Netkiller Python 手札

陈景峯 著

python

Netkiller Python 手札

目录

- 自述
  - 1. 写给读者
  - 2. 作者简介
  - 3. 如何获得文档
  - 4. 打赏（Donations）
  - 5. 联系方式
- I. Python 入门
  - 1. Python 入门
    - 1. 如何快速学习 Python 语言
      - 1.1. 学习编程的目的是什么?
      - 1.2. 很多公司是怎么死的?
      - 1.3. 学习定位问题?
      - 1.4. 小白怎么入门呢?
      - 1.5. 从入门到放弃
      - 1.6. 为什么学不会?
      - 1.7. 如何快速高效的学习一门语言?
      - 1.8. 碎片化学习
    - 2. install
      - 2.1. venv
      - 2.2. Docker 安装
      - 2.3. dnf 安装 python3.11
      - 2.4. 编译安装 Python3.x
      - 2.5. Ubunut 13.04 环境安装python3
      - 2.6. pypy - fast alternative implementation of Python - PyPy interpreter
        - Ubuntu 环境安装
        - CentOS 环境安装
      - 2.7. Eric Python IDE
      - 2.8. python to exe
        - pyinstaller

Linux
py2exe
2.9. Python2.x
编译安装
Ubuntu 安装
3. Python Package Index (PyPI)
3.1. 什么是 PyPI
3.2. 准备目录
3.3. 创建LICENSE文件
3.4. 项目描述文件
3.5. 库代码
3.6. setuptools 构建脚本
3.7. 构建包
3.8. 查看包
3.9. 上传包
3.10. 安装包
3.11. 使用包
3.12. 卸载包
3.13. Built distribution
3.14. 免密登录
3.15. 其他 Python 包管理工具
distutils
distribute
3.16. pip - A tool for installing and managing Python packages
安装 pip
使用 easy_install 安装 pip
Ubuntu 安装 pip
Mac
查看版本
升级 pip 命令
查询包
安装包
指定版本号
安装 wheel文件
卸载包

升级包
显示包详细信息
列出已经安装的包
批量安装库
导出已安装的包
兼容性检查
从 PyPI 下载 whl 文件到本地硬盘
切换 pip 镜像
4. Python 模块
4.1. 同级目录
4.2. 一级目录
4.3. 二级子目录
4.4. 子目录调用上级目录
4.5. 导入类
5. 数据类型
5.1. type 数据类型检测
5.2. 字符串
String function
str.find()
str.find()
format 方法
格式化字典输出
Convert str to bytes in python
% 字符串格式化输出
f-string 格式化字符串
去除中文
去除标点符号
去除数字
5.3. float 浮点数值
5.4. Array
遍历数字
split / join
5.5. 日期和时间
当前时间
生成时间
日期格式化

- 字符串转日期
- 日期转字符串
- 日期运算
  - 天数差
  - 月/周 首尾计算
- 日期范围计算
- 日期排序

5.6. bytes 类型
- bytes to string
- BOM头
- replace
- pack/unpack

6. 数据结构
- 6.1. List
- 6.2. Set
- 6.3. Dict 字典
  - 随机选择字典的 key 和 value
  - 字典合并
  - 取最大值

7. Class
- 7.1. __init__ 构造方法
- 7.2. inner class(内嵌类)
  - 内嵌 Class 继承

8. 正则处理字符串
- 8.1. 正则替换
- 8.2. match
- 8.3. 正则查找
- 8.4. 正则匹配后返回字典

9. 循环

10. Input/Output
- 10.1. Standard
  - Standard Input
  - Standard Output
- 10.2. File

11. Pipe
- 11.1. stdin

        11.2. forkpty
        11.3. Popen
        11.4. socketpair
2. Library
    1. 序列化
    2. 队列
    3. 随机数
        3.1. 随机选择列表
        3.2. 指定随机数范围
        3.3. 指定随机数范围(整数)
        3.4. 指定随机数范围(小数)
        3.5. 打乱列表顺序
    4. Python 多线程
        4.1. 创建线程
        4.2. threading 高级线程接口
        4.3. Lock 线程锁
        4.4. Queue 队列
    5. syslog
        5.1. udp client
        5.2. udp server
    6. Socket
        6.1. UDP
            UDP Server
            UDP Clinet
    7. subprocess
        7.1. check_output
    8. YAML
        8.1. 严格按段落展示 ｜、｜+、｜-
        8.2. >、>+、>-
        8.3. PyYAML
            解决 ｜ 问题
        8.4. ruamel.yaml
            解决 ｜ 问题
            LiteralScalarString, PreservedScalarString
    9. Daemon
    10. python-memcached

11. Pyro - Pyro is short for PYthon Remote Objects
12. Python Imaging Library
13. getopt – Command line option parsing
14. syslog
    14.1. udp client
    14.2. udp server
15. python-subversion
16. SimpleHTTPServer
17. fuse-python.x86_64 : Python bindings for FUSE - filesystem in userspace
18. Network
    18.1. gevent - A coroutine-based network library for Python
19. Python-spdylay - Spdylay Python Extension Module
20. mechanize
21. Dominate
22. dbm Key-Value 数据库
23. keyboard
    23.1. 读取键盘值
    23.2. 功能键
24. httpx
    24.1. 安装 https
    24.2. 操作演示
    24.3. Restful CRUD 操作
    24.4. HTTP 2
    24.5. 异步请求
    24.6. 日志输出
25. 日志彩色输出

3. 终端环境开发
    1. ANSI Color
        1.1. ansicolors
        1.2. termcolor
        1.3. Colorama
            初始化操作
            常用格式

2. 进度条
    2.1. progress
        条形进度条（Bars）
        方块进度条（ChargingBar）
        填充方块进度条（FillingSquaresBar）
        填充圆圈进度条（FillingCirclesBar）
        使用 Incremental 展示内存使用率
    2.2. tqdm
        tqdm 命令
        演示
        设置进度条长度
        设置任务数量
        多进程进度监控
    2.3. alive-progress
3. texttable - module for creating simple ASCII tables
    3.1. 对齐设置
    3.2. 设置表格风格
    3.3. 自定义风格
    3.4. 设置列数据类型
    3.5. 彩色表格
4. prompt_toolkit
    4.1. 安装
5. Simple Terminal Menu
6. picotui
7. TUI
    7.1. Console
    7.2. urwid
    7.3. pycdk
    7.4. python-newt - A NEWT module for Python
4. Python 调试技巧
    1. 显示代码所在文件行
II. Frameworks
5. Uvicorn
    1. 代码启动
    2. 命令行启动
    3. 日志

- 4. FastAPI
  - 4.1. Post Request
    - From 数据
    - Json 数据转为 dict
    - Data 原始数据
    - POST 接收 JSON 数据
  - 4.2. api_route
  - 4.3. slowapi 流向控制
  - 4.4. 异步执行
  - 4.5. 缓存
    - 缓存 Json 数据结构
    - 自定义 key
  - 4.6. HTTP Auth
  - 4.7. SSE
  - 4.8. Fief 认证框架
- 6. Robot Framework 测试框架
- 7. Web framework
  - 1. Django
  - 2. Pylons
    - 2.1. Getting Started with Pylons
      - Installing
      - Debian/Ubuntu
    - 2.2. config/routing.py
    - 2.3. mako template
      - include
      - inherit
  - 3. Pyramid
    - 3.1. Getting Started
      - virtualenv - create virtual Python instances
      - Hello world
      - MongoDB
    - 3.2. Creating a Pyramid Project
      - mongodb
- 9. Sqlalchemy
  - 1. 安装 Sqlalchemy
  - 2. 链接测试

3. 创建表
4. Session
5. 模型定义
    5.1. 定义字段
        server_default
        给表加注释
        修改记录的时候触发更新
    5.2. 外键
        ON DELETE 删除外键约束
        ON UPDATE 更新外键约束
6. 增删改
7. 查询
8. 标签
9. 统计数量
10. 排序
11. 查询数据是否存在
    11.1. 返回 exists SQL 语句
    11.2. exists()
    11.3. query.exists()
12.
    12.1. count
    12.2. min/max
    12.3. 平均值/求和
    12.4.

III. Python 数据分析

10. Crawler
    1. Requests
11. Scrapy - Python web scraping and crawling framework
    1. 安装 scrapy 开发环境
        1.1. Mac
        1.2. Ubuntu
        1.3. 使用 pip 安装 scrapy
        1.4. 测试 scrapy
    2. scrapy 命令
        2.1.
        2.2. 新建 spider

        2.3. 列出可用的 spiders
        2.4. 运行 spider
    3. Scrapy Shell
        3.1. response
            当前URL地址
            status HTTP 状态
            text 正文
            css
                获取 html 属性
            xpath
            headers
    4. 爬虫项目
        4.1. 创建项目
        4.2. Spider
            翻页操作
            采集内容保存到文件
        4.3. settings.py 爬虫配置文件
            忽略 robots.txt 规则
        4.4. Item
        4.5. Pipeline
    5. 下载图片
        5.1. 配置 settings.py
        5.2. 修改 pipelines.py 文件
        5.3. 编辑 items.py
        5.4. Spider 爬虫文件
    6. xpath
        6.1. 逻辑运算符
            and
            or
        6.2. function
            text()
            contains()
12. Pandas - Python Data Analysis Library
    1. 安装 pandas
    2. 数据输入与输出
        2.1. Pandas 处理 HTML

HTML 表格处理
安装依赖包
read_html 参数详解
从文本变量中提取数据
从文件获取表格数据
从网址获取表格数据
处理多个表格
获取指定属性的表格
结合 Xpath 使用
指定编码
使用 Dominate 生成 HTML
2.2. Excel 处理
安装依赖库
创建 Excel 文档
读取 Excel 文件
读取指定列
工作表
显示所有工作表
打开工作表
合并工作表
打开工作表，指定返回列数据
跳过不需要的数据
数据操作
打印头部/尾部数据
打印列标题
打印行
描述数据
修改 Excel 数据
新增行/列
数据筛选
数据排序
Excel 设置项
大数据写入Excel问题
2.3. Pandas 读写 CSV 文件
将数据保存到CSV文件
写入表头列名

分隔符
格式化
指定列输出
留行索引
替换空值
2.4. Pandas SQL
建立数据库链接
sqlite3
SQLAlchemy
DataFrame数据写入到数据库
分批写入数据
3. 数据帧(DataFrame)
3.1. 什么是 DataFrame
3.2. 如何创建 DataFrame
3.3. 行与列操作 index/columns
方法一，指定 index / columes 名称
方法二，指定 index / columes 名称
获取 columns 名称
插入列
迭代行
3.4. 数据访问
head() 与 tail()
iloc 访问数据
loc 访问数据
Axis（轴）
3.5. 添加操作
添加列
追加数据
3.6. 删除操作
3.7. 插入数据
插入一列数据
3.8. 替换操作
3.9. 筛选
3.10. 排序
3.11. 分类聚合
3.12. 数据去重

3.13. 数据格式化
日期格式化
浮点格式化
小数位数
3.14. 迭代输出
4. 序列
4.1. 创建 Series 对象
4.2. Series 运算
4.3. Series 常用方法
head() /tail()
isnull() / notnull()
去重复数据
5. 数据可视化
5.1. 演示代码
折线图
条形图
直方图
区域图
饼形图
XY散点图
箱形图
核密度估计图（Kernel Density Estimation, KDE）
5.2. 中文支持
查看系统支持的字体
设置字体
中文演示代码
5.3. 开启网格
5.4. 坐标轴
轴标签旋转

标题/X标签/Y标签
设置X/Y坐标范围
5.5. 边框设置
5.6. plot 设置

            隐藏图例
        5.7. 保存为图片
        5.8. matplotlib 绘图风格
    6. Pandas 实用函数
        6.1. 日期范围
        6.2.
    7. FAQ
        7.1. xlrd.biffh.XLRDError: Excel xlsx file; not supported
        7.2. Missing optional dependency 'xlrd'
13. 股票
    1. easyquotation - 快速获取新浪/腾讯的全市场行情
        1.1. 安装
        1.2. 演示
    2. akshare
14. 数据可视化
    1. matplotlib
        1.1. 直方图
        1.2. 显示中文
    2. pyecharts
IV. 人工智能 AI
15. AI 相关
    1. tokenizers
        1.1. Normalization
        1.2. Pre-Tokenization
    2. transformers
        2.1. 安装 transformers
        2.2. 加载本地模型
        2.3. 自动下载模型
        2.4. 编码
        2.5. 计算向量
        2.6. FAQ
            隐藏警告
16. OCR
    1. EasyOCR
        1.1. 安装 EasyOCR

        1.2. 操作演示
        1.3. 命令行运行
        1.4. 函数
            Reader()
            readtext() 函数
        1.5. urllib.error.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:833)>
    2. Tesseract
        2.1. 安装 Tesseract
        2.2. 演示 Tesseract
        2.3.
17. 语音处理
    1. TTS(Text To Speech) 文本转语音
        1.1. 安装 pyttsx3
            Linux
        1.2. 演示
        1.3. 方法详解
            say() 方法
            save_to_file()
            调整人声类型
            调整语速
            调整声量
            查看语音引擎
        1.4. 例子
    2. STT(Speech To Text) 语音转文本
        2.1. SpeechRecognition
            安装
            查看麦克风列表
            PocketSphinx 文件转文本
            Google Cloud Speech API
            IBM Speech to Text
        2.2. DeepSpeech
    3. Baidu AipSpeech
    4. AI文字转语音模型Bark
    5. Automatic Speech Recognition

        5.1. kaldi
        5.2. OpenAI Whisper
18. 视频
    1. 摄像头
    2. MoviePy
        2.1. 合成视频
        2.2. 提取视频中的音频
        2.3. 加字幕
        2.4. 音量大小调节
        2.5. 音频合成剪辑
        2.6. 视频中设置背景音乐
        2.7. 画面截图
        2.8.
19. 人脸识别
    1. 安装
    2. 命令行工具
        2.1. face_detection - 在单张图片或一个图片文件夹中定位人脸位置
        2.2. face_detection - 在单张图片或一个图片文件夹中定位人脸位置
    3. 定位头像位置
    4. 人脸比较
    5. 摄像头识别人脸
20. 自然语言处理
    1. 全文索引（full-text index）
    2. 人机对话
    3. 情感分析
    4. 常用的 Python 分词库
    5. 结巴中文分词
        5.1. 分词演示
        5.2. 日志设置
        5.3. 返回 generator
        5.4. 返回 list
        5.5. 精准模式与全模式比较
        5.6. 精准模式与搜索引擎模式比较
        5.7. 词性标注

5.8. 词典管理
添加/删除词语
用户词典
自定义词库
5.9. 抽取文本标签
提取标签
基于 TextRank 算法的关键词抽取
5.10. 返回词语在原文的起止位置
6. wordcloud
6.1. wordcloud_cli
6.2. WordCloud 对象配置参数
6.3. 与分词共用
6.4. 遮罩图
7. Transformers 自然语言处理
8. 汉字转拼音
8.1.
8.2. pypinyin
21. OpenAI
1. ChatGPT
1.1. gpt-3.5-turbo
1.2. 流式输出
2. Embedding
22. LangChain
1. 拆分文档
2. 拆分文档
2.1. 拆分文本
2.2. 拆分代码
2.3. 拆分 Markdown 文档
2.4. 按token拆分
3. ChatGPT
4. 相似度搜索
23. 自动化运维
1. 日志中心
1.1. 什么是日志中心
1.2. 工作原理
1.3. 安装

1.4. 命令
日志采集端
日志接收端
1.5. 操作演示
从标准输出采集日志
发送日志文件
接收日志并保存到文件
发送动态日志文件
2. Python 开发防火墙
2.1. 我为什么要开发一个防火墙软件?
2.2. 安装 Python 防火墙
2.3. 切换防火墙规则
2.4. 规则库
3. 监视文件系统
3.1. watchdog
Observer
创建/删除/修改/移动
多事件绑定
自动备份程序
3.2. pyinotify
4. 容器
4.1. 在 Docker 容器中运行 Python 项目
4.2. 在 kubernetes 中部署项目
24. 办公自动化
1. Python 处理 PDF 文件
1.1. Word 转 PDF
1.2. 提取 PDF 文件中的文字和表格
安装 pdfplumber
获取PDF文档信息
获取PDF总页数
查看PDF页面信息
提取文本内容
提取pdf中的表格数据
保存数据到 Excel
1.3. PyPDF2
2. Word 文字处理

- 2.1. 安装
- 2.2. 创建空白文档
- 2.3. 添加标题
- 2.4. 添加段落
- 2.5. 列表
- 2.6. 表格
- 2.7. 添加图片
- 2.8. 强制分页
- 2.9. 样式
  - 对齐
  - 首行缩进
  - 段落间距
  - 行间距
  - 粗体，斜体
  - 字体大小
  - 查看段落样式
  - 文档样式
  - 自动分页设置
  - 样式演示
- 2.10. 演示例子
  - 官方演示例子
  - 完整的演示例子
- 2.11. 另存操作
- 2.12. 读取 Word 文档
  - 风格筛选
- 2.13. Word 模版合并
  - 安装 docx-mailmerge

3. Python 处理 Excel
- 3.1. openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files
  - 创建空文档
  - 工作表
    - 默认工作表
    - 创建新工作表
    - 遍历工作表
    - 删除工作表

- 单元格
  - 单元格填充数据
  - 获取工作表行数和列数
  - 读取单元格
  - 修改单元格
  - 单元格合并/取消合并
  - 单元格格式化
  - 使用公式
  - 插入图片
  - 行高和列宽
  - 行列隐藏
- 样式设置
  - sheet选项卡背景色
  - 字体
  - 单元格背景色
  - 设置单元格样式
  - 综合应用
- 工具
  - 数字列转标签

3.2. xlrd/xlwt/xlutils

- 读 Excel
- 写 Excel
  - 添加工作表
  - 合并单元格
  - 运行公式
  - 超链接
  - 样式设置
    - 单元格对齐
    - 字体和颜色
    - 设置边框
    - 设置背景色
    - 单元格格式化
- xlutils

3.3. xlwings

25. OpenCV

1. 安装 OpenCV

    2. 显示图片
    3. 摄像头捕捉图像
    4. imread()
26. 图形开发
    1. SVG 图形库
        1.1. 安装
        1.2. 绘制多边形
        1.3. SVG 事件
    2. PIL
    3. 二维码
        3.1. qrcode
            设置颜色
            qr - script to create QR codes at the command line
        3.2. MyQR
        3.3. 从图片识别二维码
        3.4. 从摄像头识别二维码
    4. graphviz
        4.1. 安装 graphviz 环境
        4.2. 例子
27. 3rdparty toolkit
    1. ZeroRPC
    2. 表情符号 emoji
    3. Markdown
        3.1. 安装
29. 实用代码
    1. 随机生成姓名
    2. 心知天气
30. FAQ
    1. ImportError: No module named 'zlib'
    2. UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb2 in position 679: invalid start byte
    3. ERROR: Could not install packages due to an EnvironmentError: [Errno 28] No space left on device

范例清单

2.1. memcached.py
2.2. SimpleHTTPServer
7.1. __init__.py
12.1. Pandas 打开工作表的四种方法
19.1. 找出图片中头像
29.1. 随机生成姓名
29.2. 词库生成姓名

Netkiller Python 手札

《Netkiller Python 手札》视频教程（2023版）

ISBN#

Mr. Neo Chan, 陈景峯(BG7NYT)

中国广东省深圳市望海路半岛城邦三期
518067
+86 13113668890

<netkiller@msn.com>

$Id: book.xml 640 2013-07-18 03:27:47Z netkiller $

电子书最近一次更新于 2023-11-19 11:30:46

版权 © 2008-2023 Netkiller(Neo Chan). All rights reserved.

版权声明

转载请与作者联系，转载时请务必标明文章原始出处和作者信息及本声明。

http://www.netkiller.cn
http://netkiller.github.io
http://netkiller.sourceforge.net
微信公众号: netkiller
微信：13113668890 请注明“读者”
QQ：13721218 请注明“读者”
QQ群：128659835 请注明“读者”
知乎专栏 | 多维度架构

$Date: 2013-07-18 11:27:47 +0800 (Thu, 18 Jul 2013) $

我的系列文档

编程语言

Netkiller Architect 手札	Netkiller Developer 手札	Netkiller Java 手札	Netkiller Spring 手札	Netkiller PHP 手札	Netkiller Python 手札
Netkiller Testing 手札	Netkiller Cryptography 手札	Netkiller Perl 手札	Netkiller Docbook 手札	Netkiller Project 手札	Netkiller Database 手札

自述

《Netkiller 系列 手札》是一套免费系列电子书，netkiller 是 nickname 从1999 开使用至今，“手札” 是札记，手册的含义。

2003年之前我还是以文章形式在BBS上发表各类技术文章，后来发现文章不够系统，便尝试写长篇技术文章加上章节目录等等。随着内容增加，不断修订，开始发布第一版，第二版.....

IT知识变化非常快，而且具有时效性，这样发布非常混乱，经常有读者发现第一版例子已经过时，但他不知道我已经发布第二版。

我便有一种想法，始终维护一个文档，不断更新，使他保持较新的版本不过时。

第一部电子书是《PostgreSQL 实用实例参考》开始我使用 Microsoft Office Word 慢慢随着文档尺寸增加 Word 开始表现出力不从心。

我看到PostgreSQL 中文手册使用SGML编写文档，便开始学习 Docbook SGML。使用Docbook写的第一部电子书是《Netkiller Postfix Integrated Solution》这是Netkiller 系列手札的原型。

至于“手札”一词的来历，是因为我爱好摄影，经常去一个台湾摄影网站，名字就叫“摄影家手札”。

由于硬盘损坏数据丢失 《Netkiller Postfix Integrated Solution》 的 SGML文件已经不存在; Docbook SGML存在很多缺陷 UTF-8支持不好，转而使用Docbook XML.

目前技术书籍的价格一路飙升，动则￥80，￥100，少则￥50，￥60. 技术书籍有时效性，随着技术的革新或淘汰，大批书记成为废纸垃圾。并且这些书技术内容雷同，相互抄袭，质量越来越差，甚至里面给出的例子错误百出，只能购买影印版，或者翻译的版本。

在这种背景下我便萌生了自己写书的想法，资料主要来源是我的笔记与例子。我并不想出版，只为分享，所有我制作了基于CC License 发行的系列电子书。

本书注重例子，少理论（捞干货），只要你对着例子一步一步操作，就会成功，会让你有成就感并能坚持学下去，因为很多人遇到障碍就会放弃，其实我就是这种人，只要让他看到希望，就能坚持下去。

1. 写给读者

为什么写这篇文章

有很多想法,工作中也用不到所以未能实现，所以想写出来,和大家分享.有一点写一点,写得也不好,只要能看懂就行,就当学习笔记了.

开始零零碎碎写过一些文档，也向维基百科供过稿，但维基经常被ZF封锁，后来发现sf.net可以提供主机存放文档，便做了迁移。并开始了我的写作生涯。

这篇文档是作者20年来对工作的总结,是作者一点一滴的积累起来的，有些笔记已经丢失，所以并不完整。

因为工作太忙整理比较缓慢。目前的工作涉及面比较窄所以新文档比较少。

我现在花在技术上的时间越来越少，兴趣转向摄影，无线电。也想写写摄影方面的心得体会。

写作动力:

曾经在网上看到外国开源界对中国的评价，中国人对开源索取无度，但贡献却微乎其微.这句话一直记在我心中，发誓要为中国开源事业做我仅有的一点微薄贡献

另外写文档也是知识积累，还可以增加在圈内的影响力.

人跟动物的不同,就是人类可以把自己学习的经验教给下一代人.下一代在上一代的基础上再创新,不断积累才有今天.

所以我把自己的经验写出来,可以让经验传承

没有内容的章节:

目前我自己一人维护所有文档，写作时间有限，当我发现一个好主题就会加入到文档中，待我有时间再完善章节，所以你会发现很多章节是空无内容的.

文档目前几乎是流水帐试的写作，维护量很大，先将就着看吧.

我想到哪写到哪,你会发现文章没一个中心,今天这里写点,明天跳过本章写其它的.

文中例子绝对多,对喜欢复制然后粘贴朋友很有用,不用动手写,也省时间.

理论的东西,网上大把,我这里就不写了,需要可以去网上查.

我爱写错别字,还有一些是打错的,如果发现请指正.

文中大部分试验是在Debian/Ubuntu/Redhat AS上完成.

写给读者

至读者:

我不知道什么时候，我不再更新文档或者退出IT行业去从事其他工作，我必须给这些文档找一个归宿，让他能持续更新下去。

我想捐赠给某些基金会继续运转，或者建立一个团队维护它。

我用了20年时间坚持不停地写作，持续更新，才有今天你看到的《Netkiller 手扎》系列文档，在中国能坚持20年，同时没有任何收益的技术类文档，是非常不容易的。

有很多时候想放弃，看到外国读者的支持与国内社区的影响，我坚持了下来。

中国开源事业需要各位参与，不要成为局外人，不要让外国人说：中国对开源索取无度，贡献却微乎其微。

我们参与内核的开发还比较遥远，但是进个人能力，写一些文档还是可能的。

系列文档

下面是我多年积累下来的经验总结，整理成文档供大家参考:

Netkiller Architect 手札

Netkiller Developer 手札

Netkiller PHP 手札

Netkiller Python 手札

Netkiller Testing 手札

Netkiller Cryptography 手札
Netkiller Linux 手札
Netkiller FreeBSD 手札
Netkiller Shell 手札
Netkiller Security 手札
Netkiller Web 手札
Netkiller Monitoring 手札
Netkiller Storage 手札
Netkiller Mail 手札
Netkiller Docbook 手札
Netkiller Version 手札
Netkiller Database 手札
Netkiller PostgreSQL 手札
Netkiller MySQL 手札
Netkiller NoSQL 手札
Netkiller LDAP 手札
Netkiller Network 手札
Netkiller Cisco IOS 手札
Netkiller H3C 手札
Netkiller Multimedia 手札
Netkiller Management 手札
Netkiller Spring 手札
Netkiller Perl 手札
Netkiller Amateur Radio 手札

2. 作者简介

陈景峯 (ㄔㄣˊ ㄐㄧㄥˇ ㄈㄥ)

Nickname： netkiller | English name: Neo chen | Nippon name: ちんけいほう (音訳) | Korean name: 천징봉 | Thailand name: ภูมิภาพภูเขา | Vietnam: Tr`ần Cảnh Phong

Callsign: BG7NYT | QTH: ZONE CQ24 ITU44 ShenZhen, China

程序猿，攻城狮，挨踢民工, Full Stack Developer, UNIX like Evangelist, 业余无线电爱好者（呼号：BG7NYT）,户外运动，山地骑行以及摄影爱好者。

《Netkiller 系列 手札》的作者

成长阶段

1981年1月19日(庚申年腊月十四)出生于黑龙江省青冈县建设乡双富大队第一小队

1989年9岁随父母迁居至黑龙江省伊春市，悲剧的天朝教育，不知道那门子归定，转学必须降一级，我本应该上一年级，但体制让我上学前班，那年多都10岁了

1995年小学毕业，体制规定借读要交3000两银子(我曾想过不升初中)，亲戚单位分楼告别平房，楼里没有地方放东西，把2麻袋书送给我，无意中发现一本电脑书BASIC语言，我竟然看懂了，对于电脑知识追求一发而不可收，后面顶零花钱，压岁钱主要用来买电脑书《MSDOS 6.22》《新编Unix实用大全》《跟我学Foxbase》。。。。。。

1996年第一次接触UNIX操作系统，BSD UNIX, Microsoft Xinux(盖茨亲自写的微软Unix，知道的人不多)

1997年自学Turbo C语言，苦于没有电脑，后来学校建了微机室才第一次使用QBASIC(DOS 6.22 自带命令)，那个年代只能通过软盘拷贝转播，Trubo C编译器始终没有搞到，

1997年第一次上Internet网速只有9600Bps, 当时全国兴起各种信息港域名格式是www.xxxx.info.net, 访问的第一个网站是NASA下载了很多火星探路者拍回的照片，还有“淞沪”sohu的前身

1998~2000年在哈尔滨学习计算机，充足的上机时间，但老师让我们练打字（明伦五笔/WT）打字不超过80个/每分钟还要强化训练，不过这个给我的键盘功夫打了好底。

1999年学校的电脑终于安装了光驱，在一张工具盘上终于找到了Turbo C, Borland C++与Quick Basic编译器，当时对VGA图形编程非常感兴趣，通过INT33中断控制鼠标，使用绘图函数模仿windows界面。还有操作 UCDOS 中文字库，绘制矢量与点阵字体。

2000年沉迷于Windows NT与Back Office各种技术，神马主域控制器，DHCP，WINS，IIS，域名服务器，Exchange邮件服务器，MS Proxy, NetMeeting...以及ASP+MS SQL开发；用56K猫下载了一张LINUX。ISO镜像，安装后我兴奋的24小时没有睡觉。

职业生涯

2001 年来深圳进城打工,成为一名外来务工者. 在一个4人公司做PHP开发，当时PHP的版本是2.0, 开始使用Linux Redhat 6.2.当时很多门户网站都是用FreeBSD,但很难搞到安装盘，在网易社区认识了一个网友,从广州给我寄了一张光盘，FreeBSD 3.2

2002 年我发现不能埋头苦干,还要学会"做人".后辗转广州工作了半年，考了一个Cisco CCNA认证。回到深圳重新开始，在车公庙找到一家工作做Java开发

2003 年这年最惨,公司拖欠工资16000元,打过两次官司2005才付清.

