候可以用子类对象替换掉父类对象
- 接口隔离原则 ( ISP ) - 接口要小而专不要大而全 ( Python 中没有接口的概念 )

- 合成聚合复用原则 ( CARP ) - 优先使用强关联关系而不是继承关系复用代码
- 最少知识原则 ( 迪米特法则 , LoD ) - 不要给没有必然联系的对象发消息  
说明 : 上面加粗的字母放在一起称为面向对象的SOLID原则。

- GoF设计模式

- 创建型模式 : 单例、工厂、建造者、原型
- 结构型模式 : 适配器、门面 ( 外观 ) 、代理
- 行为型模式 : 迭代器、观察者、状态、策略

例子 : 可插拔的哈希算法。

```
class StreamHasher():
    """哈希摘要生成器(策略模式)"""

    def __init__(self, alg='md5', size=4096):
        self.size = size
        alg = alg.lower()
        self.hasher = getattr(__import__('hashlib'), alg.lower())()

    def __call__(self, stream):
        return self.to_digest(stream)

    def to_digest(self, stream):
        """生成十六进制形式的摘要"""
        for buf in iter(lambda: stream.read(self.size), b''):
            self.hasher.update(buf)
        return self.hasher.hexdigest()

def main():
    """主函数"""
    hasher1 = StreamHasher()
    with open('Python-3.7.1.tgz', 'rb') as stream:
        print(hasher1.to_digest(stream))
    hasher2 = StreamHasher('sha1')
    with open('Python-3.7.1.tgz', 'rb') as stream:
        print(hasher2(stream))

if __name__ == '__main__':
    main()
```

#### 4. 迭代器和生成器

- 和迭代器相关的魔术方法 ( \_\_iter\_\_ 和 \_\_next\_\_ )
- 两种创建生成器的方式 ( 生成器表达式和 yield 关键字 )

```

def fib(num):
    """生成器"""
    a, b = 0, 1
    for _ in range(num):
        a, b = b, a + b
        yield a

class Fib(object):
    """迭代器"""

    def __init__(self, num):
        self.num = num
        self.a, self.b = 0, 1
        self.idx = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.idx < self.num:
            self.a, self.b = self.b, self.a + self.b
            self.idx += 1
            return self.a
        raise StopIteration()

```

## 5. 并发编程

Python中实现并发编程的三种方案：多线程、多进程和异步I/O。并发编程的好处在于可以提升程序的执行效率以及改善用户体验；坏处在于并发的程序不容易开发和调试，同时对其他程序来说它并不友好。

- 多线程：Python中提供了Thread类并辅以Lock、Condition、Event、Semaphore和Barrier。Python中有GIL来防止多个线程同时执行本地字节码，这个锁对于CPython是必须的，因为CPython的内存管理并不是线程安全的，因为GIL的存在多线程并不能发挥CPU的多核特性。

.....

面试题：进程和线程的区别和联系？

进程 - 操作系统分配内存的基本单位 - 一个进程可以包含一个或多个线程

线程 - 操作系统分配CPU的基本单位

并发编程（concurrent programming）

1. 提升执行性能 - 让程序中没有因果关系的部分可以并发的执行
2. 改善用户体验 - 让耗时间的操作不会造成程序的假死

.....

```

import glob
import os
import threading

```

```

from PIL import Image

```

```

PREFIX = 'thumbnails'

def generate_thumbnail(infile, size, format='PNG'):
    """生成指定图片文件的缩略图"""
    file, ext = os.path.splitext(infile)
    file = file[file.rfind('/') + 1:]
    outfile = f'{PREFIX}/{file}_{size[0]}_{size[1]}.{ext}'
    img = Image.open(infile)
    img.thumbnail(size, Image.ANTIALIAS)
    img.save(outfile, format)

def main():
    """主函数"""
    if not os.path.exists(PREFIX):
        os.mkdir(PREFIX)
    for infile in glob.glob('images/*.png'):
        for size in (32, 64, 128):
            # 创建并启动线程
            threading.Thread(
                target=generate_thumbnail,
                args=(infile, (size, size))
            ).start()

if __name__ == '__main__':
    main()

```

## 多个线程竞争资源的情况

```

"""
多线程程序如果没有竞争资源处理起来通常也比较简单
当多个线程竞争临界资源的时候如果缺乏必要的保护措施就会导致数据错乱
说明：临界资源就是被多个线程竞争的资源
"""

import time
import threading

from concurrent.futures import ThreadPoolExecutor

class Account(object):
    """银行账户"""

    def __init__(self):
        self.balance = 0.0
        self.lock = threading.Lock()

    def deposit(self, money):
        # 通过锁保护临界资源
        with self.lock:
            new_balance = self.balance + money

```

```

        time.sleep(0.001)
        self.balance = new_balance

class AddMoneyThread(threading.Thread):
    """自定义线程类"""

    def __init__(self, account, money):
        self.account = account
        self.money = money
        # 自定义线程的初始化方法中必须调用父类的初始化方法
        super().__init__()

    def run(self):
        # 线程启动之后要执行的操作
        self.account.deposit(self.money)

def main():
    """主函数"""
    account = Account()
    # 创建线程池
    pool = ThreadPoolExecutor(max_workers=10)
    futures = []
    for _ in range(100):
        # 创建线程的第1种方式
        # threading.Thread(
        #     target=account.deposit, args=(1, )
        # ).start()
        # 创建线程的第2种方式
        # AddMoneyThread(account, 1).start()
        # 创建线程的第3种方式
        # 调用线程池中的线程来执行特定的任务
        future = pool.submit(account.deposit, 1)
        futures.append(future)
    # 关闭线程池
    pool.shutdown()
    for future in futures:
        future.result()
    print(account.balance)

if __name__ == '__main__':
    main()

```

修改上面的程序，启动5个线程向账户中存钱，5个线程从账户中取钱，取钱时如果余额不足就暂停线程进行等待。为了达到上述目标，需要对存钱和取钱的线程进行调度，在余额不足时取钱的线程暂停并释放锁，而存钱的线程将钱存入后要通知取钱的线程，使其从暂停状态被唤醒。可以使用 `threading` 模块的 `Condition` 来实现线程调度，该对象也是基于锁来创建的，代码如下所示：

....

多个线程竞争一个资源 - 保护临界资源 - 锁（Lock/RLock）

多个线程竞争多个资源（线程数>资源数） - 信号量（Semaphore）  
多个线程的调度 - 暂停线程执行/唤醒等待中的线程 - Condition

```
"""
from concurrent.futures import ThreadPoolExecutor
from random import randint
from time import sleep

import threading


class Account():
    """银行账户"""

    def __init__(self, balance=0):
        self.balance = balance
        lock = threading.Lock()
        self.condition = threading.Condition(lock)

    def withdraw(self, money):
        """取钱"""
        with self.condition:
            while money > self.balance:
                self.condition.wait()
            new_balance = self.balance - money
            sleep(0.001)
            self.balance = new_balance

    def deposit(self, money):
        """存钱"""
        with self.condition:
            new_balance = self.balance + money
            sleep(0.001)
            self.balance = new_balance
            self.condition.notify_all()

    def add_money(account):
        while True:
            money = randint(5, 10)
            account.deposit(money)
            print(threading.current_thread().name,
                  ':', money, '====>', account.balance)
            sleep(0.5)

    def sub_money(account):
        while True:
            money = randint(10, 30)
            account.withdraw(money)
            print(threading.current_thread().name,
                  ':', money, '<====', account.balance)
            sleep(1)

def main():
```

```

account = Account()
with ThreadPoolExecutor(max_workers=10) as pool:
    for _ in range(5):
        pool.submit(add_money, account)
        pool.submit(sub_money, account)

if __name__ == '__main__':
    main()

```

- 多进程：多进程可以有效的解决GIL的问题，实现多进程主要的类是Process，其他辅助的类跟threading模块中的类似，进程间共享数据可以使用管道、套接字等，在multiprocessing模块中有一个Queue类，它基于管道和锁机制提供了多个进程共享的队列。下面是官方文档上关于多进程和进程池的一个示例。

```

"""
多进程和进程池的使用
多线程因为GIL的存在不能够发挥CPU的多核特性
对于计算密集型任务应该考虑使用多进程
time python3 example22.py
real    0m11.512s
user    0m39.319s
sys     0m0.169s
使用多进程后实际执行时间为11.512秒，而用户时间39.319秒约为实际执行时间的4倍
这就证明我们的程序通过多进程使用了CPU的多核特性，而且这台计算机配置了4核的CPU
"""

import concurrent.futures
import math

PRIMES = [
    1116281,
    1297337,
    104395303,
    472882027,
    533000389,
    817504243,
    982451653,
    112272535095293,
    112582705942171,
    112272535095293,
    115280095190773,
    115797848077099,
    1099726899285419
] * 5

def is_prime(n):
    """判断素数"""
    if n % 2 == 0:
        return False

    sqrt_n = int(math.floor(math.sqrt(n)))

```

```

for i in range(3, sqrt_n + 1, 2):
    if n % i == 0:
        return False
return True

def main():
    """主函数"""
    with concurrent.futures.ProcessPoolExecutor() as executor:
        for number, prime in zip(PRIMES, executor.map(is_prime, PRIMES)):
            print('%d is prime: %s' % (number, prime))

if __name__ == '__main__':
    main()

```

### 说明：多线程和多进程的比较。

以下情况需要使用多线程：

- a. 程序需要维护许多共享的状态（尤其是可变状态），Python中的列表、字典、集合都是线程安全的，所以使用线程而不是进程维护共享状态的代价相对较小。
- b. 程序会花费大量时间在I/O操作上，没有太多并行计算的需求且不需占用太多的内存。

以下情况需要使用多进程：

- a. 程序执行计算密集型任务（如：字节码操作、数据处理、科学计算）。
- b. 程序的输入可以并行的分成块，并且可以将运算结果合并。
- c. 程序在内存使用方面没有任何限制且不强依赖于I/O操作（如：读写文件、套接字等）。
- 异步处理：从调度程序的任务队列中挑选任务，该调度程序以交叉的形式执行这些任务，我们并不能保证任务将以某种顺序去执行，因为执行顺序取决于队列中的一项任务是否愿意将CPU处理时间让位给另一项任务。异步任务通常通过多任务协作处理的方式来实现，由于执行时间和顺序的不确定，因此需要通过回调式编程或者 `future` 对象来获取任务执行的结果。Python 3通过 `asyncio` 模块和 `await` 和 `async` 关键字（在Python 3.7中正式被列为关键字）来支持异步处理。

```

...
异步I/O - async / await
...
import asyncio

```

```

def num_generator(m, n):
    """指定范围的数字生成器"""
    yield from range(m, n + 1)

async def prime_filter(m, n):
    """素数过滤器"""
    primes = []
    for i in num_generator(m, n):
        flag = True
        for j in range(2, int(i ** 0.5 + 1)):
            if i % j == 0:
                flag = False
                break
        if flag:
            print('Prime =>', i)
            primes.append(i)

        await asyncio.sleep(0.001)
    return tuple(primes)

async def square_mapper(m, n):
    """平方映射器"""
    squares = []
    for i in num_generator(m, n):
        print('Square =>', i * i)
        squares.append(i * i)

    await asyncio.sleep(0.001)
    return squares

def main():
    """主函数"""
    loop = asyncio.get_event_loop()
    future = asyncio.gather(prime_filter(2, 100), square_mapper(1, 100))
    future.add_done_callback(lambda x: print(x.result()))
    loop.run_until_complete(future)
    loop.close()

if __name__ == '__main__':
    main()

```

说明：上面的代码使用 `get_event_loop` 函数获得系统默认的事件循环，通过 `gather` 函数可以获得一个 `future` 对象，`future` 对象的 `add_done_callback` 可以添加执行完成时的回调函数，`loop` 对象的 `run_until_complete` 方法可以等待通过 `future` 对象获得协程执行结果。

Python中有一个名为[aiohttp](#)的三方库，它提供了异步的HTTP客户端和服务器，这个三方库可以跟[asyncio](#)模块一起工作，并提供了对[Future](#)对象的支持。Python 3.6中引入了[async](#)和[await](#)来定义异步执行的函数以及创建异步上下文，在Python 3.7中它们正式成为了关键字。下面的代码异步的从5个URL中获取页面并通过正则表达式的命名捕获组提取了网站的标题。

```
import asyncio
import re

import aiohttp

PATTERN = re.compile(r'<title>(?P<title>.**)</title>')

async def fetch_page(session, url):
    async with session.get(url, ssl=False) as resp:
        return await resp.text()

async def show_title(url):
    async with aiohttp.ClientSession() as session:
        html = await fetch_page(session, url)
        print(PATTERN.search(html).group('title'))

def main():
    urls = ('https://www.python.org/',
            'https://git-scm.com/',
            'https://www.jd.com/',
            'https://www.taobao.com/',
            'https://www.douban.com/')
    loop = asyncio.get_event_loop()
    tasks = [show_title(url) for url in urls]
    loop.run_until_complete(asyncio.wait(tasks))
    loop.close()

if __name__ == '__main__':
    main()
```

### 说明：异步I/O与多进程的比较。

当程序不需要真正的并发性或并行性，而是更多的依赖于异步处理和回调时，[asyncio](#)就是一种很好的选择。如果程序中有大量的等待与休眠时，也应该考虑[asyncio](#)，它很适合编写没有实时数据处理需求的Web应用服务器。

Python还有很多用于处理并行任务的三方库，例如：joblib、PyMP等。实际开发中，要提升系统的可扩展性和并发性通常有垂直扩展（增加单个节点的处理能力）和水平扩展（将单个节点变成多个节点）两种做法。可以通过消息队列来实现应用程序的解耦合，消息队列相当于是多线程同步队列的扩展版本，不同机器上的应用程序相当于就是线程，而共享的分布式消息队列就是原来程序中的Queue。消息队列（面向消息的中间件）的最流行和最标准化的实现是AMQP（高级消息队列协议），AMQP源于金融行业，提供了排队、路由、可靠传输、安全等功能，最著名的实现包括：Apache的ActiveMQ、RabbitMQ等。

要实现任务的异步化，可以使用名为Celery的三方库。Celery是Python编写的分布式任务队列，它使用分布式消息进行工作，可以基于RabbitMQ或Redis来作为后端的消息代理。

## Python-100-Days / Day21-30 / Web前端概述.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



881 lines (720 sloc) 22.4 KB

## Web前端概述

说明：本文使用的一部分插图来自Jon Duckett先生的\*HTML and CSS: Design and Build Websites\*一书，这是一本非常棒的前端入门书，有兴趣的读者可以在亚马逊或者其他网站上找到该书的购买链接。

### HTML简史

1. 1991年10月：一个非正式CERN（[欧洲核子研究中心](#)）文件首次公开18个HTML标签，这个文件的作者是物理学家[蒂姆·伯纳斯-李](#)，因此他是[万维网](#)的发明者，也是[万维网联盟](#)的主席。
2. 1995年11月：HTML 2.0标准发布（RFC 1866）。
3. 1997年1月：HTML 3.2作为[W3C](#)推荐标准发布。
4. 1997年12月：HTML 4.0作为W3C推荐标准发布。
5. 1999年12月：HTML4.01作为W3C推荐标准发布。
6. 2008年1月：HTML5由W3C作为工作草案发布。
7. 2011年5月：W3C将HTML5推进至“最终征求”（Last Call）阶段。
8. 2012年12月：W3C指定HTML5作为“候选推荐”阶段。
9. 2014年10月：HTML5作为稳定W3C推荐标准发布，这意味着HTML5的标准化已经完成。

### HTML5新特性

1. 引入原生多媒体支持（audio和video标签）
2. 引入可编程内容（canvas标签）
3. 引入语义Web（article、aside、details、figure、footer、header、nav、section、summary等标签）
4. 引入新的表单控件（日历、邮箱、搜索、滑条等）
5. 引入对离线存储更好的支持（localStorage和sessionStorage）

## 6. 引入对定位、拖放、WebSocket、后台任务等的支持

### 使用标签承载内容

#### 结构

- head
  - title
  - meta
- body

#### 文本

- 标题和段落
  - h1 ~ h6
  - p
- 上标和下标
  - sup
  - sub
- 空白 ( 白色空间折叠 )
- 折行和水平标尺
  - br
  - hr
- 语义化标签
  - 加粗和强调 - strong
  - 引用 - blockquote
  - 缩写词和首字母缩写词 - abbr / acronym
  - 引文 - cite
  - 所有者联系信息 - address
  - 内容的修改 - ins / del

#### 列表 ( list )

- 有序列表 ( ordered list ) - ol / li
- 无序列表 ( unordered list ) - ul / li
- 定义列表 ( definition list ) - dl / dt / dd

#### 链接 ( anchor )

- 页面链接
- 锚链接
- 功能链接

## 图像 ( image )

- 图像存储位置
- 图像及其宽高
- 选择正确的图像格式
  - JPEG
  - GIF
  - PNG
- 矢量图
- 语义化标签 - figure / figcaption

## 表格 ( table )

- 基本的表格结构 - table / tr / td
- 表格的标题 - caption
- 跨行和跨列 - rowspan属性 / colspan属性
- 长表格 - thead / tbody / tfoot

## 表单 ( form )

- 重要属性 - action / method
- 表单控件 ( input ) - type属性
  - 文本框 - text / 密码框 - password / 数字框 - number
  - 邮箱 - email / 电话 - tel / 日期 - date / 滑条 - range / URL - url / 搜索 - search
  - 单选按钮 - radio / 复选按钮 - checkbox
  - 文件上传 - file / 隐藏域 ( 埋点 ) - hidden
  - 提交按钮 - submit / 图像按钮 - image / 重置按钮 - reset
- 下拉列表 - select / option
- 文本域 ( 多行文本 ) - textarea
- 组合表单元素 - fieldset / legend

## 音视频 ( audio / video )

- 视频格式和播放器
- 视频托管服务
- 添加视频的准备工作
- video标签和属性 - autoplay / controls / loop / muted / preload / src

- audio标签和属性 - autoplay / controls / loop / muted / preload / src / width / height / poster

## 其他

- 文档类型
- 注释
- 属性
  - id
  - class
- 块级元素 / 行级元素
- 内联框架 ( internal frame )
- 字符实体 ( 实体替换符 )

<	Less-than sign amp;lt; amp;#60;	¢	Cent sign amp;cent; amp;#162;	'	Left single quote amp;lsquo; amp;#8216;
>	Greater-than sign amp;gt; amp;#62;	£	Pound sign amp;pound; amp;#163;	'	Right single quote amp;rsquo; amp;#8217;
&	Ampersand amp; amp;#38;	¥	Yen sign amp;yen; amp;#165;	"	Left double quotes amp;ldquo; amp;#8220;
"	Quotation mark amp;quot; amp;#34;	€	Euro sign amp;euro; amp;#8364;	"	Right double quotes amp;rdquo; amp;#8221;
(C)	Copyright symbol amp;copy; amp;#169;	X	Multiplication sign amp;times; amp;#215;		
(R)	Registered trademark amp;reg; amp;#174;	÷	Division sign amp;divide; amp;#247;		
TM	Trademark amp;trade; amp;#8482;				

## 使用CSS渲染页面

## 简介

- CSS的作用
- CSS的工作原理
- 规则、属性和值



- 常用选择器

SELECTOR	MEANING	EXAMPLE
UNIVERSAL SELECTOR	Applies to all elements in the document	* {} Targets all elements on the page
TYPE SELECTOR	Matches element names	h1, h2, h3 {} Targets the <h1>, <h2> and <h3> elements
CLASS SELECTOR	Matches an element whose class attribute has a value that matches the one specified after the period (or full stop) symbol	.note {} Targets any element whose class attribute has a value of note p.note {} Targets only <p> elements whose class attribute has a value of note
ID SELECTOR	Matches an element whose id attribute has a value that matches the one specified after the pound or hash symbol	#introduction {} Targets the element whose id attribute has a value of introduction
CHILD SELECTOR	Matches an element that is a direct child of another	li>a {} Targets any <a> elements that are children of an <li> element (but not other <a> elements in the page)
DESCENDANT SELECTOR	Matches an element that is a descendent of another specified element (not just a direct child of that element)	p a {} Targets any <a> elements that sit inside a <p> element, even if there are other elements nested between them
ADJACENT SIBLING SELECTOR	Matches an element that is the next sibling of another	h1+p {} Targets the first <p> element after any <h1> element (but not other <p> elements)
GENERAL SIBLING SELECTOR	Matches an element that is a sibling of another, although it does not have to be the directly preceding element	h1~p {} If you had two <p> elements that are siblings of an <h1> element, this rule would apply to both

## 颜色 ( color )

- 如何指定颜色
- 颜色术语和颜色对比
- 背景色

## 文本 ( text / font )

- 文本的大小和字型(font-size / font-family)

PIXELS	PERCENTAGES	EMS		
<b>TWELVE PIXEL SCALE</b>				
<pre> h1    24px h2    18px h3    14px body  12px </pre>	=	<pre> h1    200% h2    150% h3    117% body  75% </pre>	=	<pre> h1    1.5em h2    1.3em h3    1.17em body  100% p     0.75em </pre>
<b>SIXTEEN PIXEL SCALE</b>				
<pre> h1    32px h2    24px h3    18px body  16px </pre>	=	<pre> h1    200% h2    150% h3    133% body  100% </pre>	=	<pre> h1    2em h2    1.5em h3    1.125em body  100% p     1em </pre>

SERIF	SANS-SERIF	MONOSPACE
<p>Serif fonts have extra details on the ends of the main strokes of the letters. These details are known as serifs.</p> 	<p>Sans-serif fonts have straight ends to letters, and therefore have a much cleaner design.</p> 	<p>Every letter in a monospace (or fixed-width) font is the same width. (Non-monospace fonts have different widths.)</p> 
<p>In print, serif fonts were traditionally used for long passages of text because they were considered easier to read.</p>	<p>Screens have a lower resolution than print. So, if the text is small, sans-serif fonts can be clearer to read.</p>	<p>Monospace fonts are commonly used for code because they align nicely, making the text easier to follow.</p>

- 粗细、样式、拉伸和装饰(font-weight / font-style / font-stretch / text-decoration)



- 行间距(line-height)、字母间距(letter-spacing)和单词间距(word-spacing)
- 对齐(text-align)方式和缩进(text-indent)
- 链接样式 (:link / :visited / :active / :hover )
- CSS3新属性
  - 阴影效果 - text-shadow
  - 首字母和首行文本(:first-letter / :first-line)
  - 响应用户

## 盒子 ( box model )

- 盒子大小的控制 ( width / height )

PIXELS	PERCENTAGES	EMS
<b>TWELVE PIXEL SCALE</b>		
h1 24px h2 18px h3 14px body 12px	= h1 200% h2 150% h3 117% body 75%	 h1 1.5em h2 1.3em h3 1.17em body 100% p 0.75em
<b>SIXTEEN PIXEL SCALE</b>		
h1 32px h2 24px h3 18px body 16px	= h1 200% h2 150% h3 133% body 100%	 h1 2em h2 1.5em h3 1.125em body 100% p 1em

- 盒子的边框、外边距和内边距 ( border / margin / padding )



- 盒子的显示和隐藏 ( display / visibility )
- CSS3新属性
  - 边框图像 ( border-image )
  - 投影 ( border-shadow )
  - 圆角 ( border-radius )

## 列表、表格和表单

- 列表的项目符号 ( list-style )
- 表格的边框和背景 ( border-collapse )
- 表单控件的外观
- 表单控件的对齐
- 浏览器的开发者工具

## 图像

- 控制图像的大小 ( display: inline-block )
- 对齐图像

- 背景图像 ( background / background-image / background-repeat / background-position )

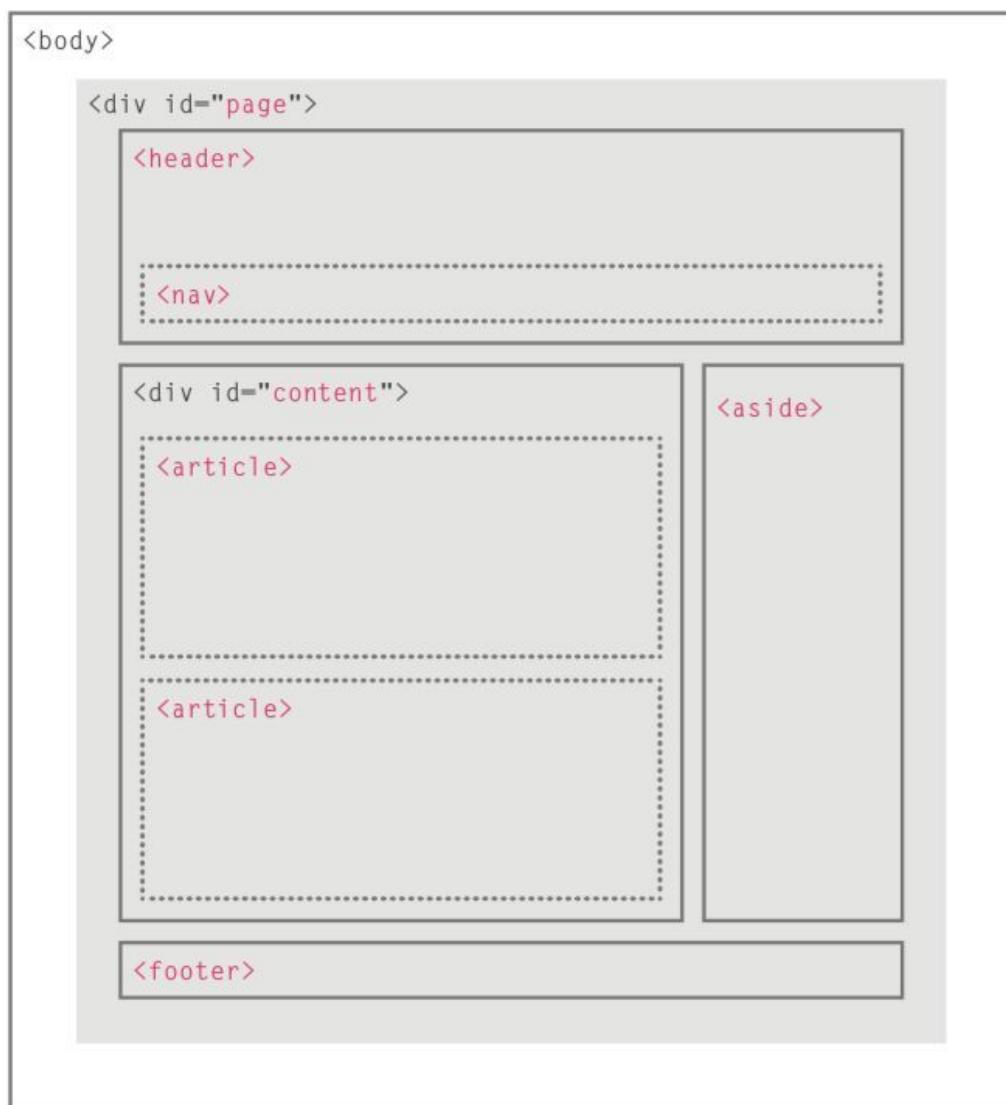
## 布局

- 控制元素的位置 ( position / z-index )

- 普通流
- 相对定位
- 绝对定位
- 固定定位
- 浮动元素 ( float / clear )

- 网站布局

- HTML5布局



- 适配屏幕尺寸

- 固定宽度布局
- 流体布局
- 布局网格

## 使用JavaScript控制行为

### JavaScript基本语法

- 语句和注释
- 变量和数据类型
  - 声明和赋值
  - 简单数据类型和复杂数据类型
  - 变量的命名规则
- 表达式和运算符
  - 赋值运算符
  - 算术运算符
  - 比较运算符
  - 逻辑运算符
- 分支结构
  - if...else...
  - switch...cas...default...
- 循环结构
  - for 循环
  - while 循环
  - do...while 循环
- 数组
  - 创建数组
  - 操作数组中的元素
- 函数
  - 声明函数
  - 调用函数
  - 参数和返回值
  - 匿名函数
  - 立即调用函数

### 面向对象

- 对象的概念
- 创建对象的字面量语法
- 访问成员运算符

- 创建对象的构造函数语法
  - `this` 关键字
- 添加和删除属性
  - `delete` 关键字
- 标准对象
  - `Number / String / Boolean / Symbol / Array / Function`
  - `Date / Error / Math / RegEx / Object / Map / Set`
  - `JSON / Promise / Generator / Reflect / Proxy`

## BOM

- `window` 对象的属性和方法
- `history` 对象
  - `forward() / back() / go()`
- `location` 对象
- `navigator` 对象
- `screen` 对象

## DOM

- DOM树
- 访问元素
  - `getElementById() / querySelector()`
  - `getElementsByClassName() / getElementsByTagName() / querySelectorAll()`
  - `parentNode / previousSibling / nextSibling / children / firstChild / lastChild`
- 操作元素
  - `nodeValue`
  - `innerHTML / textContent / createElement() / createTextNode() / appendChild() / insertBefore() / removeChild()`
  - `className / id / hasAttribute() / getAttribute() / setAttribute() / removeAttribute()`
- 事件处理
  - 事件类型
    - UI事件：`load / unload / error / resize / scroll`
    - 键盘事件：`keydown / keyup / keypress`
    - 鼠标事件：`click / dblclick / mousedown / mouseup / mousemove / mouseover / mouseout`
    - 焦点事件：`focus / blur`
    - 表单事件：`input / change / submit / reset / cut / copy / paste / select`

- 事件绑定
  - HTML事件处理程序（不推荐使用，因为要做到标签与代码分离）
  - 传统的DOM事件处理程序（只能附加一个回调函数）
  - 事件监听器（旧的浏览器中不被支持）
- 事件流：事件捕获 / 事件冒泡
- 事件对象（低版本IE中的window.event）
  - target（有些浏览器使用srcElement）
  - type
  - cancelable
  - preventDefault()
  - stopPropagation()（低版本IE中的cancelBubble）
- 鼠标事件 - 事件发生的位置
  - 屏幕位置：screenX 和 screenY
  - 页面位置：pageX 和 pageY
  - 客户端位置：clientX 和 clientY
- 键盘事件 - 哪个键被按下了
  - keyCode 属性（有些浏览器使用 which）
  - String.fromCharCode(event.keyCode)
- HTML5事件
  - DOMContentLoaded
  - hashchange
  - beforeunload

## JavaScript API

- 客户端存储 - localStorage 和 sessionStorage

```
localStorage.colorSetting = '#a4509b';
localStorage['colorSetting'] = '#a4509b';
localStorage.setItem('colorSetting', '#a4509b');
```

- 获取位置信息 - geolocation

```
navigator.geolocation.getCurrentPosition(function(pos) {
  console.log(pos.coords.latitude)
  console.log(pos.coords.longitude)
})
```

- 从服务器获取数据 - Fetch API
- 绘制图形 - <canvas> 的API

- 音视频 - <audio> 和 <video> 的API

## 使用jQuery

### jQuery概述

1. Write Less Do More ( 用更少的代码来完成更多的工作 )
2. 使用CSS选择器来查找元素 ( 更简单更方便 )
3. 使用jQuery方法来操作元素 ( 解决浏览器兼容性问题、应用于所有元素并施加多个方法 )

### 引入jQuery

- 下载jQuery的开发版和压缩版
- 从CDN加载jQuery

```
<script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>
<script>
    window.jQuery ||
        document.write('<script src="js/jquery-3.3.1.min.js"></script>')
</script>
```

## 查找元素

- 选择器
  - \* / element / #id / .class / selector1, selector2
  - ancestor descendant / parent>child / previous+next / previous~siblings
- 筛选器
  - 基本筛选器 :not(selector) / :first / :last / :even / :odd / :eq(index) / :gt(index) / :lt(index) / :animated / :focus
  - 内容筛选器 :contains('...') / :empty / :parent / :has(selector)
  - 可见性筛选器 :hidden / :visible
  - 子节点筛选器 :nth-child(expr) / :first-child / :last-child / :only-child
  - 属性筛选器 :[attribute] / [attribute='value'] / [attribute!='value'] / [attribute^='value'] / [attribute\$='value'] / [attribute]=['value'] / [attribute~='value']
- 表单 :input / :text / :password / :radio / :checkbox / :submit / :image / :reset / :button / :file / :selected / :enabled / :disabled / :checked

## 执行操作

- 内容操作
  - 获取/修改内容 : html() / text() / replaceWith() / remove()

- 获取/设置元素 : before() / after() / prepend() / append() / remove() / clone() / unwrap() / detach() / empty() / add()
- 获取/修改属性 : attr() / removeAttr() /addClass() /removeClass() / css()
- 获取/设置表单值 : val()
- 查找操作
  - 查找方法 : find() / parent() / children() / siblings() / next() / nextAll() / prev() / prevAll()
  - 筛选器 : filter() / not() / has() / is() / contains()
  - 索引编号 : eq()
- 尺寸和位置
  - 尺寸相关 : height() / width() / innerHeight() / innerWidth() / outerWidth() / outerHeight()
  - 位置相关 : offset() / position() / scrollLeft() / scrollTop()
- 特效和动画
  - 基本动画 : show() / hide() / toggle()
  - 消失出现 : fadeIn() / fadeOut() / fadeTo() / fadeToggle()
  - 滑动效果 : slideDown() / slideUp() / slideToggle()
  - 自定义 : delay() / stop() / animate()
- 事件
  - 文档加载 : ready() / load()
  - 用户交互 : on() / off()

## 链式操作

### 检测页面是否可用

```
<script>
$(document).ready(function() {

});

</script>

<script>
$(function() {

});

</script>
```

## jQuery插件

- jQuery Validation
- jQuery Treeview

- jQuery Autocomplete
- jQuery UI

## 避免和其他库的冲突

先引入其他库再引入jQuery的情况。

```
<script src="other.js"></script>
<script src="jquery.js"></script>
<script>
    jQuery.noConflict();
    jQuery(function() {
        jQuery('div').hide();
    });
</script>
```

先引入jQuery再引入其他库的情况。

```
<script src="jquery.js"></script>
<script src="other.js"></script>
<script>
    jQuery(function() {
        jQuery('div').hide();
    });
</script>
```

## 使用Ajax

Ajax是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术。

- 原生的Ajax
- 基于jQuery的Ajax
  - 加载内容
  - 提交表单

## 前端框架

### 渐进式框架 - [Vue.js](#)

前后端分离开发（前端渲染）必选框架。

#### 快速上手

1. 引入Vue的JavaScript文件，我们仍然推荐从CDN服务器加载它。

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

## 2. 数据绑定（声明式渲染）。

```
<div id="app">
    <h1>{{ product }}库存信息</h1>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            product: 'iPhone X'
        }
    });
</script>
```

## 3. 条件与循环。

```
<div id="app">
    <h1>库存信息</h1>
    <hr>
    <ul>
        <li v-for="product in products">
            {{ product.name }} - {{ product.quantity }}
            <span v-if="product.quantity === 0">
                已经售罄
            </span>
        </li>
    </ul>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            products: [
                {"id": 1, "name": "iPhone X", "quantity": 20},
                {"id": 2, "name": "华为 Mate20", "quantity": 0},
                {"id": 3, "name": "小米 Mix3", "quantity": 50}
            ]
        }
    });
</script>
```

## 4. 计算属性。

```
<div id="app">
    <h1>库存信息</h1>
```

```

<hr>
<ul>
    <li v-for="product in products">
        {{ product.name }} - {{ product.quantity }}
        <span v-if="product.quantity === 0">
            已经售罄
        </span>
    </li>
</ul>
<h2>库存总量: {{ totalQuantity }}台</h2>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            products: [
                {"id": 1, "name": "iPhone X", "quantity": 20},
                {"id": 2, "name": "华为 Mate20", "quantity": 0},
                {"id": 3, "name": "小米 Mix3", "quantity": 50}
            ],
            computed: {
                totalQuantity() {
                    return this.products.reduce((sum, product) => {
                        return sum + product.quantity
                    }, 0);
                }
            }
        });
</script>

```

## 5. 处理事件。

```

<div id="app">
    <h1>库存信息</h1>
    <hr>
    <ul>
        <li v-for="product in products">
            {{ product.name }} - {{ product.quantity }}
            <span v-if="product.quantity === 0">
                已经售罄
            </span>
            <button @click="product.quantity += 1">
                增加库存
            </button>
        </li>
    </ul>
    <h2>库存总量: {{ totalQuantity }}台</h2>
</div>

```

```

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            products: [
                {"id": 1, "name": "iPhone X", "quantity": 20},
                {"id": 2, "name": "华为 Mate20", "quantity": 0},
                {"id": 3, "name": "小米 Mix3", "quantity": 50}
            ]
        },
        computed: {
            totalQuantity() {
                return this.products.reduce((sum, product) => {
                    return sum + product.quantity
                }, 0);
            }
        }
    });
</script>

```

## 6. 用户输入。

```

<div id="app">
    <h1>库存信息</h1>
    <hr>
    <ul>
        <li v-for="product in products">
            {{ product.name }} -
            <input type="number" v-model.number="product.quantity" min="0" max="100" />
            <span v-if="product.quantity === 0">
                已经售罄
            </span>
            <button @click="product.quantity += 1">
                增加库存
            </button>
        </li>
    </ul>
    <h2>库存总量: {{ totalQuantity }}台</h2>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            products: [
                {"id": 1, "name": "iPhone X", "quantity": 20},
                {"id": 2, "name": "华为 Mate20", "quantity": 0},
                {"id": 3, "name": "小米 Mix3", "quantity": 50}
            ]
        },
    });
</script>

```

```

        computed: {
            totalQuantity() {
                return this.products.reduce((sum, product) => {
                    return sum + product.quantity
                }, 0);
            }
        });
    </script>

```

## 7. 通过网络加载JSON数据。

```

<div id="app">
    <h2>库存信息</h2>
    <ul>
        <li v-for="product in products">
            {{ product.name }} - {{ product.quantity }}
            <span v-if="product.quantity === 0">
                已经售罄
            </span>
        </li>
    </ul>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            products: []
        },
        created() {
            fetch('https://jackfrued.top/api/products')
                .then(response => response.json())
                .then(json => {
                    this.products = json
                });
        }
    });
</script>

```

## 使用脚手架 - vue-cli

Vue为商业项目开发提供了非常便捷的脚手架工具vue-cli，通过工具可以省去手工配置开发环境、测试环境和运行环境的步骤，让开发者只需要关注要解决的问题。

1. 安装脚手架。
2. 创建项目。
3. 安装依赖包。

#### 4. 运行项目。

##### UI框架 - Element

基于Vue 2.0的桌面端组件库，用于构造用户界面，支持响应式布局。

###### 1. 引入Element的CSS和JavaScript文件。

```
<!-- 引入样式 -->
<link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css">
<!-- 引入组件库 -->
<script src="https://unpkg.com/element-ui/lib/index.js"></script>
```

###### 2. 一个简单的例子。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css">
  </head>
  <body>
    <div id="app">
      <el-button @click="visible = true">点我</el-button>
      <el-dialog :visible.sync="visible" title="Hello world">
        <p>开始使用Element吧</p>
      </el-dialog>
    </div>
  </body>
  <script src="https://unpkg.com/vue/dist/vue.js"></script>
  <script src="https://unpkg.com/element-ui/lib/index.js"></script>
  <script>
    new Vue({
      el: '#app',
      data: {
        visible: false,
      }
    })
  </script>
</html>
```

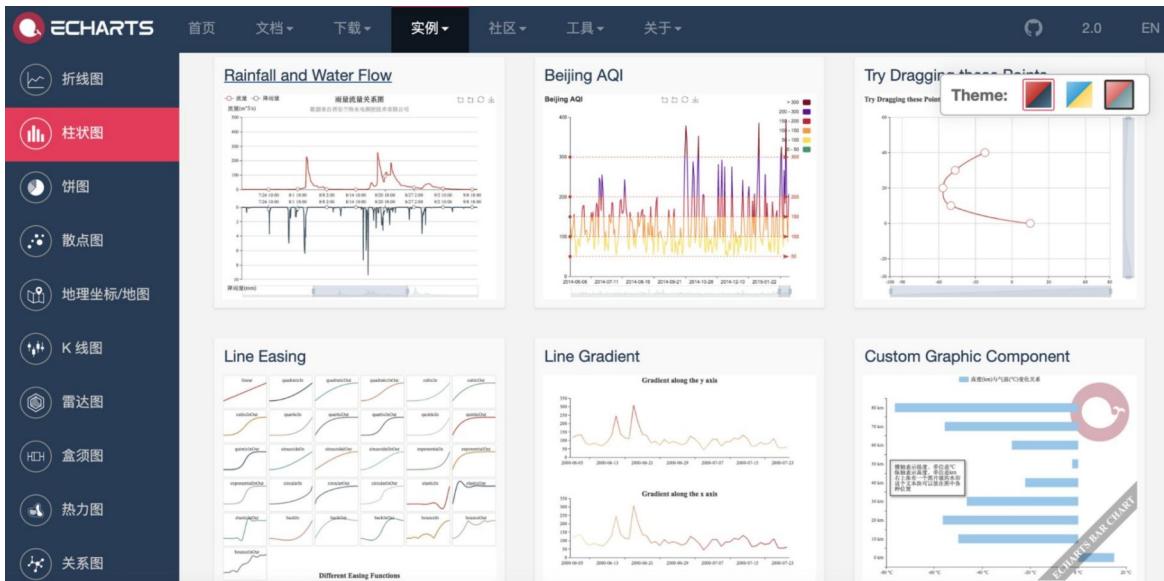
###### 3. 使用组件。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css">
```

```
</head>
<body>
    <div id="app">
        <el-table :data="tableData" stripe style="width: 100%">
            <el-table-column prop="date" label="日期" width="150">
            </el-table-column>
            <el-table-column prop="name" label="姓名" width="150">
            </el-table-column>
            <el-table-column prop="address" label="地址">
            </el-table-column>
        </el-table>
    </div>
</body>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/element-ui/lib/index.js"></script>
<script>
    new Vue({
        el: '#app',
        data: {
            tableData: [
                {
                    date: '2016-05-02',
                    name: '王一霸',
                    address: '上海市普陀区金沙江路 151弄'
                },
                {
                    date: '2016-05-04',
                    name: '刘二狗',
                    address: '上海市普陀区金沙江路 151弄'
                },
                {
                    date: '2016-05-01',
                    name: '杨三萌',
                    address: '上海市普陀区金沙江路 151弄'
                },
                {
                    date: '2016-05-03',
                    name: '陈四吹',
                    address: '上海市普陀区金沙江路 151弄'
                }
            ]
        }
    })
</script>
</html>
```

## 报表框架 - ECharts

百度出品的开源可视化库，常用于生成各种类型的报表。



## 基于弹性盒子的CSS框架 - Bulma

Bulma是一个基于Flexbox的现代化的CSS框架，其初衷就是移动优先（Mobile First），模块化设计，可以轻松用来实现各种简单或者复杂的内容布局，即使不懂CSS的开发者也能够使用它定制出漂亮的页面。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Bulma</title>
    <link href="https://cdn.bootcss.com/bulma/0.7.4/css/bulma.min.css" rel="stylesheet"/>
    <style type="text/css">
        div { margin-top: 10px; }
        .column { color: #fff; background-color: #063; margin: 10px 10px; text-align: center; }
    </style>
</head>
<body>
    <div class="columns">
        <div class="column">1</div>
        <div class="column">2</div>
        <div class="column">3</div>
        <div class="column">4</div>
    </div>
    <div>
        <a class="button is-primary">Primary</a>
        <a class="button is-link">Link</a>
        <a class="button is-info">Info</a>
        <a class="button is-success">Success</a>
        <a class="button is-warning">Warning</a>
        <a class="button is-danger">Danger</a>
    </div>
    <div>
        <progress class="progress is-danger is-medium" max="100">60%</progress>
    </div>
</body>
```

```
<div>
  <table class="table is-hoverable">
    <tr>
      <th>One</th>
      <th>Two</th>
    </tr>
    <tr>
      <td>Three</td>
      <td>Four</td>
    </tr>
    <tr>
      <td>Five</td>
      <td>Six</td>
    </tr>
    <tr>
      <td>Seven</td>
      <td>Eight</td>
    </tr>
    <tr>
      <td>Nine</td>
      <td>Ten</td>
    </tr>
    <tr>
      <td>Eleven</td>
      <td>Twelve</td>
    </tr>
  </table>
</div>
</body>
</html>
```

## 响应式布局框架 - Bootstrap

用于快速开发Web应用程序的前端框架，支持响应式布局。

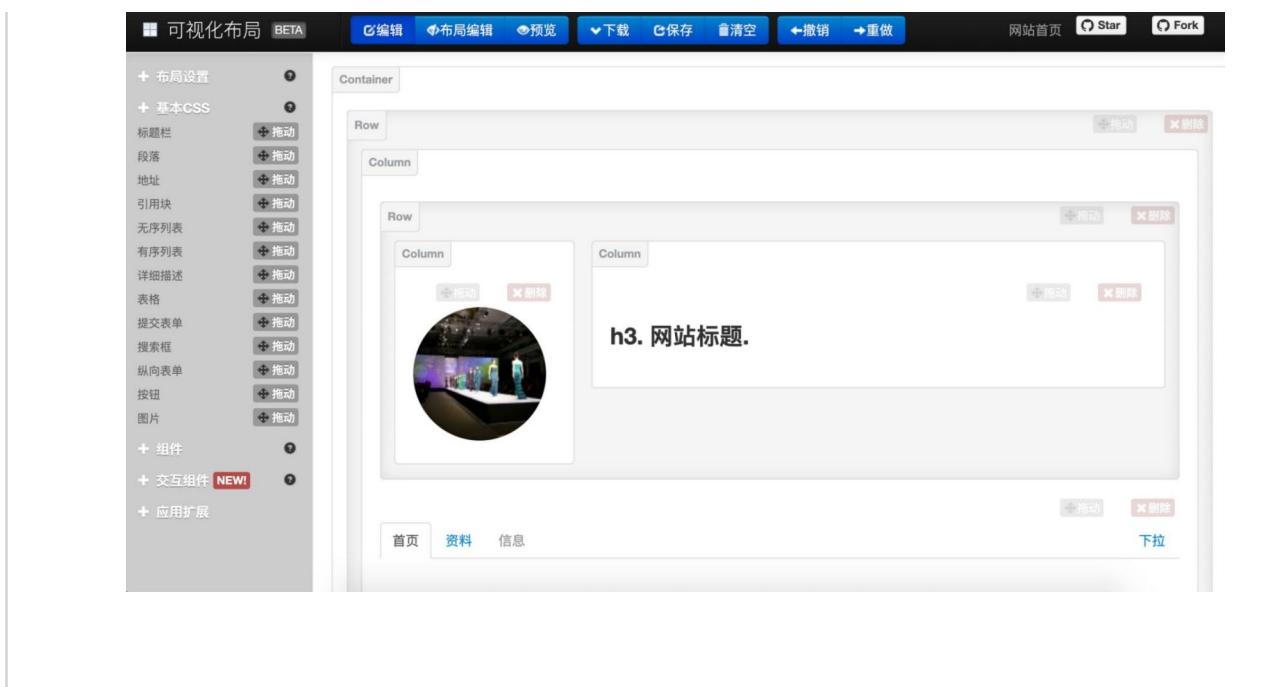
### 1. 特点

- 支持主流的浏览器和移动设备
- 容易上手
- 响应式设计

### 2. 内容

- 网格系统
- 封装的CSS
- 现成的组件
- JavaScript插件

### 3. 可视化



4

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Day31-35 / 玩转Linux操作系统.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

1246 lines (988 sloc) 51.2 KB

# 玩转Linux操作系统

## 操作系统发展史

只有硬件没有软件的计算机系统被称之为“裸机”，我们很难用“裸机”来完成计算机日常的工作（如存储和运算），所以必须用特定的软件来控制硬件的工作。最靠近计算机硬件的软件是系统软件，其中最为重要的就是“操作系统”。“操作系统”是控制和管理整个计算机系统的硬件和软件资源，合理的分配资源和调配任务，为系统用户和其他软件提供接口和环境的程序的集合。

## 没有操作系统（手工操作）

在计算机诞生之初没有操作系统的年代，人们先把程序纸带（或卡片）装上计算机，然后启动输入机把程序和送入计算机，接着通过控制台开关启动程序运行。当程序执行完毕，打印机输出计算的结果，用户卸下并取走纸带（或卡片）。第二个用户上机，重复同样的步骤。在整个过程中用户独占机器，CPU等待手工操作，资源利用率极低。下图是IBM生产的书写Fortran程序的80栏打孔卡，当然这个已经是比较先进的打孔卡了。

IBM

FORTAN Coding Form

Program  
Punching  
Sequence

Date \_\_\_\_\_

PUNCH INSTRUCTIONS: GRAPHIC PUNCH

PAGE OF CARDS ELECTRIC NUMBER

FORTRAN STATEMENT

DISPATCHER SEQUENCE

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

A punched card form, like excess data. It is suitable for punching statements from this form.

## 批处理系统

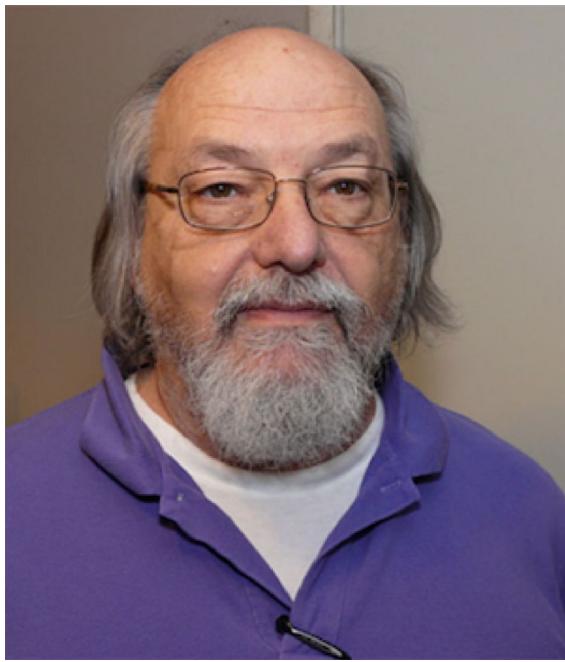
首先启动计算机上的一个监督程序，在监督程序的控制下，计算机能够自动的、成批的处理一个或多个用户的作业。完成一批作业后，监督程度又从输入机读取作业存入磁带机。按照上面的步骤重复处理任务。监督程序不停的处理各个作业，实现了作业的自动转接，减少了作业的建立时间和手工操作时间，提高了计算机资源的利用率。批处理系统又可以分为单道批处理系统、多道批处理系统、联机批处理系统、脱机批处理系统。

## 分时系统和实时系统

分时系统是把处理器的运行时间分成很短的时间片，按时间片轮流把处理机分配给各联机作业使用。若某个作业在分配给它的时间片内不能完成其计算，则该作业暂时中断，把处理机让给另一作业使用，等待下一轮调度时再继续其运行。由于计算机速度很快，作业运行轮转得很快，给每个用户的感觉是他独占了一台计算机。而每个用户可以通过自己的终端向系统发出各种操作控制命令，在充分的人机交互情况下，完成作业的运行。为了解决分时系统不能及时响应用户指令的情况，又出现了能够在在严格的时间范围内完成事件处理，及时响应随机外部事件的实时系统。

## 通用操作系统

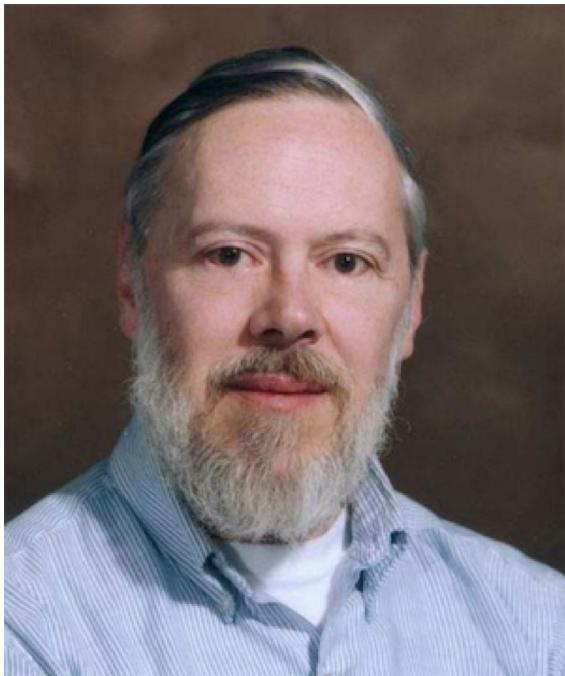
1. 1960s : IBM的System/360系列的机器有了统一的操作系统OS/360。
2. 1965年 : AT&T的贝尔实验室加入GE和MIT的合作计划开始开发MULTICS。
3. 1969年 : Ken Tompson为了玩“Space Travel”游戏用汇编语言在PDP-7上开发了Unics。



4. 1970年~1971年：Ken Tompson和Dennis Ritchie用B语言在PDP-11上重写了Unics，并在Brian Kernighan的建议下将其更名为Unix。



5. 1972年~1973年：Dennis Ritchie发明了C语言来取代可移植性较差的B语言，并开启了用C语言重写Unix的工作。



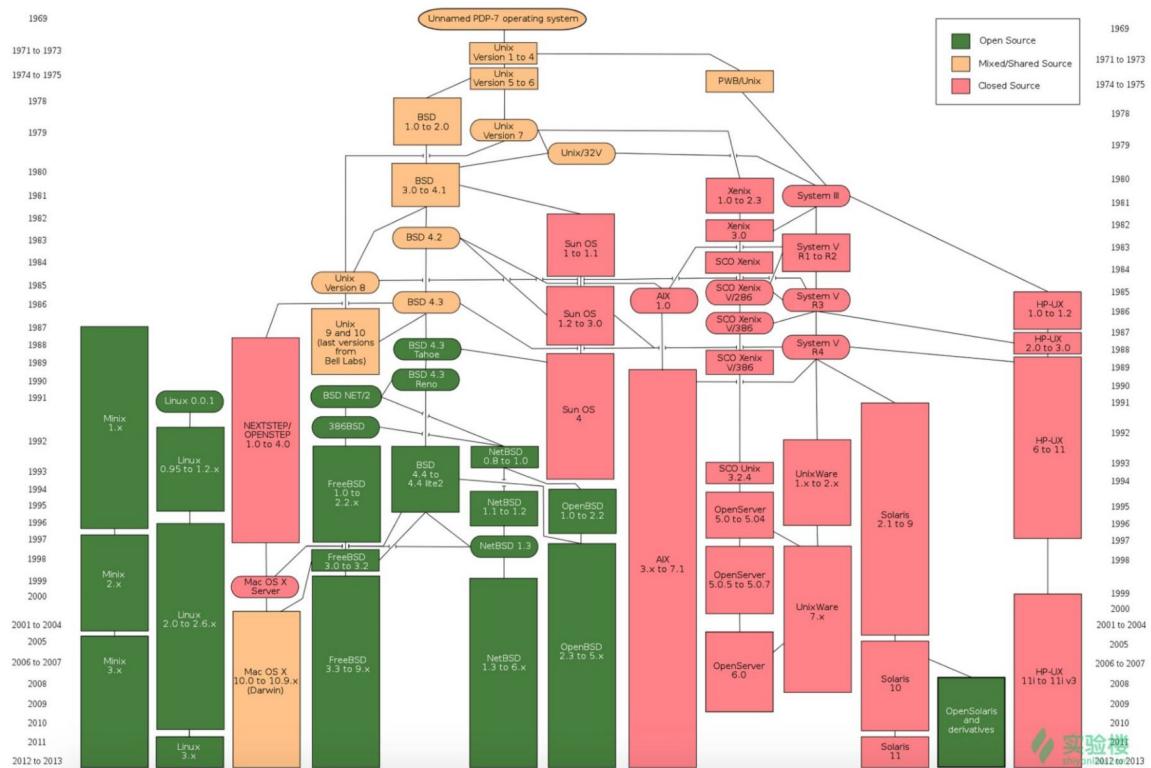
6. 1974年：Unix推出了里程碑意义的第5版，几乎完全用C语言来实现。
7. 1979年：从Unix第7版开始，AT&T发布新的使用条款，将Unix私有化。
8. 1987年：Andrew S. Tanenbaum教授为了能在课堂上教授学生操作系统运作的细节，决定在不使用任何AT&T的源代码前提下，自行开发与Unix兼容的操作系统，以避免版权上的争议并将其命名为Minix。



9. 1991年：Linus Torvalds就读于芬兰赫尔辛基大学期间，尝试在Minix上做一些开发工作，但因为Minix只是作为教学用途的操作系统，功能并不强大，为了方便在学校的主机的新闻组和邮件系统中读写和下载文件，Linus编写了磁盘驱动程序和文件系统，这些成为了Linux系统内核的雏形。



下图是Unix操作系统家族的图谱。



## Linux概述

Linux是一个通用操作系统。一个操作系统要负责任务调度、内存分配、处理外围设备I/O等操作。操作系统通常由内核（运行其他程序，管理像磁盘、打印机等硬件设备的核心程序）和系统程序（设备驱动、底层库、shell、服务程序等）两部分组成。

Linux内核是芬兰人Linus Torvalds开发的，于1991年9月发布。而Linux操作系统作为Internet时代的产物，它是由全世界许多开发者共同合作开发的，是一个自由的操作系统（注意自由和免费并不是同一个概念，想了解二者的差别可以[点击这里](#)）。

## Linux系统优点

1. 通用操作系统，不跟特定的硬件绑定。
2. 用C语言编写，有可移植性，有内核编程接口。
3. 支持多用户和多任务，支持安全的分层文件系统。
4. 大量的实用程序，完善的网络功能以及强大的支持文档。
5. 可靠的安全性和良好的稳定性，对开发者更友好。

## Linux系统发行版本

1. [Redhat](#)
2. [Ubuntu](#)
3. [CentOS](#)
4. [Fedora](#)
5. [Debian](#)
6. [openSUSE](#)

## 基础命令

Linux系统的命令通常都是如下所示的格式：

命令名称 [命名参数] [命令对象]

1. 获取登录信息 - w / who / last。

```
[root@izwz97tbgo91kabnat2lo8z ~]# w
23:31:16 up 12:16,  2 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM          LOGIN@    IDLE   JCPU   PCPU WHAT
root     pts/0    182.139.66.250  23:03    4.00s  0.02s  0.00s w
jackfrue pts/1    182.139.66.250  23:26    3:56   0.00s  0.00s -bash
[root@izwz97tbgo91kabnat2lo8z ~]# who
root     pts/0        2018-04-12 23:03 (182.139.66.250)
jackfrued pts/1       2018-04-12 23:26 (182.139.66.250)
[root@izwz97tbgo91kabnat2lo8z ~]# who am i
root     pts/0        2018-04-12 23:03 (182.139.66.250)
```

2. 查看自己使用的Shell - ps。

Shell也被称为“壳”，它是用户与内核交流的翻译官，简单的说就是人与计算机交互的接口。目前很多Linux系统默认的Shell都是bash ( Bourne Again SHell )，因为它可以使用Tab键进行命令补全、可以保存历史命令、可以方便的配置环境变量以及执行批处理操作等。

```
[root@izwz97tbgo91kabnat2lo8z ~]# ps
  PID TTY      TIME CMD
 3531 pts/0    00:00:00 bash
 3553 pts/0    00:00:00 ps
```

### 3. 查看命令的说明 - whatis。

```
[root@izwz97tbgo91kabnat2lo8z ~]# whatis ps
ps (1)      - report a snapshot of the current processes.
[root@izwz97tbgo91kabnat2lo8z ~]# whatis python
python (1)   - an interpreted, interactive, object-oriented programming language
```

### 4. 查看命令的位置 - which / whereis。

```
[root@izwz97tbgo91kabnat2lo8z ~]# whereis ps
ps: /usr/bin/ps /usr/share/man/man1/ps.1.gz
[root@izwz97tbgo91kabnat2lo8z ~]# whereis python
python: /usr/bin/python /usr/bin/python2.7 /usr/lib/python2.7 /usr/lib64/python2.7
[root@izwz97tbgo91kabnat2lo8z ~]# which ps
/usr/bin/ps
[root@izwz97tbgo91kabnat2lo8z ~]# which python
/usr/bin/python
```

### 5. 查看帮助文档 - man / info / apropos。

```
[root@izwz97tbgo91kabnat2lo8z ~]# ps --help
Usage:
  ps [options]
  Try 'ps --help <simple|list|output|threads|misc|all>'
  or 'ps --help <s|l|o|t|m|a>'
  for additional help text.
  For more details see ps(1).
[root@izwz97tbgo91kabnat2lo8z ~]# man ps
PS(1)                               User Commands
NAME
  ps - report a snapshot of the current processes.
SYNOPSIS
  ps [options]
DESCRIPTION
  ...
```

```
[root@izwz97tbgo91kabnat2lo8z ~]# info ps
```

...



## 6. 切换用户 - su。

```
[root@izwz97tbgo91kabnat2lo8z ~]# su hellokitty  
[hellokitty@izwz97tbgo91kabnat2lo8z root]$
```

## 7. 以管理员身份执行命令 - sudo。

```
[jackfrued@izwz97tbgo91kabnat2lo8z ~]$ ls /root  
ls: cannot open directory /root: Permission denied  
[jackfrued@izwz97tbgo91kabnat2lo8z ~]$ sudo ls /root  
[sudo] password for jackfrued:  
calendar.py code error.txt hehe hello.c index.html myconf result.txt
```

**说明**：如果希望用户能够以管理员身份执行命令，用户必须被添加到sudoers名单中，该文件在 /etc 目录下。

## 8. 登入登出相关 - logout / exit / adduser / userdel / passwd / ssh。

```
[root@izwz97tbgo91kabnat2lo8z ~]# adduser hellokitty  
[root@izwz97tbgo91kabnat2lo8z ~]# passwd hellokitty  
Changing password for user jackfrued.  
New password:  
Retype new password:  
passwd: all authentication tokens updated successfully.  
[root@izwz97tbgo91kabnat2lo8z ~]# ssh hellokitty@1.2.3.4  
hellokitty@1.2.3.4's password:  
Last login: Thu Apr 12 23:05:32 2018 from 10.12.14.16  
[hellokitty@izwz97tbgo91kabnat2lo8z ~]$ logout  
Connection to 1.2.3.4 closed.  
[root@izwz97tbgo91kabnat2lo8z ~]#
```

## 9. 查看系统和主机名 - uname / hostname。

```
[root@izwz97tbgo91kabnat2lo8z ~]# uname  
Linux  
[root@izwz97tbgo91kabnat2lo8z ~]# hostname  
izwz97tbgo91kabnat2lo8z  
[root@izwz97tbgo91kabnat2lo8z ~]# cat /etc/centos-release  
CentOS Linux release 7.4.1708 (Core)
```

## 10. 重启和关机 - reboot / init 6 / shutdown / init 0。

## 11. 查看历史命令 - history。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# history
...
452  ls
453  cd Python-3.6.5/
454  clear
455  history
[root@iZwz97tbgo91kabnat2lo8Z ~]# !454
```

说明：查看到历史命令之后，可以用 !历史命令编号 来重新执行该命令；通过 history -c 可以清除历史命令。

## 实用程序

### 文件和文件夹操作

#### 1. 创建/删除目录 - mkdir / rmdir。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# mkdir abc
[root@iZwz97tbgo91kabnat2lo8Z ~]# mkdir -p xyz/abc
[root@iZwz97tbgo91kabnat2lo8Z ~]# rmdir abc
```

#### 2. 创建/删除文件 - touch / rm。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# touch readme.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]# touch error.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]# rm error.txt
rm: remove regular empty file ‘error.txt’? y
[root@iZwz97tbgo91kabnat2lo8Z ~]# rm -rf xyz
```

- touch命令用于创建空白文件或修改文件时间。在Linux系统中一个文件有三种时间：
  - 更改内容的时间 - mtime。
  - 更改权限的时间 - ctime。
  - 最后访问时间 - atime。
- rm的几个重要参数：
  - -i：交互式删除，每个删除项都会进行询问。
  - -r：删除目录并递归的删除目录中的文件和目录。
  - -f：强制删除，忽略不存在的文件，没有任何提示。

#### 3. 切换和查看当前工作目录 - cd / pwd。

说明： cd 命令后面可以跟相对路径（以当前路径作为参照）或绝对路径（以 / 开头）来切换到指定的目录，也可以用 cd .. 来返回上一级目录。

#### 4. 查看目录内容 - ls。

- -l : 以长格式查看文件和目录。
- -a : 显示以点开头的文件和目录 ( 隐藏文件 ) 。
- -R : 遇到目录要进行递归展开 ( 继续列出目录下面的文件和目录 ) 。
- -d : 只列出目录 , 不列出其他内容。
- -S/-t : 按大小/时间排序。

## 5. 查看文件内容 - cat / head / tail / more / less.

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# wget http://www.sohu.com/ -O sohu.html
--2018-06-20 18:42:34--  http://www.sohu.com/
Resolving www.sohu.com (www.sohu.com)... 14.18.240.6
Connecting to www.sohu.com (www.sohu.com)|14.18.240.6|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 212527 (208K) [text/html]
Saving to: 'sohu.html'
100%[=====] 212,527      --.-K/s   ir
2018-06-20 18:42:34 (7.48 MB/s) - 'sohu.html' saved [212527/212527]
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat sohu.html
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# head -10 sohu.html
<!DOCTYPE html>
<html>
<head>
<title>搜狐</title>
<meta name="Keywords" content="搜狐,门户网站,新媒体,网络媒体,新闻,财经,体育,娱乐,影
<meta name="Description" content="搜狐网为用户提供24小时不间断的最新资讯,及搜索、
<meta name="shenma-site-verification" content="1237e4d02a3d8d73e96cbd97b699e9c3_15
<meta charset="utf-8"/>
<meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1"/>
[root@iZwz97tbgo91kabnat2lo8Z ~]# tail -2 sohu.html
</body>
</html>
[root@iZwz97tbgo91kabnat2lo8Z ~]# less sohu.html
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat -n sohu.html | more
...
```

## 6. 拷贝/移动文件 - cp / mv.

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# mkdir backup
[root@iZwz97tbgo91kabnat2lo8Z ~]# cp sohu.html backup/
[root@iZwz97tbgo91kabnat2lo8Z ~]# cd backup
[root@iZwz97tbgo91kabnat2lo8Z backup]# ls
sohu.html
[root@iZwz97tbgo91kabnat2lo8Z backup]# mv sohu.html sohu_index.html
[root@iZwz97tbgo91kabnat2lo8Z backup]# ls
sohu_index.html
```

## 7. 查找文件和查找内容 - find / grep。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# find / -name "*.html"
/root/sohu.html
/root/backup/sohu_index.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# find . -atime 7 -type f -print
[root@iZwz97tbgo91kabnat2lo8Z ~]# find . -type f -size +2k
[root@iZwz97tbgo91kabnat2lo8Z ~]# find . -type f -name "*.swp" -delete
[root@iZwz97tbgo91kabnat2lo8Z ~]# grep "<script>" sohu.html -n
20:<script>
[root@iZwz97tbgo91kabnat2lo8Z ~]# grep -E \<\?script.*\> sohu.html -n
20:<script>
22:</script>
24:<script src="//statics.itc.cn/web/v3/static/js/es5-shim-08e41cf3e.min.js"></sc
25:<script src="//statics.itc.cn/web/v3/static/js/es5-sham-1d5fa1124b.min.js"></sc
26:<script src="//statics.itc.cn/web/v3/static/js/html5shiv-21fc8c2ba6.js"></scrip
29:<script type="text/javascript">
52:</script>
...
...
```

说明： grep 在搜索字符串时可以使用正则表达式，如果需要使用正则表达式可以用 grep -E 或者直接使用 egrep 。

## 8. 链接 - ln。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sohu.html
-rw-r--r-- 1 root root 212131 Jun 20 19:15 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ln /root/sohu.html /root/backup/sohu_backup
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sohu.html
-rw-r--r-- 2 root root 212131 Jun 20 19:15 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ln /root/sohu.html /root/backup/sohu_backup2
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sohu.html
-rw-r--r-- 3 root root 212131 Jun 20 19:15 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ln -s /etc/centos-release sysinfo
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sysinfo
lrwxrwxrwx 1 root root 19 Jun 20 19:21 sysinfo -> /etc/centos-release
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat sysinfo
CentOS Linux release 7.4.1708 (Core)
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat /etc/centos-release
CentOS Linux release 7.4.1708 (Core)
```

说明：链接可以分为硬链接和软链接（符号链接）。硬链接可以认为是一个指向文件数据的指针，就像Python中对象的引用计数，每添加一个硬链接，文件的对应链接数就增加1，只有当文件的链接数为0时，文件所对应的存储空间才有可能被其他文件覆盖。我们平常删除文件时其实并没有删除硬盘上的数据，我们删除的只是一个指针，或者说是数据的一条使用记录，所以类似于“文件粉碎机”之类的软件在“粉碎”文件时除了删除文件指针，还会在文件对应的存储区域填入数据来保证文件无法再恢复。软链接类似于Windows系统下的快捷方式，当软链接链接的文件被删除时，软链接也就失效了。

## 9. 压缩/解压缩和归档/解归档 - gzip / gunzip / xz / tar。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# wget http://download.redis.io/releases/redis-4.0.10.tar.gz
--2018-06-20 19:29:59--  http://download.redis.io/releases/redis-4.0.10.tar.gz
Resolving download.redis.io (download.redis.io)... 109.74.203.151
Connecting to download.redis.io (download.redis.io)|109.74.203.151|:80... connected
HTTP request sent, awaiting response... 200 OK
Length: 1738465 (1.7M) [application/x-gzip]
Saving to: ‘redis-4.0.10.tar.gz’

100%[=====] 1,738,465 70.1KB/s
2018-06-20 19:31:14 (22.9 KB/s) - ‘redis-4.0.10.tar.gz’ saved [1738465/1738465]
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls redis*
redis-4.0.10.tar.gz
[root@iZwz97tbgo91kabnat2lo8Z ~]# gunzip redis-4.0.10.tar.gz
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls redis*
redis-4.0.10.tar
[root@iZwz97tbgo91kabnat2lo8Z ~]# tar -xvf redis-4.0.10.tar
redis-4.0.10/
redis-4.0.10/.gitignore
redis-4.0.10/00-RELEASENOTES
redis-4.0.10/BUGS
redis-4.0.10/CONTRIBUTING
redis-4.0.10/COPYING
redis-4.0.10/INSTALL
redis-4.0.10/MANIFESTO
redis-4.0.10/Makefile
redis-4.0.10/README.md
redis-4.0.10/deps/
redis-4.0.10/deps/Makefile
redis-4.0.10/deps/README.md
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls redis*
redis-4.0.10.tar
redis-4.0.10:
00-RELEASENOTES  COPYING  Makefile  redis.conf      runtest-sentinel  tests
BUGS            deps     MANIFESTO  runtest        sentinel.conf    utils
CONTRIBUTING   INSTALL  README.md  runtest-cluster src
```

## 10. 其他工具 - sort / uniq / diff / tr / cut / paste / file / wc。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat foo.txt
grape
apple
pitaya
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat bar.txt
100
200
300
400
[root@iZwz97tbgo91kabnat2lo8Z ~]# paste foo.txt bar.txt
grape    100
apple    200
pitaya   300
        400
[root@iZwz97tbgo91kabnat2lo8Z ~]# paste foo.txt bar.txt > hello.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]# cut -b 4-8 hello.txt
pe      10
le      20
aya    3
0
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat hello.txt | tr '\t' ','
grape,100
apple,200
pitaya,300
,400
[root@iZwz97tbgo91kabnat2lo8Z ~]# wget https://www.baidu.com/img/bd_logo1.png
--2018-06-20 18:46:53--  https://www.baidu.com/img/bd_logo1.png
Resolving www.baidu.com (www.baidu.com)... 220.181.111.188, 220.181.112.244
Connecting to www.baidu.com (www.baidu.com)|220.181.111.188|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7877 (7.7K) [image/png]
Saving to: 'bd_logo1.png'

100%[=====] 7,877          --.-K/s   in 0s
2018-06-20 18:46:53 (118 MB/s) - ‘bd_logo1.png’ saved [7877/7877][root@iZwz97tbgo91kab
bd_logo1.png: PNG image data, 540 x 258, 8-bit colormap, non-interlaced
[root@iZwz97tbgo91kabnat2lo8Z ~]# wc sohu.html
2979 6355 212527 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# wc -l sohu.html
2979 sohu.html
```

## 管道和重定向

### 1. 管道的使用 - |。

例子：查找当前目录下文件个数。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# find ./ | wc -l
6152
```

例子：列出当前路径下的文件和文件夹，给每一项加一个编号。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls | cat -n
1  dump.rdb
2  mongodb-3.6.5
3  Python-3.6.5
4  redis-3.2.11
5  redis.conf
```

例子：查找record.log中包含AAA，但不包含BBB的记录的总数

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat record.log | grep AAA | grep -v BBB | wc -l
```

2. 输出重定向和错误重定向 - > / >> / 2>。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat readme.txt
banana
apple
grape
apple
grape
watermelon
pear
pitaya
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat readme.txt | sort | uniq > result.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat result.txt
apple
banana
grape
pear
pitaya
watermelon
```

3. 输入重定向 - <。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# echo 'hello, world!' > hello.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]# wall < hello.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]#
Broadcast message from root@iZwz97tbgo91kabnat2lo8Z (Wed Jun 20 19:43:05 2018):
hello, world!
[root@iZwz97tbgo91kabnat2lo8Z ~]# echo 'I will show you some code.' >> hello.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]# wall < hello.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]#
Broadcast message from root@iZwz97tbgo91kabnat2lo8Z (Wed Jun 20 19:43:55 2018):
hello, world!
I will show you some code.
```

## 别名

### 1. alias

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# alias ll='ls -l'
[root@iZwz97tbgo91kabnat2lo8Z ~]# alias frm='rm -rf'
[root@iZwz97tbgo91kabnat2lo8Z ~]# ll
...
drwxr-xr-x  2 root      root   4096 Jun 20 12:52 abc
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# frm abc
```

### 2. unalias

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# unalias frm
[root@iZwz97tbgo91kabnat2lo8Z ~]# frm sohu.html
-bash: frm: command not found
```

## 其他程序

### 1. 时间和日期 - date / cal。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# date
Wed Jun 20 12:53:19 CST 2018
[root@iZwz97tbgo91kabnat2lo8Z ~]# cal
June 2018
Su Mo Tu We Th Fr Sa
      1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
[root@iZwz97tbgo91kabnat2lo8Z ~]# cal 5 2017
May 2017
Su Mo Tu We Th Fr Sa
      1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

### 2. 录制操作脚本 - script。

### 3. 给用户发送消息 - mesg / write / wall / mail。

## 文件系统

### 文件和路径

1. 命名规则：文件名的最大长度与文件系统类型有关，一般情况下，文件名不应该超过255个字符，虽然绝大多数的字符都可以用于文件名，但是最好使用英文大小写字母、数字、下划线、点这样的符号。文件名中虽然可以使用空格，但应该尽可能避免使用空格，否则在输入文件名时需要用将文件名放在双引号中或者通过\对空格进行转义。
2. 扩展名：在Linux系统下文件的扩展名是可选的，但是使用扩展名有助于对文件内容的理解。有些应用程序要通过扩展名来识别文件，但是更多的应用程序并不依赖文件的扩展名，就像 file 命令在识别文件时并不是依据扩展名来判定文件的类型。
3. 隐藏文件：以点开头的文件在Linux系统中是隐藏文件（不可见文件）。

## 目录结构

1. /bin - 基本命令的二进制文件。
2. /boot - 引导加载程序的静态文件。
3. /dev - 设备文件。
4. /etc - 配置文件。
5. /home - 普通用户主目录的父目录。
6. /lib - 共享库文件。
7. /lib64 - 共享64位库文件。
8. /lost+found - 存放未链接文件。
9. /media - 自动识别设备的挂载目录。
10. /mnt - 临时挂载文件系统的挂载点。
11. /opt - 可选插件软件包安装位置。
12. /proc - 内核和进程信息。
13. /root - 超级管理员用户主目录。
14. /run - 存放系统运行时需要的东西。
15. /sbin - 超级用户的二进制文件。
16. /sys - 设备的伪文件系统。
17. /tmp - 临时文件夹。
18. /usr - 用户应用目录。
19. /var - 变量数据目录。

## 访问权限

1. chmod - 改变文件模式比特。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l
...
-rw-r--r-- 1 root      root 211878 Jun 19 16:06 sohu.html
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# chmod g+w,o+w sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l
```

```

...
-rw-rw-rw- 1 root      root 211878 Jun 19 16:06 sohu.html
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# chmod 644 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l
...
-rw-r--r-- 1 root      root 211878 Jun 19 16:06 sohu.html
...

```

说明：通过上面的例子可以看出，用 `chmod` 改变文件模式比特有两种方式：一种是字符设定法，另一种是数字设定法。除了 `chmod` 之外，可以通过 `umask` 来设定哪些权限将在新文件的默认权限中被删除。

长格式查看目录或文件时显示结果及其对应权限的数值如下表所示。

文件类型	所有者权限	同组用户权限	其他用户权限
d (目录)	r w x 读写执行 7	r - x 读执行 5	r - x 读执行 5
- (文件)	r w - 读写 6	r - - 读 4	r - - 读 4
l (链接)	r w x 读写执行 7	r w x 读写执行 7	r - x 读执行 5

## 2. `chown` - 改变文件所有者。

```

[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l
...
-rw-r--r-- 1 root root 54 Jun 20 10:06 readme.txt
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# chown hellokitty readme.txt
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l
...
-rw-r--r-- 1 hellokitty root 54 Jun 20 10:06 readme.txt
...

```

## 磁盘管理

### 1. 列出文件系统的磁盘使用状况 - `df`。

```

[root@iZwz97tbgo91kabnat2lo8Z ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        40G   5.0G   33G  14% /
devtmpfs        486M     0  486M   0% /dev
tmpfs          497M     0  497M   0% /dev/shm
tmpfs          497M   356K  496M   1% /run
tmpfs          497M     0  497M   0% /sys/fs/cgroup
tmpfs         100M     0  100M   0% /run/user/0

```

## 2. 磁盘分区表操作 - **fdisk**。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# fdisk -l
Disk /dev/vda: 42.9 GB, 42949672960 bytes, 83886080 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000a42f4

Device Boot Start End Blocks Id System
/dev/vda1 * 2048 83884031 41940992 83 Linux

Disk /dev/vdb: 21.5 GB, 21474836480 bytes, 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

## 3. 格式化文件系统 - **mkfs**。

## 4. 文件系统检查 - **fsck**。

## 5. 挂载/卸载 - **mount / umount**。

## 编辑器 - **vim**

- 启动vim。可以通过 **vi** 或 **vim** 命令来启动vim，启动时可以指定文件名来打开一个文件，如果没有指定文件名，也可以在保存的时候指定文件名。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# vim guess.py
```

- 命令模式、编辑模式和末行模式：启动vim进入的是命令模式（也称为Normal模式），在命令模式下输入英文字母 **i** 会进入编辑模式（Insert模式），屏幕下方出现 **-- INSERT --** 提示；在编辑模式下按下 **Esc** 会回到命令模式，此时如果输入英文：**:**会进入末行模式，在末行模式下输入 **q!** 可以在不保存当前工作的情况下强行退出vim；在命令模式下输入 **v** 会进入可视模式（Visual模式），可以用光标选择一个区域再完成对应的操作。
- 保存和退出vim：在命令模式下输入：**:**进入末行模式，输入 **wq** 可以实现保存退出；如果想放弃编辑的内容输入 **q!** 强行退出，这一点刚才已经提到过了；在命令模式下也可以直接输入 **zz** 实现保存退出。如果只想保存文件不退出，那么可以在末行模式下输入 **w**；可以在 **w** 后面输入空格再指定要保存的文件名。
- 光标操作。
  - 在命令模式下可以通过 **h**、**j**、**k**、**l** 来控制光标向左、下、上、右的方向移动，可以在字母前输入数字来表示移动的距离，例如：**10h** 表示向左移动10个字符。

- 在命令模式下可以通过 `Ctrl+y` 和 `Ctrl+e` 来实现向上、向下滚动一行文本的操作，可以通过 `Ctrl+f` 和 `Ctrl+b` 来实现向前和向后翻页的操作。
- 在命令模式下可以通过输入英文字母 `G` 将光标移到文件的末尾，可以通过 `gg` 将光标移到文件的开始，也可以通过在 `G` 前输入数字来将光标移动到指定的行。

## 5. 文本操作。

- 删除：在命令模式下可以用 `dd` 来删除整行；可以在 `dd` 前加数字来指定删除的行数；可以用 `d$` 来实现删除从光标处删到行尾的操作，也可以通过 `d0` 来实现从光标处删到行首的操作；如果想删除一个单词，可以使用 `dw`；如果要删除全文，可以在输入 `:%d`（其中 `:` 用来从命令模式进入末行模式）。
- 复制和粘贴：在命令模式下可以用 `yy` 来复制整行；可以在 `yy` 前加数字来指定复制的行数；可以通过 `p` 将复制的内容粘贴到光标所在的地方。
- 撤销和恢复：在命令模式下输入 `u` 可以撤销之前的操作；通过 `Ctrl+r` 可以恢复被撤销的操作。
- 对内容进行排序：在命令模式下输入 `%!sort`。

## 6. 查找和替换。

- 查找操作需要输入 `/` 进入末行模式并提供正则表达式来匹配与之对应的内容，例如：`/doc.*\.`，输入 `n` 来向前搜索，也可以输入 `N` 来向后搜索。
- 替换操作需要输入：进入末行模式并指定搜索的范围、正则表达式以及替换后的内容和匹配选项，例如：`:1,$s/doc.*/hello/gice`，其中：
  - `g` - global：全局匹配。
  - `i` - ignore case：忽略大小写匹配。
  - `c` - confirm：替换时需要确认。
  - `e` - error：忽略错误。

## 7. 参数设定：在输入 `:` 进入末行模式后可以对 vim 进行设定。

- 设置 Tab 键的空格数：`set ts=4`
- 设置显示/不显示行号：`set nu` / `set nonu`
- 设置启用/关闭高亮语法：`syntax on` / `syntax off`
- 设置显示标尺（光标所在的行和列）：`set ruler`
- 设置启用/关闭搜索结果高亮：`set hls` / `set nohls`

说明：如果希望上面的这些设定在每次启动 vim 时都能生效，需要将这些设定写到用户主目录下的 `.vimrc` 文件中。

## 8. 高级技巧

- 比较多个文件。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# vim -d foo.txt bar.txt
```

The screenshot shows a Vim session with two windows side-by-side. The left window is titled 'foo.txt' and displays the following text:  
1 grape  
2 apple  
3 pitaya

The right window is titled 'bar.txt' and displays the following text:  
1 100  
2 200  
3 300  
4 400

Both windows have their status bars at the bottom showing '1,1' and 'All'.