2004 年开始加入分布式计算团队,目前成绩，工作仍然是Java开发并且开始使用PostgreSQL数据库。

2004-10月开始玩户外和摄影

2005-6月成为中国无线电运动协会会员,呼号BG7NYT,进了一部Yaesu FT-60R手台。公司的需要转回PHP与MySQL，相隔几年发现PHP进步很大。在前台展现方面无人能敌，于是便前台使用PHP，后台采用Java开发。

2006 年单身生活了这么多年,终于找到归宿. 工作更多是研究PHP各种框架原理

2007 物价上涨,金融危机，休息了4个月（其实是找不到工作），关外很难上439.460中继，搞了一台Yaesu FT-7800.

2008 终于找到英文学习方法， 《Netkiller Developer 手札》，《Netkiller Document 手札》

2008-8-8 08:08:08 结婚,后全家迁居湖南省常德市

2009《Netkiller Database 手札》,2009-6-13学车，年底拿到C1驾照

2010 对电子打击乐产生兴趣，计划学习爵士鼓。由于我对Linux热爱，我轻松的接管了公司的运维部，然后开发运维两把抓。我印象最深刻的是公司一次上架10个机柜，我们用买服务器纸箱的钱改善伙食。我将40多台服务器安装BOINC做压力测试，获得了中国第二的名次。

2011 平凡的一年，户外运动停止，电台很少开，中继很少上，摄影主要是拍女儿与家人，年末买了一辆山地车

2012 对油笔画产生了兴趣，活动基本是骑行银湖山绿道，

2013 开始学习民谣吉他，同时对电吉他也极有兴趣；最终都放弃了。这一年深圳开始推数字中继2013-7-6日入手Motorola

MOTOTRBO XIR P8668，Netkiller 系列手札从Sourceforge向Github迁移；年底对MYSQL UDF，Engine与PHP扩展开发产生很浓的兴趣，拾起遗忘10+年的C，写了几个mysql扩展（图片处理，fifo管道与ZeroMQ），10月份入Toyota Rezi 2.5V并写了一篇《攻城狮的苦逼选车经历》

2014-9-8 在淘宝上买了一架电钢琴 Casio Privia PX-5S pro 开始陪女儿学习钢琴，由于这家钢琴是合成器电钢，里面有打击乐，我有对键盘鼓产生了兴趣。

2014-10-2号罗浮山两日游，对中国道教文化与音乐产生了兴趣，10月5号用了半天时间学会了简谱。10月8号入Canon 5D Mark III + Canon Speedlite 600EX-RT香港过关被查。

2014-12-20号对乐谱制作产生兴趣（https://github.com/SheetMusic/Piano），给女儿做了几首钢琴伴奏曲，MuseScore制谱然后生成MIDI与WAV文件。

2015-09-01 晚饭后拿起爵士鼓基础教程尝试在Casio Privia PX-5S pro演练，经过反复琢磨加上之前学钢琴的乐理知识，终于在02号晚上，打出了简单的基本节奏，迈出了第一步。

2016 对弓箭（复合弓）产生兴趣，无奈天朝法律法规不让玩。每周游泳轻松1500米无压力，年底入 xbox one s 和 Yaesu FT-2DR, 同时开始关注功放音响这块

2017 7月9号入 Yamaha RX-V581 功放一台，连接Xbox打游戏爽翻了，入Kindle电子书，计划学习蝶泳，果断放弃运维和开发知识体系转攻区块链。

2018 从溪山美地搬到半岛城邦，丢弃了多年攒下的家底。11 月开始玩 MMDVM，使用 Yaesu FT-7800 发射，连接MMDVM中继板，树莓派，覆盖深圳湾，散步骑车通联两不误。

2019 卖了常德的房子，住了5次院，哮喘反复发作，决定停止电子书更新，兴趣转到知乎，B站

2020 准备找工作

职业生涯路上继续打怪升级

3. 如何获得文档

下载 Netkiller 手札 (epub,kindle,chm,pdf)
EPUB
https://github.com/netkiller/netkiller.github.io/tree/master/download/epub
MOBI
https://github.com/netkiller/netkiller.github.io/tree/master/download/mobi
PDF
https://github.com/netkiller/netkiller.github.io/tree/master/download/pdf
CHM
https://github.com/netkiller/netkiller.github.io/tree/master/download/chm

通过 GIT 镜像整个网站

https://github.com/netkiller/netkiller.github.com.git

$ git clone https://github.com/netkiller/netkiller.github.com.git

镜像下载

整站下载

```
wget -m http://www.netkiller.cn/index.html
```

指定下载

```
wget -m wget -m http://www.netkiller.cn/linux/index.html
```

Yum 下载文档

获得光盘介质，RPM包，DEB包，如有特别需要，请联系我

YUM 在线安装电子书

http://netkiller.sourceforge.net/pub/repo/

```
# cat >> /etc/yum.repos.d/netkiller.repo <<EOF
[netkiller]
name=Netkiller Free Books
baseurl=http://netkiller.sourceforge.net/pub/repo/
enabled=1
gpgcheck=0
gpgkey=
EOF
```

查找包

```
# yum search netkiller

netkiller-centos.x86_64 : Netkiller centos Cookbook
netkiller-cryptography.x86_64 : Netkiller cryptography
Cookbook
netkiller-docbook.x86_64 : Netkiller docbook Cookbook
netkiller-linux.x86_64 : Netkiller linux Cookbook
netkiller-mysql.x86_64 : Netkiller mysql Cookbook
netkiller-php.x86_64 : Netkiller php Cookbook
netkiller-postgresql.x86_64 : Netkiller postgresql Cookbook
netkiller-python.x86_64 : Netkiller python Cookbook
netkiller-version.x86_64 : Netkiller version Cookbook
```

安装包

```
yum install netkiller-docbook
```

4. 打赏（Donations）

If you like this documents, please make a donation to support the authors' efforts. Thank you!

您可以通过微信，支付宝，贝宝给作者打赏。

银行(Bank)

招商银行(China Merchants Bank)

开户名：陈景峰

账号：9555500000007459

微信（Wechat）

支付宝（Alipay）

PayPal Donations

https://www.paypal.me/netkiller

5. 联系方式

主站 http://www.netkiller.cn/

备用 http://netkiller.github.io/

繁体网站 http://netkiller.sourceforge.net/

联系作者

Mobile: +86 13113668890

Email: netkiller@msn.com

QQ群: 128659835 请注明“读者”

QQ: 13721218

ICQ: 101888222

注：请不要问我安装问题！

博客 **Blogger**

知乎专栏 https://zhuanlan.zhihu.com/netkiller

LinkedIn: http://cn.linkedin.com/in/netkiller

OSChina: http://my.oschina.net/neochen/

Facebook: https://www.facebook.com/bg7nyt

Flickr: http://www.flickr.com/photos/bg7nyt/

Disqus: http://disqus.com/netkiller/

solidot: http://solidot.org/~netkiller/

SegmentFault: https://segmentfault.com/u/netkiller

Reddit: https://www.reddit.com/user/netkiller/

Digg: http://www.digg.com/netkiller

Twitter: http://twitter.com/bg7nyt

weibo: http://weibo.com/bg7nyt

Xbox club

我的 xbox 上的ID是 netkiller xbox，我创建了一个俱乐部 netkiller 欢迎加入。

Radio

CQ CQ CQ DE BG7NYT:

如果这篇文章对你有所帮助,请寄给我一张QSL卡片, qrz.cn or qrz.com or hamcall.net

Personal Amateur Radiostations of P.R.China

ZONE CQ24 ITU44 ShenZhen, China

Best Regards, VY 73! OP. BG7NYT

守听频率 DMR 438.460 -8 Color 12 Slot 2 Group 46001

守听频率 C4FM 439.360 -5 DN/VW

MMDVM Hotspot:

Callsign: BG7NYT QTH: Shenzhen, China

YSF: YSF80337 - CN China 1 - W24166/TG46001

DMR: BM_China_46001 - DMR Radio ID: 4600441

部分 I. Python 入门

第 1 章 Python 入门

1. 如何快速学习 Python 语言

1.1. 学习编程的目的是什么?

学习编程的目的要明确，解决工作中遇到的痛点

1.2. 很多公司是怎么死的?

很多老板创业，他的想法是先生产产品，然后再找销路，这时发现市场根本不需要他们生产的产品，最后一个都卖不出去，工厂就倒闭了。

学习也是一样，不要想着，我先学，等我学会了，在去找对口工作，结果你会发现可能等你学会了，企业已经不需要这种技术了，或者学的人特别多，竞争非常激烈，很多人抢一个岗位。

所以我们应该是以解决工作痛点为出发点，去学习编程。员工的工资取决于岗位的稀缺性和不可替代性以及为公司创造的价值。我们学习编程，就是为了在该岗位上为公司创造价值，增加岗位竞争力，从而提高工资收入。

1.3. 学习定位问题?

编程是个很宽泛的说法，在编程界分成很多领域，两个领域的程序员相互都不会对方领域的知识。例如开发显卡驱动的跟H5前端程序员是两个隔行如隔山的领域。

1.4. 小白怎么入门呢?

对于没有经验的人，怎么学习编程？其实学习语言很简单，可能最大的阻力是你心理上的，你从心里上否定了你学不会，学习难……

儿童都能学习，你为什么学不了呢?

所谓编程，就是流程控制，我们将现实中的流程搬到电脑上，让电脑替代人去完成。所以就必须用电脑能理解的语言去描述工作流程，这就叫编程。

1.5. 从入门到放弃

为什么很多人学了一半最终放弃了?

答案是，没有应用场景。没有应用场景就等于你在学习屠龙术，在消耗你时间和生命。

我们为什么去学一门编程语言？提高自身涵养吗？修身养性吗？我们学习语言的目的非常明确，解决工作中遇到的问题啊，提高自己职场竞争力，升职加薪，为了以后可以不工作，为了实现财务自由。其他全TMD扯淡。

1.6. 为什么学不会?

学不会真的不是你的问题，是教的人有问题。我们中国的教育是畸形的，在这种畸形教育下培养出来的老师，会让这种教育方式继续在你的身上延续。

我举一个例子，学电脑，第一周会教你什么？计算机原理，认识计算机，然后学习打字，学习操作系统，在学习办公软件，我当年学了三年，毕业的。话说以前用步话机都需要学习，还要培训一个月，就一个按钮的机器，他会叫你通信原理，使用手册，通联用语等等，你很难想想，就如同电话手机，让你先学一周的通信原理，基站原理，手机使用注意事项…… 这就是我们的教育，教你十几年的英语，还是学不会。

现在是否发现，手机是不需要学习，就能使用的，你只需要买一部手机，差一张SIM卡，就可以打电话。上面APP你也从来没有上过什么培训班去学吧?

1.7. 如何快速高效的学习一门语言?

任何一门语言都是由下面几部分组成的:

- 数据类型、数据结构
- 逻辑判断，循环
- 类、函数
- 自带库
- 第三方库

对于没有经验的人来说，1，2，3 可以需要花点时间，1~3个月足以。对于有经验的人（之前使用过其他语言），1，2，3 只需要一周时间。

接下来重点就是类库和函数的学习，你能否熟练写程序就取决于对各种开发库的熟悉程度。新手需要不停的翻越手册，查看每个函数和参数，用过了再知道怎么使用。

我在学习Python 的时候，用了3天快速翻阅，掌握Python基本的数据类型，数据结构和语法。然后直奔开发库，从手册中找出我需要干活用到的函数，对着手册，用了一周就开发出高质量的程序，解决了工作中的实际问题。

记住在职场上，薪资的高低永远取决于所在岗位的稀缺性和不可替代性以及一位公司创造的价值。能找到公司的痛点，并解决掉，这样的员工才有价值。所以我在公司专挑骨头啃，没挑战的活我不干。

职场上从来没有「没有功劳，也有苦劳」一说，升职加薪也不会给干苦活的人。

我第一个Python程序，就涉及了多线程，进程间通信，TCP Socket，本应该使用 C 来完成的程序，使用C语言写估计要搞上半个月到一个月，我用Python 只用了一周完成。爽爆了！！！

我也用这种学习方法去学习 Go 语言，当时做区块链项目，Hyperledger Fabric 的智能合约需要用 Go 语言编写，用了不到一周就把智能合约写完。

后来又有一个项目用到了 EOS 区块链，需要使用 C++ 写智能合约，我也如法炮制。

1.8. 碎片化学习

我会说这样的学习方法，缺点是不扎实，但解决实际问题。扎实与否取决于后面你是否能坚持不懈的学习。

这种学习方式非常适合非职业程序猿或者晋升到管理层脱落一线的程序猿。即我们必须解决工作中遇到的问题，我们又不是天天都在写程序。当然前提是你需要有一个好的基础，至少在此前你做过多年的程序猿。

没有经验的程序猿，首先要掌握前面所说的 1，2，3 三个步骤，只要突破了这三个阶段学习，后面如履平地。

2. install

2.1. venv

```
python3 -m venv /srv/python
```

```
root@debian:~# update-alternatives --install /usr/bin/python
python /srv/python/bin/python 1
```

2.2. Docker 安装

```
$ docker run --name python --rm -it python
Python 3.10.1 (main, Dec 21 2021, 09:01:08) [GCC 10.2.1
20210110] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Dockerfile 制作自己的镜像

```
FROM python:latest

MAINTAINER Netkiller <netkiller@msn.com>

RUN pip install pymysql sqlalchemy openpyxl pandas -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

```
RUN mkdir /data
ADD export.py /srv
VOLUME ["/tmp"]
WORKDIR /data
#EXPOSE 80 443
ENTRYPOINT python3 /srv/export.py
```

2.3. dnf 安装 python3.11

安装 Python 3.11 适用鱼 Rocky Linux 9.2 和 AlmaLinux 9.2

```
dnf install -y python3.11 python3.11-pip
```

系统默认是 python3.9，将其切换到 python3.11

```
update-alternatives --install /usr/bin/python python
/usr/bin/python3.11 1
update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.11 2
```

切换 pip 到 3.11

```
[root@iZwz9cug5b7jbx1dc4nwniZ ~]# mv /usr/bin/pip{,.backup}
[root@iZwz9cug5b7jbx1dc4nwniZ ~]# mv /usr/bin/pip3{,.backup}
[root@iZwz9cug5b7jbx1dc4nwniZ ~]# alternatives --install
/usr/bin/pip pip /usr/bin/pip3.11 1
[root@iZwz9cug5b7jbx1dc4nwniZ ~]# alternatives --install
/usr/bin/pip3 pip3 /usr/bin/pip3.11 2
[root@iZwz9cug5b7jbx1dc4nwniZ ~]# pip -V
pip 22.3.1 from /usr/lib/python3.11/site-packages/pip (python
```

```
3.11)
```

检查 python 是否工作正常

```
[root@iZwz9cug5b7jbx1dc4nwniZ srv]# alternatives --list
libnssckbi.so.x86_64    auto    /usr/lib64/pkcs11/p11-kit-
trust.so
soelim                  auto    /usr/bin/soelim.groff
cifs-idmap-plugin       auto    /usr/lib64/cifs-
utils/cifs_idmap_sss.so
iptables                auto    /usr/sbin/iptables-nft
ebtables                auto    /usr/sbin/ebtables-nft
arptables               auto    /usr/sbin/arptables-nft
ld                      auto    /usr/bin/ld.bfd
man                     auto    /usr/bin/man.man-db
nc                      auto    /usr/bin/ncat
man.7.gz                auto    /usr/share/man/man7/man.man-
pages.7.gz
libwbclient.so.0.15-64  auto
/usr/lib64/samba/wbclient/libwbclient.so.0.15
python                  auto    /usr/bin/python3.11

[root@iZwz9cug5b7jbx1dc4nwniZ srv]# alternatives --display
python
python - status is auto.
 link currently points to /usr/bin/python3.11
/usr/bin/python3.11 - priority 1
Current `best' version is /usr/bin/python3.11.

[root@iZwz9cug5b7jbx1dc4nwniZ srv]# python
Python 3.11.2 (main, May 24 2023, 00:00:00) [GCC 11.3.1
20221121 (Red Hat 11.3.1-4)] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

2.4. 编译安装 Python3.x

```
./configure --prefix=/usr/local/python-3.1.2
make && make install
```

2.5. Ubunut 13.04 环境安装python3

```
$ sudo apt-get install python3
```

python 3.3

```
$ sudo apt-get install python3.3
```

2.6. pypy - fast alternative implementation of Python - PyPy interpreter

http://pypy.org/

Ubuntu 环境安装

```
$ apt-cache search pypy | grep pypy
pypy - fast alternative implementation of Python - PyPy
interpreter
pypy-dev - header files for PyPy (an alternative Python
interpreter)
pypy-doc - developer Documentation for PyPy (an alternative
Python interpreter)
pypy-lib - standard library for PyPy (an alternative Python
interpreter)
python-pypy.translator.sandbox - sandboxed PyPy interpreter
```

```
$ sudo apt-get install pypy
```

CentOS 环境安装

```
# yum install pypy
```

2.7. Eric Python IDE

```
apt-get install eric
```

2.8. python to exe

pyinstaller

http://www.pyinstaller.org/

Linux

安装

```
pip install pyinstaller
```

制作EXE文件

```
$ pyinstaller test.py
```

dist目录下会生成可执行文件

py2exe

http://www.py2exe.org/

此软件 2008-11-16 后不再更新

2.9. Python2.x

编译安装

Python2.x

```
wget http://www.python.org/ftp/python/2.x.x/Python-2.x.x.tgz
tar zxvf Python-2.x.x.tgz
cd Python-2.x.x
./configure --prefix=/usr/local/python2.x
make
make install

ln -s /usr/local/python/bin/python2.5 /usr/bin/
ln -s /usr/local/python/bin/* /usr/local/bin/
```

Ubuntu 安装

```
sudo apt-get install python
sudo apt-get install python-setuptools
```

3. Python Package Index (PyPI)

3.1. 什么是 PyPI

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community.

Package authors use PyPI to distribute their software.

PyPI 是一个 Python 语言软件仓库，开发包的作者通过Python社区共享他们的作品。软件开发者可以在平台上搜索、安装和使用这些共享资源。

3.2. 准备目录

发布 Python 包到 PyPI（Python Package Index） 的步骤

下面开始介绍如何将自己的 Python 包发布到Python Package Index。

准备一个 Python 项目，目录结构如下

```
neo@MacBook-Pro-Neo ~/workspace % mkdir netkiller-project
neo@MacBook-Pro-Neo ~/workspace % cd netkiller-project
```

3.3. 创建LICENSE文件

上传到 Python Package Index 的包必须包含该文件，外国法律是很严的，License 文件是告诉用户该软件包的授权条款。你可以在下面网

站找到 License 复制粘贴到 LICENSE文件中。

Choose an open source license 我使用的是MIT License

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % curl -s
https://mit-license.org | sed 's/<[^>]*>//g ; /^$/d' > LICENSE
```

3.4. 项目描述文件

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % vim
README.md
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % cat
README.md
= Netkiller-Project
演示如何上传包到PyPI
```

3.5. 库代码

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % mkdir
netkiller
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % vim
netkiller/__init__.py
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % cat
netkiller/__init__.py
name = "netkiller"
__version__ = "0.0.1"
```

3.6. setuptools 构建脚本

setup.py 文件是 setuptools 打包工具的构建脚本，主要用来描述项目名称，版本，作者邮箱，项目描述信息，网址，运行环境，依赖哪些库，支持哪些操作系统，等等。

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % vim
setup.py
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % cat
setup.py
import setuptools

with open("README.md", "r") as fh:
  long_description = fh.read()

setuptools.setup(
  name="netkiller-project",
  version="0.0.1",
  author="Neo Chen",
  author_email="netkiller@msn.com",
  description="Netkiller Python Package",
  long_description=long_description,
  long_description_content_type="text/markdown",
  url="https://github.com/netkiller/netkiller.netkiller.io",
  packages=setuptools.find_packages(),
  classifiers=[
  "Programming Language :: Python :: 3",
  "License :: OSI Approved :: MIT License",
  "Operating System :: OS Independent",
  ],
)
```

3.7. 构建包

使用命令 python3 setup.py sdist bdist_wheel 打包

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % python3
setup.py sdist bdist_wheel
running sdist
```

```
running egg_info
creating netkiller_project.egg-info
writing netkiller_project.egg-info/PKG-INFO
writing dependency_links to netkiller_project.egg-
info/dependency_links.txt
writing top-level names to netkiller_project.egg-
info/top_level.txt
writing manifest file 'netkiller_project.egg-info/SOURCES.txt'
reading manifest file 'netkiller_project.egg-info/SOURCES.txt'
writing manifest file 'netkiller_project.egg-info/SOURCES.txt'
running check
creating netkiller-project-0.0.1
creating netkiller-project-0.0.1/netkiller_project.egg-info
copying files to netkiller-project-0.0.1...
copying README.md -> netkiller-project-0.0.1
copying setup.py -> netkiller-project-0.0.1
copying netkiller_project.egg-info/PKG-INFO -> netkiller-
project-0.0.1/netkiller_project.egg-info
copying netkiller_project.egg-info/SOURCES.txt -> netkiller-
project-0.0.1/netkiller_project.egg-info
copying netkiller_project.egg-info/dependency_links.txt ->
netkiller-project-0.0.1/netkiller_project.egg-info
copying netkiller_project.egg-info/top_level.txt -> netkiller-
project-0.0.1/netkiller_project.egg-info
Writing netkiller-project-0.0.1/setup.cfg
creating dist
Creating tar archive
removing 'netkiller-project-0.0.1' (and everything under it)
running bdist_wheel
running build
installing to build/bdist.macosx-11-x86_64/wheel
running install
running install_egg_info
Copying netkiller_project.egg-info to build/bdist.macosx-11-
x86_64/wheel/netkiller_project-0.0.1-py3.9.egg-info
running install_scripts
adding license file "LICENSE" (matched pattern "LICEN[CS]E*")
creating build/bdist.macosx-11-x86_64/wheel/netkiller_project-
0.0.1.dist-info/WHEEL
creating 'dist/netkiller_project-0.0.1-py3-none-any.whl' and
adding 'build/bdist.macosx-11-x86_64/wheel' to it
adding 'netkiller_project-0.0.1.dist-info/LICENSE'
adding 'netkiller_project-0.0.1.dist-info/METADATA'
adding 'netkiller_project-0.0.1.dist-info/WHEEL'
adding 'netkiller_project-0.0.1.dist-info/top_level.txt'
```

```
adding 'netkiller_project-0.0.1.dist-info/RECORD'
removing build/bdist.macosx-11-x86_64/wheel
```

3.8. 查看包

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % ls dist
netkiller-project-0.0.1.tar.gz           netkiller_project-
0.0.1-py3-none-any.whl
```

3.9. 上传包

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % twine
upload dist/*
Uploading distributions to https://upload.pypi.org/legacy/
Enter your username: netkiller
Enter your password:
Uploading netkiller_project-0.0.1-py3-none-any.whl
100%|
━━━
```

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % pip3
install netkiller-project
Collecting netkiller-project
  Downloading netkiller_project-0.0.1-py3-none-any.whl (2.8 kB)
Installing collected packages: netkiller-project
Successfully installed netkiller-project-0.0.1
```

3.11. 使用包

```
neo@MacBook-Pro-Neo ~/tmp % cat test.py
import netkiller
print(netkiller.name)
```

3.12. 卸载包

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % pip3
uninstall netkiller-project
Found existing installation: netkiller-project 0.0.3
Uninstalling netkiller-project-0.0.3:
  Would remove:
    /usr/local/lib/python3.9/site-packages/netkiller/*
    /usr/local/lib/python3.9/site-packages/netkiller_project-
0.0.3.dist-info/*
Proceed (y/n)? y
  Successfully uninstalled netkiller-project-0.0.3
```

3.13. Built distribution

```
Format  Description     Notes
gztar   gzipped tar file (.tar.gz)      Unix 默认
bztar   bzipped tar file (.tar.bz2)
xztar   xzipped tar file (.tar.xz)
ztar    compressed tar file (.tar.Z)
tar     tar file (.tar)
zip     zip file (.zip) Windows 默认
rpm     RPM
pkgtool Solaris pkgtool
sdux    HP-UX swinstall
wininst self-extracting ZIP file for Windows
msi     Microsoft Installer.


$ python setup.py bdist --formats=rpm

setuptools 提供了如下简化命令:

Command Formats Notes
bdist_dumb      tar, gztar, bztar, xztar, ztar, zip     Windows
默认 zip, Unix 默认 gztar
bdist_rpm       rpm, srpm
bdist_wininst   wininst
bdist_msi       msi

打 rpm 包可以使用:

$ python setup.py bdist_rpm
```

3.14. 免密登录

创建 ~/.pypirc 文件

```
neo@MacBook-Pro-Neo ~/git/kindle % cat ~/.pypirc
[pypi]
username:netkiller
password:******
```

```
```

****** 改为你的密码

3.15. 其他 Python 包管理工具

distutils

http://docs.python.org/3/install/index.html

http://docs.python.org/3/distutils/index.html

创建 setup.py 如下：

```
# cat setup.py
from distutils.core import setup

setup (
    name = 'firewall',
    version = '1.0.0',
    py_modules = ['firewall'],
    author = 'neo.chen',
    author_email = 'netkiller@msn.com',
    description = 'Module firewall',
    url='http://netkiller.github.io/',
)
```

注意：name和py_modules这两个参数一定要与firewall.py文件名相同，否则发布安装时会发出模块文件找不到的错误.