- 打开多个文件。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# vim foo.txt bar.txt hello.txt
```

启动vim后只有一个窗口显示的是foo.txt，可以在末行模式中输入 `ls` 查看到打开的三个文件，也可以在末行模式中输入 `b <num>` 来显示另一个文件，例如可以用 `:b 2` 将bar.txt显示出来，可以用 `:b 3` 将hello.txt显示出来。

- 拆分和切换窗口。

可以在末行模式中输入 `sp` 或 `vs` 来实现对窗口的水平或垂直拆分，这样我们就可以同时打开多个编辑窗口，通过按两次 `Ctrl+w` 就可以实现编辑窗口的切换，在一个窗口中执行退出操作只会关闭对应的窗口，其他的窗口继续保留。

```

1 #!/usr/bin/python3
2 # coding: utf-8
3 from random import randint
4
5
6 def main():
7     answer = randint(1, 100)
8     while True:
9         number = int(input('请输入：'))
10        if number < answer:
11            print('大一点')
12        elif number > answer:
guess.py          1,1      Top
1 class Hello {
2
3     public static void main(String[] args) {
4         System.out.println("Hello, world!");
5         System.out.println("Goodbye, world!");
6     }
7 }

Hello.java        1,1      All mycal.py          1,1      Top

```

- 映射快捷键：在vim下可以将一些常用操作映射为快捷键来提升工作效率。

- 例子1：在命令模式下输入 F4 执行从第一行开始删除10000行代码的操作。

```
:map <F4> gg10000dd .
```

例子2：在编辑模式下输入 \_\_main\_\_ 直接补全为 if \_\_name\_\_ == '\_\_main\_\_':。

```
:inoremap __main__ if __name__ == '__main__':
```

说明：上面例子2的 inoremap 中的 i 表示映射的键在编辑模式使用，nore 表示不要递归，这一点非常重要，否则如果键对应的内容中又出现键本身，就会引发递归（相当于进入了死循环）。如果希望映射的快捷键每次启动vim时都能生效，需要将映射写到用户主目录下的.vimrc文件中。

- 录制宏。

- 在命令模式下输入 qa 开始录制宏（其中 a 是寄存器的名字，也可以是其他英文字母或0-9的数字）。
- 执行你的操作（光标操作、编辑操作等），这些操作都会被录制下来。
- 如果录制的操作已经完成了，按 q 结束录制。
- 通过 @a （ a 是刚才使用的寄存器的名字）播放宏，如果要多次执行宏可以在前面加数字，例如 100@a 表示将宏播放100次。
- 可以试一试下面的例子来体验录制宏的操作，该例子来源于[Harttle Land网站](#)，该网站上提供了很多关于vim的使用技巧，有兴趣的可以去了解一下。

```
1 | BOOL | Boolean
2 | SINT | Short integer
3 | INT | Integer
4 | DINT | Double integer
5 | LINT | Long integer
6 | USINT | Unsigned short integer
7 | UINT | Unsigned integer
```

1. 首先按几次`<Esc>`进入normal模式，光标移到第一行，开始录制并存入m寄存器`qm`。
2. 光标到行首`^`，到第二列词首`ww`，进入插入模式`i`，插入分隔符`,`，退出到normal模式`<Esc>`，到词尾`e`，进入插入模式`i`，插入分隔符`,`，退出到normal模式`<Esc>`，光标到下一行`j`。
3. 结束录制`q`。
4. 光标到第二行，在normal模式执行100次寄存器m中的宏`100@m`。

宏会在`j`执行错误后自动结束，得到如下文件：

```
1 | `BOOL` | Boolean
2 | `SINT` | Short integer
3 | `INT` | Integer
4 | `DINT` | Double integer
5 | `LINT` | Long integer
6 | `USINT` | Unsigned short integer
7 | `UINT` | Unsigned integer
```

## 软件安装和配置

### 使用包管理工具

#### 1. yum - Yellowdog Updater Modified。

- `yum search`：搜索软件包，例如`yum search nginx`。
- `yum list installed`：列出已经安装的软件包，例如`yum list installed | grep zlib`。
- `yum install`：安装软件包，例如`yum install nginx`。
- `yum remove`：删除软件包，例如`yum remove nginx`。
- `yum update`：更新软件包，例如`yum update`可以更新所有软件包，而`yum update tar`只会更新tar。
- `yum check-update`：检查有哪些可以更新的软件包。
- `yum info`：显示软件包的相关信息，例如`yum info nginx`。

#### 2. rpm - Redhat Package Manager。

- 安装软件包：`rpm -ivh <packagename>.rpm`。
- 移除软件包：`rpm -e <packagename>`。
- 查询软件包：`rpm -qa`，例如可以用`rpm -qa | grep mysql`来检查是否安装了MySQL相关的软件包。

下面以Nginx为例，演示如何使用yum安装软件。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# yum -y install nginx
...
Installed:
  nginx.x86_64 1:1.12.2-2.el7
Dependency Installed:
  nginx-all-modules.noarch 1:1.12.2-2.el7
```

```
nginx-mod-http-geoip.x86_64 1:1.12.2-2.el7
nginx-mod-http-image-filter.x86_64 1:1.12.2-2.el7
nginx-mod-http-perl.x86_64 1:1.12.2-2.el7
nginx-mod-http-xslt-filter.x86_64 1:1.12.2-2.el7
nginx-mod-mail.x86_64 1:1.12.2-2.el7
nginx-mod-stream.x86_64 1:1.12.2-2.el7
Complete!
[root@iZwz97tbgo91kabnat2lo8Z ~]# yum info nginx
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Installed Packages
Name        : nginx
Arch       : x86_64
Epoch      : 1
Version    : 1.12.2
Release    : 2.el7
Size       : 1.5 M
Repo       : installed
From repo  : epel
Summary    : A high performance web server and reverse proxy server
URL        : http://nginx.org/
License    : BSD
Description: Nginx is a web server and a reverse proxy server for HTTP, SMTP, POP3 and
             : IMAP protocols, with a strong focus on high concurrency, performance and
             : memory usage.
[root@iZwz97tbgo91kabnat2lo8Z ~]# nginx -v
nginx version: nginx/1.12.2
```

移除Nginx。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# nginx -s stop
[root@iZwz97tbgo91kabnat2lo8Z ~]# yum -y remove nginx
```

下面以MySQL为例，演示如何使用rpm安装软件。要安装MySQL需要先到[MySQL官方网站](#)下载对应的[RPM文件](#)，当然要选择和你使用的Linux系统对应的版本。MySQL现在是Oracle公司旗下的产品，在MySQL被收购后，MySQL的作者重新制作了一个MySQL的分支MariaDB，可以通过yum进行安装。如果要安装MySQL需要先通过yum删除 mariadb-libs 这个可能会跟MySQL底层库冲突的库，然后还需要安装一个名为 libaio 的依赖库。

```
[root@iZwz97tbgo91kabnat2lo8Z mysql]# ls
mysql-community-client-5.7.22-1.el7.x86_64.rpm
mysql-community-common-5.7.22-1.el7.x86_64.rpm
mysql-community-libs-5.7.22-1.el7.x86_64.rpm
mysql-community-server-5.7.22-1.el7.x86_64.rpm
[root@iZwz97tbgo91kabnat2lo8Z mysql]# yum -y remove mariadb-libs
[root@iZwz97tbgo91kabnat2lo8Z mysql]# yum -y install libaio
[root@iZwz97tbgo91kabnat2lo8Z mysql]# ls | xargs rpm -ivh
warning: mysql-community-client-5.7.22-1.el7.x86_64.rpm: Header V3 DSA/SHA1 Signature,
```

Preparing... ##### [100%]

...

说明：由于MySQL和MariaDB的底层依赖库是有冲突的，所以上面我们首先用 yum 移除了名为mariadb-libs的依赖库并安装了名为libaio的依赖库。由于我们将安装MySQL所需的rpm文件放在一个独立的目录中，所以可以通过 ls 命令查看到安装文件并用 xargs 将 ls 的输出作为参数交给 rpm -ivh 来进行安装。关于MySQL和MariaDB之间的关系，可以阅读维基百科上关于MariaDB的介绍。

移除安装的MySQL。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# rpm -qa | grep mysql | xargs rpm -e
```

## 下载解压配置环境变量

下面以安装MongoDB为例，演示这类软件应该如何安装。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# wget https://fastdl.mongodb.org/linux/mongodb-linux- --2018-06-21 18:32:53-- https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70- Resolving fastdl.mongodb.org (fastdl.mongodb.org)... 52.85.83.16, 52.85.83.228, 52.85. Connecting to fastdl.mongodb.org (fastdl.mongodb.org)|52.85.83.16|:443... connected. HTTP request sent, awaiting response... 200 OK Length: 100564462 (96M) [application/x-gzip] Saving to: 'mongodb-linux-x86_64-rhel70-3.6.5.tgz' 100%[=====] 100,564,462 630KB/s in 2m 2018-06-21 18:35:04 (760 KB/s) - 'mongodb-linux-x86_64-rhel70-3.6.5.tgz' saved [100564 [root@iZwz97tbgo91kabnat2lo8Z ~]# gunzip mongodb-linux-x86_64-rhel70-3.6.5.tgz [root@iZwz97tbgo91kabnat2lo8Z ~]# tar -xvf mongodb-linux-x86_64-rhel70-3.6.5.tar mongodb-linux-x86_64-rhel70-3.6.5/README mongodb-linux-x86_64-rhel70-3.6.5/THIRD-PARTY-NOTICES mongodb-linux-x86_64-rhel70-3.6.5/MPL-2 mongodb-linux-x86_64-rhel70-3.6.5/GNU-AGPL-3.0 mongodb-linux-x86_64-rhel70-3.6.5/bin/mongodump mongodb-linux-x86_64-rhel70-3.6.5/bin/mongorestore mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoexport mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoimport mongodb-linux-x86_64-rhel70-3.6.5/bin/mongostat mongodb-linux-x86_64-rhel70-3.6.5/bin/mongotop mongodb-linux-x86_64-rhel70-3.6.5/bin/bsondump mongodb-linux-x86_64-rhel70-3.6.5/bin/mongofiles mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoreplay mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoperf mongodb-linux-x86_64-rhel70-3.6.5/bin/mongod mongodb-linux-x86_64-rhel70-3.6.5/bin/mongos mongodb-linux-x86_64-rhel70-3.6.5/bin/mongo mongodb-linux-x86_64-rhel70-3.6.5/bin/install_compass [root@iZwz97tbgo91kabnat2lo8Z ~]# vim .bash_profile ... PATH=$PATH:$HOME/bin:$HOME/mongodb-linux-x86_64-rhel70-3.6.5/bin
```

```
export PATH
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# source .bash_profile
[root@iZwz97tbgo91kabnat2lo8Z ~]# mongod --version
db version v3.6.5
git version: a20ecd3e3a174162052ff99913bc2ca9a839d618
OpenSSL version: OpenSSL 1.0.1e-fips 11 Feb 2013
allocator: tcmalloc
modules: none
build environment:
  distmod: rhel70
  distarch: x86_64
  target_arch: x86_64
[root@iZwz97tbgo91kabnat2lo8Z ~]# mongo --version
MongoDB shell version v3.6.5
git version: a20ecd3e3a174162052ff99913bc2ca9a839d618
OpenSSL version: OpenSSL 1.0.1e-fips 11 Feb 2013
allocator: tcmalloc
modules: none
build environment:
  distmod: rhel70
  distarch: x86_64
  target_arch: x86_64
```

说明：当然也可以通过yum来安装MongoDB，具体可以参照[官方网站](#)上给出的说明。

## 源代码构建安装

### 1. 安装Python 3.6。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# yum install gcc
[root@iZwz97tbgo91kabnat2lo8Z ~]# wget https://www.python.org/ftp/python/3.6.5/Pyt
[root@iZwz97tbgo91kabnat2lo8Z ~]# gunzip Python-3.6.5.tgz
[root@iZwz97tbgo91kabnat2lo8Z ~]# tar -xvf Python-3.6.5.tar
[root@iZwz97tbgo91kabnat2lo8Z ~]# cd Python-3.6.5
[root@iZwz97tbgo91kabnat2lo8Z ~]# ./configure --prefix=/usr/local/python36 --enab
[root@iZwz97tbgo91kabnat2lo8Z ~]# yum -y install zlib-devel bzip2-devel openssl-de
[root@iZwz97tbgo91kabnat2lo8Z ~]# make && make install
... 配置环境变量 ...
[root@iZwz97tbgo91kabnat2lo8Z ~]# ln -s /usr/local/python36/bin/python3.6 /usr/bir
[root@iZwz97tbgo91kabnat2lo8Z ~]# python3 --version
Python 3.6.5
[root@iZwz97tbgo91kabnat2lo8Z ~]# python3 -m pip install -U pip
[root@iZwz97tbgo91kabnat2lo8Z ~]# pip3 --version
```

### 2. 安装Redis-3.2.12。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# wget http://download.redis.io/releases/redis-3.2.12.tar.gz
[root@iZwz97tbgo91kabnat2lo8Z ~]# gunzip redis-3.2.12.tar.gz
[root@iZwz97tbgo91kabnat2lo8Z ~]# tar -xvf redis-3.2.12.tar
[root@iZwz97tbgo91kabnat2lo8Z ~]# cd redis-3.2.12
[root@iZwz97tbgo91kabnat2lo8Z ~]# make && make install
[root@iZwz97tbgo91kabnat2lo8Z ~]# redis-server --version
Redis server v=3.2.12 sha=00000000:0 malloc=jemalloc-4.0.3 bits=64 build=5bc5cd3ce
[root@iZwz97tbgo91kabnat2lo8Z ~]# redis-cli --version
redis-cli 3.2.12
```



## 配置服务

1. 启动服务。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# systemctl start firewalld
```

2. 终止服务。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# systemctl stop firewalld
```

3. 重启服务。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# systemctl restart firewalld
```

4. 查看服务。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# systemctl status firewalld
```

5. 设置是否开机自启。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# systemctl enable firewalld
Created symlink from /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service to /etc/systemd/system/multi-user.target.wants/firewalld.service.
[root@iZwz97tbgo91kabnat2lo8Z ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```



## 计划任务

1. crontab命令。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# crontab -e
* * * * * echo "hello, world!" >> /root/hello.txt
59 23 * * * rm -f /root/*.log
```

说明：输入 crontab -e 命令会打开vim来编辑Cron表达式并指定触发的任务，上面我们定制了两个计划任务，一个是每分钟向/root目录下的hello.txt中追加输出 hello, world!；另一个是每天23时59分执行删除/root目录下以log为后缀名的文件。如果不知道Cron表达式如何书写，可以参照/etc/crontab文件中的提示（下面会讲到）或者用谷歌或百度搜索一下，也可以使用Cron表达式在线生成器来生成Cron表达式。

## 2. crontab相关文件。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# cd /etc
[root@iZwz97tbgo91kabnat2lo8Z etc]# ls -l | grep cron
-rw-----. 1 root root      541 Aug  3  2017 anacrontab
drwxr-xr-x. 2 root root    4096 Mar 27 11:56 cron.d
drwxr-xr-x. 2 root root    4096 Mar 27 11:51 cron.daily
-rw-----. 1 root root      0 Aug  3  2017 cron.deny
drwxr-xr-x. 2 root root    4096 Mar 27 11:50 cron.hourly
drwxr-xr-x. 2 root root    4096 Jun 10  2014 cron.monthly
-rw-r--r--  1 root root     493 Jun 23 15:09 crontab
drwxr-xr-x. 2 root root    4096 Jun 10  2014 cron.weekly
[root@iZwz97tbgo91kabnat2lo8Z etc]# vim crontab
 1 SHELL=/bin/bash
 2 PATH=/sbin:/bin:/usr/sbin:/usr/bin
 3 MAILTO=root
 4
 5 # For details see man 4 crontabs
 6
 7 # Example of job definition:
 8 # ----- minute (0 - 59)
 9 # | ----- hour (0 - 23)
10 # | | ----- day of month (1 - 31)
11 # | | | ----- month (1 - 12) OR jan,feb,mar,apr ...
12 # | | | | ----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu
13 # | | | |
14 # * * * * * user-name command to be executed
```

通过修改 /etc 目录下的crontab文件也能够定制计划任务。

## 网络访问和管理

1. 安全远程连接 - ssh。
2. 通过网络获取资源 - wget。
  - -b 后台下载模式

- o -O 下载到指定的目录
- o -r 递归下载

### 3. 显示/操作网络配置 (旧) - ifconfig。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.18.61.250 netmask 255.255.240.0 broadcast 172.18.63.255
              ether 00:16:3e:02:b6:46 txqueuelen 1000 (Ethernet)
                    RX packets 1067841 bytes 1296732947 (1.2 GiB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 409912 bytes 43569163 (41.5 MiB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions
```

### 4. 显示/操作网络配置 (新) - ip。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1
    link/ether 00:16:3e:02:b6:46 brd ff:ff:ff:ff:ff:ff
      inet 172.18.61.250/20 brd 172.18.63.255 scope global eth0
        valid_lft forever preferred_lft forever
```

### 5. 网络可达性检查 - ping。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ping www.baidu.com -c 3
PING www.a.shifen.com (220.181.111.188) 56(84) bytes of data.
64 bytes from 220.181.111.188 (220.181.111.188): icmp_seq=1 ttl=51 time=36.3 ms
64 bytes from 220.181.111.188 (220.181.111.188): icmp_seq=2 ttl=51 time=36.4 ms
64 bytes from 220.181.111.188 (220.181.111.188): icmp_seq=3 ttl=51 time=36.4 ms
--- www.a.shifen.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 36.392/36.406/36.427/0.156 ms
```

### 6. 查看网络服务和端口 - netstat。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# netstat -nap | grep nginx
```

### 7. 安全文档拷贝 - scp。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# scp root@1.2.3.4:/root/guido.jpg hellokitty@4.3.
```

## 8. 安全文件传输 - sftp。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# sftp root@120.77.222.217
root@120.77.222.217's password:
Connected to 120.77.222.217.
sftp>
```

- o help : 显示帮助信息。
- o ls / ll : 显示远端/本地目录列表。
- o cd / lcd : 切换远端/本地路径。
- o mkdir / lmkdir : 创建远端/本地目录。
- o pwd / lpwd : 显示远端/本地当前工作目录。
- o get : 下载文件。
- o put : 上传文件。
- o rm : 删除远端文件。
- o bye / exit / quit : 退出sftp。

## 进程管理

### 1. ps - 查询进程。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ps -ef
UID      PID  PPID  C STIME TTY          TIME CMD
root        1      0  0 Jun23 ?        00:00:05 /usr/lib/systemd/systemd --switchch
root        2      0  0 Jun23 ?        00:00:00 [kthreadd]
...
[root@iZwz97tbgo91kabnat2lo8Z ~]# ps -ef | grep mysqld
root      4943  4581  0 22:45 pts/0    00:00:00 grep --color=auto mysqld
mysql    25257      1  0 Jun25 ?        00:00:39 /usr/sbin/mysqld --daemonize --pic
```

### 2. kill - 终止进程。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# kill 1234
[root@iZwz97tbgo91kabnat2lo8Z ~]# kill -9 1234
```

例子：用一条命令强制终止正在运行的Redis进程。

```
ps -ef | grep redis | grep -v grep | awk '{print $2}' | xargs kill
```

### 3. 将进程置于后台运行。

- o Ctrl+Z
- o &

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# mongod &  
[root@iZwz97tbgo91kabnat2lo8Z ~]# redis-server  
...  
^Z  
[4]+ Stopped redis-server
```

### 4. jobs - 查询后台进程。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# jobs  
[2] Running mongod &  
[3]- Stopped cat  
[4]+ Stopped redis-server
```

### 5. bg - 让进程在后台继续运行。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# bg %4  
[4]+ redis-server &  
[root@iZwz97tbgo91kabnat2lo8Z ~]# jobs  
[2] Running mongod &  
[3]+ Stopped cat  
[4]- Running redis-server &
```

### 6. fg - 将后台进程置于前台。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# fg %4  
redis-server  
^C5554:signal-handler (1530025281) Received SIGINT scheduling shutdown...  
5554:M 26 Jun 23:01:21.413 # User requested shutdown...  
5554:M 26 Jun 23:01:21.413 * Saving the final RDB snapshot before exiting.  
5554:M 26 Jun 23:01:21.415 * DB saved on disk  
5554:M 26 Jun 23:01:21.415 # Redis is now ready to exit, bye bye...
```

说明：置于前台的进程可以使用 Ctrl+C 来终止它。

### 7. top - 进程监控。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# top  
top - 23:04:23 up 3 days, 14:10, 1 user, load average: 0.00, 0.01, 0.05  
Tasks: 65 total, 1 running, 64 sleeping, 0 stopped, 0 zombie
```

```
%Cpu(s): 0.3 us, 0.3 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st  
KiB Mem : 1016168 total, 191060 free, 324700 used, 500408 buff/cache  
KiB Swap: 0 total, 0 free, 0 used. 530944 avail Mem  
...
```

## 系统性能

1. 查看系统活动信息 - sar。

2. 查看内存使用情况 - free。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# free  
total used free shared buff/cache available  
Mem: 1016168 323924 190452 356 501792 531800  
Swap: 0 0 0
```

3. 查看进程使用内存状况 - pmap。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ps  
PID TTY TIME CMD  
4581 pts/0 00:00:00 bash  
5664 pts/0 00:00:00 ps  
[root@iZwz97tbgo91kabnat2lo8Z ~]# pmap 4581  
4581: -bash  
000000000400000 884K r-x-- bash  
00000000006dc000 4K r---- bash  
000000000006dd000 36K rw--- bash  
000000000006e6000 24K rw--- [ anon ]  
0000000001de0000 400K rw--- [ anon ]  
00007f82fe805000 48K r-x-- libnss_files-2.17.so  
00007f82fe811000 2044K ----- libnss_files-2.17.so  
...
```

4. 报告设备CPU和I/O统计信息 - iostat。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# iostat  
Linux 3.10.0-693.11.1.el7.x86_64 (iZwz97tbgo91kabnat2lo8Z) 06/26/2018  
avg-cpu: %user %nice %system %iowait %steal %idle  
          0.79   0.00   0.20   0.04   0.00  98.97  
Device:    tps kB_read/s kB_wrtn/s kB_read kB_wrtn  
vda       0.85      6.78     21.32  2106565  6623024  
vdb       0.00      0.01      0.00    2088      0
```



## Python-100-Days / Day36-40 / 关系型数据库MySQL.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



699 lines (533 sloc) 24.7 KB

# 关系数据库入门

## 关系数据库概述

1. 数据持久化 - 将数据保存到（在掉电情况下）能够长久保存数据的存储介质中。
2. 数据库发展史 - 网状数据库、层次数据库、关系数据库、NoSQL数据库。

1970年，IBM的研究员E.F.Codd在*Communication of the ACM*上发表了名为*A Relational Model of Data for Large Shared Data Banks*的论文，提出了关系模型的概念，奠定了关系模型的理论基础。后来Codd又陆续发表多篇文章，论述了范式理论和衡量关系系统的12条标准，用数学理论奠定了关系数据库的基础。

3. 关系数据库特点。
  - 理论基础：集合论和关系代数。
  - 具体表象：用二维表（有行和列）组织数据。
  - 编程语言：结构化查询语言（SQL）。
4. E-R图。
  - 实体 - 矩形框
  - 属性 - 椭圆框
  - 关系 - 菱形框
  - 重数 - 1:1 / 1:N / M:N
5. 关系数据库产品。
  - **Oracle** - 目前世界上使用最为广泛的数据库管理系统，作为一个通用的数据库系统，它具有完整的数据管理功能；作为一个关系数据库，它是一个完备关系的

产品；作为分布式数据库，它实现了分布式处理的功能。在Oracle最新的12c版本中，还引入了多租户架构，使用该架构可轻松部署和管理数据库云。

- [DB2](#) - IBM公司开发的、主要运行于Unix（包括IBM自家的[AIX](#)）、Linux、以及Windows服务器版等系统的关系数据库产品。DB2历史悠久且被认为是最早使用SQL的数据库产品，它拥有较为强大的商业智能功能。
- [SQL Server](#) - 由Microsoft开发和推广的关系型数据库产品，最初适用于中小企业的数据管理，但是近年来它的应用范围有所扩展，部分大企业甚至是跨国公司也开始基于它来构建自己的数据管理系统。
- [MySQL](#) - MySQL是开放源代码的，任何人都可以在GPL（General Public License）的许可下下载并根据个性化的需要对其进行修改。MySQL因为其速度、可靠性和适应性而备受关注。
- [PostgreSQL](#) - 在BSD许可证下发行的开放源代码的关系数据库产品。

## MySQL简介

1. 安装和配置（以CentOS Linux环境为例）。

- Linux下有一个MySQL的分支版本，名为MariaDB，它由MySQL的一些原始开发者开发，有商业支持，旨在继续保持MySQL数据库在[GNU GPL](#)下开源（因为大家担心MySQL被甲骨文收购后会不再开源）。如果决定要直接使用MariaDB作为MySQL的替代品，可以使用下面的命令进行安装。

```
yum install mariadb mariadb-server
```

- 如果要安装官方版本的MySQL，可以在[MySQL官方网站](#)下载安装文件。首先在下载页面中选择平台和版本，然后找到对应的下载链接。下面以MySQL 5.7.26版本和Red Hat Enterprise Linux为例，直接下载包含所有安装文件的归档文件，解归档之后通过包管理工具进行安装。

```
wget https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-5.7.26-1.el7.x86_64.tar -xvf mysql-5.7.26-1.el7.x86_64.rpm-bundle.tar
```



如果系统上有MariaDB相关的文件，需要先移除MariaDB相关的文件。

```
yum list installed | grep mariadb | awk '{print $1}' | xargs yum erase -y
```

接下来可以按照如下所示的顺序用RPM（Redhat Package Manager）工具安装MySQL。

```
rpm -ivh mysql-community-common-5.7.26-1.el7.x86_64.rpm  
rpm -ivh mysql-community-libs-5.7.26-1.el7.x86_64.rpm
```

```
rpm -ivh mysql-community-client-5.7.26-1.el7.x86_64.rpm  
rpm -ivh mysql-community-server-5.7.26-1.el7.x86_64.rpm
```

可以使用下面的命令查看已经安装的MySQL相关的包。

```
rpm -qa | grep mysql
```

- 启动MySQL服务。

先修改MySQL的配置文件 ( /etc/my.cnf ) 添加一行 skip-grant-tables , 可以设置不进行身份验证即可连接MySQL服务器 , 然后就可以以超级管理员 ( root ) 身份登录。

```
vim /etc/my.cnf
```

```
[mysqld]  
skip-grant-tables  
  
datadir=/var/lib/mysql  
socket=/var/lib/mysql/mysql.sock  
  
symbolic-links=0  
  
log-error=/var/log/mysqld.log  
pid-file=/var/run/mysqld/mysqld.pid
```

接下来可以使用下面的命令来启动MySQL。

```
service mysqld start
```

在CentOS 7中建议使用下面的命令来启动MySQL。

```
systemctl start mysqld
```

- 使用MySQL客户端工具连接服务器。

命令行工具 :

```
mysql -u root
```

修改超级管理员 ( root ) 的访问口令为i\_LOVE\_macos\_123。

```
use mysql;
update user set authentication_string=password('i_LOVE_macos_123') where user='root';
flush privileges;
```

将MySQL配置文件中的 skip-grant-tables 去掉，然后重启服务器，重新登录。这一次需要提供用户名和口令才能连接MySQL服务器。

```
systemctl restart mysqld
mysql -u root -p
```

也可以选择图形化的客户端工具来连接MySQL服务器，可以选择下列工具之一：

- MySQL Workbench ( 官方提供的工具 )
- Navicat for MySQL ( 界面简单优雅，功能直观强大 )
- SQLyog for MySQL ( 强大的MySQL数据库管理員工具 )

## 2. 常用命令。

- 查看服务器版本。

```
select version();
```

- 查看所有数据库。

```
show databases;
```

- 切换到指定数据库。

```
use mysql;
```

- 查看数据库下所有表。

```
show tables;
```

- 获取帮助。

```
? contents;
? functions;
? numeric functions;
? round;
```

```
? data types;  
? longblob;
```

## SQL详解

### 1. DDL

```
-- 如果存在名为school的数据库就删除它  
drop database if exists school;  
  
-- 创建名为school的数据库并设置默认的字符集和排序方式  
create database school default charset utf8 collate utf8_bin;  
  
-- 切换到school数据库上下文环境  
use school;  
  
-- 创建学院表  
create table tb_college  
(  
    collid int not null auto_increment comment '编号',  
    collname varchar(50) not null comment '名称',  
    collmaster varchar(20) not null comment '院长',  
    collweb varchar(511) default '' comment '网站',  
    primary key (collid)  
);  
  
-- 创建学生表  
create table tb_student  
(  
    stuid int not null comment '学号',  
    stuname varchar(20) not null comment '姓名',  
    stusex bit default 1 comment '性别',  
    stubirth date not null comment '出生日期',  
    stuaddr varchar(255) default '' comment '籍贯',  
    collid int not null comment '所属学院',  
    primary key (stuid),  
    foreign key (collid) references tb_college (collid)  
);  
  
-- alter table tb_student add constraint fk_student_collid foreign key (collid) re  
  
-- 创建教师表  
create table tb_teacher  
(  
    teaid int not null comment '工号',  
    teaname varchar(20) not null comment '姓名',  
    teatitle varchar(10) default '助教' comment '职称',  
    collid int not null comment '所属学院',  
    primary key (teaid),  
    foreign key (collid) references tb_college (collid)  
);
```

```

-- 创建课程表
create table tb_course
(
couid int not null comment '编号',
couname varchar(50) not null comment '名称',
coucredit int not null comment '学分',
teaid int not null comment '授课老师',
primary key (couid),
foreign key (teaid) references tb_teacher (teaid)
);

-- 创建选课记录表
create table tb_score
(
scid int auto_increment comment '选课记录编号',
stuid int not null comment '选课学生',
couid int not null comment '所选课程',
scdate datetime comment '选课时间日期',
scmark decimal(4,1) comment '考试成绩',
primary key (scid),
foreign key (stuid) references tb_student (stuid),
foreign key (couid) references tb_course (couid)
);

-- 添加唯一性约束 (一个学生选某个课程只能选一次)
alter table tb_score add constraint uni_score_stuid_couid unique (stuid, couid);

```

## 2. DML

```

-- 插入学院数据
insert into tb_college (collname, collmaster, collweb) values
('计算机学院', '左冷禅', 'http://www.abc.com'),
('外国语学院', '岳不群', 'http://www.xyz.com'),
('经济管理学院', '风清扬', 'http://www.foo.com');

-- 插入学生数据
insert into tb_student (stuid, stuname, stusex, stubirth, stuaddr, collid) values
(1001, '杨逍', 1, '1990-3-4', '四川成都', 1),
(1002, '任我行', 1, '1992-2-2', '湖南长沙', 1),
(1033, '王语嫣', 0, '1989-12-3', '四川成都', 1),
(1572, '岳不群', 1, '1993-7-19', '陕西咸阳', 1),
(1378, '纪嫣然', 0, '1995-8-12', '四川绵阳', 1),
(1954, '林平之', 1, '1994-9-20', '福建莆田', 1),
(2035, '东方不败', 1, '1988-6-30', null, 2),
(3011, '林震南', 1, '1985-12-12', '福建莆田', 3),
(3755, '项少龙', 1, '1993-1-25', null, 3),
(3923, '杨不悔', 0, '1985-4-17', '四川成都', 3),
(4040, '隔壁老王', 1, '1989-1-1', '四川成都', 2);

-- 删除学生数据
delete from tb_student where stuid=4040;

```

```

-- 更新学生数据
update tb_student set stuname='杨过', stuaddr='湖南长沙' where stuid=1001;

-- 插入老师数据
insert into tb_teacher (teaid, teaname, teatitle, collid) values
(1122, '张三丰', '教授', 1),
(1133, '宋远桥', '副教授', 1),
(1144, '杨逍', '副教授', 1),
(2255, '范遥', '副教授', 2),
(3366, '韦一笑', '讲师', 3);

-- 插入课程数据
insert into tb_course (couid, couname, coucredit, teaid) values
(1111, 'Python程序设计', 3, 1122),
(2222, 'Web前端开发', 2, 1122),
(3333, '操作系统', 4, 1122),
(4444, '计算机网络', 2, 1133),
(5555, '编译原理', 4, 1144),
(6666, '算法和数据结构', 3, 1144),
(7777, '经贸法语', 3, 2255),
(8888, '成本会计', 2, 3366),
(9999, '审计学', 3, 3366);

-- 插入选课数据
insert into tb_score (stuid, couid, scdate, scmark) values
(1001, 1111, '2017-09-01', 95),
(1001, 2222, '2017-09-01', 87.5),
(1001, 3333, '2017-09-01', 100),
(1001, 4444, '2018-09-03', null),
(1001, 6666, '2017-09-02', 100),
(1002, 1111, '2017-09-03', 65),
(1002, 5555, '2017-09-01', 42),
(1033, 1111, '2017-09-03', 92.5),
(1033, 4444, '2017-09-01', 78),
(1033, 5555, '2017-09-01', 82.5),
(1572, 1111, '2017-09-02', 78),
(1378, 1111, '2017-09-05', 82),
(1378, 7777, '2017-09-02', 65.5),
(2035, 7777, '2018-09-03', 88),
(2035, 9999, curdate(), null),
(3755, 1111, date(now()), null),
(3755, 8888, date(now()), null),
(3755, 9999, '2017-09-01', 92);

```

### 3. DQL

```

-- 查询所有学生信息
select * from tb_student;

-- 查询所有课程名称及学分(投影和别名)
select couname, coucredit from tb_course;

```

```
select couname as 课程名称, coucredit as 学分 from tb_course;

-- 查询所有学生的姓名和性别(条件运算)
select stuname as 姓名, case stusex when 1 then '男' else '女' end as 性别 from tb_student;
select stuname as 姓名, if(stusex, '男', '女') as 性别 from tb_student;

-- 查询所有女学生的姓名和出生日期(筛选)
select stuname, stubirth from tb_student where stusex=0;

-- 查询所有80后学生的姓名、性别和出生日期(筛选)
select stuname, stusex, stubirth from tb_student where stubirth>='1980-1-1' and stusex=1;
select stuname, stusex, stubirth from tb_student where stubirth between '1980-1-1' and '2000-1-1';

-- 查询姓"杨"的学生姓名和性别(模糊)
select stuname, stusex from tb_student where stuname like '杨%';

-- 查询姓"杨"名字两个字的学生姓名和性别(模糊)
select stuname, stusex from tb_student where stuname like '杨_';

-- 查询姓"杨"名字三个字的学生姓名和性别(模糊)
select stuname, stusex from tb_student where stuname like '杨__';

-- 查询名字中有"不"字或"嫣"字的学生成绩(模糊)
select stuname, stusex from tb_student where stuname like '%不%' or stuname like '%嫣%';

-- 查询没有录入家庭住址的学生姓名(空值)
select stuname from tb_student where stuaddr is null;

-- 查询录入了家庭住址的学生姓名(空值)
select stuname from tb_student where stuaddr is not null;

-- 查询学生选课的所有日期(去重)
select distinct scdate from tb_score;

-- 查询学生的家庭住址(去重)
select distinct stuaddr from tb_student where stuaddr is not null;

-- 查询男学生的姓名和生日按年龄从大到小排列(排序)
-- asc (ascending) - 升序 (从小到大) / desc (descending) - 降序 (从大到小)
select stuname as 姓名, year(now())-year(stubirth) as 年龄 from tb_student where stusex=1 order by age desc;

-- 聚合函数: max / min / count / sum / avg
-- 查询年龄最大的学生的出生日期(聚合函数)
select min(stubirth) from tb_student;

-- 查询年龄最小的学生成绩(聚合函数)
select max(stubirth) from tb_student;

-- 查询男女学生的人数(分组和聚合函数)
select stusex, count(*) from tb_student group by stusex;

-- 查询课程编号为1111的课程的平均成绩(筛选和聚合函数)
select avg(scmark) from tb_score where couid=1111;

-- 查询学号为1001的学生所有课程的平均分(筛选和聚合函数)
```

```

select avg(scmark) from tb_score where stuid=1001;

-- 查询每个学生的学号和平均成绩(分组和聚合函数)
select stuid as 学号, avg(scmark) as 平均分 from tb_score group by stuid;

-- 查询平均成绩大于等于90分的学生的学号和平均成绩
-- 分组以前的筛选使用where子句 / 分组以后的筛选使用having子句
select stuid as 学号, avg(scmark) as 平均分 from tb_score group by stuid having 平均分 >= 90;

-- 查询年龄最大的学生的姓名(子查询/嵌套的查询)
select stuname from tb_student where stubirth=(  

    select min(stubirth) from tb_student  

);

-- 查询年龄最大的学生姓名和年龄(子查询+运算)
select stuname as 姓名, year(now())-year(stubirth) as 年龄 from tb_student where stuid=(  

    select min(stubirth) from tb_student  

);

-- 查询选了两门以上的课程的学生姓名(子查询/分组条件/集合运算)
select stuname from tb_student where stuid=(  

    select stuid from tb_score group by stuid having count(stuid)>2  

);

-- 查询学生姓名、课程名称以及成绩(连接查询)
select stuname, couname, scmark from tb_student t1, tb_course t2, tb_score t3 where t1.stuid=t2.id and t1.stuid=t3.stuid;

-- 查询学生姓名、课程名称以及成绩按成绩从高到低查询第11-15条记录(内连接+分页)
select stuname, couname, scmark from tb_student t1 inner join tb_score t3 on t1.stuid=t3.stuid limit 10,5;

select stuname, couname, scmark from tb_student t1 inner join tb_score t3 on t1.stuid=t3.stuid order by scmark desc limit 10,5;

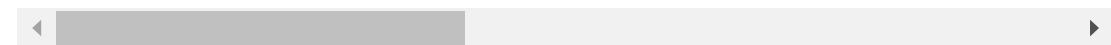
-- 查询选课学生的姓名和平均成绩(子查询和连接查询)
select stuname, avgmark from tb_student t1, (select stuid, avg(scmark) as avgmark from tb_score group by stuid) t2 on t1.stuid=t2.stuid;

select stuname, avgmark from tb_student t1 inner join  

(select stuid, avg(scmark) as avgmark from tb_score group by stuid) t2 on t1.stuid=t2.stuid;

-- 内连接 (inner join) - 只有满足连接条件的记录才会被查出来
-- 外连接 (outer join) - 左外连接(left outer join) / 右外连接(right outer join)
-- 查询每个学生的姓名和选课数量(左外连接和子查询)
select stuname, ifnull(total, 0) from tb_student t1 left outer join (select stuid, count(stuid) as total from tb_score group by stuid) t2 on t1.stuid=t2.stuid;

```



#### 4. DCL

```

-- 创建名为hellokitty的用户
create user 'hellokitty'@'%' identified by '123123';

-- 将对school数据库所有对象的所有操作权限授予hellokitty
grant all privileges on school.* to 'hellokitty'@'%';

```

```
-- 召回hellokitty对school数据库所有对象的insert/delete/update权限  
revoke insert, delete, update on school.* from 'hellokitty'@'%';
```

## 相关知识

### 范式理论 - 设计二维表的指导思想

1. 第一范式：数据表的每个列的值域都是由原子值组成的，不能够再分割。
2. 第二范式：数据表里的所有数据都要和该数据表的键（主键与候选键）有完全依赖关系。
3. 第三范式：所有非键属性都只和候选键有相关性，也就是说非键属性之间应该是独立无关的。

### 数据完整性

1. 实体完整性 - 每个实体都是独一无二的
  - 主键 ( primary key ) / 唯一约束 / 唯一索引 ( unique )
2. 引用完整性 ( 参照完整性 ) - 关系中不允许引用不存在的实体
  - 外键 ( foreign key )
3. 域完整性 - 数据是有效的
  - 数据类型及长度
  - 非空约束 ( not null )
  - 默认值约束 ( default )
  - 检查约束 ( check )

### 数据一致性

1. 事务：一系列对数据库进行读/写的操作。
2. 事务的ACID特性
  - 原子性：事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行
  - 一致性：事务应确保数据库的状态从一个一致状态转变为另一个一致状态
  - 隔离性：多个事务并发执行时，一个事务的执行不应影响其他事务的执行
  - 持久性：已被提交的事务对数据库的修改应该永久保存在数据库中

## Python数据库编程

我们用如下所示的数据库来演示在Python中如何访问MySQL数据库。

```
drop database if exists hrs;  
create database hrs default charset utf8;
```

```
use hrs;

drop table if exists tb_dept;
drop table if exists tb_emp;

create table tb_dept
(
dno int not null comment '编号',
dname varchar(10) not null comment '名称',
dloc varchar(20) not null comment '所在地',
primary key (dno)
);

insert into tb_dept values
(10, '会计部', '北京'),
(20, '研发部', '成都'),
(30, '销售部', '重庆'),
(40, '运维部', '深圳');

create table tb_emp
(
eno int not null comment '员工编号',
ename varchar(20) not null comment '员工姓名',
job varchar(20) not null comment '员工职位',
mgr int comment '主管编号',
sal int not null comment '员工月薪',
comm int comment '每月补贴',
dno int comment '所在部门编号',
primary key (eno)
);

alter table tb_emp add constraint fk_emp_dno foreign key (dno) references tb_dept (dno)

insert into tb_emp values
(7800, '张三丰', '总裁', null, 9000, 1200, 20),
(2056, '乔峰', '分析师', 7800, 5000, 1500, 20),
(3088, '李莫愁', '设计师', 2056, 3500, 800, 20),
(3211, '张无忌', '程序员', 2056, 3200, null, 20),
(3233, '丘处机', '程序员', 2056, 3400, null, 20),
(3251, '张翠山', '程序员', 2056, 4000, null, 20),
(5566, '宋远桥', '会计师', 7800, 4000, 1000, 10),
(5234, '郭靖', '出纳', 5566, 2000, null, 10),
(3344, '黄蓉', '销售主管', 7800, 3000, 800, 30),
(1359, '胡一刀', '销售员', 3344, 1800, 200, 30),
(4466, '苗人凤', '销售员', 3344, 2500, null, 30),
(3244, '欧阳锋', '程序员', 3088, 3200, null, 20),
(3577, '杨过', '会计', 5566, 2200, null, 10),
(3588, '朱九真', '会计', 5566, 2500, null, 10);
```

在Python 3中，我们通常使用纯Python的三方库PyMySQL来访问MySQL数据库，它应该是目前最好的选择。

## 1. 安装PyMySQL。

```
pip install pymysql
```

## 2. 添加一个部门。

```
import pymysql

def main():
    no = int(input('编号: '))
    name = input('名字: ')
    loc = input('所在地: ')
    # 1. 创建数据库连接对象
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='root', password='123456')
    try:
        # 2. 通过连接对象获取游标
        with con.cursor() as cursor:
            # 3. 通过游标执行SQL并获得执行结果
            result = cursor.execute(
                'insert into tb_dept values (%s, %s, %s)',
                (no, name, loc)
            )
            if result == 1:
                print('添加成功!')
        # 4. 操作成功提交事务
        con.commit()
    finally:
        # 5. 关闭连接释放资源
        con.close()

if __name__ == '__main__':
    main()
```

## 3. 删除一个部门。

```
import pymysql

def main():
    no = int(input('编号: '))
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='root', password='123456',
                          autocommit=True)
    try:
        with con.cursor() as cursor:
```

```
        result = cursor.execute(  
            'delete from tb_dept where dno=%s',  
            (no, ))  
    )  
    if result == 1:  
        print('删除成功！')  
finally:  
    con.close()  
  
if __name__ == '__main__':  
    main()
```

#### 4. 更新一个部门。

```
import pymysql

def main():
    no = int(input('编号: '))
    name = input('名字: ')
    loc = input('所在地: ')
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='root', password='123456',
                          autocommit=True)

    try:
        with con.cursor() as cursor:
            result = cursor.execute(
                'update tb_dept set dname=%s, dloc=%s where dno=%s',
                (name, loc, no)
            )
            if result == 1:
                print('更新成功!')
    finally:
        con.close()

if __name__ == '__main__':
    main()
```

5. 查询所有部门。

```

try:
    with con.cursor(cursor=DictCursor) as cursor:
        cursor.execute('select dno as no, dname as name, dloc as loc from tb_c
results = cursor.fetchall()
print(results)
print('编号\t名称\t所在地')
for dept in results:
    print(dept['no'], end='\t')
    print(dept['name'], end='\t')
    print(dept['loc'])
finally:
    con.close()

if __name__ == '__main__':
    main()

```

## 6. 分页查询员工信息。

```

import pymysql
from pymysql.cursors import DictCursor

class Emp(object):

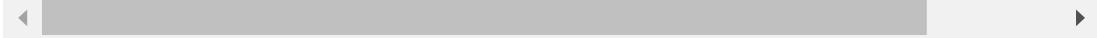
    def __init__(self, no, name, job, sal):
        self.no = no
        self.name = name
        self.job = job
        self.sal = sal

    def __str__(self):
        return f'\n编号: {self.no}\n姓名: {self.name}\n职位: {self.job}\n月薪: {self.sal}'

def main():
    page = int(input('页码: '))
    size = int(input('大小: '))
    con = pymysql.connect(host='localhost', port=3306,
                          database='hrs', charset='utf8',
                          user='root', password='123456')
    try:
        with con.cursor() as cursor:
            cursor.execute(
                'select eno as no, ename as name, job, sal from tb_emp limit %s,%s'
                '(%(page - 1) * size, size)')
    for emp_tuple in cursor.fetchall():
        emp = Emp(*emp_tuple)
        print(emp)
    finally:
        con.close()

```

```
if __name__ == '__main__':
    main()
```



Branch: master ▾

[Find file](#)[Copy path](#)

## Python-100-Days / Day36-40 / NoSQL入门.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

547 lines (459 sloc) 22.4 KB

# NoSQL入门

## NoSQL概述

如今，大多数的计算机系统（包括服务器、PC、移动设备等）都会产生庞大的数据量。其实，早在2012年的时候，全世界每天产生的数据量就达到了2.5EB（艾字节， $\$1EB \approx 10^{18} B\$$ ）。这些数据有很大一部分是由关系型数据库来存储和管理的。早在1970年，E.F.Codd发表了论述关系型数据库的著名论文“*A relational model of data for large shared data banks*”，这篇文章奠定了关系型数据库的基础并在接下来的数十年时间内产生了深远的影响。实践证明，关系型数据库是实现数据持久化最为重要的方式，它也是大多数应用在选择持久化方案时的首选技术。

NoSQL是一项全新的数据库革命性运动，虽然它的历史可以追溯到1998年，但是NoSQL真正深入人心并得到广泛的应用是在进入大数据时候以后，业界普遍认为NoSQL是更适合大数据存储的技术方案，这才使得NoSQL的发展达到了前所未有的高度。2012年《纽约时报》的一篇专栏中写到，大数据时代已经降临，在商业、经济及其他领域中，决策将不再基于经验和直觉而是基于数据和分析而作出。事实上，在天文学、气象学、基因组学、生物学、社会学、互联网搜索引擎、金融、医疗、社交网络、电子商务等诸多领域，由于数据过于密集和庞大，在数据的分析和处理上也遇到了前所未有的限制和阻碍，这一切都使得对大数据处理技术的研究被提升到了新的高度，也使得各种NoSQL的技术方案进入到了公众的视野。

NoSQL数据库按照其存储类型可以大致分为以下几类：

类型	部分代表	特点

列族数据库	HBase Cassandra Hypertable	顾名思义是按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的I/O优势，适合于批量数据处理和即时查询。
文档数据库	MongoDB CouchDB ElasticSearch	文档数据库一般用类JSON格式存储数据，存储的内容是文档型的。这样也就有机会对某些字段建立索引，实现关系数据库的某些功能，但不提供对参照完整性和分布事务的支持。
KV数据库	DynamoDB Redis LevelDB	可以通过key快速查询到其value，有基于内存和基于磁盘两种实现方案。
图数据库	Neo4J FlockDB JanusGraph	使用图结构进行语义查询的数据库，它使用节点、边和属性来表示和存储数据。图数据库从设计上，就可以简单快速的检索难以在关系系统中建模的复杂层次结构。
对象数据库	db4o Versant	通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。

说明：想了解更多的NoSQL数据库，可以访问<http://nosql-database.org/>。

## Redis概述

Redis是一种基于键值对的NoSQL数据库，它提供了对多种数据类型（字符串、哈希、列表、集合、有序集合、位图等）的支持，能够满足很多应用场景的需求。Redis将数据放在内存中，因此读写性能是非常惊人的。与此同时，Redis也提供了持久化机制，能够将内存中的数据保存到硬盘上，在发生意外状况时数据也不会丢掉。此外，Redis还支持键过期、地理信息运算、发布订阅、事务、管道、Lua脚本扩展等功能，总而言之，Redis的功能和性能都非常强大，如果项目中要实现高速缓存和消息队列这样的服务，直接交给Redis就可以了。目前，国内外很多著名的企业和商业项目都使用了Redis，包括：Twitter、Github、StackOverflow、新浪微博、百度、优酷土豆、美团、小米、唯品会等。

## Redis简介

2008年，一个名为Salvatore Sanfilippo的程序员为他开发的LLOOGG项目定制了专属的数据库（因为之前他无论怎样优化MySQL，系统性能已经无法再提升了），这项工作的成果就是Redis的初始版本。后来他将Redis的代码放到了全球最大的代码托管平台[Github](#)，从那以后，Redis引发了大量开发者的好评和关注，继而有数百人参与了Redis的开发和维护，这使得Redis的功能越来越强大和性能越来越好。

Redis是REmote DIctionary Server的缩写，它是一个用ANSI C编写的高性能的key-value存储系统，与其他的key-value存储系统相比，Redis有以下一些特点（也是优点）：

- Redis的读写性能极高，并且有丰富的特性（发布/订阅、事务、通知等）。
- Redis支持数据的持久化（RDB和AOF两种方式），可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
- Redis支持多种数据类型，包括：string、hash、list、set，zset、bitmap、hyperloglog等。
- Redis支持主从复制（实现读写分离）以及哨兵模式（监控master是否宕机并自动调整配置）。
- Redis支持分布式集群，可以很容易的通过水平扩展来提升系统的整体性能。
- Redis基于TCP提供的可靠传输服务进行通信，很多编程语言都提供了Redis客户端支持。

## Redis的应用场景

1. 高速缓存 - 将不常变化但又经常被访问的热点数据放到Redis数据库中，可以大大降低关系型数据库的压力，从而提升系统的响应性能。
2. 排行榜 - 很多网站都有排行榜功能，利用Redis中的列表和有序集合可以非常方便的构造各种排行榜系统。
3. 商品秒杀/投票点赞 - Redis提供了对计数操作的支持，网站上常见的秒杀、点赞等功能都可以利用Redis的计数器通过+1或-1的操作来实现，从而避免了使用关系型数据的update操作。
4. 分布式锁 - 利用Redis可以跨多台服务器实现分布式锁（类似于线程锁，但是能够被多台机器上的多个线程或进程共享）的功能，用于实现一个阻塞式操作。
5. 消息队列 - 消息队列和高速缓存一样，是一个大型网站不可缺少的基础服务，可以实现业务解耦和非实时业务削峰等特性，这些我们都会在后面的项目中为大家展示。

## Redis的安装和配置

可以使用Linux系统的包管理工具（如yum）来安装Redis，也可以通过在Redis的[官方网站](#)下载Redis的源代码，解压缩解归档之后通过make工具对源代码进行构建并安装，在更新这篇文档时，Redis官方提供的最新稳定版本是[Redis 5.0.4](#)。

```
 wget http://download.redis.io/releases/redis-5.0.4.tar.gz  
 gunzip redis-5.0.4.tar.gz  
 tar -xvf redis-5.0.4.tar
```

```
cd redis-5.0.4  
make && make install
```

在redis源代码目录下有一个名为redis.conf的配置文件，我们可以先查看一下该文件。

```
vim redis.conf
```

配置将Redis服务绑定到指定的IP地址和端口。

```
66 # IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES  
67 # JUST COMMENT THE FOLLOWING LINE.  
68 # -----  
69 bind 127.0.0.1  
70  
71 #-----
```

配置底层有多少个数据库。

```
183 # Set the number of databases. The default database is DB 0, you can select  
184 # a different one on a per-connection basis using SELECT <dbid> where  
185 # dbid is a number between 0 and 'databases'-1  
186 databases 16
```

配置Redis的持久化机制 - RDB。

```
237 # Compress string objects using LZF when dump .rdb databases?
238 # For default that's set to 'yes' as it's almost always a win.
239 # If you want to save some CPU in the saving child set it to 'no' but
240 # the dataset will likely be bigger if you have compressible values or keys.
241 rdbcompression yes
242
243 # Since version 5 of RDB a CRC64 checksum is placed at the end of the file.
244 # This makes the format more resistant to corruption but there is a performance
245 # hit to pay (around 10%) when saving and loading RDB files - so you can disable it
```

## 配置Redis的持久化机制 - AOF。

```
679 ##### APPEND ONLY MODE #####
680
681 # By default Redis asynchronously dumps the dataset on disk. This mode is
682 # good enough in many applications, but an issue with the Redis process or
683 # a power outage may result into a few minutes of writes lost (depending on
684 # the configured save points).
685 #
686 # The Append Only File is an alternative persistence mode that provides
687 # much better durability. For instance using the default data fsync policy
688 # (see later in the config file) Redis can lose just one second of writes in a
689 # dramatic event like a server power outage, or a single write if something
```

配置访问Redis服务器的验证口令。

```
494 ##### SECURITY #####
495 # Require clients to issue AUTH <PASSWORD> before processing any other
496 # commands. This is an important security feature which helps prevent
497 # denial of service attacks. It is highly recommended that you
498 # require clients to authenticate before they can issue any commands.
499 # If you don't want to use authentication at all, simply remove this
500 # directive.
```

配置Redis的主从复制，通过主从复制可以实现读写分离。

```
-----[REPLICA LIST]-----
```

配置慢查询。

上面这些内容就是Redis的基本配置，如果你对上面的内容感到困惑也没有关系，先把Redis用起来再回头去推敲这些内容就行了。如果想找一些参考书，《Redis开发与运维》是一本不错的入门读物，而《Redis实战》是不错的进阶读物。

### Redis的服务器和客户端

接下来启动Redis服务器，下面的方式将以默认的配置启动Redis服务。

```
redis-server
```

如果希望修改Redis的配置（如端口、认证口令、持久化方式等），可以通过下面两种方式。

方式一：通过参数指定认证口令和AOF持久化方式。

```
redis-server --requirepass 1qaz2wsx --appendonly yes
```

方式二：通过指定的配置文件来修改Redis的配置。

```
redis-server /root/redis-5.0.4/redis.conf
```

下面我们使用第一种方式来启动Redis并将其置于后台运行，将Redis产生的输出重定向到名为redis.log的文件中。

```
redis-server --requirepass 1qaz2wsx > redis.log &
```

可以通过ps或者netstat来检查Redis服务器是否启动成功。

```
ps -ef | grep redis-server  
netstat -nap | grep redis-server
```

接下来，我们尝试用Redis客户端去连接服务器。

```
redis-cli  
127.0.0.1:6379> auth 1qaz2wsx  
OK  
127.0.0.1:6379> ping  
PONG  
127.0.0.1:6379>
```

Redis有着非常丰富的数据类型，也有很多的命令来操作这些数据，具体的内容可以查看[Redis命令参考](#)，在这个网站上，除了Redis的命令参考，还有Redis的详细文档，其中包括了通知、事务、主从复制、持久化、哨兵、集群等内容。

说明：上面的插图来自付磊和张益军先生编著的《Redis开发与运维》一书。

```
127.0.0.1:6379> set username admin  
OK  
127.0.0.1:6379> get username  
"admin"  
127.0.0.1:6379> set password "123456" ex 300  
OK  
127.0.0.1:6379> get password  
"123456"  
127.0.0.1:6379> ttl username  
(integer) -1  
127.0.0.1:6379> ttl password  
(integer) 286
```

```
127.0.0.1:6379> hset stu1 name hao
(integer) 0
127.0.0.1:6379> hset stu1 age 38
(integer) 1
127.0.0.1:6379> hset stu1 gender male
(integer) 1
127.0.0.1:6379> hgetall stu1
1) "name"
2) "hao"
3) "age"
4) "38"
5) "gender"
6) "male"
127.0.0.1:6379> hvals stu1
1) "hao"
2) "38"
3) "male"
127.0.0.1:6379> hmset stu2 name wang age 18 gender female tel 13566778899
OK
127.0.0.1:6379> hgetall stu2
1) "name"
2) "wang"
3) "age"
4) "18"
5) "gender"
6) "female"
7) "tel"
8) "13566778899"
127.0.0.1:6379> lpush nums 1 2 3 4 5
(integer) 5
127.0.0.1:6379> lrange nums 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lpop nums
"5"
127.0.0.1:6379> lpop nums
"4"
127.0.0.1:6379> rpop nums
"1"
127.0.0.1:6379> rpop nums
"2"
127.0.0.1:6379> sadd fruits apple banana orange apple grape grape
(integer) 4
127.0.0.1:6379> scard fruits
(integer) 4
127.0.0.1:6379> smembers fruits
1) "grape"
2) "orange"
3) "banana"
4) "apple"
127.0.0.1:6379> sismember fruits apple
(integer) 1
```

```
127.0.0.1:6379> sismember fruits durian
(integer) 0
127.0.0.1:6379> sadd nums1 1 2 3 4 5
(integer) 5
127.0.0.1:6379> sadd nums2 2 4 6 8
(integer) 4
127.0.0.1:6379> sinter nums1 nums2
1) "2"
2) "4"
127.0.0.1:6379> sunion nums1 nums2
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "8"
127.0.0.1:6379> sdiff nums1 nums2
1) "1"
2) "3"
3) "5"
127.0.0.1:6379> zadd topsinger 5234 zhangxy 1978 chenyx 2235 zhoujl 3520 xuezq
(integer) 4
127.0.0.1:6379> zrange topsinger 0 -1 withscores
1) "chenyx"
2) "1978"
3) "zhoujl"
4) "2235"
5) "xuezq"
6) "3520"
7) "zhangxy"
8) "5234"
127.0.0.1:6379> zrevrange topsinger 0 -1
1) "zhangxy"
2) "xuezq"
3) "zhoujl"
4) "chenyx"
127.0.0.1:6379> geoadd pois 116.39738549206541 39.90862689286386 tiananmen 116.27172931
135172904494 yiheyuan 117.27766503308104 40.65332064313784 gubeishuizhen
(integer) 3
127.0.0.1:6379> geodist pois tiananmen gubeishuizhen km
"111.5333"
127.0.0.1:6379> geodist pois tiananmen yiheyuan km
"14.1230"
127.0.0.1:6379> georadius pois 116.86499108288572 40.40149669363615 50 km withdist
1) 1) "gubeishuizhen"
2) "44.7408"
```

可以使用pip安装redis模块。redis模块的核心是名为Redis的类，该类的对象代表一个Redis客户端，通过该客户端可以向Redis服务器发送命令并获取执行的结果。上面我们在Redis客户端中使用的命令基本上就是Redis对象可以接收的消息，所以如果了解了Redis的命令就可以在Python中玩转Redis。

```
pip3 install redis
python3

>>> import redis
>>> client = redis.Redis(host='1.2.3.4', port=6379, password='1qaz2wsx')
>>> client.set('username', 'admin')
True
>>> client.hset('student', 'name', 'hao')
1
>>> client.hset('student', 'age', 38)
1
>>> client.keys('*')
[b'username', b'student']
>>> client.get('username')
b'admin'
>>> client.hgetall('student')
{b'name': b'hao', b'age': b'38'}
```

## MongoDB概述

### MongoDB简介

MongoDB是2009年问世的一个面向文档的数据库管理系统，由C++语言编写，旨在为Web应用提供可扩展的高性能数据存储解决方案。虽然在划分类别的时候后，MongoDB被认为是NoSQL的产品，但是它更像一个介于关系数据库和非关系数据库之间的产品，在非关系数据库中它功能最丰富，最像关系数据库。

MongoDB将数据存储为一个文档，一个文档由一系列的“键值对”组成，其文档类似于JSON对象，但是MongoDB对JSON进行了二进制处理（能够更快的定位key和value），因此其文档的存储格式称为BSON。关于JSON和BSON的差别大家可以看看MongoDB官方网站的文章[《JSON and BSON》](#)。

目前，MongoDB已经提供了对Windows、MacOS、Linux、Solaris等多个平台的支持，而且也提供了多种开发语言的驱动程序，Python当然是其中之一。

### MongoDB的安装和配置

可以从MongoDB的[官方下载链接](#)下载MongoDB，官方为Windows系统提供了一个Installer程序，而Linux和MacOS则提供了压缩文件。下面简单说一下Linux系统如何安装和配置MongoDB。

```
wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-amazon-3.6.5.tgz  
gunzip mongodb-linux-x86_64-amazon-3.6.5.tgz  
mkdir mongodb-3.6.5  
tar -xvf mongodb-linux-x86_64-amazon-3.6.5.tar --strip-components 1 -C mongodb-3.6.5/  
export PATH=$PATH:~/mongodb-3.6.5/bin  
mkdir -p /data/db  
mongod --bind_ip 172.18.61.250  
  
2018-06-03T18:03:28.232+0800 I CONTROL [initandlisten] MongoDB starting : pid=1163 po  
2018-06-03T18:03:28.232+0800 I CONTROL [initandlisten] db version v3.6.5  
2018-06-03T18:03:28.232+0800 I CONTROL [initandlisten] git version: a20ecd3e3a1741620  
2018-06-03T18:03:28.232+0800 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.0  
...  
2018-06-03T18:03:28.945+0800 I NETWORK [initandlisten] waiting for connections on por
```

说明：上面的操作中，`export`命令是设置PATH环境变量，这样可以在任意路径下执行`mongod`来启动MongoDB服务器。MongoDB默认保存数据的路径是`/data/db`目录，为此要提前创建该目录。此外，在使用`mongod`启动MongoDB服务器时，`--bind_ip`参数用来将服务绑定到指定的IP地址，也可以用`--port`参数来指定端口，默认端口为27017。

## MongoDB基本概念

我们通过与关系型数据库进行对照的方式来说明MongoDB中的一些概念。

SQL	MongoDB	解释 ( SQL/MongoDB )
database	database	数据库/数据库
table	collection	二维表/集合
row	document	记录(行)/文档
column	field	字段(列)/域
index	index	索引/索引
table joins	---	表连接/嵌套文档
primary key	primary key	主键/主键(_id字段)

## 通过Shell操作MongoDB

启动服务器后可以使用交互式环境跟服务器通信，如下所示。

```
mongo --host 172.18.61.250  
  
MongoDB shell version v3.6.5  
connecting to: mongodb://172.18.61.250:27017/
```

```
...  
>
```

## 1. 查看、创建和删除数据库。

```
> // 显示所有数据库  
> show dbs  
admin 0.000GB  
config 0.000GB  
local 0.000GB  
> // 创建并切换到school数据库  
> use school  
switched to db school  
> // 删除当前数据库  
> db.dropDatabase()  
{ "ok" : 1 }  
>
```

## 2. 创建、删除和查看集合。

```
> // 创建并切换到school数据库  
> use school  
switched to db school  
> // 创建colleges集合  
> db.createCollection('colleges')  
{ "ok" : 1 }  
> // 创建students集合  
> db.createCollection('students')  
{ "ok" : 1 }  
> // 查看所有集合  
> show collections  
colleges  
students  
> // 删除colleges集合  
> db.colleges.drop()  
true  
>
```

说明：在MongoDB中插入文档时如果集合不存在会自动创建集合，所以也可以按照下面的方式通过创建文档来创建集合。

## 3. 文档的CRUD操作。

```
> // 向students集合插入文档  
> db.students.insert({stuid: 1001, name: '骆昊', age: 38})  
WriteResult({ "nInserted" : 1 })  
> // 向students集合插入文档  
> db.students.save({stuid: 1002, name: '王大锤', tel: '13012345678', gender: '男'}  
WriteResult({ "nInserted" : 1 })
```

```
> // 查看所有文档
> db.students.find()
{
  "_id" : ObjectId("5b13c72e006ad854460ee70b"), "stuid" : 1001, "name" : "骆昊", "
  {"_id" : ObjectId("5b13c790006ad854460ee70c"), "stuid" : 1002, "name" : "王大锤",
> // 更新stuid为1001的文档
> db.students.update({stuid: 1001}, {'$set': {tel: '13566778899', gender: '男'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> // 插入或更新stuid为1003的文档
> db.students.update({stuid: 1003}, {'$set': {name: '白元芳', tel: '13022223333',
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("5b13c92dd185894d7283efab")
})
> // 查询所有文档
> db.students.find().pretty()
{
  "_id" : ObjectId("5b13c72e006ad854460ee70b"),
  "stuid" : 1001,
  "name" : "骆昊",
  "age" : 38,
  "gender" : "男",
  "tel" : "13566778899"
}
{
  "_id" : ObjectId("5b13c790006ad854460ee70c"),
  "stuid" : 1002,
  "name" : "王大锤",
  "tel" : "13012345678",
  "gender" : "男"
}
{
  "_id" : ObjectId("5b13c92dd185894d7283efab"),
  "stuid" : 1003,
  "gender" : "男",
  "name" : "白元芳",
  "tel" : "13022223333"
}
> // 查询stuid大于1001的文档
> db.students.find({stuid: {'$gt': 1001}}).pretty()
{
  "_id" : ObjectId("5b13c790006ad854460ee70c"),
  "stuid" : 1002,
  "name" : "王大锤",
  "tel" : "13012345678",
  "gender" : "男"
}
{
  "_id" : ObjectId("5b13c92dd185894d7283efab"),
  "stuid" : 1003,
  "gender" : "男",
  "name" : "白元芳",
  "tel" : "13022223333"
}
```

```

> // 查询stuid大于1001的文档只显示name和tel字段
> db.students.find({stuid: {'$gt': 1001}}, {_id: 0, name: 1, tel: 1}).pretty()
{
  "name" : "王大锤",
  "tel" : "13012345678"
}
{
  "name" : "白元芳",
  "tel" : "13022223333"
}
> // 查询name为“骆昊”或者tel为“13022223333”的文档
> db.students.find({'$or': [{name: '骆昊'}, {tel: '13022223333'}]}), {_id: 0, name: 1}
{
  "name" : "骆昊",
  "tel" : "13566778899"
}
{
  "name" : "白元芳",
  "tel" : "13022223333"
}
> // 查询学生文档跳过第1条文档只查1条文档
> db.students.find().skip(1).limit(1).pretty()
{
  "_id" : ObjectId("5b13c790006ad854460ee70c"),
  "stuid" : 1002,
  "name" : "王大锤",
  "tel" : "13012345678",
  "gender" : "男"
}
> // 对查询结果进行排序(1表示升序, -1表示降序)
> db.students.find({}, {_id: 0, stuid: 1, name: 1}).sort({stuid: -1})
{
  "stuid" : 1003,
  "name" : "白元芳"
}
{
  "stuid" : 1002,
  "name" : "王大锤"
}
{
  "stuid" : 1001,
  "name" : "骆昊"
}
> // 在指定的一个或多个字段上创建索引
> db.students.ensureIndex({name: 1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
>

```

使用MongoDB可以非常方便的配置数据复制，通过冗余数据来实现数据的高可用以及灾难恢复，也可以通过数据分片来应对数据量迅速增长的需求。关于MongoDB更多的操作可以查阅[官方文档](#)，同时推荐大家阅读Kristina Chodorow写的《[MongoDB权威指南](#)》。

#### ####在Python程序中操作MongoDB

可以通过pip安装pymongo来实现对MongoDB的操作。

```

pip3 install pymongo
python3

```

```

>>> from pymongo import MongoClient
>>> client = MongoClient('mongodb://120.77.222.217:27017')
>>> db = client.school
>>> for student in db.students.find():
...     print('学号:', student['stuid'])
...     print('姓名:', student['name'])
...     print('电话:', student['tel'])

```

```
...
学号: 1001.0
姓名: 骆昊
电话: 13566778899
学号: 1002.0
姓名: 王大锤
电话: 13012345678
学号: 1003.0
姓名: 白元芳
电话: 13022223333
>>> db.students.find().count()
3
>>> db.students.remove()
{'n': 3, 'ok': 1.0}
>>> db.students.find().count()
0
>>> coll = db.students
>>> from pymongo import ASCENDING
>>> coll.create_index([('name', ASCENDING)], unique=True)
'name_1'
>>> coll.insert_one({'stuid': int(1001), 'name': '骆昊', 'gender': True})
<pymongo.results.InsertOneResult object at 0x1050cc6c8>
>>> coll.insert_many([{'stuid': int(1002), 'name': '王大锤', 'gender': False}, {'stuid': int(1003), 'name': '白元芳', 'gender': True}])
<pymongo.results.InsertManyResult object at 0x1050cc8c8>
>>> for student in coll.find({'gender': True}):
...     print('学号:', student['stuid'])
...     print('姓名:', student['name'])
...     print('性别:', '男' if student['gender'] else '女')
...
学号: 1001
姓名: 骆昊
性别: 男
学号: 1003
姓名: 白元芳
性别: 男
>>>
```

关于PyMongo更多的知识可以通过它的[官方文档](#)进行了解。

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

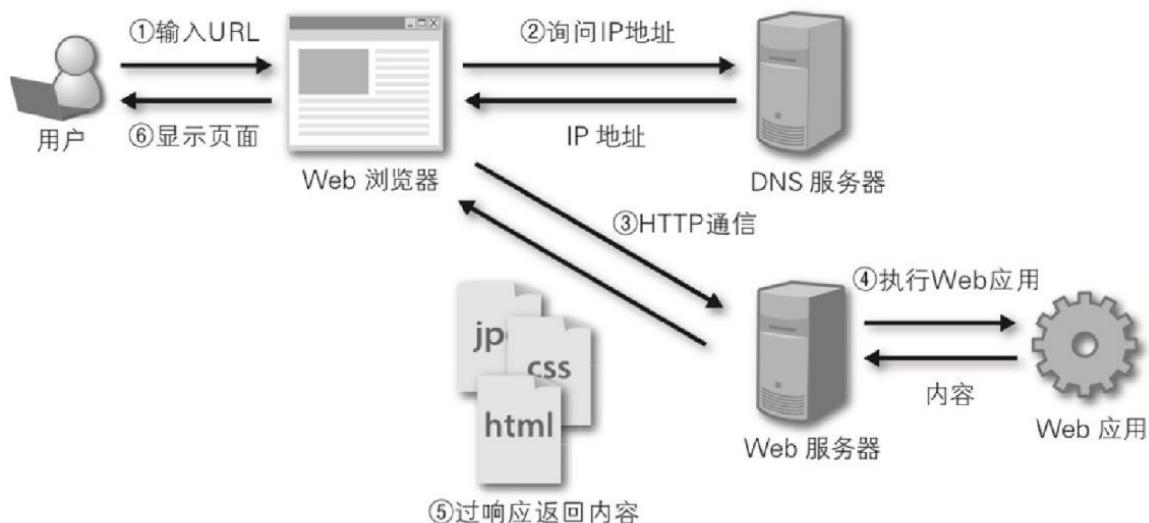
425 lines (306 sloc) 19.4 KB

## 快速上手

Web开发的早期阶段，开发者需要手动编写每个页面，例如一个新闻门户网站，每天都要修改它的HTML页面，随着网站规模和体量的增大，这种方式就变得极度糟糕。为了解决这个问题，开发人员想到了用外部程序来为Web服务器生成动态内容，也就是说HTML页面以及页面中的动态内容不再通过手动编写而是通过程序自动生成。最早的时候，这项技术被称为CGI（公共网关接口），当然随着时间的推移，CGI暴露出的问题也越来越多，例如大量重复的样板代码，总体性能较为低下等，因此在时代呼唤新英雄的背景下，PHP、ASP、JSP这类Web应用开发技术在上世纪90年代中后期如雨后春笋般涌现。通常我们说的Web应用是指通过浏览器来访问网络资源的应用程序，因为浏览器的普及性以及易用性，Web应用使用起来方便简单，免除了安装和更新应用程序带来的麻烦，而且也不用关心用户到底用的是什么操作系统，甚至不用区分是PC端还是移动端。

## Web应用机制和术语

下图向我们展示了Web应用的工作流程，其中涉及到的术语如下表所示。



说明：相信有经验的读者会发现，这张图中其实还少了很多东西，例如反向代理服务器、数据库服务器、防火墙等，而且图中的每个节点在实际项目部署时可能是一组节点组成的集群。当然，如果你对这些没有什么概念也不要紧，继续下去就行了，后面会给大家一一讲解的。

术语	解释
URL/URI	统一资源定位符/统一资源标识符，网络资源的唯一标识
域名	与Web服务器地址对应的一个易于记忆的字符串名字
DNS	域名解析服务，可以将域名转换成对应的IP地址
IP地址	网络上的主机的身份标识，通过IP地址可以区分不同的主机
HTTP	超文本传输协议，构建在TCP之上的应用级协议，万维网数据通信的基础
反向代理	代理客户端向服务器发出请求，然后将服务器返回的资源返回给客户端
Web服务器	接受HTTP请求，然后返回HTML文件、纯文本文件、图像等资源给请求者
Nginx	高性能的Web服务器，也可以用作反向代理， <a href="#">负载均衡</a> 和 <a href="#">HTTP缓存</a>

## HTTP协议

这里我们稍微费一些笔墨来谈谈上面提到的HTTP。HTTP（超文本传输协议）是构建于TCP（传输控制协议）之上应用级协议，它利用了TCP提供的可靠的传输服务实现了Web应用中的数据交换。按照维基百科上的介绍，设计HTTP最初的目的为了提供一种发布和接收HTML页面的方法，也就是说这个协议是浏览器和Web服务器之间传输的数据的载体。关于这个协议的详细信息以及目前的发展状况，大家可以阅读阮一峰老师的[《HTTP协议入门》](#)、[《互联网协议入门》](#)系列以及[《图解HTTPS协议》](#)进行了解。下图是我于2009年9月10日凌晨4点在四川省网络通信技术重点实验室用开源协议分析工具Ethereal（抓包工具WireShark的前身）截取的访问百度首页时的HTTP请求和响应的报文（协议数据），由于Ethereal截取的是经过网络适配器的数据，因此可以清晰的看到从物理链路层到应用层的协议数据。

HTTP请求（请求行+请求头+空行+[消息体]）：

```
Frame 4 (563 bytes on wire, 563 bytes captured)
Ethernet II, Src: Elitegro_f9:5d:82 (00:11:5b:f9:5d:82), Dst: Cisco_50:14:71 (00:1b:2a:50:14:71)
Internet Protocol, Src: 192.168.58.136 (192.168.58.136), Dst: 119.75.213.51 (119.75.213.51)
Transmission Control Protocol, Src Port: voispeed-port (3541), Dst Port: http (80), seq: 1, Ack: 1, Len: 509
Hypertext Transfer Protocol
    GET / HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Language: zh-cn\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727)\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: www.baidu.com\r\n
    Connection: Keep-Alive\r\n
    [truncated] Cookie: BAIDUID=72675E110453F51BEAC13B6277CE022F:FG=1; BDLOFONT=0; BDUSS=VFJenJQbGZuS21EM1dja3Vv
\r\n
```

HTTP响应（响应行+响应头+空行+消息体）：

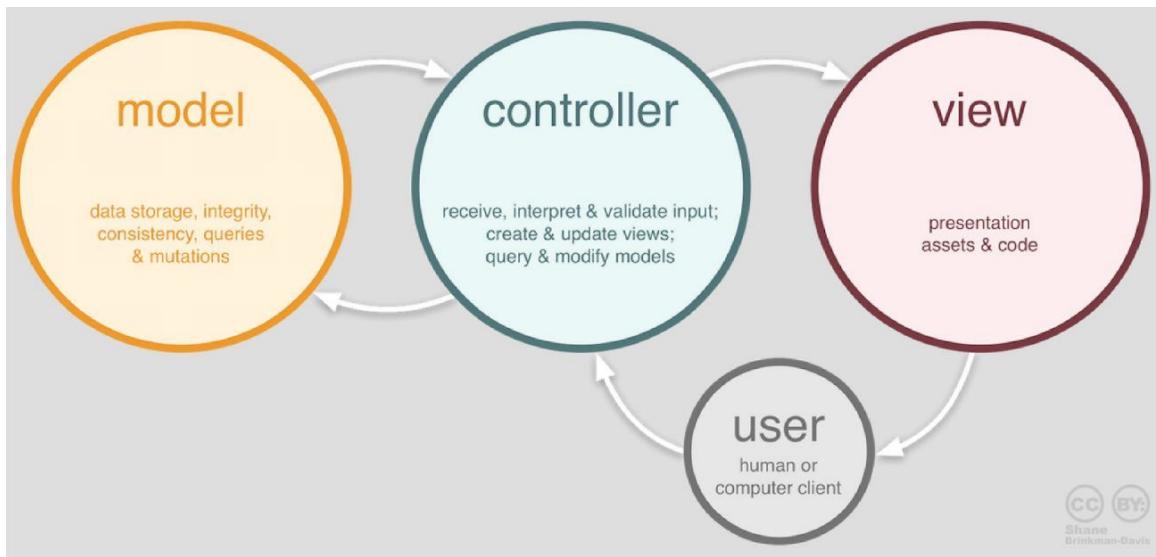
```
Frame 7 (668 bytes on wire, 668 bytes captured)
Ethernet II, Src: Cisco_50:14:71 (00:1b:2a:50:14:71), Dst: Elitegro_f9:5d:82 (00:11:5b:f9:5d:82)
Internet Protocol, Src: 119.75.213.51 (119.75.213.51), Dst: 192.168.58.136 (192.168.58.136)
Transmission Control Protocol, Src Port: http (80), Dst Port: voispeed-port (3541), seq: 1421, Ack: 510, Len: 510
[Reassembled TCP Segments (2034 bytes): #6(1420), #7(614)]
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Thu, 10 Sep 2009 04:02:47 GMT\r\n
    Server: BWS/1.0\r\n
    Content-Length: 1826\r\n
    Content-Type: text/html\r\n
    Cache-Control: private\r\n
    Expires: Thu, 10 Sep 2009 04:02:47 GMT\r\n
    Content-Encoding: gzip\r\n
\r\n
    Content-encoded entity body (gzip): 1826 bytes -> 3719 bytes
Line-based text data: text/html
```

说明：但愿这两张如同泛黄的照片般的截图能帮助你了解HTTP到底是什么样子的。

## Django概述

Python的Web框架有上百个，比它的关键字还要多。所谓Web框架，就是用于开发Web服务器端应用的基础设施（通常指封装好的模块和一系列的工具）。事实上，即便没有Web框架，我们仍然可以通过socket或CGI来开发Web服务器端应用，但是这样做的成本和代价在实际开发中通常是不能接受的。通过Web框架，我们可以化繁为简，同时降低创建、更新、扩展应用程序的工作量。Python的Web框架中比较有名的有：Flask、Django、Tornado、Sanic、Pyramid、Bottle、Web2py、web.py等。

在基于Python的Web框架中，Django是所有重量级选手中最有代表性的一位，开发者可以基于Django快速的开发可靠的Web应用程序，因为它减少了Web开发中不必要的开销，对常用的设计和开发模式进行了封装，并对MVC架构提供了支持（MTV）。许多成功的网站和App都是基于Django框架构建的，国内比较有代表性的网站包括：知乎、豆瓣网、果壳网、搜狐闪电邮箱、101围棋网、海报时尚网、背书吧、堆糖、手机搜狐网、咕咚、爱福窝、果库等。



Django诞生于2003年，它是一个在真正的应用中成长起来的项目，由劳伦斯出版集团旗下在线新闻网站的内容管理系统（CMS）研发团队编写（主要是Adrian Holovaty和Simon Willison），以比利时的吉普赛爵士吉他手Django Reinhardt来命名，在2005年夏天作为开源框架发布。使用Django能用很短的时间构建出功能完备的网站，因为它代替程序员完成了所有乏味和重复的劳动，剩下真正有意义的核心业务给程序员，这一点就是对DRY（Don't Repeat Yourself）理念的最好践行。

## 快速上手

### 准备工作

1. 检查Python环境：Django 1.11需要Python 2.7或Python 3.4以上的版本；Django 2.0需要Python 3.4以上的版本；Django 2.1需要Python 3.5以上的版本。

```
$ python3 --version
```

```
$ python3
>>> import sys
>>> sys.version
>>> sys.version_info
```

2. 创建项目文件夹并切换到该目录，例如我们要实例一个OA（办公自动化）项目。

```
$ mkdir oa
$ cd oa
```

3. 创建并激活虚拟环境。

```
$ python3 -m venv venv
$ source venv/bin/activate
```

说明：上面使用了Python自带的venv模块完成了虚拟环境的创建，当然也可以使用其他的工具，例如：virtualenv或pipenv等。要激活虚拟环境，在Windows系统下是通过"venv/Scripts/activate"执行批处理文件来实现。

#### 4. 更新包管理工具pip。

```
(venv)$ pip install -U pip
```

或

```
(venv)$ python -m pip install -U pip
```

注意：请注意终端提示符发生的变化，前面的(venv)说明我们已经进入虚拟环境，而虚拟环境下的python和pip已经是Python 3的解释器和包管理工具了。

#### 5. 安装Django。

```
(venv)$ pip install django
```

或指定版本号来安装对应的Django的版本。

```
(venv)$ pip install django==1.11
```

#### 6. 检查Django的版本。

```
(venv)$ python -m django --version  
(venv)$ django-admin --version
```

或

```
(venv)$ python  
>>> import django  
>>> django.get_version()
```

当然，也可以通过pip来查看安装的依赖库及其版本，如：

```
(venv)$ pip freeze  
(venv)$ pip list
```

下图展示了Django版本和Python版本的对应关系，如果在安装时没有指定版本号，将自动选择最新的版本（在写作这段内容时，最新的版本是2.0；目前最新的版本已经更新到2.2）。

Django版本	Python版本
1.8	2.7、3.2、3.3、3.4、3.5
1.9、1.10	2.7、3.4、3.5
1.11	2.7、3.4、3.5、3.6、3.7
2.0	3.4、3.5、3.6、3.7
2.1、2.2	3.5、3.6、3.7

#### 7. 使用 django-admin 创建项目，项目命名为oa。

```
(venv)$ django-admin startproject oa .
```

注意：上面的命令最后的那个点，它表示在当前路径下创建项目。

执行上面的命令后看看生成的文件和文件夹，它们的作用如下所示：

- manage.py：一个让你用各种方式管理 Django 项目的命令行工具。
- oa/\_\_init\_\_.py：一个空文件，告诉 Python 这个目录应该被认为是一个 Python 包。
- oa/settings.py：Django 项目的配置文件。
- oa/urls.py：Django 项目的 URL 声明，就像你网站的“目录”。
- oa/wsgi.py：作为你的项目的运行在 WSGI 兼容的Web服务器上的入口。

#### 8. 启动服务器运行项目。

```
(venv)$ python manage.py runserver
```

在浏览器中输入<http://127.0.0.1:8000>访问我们的服务器，效果如下图所示。



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in  
your settings file and you have not configured any  
URLs.



[Django Documentation](#)

Topics, references, & how-to's



[Tutorial: A Polling App](#)

Get started with Django



[Django Community](#)

Connect, get help, or contribute

说明1：刚刚启动的是Django自带的用于开发和测试的服务器，它是一个用纯Python编写的轻量级Web服务器，但它并不是真正意义上的生产级别的服务器，千万不要将这个服务器用于和生产环境相关的任何地方。

说明2：用于开发的服务器在需要的情况下会对每一次的访问请求重新载入一遍Python代码。所以你不需要为了让修改的代码生效而频繁的重新启动服务器。然而，一些动作，比如添加新文件，将不会触发自动重新加载，这时你得自己手动重启服务器。

说明3：可以通过 `python manage.py help` 命令查看可用命令列表；在启动服务器时，也可以通过 `python manage.py runserver 1.2.3.4:5678` 来指定绑定的IP地址和端口。

说明4：可以通过Ctrl+C来终止服务器的运行。

9. 接下来我们修改项目的配置文件`settings.py`，Django是一个支持国际化和本地化的框架，因此刚才我们看到的默认首页也是支持国际化的，我们将默认语言修改为中文，时区设置为东八区。

```
(venv)$ vim oa/settings.py
```

```
# 此处省略上面的内容
```

```
# 设置语言代码
LANGUAGE_CODE = 'zh-hans'
# 设置时区
```

```
TIME_ZONE = 'Asia/Chongqing'
```

```
# 此处省略下面的内容
```

## 10. 刷新刚才的页面。

[django](#)

[查看 Django 2.0 的 release notes](#)



安装成功！祝贺！

您现在看见这个页面，因为您设置了 `DEBUG=True` 并且  
您还没有配置任何URLs。



[Django 文档](#)

主题, 参考和指南



[教程：投票应用](#)

开始使用 Django



[Django 社区](#)

联系, 获取帮助, 贡献代码

## 动态页面

1. 创建名为hrs（人力资源系统）的应用，一个Django项目可以包含一个或多个应用。

```
(venv)$ python manage.py startapp hrs
```

执行上面的命令会在当前路径下创建hrs目录，其目录结构如下所示：

- `__init__.py`：一个空文件，告诉 Python 这个目录应该被认为是一个 Python 包。
- `admin.py`：可以用来注册模型，用于在 Django 的管理界面管理模型。
- `apps.py`：当前应用的配置。
- `migrations`：存放与模型有关的数据库迁移信息。
  - `__init__.py`：一个空文件，告诉 Python 这个目录应该被认为是一个 Python 包。
- `models.py`：存放应用的数据模型，即实体类及其之间的关系（MVC/MVT中的 M）。

- o tests.py : 包含测试应用各项功能的测试类和测试函数。
- o views.py : 处理请求并返回响应的函数 ( MVC中的C , MVT中的V ) 。

2. 修改应用目录下的视图文件views.py。

```
(venv)$ vim hrs/views.py
```

```
from django.http import HttpResponse

def index(request):
    return HttpResponse('<h1>Hello, Django!</h1>')
```

3. 在应用目录创建一个urls.py文件并映射URL。

```
(venv)$ touch hrs/urls.py
(venv)$ vim hrs/urls.py
```

```
from django.urls import path

from hrs import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

说明：上面使用的 path 函数是Django 2.x中新添加的函数，除此之外还可以使用支持正则表达式的URL映射函数 re\_path 函数；Django 1.x中是用名为 url 函数来设定URL映射。

4. 切换到项目目录，修改该目录下的urls.py文件，对应用中设定的URL进行合并。

```
(venv) $ vim oa/urls.py
```

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('hrs/', include('hrs.urls')),
]
```

5. 重新运行项目，并打开浏览器中访问<http://localhost:8000/hr>。

```
(venv)$ python manage.py runserver
```

## 6. 修改views.py生成动态内容。

```
(venv)$ vim hrs/views.py
```

```
from io import StringIO

from django.http import HttpResponse

dept_list = [
    {'no': 10, 'name': '财务部', 'location': '北京'},
    {'no': 20, 'name': '研发部', 'location': '成都'},
    {'no': 30, 'name': '销售部', 'location': '上海'},
]

def index(request):
    output = StringIO()
    output.write('<html>\n')
    output.write('<head>\n')
    output.write('\t<meta charset="utf-8">\n')
    output.write('\t<title>首页</title>')
    output.write('</head>\n')
    output.write('<body>\n')
    output.write('\t<h1>部门信息</h1>\n')
    output.write('\t<hr>\n')
    output.write('\t<table>\n')
    output.write('\t\t<tr>\n')
    output.write('\t\t\t<th width=120>部门编号</th>\n')
    output.write('\t\t\t<th width=180>部门名称</th>\n')
    output.write('\t\t\t<th width=180>所在地</th>\n')
    output.write('\t\t</tr>\n')
    for dept in dept_list:
        output.write('\t\t<tr>\n')
        output.write(f'\t\t\t<td align=center>{dept["no"]}</td>\n')
        output.write(f'\t\t\t<td align=center>{dept["name"]}</td>\n')
        output.write(f'\t\t\t<td align=center>{dept["location"]}</td>\n')
        output.write('\t\t</tr>\n')
    output.write('\t</table>\n')
    output.write('</body>\n')
    output.write('</html>\n')
    return HttpResponse(output.getvalue())
```

## 7. 刷新页面查看程序的运行结果。

# 部门信息

---

部门编号	部门名称	所在地
10	财务部	北京
20	研发部	成都
30	销售部	上海

## 使用视图模板

上面通过拼接HTML代码的方式生成动态视图的做法在实际开发中是无能接受的，这一点大家一定能够想到。为了解决这个问题，我们可以提前准备一个模板页，所谓模板页就是一个带占位符的HTML页面，当我们将程序中获得的数据替换掉页面中的占位符时，一个动态页面就产生了。

我们可以用Django框架中template模块的Template类创建模板对象，通过模板对象的render方法实现对模板的渲染。所谓的渲染就是用数据替换掉模板页中的占位符，当然这里的渲染称为后端渲染，即在服务器端完成页面的渲染再输出到浏览器中，这种做法的主要坏处是当并发访问量较大时，服务器会承受较大的负担，所以今天有很多的Web应用都使用了前端渲染，即服务器只为浏览器提供所需的数据（通常是JSON格式），在浏览器中通过JavaScript获取这些数据并渲染到页面上，这些内容在后面为大家呈现。

Django框架通过shortcuts模块的快捷函数 render 简化了渲染模板的操作，具体的用法如下所示。

1. 先回到manage.py文件所在的目录创建名为templates文件夹。

```
(venv)$ mkdir templates
```

2. 创建模板页index.html。

```
(venv)$ touch templates/index.html  
(venv)$ vim templates/index.html
```

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>首页</title>  
</head>  
<body>  
    <h1>部门信息</h1>  
    <hr>  
    <table>
```

```

<tr>
<th>部门编号</th>
<th>部门名称</th>
<th>所在地</th>
</tr>
{% for dept in depts_list %}
<tr>
<td>{{ dept.no }}</td>
<td>{{ dept.name }}</td>
<td>{{ dept.location }}</td>
</tr>
{% endfor %}
</table>
</body>
</html>

```

在上面的模板页中我们使用了 `{{ greeting }}` 这样的模板占位符语法，也使用了 `{% for %}` 这样的模板指令，这些都是Django模板语言（DTL）的一部分。如果对此不熟悉并不要紧，我们会在后续的内容中进一步的讲解，而且我们刚才也说到了，还有更好的选择就是使用前端渲染，当然这是后话。

### 3. 回到应用目录，修改views.py文件。

```
(venv)$ vim hrs/views.py
```

```

from django.shortcuts import render

depts_list = [
    {'no': 10, 'name': '财务部', 'location': '北京'},
    {'no': 20, 'name': '研发部', 'location': '成都'},
    {'no': 30, 'name': '销售部', 'location': '上海'},
]

def index(request):
    return render(request, 'index.html', {'depts_list': depts_list})

```

到此为止，我们还没有办法让views.py中的 `render` 函数找到模板文件index.html，为此我们需要修改settings.py文件，配置模板文件所在的路径。

### 4. 切换到项目目录修改settings.py文件。

```
(venv)$ vim oa/settings.py
```

```
# 此处省略上面的内容
```

```
TEMPLATES = [
```

```
{  
    'BACKEND': 'django.template.backends.django.DjangoTemplates',  
    'DIRS': [os.path.join(BASE_DIR, 'templates')],  
    'APP_DIRS': True,  
    'OPTIONS': {  
        'context_processors': [  
            'django.template.context_processors.debug',  
            'django.template.context_processors.request',  
            'django.contrib.auth.context_processors.auth',  
            'django.contrib.messages.context_processors.messages',  
        ],  
    },  
},  
]  
  
# 此处省略下面的内容
```

## 5. 重新运行项目或直接刷新页面查看结果。

```
(venv)$ python manage.py runserver
```

## 总结

至此，我们已经利用Django框架完成了一个非常小的Web应用，虽然它并没有任何的实际价值，但是可以通过这个项目对Django框架有一个感性的认识。当然，实际开发中我们可以用PyCharm来创建项目，如果使用专业版的PyCharm，可以直接创建Django项目。使用PyCharm的好处在于编写代码时可以获得代码提示、错误修复、自动导入等功能，从而提升开发效率，但是专业版的PyCharm需要按年支付相应的费用，社区版的PyCharm中并未包含对Django框架直接的支持，但是我们仍然可以使用它来创建Django项目，只是在使用上没有专业版的方便。关于PyCharm的使用，可以参考[《玩转PyCharm》一文](#)。

此外，学习Django最好的资料肯定是它的[官方文档](#)，除此之外图灵社区出版的[《Django基础教程》](#)也是非常适合初学者的读物。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day41-55 / 02.深入模型.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



543 lines (399 sloc) 22.7 KB

## 深入模型

在上一个章节中，我们提到了Django是基于MVC架构的Web框架，MVC架构追求的是“模型”和“视图”的解耦合。所谓“模型”说得更直白一些就是数据，所以通常也被称作“数据模型”。在实际的项目中，数据模型通常通过数据库实现持久化操作，而关系型数据库在很长一段时间都是持久化的首选方案，下面我们以MySQL为例来说明如何使用关系型数据库来实现持久化操作。

### 配置关系型数据库MySQL

我们继续来完善上一个章节中的OA项目，首先从配置项目使用的数据库开始。

1. 修改项目的settings.py文件，首先将我们之前创建的应用hrs添加已安装的项目中，然后配置MySQL作为持久化方案。

```
(venv)$ cd oa/settings.py
```

```
# 此处省略上面的代码
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'hrs',
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'oa',
```

```
        'HOST': 'localhost',
        'PORT': 3306,
        'USER': 'root',
        'PASSWORD': '123456',
    }
}

# 此处省略下面的代码
```

在配置ENGINE属性时，常用的可选值包括：

- 'django.db.backends.sqlite3'：SQLite嵌入式数据库。
- 'django.db.backends.postgresql'：BSD许可证下发行的开源关系型数据库产品。
- 'django.db.backends.mysql'：转手多次目前属于甲骨文公司的经济高效的数据产品。
- 'django.db.backends.oracle'：甲骨文公司的关系型数据库旗舰产品。

其他的配置可以参考官方文档中[数据库配置](#)的部分。

NAME属性代表数据库的名称，如果使用SQLite它对应着一个文件，在这种情况下NAME的属性值应该是一个绝对路径；使用其他关系型数据库，则要配置对应的HOST（主机）、PORT（端口）、USER（用户名）、PASSWORD（口令）等属性。

## 2. 安装MySQL客户端工具，Python 3中使用PyMySQL，Python 2中用MySQLdb。

```
(venv)$ pip install pymysql
```

如果使用Python 3需要修改项目的`__init__.py`文件并加入如下所示的代码，这段代码的作用是将PyMySQL视为MySQLdb来使用，从而避免Django找不到连接MySQL的客户端工具而询问你：“Did you install mysqlclient?”（你安装了mysqlclient吗？）。

```
import pymysql
pymysql.install_as_MySQLdb()
```

## 3. 运行manage.py并指定migrate参数实现数据库迁移，为应用程序创建对应的数据表，当然在此之前需要先启动MySQL数据库服务器并创建名为oa的数据库，在MySQL中创建数据库的语句如下所示。

```
drop database if exists oa;
create database oa default charset utf8;
```

```
(venv)$ cd ..
(venv)$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

4. 可以看到，Django帮助我们创建了10张表，这些都是使用Django框架需要的东西，稍后我们就会用到这些表。除此之外，我们还应该为我们自己的应用创建数据模型。如果要在hrs应用中实现对部门和员工的管理，我们可以创建如下所示的数据模型。

```
(venv)$ vim hrs/models.py
```

```
from django.db import models

class Dept(models.Model):
    """部门类"""

    no = models.IntegerField(primary_key=True, db_column='dno', verbose_name='部门号')
    name = models.CharField(max_length=20, db_column='dname', verbose_name='部门名称')
    location = models.CharField(max_length=10, db_column='dloc', verbose_name='部门地址')

    class Meta:
        db_table = 'tb_dept'

class Emp(models.Model):
    """员工类"""

    no = models.IntegerField(primary_key=True, db_column='eno', verbose_name='员工号')
    name = models.CharField(max_length=20, db_column='ename', verbose_name='员工姓名')
    job = models.CharField(max_length=10, verbose_name='职位')
    # 自参照完整性多对一外键关联
    mgr = models.ForeignKey('self', on_delete=models.SET_NULL, null=True, blank=True)
    sal = models.DecimalField(max_digits=7, decimal_places=2, verbose_name='月薪')
```

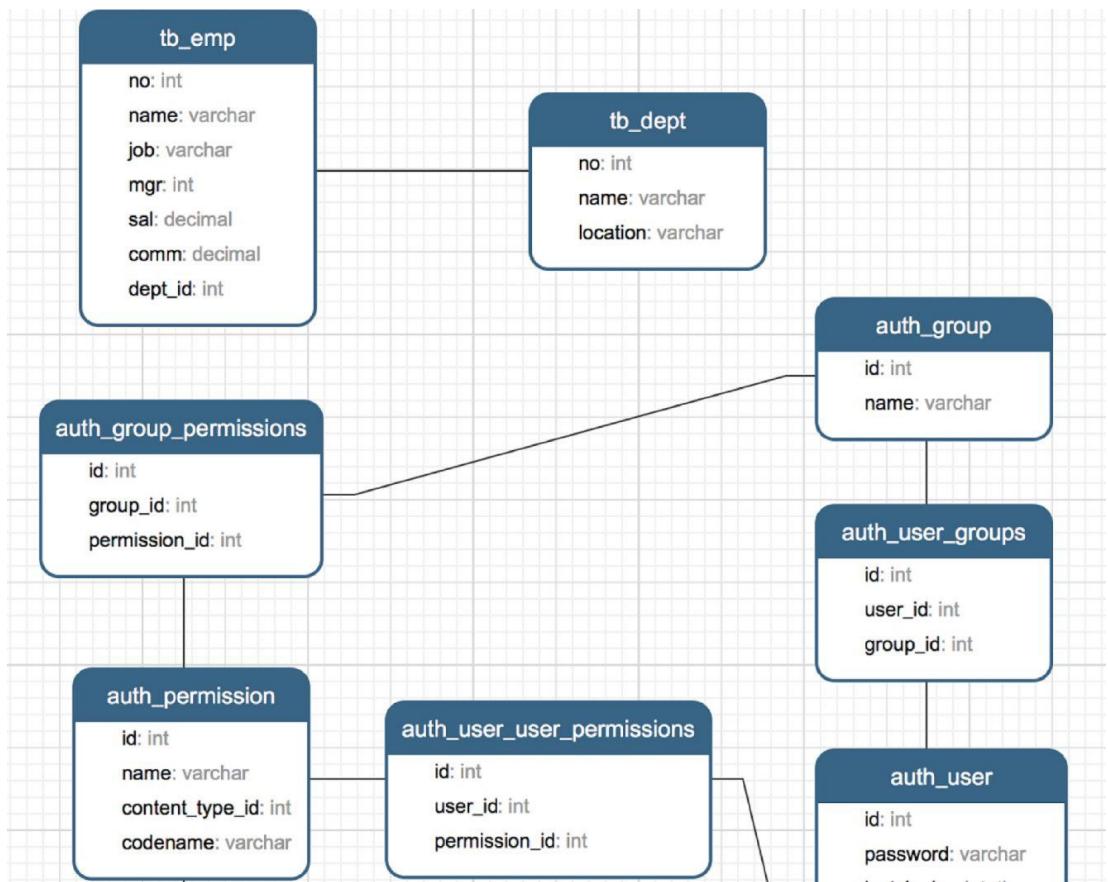
```
comm = models.DecimalField(max_digits=7, decimal_places=2, null=True, blank=True)
# 多对一外键关联
dept = models.ForeignKey(Dept, db_column='dno', on_delete=models.PROTECT, verbos
class Meta:
    db_table = 'tb_emp'
```

说明：上面定义模型时使用了字段类及其属性，其中IntegerField对应数据库中的integer类型，CharField对应数据库的varchar类型，DecimalField对应数据库的decimal类型，ForeignKey用来建立多对一外键关联。字段属性primary\_key用于设置主键，max\_length用来设置字段的最大长度，db\_column用来设置数据库中与字段对应的列，verbose\_name则设置了Django后台管理系统中该字段显示的名称。如果对这些东西感到很困惑也不要紧，文末提供了字段类、字段属性、元数据选项等设置的相关说明，不清楚的读者可以稍后查看对应的参考指南。

## 5. 通过模型创建数据表。

```
(venv)$ python manage.py makemigrations hrs
Migrations for 'hrs':
    hrs/migrations/0001_initial.py
        - Create model Dept
        - Create model Emp
(venv)$ python manage.py migrate
Operations to perform:
    Apply all migrations: admin, auth, contenttypes, hrs, sessions
Running migrations:
    Applying hrs.0001_initial... OK
```

执行完数据模型迁移操作之后，可以在通过图形化的MySQL客户端工具查看到E-R图（实体关系图）。



## 在后台管理模型

1. 创建超级管理员账号。

```
(venv)$ python manage.py createsuperuser
Username (leave blank to use 'hao'): jackfrued
Email address: jackfrued@126.com
Password:
Password (again):
Superuser created successfully.
```

2. 启动Web服务器，登录后台管理系统。

```
(venv)$ python manage.py runserver
```

访问<http://127.0.0.1:8000/admin>，会来到如下图所示的登录界面。



登录后进入管理员操作平台。

A screenshot of the Django admin dashboard. The top navigation bar is dark blue with the text "Django 管理" and a welcome message for the user "JACKFRUED". Below the navigation, the main content area has a light gray background. On the left, there's a sidebar titled "站点管理" (Site Management) with sections for "认证和授权" (Authentication and Authorization), "用户" (User) with "增加" (Add) and "修改" (Change) buttons, and "组" (Group) with "增加" (Add) and "修改" (Change) buttons. On the right, there's a "最近动作" (Recent Actions) sidebar with a section titled "我的动作" (My Actions) showing the message "无可用的" (None available).

至此我们还没有看到之前创建的模型类，需要在应用的admin.py文件中模型进行注册。

### 3. 注册模型类。

```
(venv)$ vim hrs/admin.py
```

```
from django.contrib import admin  
  
from hrs.models import Emp, Dept
```

```
admin.site.register(Dept)
admin.site.register(Emp)
```

注册模型类后，就可以在后台管理系统中看到它们。

The screenshot shows the Django Admin interface for the 'HRS' site. On the left, there are two sections: 'Depts' and 'Emps'. Each section has a '增加' (Add) button and a '修改' (Change) button. On the right, there is a sidebar titled '最近动作' (Recent Actions) which says '我的动作' (My Actions) and '无可用的' (No available actions).

#### 4. 对模型进行CRUD操作。

可以在管理员平台对模型进行C（新增）R（查看）U（更新）D（删除）操作，如下图所示。

添加新的部门。

The screenshot shows the '增加 dept' (Add Dept) form. It has three fields: '部门编号:' with value '30', '部门名称:' with value '销售1部', and '部门所在地:' with value '重庆'. At the bottom, there are three buttons: '保存并增加另一个' (Save and add another), '保存并继续编辑' (Save and continue edit), and '保存' (Save).

查看所有部门。

Django 管理

欢迎, JACKFRUED. 查看站点 / 修改密码 / 注销

首页 > Hrs > Depts

选择 dept 来修改

增加 DEPT +

动作  执行 3 个中 0 个被选

DEPT

Dept object (30)

Dept object (20)

Dept object (10)

3 depts

更新和删除部门。

Django 管理

欢迎, JACKFRUED. 查看站点 / 修改密码 / 注销

首页 > Hrs > Depts > Dept object (30)

修改 dept

历史

部门编号:

30

部门名称:

运维1部

部门所在地:

重庆

## 5. 注册模型管理类。

再次修改admin.py文件，通过注册模型管理类，可以在后台管理系统中更好的管理模型。

```
from django.contrib import admin

from hrs.models import Emp, Dept

class DeptAdmin(admin.ModelAdmin):

    list_display = ('no', 'name', 'location')
    ordering = ('no', )

class EmpAdmin(admin.ModelAdmin):
```

```
list_display = ('no', 'name', 'job', 'mgr', 'sal', 'comm', 'dept')
search_fields = ('name', 'job')
```

```
admin.site.register(Dept, DeptAdmin)
admin.site.register(Emp, EmpAdmin)
```

Django 管理

欢迎, JACKFRUED. 查看站点 / 修改密码 / 注销

首页 > Hrs > Depts

选择 dept 来修改 增加 DEPT +

动作	部门编号	部门名称	部门所在地
<input type="checkbox"/>	10	研发1部	成都
<input type="checkbox"/>	20	销售1部	重庆
<input type="checkbox"/>	30	运维1部	重庆

3 depts

Django 管理

欢迎, JACKFRUED. 查看站点 / 修改密码 / 注销

首页 > Hrs > Emps

选择 emp 来修改 增加 EMP +

动作	员工编号	员工姓名	职位	主管编号	月薪	补贴	所在部门
<input type="checkbox"/>	7800	张三丰	总裁	-	9000.00	1200.00	Dept object (10)
<input type="checkbox"/>	3250	张无忌	程序员	Emp object (2566)	4500.00	800.00	Dept object (10)

为了更好的查看模型数据，可以为Dept和Emp两个模型类添加 `__str__` 魔法方法。

```
from django.db import models
```

```
class Dept(models.Model):
    """部门类"""

    # 此处省略上面的代码
```

```

def __str__(self):
    return self.name

# 此处省略下面的代码


class Emp(models.Model):
    """员工类"""

    # 此处省略上面的代码

    mgr = models.ForeignKey('self', on_delete=models.SET_NULL, null=True, blank=True)

    # 此处省略下面的代码

    # 此处省略上面的代码

    def __str__(self):
        return self.name

    # 此处省略下面的代码

```

修改代码后刷新查看Emp模型的页面，效果如下图所示。

动作	员工编号	员工姓名	职位	直接主管	月薪	补贴	所在部门
<input type="checkbox"/>	7800	张三丰	总裁	-	9000.00	1200.00	研发1部
<input type="checkbox"/>	3344	黄蓉	销售主管	张三丰	3500.00	2000.00	销售1部
<input type="checkbox"/>	3250	张无忌	程序员	乔峰	4500.00	800.00	研发1部
<input type="checkbox"/>	2566	乔峰	架构师	张三丰	7500.00	1500.00	研发1部

4 emps

## 使用ORM完成模型的CRUD操作

在了解了Django提供的模型管理平台之后，我们来看看如何从代码层面完成对模型的CRUD ( Create / Read / Update / Delete ) 操作。我们可以通过manage.py开启Shell交互式环境，然后使用Django内置的ORM框架对模型进行CRUD操作。

```
(venv)$ python manage.py shell
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

## 新增

```
>>> from hrs.models import Dept, Emp
>>> dept = Dept(40, '研发2部', '深圳')
>>> dept.save()
```

## 更新

```
>>> dept.name = '研发3部'
>>> dept.save()
```

## 查询

查询所有对象。

```
>>> Dept.objects.all()
<QuerySet [<Dept: 研发1部>, <Dept: 销售1部>, <Dept: 运维1部>, <Dept: 研发3部>]>
```

过滤数据。

```
>>> Dept.objects.filter(name='研发3部') # 查询部门名称为“研发3部”的部门
<QuerySet [<Dept: 研发3部>]>
>>>
>>> Dept.objects.filter(name__contains='研发') # 查询部门名称包含“研发”的部门(模糊查询
<QuerySet [<Dept: 研发1部>, <Dept: 研发3部>]>
>>>
>>> Dept.objects.filter(no__gt=10).filter(no__lt=40) # 查询部门编号大于10小于40的部门
<QuerySet [<Dept: 销售1部>, <Dept: 运维1部>]>
>>>
>>> Dept.objects.filter(no__range=(10, 30)) # 查询部门编号在10到30之间的部门
<QuerySet [<Dept: 研发1部>, <Dept: 销售1部>, <Dept: 运维1部>]>
```

查询单个对象。

```
>>> Dept.objects.get(pk=10)
<Dept: 研发1部>
>>>
```

```
>>> Dept.objects.get(no=20)
<Dept: 销售1部>
>>>
>>> Dept.objects.get(no_exact=30)
<Dept: 运维1部>
>>>
>>> Dept.objects.filter(no=10).first()
<Dept: 研发1部>
```

排序数据。

```
>>> Dept.objects.order_by('no') # 查询所有部门按部门编号升序排列
<QuerySet [<Dept: 研发1部>, <Dept: 销售1部>, <Dept: 运维1部>, <Dept: 研发3部>]>
>>>
>>> Dept.objects.order_by('-no') # 查询所有部门按部门编号降序排列
<QuerySet [<Dept: 研发3部>, <Dept: 运维1部>, <Dept: 销售1部>, <Dept: 研发1部>]>
```

切片数据。

```
>>> Dept.objects.order_by('no')[0:2] # 按部门编号排序查询1~2部门
<QuerySet [<Dept: 研发1部>, <Dept: 销售1部>]>
>>>
>>> Dept.objects.order_by('no')[2:4] # 按部门编号排序查询3~4部门
<QuerySet [<Dept: 运维1部>, <Dept: 研发3部>]>
```

高级查询。

```
>>> Emp.objects.filter(dept_no=10) # 根据部门编号查询该部门的员工
<QuerySet [<Emp: 乔峰>, <Emp: 张无忌>, <Emp: 张三丰>]>
>>>
>>> Emp.objects.filter(dept_name_contains='销售') # 查询名字包含“销售”的部门的员工
<QuerySet [<Emp: 黄蓉>]>
>>>
>>> Dept.objects.get(pk=10).emp_set.all() # 通过部门反查部门所有的员工
<QuerySet [<Emp: 乔峰>, <Emp: 张无忌>, <Emp: 张三丰>]>
```

说明1：由于员工与部门之间存在多对一外键关联，所以也能通过部门反向查询该部门的员工（从一对多关系中“一”的一方查询“多”的一方），反向查询属性默认的名字是类名小写\_set（如上面例子中的 emp\_set），当然也可以在创建模型时通过 ForeignKey 的 related\_name 属性指定反向查询属性的名字。如果不希望执行反向查询可以将 related\_name 属性设置为 '+' 或以 '+' 开头的字符串。

说明2：查询多个对象的时候返回的是QuerySet对象，QuerySet使用了惰性查询，即在创建QuerySet对象的过程中不涉及任何数据库活动，等真正用到对象时（求值 QuerySet）才向数据库发送SQL语句并获取对应的结果，这一点在实际开发中需要引起注意！

说明3：可以在QuerySet上使用 `update()` 方法一次更新多个对象。

## 删除

```
>>> Dept.objects.get(pk=40).delete()
(1, {'hrs.Dept': 1})
```

## Django模型最佳实践

1. 正确的为模型和关系字段命名。
2. 设置适当的 `related_name` 属性。
3. 用 `OneToOneField` 代替 `ForeignKeyField(unique=True)`。
4. 通过“迁移操作”（`migrate`）来添加模型。
5. 用NoSQL来应对需要降低范式级别的场景。
6. 如果布尔类型可以为空要使用 `NullBooleanField`。
7. 在模型中放置业务逻辑。
8. 用 `<ModelName>.DoesNotExist` 取代 `ObjectDoesNotExist`。
9. 在数据库中不要出现无效数据。
10. 不要对 `QuerySet` 调用 `len()` 函数。
11. 将 `QuerySet` 的 `exists()` 方法的返回值用于 `if` 条件。
12. 用 `DecimalField` 来存储货币相关数据而不是 `FloatField`。
13. 定义 `__str__` 方法。
14. 不要将数据文件放在同一个目录中。

说明：以上内容来自于STEELKIWI网站的*Best Practice working with Django models in Python*，有兴趣的小伙伴可以阅读原文。

## 模型定义参考

### 字段

#### 对字段名称的限制

- 字段名不能是Python的保留字，否则会导致语法错误
- 字段名不能有多个连续下划线，否则影响ORM查询操作

#### Django模型字段类

字段类	说明
AutoField	自增ID字段
BigIntegerField	64位有符号整数

字段类	说明
BinaryField	存储二进制数据的字段，对应Python的bytes类型
BooleanField	存储True或False
CharField	长度较小的字符串
DateField	存储日期，有auto_now和auto_now_add属性
DateTimeField	存储日期和时间，两个附加属性同上
DecimalField	存储固定精度小数，有max_digits（有效位数）和decimal_places（小数点后面）两个必要的参数
DurationField	存储时间跨度
EmailField	与CharField相同，可以用EmailValidator验证
FileField	文件上传字段
FloatField	存储浮点数
ImageField	其他同FileFiled，要验证上传的是不是有效图像
IntegerField	存储32位有符号整数。
GenericIPAddressField	存储IPv4或IPv6地址
NullBooleanField	存储True、False或null值
PositiveIntegerField	存储无符号整数（只能存储正数）
SlugField	存储slug（简短标注）
SmallIntegerField	存储16位有符号整数
TextField	存储数据量较大的文本
TimeField	存储时间
URLField	存储URL的CharField
UUIDField	存储全局唯一标识符

## 字段属性

### 通用字段属性

选项	说明
null	数据库中对应的字段是否允许为NULL，默认为False

选项	说明
blank	后台模型管理验证数据时，是否允许为NULL，默认为False
choices	设定字段的选项，各元组中的第一个值是设置在模型上的值，第二值是人类可读的值
db_column	字段对应到数据库表中的列名，未指定时直接使用字段的名称
db_index	设置为True时将在该字段创建索引
db_tablespace	为有索引的字段设置使用的表空间，默认为DEFAULT_INDEX_TABLESPACE
default	字段的默认值
editable	字段在后台模型管理或ModelForm中是否显示，默认为True
error_messages	设定字段抛出异常时的默认消息的字典，其中的键包括null、blank、invalid、invalid_choice、unique和unique_for_date
help_text	表单小组件旁边显示的额外的帮助文本。
primary_key	将字段指定为模型的主键，未指定时会自动添加AutoField用于主键，只读。
unique	设置为True时，表中字段的值必须是唯一的
verbose_name	字段在后台模型管理显示的名称，未指定时使用字段的名称

### ForeignKey属性

1. limit\_choices\_to : 值是一个Q对象或返回一个Q对象，用于限制后台显示哪些对象。
2. related\_name : 用于获取关联对象的关联管理器对象（反向查询），如果不允许反向，该属性应该被设置为 '+'，或者以 '+' 结尾。
3. to\_field : 指定关联的字段，默认关联对象的主键字段。
4. db\_constraint : 是否为外键创建约束，默认值为True。
5. on\_delete : 外键关联的对象被删除时对应的动作，可取的值包括django.db.models中定义的：
  - CASCADE : 级联删除。
  - PROTECT : 抛出ProtectedError异常，阻止删除引用的对象。
  - SET\_NULL : 把外键设置为null，当null属性被设置为True时才能这么做。
  - SET\_DEFAULT : 把外键设置为默认值，提供了默认值才能这么做。

### ManyToManyField属性

1. symmetrical : 是否建立对称的多对多关系。
2. through : 指定维持多对多关系的中间表的Django模型。

3. `throughfields` : 定义了中间模型时可以指定建立多对多关系的字段。

4. `db_table` : 指定维持多对多关系的中间表的表名。

## 模型元数据选项

选项	说明
<code>abstract</code>	设置为True时模型是抽象父类
<code>app_label</code>	如果定义模型的应用不在INSTALLED_APPS中可以用该属性指定
<code>db_table</code>	模型使用的数据表名称
<code>db_tablespace</code>	模型使用的数据表空间
<code>default_related_name</code>	关联对象回指这个模型时默认使用的名称，默认为 <code>&lt;model_name&gt;_set</code>
<code>get_latest_by</code>	模型中可排序字段的名称。
<code>managed</code>	设置为True时，Django在迁移中创建数据表并在执行flush管理命令时把表移除
<code>order_with_respect_to</code>	标记对象为可排序的
<code>ordering</code>	对象的默认排序
<code>permissions</code>	创建对象时写入权限表的额外权限
<code>default_permissions</code>	默认为 ('add', 'change', 'delete')
<code>unique_together</code>	设定组合在一起时必须独一无二的字段名
<code>index_together</code>	设定一起建立索引的多个字段名
<code>verbose_name</code>	为对象设定人类可读的名称
<code>verbose_name_plural</code>	设定对象的复数名称

## 查询参考

按字段查找可以用的条件：

1. `exact / iexact` : 精确匹配/忽略大小写的精确匹配查询

2. `contains / icontains / startswith / istartswith / endswith / iendswith` : 基于 like 的模糊查询

3. `in` : 集合运算

4. `gt / gte / lt / lte` : 大于/大于等于/小于/小于等于关系运算

5. `range` : 指定范围查询 ( SQL中的 between...and... )

- 6. year / month / day / week\_day / hour / minute / second : 查询时间日期
- 7. isnull : 查询空值 ( True ) 或非空值 ( False )
- 8. search : 基于全文索引的全文检索
- 9. regex / iregex : 基于正则表达式的模糊匹配查询

Q对象（用于执行复杂查询）的使用：

```
>>> from django.db.models import Q
>>> Emp.objects.filter(
...     Q(name__startswith='张'),
...     Q(sal__gte=5000) | Q(comm__gte=1000)
... ) # 查询名字以“张”开头且工资大于等于5000或补贴大于等于1000的员工
<QuerySet [<Emp: 张三丰>]>
```

Branch: master ▾

[Find file](#)[Copy path](#)

## Python-100-Days / Day41-55 / 03.静态资源和Ajax请求.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

304 lines (244 sloc) 16.6 KB

## 静态资源和Ajax请求

基于前面两个章节讲解的知识，我们已经可以使用Django框架来实现Web应用的开发了。接下来我们就尝试实现一个投票应用，具体的需求是用户进入应用首先查看到“学科介绍”页面，该页面显示了一个学校所开设的所有学科；通过点击某个学科，可以进入“老师介绍”页面，该页面展示了该学科所有老师的详细情况，可以在该页面上给老师点击“好评”或“差评”，但是会先跳转到“登录页”要求用户登录，登录成功才能投票；对于未注册的用户，可以在“登录页”点击“新用户注册”进入“注册页”完成用户注册，注册成功后会跳转到“登录页”，注册失败会获得相应的提示信息。

## 准备工作

由于之前已经详细的讲解了如何创建Django项目以及项目的相关配置，因此我们略过这部分内容，唯一需要说明的是，从上面对投票应用需求的描述中我们可以分析出三个业务实体：学科、老师和用户。学科和老师之间通常是一对多关联关系（一个学科有多个老师，一个老师通常只属于一个学科），用户因为要给老师投票，所以跟老师之间是多对多关联关系（一个用户可以给多个老师投票，一个老师也可以收到多个用户的投票）。首先修改应用下的models.py文件来定义数据模型，先给出学科和老师的模型。

```
from django.db import models

class Subject(models.Model):
    """学科"""
    no = models.AutoField(primary_key=True, verbose_name='编号')
    name = models.CharField(max_length=31, verbose_name='名称')
    intro = models.CharField(max_length=511, verbose_name='介绍')

    def __str__(self):
        return self.name

    class Meta:
        db_table = 'tb_subject'
```

```

verbose_name_plural = '学科'

class Teacher(models.Model):
    """老师"""
    no = models.AutoField(primary_key=True, verbose_name='编号')
    name = models.CharField(max_length=15, verbose_name='姓名')
    gender = models.BooleanField(default=True, choices=((True, '男'), (False, '女')))
    birth = models.DateField(null=True, verbose_name='出生日期')
    intro = models.CharField(max_length=511, default='', verbose_name='')
    good_count = models.IntegerField(default=0, verbose_name='好评数')
    bad_count = models.IntegerField(default=0, verbose_name='差评数')
    photo = models.CharField(max_length=255, verbose_name='照片')
    subject = models.ForeignKey(to=Subject, on_delete=models.PROTECT, db_column='sno',

    def __str__(self):
        return self.name

    class Meta:
        db_table = 'tb_teacher'
        verbose_name_plural = '老师'

```

模型定义完成后，可以通过“生成迁移”和“执行迁移”来完成关系型数据库中二维表的创建，当然这需要提前启动数据库服务器并创建好对应的数据库，同时我们在项目中已经安装了PyMySQL而且完成了相应的配置，这些内容此处不再赘述。

```
(venv)$ python manage.py makemigrations vote
...
(venv)$ python manage.py migrate
...
```

**注意：**为了给vote应用生成迁移，需要先修改Django项目的配置文件settings.py，在INSTALLED\_APPS中添加vote应用。

完成模型迁移之后，我们可以通过下面的SQL语句来添加学科和老师测试的数据。

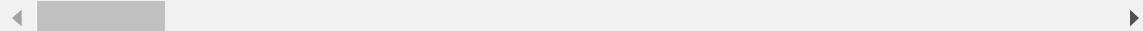
```

INSERT INTO `tb_subject` (`no`, `name`, `intro`)
VALUES
(1, 'Python全栈+人工智能', 'Python是一种面向对象的解释型计算机程序设计语言，由荷兰人Gu
(2, 'JavaEE+分布式服务', 'Java是一门面向对象编程语言，不仅吸收了C++语言的各种优点，还拥
(3, 'HTML5大前端', 'HTML5 将成为 HTML、XHTML 以及 HTML DOM 的新标准。'),
(4, '全栈软件测试', '在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其实
(5, '全链路UI/UE', '全链路要求设计师关注整个业务链中的每一个环节，将设计的价值融入每一

INSERT INTO `tb_teacher` (`no`, `name`, `gender`, `birth`, `intro`, `good_count`, `bad_count
VALUES
(1, '骆昊', 1, '1980-11-28', '10年以上软硬件产品设计、研发、架构和管理经验，2003年毕业
(2, '王海飞', 1, '1993-05-24', '5年以上Python开发经验，先后参与了O2O商城、CRM系统、CMS
(3, '余婷', 0, '1992-03-12', '5年以上移动互联网项目开发经验和教学经验，曾担任上市游戏公

```

```
(4, '肖世荣', 1, '1977-07-02', '10年以上互联网和移动互联网产品设计、研发、技术架构和项  
(5, '张无忌', 1, '1987-07-07', '出生起便在冰火岛过着原始生活，踏入中土后因中玄冥神掌命  
(6, '韦一笑', 1, '1975-12-15', '外号“青翼蝠王”，为明教四大护教法王之一。身披青条子白·
```



当然也可以直接使用Django提供的后台管理应用来添加学科和老师信息，这需要先注册模型类和模型管理类。

```
from django.contrib import admin
from django.contrib.admin import ModelAdmin

from vote.models import Teacher, Subject

class SubjectModelAdmin(ModelAdmin):
    """学科模型管理"""
    list_display = ('no', 'name')
    ordering = ('no', )

class TeacherModelAdmin(ModelAdmin):
    """老师模型管理"""
    list_display = ('no', 'name', 'gender', 'birth', 'good_count', 'bad_count', 'subject')
    ordering = ('no', )
    search_fields = ('name', )

admin.site.register(Subject, SubjectModelAdmin)
admin.site.register(Teacher, TeacherModelAdmin)
```



接下来，我们就可以修改views.py文件，通过编写视图函数先实现“学科介绍”页面。

```
def show_subjects(request):
    """查看所有学科"""
    subjects = Subject.objects.all()
    return render(request, 'subject.html', {'subjects': subjects})
```

至此，我们还需要一个模板页，模板的配置以及模板页中模板语言的用法在之前已经进行过简要的介绍，如果不熟悉可以看看下面的代码，相信这并不是一件困难的事情。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>学科信息</title>
    <style>/* 此处略去了层叠样式表的选择器 */</style>
</head>
<body>
```

```

<h1>所有学科</h1>
<hr>
<div id="container">
    {% for subject in subjects %}
        <dl>
            <dt>
                <a href="/teachers?sno={{ subject.no }}">
                    {{ subject.name }}
                </a>
            </dt>
            <dd>{{ subject.intro }}</dd>
        </dl>
    {% endfor %}
</div>
</body>
</html>

```

在上面的模板中，我们为每个学科添加了一个超链接，点击超链接可以查看该学科的讲师信息，为此需要再编写一个视图函数来处理查看指定学科老师信息。

```

def show_teachers(request):
    """查看指定学科的老师"""
    try:
        sno = int(request.GET['sno'])
        subject = Subject.objects.get(no=sno)
        teachers = Teacher.objects.filter(subject_no=sno)
        context = {'subject': subject, 'teachers': teachers}
        return render(request, 'teacher.html', context)
    except (KeyError, ValueError, Subject.DoesNotExist):
        return redirect('/')

```

显示老师信息的模板页。

```

<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>老师信息</title>
    <style>/* 此处略去了层叠样式表的选择器 */</style>
</head>
<body>
    <h1>{{ subject.name }}的老师信息</h1>
    <hr>
    {% if teachers %}
        <div id="container">
            {% for teacher in teachers %}
                <div class="teacher">
                    <div class="photo">
                        
                    </div>

```

```

        <div class="info">
            <div>
                <span><strong>姓名: {{ teacher.name }}</strong></span>
                <span>性别: {{ teacher.gender | yesno:'男,女' }}</span>
                <span>出生日期: {{ teacher.birth }}</span>
            </div>
            <div class="intro">{{ teacher.intro }}</div>
            <div class="comment">
                <a href="">好评 ({{ teacher.good_count }}) </a>
                <a href="">差评 ({{ teacher.bad_count }}) </a>
            </div>
        </div>
    </div>
    {% endfor %}
</div>
{% else %}
<h2>暂时没有该学科的老师信息</h2>
{% endif %}
<div class="back">
    <a href="/">&lt;&lt;&nbsp;返回学科</a>
</div>
</body>
</html>

```

## 加载静态资源

在上面的模板页面中，我们使用了 `<img>` 标签来加载老师的照片，其中使用了引用静态资源的模板指令 `{% static %}`，要使用该指令，首先要使用 `{% load static %}` 指令来加载静态资源，我们将这段代码放在了页码开始的位置。在上面的项目中，我们将静态资源置于名为static的文件夹中，在该文件夹下又创建了三个文件夹：css、js和images，分别用来保存外部层叠样式表、外部JavaScript文件和图片资源。为了能够找到保存静态资源的文件夹，我们还需要修改Django项目的配置文件settings.py，如下所示：

```

# 此处省略上面的代码

STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static'), ]
STATIC_URL = '/static/'

# 此处省略下面的代码

```

接下来修改urls.py文件，配置用户请求的URL和视图函数的对应关系。

```

from django.contrib import admin
from django.urls import path

from vote import views

urlpatterns = [
    path('', views.show_subjects),

```

```
        path('teachers/', views.show_teachers),
        path('admin/', admin.site.urls),
    ]
```

启动服务器运行项目，进入首页查看学科信息。



点击学科查看老师信息。

## Python全栈+人工智能学科老师信息



姓名：骆昊 性别：男 出生日期：1980年11月28日

10年以上软硬件产品设计、研发、架构和管理经验，2003年毕业于四川大学，四川大学Java技术俱乐部创始人，四川省优秀大学毕业生，在四川省网络通信技术重点实验室工作期间，参与了2项国家自然科学基金项目、1项中国科学院中长期研究项目和多项四川省科技攻关项目，在国际会议和国内顶级期刊上发表多篇论文（1篇被SCI收录，3篇被EI收录），大规模网络性能测量系统DMC-TS的设计者和开发者，perf-TTCN语言的发明者。国内最大程序员社区CSDN的博客专家，在Github上参与和维护了多个高质量开源项目，精通C/C++、Java、Python、R、Swift、JavaScript等编程语言，擅长OOAD、系统架构、算法设计、协议分析和网络测量，主持和参与过电子政务系统、KPI考核系统、P2P借贷平台等产品的研发，一直践行“用知识创造快乐”的教学理念，善于总结，乐于分享。

[好评\( 21 \)](#) [差评\( 6 \)](#)



姓名：王海飞 性别：男 出生日期：1993年5月24日

5年以上Python开发经验，先后参与了O2O商城、CRM系统、CMS平台、ERP系统等项目的设计与研发，曾在全国最大最专业的汽车领域相关服务网站担任Python高级研发工程师、项目经理等职务，擅长基于Python、Java、PHP等开发语言的企业级应用开发，全程参与了多个企业级应用从需求到上线所涉及的各种工作，精通Django、Flask等框架，熟悉基于微服务的企业级项目开发，拥有丰富的项目实战经验。善于用浅显易懂的方式在课堂上传授知识点，在授课过程中经常穿插企业开发的实际案例并分析其中的重点和难点，通过这种互动性极强的教学模式帮助学员找到解决问题的办法并提升学员的综合素质。

[好评\( 2 \)](#) [差评\( 1 \)](#)



姓名：余婷 性别：女 出生日期：1992年3月12日

5年以上移动互联网项目开发经验和教学经验，曾担任上市游戏公司高级软件研发工程师和移动端（iOS）技术负责人，参了多个企业级应用和游戏类应用的移动端开发和后台服务器开发，拥有丰富的开发经验和项目管理经验，以个

## Ajax请求

接下来就可以实现“好评”和“差评”的功能了，很明显如果能够在不刷新页面的情况下实现这两个功能会带来更好的用户体验，因此我们考虑使用Ajax技术来实现“好评”和“差评”，Ajax技术我们在之前的章节中已经介绍过了，此处不再赘述。

首先修改项目的urls.py文件，为“好评”和“差评”功能映射对应的URL。

```
from django.contrib import admin
from django.urls import path

from vote import views

urlpatterns = [
    path('', views.show_subjects),
    path('teachers/', views.show_teachers),
    path('praise/', views.praise_or_criticize),
    path('criticize/', views.praise_or_criticize),
    path('admin/', admin.site.urls),
]
```

设计视图函数 `praise_or_criticize` 来支持“好评”和“差评”功能，该视图函数通过Django封装的`JsonResponse`类将字典序列化成JSON字符串作为返回给浏览器的响应内容。

```
def praise_or_criticize(request):
    """好评"""
    try:
        tno = int(request.GET['tno'])
        teacher = Teacher.objects.get(no=tno)
        if request.path.startswith('/prise'):
            teacher.good_count += 1
        else:
            teacher.bad_count += 1
        teacher.save()
        data = {'code': 200, 'hint': '操作成功'}
    except (KeyError, ValueError, Teacher.DoesNotExist):
        data = {'code': 404, 'hint': '操作失败'}
    return JsonResponse(data)
```

修改显示老师信息的模板页，引入jQuery库来实现事件处理、Ajax请求和DOM操作。

```
<script src="{% static 'js/jquery.min.js' %}"></script>
<script>
$(() => {
    $('.comment>a').on('click', (evt) => {
        evt.preventDefault();
        let a = $(evt.target)
        let span = a.next()
        $.getJSON(a.attr('href'), (json) => {
            if (json.code == 200) {
                span.text(parseInt(span.text()) + 1)
            } else {
                alert(json.hint)
            }
        })
    })
})
</script>
```

## 小结

到此为止，这个投票项目的核心功能已然完成，在下一个章节中我们要求用户必须登录才能投票，没有账号的用户可以通过注册功能注册一个账号。

Branch: master ▾ Python-100-Days / Day41-55 /

Create new file Upload files Find file History

This branch is even with jackfrued/master.

Pull request Compare

Cannot retrieve the latest commit at this time.

..

code

res

01.快速上手.md

02.深入模型.md

03.静态资源和Ajax请求.md

04.表单的应用.md

05.Cookie和Session.md

06.中间件的应用.md

07.日志和调试.md

08.文件上传和富文本编辑.md

09.文件下载和报表.md

10.RESTful架构和DRF入门.md

11.RESTful架构和DRF进阶.md

12.使用缓存.md

13.短信和邮件.md

14.异步任务和定时任务.md

15.单元测试和项目上线.md

Branch: master ▾

Find file Copy path

## Python-100-Days / Day41-55 / 04.表单的应用.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



444 lines (375 sloc) 18 KB

## 表单的应用

我们继续来完成上一章节中的项目，实现“用户注册”和“用户登录”的功能，并限制只有登录的用户才能为老师投票。Django框架中提供了对表单的封装，而且提供了多种不同的使用方式。

首先添加用户模型。

```
class User(models.Model):
    """用户"""
    no = models.AutoField(primary_key=True, verbose_name='编号')
    username = models.CharField(max_length=20, unique=True, verbose_name='用户名')
    password = models.CharField(max_length=32, verbose_name='密码')
    regdate = models.DateTimeField(auto_now_add=True, verbose_name='注册时间')

    class Meta:
        db_table = 'tb_user'
        verbose_name_plural = '用户'
```

通过生成迁移和执行迁移操作，在数据库中创建对应的用户表。

```
python manage.py makemigrations 应用名
python manage.py migrate
```

定制一个非常简单的注册模板页面。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
    <style>/* 此处省略层叠样式表选择器 */</style>
</head>
```

```

<body>
    <h1>用户注册</h1>
    <hr>
    <p class="hint">{{ hint }}</p>
    <form action="/register/" method="post">
        {% csrf_token %}
        <div class="input">
            <label for="username">用户名: </label>
            <input type="text" id="username" name="username">
        </div>
        <div class="input">
            <label for="password">密码: </label>
            <input type="password" id="password" name="password">
        </div>
        <div class="input">
            <label for="repassword">确认密码: </label>
            <input type="password" id="repASSWORD" name="repASSWORD">
        </div>
        <div class="input">
            <input type="submit" value="注册">
            <input type="reset" value="重置">
        </div>
    </form>
    <a href="/login">返回登录</a>
</body>
</html>

```

注意，在上面的表单中，我们使用了模板指令 `{% csrf_token %}`，它的作用是在表单中生成一个随机令牌（token）来防范跨站请求伪造（通常简称为CSRF），这也是Django在提交表单时的硬性要求，除非我们专门设置了免除CSRF令牌。

用户在提交注册表单时，我们还需要对用户的输入进行验证，例如我们的网站要求用户名必须由字母、数字、下划线构成且长度在4-20个字符之间，密码的长度为8-20个字符，确认密码必须跟密码保持一致。这些验证操作首先可以通过浏览器中的JavaScript代码来完成，但是即便如此，在服务器端仍然要对用户输入再次进行验证来避免将无效的数据库交给数据库，因为用户可能会禁用浏览器的JavaScript功能，也有可能绕过浏览器的输入检查将注册数据提交给服务器，所以服务器端的用户输入检查仍然是必要的。

我们可以利用Django框架封装的表单功能来对用户输入的有效性进行检查，虽然Django封装的表单还能帮助我们定制出页面上的表单元素，但是这显然是一种灵活性很差的设计，这样的功能在实际开发中基本不考虑，所以表单主要的作用就在于数据验证，具体的做法如下所示。

```

USERNAME_PATTERN = re.compile(r'\w{4,20}')

class RegisterForm(forms.ModelForm):
    repassword = forms.CharField(min_length=8, max_length=20)

    def clean_username(self):
        username = self.cleaned_data['username']

```

```

if not USERNAME_PATTERN.fullmatch(username):
    raise ValidationError('用户名由字母、数字和下划线构成且长度为4-20个字符')
return username

def clean_password(self):
    password = self.cleaned_data['password']
    if len(password) < 8 or len(password) > 20:
        raise ValidationError('无效的密码，密码长度为8-20个字符')
    return to_md5_hex(self.cleaned_data['password'])

def clean_repassword(self):
    repassword = to_md5_hex(self.cleaned_data['repassword'])
    if repassword != self.cleaned_data['password']:
        raise ValidationError('密码和确认密码不一致')
    return repassword

class Meta:
    model = User
    exclude = ('no', 'regdate')

```

上面，我们定义了一个与User模型绑定的表单（继承自ModelForm），我们排除了用户编号（no）和注册日期（regdate）这两个属性，并添加了一个repassword属性用来接收从用户表单传给服务器的确认密码。我们在定义User模型时已经对用户名的最大长度进行了限制，上面我们又对确认密码的最小和最大长度进行了限制，但是这些都不足以完成我们对用户输入的验证。上面以clean\_打头的方法就是我们自定义的验证规则。很明显，clean\_username是对用户名的检查，而clean\_password是对密码的检查。由于数据库二维表中不应该保存密码的原文，所以对密码做了一个简单的MD5摘要处理（实际开发中这样处理还不够，因为有被实施反向查表法（利用彩虹表反向查询）破解用户密码的风险）。生成MD5摘要的代码如下所示。

```

def to_md5_hex(message):
    return hashlib.md5(message.encode()).hexdigest()

```

新增一个视图函数实现用户注册的功能。

```

def register(request):
    page, hint = 'register.html', ''
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            form.save()
            page = 'login.html'
            hint = '注册成功，请登录'
        else:
            hint = '请输入有效的注册信息'
    return render(request, page, {'hint': hint})

```

如果用户发起GET请求，将直接跳转到注册的页面；如果用户以POST方式提交注册表单，则创建自定义的注册表单对象并获取用户输入。可以通过表单对象的 `is_valid` 方法对表单进行验证，如果用户输入没有问题，该方法返回True，否则返回False；由于我们定义的`RegisterForm`继承自`ModelForm`，因此也可以直接使用表单对象的 `save` 方法来保存模型。下面是注册请求的URL配置。

```
from django.contrib import admin
from django.urls import path

from vote import views

urlpatterns = [
    path('', views.show_subjects),
    path('captcha/', views.get_captcha),
    path('teachers/', views.show_teachers),
    path('prise/', views.praise_or_criticize),
    path('criticize/', views.praise_or_criticize),
    path('login/', views.login, name='login'),
    path('register/', views.register, name='register'),
    path('admin/', admin.site.urls),
]
```

说明：上面的代码中我们把待会要用到的登录和验证码的URL也顺便做了映射。  
`path` 函数还可以通过`name`参数给URL绑定一个逆向解析的名字，也就是说，如果需要可以从后面给的名字逆向得到对应的URL。

我们再来定制一个非常简单的登录页。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>用户登录</title>
    <style> /* 此处省略层叠样式表选择器 */ </style>
</head>
<body>
    <h1>用户登录</h1>
    <hr>
    <p class="hint">{{ hint }}</p>
    <form action="/login/" method="post">
        <input type="hidden" name="backurl" value="{{ backurl }}">
        {% csrf_token %}
        <div class="input">
            <label for="username">用户名: </label>
            <input type="text" id="username" name="username">
        </div>
        <div class="input">
            <label for="password">密码: </label>
            <input type="password" id="password" name="password">
        </div>
    </form>
</body>
```

```

<div class="input captcha">
    <label for="captcha">验证码: </label>
    <input type="text" id="captcha" name="captcha">
    
</div>
<div class="input">
    <input type="submit" value="登录">
    <input type="reset" value="重置">
</div>
</form>
<a href="/register">注册新用户</a>
</body>
</html>

```

上面的登录页中，我们要求用户提供验证码，验证码全称是**全自动区分计算机和人类的公开图灵测试**，它是一种用来区分系统的使用者是计算机还是人类的程序。简单的说就是程序出一个只有人类能够回答的问题，由系统使用者来解答，由于计算机理论上无法解答程序提出的问题，所以回答出问题的用户就可以被认为是人类。大多数的网站都使用了不同类型的验证码技术来防范计算机自动注册用户或模拟用户登录（暴力破解用户密码），因为验证码具有一次消费性，而没有通过图灵测试的计算机是不能够注册或登录的。

在Python程序中生成验证码并不算特别复杂，但需要三方库pillow的支持（PIL的分支）。我们可以借鉴现有的方法用Python稍作封装即可。下面的代码已经实现了生成验证码图片并得到图片二进制数据的功能。

```

"""
图片验证码
"""

import os
import random

from io import BytesIO

from PIL import Image
from PIL import ImageFilter
from PIL.ImageDraw import Draw
from PIL.ImageFont import truetype


class Bezier(object):
    """贝塞尔曲线"""

    def __init__(self):
        self.tsequence = tuple([t / 20.0 for t in range(21)])
        self.beziers = {}

    def make_bezier(self, n):
        """绘制贝塞尔曲线"""
        try:
            return self.beziers[n]
        
```

```

        except KeyError:
            combinations = pascal_row(n - 1)
            result = []
            for t in self.tsequence:
                tpowers = (t ** i for i in range(n))
                upowers = ((1 - t) ** i for i in range(n - 1, -1, -1))
                coefs = [c * a * b for c, a, b in zip(combinations,
                                                       tpowers, upowers)]
                result.append(coefs)
            self.beziers[n] = result
        return result

class Captcha(object):
    """验证码"""

    def __init__(self, width, height, fonts=None, color=None):
        self._image = None
        self._fonts = fonts if fonts else \
            [os.path.join(os.path.dirname(__file__), 'fonts', font)
             for font in ['Action.ttf', 'Silom.ttf', 'Verdana.ttf']]
        self._color = color if color else random_color(0, 200, random.randint(220, 255))
        self._width, self._height = width, height

    @classmethod
    def instance(cls, width=200, height=75):
        if not hasattr(Captcha, "_instance"):
            cls._instance = cls(width, height)
        return cls._instance

    def background(self):
        """绘制背景"""
        Draw(self._image).rectangle([(0, 0), self._image.size],
                                   fill=random_color(230, 255))

    def smooth(self):
        """半滑图像"""
        return self._image.filter(ImageFilter.SMOOTH)

    def curve(self, width=4, number=6, color=None):
        """绘制曲线"""
        dx, height = self._image.size
        dx /= number
        path = [(dx * i, random.randint(0, height))
                for i in range(1, number)]
        bcoefs = Bezier().make_bezier(number - 1)
        points = []
        for coefs in bcoefs:
            points.append(tuple(sum([coef * p for coef, p in zip(coefs, ps)])
                               for ps in zip(*path)))
        Draw(self._image).line(points, fill=color if color else self._color, width=wid)

    def noise(self, number=62, level=2, color=None):
        """绘制扰码"""
        width, height = self._image.size

```

```

        dx, dy = width / 10, height / 10
        width, height = width - dx, height - dy
        draw = Draw(self._image)
        for i in range(number):
            x = int(random.uniform(dx, width))
            y = int(random.uniform(dy, height))
            draw.line(((x, y), (x + level, y)),
                      fill=color if color else self._color, width=level)

    def text(self, captcha_text, fonts, font_sizes=None, drawings=None, squeeze_factor=1.0):
        """绘制文本"""
        color = color if color else self._color
        fonts = tuple([truetype(name, size)
                      for name in fonts
                      for size in font_sizes or (65, 70, 75)])
        draw = Draw(self._image)
        char_images = []
        for c in captcha_text:
            font = random.choice(fonts)
            c_width, c_height = draw.textsize(c, font=font)
            char_image = Image.new('RGB', (c_width, c_height), (0, 0, 0))
            char_draw = Draw(char_image)
            char_draw.text((0, 0), c, font=font, fill=color)
            char_image = char_image.crop(char_image.getbbox())
            for drawing in drawings:
                d = getattr(self, drawing)
                char_image = d(char_image)
            char_images.append(char_image)
        width, height = self._image.size
        offset = int((width - sum(int(i.size[0]) * squeeze_factor)
                      for i in char_images[:-1]) -
                     char_images[-1].size[0]) / 2
        for char_image in char_images:
            c_width, c_height = char_image.size
            mask = char_image.convert('L').point(lambda i: i * 1.97)
            self._image.paste(char_image,
                              (offset, int((height - c_height) / 2)),
                              mask)
        offset += int(c_width * squeeze_factor)

    @staticmethod
    def warp(image, dx_factor=0.3, dy_factor=0.3):
        """图像扭曲"""
        width, height = image.size
        dx = width * dx_factor
        dy = height * dy_factor
        x1 = int(random.uniform(-dx, dx))
        y1 = int(random.uniform(-dy, dy))
        x2 = int(random.uniform(-dx, dx))
        y2 = int(random.uniform(-dy, dy))
        warp_image = Image.new(
            'RGB',
            (width + abs(x1) + abs(x2), height + abs(y1) + abs(y2)))
        warp_image.paste(image, (abs(x1), abs(y1)))
        width2, height2 = warp_image.size

```

```

        return warp_image.transform(
            (width, height),
            Image.QUAD,
            (x1, y1, -x1, height2 - y2, width2 + x2, height2 + y2, width2 - x2, -y1))

    @staticmethod
    def offset(image, dx_factor=0.1, dy_factor=0.2):
        """图像偏移"""
        width, height = image.size
        dx = int(random.random() * width * dx_factor)
        dy = int(random.random() * height * dy_factor)
        offset_image = Image.new('RGB', (width + dx, height + dy))
        offset_image.paste(image, (dx, dy))
        return offset_image

    @staticmethod
    def rotate(image, angle=25):
        """图像旋转"""
        return image.rotate(random.uniform(-angle, angle),
                            Image.BILINEAR, expand=1)

    def generate(self, captcha_text='', fmt='PNG'):
        """生成验证码(文字和图片)"""
        self._image = Image.new('RGB', (self._width, self._height), (255, 255, 255))
        self.background()
        self.text(captcha_text, self._fonts,
                  drawings=['warp', 'rotate', 'offset'])
        self.curve(), self.noise(), self.smooth()
        image_bytes = BytesIO()
        self._image.save(image_bytes, format=fmt)
        return image_bytes.getvalue()

    def pascal_row(n=0):
        """生成Pascal三角第n行"""
        result = [1]
        x, numerator = 1, n
        for denominator in range(1, n // 2 + 1):
            x *= numerator
            x /= denominator
            result.append(x)
            numerator -= 1
        if n & 1 == 0:
            result.extend(reversed(result[:-1]))
        else:
            result.extend(reversed(result))
        return result

    def random_color(start=0, end=255, opacity=255):
        """获得随机颜色"""
        red = random.randint(start, end)
        green = random.randint(start, end)
        blue = random.randint(start, end)
        if opacity is None:

```

```
    return red, green, blue
return red, green, blue, opacity
```

下面的视图函数用来生成验证码并通过HttpResponse对象输出到用户浏览器中。

```
ALL_CHARS = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

def get_captcha_text(length=4):
    selected_chars = random.choices(ALL_CHARS, k=length)
    return ''.join(selected_chars)

def get_captcha(request):
    """获得验证码"""
    captcha_text = get_captcha_text()
    image = Captcha.instance().generate(captcha_text)
    return HttpResponse(image, content_type='image/png')
```

生成的验证码如下图所示。



为了验证用户提交的登录表单，我们再定义个表单类。

```
class LoginForm(forms.Form):
    username = forms.CharField(min_length=4, max_length=20)
    password = forms.CharField(min_length=8, max_length=20)
    captcha = forms.CharField(min_length=4, max_length=4)

    def clean_username(self):
        username = self.cleaned_data['username']
        if not USERNAME_PATTERN.fullmatch(username):
            raise ValidationError('无效的用户名')
        return username

    def clean_password(self):
        return to_md5_hex(self.cleaned_data['password'])
```

跟之前我们定义的注册表单类略有区别，登录表单类直接继承自Form没有跟模型绑定，定义了三个字段分别对应登录表单中的用户名、密码和验证码。接下来是处理用户登录的视图函数。

```
def login(request):
    hint = ''
```

```
if request.method == 'POST':
    form = LoginForm(request.POST)
    if form.is_valid():
        username = form.cleaned_data['username']
        password = form.cleaned_data['password']
        user = User.objects.filter(username=username, password=password).first()
        if user:
            return redirect('/')
        else:
            hint = '用户名或密码错误'
    else:
        hint = '请输入有效的登录信息'
return render(request, 'login.html', {'hint': hint})
```

需要指出，上面我们设定用户登录成功时直接返回首页，而且在用户登录时并没有验证用户输入的验证码是否正确，这些我们留到下一个单元再为大家讲解。

Branch: master ▾ Python-100-Days / Day41-55 /

Create new file Upload files Find file History

This branch is even with jackfrued:master.

Pull request Compare

Cannot retrieve the latest commit at this time.

..

code

res

01.快速上手.md

注：django只更新了四节，

02.深入模型.md

03.静态资源和Ajax请求.md

04.表单的应用.md

05.Cookie和Session.md

06.中间件的应用.md

07.日志和调试.md

08.文件上传和富文本编辑.md

09.文件下载和报表.md

10.RESTful架构和DRF入门.md

11.RESTful架构和DRF进阶.md

12.使用缓存.md

13.短信和邮件.md

14.异步任务和定时任务.md

15.单元测试和项目上线.md

# 7:Flask

liupeng678 / Python-100-Days  
forked from jackfrued/Python-100-Days

Branch: master ▾ Python-100-Days / Day56-60 /

Create new file Upload files Find file History

This branch is even with jackfrued:master.

Pull request Compare

Cannot retrieve the latest commit at this time.

..

01.Flask入门.md  
 02.模板的使用.md  
 03.表单的处理.md  
 04.数据库操作.md  
 05.项目实战.md

注： flask未更新



Fetching contributors...

Cannot retrieve contributors at this time.

Raw

Blame

History



141 lines (105 sloc) 9.15 KB

## 预备知识

### 并发编程

所谓并发编程就是让程序中有多个部分能够并发或同时执行，并发编程带来的好处不言而喻，其中最为关键的两点是提升了执行效率和改善了用户体验。下面简单阐述一下Python中实现并发编程的三种方式：

1. 多线程：Python中通过 `threading` 模块的 `Thread` 类并辅以 `Lock`、`Condition`、`Event`、`Semaphore` 和 `Barrier` 等类来支持多线程编程。Python解释器通过GIL（全局解释器锁）来防止多个线程同时执行本地字节码，这个锁对于CPython（Python解释器的官方实现）是必须的，因为CPython的内存管理并不是线程安全的。因为GIL的存在，Python的多线程并不能利用CPU的多核特性。
2. 多进程：使用多进程可以有效的解决GIL的问题，Python中的 `multiprocessing` 模块提供了 `Process` 类来实现多进程，其他的辅助类跟 `threading` 模块中的类类似，由于进程间的内存是相互隔离的（操作系统对进程的保护），进程间通信（共享数据）必须使用管道、套接字等方式，这一点从编程的角度来讲是比较麻烦的，为此，Python的 `multiprocessing` 模块提供了一个名为 `Queue` 的类，它基于管道和锁机制提供了多个进程共享的队列。

"""

用下面的命令运行程序并查看执行时间，例如：

```
time python3 example06.py
real    0m20.657s
user    1m17.749s
sys     0m0.158s
```

使用多进程后实际执行时间为20.657秒，而用户时间1分17.749秒约为实际执行时间的4倍  
这就证明我们的程序通过多进程使用了CPU的多核特性，而且这台计算机配置了4核的CPU

"""

```
import concurrent.futures
import math
```

```

PRIMES = [
    1116281,
    1297337,
    104395303,
    472882027,
    533000389,
    817504243,
    982451653,
    112272535095293,
    112582705942171,
    112272535095293,
    115280095190773,
    115797848077099,
    1099726899285419
] * 5

def is_prime(num):
    """判断素数"""
    assert num > 0
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return False
    return num != 1

def main():
    """主函数"""
    with concurrent.futures.ProcessPoolExecutor() as executor:
        for number, prime in zip(PRIMES, executor.map(is_prime, PRIMES)):
            print('%d is prime: %s' % (number, prime))

if __name__ == '__main__':
    main()

```

3. 异步编程（异步I/O）：所谓异步编程是通过调度程序从任务队列中挑选任务，调度程序以交叉的形式执行这些任务，我们并不能保证任务将以某种顺序去执行，因为执行顺序取决于队列中的一项任务是否愿意将CPU处理时间让位给另一项任务。异步编程通常通过多任务协作处理的方式来实现，由于执行时间和顺序的不确定，因此需要通过钩子函数（回调函数）或者 Future 对象来获取任务执行的结果。目前我们使用的Python 3通过 asyncio 模块以及 await 和 async 关键字（Python 3.5中引入，Python 3.7中正式成为关键字）提供了对异步I/O的支持。

```

import asyncio

async def fetch(host):
    """从指定的站点抓取信息(协程函数)"""
    print(f'Start fetching {host}\n')
    # 跟服务器建立连接

```

```

reader, writer = await asyncio.open_connection(host, 80)
# 构造请求行和请求头
writer.write(b'GET / HTTP/1.1\r\n')
writer.write(f'Host: {host}\r\n'.encode())
writer.write(b'\r\n')
# 清空缓存区(发送请求)
await writer.drain()
# 接收服务器的响应(读取响应行和响应头)
line = await reader.readline()
while line != b'\r\n':
    print(line.decode().rstrip())
    line = await reader.readline()
print('\n')
writer.close()

def main():
    """主函数"""
    urls = ('www.sohu.com', 'www.douban.com', 'www.163.com')
    # 获取系统默认的事件循环
    loop = asyncio.get_event_loop()
    # 用生成式语法构造一个包含多个协程对象的列表
    tasks = [fetch(url) for url in urls]
    # 通过asyncio模块的wait函数将协程列表包装成Task（Future子类）并等待其执行完成
    # 通过事件循环的run_until_complete方法运行任务直到Future完成并返回它的结果
    loop.run_until_complete(asyncio.wait(tasks))
    loop.close()

if __name__ == '__main__':
    main()

```

说明：目前大多数网站都要求基于HTTPS通信，因此上面例子中的网络请求不一定能收到正常的响应，也就是说响应状态码不一定是200，有可能是3xx或者4xx。当然我们这里的重点不在于获得网站响应的内容，而是帮助大家理解asyncio模块以及async和await两个关键字的使用。

我们对三种方式的使用场景做一个简单的总结。

以下情况需要使用多线程：

1. 程序需要维护许多共享的状态（尤其是可变状态），Python中的列表、字典、集合都是线程安全的，所以使用线程而不是进程维护共享状态的代价相对较小。
2. 程序会花费大量时间在I/O操作上，没有太多并行计算的需求且不需占用太多的内存。

以下情况需要使用多进程：

1. 程序执行计算密集型任务（如：字节码操作、数据处理、科学计算）。
2. 程序的输入可以并行的分成块，并且可以将运算结果合并。

3. 程序在内存使用方面没有任何限制且不强依赖于I/O操作（如：读写文件、套接字等）。

最后，如果程序不需要真正的并发性或并行性，而是更多的依赖于异步处理和回调时，异步I/O就是一种很好的选择。另一方面，当程序中有大量的等待与休眠时，也应该考虑使用异步I/O。

扩展：关于进程，还需要做一些补充说明。首先，为了控制进程的执行，操作系统内核必须有能力挂起正在CPU上运行的进程，并恢复以前挂起的某个进程使之继续执行，这种行为被称为进程切换（也叫调度）。进程切换是比较耗费资源的操作，因为在进行切换时首先要保存当前进程的上下文（内核再次唤醒该进程时所需要的状态，包括：程序计数器、状态寄存器、数据栈等），然后还要恢复准备执行的进程的上下文。正在执行的进程由于期待的某些事件未发生，如请求系统资源失败、等待某个操作完成、新数据尚未到达等原因会主动由运行状态变为阻塞状态，当进程进入阻塞状态，是不占用CPU资源的。这些知识对于理解到底选择哪种方式进行并发编程也是很重要的。

## I/O模式和事件驱动

对于一次I/O操作（以读操作为例），数据会先被拷贝到操作系统内核的缓冲区中，然后从操作系统内核的缓冲区拷贝到应用程序的缓冲区（这种方式称为标准I/O或缓存I/O，大多数文件系统的默认I/O都是这种方式），最后交给进程。所以说，当一个读操作发生时（写操作与之类似），它会经历两个阶段：(1)等待数据准备就绪；(2)将数据从内核拷贝到进程中。

由于存在这两个阶段，因此产生了以下几种I/O模式：

1. 阻塞 I/O ( blocking I/O )：进程发起读操作，如果内核数据尚未就绪，进程会阻塞等待数据直到内核数据就绪并拷贝到进程的内存中。
2. 非阻塞 I/O ( non-blocking I/O )：进程发起读操作，如果内核数据尚未就绪，进程不阻塞而是收到内核返回的错误信息，进程收到错误信息可以再次发起读操作，一旦内核数据准备就绪，就立即将数据拷贝到了用户内存中，然后返回。
3. 多路I/O复用 ( I/O multiplexing )：监听多个I/O对象，当I/O对象有变化（数据就绪）的时候就通知用户进程。多路I/O复用的优势并不在于单个I/O操作能处理得更快，而是在于能处理更多的I/O操作。
4. 异步 I/O ( asynchronous I/O )：进程发起读操作后就可以去做别的事情了，内核收到异步读操作后会立即返回，所以用户进程不阻塞，当内核数据准备就绪时，内核发送一个信号给用户进程，告诉它读操作完成了。

通常，我们编写一个处理用户请求的服务器程序时，有以下三种方式可供选择：

1. 每收到一个请求，创建一个新的进程，来处理该请求；
2. 每收到一个请求，创建一个新的线程，来处理该请求；
3. 每收到一个请求，放入一个事件列表，让主进程通过非阻塞I/O方式来处理请求

第1种方式实现比较简单，但由于创建进程开销比较大，会导致服务器性能比较差；第2种方式，由于要涉及到线程的同步，有可能会面临竞争、死锁等问题；第3种方式，就是所谓事件驱动的方式，它利用了多路I/O复用和异步I/O的优点，虽然代码逻辑比前面两种都复杂，但能达到最好的性能，这也是目前大多数网络服务器采用的方式。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day61-65 / 02.Tornado入门.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



378 lines (287 sloc) 13.7 KB

# Tornado入门

## Tornado概述

Python的Web框架种类繁多（比Python语言的关键字还要多），但在众多优秀的Web框架中，Tornado框架最适合用来开发需要处理长连接和应对高并发的Web应用。Tornado框架在设计之初就考虑到性能问题，通过对非阻塞I/O和epoll（Linux 2.5.44内核引入的一种多路I/O复用方式，旨在实现高性能网络服务，在BSD和macOS中是kqueue）的运用，Tornado可以处理大量的并发连接，更轻松的应对C10K（万级并发）问题，是非常理想的实时通信Web框架。

扩展：基于线程的Web服务器产品（如：Apache）会维护一个线程池来处理用户请求，当用户请求到达时就为该请求分配一个线程，如果线程池中没有空闲线程了，那么可以通过创建新的线程来应付新的请求，但前提是系统尚有空闲的内存空间，显然这种方式很容易将服务器的空闲内存耗尽（大多数Linux发行版本中，默认的线程栈大小为8M）。想象一下，如果我们要开发一个社交类应用，这类应用中，通常需要显示实时更新的消息、对象状态的变化和各种类型的通知，那也就意味着客户端需要保持请求连接来接收服务器的各种响应，在这种情况下，服务器上的工作线程很容易被耗尽，这也就意味着新的请求很有可能无法得到响应。

Tornado框架源于FriendFeed网站，在FriendFeed网站被Facebook收购之后得以开源，正式发布的日期是2009年9月10日。Tornado能让你能够快速开发高速的Web应用，如果你想编写一个可扩展的社交应用、实时分析引擎，或RESTful API，那么Tornado框架就是很好的选择。Tornado其实不仅仅是一个Web开发的框架，它还是一个高性能的事件驱动网络访问引擎，内置了高性能的HTTP服务器和客户端（支持同步和异步请求），同时还对WebSocket提供了完美的支持。

了解和学习Tornado最好的资料就是它的官方文档，在[tornadoweb.org](http://tornadoweb.org)上面有很多不错的例子，你也可以在Github上找到Tornado的源代码和历史版本。

## 5分钟上手Tornado

## 1. 创建并激活虚拟环境。