打包操作

```
# python setup.py sdist
running sdist
warning: sdist: missing required meta-data: url
warning: sdist: manifest template 'MANIFEST.in' does not exist
(using default file list)
```

```
warning: sdist: standard file not found: should have one of
README, README.txt
writing manifest file 'MANIFEST'
creating firewall-1.0.0
making hard links in firewall-1.0.0...
hard linking firewall.py -> firewall-1.0.0
hard linking setup.py -> firewall-1.0.0
creating dist
tar -cf dist/firewall-1.0.0.tar firewall-1.0.0
gzip -f9 dist/firewall-1.0.0.tar
tar -cf dist/firewall-1.0.0.tar firewall-1.0.0
gzip -f9 dist/firewall-1.0.0.tar
removing 'firewall-1.0.0' (and everything under it)
```

sdist 生成的文件

```
# ll dist/
total 4
-rw-r--r--. 1 root root 2123 Aug  9 12:41 firewall-1.0.0.tar.gz
```

安装包

```
# python setup.py install
running install
running build
running build_py
creating build
creating build/lib
copying firewall.py -> build/lib
running install_lib
copying build/lib/firewall.py -> /usr/lib/python2.6/site-
packages
byte-compiling /usr/lib/python2.6/site-packages/firewall.py to
firewall.pyc
running install_egg_info
Writing /usr/lib/python2.6/site-packages/firewall-1.0.0-
py2.6.egg-info
```

安装后

```
# ll /usr/lib/python2.6/site-packages/firewall*
-rw-r--r--. 1 root root   201 Aug  9 12:42
/usr/lib/python2.6/site-packages/firewall-1.0.0-py2.6.egg-info
-rw-r--r--. 1 root root  6145 Aug  9 11:28
/usr/lib/python2.6/site-packages/firewall.py
-rw-r--r--. 1 root root 11858 Aug  9 12:42
/usr/lib/python2.6/site-packages/firewall.pyc
```

distribute

distribute_setup 是 setuptools 的替代品

```
$ wget http://python-distribute.org/distribute_setup.py
$ sudo python distribute_setup.py
```

或者

```
curl -sS http://python-distribute.org/distribute_setup.py |
python
```

3.16. pip - A tool for installing and managing Python packages

https://pip.readthedocs.io/en/stable/installing/

安装 pip

自从Python 3.4版本开始，pip已经被内置在Python中，所以无需再次安装。如果你的系统中没有pip/pip3命令，或者不小心卸载了，可以使用下面方式安装。

使用 easy_install 安装 pip

使用easy_install安装pip， 同时pip也是easy_install替代品

```
# easy_install pip
```

Ubuntu 安装 pip

```
$ sudo apt install python3-pip
```

Mac

```
neo@MacBook-Pro ~ % brew install python3
neo@MacBook-Pro ~ % pip3 install scrapy
```

安装 python3 会携带 pip3 命令

查看版本

```
neo@MacBook-Pro-Neo ~ % pip --version
pip 21.1 from /usr/local/lib/python3.9/site-packages/pip
(python 3.9)

neo@MacBook-Pro-Neo ~ % pip3 --version
pip 21.1 from /usr/local/lib/python3.9/site-packages/pip
(python 3.9)
```

升级 pip 命令

```
neo@MacBook-Pro-Neo ~ % pip install --upgrade pip
```

查询包

查询包

```
$ pip search "Markdown"
```

安装包

安装包

```
$ pip install Markdown
```

指定版本号

安装特定版本的包， 通过使用 ==, >=, <=, >, < 来指定一个版本号。

比如说我要安装3.4.1版本的matplotlib

```
pip install matplotlib==3.4.1
```

```
$ pip install 'Markdown<2.0'
$ pip install 'Markdown>2.0,<2.0.3'
```

用户级包管理，默认root是全局安装

```
pip3 install --user netkiller-firewall
```

安装 wheel文件

```
pip install dist/netkiller_firewall-0.0.1-py3-none-any.whl
```

操作演示

```
neo@MacBook-Pro-Neo ~ % cd workspace/firewall
neo@MacBook-Pro-Neo ~/workspace/firewall % ls dist
netkiller-firewall-0.0.1.tar.gz           netkiller_firewall-
0.0.1-py3.9.egg
netkiller_firewall-0.0.1-py3-none-any.whl
neo@MacBook-Pro-Neo ~/workspace/firewall % pip install
dist/netkiller_firewall-0.0.1-py3-none-any.whl
Processing ./dist/netkiller_firewall-0.0.1-py3-none-any.whl
Installing collected packages: netkiller-firewall
Successfully installed netkiller-firewall-0.0.1
```

卸载包

卸载包

```
$ pip uninstall Markdown
```

升级包

升级包到当前最新的版本，可以使用-U 或者 --upgrade

```
# pip install distribute --upgrade
$ pip install -U Markdown
```

显示包详细信息

```
neo@MacBook-Pro-Neo ~ % pip show netkiller-logging
Name: netkiller-logging
Version: 0.0.5
Summary: log send to remote
Home-page: http://netkiller.github.io
Author: None
Author-email: netkiller@msn.com
License: BSD
Location: /usr/local/lib/python3.9/site-packages
Requires:
Required-by:
```

列出已经安装的包

```
neo@MacBook-Pro-Neo ~ % pip list
Package                                            Version
-------------------------------------------------- ---------
autopep8                                           1.5.6
bleach                                             3.3.0
certifi                                            2020.12.5
chardet                                            4.0.0
click                                              7.1.2
colorama                                           0.4.4
cv                                                 1.0.0
cycler                                             0.10.0
decorator                                          4.4.2
dlib                                               19.22.0
docutils                                           0.17
```

查看需要升级的包

```
neo@MacBook-Pro-Neo ~ % pip list -o
```

目前已经安装的库中，看哪些需要版本升级

```
neo@MacBook-Pro-Neo ~ % pip list -o
Package                                            Version
Latest   Type
-------------------------------------------------- --------- ---
----- -----
autopep8                                           1.5.6
1.5.7    wheel
decorator                                          4.4.2
5.0.7    wheel
docutils                                           0.17
0.17.1   wheel
easyocr                                            1.3.0.1
1.3.1    wheel
```

```
et-xmlfile                                          1.0.1
1.1.0    wheel
idna                                                2.10      3.1
wheel
importlib-metadata                                  3.10.0
4.0.1    wheel
Pygments                                            2.8.1
2.9.0    wheel
pyobjc                                              7.1       7.2
wheel
```

批量安装库

如果一个项目需要安装很多库，可以将他们放入一个文件 requirements.txt，例如那可以批量安装：

```
pip install -r requirements.txt
```

requirements.txt 文件内容格式如下：

```
netkiller-logging==0.0.4
netkiller-firewall
```

操作演示

```
neo@MacBook-Pro-Neo ~ % vim requirements.txt

neo@MacBook-Pro-Neo ~ % cat requirements.txt
netkiller-logging
netkiller-firewall
```

```
neo@MacBook-Pro-Neo ~ % pip install -r requirements.txt
Collecting netkiller-logging
  Downloading netkiller_logging-0.0.5-py3-none-any.whl (14 kB)
Collecting netkiller-firewall
  Downloading netkiller_firewall-0.0.1-py3-none-any.whl (5.4
kB)
Installing collected packages: netkiller-logging, netkiller-
firewall
```

导出已安装的包

列出安装包的描述信息

```
$ pip freeze
```

操作演示

```
neo@MacBook-Pro-Neo ~ % pip freeze
autopep8==1.5.6
bleach==3.3.0
certifi==2020.12.5
chardet==4.0.0
click==7.1.2
colorama==0.4.4
cv==1.0.0
cycler==0.10.0
decorator==4.4.2
dlib==19.22.0
docutils==0.17
easyocr==1.3.0.1
et-xmlfile==1.0.1
face-recognition==1.3.0
face-recognition-models==0.3.0
idna==2.10
```

```
imageio==2.9.0
imageio-ffmpeg==0.4.3
importlib-metadata==3.10.0
keyboard==0.13.5
keyring==23.0.1
kiwisolver==1.3.1
lxml==4.6.3
matplotlib==3.4.1
moviepy==1.0.3
netkiller-devops==0.0.1
netkiller-kindle==0.0.3
networkx==2.5.1
numpy==1.20.2
opencv-python==4.5.1.48
openpyxl==3.0.7
packaging==20.9
pandas==1.2.4
Pillow==8.2.0
pkginfo==1.7.0
proglog==0.1.9
progress==1.5
PyAudio==0.2.11
pycodestyle==2.7.0
Pygments==2.8.1
pyobjc==7.1
python-dateutil==2.8.1
python-docx==0.8.10
pytz==2021.1
PyWavelets==1.1.1
PyYAML==5.4.1
readme-renderer==29.0
requests==2.25.1
requests-toolbelt==0.9.1
rfc3986==1.4.0
scikit-image==0.18.1
scipy==1.6.2
six==1.15.0
SpeechRecognition==3.8.1
tabulate==0.8.9
tifffile==2021.3.31
toml==0.10.2
torch==1.8.1
torchvision==0.9.1
tqdm==4.59.0
twine==3.4.1
```

```
typing-extensions==3.7.4.3
urllib3==1.26.4
webencodings==0.5.1
xlrd==2.0.1
zipp==3.4.1
```

导出已经安装的包列表，并保存到到本地文件中：

```
pip freeze > requirements.txt
```

兼容性检查

验证已安装的包是否有兼容依赖问题

```
pip check package-name
```

操作演示

```
neo@MacBook-Pro-Neo ~ % pip check netkiller-firewall
No broken requirements found.
```

从 PyPI 下载 whl 文件到本地硬盘

将库下载到本地指定目录

```
```

```
pip download package_name -d "要保存的文件路径"
```

操作演示

```
neo@MacBook-Pro-Neo ~ % pip download netkiller-logging -d
/tmp/netkiller-logging
Collecting netkiller-logging
  Using cached netkiller_logging-0.0.5-py3-none-any.whl (14 kB)
Saved /private/tmp/netkiller-logging/netkiller_logging-0.0.5-
py3-none-any.whl
Successfully downloaded netkiller-logging

neo@MacBook-Pro-Neo ~ % ls /tmp/netkiller-logging
netkiller_logging-0.0.5-py3-none-any.whl
```

切换 pip 镜像

由于 pip 源服务器放在外国，所以我们安装包的时候速度非常慢。

国内一些企业和组织做了 pip 镜像，他们每个一定时间从外国服务器同步一次数据到国内服务器，我们将 pip 切换到国内服务器后，再下载包就不会去外国服务器，所以下载速度大大提高。

临时下载一个包

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple
matplotlib
```

永久性切换

```
pip config set global.index-url
https://pypi.tuna.tsinghua.edu.cn/simple
```

设为默认后，以后安装库都将从清华镜像源下载

下面是国内常用镜像服务器地址：

清华镜像 https://pypi.tuna.tsinghua.edu.cn/simple

阿里云镜像 https://mirrors.aliyun.com/pypi/simple/

中科大镜像 https://mirrors.bfsu.edu.cn/pypi/web/simple/

豆瓣镜像 http://pypi.doubanio.com/simple/

4. Python 模块

4.1. 同级目录

同级目录下有两个文件，分别是 a.py 和 test.py。我们将在 test.py 中导入 a.py

```
a.py
test.py
```

a.py 文件

```
neo@MacBook-Pro-Neo ~/workspace/devops % cat a.py
def say():
    print("hello")%
```

test.py 文件

```
neo@MacBook-Pro-Neo ~/workspace/devops % cat test.py
import a
a.say()
```

运行结果

```
neo@MacBook-Pro-Neo ~/workspace/devops % python3 test.py
hello
```

4.2. 一级目录

```
dir/a.py
dir/__init__.py
test.py
```

```
neo@MacBook-Pro-Neo ~/workspace/devops % cat dir/a.py
def say():
    print("hello")

neo@MacBook-Pro-Neo ~/workspace/devops % cat test.py
from dir import a
a.say()
```

```
neo@MacBook-Pro-Neo ~/workspace/devops % python3 test.py
hello
```

4.3. 二级子目录

```
test.py
dir
  |--subdir
    |-- b.py
```

```
neo@MacBook-Pro-Neo ~/workspace/devops % cat dir/subdir/b.py
def sayhello():
    print("helloworld!!!")

neo@MacBook-Pro-Neo ~/workspace/devops % cat test.py
from dir.subdir import b
b.sayhello()%
```

```
neo@MacBook-Pro-Neo ~/workspace/devops % python3 test.py
helloworld!!!
```

4.4. 子目录调用上级目录

```
neo@MacBook-Pro-Neo ~/workspace/devops % find dir
dir
dir/a.py
dir/subdir
dir/subdir/c.py
dir/subdir/b.py
```

```
neo@MacBook-Pro-Neo ~/workspace/devops % cat dir/subdir/c.py
import b
b.sayhello()

import sys,os
path =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

```
# print(path)
sys.path.insert(0, path)
import a
a.say()

sys.path.append(os.path.dirname(os.path.dirname(os.path.dirname
(os.path.abspath(__file__)))))
from dir import a
a.say()
```

```
neo@MacBook-Pro-Neo ~/workspace/devops % python3
dir/subdir/c.py
helloworld!!!
hello
hello
```

4.5. 导入类

```
neo@MacBook-Pro-Neo ~/workspace/devops % cat
netkiller/dir/class1.py
class TestClass():
        def __init__(self, name="None"):
                self.name = name
        def say(self):
                print(self.name)
```

```
from netkiller.dir.class1 import *
tc = TestClass('neo')
tc.say()
```

5. 数据类型

5.1. type 数据类型检测

http://docs.python.org/library/types.html

```
>>> type( [] ) == list
True
>>> type( {} ) == dict
True
>>> type( "" ) == str
True
>>> type( 0 ) == int
True
>>> class Test1 ( object ):
    pass
>>> class Test2 ( Test1 ):
    pass
>>> a = Test1()
>>> b = Test2()
>>> type( a ) == Test1
True
>>> type( b ) == Test2
True
>>> type( b ) == Test1
False
>>> isinstance( b, Test1 )
True
>>> isinstance( b, Test2 )
True
>>> isinstance( a, Test1 )
True
>>> isinstance( a, Test2 )
False
>>> isinstance( [], list )
True
>>> isinstance( {}, dict )
True
```

```
>>> a = []
>>> type(a)
<type 'list'>
>>> f = ()
>>> type(f)
<type 'tuple'>
```

5.2. 字符串

String function

str.find()

找到字符，返回字符串位置。没有找到返回 -1

```
"aaa bbb ccc".find('aaa')
```

str.find()

查找并替换字符串

```
a = 'hello word'
a.replace('word','python')
```

format 方法

```
DB_URL = 'mysql+pymysql://{}:{}@{}:{}/{}?
```

```
charset=utf8'.format(
    USERNAME, PASSWORD, HOST_NAME, PORT, DB_NAME
)
```

格式化字典输出

```
member = {'name':'neo','age':18}
'my name is {name},age is {age}'.format(**member)
#**dict
```

输出

```
>>> member = {'name':'neo','age':18}
>>> 'my name is {name},age is {age}'.format(**member)
'my name is neo,age is 18'
```

Convert str to bytes in python

```
>>> b = str.encode(y)
>>> type(b) >>> b b'Hello World!'
To alter from bytes to str, capitalize on bytes.decode().
>>> z = b"Hello World!"
>>> y = "Hello World!"
>>> type(z)

>>> type(y)

To alter from str to bytes, capitalize on str.encode().
>>> a = bytes.decode(z)
>>> type(a)
```

```
>>> a
‘Hello World!’


# to utf-8
'BG7NYT'.encode('utf-8')
# to utf-16
'BG7NYT'.encode('utf-16')
```

% 字符串格式化输出

```
strHello = "the length of (%s) is %d" %('Hello
World',len('Hello World'))
print strHello
```

前导字符串加0

```
for i in range(5):
    print("%03d" % i)

for i in range(100):
    print('{0:0>3d}'.format(i))
```

f-string 格式化字符串

```
text = "Hello"
print(f"{text:-^15}")
```

```
number = 1
print(f"{number:0<5}")
print(f"{number:0>5}")
```

```
-----Hello-----
10000
00001
```

打印99乘法表

```
for i in range(1,10):
    for j in range(1,i+1):
        print(f"{j}*{i}={j*i}",end=" ")
    print("")
```

```
1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

格式化日期

```
>>> from datetime import *
>>> a = date.today()
>>> a
datetime.date(2023, 9, 2)
>>> f"{a:%Y-%m-%d}"
'2023-09-02'
```

去除中文

```
import re
prompt='给我画一张 An octopus catches crabs in the sea 3D'
string=re.sub('[\u4e00-\u9fa5]', '', prompt)
print(string)
```

去除标点符号

```
simple_punctuation = '[’!"#$%&\'()*+,-/:;<=>?
@[\\]^_`{|}~，。,]'
line = re.sub(simple_punctuation, '', string)
```

去除数字

```
re.sub("[0-9]", " ", line)
```

5.3. float 浮点数值

```
for i in [12.12300, 12.00, 200.12000, 200.0]:
    print('{:g}'.format(i))
```

5.4. Array

遍历数字

```
colours = ["RED","GREEN","BLUE"]
for colour in colours:
    print colour
```

```
colours = ["RED","GREEN","BLUE"]
for i in range(0, len(colours)):
    print i, colour[i]
```

split / join

```
>>> str = 'a|b|c|d|e'
>>> str.split("|")
```

```
['a', 'b', 'c', 'd', 'e']

>>> list = ['a', 'b', 'c', 'd', 'e']
>>> "|".join(list)
'a|b|c|d|e'
```

5.5. 日期和时间

当前时间

当前日期

```
import time
dt = time.strftime('%Y-%m-%d.%X',time.localtime(time.time()))
print(dt)
```

```
>>> import time
>>> dt = time.strftime('%Y-%m-
%d.%X',time.localtime(time.time()))
>>> print(dt)
2014-01-23.11:07:28
```

```
from datetime import datetime

print(datetime.today())
print(datetime.now())
print(datetime.now().date())
print(datetime.now().time())
```

```
2023-03-11 18:30:34.657155
2023-03-11 18:30:34.657176
2023-03-11
18:30:34.657182
```

```
today = datetime.today()
yesterday = (today - timedelta(days=1)).date()
```

生成时间

```
from datetime import date
print(date(2002, 12, 31))
```

日期格式化

```
from datetime import datetime
datetime.today().strftime('%Y-%m-%d.%H:%M:%S')

timepoint = time.strftime('%Y-%m-
%d.%H:%M:%S',time.localtime(time.time()))
```

字符串转日期

```
from datetime import datetime

dt_str = '27/10/20 05:23:20'
dt_obj = datetime.strptime(dt_str, '%d/%m/%y %H:%M:%S')

print("The type of the date is now",  type(dt_obj))
print("The date is", dt_obj)
```

日期转字符串

```
from datetime import date
print(date(2002, 12, 31).strftime('%Y-%m-%d'))
```

日期运算

天数差

```
from datetime import datetime

begin='2023-02-01'
end='2023-02-28'

interval=datetime.strptime(end,'%Y-%m-%d').date() -
datetime.strptime(begin,'%Y-%m-%d').date()
print(interval.days)
```

月/周 首尾计算

```
from datetime import timedelta,datetime

now = datetime.now()

# 获取昨天日期:
yesterday = now - timedelta(days=1)
# 获取明天日期:
tomorrow = now + timedelta(days=1)
```

```
# 获取本周第一天和最后一天:
this_week_start = now - timedelta(days=now.weekday())
this_week_end = now + timedelta(days=6-now.weekday())

# 获取上周第一天和最后一天:
last_week_start = now - timedelta(days=now.weekday()+7)
last_week_end = now - timedelta(days=now.weekday()+1)

# 获取上月第一天和最后一天:

last_month_end = this_month_start - timedelta(days=1)
last_month_start = datetime.datetime(last_month_end.year,
last_month_end.month, 1)

# 获取本季第一天和最后一天:

month = (now.month - 1) - (now.month - 1) % 3 + 1
this_quarter_start = datetime.datetime(now.year, month, 1)
this_quarter_end = datetime.datetime(now.year, month,
calendar.monthrange(now.year, now.month)[1])

# 获取上季第一天和最后一天:

last_quarter_end = this_quarter_start - timedelta(days=1)
last_quarter_start = datetime.datetime(last_quarter_end.year,
last_quarter_end.month - 2, 1)

# 获取本年第一天和最后一天:
```

```
this_year_start = datetime.datetime(now.year, 1, 1)
this_year_end = datetime.datetime(now.year + 1, 1, 1) -
timedelta(days=1)

# 获取去年第一天和最后一天:

last_year_end = this_year_start - timedelta(days=1)
last_year_start = datetime.datetime(last_year_end.year, 1, 1)
```

日期范围计算

计算两个日期之间的月份

```
import calendar
import datetime

begin = "2017-11-15"
end = "2018-04-23"


def monthlist(begin, end):
    begin = datetime.datetime.strptime(begin, "%Y-%m-%d")
    end = datetime.datetime.strptime(end, "%Y-%m-%d")

    result = []
    while True:
        if begin.month == 12:
            next = begin.replace(year=begin.year+1, month=1,
day=1)
        else:
            next = begin.replace(month=begin.month+1, day=1)
        if next > end:
            break

        day = calendar.monthrange(begin.year, begin.month)[1]

        result.append((begin.strftime("%Y-%m-%d"),
```

```
                        begin.replace(day=day).strftime("%Y-%m-
%d")))
        begin = next
    result.append((begin.strftime("%Y-%m-%d"),
end.strftime("%Y-%m-%d")))
    return result


lists = monthlist(begin, end)
print(lists)
for (b, e) in lists:
    print(b, e)
```

日期排序

```
import datetime

list_time=["2018-05-01","2018-02-01","2018-07-10","2019-06-
01"]

list_time1=sorted(list_time,key=lambda
date:datetime.datetime.strptime(date,"%Y-%m-%d").timestamp())

print('before', list_time)
print('after', list_time1)

print('before', list_time)
print('min', min(list_time1), 'max', max(list_time1))
```

5.6. bytes 类型

bytes to string

```
.decode()
```

BOM头

```
BOM : fffe
BOM_BE : feff
BOM_LE : fffe
BOM_UTF8 : efbb bf
BOM_UTF16 : fffe
BOM_UTF16_BE : feff
BOM_UTF16_LE : fffe
BOM_UTF32 : fffe 0000
BOM_UTF32_BE : 0000 feff
BOM_UTF32_LE : fffe 0000
```

replace

bytes类型的数据，替换方法，不要忘记b""

```
a = b"abc"
b = a.replace(b"a", b"f")
print(b)
```

pack/unpack

6. 数据结构

6.1. List

随机抽取list成员

```
import random

# random choice from a list
for i in range(5):
    print random.choice([1, 2, 3, 5, 9])
```

```
test = []
test.append('1')
test.append('b')
test.append('c')
print(test)
```

6.2. Set

```
>>> set = {'a','b','c'}
>>> set
set(['a', 'c', 'b'])

>>> set = {"one", "two", "three","one"}
>>> set
set(['three', 'two', 'one'])
```

```
>>> set.add('four')
>>> set
set(['four', 'three', 'two', 'one'])
```

```
>>> ','.join(set);
'four,three,two,one'

>>> set('four,three,two,one'.split(','));
{'one', 'four', 'three', 'two'}

>>> fruit = [ 'banana', 'orange', 'pineapple' ]
>>> set(fruit)
{'banana', 'pineapple', 'orange'}
```

6.3. Dict 字典

随机选择字典的 key 和 value

随机字典的 key 和 value

```
<![CDATA[
names = {1: '张三', 2: '李四', 3: '王五', 4: '赵六', 5: '牛七', 6:
'马八'}
print(random.choice(list(names.keys())))
print(random.choice(list(names.values())))
```

字典合并

```
dict1 = {'name','neo'}
dict2 = {'nickname','netkiller'}
dict1.update(dict2)
print(dict1)
```

```
{'name', 'neo', 'nickname', 'netkiller'}
```

取最大值

```
data = [
    {
      "question": "为什么过只加汇摇晃",
      "answer": "因为支架是由很多支索小圆轮组成的，所以吊厢过索轮时会有轻微的震动和摇晃，这些都是正常的",
      "ratio": 0,
      "distance": 41.112388610839844
    },
    {
      "question": "吊厢过之家为什么会摇晃",
      "answer": "因为支架是由很多支索小圆轮组成的，所以吊厢过索轮时会有轻微的震动和摇晃，这些都是正常的",
      "ratio": 0,
      "distance": 42.78510284423828
    },
    {
      "question": "嗦道为啥会摇晃",
      "answer": "因为支架是由很多支索小圆轮组成的，所以吊厢过索轮时会有轻微的震动和摇晃，这些都是正常的",
      "ratio": 1,
      "distance": 44.00849914550781
    }
```

```
  ]

ret = max(data, key=lambda dic: dic['ratio'])
print(ret)
```

7. Class

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'

x = MyClass()
```

7.1. __init__ 构造方法

```
def __init__(self):
    self.data = []
```

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
```

7.2. inner class(内嵌类)

```
#!/usr/bin/env python3

import os, sys

class parent:
        def __init__(self):
```

```
                self.name = 'parent'

        def getName(self):
                print(self.name)

        class child:
                def __init__(self):
                        self.name = 'child'

                def getName(self):
                        print(self.name)
                def getParentName(self):
                        print(parent.name)

if __name__ == '__main__':
        child =  parent.child()
        child.getName()

        p = parent()
        p.getName()
        c =  p.child()
        c.getName()
        c.getParentName()
```

内嵌 Class 继承

```
class Geometry:
    class Curve:
        def __init__(self,c=1):
            self.c = c                          #curvature
parameter
            print ('Curvature %g'%self.c)
            pass                                #some codes

    class Line(Curve):
        def __init__(self):
            Geometry.Curve.__init__(self,0)     #the key
point
            pass                                #some codes
```

```
g = Geometry()
C = g.Curve(0.5)
L = g.Line()
```

8. 正则处理字符串

8.1. 正则替换

```
import re

string = "TMP AAAAAA\r\nMSG BBBB"

print(re.sub(r'(TMP|MSG)\s', "", string))
```

8.2. match

```
#!/usr/bin/python
import re

string = "/aaa/bbb/ccc"
match = re.match(r"/aaa/bbb/(.*)", string)
print(match)
print(match.span())
print(match.string)
print(match.group())
print(match.group(1))
print(match.groups())
print(match.groupdict())
```

输出

```
<re.Match object; span=(0, 12), match='/aaa/bbb/ccc'>
(0, 12)
```

```
/aaa/bbb/ccc
/aaa/bbb/ccc
ccc
('ccc',)
{}
```

8.3. 正则查找

```
#!/usr/bin/python
import re

regular = "打开|关闭|结束|暂停"
question = "打开电灯，然后关闭"
pattern = re.compile(regular)
result = pattern.findall(question)
print(result)
```

```
#!/usr/bin/python
import re

regular = "(打开|播放)圣经第(.+)卷第(.*)章"
question = "打开圣经第三卷第二章"
pattern = re.compile(regular)
result = pattern.findall(question)
print(result)
```

8.4. 正则匹配后返回字典

```
#!/usr/bin/python
```

```
import re

m = re.match(r"(?P<firstname>\w+) (?P<lastname>\w+)", "Neo
Chen")
print(m.group("firstname"), m.group("lastname"))
```

```
>>> match = re.search(r'(foo)?(bar)?', 'foo')
>>> match and match.groups()[1]
None
>>> match = re.search(r'(foo)?(bar)?', 'foobar')
>>> match and match.groups()[1]
'bar'

match = re.search(r'(?P<foo>foo)?(?P<bar>bar)?', 'foo')
match and match.groupdict().get('bar')

match = re.search(r'(?P<foo>foo)?(?P<bar>bar)?', 'foobar')
match and match.groupdict().get('bar')
```

9. 循环

```
for alt in range(100, 700, 10):
print(alt)

for alt in range(1000, 500, -10):
print(alt)
```

10. Input/Output

10.1. Standard

Standard Input

```
sys.stdin.readline().strip()
```

Standard Output

```
sys.stdout.write("输出的字串")
```

10.2. File

```
f = open('/tmp/workfile', 'r+')
f.write('0123456789abcdef')
f.seek(5)     # Go to the 6th byte in the file
f.read(1)
f.seek(-3, 2) # Go to the 3rd byte before the end
f.read(1)
f.readline()
f.close()
```

```
for line in open("myfile.txt"):
    print line

with open("myfile.txt") as f:
    for line in f:
```

```
        print line
```

```
with open('beak', 'rb+') as f:
    content = f.read()
    f.seek(0)
    f.write(content.replace(b'\r', b''))
    f.truncate()
```

11. Pipe

11.1. stdin

```
#!/usr/bin/python
#filename:demo.py

import sys

for line in sys.stdin:
    print line,


#ls -lh | python demo.py
```

11.2. forkpty

```
#!/usr/bin/env python

import os
import sys

pid, fd = os.forkpty()

if pid == 0:
    # child
    os.execlp("ssh","ssh","hostname","uname")
else:
    # parent
    print os.read(fd, 1000)
    os.write(fd,"password\n")

    c = os.read(fd, 1)
    while c:
        c = os.read(fd, 1)
        sys.stdout.write(c)
```

11.3. Popen

```
#!/usr/bin/python

from  subprocess import *
p = Popen(["cat", "-n"], bufsize=1024,stdin=PIPE,
                    stdout=PIPE, close_fds=True)

(fin, fout) =  (p.stdin, p.stdout)
for i in range(10):
   fin.write("line" + str(i))
   fin.write('\n')
   fin.flush()
   print fout.readline(),
```

11.4. socketpair

socketpair 实现双向管道

第 2 章 Library

1. 序列化

```
import shelve

with shelve.open("/var/tmp/test") as db:
        db["hello"] = "helloworld"

with shelve.open("/var/tmp/test") as db:
        for k, v in db.items():
                print(k, ":", v)
```

```
import shelve, json

data = json.loads(json.dumps({"800": "AAAA", "900": "BBBB",
"1000": "CCCC"}))
print(data, type(data))
with shelve.open("/var/tmp/test", "n") as db:
        for k, v in data.items():
                db[k] = v

with shelve.open("/var/tmp/test") as db:
        for k, v in db.items():
                print(k, ":", v)
```

2. 队列

```
import queue

q = queue.SimpleQueue()

for n in range(10):
        q.put(n)

print("队列数量:", q.qsize())

while not q.empty():
        print(q.get())

print("队列状态: ", q.empty())
```

先进先出队列，保持队列最大尺寸，超过尺寸自动出队

```
from queue import Queue

q = Queue(maxsize=3)

for i in range(10):
        if q.full():
                q.get()
        q.put(i, block=False)


print(type(q.queue), q.queue)

while not q.empty():
        i = q.get(block=False)
        print(i)
```

```
from queue import Queue

q = Queue(maxsize=10)

for i in range(20):
        if q.full():
                q.get()
        q.put(i, block=False)


print(type(q.queue), q.queue)

for n in range(5):
        if not q.empty():
                i = q.get(block=False)
                print(i)
```

3. 随机数

3.1. 随机选择列表

随机返回参数列表中任意一个元素

```
>>> print(random.choice(['China','Japan','Korea']))
Korea
>>> print(random.choice(['China','Japan','Korea']))
Japan
```

随机返回参数列表中任意两个元素，参数二指定返回的数量

```
print(random.sample(['China','Japan','Korea'],2))

>>> print(random.sample(['China','Japan','Korea'],2))
['Japan', 'Korea']
```

3.2. 指定随机数范围

random.randrange ([start,] stop [,step])

```
import random
random.randrange ([start,] stop [,step])
参数
start -- 指定范围内的开始值，包含在范围内。
stop -- 指定范围内的结束值，不包含在范围内。
step -- 指定递增基数。
```

```
#!/usr/bin/python3
import random

# 输出 1 <= number < 100 间的偶数
print "randrange(1, 100) : ", random.randrange(1, 100)

# 输出 100 <= number < 1000 间的偶数
print "randrange(100, 1000, 5) : ", random.randrange(100,
1000, 5)

# 输出 100 <= number < 1000 间的其他数
print "randrange(100, 1000, 3) : ", random.randrange(100,
1000, 3)
```

3.3. 指定随机数范围(整数)

```
>>> print(random.randint(1,5))
1
>>> print(random.randint(1,5))
3
>>> print(random.randint(1,5))
5
```

3.4. 指定随机数范围(小数)

```
>>> import random
>>> print(random.uniform(0,9))
7.36185883349121
```

```
>>> print(random.uniform(0.1,0.9))
0.3972467892993786
```

保留两位小数

```
import random

number = random.uniform(-0.0, -10)
print(round(number, 2))
```

字符串格式化方法

```
import random

randnum = random.uniform(0, 100)

print(randnum)
print(f'方法1：{randnum:.2f}')
print('方法2：{:.2f}'.format(randnum))
print('方法3：%.2f' % randnum)
print('方法4：' + str(round(randnum, 2)))
```

输出结果