```
mkdir hello-tornado
cd hello-tornado
python3 -m venv venv
source venv/bin/activate
```

## 2. 安装Tornado。

```
pip install tornado
```

## 3. 编写Web应用。

```
"""
example01.py
"""

import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):

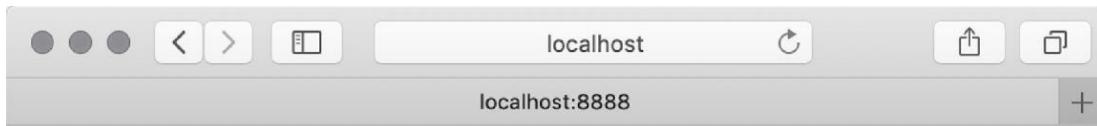
    def get(self):
        self.write('<h1>Hello, world!</h1>')


def main():
    app = tornado.web.Application(handlers=[(r'/', MainHandler), ])
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()

if __name__ == '__main__':
    main()
```

## 4. 运行并访问应用。

```
python example01.py
```



## Hello, world!

在上面的例子中，代码example01.py通过定义一个继承自 RequestHandler 的类（ MainHandler ）来处理用户请求，当请求到达时，Tornado会实例化这个类（创建 MainHandler 对象），并调用与HTTP请求方法（ GET、POST等 ）对应的方法，显然上面的 MainHandler 只能处理GET请求，在收到GET请求时，它会将一段HTML的内容写入到HTTP响应中。 main 函数的第1行代码创建了Tornado框架中 Application 类的实例，它代表了我们的Web应用，而创建该实例最为重要的参数就是 handlers ，该参数告知 Application 对象，当收到一个请求时应该通过哪个类的对象来处理这个请求。在上面的例子中，当通过HTTP的GET请求访问站点根路径时，就会调用 MainHandler 的 get 方法。 main 函数的第2行代码通过 Application 对象的 listen 方法指定了监听HTTP请求的端口。 main 函数的第3行代码用于获取Tornado框架的 IOLoop 实例并启动它，该实例代表一个条件触发的I/O循环，用于持续的接收来自于客户端的请求。

扩展：在Python 3中， IOLoop 实例的本质就是 asyncio 的事件循环，该事件循环在非Windows系统中就是 SelectorEventLoop 对象，它基于 selectors 模块（高级I/O复用模块），会使用当前操作系统最高效的I/O复用选择器，例如在Linux环境下它使用 EpollSelector ，而在macOS和BSD环境下它使用的是 KqueueSelector ；在Python 2中， IOLoop 直接使用 select 模块（低级I/O复用模块）的 epoll 或 kqueue 函数，如果这两种方式都不可用，则调用 select 函数实现多路I/O复用。当然，如果要支持高并发，你的系统最好能够支持epoll或者kqueue这两种多路I/O复用方式中的一种。

如果希望通过命令行参数来指定Web应用的监听端口，可以对上面的代码稍作修改。

```
"""
example01.py
"""

import tornado.ioloop
import tornado.web

from tornado.options import define, options, parse_command_line

# 定义默认端口
```

```

define('port', default=8000, type=int)

class MainHandler(tornado.web.RequestHandler):

    def get(self):
        self.write('<h1>Hello, world!</h1>')

def main():
    # python example01.py --port=8000
    parse_command_line()
    app = tornado.web.Application(handlers=[(r'/', MainHandler), ])
    app.listen(options.port)
    tornado.ioloop.IOLoop.current().start()

if __name__ == '__main__':
    main()

```

在启动Web应用时，如果没有指定端口，将使用 `define` 函数中设置的默认端口8000，如果要指定端口，可以使用下面的方式来启动Web应用。

```
python example01.py --port=8000
```

## 路由解析

上面我们曾经提到过创建 `Application` 实例时需要指定 `handlers` 参数，这个参数非常重要，它应该是一个元组的列表，元组中的第一个元素是正则表达式，它用于匹配用户请求的资源路径；第二个元素是 `RequestHandler` 的子类。在刚才的例子中，我们只在 `handlers` 列表中放置了一个元组，事实上我们可以放置多个元组来匹配不同的请求（资源路径），而且可以使用正则表达式的捕获组来获取匹配的内容并将其作为参数传入到 `get`、`post` 这些方法中。

```

"""
example02.py
"""

import os
import random

import tornado.ioloop
import tornado.web

from tornado.options import define, options, parse_command_line

# 定义默认端口
define('port', default=8000, type=int)

```

```

class SayingHandler(tornado.web.RequestHandler):
    """自定义请求处理器"""

    def get(self):
        sayings = [
            '世上没有绝望的处境，只有对处境绝望的人',
            '人生的道路在态度的岔口一分为二，从此通向成功或失败',
            '所谓措手不及，不是说没有时间准备，而是有时间的时候没有准备',
            '那些你认为不靠谱的人生里，充满你没有勇气做的事',
            '在自己喜欢的时间里，按照自己喜欢的方式，去做自己喜欢做的事，这便是自由',
            '有些人不属于自己，但是遇见了也弥足珍贵'
        ]
        # 渲染index.html模板页
        self.render('index.html', message=random.choice(sayings))

class WeatherHandler(tornado.web.RequestHandler):
    """自定义请求处理器"""

    def get(self, city):
        # Tornado框架会自动处理百分号编码的问题
        weathers = {
            '北京': {'temperature': '-4~4', 'pollution': '195 中度污染'},
            '成都': {'temperature': '3~9', 'pollution': '53 良'},
            '深圳': {'temperature': '20~25', 'pollution': '25 优'},
            '广州': {'temperature': '18~23', 'pollution': '56 良'},
            '上海': {'temperature': '6~8', 'pollution': '65 良'}
        }
        if city in weathers:
            self.render('weather.html', city=city, weather=weathers[city])
        else:
            self.render('index.html', message=f'没有{city}的天气信息')

class ErrorHandler(tornado.web.RequestHandler):
    """自定义请求处理器"""

    def get(self):
        # 重定向到指定的路径
        self.redirect('/saying')

def main():
    """主函数"""
    parse_command_line()
    app = tornado.web.Application(
        # handlers是按列表中的顺序依次进行匹配的
        handlers=[
            (r'/saying/?', SayingHandler),
            (r'/weather/([^\?]{2,})/?', WeatherHandler),
            (r'/.+', ErrorHandler),
        ],
        # 通过template_path参数设置模板页的路径
        template_path=os.path.join(os.path.dirname(__file__), 'templates')
    )

```

```
)  
app.listen(options.port)  
tornado.ioloop.IOLoop.current().start()  
  
if __name__ == '__main__':  
    main()
```

模板页index.html。

```
<!-- index.html -->  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="utf-8">  
    <title>Tornado基础</title>  
</head>  
<body>  
    <h1>{{message}}</h1>  
</body>  
</html>
```

模板页weather.html。

```
<!-- weather.html -->  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="utf-8">  
    <title>Tornado基础</title>  
</head>  
<body>  
    <h1>{{city}}</h1>  
    <hr>  
    <h2>温度: {{weather['temperature']}}摄氏度</h2>  
    <h2>污染指数: {{weather['pollution']}}</h2>  
</body>  
</html>
```

Tornado的模板语法与其他的Web框架中使用的模板语法并没有什么实质性的区别，而且目前的Web应用开发更倡导使用前端渲染的方式来减轻服务器的负担，所以这里我们并不对模板语法和后端渲染进行深入的讲解。

## 请求处理器

通过上面的代码可以看出，RequestHandler 是处理用户请求的核心类，通过重写 get、post、put、delete 等方法可以处理不同类型的HTTP请求，除了这些方法之外，RequestHandler 还实现了很多重要的方法，下面是部分方法的列表：

1. `get_argument / get_arguments / get_body_argument / get_body_arguments / get_query_arugment / get_query_arguments` : 获取请求参数。
2. `set_status / send_error / set_header / add_header / clear_header / clear` : 操作状态码和响应头。
3. `write / flush / finish / write_error` : 和输出相关的方法。
4. `render / render_string` : 渲染模板。
5. `redirect` : 请求重定向。
6. `get_cookie / set_cookie / get_secure_cookie / set_secure_cookie / create_signed_value / clear_cookie / clear_all_cookies` : 操作Cookie。

我们用上面讲到的这些方法来完成下面的需求，访问页面时，如果Cookie中没有读取到用户信息则要求用户填写个人信息，如果从Cookie中读取到用户信息则直接显示用户信息。

```
"""
example03.py
"""

import os
import re

import tornado.ioloop
import tornado.web

from tornado.options import define, options, parse_command_line


# 定义默认端口
define('port', default=8000, type=int)

users = {}


class User(object):
    """用户"""

    def __init__(self, nickname, gender, birthday):
        self.nickname = nickname
        self.gender = gender
        self.birthday = birthday


class MainHandler(tornado.web.RequestHandler):
    """自定义请求处理器"""

    def get(self):
        # 从Cookie中读取用户昵称
        nickname = self.get_cookie('nickname')
        if nickname in users:
            self.render('userinfo.html', user=users[nickname])
        else:
```

```

        self.render('userform.html', hint='请填写个人信息')

class UserHandler(tornado.web.RequestHandler):
    """自定义请求处理器"""

    def post(self):
        # 从表单参数中读取用户昵称、性别和生日信息
        nickname = self.get_body_argument('nickname').strip()
        gender = self.get_body_argument('gender')
        birthday = self.get_body_argument('birthday')
        # 检查用户名是否有效
        if not re.fullmatch(r'\w{6,20}', nickname):
            self.render('userform.html', hint='请输入有效的昵称')
        elif nickname in users:
            self.render('userform.html', hint='昵称已经被使用过')
        else:
            users[nickname] = User(nickname, gender, birthday)
            # 将用户名写入Cookie并设置有效期为7天
            self.set_cookie('nickname', nickname, expires_days=7)
            self.render('userinfo.html', user=users[nickname])

    def main():
        """主函数"""
        parse_command_line()
        app = tornado.web.Application(
            handlers=[
                (r'/', MainHandler), (r'/register', UserHandler)
            ],
            template_path=os.path.join(os.path.dirname(__file__), 'templates')
        )
        app.listen(options.port)
        tornado.ioloop.IOLoop.current().start()

    if __name__ == '__main__':
        main()

```

模板页userform.html。

```

<!-- userform.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tornado基础</title>
    <style>
        .em { color: red; }
    </style>
</head>
<body>
    <h1>填写用户信息</h1>

```

```
<hr>
    <p class="em">{{hint}}</p>
<form action="/register" method="post">
    <p>
        <label>昵称: </label>
        <input type="text" name="nickname">
        (字母数字下划线, 6-20个字符)
    </p>
    <p>
        <label>性别: </label>
        <input type="radio" name="gender" value="男" checked>男
        <input type="radio" name="gender" value="女">女
    </p>
    <p>
        <label>生日: </label>
        <input type="date" name="birthday" value="1990-01-01">
    </p>
    <p>
        <input type="submit" value="确定">
    </p>
</form>
</body>
</html>
```

模板页 userinfo.html。

```
<!-- userinfo.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tornado基础</title>
</head>
<body>
    <h1>用户信息</h1>
    <hr>
    <h2>昵称: {{user.nickname}}</h2>
    <h2>性别: {{user.gender}}</h2>
    <h2>出生日期: {{user.birthday}}</h2>
</body>
</html>
```

Branch: master ▾

[Find file](#)[Copy path](#)

## Python-100-Days / Day61-65 / 03.异步化.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#)[Blame](#)[History](#)

152 lines (116 sloc) 4.87 KB

## 异步化

在前面的例子中，我们并没有对 RequestHandler 中的 get 或 post 方法进行异步处理，这就意味着，一旦在 get 或 post 方法中出现了耗时间的操作，不仅仅是当前请求被阻塞，按照Tornado框架的工作模式，其他的请求也会被阻塞，所以我们需要对耗时间的操作进行异步化处理。

在Tornado稍早一些的版本中，可以用装饰器实现请求方法的异步化或协程化来解决这个问题。

- 给 RequestHandler 的请求处理函数添加 @tornado.web.asynchronous 装饰器，如下所示：

```
class AsyncReqHandler(RequestHandler):  
  
    @tornado.web.asynchronous  
    def get(self):  
        http = httpclient.AsyncHTTPClient()  
        http.fetch("http://example.com/", self._on_download)  
  
    def _on_download(self, response):  
        do_something_with_response(response)  
        self.render("template.html")
```

- 给 RequestHandler 的请求处理函数添加 @tornado.gen.coroutine 装饰器，如下所示：

```
class GenAsyncHandler(RequestHandler):  
  
    @tornado.gen.coroutine  
    def get(self):  
        http_client = AsyncHTTPClient()  
        response = yield http_client.fetch("http://example.com")  
        do_something_with_response(response)  
        self.render("template.html")
```

- 使用 @return\_future 装饰器，如下所示：

```
@return_future
def future_func(arg1, arg2, callback):
    # Do stuff (possibly asynchronous)
    callback(result)

async def caller():
    await future_func(arg1, arg2)
```

在Tornado 5.x版本中，这几个装饰器都被标记为**depricated**（过时），我们可以通过Python 3.5中引入的 `async` 和 `await`（在Python 3.7中已经成为正式的关键字）来达到同样的效果。当然，要实现异步化还得靠其他的支撑异步操作的三方库来支持，如果请求处理函数中用到了不支持异步操作的三方库，就需要靠自己写包装类来支持异步化。

下面的代码演示了在读写数据库时如何实现请求处理的异步化。我们用到的数据库建表语句如下所示：

```
create database hrs default charset utf8;

use hrs;

/* 创建部门表 */
create table tb_dept
(
    dno      int not null comment '部门编号',
    dname    varchar(10) not null comment '部门名称',
    dloc     varchar(20) not null comment '部门所在地',
    primary key (dno)
);

insert into tb_dept values
(10, '会计部', '北京'),
(20, '研发部', '成都'),
(30, '销售部', '重庆'),
(40, '运维部', '深圳');
```

我们通过下面的代码实现了查询和新增部门两个操作。

```
import json

import aiomysql
import tornado
import tornado.web

from tornado.ioloop import IOLoop
from tornado.options import define, parse_command_line, options
```

```
define('port', default=8000, type=int)

async def connect_mysql():
    return await aiomysql.connect(
        host='120.77.222.217',
        port=3306,
        db='hrs',
        user='root',
        password='123456',
    )

class HomeHandler(tornado.web.RequestHandler):

    async def get(self, no):
        async with self.settings['mysql'].cursor(aiomysql.DictCursor) as cursor:
            await cursor.execute("select * from tb_dept where dno=%s", (no, ))
        if cursor.rowcount == 0:
            self.finish(json.dumps({
                'code': 20001,
                'mesg': f'没有编号为{no}的部门'
            }))
            return
        row = await cursor.fetchone()
        self.finish(json.dumps(row))

    async def post(self, *args, **kwargs):
        no = self.get_argument('no')
        name = self.get_argument('name')
        loc = self.get_argument('loc')
        conn = self.settings['mysql']
        try:
            async with conn.cursor() as cursor:
                await cursor.execute('insert into tb_dept values (%s, %s, %s)',
                                     (no, name, loc))
            await conn.commit()
        except aiomysql.MySQLError:
            self.finish(json.dumps({
                'code': 20002,
                'mesg': '添加部门失败请确认部门信息'
            }))
        else:
            self.set_status(201)
            self.finish()

def make_app(config):
    return tornado.web.Application(
        handlers=[(r'/api/depts/(.*)', HomeHandler), ],
        **config
    )

def main():
```

```
parse_command_line()
app = make_app({
    'debug': True,
    'mysql': IOLoop.current().run_sync(connect_mysql)
})
app.listen(options.port)
IOLoop.current().start()

if __name__ == '__main__':
    main()
```

上面的代码中，我们用到了 `aiomysql` 这个三方库，它基于 `pymysql` 封装，实现了对 MySQL 操作的异步化。操作 Redis 可以使用 `aioredis`，访问 MongoDB 可以使用 `motor`，这些都是支持异步操作的三方库。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day61-65 / 04.WebSocket的应用.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



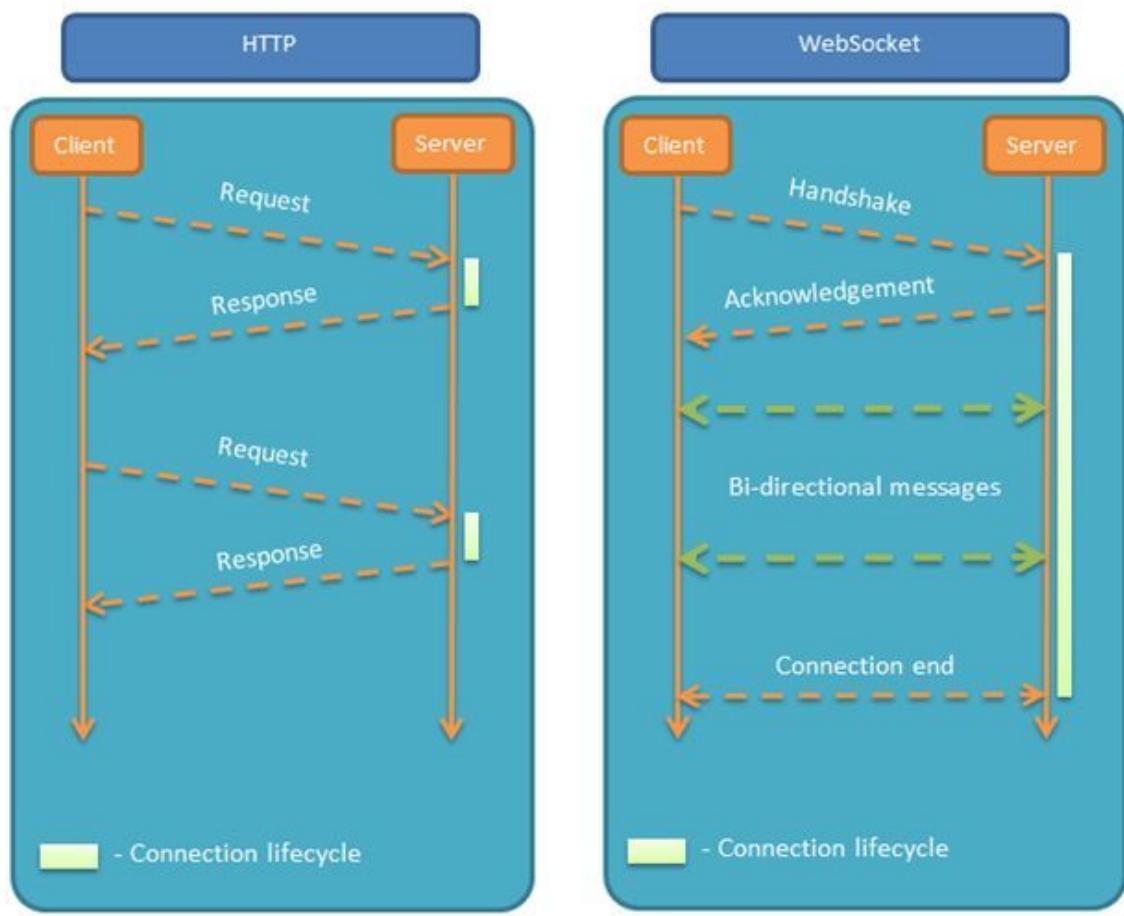
229 lines (185 sloc) 9.78 KB

# WebSocket的应用

Tornado的异步特性使其非常适合处理高并发的业务，同时也适合那些需要在客户端和服务器之间维持长连接的业务。传统的基于HTTP协议的Web应用，服务器和客户端（浏览器）的通信只能由客户端发起，这种单向请求注定了如果服务器有连续的状态变化，客户端（浏览器）是很难得知的。事实上，今天的很多Web应用都需要服务器主动向客户端（浏览器）发送数据，我们将这种通信方式称之为“推送”。过去很长一段时间，程序员都是用定时轮询（Polling）或长轮询（Long Polling）等方式来实现“推送”，但是这些都不是真正意义上的“推送”，而且浪费资源且效率低下。在HTML5时代，可以通过一种名为WebSocket的技术在服务器和客户端（浏览器）之间维持传输数据的长连接，这种方式可以实现真正的“推送”服务。

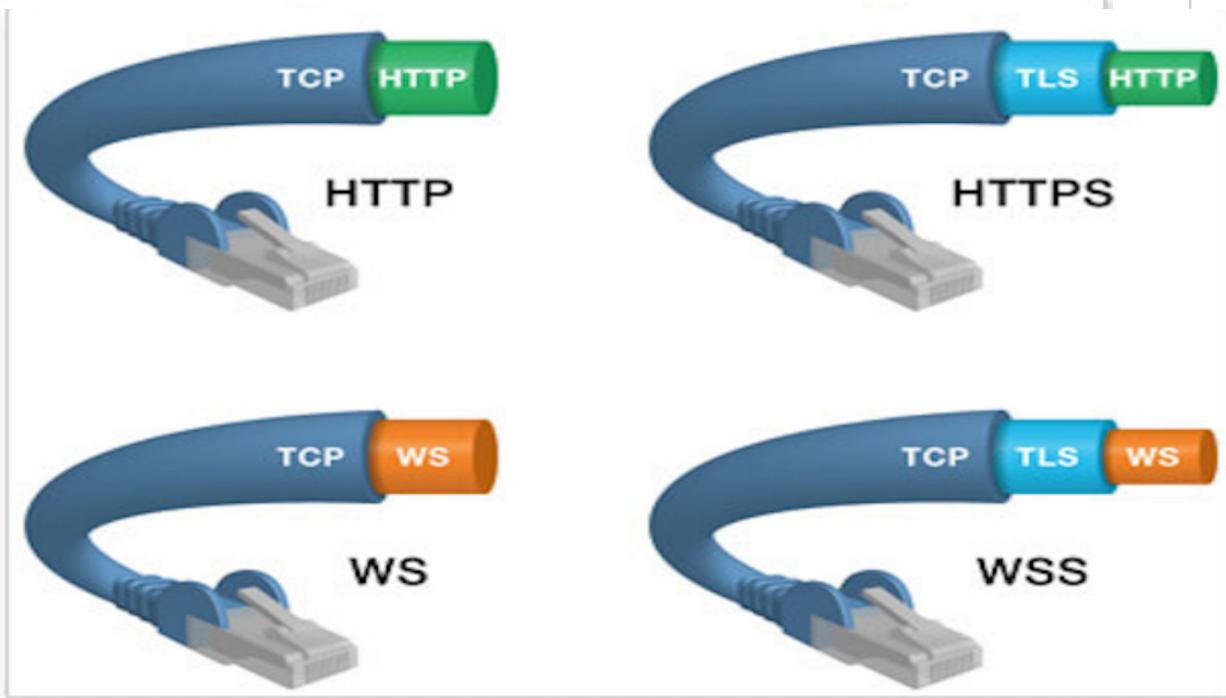
## WebSocket简介

WebSocket 协议在2008年诞生，2011年成为国际标准（[RFC 6455](#)），现在的浏览器都能够支持它，它可以实现浏览器和服务器之间的全双工通信。我们之前学习或了解过 Python 的 Socket 编程，通过 Socket 编程，可以基于 TCP 或 UDP 进行数据传输；而 WebSocket 与之类似，只不过它是基于 HTTP 来实现通信握手，使用 TCP 来进行数据传输。WebSocket 的出现打破了 HTTP 请求和响应只能一对一通信的模式，也改变了服务器只能被动接受客户端请求的状况。目前有很多 Web 应用是需要服务器主动向客户端发送信息的，例如股票信息的网站可能需要向浏览器发送股票涨停通知，社交网站可能需要向用户发送好友上线提醒或聊天信息。



WebSocket的特点如下所示：

1. 建立在TCP协议之上，服务器端的实现比较容易。
2. 与HTTP协议有着良好的兼容性，默认端口是80（WS）和443（WSS），通信握手阶段采用HTTP协议，能通过各种HTTP代理服务器（不容易被防火墙阻拦）。
3. 数据格式比较轻量，性能开销小，通信高效。
4. 可以发送文本，也可以发送二进制数据。
5. 没有同源策略的限制，客户端（浏览器）可以与任意服务器通信。



## WebSocket服务器端编程

Tornado框架中有一个 `tornado.websocket.WebSocketHandler` 类专门用于处理来自 WebSocket的请求，通过继承该类并重写 `open`、`on_message`、`on_close` 等方法来处理 WebSocket通信，下面我们将对 `WebSocketHandler` 的核心方法做一个简单的介绍。

1. `open(*args, **kwargs)` 方法：建立新的WebSocket连接后，Tornado框架会调用该方法，该方法的参数与 `RequestHandler` 的 `get` 方法的参数类似，这也就意味着在 `open` 方法中可以执行获取请求参数、读取Cookie信息这样的操作。
2. `on_message(message)` 方法：建立WebSocket之后，当收到来自客户端的消息时，Tornado框架会调用该方法，这样就可以对收到的消息进行对应的处理，必须重写这个方法。
3. `on_close()` 方法：当WebSocket被关闭时，Tornado框架会调用该方法，在该方法中可以通过 `close_code` 和 `close_reason` 了解关闭的原因。
4. `write_message(message, binary=False)` 方法：将指定的消息通过WebSocket发送给客户端，可以传递utf-8字符序列或者字节序列，如果message是一个字典，将会执行JSON序列化。正常情况下，该方法会返回一个 `Future` 对象；如果WebSocket被关闭了，将引发 `WebSocketClosedError`。
5. `set_nodelay(value)` 方法：默认情况下，因为TCP的Nagle算法会导致短小的消息被延迟发送，在考虑到交互性的情况下就要通过将该方法的参数设置为 `True` 来避免延迟。

6. `close(code=None, reason=None)` 方法：主动关闭WebSocket，可以指定状态码（详见[RFC 6455 7.4.1节](#)）和原因。

## WebSocket客户端编程

1. 创建WebSocket对象。

```
var webSocket = new WebSocket('ws://localhost:8000/ws');
```

说明：webSocket对象的readyState属性表示该对象当前状态，取值为CONNECTING-正在连接，OPEN-连接成功可以通信，CLOSING-正在关闭，CLOSED-已经关闭。

2. 编写回调函数。

```
webSocket.onopen = function(evt) { webSocket.send('...'); };
webSocket.onmessage = function(evt) { console.log(evt.data); };
webSocket.onclose = function(evt) {};
webSocket.onerror = function(evt) {};
```

说明：如果要绑定多个事件回调函数，可以用`addEventListener`方法。另外，通过事件对象的`data`属性获得的数据可能是字符串，也有可能是二进制数据，可以通过`webSocket`对象的`binaryType`属性（`blob`、`arraybuffer`）或者通过`typeof`、`instanceof`运算符检查类型进行判定。

## 项目：Web聊天室

```
"""
handlers.py - 用户登录和聊天的处理器
"""

import tornado.web
import tornado.websocket

nicknames = set()
connections = {}

class LoginHandler(tornado.web.RequestHandler):

    def get(self):
        self.render('login.html', hint='')

    def post(self):
        nickname = self.get_argument('nickname')
        if nickname in nicknames:
            self.render('login.html', hint='昵称已被使用，请更换昵称')
        self.set_secure_cookie('nickname', nickname)
```

```

    self.render('chat.html')

class ChatHandler(tornado.websocket.WebSocketHandler):

    def open(self):
        nickname = self.get_secure_cookie('nickname').decode()
        nicknames.add(nickname)
        for conn in connections.values():
            conn.write_message(f'~~~{nickname}进入了聊天室~~~')
        connections[nickname] = self

    def on_message(self, message):
        nickname = self.get_secure_cookie('nickname').decode()
        for conn in connections.values():
            if conn is not self:
                conn.write_message(f'{nickname}说: {message}')

    def on_close(self):
        nickname = self.get_secure_cookie('nickname').decode()
        del connections[nickname]
        nicknames.remove(nickname)
        for conn in connections.values():
            conn.write_message(f'~~~{nickname}离开了聊天室~~~')

"""

run_chat_server.py - 聊天服务器
"""

import os

import tornado.web
import tornado.ioloop

from handlers import LoginHandler, ChatHandler

if __name__ == '__main__':
    app = tornado.web.Application(
        handlers=[(r'/login', LoginHandler), (r'/chat', ChatHandler)],
        template_path=os.path.join(os.path.dirname(__file__), 'templates'),
        static_path=os.path.join(os.path.dirname(__file__), 'static'),
        cookie_secret='MWM2MzEyOWF10Wri0WM2MGMzZThhYTk0ZDNlMDA0OTU=',
    )
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()

<!-- login.html --&gt;
&lt;!DOCTYPE html&gt;
&lt;html lang="en"&gt;
&lt;head&gt;
    &lt;meta charset="utf-8"&gt;
    &lt;title&gt;Tornado聊天室&lt;/title&gt;
</pre>

```

```
<style>
    .hint { color: red; font-size: 0.8em; }
</style>
</head>
<body>
    <div>
        <div id="container">
            <h1>进入聊天室</h1>
            <hr>
            <p class="hint">{{hint}}</p>
            <form method="post" action="/login">
                <label>昵称: </label>
                <input type="text" placeholder="请输入你的昵称" name="nickname">
                <button type="submit">登录</button>
            </form>
        </div>
    </div>
</body>
</html>
```

```
<!-- chat.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tornado聊天室</title>
</head>
<body>
    <h1>聊天室</h1>
    <hr>
    <div>
        <textarea id="contents" rows="20" cols="120" readonly></textarea>
    </div>
    <div class="send">
        <input type="text" id="content" size="50">
        <input type="button" id="send" value="发送">
    </div>
    <p>
        <a id="quit" href="javascript:void(0);">退出聊天室</a>
    </p>
    <script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>
    <script>
        $(function() {
            // 将内容追加到指定的文本区
            function appendContent($ta, message) {
                var contents = $ta.val();
                contents += '\n' + message;
                $ta.val(contents);
                $ta[0].scrollTop = $ta[0].scrollHeight;
            }
            // 通过WebSocket发送消息
            function sendMessage() {
                message = $('#content').val().trim();
```

```
if (message.length > 0) {
    ws.send(message);
    appendContent($('#contents'), '我说: ' + message);
    $('#content').val('');
}
// 创建WebSocket对象
var ws= new WebSocket('ws://localhost:8888/chat');
// 连接建立后执行的回调函数
ws.onopen = function(evt) {
    $('#contents').val('~~~欢迎您进入聊天室~~~');
};
// 收到消息后执行的回调函数
ws.onmessage = function(evt) {
    appendContent($('#contents'), evt.data);
};
// 为发送按钮绑定点击事件回调函数
$('#send').on('click', sendMessage);
// 为文本框绑定按下回车事件回调函数
$('#content').on('keypress', function(evt) {
    keycode = evt.keyCode || evt.which;
    if (keycode == 13) {
        sendMessage();
    }
});
// 为退出聊天室超链接绑定点击事件回调函数
$('#quit').on('click', function(evt) {
    ws.close();
    location.href = '/login';
});
});
</script>
</body>
</html>
```

Branch: master ▾

[Find file](#) [Copy path](#)

## [Python-100-Days](#) / [Day61-65](#) / [05.项目实战.md](#)

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



3 lines (1 sloc) 17 Bytes

### 项目实战

注：未更新

# 9: 爬虫

Branch: master ▾

[Find file](#) [Copy path](#)

[Python-100-Days / Day66-75 / 01.网络爬虫和相关工具.md](#)

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



316 lines (235 sloc) 16.4 KB

## 网络爬虫和相关工具

### 网络爬虫

网络爬虫（ web crawler ），以前经常称之为网络蜘蛛（ spider ），是按照一定的规则自动浏览万维网并获取信息的机器人程序（或脚本），曾经被广泛的应用于互联网搜索引擎。使用过互联网和浏览器的人都知道，网页中除了供用户阅读的文字信息之外，还包含一些超链接。网络爬虫系统正是通过网页中的超链接信息不断获得网络上的其它页面。正因如此，网络数据采集的过程就像一个爬虫或者蜘蛛在网络上漫游，所以才被形象的称为网络爬虫或者网络蜘蛛。

### 爬虫的应用领域

在理想的状态下，所有ICP（ Internet Content Provider ）都应该为自己的网站提供API接口来共享它们允许其他程序获取的数据，在这种情况下爬虫就不是必需品，国内比较有名的电商平台（如淘宝、京东等）、社交平台（如腾讯微博等）等网站都提供了自己的Open API，但是这类Open API通常会对可以抓取的数据以及抓取数据的频率进行限制。对于大多数的公司而言，及时的获取行业相关数据是企业生存的重要环节之一，然而大部分企业在行业数据方面的匮乏是其与生俱来的短板，合理的利用爬虫来获取数据并从中提取出有商业价值的信息是至关重要的。当然爬虫还有很多重要的应用领域，下面列举了其中的一部分：

1. 搜索引擎
2. 新闻聚合
3. 社交应用
4. 舆情监控
5. 行业数据

### 合法性和背景调研

## 爬虫合法性探讨

1. 网络爬虫领域目前还属于拓荒阶段，虽然互联网世界已经通过自己的游戏规则建立起一定的道德规范(Robots协议，全称是“网络爬虫排除标准”)，但法律部分还在建立和完善中，也就是说，现在这个领域暂时还是灰色地带。
2. “法不禁止即为许可”，如果爬虫就像浏览器一样获取的是前端显示的数据（网页上的公开信息）而不是网站后台的私密敏感信息，就不太担心法律法规的约束，因为目前大数据产业链的发展速度远远超过了法律的完善程度。
3. 在爬取网站的时候，需要限制自己的爬虫遵守Robots协议，同时控制网络爬虫程序的抓取数据的速度；在使用数据的时候，必须要尊重网站的知识产权（从Web 2.0时代开始，虽然Web上的数据很多都是由用户提供的，但是网站平台是投入了运营成本的，当用户在注册和发布内容时，平台通常就已经获得了对数据的所有权、使用权和分发权）。如果违反了这些规定，在打官司的时候败诉几率相当高。

### Robots.txt文件

大多数网站都会定义robots.txt文件，下面以淘宝的[robots.txt文件](#)为例，看看该网站对爬虫有哪些限制。

```
User-agent: Baiduspider
Allow: /article
Allow: /oshtml
Disallow: /product/
Disallow: /
```

```
User-Agent: Googlebot
Allow: /article
Allow: /oshtml
Allow: /product
Allow: /spu
Allow: /dianpu
Allow: /oversea
Allow: /list
Disallow: /
```

```
User-agent: Bingbot
Allow: /article
Allow: /oshtml
Allow: /product
Allow: /spu
Allow: /dianpu
Allow: /oversea
Allow: /list
Disallow: /
```

```
User-Agent: 360Spider
Allow: /article
Allow: /oshtml
Disallow: /
```

```
User-Agent: Yisouspider  
Allow: /article  
Allow: /oshtml  
Disallow: /
```

```
User-Agent: Sogouspider  
Allow: /article  
Allow: /oshtml  
Allow: /product  
Disallow: /
```

```
User-Agent: Yahoo! Slurp  
Allow: /product  
Allow: /spu  
Allow: /dianpu  
Allow: /oversea  
Allow: /list  
Disallow: /
```

```
User-Agent: *  
Disallow: /
```

注意上面robots.txt第一段的最后一行，通过设置“Disallow: /”禁止百度爬虫访问除了“Allow”规定页面外的其他所有页面。因此当你在百度搜索“淘宝”的时候，搜索结果下方会出现：“由于该网站的robots.txt文件存在限制指令（限制搜索引擎抓取），系统无法提供该页面的内容描述”。百度作为一个搜索引擎，至少在表面上遵守了淘宝网的robots.txt协议，所以用户不能从百度上搜索到淘宝内部的产品信息。



淘宝



百度一下

### 淘 淘宝网 - 淘!我喜欢 官网

<https://www.taobao.com/> - 8262条评论

由于该网站的robots.txt文件存在限制指令（限制搜索引擎抓取），系统无法提供该页面的内容  
描述 - [了解详情](#)

### 淘宝网触屏版

<https://h5.m.taobao.com/> - 74条评论

由于该网站的robots.txt文件存在限制指令（限制搜索引擎抓取），系统无法提供该页面的内容  
描述 - [了解详情](#)

### 淘宝\_百度百科



简介：淘宝网是亚太地区较大的网络零售、商圈，由阿里巴巴集团在2003年5月创立。淘宝网是中国深受欢迎的网购零售平台，拥有近5亿的注册用户数，每天有超过6000万的固定访客，同时每天的在线商品数已经超过了8亿件，平均每分钟售出4.8万件商品。截止2011年年底，淘宝网单日交易额峰值达到43.8亿元，创造270.8...

[发展沿革](#) [网站规模](#) [经济收入](#) [组织架构](#) [公司文化](#) [更多>>](#)

<https://baike.baidu.com/>

### 淘宝网 - 淘!我喜欢

<https://login.taobao.com/> - 8262条评论

由于该网站的robots.txt文件存在限制指令（限制搜索引擎抓取），系统无法提供该页面的内容  
描述 - [了解详情](#)

## 相关工具介绍

### HTTP协议

在开始讲解爬虫之前，我们稍微对HTTP（超文本传输协议）做一些回顾，因为我们在网页上看到的内容通常是浏览器执行HTML语言得到的结果，而HTTP就是传输HTML数据的协议。HTTP和其他很多应用级协议一样是构建在TCP（传输控制协议）之上的，它利用了TCP提供的可靠的传输服务实现了Web应用中的数据交换。按照维基百科上的介绍，设计HTTP最初的目的为了提供一种发布和接收HTML页面的方法，也就是说这个协议是浏览器和Web服务器之间传输的数据的载体。关于这个协议的详细信息以及目前的发展状况，大家可以阅读阮一峰老师的《HTTP 协议入门》、《互联网协议入门》系列以及《图解HTTPS协议》进行了解，下图是我在四川省网络通信技术重点实验室工作期间用开源协议分析工具Ethereal（抓包工具WireShark的前身）截取的访问百度首页时的HTTP请求和响应的报文（协议数据），由于Ethereal截取的是经过网络适配器的数据，因此可以清晰的看到从物理链路层到应用层的协议数据。

HTTP请求（请求行+请求头+空行+[消息体]）：

```

# Frame 4 (563 bytes on wire, 563 bytes captured)
# Ethernet II, Src: Elitegro_f9:5d:82 (00:11:5b:f9:5d:82), Dst: Cisco_50:14:71 (00:1b:2a:50:14:71)
# Internet Protocol, Src: 192.168.58.136 (192.168.58.136), Dst: 119.75.213.51 (119.75.213.51)
# Transmission Control Protocol, Src Port: voispeed-port (3541), Dst Port: http (80), seq: 1, Ack: 1, Len: 509
# Hypertext Transfer Protocol
# GET / HTTP/1.1\r\n
Accept: */*\r\n
Accept-Language: zh-cn\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727)\r\n
Accept-Encoding: gzip, deflate\r\n
Host: www.baidu.com\r\n
Connection: Keep-Alive\r\n
[truncated] Cookie: BAIDUID=72675E110453F51BEAC13B6277CE022F:FG=1; BDLOFONT=0; BDUSS=VFJenJQbGZuS21EM1dja3Vv
\r\n

```

HTTP响应（响应行+响应头+空行+消息体）：

```

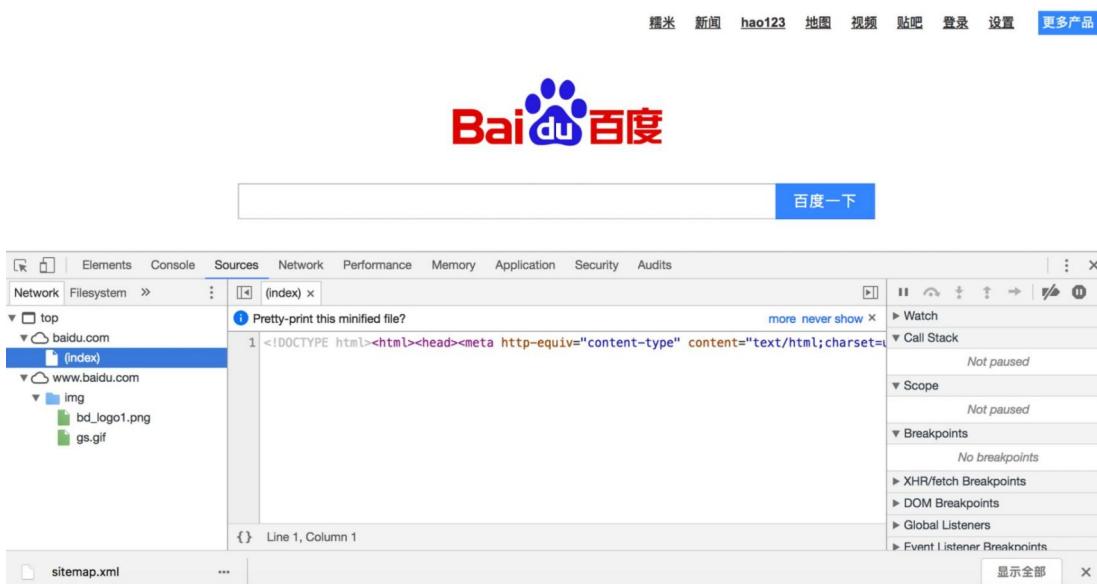
# Frame 7 (668 bytes on wire, 668 bytes captured)
# Ethernet II, Src: Cisco_50:14:71 (00:1b:2a:50:14:71), Dst: Elitegro_f9:5d:82 (00:11:5b:f9:5d:82)
# Internet Protocol, Src: 119.75.213.51 (119.75.213.51), Dst: 192.168.58.136 (192.168.58.136)
# Transmission Control Protocol, Src Port: http (80), Dst Port: voispeed-port (3541), seq: 1421, Ack: 510, Len: 509
# [Reassembled TCP Segments (2034 bytes): #6(1420), #7(614)]
# Hypertext Transfer Protocol
# HTTP/1.1 200 OK\r\n
Date: Thu, 10 Sep 2009 04:02:47 GMT\r\n
Server: BWS/1.0\r\n
Content-Length: 1826\r\n
Content-Type: text/html\r\n
Cache-Control: private\r\n
Expires: Thu, 10 Sep 2009 04:02:47 GMT\r\n
Content-Encoding: gzip\r\n
\r\n
Content-encoded entity body (gzip): 1826 bytes -> 3719 bytes
# Line-based text data: text/html

```

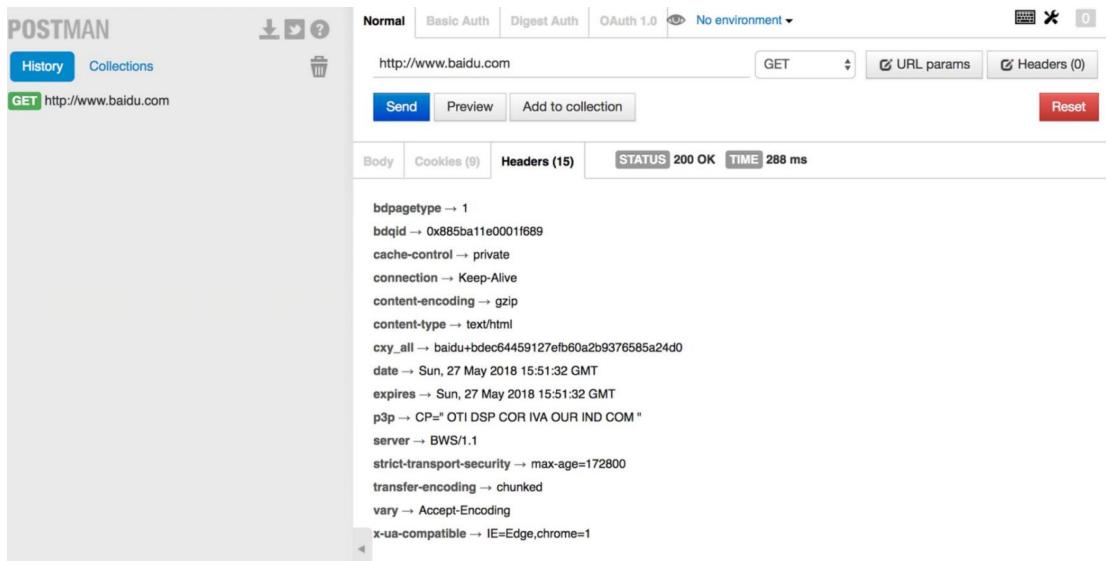
说明：但愿这两张如同泛黄的照片般的截图帮助你大概的了解到HTTP是一个怎样的协议。

## 相关工具

1. Chrome Developer Tools : 谷歌浏览器内置的开发者工具。



2. POSTMAN : 功能强大的网页调试与RESTful请求工具。



### 3. HTTPie : 命令行HTTP客户端。

```
$ http --header http://www.scu.edu.cn
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: private, max-age=600
Connection: Keep-Alive
Content-Encoding: gzip
Content-Language: zh-CN
Content-Length: 14403
Content-Type: text/html
Date: Sun, 27 May 2018 15:38:25 GMT
ETag: "e6ec-56d3032d70a32-gzip"
Expires: Sun, 27 May 2018 15:48:25 GMT
Keep-Alive: timeout=5, max=100
Last-Modified: Sun, 27 May 2018 13:44:22 GMT
Server: vWebServer
Vary: User-Agent,Accept-Encoding
X-Frame-Options: SAMEORIGIN
```

### 4. BuiltWith : 识别网站所用技术的工具。

```
>>> import builtwith
>>> builtwith.parse('http://www.bootcss.com/')
{'web-servers': ['Nginx'], 'font-scripts': ['Font Awesome'], 'javascript-framework':
>>>
>>> import ssl
>>> ssl._create_default_https_context = ssl._create_unverified_context
>>> builtwith.parse('https://www.jianshu.com/')
{'web-servers': ['Tengine'], 'web-frameworks': ['Twitter Bootstrap', 'Ruby on Rai']}
```

### 5. python-whois : 查询网站所有者的工具。

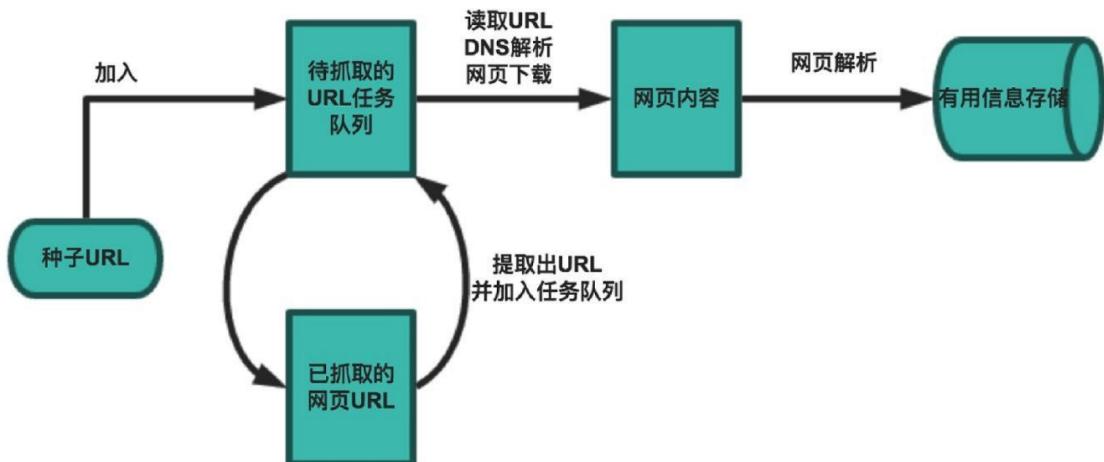
```
>>> import whois  
>>> whois.whois('baidu.com')  
{'domain_name': ['BAIDU.COM', 'baidu.com'], 'registrar': 'MarkMonitor, Inc.', 'whc
```

## 6. robotparser : 解析robots.txt的工具。

```
>>> from urllib import robotparser  
>>> parser = robotparser.RobotFileParser()  
>>> parser.set_url('https://www.taobao.com/robots.txt')  
>>> parser.read()  
>>> parser.can_fetch('Hellokitty', 'http://www.taobao.com/article')  
False  
>>> parser.can_fetch('Baiduspider', 'http://www.taobao.com/article')  
True  
>>> parser.can_fetch('Baiduspider', 'http://www.taobao.com/product')  
False
```

## 一个简单的爬虫

一个基本的爬虫通常分为数据采集（网页下载）、数据处理（网页解析）和数据存储（将有用的信息持久化）三个部分的内容，当然更为高级的爬虫在数据采集和处理时会使用并发编程或分布式技术，这就需要有调度器（安排线程或进程执行对应的任务）、后台管理程序（监控爬虫的工作状态以及检查数据抓取的结果）等的参与。



一般来说，爬虫的工作流程包括以下几个步骤：

1. 设定抓取目标（种子页面/起始页面）并获取网页。
2. 当服务器无法访问时，按照指定的重试次数尝试重新下载页面。
3. 在需要的时候设置用户代理或隐藏真实IP，否则可能无法访问页面。
4. 对获取的页面进行必要的解码操作然后抓取出需要的信息。

- 在获取的页面中通过某种方式（如正则表达式）抽取出页面中的链接信息。
  - 对链接进行进一步的处理（获取页面并重复上面的动作）。
  - 将有用的信息进行持久化以备后续的处理。

下面的例子给出了一个从“搜狐体育”上获取NBA新闻标题和链接的爬虫。

```

try:
    with conn.cursor() as cursor:
        url_list = [seed_url]
        # 通过下面的字典避免重复抓取并控制抓取深度
        visited_url_list = {seed_url: 0}
        while url_list:
            current_url = url_list.pop(0)
            depth = visited_url_list[current_url]
            if depth != max_depth:
                # 尝试用utf-8/gbk/gb2312三种字符集进行页面解码
                page_html = get_page_html(current_url, charsets=('utf-8', 'gbk', 'gb2312'))
                links_list = get_matched_parts(page_html, match_pattern)
                param_list = []
                for link in links_list:
                    if link not in visited_url_list:
                        visited_url_list[link] = depth + 1
                        page_html = get_page_html(link, charsets=('utf-8', 'gbk', 'gb2312'))
                        headings = get_matched_parts(page_html, r'<h1>(.*)<span>')
                        if headings:
                            param_list.append((headings[0], link))
                cursor.executemany('insert into tb_result values (default, %s, %s)', param_list)
            conn.commit()
except Error:
    pass
# logging.error('SQL:', error)
finally:
    conn.close()

def main():
    ssl._create_default_https_context = ssl._create_unverified_context
    start_crawl('http://sports.sohu.com/nba_a.shtml',
                r'<a[^>]+test=a\s[^>]*href=[\"\\'][^.?]*[\"\\"]',
                max_depth=2)

if __name__ == '__main__':
    main()

```

由于使用了MySQL实现持久化操作，所以要先启动MySQL服务器再运行该程序。

## 爬虫注意事项

通过上面的例子，我们对爬虫已经有了一个感性的认识，在编写爬虫时有以下一些注意事项：

1. 处理相对链接。有的时候我们从页面中获取的链接不是一个完整的绝对链接而是一个相对链接，这种情况下需要将其与URL前缀进行拼接（urllib.parse中的urljoin()函数可以完成此项操作）。

2. 设置代理服务。有些网站会限制访问的区域（例如美国的Netflix屏蔽了很多国家的访问），有些爬虫需要隐藏自己的身份，在这种情况下可以设置使用代理服务器，代理服务器有免费（如[西刺代理](#)、[快代理](#)）和付费两种（如[讯代理](#)、[阿布云代理](#)），付费的一般稳定性和可用性都更好，可以通过 `urllib.request` 中的 `ProxyHandler` 来为请求设置代理。
3. 限制下载速度。如果我们的爬虫获取网页的速度过快，可能就会面临被封禁或者产生“损害动产”的风险（这个可能会导致吃官司且败诉），可以在两次下载之间添加延时从而对爬虫进行限速。
4. 避免爬虫陷阱。有些网站会动态生成页面内容，这会导致产生无限多的页面（例如在线万年历通常会有无穷无尽的链接）。可以通过记录到达当前页面经过了多少个链接（链接深度）来解决该问题，当达到事先设定的最大深度时爬虫就不再像队列中添加该网页中的链接了。
5. SSL相关问题。在使用 `urlopen` 打开一个HTTPS链接时会验证一次SSL证书，如果不做出处理会产生错误提示“SSL: CERTIFICATE\_VERIFY\_FAILED”，可以通过以下两种方式加以解决：
  - 使用未经验证的上下文

```
import ssl

request = urllib.request.Request(url='...', headers={...})
context = ssl._create_unverified_context()
web_page = urllib.request.urlopen(request, context=context)
```

- 设置全局的取消证书验证

```
import ssl

ssl._create_default_https_context = ssl._create_unverified_context
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 02.数据采集和解析.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



171 lines (117 sloc) 5.49 KB

## 数据采集和解析

通过《[网络爬虫和相关工具](#)》一文，我们已经了解到了开发一个爬虫需要做的工作以及一些常见的问题，至此我们可以对爬虫开发需要做的工作以及相关的技术做一个简单的汇总，这其中可能会有一些我们之前没有使用过的第三方库，不过别担心，这些内容我们稍后都会一一讲到。

1. 下载数据 - `urllib / requests / aiohttp`。
2. 解析数据 - `re / lxml / BeautifulSoup ( bs4 ) / pyquery`。
3. 缓存和持久化 - `pymysql / sqlalchemy / peewee / redis / pymongo`。
4. 生成数字签名 - `hashlib`。
5. 序列化和压缩 - `pickle / json / zlib`。
6. 调度器 - 进程 ( `multiprocessing` ) / 线程 ( `threading` ) / 协程 ( `coroutine` ) 。

## HTML页面分析

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>首页</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p>这是一个神奇的网站！</p>
    <hr>
    <div>
      <h2>这是一个例子程序</h2>
      <p>静夜思</p>
      <p class="foo">床前明月光</p>
      <p id="bar">疑似地上霜</p>
      <p class="foo">举头望明月</p>
      <div><a href="http://www.baidu.com"><p>低头思故乡</p></a></div>
```

```
</div>
<a class="foo" href="http://www.qq.com">腾讯网</a>



<table>
  <tr>
    <th>姓名</th>
    <th>上场时间</th>
    <th>得分</th>
    <th>篮板</th>
    <th>助攻</th>
  </tr>
</table>
</body>
</html>
```

如果你对上面的代码并不感到陌生，那么你一定知道HTML页面通常由三部分构成，分别是用来承载内容的Tag（标签）、负责渲染页面的CSS（层叠样式表）以及控制交互式行为的JavaScript。通常，我们可以在浏览器的右键菜单中通过“查看网页源代码”的方式获取网页的代码并了解页面的结构；当然，我们也可以通过浏览器提供的开发人员工具来了解网页更多的信息。

## 使用requests获取页面

1. GET请求和POST请求。
  
  
  
  
2. URL参数和请求头。
  
  
  
  
3. 复杂的POST请求（文件上传）。
  
  
  
  
4. 操作Cookie。
  
  
  
  
5. 设置代理服务器。

说明：关于requests的详细用法可以参考它的[官方文档](#)。

## 四种采集方式

## 四种采集方式的比较

抓取方法	速度	使用难度	备注
正则表达式	快	困难	常用正则表达式 在线正则表达式测试
lxml	快	一般	需要安装C语言依赖库 唯一支持XML的解析器
Beautiful	较快/较慢 ( 取决于解析器 )	简单	
PyQuery	较快	简单	Python版的jQuery

说明：Beautiful的解析器包括：Python标准库（html.parser）、lxml的HTML解析器、lxml的XML解析器和html5lib。

## 使用正则表达式

如果你对正则表达式没有任何的概念，那么推荐先阅读[《正则表达式30分钟入门教程》](#)，然后再阅读我们之前讲解在Python中如何使用正则表达式一文。

## 使用XPath和lxml

### BeautifulSoup的使用

BeautifulSoup是一个可以从HTML或XML文件中提取数据的Python库。它能够通过你喜欢的转换器实现惯用的文档导航、查找、修改文档的方式。

#### 1. 遍历文档树

- 获取标签
- 获取标签属性
- 获取标签内容
- 获取子（孙）节点
- 获取父节点/祖先节点
- 获取兄弟节点

#### 2. 搜索树节点

- find / find\_all : 字符串、正则表达式、列表、True、函数或Lambda。
- select\_one / select : CSS选择器

说明：更多内容可以参考BeautifulSoup的[官方文档](#)。

## PyQuery的使用

pyquery相当于jQuery的Python实现，可以用于解析HTML网页。

## 实例 - 获取知乎发现上的问题链接

```
from urllib.parse import urljoin

import re
import requests

from bs4 import BeautifulSoup


def main():
    headers = {'user-agent': 'Baiduspider'}
    proxies = {
        'http': 'http://122.114.31.177:808'
    }
    base_url = 'https://www.zhihu.com/'
    seed_url = urljoin(base_url, 'explore')
    resp = requests.get(seed_url,
                         headers=headers,
                         proxies=proxies)
    soup = BeautifulSoup(resp.text, 'lxml')
    href_regex = re.compile(r'^/question')
    link_set = set()
    for a_tag in soup.find_all('a', {'href': href_regex}):
        if 'href' in a_tag.attrs:
            href = a_tag.attrs['href']
            full_url = urljoin(base_url, href)
            link_set.add(full_url)
    print('Total %d question pages found.' % len(link_set))

if __name__ == '__main__':
    main()
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 03.存储数据.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



67 lines (51 sloc) 2.93 KB

# 存储数据

## 存储海量数据

数据持久化的首选方案应该是关系型数据库，关系型数据库的产品很多，包括：Oracle、MySQL、SQLServer、PostgreSQL等。如果要存储海量的低价值数据，文档数据库也是不错的选择，MongoDB是文档数据库中的佼佼者，之前我们已经讲解过MongoDB的相关知识，在此不再进行赘述。

## 数据缓存

通过《[网络数据采集和解析](#)》一文，我们已经知道了如何从指定的页面中抓取数据，以及如何保存抓取的结果，但是我们没有考虑过这么一种情况，就是我们可能需要从已经抓取过的页面中提取出更多的数据，重新去下载这些页面对于规模不大的网站倒是问题也不大，但是如果能够把这些页面缓存起来，对应用的性能会有明显的改善。可以使用Redis来提供高速缓存服务，关于Redis的知识，我们在《[NoSQL入门](#)》一文中已经做过简要的介绍。

## 实例 - 缓存知乎发现上的链接和页面代码

```
from hashlib import sha1
from urllib.parse import urljoin

import pickle
import re
import requests
import zlib

from bs4 import BeautifulSoup
from redis import Redis

def main():
```

```
# 指定种子页面
base_url = 'https://www.zhihu.com/'
seed_url = urljoin(base_url, 'explore')
# 创建Redis客户端
client = Redis(host='1.2.3.4', port=6379, password='1qaz2wsx')
# 设置用户代理(否则访问会被拒绝)
headers = {'user-agent': 'Baiduspider'}
# 通过requests模块发送GET请求并指定用户代理
resp = requests.get(seed_url, headers=headers)
# 创建BeautifulSoup对象并指定使用lxml作为解析器
soup = BeautifulSoup(resp.text, 'lxml')
href_regex = re.compile(r'^/question')
# 将URL处理成SHA1摘要(长度固定更简短)
hasher_proto = sha1()
# 查找所有href属性以/question打头的a标签
for a_tag in soup.find_all('a', {'href': href_regex}):
    # 获取a标签的href属性值并组装完整的URL
    href = a_tag.attrs['href']
    full_url = urljoin(base_url, href)
    # 传入URL生成SHA1摘要
    hasher = hasher_proto.copy()
    hasher.update(full_url.encode('utf-8'))
    field_key = hasher.hexdigest()
    # 如果Redis的键'zhihu'对应的hash数据类型中没有URL的摘要就访问页面并缓存
    if not client.exists('zhihu', field_key):
        html_page = requests.get(full_url, headers=headers).text
        # 对页面进行序列化和压缩操作
        zipped_page = zlib.compress(pickle.dumps(html_page))
        # 使用hash数据类型保存URL摘要及其对应的页面代码
        client.hset('zhihu', field_key, zipped_page)
    # 显示总共缓存了多少个页面
    print('Total %d question pages found.' % client.hlen('zhihu'))

if __name__ == '__main__':
    main()
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 04.并发下载.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



376 lines (269 sloc) 13.2 KB

# 并发下载

## 多线程和多进程回顾

在前面的《进程和线程》一文中，我们已经对在Python中使用多进程和多线程实现并发编程进行了简明的讲解，在此我们补充几个知识点。

### threading.local类

使用线程时最不愿意遇到的情况就是多个线程竞争资源，在这种情况下为了保证资源状态的正确性，我们可能需要对资源进行加锁保护的处理，这一方面会导致程序失去并发性，另外如果多个线程竞争多个资源时，还有可能因为加锁方式的不当导致死锁。要解决多个线程竞争资源的问题，其中一个方案就是让每个线程都持有资源的副本（拷贝），这样每个线程可以操作自己所持有的资源，从而规避对资源的竞争。

要实现将资源和持有资源的线程进行绑定的操作，最简单的做法就是使用threading模块的local类，在网络爬虫开发中，就可以使用local类为每个线程绑定一个MySQL数据库连接或Redis客户端对象，这样通过线程可以直接获得这些资源，既解决了资源竞争的问题，又避免了在函数和方法调用时传递这些资源。具体的请参考本章多线程爬取“手机搜狐网”（Redis版）的实例代码。

### concurrent.futures模块

Python3.2带来了concurrent.futures模块，这个模块包含了线程池和进程池、管理并行编程任务、处理非确定性的执行流程、进程/线程同步等功能。关于这部分的内容推荐阅读《Python并行编程》。

### 分布式进程

使用多进程的时候，可以将进程部署在多个主机节点上，Python的 `multiprocessing` 模块不但支持多进程，其中 `managers` 子模块还支持把多进程部署到多个节点上。当然，要部署分布式进程，首先需要一个服务进程作为调度者，进程之间通过网络进行通信来实现对进程的控制和调度，由于 `managers` 模块已经对这些做出了很好的封装，因此在无需了解网络通信细节的前提下，就可以编写分布式多进程应用。具体的请参照本章分布式多进程爬取“手机搜狐网”的实例代码。

## 协程和异步I/O

### 协程的概念

协程（coroutine）通常又称之为微线程或纤程，它是相互协作的一组子程序（函数）。所谓相互协作指的是在执行函数A时，可以随时中断去执行函数B，然后又中断继续执行函数A。注意，这一过程并不是函数调用（因为没有调用语句），整个过程看似像多线程，然而协程只有一个线程执行。协程通过 `yield` 关键字和 `send()` 操作来转移执行权，协程之间不是调用者与被调用者的关系。

协程的优势在于以下两点：

1. 执行效率极高，因为子程序（函数）切换不是线程切换，由程序自身控制，没有切换线程的开销。
2. 不需要多线程的锁机制，因为只有一个线程，也不存在竞争资源的问题，当然也就不需要对资源加锁保护，因此执行效率高很多。

说明：协程适合处理的是I/O密集型任务，处理CPU密集型任务并不是它的长处，如果要提升CPU的利用率可以考虑“多进程+协程”的模式。

### 历史回顾

1. Python 2.2：第一次提出了生成器（最初称之为迭代器）的概念（PEP 255）。
2. Python 2.5：引入了将对象发送回暂停了的生成器这一特性即生成器的 `send()` 方法（PEP 342）。
3. Python 3.3：添加了 `yield from` 特性，允许从迭代器中返回任何值（注意生成器本身也是迭代器），这样我们就可以串联生成器并且重构出更好的生成器。
4. Python 3.4：引入 `asyncio.coroutine` 装饰器用来标记作为协程的函数，协程函数和 `asyncio` 及其事件循环一起使用，来实现异步I/O操作。
5. Python 3.5：引入了 `async` 和 `await`，可以使用 `async def` 来定义一个协程函数，这个函数中不能包含任何形式的 `yield` 语句，但是可以使用 `return` 或 `await` 从协程中返回值。

### 示例代码

1. 生成器 - 数据的生产者。

```

from time import sleep

# 倒计数生成器
def countdown(n):
    while n > 0:
        yield n
        n -= 1

def main():
    for num in countdown(5):
        print(f'Countdown: {num}')
        sleep(1)
    print('Countdown Over!')


if __name__ == '__main__':
    main()

```

生成器还可以叠加来组成生成器管道，代码如下所示。

```

# Fibonacci数生成器
def fib():
    a, b = 0, 1
    while True:
        a, b = b, a + b
        yield a

# 偶数生成器
def even(gen):
    for val in gen:
        if val % 2 == 0:
            yield val

def main():
    gen = even(fib())
    for _ in range(10):
        print(next(gen))

if __name__ == '__main__':
    main()

```

## 2. 协程 - 数据的消费者。

```
from time import sleep
```

```

# 生成器 - 数据生产者
def countdown_gen(n, consumer):
    consumer.send(None)
    while n > 0:
        consumer.send(n)
        n -= 1
    consumer.send(None)

# 协程 - 数据消费者
def countdown_con():
    while True:
        n = yield
        if n:
            print(f'Countdown {n}')
            sleep(1)
        else:
            print('Countdown Over!')

def main():
    countdown_gen(5, countdown_con())

if __name__ == '__main__':
    main()

```

说明：上面代码中countdown\_gen函数中的第1行consumer.send(None)是为了激活生成器，通俗的说就是让生成器执行到有yield关键字的地方挂起，当然也可以通过next(consumer)来达到同样的效果。如果不愿意每次都用这样的代码来“预激”生成器，可以写一个包装器来完成该操作，代码如下所示。

```

from functools import wraps

def coroutine(fn):

    @wraps(fn)
    def wrapper(*args, **kwargs):
        gen = fn(*args, **kwargs)
        next(gen)
        return gen

    return wrapper

```

这样就可以使用 @coroutine 装饰器对协程进行预激操作，不需要再写重复代码来激活协程。

### 3. 异步I/O - 非阻塞式I/O操作。

```

import asyncio

@asyncio.coroutine
def countdown(name, n):
    while n > 0:
        print(f'Countdown[{name}]: {n}')
        yield from asyncio.sleep(1)
        n -= 1

def main():
    loop = asyncio.get_event_loop()
    tasks = [
        countdown("A", 10), countdown("B", 5),
    ]
    loop.run_until_complete(asyncio.wait(tasks))
    loop.close()

if __name__ == '__main__':
    main()

```

#### 4. async 和 await。

```

import asyncio
import aiohttp

async def download(url):
    print('Fetch:', url)
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as resp:
            print(url, '--->', resp.status)
            print(url, '--->', resp.cookies)
            print('\n\n', await resp.text())

def main():
    loop = asyncio.get_event_loop()
    urls = [
        'https://www.baidu.com',
        'http://www.sohu.com/',
        'http://www.sina.com.cn/',
        'https://www.taobao.com/',
        'https://www.jd.com/'
    ]
    tasks = [download(url) for url in urls]
    loop.run_until_complete(asyncio.wait(tasks))
    loop.close()

```

```
if __name__ == '__main__':
    main()
```

上面的代码使用了[AIOHTTP](#)这个非常著名的第三方库，它实现了HTTP客户端和HTTP服务器的功能，对异步操作提供了非常好的支持，有兴趣可以阅读它的[官方文档](#)。

## 实例 - 多线程爬取“手机搜狐网”所有页面

下面我们把之前讲的所有知识结合起来，用面向对象的方式实现一个爬取“手机搜狐网”的多线程爬虫。

```
import pickle
import zlib
from enum import Enum, unique
from hashlib import sha1
from random import random
from threading import Thread, current_thread, local
from time import sleep
from urllib.parse import urlparse

import pymongo
import redis
import requests
from bs4 import BeautifulSoup
from bson import Binary

@unique
class SpiderStatus(Enum):
    IDLE = 0
    WORKING = 1

def decode_page(page_bytes, charsets=('utf-8',)):
    page_html = None
    for charset in charsets:
        try:
            page_html = page_bytes.decode(charset)
            break
        except UnicodeDecodeError:
            pass
    return page_html

class Retry(object):

    def __init__(self, *, retry_times=3,
                 wait_secs=5, errors=(Exception, )):
        self.retry_times = retry_times
        self.wait_secs = wait_secs
        self.errors = errors
```

```

def __call__(self, fn):

    def wrapper(*args, **kwargs):
        for _ in range(self.retry_times):
            try:
                return fn(*args, **kwargs)
            except self.errors as e:
                print(e)
                sleep((random() + 1) * self.wait_secs)
        return None

    return wrapper


class Spider(object):

    def __init__(self):
        self.status = SpiderStatus.IDLE

    @Retry()
    def fetch(self, current_url, *, charsets=('utf-8', ), user_agent=None, proxies=None):
        thread_name = current_thread().name
        print(f'[{thread_name}]: {current_url}')
        headers = {'user-agent': user_agent} if user_agent else {}
        resp = requests.get(current_url,
                             headers=headers, proxies=proxies)
        return decode_page(resp.content, charsets) \
            if resp.status_code == 200 else None

    def parse(self, html_page, *, domain='m.sohu.com'):
        soup = BeautifulSoup(html_page, 'lxml')
        for a_tag in soup.body.select('a[href]'):
            parser = urlparse(a_tag.attrs['href'])
            scheme = parser.scheme or 'http'
            netloc = parser.netloc or domain
            if scheme != 'javascript' and netloc == domain:
                path = parser.path
                query = '?' + parser.query if parser.query else ''
                full_url = f'{scheme}://{netloc}{path}{query}'
                redis_client = thread_local.redis_client
                if not redis_client.sismember('visited_urls', full_url):
                    redis_client.rpush('m_sohu_task', full_url)

    def extract(self, html_page):
        pass

    def store(self, data_dict):
        # redis_client = thread_local.redis_client
        # mongo_db = thread_local.mongo_db
        pass

class SpiderThread(Thread):

```

```

def __init__(self, name, spider):
    super().__init__(name=name, daemon=True)
    self.spider = spider

def run(self):
    redis_client = redis.Redis(host='1.2.3.4', port=6379, password='1qaz2wsx')
    mongo_client = pymongo.MongoClient(host='1.2.3.4', port=27017)
    thread_local.redis_client = redis_client
    thread_local.mongo_db = mongo_client.msohu
    while True:
        current_url = redis_client.lpop('m_sohu_task')
        while not current_url:
            current_url = redis_client.lpop('m_sohu_task')
        self.spider.status = SpiderStatus.WORKING
        current_url = current_url.decode('utf-8')
        if not redis_client.sismember('visited_urls', current_url):
            redis_client.sadd('visited_urls', current_url)
            html_page = self.spider.fetch(current_url)
            if html_page not in [None, '']:
                hasher = hasher_proto.copy()
                hasher.update(current_url.encode('utf-8'))
                doc_id = hasher.hexdigest()
                sohu_data_coll = mongo_client.msohu.webpages
                if not sohu_data_coll.find_one({'_id': doc_id}):
                    sohu_data_coll.insert_one({
                        '_id': doc_id,
                        'url': current_url,
                        'page': Binary(zlib.compress(pickle.dumps(html_page)))
                    })
                self.spider.parse(html_page)
            self.spider.status = SpiderStatus.IDLE

def is_any_alive(spider_threads):
    return any([spider_thread.spider.status == SpiderStatus.WORKING
               for spider_thread in spider_threads])

thread_local = local()
hasher_proto = sha1()

def main():
    redis_client = redis.Redis(host='1.2.3.4', port=6379, password='1qaz2wsx')
    if not redis_client.exists('m_sohu_task'):
        redis_client.rpush('m_sohu_task', 'http://m.sohu.com/')

    spider_threads = [SpiderThread('thread-%d' % i, Spider())
                      for i in range(10)]
    for spider_thread in spider_threads:
        spider_thread.start()

    while redis_client.exists('m_sohu_task') or is_any_alive(spider_threads):
        sleep(5)

```

```
print('Over!')  
  
if __name__ == '__main__':  
    main()
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 05.解析动态内容.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



83 lines (51 sloc) 5.09 KB

## 解析动态内容

根据权威机构发布的全球互联网可访问性审计报告，全球约有四分之三的网站其内容或部分内容是通过JavaScript动态生成的，这就意味着在浏览器窗口中“查看网页源代码”时无法在HTML代码中找到这些内容，也就是说我们之前用的抓取数据的方式无法正常运转了。解决这样的问题基本上有两种方案，一是JavaScript逆向工程；另一种是渲染JavaScript获得渲染后的内容。

### JavaScript逆向工程

下面我们以“360图片”网站为例，说明什么是JavaScript逆向工程。其实所谓的JavaScript逆向工程就是找到通过Ajax技术动态获取数据的接口。在浏览器中输入<http://image.so.com/z?ch=beauty>就可以打开“360图片”的“美女”版块，如下图所示。

The screenshot shows the homepage of the 360 Images website. At the top, there is a search bar with a camera icon and a green 'Search' button. Below the search bar, there is a navigation menu with links: 首页 (Home), 美女 (Beautiful Women), 图说世界 (Photo Stories), 壁纸 (Wallpapers), 搞笑 (Funny), 图解电影 (Movie Explanations), 旅游 (Travel), 艺术 (Art), 汽车 (Cars), 摄影 (Photography), 美食 (Food), 家居 (Home), and 更多 (More). The 'Beautiful Women' link is highlighted with a green background. Below the menu, there is a secondary navigation bar with categories: 萌女 (Cute Girls), 明星 (Celebrities), 粉嫩 (Pink), 车模 (Car Models), 街拍 (Street Photography), 婚纱 (Wedding Dresses), 时装秀 (Fashion Shows), showgirl (Showgirls), cosplay (Cosplay), and 主播秀 (Streaming Shows). The '萌女' link is also highlighted with a green background. At the bottom of the page, there are three large image thumbnails. The first thumbnail on the left shows a woman sitting in a chair. The middle thumbnail shows a woman in a light blue dress. The third thumbnail on the right shows a woman sitting next to a white unicorn statue. The overall layout is clean and modern, typical of a search engine's image results page.

但是当我们在浏览器中通过右键菜单“显示网页源代码”的时候，居然惊奇的发现页面的HTML代码中连一个 `<img>` 标签都没有，那么我们看到的图片是怎么显示出来的呢？原来所有的图片都是通过JavaScript动态加载的，而在浏览器的“开发人员工具”的“网络”中可以找到获取这些图片数据的网络API接口，如下图所示。

The screenshot shows the Network tab in the Chrome DevTools developer tools. A specific request for 'zj?ch=beauty&t1=595&listtype=new&sn=30&l...' is selected. The response is a JSON object with the following structure:

```
zj?ch=beauty&t1=595&listtype=new&sn=30&l...  
  {end: false, count: 30, lastid: 60,...}  
    count: 30  
    end: false  
    lastid: 60  
    list: [{id: "ff276e8056f58896e7cf92505a312749", imageid: "cc48190e57100864abc4d55a2c4104c6",...},...]  
      id: "ff276e8056f58896e7cf92505a312749"  
      imageid: "cc48190e57100864abc4d55a2c4104c6",...  
      cover_height: 586  
      cover_imgeurl: "http://i1.umei.cc/uploads/tu/201701/798/rfzhiltiisv.jpg"  
      cover_width: 880  
      dsptime: ""  
      group_title: "台湾美女Kila户外唯美写真"  
      grpseq: 1  
      id: "ff276e8056f58896e7cf92505a312749"  
      imageid: "cc48190e57100864abc4d55a2c4104c6"  
      index: 31
```

At the bottom left, it says '1 / 32 requests | 4.1 KB / 4.1 KB transferred'.

那么结论就很简单了，只要我们找到了这些网络API接口，那么就能通过这些接口获取到数据，当然实际开发的时候可能还要对这些接口的参数以及接口返回的数据进行分析，了解每个参数的意义以及返回的JSON数据的格式，这样才能在我们的爬虫中使用这些数据。

关于如何从网络API中获取JSON格式的数据并提取出我们需要的内容，在之前的[《文件和异常》](#)一文中已经讲解过了，这里不再进行赘述。

## 使用Selenium

尽管很多网站对自己的网络API接口进行了保护，增加了获取数据的难度，但是只要经过足够的努力，绝大多数还是可以被逆向工程的，但是在实际开发中，我们可以通过浏览器渲染引擎来避免这些繁琐的工作，WebKit就是一个利用的渲染引擎。

WebKit的代码始于1998年的KHTML项目，当时它是Konqueror浏览器的渲染引擎。2001年，苹果公司从这个项目的代码中衍生出了WebKit并应用于Safari浏览器，早期的Chrome浏览器也使用了该内核。在Python中，我们可以通过Qt框架获得WebKit引擎并使用它来渲染页面获得动态内容，关于这个内容请大家自行阅读[《爬虫技术:动态页面抓取超级指南》](#)一文。

如果没有打算用上面所说的方式来渲染页面并获得动态内容，其实还有一种替代方案就是使用自动化测试工具Selenium，它提供了浏览器自动化的API接口，这样就可以通过操控浏览器来获取动态内容。首先可以使用pip来安装Selenium。

```
pip3 install selenium
```

下面以“阿里V任务”的“直播服务”为例，来演示如何使用Selenium获取到动态内容并抓取主播图片。

```
import requests

from bs4 import BeautifulSoup

def main():
    resp = requests.get('https://v.taobao.com/v/content/live?catetype=704&from=taonvlang')
    soup = BeautifulSoup(resp.text, 'lxml')
    for img_tag in soup.select('img[src]'):
        print(img_tag.attrs['src'])

if __name__ == '__main__':
    main()
```

运行上面的程序会发现没有任何的输出，因为页面的HTML代码上根本找不到 `<img>` 标签。接下来我们使用Selenium来获取到页面上的动态内容，再提取主播图片。

```
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

def main():
    driver = webdriver.Chrome()
    driver.get('https://v.taobao.com/v/content/live?catetype=704&from=taonvlang')
    soup = BeautifulSoup(driver.page_source, 'lxml')
    for img_tag in soup.body.select('img[src]'):
        print(img_tag.attrs['src'])

if __name__ == '__main__':
    main()
```

在上面的程序中，我们通过Selenium实现对Chrome浏览器的操控，如果要操控其他的浏览器，可以创对应的浏览器对象，例如Firefox、IE等。运行上面的程序，如果看到如下所示的错误提示，那是说明我们还没有将Chrome浏览器的驱动添加到PATH环境变量中，也没有在程序中指定Chrome浏览器驱动所在的位置。

```
selenium.common.exceptions.WebDriverException: Message: 'chromedriver' executable need
```

为了解决上面的问题，可以到Selenium的[官方网站](#)找到浏览器驱动的下载链接并下载需要的驱动，在Linux或macOS系统下可以通过下面的命令来设置PATH环境变量，Windows下配置环境变量也非常简单，不清楚的可以自行了解。

```
export PATH=$PATH:/Users/Hao/Downloads/Tools/chromedriver/
```

其中 /Users/Hao/Downloads/Tools/chromedriver/ 就是chromedriver所在的路径。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 06.表单交互和验证码处理.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



35 lines (13 sloc) 1.04 KB

# 表单交互和验证码处理

## 提交表单

### 手动提交

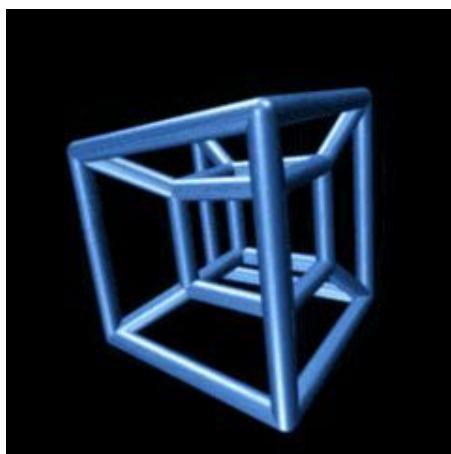
### 自动提交

## 验证码处理

### 加载验证码

### 光学字符识别

光学字符识别（OCR）是从图像中抽取文本的工具，可以应用于公安、电信、物流、金融等诸多行业，例如识别车牌、身份证扫描识别、名片信息提取等。在爬虫开发中，如果遭遇了有文字验证码的表单，就可以利用OCR来进行验证码处理。Tesseract-OCR引擎最初是由惠普公司开发的光学字符识别系统，目前发布在Github上，由Google赞助开发。



## 改善OCR

### 处理更复杂的验证码

很多网站为了分别出提供验证码的是人还是机器使用了更为复杂的验证码，例如拼图验证码、点触验证码、九宫格验证码等。关于这方面的知识，在崔庆才同学的[《Python 3 网络爬虫开发实战》](#)有较为详细的讲解，有兴趣的可以购买阅读。

### 验证码处理服务

## Python-100-Days / Day66-75 / 07.Scrapy入门.md

Fetching contributors...