```
0.15161848483329354
方法1：0.15
方法2：0.15
方法3：0.15
方法4：0.15
```

3.5. 打乱列表顺序

```
lists = list(range(10))
print(lists)
random.shuffle(lists)
print(lists)
```

演示

```
>>> lists = list(range(10))
>>> print(lists)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> random.shuffle(lists)
>>> print(lists)
[8, 1, 7, 0, 5, 3, 2, 9, 6, 4]
>>>
```

4. Python 多线程

4.1. 创建线程

```
from threading import Thread
import time


def fun1():
    print("fun1 begin")
    time.sleep(2)
    print("fun1 end")


def fun2():
    print("fun2 begin")
    time.sleep(6)
    print("fun2 end")


threads = []
threads.append(Thread(target=fun1))
threads.append(Thread(target=fun2))
print(threads)

if __name__ == "__main__":
    for t in threads:
        print(t)
        t.start()
    print("Done")
```

4.2. threading 高级线程接口

threading — Higher-level threading interface

```
import threading
class MyThread(threading.Thread):
    def __init__(self, name=None):
        threading.Thread.__init__(self)
        self.name = name

    def run(self):
        print self.name

def test():
    for i in range(0, 100):
        t = MyThread("thread_" + str(i))
        t.start()

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

4.3. Lock 线程锁

这里实现了一个计数器 count 这个全局变量会被多个线程同时操作，使其能够被顺序相加，需要靠线程锁的帮助。

```
#-*- encoding: utf-8 -*-
import threading
import time

class Test(threading.Thread):
    def __init__(self, num):
        threading.Thread.__init__(self)
        self._run_num = num

    def run(self):
        global count, mutex
        threadname = threading.currentThread().getName()

        for x in range(int(self._run_num)):
            mutex.acquire()
            count = count + 1
```

```
            mutex.release()
            print (threadname, x, count)
            time.sleep(1)

if __name__ == '__main__':
    global count, mutex
    threads = []
    num = 5
    count = 0
    # 创建锁
    mutex = threading.Lock()
    # 创建线程对象
    for x in range(num):
        threads.append(Test(10))
    # 启动线程
    for t in threads:
        t.start()
    # 等待子线程结束
    for t in threads:
        t.join()
```

4.4. Queue 队列

ref: http://www.ibm.com/developerworks/aix/library/au-threadingpython/

```
 #!/usr/bin/env python
import Queue
import threading
import urllib2
import time

hosts = ["http://yahoo.com", "http://google.com",
"http://amazon.com",
"http://ibm.com", "http://apple.com"]

queue = Queue.Queue()
```

```
class ThreadUrl(threading.Thread):
        """Threaded Url Grab"""
        def __init__(self, queue):
          threading.Thread.__init__(self)
          self.queue = queue

        def run(self):
          while True:
                #grabs host from queue
                host = self.queue.get()

                #grabs urls of hosts and prints first 1024
bytes of page
                url = urllib2.urlopen(host)
                print url.read(1024)

                #signals to queue job is done
                self.queue.task_done()

start = time.time()
def main():

#spawn a pool of threads, and pass them queue instance
for i in range(5):
  t = ThreadUrl(queue)
  t.setDaemon(True)
  t.start()

#populate queue with data
  for host in hosts:
    queue.put(host)

#wait on the queue until everything has been processed
queue.join()

main()
print "Elapsed Time: %s" % (time.time() - start)
```

5. syslog

```
import syslog

syslog.syslog('Processing started')
if error:
    syslog.syslog(syslog.LOG_ERR, 'Processing started')

syslog.openlog(logopt=syslog.LOG_PID, facility=syslog.LOG_MAIL)
syslog.syslog('E-mail processing initiated...')
```

5.1. udp client

```
#!/usr/bin/python
# -*- encoding: iso-8859-1 -*-

"""
Python syslog client.

This code is placed in the public domain by the author.
Written by Christian Stigen Larsen.

This is especially neat for Windows users, who (I think) don't
get any syslog module in the default python installation.

See RFC3164 for more info -- http://tools.ietf.org/html/rfc3164

Note that if you intend to send messages to remote servers,
their
syslogd must be started with -r to allow to receive UDP from
the network.
"""

import socket

# I'm a python novice, so I don't know of better ways to define
enums
```

```
FACILITY = {
        'kern': 0, 'user': 1, 'mail': 2, 'daemon': 3,
        'auth': 4, 'syslog': 5, 'lpr': 6, 'news': 7,
        'uucp': 8, 'cron': 9, 'authpriv': 10, 'ftp': 11,
        'local0': 16, 'local1': 17, 'local2': 18, 'local3': 19,
        'local4': 20, 'local5': 21, 'local6': 22, 'local7': 23,
}

LEVEL = {
        'emerg': 0, 'alert':1, 'crit': 2, 'err': 3,
        'warning': 4, 'notice': 5, 'info': 6, 'debug': 7
}

def syslog(message, level=LEVEL['notice'],
facility=FACILITY['daemon'],
        host='localhost', port=514):

        """
        Send syslog UDP packet to given host and port.
        """

        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        data = '<%d>%s' % (level + facility*8, message)
        sock.sendto(data, (host, port))
        sock.close()
```

Example usage:

```
from syslog import syslog
message = 'There were zwei peanuts walking down der strasse...'
syslog(message, level=5, facility=3, host='localhost',
port=514)
```

```
# -*- Mode: Python; tab-width: 4 -*-

#
==================================================================
```

```
=======
# Copyright 1997 by Sam Rushing
#
#                         All Rights Reserved
#
# Permission to use, copy, modify, and distribute this software
and
# its documentation for any purpose and without fee is hereby
# granted, provided that the above copyright notice appear in
all
# copies and that both that copyright notice and this
permission
# notice appear in supporting documentation, and that the name
of Sam
# Rushing not be used in advertising or publicity pertaining to
# distribution of the software without specific, written prior
# permission.
#
# SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE,
# INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN
# NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL,
INDIRECT OR
# CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
FROM LOSS
# OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
# NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
# CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
#
=================================================================
=======

"""socket interface to unix syslog.
On Unix, there are usually two ways of getting to syslog: via a
local unix-domain socket, or via the TCP service.

Usually "/dev/log" is the unix domain socket.  This may be
different
for other systems.

>>> my_client = syslog_client ('/dev/log')

Otherwise, just use the UDP version, port 514.
```

```
>>> my_client = syslog_client (('my_log_host', 514))

On win32, you will have to use the UDP version.  Note that
you can use this to log to other hosts (and indeed, multiple
hosts).

This module is not a drop-in replacement for the python
<syslog> extension module - the interface is different.

Usage:

>>> c = syslog_client()
>>> c = syslog_client ('/strange/non_standard_log_location')
>>> c = syslog_client (('other_host.com', 514))
>>> c.log ('testing', facility='local0', priority='debug')

"""

# TODO: support named-pipe syslog.
# [see ftp://sunsite.unc.edu/pub/Linux/system/Daemons/syslog-
fifo.tar.z]

# from <linux/sys/syslog.h>:
#
==================================================================
============
# priorities/facilities are encoded into a single 32-bit
quantity, where the
# bottom 3 bits are the priority (0-7) and the top 28 bits are
the facility
# (0-big number).  Both the priorities and the facilities map
roughly
# one-to-one to strings in the syslogd(8) source code.  This
mapping is
# included in this file.
#
# priorities (these are ordered)

LOG_EMERG               = 0             #  system is unusable
LOG_ALERT               = 1             #  action must be taken
immediately
LOG_CRIT                = 2             #  critical conditions
LOG_ERR                 = 3             #  error conditions
LOG_WARNING             = 4             #  warning conditions
LOG_NOTICE              = 5             #  normal but
```

```
significant condition
LOG_INFO                = 6             #  informational
LOG_DEBUG               = 7             #  debug-level messages

#  facility codes
LOG_KERN                = 0             #  kernel messages
LOG_USER                = 1             #  random user-level
messages
LOG_MAIL                = 2             #  mail system
LOG_DAEMON              = 3             #  system daemons
LOG_AUTH                = 4             #
security/authorization messages
LOG_SYSLOG              = 5             #  messages generated
internally by syslogd
LOG_LPR                 = 6             #  line printer
subsystem
LOG_NEWS                = 7             #  network news
subsystem
LOG_UUCP                = 8             #  UUCP subsystem
LOG_CRON                = 9             #  clock daemon
LOG_AUTHPRIV    = 10    #  security/authorization messages
(private)

#  other codes through 15 reserved for system use
LOG_LOCAL0              = 16            #  reserved for local
use
LOG_LOCAL1              = 17            #  reserved for local
use
LOG_LOCAL2              = 18            #  reserved for local
use
LOG_LOCAL3              = 19            #  reserved for local
use
LOG_LOCAL4              = 20            #  reserved for local
use
LOG_LOCAL5              = 21            #  reserved for local
use
LOG_LOCAL6              = 22            #  reserved for local
use
LOG_LOCAL7              = 23            #  reserved for local
use

priority_names = {
        "alert":        LOG_ALERT,
        "crit":         LOG_CRIT,
        "debug":        LOG_DEBUG,
```

```
        "emerg":        LOG_EMERG,
        "err":          LOG_ERR,
        "error":        LOG_ERR,                #  DEPRECATED
        "info":         LOG_INFO,
        "notice":       LOG_NOTICE,
        "panic":        LOG_EMERG,              #  DEPRECATED
        "warn":         LOG_WARNING,            #  DEPRECATED
        "warning":      LOG_WARNING,
        }

facility_names = {
        "auth":         LOG_AUTH,
        "authpriv":     LOG_AUTHPRIV,
        "cron":         LOG_CRON,
        "daemon":       LOG_DAEMON,
        "kern":         LOG_KERN,
        "lpr":          LOG_LPR,
        "mail":         LOG_MAIL,
        "news":         LOG_NEWS,
        "security":     LOG_AUTH,               #  DEPRECATED
        "syslog":       LOG_SYSLOG,
        "user":         LOG_USER,
        "uucp":         LOG_UUCP,
        "local0":       LOG_LOCAL0,
        "local1":       LOG_LOCAL1,
        "local2":       LOG_LOCAL2,
        "local3":       LOG_LOCAL3,
        "local4":       LOG_LOCAL4,
        "local5":       LOG_LOCAL5,
        "local6":       LOG_LOCAL6,
        "local7":       LOG_LOCAL7,
        }

import socket

class syslog_client:
        def __init__ (self, address='/dev/log'):
                self.address = address
                if type (address) == type(''):
                        self.socket = socket.socket
(socket.AF_UNIX, socket.SOCK_STREAM)
                        self.socket.connect (address)
                        self.unix = 1
                else:
                        self.socket = socket.socket
```

```
(socket.AF_INET, socket.SOCK_DGRAM)
                        self.unix = 0

        # curious: when talking to the unix-domain '/dev/log'
socket, a
        #   zero-terminator seems to be required.  this string
is placed
        #   into a class variable so that it can be overridden
if
        #   necessary.

        log_format_string = '<%d>%s\000'

        def log (self, message, facility=LOG_USER,
priority=LOG_INFO):
                message = self.log_format_string % (
                        self.encode_priority (facility,
priority),
                        message
                        )
                if self.unix:
                        self.socket.send (message)
                else:
                        self.socket.sendto (message,
self.address)

        def encode_priority (self, facility, priority):
                if type(facility) == type(''):
                        facility = facility_names[facility]
                if type(priority) == type(''):
                        priority = priority_names[priority]
                return (facility<<3) | priority

        def close (self):
                if self.unix:
                        self.socket.close()
```

5.2. udp server

```
import os,socket,sys,time,string
```

```
import MySQLdb
bufsize=1500
port=514
syslog_serverty={ 0:"emergency",
                  1:"alert",
                  2:"critical",
                  3:"error",
                  4:"warning",
                  5:"notice",
                  6:"info",
                  7:"debug"
                }
syslog_facility={ 0:"kernel",
                  1:"user",
                  2:"mail",
                  3:"daemaon",
                  4:"auth",
                  5:"syslog",
                  6:"lpr",
                  7:"news",
                  8:"uucp",
                  9:"cron",
                  10:"authpriv",
                  11:"ftp",
                  12:"ntp",
                  13:"security",
                  14:"console",
                  15:"cron",
                  16:"local 0",
                  17:"local 1",
                  18:"local 2",
                  19:"local 3",
                  20:"local 4",
                  21:"local 5",
                  22:"local 6",
                  23:"local 7"
                }
try:
  sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
  sock.bind(("0.0.0.0",port))
except:
  print("error bind")
  sys.exit(1)
sql_em="insert into emergency values(%s,%s,%s,%s,%s,%s)"
sql_al="insert into alert     values(%s,%s,%s,%s,%s,%s)"
```

```
sql_cr="insert into critical  values(%s,%s,%s,%s,%s,%s)"
sql_er="insert into error     values(%s,%s,%s,%s,%s,%s)"
sql_wa="insert into warning   values(%s,%s,%s,%s,%s,%s)"
conn=MySQLdb.connect(host="127.0.0.1",db="syslog",port=18888,us
er="root",passwd="cinda")
curs=conn.cursor()
#f=file("syslog.txt","w")
print ("----------------syslog is start----------------\n")
try:
  while 1:
    try:
      data,addr=sock.recvfrom(bufsize)
      #print data,addr
      syslog=str(data)
      n=syslog.find('>')
      serverty=string.atoi(syslog[1:n])&0x0007
      facility=(string.atoi(syslog[1:n])&0x03f8)>>3
      syslog_msg=syslog[26:]
      dev_name=syslog_msg[:syslog_msg.find(' ')]
      dev_msg=syslog_msg[syslog_msg.find(' '):]
      param=(time.strftime("%Y-%m-%d
%H:%M:%S",time.localtime()),dev_name,addr[0],syslog_facility[fa
cility],syslog_serverty[serverty],dev_msg)

      if serverty==0:
        curs.execute(sql_em,param)
        print syslog_msg
      elif serverty==1:
        curs.execute(sql_al,param)
        print syslog_msg
      elif serverty==2:
        curs.execute(sql_cr,param)
        print syslog_msg
      elif serverty==3:
        curs.execute(sql_er,param)
        print syslog_msg
      elif serverty==4:
        curs.execute(sql_wa,param)
        print syslog_msg

      conn.commit()

      #print dev_msg,time.strftime("%Y-%m-%d
%H:%M:%S",time.localtime())
      #print
```

```
syslog_serverty[serverty],syslog_facility[facility],syslog[26:]
      #f.writelines(syslog_serverty[serverty]+"
"+syslog_facility[facility]+" "+syslog[26:]+'\n')
    except socket.error:
      pass
except KeyboardInterrupt:
  curs.close()
  conn.close()
  print ("------------------syslogd stop-------------\n")
  print "good bye"
  sys.exit()
#f.close
```

6. Socket

6.1. UDP

UDP Server

```
import asyncore, socket

class AsyncoreServerUDP(asyncore.dispatcher):
   def __init__(self):
      asyncore.dispatcher.__init__(self)

      # Bind to port 5005 on all interfaces
      self.create_socket(socket.AF_INET, socket.SOCK_DGRAM)
      self.bind(('', 5005))

   # Even though UDP is connectionless this is called when it
binds to a port
   def handle_connect(self):
      print "Server Started..."

   # This is called everytime there is something to read
   def handle_read(self):
      data, addr = self.recvfrom(2048)
      print str(addr)+" >> "+data

   # This is called all the time and causes errors if you
leave it out.
   def handle_write(self):
      pass

AsyncoreServerUDP()
asyncore.loop()
```

UDP Clinet

```
import socket, asyncore

class AsyncoreClientUDP(asyncore.dispatcher):

   def __init__(self, server, port):
      asyncore.dispatcher.__init__(self)
      self.server = server
      self.port = port
      self.buffer = ""

      # Network Connection Magic!
      asyncore.dispatcher.__init__(self)
      self.create_socket(socket.AF_INET, socket.SOCK_DGRAM)
      self.bind( ('', 0) ) # bind to all interfaces and a
"random" free port.
      print "Connecting..."

   # Once a "connection" is made do this stuff.
   def handle_connect(self):
      print "Connected"

   # If a "connection" is closed do this stuff.
   def handle_close(self):
      self.close()

   # If a message has arrived, process it.
   def handle_read(self):
      data, addr = self.recv(2048)
      print data

   # Actually sends the message if there was something in the
buffer.
   def handle_write(self):
      if self.buffer != "":
         print self.buffer
         sent = self.sendto(self.buffer, (self.server,
self.port))
         self.buffer = self.buffer[sent:]

connection = AsyncoreClientUDP("127.0.0.1",5005) # create the
"connection"
while 1:
```

```
   asyncore.loop(count = 10) # Check for upto 10 packets this
call?
   connection.buffer += raw_input(" Chat > ") # raw_input
(this is a blocking call)
```

7. subprocess

7.1. check_output

```
sp = subprocess.check_output(cmd)
text = sp.decode('UTF8')
print(text)
```

获取IP地址

```
import subprocess

command = "/usr/bin/ip addr show eth0 | grep 'inet ' | awk
'{print $2}' | cut -d/ -f1"
screen = subprocess.check_output(command, shell=True)
print(screen.decode().replace("\n", ""))
```

制定运行目录

```
#!/usr/bin/python
# -*-coding: utf-8-*-
import subprocess

output = subprocess.check_output("ls", cwd="/")
print(output.decode())

output = subprocess.check_output("/usr/bin/git pull",
cwd="/opt/netkiller", shell=True)
print(output.decode())
```

8. YAML

8.1. 严格按段落展示 ｜、｜+、｜-

```
|: 文中自动换行 + 文末新增一空行
|+: 文中自动换行 + 文末新增两空行
|-: 文中自动换行 + 文末不新增行
```

8.2. >、>+、>-

```
>: 文中不自动换行 + 文末新增一空行
>+: 文中不自动换行 + 文末新增两空行
>-: 文中不自动换行 + 文末不新增行
```

8.3. PyYAML

解决 ｜ 问题

```
import sys
import yaml

x = u"""\
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA6oySC+8/N9VNpk0gJS7Gk8vn9sYN7FhjpAQnoHRqTN/Oaiy
x
xk2AleP2vXpojA/DHldT1JO+o3j56AHD+yfNFFeYvgWKDY35g49HsZZhbyCEAB4
5
...
"""
```

```
yaml.SafeDumper.org_represent_str =
yaml.SafeDumper.represent_str

def repr_str(dumper, data):
    if '\n' in data:
        return
dumper.represent_scalar(u'tag:yaml.org,2002:str', data,
style='|')
    return dumper.org_represent_str(data)

yaml.add_representer(str, repr_str, Dumper=yaml.SafeDumper)

yaml.safe_dump(dict(a=1, b='hello world', c=x), sys.stdout)

print('-'*50)

class PSS(str):
    pass

x = PSS("""\
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA6oySC+8/N9VNpk0gJS7Gk8vn9sYN7FhjpAQnoHRqTN/Oaiy
x
xk2AleP2vXpojA/DHldT1JO+o3j56AHD+yfNFFeYvgWKDY35g49HsZZhbyCEAB4
5
...
""")

def pss_representer(dumper, data):
        style = '|'
        # if sys.versioninfo < (3,) and not isinstance(data,
unicode):
        #     data = unicode(data, 'ascii')
        tag = u'tag:yaml.org,2002:str'
        return dumper.represent_scalar(tag, data, style=style)

yaml.add_representer(PSS, pss_representer,
Dumper=yaml.SafeDumper)

yaml.safe_dump(dict(a=1, b='hello world', c=x), sys.stdout)
```

8.4. ruamel.yaml

```
pip install ruamel.yaml
```

解决 | 问题

```
from ruamel.yaml import YAML
from ruamel.yaml.scalarstring import PreservedScalarString as
pss

x = pss("""\
external_url 'https://gitlab.example.com'
gitlab_rails['time_zone'] = 'Asia/Shanghai'
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "smtp.aliyun.com"
gitlab_rails['smtp_port'] = 465
gitlab_rails['smtp_user_name'] = "netkiller@msn.com"
gitlab_rails['smtp_password'] = "******"
gitlab_rails['smtp_domain'] = "aliyun.com"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
gitlab_rails['smtp_tls'] = true
gitlab_rails['gitlab_email_from'] = 'netkiller@msn.com'
gitlab_rails['gitlab_shell_ssh_port'] = 22
""")

yaml = YAML()

yaml.dump(dict(a=1, b='hello world', c=x), sys.stdout)
```

输出结果

```
a: 1
b: hello world
c: |
  external_url 'https://gitlab.example.com'
  gitlab_rails['time_zone'] = 'Asia/Shanghai'
  gitlab_rails['smtp_enable'] = true
  gitlab_rails['smtp_address'] = "smtp.aliyun.com"
  gitlab_rails['smtp_port'] = 465
  gitlab_rails['smtp_user_name'] = "netkiller@msn.com"
  gitlab_rails['smtp_password'] = "******"
  gitlab_rails['smtp_domain'] = "aliyun.com"
  gitlab_rails['smtp_authentication'] = "login"
  gitlab_rails['smtp_enable_starttls_auto'] = true
  gitlab_rails['smtp_tls'] = true
  gitlab_rails['gitlab_email_from'] = 'netkiller@msn.com'
  gitlab_rails['gitlab_shell_ssh_port'] = 22
```

LiteralScalarString, PreservedScalarString

```
from ruamel.yaml.scalarstring import LiteralScalarString,
PreservedScalarString
from ruamel.yaml import YAML
import sys

yaml = YAML()
data = {}
data['data'] = PreservedScalarString("""\
0.0, 1.0
0.1, 1.5
0.2, 1.7
""")

data['data_points'] = LiteralScalarString("""\
0.0, 1.0
0.1, 1.5
```

```
0.2, 1.7""")

yaml.dump(data, sys.stdout)

print("=" * 50)

yaml_str = """\
any_value: 123.4
data_points: |2-
  a
  b
"""

data = yaml.load(yaml_str)
yaml.dump(data, sys.stdout)
```

9. Daemon

http://www.myelin.co.nz/post/2003/3/13/#200303135

```
#!/usr/bin/env python

import os, sys

print "I'm going to fork now - the child will write something
to a pipe, and the parent will read it back"

r, w = os.pipe()            # r,w是文件描述符, 不是文件对象

pid = os.fork()
if pid:
    # 父进程
    os.close(w)            # 关闭一个文件描述符
    r = os.fdopen(r)       # 将r转化为文件对象
    print "parent: reading"
    txt = r.read()
    os.waitpid(pid, 0)   # 确保子进程被撤销
else:
    # 子进程
    os.close(r)
    w = os.fdopen(w, 'w')
    print "child: writing"
    w.write("here's some text from the child")
    w.close()
    print "child: closing"
    sys.exit(0)

print "parent: got it; text =", txt
```

```
import sys, os
```

```
if __name__ == "__main__":
    # do the UNIX double-fork magic, see Stevens' "Advanced
    # Programming in the UNIX Environment" for details (ISBN
0201563177)
    try:
        pid = os.fork()
        if pid > 0:
            # exit first parent
            sys.exit(0)
    except OSError, e:
        print >>sys.stderr, "fork #1 failed: %d (%s)" %
(e.errno, e.strerror)
        sys.exit(1)

    # decouple from parent environment
    os.chdir("/")
    os.setsid()
    os.umask(0)

    # do second fork
    try:
        pid = os.fork()
        if pid > 0:
            # exit from second parent, print eventual PID
before
            print "Daemon PID %d" % pid
            sys.exit(0)
    except OSError, e:
        print >>sys.stderr, "fork #2 failed: %d (%s)" %
(e.errno, e.strerror)
        sys.exit(1)

    # start the daemon main loop
```

```
# Redirect standard file descriptors
sys.stdin = open('/dev/null', 'r')
sys.stdout = open('/dev/null', 'w')
sys.stderr = open('/dev/null', 'w')
```

10. python-memcached

参考Python安装

```
wget ftp://ftp.tummy.com/pub/python-memcached/python-memcached-
1.34.tar.gz
tar zxvf python-memcached-1.34.tar.gz
cd python-memcached-1.34
```

python setup.py install

```
running install
running build
running build_py
creating build
creating build/lib
copying memcache.py -> build/lib
running install_lib
copying build/lib/memcache.py -> /usr/lib/python2.3/site-
packages
byte-compiling /usr/lib/python2.3/site-packages/memcache.py to
memcache.pyc
```

例 2.1. memcached.py

```
import memcache
mc = memcache.Client(['127.0.0.1:11211'], debug=1)

mc.set("some_key", "Some value")
value = mc.get("some_key")
print value
```

```
mc.set("another_key", 3)
mc.delete("another_key")

mc.set("key", "1")   # note that the key used for incr/decr
must be a string.
mc.incr("key")
mc.decr("key")
```

11. Pyro - Pyro is short for PYthon Remote Objects

http://pyro.sourceforge.net/

12. Python Imaging Library

Debian/Ubuntu

```
sudo apt-get install libjpeg62-dev
sudo apt-get install python-imaging
```

采用源码安装

```
tar zxvf Imaging-1.1.6.tar.gz
cd Imaging-1.1.6/
```

sudo python setup.py install

decoder jpeg not available

首先确认jpeg库是否安装

find / -name jpeglib.h

然后修改头文件

Imaging-1.1.6/libImaging

修改Jpeg.h, #include "jpeglib.h" 改为

#include "/usr/include/jpeglib.h"

13. getopt – Command line option parsing

14. syslog

```
import syslog

syslog.syslog('Processing started')
if error:
    syslog.syslog(syslog.LOG_ERR, 'Processing started')

syslog.openlog(logopt=syslog.LOG_PID, facility=syslog.LOG_MAIL)
syslog.syslog('E-mail processing initiated...')
```

14.1. udp client

```
#!/usr/bin/python
# -*- encoding: iso-8859-1 -*-

"""
Python syslog client.

This code is placed in the public domain by the author.
Written by Christian Stigen Larsen.

This is especially neat for Windows users, who (I think) don't
get any syslog module in the default python installation.

See RFC3164 for more info -- http://tools.ietf.org/html/rfc3164

Note that if you intend to send messages to remote servers,
their
syslogd must be started with -r to allow to receive UDP from
the network.
"""

import socket

# I'm a python novice, so I don't know of better ways to define
enums
```

```
FACILITY = {
        'kern': 0, 'user': 1, 'mail': 2, 'daemon': 3,
        'auth': 4, 'syslog': 5, 'lpr': 6, 'news': 7,
        'uucp': 8, 'cron': 9, 'authpriv': 10, 'ftp': 11,
        'local0': 16, 'local1': 17, 'local2': 18, 'local3': 19,
        'local4': 20, 'local5': 21, 'local6': 22, 'local7': 23,
}

LEVEL = {
        'emerg': 0, 'alert':1, 'crit': 2, 'err': 3,
        'warning': 4, 'notice': 5, 'info': 6, 'debug': 7
}

def syslog(message, level=LEVEL['notice'],
facility=FACILITY['daemon'],
        host='localhost', port=514):

        """
        Send syslog UDP packet to given host and port.
        """

        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        data = '<%d>%s' % (level + facility*8, message)
        sock.sendto(data, (host, port))
        sock.close()
```