Cannot retrieve contributors at this time.

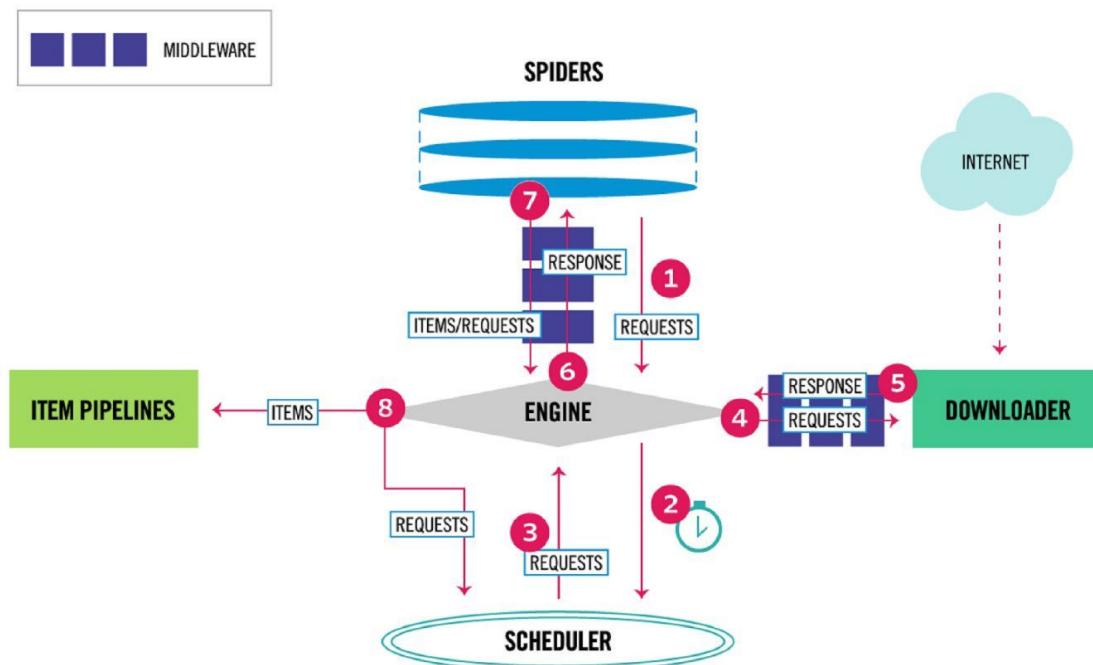
[Raw](#) [Blame](#) [History](#)

378 lines (283 sloc) 15.9 KB

# Scrapy爬虫框架入门

## Scrapy概述

Scrapy是Python开发的一个非常流行的网络爬虫框架，可以用来抓取Web站点并从页面中提取结构化的数据，被广泛的用于数据挖掘、数据监测和自动化测试等领域。下图展示了Scrapy的基本架构，其中包含了主要组件和系统的数据处理流程（图中带数字的红色箭头）。



## 组件

1. Scrapy引擎（Engine）：Scrapy引擎是用来控制整个系统的数据处理流程。

2. 调度器 ( Scheduler ) : 调度器从Scrapy引擎接受请求并排序列入队列，并在Scrapy引擎发出请求后返还给它们。
3. 下载器 ( Downloader ) : 下载器的主要职责是抓取网页并将网页内容返还给蜘蛛 ( Spiders )。
4. 蜘蛛 ( Spiders ) : 蜘蛛是有Scrapy用户自定义的用来解析网页并抓取特定URL返回的内容的类，每个蜘蛛都能处理一个域名或一组域名，简单的说就是用来定义特定网站的抓取和解析规则。
5. 条目管道 ( Item Pipeline ) : 条目管道的主要责任是负责处理有蜘蛛从网页中抽取的数据条目，它的主要任务是清理、验证和存储数据。当页面被蜘蛛解析后，将被发送到条目管道，并经过几个特定的次序处理数据。每个条目管道组件都是一个Python类，它们获取了数据条目并执行对数据条目进行处理的方法，同时还需要确定是否需要在条目管道中继续执行下一步或是直接丢弃掉不处理。条目管道通常执行的任务有：清理HTML数据、验证解析到的数据（检查条目是否包含必要的字段）、检查是不是重复数据（如果重复就丢弃）、将解析到的数据存储到数据库（关系型数据库或NoSQL数据库）中。
6. 中间件 ( Middlewares ) : 中间件是介于Scrapy引擎和其他组件之间的一个钩子框架，主要是为了提供自定义的代码来拓展Scrapy的功能，包括下载器中间件和蜘蛛中间件。

## 数据处理流程

Scrapy的整个数据处理流程由Scrapy引擎进行控制，通常的运转流程包括以下的步骤：

1. 引擎询问蜘蛛需要处理哪个网站，并让蜘蛛将第一个需要处理的URL交给它。
2. 引擎让调度器将需要处理的URL放在队列中。
3. 引擎从调度那获取接下来进行爬取的页面。
4. 调度将下一个爬取的URL返回给引擎，引擎将它通过下载中间件发送到下载器。
5. 当网页被下载器下载完成以后，响应内容通过下载中间件被发送到引擎；如果下载失败了，引擎会通知调度器记录这个URL，待会再重新下载。
6. 引擎收到下载器的响应并将它通过蜘蛛中间件发送到蜘蛛进行处理。
7. 蜘蛛处理响应并返回爬取到的数据条目，此外还要将需要跟进的新的URL发送给引擎。
8. 引擎将抓取到的数据条目送入条目管道，把新的URL发送给调度器放入队列中。

上述操作中的2-8步会一直重复直到调度器中没有需要请求的URL，爬虫停止工作。

## 安装和使用Scrapy

可以先创建虚拟环境并在虚拟环境下使用pip安装scrapy。

项目的目录结构如下图所示。

```
(venv) $ tree
.
├── scrapy.cfg
└── douban
    ├── spiders
    │   ├── __init__.py
    │   ├── __pycache__
    │   │   __init__.py
    │   │   __pycache__
    │   ├── middlewares.py
    │   ├── settings.py
    │   ├── items.py
    │   └── pipelines.py
```

说明：Windows系统的命令行提示符下有tree命令，但是Linux和MacOS的终端是没有tree命令的，可以用下面给出的命令来定义tree命令，其实是对find命令进行了定制并别名为tree。

```
alias tree="find . -print | sed -e 's;[^/]*/;|____;g;s;____|; |;g'"
```

Linux系统也可以通过yum或其他的包管理工具来安装tree。

```
yum install tree
```

根据刚才描述的数据处理流程，基本上需要我们做的有以下几件事情：

1. 在items.py文件中定义字段，这些字段用来保存数据，方便后续的操作。

```
# -*- coding: utf-8 -*-
# Define here the models for your scraped items
#
# See documentation in:
# https://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class DoubanItem(scrapy.Item):

    name = scrapy.Field()
    year = scrapy.Field()
    score = scrapy.Field()
    director = scrapy.Field()
    classification = scrapy.Field()
    actor = scrapy.Field()
```

## 2. 在spiders文件夹中编写自己的爬虫。

```
(venv) $ scrapy genspider movie movie.douban.com --template=crawl

# -*- coding: utf-8 -*-
import scrapy
from scrapy.selector import Selector
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

from douban.items import DoubanItem


class MovieSpider(CrawlSpider):
    name = 'movie'
    allowed_domains = ['movie.douban.com']
    start_urls = ['https://movie.douban.com/top250']
    rules = (
        Rule(LinkExtractor(allow=(r'https://movie.douban.com/top250\?start=\d+.*')),
        Rule(LinkExtractor(allow=(r'https://movie.douban.com/subject/\d+')), callback='parse_item')
    )

    def parse_item(self, response):
        sel = Selector(response)
        item = DoubanItem()
        item['name'] = sel.xpath('//*[@id="content"]/h1/span[1]/text()').extract()
        item['year'] = sel.xpath('//*[@id="content"]/h1/span[2]/text()').re(r'\((\d+)')
        item['score'] = sel.xpath('//*[@id="interest_sect1"]/div/p[1]/strong/text()')
        item['director'] = sel.xpath('//*[@id="info"]/span[1]/a/text()').extract()
        item['classification'] = sel.xpath('//span[@property="v:genre"]/text()').extract()
        item['actor'] = sel.xpath('//*[@id="info"]/span[3]/a[1]/text()').extract()
        return item
```

说明：上面我们通过Scrapy提供的爬虫模板创建了Spider，其中的rules中的LinkExtractor对象会自动完成对新的链接的解析，该对象中有一个名为extract\_link的回调方法。Scrapy支持用XPath语法和CSS选择器进行数据解析，对应的方法分别是xpath和css，上面我们使用了XPath语法对页面进行解析，如果不熟悉XPath语法可以看看后面的补充说明。

到这里，我们已经可以通过下面的命令让爬虫运转起来。

```
(venv)$ scrapy crawl movie
```

可以在控制台看到爬取到的数据，如果想将这些数据保存到文件中，可以通过 -o 参数来指定文件名，Scrapy支持我们将爬取到的数据导出成JSON、CSV、XML、pickle、marshal等格式。

```
(venv)$ scrapy crawl moive -o result.json
```

3. 在pipelines.py中完成对数据进行持久化的操作。

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: https://doc.scrapy.org/en/latest/topics/item-pipeline.html
import pymongo

from scrapy.exceptions import DropItem
from scrapy.conf import settings
from scrapy import log


class DoubanPipeline(object):

    def __init__(self):
        connection = pymongo.MongoClient(settings['MONGODB_SERVER'], settings['MON
db = connection[settings['MONGODB_DB']]
self.collection = db[settings['MONGODB_COLLECTION']]

    def process_item(self, item, spider):
        #Remove invalid data
        valid = True
        for data in item:
            if not data:
                valid = False
            raise DropItem("Missing %s of blogpost from %s" %(data, item['url']))
        if valid:
            #Insert data into database
            new_moive=[{
                "name":item['name'][0],
                "year":item['year'][0],
                "score":item['score'],
                "director":item['director'],
                "classification":item['classification'],
                "actor":item['actor']
            }]
            self.collection.insert(new_moive)
            log.msg("Item wrote to MongoDB database %s/%s" %
(settings['MONGODB_DB'], settings['MONGODB_COLLECTION']),
            level=log.DEBUG, spider=spider)
        return item
```

利用Pipeline我们可以完成以下操作：

- 清理HTML数据，验证爬取的数据。

- 丢弃重复的不必要的内容。
- 将爬取的结果进行持久化操作。

#### 4. 修改settings.py文件对项目进行配置。

```
# -*- coding: utf-8 -*-

# Scrapy settings for douban project
#
# For simplicity, this file contains only settings considered important or
# commonly used. You can find more settings consulting the documentation:
#
#     https://doc.scrapy.org/en/latest/topics/settings.html
#     https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
#     https://doc.scrapy.org/en/latest/topics/spider-middleware.html

BOT_NAME = 'douban'

SPIDER_MODULES = ['douban.spiders']
NEWSPIDER_MODULE = 'douban.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent
USER_AGENT = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/20.0.1062.52 Safari/536.5'

# Obey robots.txt rules
ROBOTSTXT_OBEY = True

# Configure maximum concurrent requests performed by Scrapy (default: 16)
# CONCURRENT_REQUESTS = 32

# Configure a delay for requests for the same website (default: 0)
# See https://doc.scrapy.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
DOWNLOAD_DELAY = 3
RANDOMIZE_DOWNLOAD_DELAY = True
# The download delay setting will honor only one of:
# CONCURRENT_REQUESTS_PER_DOMAIN = 16
# CONCURRENT_REQUESTS_PER_IP = 16

# Disable cookies (enabled by default)
COOKIES_ENABLED = True

MONGODB_SERVER = '120.77.222.217'
MONGODB_PORT = 27017
MONGODB_DB = 'douban'
MONGODB_COLLECTION = 'movie'

# Disable Telnet Console (enabled by default)
# TELNETCONSOLE_ENABLED = False

# Override the default request headers:
# DEFAULT_REQUEST_HEADERS = {
```

```
#   'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
#   'Accept-Language': 'en',
# }

# Enable or disable spider middlewares
# See https://doc.scrapy.org/en/latest/topics/spider-middleware.html
# SPIDER_MIDDLEWARES = {
#     'douban.middlewares.DoubanSpiderMiddleware': 543,
# }

# Enable or disable downloader middlewares
# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
# DOWNLOADER_MIDDLEWARES = {
#     'douban.middlewares.DoubanDownloaderMiddleware': 543,
# }

# Enable or disable extensions
# See https://doc.scrapy.org/en/latest/topics/extensions.html
# EXTENSIONS = {
#     'scrapy.extensions.telnet.TelnetConsole': None,
# }

# Configure item pipelines
# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'douban.pipelines.DoubanPipeline': 400,
}

LOG_LEVEL = 'DEBUG'

# Enable and configure the AutoThrottle extension (disabled by default)
# See https://doc.scrapy.org/en/latest/topics/autothrottle.html
#AUTOTHROTTLE_ENABLED = True
# The initial download delay
#AUTOTHROTTLE_START_DELAY = 5
# The maximum download delay to be set in case of high latencies
#AUTOTHROTTLE_MAX_DELAY = 60
# The average number of requests Scrapy should be sending in parallel to
# each remote server
#AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0
# Enable showing throttling stats for every response received:
#AUTOTHROTTLE_DEBUG = False

# Enable and configure HTTP caching (disabled by default)
# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache
HTTPCACHE_ENABLED = True
HTTPCACHE_EXPIRATION_SECS = 0
HTTPCACHE_DIR = 'httpcache'
HTTPCACHE_IGNORE_HTTP_CODES = []
HTTPCACHE_STORAGE = 'scrapy.extensions.httpcache.FilesystemCacheStorage'
```

## 补充说明

## XPath语法

1. XPath路径表达式：XPath使用路径表达式来选取XML文档中的节点或者节点集。
2. XPath节点：元素、属性、文本、命名空间、处理指令、注释、根节点。
3. XPath语法。（注：下面的例子来自于[菜鸟教程网站的XPath教程](#)。）

XML文件。

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

    <book>
        <title lang="eng">Harry Potter</title>
        <price>29.99</price>
    </book>

    <book>
        <title lang="eng">Learning XML</title>
        <price>39.95</price>
    </book>

</bookstore>
```

XPath语法。

路径表达式	结果
bookstore	选取 bookstore 元素的所有子节点。
/bookstore	选取根元素 bookstore。注释：假如路径起始于正斜杠( / )，则此路径始终代表到某元素的绝对路径！
bookstore/book	选取属于 bookstore 的子元素的所有 book 元素。
//book	选取所有 book 子元素，而不管它们在文档中的位置。
bookstore//book	选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。
//@lang	选取名为 lang 的所有属性。

XPath谓词。

路径表达式	结果
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素。

路径表达式	结果
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素。
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素。
/bookstore/book[position()<3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。
//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

通配符用法。

路径表达式	结果
/bookstore/*	选取 bookstore 元素的所有子元素。
/*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。

选取多个路径。

路径表达式	结果
//book/title   //book/price	选取 book 元素的所有 title 和 price 元素。
//title   //price	选取文档中的所有 title 和 price 元素。
/bookstore/book/title   //price	选取属于 bookstore 元素的 book 元素的所有 title 元素，以及文档中所有的 price 元素。

在Chrome浏览器中查看元素XPath语法

豆瓣 读书 电影 音乐 同城 小组 阅读 FM 时间 市集 更多

# 豆瓣电影

搜索电影、电视剧、综艺、影人

影讯&购票 选电影 电视剧 排行榜 分类 影评 2017年度榜单 2017观影报告

span | 517.16×30 50

## 肖申克的救赎 The Shawshank Redemption (1994)

导演: 弗兰克·德拉邦特  
编剧: 弗兰克·德拉邦特 / 斯蒂芬·金  
主演: 蒂姆·罗宾斯 / 摩根·弗里曼 / 鲍勃·冈顿 / 威廉姆·赛德勒 / 克兰西·布朗 / 更多...  
类型: 剧情 / 犯罪  
制片国家/地区: 美国

豆瓣评分  
**9.6** ★★★★★ 1041463人评价  
5星 82.9%  
4星 15.1%  
3星 1.8%

在聊  
优酷  
搜狐  
爱奇

Elements Console Sources Network Performance Memory Application Security Audits

Copy

Edit text  
Edit as HTML  
Delete element  
Cut element  
Copy element  
Paste element  
Copy outerHTML  
Copy selector  
Collapse children  
Copy XPath

```
<div id="content">
  <!-- top 250 begin -->
  <div class="top250">...</div>
  <!-- top 250 end -->
  <div id="dale_movie_subject_top_icon" ad-status="loaded"></div>
  <h1>
    <span property="v:itemreviewed">肖申克的救赎 The Shawshank</span>
    <span class="year">(1994)</span>
  </h1>
  <div class="grid-16-8 clearfix">...</div>
</div>
<div id="footer">...</div>
</div>
<script type="text/javascript" src="https://img3.doubanio.com/html.ua-mac.ua-webkit/body/div#content/h1/span">
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 08.Scrapy高级应用.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



33 lines (17 sloc) 1.59 KB

# Scrapy爬虫框架高级应用

## Spider的用法

在Scrapy框架中，我们自定义的蜘蛛都继承自scrapy.spiders.Spider，这个类有一系列的属性和方法，具体如下所示：

1. name：爬虫的名字。
2. allowed\_domains：允许爬取的域名，不在此范围的链接不会被跟进爬取。
3. start\_urls：起始URL列表，当我们没有重写start\_requests()方法时，就会从这个列表开始爬取。
4. custom\_settings：用来存放蜘蛛专属配置的字典，这里的设置会覆盖全局的设置。
5. crawler：由from\_crawler()方法设置的和蜘蛛对应的Crawler对象，Crawler对象包含了很多项目组件，利用它我们可以获取项目的配置信息，如调用crawler.settings.get()方法。
6. settings：用来获取爬虫全局设置的变量。
7. start\_requests()：此方法用于生成初始请求，它返回一个可迭代对象。该方法默认是使用GET请求访问起始URL，如果起始URL需要使用POST请求来访问就必须重写这个方法。
8. parse()：当Response没有指定回调函数时，该方法就会被调用，它负责处理Response对象并返回结果，从中提取出需要的数据和后续的请求，该方法需要返回类型为Request或Item的可迭代对象（生成器当前也包含在其中，因此根据实际需要可以用return或yield来产生返回值）。
9. closed()：当蜘蛛关闭时，该方法会被调用，通常用来做一些释放资源的善后操作。

## 中间件的应用

### 下载中间件

[蜘蛛中间件](#)

[Scrapy对接Selenium](#)

[Scrapy部署到Docker](#)

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 09.Scrapy分布式实现.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



31 lines (22 sloc) 834 Bytes

# Scrapy爬虫框架分布式实现

## 分布式爬虫原理

### Scrapy分布式实现

1. 安装Scrapy-Redis。
2. 配置Redis服务器。
3. 修改配置文件。
  - SCHEDULER = 'scrapy\_redis.scheduler.Scheduler'
  - DUPEFILTER\_CLASS = 'scrapy\_redis.dupefilter.RFPDupeFilter'
  - REDIS\_HOST = '1.2.3.4'
  - REDIS\_PORT = 6379
  - REDIS\_PASSWORD = '1qaz2wsx'
  - SCHEDULER\_QUEUE\_CLASS = 'scrapy\_redis.queue.FifoQueue'
  - SCHEDULER\_PERSIST = True ( 通过持久化支持接续爬取 )
  - SCHEDULER\_FLUSH\_ON\_START = True ( 每次启动时重新爬取 )

## Scrapyd分布式部署

1. 安装Scrapyd
2. 修改配置文件
  - mkdir /etc/scrapyd
  - vim /etc/scrapyd/scrapyd.conf
3. 安装Scrapyd-Client
  - 将项目打包成Egg文件。
  - 将打包的Egg文件通过addversion.json接口部署到Scrapyd上。

Branch: master ▾

[Find file](#) [Copy path](#)

## [Python-100-Days / Day66-75 / 10.爬虫项目实战.md](#)

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



2 lines (1 sloc) 22 Bytes

# 爬虫项目实战

Branch: master ▾

Find file Copy path

## Python-100-Days / Day66-75 / 常见反爬策略及应对方案.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



111 lines (91 sloc) 4.02 KB

# 常见反爬策略及应对方案

## 1. 构造合理的HTTP请求头。

- Accept
- User-Agent - 三方库fake-useragent

```
from fake_useragent import UserAgent
ua = UserAgent()

ua.ie
# Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US);
ua.msie
# Mozilla/5.0 (compatible; MSIE 10.0; Macintosh; Intel Mac OS X 10_7_3; Trident/6.0)
ua['Internet Explorer']
# Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; GTB7.4; InfoPath.3)
ua.opera
# Opera/9.80 (X11; Linux i686; U; ru) Presto/2.8.131 Version/11.11
ua.chrome
# Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko) Chrome/26.0.1410.63 Safari/537.2
ua.google
# Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/537.13 (KHTML, like Gecko) Chrome/26.0.1410.43 Safari/537.13
ua['google chrome']
# Mozilla/5.0 (X11; CrOS i686 2268.111.0) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/26.0.1410.43 Safari/536.11
ua.firefox
# Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:16.0.1) Gecko/20121011 Firefox/16.0.1
ua.ff
# Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:15.0) Gecko/20100101 Firefox/15.0
ua.safari
# Mozilla/5.0 (iPad; CPU OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Mobile/10A5376e

# and the best one, random via real world browser usage statistic
ua.random
```



- Referer
- Accept-Encoding
- Accept-Language

2. 检查网站生成的Cookie。

- 有用的插件：[EditThisCookie](#)
- 如何处理脚本动态生成的Cookie

3. 抓取动态内容。

- Selenium + WebDriver
- Chrome / Firefox - Driver

4. 限制爬取的速度。

5. 处理表单中的隐藏域。

- 在读取到隐藏域之前不要提交表单
- 用RoboBrowser这样的工具辅助提交表单

6. 处理表单中的验证码。

- OCR ( Tesseract ) - 商业项目一般不考虑
- 专业识别平台 - 超级鹰 / 云打码

```
from hashlib import md5

class ChaoClient(object):

    def __init__(self, username, password, soft_id):
        self.username = username
        password = password.encode('utf-8')
        self.password = md5(password).hexdigest()
        self.soft_id = soft_id
        self.base_params = {
            'user': self.username,
            'pass2': self.password,
            'softid': self.soft_id,
        }
        self.headers = {
            'Connection': 'Keep-Alive',
            'User-Agent': 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1;',
        }

    def post_pic(self, im, codetype):
        params = {
            'codetype': codetype,
```

```
}

params.update(self.base_params)
files = {'userfile': ('captcha.jpg', im)}
r = requests.post('http://upload.chaojiying.net/Upload/Processing.php')
return r.json()

if __name__ == '__main__':
    client = ChaoClient('用户名', '密码', '软件ID')
    with open('captcha.jpg', 'rb') as file:
        print(client.post_pic(file, 1902))
```



## 7. 绕开“陷阱”。

- 网页上有诱使爬虫爬取的爬取的隐藏链接（陷阱或蜜罐）
- 通过Selenium+WebDriver+Chrome判断链接是否可见或在可视区域

## 8. 隐藏身份。

- 代理服务 - 快代理 / 讯代理 / 芝麻代理 / 蘑菇代理 / 云代理  
[《爬虫代理哪家强？十大付费代理详细对比评测出炉！》](#)
- 洋葱路由 - 国内需要翻墙才能使用

```
yum -y install tor
useradd admin -d /home/admin
passwd admin
chown -R admin:admin /home/admin
chown -R admin:admin /var/run/tor
tor
```

10 :

Branch: master ▾

Find file Copy path

Python-100-Days / Day76-90 / 01.机器学习基础.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw

Blame

History



34 lines (21 sloc) 2.92 KB

## 机器学习基础

所谓“机器学习”就是利用计算机将纷繁复杂的数据处理成有用的信息，这样就可以发掘出数据带来的意义以及隐藏在数据背后的规律。现如今，“机器学习”和“大数据”可以说是IT行业中最热点的两个词汇，而无论是“机器学习”还是“大数据”最终要解决的问题本质上是一样的，用最为直白的话来说就是用现有的数据去预测将来的状况。

按照问题的“输入”和“输出”，我们可以将用计算机解决的问题分为四大类：

1. 输入的信息是精确的，要求输出最优解。
2. 输入的信息是精确的，无法找到最优解。
3. 输入的信息是模糊的，要求输出最优解。
4. 输入的信息是模糊的，无法找到最优解。

在上面的四大类问题中，第1类问题是计算机最擅长解决的，这类问题其实就是“数值计算”和“逻辑推理”方面的问题，而传统意义上的人工智能也就是利用逻辑推理来解决问题（如早期的“人机对弈”）。一直以来，我们都习惯于将计算机称为“电脑”，而基于“冯诺依曼”体系结构的“电脑”实际上只是实现了“人脑”理性思维这部分的功能，而在这一点上“电脑”通常是优于“人脑”的，而“人脑”在处理输入模糊信息时表现出来的强大的处理能力，在今天看来也不是“电脑”可以完全企及的。所以我们研究人工智能也好，研究机器学习也好，是希望输入模糊信息时，计算机能够给出满意的甚至是最优的答案。

至此，我们可以给“机器学习”下一个定义：机器学习是一门专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身性能的学科。机器学习目前已经广泛的应用到生产生活的各个领域，以下列举了一些经典的场景：

1. 搜索引擎：根据搜索和使用习惯，优化下一次搜索的结果。
2. 电商网站：自动推荐你可能感兴趣的的商品。
3. 贷款申请：通过你最近的金融活动信息进行综合评定。
4. 图像识别：自动识别图片中有没有不和谐的内容。

机器学习可以分为监督学习和非监督学习。监督学习是从给定的训练数据集中学习得到一个函数，当新的数据到来时，可以根据这个函数预测结果，监督学习的训练集包括输入和输出，也可以说是特征和目标。监督学习的目标是由人来标注的，而非监督学习的数据没有类别信息，训练集也没有人为标注结果，通过无监督学习可以减少数据特征的维度，以便我们可以使用二维或三维图形更加直观地展示数据信息。

实现机器学习的一般步骤：

1. 数据收集
2. 数据准备
3. 数据分析
4. 训练算法
5. 测试算法
6. 应用算法

Branch: master ▾

[Find file](#) [Copy path](#)

## [Python-100-Days / Day76-90 / 02.Pandas的应用.md](#)

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



3 lines (1 sloc) 20 Bytes

# Pandas的应用

Branch: master ▾

[Find file](#) [Copy path](#)

## [Python-100-Days / Day76-90 / 02.Pandas的应用.md](#)

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



3 lines (1 sloc) 20 Bytes

### Pandas的应用

Branch: master ▾

[Find file](#) [Copy path](#)

## [Python-100-Days / Day76-90 / 03.NumPy和SciPy的应用.md](#)

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



3 lines (1 sloc) 27 Bytes

# NumPy和SciPy的应用

Branch: master ▾

Find file Copy path

## Python-100-Days / Day76-90 / 04.Matplotlib和数据可视化.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



234 lines (154 sloc) 7.28 KB

# Matplotlib和数据可视化

数据的处理、分析和可视化已经成为Python近年来最为重要的应用领域之一，其中数据的可视化指的是将数据呈现为漂亮的统计图表，然后进一步发现数据中包含的规律以及隐藏的信息。数据可视化又跟数据挖掘和大数据分析紧密相关，而这些领域以及当下被热议的“深度学习”其最终的目标都是为了实现从过去的数据去对未来的状况进行预测。Python在实现数据可视化方面是非常棒的，即便是使用个人电脑也能够实现对百万级甚至更大体量的数据进行探索的工作，而这些工作都可以在现有的第三方库的基础上来完成（无需“重复的发明轮子”）。[Matplotlib](#)就是Python绘图库中的佼佼者，它包含了大量的工具，你可以使用这些工具创建各种图形（包括散点图、折线图、直方图、饼图、雷达图等），Python科学计算社区也经常使用它来完成数据可视化的工作。

## 安装matplotlib

可以使用pip来安装matplotlib，命令如下所示。

```
pip install matplotlib
```

## 绘制折线图

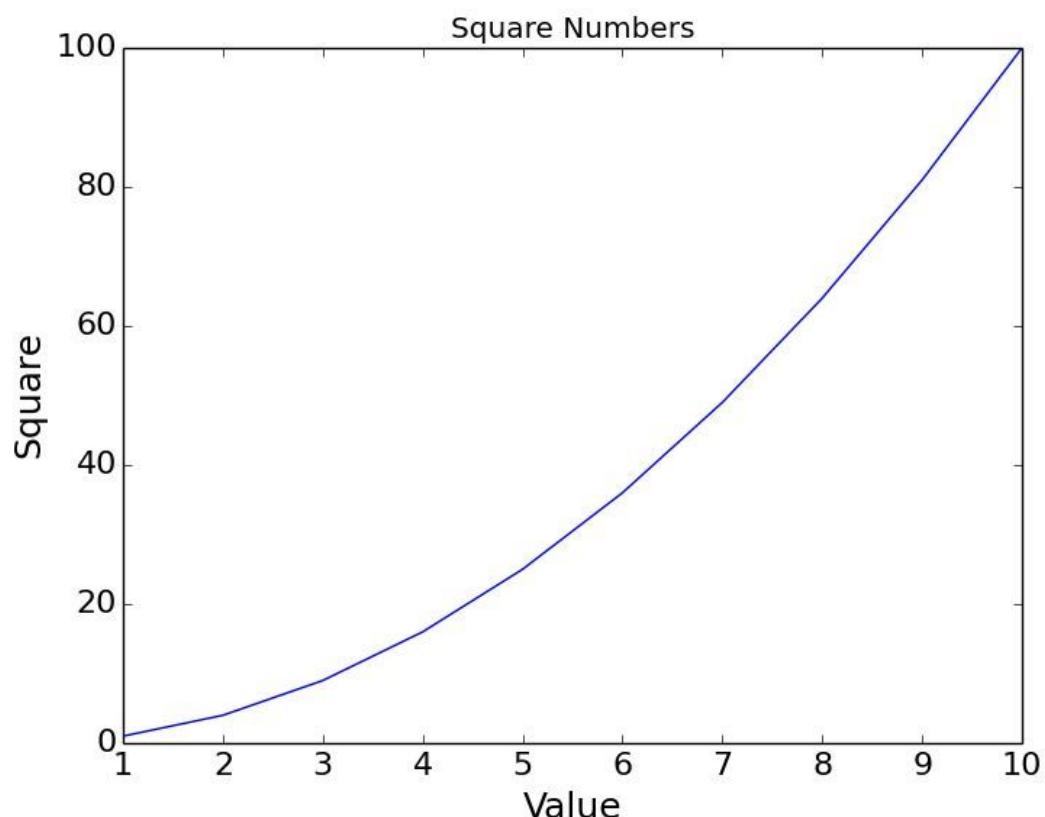
```
# coding: utf-8
import matplotlib.pyplot as plt

def main():
    # 保存x轴数据的列表
    x_values = [x for x in range(1, 11)]
    # 保存y轴数据的列表
    y_values = [x ** 2 for x in range(1, 11)]
    # 设置图表的标题以及x和y轴的说明
    plt.title('Square Numbers')
    plt.xlabel('Value', fontsize=18)
```

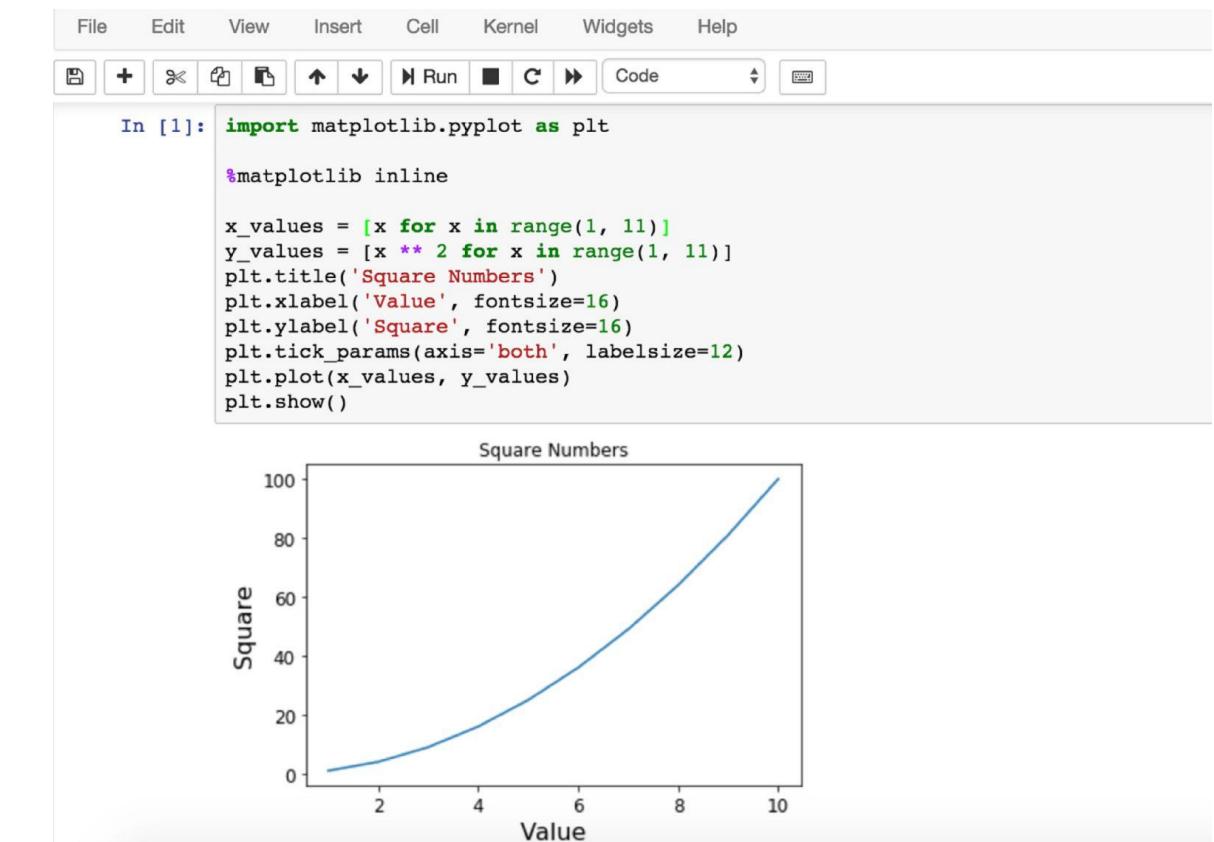
```
plt.ylabel('Square', fontsize=18)
# 设置刻度标记的文字大小
plt.tick_params(axis='both', labelsize=16)
# 绘制折线图
plt.plot(x_values, y_values)
plt.show()

if __name__ == '__main__':
    main()
```

运行程序，效果如下图所示。

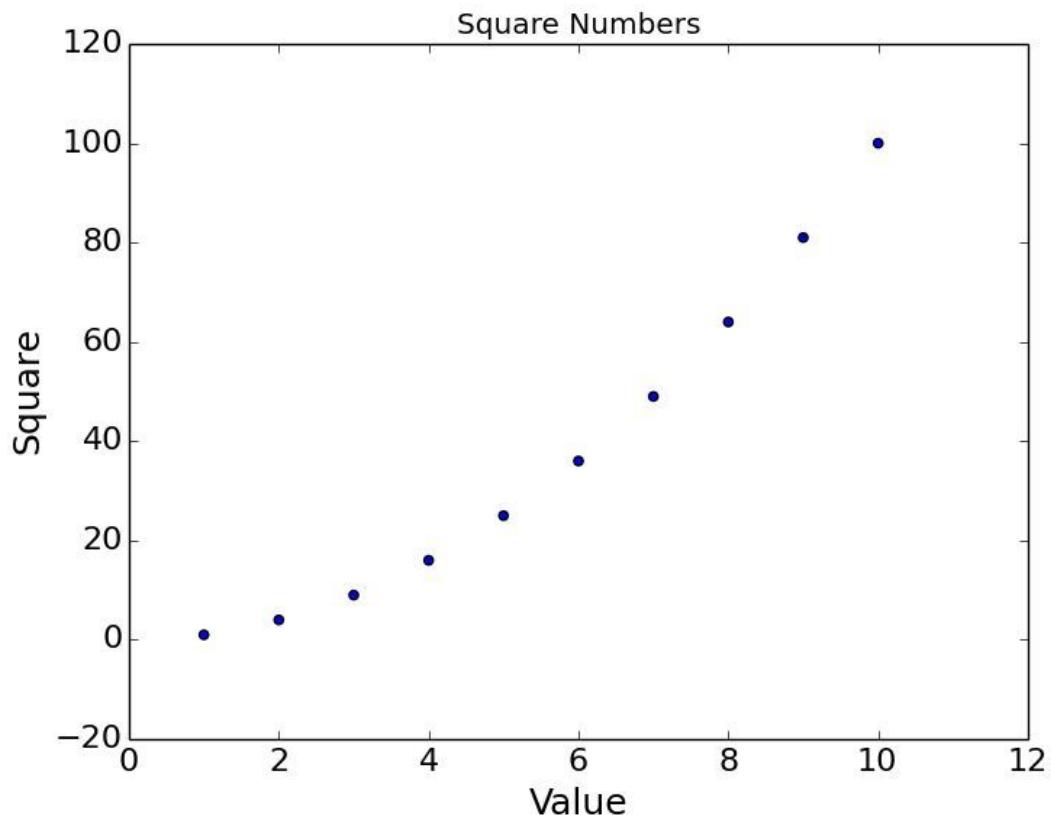


如果使用jupyter的notebook，需要使用魔法指令 %matplotlib inline 来设置在页面中显示图表，效果如下所示。



## 绘制散点图

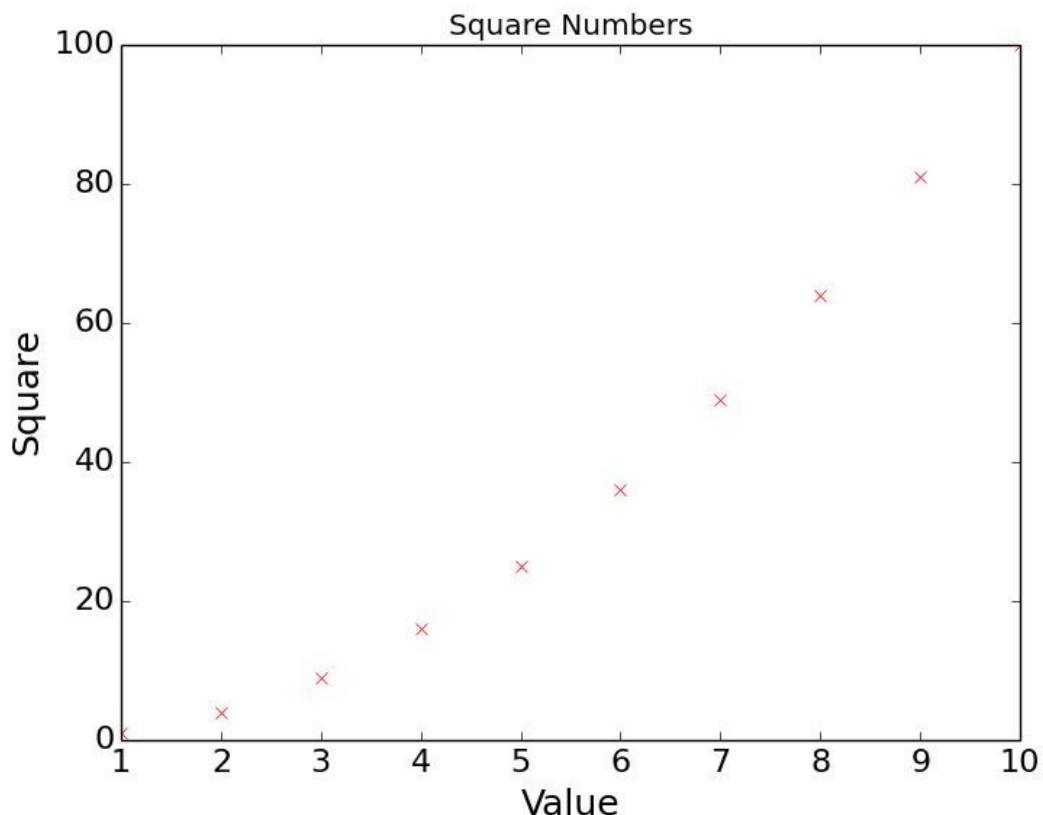
可以将上面代码中的的 `plot` 函数换成 `scatter` 函数来绘制散点图，效果如下图所示。



当然，也可以直接通过 `plot` 函数设置绘图的颜色和线条的形状将折线图改造为散点图，对应的代码如下所示，其中参数'xr'表示每个点的记号是'x'图形，颜色是红色（red）。

```
plt.plot(x_values, y_values, 'xr')
```

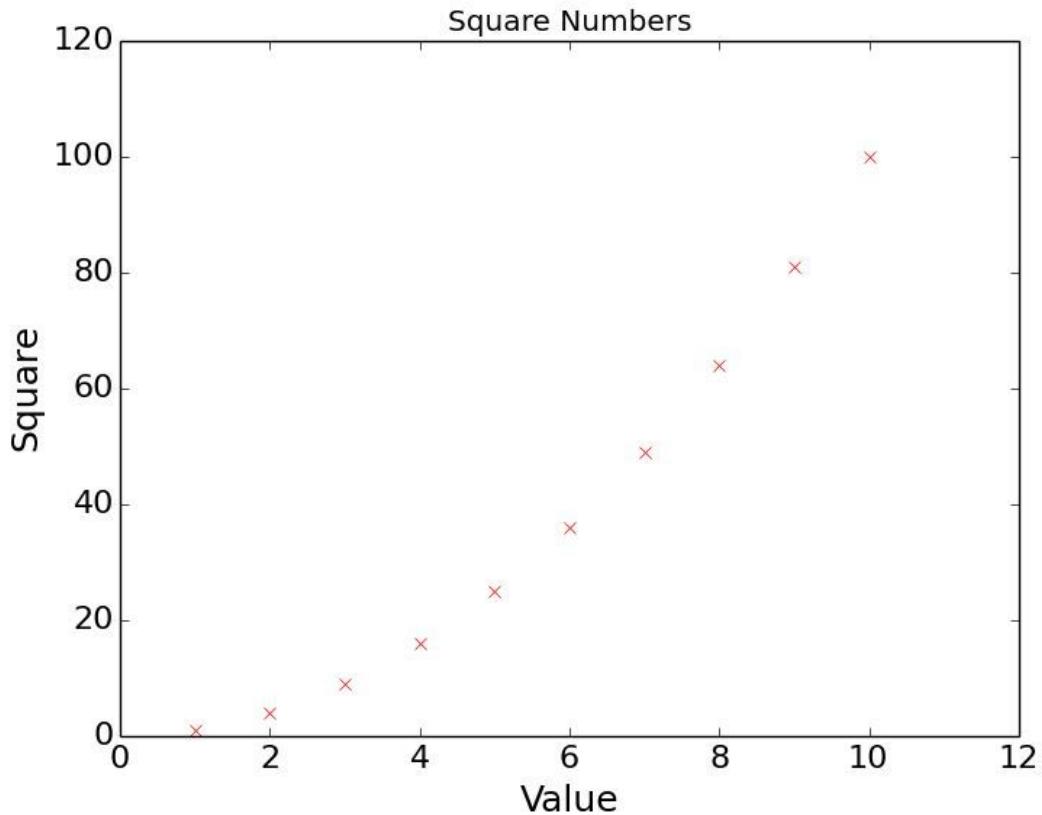
重新运行程序，效果如下图所示。



可能大家已经注意到了，1和10对应的'x'记号在图形边角的位置不太明显，要解决这个问题可以通过添加下面的代码调整x轴和y轴的坐标范围。

```
plt.axis([0, 12, 0, 120])
```

调整后的效果如下图所示。



## 绘制正弦曲线

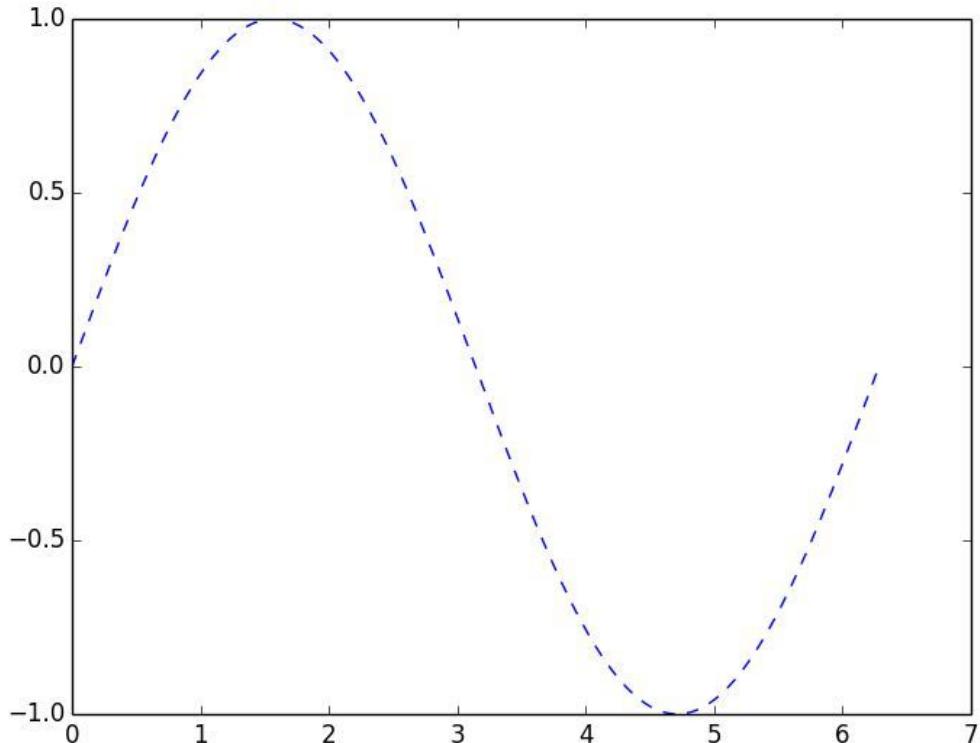
在下面的程序中，我们使用了名为NumPy的第三方库来产生样本并计算正弦值。NumPy是一个运行速度非常快的数学库，主要用于数组计算。它可以在Python中使用向量和数学矩阵，以及许多用C语言实现的底层函数。如果想通过Python学习数据科学或者机器学习相关的内容，那么就得先学会使用NumPy。

```
# coding: utf-8
import matplotlib.pyplot as plt
import numpy as np

def main():
    # 指定采样的范围以及样本的数量
    x_values = np.linspace(0, 2 * np.pi, 1000)
    # 计算每个样本对应的正弦值
    y_values = np.sin(x_values)
    # 绘制折线图(线条形状为--，颜色为蓝色)
    plt.plot(x_values, y_values, '--b')
    plt.show()

if __name__ == '__main__':
    main()
```

运行程序，效果如下图所示。



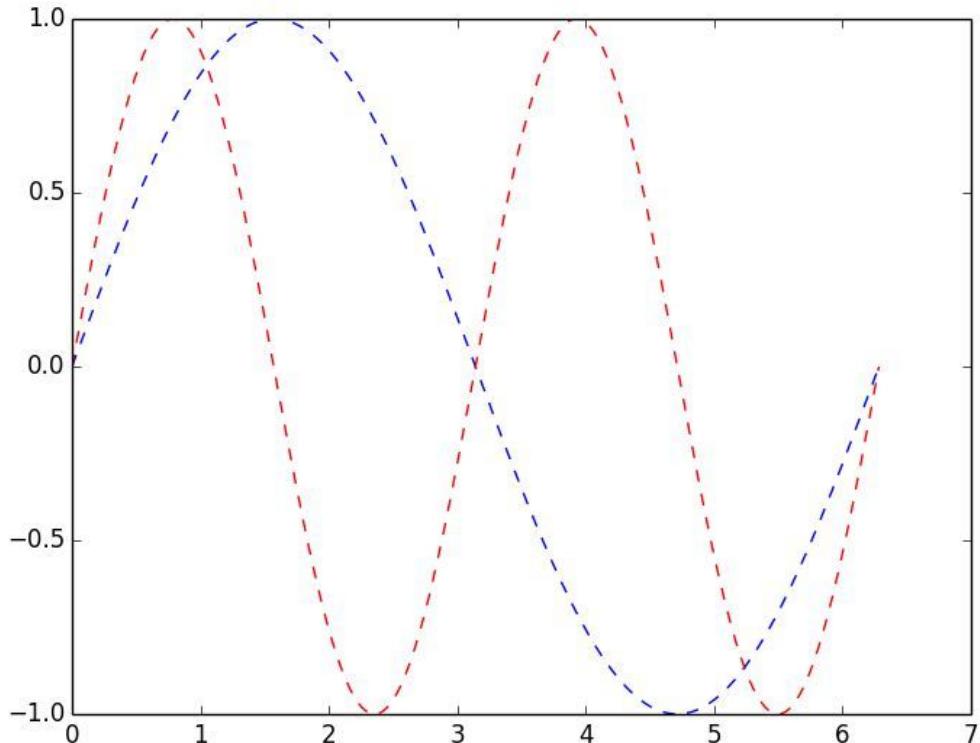
如果要在一个坐标系上绘制多个图像，可以按照如下的方式修改代码。

```
# coding: utf-8
import matplotlib.pyplot as plt
import numpy as np

def main():
    x_values = np.linspace(0, 2 * np.pi, 1000)
    plt.plot(x_values, np.sin(x_values), '--b')
    plt.plot(x_values, np.sin(2 * x_values), '--r')
    plt.show()

if __name__ == '__main__':
    main()
```

修改后的代码运行效果如下图所示。



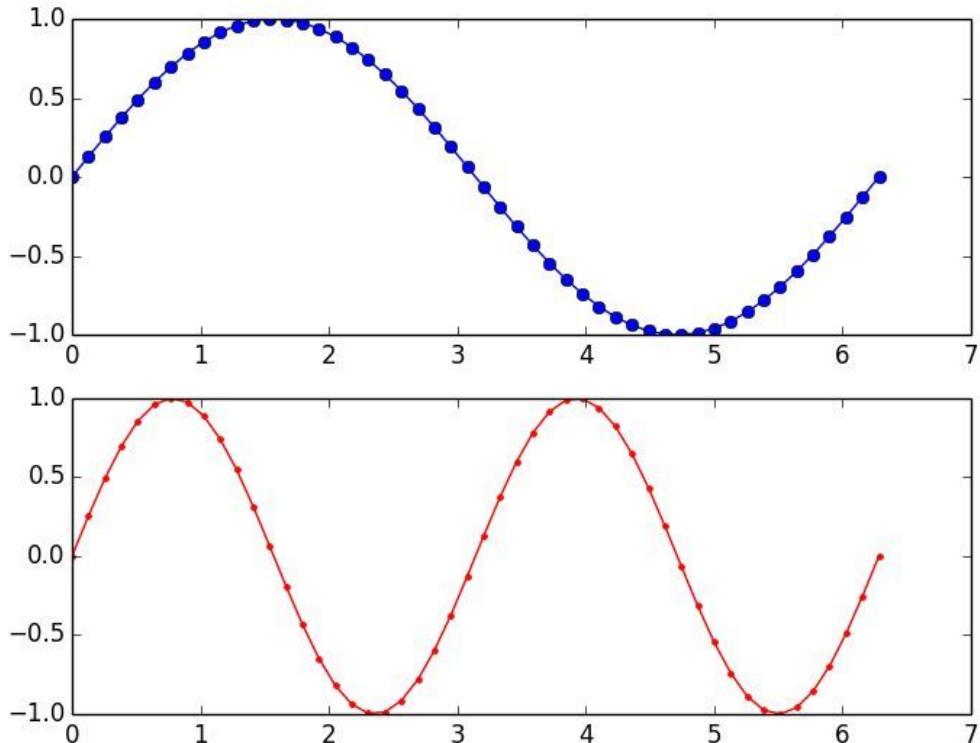
如果需要分别在两个坐标系上绘制出两条曲线，可以按照如下的方式操作。

```
# coding: utf-8
import matplotlib.pyplot as plt
import numpy as np

def main():
    # 将样本数量减少为50个
    x_values = np.linspace(0, 2 * np.pi, 50)
    # 设置绘图为2行1列活跃区为1区(第一个图)
    plt.subplot(2, 1, 1)
    plt.plot(x_values, np.sin(x_values), 'o-b')
    # 设置绘图为2行1列活跃区为2区(第二个图)
    plt.subplot(2, 1, 2)
    plt.plot(x_values, np.sin(2 * x_values), '.-r')
    plt.show()

if __name__ == '__main__':
    main()
```

效果如下图所示。



## 绘制直方图

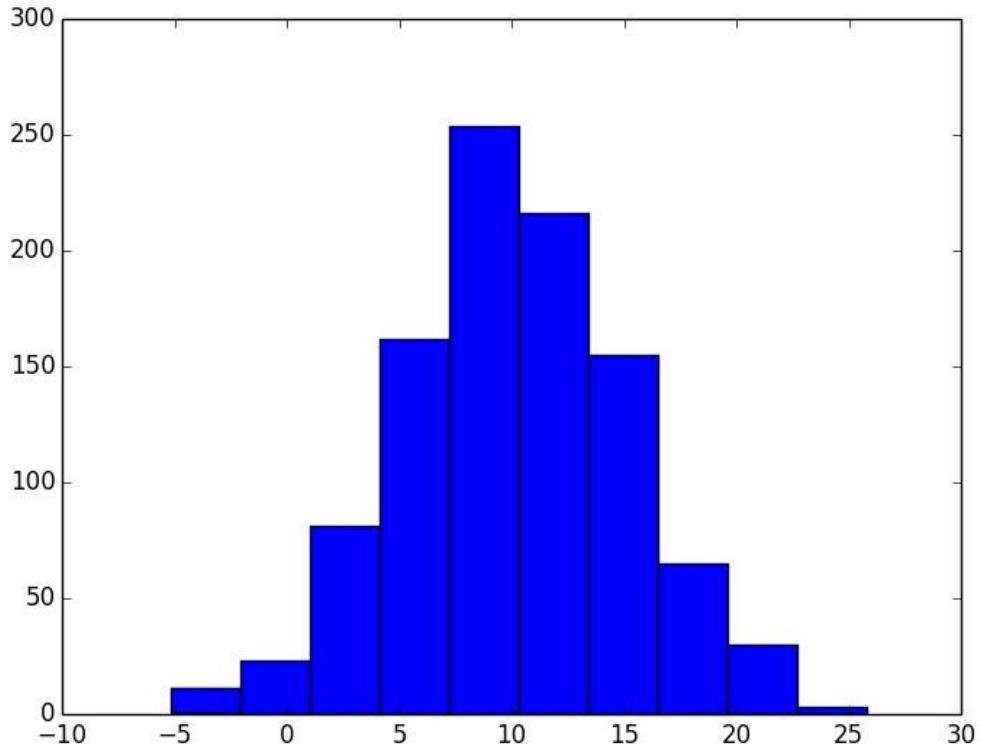
我们可以通过NumPy的random模块的normal函数来生成[正态分布](#)的采样数据，其中的三个参数分别表示期望、标准差和样本数量，然后绘制成直方图，代码如下所示。

```
# coding: utf-8
import matplotlib.pyplot as plt
import numpy as np

def main():
    # 通过random模块的normal函数产生1000个正态分布的样本
    data = np.random.normal(10.0, 5.0, 1000)
    # 绘制直方图(直方的数量为10个)
    plt.hist(data, 10)
    plt.show()

if __name__ == '__main__':
    main()
```

运行效果如下图所示。



## 使用Pygal绘制矢量图

矢量图（SVG）是[计算机图形学](#)中用点、直线或者多边形等基于数学方程的几何图元表示的图像，也是目前应用得非常多的一种图像文件格式，全称是“Scalable Vector Graphics”。和使用像素表示图像的位图不同，SVG基于XML存储图像数据，它是W3C定义的一种开放标准的矢量图形语言，可以用来设计更为清晰的Web图像，因为SVG与分辨率无关，在任意放大时不会丢失细节或影响清晰度。SVG可以直接用代码来描绘图像，也可以用任何文字处理工具来打开它，通过改变SVG的代码我们可以让图像具备交互功能。

Python中可以使用Pygal来生成SVG，可以通过pip来安装它。

```
from random import randint
import pygal

def roll_dice(n=1):
    total = 0
    for _ in range(n):
        total += randint(1, 6)
    return total

def main():
    results = []
    # 将两颗色子摇10000次记录点数
```

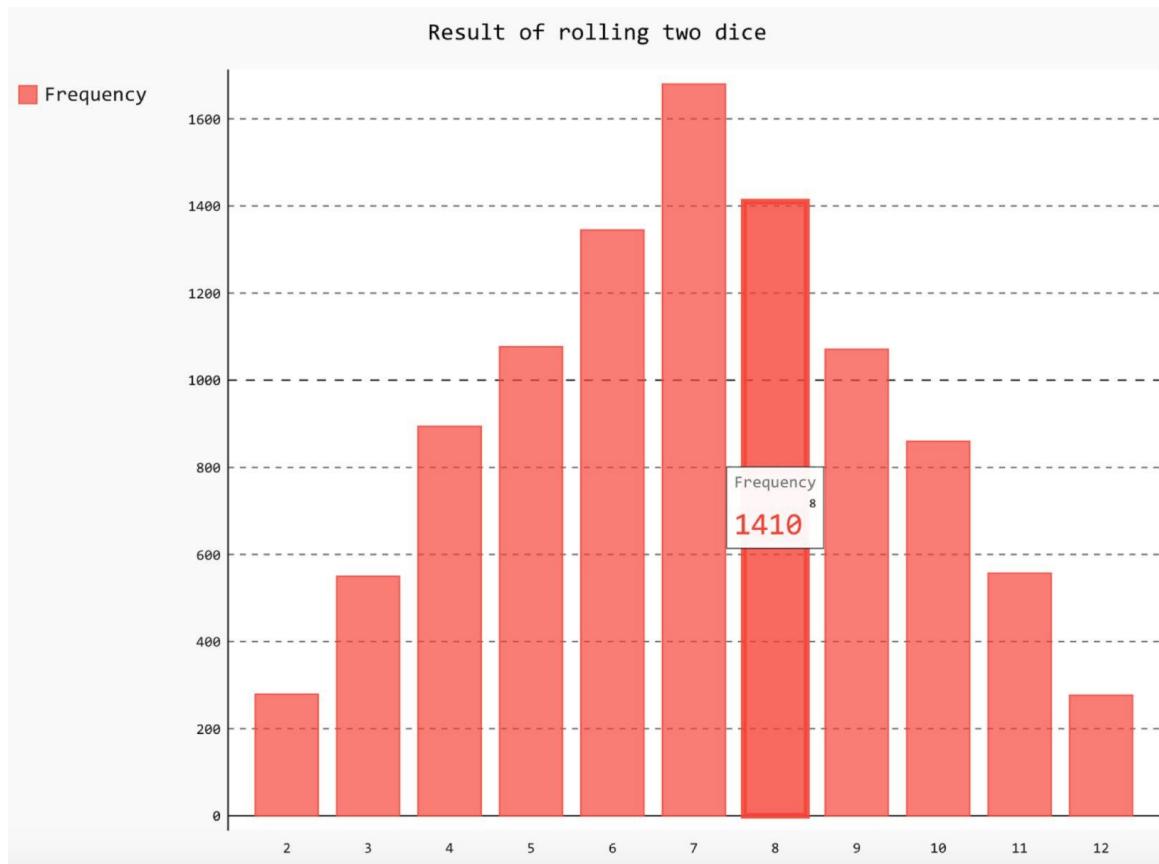
```

for _ in range(10000):
    face = roll_dice(2)
    results.append(face)
freqs = []
# 统计2~12点各出现了多少次
for value in range(2, 13):
    freq = results.count(value)
    freqs.append(freq)
# 绘制柱状图
hist = pygal.Bar()
hist.title = 'Result of rolling two dice'
hist.x_labels = [x for x in range(2, 13)]
hist.add('Frequency', freqs)
# 保存矢量图
hist.render_to_file('result.svg')

if __name__ == '__main__':
    main()

```

运行上面的程序，效果如下图所示。



## 后记

Matplotlib和NumPy的强大我们在这里也只是窥视了其冰山一角，我们在后续的内容里面还会使用到这两个第三方库，到时候我们再续点为大家介绍其他的功能。

Branch: master ▾ Python-100-Days / Day76-90 /

Create new file Upload files Find file History

This branch is even with jackfrued:master.

Pull request Compare

Cannot retrieve the latest commit at this time.

..

res

- 01.机器学习基础.md
- 02.Pandas的应用.md
- 03.NumPy和SciPy的应用.md
- 04.Matplotlib和数据可视化.md
- 05.k最近邻分类.md
- 06.决策树.md
- 07.贝叶斯分类.md
- 08.支持向量机.md
- 09.K-均值聚类.md
- 10.回归分析.md
- 11.大数据分析入门.md
- 12.大数据分析进阶.md
- 13.Tensorflow入门.md
- 14.Tensorflow实战.md
- 15.推荐系统实战.md

注：只更新一小部分

错误信息：无法解析服务器的 DNS 地址

[重新打开](#)

搜索

# 11: 几个知识点总结和团队开发

liupeng678 / Python-100-Days  
forked from jackfrued/Python-100-Days

Branch: master ▾ Python-100-Days / Day91-100 /

Create new file Upload files Find file History

This branch is even with jackfrued:master.

Pull request Compare

Cannot retrieve the latest commit at this time.

..

res

Django知识点概述.md

Docker简易上手指南.md

MySQL相关知识.md

关于测试.md

团队项目开发.md

电商网站技术要点剖析.md

网络API接口设计.md

英语面试.md

面试中的公共问题.md

项目部署上线指南.md

Branch: master ▾

Find file Copy path

## Python-100-Days / Day91-100 / Django知识点概述.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



2499 lines (1900 sloc) 81 KB

## Django知识点概述

### Web应用

问题1：描述一个Web应用的工作流程。

问题2：描述项目的物理架构。（上图中补充负载均衡（反向代理）服务器、数据库服务器、文件服务器、邮件服务器、缓存服务器、防火墙等，而且每个节点都有可能是多节点构成的集群，如下图所示，架构并不是一开始就是这样，而是逐步演进的）

问题3：描述Django项目的工作流程。（如下图所示）

### MVC架构模式

问题1：为什么要使用MVC架构模式？（模型和视图解耦合）

问题2：MVC架构中每个部分的作用？（如下图所示）

### HTTP请求和响应

HTTP请求 = 请求行+请求头+空行+[消息体]

HTTP响应 = 响应行+响应头+空行+消息体

## 1. `HttpRequest` 对象的属性和方法：

- `method` - 获取请求方法
- `path` / `get_full_path()` - 获取请求路径/带查询字符串的路径
- `scheme` / `is_secure()` / `get_host()` / `get_port()` - 获取请求的协议/主机/端口
- `META` / `COOKIES` - 获取请求头/Cookie信息
- `GET` / `POST` / `FILES` - 获取GET或POST请求参数/上传的文件
- `get_signed_cookie()` - 获取带签名的Cookie
- `is_ajax()` - 是不是Ajax异步请求
- `body` / `content_type` / `encoding` - 获取请求的消息体 ( bytes流 ) /MIME类型/编码

## 2. 中间件添加的属性：

- `session` / `user` / `site`

## 3. `HttpResponse` 对象的属性和方法：

- `set_cookie()` / `set_signed_cookie()` / `delete_cookie()` - 添加/删除Cookie
- `__setitem__` / `__getitem__` / `__delitem__` - 添加/获取/删除响应头
- `charset` / `content` / `status_code` - 响应的字符集/消息体 ( bytes流 ) /状态码
  - 1xx：请求已经收到，继续处理
  - 2xx（成功）：请求已经成功收到、理解和接收。
  - 3xx（重定向）：为完成请求要继续执行后续的操作。
  - 4xx（客户端错误）：请求不正确或不能够被受理。
  - 5xx（服务器错误）：服务器处理请求失败。

## 4. `JsonResponse` ( `HttpResponse` 的子类型) 对象

```
>>> from django.http import HttpResponse, JsonResponse
>>>
>>> response = JsonResponse({'foo': 'bar'})
>>> response.content
>>>
>>> response = JsonResponse([1, 2, 3], safe=False)
>>> response.content
>>>
>>> response = HttpResponse(b'...')
>>> response['content-type'] = 'application/pdf';
>>> response['content-disposition'] = 'inline; filename="xyz.pdf"'
>>> response['content-disposition'] = 'attachment; filename="xyz.pdf"'
>>>
>>> response.set_signed_cookie('foo', 'bar', salt='')
>>> response.status_code = 200
```

## 数据模型(Model)

问题1：关系型数据库表的设计应该注意哪些问题（范式理论和逆范式）？如何通过表来创建模型类（反向工程）？如何通过模型类来创建表（正向工程）？

```
python manage.py makemigrations <appname>
python manage.py migrate

python manage.py inspectdb > <appname>/models.py
```

问题2：关系型数据库中数据完整性指的是什么？什么时候需要牺牲数据完整性？（实体完整性/参照完整性/域完整性）

问题3：ORM是什么以及解决了什么问题？（对象模型-关系模型双向转换）

1. Field 及其子类的属性：

- 通用选项：
  - db\_column / db\_tablespace
  - null / blank / default
  - primary\_key
  - db\_index / unique
  - choices / help\_text / error\_message / editable / hidden

○ 其他选项：

- CharField : max\_length
- DateField : auto\_now / auto\_now\_add
- DecimalField : max\_digits / decimal\_places
- FileField : storage / upload\_to
- ImageField : height\_field / width\_field

2. ForeignKey 的属性：

- 重要属性：
  - db\_constraint ( 提升性能或者数据分片的情况可能需要设置为 False )
  - on\_delete
    - CASCADE : 级联删除。
    - PROTECT : 抛出 ProtectedError 异常，阻止删除引用的对象。
    - SET\_NULL : 把外键设置为 null ，当 null 属性被设置为 True 时才能这么做。
    - SET\_DEFAULT : 把外键设置为默认值，提供了默认值才能这么做。

- related\_name

```
class Dept(models.Model):  
    pass  
  
class Emp(models.Model):  
    dept = models.ForeignKey(related_name='+', ...)  
  
Dept.objects.get(no=10).emp_set.all()  
Emp.objects.filter(dept_no=10)
```

说明：related\_name 设置为 '+'，可以防止一对多外键关联从“一”的一方查询“多”的一方。

- 其他属性：

- to\_field / limit\_choices\_to / swappable

### 3. Model 的属性和方法

- objects / pk
- save() / delete()
- clean() / validate\_unique() / full\_clean()

### 4. QuerySet 的方法

- get() / all() / values()

说明：values() 返回的 QuerySet 中不是模型对象而是字典

- count() / order\_by() / exists() / reverse()
- filter() / exclude()
  - exact / iexact : 精确匹配/忽略大小写的精确匹配查询
  - contains / icontains / startswith / istartswith / endswith / iendswith : 基于 like 的模糊查询
  - in : 集合运算
  - gt / gte / lt / lte : 大于/大于等于/小于/小于等于关系运算
  - range : 指定范围查询 ( SQL 中的 between...and... )

- year / month / day / week\_day / hour / minute / second : 查询时间日期
- isnull : 查询空值 ( True ) 或非空值 ( False )
- search : 基于全文索引的全文检索
- regex / iregex : 基于正则表达式的模糊匹配查询
- aggregate() / annotate()
- Avg / Count / Sum / Max / Min

```
>>> from django.db.models import Avg
>>> Emp.objects.aggregate(avg_sal=Avg('sal'))
(0.001) SELECT AVG(`TbEmp`.`sal`) AS `avg_sal` FROM `TbEmp`; args=()
{'avg_sal': 3521.4286}
```

```
>>> Emp.objects.values('dept').annotate(total=Count('dept'))
(0.001) SELECT `TbEmp`.`dno`, COUNT(`TbEmp`.`dno`) AS `total` FROM `TbEmp`
<QuerySet [{`dept`: 10, `total`: 4}, {"dept": 20, "total": 7}, {"dept": 30, "total": 3}, {"dept": 40, "total": 2}, {"dept": 50, "total": 1}...]
```

- first() / last()

说明：调用 first() 方法相当于用 [0] 对 QuerySet 进行切片。

- only() / defer()

```
>>> Emp.objects.filter(pk=7800).only('name', 'sal')
(0.001) SELECT `TbEmp`.`empno`, `TbEmp`.`ename`, `TbEmp`.`sal` FROM `TbEmp` WHERE `TbEmp`.`empno` = 7800
<QuerySet [

```

- create() / update() / raw()

```
>>> Emp.objects.filter(dept_no=20).update(sal=F('sal') + 100)
(0.011) UPDATE `TbEmp` SET `sal` = (`TbEmp`.`sal` + 100) WHERE `TbEmp`.`dno` = 20
>>>
>>> Emp.objects.raw('select empno, ename, job from TbEmp where dno=10')
<RawQuerySet: select empno, ename, job from TbEmp where dno=10>
```

## 5. Q 对象和 F 对象

说明：Q对象主要用来解决多条件组合的复杂查询；F对象主要用于更新数据。

```
>>> from django.db.models import Q
>>> Emp.objects.filter(
...     Q(name__startswith='张'),
...     Q(sal__lte=5000) | Q(comm__gte=1000)
... ) # 查询名字以“张”开头且工资小于等于5000或补贴大于等于1000的员工
<QuerySet [<Emp: 张三丰>]>
```

```
>>> from backend.models import Emp, Dept
>>> emps = Emp.objects.filter(dept_no=20)
>>> from django.db.models import F
>>> emps.update(sal=F('sal') + 100)
```

## 6. 原生SQL查询

```
from django.db import connections

with connections['...'].cursor() as cursor:
    cursor.execute("UPDATE TbEmp SET sal=sal+10 WHERE dno=30")
    cursor.execute("SELECT ename, job FROM TbEmp WHERE dno=10")
    row = cursor.fetchall()
```

## 7. 模型管理器

```
class BookManager(models.Manager):

    def title_count(self, keyword):
        return self.filter(title__icontains=keyword).count()

class Book(models.Model):

    objects = BookManager()
```

## 视图函数(Controller)

### 如何设计视图函数

1. 用户的每个操作（用户故事）对应一个视图函数。
2. 每个视图函数可以构成一个事务边界。
  - 事务的ACID特性。
    - 原子性（Atomicity）：事务中各项的操作要么全做要么全不做；

- 一致性 ( Consistency ) : 事务前后系统状态是一致的 ;
  - 隔离性 ( Isolation ) : 并发执行的事务无法看到彼此的中间状态 ;
  - 持久性 ( Duration ) : 事务完成后所做的改动都会被持久化。
- 事务隔离级别 - 设置事务隔离级别是为了数据库底层依据事务隔离级别为数据加上适当的锁。如果需要保证数据的强一致性，那么关系型数据库仍然是唯一的也是最好的选择，因为关系型数据库可以通过锁机制来保护数据。事务隔离级别从低到高依次是：Read Uncommitted ( 读未提交 )、Read Committed ( 读提交 )、Repeatable Read ( 可重复读 )、Serializable ( 串行化 )。事务隔离级别越高，数据并发访问的问题越少，但是性能越差；事务隔离级别越低，数据并发访问的问题越多，但是性能越好。
- 数据并发访问会产生5种问题（请参考我的[《Java面试题全集（上）》第80题对该问题的讲解](#)）：
- 第1类丢失更新 ( A 事务撤销覆盖 B 事务更新的数据 ) 和第2类丢失更新 ( A 事务提交覆盖 B 事务更新的数据 )。
  - 脏读 ( 读脏数据 ) : 一个事务读取到其他尚未提交的事务的数据。
  - 不可重复读 : 一个事务在读取它的查询结果时，被另一个事务更新了它的查询记录导致无法读到数据。
  - 幻读 : 一个事务在读取它的查询结果时，发现读到了被另一个事务提交的新数据。
- ```

-- 设置全局默认的事务隔离级别
set global transaction isolation level repeatable read;
-- 设置当前会话的事务隔离级别
set session transaction isolation level read committed;
-- 查询当前会话的事务隔离级别
select @@tx_isolation;

```
- Django中的事务控制。
- 给每个请求绑定事务环境 ( 反模式 )。
- ```

ATOMIC_REQUESTS = True

```
- 使用事务装饰器 ( 简单易用 ) - 粗粒度 ( 控制不够精细 )。
- ```

@transaction.non_atomic_requests
@transaction.atomic

```
- 使用上下文语法 ( 细粒度 - 事务控制的范围更加精准 )。

```
with transaction.atomic():
    pass
```

- 关闭自动提交使用手动提交。

```
AUTOCOMMIT = False
```

```
transaction.commit()
transaction.rollback()
```

## URL配置

1. 可以让部分URL只在调试模式下生效。

```
from django.conf import settings

urlpatterns = [
    ...
]

if settings.DEBUG:
    urlpatterns += [ ... ]
```

2. 可以使用命名捕获组捕获路径参数。

```
url(r'^api/code/(?P<mobile>1[3-9]\d{9})'),
path('api/code/<str:mobile>'),
```

3. URL配置不关心请求使用的方法（一个视图函数可以处理不同的请求方式）。
4. 如果使用 url 函数捕获的路径参数都是字符串， path 函数可以指定路径参数类型。
5. 可以使用 include 函数引入其他URL配置，捕获的参数会向下传递。
6. 在 url 和 path 函数甚至是 include 函数中都可以用字典向视图传入额外的参数，如果参数与捕获的参数同名，则使用字典中的参数。
7. 可以用 reverse 函数实现URL的逆向解析（从名字解析出URL），在模板中也可以用 { % url % } 实现同样的操作。

```
path('', views.index, name='index')

return redirect(reverse('index'))
return redirect('index')
```

## 模板(View)

### 后端渲染

1. 模板的配置和渲染函数。

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates'), ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

resp = render(request, 'index.html', {'foo': ...})
```

2. 模板遇到变量名的查找顺序。

- 字典查找 (如： foo['bar'] )
- 属性查找 (如： foo.bar )
- 方法调用 (如： foo.bar() )
  - 方法不能有必须传值的参数
  - 在模板中不能够给方法传参
  - 如果方法的 `alters_data` 被设置为 `True` 则不能调用该方法 (避免误操作的风险)，模型对象动态生成的 `delete()` 和 `save()` 方法都设定了 `alters_data = True`。
- 列表索引查找 (如： foo[0] )

3. 模板标签的使用。

- `{% if %}` / `{% else %}` / `{% endif %}`
- `{% for %}` / `{% endfor %}`
- `{% ifequal %}` / `{% endifequal %}` / `{% ifnotequal %}` / `{% endifnotequal %}`
- `{# comment #}` / `{% comment %}` / `{% endcomment %}`

4. 过滤器的使用。

- o lower / upper / first / last / truncatewords / date / time / length / pluralize / center / ljust / rjust / cut / urlencode / default\_if\_none / filesizeformat / join / slice / slugify

## 5. 模板的包含和继承。

- o {% include %} / {% block %}
- o {% extends %}

## 6. 模板加载器（后面优化部分会讲到）。

- o 文件系统加载器

```
TEMPLATES = [
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [os.path.join(BASE_DIR, 'templates')],
]
```

- o 应用目录加载器

```
TEMPLATES = [
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'APP_DIRS': True,
]
```

## 前端渲染

1. 前端模板引擎 : Handlebars / Mustache。
2. 前端MV\*框架。
  - o MVC - AngularJS
  - o MVVM(Model-View-ViewModel) - Vue.js

## 其他视图

1. MIME ( 多用途Internet邮件扩展 ) 类型 - 告知浏览器传输的数据类型。

| Content-Type     | 说明                                  |
|------------------|-------------------------------------|
| application/json | JSON ( JavaScript Object Notation ) |
| application/pdf  | PDF ( Portable Document Format )    |
| audio/mpeg       | MP3或其他MPEG音频文件                      |
| audio/vnd.wave   | WAV音频文件                             |
| image/gif        | GIF图像文件                             |

| Content-Type    | 说明            |
|-----------------|---------------|
| image/jpeg      | JPEG图像文件      |
| image/png       | PNG图像文件       |
| text/html       | HTML文件        |
| text/xml        | XML           |
| video/mp4       | MP4视频文件       |
| video/quicktime | QuickTime视频文件 |

## 2. 如何处置生成的内容 ( inline / attachment ) 。

```
>>> from urllib.parse import quote
>>>
>>> response['content-type'] = 'application/pdf'
>>> filename = quote('Python语言规范.pdf')
>>> filename
'Python%E8%AF%AD%E8%A8%80%E8%A7%84%E8%8C%83.pdf'
>>> response['content-disposition'] = f'attachment; filename="{filename}"'
```

提醒：URL以及请求和响应头中的中文都应该处理成百分号编码。

## 3. 生成CSV / Excel / PDF / 统计报表。

- 向浏览器传输二进制数据。

```
buffer = BytesIO()

resp = HttpResponse(content_type='...')
resp['Content-Disposition'] = 'attachment; filename="..."'
resp.write(buffer.getvalue())


def get_style(name, color=0, bold=False, italic=False):
    style = xlwt.XFStyle()
    font = xlwt.Font()
    font.name = name
    font.colour_index = color
    font.bold = bold
    font.italic = italic
    style.font = font
    return style


def export_emp_excel(request):
    # 创建Excel工作簿(使用三方库xlwt)
    workbook = xlwt.Workbook()
```

```

# 向工作簿中添加工作表
sheet = workbook.add_sheet('员工详细信息')
# 设置表头
titles = ['编号', '姓名', '主管', '职位', '工资', '部门名称']
for col, title in enumerate(titles):
    sheet.write(0, col, title, get_style('HanziPenSC-W3', 2, True))
# 使用Django的ORM框架查询员工数据
emps = Emp.objects.all().select_related('dept').select_related('mgr')
cols = ['no', 'name', 'mgr', 'job', 'sal', 'dept']
# 通过嵌套的循环将员工表的数据写入Excel工作表的单元格
for row, emp in enumerate(emps):
    for col, prop in enumerate(cols):
        val = getattr(emp, prop, '')
        if isinstance(val, (Dept, Emp)):
            val = val.name
        sheet.write(row + 1, col, val)
# 将Excel文件的二进制数据写入内存
buffer = BytesIO()
workbook.save(buffer)
# 通过HttpResponse对象向浏览器输出Excel文件
resp = HttpResponse(buffer.getvalue())
resp['content-type'] = 'application/msexcel'
# 如果文件名有中文需要处理成百分号编码
resp['content-disposition'] = 'attachment; filename="detail.xls"'
return resp