Example usage:

```
from syslog import syslog
message = 'There were zwei peanuts walking down der strasse...'
syslog(message, level=5, facility=3, host='localhost',
port=514)
```

```
# -*- Mode: Python; tab-width: 4 -*-

#
==================================================================
```

```
=======
# Copyright 1997 by Sam Rushing
#
#                         All Rights Reserved
#
# Permission to use, copy, modify, and distribute this software
and
# its documentation for any purpose and without fee is hereby
# granted, provided that the above copyright notice appear in
all
# copies and that both that copyright notice and this
permission
# notice appear in supporting documentation, and that the name
of Sam
# Rushing not be used in advertising or publicity pertaining to
# distribution of the software without specific, written prior
# permission.
#
# SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE,
# INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN
# NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL,
INDIRECT OR
# CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
FROM LOSS
# OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
# NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
# CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
#
=================================================================
=======

"""socket interface to unix syslog.
On Unix, there are usually two ways of getting to syslog: via a
local unix-domain socket, or via the TCP service.

Usually "/dev/log" is the unix domain socket.  This may be
different
for other systems.

>>> my_client = syslog_client ('/dev/log')

Otherwise, just use the UDP version, port 514.
```

```
>>> my_client = syslog_client (('my_log_host', 514))

On win32, you will have to use the UDP version.  Note that
you can use this to log to other hosts (and indeed, multiple
hosts).

This module is not a drop-in replacement for the python
<syslog> extension module - the interface is different.

Usage:

>>> c = syslog_client()
>>> c = syslog_client ('/strange/non_standard_log_location')
>>> c = syslog_client (('other_host.com', 514))
>>> c.log ('testing', facility='local0', priority='debug')

"""

# TODO: support named-pipe syslog.
# [see ftp://sunsite.unc.edu/pub/Linux/system/Daemons/syslog-
fifo.tar.z]

# from <linux/sys/syslog.h>:
#
==================================================================
============
# priorities/facilities are encoded into a single 32-bit
quantity, where the
# bottom 3 bits are the priority (0-7) and the top 28 bits are
the facility
# (0-big number).  Both the priorities and the facilities map
roughly
# one-to-one to strings in the syslogd(8) source code.  This
mapping is
# included in this file.
#
# priorities (these are ordered)

LOG_EMERG               = 0             #  system is unusable
LOG_ALERT               = 1             #  action must be taken
immediately
LOG_CRIT                = 2             #  critical conditions
LOG_ERR                 = 3             #  error conditions
LOG_WARNING             = 4             #  warning conditions
LOG_NOTICE              = 5             #  normal but
```

```
significant condition
LOG_INFO                = 6             #  informational
LOG_DEBUG               = 7             #  debug-level messages

#  facility codes
LOG_KERN                = 0             #  kernel messages
LOG_USER                = 1             #  random user-level
messages
LOG_MAIL                = 2             #  mail system
LOG_DAEMON              = 3             #  system daemons
LOG_AUTH                = 4             #
security/authorization messages
LOG_SYSLOG              = 5             #  messages generated
internally by syslogd
LOG_LPR                 = 6             #  line printer
subsystem
LOG_NEWS                = 7             #  network news
subsystem
LOG_UUCP                = 8             #  UUCP subsystem
LOG_CRON                = 9             #  clock daemon
LOG_AUTHPRIV    = 10    #  security/authorization messages
(private)

#  other codes through 15 reserved for system use
LOG_LOCAL0              = 16            #  reserved for local
use
LOG_LOCAL1              = 17            #  reserved for local
use
LOG_LOCAL2              = 18            #  reserved for local
use
LOG_LOCAL3              = 19            #  reserved for local
use
LOG_LOCAL4              = 20            #  reserved for local
use
LOG_LOCAL5              = 21            #  reserved for local
use
LOG_LOCAL6              = 22            #  reserved for local
use
LOG_LOCAL7              = 23            #  reserved for local
use

priority_names = {
        "alert":        LOG_ALERT,
        "crit":         LOG_CRIT,
        "debug":        LOG_DEBUG,
```

```
        "emerg":        LOG_EMERG,
        "err":          LOG_ERR,
        "error":        LOG_ERR,                #  DEPRECATED
        "info":         LOG_INFO,
        "notice":       LOG_NOTICE,
        "panic":        LOG_EMERG,              #  DEPRECATED
        "warn":         LOG_WARNING,            #  DEPRECATED
        "warning":      LOG_WARNING,
        }

facility_names = {
        "auth":         LOG_AUTH,
        "authpriv":     LOG_AUTHPRIV,
        "cron":         LOG_CRON,
        "daemon":       LOG_DAEMON,
        "kern":         LOG_KERN,
        "lpr":          LOG_LPR,
        "mail":         LOG_MAIL,
        "news":         LOG_NEWS,
        "security":     LOG_AUTH,               #  DEPRECATED
        "syslog":       LOG_SYSLOG,
        "user":         LOG_USER,
        "uucp":         LOG_UUCP,
        "local0":       LOG_LOCAL0,
        "local1":       LOG_LOCAL1,
        "local2":       LOG_LOCAL2,
        "local3":       LOG_LOCAL3,
        "local4":       LOG_LOCAL4,
        "local5":       LOG_LOCAL5,
        "local6":       LOG_LOCAL6,
        "local7":       LOG_LOCAL7,
        }

import socket

class syslog_client:
        def __init__ (self, address='/dev/log'):
                self.address = address
                if type (address) == type(''):
                        self.socket = socket.socket
(socket.AF_UNIX, socket.SOCK_STREAM)
                        self.socket.connect (address)
                        self.unix = 1
                else:
                        self.socket = socket.socket
```

```
(socket.AF_INET, socket.SOCK_DGRAM)
                        self.unix = 0

        # curious: when talking to the unix-domain '/dev/log'
socket, a
        #   zero-terminator seems to be required.  this string
is placed
        #   into a class variable so that it can be overridden
if
        #   necessary.

        log_format_string = '<%d>%s\000'

        def log (self, message, facility=LOG_USER,
priority=LOG_INFO):
                message = self.log_format_string % (
                        self.encode_priority (facility,
priority),
                        message
                        )
                if self.unix:
                        self.socket.send (message)
                else:
                        self.socket.sendto (message,
self.address)

        def encode_priority (self, facility, priority):
                if type(facility) == type(''):
                        facility = facility_names[facility]
                if type(priority) == type(''):
                        priority = priority_names[priority]
                return (facility<<3) | priority

        def close (self):
                if self.unix:
                        self.socket.close()
```

14.2. udp server

```
import os,socket,sys,time,string
```

```
import MySQLdb
bufsize=1500
port=514
syslog_serverty={ 0:"emergency",
                  1:"alert",
                  2:"critical",
                  3:"error",
                  4:"warning",
                  5:"notice",
                  6:"info",
                  7:"debug"
                }
syslog_facility={ 0:"kernel",
                  1:"user",
                  2:"mail",
                  3:"daemaon",
                  4:"auth",
                  5:"syslog",
                  6:"lpr",
                  7:"news",
                  8:"uucp",
                  9:"cron",
                  10:"authpriv",
                  11:"ftp",
                  12:"ntp",
                  13:"security",
                  14:"console",
                  15:"cron",
                  16:"local 0",
                  17:"local 1",
                  18:"local 2",
                  19:"local 3",
                  20:"local 4",
                  21:"local 5",
                  22:"local 6",
                  23:"local 7"
                }
try:
  sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
  sock.bind(("0.0.0.0",port))
except:
  print("error bind")
  sys.exit(1)
sql_em="insert into emergency values(%s,%s,%s,%s,%s,%s)"
sql_al="insert into alert     values(%s,%s,%s,%s,%s,%s)"
```

```
sql_cr="insert into critical  values(%s,%s,%s,%s,%s,%s)"
sql_er="insert into error     values(%s,%s,%s,%s,%s,%s)"
sql_wa="insert into warning   values(%s,%s,%s,%s,%s,%s)"
conn=MySQLdb.connect(host="127.0.0.1",db="syslog",port=18888,us
er="root",passwd="cinda")
curs=conn.cursor()
#f=file("syslog.txt","w")
print ("----------------syslog is start----------------\n")
try:
  while 1:
    try:
      data,addr=sock.recvfrom(bufsize)
      #print data,addr
      syslog=str(data)
      n=syslog.find('>')
      serverty=string.atoi(syslog[1:n])&0x0007
      facility=(string.atoi(syslog[1:n])&0x03f8)>>3
      syslog_msg=syslog[26:]
      dev_name=syslog_msg[:syslog_msg.find(' ')]
      dev_msg=syslog_msg[syslog_msg.find(' '):]
      param=(time.strftime("%Y-%m-%d
%H:%M:%S",time.localtime()),dev_name,addr[0],syslog_facility[fa
cility],syslog_serverty[serverty],dev_msg)

      if serverty==0:
        curs.execute(sql_em,param)
        print syslog_msg
      elif serverty==1:
        curs.execute(sql_al,param)
        print syslog_msg
      elif serverty==2:
        curs.execute(sql_cr,param)
        print syslog_msg
      elif serverty==3:
        curs.execute(sql_er,param)
        print syslog_msg
      elif serverty==4:
        curs.execute(sql_wa,param)
        print syslog_msg

      conn.commit()

      #print dev_msg,time.strftime("%Y-%m-%d
%H:%M:%S",time.localtime())
      #print
```

```
syslog_serverty[serverty],syslog_facility[facility],syslog[26:]
      #f.writelines(syslog_serverty[serverty]+"
"+syslog_facility[facility]+" "+syslog[26:]+'\n')
    except socket.error:
      pass
except KeyboardInterrupt:
  curs.close()
  conn.close()
  print ("------------------syslogd stop-------------\n")
  print "good bye"
  sys.exit()
#f.close
```

15. python-subversion

```
$ sudo apt-get install python-subversion
```

```
$ dpkg -L python-subversion
/.
/usr
/usr/share
/usr/share/python-support
/usr/share/python-support/python-subversion.public
/usr/share/doc
/usr/share/doc/python-subversion
/usr/share/doc/python-subversion/examples
/usr/share/doc/python-subversion/examples/svnlook.py.gz
/usr/share/doc/python-subversion/examples/dumpprops.py
/usr/share/doc/python-subversion/examples/blame.py
/usr/share/doc/python-subversion/examples/svnshell.py.gz
/usr/share/doc/python-subversion/examples/revplist.py
/usr/share/doc/python-subversion/examples/putfile.py
/usr/share/doc/python-subversion/examples/getfile.py
/usr/share/doc/python-subversion/examples/check-modified.py
/usr/share/doc/python-subversion/examples/get-location-
segments.py
/usr/share/doc/python-subversion/examples/geturl.py
/usr/share/doc/python-subversion/changelog.gz
/usr/share/doc/python-subversion/copyright
/usr/share/doc/python-subversion/changelog.Debian.gz
/usr/share/doc/python-subversion/README.Debian
/usr/share/lintian
/usr/share/lintian/overrides
/usr/share/lintian/overrides/python-subversion
/usr/share/pyshared
/usr/share/pyshared/libsvn
/usr/share/pyshared/libsvn/delta.py
/usr/share/pyshared/libsvn/client.py
/usr/share/pyshared/libsvn/core.py
```

```
/usr/share/pyshared/libsvn/ra.py
/usr/share/pyshared/libsvn/fs.py
/usr/share/pyshared/libsvn/wc.py
/usr/share/pyshared/libsvn/__init__.py
/usr/share/pyshared/libsvn/repos.py
/usr/share/pyshared/libsvn/diff.py
/usr/share/pyshared/svn
/usr/share/pyshared/svn/delta.py
/usr/share/pyshared/svn/client.py
/usr/share/pyshared/svn/core.py
/usr/share/pyshared/svn/ra.py
/usr/share/pyshared/svn/fs.py
/usr/share/pyshared/svn/wc.py
/usr/share/pyshared/svn/__init__.py
/usr/share/pyshared/svn/repos.py
/usr/share/pyshared/svn/diff.py
/usr/bin
/usr/bin/svnshell
/usr/lib
/usr/lib/libsvn_swig_py2.6-1.so.1.0.0
/usr/lib/pyshared
/usr/lib/pyshared/python2.6
/usr/lib/pyshared/python2.6/libsvn
/usr/lib/pyshared/python2.6/libsvn/_delta.so
/usr/lib/pyshared/python2.6/libsvn/_repos.so
/usr/lib/pyshared/python2.6/libsvn/_fs.so
/usr/lib/pyshared/python2.6/libsvn/_client.so
/usr/lib/pyshared/python2.6/libsvn/_core.so
/usr/lib/pyshared/python2.6/libsvn/_diff.so
/usr/lib/pyshared/python2.6/libsvn/_wc.so
/usr/lib/pyshared/python2.6/libsvn/_ra.so
/usr/lib/libsvn_swig_py2.6-1.so.1
```

16. SimpleHTTPServer

例 2.2. SimpleHTTPServer

```
python -m SimpleHTTPServer &
```

```
curl http://localhost:8000/
```

17. fuse-python.x86_64 : Python bindings for FUSE - filesystem in userspace

18. Network

18.1. gevent - A coroutine-based network library for Python

http://www.gevent.org/

19. Python-spdylay - Spdylay Python Extension Module

http://spdylay.sourceforge.net/

20. mechanize

http://wwwsearch.sourceforge.net/mechanize/

Ubuntu

```
$ sudo apt-get install python-mechanize
```

Pip

```
$ sudo pip install mechanize
```

Python 3

```
git clone https://github.com/adevore/mechanize.git
cd mechanize/
git checkout python3
```

21. Dominate

Dominate is a Python library for creating and manipulating HTML documents using an elegant DOM API. It allows you to write HTML pages in pure Python very concisely, which eliminates the need to learn another template language, and lets you take advantage of the more powerful features of Python.

https://pypi.org/project/dominate/

```
pip install dominate
```

22. dbm Key-Value 数据库

```
import dbm

with dbm.open(file="/tmp/cache", flag="c") as db:
    print(db.keys())
    print("=" * 20)
    if "hello".encode() in db.keys():
        print(f"找到: { db['hello']}")
    else:
        print(f"没找到")
        db["hello"] = "world"
```

23. keyboard

```
pip install keyboard
```

23.1. 读取键盘值

读取键盘值

```
root@debian:~# cat test.py
import keyboard

while True:
    k = keyboard.read_key()
    print(k)
```

按键按下时触发

```
root@debian:~# cat test.py
import keyboard

def callback(x):
    print(x)
    print()

keyboard.on_press(callback)
keyboard.wait()
```

按键释放触发

```
root@debian:~# cat test.py
import keyboard

def callback(x):
    print(x)
    print()

keyboard.on_release(callback)
keyboard.wait()
```

23.2. 功能键

```
>>> keyboard.get_hotkey_name(['+', 'left ctrl', 'shift'])
'ctrl+shift+plus'

>>> keyboard.get_hotkey_name(['+', 'left ctrl', 'f13'])
'ctrl+f13+plus'
```

24. httpx

24.1. 安装 https

```
python3 -m pip install httpx

# HTTP/2 支持，我们需要额外安装一个库

python3 -m pip install httpx[http2]
```

24.2. 操作演示

```
import httpx
r = httpx.get('https://www.example.org/')
r.text
r.content
r.json()
r.status_code
```

24.3. Restful CRUD 操作

```
r = httpx.get('https://netkiller.cn/get')
r = httpx.post('https://netkiller.cn/post', data={'key':
'value'})
r = httpx.put('https://netkiller.cn/put', data={'key':
'value'})
r = httpx.delete('https://netkiller.cn/delete')
r = httpx.head('https://netkiller.cn/head')
```

```
r = httpx.options('https://netkiller.cn/options')
```

24.4. HTTP 2

```
import httpx
client = httpx.Client(http2=True, verify=False)
headers = {
    'Host': 'netkiller.com',
    'upgrade-insecure-requests': '1',
    'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X
10_14_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.121 Safari/537.36',
    'accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,image/
avif,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9',
    'sec-fetch-site': 'none',
    'sec-fetch-mode': 'navigate',
    'sec-fetch-dest': 'document',
    'accept-language': 'zh-CN,zh;q=0.9'
}

response = client.get('https://www.netkiller.cn/linux/',
headers=headers)
print(response.text)
```

24.5. 异步请求

```
async with httpx.AsyncClient() as client:
    resp = await
client.get('https://www.netkiller.cn/index.html')
    assert resp.status_code == 200
    html = resp.text
```

asyncio

```
import httpx
import asyncio

async def main():
    async with httpx.AsyncClient() as client:
        resp = await client.get("https://www.netkiller.cn")
        result = resp.text
        print(result)

asyncio.run(main())
```

24.6. 日志输出

```
import logging.config
import httpx

LOGGING_CONFIG = {
    "version": 1,
    "handlers": {
        "default": {
            "class": "logging.StreamHandler",
            "formatter": "http",
            "stream": "ext://sys.stderr"
        }
    },
    "formatters": {
        "http": {
            "format": "%(levelname)s [%(asctime)s] %(name)s -
%(message)s",
            "datefmt": "%Y-%m-%d %H:%M:%S",
        }
```

```
    },
    'loggers': {
        'httpx': {
            'handlers': ['default'],
            'level': 'DEBUG',
        },
        'httpcore': {
            'handlers': ['default'],
            'level': 'DEBUG',
        },
    }
}

logging.config.dictConfig(LOGGING_CONFIG)
httpx.get('https://www.example.com')
```

25. 日志彩色输出

```
# Create a logger object.
import logging
logger = logging.getLogger('your-module')

# Initialize coloredlogs.
import coloredlogs
coloredlogs.install(level='DEBUG')

# Some examples.
logger.debug("this is a debugging message")
logger.info("this is an informational message")
logger.warn("this is a warning message")
logger.error("this is an error message")
logger.critical("this is a critical message")
```

第 3 章 终端环境开发

1. ANSI Color

1.1. ansicolors

1.2. termcolor

```
from termcolor import colored

# then use Termcolor for all colored text output
print(colored('Hello, World!', 'green', 'on_red'))
```

```
import sys
from termcolor import colored, cprint

text = colored('Hello, World!', 'red', attrs=['reverse',
'blink'])
print(text)
cprint('Hello, World!', 'green', 'on_red')

print_red_on_cyan = lambda x: cprint(x, 'red', 'on_cyan')
print_red_on_cyan('Hello, World!')
print_red_on_cyan('Hello, Universe!')

for i in range(10):
    cprint(i, 'magenta', end=' ')

cprint("Attention!", 'red', attrs=['bold'], file=sys.stderr)
```

1.3. Colorama

https://pypi.org/project/colorama/

```
pip install colorama
```

初始化操作

init(autoreset = False)，当 autoreset = True 时自动恢复到默认颜色

```
#!/usr/bin/env python
from colorama import init, Fore, Back, Style

if __name__ == "__main__":

    init(autoreset=True)    #  初始化，自动恢复到默认颜色
    print(Fore.RED + 'some red text')
    print(Back.GREEN + 'and with a green background')
    print(Style.DIM + 'and in dim text')
```

常用格式

Fore是针对字体颜色，Back是针对字体背景颜色，Style是针对字体格式

- Fore: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, RESET.
- Back: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, RESET.
- Style: DIM, NORMAL, BRIGHT, RESET_ALL

2. 进度条

2.1. progress

CLI 和 TUI 开发中我们常常会用到进度条，用于展示下载或执行进度等等。

```
Bar |################################| 200/200 [2 / 0 /
0:00:00] (None)
ChargingBar ██████████████████████████████████ 200/200 [2 / 0 /
0:00:00] (None)
FillingSquaresBar ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ 200/200 [2 /
0 / 0:00:00] (None)
FillingCirclesBar ●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●● 200/200 [2 /
0 / 0:00:00] (None)
IncrementalBar |▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉| 100% [0:00:02
/ 0 / 0:00:00]
PixelBar |⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿| 100% [0:00:02
/ 0 / 0:00:00]
ShadyBar |▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒| 100% [0:00:02 / 0 /
0:00:00]
Corolored |▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒| 200/200
Spinner %(index)d -
PieSpinner %(index)d ◷
MoonSpinner %(index)d ◑
LineSpinner %(index)d ⎯
PixelSpinner %(index)d ⣿
Counter 100
Countdown 0
Stack ▉
Pie ●
Random |▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ | 96
```

安装

```
```

```
pip install progress
```

演示

```
from progress.bar import Bar

bar = Bar('Processing', max=20)
for i in range(20):
    # Do some work
    bar.next()
bar.finish()
```

执行结果

```
neo@MacBook-Pro-Neo ~/workspace/python % python3.9 progress-
demo.py
Processing |################################| 20/20
```

条形进度条（Bars）

```
from progress.bar import Bar
import time
import random

length = 100

bar = Bar('Processing', max=length)
for i in range(length):
```

```
    # Do some work
    time.sleep(random.randrange(1, 5)*0.05)
    bar.next()
bar.finish()
```

运行结果

```
Processing |#########                          | 31/100
```

将 # 换成 @ 符号，并且进度改为百分比。

```
bar = Bar('Loading', fill='@', suffix='%(percent)d%%')

演示

Loading |@@@@@@@@@@@@@                    | 42%
```

方块进度条（ChargingBar）

```
from progress.bar import ChargingBar
import time
import random
length = 100
with ChargingBar('Processing', max=length) as bar:
    for i in range(length):
        # Do some work
        time.sleep(random.randrange(1, 5)*0.05)
        bar.next()
```

运行结果

```
Processing ▉▉▉
```

```
bar = FillingCirclesBar('Processing', max=length)
for i in range(length):
    # Do some work
    time.sleep(random.randrange(1, 5)*0.05)
    bar.next()
bar.finish()
```

运行结果

```
Processing ●●●●●●●●●●●●●○○○
```

```
Memory |                                  | 52%
```

2.2. tqdm

安装

```
neo@MacBook-Pro-Neo ~ % pip install tqdm
```

tqdm 命令

```
neo@MacBook-Pro-Neo ~ % tqdm -h
Usage:
  tqdm [--help | options]

Options:
  -h, --help     Print this help and exit.
  -v, --version  Print version and exit.
  --desc=<desc>  : str, optional
            Prefix for the progressbar.
  --total=<total>  : int or float, optional
            The number of expected iterations. If unspecified,
            len(iterable) is used if possible. If float("inf")
or as a last
            resort, only basic progress statistics are
displayed
            (no ETA, no progressbar).
            If `gui` is True and this parameter needs
subsequent updating,
            specify an initial arbitrary large positive number,
            e.g. 9e9.
  --leave  : bool, optional
            If [default: True], keeps all traces of the
```

```
progressbar
            upon termination of iteration.
            If `None`, will leave only if `position` is `0`.
  --ncols=<ncols>  : int, optional
            The width of the entire output message. If
specified,
            dynamically resizes the progressbar to stay within
this bound.
            If unspecified, attempts to use environment width.
The
            fallback is a meter width of 10 and no limit for
the counter and
            statistics. If 0, will not print any meter (only
stats).
  --mininterval=<mininterval>  : float, optional
            Minimum progress display update interval [default:
0.1] seconds.
  --maxinterval=<maxinterval>  : float, optional
            Maximum progress display update interval [default:
10] seconds.
            Automatically adjusts `miniters` to correspond to
`mininterval`
            after long display update lag. Only works if
`dynamic_miniters`
            or monitor thread is enabled.
  --miniters=<miniters>  : int or float, optional
            Minimum progress display update interval, in
iterations.
            If 0 and `dynamic_miniters`, will automatically
adjust to equal
            `mininterval` (more CPU efficient, good for tight
loops).
            If > 0, will skip display of specified number of
iterations.
            Tweak this and `mininterval` to get very efficient
loops.
            If your progress is erratic with both fast and slow
iterations
            (network, skipping items, etc) you should set
miniters=1.
  --ascii=<ascii>  : bool or str, optional
            If unspecified or False, use unicode (smooth
blocks) to fill
            the meter. The fallback is to use ASCII characters
" 123456789#".
```

```
  --disable  : bool, optional
             Whether to disable the entire progressbar wrapper
             [default: False]. If set to None, disable on non-
TTY.
  --unit=<unit>  : str, optional
             String that will be used to define the unit of each
iteration
             [default: it].
  --unit-scale=<unit_scale>  : bool or int or float, optional
             If 1 or True, the number of iterations will be
reduced/scaled
             automatically and a metric prefix following the
             International System of Units standard will be
added
             (kilo, mega, etc.) [default: False]. If any other
non-zero
             number, will scale `total` and `n`.
  --dynamic-ncols  : bool, optional
             If set, constantly alters `ncols` and `nrows` to
the
             environment (allowing for window resizes) [default:
False].
  --smoothing=<smoothing>  : float, optional
             Exponential moving average smoothing factor for
speed estimates
             (ignored in GUI mode). Ranges from 0 (average
speed) to 1
             (current/instantaneous speed) [default: 0.3].
  --bar-format=<bar_format>  : str, optional
             Specify a custom bar string formatting. May impact
performance.
             [default: '{l_bar}{bar}{r_bar}'], where
             l_bar='{desc}: {percentage:3.0f}%|' and
             r_bar='| {n_fmt}/{total_fmt} [{elapsed}
<{remaining}, '
               '{rate_fmt}{postfix}]'
             Possible vars: l_bar, bar, r_bar, n, n_fmt, total,
total_fmt,
               percentage, elapsed, elapsed_s, ncols, nrows,
desc, unit,
               rate, rate_fmt, rate_noinv, rate_noinv_fmt,
               rate_inv, rate_inv_fmt, postfix, unit_divisor,
               remaining, remaining_s, eta.
             Note that a trailing ": " is automatically removed
after {desc}
```

```
            if the latter is empty.
  --initial=<initial>  : int or float, optional
            The initial counter value. Useful when restarting a
progress
            bar [default: 0]. If using float, consider
specifying `{n:.3f}`
            or similar in `bar_format`, or specifying
`unit_scale`.
  --position=<position>  : int, optional
            Specify the line offset to print this bar (starting
from 0)
            Automatic if unspecified.
            Useful to manage multiple bars at once (eg, from
threads).
  --postfix=<postfix>  : dict or *, optional
            Specify additional stats to display at the end of
the bar.
            Calls `set_postfix(**postfix)` if possible (dict).
  --unit-divisor=<unit_divisor>  : float, optional
            [default: 1000], ignored unless `unit_scale` is
True.
  --write-bytes  : bool, optional
            If (default: None) and `file` is unspecified,
            bytes will be written in Python 2. If `True` will
also write
            bytes. In all other cases will default to unicode.
  --lock-args=<lock_args>  : tuple, optional
            Passed to `refresh` for intermediate output
            (initialisation, iterating, and updating).
  --nrows=<nrows>  : int, optional
            The screen height. If specified, hides nested bars
outside this
            bound. If unspecified, attempts to use environment
height.
            The fallback is 20.
  --colour=<colour>  : str, optional
            Bar colour (e.g. 'green', '#00ff00').
  --delay=<delay>  : float, optional
            Don't display until [default: 0] seconds have
elapsed.
  --delim=<delim>  : chr, optional
            Delimiting character [default: '\n']. Use '\0' for
null.
            N.B.: on Windows systems, Python converts '\n' to
'\r\n'.
```

```
  --buf-size=<buf_size>  : int, optional
            String buffer size in bytes [default: 256]
            used when `delim` is specified.
  --bytes  : bool, optional
            If true, will count bytes, ignore `delim`, and
default
            `unit_scale` to True, `unit_divisor` to 1024, and
`unit` to 'B'.
  --tee  : bool, optional
            If true, passes `stdin` to both `stderr` and
`stdout`.
  --update  : bool, optional
            If true, will treat input as newly elapsed
iterations,
            i.e. numbers to pass to `update()`. Note that this
is slow
            (~2e5 it/s) since every input must be decoded as a
number.
  --update-to  : bool, optional
            If true, will treat input as total elapsed
iterations,
            i.e. numbers to assign to `self.n`. Note that this
is slow
            (~2e5 it/s) since every input must be decoded as a
number.
  --null  : bool, optional
            If true, will discard input (no stdout).
  --manpath=<manpath>  : str, optional
            Directory in which to install tqdm man pages.
  --comppath=<comppath>  : str, optional
            Directory in which to place tqdm completion.
  --log=<log>  : str, optional
            CRITICAL|FATAL|ERROR|WARN(ING)|[default:
'INFO']|DEBUG|NOTSET.
```

演示

```
from tqdm import tqdm
```

```
import time

for i in tqdm(range(100)):
  time.sleep(0.1)
  pass
```

演示效果

```
neo@MacBook-Pro-Neo ~/workspace/python % python3.9
/Users/neo/workspace/python/terminal/tqdm/demo.py
 31%|                                                  | 31/100
[00:03<00:07,  9.66it/s]
```

设置进度条长度

```
from tqdm import tqdm,trange
import time

for i in trange(100):
  time.sleep(0.1)
  pass
```

```
from tqdm import tqdm
import time

length = 50
# total 参数设置进度条的总长度
with tqdm(total=length) as bar:
    for i in range(length):
```

```
        time.sleep(0.05)
        # 每次更新进度条的长度
        bar.update(1)
```

```
from tqdm import tqdm
import time
length = 50
#total参数设置进度条的总长度
bar = tqdm(total=length)
for i in range(length):
  time.sleep(0.05)
  #每次更新进度条的长度
  bar.update(1)
#关闭进度条
bar.close()
```

设置任务数量

```
from tqdm import tqdm
import time

bar = tqdm(["加载", "执行", "保存", "完成"])
for step in bar:
    time.sleep(1)
    bar.set_description("当前进度 %s" % step)
```

多进程进度监控

```
from time import sleep
from tqdm import trange, tqdm
import multiprocessing
import random

threads = list(range(5))
total = 100


def run(n):
    name = multiprocessing.current_process().name
    description = "#{} - {}".format(n, name)
    for i in trange(total, desc=description, position=n,
ascii=True):
        sleep(random.randrange(1, 9)*0.1)


if __name__ == '__main__':

    pool = multiprocessing.Pool(len(threads))
    pool.map(run, threads)
    pool.close()
    pool.join()
```

效果展示

```
#0 - SpawnPoolWorker-1:   9%|#########
| 9/100 [00:04<00:45,  1.98it/s]
#1 - SpawnPoolWorker-4:  10%|#########9
| 10/100 [00:04<00:44,  2.01it/s]
#2 - SpawnPoolWorker-2:  13%|############8
| 13/100 [00:04<00:34,  2.54it/s]
#3 - SpawnPoolWorker-3:  10%|#########9
| 10/100 [00:04<00:50,  1.78it/s]
#4 - SpawnPoolWorker-5:  14%|#############8
| 14/100 [00:04<00:28,  2.99it/s]
```

2.3. alive-progress

```
pip install alive-progress
```

```
from alive_progress import alive_bar
import time
items = range(10)                    # retrieve your set of
items
with alive_bar(len(items)) as bar:  # declare your expected
total
    for item in items:               # iterate as usual
        # process each item
        bar()                        # call after consuming
one item
        time.sleep(1)
```

效果展示

```
|                                   | ▅▃▁ 4/10 [40%] in 3s
(1.2/s, eta: 4s)
```