```

- 大文件的流式处理：StreamingHttpResponse。

```

def download_file(request):
    file_stream = open('...', 'rb')
    # 如果文件的二进制数据较大则最好用迭代器进行处理避免过多的占用服务器内存
    file_iter = iter(lambda: file_stream.read(4096), b'')
    resp = StreamingHttpResponse(file_iter)
    # 中文文件名要处理成百分号编码
    filename = quote('...', 'utf-8')
    resp['Content-Type'] = '...'
    resp['Content-Disposition'] = f'attachment; filename="{filename}"'
    return resp

```

说明：如果需要生成PDF文件，可以需要安装 reportlab。另外，使用 StreamingHttpResponse只能减少内存的开销，但是如果下载一个大文件会导致一个请求长时间占用服务器资源，比较好的做法还是把报表提前生成好（可以考虑使用定时任务），放在静态资源服务器或者是云存储服务器上以访问静态资源的方式访问。

- ECharts或Chart.js。

- 思路：后端只提供JSON格式的数据，前端JavaScript渲染生成图表。

```
def get_charts_data(request):
    """获取统计图表JSON数据"""
    names = []
    totals = []
    # 通过connections获取指定数据库连接并创建游标对象
    with connections['backend'].cursor() as cursor:
        # 在使用ORM框架时可以使用对象管理器的aggregate()和annotate()方法实现
        # 执行原生SQL查询(如果ORM框架不能满足业务或性能需求)
        cursor.execute('select dname, total from vw_dept_emp')
        for row in cursor.fetchall():
            names.append(row[0])
            totals.append(row[1])
    return JsonResponse({'names': names, 'totals': totals})
```



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>统计图表</title>
    <style>
        #main {
            width: 600px;
            height: 400px;
        }
    </style>
</head>
<body>
    <div id="main"></div>
    <script src="https://cdn.bootcss.com/echarts/4.2.0-rc.2/echarts.min.js"><
    <script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script
    <script>
        var myChart = echarts.init($('#main')[0]);
        $.ajax({
            'url': 'charts_data',
            'type': 'get',
            'data': {},
            'dataType': 'json',
            'success': function(json) {
                var option = {
                    title: {
                        text: '员工分布统计图'
                    },
                    tooltip: {},
                    legend: {
                        data:['人数']
                    },
                    xAxis: {
                        data: json.names
                    },
                    yAxis: {},
                    series: [{


```

```
        name: '人数',
        type: 'bar',
        data: json.totals
    }]
},
myChart.setOption(option);
},
'error': function() {}
});
</script>
</body>
</html>
```



## 中间件

问题1：中间件背后的设计理念是什么？（分离横切关注功能/拦截过滤器模式）

问题2：中间件有哪些不同的实现方式？（参考下面的代码）

问题3：描述Django内置的中间件及其执行顺序。（推荐阅读：[Django官方文档 - 中间件 - 中间件顺序](#)）

### 激活中间件

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'common.middlewares.block_sms_middleware',
]
```

### 自定义中间件

```
def simple_middleware(get_response):

    def middleware(request, *args, **kwargs):

        response = get_response(request, *args, **kwargs)

        return response

    return middleware
```

```

class MyMiddleware(object):

    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)
        return response


class MyMiddleware(object):

    def __init__(self):
        pass

    def process_request(self, request):
        pass

    def process_view(self, request, view_func, view_args, view_kwargs):
        pass

    def process_template_response(self, request, response):
        pass

    def process_response(self, request, response):
        pass

    def process_exception(self, request, exception):
        pass

```

## 内置中间件

### 1. CommonMiddleware - 基础设置中间件

- DISALLOWED\_USER\_AGENTS - 不被允许的用户代理 ( 浏览器 )
- APPEND\_SLASH - 是否追加 /
- USE\_ETAG - 浏览器缓存相关

### 2. SecurityMiddleware - 安全相关中间件

- SECURE\_HSTS\_SECONDS - 强制使用HTTPS的时间
- SECURE\_HSTS\_INCLUDE\_SUBDOMAINS - HTTPS是否覆盖子域名
- SECURE\_CONTENT\_TYPE\_NOSNIFF - 是否允许浏览器推断内容类型
- SECURE\_BROWSER\_XSS\_FILTER - 是否启用跨站脚本攻击过滤器
- SECURE\_SSL\_REDIRECT - 是否重定向到HTTPS连接
- SECURE\_REDIRECT\_EXEMPT - 免除重定向到HTTPS

3. SessionMiddleware - 会话中间件
4. CsrfViewMiddleware - 防范跨站身份伪造中间件
5. XFrameOptionsMiddleware - 防范点击劫持攻击中间件

## 表单

1. 用法：通常不要用来生成页面上的表单控件（耦合度太高不容易定制），主要用来验证数据。
2. Form的属性和方法：
  - o `is_valid()` / `is_multipart()`
  - o `errors` / `fields` / `is_bound` / `changed_data` / `cleaned_data`
  - o `add_error()` / `has_errors()` / `non_field_errors()`
  - o `clean()`
3. Form.errors的方法：
  - o `as_data()` / `as_json()` / `get_json_data()`

问题1：Django中的 `Form` 和 `ModelForm` 有什么作用？（通常不用来生成表单主要用来验证数据）

问题2：表单上传文件时应该注意哪些问题？（表单的设置、多文件上传、图片预览（`FileReader`）、Ajax上传文件、上传后的文件如何存储、调用云存储（如[阿里云OSS](#)、[七牛云](#)、[LeanCloud](#)等））

```
<form action="" method="post" enctype="multipart/form-data">
    <input type="file" name="..." multiple>
    <input type="file" name="foo">
    <input type="file" name="foo">
    <input type="file" name="foo">
</form>
```

说明：上传图片文件的预览效果可以通过HTML5的`FileReader`来实现。

说明：使用云存储通常是比自己配置分布式文件系统这种方式更靠谱的做法，而且云存储通常成本并不太高，不仅如此大多数云存储还提供了如图片剪裁、生成水印、视频转码、CDN等服务。如果要自己做上传的视频文件转码，需要安装三方库 `ffmpeg`，在程序中调用该三方库可以实现转码。

## Cookie和Session

问题1：使用Cookie能解决什么问题？（用户跟踪，解决HTTP协议无状态问题）

## 1. URL重写

http://www.abc.com/path/resource?foo=bar

## 2. 隐藏域 ( 隐式表单域 ) - 埋点

```
<form action="" method="post">  
    <input type="hidden" name="foo" value="bar">  
</form>
```

## 3. Cookie - 浏览器中的临时文件 ( 文本文件 ) - BASE64

问题2：Cookie和Session之间关系是什么？( Session的标识通过Cookie保存和传输 )

### Session的配置

1. Session对应的中间件： django.contrib.sessions.middleware.SessionMiddleware。

2. Session引擎。

- 基于数据库 ( 默认方式 )

```
INSTALLED_APPS = [  
    'django.contrib.sessions',  
]
```

- 基于缓存 ( 推荐使用 )

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'  
SESSION_CACHE_ALIAS = 'session'
```

- 基于文件 ( 基本不考虑 )

- 基于Cookie ( 不靠谱 )

```
SESSION_ENGINE = 'django.contrib.sessions.backends.signed_cookies'
```

## 3. Cookie相关的配置。

```
SESSION_COOKIE_NAME = 'djang_session_id'  
SESSION_COOKIE_AGE = 1209600  
# 如果设置为True, Cookie就是基于浏览器窗口的Cookie, 不会持久化  
SESSION_EXPIRE_AT_BROWSER_CLOSE = False
```

```
SESSION_SAVE_EVERY_REQUEST = False  
SESSION_COOKIE_HTTPONLY = True
```

#### 4. session的属性和方法。

- session\_key / session\_data / expire\_date
- \_\_getitem\_\_ / \_\_setitem\_\_ / \_\_delitem\_\_ / \_\_contains\_\_
- set\_expiry() / get\_expiry\_age() / get\_expiry\_date() - 设置/获取会话超期时间
- flush() - 销毁会话
- set\_test\_cookie() / test\_cookie\_worked() / delete\_test\_cookie() - 测试浏览器是否支持Cookie ( 提示用户如果浏览器禁用Cookie可能会影响网站的使用 )

#### 5. session的序列化。

```
SESSION_SERIALIZER = 'django.contrib.sessions.serializers.JSONSerializer'
```

- JSONSerializer ( 1.6及以后默认 ) - 如果想将自定义的对象放到session中，会遇到“Object of type 'XXX' is not JSON serializable”的问题 ( 如果配置使用Redis保存Session，django-redis使用了Pickle序列化，这个问题就不存在了 )。
- PickleSerializer ( 1.6以前的默认 ) - 因为安全问题不推荐使用，但是只要不去反序列化用户构造的恶意的Payload其实也没有什么风险。关于这种方式的安全漏洞，可以参考《[Python Pickle的任意代码执行漏洞实践和Payload构造](#)》一文或《软件架构-Python语言实现》上关于这个问题的讲解。

说明：如果使用了django\_redis整合Redis作为session的存储引擎，那么由于django\_redis又封装了一个PickleSerializer来提供序列化，所以不会存在上述的问题，且Redis中保存的value是pickle序列化之后的结果。

## 缓存

### 配置缓存

```
CACHES = {  
    # 默认缓存  
    'default': {  
        'BACKEND': 'django_redis.cache.RedisCache',  
        'LOCATION': [  
            'redis://1.2.3.4:6379/0',  
        ],  
        'KEY_PREFIX': 'teamproject',  
        'OPTIONS': {  
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',  
            'CONNECTION_POOL_KWARGS': {  
                'max_connections': 1000,  
            },  
        },  
    },  
}
```

```
        },
        'PASSWORD': '1qaz2wsx',
    }
},
# 页面缓存
'page': {
    'BACKEND': 'django_redis.cache.RedisCache',
    'LOCATION': [
        'redis://1.2.3.4:6379/1',
    ],
    'KEY_PREFIX': 'teamproject:page',
    'OPTIONS': {
        'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        'CONNECTION_POOL_KWARGS': {
            'max_connections': 500,
        },
        'PASSWORD': '1qaz2wsx',
    }
},
# 会话缓存
'session': {
    'BACKEND': 'django_redis.cache.RedisCache',
    'LOCATION': [
        'redis://1.2.3.4:6379/2',
    ],
    'KEY_PREFIX': 'teamproject:session',
    'TIMEOUT': 1209600,
    'OPTIONS': {
        'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        'CONNECTION_POOL_KWARGS': {
            'max_connections': 2000,
        },
        'PASSWORD': '1qaz2wsx',
    }
},
# 接口数据缓存
'api': {
    'BACKEND': 'django_redis.cache.RedisCache',
    'LOCATION': [
        'redis://1.2.3.4:6379/3',
    ],
    'KEY_PREFIX': 'teamproject:api',
    'OPTIONS': {
        'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        'CONNECTION_POOL_KWARGS': {
            'max_connections': 500,
        },
        'PASSWORD': '1qaz2wsx',
    }
},
}
```

说明：通过Redis底层提供的多个数据库来隔离缓存数据有助于缓存数据的管理。如果配置了Redis的主从复制（读写分离），LOCATION列表中可以配置多个Redis连接，第一个被视为master用来进行写操作，后面的被视为slave用来进行读操作。

## 全站缓存

```
MIDDLEWARE_CLASSES = [
    'django.middleware.cache.UpdateCacheMiddleware',
    ...
    'django.middleware.common.CommonMiddleware',
    ...
    'django.middleware.cache.FetchFromCacheMiddleware',
]

CACHE_MIDDLEWARE_ALIAS = 'default'
CACHE_MIDDLEWARE_SECONDS = 300
CACHE_MIDDLEWARE_KEY_PREFIX = 'djangocache'
```

## 视图层缓存

```
from django.views.decorators.cache import cache_page
from django.views.decorators.vary import vary_on_cookie

@cache_page(timeout=60 * 15, cache='page')
@vary_on_cookie
def my_view(request):
    pass

from django.views.decorators.cache import cache_page

urlpatterns = [
    url(r'^foo/([0-9]{1,2})/$', cache_page(60 * 15)(my_view)),
]
```

## 其他内容

### 1. 模板片段缓存。

- { % load cache %}
- { % cache %} / { % endcache %}

### 2. 使用底层API访问缓存。

```
>>> from django.core.cache import cache
>>>
>>> cache.set('my_key', 'hello, world!', 30)
```

```

>>> cache.get('my_key')
>>> cache.clear()

>>> from django.core.cache import caches
>>> cache1 = caches['page']
>>> cache2 = caches['page']
>>> cache1 is cache2
True
>>> cache3 = caches['session']
>>> cache2 is cache3
False

>>> from django_redis import get_redis_connection
>>>
>>> redis_client = get_redis_connection()
>>> redis_client.hgetall()

```

## 日志

### 日志级别

NOTSET < DEBUG < INFO < WARNING < ERROR < FATAL

### 日志配置

```

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    # 配置日志格式化器
    'formatters': {
        'simple': {
            'format': '%(asctime)s %(module)s.%(funcName)s: %(message)s',
            'datefmt': '%Y-%m-%d %H:%M:%S',
        },
        'verbose': {
            'format': '%(asctime)s %(levelname)s [%(process)d-%(threadName)s] '
                      '%(module)s.%(funcName)s line %(lineno)d: %(message)s',
            'datefmt': '%Y-%m-%d %H:%M:%S',
        }
    },
    # 配置日志过滤器
    'filters': {
        'require_debug_true': {
            '()': 'django.utils.log.RequireDebugTrue',
        },
    },
    # 配置日志处理器
    'handlers': {
        'console': {

```

```

        'class': 'logging.StreamHandler',
        'level': 'DEBUG',
        'filters': ['require_debug_true'],
        'formatter': 'simple',
    },
    'file1': {
        'class': 'logging.handlers.TimedRotatingFileHandler',
        'filename': 'access.log',
        'when': 'W0',
        'backupCount': 12,
        'formatter': 'simple',
        'level': 'INFO',
    },
    'file2': {
        'class': 'logging.handlers.TimedRotatingFileHandler',
        'filename': 'error.log',
        'when': 'D',
        'backupCount': 31,
        'formatter': 'verbose',
        'level': 'WARNING',
    },
},
# 配置日志器
'loggers': {
    'django': {
        'handlers': ['console', 'file1', 'file2'],
        'propagate': True,
        'level': 'DEBUG',
    },
}
}

```

日志配置官方示例。

## 日志分析

1. Linux相关命令：head、tail、grep、awk、uniq、sort

```
tail -10000 access.log | awk '{print $1}' | uniq -c | sort -r
```

2. 实时日志文件分析：Python + 正则表达式 + Crontab

3. [《Python日志分析工具》](#)。

4. [《集中式日志系统ELK》](#)。

- ElasticSearch：搜索引擎，实现全文检索。
- Logstash：负责从指定节点收集日志。
- Kibana：日志可视化工具。

5. 大数据日志处理：Flume+Kafka日志采集、Storm / Spark实时数据处理、Impala实时查询。

## RESTful

问题1：RESTful架构到底解决了什么问题？（URL具有自描述性、资源表述与视图的解耦和、互操作性利用构建微服务以及集成第三方系统、无状态性提高水平扩展能力）

问题2：项目在使用RESTful架构时有没有遇到一些问题或隐患？（对资源访问的限制、资源从属关系检查、避免泄露业务信息、防范可能的攻击）

补充：下面的几个和安全性相关的响应头在前面讲中间件的时候提到过的。

- X-Frame-Options: DENY
- X-Content-Type-Options: nosniff
- X-XSS-Protection: 1; mode=block;
- Strict-Transport-Security: max-age=31536000;

问题3：如何保护API中的敏感信息以及防范重放攻击？（摘要和令牌）

推荐阅读：[《如何有效防止API的重放攻击》](#)。

## 使用djangorestframework

安装djangorestframework（为了描述方便，以下统一简称为drf）。

```
pip install djangorestframework
```

配置drf。

```
INSTALLED_APPS = [  
    'rest_framework',  
]  
  
REST_FRAMEWORK = {  
    # 配置默认页面大小  
    'PAGE_SIZE': 10,  
    # 配置默认的分页类  
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',  
    # 配置异常处理器  
    # 'EXCEPTION_HANDLER': 'api.exceptions.exception_handler',  
    # 配置默认解析器  
    # 'DEFAULT_PARSER_CLASSES': (  
    #     'rest_framework.parsers.JSONParser',  
    #     'rest_framework.parsers.FormParser',  
    #     'rest_framework.parsers.MultiPartParser',
```

```

    # ),
    # 配置默认限流类
    # 'DEFAULT_THROTTLE_CLASSES': (),
    # 配置默认授权类
    # 'DEFAULT_PERMISSION_CLASSES': (
    #     'rest_framework.permissions.IsAuthenticated',
    # ),
    # 配置默认认证类
    # 'DEFAULT_AUTHENTICATION_CLASSES': (
    #     'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
    # ),
}

```

## 编写序列化器

```

from rest_framework import serializers
from rest_framework.serializers import ModelSerializer

from common.models import District, HouseType, Estate, Agent


class DistrictSerializer(ModelSerializer):

    class Meta:
        model = District
        fields = ('distid', 'name')


class HouseTypeSerializer(ModelSerializer):

    class Meta:
        model = HouseType
        fields = '__all__'


class AgentSerializer(ModelSerializer):

    class Meta:
        model = Agent
        fields = ('agentid', 'name', 'tel', 'servstar', 'certificated')


class EstateSerializer(ModelSerializer):
    district = serializers.SerializerMethodField()
    agents = serializers.SerializerMethodField()

    @staticmethod
    def get_agents(estate):
        return AgentSerializer(estate.agents, many=True).data

    @staticmethod
    def get_district(estate):
        return DistrictSerializer(estate.district).data

```

```
class Meta:  
    model = Estate  
    fields = '__all__'
```

## 方法1：使用装饰器

```
@api_view(['GET'])  
@cache_page(timeout=None, cache='api')  
def provinces(request):  
    queryset = District.objects.filter(parent__isnull=True)  
    serializer = DistrictSerializer(queryset, many=True)  
    return Response(serializer.data)  
  
@api_view(['GET'])  
@cache_page(timeout=300, cache='api')  
def cities(request, provid):  
    queryset = District.objects.filter(parent__distid=provid)  
    serializer = DistrictSerializer(queryset, many=True)  
    return Response(serializer.data)  
  
urlpatterns = [  
    path('districts/', views.provinces, name='districts'),  
    path('districts/<int:provid>/', views.cities, name='cities'),  
]
```

说明：上面使用了Django自带的视图装饰器（@cache\_page）来实现对API接口返回数据的缓存。

## 方法2：使用APIView及其子类

更好的复用代码，不要重“复发明轮子”。

```
class HouseTypeAPIView(CacheResponseMixin, ListAPIView):  
    queryset = HouseType.objects.all()  
    serializer_class = HouseTypeSerializer  
  
urlpatterns = [  
    path('housetypes/', views.HouseTypeAPIView.as_view(), name='housetypes'),  
]
```

说明：上面使用了drf\_extensions提供的CacheResponseMixin混入类实现了对接口数据的缓存。如果重写了获取数据的方法，可以使用drf\_extensions提供的@cache\_response来实现对接口数据的缓存，也可以用自定义的函数来生成缓存中的key。当然还有一个选择就是通过Django提供的@method\_decorator装饰器，将@cache\_page装饰器处理为装饰方法的装饰器，这样也能提供使用缓存服务。

drf-extensions 配置如下所示。

```
# 配置DRF扩展来支持缓存API接口调用结果
REST_FRAMEWORK_EXTENSIONS = {
    'DEFAULT_CACHE_RESPONSE_TIMEOUT': 300,
    'DEFAULT_USE_CACHE': 'default',
    # 配置默认缓存单个对象的key函数
    'DEFAULT_OBJECT_CACHE_KEY_FUNC': 'rest_framework_extensions.utils.default_object_c
    # 配置默认缓存对象列表的key函数
    'DEFAULT_LIST_CACHE_KEY_FUNC': 'rest_framework_extensions.utils.default_list_cache_
}
```

### 方法3：使用ViewSet及其子类

```
class HouseTypeViewSet(CacheResponseMixin, viewsets.ModelViewSet):
    queryset = HouseType.objects.all()
    serializer_class = HouseTypeSerializer
    pagination_class = None

    router = DefaultRouter()
    router.register('housetypes', views.HouseTypeViewSet)

    urlpatterns += router.urls
```

djangorestframework提供了基于Bootstrap定制的页面来显示接口返回的JSON数据，当然也可以使用POSTMAN这样的工具对API接口进行测试。

### 补充说明

在这里顺便提一下跟前端相关的几个问题。

问题1：如何让浏览器能够发起DELETE/PUT/PATCH？

```
<form method="post">
    <input type="hidden" name="_method" value="delete">
</form>
```

```

if request.method == 'POST' and '_method' in request.POST:
    request.method = request.POST['_method'].upper()

<script>
    $.ajax({
        'url': '/api/provinces',
        'type': 'put',
        'data': {},
        'dataType': 'json',
        'success': function(json) {
            // Web = 标签(内容) + CSS(显示) + JS(行为)
            // JavaScript = ES + BOM + DOM
            // DOM操作实现页面的局部刷新
        },
        'error': function() {}
    });
    $.getJSON('/api/provinces', function(json) {
        // DOM操作实现页面的局部刷新
    });
</script>

```

问题2：如何解决多个JavaScript库之间某个定义（如\$函数）冲突的问题？

```

<script src="js/jquery.min.js"></script>
<script src="js/abc.min.js"></script>
<script>
    // $已经被后加载的JavaScript库占用了
    // 但是可以直接用绑定在window对象上的jQuery去代替$
    jQuery(function() {
        jQuery('#okBtn').on('click', function() {});
    });
</script>

<script src="js/abc.min.js"></script>
<script src="js/jquery.min.js"></script>
<script>
    // 将$让给出其他的JavaScript库使用
    jQuery.noConflict();
    jQuery(function() {
        jQuery('#okBtn').on('click', function() {});
    });
</script>

```

问题3：jQuery对象与原生DOM对象之间如何转换？

```

<button id="okBtn">点我</button>
<script src="js/jquery.min.js"></script>

```

```
<script>
    var btn = document.getElementById('okBtn'); // 原生JavaScript对象(使用相对麻烦)
    var $btn = $('#okBtn'); // jQuery对象(拥有更多的属性和方法而且没有浏览器兼容性
    $btn.on('click', function() {});
    // $(btn)可以将原生JavaScript对象转成jQuery对象
    // $btn.get(0)或$btn[0]可以获得原生的JavaScript对象
</script>
```

## 过滤数据

如果需要过滤数据（对数据接口设置筛选条件、排序条件等），可以使用 `django-filter` 三方库来实现。

```
pip install django-filter
```

```
INSTALLED_APPS = [
    'django_filters',
]

REST_FRAMEWORK = {
    'DEFAULT_FILTER_BACKENDS': (
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.OrderingFilter',
    ),
}

from django.utils.decorators import method_decorator
from django.views.decorators.cache import cache_page
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.filters import OrderingFilter
from rest_framework.generics import RetrieveAPIView, ListCreateAPIView

from api.serializers import EstateSerializer
from common.models import Estate

@method_decorator(decorator=cache_page(timeout=120, cache='api', key_prefix='estates'))
class EstateView(RetrieveAPIView, ListCreateAPIView):
    queryset = Estate.objects.all().select_related('district').prefetch_related('agent')
    serializer_class = EstateSerializer
    filter_backends = (DjangoFilterBackend, OrderingFilter)
    filter_fields = ('name', 'district')
    ordering = ('-hot', )
    ordering_fields = ('hot', 'estateid')
```

```

from django_filters import rest_framework as drf
from common.models import HouseInfo

class HouseInfoFilter(drf.FilterSet):
    """自定义房源数据过滤器"""

    title = drf.CharFilter(lookup_expr='starts')
    dist = drf.NumberFilter(field_name='district')
    min_price = drf.NumberFilter(field_name='price', lookup_expr='gte')
    max_price = drf.NumberFilter(field_name='price', lookup_expr='lte')
    type = drf.NumberFilter()

    class Meta:
        model = HouseInfo
        fields = ('title', 'district', 'min_price', 'max_price', 'type')

class HouseInfoViewSet(CacheResponseMixin, ReadOnlyModelViewSet):
    queryset = HouseInfo.objects.all() \
        .select_related('type', 'district', 'estate', 'agent') \
        .prefetch_related('tags').order_by('-pubdate')
    serializer_class = HouseInfoSerializer
    filter_backends = (DjangoFilterBackend, OrderingFilter)
    filterset_class = HouseInfoFilter
    ordering = ('price',)
    ordering_fields = ('price', 'area')

```

## 身份认证

查看drf中APIView类的代码可以看出，drf默认的认证方案是  
DEFAULT\_AUTHENTICATION\_CLASSES，如果修改authentication\_classes就可以自行定制身份认证的方案。

```

class APIView(View):

    # The following policies may be set at either globally, or per-view.
    renderer_classes = api_settings.DEFAULT_RENDERER_CLASSES
    parser_classes = api_settings.DEFAULT_PARSER_CLASSES
    authentication_classes = api_settings.DEFAULT_AUTHENTICATION_CLASSES
    throttle_classes = api_settings.DEFAULT_THROTTLE_CLASSES
    permission_classes = api_settings.DEFAULT_PERMISSION_CLASSES
    content_negotiation_class = api_settings.DEFAULT_CONTENT_NEGOTIATION_CLASS
    metadata_class = api_settings.DEFAULT_METADATA_CLASS
    versioning_class = api_settings.DEFAULT_VERSIONING_CLASS

    # 此处省略下面的代码

```

```

DEFAULTS = {
    # Base API policies
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
    ),
    'DEFAULT_PARSER_CLASSES': (
        'rest_framework.parsers.JSONParser',
        'rest_framework.parsers.FormParser',
        'rest_framework.parsers.MultiPartParser'
    ),
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication'
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.AllowAny',
    ),
    'DEFAULT_THROTTLE_CLASSES': (),
    'DEFAULT_CONTENT_NEGOTIATION_CLASS': 'rest_framework.negotiation.DefaultContentNeg',
    'DEFAULT_METADATA_CLASS': 'rest_framework.metadata.SimpleMetadata',
    'DEFAULT_VERSIONING_CLASS': None,
}

# 此处省略下面的代码
}

```

自定义认证类，继承 `BaseAuthentication` 并重写 `authenticate(self, request)` 方法，通过请求中的 `userid` 和 `token` 来确定用户身份。如果认证成功，该方法应返回一个二元组（用户和令牌的信息），否则产生异常。也可以重写 `authenticate_header(self, request)` 方法来返回一个字符串，该字符串将用于 `HTTP 401 Unauthorized` 响应中的 `WWW-Authenticate` 响应头的值。如果未重写该方法，那么当未经身份验证的请求被拒绝访问时，身份验证方案将返回 `HTTP 403 Forbidden` 响应。

```

class MyAuthentication(BaseAuthentication):
    """自定义用户身份认证类"""

    def authenticate(self, request):
        try:
            token = request.GET['token'] or request.POST['token']
            user_token = UserToken.objects.filter(token=token).first()
            if user_token:
                return user_token.user, user_token
            else:
                raise AuthenticationFailed('请提供有效的用户身份标识')
        except KeyError:
            raise AuthenticationFailed('请提供有效的用户身份标识')

    def authenticate_header(self, request):
        pass

```

使用自定义的认证类。

```
class EstateViewSet(CacheResponseMixin, ModelViewSet):
    # 通过queryset指定如何获取数据（资源）
    queryset = Estate.objects.all().select_related('district').prefetch_related('agent')
    # 通过serializer_class指定如何序列化数据
    serializer_class = EstateSerializer
    # 指定根据哪些字段进行数据筛选
    filter_fields = ('district', 'name')
    # 指定根据哪些字段对数据进行排序
    ordering_fields = ('hot', )
    # 指定用于进行用户身份验证的类
    authentication_classes = (MyAuthentication, )
```

说明：也可以在Django配置文件中将自定义的认证类设置为默认认证方式。

## 授予权限

权限检查总是在视图的最开始处运行，在任何其他代码被允许进行之前。最简单的权限是允许通过身份验证的用户访问，并拒绝未经身份验证的用户访问，这对应于dfr中的`IsAuthenticated`类，可以用它来取代默认的`AllowAny`类。权限策略可以在Django的drf配置中用`DEFAULT_PERMISSION_CLASSES`全局设置。

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    )
}
```

也可以在基于`APIView`类的视图上设置身份验证策略。

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView

class ExampleView(APIView):
    permission_classes = (IsAuthenticated, )
    # 此处省略其他代码
```

或者在基于`@api_view`装饰器的视图函数上设置。

```
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated

@api_view(['GET'])
@permission_classes((IsAuthenticated, ))
```

```
def example_view(request, format=None):
    # 此处省略其他代码
```

自定义权限需要继承 `BasePermission` 并实现以下方法中的一个或两个，下面是 `BasePermission` 的代码。

```
@six.add_metaclass(BasePermissionMetaclass)
class BasePermission(object):
    """
    A base class from which all permission classes should inherit.
    """

    def has_permission(self, request, view):
        """
        Return `True` if permission is granted, `False` otherwise.
        """
        return True

    def has_object_permission(self, request, view, obj):
        """
        Return `True` if permission is granted, `False` otherwise.
        """
        return True
```

如果请求被授予访问权限，则方法应该返回 `True`，否则返 `False`。下面的例子演示了阻止黑名单中的IP地址访问接口数据（这个在反爬虫的时候很有用哟）。

```
from rest_framework import permissions

class BlacklistPermission(permissions.BasePermission):
    """
    Global permission check for blacklisted IPs.
    """

    def has_permission(self, request, view):
        ip_addr = request.META['REMOTE_ADDR']
        blacklisted = Blacklist.objects.filter(ip_addr=ip_addr).exists()
        return not blacklisted
```

如果要实现更为完整的权限验证，可以考虑RBAC或ACL。

1. RBAC - 基于角色的访问控制，如下图所示。

2. ACL - 访问控制列表（每个用户绑定自己的访问白名单）。

## 访问限流

可以修改dfr配置的 `DEFAULT_THROTTLE_CLASSES` 和 `DEFAULT_THROTTLE_RATES` 两个值来设置全局默认限流策略。例如：

```
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': (
        'rest_framework.throttling.AnonRateThrottle',
        'rest_framework.throttling.UserRateThrottle'
    ),
    'DEFAULT_THROTTLE_RATES': {
        'anon': '3/min',
        'user': '10000/day'
    }
}
```

`DEFAULT_THROTTLE_RATES` 中使用的频率描述可能包括 `second`、`minute`、`hour` 或 `day`。

如果要为接口单独设置限流，可以在每个视图或视图集上设置限流策略，如下所示：

```
from rest_framework.throttling import UserRateThrottle
from rest_framework.views import APIView

class ExampleView(APIView):
    throttle_classes = (UserRateThrottle, )
    # 此处省略下面的代码
```

或

```
@api_view(['GET'])
@throttle_classes([UserRateThrottle, ])
def example_view(request, format=None):
    # 此处省略下面的代码
```

当然也可以通过继承 `BaseThrottle` 来自定义限流策略，通常需要重写 `allow_request` 和 `wait` 方法。

## 异步任务和计划任务

### Celery的应用

Celery 是一个简单、灵活且可靠的，处理大量消息的分布式系统，并且提供维护这样一个系统的必需工具。它是一个专注于实时处理的任务队列，同时也支持任务调度。

推荐阅读：[《Celery官方文档中文版》](#)，上面有极为详细的配置和使用指南。

Celery是一个本身不提供队列服务，官方推荐使用RabbitMQ或Redis来实现消息队列服务，前者是更好的选择，它对AMQP（高级消息队列协议）做出了非常好的实现。

#### 1. 安装RabbitMQ。

```
docker pull rabbitmq
docker run -d -p 5672:5672 --name myrabbit rabbitmq
docker container exec -it myrabbit /bin/bash
```

#### 2. 创建用户、资源以及分配操作权限。

```
rabbitmqctl add_user luohao 123456
rabbitmqctl set_user_tags luohao administrator
rabbitmqctl add_vhost vhost1
rabbitmqctl set_permissions -p vhost1 luohao ".*" ".*" ".*"
```

#### 3. 创建Celery实例。

```
# 注册环境变量
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'fangtx.settings')

# 创建Celery实例
app = celery.Celery(
    'fangtx',
    broker='amqp://luohao:123456@120.77.222.217:5672/vhost1'
)

# 从项目的配置文件读取Celery配置信息
app.config_from_object('django.conf:settings')
# 从指定的文件(例如celery_config.py)中读取Celery配置信息
# app.config_from_object('celery_config')

# 让Celery自动从参数指定的应用中发现异步任务/定时任务
app.autodiscover_tasks(['common',])
# 让Celery自动从所有注册的应用中发现异步任务/定时任务
# app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)
```

#### 4. 启动Celery创建worker（消息的消费者）。

```
celery -A <name> worker -l debug &
```

#### 5. 执行异步任务。

```
@app.task
def send_email(from, to, cc, subject, content):
```

```
pass

send_email.delay('', [], [], '', '')
```

## 6. 创建定时任务。

```
# 配置定时任务（计划任务）
app.conf.update(
    timezone=settings.TIME_ZONE,
    enable_utc=True,
    # 定时任务（计划任务）相当于是消息的生产者
    # 如果只有生产者没有消费者那么消息就会在消息队列中积压
    # 将来实际部署项目的时候生产者、消费者、消息队列可能都是不同节点
    beat_schedule={
        'task1': {
            'task': 'common.tasks.show_msg',
            'schedule': crontab(),
            'args': ('刘强东，奶茶妹妹喊你回家喝奶啦', )
        },
    },
)

@app.task
def show_msg(content):
    print(content)
```

## 7. 启动Celery创建执行定时任务的beat（消息的生产者）。

```
celery -A <name> beat -l info
```

## 8. 检查消息队列状况。

```
rabbitmqctl list_queues -p vhost1
```

## 9. 监控Celery - 可以通过flower来对Celery进行监控。

```
pip install flower
celery flower --broker=amqp://luohao:123456@120.77.222.217:5672/vhost1
```

## 其他问题

问题1：如何解决JavaScript跨域获取数据的问题？( django-cors-headers )

```

INSTALLED_APPS = [
    'corsheaders',
]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
]

CORS_ORIGIN_ALLOW_ALL = True
# 配置跨域白名单
# CORS_ORIGIN_WHITELIST = ('www.abc.com', 'www.baidu.com')
# CORS_ORIGIN_REGEX_WHITELIST = ('...', )
# CORS_ALLOW_CREDENTIALS = True
# CORS_ALLOW_METHODS = ('GET', 'POST', 'PUT', 'DELETE')

```

问题2：网站图片（水印、剪裁）和视频（截图、水印、转码）是如何处理的？（云存储、FFmpeg）

问题3：网站如何架设（静态资源）文件系统？（FastDFS、云存储、CDN）

## 安全保护

问题1：什么是跨站脚本攻击，如何防范？（对提交的内容进行消毒）

问题2：什么是跨站身份伪造，如何防范？（使用随机令牌）

问题3：什么是SQL注入攻击，如何防范？（不拼接SQL语句，避免使用单引号）

问题4：什么是点击劫持攻击，如何防范？（不允许`<iframe>`加载非同源站点内容）

### Django提供的安全措施

#### 签名数据的API

```

>>> from django.core.signing import Signer
>>> signer = Signer()
>>> value = signer.sign('hello, world!')
>>> value
'hello, world!:BYMlgvWMTSPLxC-DqxByleiMVXU'
>>> signer.unsign(value)
'hello, world!'
>>>
>>> signer = Signer(salt='1qaz2wsx')
>>> signer.sign('hello, world!')
'hello, world!:9vEvG6EA05hjMDB5MtUr33nRA_M'
>>>
>>> from django.core.signing import TimestampSigner
>>> signer = TimestampSigner()
>>> value = signer.sign('hello, world!')
>>> value

```

```
'hello, world!:1fpmcQ:STwj464IFE6eUB--hyUVF3d2So'
>>> signer.unsign(value, max_age=5)
Traceback (most recent call last):
File "<console>", line 1, in <module>
File "/Users/Hao/Desktop/fang.com/venv/lib/python3.6/site-packages/django/core/signing.py", line 109, in unsign
    'Signature age %s > %s seconds' % (age, max_age))
django.core.signing.SignatureExpired: Signature age 21.020604848861694 > 5 seconds
>>> signer.unsign(value, max_age=120)
'hello, world!'
```

## CSRF令牌和小工具

```
{% csrf_token %}
```

- @csrf\_exempt : 免除令牌
- @csrf\_protect : 提供令牌保护
- @require\_csrf\_token : 提供令牌保护
- @ensure\_csrf\_cookie : 强制视图发送带令牌的cookie

说明：可以在Chrome浏览器中安装EditThisCookie插件来方便的查看Cookie。

## 用户敏感信息的保护

### 1. 哈希摘要（签名）

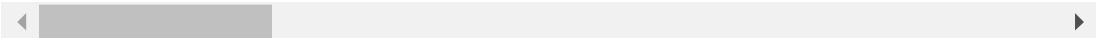
```
>>> import hashlib
>>>
>>> md5_hasher = hashlib.md5()
>>> md5_hasher.update('hello, world!'.encode())
>>> md5_hasher.hexdigest()
'3adbbad1791fbae3ec908894c4963870'
>>>
>>> sha1_hasher = hashlib.sha1()
>>> sha1_hasher.update('hello, world!'.encode())
>>> sha1_hasher.update('goodbye, world!'.encode())
>>> sha1_hasher.hexdigest()
'1f09d30c707d53f3d16c530dd73d70a6ce7596a9'
```

### 2. 加密和解密（对称加密和非对称加密）

```
pip install rsa
```

```
>>> pub_key, pri_key = rsa.newkeys(1024)
>>> message = 'hello, world!'
>>> crypto = rsa.encrypt(message.encode(), pub_key)
```

```
>>> crypto
b'0u{gH\x{a9}\x{a8}}0\x{e3}\x{1d}\x{052}|M\x{9d9?}\xd{c}\xd{8}\x{ec}{F}\xd{3}{v}\x{9b}\xde\x{8e}\x{12}\xe{6}{M}\xeb{v}
>>> origin = rsa.decrypt(crypto, pri_key).decode()
>>> origin
'hello, world!'
```



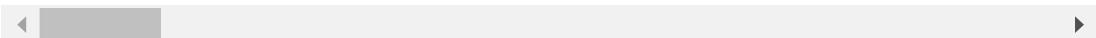
```
pip install pycrypto
```

### AES对称加密：

```
>>> from hashlib import md5
>>>
>>> from Crypto.Cipher import AES
>>> from Crypto import Random
>>>
>>> key = md5(b'mysecret').hexdigest()
>>> iv = Random.new().read(AES.block_size)
>>> str1 = '我爱你们！'
>>> str2 = AES.new(key, AES.MODE_CFB, iv).encrypt(str1)
b'p\x{96}\x{85}\x{0b}\x{c4}-Y\x{c4}\xb{cp}\n&'
>>> str3 = AES.new(key, AES.MODE_CFB, iv).decrypt(str2).decode()
'我爱你们！'
```

### RSA非对称加密：

```
>>> from Crypto.PublicKey import RSA
>>> # 生成密钥对
>>> key_pair = RSA.generate(2048)
>>> # 导入公钥
>>> pub_key = RSA.importKey(key_pair.publickey().exportKey())
>>> # 导入私钥
>>> pri_key = RSA.importKey(key_pair.exportKey())
>>> # 明文
>>> message1 = 'hello, world!'.encode()
>>> # 加密数据
>>> message2 = pub_key.encrypt(message1, None)
(b'\x{03}\x{86}\t\x{a0}\x{00}\x{c4}\x{ea}\x{d2}\x{80}\x{ed}\x{a7}\x{7f}\x{07}\x{ff}\x{88}\x{aa}\x{1e}\x{0cm}\x{0}\x{06}\x{a}
>>> # 解密数据
>>> message3 = pri_key.decrypt(message2)
'hello, world!'
```



## 安全相关建议

1. 虽然 Django 自带了稳固的安全保护措施，但是依然要采用正确的方式部署应用程序，利用 Web 服务器、操作系统和其他组件提供的安全保护措施。
2. 记得把 Python 代码放在 Web 服务器的文档根目录之外，避免代码意外泄露。

3. 谨慎处理用户上传的文件。
4. Django本身没有对请求次数加以限制（包括验证用户身份的请求），为了防止暴力攻击和破解，可以考虑使用具有一次消费性的验证码或对这类请求的次数进行限制。
5. 将缓存系统、数据库服务器以及重要的资源服务器都放在第二级防火墙之后（不要放在DMZ）。

## 测试相关

测试是发现和标记缺陷的过程。所谓的缺陷是指实际结果和期望结果之间的任何差别。有的地方，测试也被认为是执行以找出错误为目的的程序的过程。测试是为了让产品达到以下目标：

1. 满足需求用户满意
2. 改善产品的市场占有率
3. 树立对产品的信任
4. 减少开发和维护的成本

## 功能测试

如果一个软件单元的行为方式与它的开发规范完全一样，那么该软件单元就通过了它的功能测试。

- 白盒测试：开发人员自己实现，最基本的形式是单元测试，还有集成测试和系统测试。
- 黑盒测试：由开发团队之外的人执行，对测试代码没有可见性，将被测系统视为黑盒子。通常由测试人员或QA工程师来执行，Web应用可以通过Selenium这样的测试框架自动化实施。

## 性能测试

软件在高工作负载下对其响应性和健壮性展开的测试。

- 负载测试：在特定负载下执行的测试。
- 压力测试：突发条件或极限条件下的性能测试。

## 安全性测试

系统的敏感数据都是经过认证和授权之后才能访问。

## 其他测试

易用性测试 / 安装测试 / 可访问性测试

## 单元测试

测试函数和对象的方法（程序中最小最基本的单元）。通过对实际输出和预期输出的比对以及各种的断言条件来判定被测单元是否满足设计需求。

- 测试用例
- 测试固件 - 每次测试时都要使用的东西。
- 测试套件（测试集）- 组合了多个测试用例而构成的集合。

```
class UtilTest(TestCase):

    def setUp(self):
        self.pattern = re.compile(r'\d{6}')

    def test_gen_mobile_code(self):
        for _ in range(100):
            self.assertIsNotNone(self.pattern.match(gen_mobile_code()))

    def test_to_md5_hex(self):
        md5_dict = {
            '123456': 'e10adc3949ba59abbe56e057f20f883e',
            '123123123': 'f5bb0c8de146c67b44babbf4e6584cc0',
            '1qaz2wsx': '1c63129ae9db9c60c3e8aa94d3e00495',
        }
        for key, value in md5_dict.items():
            self.assertEqual(value, to_md5_hex(key))
```

TestCase 的断言方法：

- assertEquals / assertNotEqual
- assertTrue / assertFalse / assertIsNot
- assertRaise / assertRaiseRegexp
- assertAlmostEqual / assertNotAlmostEqual
- assertGreater / assertGreaterEqual / assertLess / assertLessEqual
- assertRegexpMatches / assertNotRegexpMatches
- assertListEqual / assertSetEqual / assertTupleEqual / assertDictEqual

可以使用nose2或pytest来辅助执行单元测试，同时通过cov-core或pytest-cov可以对测试覆盖度进行评估。覆盖率由百分比表示。比如测试代码执行过了程序的每一行，那么覆盖率就是100%。这种时候，几乎不会出现新程序上线后突然无法运行的尴尬情况。覆盖率不关心代码内容究竟是什么，覆盖率是用来检查“测试代码不足、测试存在疏漏”的一个指标，“测试内容是否妥当”并不归它管。

```
pip install nose2 pytest cov-core pytest-cov
```

可以使用Selenium来实现Web应用的自动化测试，它还可以用于屏幕抓取与浏览器行为模拟，通过爬虫抓取页面上的动态数据也可以使用它。Selenium其实包括三个部分：

- Selenium IDE：嵌入到浏览器的插件，可以录制和回放脚本。
- Selenium WebDriver：支持多种语言可以操控浏览器的API。
- Selenium Standalone Server：Selenium Grid、远程控制、分布式部署。

```
pip install selenium
```

```
from selenium import webdriver
import pytest
import contextlib

@pytest.fixture(scope='session')
def chrome():
    # 设置使用无头浏览器(不会打开浏览器窗口)
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')
    driver = webdriver.Chrome(options=options)
    yield driver
    driver.quit()

def test_baidu_index(chrome):
    chrome.get('https://www.baidu.com')
    assert chrome.title == '百度一下，你就知道'
```

除了Selenium之外，还有一个Web自动化测试工具名叫Robot Framework。

```
nose2 -v -C
pytest --cov
```

```
Ran 7 tests in 0.002s
```

OK	Name	Stmts	Miss	Cover
	example01.py	15	0	100%
	example02.py	49	49	0%
	example03.py	22	22	0%
	example04.py	61	61	0%
	example05.py	29	29	0%
	example06.py	39	39	0%
	example07.py	19	19	0%
	example08.py	27	27	0%
	example09.py	18	18	0%
	example10.py	19	19	0%

example11.py	22	22	0%
example12.py	28	28	0%
example13.py	28	28	0%
test_ddt_example.py	18	0	100%
test_pytest_example.py	11	6	45%
test_unittest_example.py	22	0	100%
<hr/>			
TOTAL	427	367	14%

在测试过程中需要孤立各种外部依赖（数据库、外部接口调用、时间依赖），具体又包括两个方面：

1. 数据源：数据本地化 / 置于内存中 / 测试之后回滚
2. 资源虚拟化：存根/桩（stub）、仿制/模拟（mock）、伪造（fake）
  - stub：测试期间为提供响应的函数生成的替代品
  - mock：代替实际对象（以及该对象的API）的对象
  - fake：没有达到生产级别的轻量级对象

## 集成测试

集成多个函数或方法的输入输出的测试，测试时需要将多个测试对象组合在一起。

- 测试组件互操作性 / 需求变更测试 / 外部依赖和API / 调试硬件问题 / 在代码路径中发现异常

## 系统测试

对需求的测试，测试成品是否最终满足了所有需求，在客户验收项目时进行。

## 数据驱动测试

使用外部数据源实现对输入值与期望值的参数化，避免在测试中使用硬编码的数据。

被测函数：

```
def add(x, y):
    return x + y
```

data.csv文件：

```
3,1,2
0,1,-1
100,50,50
100,1,99
15,7,8
```

测试代码：

```
import csv

from unittest import TestCase
from ddt import ddt, data, unpack

@ddt
class TestAdd(TestCase):

    def load_data_from_csv(filename):
        data_items = []
        with open(filename, 'r', newline='') as fs:
            reader = csv.reader(fs)
            for row in reader:
                data_items.append(list(map(int, row)))
        return data_items

    @data(*load_data_from_csv('data.csv'))
    @unpack
    def test_add(self, result, param1, param2):
        self.assertEqual(result, add(param1, param2))
```

## Django中的测试

1. 测试Django视图 - Django中提供的 `TestCase` 扩展了 `unittest` 中的 `TestCase`，绑定了一个名为 `client` 的属性，可以用来模拟浏览器发出的GET、POST、DELETE、PUT等请求。

```
class SomeViewTest(TestCase):

    def test_example_view(self):
        resp = self.client.get(reverse('index'))
        self.assertEqual(200, resp.status_code)
        self.assertEqual(5, resp.context['num'])
```

2. 运行测试 - 配置测试数据库。

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'HOST': 'localhost',
        'PORT': 3306,
        'NAME': 'DbName',
        'USER': os.environ['DB_USER'],
        'PASSWORD': os.environ['DB_PASS'],
        'TEST': {
            'NAME': 'DbName_for_testing',
```

```

        'CHARSET': 'utf8',
    },
}
}

python manage.py test
python manage.py test common
python manage.py test common.tests.UtilsTest
python manage.py test common.tests.UtilsTest.test_to_md5_hex

```

### 3. 评估测试覆盖度

```

pip install coverage
coverage run --source=<path1> --omit=<path2> manage.py test common
coverage report

```

Name	Stmts	Miss	Cover
<hr/>			
common/__init__.py	0	0	100%
common/admin.py	1	0	100%
common/apps.py	3	3	0%
common/forms.py	16	16	0%
common/helper.py	32	32	0%
common/middlewares.py	19	19	0%
common/migrations/__init__.py	0	0	100%
common/models.py	71	2	97%
common/serializers.py	14	14	0%
common/tests.py	14	8	43%
common/urls_api.py	3	3	0%
common/urls_user.py	3	3	0%
common/utils.py	22	7	68%
common/views.py	69	69	0%
<hr/>			
TOTAL	267	176	34%

## 性能测试

问题1：性能测试的指标有哪些？

1. ab ( Apache Benchmark ) / webbench / httpperf

```

yum -y install httpd
ab -c 10 -n 1000 http://www.baidu.com/
...
Benchmarking www.baidu.com (be patient).....done
Server Software:      BWS/1.1
Server Hostname:     www.baidu.com
Server Port:          80
Document Path:        /

```

```

Document Length:      118005 bytes
Concurrency Level:    10
Time taken for tests: 0.397 seconds
Complete requests:   100
Failed requests:     98
    (Connect: 0, Receive: 0, Length: 98, Exceptions: 0)
Write errors:         0
Total transferred:   11918306 bytes
HTML transferred:    11823480 bytes
Requests per second: 252.05 [#/sec] (mean)
Time per request:    39.675 [ms] (mean)
Time per request:    3.967 [ms] (mean, across all concurrent requests)
Transfer rate:       29335.93 [Kbytes/sec] received

Connection Times (ms)
                      min  mean[+/-sd] median   max
Connect:             6    7   0.6     7     9
Processing:          20   27  22.7    24   250
Waiting:             8   11  21.7     9   226
Total:               26   34  22.8    32   258

Percentage of the requests served within a certain time (ms)
 50%    32
 66%    34
 75%    34
 80%    34
 90%    36
 95%    39
 98%    51
 99%   258
100%   258 (longest request)

```

## 2. mysqlslap

```

mysqlslap -a -c 100 -h 1.2.3.4 -u root -p
mysqlslap -a -c 100 --number-of-queries=1000 --auto-generate-sql-load-type=read -t
mysqlslap -a --concurrency=50,100 --number-of-queries=1000 --debug-info --auto-ge

```

## 3. sysbench

```

sysbench --test=threads --num-threads=64 --thread-yields=100 --thread-locks=2 run
sysbench --test=memory --num-threads=512 --memory-block-size=256M --memory-total-s

```

## 4. jmeter

请查看 [《使用JMeter进行性能测试》](#)。

## 5. LoadRunner / QTP

## 项目调试

可以使用django-debug-toolbar来辅助项目调试。

### 1. 安装

```
pip install django-debug-toolbar
```

### 2. 配置 - 修改settings.py。

```
INSTALLED_APPS = [
    'debug_toolbar',
]

MIDDLEWARE = [
    'debug_toolbar.middleware.DebugToolbarMiddleware',
]

DEBUG_TOOLBAR_CONFIG = {
    # 引入jQuery库
    'JQUERY_URL': 'https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js',
    # 工具栏是否折叠
    'SHOW_COLLAPSED': True,
    # 是否显示工具栏
    'SHOW_TOOLBAR_CALLBACK': lambda x: True,
}
```

### 3. 配置 - 修改urls.py。

```
if settings.DEBUG:

    import debug_toolbar

    urlpatterns.insert(0, path('__debug__/', include(debug_toolbar.urls)))
```

### 4. 使用 - 在页面右侧可以看到一个调试工具栏，上面包括了执行时间、项目设置、请求头、SQL、静态资源、模板、缓存、信号等调试信息，查看起来非常的方便。

## 部署相关

请参考《Django项目上线指南》。

## 性能相关

网站优化两大定律：

1. 尽可能的使用缓存 - 牺牲空间换取时间（普适策略）。

2. 能推迟的都推迟 - 使用消息队列将并行任务串行来缓解服务器压力。

- 服务器CPU利用率出现瞬时峰值 - 削峰 ( CPU利用率平缓的增长 )
- 上下游节点解耦合 ( 下订单和受理订单的系统通常是分离的 )

## Django框架

1. 配置缓存来缓解数据库的压力，并有合理的机制应对[缓存穿透和缓存雪崩](#)。

2. 开启[模板缓存](#)来加速模板的渲染。

```
TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [os.path.join(BASE_DIR, 'templates'), ],
    # 'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
        'loaders': [
            ('django.template.loaders.cached.Loader', [
                'django.template.loaders.filesystem.Loader',
                'django.template.loaders.app_directories.Loader',
            ], ),
        ],
    },
},
]
```

3. 用惰性求值、迭代器、`defer()`、`only()` 等缓解内存压力。

4. 用 `select_related()` 和 `prefetch_related()` 执行预加载避免“1+N查询问题”。

## 数据库

1. 用ID生成器代替自增主键 ( 性能更好、适用于分布式环境 )。

- 自定义ID生成器
- UUID

```
>>> my_uuid = uuid.uuid1()
>>> my_uuid
UUID('63f859d0-a03a-11e8-b0ad-60f81da8d840')
>>> my_uuid.hex
'63f859d0a03a11e8b0ad60f81da8d840'
```

2. 避免不必要的外键列上的约束（除非必须保证参照完整性），更不要使用触发器之类的机制。
3. 使用索引来优化查询性能（索引放在要用于查询的字段上）。InnoDB用的是BTREE索引，使用>、<、>=、<=、BETWEEN或者LIKE 'pattern'（pattern不以通配符开头）时都可以用到索引。因为建立索引需要额外的磁盘空间，而主键上是有默认的索引，所以主键要尽可能选择较短的数据类型来减少磁盘占用，提高索引的缓存效果。

```
create index idx_goods_name on tb_goods (gname(10));
```

```
-- 无法使用索引
select * from tb_goods where gname like '%iPhone%';
-- 可以使用索引
select * from tb_goods where gname like 'iPhone%';
```

```
# 无法使用索引
Goods.objects.filter(name__icontains='iPhone')
# 可以使用索引
Goods.objects.filter(name__istartswith='iPhone');
```

4. 使用存储过程（存储在服务器端编译过的一组SQL语句）。

```
drop procedure if exists sp_avg_sal_by_dept;

create procedure sp_avg_sal_by_dept(deptno integer, out avg_sal float)
begin
    select avg(sal) into avg_sal from TbEmp where dno=deptno;
end;

call sp_avg_sal_by_dept(10, @a);

select @a;
```

```
>>> from django.db import connection
>>> cursor = connection.cursor()
>>> cursor.callproc('sp_avg_sal_by_dept', (10, 0))
>>> cursor.execute('select @_sp_avg_sal_by_dept_1')
>>> cursor.fetchone()
(2675.0,)
```

5. 使用数据分区。通过分区可以存储更多的数据、优化查询更大的吞吐量、可以快速删除过期的数据。关于这个知识点可以看看MySQL的[官方文档](#)。

- RANGE分区：基于连续区间范围，把数据分配到不同的分区。
- LIST分区：基于枚举值的范围，把数据分配到不同的分区。
- HASH分区 / KEY分区：基于分区个数，把数据分配到不同的分区。

```

CREATE TABLE tb_emp (
    eno INT NOT NULL,
    ename VARCHAR(20) NOT NULL,
    job VARCHAR(10) NOT NULL,
    hiredate DATE NOT NULL,
    dno INT NOT NULL
)
PARTITION BY HASH(dno)
PARTITIONS 4;

CREATE TABLE tb_emp (
    eno INT NOT NULL,
    ename VARCHAR(20) NOT NULL,
    job VARCHAR(10) NOT NULL,
    hiredate DATE NOT NULL,
    dno INT NOT NULL
)
PARTITION BY RANGE( YEAR(hiredate) ) (
    PARTITION p0 VALUES LESS THAN (1960),
    PARTITION p1 VALUES LESS THAN (1970),
    PARTITION p2 VALUES LESS THAN (1980),
    PARTITION p3 VALUES LESS THAN (1990),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);

```

## 6. 使用 explain 来分析查询性能 - 执行计划。

```
explain select * from ...;
```

explain 结果解析：

- select\_type：表示select操作的类型，常见的值有SIMPLE（简单查询，没有使用子查询或者表连接查询）、PRIMARY（主查询，外层的查询）、UNION（并集操作中的第二个或者后面的查询）、SUBQUERY（子查询中的第一个SELECT）等。
- table：输出结果的表。
- type：MySQL在表中找到所需行的方式，也称为访问类型，常见的值有：
  - ALL：全表扫描（遍历全表找到匹配的行）
  - index：索引全扫描（遍历整个索引）
  - range：索引范围扫描
  - ref：非唯一索引扫描或唯一索引的前缀扫描

- eq\_ref : 唯一索引扫描
- const / system : 表中最多有一行匹配
- NULL : 不用访问表或者索引
- possible\_keys : 查询时可能用到的索引。
- key : 实际使用的索引。
- key\_len : 使用到索引字段的长度。
- rows : 扫描行的数量。
- Extra : 额外的信息 ( 执行情况的说明或描述 ) 。

说明 : 关于 MySQL 更多的知识尤其是性能调优和运维方面的内容 , 推荐大家阅读网易出品的《深入浅出 MySQL ( 第 2 版 ) 》 , 网易出品必属精品。

## 7. 使用慢查询日志来发现性能低下的查询。

```
mysql> show variables like 'slow_query%';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| slow_query_log     | OFF    |
| slow_query_log_file | /mysql/data/localhost-slow.log |
+-----+-----+
```

```
mysql> show variables like 'long_query_time';
+-----+-----+
| Variable_name | Value   |
+-----+-----+
| long_query_time | 10.000000 |
+-----+-----+
```

```
mysql> set global slow_query_log='ON';
mysql> set global long_query_time=1;
```

```
[mysqld]
slow_query_log=ON
slow_query_log_file=/usr/local/mysql/data/slow.log
long_query_time=1
```

## 其他

请参考《Python 性能调优》。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day91-100 / Docker简易上手指南.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



251 lines (174 sloc) 9.96 KB

# Docker简易上手指南

## Docker简介

软件开发中最为麻烦的事情可能就是配置环境了。由于用户使用的操作系统具有多样性，即便使用跨平台的开发语言（如Java和Python）都不能保证代码能够在各种平台下都可以正常的运转，而且可能在不同的环境下我们的软件需要依赖的其他软件包也是不一样的。

那么问题来了，我们安装软件的时候可不可以把软件运行的环境一并安装？我们是不是可以把原始环境一模一样地复制过来呢？

虚拟机（virtual machine）就是带环境安装的一种解决方案，它可以在一种操作系统里面运行另一种操作系统，比如在Windows系统里面运行Linux系统，在macOS上运行Windows，而应用程序对此毫无感知。使用过虚拟机的人都知道，虚拟机用起来跟真实系统一模一样，而对于虚拟机的宿主系统来说，虚拟机就是一个普通文件，不需要了就删掉，对宿主系统或者其他的应用程序并没有影响。但是虚拟机通常会占用较多的系统资源，启动和关闭也非常的缓慢，总之用户体验没有想象中的那么好。

Docker属于对Linux容器技术的一种封装（利用了Linux的namespace和cgroup技术），它提供了简单易用的容器使用接口，是目前最流行的Linux容器解决方案。Docker将应用程序与该程序的依赖打包在一个文件里面，运行这个文件，就会生成一个虚拟容器。程序在这个虚拟容器里运行，就好像在真实的物理机上运行一样。有了Docker就再也不用担心环境问题了。

目前，Docker主要用于以下几个方面：

1. 提供一次性的环境。
2. 提供弹性的云服务（利用Docker很容易实现扩容和收缩）。
3. 实践微服务架构（隔离真实环境在容器中运行多个服务）。

## 安装Docker

下面的讲解以CentOS为例，使用Ubuntu、macOS或Windows的用户可以通过点击链接了解这些平台下如何安装和使用Docker。

0. 确定操作系统内核版本（CentOS 7要求64位，内核版本3.10+；CentOS 6要求64位，内核版本2.6+）。

```
uname -r
```

1. 在CentOS下使用yum安装Docker并启动。

```
yum -y install docker-io  
systemctl start docker
```

2. 检视Docker的信息和版本。

```
docker version  
docker info
```

3. 运行Hello-World项目来测试Docker。第一次运行时由于本地没有hello-world的镜像因此需要联网进行下载。

```
docker run hello-world
```

也可以先用下面的命令下载镜像，然后再来运行。

```
docker pull <name>
```

4. 运行镜像文件。

```
docker run <image-id>  
docker run -p <port1>:<port2> <name>
```

6. 查看镜像文件。

```
docker image ls  
docker images
```

7. 删除镜像文件。

```
docker rmi <name>
```

8. 查看正在运行容器。

```
docker ps
```

9. 停止运行的容器。

```
docker stop <container-id>
docker stop <name>
```

对于那些不会自动终止的容器，就可以用下面的方式来停止。

```
docker container kill <container-id>
```

在Ubuntu（内核版本3.10+）下面安装和启动Docker，可以按照如下的步骤进行。

```
apt update
apt install docker-ce
service docker start
```

在有必要的情况下，可以更换Ubuntu软件下载源来提升下载速度，具体的做法请参照<https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu/>。

安装Docker后，由于直接访问dockerhub下载镜像会非常缓慢，建议更换国内镜像，可以通过修改 /etc/docker/daemon.json 文件来做到。如果使用云服务器（如：阿里云），通常云服务器提供商会提供默认的镜像服务器，并不需要手动进行指定。

```
{
  "registry-mirrors": [
    "http://hub-mirror.c.163.com",
    "https://registry.docker-cn.com"
  ]
}
```

## 使用Docker

### 安装Nginx

下面我们就基于Docker来运行一台HTTP服务器，我们选择用Nginx来搭建该服务，因为Nginx是高性能的Web服务器，同时也是做反向代理服务器的上佳选择。要做到这件事情，只需要下面这条命令在Docker中创建一个容器即可。

```
docker container run -d -p 80:80 --rm --name mynginx nginx
```

说明：上面的参数 `-d` 表示容器在后台运行（不产生输出到Shell）并显示容器的ID；`-p` 是用来映射容器的端口到宿主机的端口；`--rm` 表示容器停止后自动删除容器，例如通过 `docker container stop mynginx` 以后，容器就没有了；`--name` 后面的 `mynginx` 就是自定义容器的名字。在创建容器的过程中，需要用到nginx的镜像文件，镜像文件的下载是自动完成的，如果没有指定版本号，默认是最新版本（`latest`）。

如果需要将自己的Web项目（页面）部署到Nginx上，可以使用容器拷贝命令将指定路径下所有的文件和文件夹拷贝到容器的指定目录中。

```
docker container cp /root/web/index.html mynginx:/usr/share/nginx/html
```

如果不愿意拷贝文件也可以在创建容器时通过数据卷操作 `--volume` 将指定的文件夹映射到容器的某个目录中，例如将Web项目的文件夹直接映射到 `/usr/share/nginx/html` 目录。

```
docker container run -d -p 80:80 --rm --name mynginx --volume $PWD/html:/usr/share/nginx/html
```

说明：上面创建容器和拷贝文件的命令中，`container` 是可以省略的，也就是说 `docker container run` 和 `docker run` 是一样的，而 `docker container cp` 和 `docker cp` 是一样的。此外，命令中的 `--volume` 也可以缩写为 `-v`，就如同 `-d` 是 `--detach` 的缩写，`-p` 是 `--publish` 的缩写。`$PWD` 代表宿主系统当前文件夹，这个用过Linux系统的人相信很容易理解。

要查看运行中的容器，可以使用下面的命令。

```
docker ps
```

要让刚才创建的容器停下来，可以使用下面的命令。

```
docker stop mynginx
```

由于在创建容器时使用了 `--rm` 选项，容器在停止时会被移除，当我们使用下面的命令查看所有容器时，应该已经看不到刚才的 `mynginx` 容器了。

```
docker container ls -a
```

如果在创建容器时没有指定 `--rm` 选项，那么也可以使用下面的命令来删除容器。

```
docker rm mynginx
```

## 安装MySQL

我们可以先检查一下服务器上有没有MySQL的镜像文件。

```
docker search mysql
```

下载MySQL镜像并指定镜像的版本号。

```
docker pull mysql:5.7
```

如果需要查看已经下载的镜像文件，可以使用下面的命令。

```
docker images
```

创建并运行MySQL容器。

```
docker run -d -p 3306:3306 --name mysql57 -v $PWD/mysql/conf:/etc/mysql/mysql.cnf.d -v
```

注意，上面创建容器时我们又一次使用了数据卷操作，那是因为通常容器是随时创建随时删除的，而数据库中的数据却是需要保留下来的，所以上面的两个数据卷操作一个是映射了MySQL配置文件所在的文件夹，一个是映射了MySQL数据所在的文件夹，这里的数据卷操作非常重要。我们可以将MySQL的配置文件放在 \$PWD/mysql/conf 目录下，配置文件的具体内容如下所示：

```
[mysqld]
pid-file=/var/run/mysqld/mysqld.pid
socket=/var/run/mysqld/mysqld.sock
datadir=/var/lib/mysql
log-error=/var/log/mysql/error.log
server-id=1
log-bin=/var/log/mysql/mysql-bin.log
expire_logs_days=30
max_binlog_size=256M
symbolic-links=0
```

如果安装了MySQL 8.x版本（目前的最新版本），在使用客户端工具连接服务器时可能会遇到“error 2059: Authentication plugin 'caching\_sha2\_password' cannot be loaded”的问题，这是因为MySQL 8.x默认使用了名为“caching\_sha2\_password”的机制对用户口令进行了更好的保护，但是如果客户端工具不支持新的认证方式，连接就会失败。解决这个问题有两种方式：一是升级客户端工具来支持MySQL 8.x的认证方式；二是进入容器，修改MySQL的用户口令认证方式。下面是具体的步骤，我们先用 `docker exec` 命令进入容器的交互式环境，假设运行MySQL 8.x的容器名字叫 `mysql8x`。

```
docker exec -it mysql8x /bin/bash
```

进入容器的交互式Shell之后，可以首先利用MySQL的客户端工具连接MySQL服务器。

```
mysql -u root -p
Enter password:
Your MySQL connection id is 16
Server version: 8.0.12 MySQL Community Server - GPL
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

接下来通过SQL来修改用户口令就可以了。

```
alter user 'root'@'%' identified with mysql_native_password by '123456' password expire
```

当然，如果愿意你也可以查看一下用户表检查是否修改成功。

```
use mysql;
select user, host, plugin, authentication_string from user where user='root';
+-----+-----+-----+
| user | host      | plugin           | authentication_string
+-----+-----+-----+
| root | %        | mysql_native_password | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9
| root | localhost | mysql_native_password | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9
+-----+-----+-----+
2 rows in set (0.00 sec)
```

在完成上面的步骤后，现在即便不更新客户端工具也可以连接MySQL 8.x了。

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Day91-100 / MySQL相关知识.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



136 lines (94 sloc) 2.38 KB

# MySQL相关知识

## 存储引擎

1. InnoDB
2. MyISAM

## 数据类型

## 索引

### 索引的类型

1. B-Tree索引
2. HASH索引
3. R-Tree索引 ( 空间索引 )
4. Full-text索引 ( 全文索引 )

## 视图

查询的快照，可以将访问权限控制到列上。

```
create view ... as select ...
drop view ...
```

## 存储过程

```
create procedure ... (params)
begin
...
end
```

```
end;  
  
call ...  
  
cursor.callproc('...')
```

## 触发器

不能用，因为多个行锁可能直接升级为表锁，导致性能低下。

## 事务控制

## SQL注入攻击

## 数据分区

## SQL优化

### 优化步骤

1. 通过 `show status` 了解各种SQL的执行频率。

```
show status like 'com_%';  
show status like 'innodb_%';  
show status like 'connections';  
show status like 'slow_queries';
```

2. 定位低效率的SQL语句 - 慢查询日志。

```
show processlist
```

3. 通过 `explain` 了解SQL的执行计划。

- `select_type` : 查询类型 ( simple、primary、union、subquery )
- `table` : 输出结果集的表
- `type` : 访问类型 ( ALL、index、range、ref、eq\_ref、const、NULL )
- `possible_keys` : 查询时可能用到的索引
- `key` : 实际使用的索引
- `key_len` : 索引字段的长度
- `rows` : 扫描的行数
- `extra` : 额外信息

4. 通过 `show profiles` 和 `show profile for query` 分析SQL。

## SQL优化

1. 优化`insert`语句
2. 优化`order by`语句
3. 优化`group by`语句
4. 优化嵌套查询
5. 优化`or`条件
6. 优化分页查询
7. 使用SQL提示
  - `USE INDEX`
  - `IGNORE INDEX`
  - `FORCE INDEX`

## 配置优化

1. 调整`max_connections`
2. 调整`back_log`
3. 调整`table_open_cache`
4. 调整`thread_cache_size`
5. 调整`innodb_lock_wait_timeout`

## 架构优化

1. 通过拆分提高表的访问效率
  - 垂直拆分
  - 水平拆分
2. 逆范式理论
  - 数据表设计的规范程度称之为范式 ( Normal Form )
    - 1NF : 列不能再拆分
    - 2NF : 所有的属性都依赖于主键
    - 3NF : 所有的属性都直接依赖于主键 ( 消除传递依赖 )
    - BCNF : 消除非平凡多值依赖
3. 使用中间表提高统计查询速度

## 数据备份

### 导入和导出

1. `select ... into outfile ...`
2. `load data infile ... into table ...`

3. mysqldump
4. mysqlimport

**ibbackup工具**

**xtrabackup工具**

**主从复制**

**集群**

Branch: master ▾

[Find file](#) [Copy path](#)

[Python-100-Days](#) / [Day91-100](#) / **关于测试.md**

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



74 lines (42 sloc) 6.06 KB

# 关于测试

## 软件测试概述

软件测试是一种用来促进鉴定软件的正确性、完整性、安全性和品质的过程，也就是在规定的条件下对程序进行操作以发现程序中的错误，衡量软件的品质并对其是否能满足设计要求进行评估的过程。

### 测试的方法

黑盒测试：测试应用程序的功能，而不是其内部结构或运作。测试者不需具备应用程序的代码、内部结构和编程语言的专门知识。测试者只需知道什么是系统应该做的事，即当键入一个特定的输入，可得到一定的输出。测试案例是依应用系统应该做的功能，照规范、规格或要求等设计。测试者选择有效输入和无效输入来验证是否正确的输出。此测试方法可适合大部分的软件测试，例如集成测试和系统测试。

白盒测试：测试应用程序的内部结构或运作，而不是测试应用程序的功能（即黑箱测试）。在白箱测试时，以编程语言的角度来设计测试案例。测试者输入数据验证数据流在程序中的流动路径，并确定适当的输出，类似测试电路中的节点。

### 测试的种类（阶段）

单元测试：对软件组成单元进行测试，其目的是检验软件基本组成单位的正确性，测试的对象是软件设计的最小单位 - 函数。

集成测试：将程序模块采用适当的集成策略组装起来，对系统的接口及集成后的功能进行正确性检测的测试工作。其主要目的是检查软件单位之间的接口是否正确，集成测试的对象是已经经过单元测试的模块。

系统测试：系统测试主要包括功能测试、界面测试、可靠性测试、易用性测试、性能测试。

回归测试：为了检测代码修改而引入的错误所进行的测试活动。回归测试是软件维护阶段的重要工作，有研究表明，回归测试带来的耗费占软件生命周期的1/3总费用以上。

## 测试驱动开发

测试驱动开发包括以下三个步骤：

1. 为未实现的新功能或者改进编写自动化测试。
2. 提供通过所有定义的测试的最小代码量。
3. 重构代码以满足所需的质量标准。

测试驱动开发的好处在于可以有效的防止软件回归以及提供更有质量的代码。

Python的标准库里有为编写单元测试而准备的unittest模块，执行测试时建议使用[pytest](#)或nose2。pytest是一款能够自动搜索并执行测试的测试执行工具，并且会输出详细的错误报告。关于单元测试可以看看[《Python必会的单元测试框架 - unittest》](#)。

可以安装[testfixtures](#)库来辅助单元测试，它整合了多种典型配置器，提供了生成目录、更改系统日期、生成mock对象的功能模块，这些模块能够帮助我们将单元测试与单元测试所依赖的环境分离开。[mock](#)是将测试对象所依赖的对象替换为虚拟对象的库，在测试的时候，我们可以为虚拟对象指定其在被调用时的返回值以及是否发生异常等。

[WebTest](#)是用于Web应用功能测试的库。它会对WSGI应用执行模拟请求并获取结果。基本上所有WSGI应用的测试都可以用它。

tox能便捷地为我们准备好执行测试所需的环境。tox会在多个virtualenv环境中搭建测试环境，然后在这些环境中执行测试并显示结果。它能够把测试工具的选项及环境变量等内容统一起来，所以我们只需执行tox命令即能轻松完成所需的测试。

## Selenium/Robot Framework

Selenium是实现Web应用程序的功能测试以及集成测试自动化的浏览器驱动测试工具群。和使用浏览器的用户相同，Selenium可以在浏览器进行的鼠标操作、在表单中输入文字、验证表单的值等，利用这一点就可以将手动操作变成自动化操作。

### Selenium优点

1. 自动化测试用例制作简单。Selenium提供了Selenium IDE工具，该工具可以捕获鼠标、键盘的操作，然后通过重放功能来重复这些操作，这样就可以简单的制作测试用例。
2. 支持多种浏览器和操作系统。

### Selenium的组件

1. [Selenium IDE](#)。
2. [Selenium Remote Control](#)。

### 3. Selenium WebDriver。

## 与持续集成工具协作

持续集成指的是频繁的将代码集成到主干。它的好处主要有两个：

1. 快速发现错误。每完成一点更新，就集成到主干，可以快速发现错误，定位错误也比较容易。
2. 防止分支大幅偏离主干。如果不是经常集成，主干又在不断更新，会导致以后集成的难度变大，甚至难以集成。

持续集成的目的，就是让产品可以快速迭代，同时还能保持高质量。它的核心措施是代码集成到主干之前，必须通过自动化测试，只要有一个测试用例失败，就不能集成。编程大师Martin Fowler曾经说过：“持续集成并不能消除Bug，而是让它们非常容易发现和改正。”

可以在Jenkins中安装“Seleniumhq Plugin”插件，这样就可以将Selenium IDE制作的测试用例保存为HTML格式并提供给Jenkins来使用，基本步骤是：

1. 在执行测试的机器上，从版本控制系统中下载测试套件和测试用例。
2. 在执行测试的机器上下载Selenium Server。
3. 从Jenkins的“系统管理”中选择“插件管理”来安装“Seleniumhq Plugin”。
4. 在Jenkins的“系统管理”中选择“系统设置”并配置“Selenium Remote Control”下的“HTMLSuite Runner”。
5. 新建测试用的Jenkins任务并进行配置，配置的内容包括：浏览器、起始URL、测试套件和测试结果输出文件。

配置完成后，就可以执行Jenkins的“立即构建”了。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day91-100 / 团队项目开发.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



233 lines (144 sloc) 12.6 KB

## 团队项目开发

我们经常听到个人开发和团队开发这两个词，所谓个人开发就是一个人把控产品的所有内容；而团队开发则是由多个人组成团队并完成产品的开发。要实施团队开发以下几点是必不可少的：

1. 必须对开发过程中的各种事件（例如：谁到什么时间完成了什么事情）进行管理和共享。
2. 各类工作成果以及新的知识技巧等必须在团队内部共享。
3. 管理工作成果的变更，既要防止成果被破坏，又要保证各个成员利用现有成果并行作业。
4. 能够证明团队开发出的软件在任何时候都是可以正常运行的。
5. 尽可能的使用自动化的工作流程，让团队成员能够正确的实施开发、测试和部署。

## 团队项目开发常见问题

### 问题1：传统的沟通方式无法确定处理的优先级

例如：使用邮件进行沟通可能出现邮件数量太多导致重要的邮件被埋没，无法管理状态，不知道哪些问题已经解决，哪些问题尚未处理，如果用全文检索邮件的方式来查询相关问题效率过于低下。

解决方案：使用缺陷管理工具。

### 问题2：没有能够用于验证的环境

例如：收到项目正式环境中发生的故障报告后，需要还原正式环境需要花费很长的时间。

解决方法：实施持续交付。

### 问题3：用别名目录管理项目分支

解决方法：实施版本控制。

#### **问题4：重新制作数据库非常困难**

例如：正式环境和开发环境中数据库表结构不一致或者某个表列的顺序不一致。

解决方法：实施版本控制。

#### **问题5：不运行系统就无法察觉问题**

例如：解决一个bug可能引入其他的bug或者造成系统退化，不正确的使用版本系统覆盖了其他人的修改，修改的内容相互发生了干扰，如果问题不能尽早发现，那么等过去几个月后再想追溯问题就非常麻烦了。

解决方法：实施持续集成，将团队成员的工作成果经常、持续的进行构建和测试。

#### **问题6：覆盖了其他成员修正的代码**

解决方法：实施版本控制。

#### **问题7：无法实施代码重构**

重构：在不影响代码产生的结果的前提下对代码内部的构造进行调整。

例如：在实施代码重构时可能引发退化。

解决方法：大量的可重用的测试并实施持续集成。

#### **问题8：不知道bug的修正日期无法追踪退化**

解决方法：版本控制系统、缺陷管理系统和持续集成之间需要交互，最好能够和自动化部署工具集成到一起来使用。

#### **问题9：发布过程太复杂**

解决方法：实施持续交付。

基于对上述问题的阐述和分析，我们基本上可以得到以下的结论，在团队开发中版本控制、缺陷管理和持续集成都是非常的重要且不可或缺的。

## **版本控制**

针对上面提到的一些问题，在团队开发的首要前提就是实施版本控制，对必要的信息进行管理，需要管理的内容包括：

1. 代码。
2. 需求和设计的相关文档。
3. 数据库模式和初始数据。

4. 配置文件。
5. 库的依赖关系定义。

## Git简介



Git是诞生于2005年的一个开源分布式版本控制系统，最初是Linus Torvalds（Linux之父）为了帮助管理Linux内核开发而开发的一个版本控制软件。Git与常用的版本控制工具Subversion等不同，它采用了分布式版本控制的方式，在没有中央服务器支持的环境下也能够实施版本控制。

对于有使用Subversion（以下简称为SVN）经验的人来说，Git和SVN一样摒弃了基于锁定模式的版本控制方案（早期的CVS和VSS使用的就是锁定模式）采用了合并模式，而二者的区别在于：1. Git是分布式的，SVN是集中式的，SVN需要中央服务器才能工作。2. Git把内容按元数据方式存储，而SVN是按文件，即把文件的元信息隐藏在一个.svn文件夹里。3. Git分支和SVN的分支不同。4. Git没有一个全局版本号而SVN有。5. Git的内容完整性要优于SVN，Git的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

## 安装Git

可以在[Git官方网站](#)找到适合自己系统的Git下载链接并进行安装，安装成功后可以在终端中键入下面的命令检查自己的Git版本。

```
git --version
```

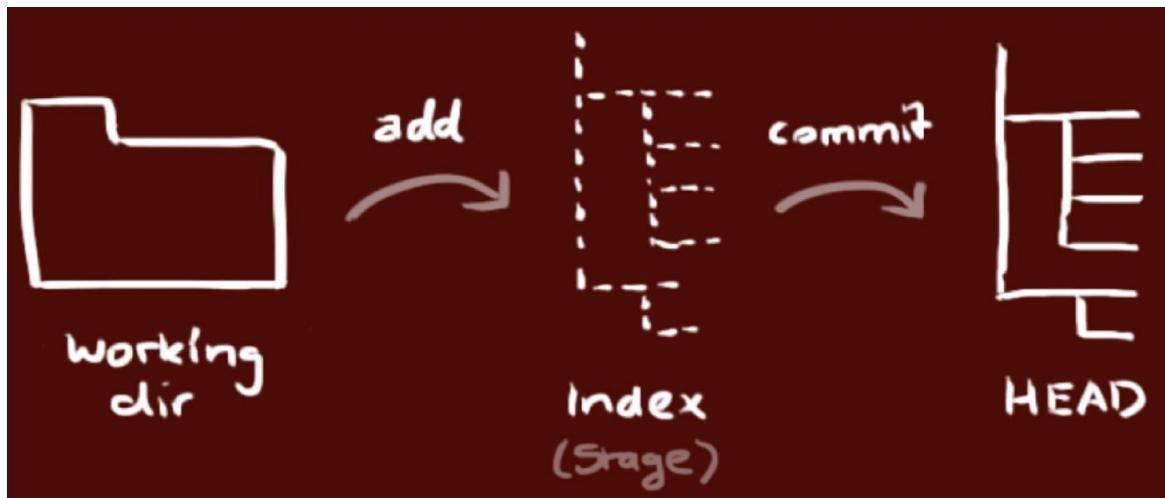
如果之前完全没有接触过Git，可以先阅读[《git - 简易指南》](#)来对Git有一个大致的了解。

## 本地实施版本控制

可以使用下面的命令将目录创建为Git仓库。