3. texttable - module for creating simple ASCII tables

https://github.com/foutaise/texttable/

```
pip install texttable
```

程序演示

```
from texttable import Texttable

table = Texttable()
table.add_rows([["Name", "Age", "Nickname"],
                ["Neo", 35, "netkiller"],
                ["李磊", 23, "Lee"],
                ["韩美美", 28, "May"]])
print(table.draw())
```

```
+--------+-----+-----------+
|  Name  | Age | Nickname  |
+========+=====+===========+
| Neo    |  35 | netkiller |
+--------+-----+-----------+
| 李磊   |  23 |    Lee    |
+--------+-----+-----------+
| 韩美美 |  28 |    May    |
+--------+-----+-----------+
```

3.1. 对齐设置

set_header_align(self, array) 设置水平对齐

- l 表示左对齐
- r 表示右对齐
- c 表示居中对齐

set_cols_align(self, array) 设置水平对齐

- l 表示左对齐
- r 表示右对齐
- c 表示居中对齐

set_cols_valign(self, array) 设置垂直对齐

- t 表示顶部对齐
- m 表示中间对齐
- b 表示底部对齐

```
from texttable import Texttable

table = Texttable()
table.set_cols_align(["l", "r", "c"])
table.set_cols_valign(["t", "m", "b"])
table.add_rows([["Name", "Age", "Nickname"],
                ["Mr\nXavier\nHuon", 32, "Xav'"],
                ["Mr\nBaptiste\nClement", 1, "Baby"],
                ["Mme\nLouise\nBourgeau", 28,
"Lou\n\nLoue"]])
print(table.draw())
print()
```

输出结果

```
+----------+-----+----------+
|   Name   | Age | Nickname |
+==========+=====+==========+
| Mr       |     |          |
| Xavier   |  32 |          |
| Huon     |     |   Xav'   |
+----------+-----+----------+
| Mr       |     |          |
| Baptiste |   1 |          |
| Clement  |     |   Baby   |
+----------+-----+----------+
| Mme      |     |   Lou    |
| Louise   |  28 |          |
| Bourgeau |     |   Loue   |
+----------+-----+----------+
```

3.2. 设置表格风格

```
from texttable import Texttable
table = Texttable()
for header in (Texttable.BORDER, Texttable.HEADER,
Texttable.HLINES, Texttable.VLINES):
    table.set_deco(header)

    table.set_cols_align(["l", "r", "c"])
    table.set_cols_valign(["t", "m", "b"])
    table.add_rows([["Name", "Age", "Nickname"],
                    ["Neo", 35, "netkiller"],
                    ["李磊", 23, "Lee"],
                    ["韩美美", 28, "May"]])
    print(table.draw())
    print("\n\n")
```

输出结果

```
+-------------------------+
|  Name    Age   Nickname  |
| Neo       35   netkiller |
| 李磊      23      Lee    |
| 韩美美    28      May    |
+-------------------------+



 Name    Age   Nickname
========================
Neo       35   netkiller
李磊      23      Lee
韩美美    28      May
Neo       35   netkiller
李磊      23      Lee
韩美美    28      May



 Name    Age   Nickname
Neo       35   netkiller
+-------------------------+
李磊      23      Lee
+-------------------------+
韩美美    28      May
+-------------------------+
Neo       35   netkiller
+-------------------------+
李磊      23      Lee
+-------------------------+
韩美美    28      May
+-------------------------+
Neo       35   netkiller
+-------------------------+
李磊      23      Lee
+-------------------------+
韩美美    28      May



 Name   | Age | Nickname
```

```
Neo     |  35 | netkiller
李磊    |  23 |    Lee
韩美美  |  28 |    May
Neo     |  35 | netkiller
李磊    |  23 |    Lee
韩美美  |  28 |    May
Neo     |  35 | netkiller
李磊    |  23 |    Lee
韩美美  |  28 |    May
Neo     |  35 | netkiller
李磊    |  23 |    Lee
韩美美  |  28 |    May
```

3.3. 自定义风格

自定义行列线条字符

```
set_chars(self, array)
     |      Set the characters used to draw lines between rows
and columns
     |
     |      - the array should contain 4 fields:
     |
     |          [horizontal, vertical, corner, header]
     |
     |      - default is set to:
     |
     |          ['-', '|', '+', '=']
```

set_chars(self, array) 数字的四个参数分别是:

- horizontal 水平画线字符
- vertical 垂直画线字符
- corner 转角画线字符
- header 表头画线字符

默认是 ['-', '|', '+', '=']

下面这段代码模仿 MySQL 终端输出样式

```
table = Texttable()
table.set_cols_align(["r", "l", "c", "l", "l"])
table.set_cols_valign(["m", "m", "m", "m", "m"])
table.set_chars(['-', '|', '+', '-'])
table.set_cols_dtype(['i', 't', 'i', 't', 'a'])
table.add_rows([["id", "name", "age", "nickname", "ctime"],
                [1, "Neo", 35, "netkiller", "2021-05-16
10:14:00"],
                [2, "Tom", 23, "Lee", "2021-05-16 10:14:00"],
                [3, "Jerry", 28, "May", "2021-05-16
10:14:00"]])
print(table.draw())
print()
```

```
+----+-------+-----+-----------+---------------------+
| id | name  | age | nickname  |        ctime        |
+----+-------+-----+-----------+---------------------+
|  1 | Neo   | 35  | netkiller | 2021-05-16 10:14:00 |
+----+-------+-----+-----------+---------------------+
|  2 | Tom   | 23  | Lee       | 2021-05-16 10:14:00 |
+----+-------+-----+-----------+---------------------+
|  3 | Jerry | 28  | May       | 2021-05-16 10:14:00 |
+----+-------+-----+-----------+---------------------+
```

怎么样，似曾相识吧？跟 mysql 命令中输出结果一致。

3.4. 设置列数据类型

```
from texttable import Texttable
table = Texttable()
table.set_deco(Texttable.HEADER)
table.set_cols_dtype(['t',  # text
                      'f',  # float (decimal)
                      'e',  # float (exponent)
                      'i',  # integer
                      'a'])  # automatic
table.set_cols_align(["l", "r", "r", "r", "l"])
table.add_rows([["text",    "float", "exp", "int", "auto"],
                ["abcd",    "67",    654,   89,    128.001],
                ["efghijk", 67.5434, .654,  89.6,
                 12800000000000000000000.00023],
                ["lmn",     5e-78,   5e-78, 89.4,
.000000000000128],
                ["opqrstu", .023,    5e+78, 92.,
12800000000000000000000]])
print(table.draw())
```

输出结果

```
 text     float       exp      int     auto
===============================================
abcd      67.000   6.540e+02    89   128.001
efghijk   67.543   6.540e-01    90   1.280e+22
lmn        0.000   5.000e-78    89   0.000
opqrstu    0.023   5.000e+78    92   1.280e+22
```

3.5. 彩色表格

texttable 本身不支持 ANSI 彩色文本输出，我以修复了该 Bug，已经想修复代码pull request 给作者。

Pull Request: https://github.com/foutaise/texttable/pull/75

我的代码库地址： https://github.com/netkiller/texttable

```
from texttable import Texttable
from colorama import Fore, Back, Style, init
table = Texttable()
table.set_chars(['-', '|', '+', '-'])
# table.set_cols_width([8, 5, 19])
table.add_rows([["Name", "Age", "Nickname"],
                ["Neo", 35, Fore.RED+"netkiller"+Fore.RESET],
                ["李磊", 23, Fore.GREEN+"Lee"+Fore.RESET],
                ["韩美美", 28, Fore.BLUE+"May"+Fore.RESET]])
print(table.draw())
```

4. prompt_toolkit

prompt_toolkit is a library for building powerful interactive command line applications in Python.

https://github.com/prompt-toolkit/python-prompt-toolkit

https://github.com/prompt-toolkit/pymux

https://github.com/prompt-toolkit/pyvim

4.1. 安装

```
pip install prompt-toolkit
```

5. Simple Terminal Menu

```
pip install simple-term-menu
```

6. picotui

```
pip install picotui
```

7. TUI

7. TUI

7.1. Console

http://www.effbot.org/zone/console-handbook.htm

7.2. urwid

http://excess.org/urwid/

7.3. pycdk

http://pycdk.sourceforge.net/

7.4. python-newt - A NEWT module for Python

https://fedorahosted.org/newt/

第 4 章 Python 调试技巧

1. 显示代码所在文件行

```
#!/usr/bin/python
import sys

print("=" * 50)
print("here is :", __file__, sys._getframe().f_lineno)
print("=" * 50)
```

部分 II. Frameworks

第 5 章 Uvicorn

1. 代码启动

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app=app, host="127.0.0.1", port=8000)
```

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app='main:app', host="127.0.0.1", port=8000,
reload=True, debug=True)
```

2. 命令行启动

```
uvicorn main:app --host=127.0.0.1 --port=8000 --reload
```

3. 日志

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##############################################
# Home  : https://www.netkiller.cn
# Author: Neo <netkiller@msn.com>
# Upgrade: 2023-07-07
##############################################
try:
    import uvicorn, logging, sys

    from config import LOGFILE
except ImportError as err:
    print("ImportError: %s" % (err))
    exit()

LOGGING_CONFIG = {
    "version": 1,
    "disable_existing_loggers": False,
    "formatters": {
        "default": {
            "()": "uvicorn.logging.DefaultFormatter",
            "fmt": "%(levelprefix)s %(message)s",
            "use_colors": None,
        },
        "access": {
            "()": "uvicorn.logging.AccessFormatter",
            "fmt": '%(levelprefix)s %(client_addr)s - "%
(request_line)s" %(status_code)s',
        },
    },
    "handlers": {
        "default": {"formatter": "default", "class":
"logging.handlers.TimedRotatingFileHandler", "filename":
LOGFILE},
        "access": {"formatter": "access", "class":
"logging.handlers.TimedRotatingFileHandler", "filename": f"
{LOGFILE}.access.log"},
    },
```

```
    "loggers": {
        "": {"handlers": ["default"], "level": "INFO"},
        "uvicorn.error": {"level": "INFO"},
        "uvicorn.access": {"handlers": ["access"], "level":
"INFO", "propagate": False},
    },
}

if __name__ == "__main__":
    try:
        uvicorn.run(app="main:app", host="0.0.0.0", port=8000,
workers=4, log_config=LOGGING_CONFIG)
    except KeyboardInterrupt:
        print("Crtl+C Pressed. Shutting down.")
```

输出结果

```
docker@debian:~$ tail /tmp/uvicorn.access.log
INFO:     172.16.0.102:61485 - "GET / HTTP/1.1" 200 OK
INFO:     172.16.0.102:61485 - "GET / HTTP/1.1" 304 Not
Modified
INFO:     172.16.0.102:61485 - "GET / HTTP/1.1" 304 Not
Modified
INFO:     172.16.0.102:61485 - "GET / HTTP/1.1" 304 Not
Modified
INFO:     172.16.0.102:61485 - "GET / HTTP/1.1" 304 Not
Modified

docker@debian:~$ tail /tmp/uvicorn.log
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     OPENAI_API_KEY=sk-
DNsMaVmxxIm3Xp7nev5OT3BlbkFJ8wb3Y8ZTZrZw2mcGgDF1
INFO:     Started server process [21112]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     OPENAI_API_KEY=sk-
DNsMaVmxxIm3Xp7nev5OT3BlbkFJ8wb3Y8ZTZrZw2mcGgDF1
INFO:     Started server process [21110]
INFO:     Waiting for application startup.
```

```
INFO:     Application startup complete.
```

4. FastAPI

4.1. Post Request

From 数据

```
pip install python-multipart
```

```
from fastapi import FastAPI, Form
# from starlette.requests import Request
from starlette.responses import Response
from starlette.testclient import TestClient

app = FastAPI()

@app.post("/form")
async def login(username: str = Form(), password: str =
Form()):
    return {"username": username, "password": password}

client = TestClient(app)
data = {"username": "netkiller", "password": "123456"}
response = client.post("/form", data=data)
print(response.content.decode())
```

Json 数据转为 dict

```
from fastapi import FastAPI, Request
from typing import Dict
```

```
# from starlette.requests import Request
from starlette.responses import Response
from starlette.testclient import TestClient

app = FastAPI()


@app.post("/json")
async def json(item: dict):
    print(item)
    return "OK"


client = TestClient(app)
data = {"key": "value"}
response = client.post("/json", json=data)
print(response.content.decode())
```

Data 原始数据

```
from fastapi import FastAPI, Request

# from starlette.requests import Request
from starlette.responses import Response
from starlette.testclient import TestClient

app = FastAPI()


@app.post("/webhook")
async def the_webhook(request: Request):
    return await request.body()


data = b"""EURUSD Less Than 1.09092
{"Condition": "value"}
[3,4,5,]
{}"""
```

```
data = b"""EURUSD Less Than 1.09092"""

client = TestClient(app)
response = client.post("/webhook", data=data)
print(response.content.decode())
```

POST 接收 JSON 数据

```
@app.post("/android/notification", summary="通知",
description=f"通知接口", tags=["android"])
async def notification(request: Request):
    jsonText = (await request.body()).decode()
    print(jsonText)
    print(await request.json())
    return {"status": true, "data": {}, "msg": "成功"}
```

4.2. api_route

```
@app.api_route("/", methods=["GET", "POST"])
async def handler():
    return {}
```

4.3. slowapi 流向控制

```
pip install slowapi
```

```
from fastapi import FastAPI
from slowapi import Limiter, _rate_limit_exceeded_handler
from slowapi.util import get_remote_address


limiter = Limiter(key_func=get_remote_address)
app = FastAPI()
app.state.limiter = limiter
app.add_exception_handler(RateLimitExceeded,
_rate_limit_exceeded_handler)

@app.get("/home")
@limiter.limit("5/minute")
async def homepage(request: Request):
    return PlainTextResponse("test")

@app.get("/mars")
@limiter.limit("5/minute")
async def homepage(request: Request, response: Response):
    return {"key": "value"}
```

4.4. 异步执行

```
from fastapi import APIRouter
import time
import asyncio

router = APIRouter()


@router.get("/a")
async def a():
    time.sleep(1)
```

```
    return {"message": "异步模式，但是同步执行sleep函数，执行过程是串行的"}


@router.get("/b")
async def b():
    loop = asyncio.get_event_loop()
    await loop.run_in_executor(None, time.sleep, 1)
    return {"message": "异步线程池中运行sleep函数"}


@router.get("/c")
async def c():
    await asyncio.sleep(1)
    return {"message": "异步模式，且异步执行sleep函数"}


@router.get("/d")
def d():
    time.sleep(1)
    return {"message": "同步模式"}
```

4.5. 缓存

```
pip install fastapi-cache2
```

```
from fastapi import FastAPI
from starlette.requests import Request
from starlette.responses import Response

from fastapi_cache import FastAPICache
from fastapi_cache.backends.redis import RedisBackend
from fastapi_cache.decorator import cache

from redis import asyncio as aioredis
```

```
app = FastAPI()


@cache()
async def get_cache():
    return 1


@app.get("/")
@cache(expire=60)
async def index():
    return dict(hello="world")


@app.on_event("startup")
async def startup():
    redis = aioredis.from_url("redis://localhost",
encoding="utf8", decode_responses=True)
    FastAPICache.init(RedisBackend(redis), prefix="fastapi-
cache")
```

缓存 Json 数据结构

```
@app.get("/")
@cache(expire=60, coder=JsonCoder)
async def index():
    return dict(hello="world")
```

自定义 key

```
def my_key_builder(
        func,
```

```
        namespace: Optional[str] = "",
        request: Request = None,
        response: Response = None,
        *args,
        **kwargs,
):
    prefix = FastAPICache.get_prefix()
    cache_key = f"{prefix}:{namespace}:{func.__module__}:
{func.__name__}:{args}:{kwargs}"
    return cache_key


@app.get("/")
@cache(expire=60, coder=JsonCoder,
key_builder=my_key_builder)
async def index():
    return dict(hello="world")
```

4.6. HTTP Auth

```
security = HTTPBasic()

def auth(credentials: Annotated[HTTPBasicCredentials,
Depends(security)]):
    current_username_bytes =
credentials.username.encode("utf8")
    correct_username_bytes = b"admin"
    is_correct_username =
compare_digest(current_username_bytes,
correct_username_bytes)
    current_password_bytes =
credentials.password.encode("utf8")
    correct_password_bytes = b"admin"
    is_correct_password =
compare_digest(current_password_bytes,
correct_password_bytes)
    if not (is_correct_username and is_correct_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
```

```
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Basic"},
        )
    return credentials.username
```

应用与方法

```
@app.get("/who")
@cache(expire=60)
def read_current_user(username: Annotated[str,
Depends(auth)]):
    return {"username": username}
```

全局配置

```
app = FastAPI(title="netkiller", description="HTTP Auth 测试",
dependencies=[Depends(auth)])
```

4.7. SSE

SSE 协议格式

协议字段之间使用\r\n 分割，数据结尾处使用两个\r\n。

event: message\r\ndata: \xe4\xb8\x83\r\nretry: 15000\r\n\r\n
event: message\r\ndata: \xe5\xa4\x95\r\nretry: 15000\r\n\r\n

event: 表示事件，message和error，对应前端会分别触发onmessage或onerror事件。
retry: 重试时间，让客户端在retry时间后进行重试，单位是毫秒。
data: 具体的数据。

```
pip install sse_starlette
```

服务器端

```
from fastapi import FastAPI, Request
from sse_starlette.sse import EventSourceResponse
import asyncio
import uvicorn

app = FastAPI()


@app.get("/")
async def home():
    return {"message": "Hello World"}


@app.get("/sse")
async def sse(request: Request):
    async def ServerSendEvents(request: Request):
        books = ["Netkiller Linux 手札", "Netkiller MySQL 手札", "Netkiller Python 手札", "Netkiller Spring 手札", "Netkiller Java 手札", "Netkiller FreeBSD 手札", "Netkiller Network 手札", "Netkiller Blockchain 手札"]
        for book in books:
            if await request.is_disconnected():
                print("连接已中断")
                break
            yield {"event": "message", "retry": 15000,
```

```
"data": book}

            await asyncio.sleep(0.5)

    g = ServerSendEvents(request)
    return EventSourceResponse(g)


if __name__ == "__main__":
    try:
        uvicorn.run(app=app, host="0.0.0.0", port=8080,
log_level="info")
    except KeyboardInterrupt:
        print("Crtl+C Pressed. Shutting down.")
```

客户端

```
#!/usr/bin/python
# -*-coding: utf-8-*-
import requests


def test():
    url = r"http://127.0.0.1:8080/sse"
    headers = {"Content-Type": "text/event-stream"}
    response = requests.get(url, headers=headers,
stream=True)
    for chunk in response.iter_content(chunk_size=1024,
decode_unicode=True):
        print(chunk)


if __name__ == "__main__":
    test()
```

4.8. Fief 认证框架

https://docs.fief.dev/integrate/python/fastapi/

第 6 章 Robot Framework 测试框架

http://robotframework.org/

第 7 章 Web framework

http://wiki.secondlife.com/wiki/Mulib

1. Django

```
wget http://www.djangoproject.com/download/0.96/tarball/
tar zxvf Django-0.96.tar.gz
cd Django-0.96
python setup.py install
```

生成项目

```
django-admin.py startproject newtest
```

web server

```
cd newtest/
./manage.py runserver
```

helloworld.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, Django.")
```

urls.py

```
from django.conf.urls.defaults import *
```

```
urlpatterns = patterns('',
    # Example:
    # (r'^newtest/', include('newtest.foo.urls')),
    (r'^$', 'newtest.helloworld.index'),

    # Uncomment this for admin:
#     (r'^admin/', include('django.contrib.admin.urls')),
)
```

启动Web Server

```
# ./manage.py runserver
Validating models...
0 errors found.

Django version 0.96, using settings 'newtest.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

curl http://127.0.0.1:8000/

2. Pylons

2.1. Getting Started with Pylons

Installing

```
[neo@development ~]$ curl http://pylonshq.com/download/1.0/go-
pylons.py | python - mydevenv
[neo@development ~]$ source mydevenv/bin/activate
(mydevenv)[neo@development ~]$

(mydevenv)[neo@development ~]$ paster create -t pylons
helloworld
```

Running the application

```
(mydevenv)[neo@development ~]$ cd helloworld
(mydevenv)[neo@development helloworld]$ paster serve --reload
development.ini
Starting subprocess with file monitor
Starting server in PID 26895.
16:38:39,051 INFO  [paste.httpserver.ThreadPool] Cannot use
kill_thread_limit as ctypes/killthread is not available
serving on http://127.0.0.1:5000
```

create a controller

```
(mydevenv)[neo@development helloworld]$ paster controller hello
Creating /home/neo/helloworld/helloworld/controllers/hello.py
Creating
/home/neo/helloworld/helloworld/tests/functional/test_hello.py
```

http://192.168.3.9:5000/hello/index

Debian/Ubuntu

```
$ sudo apt-cache search pylons
$ sudo apt-get install python-pylons
$ paster create -t pylons helloworld
/usr/lib/pymodules/python2.6/pylons/templating.py:610:
UserWarning: Unbuilt egg for setuptools [unknown version]
(/usr/lib/python2.6/dist-packages)
  Engine = entry_point.load()
Selected and implied templates:
  Pylons#pylons  Pylons application template

Variables:
  egg:      helloworld
  package:  helloworld
  project:  helloworld
Enter template_engine (mako/genshi/jinja2/etc: Template
language) ['mako']:
Enter sqlalchemy (True/False: Include SQLAlchemy 0.5
configuration) [False]: True
Creating template pylons
Creating directory ./helloworld
  Recursing into +package+
    Creating ./helloworld/helloworld/
    Copying __init__.py_tmpl to
./helloworld/helloworld/__init__.py
    Recursing into config
      Creating ./helloworld/helloworld/config/
      Copying __init__.py_tmpl to
./helloworld/helloworld/config/__init__.py
      Copying deployment.ini_tmpl_tmpl to
./helloworld/helloworld/config/deployment.ini_tmpl
      Copying environment.py_tmpl to
./helloworld/helloworld/config/environment.py
      Copying middleware.py_tmpl to
./helloworld/helloworld/config/middleware.py
      Copying routing.py_tmpl to
./helloworld/helloworld/config/routing.py
    Recursing into controllers
      Creating ./helloworld/helloworld/controllers/
      Copying __init__.py_tmpl to
./helloworld/helloworld/controllers/__init__.py
      Copying error.py_tmpl to
```

```
./helloworld/helloworld/controllers/error.py
    Recursing into lib
      Creating ./helloworld/helloworld/lib/
      Copying __init__.py_tmpl to
./helloworld/helloworld/lib/__init__.py
      Copying app_globals.py_tmpl to
./helloworld/helloworld/lib/app_globals.py
      Copying base.py_tmpl to
./helloworld/helloworld/lib/base.py
      Copying helpers.py_tmpl to
./helloworld/helloworld/lib/helpers.py
    Recursing into model
      Creating ./helloworld/helloworld/model/
      Copying __init__.py_tmpl to
./helloworld/helloworld/model/__init__.py
      Copying meta.py_tmpl to
./helloworld/helloworld/model/meta.py
    Recursing into public
      Creating ./helloworld/helloworld/public/
      Copying bg.png to ./helloworld/helloworld/public/bg.png
      Copying favicon.ico to
./helloworld/helloworld/public/favicon.ico
      Copying index.html_tmpl to
./helloworld/helloworld/public/index.html
      Copying pylons-logo.gif to
./helloworld/helloworld/public/pylons-logo.gif
    Recursing into templates
      Creating ./helloworld/helloworld/templates/
    Recursing into tests
      Creating ./helloworld/helloworld/tests/
      Copying __init__.py_tmpl to
./helloworld/helloworld/tests/__init__.py
      Recursing into functional
        Creating ./helloworld/helloworld/tests/functional/
        Copying __init__.py_tmpl to
./helloworld/helloworld/tests/functional/__init__.py
      Copying test_models.py_tmpl to
./helloworld/helloworld/tests/test_models.py
    Copying websetup.py_tmpl to
./helloworld/helloworld/websetup.py
  Copying MANIFEST.in_tmpl to ./helloworld/MANIFEST.in
  Copying README.txt_tmpl to ./helloworld/README.txt
  Copying development.ini_tmpl to ./helloworld/development.ini
  Recursing into docs
    Creating ./helloworld/docs/
```

```
    Copying index.txt_tmpl to ./helloworld/docs/index.txt
  Copying ez_setup.py to ./helloworld/ez_setup.py
  Copying setup.cfg_tmpl to ./helloworld/setup.cfg
  Copying setup.py_tmpl to ./helloworld/setup.py
  Copying test.ini_tmpl to ./helloworld/test.ini
Running /usr/bin/python setup.py egg_info
```

配置host监听地址

```
$ cd helloworld
$ vim development.ini
host = 0.0.0.0
```

启动服务

```
$ paster serve --reload development.ini
```

2.2. config/routing.py

url routing 做为静态化让所有后缀加上.html

```
(mydevenv)[neo@development helloworld]$ vim
helloworld/config/routing.py

    map.connect('/{controller}/{action}.html')
    map.connect('/{controller}/{action}/{id}.html')
```

2.3. mako template

http://www.makotemplates.org/

include

```
<%include file="header.html"/>

    hello world

<%include file="footer.html"/>
```

传递参数

```
<%include file="toolbar.html"
args="current_section='members', username='ed'"/>
```

inherit

```
<%inherit file="base.html"/>
```

3. Pyramid

3.1. Getting Started

```
$ sudo apt-get install python-setuptools

$ sudo easy_install -U pyramid
```

virtualenv - create virtual Python instances

```
sudo apt-get install python-virtualenv

virtualenv --no-site-packages myenv

cd myenv

$ sudo easy_install -U pyramid
```

Hello world

```
$ vim test/helloworld.py
from pyramid.config import Configurator
from pyramid.response import Response
from paste.httpserver import serve

def hello_world(request):
    return Response('Hello world!')

def goodbye_world(request):
    return Response('Goodbye world!')

if __name__ == '__main__':
    config = Configurator()
    config.add_view(hello_world)
```

```
    config.add_view(goodbye_world, name='goodbye')
    app = config.make_wsgi_app()
    serve(app, host='0.0.0.0')
```

```
$ python test/helloworld.py
serving on 0.0.0.0:8080 view at http://127.0.0.1:8080

$ curl http://127.0.0.1:8080/
Hello world!