```
git init
```

当你完成了上述操作后，本地目录就变成了下面的样子，左边是你正在操作的工作目录，而右边是你的本地仓库，中间是工作目录和本地仓库之间的一个暂存区（也称为缓存区）。



通过 `git add` 可以将文件添加到暂存区。

```
git add <file> ...
```

可以用下面的方式将暂存区的指定文件恢复到工作区。

```
git checkout -- <file>
```

通过下面的命令可以将暂存区的内容纳入本地仓库。

```
git commit -m '本次提交的说明'
```

可以使用下面的命令查看文件状态和进行版本比较。

```
git status -s  
git diff
```

可以通过 `git log` 查看提交日志。

```
git log  
git log --graph --pretty=oneline --abbrev-commit
```

如果要回到历史版本，可以使用下面的命令。

```
git reset --hard <commit-id>  
git reset --hard HEAD^
```

其他的一些命令可以参考阮一峰老师的[《常用Git命令清单》](#)或者是码云上的[《Git大全》](#)。

## Git服务器概述

对于Git来说不像SVN那样一定需要一个中心服务器，刚才我们的操作都是在本地执行的，如果你想通过Git分享你的代码或者与其他人协作，那么就需要服务器的支持。Github为Git提供了远程仓库，它是一个基于Git的代码托管平台，企业用户（付费用户）可以创建仓库，普通用户只能创建公开仓库（代码是可以是他人可见的）。Github是在2008年4月创办的，它上面代码库惊人的增长速度已经证明了它是非常成功的，在2018年6月，微软以75亿美元的天价收购了Github。国内也有类似的代码托管平台，最有名的当属[码云](#)和[CODING](#)，目前码云和CODING对注册用户都提供了受限的使用私有仓库的功能，同时还提供了对Pull Request的支持（后面会讲到），而且目前提供代码托管服务的平台都集成了“缺陷管理”、“WebHook”等一系列的功能，让我们能做的事情不仅仅是版本控制。当然，如果公司需要也可以搭建自己的Git服务器，具体的方式我们就不在这里进行介绍了，有兴趣的可以自行了解。



我们可以在码云或者CODING上注册账号，也可以使用第三方登录（github账号、微信账号、新浪微博账号、CSDN账号等）的方式。登录成功后就可以创建项目，创建项目几乎是“傻瓜式”的，我们只说几个值得注意的地方。

1. 添加项目成员。创建项目后，可以在项目的“设置”或“管理”中找到“成员管理”功能，这样就可以将其他开发者设置为项目团队的成员，项目成员通常分为“所有者”、“管理者”、“普通成员”和“受限成员”几种角色。
2. 设置公钥实现免密操作。在项目的“设置”或“管理”中我们还可以找到“部署公钥管理”的选项，通过添加部署公钥，可以通过SSH（安全远程连接）的形式访问服务器而不用每次输入用户名和口令。可以使用 `ssh-keygen` 命令来创建密钥对。

```
ssh-keygen -t rsa -C "your_email@example.com"
```

## 使用Git进行开发

克隆服务器上的代码到本地机器。

```
git clone <url>
```

在自己的分支上进行开发。

```
git branch <branch-name>
git checkout <branch-name>
```

或者

```
git checkout -b <branch-name>
```

接下来可以先在本地实施版本控制（操作的方式与前面相同不再赘述），然后再将自己的分支Push到服务器。

```
git push origin <branch-name>
```

最后，当工作完成时，可以发起一个Pull Request，请求将代码合并到master分支。

## 分支策略的模式

上面讲解的方式，其实是一种称为github-flow的分支策略模式，这种模式的操作步骤包括：

1. master的内容都是可以进行发布的内容。
2. 开发时应该以master为基础建立新分支。
3. 分支先在本地实施版本控制，然后以同名分支定期向服务器进行Push。
4. 开发结束后向master发送Pull Request。
5. Pull Request通过代码审查之后合并到master，并从master向正式环境发布。

在使用github-flow时的注意事项有以下三点：

1. master是用于发布的，不能直接在master上进行修改。
2. 开始日常开发工作时要首先建立分支。
3. 工作完成后向master发送Pull Request。

除了上述的github-flow工作方式外，还有一种名为git-flow的分支策略模式，它借鉴了中央集权型版本控制系统的长处，为团队内部统一管理建立分支的方法、合并操作和关闭分支的方法。在这种模式下，项目有两个长线分支，分别是master和develop，其他的都是临时的、短暂的辅助分支，包括feature（开发特定功能的分支，开发结束后合并到develop）、release（从develop分离出来的为发布做准备的分支，发布结束后合并到master和develop）和hotfix（产品发布后出现问题时紧急建立的分支，直接从master分离，问题修复后合并到master并打上标签，同时还要合并到develop来避免将来的版本遗漏了这个修复工作，如果此时有正在发布的release分支，还要合并到release分支）。这套方式分支策略简单清晰且容易理解，但是在运用上会稍微有些复杂，需要一些脚本来辅助版本控制的实施。

## 缺陷管理

没有好的团队管理工具必然导致项目进展不顺利，任务管理困难，而引入缺陷管理系统正好可以解决这些问题，通常一个缺陷管理系统都包含了以下的功能：

1. 任务管理（包括必须做什么、谁来做、什么时候完成、现在处于什么状态等）。
2. 直观而且可以检索过去发生的各种问题。
3. 能够对信息进行统一的管理和共享。
4. 能够生成各类报表。
5. 能够关联到其他系统，具有可扩展性。

Redmine是基于Ruby on Rails框架的开源缺陷管理系统，提供了问题管理、代码管理、Wiki等必要的功能，而且支持插件系统，扩展起来也非常容易。

The screenshot shows the 'Create Issue' page of the Redmine application. The top navigation bar includes links for Overview, Activities, Issues, New Issue (highlighted), Gantt Chart, Calendar, News, Documents, Wiki, Files, and Configuration. The main form has the following fields:

- 跟踪 \***: Bug (dropdown menu)
- 主题 \***: 用户登录之后无法跳转 (text input)
- 描述**: Rich text editor containing the text "RT! ! !".
- 私有**: A checkbox that is unchecked.
- 状态 \***: New (dropdown menu)
- 优先级 \***: High (dropdown menu)
- 指派给**: Hao LUO (dropdown menu)
- 父任务**: A search field with a magnifying glass icon.
- 开始日期**: 2018-08-01 (date picker)
- 计划完成日期**: 2018-08-10 (date picker)
- 预期时间**: 8 小时 (time input)
- % 完成**: 0 % (progress bar)
- 文件**: A file upload field with the placeholder "选择文件" and note "(最大尺寸: 100 KB)".
- 跟踪者**: Two options: "Hao LUO" (checkbox) and "通过查找方式添加跟踪者" (radio button).

At the bottom of the form are buttons for "创建" (Create), "创建并继续" (Create and Continue), and "预览" (Preview).

如果希望了解和使用Redmine，可以关注[Redmine中文网](#)，上面提供了视频教程、经验分享以及其他安装和使用上的指导。

## 持续集成

为了快速的产生高品质的软件，在团队开发中，持续集成（CI）也是一个非常重要的基础。按照经典的软件过程模型（瀑布模型），集成的工作一般要等到所有的开发工作都结束后才能开始，但这个时候如果发现了问题，修复问题的代价是非常具体的。基本上，集成实施得越晚，代码量越大，解决问题就越困难。持续集成将版本控制、自动化构建、代码测试融入到一起，让这些工作变得自动化和可协作。由于其频繁重复整个开发流程（在指定时间内多次pull源代码并运行测试代码），所以能帮助开发者提早发现问题。

在所有的CI工具中，Jenkins和TravisCI是最具有代表性的。



Jenkins 是基于 Java 的开源 CI 工具，其安装和操作都很简单。另外，Jenkins 不仅能在面板上轻松看出任务成功或失败，还可以借助通知功能将结果以邮件或 RSS 订阅的形式发给用户。与此同时，Jenkins 也允许通过插件进行功能扩展，所需功能可以随用随添加，而且还支持主从式集群，能够轻松的进行水平扩展。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day91-100 / 电商网站技术要点剖析.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



865 lines (685 sloc) 30 KB

# 电商网站技术要点剖析

## 商业模式

1. B2B - 商家对商家，交易双方都是企业（商家），最典型的案例就是阿里巴巴。
2. C2C - 个人对个人，例如：淘宝、人人车。
3. B2C - 商家对个人，例如：唯品会，聚美优品。
4. C2B - 个人对商家，先有消费者提出需求，后有商家按需求组织生产，例如：尚品宅配。
5. O2O - 线上到线下，将线下的商务机会与互联网结合，让互联网成为线下交易的平台，例如：美团外卖、饿了么。
6. B2B2C - 商家对商家对个人，例如：天猫、京东。

## 需求要点

### 1. 用户端

- 首页（商品分类、广告轮播、滚动快讯、瀑布加载、推荐、折扣、热销、……）
- 用户（登录（第三方登录）、注册、注销、自服务（个人信息、浏览历史、收货地址、……））
- 商品（分类、列表、详情、搜索、热门搜索、搜索历史、添加到购物车、收藏、关注、……）
- 购物车（查看、编辑（修改数量、删除商品、清空））
- 订单（提交订单（支付）、历史订单、订单详情、订单评价、……）

### 2. 管理端

- 核心业务实体的CRUD

- 定时任务（周期性和非周期性）
- 报表功能（Excel、PDF、ECharts）
- 权限控制（RBAC）
- 业务流转（Activity、Airflow、Spiff、自定义）
- 三方服务（地图、短信、物流、支付、实名认证、天气、监控、……）

提示：可以通过思维导图来进行需求的整理，思维导图上的每个叶子节点都是不可再拆分的功能，而且都是动词。

## 物理模型设计

两个概念：SPU（Standard Product Unit）和SKU（Stock Keeping Unit）。

- SPU：iPhone 6s
- SKU：iPhone 6s 64G 土豪金



## 第三方登录

第三方登录是指利用第三方网站（通常是知名社交网站）的账号进行登录验证，比如国内的QQ、微博，国外的Google、Facebook等，第三方登录大部分都是使用OAuth，它是一个关于授权的开放网络标准，得到了广泛的应用，目前通常使用的是2.0版本。关于OAuth的基础知识，可以阅读阮一峰老师的[《理解OAuth 2.0》](#)。

### OAuth 2.0授权流程

1. 用户打开客户端以后，客户端要求用户（资源所有者）给予授权。
2. 用户（资源所有者）同意给予客户端授权。
3. 客户端使用上一步获得的授权，向认证服务器申请访问令牌。

4. 认证服务器对客户端进行认证以后，发放访问令牌。
5. 客户端使用访问令牌向资源服务器申请获取资源。
6. 资源服务器确认访问令牌无误，同意向客户端开放资源。



如果使用微博登录进行接入，其具体步骤可以参考微博开放平台上的[“微博登录接入”](#)文档。使用QQ登录进行接入，需要首先注册成为QQ互联开发者并通过审核，具体的步骤可以参考QQ互联上的[“接入指南”](#)，具体的步骤可以参考[“网站开发流程”](#)。

**提示：**在Gitbook上面有一本名为[《Django博客入门》](#)的书以Github为例介绍了第三方账号登录，有兴趣的可以自行阅读。

通常电商网站在使用第三方登录时，会要求与网站账号进行绑定或者根据获取到的第三方账号信息（如：手机号）自动完成账号绑定。

## 缓存预热和查询缓存

### 缓存预热

所谓缓存预热，是指在启动服务器时将数据提前加载到缓存中，为此可以在Django应用的 `apps.py` 模块中编写  `AppConfig` 的子类并重写 `ready()` 方法，代码如下所示。

```

import pymysql

from django.apps import AppConfig
from django.core.cache import cache

SELECT_PROVINCE_SQL = 'select distid, name from tb_district where pid is null'

class CommonConfig(AppConfig):
    name = 'common'

    def ready(self):
  
```

```

conn = pymysql.connect(host='1.2.3.4', port=3306,
                      user='root', password='pass',
                      database='db', charset='utf8',
                      cursorclass=pymysql.cursors.DictCursor)

try:
    with conn.cursor() as cursor:
        cursor.execute(SELECT_PROVINCE_SQL)
        provinces = cursor.fetchall()
        cache.set('provinces', provinces)
finally:
    conn.close()

```

接下来，还需要在应用的 `__init__.py` 中编写下面的代码。

```
default_app_config = 'common.apps.CommonConfig'
```

或者在项目的 `settings.py` 文件中注册应用。

```

INSTALLED_APPS = [
    ...
    'common.apps.CommonConfig',
    ...
]

```

## 查询缓存

自定义装饰器实现查询结果的缓存。

```

from pickle import dumps
from pickle import loads

from django.core.cache import caches

MODEL_CACHE_KEY = 'project:modelcache:%s'

def my_model_cache(key, section='default', timeout=None):
    """实现模型缓存的装饰器"""

    def wrapper1(func):

        def wrapper2(*args, **kwargs):
            real_key = '%s:%s' % (MODEL_CACHE_KEY % key, ':'.join(map(str, args)))
            serialized_data = caches[section].get(real_key)
            if serialized_data:
                data = loads(serialized_data)
            else:
                data = func(*args, **kwargs)
            cache.set(real_key, dumps(data), timeout=timeout)
            return data
        return wrapper2
    return wrapper1

```

```

        return data

    return wrapper2

    return wrapper1

@my_model_cache(key='provinces')
def get_all_provinces():
    return list(Province.objects.all())

```

## 购物车实现

问题一：已登录用户的购物车放在哪里？未登录用户的购物车放在哪里？

```

class CartItem(object):
    """购物车中的商品项"""

    def __init__(self, sku, amount=1, selected=False):
        self.sku = sku
        self.amount = amount
        self.selected = selected

    @property
    def total(self):
        return self.sku.price * self.amount


class ShoppingCart(object):
    """购物车"""

    def __init__(self):
        self.items = {}
        self.index = 0

    def add_item(self, item):
        if item.sku.id in self.items:
            self.items[item.sku.id].amount += item.amount
        else:
            self.items[item.sku.id] = item

    def remove_item(self, sku_id):
        if sku_id in self.items:
            self.items.remove(sku_id)

    def clear_all_items(self):
        self.items.clear()

    @property
    def cart_items(self):
        return self.items.values()

```

```
@property
def cart_total(self):
    total = 0
    for item in self.items.values():
        total += item.total
    return total
```

已登录用户的购物车可以放在数据库中（可以先在Redis中缓存）；未登录用户的购物车可以保存在Cookie、localStorage或sessionStorage中（减少服务器端内存开销）。

```
{
    '1001': {sku: {...}, 'amount': 1, 'selected': True},
    '1002': {sku: {...}, 'amount': 2, 'selected': False},
    '1003': {sku: {...}, 'amount': 3, 'selected': True},
}
```

```
request.get_signed_cookie('cart')

cart_base64 = base64.b64encode(pickle.dumps(cart))
response.set_signed_cookie('cart', cart_base64)
```

问题二：用户登录之后，如何合并购物车？（目前电商应用的购物车几乎都做了持久化处理，主要是方便在多个终端之间共享数据）

## 集成支付功能

问题一：支付信息如何持久化？（必须保证每笔交易都有记录）

问题二：如何接入支付宝？（接入其他平台基本类似）

1. [蚂蚁金服开放平台](#)。
2. [入驻平台](#)。
3. [开发者中心](#)。
4. [文档中心](#)。
5. [SDK集成 - PYPI连接](#)。
6. [API列表](#)。

网页&移动应用

 网页&移动应用接入文档

开发者中心 / 网页&移动应用 / 应用列表

#### 创建应用

+ 支付接入 快速接入支付能力 + 自定义接入 选择接入功能API + 商业消费 品牌零售/口碑 + 交通出行 停车/单车/物流 + 政务民生 城市服务/物业 + 医疗教育 医院/学校

#### 我的应用列表

应用名称	APPID	应用状态	操作
暂无数据			

## 配置文件：

```
ALIPAY_APPID = '.....'
ALIPAY_URL = 'https://openapi.alipaydev.com/gateway.do'
ALIPAY_DEBUG = False
```

## 获得支付链接（发起支付）：

```
# 创建调用支付宝的对象
alipay = AliPay(
    # 在线创建应用时分配的ID
    appid=settings.ALIPAY_APPID,
    app_notify_url=None,
    # 自己应用的私钥
    app_private_key_path=os.path.join(
        os.path.dirname(os.path.abspath(__file__)),
        'keys/app_private_key.pem'),
    # 支付宝的公钥
    alipay_public_key_path=os.path.join(
        os.path.dirname(os.path.abspath(__file__)),
        'keys/alipay_public_key.pem'),
    sign_type='RSA2',
    debug=settings.ALIPAY_DEBUG
)
# 调用获取支付页面操作
order_info = alipay.api_alipay_trade_page_pay(
    out_trade_no='...',
    total_amount='...',
    subject='...',
    return_url='http://...')
# 生成完整的支付页面URL
alipay_url = settings.ALIPAY_URL + '?' + order_info
return JsonResponse({'alipay_url': alipay_url})
```

通过上面返回的链接可以进入支付页面，支付完成后会自动跳转回上面代码中设定好的项目页面，在该页面中可以获得订单号（out\_trade\_no）、支付流水号（trade\_no）、交易金额（total\_amount）和对应的签名（sign）并请求后端验证和保存交易结果，代码如下所示：

```
# 创建调用支付宝的对象
alipay = AliPay(
    # 在线创建应用时分配的ID
    appid=settings.ALIPAY_APPID,
    app_notify_url=None,
    # 自己应用的私钥
    app_private_key_path=os.path.join(
        os.path.dirname(os.path.abspath(__file__)),
        'keys/app_private_key.pem'),
    # 支付宝的公钥
    alipay_public_key_path=os.path.join(
        os.path.dirname(os.path.abspath(__file__)),
        'keys/alipay_public_key.pem'),
    sign_type='RSA2',
    debug=settings.ALIPAY_DEBUG
)
# 请求参数（假设是POST请求）中包括订单号、支付流水号、交易金额和签名
params = request.POST.dict()
# 调用验证操作
if alipay.verify(params, params.pop('sign')):
    # 对交易进行持久化操作
```

支付宝的支付API还提供了交易查询、交易结算、退款、退款查询等一系列的接口，可以根据业务需要进行调用，此处不再进行赘述。

## 秒杀和超卖

1. 秒杀：秒杀是通常意味着要在很短的时间处理极高的并发，系统在短时间需要承受平时百倍以上的流量，因此秒杀架构是一个比较复杂的问题，其核心思路是流量控制和性能优化，需要从前端（通过JavaScript实现倒计时、避免重复提交和限制频繁刷新）到后台各个环节的配合。流量控制主要是限制只有少部分流量进入服务后端（毕竟最终只有少部分用户能够秒杀成功），同时在物理架构上使用缓存（一方面是因为读操作多写操作少；另外可以将库存放在Redis中，利用DECR原语实现减库存；同时也可利用Redis来进行限流，道理跟限制频繁发送手机验证码是一样的）和消息队列（消息队列最为重要的作用就是“削峰”和“上下游节点解耦合”）来进行优化；此外还要采用无状态服务设计，这样才便于进行水平扩展（通过增加设备来为系统扩容）。
2. 超卖现象：比如某商品的库存为1，此时用户1和用户2并发购买该商品，用户1提交订单后该商品的库存被修改为0，而此时用户2并不知道的情况下提交订单，该商品的库存再次被修改为-1这就是超卖现象。解决超卖现象有三种常见的思路：
  - 悲观锁控制：查询商品数量的时候就用 select ... for update 对数据加锁，这样的话用户1查询库存时，用户2因无法读取库存数量被阻塞，直到用户1提交或

者回滚了更新库存的操作后才能继续，从而解决了超卖问题。但是这种做法对并发访问量很高的商品来说性能太过糟糕，实际开发中可以在库存小于某个值时才考虑加锁，但是总的来说这种做法不太可取。

- 乐观锁控制：查询商品数量不用加锁，更新库存的时候设定商品数量必须与之前查询数量相同才能更新，否则说明其他事务已经更新了库存，必须重新发出请求。这种做法要求事务隔离级别为可重复读，否则仍然会产生问题。
- 尝试减库存：将上面的查询（`select`）和更新（`update`）操作合并为一条SQL操作，更新库存的时候，在`where`筛选条件中加上`库存>=购买数量`或`库存-购买数量>=0`的条件。

提示：有兴趣的可以自己在知乎上看看关于这类问题的讨论。

## 静态资源管理

静态资源的管理可以自己架设文件服务器或者分布式文件服务器（FastDFS），但是一般项目中没有必要这样做而且效果未必是最好的，我们建议使用云存储服务来管理网站的静态资源，国内外的云服务提供商如亚马逊、阿里云、腾讯云、七牛、LeanCloud、Bmob等都提供了非常优质的云存储服务，而且价格也是一般公司可以接受的。可以参考《在阿里云OSS上托管静态网站》一文来完成对网站静态资源的管理，代码相关的内容可以参考阿里云的[对象存储 OSS 开发人员指南](#)。

## 全文检索

### 方案选择

1. 使用数据库的模糊查询功能 - 效率低，每次需要全表扫描，不支持分词。
2. 使用数据库的全文检索功能 - MySQL 5.6以前只适用于MyISAM引擎，检索操作和其他的DML操作耦合在数据库中，可能导致检索操作非常缓慢，数据量达到百万级性能显著下降，查询时间很长。
3. 使用开源搜索引擎 - 索引数据和原始数据分离，可以使用ElasticSearch或Solr来提供外置索引服务，如果不考虑高并发的全文检索需求，纯Python的Whoosh也可以考虑。

### ElasticSearch

ElasticSearch既是一个分布式文档数据库又是一个高可扩展的开源全文搜索和分析引擎，它允许存储、搜索和分析大量的数据，并且这个过程是近实时的。它通常被用作底层引擎和技术，为复杂的搜索功能和要求提供动力，大家熟知的维基百科、Stack-Overflow、Github都使用了ElasticSearch。

ElasticSearch的底层是开源搜索引擎Lucene，但是直接用Lucene会非常麻烦，必须自己编写代码去调用它的接口而且只支持Java语言。ElasticSearch相当于对Lucene进行了一次全面的封装，提供了REST风格的API接口，通过基于HTTP协议的访问方式屏蔽了编程语言的差异。ElasticSearch会为数据构建倒排索引，但是ElasticSearch内置的分词器对中文分词的支持几乎为零，因此需要通过安装elasticsearch-analysis-ik插件来提供中文分词服务。

ElasticSearch的安装和配置可以参考[《ElasticSearch之Docker安装》](#)。除了ElasticSearch之外，也可以使用Solr、Whoosh等来提供搜索引擎服务，基本上Django项目中可以考虑如下两套方案：

- haystack ( django-haystack / drf-haystack ) + whoosh + Jieba
- haystack ( django-haystack / drf-haystack ) + elasticsearch

#### ####安装和使用ElasticSearch

##### 1. 使用Docker安装ElasticSearch。

```
docker pull elasticsearch:6.5.3
docker run -d -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" -e ES_JAVA_OPTS=-Xms512m -Xmx512m
```

说明：上面创建容器时通过-e参数指定了使用单机模式和Java虚拟机最小最大可用堆空间的大小，堆空间大小可以根据服务器实际能够提供给ElasticSearch的内存大小来决定，默认为2G。

##### 2. 创建数据库。

请求：PUT - <http://1.2.3.4:9200/demo>

响应：

```
{
  "acknowledged": true,
  "shards_acknowledged": true,
  "index": "demo"
}
```

##### 3. 查看创建的数据库。

请求：GET - <http://1.2.3.4:9200/demo>

响应：

```
{
  "demo": {
```

```
        "aliases": {},
        "mappings": {},
        "settings": {
            "index": {
                "creation_date": "1552213970199",
                "number_of_shards": "5",
                "number_of_replicas": "1",
                "uuid": "ny3rCn10SAmCsqW6xPP1gw",
                "version": {
                    "created": "6050399"
                },
                "provided_name": "demo"
            }
        }
    }
}
```

#### 4. 插入数据。

请求 : POST - <http://1.2.3.4:9200/demo/goods/1/>

请求头 : Content-Type: application/json

参数 :

```
{
    "no": "5089253",
    "title": "Apple iPhone X (A1865) 64GB 深空灰色 移动联通电信4G手机",
    "brand": "Apple",
    "name": "Apple iPhone X",
    "product": "中国大陆",
    "resolution": "2436 x 1125",
    "intro": "一直以来，Apple都心存一个设想，期待能够打造出这样一部iPhone：它有整[...]
}
```

响应 :

```
{
    "_index": "demo",
    "_type": "goods",
    "_id": "1",
    "_version": 4,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 3,
```

```
        "_primary_term": 1
    }
```

## 5. 删除数据。

请求 : DELETE - <http://1.2.3.4:9200/demo/goods/1/>

响应 :

```
{
    "_index": "demo",
    "_type": "goods",
    "_id": "1",
    "_version": 2,
    "result": "deleted",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 1,
    "_primary_term": 1
}
```

## 6. 更新数据。

请求 : PUT - [http://1.2.3.4:9200/demo/goods/1/\\_update](http://1.2.3.4:9200/demo/goods/1/_update)

请求头 : Content-Type: application/json

参数 :

```
{
    "doc": {
        "no": "5089253",
        "title": "Apple iPhone X (A1865) 64GB 深空灰色 移动联通电信4G手机",
        "brand": "Apple(苹果)",
        "name": "Apple iPhone X",
        "product": "美国",
        "resolution": "2436 x 1125",
        "intro": "一直以来，Apple都心存一个设想，期待能够打造出这样一部iPhone：它不"
    }
}
```

响应 :

```
{
    "_index": "demo",
```

```
_type": "goods",
"_id": "1",
"_version": 10,
"result": "updated",
"_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
},
"_seq_no": 9,
"_primary_term": 1
}
```

## 7. 查询数据。

请求 : GET - <http://1.2.3.4:9200/demo/goods/1/>

响应 :

```
{
  "_index": "demo",
  "_type": "goods",
  "_id": "1",
  "_version": 10,
  "found": true,
  "_source": {
    "doc": {
      "no": "5089253",
      "title": "Apple iPhone X (A1865) 64GB 深空灰色 移动联通电信4G手机",
      "brand": "Apple(苹果)",
      "name": "Apple iPhone X",
      "product": "美国",
      "resolution": "2436 x 1125",
      "intro": "一直以来，Apple都心存一个设想，期待能够打造出这样一部iPhone."
    }
  }
}
```

## 配置中文分词和拼音插件

### 1. 进入Docker容器的plugins目录。

```
docker exec -it es /bin/bash
```

### 2. 下载和ElasticSearch版本对应的ik和pinyin插件。

```
cd plugins/
mkdir ik
```

```
cd ik
wget https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v6.5.3/elasticsearch-analysis-ik-6.5.3.zip
unzip elasticsearch-analysis-ik-6.5.3.zip
rm -f elasticsearch-analysis-ik-6.5.3.zip
cd ..
mkdir pinyin
cd pinyin
wget https://github.com/medcl/elasticsearch-analysis-pinyin/releases/download/v6.5.3/elasticsearch-analysis-pinyin-6.5.3.zip
unzip elasticsearch-analysis-pinyin-6.5.3.zip
rm -f elasticsearch-analysis-pinyin-6.5.3.zip
```

### 3. 退出容器，重启ElasticSearch。

```
docker restart es
```

### 4. 测试中文分词效果。

请求：POST - http://1.2.3.4:9200/\_analyze

请求头：Content-Type: application/json

参数：

```
{
  "analyzer": "ik_smart",
  "text": "中国男足在2022年卡塔尔世界杯预选赛中勇夺小组最后一名"
}
```

响应：

```
{
  "tokens": [
    {
      "token": "中国",
      "start_offset": 0,
      "end_offset": 2,
      "type": "CN_WORD",
      "position": 0
    },
    {
      "token": "男足",
      "start_offset": 2,
      "end_offset": 4,
      "type": "CN_WORD",
      "position": 1
    },
    {
      "token": "在",
      "start_offset": 4,
      "end_offset": 6,
      "type": "CN_WORD",
      "position": 2
    }
  ]
}
```

```
        "start_offset": 4,
        "end_offset": 5,
        "type": "CN_CHAR",
        "position": 2
    },
    {
        "token": "2022年",
        "start_offset": 5,
        "end_offset": 10,
        "type": "TYPE_CQUAN",
        "position": 3
    },
    {
        "token": "卡塔尔",
        "start_offset": 10,
        "end_offset": 13,
        "type": "CN_WORD",
        "position": 4
    },
    {
        "token": "世界杯",
        "start_offset": 13,
        "end_offset": 16,
        "type": "CN_WORD",
        "position": 5
    },
    {
        "token": "预选赛",
        "start_offset": 16,
        "end_offset": 19,
        "type": "CN_WORD",
        "position": 6
    },
    {
        "token": "中",
        "start_offset": 19,
        "end_offset": 20,
        "type": "CN_CHAR",
        "position": 7
    },
    {
        "token": "勇夺",
        "start_offset": 20,
        "end_offset": 22,
        "type": "CN_WORD",
        "position": 8
    },
    {
        "token": "小组",
        "start_offset": 22,
        "end_offset": 24,
        "type": "CN_WORD",
        "position": 9
    },
    {
```

```
        "token": "最后",
        "start_offset": 24,
        "end_offset": 26,
        "type": "CN_WORD",
        "position": 10
    },
    {
        "token": "一名",
        "start_offset": 26,
        "end_offset": 28,
        "type": "CN_WORD",
        "position": 11
    }
]
}
```

## 5. 测试拼音分词效果。

请求 : POST - [http://1.2.3.4:9200/\\_analyze](http://1.2.3.4:9200/_analyze)

请求头 : Content-Type: application/json

参数 :

响应 :

```
{
  "tokens": [
    {
      "token": "zhang",
      "start_offset": 0,
      "end_offset": 0,
      "type": "word",
      "position": 0
    },
    {
      "token": "zxy",
      "start_offset": 0,
      "end_offset": 0,
      "type": "word",
      "position": 0
    },
    {
      "token": "xue",
      "start_offset": 0,
      "end_offset": 0,
      "type": "word",
      "position": 1
    }
  ]
}
```

```

        "token": "you",
        "start_offset": 0,
        "end_offset": 0,
        "type": "word",
        "position": 2
    }
]
}

```

## 全文检索功能

可以通过GET或者POST请求进行搜索，下面演示了搜索有“未来”关键词商品。

1. GET - [http://120.77.222.217:9200/demo/goods/\\_search?q=未来](http://120.77.222.217:9200/demo/goods/_search?q=未来)

注意：URL中的中文应该要处理成百分号编码。

```

{
    "took": 19,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": 2,
        "max_score": 0.73975396,
        "hits": [
            {
                "_index": "demo",
                "_type": "goods",
                "_id": "1",
                "_score": 0.73975396,
                "_source": {
                    "doc": {
                        "no": "5089253",
                        "title": "Apple iPhone X (A1865) 64GB 深空灰色 移动联通电信全网通",
                        "brand": "Apple(苹果)",
                        "name": "Apple iPhone X",
                        "product": "美国",
                        "resolution": "2436*1125",
                        "intro": "一直以来，Apple都心存一个设想，期待能够打造出这样一台手机。"
                    }
                }
            },
            {
                "_index": "demo",
                "_type": "goods",
                "_id": "3",
                "_score": 0.68324494,
                "_source": {
                    "doc": {
                        "no": "5089253",
                        "title": "Apple iPhone X (A1865) 64GB 深空灰色 移动联通电信全网通",
                        "brand": "Apple(苹果)",
                        "name": "Apple iPhone X",
                        "product": "美国",
                        "resolution": "2436*1125",
                        "intro": "一直以来，Apple都心存一个设想，期待能够打造出这样一台手机。"
                    }
                }
            }
        ]
    }
}

```

```

    "_source": {
        "no": "42417956432",
        "title": "小米9 透明尊享版 手机 透明尊享 全网通(12GB + 256GB)"
        "brand": "小米 (MI) ",
        "name": "小米 (MI) 小米9透明",
        "product": "中国大陆",
        "resolution": "2340*1080",
        "intro": "全面透明机身，独特科幻机甲风，来自未来的设计。"
    }
}
]
}
}

```

URL中可用的搜索参数如下表所示：

参数	说明
q	查询字符串
analyzer	分析查询字符串使用的分词器
analyze_wildcard	通配符或者前缀查询是否被分析，默认为false
default_operator	多个条件之间的关系，默认为OR，可以修改为AND
explain	在返回的结果中包含评分机制的解释
fields	只返回索引中指定的列，多个列中间用逗号隔开
sort	排序参考的字段，可以用:asc和:desc来指定升序和降序
timeout	超时时间
from	匹配结果的开始值，默认为0
size	匹配结果的条数，默认为10

2. POST - [http://120.77.222.217:9200/demo/goods/\\_search](http://120.77.222.217:9200/demo/goods/_search)

请求头：Content-Type: application/json

参数：

响应：

## Django对接ElasticSearch

Python对接ElasticSearch的第三方库是HayStack，在Django项目中可以使用django-haystack，通过HayStack可以在不修改代码对接多种搜索引擎服务。

```
pip install django-haystack elasticsearch
```

配置文件：

```
INSTALLED_APPS = [
    ...
    'haystack',
    ...
]

HAYSTACK_CONNECTIONS = {
    'default': {
        # 引擎配置
        'ENGINE': 'haystack.backends.elasticsearch_backend.ElasticsearchSearchEngine',
        # 搜索引擎服务的URL
        'URL': 'http://12.3.4: 9200',
        # 索引库的名称
        'INDEX_NAME': 'goods',
    },
}

# 添加/删除/更新数据时自动生成索引
HAYSTACK_SIGNAL_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'
```

索引类：

```
from haystack import indexes

class GoodsIndex(indexes.SearchIndex, indexes.Indexable):
    text = indexes.CharField(document=True, use_template=True)

    def get_model(self):
        return Goods

    def index_queryset(self, using=None):
        return self.get_model().objects.all()
```

编辑text字段的模板（需要放在templates/search/indexes/demo/goods\_text.txt）：

```
{{object.title}}
{{object.intro}}
```

配置URL：

```
urlpatterns = [
    # ...
    url('search/', include('haystack.urls')),
]
```

生成初始索引：

```
python manage.py rebuild_index
```

说明：可以参考[《Django Haystack 全文检索与关键词高亮》](#)一文来更深入的了解基于Haystack的全文检索操作。

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Day91-100 / 网络API接口设计.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

132 lines (96 sloc) 5.96 KB

# 网络API接口设计

手机App以及使用了Ajax技术或做了前后端分离的页面都需要通过网络API ( Application Programming Interface ) 和后台进行交互，所谓API，指的应用程序的编程接口；而网络API通畅指的是基于HTTP或HTTPS协议的一个URL ( 统一资源定位符 )，通过这个URL我们可以让服务器对某个资源进行操作并返回操作的结果。基于HTTP(S)协议最大的好处就在于访问起来非常的简单方便，而且没有编程语言和应用环境上的差别。

## 设计原则

### 关键问题

为移动端或者PC端设计网络API接口一个非常重要的原则是：根据业务实体而不是用户界面或操作来设计。如果API接口的设计是根据用户的操作或者界面上的功能设置来设计，随着需求的变更，用户界面也会进行调整，需要的数据也在发生变化，那么后端开发者就要不停的调整API，或者给一个API设计出多个版本，这些都会使项目的开发和维护成本增加。

下面是某个网站开放API的接口，可以看出API的设计是围绕业务实体来进行的，而且都做到了“见名知意”。

评论	
comments/show	获取某条微博的评论列表
comments/by_me	自己的评论列表
comments/to_me	收到的评论列表
comments/mentions	@了自己的评论列表
comments/create	创建一条评论

评论	
comments/destroy	删除一条评论
comments/reply	回复一条评论

注意：上面的API接口并不是REST风格的，关于REST的知识，可以阅读阮一峰老师的[《理解RESTful架构》](#)以及[《RESTful API设计指南》](#)。

API接口返回的数据通常都是JSON或XML格式，我们这里不讨论后者。对于JSON格式的数据，我们需要做到不要返回null这的值，因为这样的值一旦处置不当，会给移动端的开发带来麻烦（移动端可能使用强类型语言）。要解决这个问题可以从源头入手，在设计数据库的时候，尽量给每个字段都加上“not null”约束或者设置合理的默认值约束。

## 其他问题

- 更新提示问题：设计一个每次使用系统首先要访问的API，该API会向移动端返回系统更新的相关信息，这样就可以提升用户更新App了。
- 版本升级问题：API版本升级时应该考虑对低版本的兼容，同时要让新版本和旧版本都能够被访问，可以在URL中包含版本信息或者在将版本号放在HTTP(S)协议头部，关于这个问题有很多的争论，有兴趣的可以看看[stack overflow](#)上面对这个问题的讨论。
- 图片尺寸问题：移动端对于一张图片可能需要不同的尺寸，可以在获取图片时传入尺寸参数并获取对应的资源；更好的做法是直接使用云存储或CDN（直接提供了图片缩放的功能），这样可以加速对资源的访问。

## 文档撰写

下面以设计评论接口为例，简单说明接口文档应该如何撰写。

### 评论接口

全局返回状态码

返回码	返回信息	说明
10000	获取评论成功	
10001	创建评论成功	
10002	无法创建评论	创建评论时因违反审核机制而无法创建
10003	评论已被删除	查看评论时评论因不和谐因素已被删除
10004	.....	.....

1. GET /comments/{article-id}

开发者：王大锤

最后更新时间：2018年8月10日

标签：v 1.0

接口说明：获取指定文章的所有评论

使用帮助：默认返回20条数据，需要在请求头中设置身份标识（key）

请求参数：

参数名	类型	是否必填	参数位置	说明
page	整数	否	查询参数	页码，默认值1
size	整数	否	查询参数	每次获取评论数量（10~100），默认值20
key	字符串	是	请求头	用户的身份标识

响应信息：

```
{  
    "code": 10000,  
    "message": "获取评论成功",  
    "page": 1,  
    "size": 10,  
    "totalPage": 35,  
    "contents": [  
        {  
            "userId": 1700095,  
            "nickname": "王大锤",  
            "pubDate": "2018年7月31日",  
            "content": "小编是不是有病呀",  
            /* ... */  
        },  
        {  
            "userId": 1995322,  
            "nickname": "白元芳",  
            "pubDate": "2018年8月2日",  
            "content": "楼上说得好",  
            /* ... */  
        }  
    ]  
    /* ... */  
}
```

## 2. POST /comments/{article-id}

开发者：王大锤

最后更新时间：2018年8月10日

标签：v 1.0

接口说明：为指定的文章创建评论

使用帮助：暂无

请求参数：

参数名	类型	是否必填	参数位置	说明
userId	字符串	是	消息体	用户ID
key	字符串	是	请求头	用户的令牌
content	字符串	是	消息体	评论的内容

响应信息：

```
{  
    "code": 10001,  
    "message": "创建评论成功",  
    "comment": {  
        "pubDate": "2018年7月31日",  
        "content": "小编是不是有病呀"  
        /* ... */  
    }  
    /* ... */  
}
```

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Day91-100 / 英语面试.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



90 lines (56 sloc) 4.73 KB

# 英语面试

以下用I表示面试官 ( Interviewer ) , 用C表示面试者 ( Candidate ) 。

## 开场寒暄

1. I: Thanks for waiting. (Please follow me.)

C: It's no problem.

2. I: How are you doing this morning?

C: I'm great. / I'm doing fine. Thank you. / How about you?

3. I: How did you get here?

C: I took the subway here. / I drove here.

4. I: Glad to meet you.

C: Glad to meet you. / It's great to finally meet you in person. (之前电话沟通过的)

## 正式面试

### 人力面试

1. I: Can you tell me a little bit about yourself? (介绍下自己)

原则：不要谈私生活和奇怪的癖好（英雄联盟干到钻石），因为别人更想知道的是你的专业技能 ( qualifications ) 和工作经验 ( experience )，所以重点在你之前的公司 ( company name )、职位 ( title )、时间 ( years ) 和主要职责 ( major responsibilities )

C: Thank you for having me. My name is Dachui WANG. I'm 25 years old, and I'm single. I have a Bachelor's Degree of Computer Science from Tsinghua University. I was a Junior Java Programmer for ABC Technologies during my college life. Then I become an intermediate Java engineer for XYZ Corporation in last two years. Programming is my everyday life and programming is where my passion is. I think I have a good knowledge of Java enterprise application development using light-weight frameworks like Spring, Guice, Hibernate and other open source middle-ware like Dubbo, Mycat, rocketmq and so on and so forth. I love reading, travelling and playing basketball in my spare time. That's all! Thank you!

2. I: How would you describe your personality? (你的性格)

C: I'm hard working, eager to learn, and very serious about my work. I enjoy working with other people and I love challenges.

3. I: What do you know about our company? (你对我们公司有什么了解)

(需要做功课，了解公司的状况和企业文化，该公司在这个行业中的一个状况，有什么核心业务，主要的竞争对手有哪些)

C: The one thing that I like the most about our company is your core values. I think they're very important in this industry because ... (自由发挥的部分)... I personally really believe in the cause as well. Of course, I'm very interested in your products such as ... (功课部分)... and the techniques behind them.

4. I: Why are you leaving your last job? (为什么离职)

C: I want to advance my career and I think this job offers more challenges and opportunities for me do to that.

5. I: What do you see yourself in 3 or 5 years? (3-5年职业规划)

C: My long term goals involve growing with the company, where I can continue to learn, to take on additional responsibilities and to contribute as much value as I can. I intend to take advantage of all of these.

6. I: What's your salary expectation? (期望薪资)

C: My salary expectation is in line with my experience and qualifications. I believe our company will pay me and every other employee fairly. (把球踢给对方先看看对方报价是多少，如果对方非要你报价再说后面的内容) I think 15 thousands RMB or above is fitting for me to leave in Chengdu.

7. I: Do you have any questions for me? (问面试官的问题)

C: What's the growth potential for this position?

1. I: What's difference between an interface and an abstract class?
2. I: What are pass by reference and pass by value?
3. I: What's the difference between process and threads?
4. I: Explain the available thread state in high-level.
5. I: What's deadlocks? How to avoid them?
6. I: How HashMap works in Java?
7. I: What's the difference between ArrayList and LinkedList? (类似的问题还有很多，比如比较HashSet和TreeSet、HashMap和Hashtable)
8. I: Tell me what you know about garbage collection in Java.
9. I: What're two types of exceptions in Java?
10. I: What's the advantage of PreparedStatement over Statement?
11. I: What's the use of CallableStatement?
12. I: What does connection pool mean?
13. I: Explain the life cycle of a Servlet.
14. I: What's the difference between redirect and forward?
15. I: What's EL? What're implicit objects of EL?
16. I: Tell me what you know about Spring framework and its benefits.
17. I: What're different types of dependency injection.
18. I: Are singleton beans thread safe in Spring framework?
19. I: What're the benefits of Spring framework's transaction management?
20. I: Explain what's AOP.
21. I: What's a proxy and how to implement proxy pattern?
22. I: How Spring MVC works?
23. I: What's the working scenario of Hibernate and MyBatis?
24. I: How to implement SOA?
25. I: Make a brief introduction of the projects you are involved before?

上面主要是面试Java程序员的问题，但是整个流程大致如此。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day91-100 / 面试中的公共问题.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



90 lines (55 sloc) 3.96 KB

# 面试中的公共问题

## 计算机基础

1. TCP/IP模型相关问题。

建议阅读阮一峰的《互联网协议入门（一）》和《互联网协议入门（二）》。

2. HTTP和HTTPS相关问题。

建议阅读阮一峰的《HTTP 协议入门》和《SSL/TLS协议运行机制的概述》。

3. Linux常用命令和服务。

4. 进程和线程之间的关系。什么时候用多线程？什么时候用多进程？。

5. 关系型数据库相关问题（ACID、事务隔离级别、锁、SQL优化）。

6. 非关系型数据库相关问题（CAP/BASE、应用场景）。

## Python基础

1. 开发中用过哪些标准库和三方库。

标准库：sys / os / re / math / random / logging / json / pickle / shelve / socket / datetime / hashlib / configparser / urllib / itertools / collections / functools / threading / multiprocessing / timeit / atexit / abc / asyncio / base64 / concurrent.futures / copy / csv / operator / enum / heapq / http / profile / pstats / ssl / unittest / uuid

2. 装饰器的作用、原理和实现。

3. 使用过哪些魔法方法。

建议阅读《Python魔术方法指南》。

4. 生成式、生成器、迭代器的编写。
5. 列表、集合、字典的底层实现。
6. 垃圾回收相关问题。
7. 并发编程的相关问题。
8. 协程和异步I/O相关知识。

## Django和Flask

1. MVC架构（MTV）解决了什么问题。
2. 中间件的执行流程以及如何自定义中间件。
3. REST数据接口如何设计（URL、域名、版本、过滤、状态码、安全性）。

建议阅读阮一峰的《RESTful API设计指南》。

4. 使用ORM框架实现CRUD操作的相关问题。
  - 如何实现多条件组合查询 / 如何执行原生的SQL / 如何避免N+1查询问题
5. 如何执行异步任务和定时任务。
6. 如何实现页面缓存和查询缓存？缓存如何预热？

## 爬虫相关

1. Scrapy框架的组件和数据处理流程。
2. 爬取的目的（项目中哪些地方需要用到爬虫的数据）。
3. 使用的工具（抓包、下载、清理、存储、分析、可视化）。
4. 数据的来源（能够轻松的列举出10个网站）。
5. 数据的构成（抓取的某个字段在项目中有什么用）。
6. 反反爬措施（限速、请求头、Cookie池、代理池、Selenium、PhantomJS、RoboBrowser、TOR、OCR）。
7. 数据的体量（最后抓取了多少数据，多少W条数据或多少个G的数据）。
8. 后期数据处理（持久化、数据补全、归一化、格式化、转存、分类）。

## 数据分析

1. 科学运算函数库（SciPy和NumPy常用运算）。
2. 数据分析库（Pandas中封装的常用算法）。

3. 常用的模型及对应的场景（分类、回归、聚类）。
4. 提取了哪些具体的指标。
5. 如何评价模型的优劣。
6. 每种模型实际操作的步骤，对结果如何评价。

## 项目相关

1. 项目团队构成以及自己在团队中扮演的角色（在项目中的职责）。
2. 项目的业务架构（哪些模块及子模块）和技术架构（移动端、PC端、后端技术栈）。
3. 软件控制管理相关工具（版本控制、问题管理、持续集成）。
4. 核心业务实体及其属性，实体与实体之间的关系。
5. 用到哪些依赖库，依赖库主要解决哪方面的问题。
6. 项目如何部署上线以及项目的物理架构（Nginx、Gunicorn/uWSGI、Redis、MongoDB、MySQL、Supervisor等）。
7. 如何对项目进行测试，有没有做过性能调优。
8. 项目中遇到的困难有哪些，如何解决的。

Branch: master ▾

Find file Copy path

## Python-100-Days / Day91-100 / 项目部署上线指南.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



862 lines (694 sloc) 26.5 KB

# 项目部署上线指南

## 准备上线

1. 上线前的检查工作。

```
python manage.py check --deploy
```

2. 将DEBUG设置为False并配置ALLOWED\_HOSTS。

```
DEBUG = False
ALLOWED_HOSTS = ['*']
```

3. 安全相关的配置。

```
# 保持HTTPS连接的时间
SECURE_HSTS_SECONDS = 3600
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True

# 自动重定向到安全连接
SECURE_SSL_REDIRECT = True

# 避免浏览器自作聪明推断内容类型
SECURE_CONTENT_TYPE_NOSNIFF = True

# 避免跨站脚本攻击
SECURE_BROWSER_XSS_FILTER = True

# COOKIE只能通过HTTPS进行传输
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

```
# 防止点击劫持攻击手段 - 修改HTTP协议响应头  
# 当前网站是不允许使用<iframe>标签进行加载的  
X_FRAME_OPTIONS = 'DENY'
```

#### 4. 敏感信息放到环境变量或文件中。

```
SECRET_KEY = os.environ['SECRET_KEY']  
  
DB_USER = os.environ['DB_USER']  
DB_PASS = os.environ['DB_PASS']  
  
REDIS_AUTH = os.environ['REDIS_AUTH']
```

### 更新服务器Python环境到3.x

说明：如果需要清除之前的安装，就删除对应的文件和文件夹即可

#### 1. 安装底层依赖库。

```
yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel rea
```



#### 2. 下载Python源代码。

```
wget https://www.python.org/ftp/python/3.7.1/Python-3.7.1.tar.xz
```

#### 3. 解压缩和解归档。

```
xz -d Python-3.7.1.tar.xz  
tar -xvf Python-3.7.1.tar
```

#### 4. 执行配置生成Makefile（构建文件）。

```
cd Python-3.7.1  
./configure --prefix=/usr/local/python37 --enable-optimizations
```

#### 5. 构建和安装。

```
make && make install
```

#### 6. 配置PATH环境变量（用户或系统环境变量）并激活。

```
vim ~/.bash_profile  
vim /etc/profile
```

... 此处省略上面的代码...

```
export PATH=$PATH:/usr/local/python37/bin
```

... 此处省略下面的代码...

```
source ~/.bash_profile  
source /etc/profile
```

## 7. 注册软链接 ( 符号链接 ) - 这一步不是必须的 , 但通常会比较有用。

```
ln -s /usr/local/python37/bin/python3 /usr/bin/python3
```

## 8. 测试 Python 环境是否更新成功 ( 安装 Python 3 一定不能破坏原来的 Python 2 ) 。

```
python3 --version  
python --version
```

## 项目目录结构

假设项目文件夹为 project , 下面的五个子目录分别是 : code 、 conf 、 logs 、 stat 和 venv 分别用来保存项目的代码、配置文件、日志文件、静态资源和虚拟环境。其中 , conf 目录下的子目录 cert 中保存了配置 HTTPS 需要使用的证书和密钥 ; code 目录下的项目代码可以通过版本控制工具从代码仓库中检出 ; 虚拟环境可以通过工具 ( 如 : venv 、 virtualenv 、 pyenv 等 ) 进行创建。

```
project  
|__ code  
    |__ fangtx  
        |__ api  
        |__ common  
        |__ fangtx  
        |__ forum  
        |__ rent  
        |__ user  
        |__ manage.py  
        |__ README.md  
        |__ static  
        |__ templates  
|__ conf  
    |__ cert
```

```
    └── 214915882850706.key
        └── 214915882850706.pem
    ├── nginx.conf
    └── uwsgi.ini
    └── logs
        ├── access.log
        ├── error.log
        └── uwsgi.log
    └── stat
        └── css
        └── images
        └── js
    └── venv
        ├── bin
        │   ├── activate
        │   ├── activate.csh
        │   ├── activate.fish
        │   ├── celery
        │   ├── celerybeat
        │   ├── celeryd
        │   ├── celeryd-multi
        │   ├── coverage
        │   ├── coverage3
        │   ├── coverage-3.7
        │   ├── django-admin
        │   ├── django-admin.py
        │   ├── easy_install
        │   ├── easy_install-3.7
        │   ├── pip
        │   ├── pip3
        │   ├── pip3.7
        │   └── __pycache__
        ├── pyrsa-decrypt
        ├── pyrsa-decrypt-bigfile
        ├── pyrsa-encrypt
        ├── pyrsa-encrypt-bigfile
        ├── pyrsa-keygen
        ├── pyrsa-priv2pub
        ├── pyrsa-sign
        ├── pyrsa-verify
        ├── python -> python3
        ├── python3 -> /usr/bin/python3
        └── uwsgi
    └── include
    └── lib
        └── python3.7
    └── lib64 -> lib
    └── pip-selfcheck.json
    └── pyvenv.cfg
```

下面以阿里云为例，简单说明如何为项目注册域名、解析域名以及购买权威机构颁发的证书。

## 1. 注册域名。

The screenshot shows the AliCloud Domain Registration interface. At the top, there are navigation links: 阿里云首页 > 域名与网站 > 域名服务 > 域名查询结果. Below this is a search bar with the input 'jackfrued'. To the right of the search bar are dropdown menus for '单选模式' (Single Selection Mode), '批量注册' (Batch Registration), 'CN地方域名查询' (Local CN Domain Query), '域名智能推荐' (Domain Intelligent Recommendation), '白金域名' (Platinum Domains), and '价格总览>>' (Price Overview). A blue button labeled '体验新版' (Experience New Version) is also present. The main search result for 'jackfrued.com' (未注册) is displayed, showing a price of 55 元/首年 (55 Yuan/First Year) with options to add it to a list. Other related domain suggestions like '.cn', '.net', and '.top' are listed below with their respective prices and add-to-list buttons.

## 2. 域名备案。



## 3. 域名解析。

The screenshot shows the Cloud DNS Management interface. On the left, a sidebar menu includes '云解析DNS', '域名解析' (selected), 'vip实例管理', '操作记录', '辅助DNS', and 'PrivateZone'. The main area is titled '域名解析列表' (Domain Parsing List) and contains a notice: '公告: .com/.net/.cn/.xin/.top/.xyz/.vip/.club/.shop/.wang/.ren等域名注册成功后必须进行域名实名认证, 否则域名无法进行DNS解析, 查看详细' (Announcement: After successfully registering domains such as .com/.net/.cn/.xin/.top/.xyz/.vip/.club/.shop/.wang/.ren, domain real-name authentication must be performed, otherwise the domain cannot be parsed. View details). Below this is a search bar with filters for '全部域名' (All Domains), '域名快速搜索' (Quick Domain Search), '搜索' (Search), '批量操作' (Batch Operations), and '创建VIP实例' (Create VIP Instance). A table lists domains with columns for '域名' (Domain), '状态' (Status), and '操作' (Operations). A blue button at the top right says '管理自动续费' (Manage automatic renewal) and another at the bottom right says '添加域名' (Add domain).

解析设置 jackfrued.xyz

当前分配的DNS服务器是: dns27.hichina.com, dns28.hichina.com

记录类型	主机记录	解析线路(isp)	记录值	MX优先级	TTT	状态	操作
A	@	默认	192.168.1.111	--	10分钟	正常	<a href="#">修改</a>   <a href="#">暂停</a>   <a href="#">删除</a>   <a href="#">备注</a>

#### 4. 购买证书。

云盾证书服务(包年)

选择品牌: GlobalSign

GMO GlobalSign是全球最早的数字证书认证机构之一，一直致力于网络安全认证及数字证书服务，是一个备受信赖的CA和SSL数字证书提供商。

证书类型: 专业版OV SSL

OV SSL, 提供加密功能, 对申请者做严格的身份审核验证, 提供可信身份证明。

保护类型: 通配符域名

当前配置

- 选择品牌: GlobalSign
- 证书类型: 专业版OV SSL
- 保护类型: 通配符域名
- 域名个数: 1个
- 购买数量: 1
- 购买时长: 1年
- 配置费用: ￥11,090.80

¥11,090.80

可以使用类似于sftp的工具将证书上传到 conf/cert 目录，然后使用git克隆项目代码到 code 目录。

```
cd code
git clone <url>
```

回到项目目录，创建并激活虚拟环境。

```
python3 -m venv venv
source venv/bin/activate
```

重建项目依赖项。

```
pip install -r code/teamproject/requirements.txt
```

## uWSGI的配置

## 1. 安装uWSGI。

```
pip install uwsgi
```

## 2. 修改uWSGI的配置文件（ /root/project/conf/uwsgi.ini ）。

```
[uwsgi]
# 配置前导路径
base=/root/project
# 配置项目名称
name=teamproject
# 守护进程
master=true
# 进程个数
processes=4
# 虚拟环境
pythonhome=%(base)/venv
# 项目地址
chdir=%(base)/code/%(name)
# 指定python解释器
pythonpath=%(pythonhome)/bin/python
# 指定uwsgi文件
module=%(name).wsgi
# 通信的地址和端口(自己服务器的IP地址和端口)
socket=172.18.61.250:8000
# 日志文件地址
logto=%(base)/logs/uwsgi.log
```

说明：可以先将“通信的地址和端口”项等号前面改为http来进行测试，如果没有问题再改回成socket，然后通过Nginx来实现项目的“动静分离”（静态资源交给Nginx处理，动态内容交给 uWSGI 处理）。按照下面的方式可以启动uWSGI服务器。

## 3. 启动服务器。

```
uwsgi --ini conf/uwsgi.ini
```

## Nginx的配置

### 1. 安装Nginx。

```
yum -y install nginx
```

### 2. 修改全局配置文件（ /etc/nginx/nginx.conf ）。

```

# 配置用户
user root;
# 工作进程数(建议跟CPU的核数量一致)
worker_processes auto;
# 错误日志
error_log /var/log/nginx/error.log;
# 进程文件
pid /run/nginx.pid;
# 包含其他的配置
include /usr/share/nginx/modules/*.conf;
# 工作模式(多路IO复用方式)和连接上限
events {
    use epoll;
    worker_connections 1024;
}
# HTTP服务器相关配置
http {
    # 日志格式
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';
    # 访问日志
    access_log /var/log/nginx/access.log main;
    # 开启高效文件传输模式
    sendfile on;
    # 用sendfile传输文件时有利于改善性能
    tcp_nopush on;
    # 禁用Nagle来解决交互性问题
    tcp_nodelay on;
    # 客户端保持连接时间
    keepalive_timeout 30;
    types_hash_max_size 2048;
    # 包含MIME类型的配置
    include /etc/nginx/mime.types;
    # 默认使用二进制流格式
    default_type application/octet-stream;
    # 包含其他配置文件
    include /etc/nginx/conf.d/*.conf;
    # 包含项目的Nginx配置文件
    include /root/project/conf/*.conf;
}

```

### 3. 编辑局部配置文件 ( /root/project/conf/nginx.conf ) 。

```

server {
    listen 80;
    server_name _;
    access_log /root/project/logs/access.log;
    error_log /root/project/logs/error.log;
    location / {
        include uwsgi_params;
        uwsgi_pass 172.18.61.250:8000;
    }
}

```

```
        }
    location /static/ {
        alias /root/project/stat/;
        expires 30d;
    }
}
server {
    listen      443;
    server_name _;
    ssl         on;
    access_log /root/project/logs/access.log;
    error_log  /root/project/logs/error.log;
    ssl_certificate /root/project/conf/cert/214915882850706.pem;
    ssl_certificate_key /root/project/conf/cert/214915882850706.key;
    ssl_session_timeout 5m;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDSA:AES:HIGH:!NULL:!aNULL:!MD5:
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    location / {
        include uwsgi_params;
        uwsgi_pass 172.18.61.250:8000;
    }
    location /static/ {
        alias /root/project/static/;
        expires 30d;
    }
}
```

到此为止，我们可以启动Nginx来访问我们的应用程序，HTTP和HTTPS都是没有问题的，如果Nginx已经运行，在修改配置文件后，我们可以用下面的命令重新启动Nginx。

#### 4. 重启Nginx服务器。

```
nginx -s reload
```

或

```
systemctl restart nginx
```

说明：可以对Django项目使用 `python manage.py collectstatic` 命令将静态资源收集到指定目录下，要做到这点只需要在项目的配置文件 `settings.py` 中添加 `STATIC_ROOT` 配置即可。

## 负载均衡配置

下面的配置中我们使用Nginx实现负载均衡，为另外的三个Nginx服务器（通过Docker创建）提供反向代理服务。

```
docker run -d -p 801:80 --name nginx1 nginx:latest
docker run -d -p 802:80 --name nginx2 nginx:latest
docker run -d -p 803:80 --name nginx3 nginx:latest

user root;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

# 为HTTP服务配置负载均衡
http {
    upstream fangtx {
        server 172.18.61.250:801 weight=4;
        server 172.18.61.250:802 weight=2;
        server 172.18.61.250:803 weight=2;
    }

    server {
        listen      80 default_server;
        listen      [::]:80 default_server;
        listen      443 ssl;
        listen      [::]:443 ssl;

        ssl on;
        access_log /root/project/logs/access.log;
        error_log /root/project/logs/error.log;
        ssl_certificate /root/project/conf/cert/214915882850706.pem;
        ssl_certificate_key /root/project/conf/cert/214915882850706.key;
        ssl_session_timeout 5m;
        ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDSA:HIGH:!NULL:!aNULL;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        location / {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_buffering off;
            proxy_pass http://fangtx;
        }
    }
}
```

说明：Nginx在配置负载均衡时，默认使用WRR（加权轮询算法），除此之外还支持ip\_hash、fair（需要安装upstream\_fair模块）和url\_hash算法。此外，在配置upstream模块时可以指定服务器的状态值，包括：backup（备份机器，其他服务器不可用时才将请求分配到该机器）、down、fail\_timeout（请求失败达到maxfails后的暂停服务时间）、maxfails（允许请求失败的次数）和weight（轮询的权重）。

## Keepalived

当使用Nginx进行负载均衡配置时，要考虑负载均衡服务器宕机的情况。为此可以使用Keepalived来实现负载均衡主机和备机的热切换，从而保证系统的高可用性。Keepalived的配置还是比较复杂，通常由专门做运维的人进行配置，一个基本的配置可以参照[《Keepalived的配置和使用》](#)。

## MySQL主从复制

下面还是基于Docker来演示如何配置MySQL主从复制。我们事先准备好MySQL的配置文件以及保存MySQL数据和运行日志的目录，然后通过Docker的数据卷映射来指定容器的配置、数据和日志文件的位置。

```
root
└── mysql
    ├── master
    │   ├── conf
    │   └── data
    └── slave-1
        ├── conf
        └── data
    └── slave-2
        ├── conf
        └── data
    └── slave-3
        ├── conf
        └── data
```

1. MySQL的配置文件（master和slave的配置文件需要不同的server-id）。

```
[mysqld]
pid-file=/var/run/mysqld/mysqld.pid
socket=/var/run/mysqld/mysqld.sock
datadir=/var/lib/mysql
log-error=/var/log/mysql/error.log
server-id=1
log-bin=/var/log/mysql/mysql-bin.log
expire_logs_days=30
max_binlog_size=256M
symbolic-links=0
```

```
# slow_query_log=ON  
# slow_query_log_file=/var/log/mysql/slow.log  
# long_query_time=1
```

## 2. 创建和配置master。

```
docker run -d -p 3306:3306 --name mysql-master \  
-v /root/mysql/master/conf:/etc/mysql/mysql.conf.d \  
-v /root/mysql/master/data:/var/lib/mysql \  
-e MYSQL_ROOT_PASSWORD=123456 mysql:5.7  
  
docker exec -it mysql-master /bin/bash  
  
mysql -u root -p  
Enter password:  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 1  
Server version: 5.7.23-log MySQL Community Server (GPL)  
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql> grant replication slave on *.* to 'slave'@'%' identified by 'iamslave';  
Query OK, 0 rows affected, 1 warning (0.00 sec)  
  
mysql> flush privileges;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> show master status;  
+-----+-----+-----+-----+-----+  
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |  
+-----+-----+-----+-----+-----+  
| mysql-bin.000003 | 590 | | | |  
+-----+-----+-----+-----+-----+  
1 row in set (0.00 sec)  
  
mysql> quit  
Bye  
exit
```

上面创建Docker容器时使用的 -v 参数 ( --volume ) 表示映射数据卷，冒号前是宿主机的目录，冒号后是容器中的目录，这样相当于将宿主机中的目录挂载到了容器中。

## 3. 创建和配置slave。

```
docker run -d -p 3308:3306 --name mysql-slave-1 \
-v /root/mysql/slave-1/conf:/etc/mysql/mysql.conf.d \
-v /root/mysql/slave-1/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
--link mysql-master:mysql-master mysql:5.7

docker run -d -p 3309:3306 --name mysql-slave-2 \
-v /root/mysql/slave-2/conf:/etc/mysql/mysql.conf.d \
-v /root/mysql/slave-2/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
--link mysql-master:mysql-master mysql:5.7

docker run -d -p 3310:3306 --name mysql-slave-3 \
-v /root/mysql/slave-3/conf:/etc/mysql/mysql.conf.d \
-v /root/mysql/slave-3/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
--link mysql-master:mysql-master mysql:5.7

docker exec -it mysql-slave-1 /bin/bash

mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.23-log MySQL Community Server (GPL)
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> reset slave;
Query OK, 0 rows affected (0.02 sec)

mysql> change master to master_host='mysql-master', master_user='slave', master_p@%
Query OK, 0 rows affected, 2 warnings (0.03 sec)

mysql> start slave;
Query OK, 0 rows affected (0.01 sec)

mysql> show slave status\G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
    Master_Host: mysql57
    Master_User: slave
    Master_Port: 3306
    Connect_Retry: 60
    Master_Log_File: mysql-bin.000001
    Read_Master_Log_Pos: 590
        Relay_Log_File: f352f05eb9d0-relay-bin.000002
        Relay_Log_Pos: 320
    Relay_Master_Log_File: mysql-bin.000001
```

```
        Slave_IO_Running: Yes
        Slave_SQL_Running: Yes
        Replicate_Do_DB:
        Replicate_Ignore_DB:
        Replicate_Do_Table:
        Replicate_Ignore_Table:
        Replicate_Wild_Do_Table:
        Replicate_Wild_Ignore_Table:
        Last_Error:
        Skip_Counter: 0
        Exec_Master_Log_Pos: 590
        Relay_Log_Space: 534
        Until_Condition: None
        Until_Log_File:
        Until_Log_Pos: 0
        Master_SSL_Allowed: No
        Master_SSL_CA_File:
        Master_SSL_CA_Path:
        Master_SSL_Cert:
        Master_SSL_Cipher:
        Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
        Last_IO_Errno: 0
        Last_IO_Error:
        Last_SQL_Errno: 0
        Last_SQL_Error:
Replicate_Ignore_Server_Ids:
        Master_Server_Id: 1
        Master_UUID: 30c38043-ada1-11e8-8fa1-0242ac110002
        Master_Info_File: /var/lib/mysql/master.info
        SQL_Delay: 0
        SQL_Remaining_Delay: NULL
Slave_SQL_Running_State: Slave has read all relay log; waiting for more update
        Master_Retry_Count: 86400
        Master_Bind:
        Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp:
        Master_SSL_Crl:
        Master_SSL_Crlpath:
        Retrieved_Gtid_Set:
        Executed_Gtid_Set:
        Auto_Position: 0
        Replicate_Rewrite_DB:
        Channel_Name:
        Master_TLS_Version:
1 row in set (0.00 sec)
```

```
mysql> quit
Bye
exit
```

接下来可以如法炮制配置出slave2和slave3，这样就可以搭建起一个“一主带三从”的主从复制环境。上面创建容器时使用的`--link`参数用来配置容器在网络上的主机名（网络地址别名）。

配置好主从复制后，写数据的操作应该master上执行，而读数据的操作应该在slave上完成。为此，在Django项目中需要配置DATABASE\_ROUTERS并通过自定义的主从复制路由类来实现读写分离操作，如下所示：

```
DATABASE_ROUTERS = [
    # 此处省略其他配置
    'common.routers.MasterSlaveRouter',
]

class MasterSlaveRouter(object):
    """主从复制路由"""

    @staticmethod
    def db_for_read(model, **hints):
        """
        Attempts to read auth models go to auth_db.
        """
        return random.choice(['slave1', 'slave2', 'slave3'])

    @staticmethod
    def db_for_write(model, **hints):
        """
        Attempts to write auth models go to auth_db.
        """
        return 'default'

    @staticmethod
    def allow_relation(obj1, obj2, **hints):
        """
        Allow relations if a model in the auth app is involved.
        """
        return None

    @staticmethod
    def allow_migrate(db, app_label, model_name=None, **hints):
        """
        Make sure the auth app only appears in the 'auth_db'
        database.
        """
        return True
```

上面的内容参考了Django官方文档的[DATABASE\\_ROUTERS配置](#)，对代码进行了适当的调整。

事实上，项目上线中最为麻烦的事情就是配置软件运行环境，环境的差异会给软件的安装和部署带来诸多的麻烦，而Docker正好可以解决这个问题。关于Docker在之前的文档中我们已经介绍过了，接下来我们对Docker的知识做一些必要的补充。

## 1. 创建镜像文件。

将容器保存成镜像：

```
docker commit -m "..." -a "jackfrued" <container-name> jackfrued/<image-name>
```

使用Dockerfile构建镜像：

```
# 指定基础镜像文件
FROM centos:latest

# 指定维护者信息
MAINTAINER jackfrued

# 执行命令
RUN yum -y install gcc
RUN cd ~
RUN mkdir -p project/code
RUN mkdir -p project/logs

# 拷贝文件
COPY ... 

# 暴露端口
EXPOSE ... 

# 在容器启动时执行命令
CMD ~/init.sh
```

```
docker build -t jackfrued/<image-name> .
```

## 2. 镜像的导入和导出。

```
docker save -o <file-name>.tar <image-name>:<version>
docker load -i <file-name>.tar
```

## 3. 推送到DockerHub服务器。

```
docker tag <image-name>:<version> jackfrued/<name>
docker login
docker push jackfrued/<name>
```

#### 4. 容器之间的通信。

```
docker run --link <container-name>:<alias-name>
```

如果我们在Docker中完成项目的部署，并且将整个部署好的容器打包成镜像文件进行分发和安装，这样就可以解决项目在多个节点上进行部署时可能遇到的麻烦，而且整个部署可以在很短的时间内完成。

## Supervisor

Supervisor是一个用Python写的进程管理工具，可以很方便的用来在类Unix系统下启动、重启（自动重启程序）和关闭进程，目前Supervisor暂时还没有提供对Python 3的支持，可以通过Python 2来安装和运行Supervisor，再通过Supervisor来管理Python 3的程序。

### 1. 安装Supervisor。

```
pip install virtualenv
virtualenv -p /usr/bin/python venv
source venv/bin/activate
pip install supervisor
```

### 2. 查看Supervisor的配置文件。

```
vim /etc/supervisord.conf
```

```
; 此处省略上面的代码
; The [include] section can just contain the "files" setting. This
; setting can list multiple files (separated by whitespace or
; newlines). It can also contain wildcards. The filenames are
; interpreted as relative to this file. Included files *cannot*
; include files themselves.
[include]
files = supervisord.d/*.ini
```

可以看出自定义的管理配置代码可以放在 /etc/supervisord.d 目录中，并且文件名以 .ini 作为后缀即可。

### 3. 编写自己的配置文件 fangtx.ini 并放在 /etc/supervisord.d 目录中。

```
[program:project]
command=uwsgi --ini /root/project/conf/uwsgi.ini
stopsignal=QUIT
autostart=true
autorestart=true
redirect_stderr=true
```

```

[program:celery]
; Set full path to celery program if using virtualenv
command=/root/project/venv/bin/python manage.py celery -A fangtx worker
user=root
numprocs=1
stdout_logfile=/var/log/supervisor/celery.log
stderr_logfile=/var/log/supervisor/celery_error.log
autostart=true
autorestart=true
startsecs=10

; Need to wait for currently executing tasks to finish at shutdown.
; Increase this if you have very long running tasks.
;stopwaitsecs = 600

; When resorting to send SIGKILL to the program to terminate it
; send SIGKILL to its whole process group instead,
; taking care of its children as well.
killasgroup=true
; Set Celery priority higher than default (999)
; so, if rabbitmq is supervised, it will start first.
priority=1000

```

#### 4. 启动Supervisor。

```
supervisorctl -c /etc/supervisord.conf
```

## 其他服务

### 1. 常用开源软件。

功能	开源方案
版本控制工具	Git、Mercurial、SVN
缺陷管理	Redmine、Mantis
负载均衡	Nginx、LVS、HAProxy
邮件服务	Postfix、Sendmail
HTTP服务	Nginx、Apache
消息队列	RabbitMQ、ZeroMQ、Redis
文件系统	FastDFS
基于位置服务 ( LBS )	MongoDB、Redis
监控服务	Nagios、Zabbix

功能	开源方案
关系型数据库	MySQL、PostgreSQL
非关系型数据库	MongoDB、Redis、Cassandra
搜索引擎	ElasticSearch、Solr
缓存服务	Memcached、Redis

## 2. 常用云服务。

功能	可用的云服务
团队协作工具	Teambition、钉钉
代码托管平台	Github、Gitee、CODING
邮件服务	SendCloud
云存储 ( CDN )	七牛、OSS、LeanCloud、Bmob、又拍云、AWS
移动端推送	极光、友盟、百度
即时通信	环信、融云
短信服务	云片、极光、Luosimao、又拍云
第三方登录	友盟、ShareSDK
网站监控和统计	阿里云监控、监控宝、百度云观测、小鸟云

# 附录：

Branch: master ▾

Find file Copy path

Python-100-Days / 那些年我们踩过的那些坑.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



191 lines (130 sloc) 11.1 KB

## 那些年我们踩过的那些坑

### 坑1 - 整数比较的坑

在 Python 中一切都是对象，整数也是对象，在比较两个整数时有两个运算符 `==` 和 `is`，它们的区别是：

- `is` 比较的是两个整数对象的 `id` 值是否相等，也就是比较两个引用是否代表了内存中同一个地址。
- `==` 比较的是两个整数对象的内容是否相等，使用 `==` 时其实是调用了对象的 `__eq__()` 方法。

知道了 `is` 和 `==` 的区别之后，我们可以来看看下面的代码，了解 Python 中整数比较有哪些坑：

```
def main():
    x = y = -1
    while True:
        x += 1
        y += 1
        if x is y:
            print('%d is %d' % (x, y))
        else:
            print('Attention! %d is not %d' % (x, y))
            break

    x = y = 0
    while True:
        x -= 1
        y -= 1
        if x is y:
            print('%d is %d' % (x, y))
        else:
            print('Attention! %d is not %d' % (x, y))
```

```
break
```

```
if __name__ == '__main__':
    main()
```

上面代码的部分运行结果如下图所示，出现这个结果的原因是Python出于对性能的考虑所做的一项优化。对于整数对象，Python把一些频繁使用的整数对象缓存起来，保存到一个叫 `small_ints` 的链表中，在Python的整个生命周期内，任何需要引用这些整数对象的地方，都不再重新创建新的对象，而是直接引用缓存中的对象。Python把频繁使用的整数对象的值定在[-5, 256]这个区间，如果需要这个范围的整数，就直接从 `small_ints` 中获取引用而不是临时创建新的对象。因为大于256或小于-5的整数不在该范围之内，所以就算两个整数的值是一样，但它们是不同的对象。

```
253 is 253
254 is 254
255 is 255
256 is 256
Attention! 257 is not 257
-1 is -1
-2 is -2
-3 is -3
-4 is -4
-5 is -5
Attention! -6 is not -6
```

当然仅仅如此这个坑就不值一提了，如果你理解了上面的规则，我们就再看看下面的代码。

```
import dis
a = 257

def main():
    b = 257 # 第6行
    c = 257 # 第7行
    print(b is c) # True
    print(a is b) # False
    print(a is c) # False

if __name__ == "__main__":
    main()
```

程序的执行结果已经用注释写在代码上了。够坑吧！看上去 `a`、`b` 和 `c` 的值都是一样的，但是 `is` 运算的结果却不一样。为什么会出现这样的结果，首先我们来说说Python程序中的代码块。所谓代码块是程序的一个最小的基本执行单位，一个模块文件、一个函数体、一个类、交互式命令中的单行代码都叫做一个代码块。上面的代码由两个代码块构成，`a = 257` 是一个代码块，`main` 函数是另外一个代码块。Python内部为了进一步提高性能，凡是在一个代码块中创建的整数对象，如果值不在 `small_ints` 缓存范围之内，但在同一个代码块中已经存在一个值与其相同的整数对象了，那么就直接引用该对象，否则创建一个新的对象出来，这条规则对不在 `small_ints` 范围的负数并不适用，对负数值浮点数也不适用，但对非负浮点数和字符串都是适用的，这一点读者可以自行证明。所以 `b is c` 返回了 `True`，而 `a` 和 `b` 不在同一个代码块中，虽然值都是257，但却是两个不同的对象，`is` 运算的结果自然是 `False` 了。为了验证刚刚的结论，我们可以借用 `dis` 模块（听名字就知道是进行反汇编的模块）从字节码的角度来看看这段代码。如果不理解什么是字节码，可以先看看《谈谈 Python 程序的运行原理》这篇文章。可以先用 `import dis` 导入 `dis` 模块并按照如下所示的方式修改代码。

```
if __name__ == "__main__":
    main()
    dis.dis(main)
```

代码的执行结果如下图所示。可以看出代码第6行和第7行，也就是 `main` 函数中的257是从同一个位置加载的，因此是同一个对象；而代码第9行的 `a` 明显是从不同的地方加载的，因此引用的是不同的对象。

6	0 LOAD_CONST	1 (257)
	2 STORE_FAST	0 (b)
7	4 LOAD_CONST	1 (257)
	6 STORE_FAST	1 (c)
8	8 LOAD_GLOBAL	0 (print)
	10 LOAD_FAST	0 (b)
	12 LOAD_FAST	1 (c)
	14 COMPARE_OP	8 (is)
	16 CALL_FUNCTION	1
	18 POP_TOP	
9	20 LOAD_GLOBAL	0 (print)
	22 LOAD_GLOBAL	1 (a)
	24 LOAD_FAST	0 (b)
	26 COMPARE_OP	8 (is)
	28 CALL_FUNCTION	1

如果还想对这个问题进行进一步深挖，推荐大家阅读[《Python整数对象实现原理》](#)这篇文章。

## 坑2 - 嵌套列表的坑

Python中有一种内置的数据类型叫列表，它是一种容器，可以用来承载其他的对象（准确的说是其他对象的引用），列表中的对象可以称为列表的元素，很明显我们可以把列表作为列表中的元素，这就是所谓的嵌套列表。嵌套列表可以模拟出现实中的表格、矩阵、2D游戏的地图（如植物大战僵尸的花园）、棋盘（如国际象棋、黑白棋）等。但是在使用嵌套的列表时要小心，否则很可能遭遇非常尴尬的情况，下面是一个小例子。

```
def main():
    names = ['关羽', '张飞', '赵云', '马超', '黄忠']
    subjs = ['语文', '数学', '英语']
    scores = [[0] * 3] * 5
    for row, name in enumerate(names):
        print('请输入%s的成绩' % name)
        for col, subj in enumerate(subjs):
            scores[row][col] = float(input(subj + ': '))
    print(scores)

if __name__ == '__main__':
    main()
```

我们希望录入5个学生3门课程的成绩，于是定义了一个有5个元素的列表，而列表中的每个元素又是一个由3个元素构成的列表，这样一个列表的列表刚好跟一个表格是一致的，相当于有5行3列，接下来我们通过嵌套的for-in循环输入每个学生3门课程的成绩。程序执行完成后我们发现，每个学生3门课程的成绩是一模一样的，而且就是最后录入的那个学生的成绩。

要想把这个坑填平，我们首先要区分对象和对象的引用这两个概念，而要区分这两个概念，还得先说说内存中的栈和堆。我们经常会听人说起“堆栈”这个词，但实际上“堆”和“栈”是两个不同的概念。众所周知，一个程序运行时需要占用一些内存空间来存储数据和代码，那么这些内存从逻辑上又可以做进一步的划分。对底层语言（如C语言）有所了解的程序员大都知道，程序中可以使用的内存从逻辑上可以为五个部分，按照地址从高到低依次是：栈（stack）、堆（heap）、数据段（data segment）、只读数据段（static area）和代码段（code segment）。其中，栈用来存储局部、临时变量，以及函数调用时保存现场和恢复现场需要用到的数据，这部分内存从代码块开始执行时自动分配，代码块执行结束时自动释放，通常由编译器自动管理；堆的大小不固定，可以动态的分配和回收，因此如果程序中有大量的数据需要处理，这些数据通常都放在堆上，如果堆空间没有正确的被释放会引发内存泄露的问题，而像Python、Java等编程语言都使用了垃圾回收机制来实现自动化的内存管理（自动回收不再使用的堆空间）。所以下面的代码中，变量 `a` 并不是真正的对象，它是对象的引用，相当于记录了对象在堆空间的地址，通过这个地址我们可以访问到对应的对象；同理，变量 `b` 是列表容器的引用，它引用了堆空间上的列表容器，而列表容器中并没有保存真正的对象，它保存的也仅仅是对象的引用。

```
a = object()
b = ['apple', 'pitaya', 'grape']
```

知道了这一点，我们可以回过头看看刚才的程序，我们对列表进行 `[[0] * 3] * 5` 操作时，仅仅是将 `[0, 0, 0]` 这个列表的地址进行了复制，并没有创建新的列表对象，所以容器中虽然有5个元素，但是这5个元素引用了同一个列表对象，这一点可以通过 `id` 函数检查 `scores[0]` 和 `scores[1]` 的地址得到证实。所以正确的代码应该按照如下的方式进行修改。

```
def main():
    names = ['关羽', '张飞', '赵云', '马超', '黄忠']
    subjs = ['语文', '数学', '英语']
    scores = [[]] * 5
    for row, name in enumerate(names):
        print('请输入%s的成绩' % name)
        scores[row] = [0] * 3
        for col, subj in enumerate(subjs):
            scores[row][col] = float(input(subj + ': '))
    print(scores)

if __name__ == '__main__':
    main()
```

或者

```
def main():
    names = ['关羽', '张飞', '赵云', '马超', '黄忠']
    subjs = ['语文', '数学', '英语']
    scores = [[0] * 3 for _ in range(5)]
    for row, name in enumerate(names):
        print('请输入%s的成绩' % name)
        scores[row] = [0] * 3
        for col, subj in enumerate(subjs):
            scores[row][col] = float(input(subj + ': '))
    print(scores)

if __name__ == '__main__':
    main()
```

如果对内存的使用不是很理解，可以看看[PythonTutor网站](#)上提供的代码可视化执行功能，通过可视化执行，我们可以看到内存是如何分配的，从而避免在使用嵌套列表或者复制对象时可能遇到的坑。

[How do I use this?](#)

Python 3.6

```
1 scores = [[0] * 3] * 5
```

[Edit this code](#)

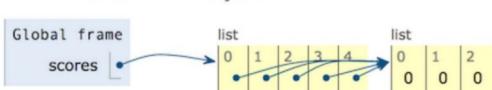
line that has just executed  
next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First < Back Program terminated Forward > Last >>

Created by [@pgbovine](#). Support with a [small donation](#).

Help improve this tool by clicking whenever you learn something:  
[I just cleared up a misunderstanding!](#) [I just fixed a bug in my code!](#)



Python 3.6

```
1 scores = [[0] * 3 for _ in range(5)]
```

[Edit this code](#)

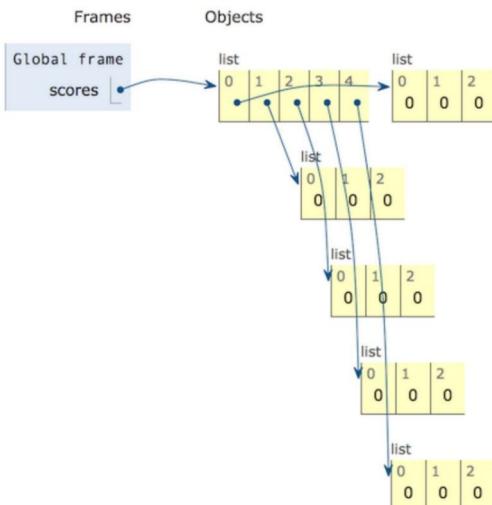
line that has just executed  
next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First < Back Program terminated Forward > Last >>

Created by [@pgbovine](#). Support with a [small donation](#).

Help improve this tool by clicking whenever you learn something:  
[I just cleared up a misunderstanding!](#) [I just fixed a bug in my code!](#)



### 坑3 - 访问修饰符的坑

用Python做过面向对象编程的人都知道，Python的类提供了两种访问控制权限，一种是公开，一种是私有（在属性或方法前加上双下划线）。而用惯了Java或C#这类编程语言的人都知道，类中的属性（数据抽象）通常都是私有的，其目的是为了将数据保护起来；而类中的方法（行为抽象）通常都是公开的，因为方法是对象向外界提供的服务。但是Python并没有从语法层面确保私有成员的私密性，因为它只是对类中所谓的私有成员进行了命名的变换，如果知道命名的规则照样可以直接访问私有成员，请看下面的代码。

```
class Student(object):

    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def __str__(self):
        return self.__name + ': ' + str(self.__age)

def main():
    stu = Student('骆昊', 38)
```

```
# 'Student' object has no attribute '__name'  
# print(stu.__name)  
# 用下面的方式照样可以访问类中的私有成员  
print(stu._Student__name)  
print(stu._Student__age)  
  
if __name__ == '__main__':  
    main()
```

Python为什么要做出这样的设定呢？用一句广为流传的格言来解释这个问题：“We are all consenting adults here”（我们都是成年人）。这句话表达了很多Python程序员的一个共同观点，那就是开放比封闭要好，我们应该自己对自己的行为负责而不是从语言层面来限制对数据或方法的访问。

所以在Python中我们实在没有必要将类中的属性或方法用双下划线开头的命名处理成私有的成员，因为这并没有任何实际的意义。如果想对属性或方法进行保护，我们建议用单下划线开头的受保护成员，虽然它也不能真正保护这些属性或方法，但是它相当于给调用者一个暗示，让调用者知道这是不应该直接访问的属性或方法，而且这样做并不影响子类去继承这些东西。

需要提醒大家注意的是，Python类中的那些魔法方法，如 `__str__`、`__repr__` 等，这些方法并不是私有成员哦，虽然它们以双下划线开头，但是他们也是以双下划线结尾的，这种命名并不是私有成员的命名，这一点对初学者来说真的很坑。

Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / 知乎问题回答.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)

97 lines (72 sloc) 6.94 KB

## 知乎问题回答

### Python学习完基础语法知识后，如何进一步提高？

如果你已经完成了Python基础语法的学习，想要知道接下来如何提高，那么你得先问问自己你要用Python来做什么？目前学习Python后可能的就业方向包括以下几个领域，我把每个领域需要的技术作为了一个简单的关键词摘要。

说明：以下数据参考了主要的招聘门户网站以及职友集。

职位	所需技能	招聘需求量
Python后端开发工程师	Python基础 Django / Flask / Tornado / Sanic RESTful / 接口文档撰写 MySQL / Redis / MongoDB / ElasticSearch Linux / Git / Scrum / PyCharm	大
Python爬虫开发工程师	Python基础 常用标准库和三方库 Scrapy / PySpider Selenium / Appnium Redis / MongoDB / MySQL 前端 / HTTP(S) / 抓包工具	较少
Python量化交易开发工程师	Python基础 数据结构 / 算法 / 设计模式 NoSQL ( KV数据库 ) 金融学 ( 两融、期权、期货、股票 ) / 数字货币	较大 ( 一线城市 )

职位	所需技能	招聘需求量
Python数据分析工程师 / Python机器学习工程师	统计学专业 / 数学专业 / 计算机专业 Python基础 / 算法设计 SQL / NoSQL / Hive / Hadoop / Spark NumPy / Scikit-Learn / Pandas / Seaborn PyTorch / Tensorflow / OpenCV	较大 ( 一线城市 )
Python自动化测试工程师	Python基础 / 单元测试 / 软件测试基础 Linux / Shell / JIRA / 禅道 / Jenkins / CI / CD Selenium / Robot Framework / Appium ab / sysbench / JMeter / LoadRunner / QTP	大
Python自动化运维工程师	Python基础 / Linux / Shell Fabric / Ansible / Playbook Zabbix / Saltstack / Puppet Docker / paramiko	较大 ( 一线城市 )
Python云平台开发工程师	Python基础 OpenStack / CloudStack Ovirt / KVM Docker / K8S	较少 ( 一线城市 )

如果弄清了自己将来要做的方向，就可以开始有针对性的学习了，下面给大家一个推荐书籍的清单。

## 1. 入门读物

- 《Python基础教程》 ( *Beginning Python From Novice to Professional* )
- 《Python学习手册》 ( *Learning Python* )
- 《Python编程》 ( *Programming Python* )
- 《Python编程从入门到实践》 ( *Python Crash Course* )
- 《Python Cookbook》

## 2. 进阶读物

- 《软件架构 - Python语言实现》 ( *Software Architecture with Python* )
- 《流畅的Python》 ( *Fluent Python* )
- 《Python设计模式》 ( *Learning Python Design Patterns* )
- 《Python高级编程》 ( *Expert Python Programming* )
- 《Python性能分析与优化》 ( *Mastering Python High Performance* )

### 3. 数据库相关

- 《MySQL必知必会》 ( *MySQL Crash Course* )
- 《深入浅出MySQL - 数据库开发、优化与管理维护》
- 《MongoDB权威指南》 ( *MongoDB: The Definitive Guide* )
- 《Redis实战》 ( *Redis in Action* )
- 《Redis开发与运维》

### 4. Linux / Shell / Docker / 运维

- 《鸟哥的Linux私房菜》
- 《Linux命令行与shell脚本编程大全》 ( *Linux Command Line and Shell Scripting Bible* )
- 《Python自动化运维:技术与最佳实践》
- 《第一本Docker书》 ( *The Docker Book* )
- 《Docker经典实例》 ( *Docker Cookbook* )

### 5. Django / Flask / Tornado

- 《Django基础教程》 ( *Tango with Django* )
- 《轻量级Django》 ( *Lightweight Django* )
- 《精通Django》 ( *Mastering Django: Core* )
- 《Python Web开发 : 测试驱动方法》 ( *Test-Driven Development with Python* )
- 《Two Scoops of Django: Best Practice of Django 1.8》
- 《Flask Web开发 : 基于Python的Web应用开发实战》 ( *Flask Web Development: Developing Web Applications with Python* )
- 《深入理解Flask》 ( *Mastering Flask* )
- 《Introduction to Tornado》

### 6. 爬虫开发

- 《用Python写网络爬虫》 ( *Web Scraping with Python* )
- 《精通Python爬虫框架Scrapy》 ( *Learning Scrapy* )
- 《Python网络数据采集》 ( *Web Scraping with Python* )
- 《Python爬虫开发与项目实战》
- 《Python 3网络爬虫开发实战》

## 7. 数据分析

- 《利用Python进行数据分析》 ( *Python for Data Analysis* )
- 《Python数据科学手册》 ( *Python Data Science Handbook* )
- 《Python金融大数据分析》 ( *Python for Finance* )
- 《Python数据可视化编程实战》 ( *Python Data Visualization Cookbook* )
- 《Python数据处理》 ( *Data Wrangling with Python* )

## 8. 机器学习

- 《Python机器学习基础教程》 ( *Introduction to Machine Learning with Python* )
- 《Python机器学习实践指南》 ( *Python Machine Learning Blueprints* )
- 《Python机器学习实践：测试驱动的开发方法》 ( *Thoughtful Machine Learning with Python A Test Driven Approach* )
- 《Python机器学习经典实例》 ( *Python Machine Learning Cookbook* )
- 《TensorFlow：实战Google深度学习框架》

## 9. 其他书籍

- 《Pro Git》
- 《Selenium自动化测试 - 基于Python语言》 ( *Learning Selenium Testing Tools with Python* )
- 《Selenium自动化测试之道》
- 《Scrum敏捷软件开发》 ( *Software Development using Scrum* )
- 《高效团队开发 - 工具与方法》

当然学习编程，最重要的通过项目实战来提升自己的综合能力，Github上有大量的优质开源项目，其中不乏优质的Python项目。有一个名为“[awesome-python-applications](#)”的项目对这些优质的资源进行了归类并提供了传送门，大家可以了解下。除此之外，还要为大家推荐一个名为“[Python-100-Days](#)”的项目，上面有大量优质的Python学习资料（包括文档、代码和相关资源）。如果自学能力不是那么强，可以通过网络上免费或者付费的视频课程来学习对应的知识；如果自律性没有那么强，那就只能建议花钱参加培训班了，因为花钱在有人监督的环境下学习对很多人来说确实是一个捷径，但是要记得：“师傅领进门，修行靠各人”。选择自己热爱的东西并全力以赴，不要盲目的跟风学习，这一点算是过来人的忠告吧。记得我自己刚开始进入软件开发这个行业时，有人跟我说过这么一句话，现在也分享出来与诸君共勉：“浮躁的人有两种：只观望而不学习的人，只学习而不坚持的人；浮躁的人都不是高手。”

Branch: master ▾

Find file Copy path

## Python-100-Days / 用函数还是用复杂的表达式.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



92 lines (66 sloc) 4.73 KB

## 要不要使用复杂表达式

Perl语言的原作者Larry Wall曾经说过，伟大的程序员都有三个优点：懒惰、暴躁和自负。乍一看这三个词语没有一个是褒义词，但在程序员的世界里，这三个词有不同的意义。首先，懒惰会促使程序员去写一些省事儿的程序来辅助自己或别人更好的完成工作，这样我们就无需做那些重复和繁琐的劳动；同理能够用3行代码解决的事情，我们也不会写出10行代码来。其次，暴躁会让程序员主动的去完成一些你还没有提出的工  
作，去优化自己的代码让它更有效率，能够3秒钟完成的任务，我们绝不能容忍1分钟的等待。最后，自负会促使程序员写出可靠无误的代码，我们写代码不是为了接受批评和指责，而是为了让其他人来膜拜。

那么接下来就有一个很有意思的问题值得探讨一下，我们需要一个程序从输入的三个数中找出最大的那个数。这个程序对任何会编程的人来说都是小菜一碟，甚至不会编程的人经过10分钟的学习也能搞定。下面是用来解决这个问题的Python代码。

```
a = int(input('a = '))
b = int(input('b = '))
c = int(input('c = '))
if a > b:
    the_max = a
else:
    the_max = b
if c > the_max:
    the_max = c
print('The max is:', the_max)
```

但是我们刚才说了，程序员都是懒惰的，很多程序员都会使用三元条件运算符来改写上面的代码。

```
a = int(input('a = '))
b = int(input('b = '))
c = int(input('c = '))
```

```
the_max = a if a > b else b  
the_max = c if c > the_max else the_max  
print('The max is:', the_max)
```

需要说明的是，Python在2.5版本以前是没有上面代码第4行和第5行中使用的三元条件运算符的，究其原因是Guido van Rossum（Python之父）认为三元条件运算符并不能帮助Python变得更加简洁，于是那些习惯了在C/C++或Java中使用三元条件运算符（在这些语言中，三元条件运算符也称为“Elvis运算符”，因为?:放在一起很像著名摇滚歌手猫王Elvis的大背头）的程序员试着用 and 和 or 运算符的短路特性来模拟出三元操作符，于是在那个年代，上面的代码是这样写的。

```
a = int(input('a = '))  
b = int(input('b = '))  
c = int(input('c = '))  
the_max = a > b and a or b  
the_max = c > the_max and c or the_max  
print('The max is:', the_max)
```

但是这种做法在某些场景下是不能成立的，且看下面的代码。

```
a = 0  
b = -100  
# 下面的代码本来预期输出a的值，结果却得到了b的值  
# 因为a的值0在进行逻辑运算时会被视为False来处理  
print(True and a or b)  
# print(a if True else b)
```

所以在Python 2.5以后引入了三元条件运算符来避免上面的风险（上面代码被注释掉的最后一句话）。那么，问题又来了，上面的代码还可以写得更简短吗？答案是肯定的。

```
a = int(input('a = '))  
b = int(input('b = '))  
c = int(input('c = '))  
print('The max is:', (a if a > b else b) if (a if a > b else b) > c else c)
```

但是，这样做真的好吗？如此复杂的表达式是不是让代码变得晦涩了很多呢？我们发现，在实际开发中很多开发者都喜欢过度的使用某种语言的特性或语法糖，于是简单的多行代码变成了复杂的单行表达式，这样做真的好吗？这个问题我也不止一次的问过自己，现在我能给出的答案是下面的代码，使用辅助函数。

```
def the_max(x, y):  
    return x if x > y else y
```

```
a = int(input('a = '))  
b = int(input('b = '))
```

```
c = int(input('c = '))
print('The max is:', the_max(the_max(a, b), c))
```

上面的代码中，我定义了一个辅助函数 `the_max` 用来找出参数传入的两个值中较大的那一个，于是下面的输出语句可以通过两次调用 `the_max` 函数来找出三个数中的最大值，现在代码的可读性是不是好了很多。用辅助函数来替代复杂的表达式真的是一个不错的选择，关键是比较大小的逻辑转移到这个辅助函数后不仅可以反复调用它，而且还可以进行级联操作。

当然，很多语言中比较大小的函数根本没有必要自己来实现（通常都是内置函数），Python也是如此。Python内置的`max`函数利用了Python对可变参数的支持，允许一次性传入多个值或者一个迭代器并找出那个最大值，所以上面讨论的问题在Python中也就是一句话的事，但是从复杂表达式到使用辅助函数简化复杂表达式这个思想是非常值得玩味的，所以分享出来跟大家做一个交流。

```
a = int(input('a = '))
b = int(input('b = '))
c = int(input('c = '))
print('The max is:', max(a, b, c))
```

Branch: master ▾

Find file Copy path

## Python-100-Days / Python惯例.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



200 lines (134 sloc) 3.82 KB

## Python惯例

“惯例”这个词指的是“习惯的做法，常规的办法，一贯的做法”，与这个词对应的英文单词叫“idiom”。由于Python跟其他很多编程语言在语法和使用上还是有比较显著的差别，因此作为一个Python开发者如果不能掌握这些惯例，就无法写出“Pythonic”的代码。下面我们总结了一些在Python开发中的惯用的代码。

1. 让代码既可以被导入又可以被执行。

```
if __name__ == '__main__':
```

2. 用下面的方式判断逻辑“真”或“假”。

```
if x:  
if not x:
```

**好的代码：**

```
name = 'jackfrued'  
fruits = ['apple', 'orange', 'grape']  
owners = {'1001': '骆昊', '1002': '王大锤'}  
if name and fruits and owners:  
    print('I love fruits!')
```

**不好的代码：**

```
name = 'jackfrued'  
fruits = ['apple', 'orange', 'grape']  
owners = {'1001': '骆昊', '1002': '王大锤'}  
if name != '' and len(fruits) > 0 and owners != {}:  
    print('I love fruits!')
```

### 3. 善于使用in运算符。

```
if x in items: # 包含  
for x in items: # 迭代
```

**好的代码：**

```
name = 'Hao LUO'  
if 'L' in name:  
    print('The name has an L in it.')
```

**不好的代码：**

```
name = 'Hao LUO'  
if name.find('L') != -1:  
    print('This name has an L in it!')
```

### 4. 不使用临时变量交换两个值。

```
a, b = b, a
```

### 5. 用序列构建字符串。

**好的代码：**

```
chars = ['j', 'a', 'c', 'k', 'f', 'r', 'u', 'e', 'd']  
name = ''.join(chars)  
print(name) # jackfrued
```

**不好的代码：**

```
chars = ['j', 'a', 'c', 'k', 'f', 'r', 'u', 'e', 'd']  
name = ''  
for char in chars:  
    name += char  
print(name) # jackfrued
```

### 6. EAFP优于LBYL。

EAFP - Easier to Ask Forgiveness than Permission.

LBYL - Look Before You Leap.

**好的代码：**

```
d = {'x': '5'}
try:
    value = int(d['x'])
    print(value)
except (KeyError, TypeError, ValueError):
    value = None
```

**不好的代码：**

```
d = {'x': '5'}
if 'x' in d and isinstance(d['x'], str) \
        and d['x'].isdigit():
    value = int(d['x'])
    print(value)
else:
    value = None
```

7. 使用enumerate进行迭代。

**好的代码：**

```
fruits = ['orange', 'grape', 'pitaya', 'blueberry']
for index, fruit in enumerate(fruits):
    print(index, ':', fruit)
```

**不好的代码：**

```
fruits = ['orange', 'grape', 'pitaya', 'blueberry']
index = 0
for fruit in fruits:
    print(index, ':', fruit)
    index += 1
```

8. 用生成式生成列表。

**好的代码：**

```
data = [7, 20, 3, 15, 11]
result = [num * 3 for num in data if num > 10]
print(result) # [60, 45, 33]
```

**不好的代码：**

```
data = [7, 20, 3, 15, 11]
result = []
for i in data:
    if i > 10:
        result.append(i * 3)
print(result) # [60, 45, 33]
```

9. 用zip组合键和值来创建字典。

**好的代码：**

```
keys = ['1001', '1002', '1003']
values = ['骆昊', '王大锤', '白元芳']
d = dict(zip(keys, values))
print(d)
```

**不好的代码：**

```
keys = ['1001', '1002', '1003']
values = ['骆昊', '王大锤', '白元芳']
d = {}
for i, key in enumerate(keys):
    d[key] = values[i]
print(d)
```

**说明**：这篇文章的内容来自于网络，有兴趣的读者可以阅读[原文](#)。

Branch: master ▾

Find file Copy path

## Python-100-Days / PEP 8风格指南.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



37 lines (26 sloc) 4.12 KB

## PEP 8风格指南

PEP是Python Enhancement Proposal的缩写，通常翻译为“Python增强提案”。每个PEP都是一份为Python社区提供的指导Python往更好的方向发展的技术文档，其中的第8号增强提案（PEP 8）是针对Python语言编订的代码风格指南。尽管我们可以在保证语法没有问题的前提下随意书写Python代码，但是在实际开发中，采用一致的风格书写出可读性强的代码是每个专业的程序员应该做到的事情，也是每个公司的编程规范中会提出的要求，这些在多人协作开发一个项目（团队开发）的时候显得尤为重要。我们可以从Python官方网站的[PEP 8链接](#)中找到该文档，下面我们将对该文档的关键部分做一个简单的总结。

### 空格的使用

1. 使用空格来表示缩进而不要用制表符（Tab）。这一点对习惯了其他编程语言的人来说简直觉得不可理喻，因为绝大多数的程序员都会用Tab来表示缩进，但是要知道Python并没有像C/C++或Java那样的用花括号来构造一个代码块的语法，在Python中分支和循环结构都使用缩进来表示哪些代码属于同一个级别，鉴于此Python代码对缩进以及缩进宽度的依赖比其他很多语言都强得多。在不同的编辑器中，Tab的宽度可能是2、4或8个字符，甚至是其他更离谱的值，用Tab来表示缩进对Python代码来说可能是一场灾难。
2. 和语法相关的每一层缩进都用4个空格来表示。
3. 每行的字符数不要超过79个字符，如果表达式因太长而占据了多行，除了首行之外的其余各行都应该在正常的缩进宽度上再加上4个空格。
4. 函数和类的定义，代码前后都要用两个空行进行分隔。
5. 在同一个类中，各个方法之间应该用一个空行进行分隔。
6. 二元运算符的左右两侧应该保留一个空格，而且只要一个空格就好。

### 标识符命名

PEP 8倡导用不同的命名风格来命名Python中不同的标识符，以便在阅读代码时能够通过标识符的名称来确定该标识符在Python中扮演了怎样的角色（在这一点上，Python自己的内置模块以及某些第三方模块都做得并不是很好）。

1. 变量、函数和属性应该使用小写字母来拼写，如果有多个单词就使用下划线进行连接。
2. 类中受保护的实例属性，应该以一个下划线开头。
3. 类中私有的实例属性，应该以两个下划线开头。
4. 类和异常的命名，应该每个单词首字母大写。
5. 模块级别的常量，应该采用全大写字母，如果有多个单词就用下划线进行连接。
6. 类的实例方法，应该把第一个参数命名为 `self` 以表示对象自身。
7. 类的类方法，应该把第一个参数命名为 `cls` 以表示该类自身。

## 表达式和语句

在Python之禅（可以使用 `import this` 查看）中有这么一句名言：“There should be one--and preferably only one --obvious way to do it.”，翻译成中文是“做一件事应该有而且最好只有一种确切的做法”，这句话传达的思想在PEP 8中也是无处不在的。

1. 采用内联形式的否定词，而不要把否定词放在整个表达式的前面。例如 `if a is not b` 就比 `if not a is b` 更容易让人理解。
2. 不要用检查长度的方式来判断字符串、列表等是否为 `None` 或者没有元素，应该用 `if not x` 这样的写法来检查它。
3. 就算 `if` 分支、`for` 循环、`except` 异常捕获等中只有一行代码，也不要将代码和 `if`、`for`、`except` 等写在一起，分开写才会让代码更清晰。
4. `import` 语句总是放在文件开头的地方。
5. 引入模块的时候，`from math import sqrt` 比 `import math` 更好。
6. 如果有多个 `import` 语句，应该将其分为三部分，从上到下分别是Python标准模块、第三方模块和自定义模块，每个部分内部应该按照模块名称的字母表顺序来排列。

Branch: master ▾

Find file Copy path

## Python-100-Days / 玩转PyCharm.md

Fetching contributors...

Cannot retrieve contributors at this time.

Raw Blame History



93 lines (47 sloc) 5.6 KB

## 玩转PyCharm

PyCharm是由JetBrains公司开发的提供给Python专业的开发者的一个集成开发环境，它最大的优点是能够大大提升Python开发者的工作效率，为开发者集成了很多用起来非常顺手的功能，包括代码调试、高亮语法、代码跳转、智能提示、自动补全、单元测试、版本控制等等。此外，PyCharm还提供了对一些高级功能的支持，包括支持基于Django框架的Web开发、。

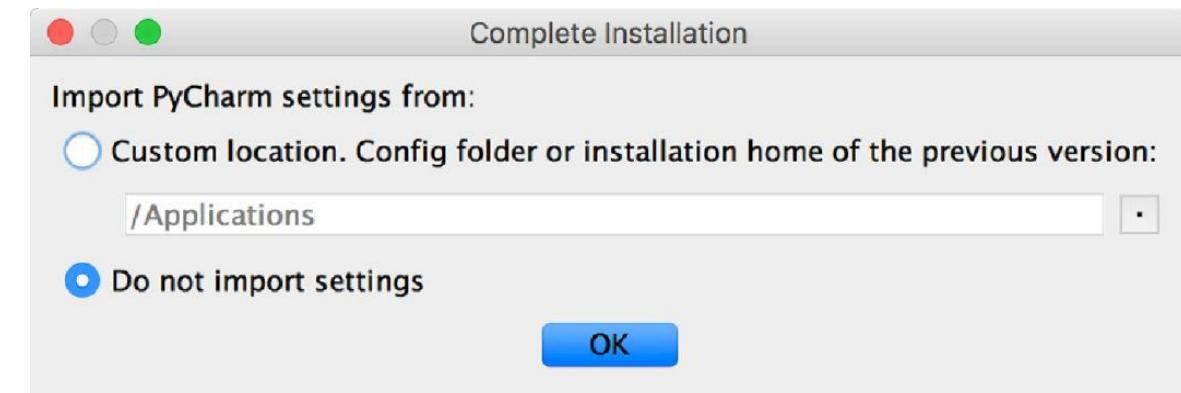
### PyCharm的安装

可以在[JetBrains公司的官方网站](#)找到PyCharm的[下载链接](#)，有两个可供下载的版本一个  
是社区版一个专业版，社区版在[Apache许可证](#)下发布，专业版在专用许可证下发布  
(需要购买授权下载后可试用30天)，其拥有许多额外功能。安装PyCharm需要有  
JRE ( Java运行时环境 ) 的支持，如果没有可以在安装过程中选择在线下载安装。

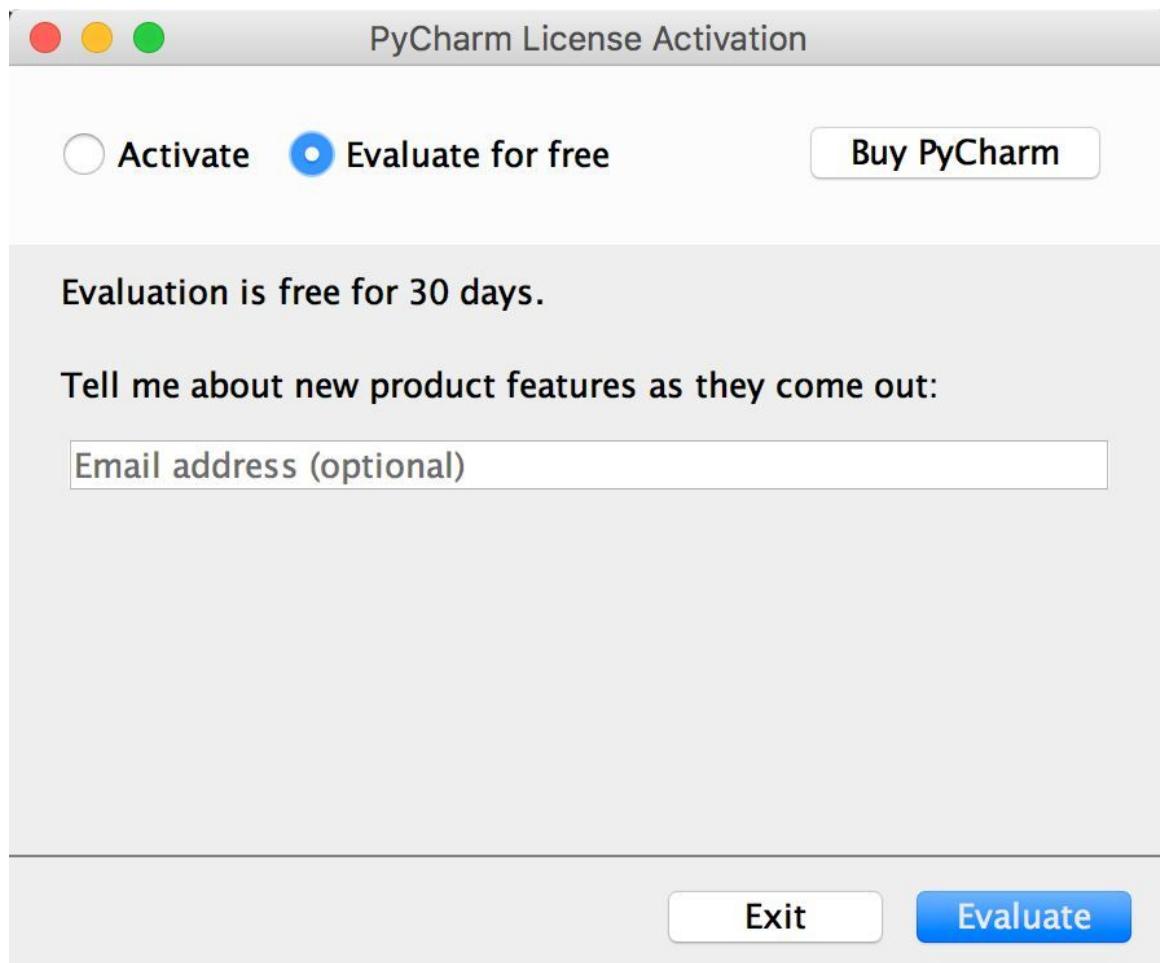
说明：如果你是一名学生，希望购买PyCharm来使用，可以看看[教育优惠官方申请指南](#)。

### 首次使用的设置

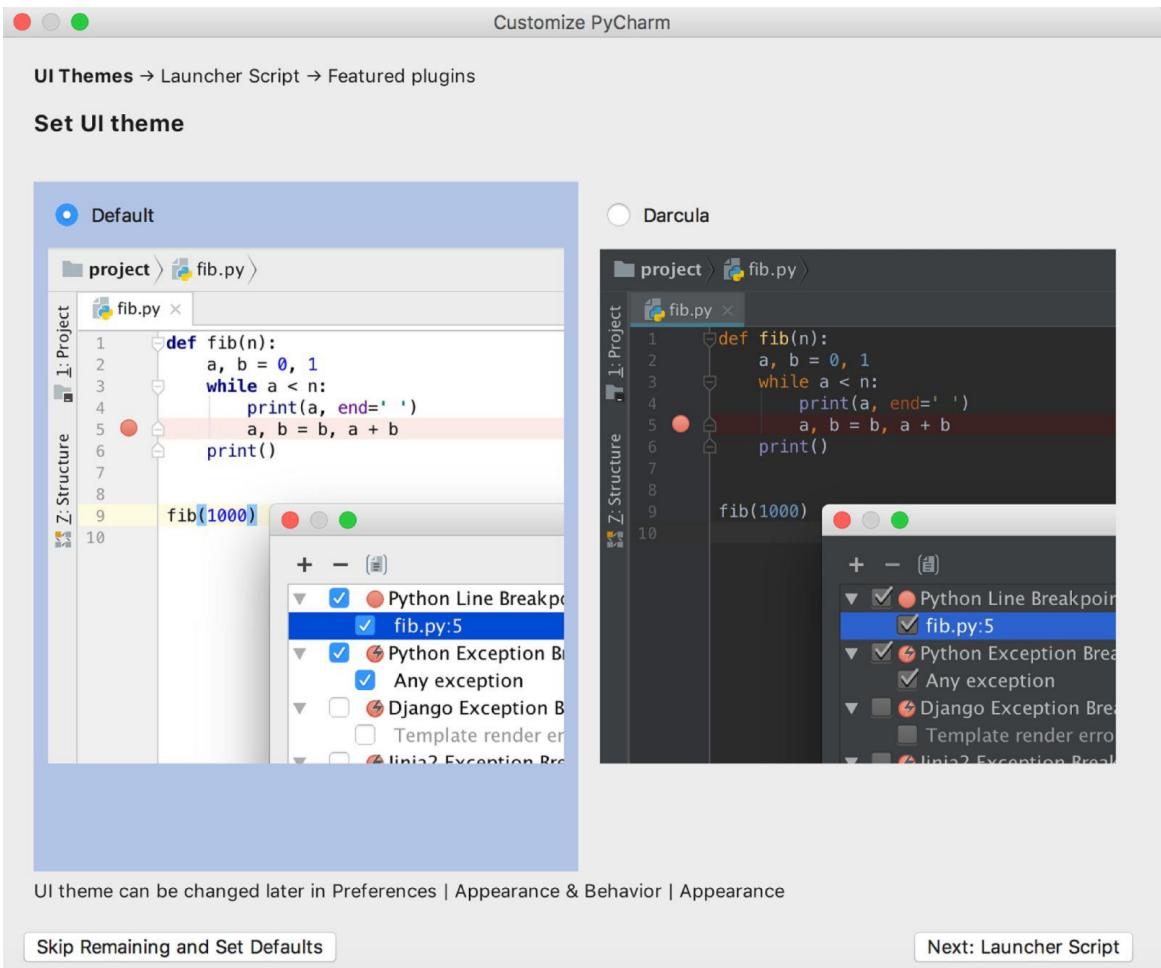
第一次使用PyCharm时，会有一个导入设置的向导，如果之前没有使用PyCharm或者没  
有保存过设置的就直接选择“Do not import settings”进入下一步即可。



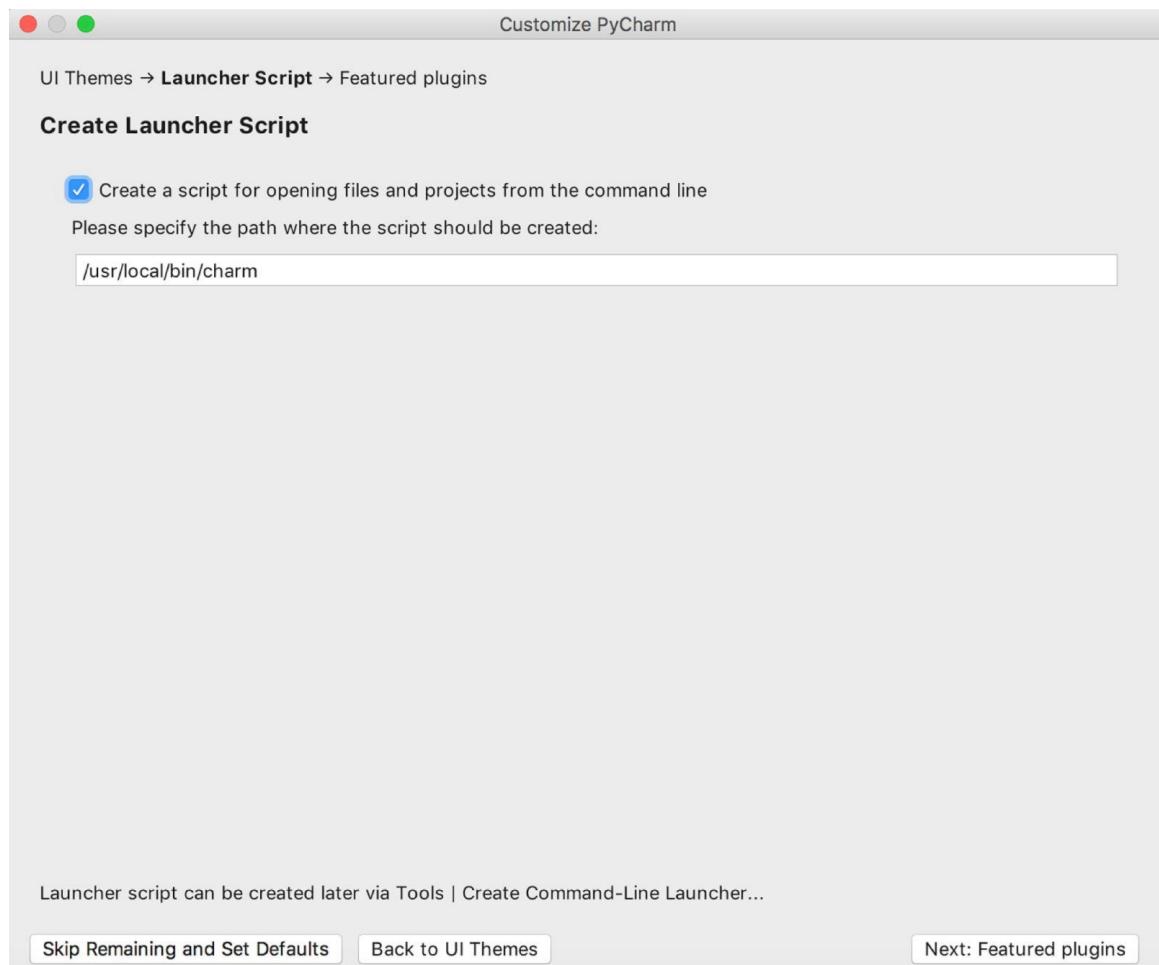
专业版的PyCharm是需要激活的，强烈建议为优秀的软件支付费用，如果不做商业用途，我们可以暂时选择试用30天或者使用社区版的PyCharm。



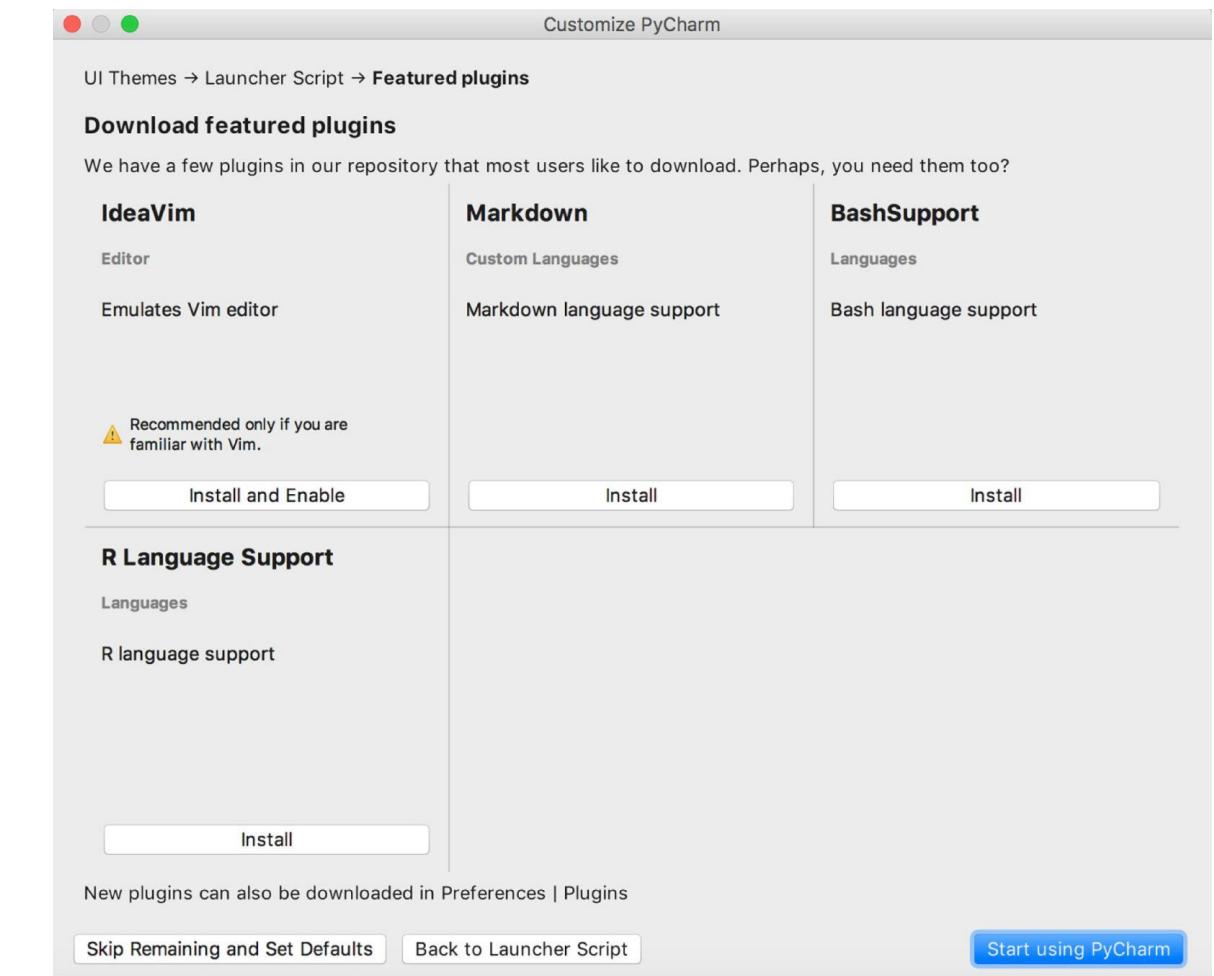
接下来是选择UI主题，这个可以根据个人喜好进行选择。



再接下来是创建可以在终端（命令行）中使用PyCharm项目的启动脚本，当然也可以直接跳过这一步。



然后可以选择需要安装哪些插件，我们可以暂时什么都不安装等需要的时候再来决定。

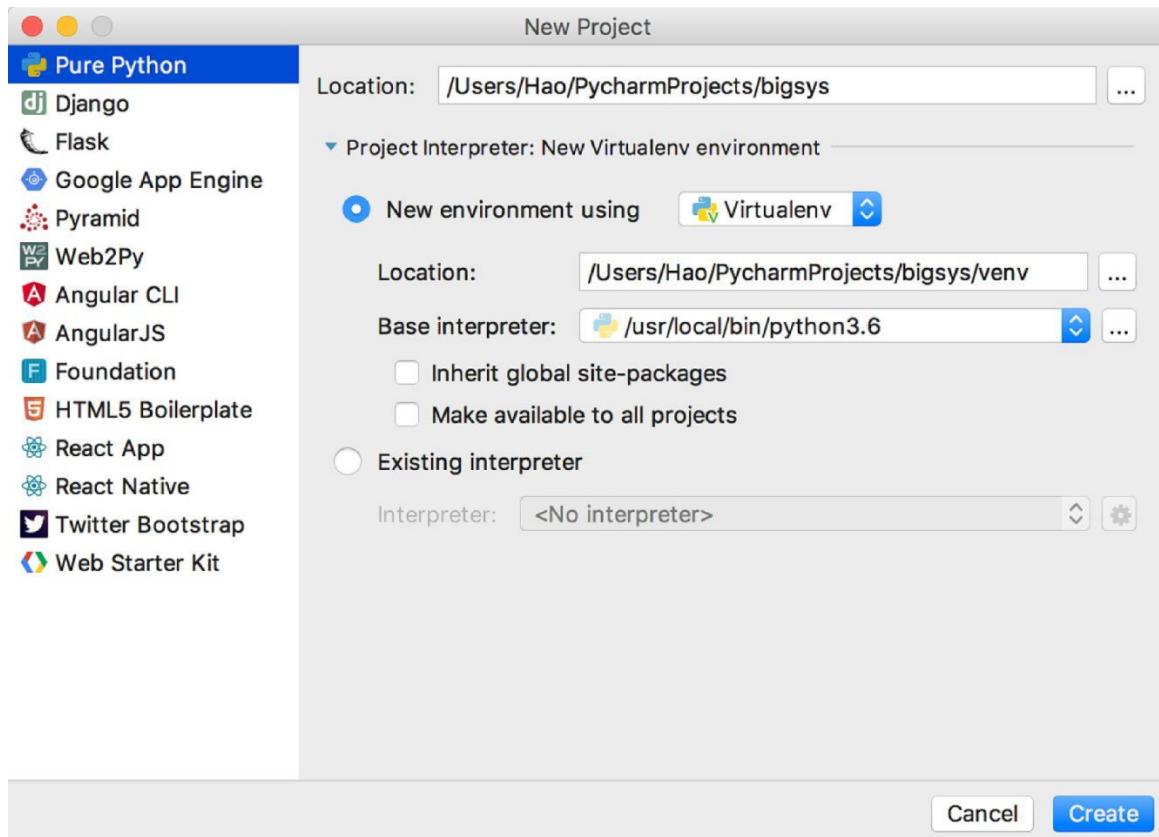


## 用PyCharm创建项目

点击上图中的“Start using PyCharm”按钮就可以开始使用PyCharm啦，首先来到的是一个欢迎页，在欢迎页上我们可以选择“创建新项目”、“打开已有项目”和“从版本控制系统中检出项目”。

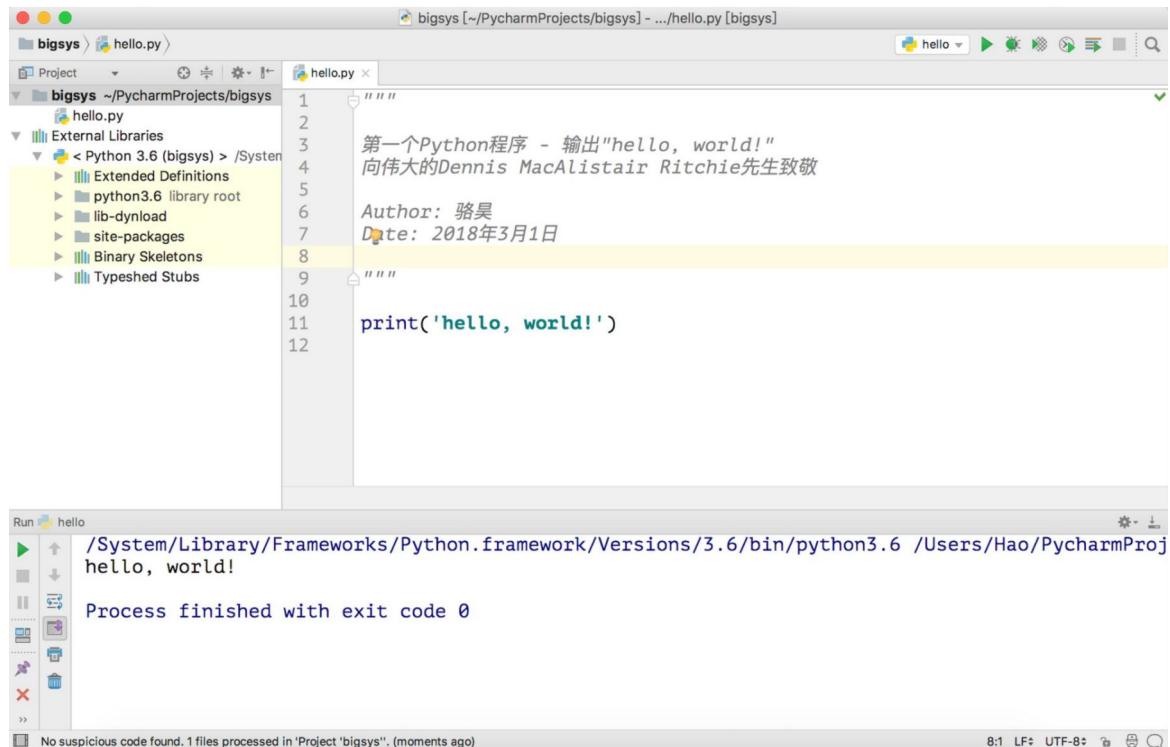


如果选择了“Create New Project”来创建新项目就会打一个创建项目的向导页。



在如上图所示的界面中，我们可以选择创建项目的模板，包括了纯Python项目、基于各种不同框架的Web项目、Web前端项目、跨平台项目等各种不同的项目模板。如果选择Python的项目，那么有一个非常重要的设定是选择“New environment...”（创建新的虚拟环境）还是使用“Existing Interpreter”（已经存在的解释器）。前者肯定是更好的选择，因为新的虚拟环境不会对系统环境变量中配置的Python环境造成影响，简单举个例子就是你在虚拟环境下安装或者更新了任何三方库，它并不会对系统原有的Python解释器造成任何的影响，但代价是需要额外的存储空间来建立这个虚拟环境。

项目创建完成后就可以开始新建各种文件来书写Python代码了。



The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure for 'bigsy' with files 'hello.py' and 'External Libraries'. The main editor window shows the code for 'hello.py':

```
'''  
第一个Python程序 - 输出"hello, world!"  
向伟大的Dennis MacAlistair Ritchie先生致敬  
Author: 骆昊  
Date: 2018年3月1日  
'''  
  
print('hello, world!')
```

Below the editor is the 'Run' tool window, which shows the output of running the code:

```
/System/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/Hao/PycharmProj  
hello, world!  
Process finished with exit code 0
```

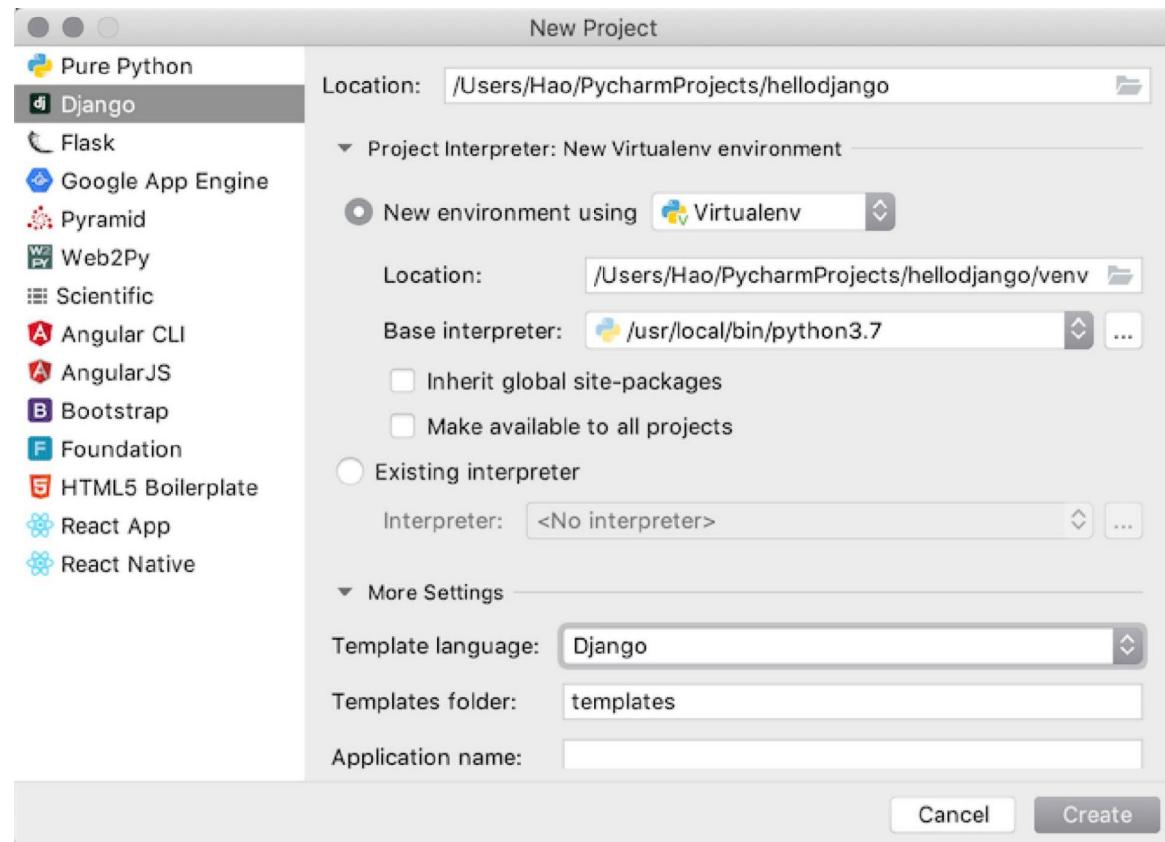
At the bottom, a status bar indicates: No suspicious code found. 1 files processed in 'Project "bigsy"', and shows file encoding as UTF-8.

在工作窗口的右键菜单中可以找到“Run ...”和“Debug ...”菜单项，通过这两个菜单项我们就可以运行和调试我们的代码啦。建议关注一下菜单栏中的“Code”、“Refactor”和“Tools”菜单，这里面为编写Python代码提供了很多有用的帮助。

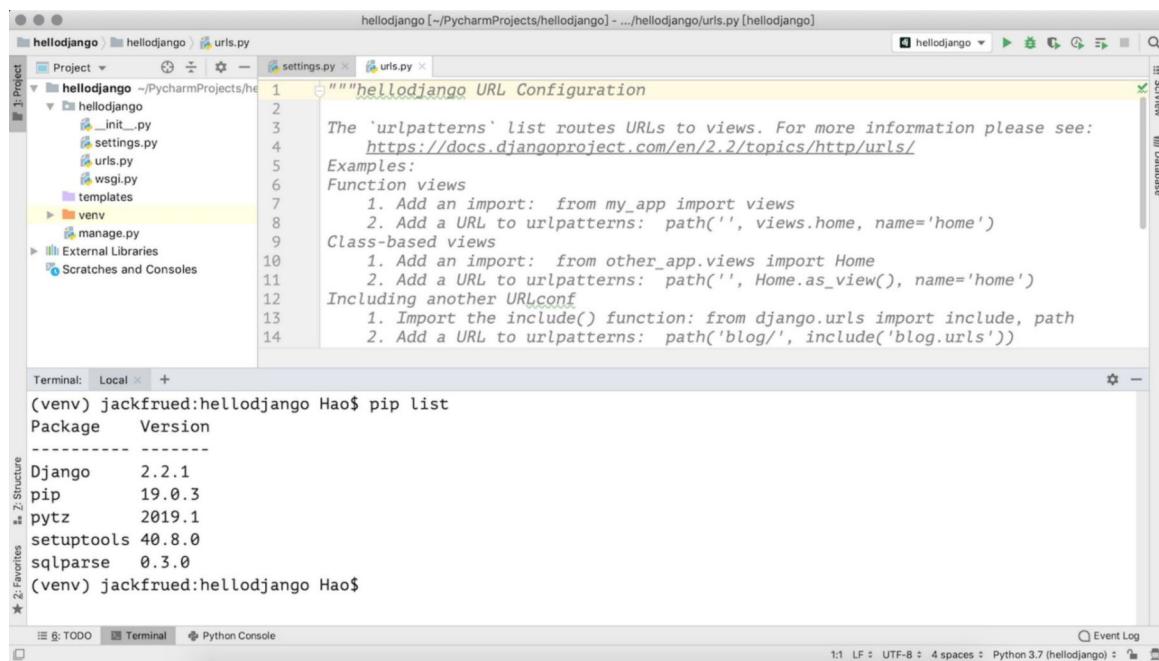
## 创建Django项目

### 专业版

PyCharm专业版提供了对Django、Flask、Google App Engine、web2py等Python Web框架以及SQL、UML、前端语言和框架、远程调试、虚拟化部署等功能的支持，如果使用PyCharm专业版，在创建项目时可以直接选择创建Django项目并设置模板语言以及放置模板页的文件夹。

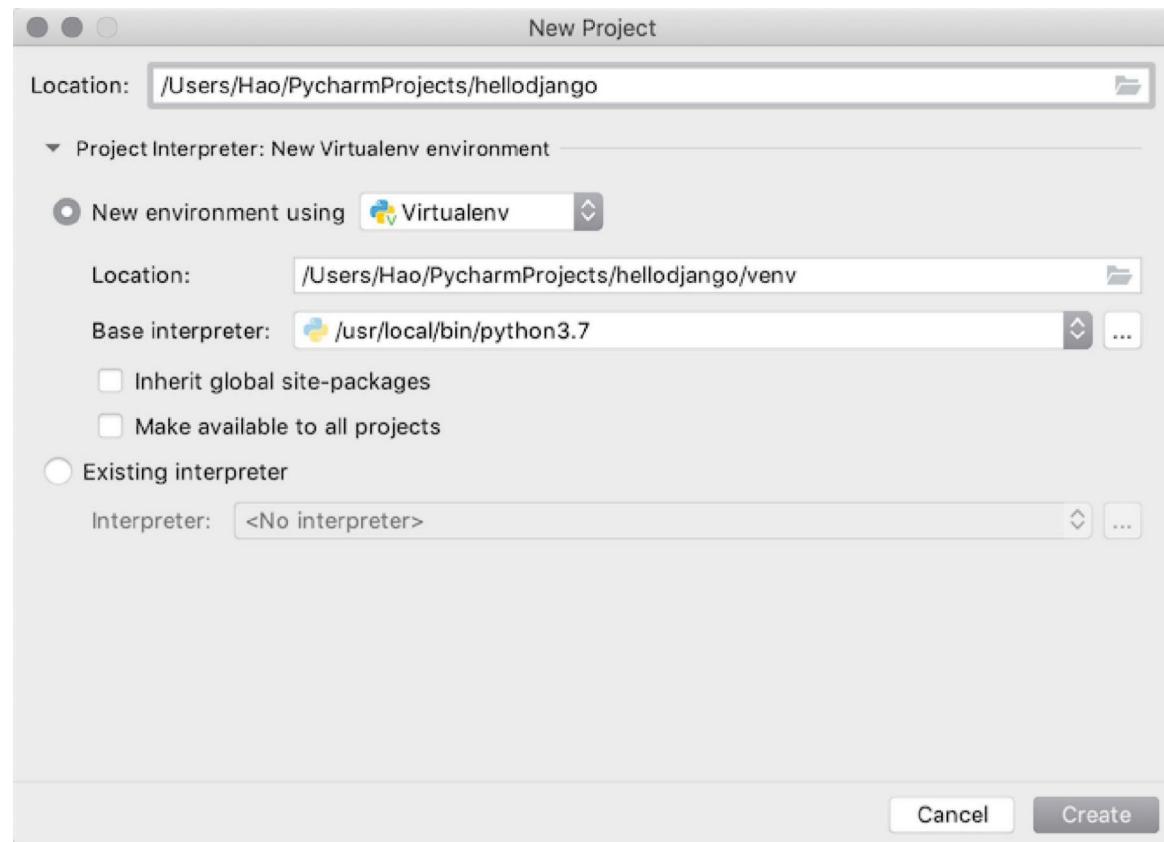


创建好项目之后，打开终端输入 pip list 命令，可以看到项目所需的依赖项已经安装好了，而且可以直接点击屏幕右上方的运行或调试按钮来直接运行Django项目。



## 社区版

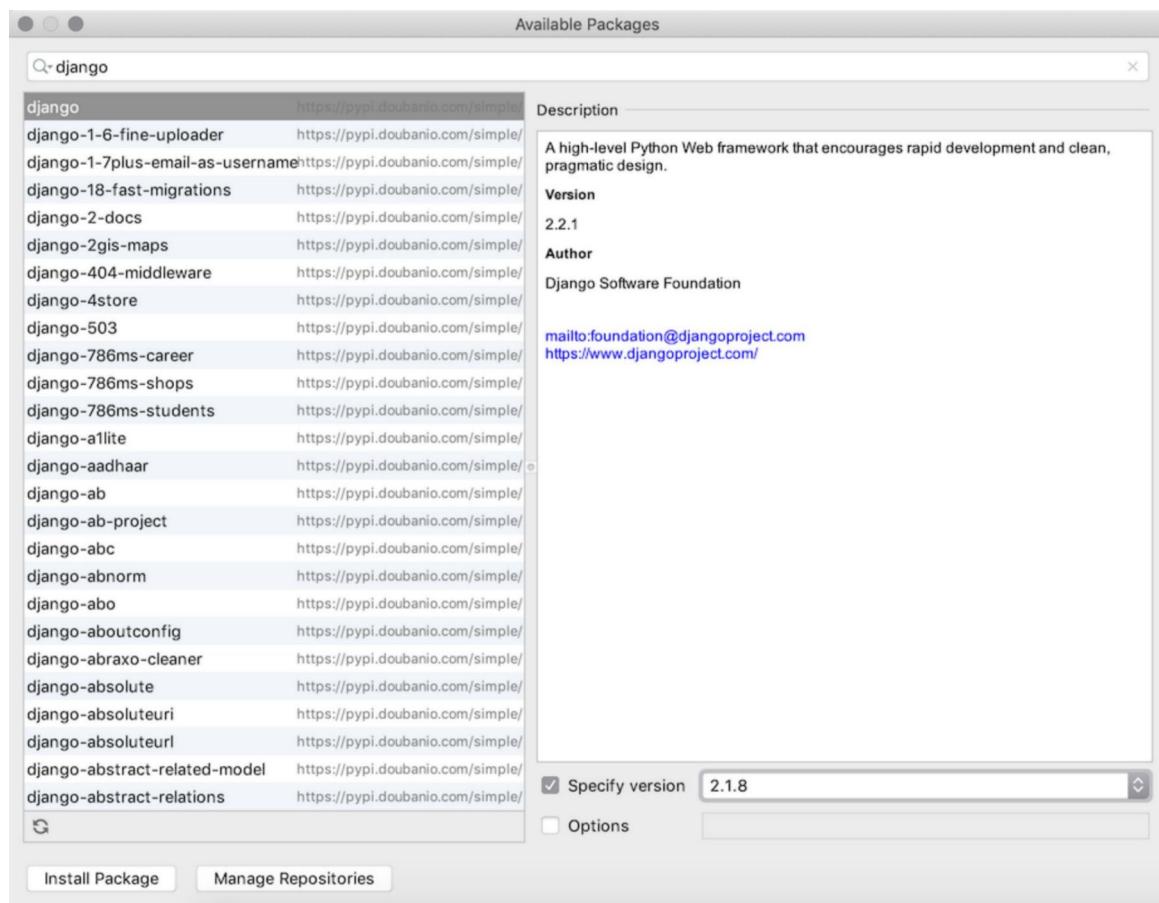
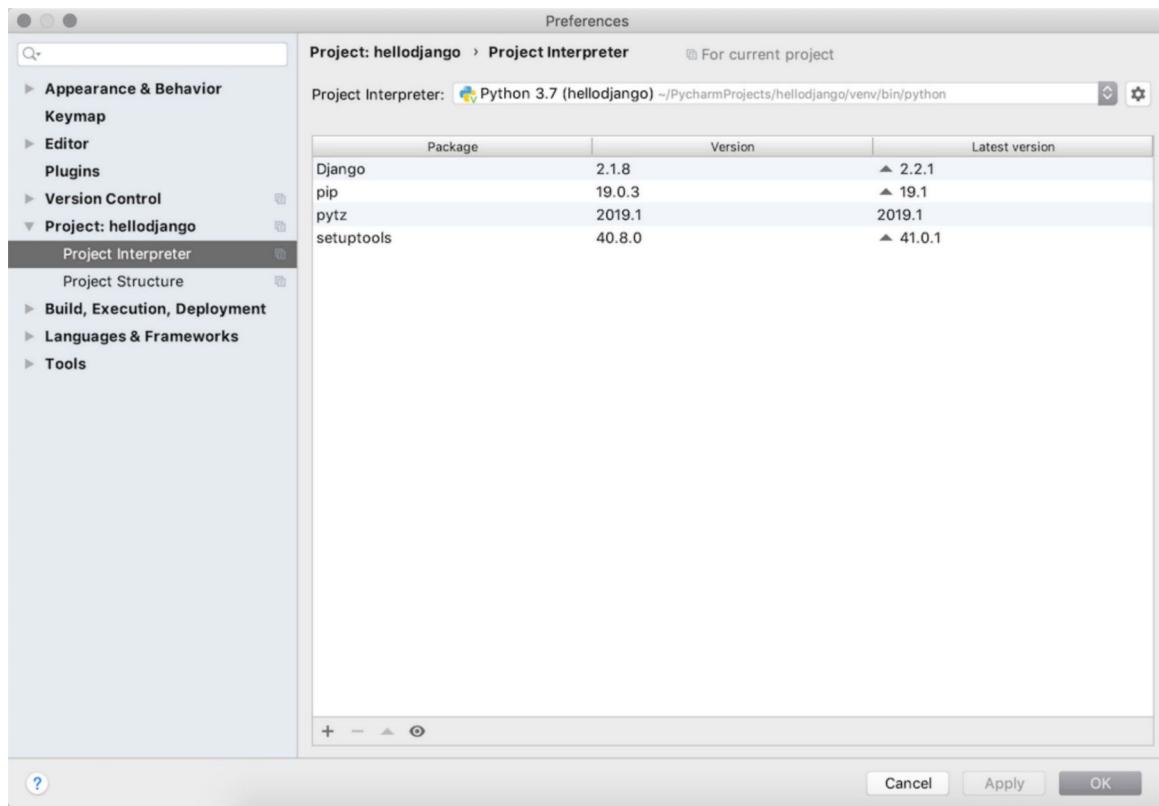
PyCharm社区版只能创建Python项目，如果项目中需要Django的支持，可以自行安装依赖库并创建Django项目。



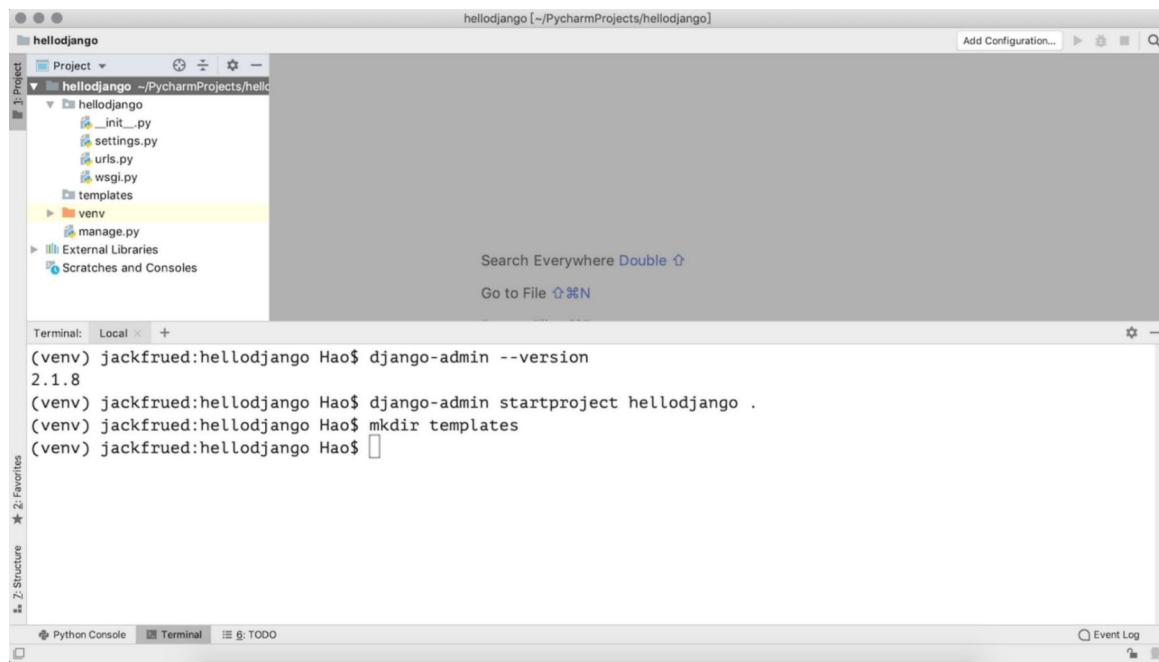
创建好Python项目之后，可以打开屏幕下方的终端（Terminal），并通过 `pip install` 安装Django项目的依赖项。



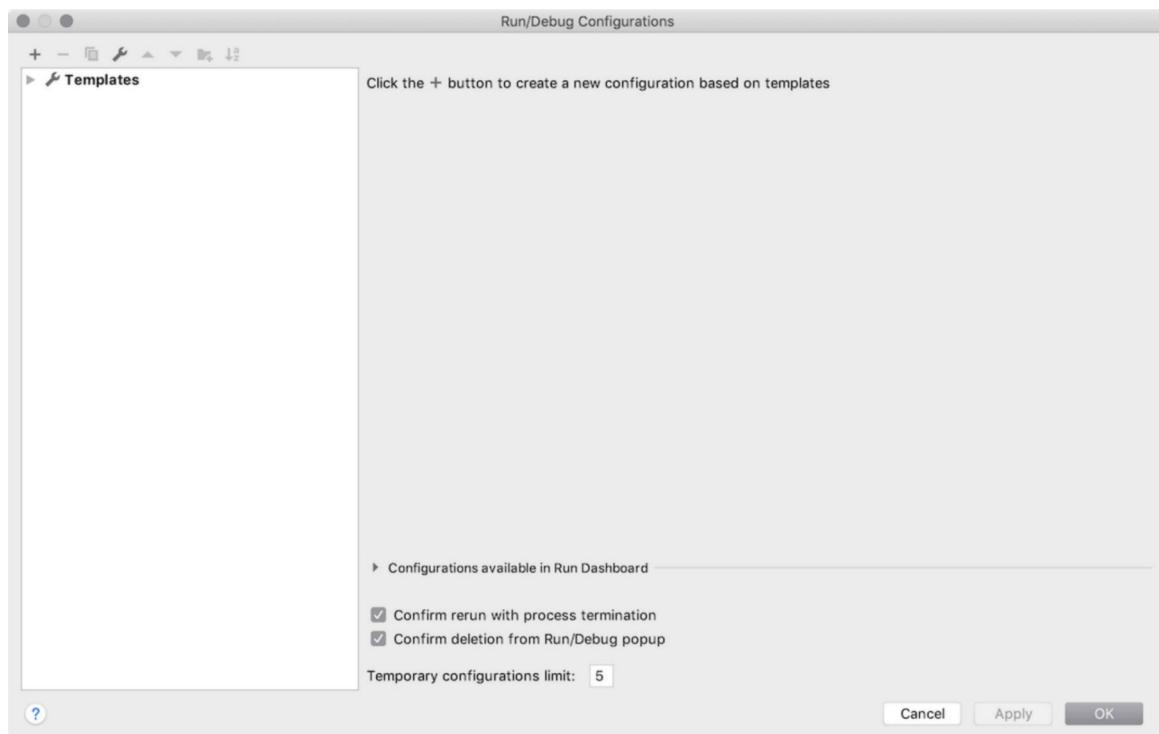
当然也可以在项目的设置菜单中找到解释器配置，并选择要添加的依赖项。



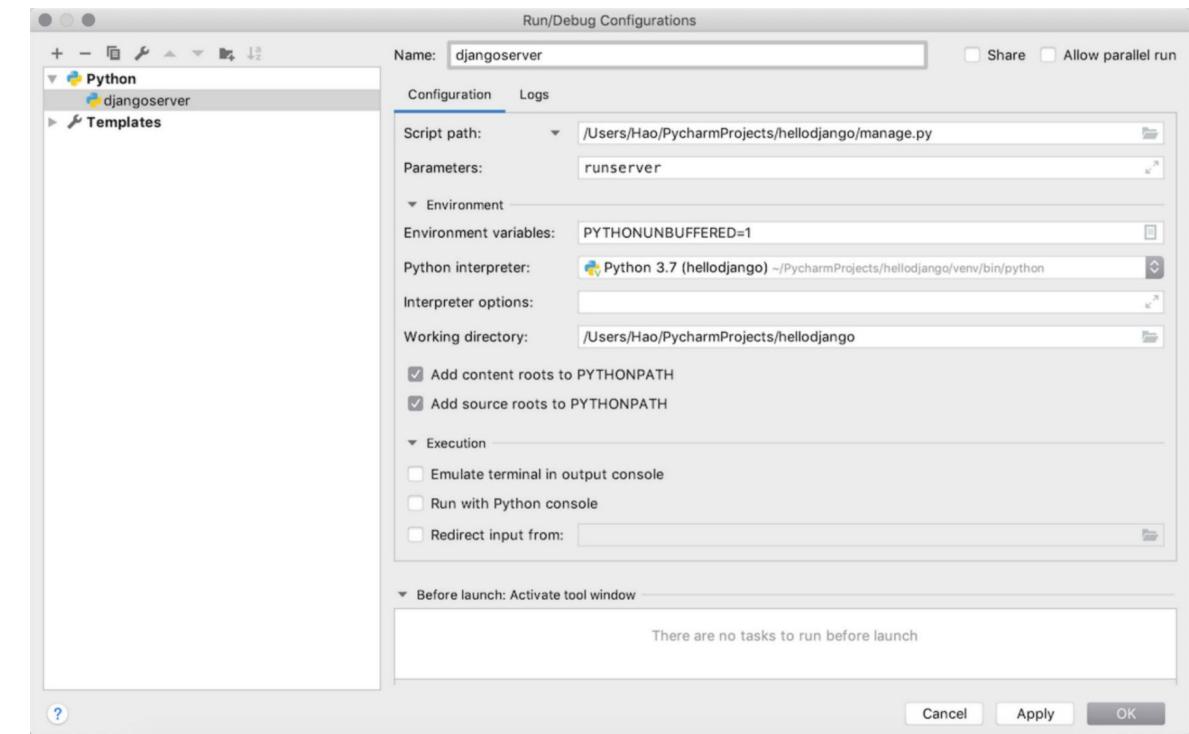
接下来可以在终端中输入 `django-admin startproject` 指令来创建项目。



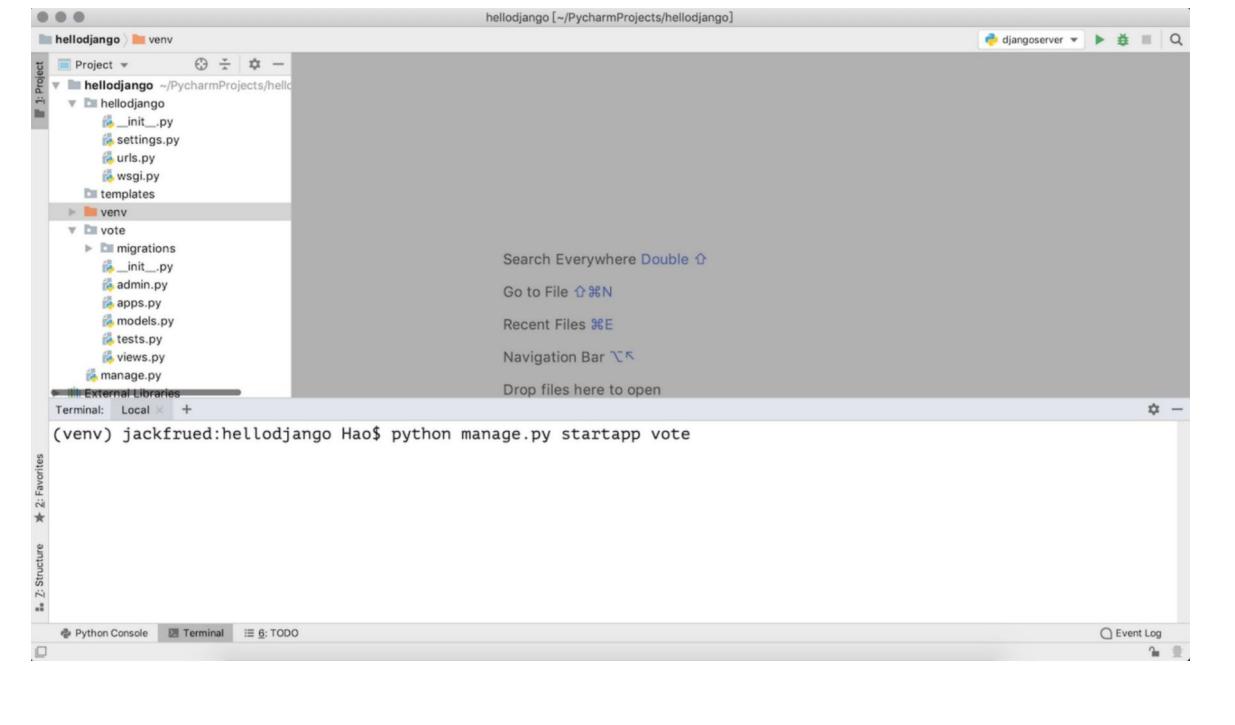
如果要运行项目，可以在终端中输入 `python manage.py runserver` 启动测试服务器。当然，也可以点击屏幕右上方的“Add Configuration”按钮，进入如下所示的配置界面，并点击窗口左上角的“+”来添加一个运行配置。



在配置窗口的右侧，指定要执行的脚本路径（Django项目的manage.py文件的位置）和运行参数（runserver）。



注意到窗口的右上角了吗？现在可以点击运行或调试按钮来启动测试服务器运行项目了。



Branch: master ▾

[Find file](#) [Copy path](#)

## Python-100-Days / Python参考书籍.md

Fetching contributors...

Cannot retrieve contributors at this time.

[Raw](#) [Blame](#) [History](#)



55 lines (43 sloc) 2.49 KB

# Python参考书籍

## 入门读物

1. 《Python基础教程》 ( *Beginning Python From Novice to Professional* )
2. 《Python学习手册》 ( *Learning Python* )
3. 《Python编程》 ( *Programming Python* )
4. 《Python Cookbook》
5. 《Python程序设计》 ( *Python Programming: An Introduction to Computer Science* )
6. 《Modern Python Cookbook》

## 进阶读物

1. 《Python核心编程》 ( *Core Python Applications Programming* )
2. 《流畅的Python》 ( *Fluent Python* )
3. 《Effective Python : 编写高质量Python代码的59个有效方法》 ( *Effective Python 59 Specific Ways to Write Better Python* )
4. 《Python设计模式》 ( *Learning Python Design Patterns* )
5. 《Python高级编程》 ( *Expert Python Programming* )
6. 《Python性能分析与优化》 ( *Mastering Python High Performance* )

## Web框架

1. 《Django基础教程》 ( *Tango with Django* )
2. 《轻量级Django》 ( *Lightweight Django* )
3. 《Python Web开发 : 测试驱动方法》 ( *Test-Driven Development with Python* )
4. 《Web Development with Django Cookbook》

5. 《Test-Driven Development with Django》
6. 《Django Project Blueprints》
7. 《Flask Web开发：基于Python的Web应用开发实战》 ( *Flask Web Development: Developing Web Applications with Python* )
8. 《深入理解Flask》 ( *Mastering Flask* )

## 爬虫开发

1. 《用Python写网络爬虫》 ( *Web Scraping with Python* )
2. 《精通Python爬虫框架Scrapy》 ( *Learning Scrapy* )
3. 《Python网络数据采集》 ( *Web Scraping with Python* )
4. 《Python爬虫开发与项目实战》
5. 《Python 3网络爬虫开发实战》

## 数据分析

1. 《利用Python进行数据分析》 ( *Python for Data Analysis* )
2. 《Python数据科学手册》 ( *Python Data Science Handbook* )
3. 《Python金融大数据分析》 ( *Python for Finance* )
4. 《Python数据可视化编程实战》 ( *Python Data Visualization Cookbook* )
5. 《Python数据处理》 ( *Data Wrangling with Python* )

## 机器学习

1. 《Python机器学习基础教程》 ( *Introduction to Machine Learning with Python* )
2. 《Python机器学习实践指南》 ( *Python Machine Learning Blueprints* )
3. 《Python Machine Learning Case Studies》
4. 《Python机器学习实践：测试驱动的开发方法》 ( *Thoughtful Machine Learning with Python A Test Driven Approach* )
5. 《Python机器学习经典实例》 ( *Python Machine Learning Cookbook* )
6. 《TensorFlow：实战Google深度学习框架》