$ curl http://127.0.0.1:8080/goodbye
Goodbye world!
```

MongoDB

```
$ sudo apt-get install python-pymongo python-gridfs
```

```
vim development.ini
[app:test]
# mongodb settings ...
db_uri = mongodb://localhost/
db_name = test
```

测试

```
from pyramid.config import Configurator
from pyramid.events import subscriber
from pyramid.events import NewRequest

from gridfs import GridFS
import pymongo

def main(global_config, **settings):
    config = Configurator(settings=settings)
```

```
    db_uri = settings['db_uri']
    conn = pymongo.Connection(db_uri)
    config.registry.settings['db_conn'] = conn
    config.add_subscriber(add_mongo_db, NewRequest)

    config.add_route('dashboard', '/')
    # other routes and more config...
    config.scan('myapp')

    return config.make_wsgi_app()

def add_mongo_db(event):
    settings = event.request.registry.settings
    db = settings['db_conn'][settings['db_name']]
    event.request.db = db
    event.request.fs = GridFS(db)
```

```
 @view_config(route_name='dashboard',
              renderer="myapp:templates/dashboard.pt")
 def dashboard(request):
     vendors = request.db['vendors'].find()
     return {'vendors':vendors}
```

3.2. Creating a Pyramid Project

```
$ paster create -t pyramid_starter test
$ cd test
$ sudo python setup.py develop
$ paster serve development.ini
```

mongodb

```
vim development.ini
[app:test]
# mongodb settings ...
mongodb_uri = mongodb://localhost/
```

```
mongodb_name = test
```

```
vim test/resources.py

from gridfs import GridFS
import pymongo

mongo_conn = pymongo.Connection()

def add_mongo(event):
    req = event.request
    req.db = mongo_conn['test']
    req.fs = GridFS(req.db)

class Root(object):
    def __init__(self, request):
        self.request = request
```

```
$ vim test/__init__.py

def main(...):
    ...
    config.add_subscriber('foo.resources.add_mongo',
                          'pyramid.events.NewRequest')
    ...
```

例 7.1. __init__.py

```
from pyramid.config import Configurator
from test.resources import Root

def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    config = Configurator(root_factory=Root, settings=settings)

config.add_subscriber('test.resources.add_mongo','pyramid.event
s.NewRequest')
```

```
    config.add_view('test.views.my_view',
                    context='test:resources.Root',
                    renderer='test:templates/mytemplate.pt')
    config.add_static_view('static', 'test:static')
    return config.make_wsgi_app()
```

第 8 章 gevent: A coroutine-based network library for Python

http://www.gevent.org/

第 9 章 Sqlalchemy

1. 安装 Sqlalchemy

```
pip install pymysql
pip install sqlalchemy
```

2. 链接测试

```
from sqlalchemy import create_engine

HOST_NAME = '127.0.0.1' # 主机
PORT = '3306'   # 端口号
DB_NAME = '数据库名称，需提前创建好'
USERNAME = '用户名'
PASSWORD = '密码'

DB_URL = 'mysql+pymysql://{}:{}@{}:{}/{}?
charset=utf8'.format(
    USERNAME, PASSWORD, HOST_NAME, PORT, DB_NAME
)
engine = create_engine(DB_URL)

if __name__ == '__main__':
    connection = engine.connect()
    result = connection.execute('select 1')
    print(result.fetchone())
```

打印 SQL 语句

```
SQLAlchemy 通过 echo=true，将连接这个数据库引擎的所有执行语句打印出
来：

engine = create_engine("<db_rul>", echo=True)
```

3. 创建表

```
models.Base.metadata.create_all(bind=engine)
```

4. Session

```
# 创建数据库会话
Session = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
```

5. 模型定义

导入数据类型

```
from sqlalchemy import Column, Integer, String, Float,
DECIMAL, Boolean, Enum, Date, DateTime, Time
from sqlalchemy.dialects.mysql import LONGTEXT
```

5.1. 定义字段

主键

```
id = Column(Integer, primary_key=True, autoincrement=True)
```

```
price = Column(Float)
# 总共有 20 位,  保留 5 位小说
price = Column(DECIMAL(20, 5))
is_delete = Column(Boolean)
create_time = Column(DateTime)
content = Column(LONGTEXT)
```

指定字段名

```
name = Column('fullname', String(60))
```

```
```

枚举字段

```
gender = Column(Enum('男', '女'))
```

默认值

```
name = Column(String(20), default=None, nullable=False,
comment="姓名")
```

server_default

default 默认是是 sqlalchemy 产生的，我们需要让数据库完成默认值的填充，就需要使用 server_default

```
class PictureBookHasPicture(Base):
    __tablename__ = "picture_book_has_picture"
    id = Column(BIGINT, autoincrement=True, primary_key=True,
comment="主键")
    picture_book_id = Column(
        Integer,
        ForeignKey("picture_book.id", ondelete="CASCADE",
onupdate="CASCADE"),
        nullable=False,
        comment="绘本ID",
    )
    picture_id = Column(
```

```
        BIGINT, ForeignKey("picture.id"), nullable=False,
comment="图片ID"
    )
    ctime = Column(DateTime, server_default=text("now()"),
comment="创建时间")
```

ON UPDATE

```
class PictureBookHasPicture(Base):
    __tablename__ = "picture_book_has_picture"
    id = Column(BIGINT, autoincrement=True, primary_key=True,
comment="主键")
    picture_book_id = Column(
        Integer,
        ForeignKey("picture_book.id", ondelete="CASCADE",
onupdate="CASCADE"),
        nullable=False,
        comment="绘本ID",
    )
    picture_id = Column(
        BIGINT, ForeignKey("picture.id"), nullable=False,
comment="图片ID"
    )
    ctime = Column(DateTime, server_default=text("now()"),
comment="创建时间")
    mtime = Column(
        DateTime,
        server_default=text("NULL ON UPDATE
CURRENT_TIMESTAMP"),
        comment="更新时间",
    )
```

输出结果

```
```

```
CREATE TABLE `picture_book_has_picture` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `picture_book_id` int NOT NULL COMMENT '绘本ID',
  `picture_id` bigint NOT NULL COMMENT '图片ID',
  `ctime` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `mtime` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`),
  KEY `picture_book_id` (`picture_book_id`),
  KEY `picture_id` (`picture_id`),
  CONSTRAINT `picture_book_has_picture_ibfk_1` FOREIGN KEY (`picture_book_id`) REFERENCES `picture_book` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `picture_book_has_picture_ibfk_2` FOREIGN KEY (`picture_id`) REFERENCES `picture` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci
```

给表加注释

给表增加注释 __table_args__ = {"comment": "安卓设备表"}

```
class Android(Base):
    __tablename__ = "android"
    __table_args__ = {"comment": "安卓设备表"}
    android_id = Column(String(16), primary_key=True, comment="安卓设备ID")
    sn = Column(String(18), nullable=False, unique=True, comment="序列号")
    version = Column(String(16), nullable=False, unique=False, comment="软件版本")
    model = Column(String(16), nullable=False, unique=False, comment="产品型号")
    mac = Column(String(48), nullable=True, unique=True, comment="MAC地址")
```

修改记录的时候触发更新

onupdate 修改记录的时候触发更新

```
update_time = Column(DateTime, onupdate=datetime.now())
```

5.2. 外键

```
class Picture(Base):
    __tablename__ = "picture"
    id = Column(BIGINT, autoincrement=True, primary_key=True,
comment="唯一ID")
    android_id = Column(String(16), nullable=False,
comment="安卓设备ID")
    session = Column(String(36), nullable=False, unique=True,
comment="回话ID")
    prompt = Column(String(250), nullable=False, comment="提示
词")
    thumbnail = Column(String(250), nullable=True, comment="缩
图")
    original = Column(String(250), nullable=True, comment="原
始图片")
    image = Column(String(250), nullable=True, comment="图片")
    story = Column(TEXT, nullable=True, comment="故事")
    share = Column(Boolean, nullable=False, default=True,
comment="共享")
    like = Column(INTEGER, nullable=False, default=0,
comment="点赞")
    ctime = Column(DateTime, default=datetime.now(),
comment="创建时间")
    mtime = Column(DateTime, default=datetime.now(),
comment="修改时间")


class PictureLike(Base):
```

```
    __tablename__ = "picture_like"
    id = Column(BIGINT, autoincrement=True, primary_key=True,
comment="唯一ID")
    android_id = Column(String(16), nullable=False,
comment="安卓设备ID")
    picture_id = Column(
        BIGINT, ForeignKey("picture.id"), nullable=False,
comment="图片ID"
    )
    ctime = Column(DateTime, default=datetime.now(),
comment="创建时间")
```

ON DELETE 删除外键约束

```
RESTRICT:        若子表中有父表对应的关联数据，删除父表对应数据，会阻止删
除。默认项
NO ACTION:       在MySQL中，同RESTRICT。
CASCADE:         级联删除。
SET NULL:        父表对应数据被删除，子表对应数据项会设置为NULL。
```

```
class PictureBook(Base):
    __tablename__ = "picture_book"
    id = Column(Integer, autoincrement=True,
primary_key=True, comment="主键")
    title = Column(String(50), nullable=False, comment="绘本名
称")
    author = Column(String(50), nullable=False, comment="作
者")
    cover = Column(String(50), nullable=False, comment="封面")
    description = Column(String(250), nullable=False,
comment="描述")
    copyright = Column(String(250), nullable=False,
comment="版权")
    isbn = Column(String(13), nullable=True, comment="ISBN")
    publisher = Column(String(13), nullable=True, comment="出
```

```
版社")
    editon = Column(String(13), nullable=True, comment="编辑")
    ctime = Column(DateTime, default=datetime.now(),
comment="创建时间")
    mtime = Column(
        DateTime, default=datetime.now(),
onupdate=datetime.now(), comment="修改时间"
    )


class PictureBookHasPicture(Base):
    __tablename__ = "picture_book_has_picture"
    id = Column(BIGINT, autoincrement=True, primary_key=True,
comment="主键")
    picture_book_id = Column(
        Integer, ForeignKey("picture_book.id",
ondelete='CASCADE'), nullable=False, comment="绘本ID"
    )
    picture_id = Column(
        BIGINT, ForeignKey("picture.id"), nullable=False,
comment="图片ID"
    )
    ctime = Column(DateTime, default=datetime.now(),
comment="创建时间")
    mtime = Column(DateTime, default=datetime.now(),
comment="修改时间")
```

输出结果

```
CREATE TABLE `picture_book_has_picture` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `picture_book_id` int NOT NULL COMMENT '绘本ID',
  `picture_id` bigint NOT NULL COMMENT '图片ID',
  `ctime` datetime DEFAULT NULL COMMENT '创建时间',
  `mtime` datetime DEFAULT NULL COMMENT '修改时间',
  PRIMARY KEY (`id`),
  KEY `picture_book_id` (`picture_book_id`),
  KEY `picture_id` (`picture_id`),
  CONSTRAINT `picture_book_has_picture_ibfk_1` FOREIGN KEY
```

```
(`picture_book_id`) REFERENCES `picture_book` (`id`) ON
DELETE CASCADE,
  CONSTRAINT `picture_book_has_picture_ibfk_2` FOREIGN KEY
(`picture_id`) REFERENCES `picture` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci
```

ON UPDATE 更新外键约束

```
class PictureBookHasPicture(Base):
    __tablename__ = "picture_book_has_picture"
    id = Column(BIGINT, autoincrement=True, primary_key=True,
comment="主键")
    picture_book_id = Column(
        Integer,
        ForeignKey("picture_book.id", ondelete="CASCADE",
onupdate="CASCADE"),
        nullable=False,
        comment="绘本ID",
    )
    picture_id = Column(
        BIGINT, ForeignKey("picture.id"), nullable=False,
comment="图片ID"
    )
    ctime = Column(DateTime, server_default=text("now()"),
comment="创建时间")
    mtime = Column(
        DateTime, server_default=None,
server_onupdate=text("now()"), comment="更新时间"
    )
```

```
CREATE TABLE `picture_book_has_picture` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `picture_book_id` int NOT NULL COMMENT '绘本ID',
```

```
  `picture_id` bigint NOT NULL COMMENT '图片ID',
  `ctime` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `mtime` datetime DEFAULT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`),
  KEY `picture_book_id` (`picture_book_id`),
  KEY `picture_id` (`picture_id`),
  CONSTRAINT `picture_book_has_picture_ibfk_1` FOREIGN KEY (`picture_book_id`) REFERENCES `picture_book` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `picture_book_has_picture_ibfk_2` FOREIGN KEY (`picture_id`) REFERENCES `picture` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci
```

6. 增删改

插入数据

```
from sqlalchemy.orm import sessionmaker

# 创建会话session
Session = sessionmaker(bind=engine)
session = Session()
# 新增数据
new_data = Employees(name='Neo', sex='男', age=25,
birth='1980-10-24', jobs='CEO')
session.add(new_data)
session.commit()
session.close()
```

删除数据

```
from sqlalchemy.orm import sessionmaker

# 创建会话session
Session = sessionmaker(bind=engine)
session = Session()
# 删除数据
data = session.query(Employees).filter_by(id=1).delete()
print('已删除数据的数据量为:', data)
session.commit()
session.close()

session.query(Students).filter(Students.name ==
'netkiller').delete()
session.commit()
```

修改数据

```
# 创建会话session
Session = sessionmaker(bind=engine)
session = Session()

# 更新数据
session.query(Employees).filter_by(id=1).update({Employees.ag
e: 12})
session.commit()
session.close()

# 查询后更新数据
data = session.query(Employees).filter_by(id=5).first()
data.name = '张三'
session.commit()
session.close()
```

7. 查询

打印SQL

```
query = session.query(PictureLike).filter(
    PictureLike.android_id == android_id,
    PictureLike.picture_id == picture_id,
)
print(f"SQL: {query}")
```

所有数据

```
results = session.query(Player).all()
for result in results:
    print(f"查询结果为: {result}")
```

第一条数据

```
first = session.query(Player).first()
print(first)
```

LIKE

```
query_result =
```

```
session.query(Player).filter(Player.name.like("%sa%"))
```

与操作符 and

```
query_result = session.query.filter(and_(Player.name ==
'Salah', Player.id > 1))
# 单个filter()中设置多个表达式
query_result = session.query.filter(Player.name == 'Salah',
Player.id > 1)

# 使用多个filter()
query_result = session.query.filter(Player.name ==
'Salah').filter(Player.id > 1)
```

或操作符 or

```
from sqlalchemy import or_

results = session.query.filter(or_(Player.name == 'Salah',
Player.id > 1))
```

IN 包含

```
query_result =
session.query.filter(Player.club.in_(["Liverpool",
"Chelsea"]))
```

NOT IN 排除

```
query_result =
session.query.filter(~Player.country.in_(["Eygpt", "China"]))
```

slice 切片操作，返回 list

```
slice(起始值，结束值)

results = session.query(Arctire).slice(90,100).all()
print(results)

我们也可以使用如下方法，获得同样的结果。
results = session.query(Arctire)[90:100]
```

其他操作

```
avg：求平均值。
max：求最大值。
min：求最小值。
sum：求和。
```

8. 标签

```
        results = (
            session.query(Picture.image.label("image"))
            .filter(Picture.android_id == android_id)
            .all()
        )
        data = [url[0] for url in results]
```

9. 统计数量

```
result = session.query(Students).count()
result = session.query(Students).filter(Students.name ==
'neo').count()
```

10. 排序

```
results =
session.query(Students).order_by(Students.id.desc()).all()
```

11. 查询数据是否存在

11.1. 返回 exists SQL 语句

```
        exist = (
            session.query(PictureLike)
            .filter(
                PictureLike.android_id == android_id,
                PictureLike.picture_id == picture_id,
            )
            .exists()
        )
```

输出结果

```
EXISTS (SELECT 1
FROM picture_like
WHERE picture_like.android_id = :android_id_1 AND
picture_like.picture_id = :picture_id_1)
```

11.2. exists()

```
from sqlalchemy import exists

it_exists = Session.query(
    exists().where( SomeObject.field==value )
).scalar()
```

11.3. query.exists()

```
query = session.query(PictureLike).filter(
    PictureLike.android_id == android_id,
    PictureLike.picture_id == picture_id,
)
print(f"SQL: {query}")
exists = session.query(query.exists()).scalar()
print(exists)
```

12.

```
from sqlalchemy import func
```

12.1. count

```
        count, min, max = (
            session.query(
                func.count().label("count"),
                func.min(Picture.id).label("min"),
                func.max(Picture.id).label("max"),
            )
            .filter(Picture.android_id == android_id)
            .one()
        )
        data = {"count": count, "min": min, "max": max}
```

12.2. min/max

```
# 最大値
result = session.query(func.max(Article.price)).first()
print(result)

# 最小値
result = session.query(func.min(Article.price)).first()
print(result)
```

```
        data = session.query(
            func.min(Picture.id).label("min"),
func.max(Picture.id).label("max")
        ).one()

        min, max = session.query(
            func.min(Picture.id).label("min"),
func.max(Picture.id).label("max")
        ).one()

        data = {"min": min, "max": max}
```

12.3. 平均值/求和

```
# 平均值
result = session.query(func.avg(Article.price)).first()
print(result)
# 求和
result = session.query(func.sum(Article.price)).first()
print(result)
```

12.4.

部分 III. Python 数据分析

第 10 章 Crawler

1. Requests

```
import requests

r = requests.get('https://api.github.com/user', auth=
('netkiller', '******'))
print(r.status_code)
print(r.headers['content-type'])
print(r.encoding)
print(r.text)
print(r.json())
```

第 11 章 Scrapy - Python web scraping and crawling framework

https://scrapy.org

1. 安装 scrapy 开发环境

1.1. Mac

```
neo@MacBook-Pro ~ % brew install python3
neo@MacBook-Pro ~ % pip3 install scrapy
```

1.2. Ubuntu

搜索 scrapy 包，scrapy 支持 Python2.7 和 Python3 我们只需要 python3 版本

```
neo@netkiller ~ % apt-cache search scrapy | grep python3
python3-scrapy - Python web scraping and crawling framework
(Python 3)
python3-scrapy-djangoitem - Scrapy extension to write scraped
items using Django models (Python3 version)
python3-w3lib - Collection of web-related functions (Python 3)
```

Ubuntu 17.04 默认 scrapy 版本为 1.3.0-1 如果需要最新的 1.4.0 请使用 pip 命令安装

```
neo@netkiller ~ % apt search python3-scrapy
Sorting... Done
Full Text Search... Done
python3-scrapy/zesty,zesty 1.3.0-1~exp2 all
  Python web scraping and crawling framework (Python 3)
```

```
python3-scrapy-djangoitem/zesty,zesty 1.1.1-1 all
  Scrapy extension to write scraped items using Django models
(Python3 version)
```

安装 scrapy

```
neo@netkiller ~ % sudo apt install python3-scrapy
[sudo] password for neo:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ipython3 libmysqlclient20 libwebpmux2 mysql-common python-
pexpect python-ptyprocess python3-attr python3-boto python3-bs4
python3-cffi-backend python3-click python3-colorama python3-
constantly
  python3-cryptography python3-cssselect python3-decorator
python3-html5lib python3-idna python3-incremental python3-
ipython python3-ipython-genutils python3-libxml2 python3-lxml
python3-mysqldb
  python3-openssl python3-pam python3-parsel python3-pexpect
python3-pickleshare python3-pil python3-prompt-toolkit python3-
ptyprocess python3-pyasn1 python3-pyasn1-modules python3-
pydispatch
  python3-pygments python3-queuelib python3-serial python3-
service-identity python3-setuptools python3-simplegeneric
python3-traitlets python3-twisted python3-twisted-bin python3-
w3lib python3-wcwidth
  python3-webencodings python3-zope.interface
Suggested packages:
  python-pexpect-doc python-attr-doc python-cryptography-doc
python3-cryptography-vectors python3-genshi python3-lxml-dbg
python-lxml-doc default-mysql-server | virtual-mysql-server
  python-egenix-mxdatetime python3-mysqldb-dbg python-openssl-
doc python3-openssl-dbg python3-pam-dbg python-pil-doc python3-
pil-dbg doc-base python-pydispatch-doc ttf-bitstream-vera
python-scrapy-doc
  python3-wxgtk3.0 | python3-wxgtk python-setuptools-doc
python3-tk python3-gtk2 python3-glade2 python3-qt4 python3-
wxgtk2.8 python3-twisted-bin-dbg
The following NEW packages will be installed:
```

```
  ipython3 libmysqlclient20 libwebpmux2 mysql-common python-
pexpect python-ptyprocess python3-attr python3-boto python3-bs4
python3-cffi-backend python3-click python3-colorama python3-
constantly
  python3-cryptography python3-cssselect python3-decorator
python3-html5lib python3-idna python3-incremental python3-
ipython python3-ipython-genutils python3-libxml2 python3-lxml
python3-mysqldb
  python3-openssl python3-pam python3-parsel python3-pexpect
python3-pickleshare python3-pil python3-prompt-toolkit python3-
ptyprocess python3-pyasn1 python3-pyasn1-modules python3-
pydispatch
  python3-pygments python3-queuelib python3-scrapy python3-
serial python3-service-identity python3-setuptools python3-
simplegeneric python3-traitlets python3-twisted python3-
twisted-bin python3-w3lib
  python3-wcwidth python3-webencodings python3-zope.interface
0 upgraded, 49 newly installed, 0 to remove and 0 not upgraded.
Need to get 7,152 kB of archives.
After this operation, 40.8 MB of additional disk space will be
used.
Do you want to continue? [Y/n]
```

输入大写 “Y” 然后回车

1.3. 使用 pip 安装 scrapy

```
neo@netkiller ~ % sudo apt install python3-pip
neo@netkiller ~ % pip3 install scrapy
```

1.4. 测试 scrapy

创建测试程序，用于验证 scrapy 安装是否存在问题。

```
$ cat > myspider.py <<EOF
import scrapy
```

```
class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://blog.scrapinghub.com']

    def parse(self, response):
        for title in response.css('h2.entry-title'):
            yield {'title': title.css('a
::text').extract_first()}

        for next_page in response.css('div.prev-post > a'):
            yield response.follow(next_page, self.parse)
EOF
```

运行爬虫

```
$ scrapy runspider myspider.py
```

2. scrapy 命令

```
neo@MacBook-Pro ~/Documents/crawler % scrapy
Scrapy 1.4.0 - project: crawler

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  check         Check spider contracts
  crawl         Run a spider
  edit          Edit spider
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  list          List available spiders
  parse         Parse URL (using its spider) and print the
results
  runspider     Run a self-contained spider (without creating a
project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy

Use "scrapy <command> -h" to see more info about a command
```

2.1.

```
neo@MacBook-Pro ~/Documents % scrapy startproject crawler
New Scrapy project 'crawler', using template directory
'/usr/local/lib/python3.6/site-
packages/scrapy/templates/project', created in:
    /Users/neo/Documents/crawler
```

```
You can start your first spider with:
    cd crawler
    scrapy genspider example example.com
```

2.2. 新建 spider

```
neo@MacBook-Pro ~/Documents/crawler % scrapy genspider
netkiller netkiller.cn
Created spider 'netkiller' using template 'basic' in module:
  crawler.spiders.netkiller
```

2.3. 列出可用的 spiders

```
neo@MacBook-Pro ~/Documents/crawler % scrapy list
bing
book
example
netkiller
```

2.4. 运行 spider

```
neo@MacBook-Pro ~/Documents/crawler % scrapy crawl netkiller
```

运行结果输出到 json 文件中

```
```

```
neo@MacBook-Pro ~/Documents/crawler % scrapy crawl netkiller -o
output.json
```

3. Scrapy Shell

Scrapy Shell 是一个爬虫命令行交互界面调试工具，可以使用它分析被爬的页面

```
neo@MacBook-Pro /tmp % scrapy shell http://www.netkiller.cn
2017-09-01 15:23:05 [scrapy.utils.log] INFO: Scrapy 1.4.0
started (bot: scrapybot)
2017-09-01 15:23:05 [scrapy.utils.log] INFO: Overridden
settings: {'DUPEFILTER_CLASS':
'scrapy.dupefilters.BaseDupeFilter', 'LOGSTATS_INTERVAL': 0}
2017-09-01 15:23:05 [scrapy.middleware] INFO: Enabled
extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.memusage.MemoryUsage']
2017-09-01 15:23:05 [scrapy.middleware] INFO: Enabled
downloader middlewares:
['scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',

'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMi
ddleware',

'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMidd
leware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',

'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMi
ddleware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
 'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
2017-09-01 15:23:05 [scrapy.middleware] INFO: Enabled spider
middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
```

```
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2017-09-01 15:23:05 [scrapy.middleware] INFO: Enabled item
pipelines:
[]
2017-09-01 15:23:05 [scrapy.extensions.telnet] DEBUG: Telnet
console listening on 127.0.0.1:6023
2017-09-01 15:23:05 [scrapy.core.engine] INFO: Spider opened
2017-09-01 15:23:05 [scrapy.core.engine] DEBUG: Crawled (200)
<GET http://www.netkiller.cn> (referer: None)
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request,
scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x103b2afd0>
[s]   item       {}
[s]   request    <GET http://www.netkiller.cn>
[s]   response   <200 http://www.netkiller.cn>
[s]   settings   <scrapy.settings.Settings object at
0x1049019e8>
[s]   spider     <DefaultSpider 'default' at 0x104be2a90>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local
objects (by default, redirects are followed)
[s]   fetch(req)                  Fetch a scrapy.Request and
update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser
>>>
```

3.1. response

response 是爬虫返回的页面，可以通过 css(), xpath() 等方法取出你需要的内容。

当前URL地址

```
>>> response.url
'https://netkiller.cn/linux/index.html'
```

```
```

status HTTP 状态

```
>>> response.status
200
```

text 正文

返回 HTML 页面正文

```
response.text
```

css

css() 这个方法可以用来选择html和css

```
>>> response.css('title')
[<Selector xpath='descendant-or-self::title'
data='<title>Netkiller ebook - Linux ebook</ti'>]

>>> response.css('title').extract()
['<title>Netkiller ebook - Linux ebook</title>']

>>> response.css('title::text').extract()
['Netkiller ebook - Linux ebook']
```

基于 class 选择

```
>>> response.css('a.ulink')[1].extract()
'<a class="ulink" href="http://netkiller.github.io/"
target="_top">http://netkiller.github.io</a>'

>>> response.css('a.ulink::text')[3].extract()
'http://netkiller.sourceforge.net'
```

数组的处理

```
>>> response.css('a::text').extract_first()
'简体中文'

>>> response.css('a::text')[1].extract()
'繁体中文'

>>> response.css('div.blockquote')
[1].css('a.ulink::text').extract()
['Netkiller Architect 手札', 'Netkiller Developer 手札',
'Netkiller PHP 手札', 'Netkiller Python 手札', 'Netkiller
Testing 手札', 'Netkiller Java 手札', 'Netkiller Cryptography 手
札', 'Netkiller Linux 手札', 'Netkiller FreeBSD 手札', 'Netkiller
Shell 手札', 'Netkiller Security 手札', 'Netkiller Web 手札',
'Netkiller Monitoring 手札', 'Netkiller Storage 手札',
'Netkiller Mail 手札', 'Netkiller Docbook 手札', 'Netkiller
Project 手札', 'Netkiller Database 手札', 'Netkiller PostgreSQL
手札', 'Netkiller MySQL 手札', 'Netkiller NoSQL 手札', 'Netkiller
LDAP 手札', 'Netkiller Network 手札', 'Netkiller Cisco IOS 手札',
'Netkiller H3C 手札', 'Netkiller Multimedia 手札', 'Netkiller
Perl 手札', 'Netkiller Amateur Radio 手札']
```

正则表达式

```
>>> response.css('title::text').re(r'Netkiller.*')
['Netkiller ebook - Linux ebook']
```

```
>>> response.css('title::text').re(r'N\w+')
['Netkiller']

>>> response.css('title::text').re(r'(\w+) (\w+)')
['Netkiller', 'ebook', 'Linux', 'ebook']
```

获取 html 属性

通过 a::attr() 可以获取 html 标记的属性值

```
>>> response.css('td a::attr(href)').extract_first()
'http://netkiller.github.io/'
```

xpath

```
>>> response.xpath('//title')
[<Selector xpath='//title' data='<title>Netkiller ebook - Linux
ebook</ti'>]

>>> response.xpath('//title/text()').extract_first()
'Netkiller ebook - Linux ebook'
```

xpath 也可以使用 re() 方法做正则处理

```
>>> response.xpath('//title/text()').re(r'(\w+)')
['Netkiller', 'ebook', 'Linux', 'ebook']

>>> response.xpath('//div[@class="time"]/text()').re('[0-9]{4}-
[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}')
```

```
['2017-09-21 02:01:38']
```

抽取HTML属性值，如图片URL。

```
>>> response.xpath('//img/@src').extract()
['graphics/spacer.gif', 'graphics/note.gif', 'graphics/by-nc-
sa.png', '/images/weixin.jpg', 'images/neo.jpg',
'/images/weixin.jpg']
```

筛选 class

```
>>> response.xpath('//a/@href')[0].extract()
'http://netkiller.github.io/'

>>> response.xpath('//a/text()')[0].extract()
'简体中文'

>>> response.xpath('//div[@class="blockquote"]')
[1].css('a.ulink::text').extract()
['Netkiller Architect 手札', 'Netkiller Developer 手札',
'Netkiller PHP 手札', 'Netkiller Python 手札', 'Netkiller
Testing 手札', 'Netkiller Java 手札', 'Netkiller Cryptography 手
札', 'Netkiller Linux 手札', 'Netkiller FreeBSD 手札', 'Netkiller
Shell 手札', 'Netkiller Security 手札', 'Netkiller Web 手札',
'Netkiller Monitoring 手札', 'Netkiller Storage 手札',
'Netkiller Mail 手札', 'Netkiller Docbook 手札', 'Netkiller
Project 手札', 'Netkiller Database 手札', 'Netkiller PostgreSQL
手札', 'Netkiller MySQL 手札', 'Netkiller NoSQL 手札', 'Netkiller
LDAP 手札', 'Netkiller Network 手札', 'Netkiller Cisco IOS 手札',
'Netkiller H3C 手札', 'Netkiller Multimedia 手札', 'Netkiller
Perl 手札', 'Netkiller Amateur Radio 手札']
```

使用 | 匹配多组规则

```
>>>
response.xpath('//ul[@class="topnews_nlist"]/li/h2/a/@href|//ul
[@class="topnews_nlist"]/li/a/@href').extract()
```

headers

```
response.headers.getlist('Set-Cookie')
```

4. 爬虫项目

4.1. 创建项目

创建爬虫项目

```
scrapy startproject project
```

在抓取之前，你需要新建一个Scrapy工程

```
neo@MacBook-Pro ~/Documents % scrapy startproject crawler
New Scrapy project 'crawler', using template directory
'/usr/local/lib/python3.6/site-
packages/scrapy/templates/project', created in:
    /Users/neo/Documents/crawler

You can start your first spider with:
    cd crawler
    scrapy genspider example example.com

neo@MacBook-Pro ~/Documents % cd crawler
neo@MacBook-Pro ~/Documents/crawler % find .
.
./crawler
./crawler/__init__.py
./crawler/__pycache__
./crawler/items.py
./crawler/middlewares.py
./crawler/pipelines.py
./crawler/settings.py
./crawler/spiders
./crawler/spiders/__init__.py
./crawler/spiders/__pycache__
./scrapy.cfg
```

Scrapy 工程目录主要有以下文件组成:

```
scrapy.cfg: 项目配置文件
middlewares.py : 项目 middlewares 文件
items.py: 项目items文件
pipelines.py: 项目管道文件
settings.py: 项目配置文件
spiders: 放置spider的目录
```

4.2. Spider

创建爬虫，名字是 netkiller, 爬行的地址是 netkiller.cn

```
neo@MacBook-Pro ~/Documents/crawler % scrapy genspider
netkiller netkiller.cn
Created spider 'netkiller' using template 'basic' in module:
  crawler.spiders.netkiller
neo@MacBook-Pro ~/Documents/crawler % find .
.
./crawler
./crawler/__init__.py
./crawler/__pycache__
./crawler/__pycache__/__init__.cpython-36.pyc
./crawler/__pycache__/settings.cpython-36.pyc
./crawler/items.py
./crawler/middlewares.py
./crawler/pipelines.py
./crawler/settings.py
./crawler/spiders
./crawler/spiders/__init__.py
./crawler/spiders/__pycache__
./crawler/spiders/__pycache__/__init__.cpython-36.pyc
./crawler/spiders/netkiller.py
./scrapy.cfg
```

打开 crawler/spiders/netkiller.py 文件，修改内容如下

```
# -*- coding: utf-8 -*-
import scrapy


class NetkillerSpider(scrapy.Spider):
    name = 'netkiller'
    allowed_domains = ['netkiller.cn']
    start_urls = ['http://www.netkiller.cn/']

    def parse(self, response):
        for link in
response.xpath('//div[@class="blockquote"]')[1].css('a.ulink'):
            # self.log('This url is %s' % link)
            yield {
                'name': link.css('a::text').extract(),
                'url':
link.css('a.ulink::attr(href)').extract()
                }

        pass
```

运行爬虫

```
neo@MacBook-Pro ~/Documents/crawler % scrapy crawl netkiller -o
output.json
2017-09-08 11:42:30 [scrapy.utils.log] INFO: Scrapy 1.4.0
started (bot: crawler)
2017-09-08 11:42:30 [scrapy.utils.log] INFO: Overridden
settings: {'BOT_NAME': 'crawler', 'FEED_FORMAT': 'json',
'FEED_URI': 'output.json', 'NEWSPIDER_MODULE':
'crawler.spiders', 'ROBOTSTXT_OBEY': True, 'SPIDER_MODULES':
['crawler.spiders']}
2017-09-08 11:42:30 [scrapy.middleware] INFO: Enabled
extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.memusage.MemoryUsage',
 'scrapy.extensions.feedexport.FeedExporter',
```

```
 'scrapy.extensions.logstats.LogStats']
2017-09-08 11:42:30 [scrapy.middleware] INFO: Enabled
downloader middlewares:
['scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware',
 'scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',

'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMi
ddleware',

'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMidd
leware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',

'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMi
ddleware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
 'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
2017-09-08 11:42:30 [scrapy.middleware] INFO: Enabled spider
middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2017-09-08 11:42:30 [scrapy.middleware] INFO: Enabled item
pipelines:
[]
2017-09-08 11:42:30 [scrapy.core.engine] INFO: Spider opened
2017-09-08 11:42:30 [scrapy.extensions.logstats] INFO: Crawled
0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2017-09-08 11:42:30 [scrapy.extensions.telnet] DEBUG: Telnet
console listening on 127.0.0.1:6023
2017-09-08 11:42:30 [scrapy.core.engine] DEBUG: Crawled (200)
<GET http://www.netkiller.cn/robots.txt> (referer: None)
2017-09-08 11:42:31 [scrapy.core.engine] DEBUG: Crawled (200)
<GET http://www.netkiller.cn/> (referer: None)
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Architect 手札'], 'url':
['../architect/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
```

```
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Developer 手札'], 'url':
['../developer/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller PHP 手札'], 'url': ['../php/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Python 手札'], 'url':
['../python/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Testing 手札'], 'url':
['../testing/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Java 手札'], 'url': ['../java/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Cryptography 手札'], 'url':
['../cryptography/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Linux 手札'], 'url':
['../linux/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller FreeBSD 手札'], 'url':
['../freebsd/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Shell 手札'], 'url':
['../shell/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Security 手札'], 'url':
['../security/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Web 手札'], 'url': ['../www/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Monitoring 手札'], 'url':
['../monitoring/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
```

```
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Storage 手札'], 'url':
['../storage/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Mail 手札'], 'url': ['../mail/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Docbook 手札'], 'url':
['../docbook/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Project 手札'], 'url':
['../project/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Database 手札'], 'url':
['../database/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller PostgreSQL 手札'], 'url':
['../postgresql/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller MySQL 手札'], 'url':
['../mysql/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller NoSQL 手札'], 'url':
['../nosql/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller LDAP 手札'], 'url': ['../ldap/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Network 手札'], 'url':
['../network/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Cisco IOS 手札'], 'url':
['../cisco/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller H3C 手札'], 'url': ['../h3c/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
```

```
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Multimedia 手札'], 'url':
['../multimedia/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Perl 手札'], 'url': ['../perl/index.html']}
2017-09-08 11:42:31 [scrapy.core.scraper] DEBUG: Scraped from
<200 http://www.netkiller.cn/>
{'name': ['Netkiller Amateur Radio 手札'], 'url':
['../radio/index.html']}
2017-09-08 11:42:31 [scrapy.core.engine] INFO: Closing spider
(finished)
2017-09-08 11:42:31 [scrapy.extensions.feedexport] INFO: Stored
json feed (28 items) in: output.json
2017-09-08 11:42:31 [scrapy.statscollectors] INFO: Dumping
Scrapy stats:
{'downloader/request_bytes': 438,
 'downloader/request_count': 2,
 'downloader/request_method_count/GET': 2,
 'downloader/response_bytes': 6075,
 'downloader/response_count': 2,
 'downloader/response_status_count/200': 2,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2017, 9, 8, 3, 42, 31,
157395),
 'item_scraped_count': 28,
 'log_count/DEBUG': 31,
 'log_count/INFO': 8,
 'memusage/max': 49434624,
 'memusage/startup': 49434624,
 'response_received_count': 2,
 'scheduler/dequeued': 1,
 'scheduler/dequeued/memory': 1,
 'scheduler/enqueued': 1,
 'scheduler/enqueued/memory': 1,
 'start_time': datetime.datetime(2017, 9, 8, 3, 42, 30,
931267)}
2017-09-08 11:42:31 [scrapy.core.engine] INFO: Spider closed
(finished)
```

你会看到返回结果

```
{'name': ['Netkiller Architect 手札'], 'url':
['../architect/index.html']}
```

翻页操作

下面我们演示爬虫翻页，例如我们需要遍历这部电子书《Netkiller Linux 手札》 https://netkiller.cn/linux/index.html，首先创建一个爬虫任务

```
neo@MacBook-Pro ~/Documents/crawler % scrapy genspider book
netkiller.cn
Created spider 'book' using template 'basic' in module:
  crawler.spiders.book
```

编辑爬虫任务

```
# -*- coding: utf-8 -*-
import scrapy


class BookSpider(scrapy.Spider):
    name = 'book'
    allowed_domains = ['netkiller.cn']
    start_urls = ['https://netkiller.cn/linux/index.html']

    def parse(self, response):
        yield {'title': response.css('title::text').extract()}
        # 这里取出下一页连接地址
        next_page =
response.xpath('//a[@accesskey="n"]/@href').extract_first()
        self.log('Next page: %s' % next_page)
        # 如果页面不为空交给 response.follow 来爬取这个页面
```

```
        if next_page is not None:
            yield response.follow(next_page,
callback=self.parse)

        pass
```

采集内容保存到文件

下面的例子是将 response.body 返回采集内容保存到文件中

```
# -*- coding: utf-8 -*-
import scrapy


class BookSpider(scrapy.Spider):
    name = 'book'
    allowed_domains = ['netkiller.cn']
    start_urls = ['https://netkiller.cn/linux/index.html']

    def parse(self, response):
        yield {'title': response.css('title::text').extract()}

        filename = '/tmp/%s' % response.url.split("/")[-1]

        with open(filename, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s' % filename)

        next_page =
response.xpath('//a[@accesskey="n"]/@href').extract_first()
        self.log('Next page: %s' % next_page)
        if next_page is not None:
            yield response.follow(next_page,
callback=self.parse)

        pass
```

任务运维结束后查看采集出来的文件

```
neo@MacBook-Pro ~/Documents/crawler % ls /tmp/*.html
/tmp/apt-get.html            /tmp/disc.html
/tmp/infomation.html         /tmp/lspci.html
/tmp/svgatextmode.html
/tmp/aptitude.html           /tmp/dmidecode.html
/tmp/install.html            /tmp/lsscsi.html
/tmp/swap.html
/tmp/author.html             /tmp/do-release-upgrade.html
/tmp/install.partition.html  /tmp/lsusb.html
/tmp/sys.html
/tmp/avlinux.html            /tmp/dpkg.html
/tmp/introduction.html       /tmp/package.html
/tmp/sysctl.html
/tmp/centos.html             /tmp/du.max-depth.html
/tmp/kernel.html             /tmp/pr01s02.html
/tmp/system.infomation.html
/tmp/cfdisk.html             /tmp/ethtool.html
/tmp/kernel.modules.html     /tmp/pr01s03.html
/tmp/system.profile.html
/tmp/console.html            /tmp/framebuffer.html
/tmp/kudzu.html              /tmp/pr01s05.html
/tmp/system.shutdown.html
/tmp/console.timeout.html    /tmp/gpt.html
/tmp/linux.html              /tmp/preface.html
/tmp/tune2fs.html
/tmp/dd.clone.html           /tmp/hdd.label.html
/tmp/locale.html             /tmp/proc.html
/tmp/udev.html
/tmp/deb.html                /tmp/hdd.partition.html
/tmp/loop.html               /tmp/rpm.html
/tmp/upgrades.html
/tmp/device.cpu.html         /tmp/hwinfo.html
/tmp/lsblk.html              /tmp/rpmbuild.html
/tmp/yum.html
/tmp/device.hba.html         /tmp/index.html
/tmp/lshw.html               /tmp/smartctl.html
```

这里只是做演示，生产环境请不要在 parse(self, response) 中处理，后面会讲到 Pipeline。

4.3. settings.py 爬虫配置文件

忽略 robots.txt 规则

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

4.4. Item

Item 在 scrapy 中的类似“实体”或者“POJO”的概念，是一个数据结构类。爬虫通过ItemLoader将数据放到Item中

下面是 items.py 文件

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class CrawlerItem(scrapy.Item):
    # define the fields for your item here like:
    title = scrapy.Field()
    author = scrapy.Field()
    content = scrapy.Field()
    ctime = scrapy.Field()

    pass
```

下面是爬虫文件

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy.loader import ItemLoader
from crawler.items import CrawlerItem
import time

class ExampleSpider(scrapy.Spider):
    name = 'example'
    allowed_domains = ['netkiller.cn']
    start_urls = ['https://netkiller.cn/java/index.html']
    def parse(self, response):

        item_selector = response.xpath('//a/@href')
        for url in item_selector.extract():
            if 'html' in url.split('.'):
                url = response.urljoin(url)
                yield response.follow( url,
callback=self.parse_item)

        next_page =
response.xpath('//a[@accesskey="n"]/@href').extract_first()
        self.log('Next page: %s' % next_page)
        if next_page is not None:
            yield response.follow(next_page,
callback=self.parse)

    def parse_item(self, response):
        l = ItemLoader(item=CrawlerItem(), response=response)
        l.add_css('title', 'title::text')
        l.add_value('ctime', time.strftime( '%Y-%m-%d %X',
time.localtime() ))
        l.add_value('content', response.body)
        return l.load_item()
```

yield response.follow(url, callback=self.parse_item) 会回调 parse_item(self, response) 将爬到的数据放置到 Item 中

4.5. Pipeline

Pipeline 管道线，主要的功能是对 Item 的数据处理，例如计算、合并等等。通常我们在这里做数据保存。下面的例子是将爬到的数据保存到 json 文件中。

默认情况 Pipeline 是禁用的，首先我们需要开启 Pipeline 支持，修改 settings.py 文件，找到下面配置项，去掉注释。

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-
pipeline.html
ITEM_PIPELINES = {
    'crawler.pipelines.CrawlerPipeline': 300,
}
```

修改 pipelines.py 文件。

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES
setting
# See: http://doc.scrapy.org/en/latest/topics/item-
pipeline.html

import json

class CrawlerPipeline(object):
    def open_spider(self, spider):
        self.file = open('items.json', 'w')
```

```
    def close_spider(self, spider):
        self.file.close()
    def process_item(self, item, spider):
        # self.log("PIPE: %s" % item)
        line = json.dumps(dict(item)) + "\n"
        self.file.write(line)
        return item
```

下面是 items.py 文件

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class CrawlerItem(scrapy.Item):
    # define the fields for your item here like:
    title = scrapy.Field()
    author = scrapy.Field()
    content = scrapy.Field()
    ctime = scrapy.Field()

    pass
```

下面是爬虫文件

```
# -*- coding: utf-8 -*-
import scrapy
```

```
from scrapy.loader import ItemLoader
from crawler.items import CrawlerItem
import time

class ExampleSpider(scrapy.Spider):
    name = 'example'
    allowed_domains = ['netkiller.cn']
    start_urls = ['https://netkiller.cn/java/index.html']
    def parse(self, response):

        item_selector = response.xpath('//a/@href')
        for url in item_selector.extract():
            if 'html' in url.split('.'):
                url = response.urljoin(url)
                yield response.follow( url,
callback=self.parse_item)

        next_page =
response.xpath('//a[@accesskey="n"]/@href').extract_first()
        self.log('Next page: %s' % next_page)
        if next_page is not None:
            yield response.follow(next_page,
callback=self.parse)

    def parse_item(self, response):
        l = ItemLoader(item=CrawlerItem(), response=response)
        l.add_css('title', 'title::text')
        l.add_value('ctime', time.strftime( '%Y-%m-%d %X',
time.localtime() ))
        l.add_value('content', response.body)
        return l.load_item()
```

items.json 文件如下


```
{"title": ["5.31.\u00a0Spring boot with Data restful"],
"ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.30.\u00a0Spring boot with Phoenix"], "ctime":
["2017-09-11 11:57:53"]}
{"title": ["5.29.\u00a0Spring boot with Apache Hive"], "ctime":
["2017-09-11 11:57:53"]}
```

{"title": ["5.28.\u00a0Spring boot with Elasticsearch 5.5.x"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.27.\u00a0Spring boot with Elasticsearch 2.x"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.23.\u00a0Spring boot with Hessian"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.22.\u00a0Spring boot with Cache"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.26.\u00a0Spring boot with HTTPS SSL"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.25.\u00a0Spring boot with Git version"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.24.\u00a0Spring boot with Apache Kafka"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.21.\u00a0Spring boot with Scheduling"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.20.\u00a0Spring boot with Oauth2"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.19.\u00a0Spring boot with Spring security"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.16.\u00a0Spring boot with PostgreSQL"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.18.\u00a0Spring boot with Velocity template"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.13.\u00a0Spring boot with MongoDB"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.11.\u00a0Spring boot with Session share"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.17.\u00a0Spring boot with Email"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.15.\u00a0Spring boot with Oracle"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.14.\u00a0Spring boot with MySQL"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.10.\u00a0Spring boot with Logging"], "ctime": ["2017-09-11 11:57:53"]}
{"title": ["5.9.\u00a0String boot with RestTemplate"], "ctime": ["2017-09-11 11:57:53"]}

5. 下载图片

创建项目

```
neo@MacBook-Pro ~/Documents % scrapy startproject photo
```

```
neo@MacBook-Pro ~/Documents % cd photo
```

安装依赖库

```
neo@MacBook-Pro ~/Documents/photo % pip3 install image
```

创建爬虫

```
neo@MacBook-Pro ~/Documents/photo % scrapy genspider jiandan
jandan.net
```

5.1. 配置 settings.py

忽略 robots.txt 规则

```
# Obey robots.txt rules
```

```
ROBOTSTXT_OBEY = False
```

配置图片保存路径与缩图

```
#图片保存路径
IMAGES_STORE='/tmp/photo'
#DOWNLOAD_DELAY = 0.25
#缩略图的尺寸，设置这个值就会产生缩略图
IMAGES_THUMBS = {
    'small': (50, 50),
    'big': (200, 200),
}
```

5.2. 修改 pipelines.py 文件

加入 process_item（）与 item_completed（） 方法

注意：PhotoPipeline(ImagesPipeline) 需要继承 ImagesPipeline

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES
setting
# See: http://doc.scrapy.org/en/latest/topics/item-
pipeline.html

import scrapy
from scrapy.pipelines.images import ImagesPipeline
from scrapy.exceptions import DropItem

class PhotoPipeline(ImagesPipeline):
    # def process_item(self, item, spider):
        # return item
```

```
    def get_media_requests(self, item, info):
        for image_url in item['image_urls']:
            yield scrapy.http.Request('http:'+image_url)

    def item_completed(self, results, item, info):
        image_paths = [x['path'] for ok, x in results if ok]
        if not image_paths:
            raise DropItem("Item contains no images")
        item['image_paths'] = image_paths
        return item
```

5.3. 编辑 items.py

忽略 robots.txt 规则

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


class PhotoItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    #图片的链接
    image_urls = scrapy.Field()
    images = scrapy.Field()
    image_paths = scrapy.Field()
    pass
```

5.4. Spider 爬虫文件

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy.loader import ItemLoader
from photo.items import PhotoItem

class JiandanSpider(scrapy.Spider):
    name = 'jiandan'
    # allowed_domains = ['jandan.net']
    allowed_domains = []
    start_urls = ['http://jandan.net/ooxx']

    def parse(self, response):

        l = ItemLoader(item=PhotoItem(), response=response)
        l.add_xpath('image_urls','//img//@src' )
        yield l.load_item()

        next_page = response.xpath('//a[@class="previous-
comment-page"]//@href').extract_first() #翻页
        if next_page:
            yield response.follow(next_page,self.parse)
        pass
    def parse_page(self, response):
        l = ItemLoader(item=PhotoItem(), response=response)
        l.add_xpath('image_urls','//img//@src' )
        return l.load_item()
```

6. xpath

6.1. 逻辑运算符

and

```
>>> response.xpath('//span[@class="time" and @id="news-
time"]/text()').extract()
['2017-10-09 09:46']
```

or

```
//*[@class='foo' or contains(@class,' foo ') or starts-
with(@class,'foo ') or substring(@class,string-
length(@class)-3)=' foo']
```

6.2. function

text()

```
>>> response.xpath('//title/text()').extract_first()
'Netkiller ebook - Linux ebook'
```

contains()

contains() 匹配含有特定字符串的 class

```
//*[contains(@class,'foo')]
```

```
>>> response.xpath('//ul[contains(@class,
"topnews_nlist")]/li/h2/a/@href|//ul[contains(@class,
"topnews_nlist")]/li/a/@href').extract()
```

内容匹配

```
>>>
response.xpath('//div[@id="epContentLeft"]/h1[contains(text(),"
10")]/text()').extract()
['美联储10月起启动渐进式缩表 维持基准利率不变']
```

第 12 章 Pandas - Python Data Analysis Library

Powerful data structures for data analysis, time series, and statistics

1. 安装 pandas

```
pip install pandas
```

安装 modin 加速插件

```
pip install modin[dask]
```

modin 的用法是将:

```
import pandas as pd
改为
import modin.pandas as pd
```

2. 数据输入与输出

我们都知道 Pandas 是做数据分析的，Pandas 在做数据分析之前需要加载数据，加载数据的方法有很多种，数据来源渠道也很多。例如数据可以从 HTML页面中的表格，Excel，JSON，CSV以及关系型数据库等等。

2.1. Pandas 处理 HTML

HTML 表格处理

工作中，我们常常需要提取HTML网页中的表格数据，多少会用到爬虫技术。

例如使用 requests 下载网页，然后使用HTML萃取工具，将HTML页面内部table表格中的数据提出去来。这种方法比较复杂，今天介绍的是 Pandas 读取网页中的表格，难度几乎是傻瓜级别的。

在不使用爬虫的情况下，这种方式是最佳选择。

安装依赖包

```
pip install lxml
```

read_html 参数详解

read_html() 可以萃取 HTML table 标签中的数据。

pandas.read_html(io, match='.+', flavor=None, header=None, index_col=None, skiprows=None, attrs=None, parse_dates=False, tupleize_cols=None, thousands=', ', encoding=None, decimal='.',

converters=None, na_values=None, keep_default_na=True, displayed_only=True)

详细参数

```
io:              str, path object 或 file-like objectURL, file-like对象或包含HTML的原始字符串。请注意，lxml仅接受http，ftp和文件url协议。如果您的网址以'https'您可以尝试删除's'。
match:                   str 或 compiled regular expression, 可选参数将返回包含与该正则表达式或字符串匹配的文本的表集。除非HTML非常简单，否则您可能需要在此处传递非空字符串。默认为“。+”(匹配任何非空字符串)。默认值将返回页面上包含的所有表。此值转换为正则表达式，以便Beautiful Soup和lxml之间具有一致的行为。
flavor:          str 或 None要使用的解析引擎。 ‘bs4’和‘html5lib’彼此同义，它们都是为了向后兼容。默认值None尝试使用lxml解析，如果失败，它会重新出现bs4+html5lib。
header:          int 或 list-like 或 None, 可选参数该行(或MultiIndex)用于创建列标题。
index_col:  int 或 list-like 或 None, 可选参数用于创建索引的列(或列列表)。
skiprows:        int 或 list-like 或 slice 或 None, 可选参数解析列整数后要跳过的行数。从0开始。如果给出整数序列或切片，将跳过该序列索引的行。请注意，单个元素序列的意思是“跳过第n行”，而整数的意思是“跳过n行”。
attrs:                   dict 或 None, 可选参数这是属性的词典，您可以传递该属性以用于标识HTML中的表。在传递给lxml或Beautiful Soup之前，不会检查它们的有效性。但是，这些属性必须是有效的HTML表属性才能正常工作。例如，attrs = {'id': 'table'} 是有效的属性字典，因为‘id’ HTML标记属性是任何HTML标记的有效HTML属性，这个文件。 attrs = {'asdf': 'table'} 不是有效的属性字典，因为‘asdf’即使是有效的XML属性，也不是有效的HTML属性。可以找到有效的HTML 4.01表属性这里。可以找到HTML 5规范的工作草案这里。它包含有关现代Web表属性的最新信息。
parse_dates:  bool, 可选参数参考read_csv()更多细节。
thousands:       str, 可选参数用来解析成千上万个分隔符。默认为','。
encoding:        str 或 None, 可选参数用于解码网页的编码。默认为NoneNone保留先前的编码行为，这取决于基础解析器库(例如，解析器库将尝试使用文档提供的编码)。
decimal:         str, 默认为 ‘.’可以识别为小数点的字符(例如，对于欧洲数据，请使用“，”)。
converters:  dict, 默认为 None用于在某些列中转换值的函数的字典。键可以是整数或列标签，值是采用一个输入参数，单元格(而非列)内容并返回转换后内容的函数。
na_values:       iterable, 默认为 None自定义NA值。
```

```
keep_default_na： bool，默认为 True如果指定了na_values并且
keep_default_na为False，则默认的NaN值将被覆盖，否则将附加它们。
displayed_only： bool，默认为 True是否应解析具有“display:none”的元
素。
```

从文本变量中提取数据

```
import pandas as pd

html = """
<table border="1">
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
</table>
"""
data = pd.read_html(html)[0]

print(data)
```

提取多个表格

```
import pandas as pd

html = """
<table border="1">
  <tr><th>月份</th><th>节约</th></tr>
  <tr><td>一月</td><td>100</td></tr>
</table>
```

```
<table border="1">
  <tr><th>月份</th><th>节约</th></tr>
  <tr><td>二月</td><td>200</td></tr>
</table>
"""
dfs = pd.read_html(html)
print("发现HTML表格数量: %s" % len(dfs))
for data in dfs:
    print(data)
```

从文件获取表格数据

准备 table.html 文件

```
<table border="1">
    <thead>
        <tr>
            <th>Month</th>
            <th>Savings</th>
        </tr>
    </thead>

    <tfoot>
        <tr>
            <td>Sum</td>
            <td>$180</td>
        </tr>
    </tfoot>

    <tbody>
        <tr>
            <td>January</td>
            <td>$100</td>
        </tr>
        <tr>
            <td>February</td>
            <td>$80</td>
```

```
        </tr>
    </tbody>
</table>
```

```
import pandas as pd

# 错误方法
html = open('table.html', 'r').read()
data = pd.read_html(html)[0]
print(data)

# 正确方法
data = pd.read_html('table.html')[0]
print(data)
```

从网址获取表格数据

获取页面中所有table的数据

```
import pandas as pd

url =
"http://www.stats.gov.cn/tjsj/zxfb/202103/t20210330_1815829.h
tml"
data = pd.read_html(url)

print(data)
```

处理多个表格

多个 table 会返回一个数组，通过数组下标可以读取指定表格。

```
import pandas as pd

url =
"http://www.stats.gov.cn/tjsj/zxfb/202103/t20210330_1815829.h
tml"
data = pd.read_html(url)[1]

print(data)
```

获取指定属性的表格

通过 id 属性，精确提取指定表格中的数据。

```
import pandas as pd

html = """
<table id="first">
  <tr><th>姓名</th><th>性别</th></tr>
  <tr><td>张三</td><td>男</td></tr>
</table>
<table id="second">
  <tr><th>姓名</th><th>性别</th></tr>
  <tr><td>李四</td><td>男</td></tr>
</table>
"""
data = pd.read_html(html, attrs={'id': 'second'})
print(data[0])
```

获取 html table 标签 id 属性为 oTable 的表格数据

```
import pandas as pd

url = "http://fund.eastmoney.com/fund.html"
data = pd.read_html(url,attrs = {'id': 'oTable'})

print(data)
```

结合 Xpath 使用

HTML 属性，只有 id 是唯一的，其他属性都允许重复使用，例如 class，所以使用 class 会选中多张 HTML 表格。

```
data = pd.read_html(text, attrs={'class': 'netkiller'})
print(len(data))
```

我的做法是使用 xpath 精准匹配，为什么部用数组下标了，因为页面的变化很可能造成数字顺序错乱。

```
import pandas as pd
from lxml import etree

text = """
<table class="netkiller">
  <tr><th>姓名</th><th>性别</th></tr>
  <tr><td>张三</td><td>男</td></tr>
</table>
<table class="netkiller">
  <tr><th>姓名</th><th>性别</th></tr>
  <tr><td>李四</td><td>男</td></tr>
</table>
"""
```

```
html = etree.HTML(text)
result = html.xpath('//table[@class="netkiller"][last()]')
text = etree.tostring(result[0], encoding='utf-
8').decode('utf-8')

data = pd.read_html(text)
print(data[0])
```

指定编码

目前仍有部分中文网站使用 GB2312和GBK编码，遇到输出乱码的情况，使用encoding指定编码即可。

建议：中文建议使用 GB18030，GB18038 包含了中日韩三国字符集，也就是说GB18030 是涵盖 GB2312和GBK的。

```
import pandas as pd

url = "http://www.tianqihoubao.com/weather/top/shenzhen.html"
data = pd.read_html(url, encoding="GB18030")
print(data[0])
```

使用 Dominate 生成 HTML

```
import dominate
from dominate.tags import *
import time
import os
import pandas as pd
```

```
os.chdir(os.path.dirname(__file__))

df = pd.read_excel("2022年6月17 Excle.xlsx", '6月17',
usecols="A:R")

print("=" * 20, '打印表头列名', "=" * 20)
header = df.columns[4:].tolist()
print(header)

print("=" * 20, '价格', "=" * 20)
price = df.iloc[0][4:].fillna('').to_dict()
print(price)

print("行: {0}".format(df.index.size))
print("列: {0}".format(df.columns.size))

doc = dominate.document(title='Dominate your HTML')
with doc:
    with div(id='content'):
        for index, row in df.iterrows():
            # print(row.to_dict())
            t = row[1:].fillna(0).to_dict()
            if t['合计'] == 0:
                continue
            br()
            with table(id=index, border=1, cellspacing="0",
cellpadding="0", width='50%'):
                caption('亲! 本期团购清单如下: {0}'.format(
                    time.strftime('%Y-%m-%d',
time.localtime())))
                h = tr(align="center")
                b = tr(align="center")
                thprice = tr(align="center")
                for key, value in t.items():
                    # print(value)
                    # print(t)
                    # if not value.isnull():
                    if value != 0:
                        h.add(th(key))
                        b.add(td(value))
                        if key in price.keys():
                            thprice.add(td(price[key]))
                        elif key == '几期':
                            thprice.add(td('单价'))
                        else:
```

```
                                thprice.add(td(''))
                thead().add(h)
                thead().add(thprice)
                tbody().add(b)
                # tfoot().add(p)

with open('doc.html', 'w') as file:
    file.write(doc.render())
# print(doc)
```

2.2. Excel 处理

本节主要介绍和excel的交互。

安装依赖库

```
neo@MacBook-Pro-Neo ~ % pip install openpyxl
```

创建 Excel 文档

```
from pandas import DataFrame

data = {
    'name': ['Neo', 'Tom', 'Jerry'],
    'age': [11, 12, 13],
    'gender': ['Y', 'Y', 'N']
}
df = DataFrame(data)
df.to_excel('netkiller.xlsx')
```

写入指定工作表

```
from pandas import DataFrame

data = {
    '姓名': ['Neo', 'Tom', 'Jerry'],
    '年龄': [11, 12, 13],
    '性别': ['Y', 'Y', 'N']
}
df = DataFrame(data)
df.to_excel('neo.xlsx', sheet_name='我的工作表')
```

设置 sheet_name 参数将数据写入指定的工作表

startrow=2 从第几行开始写入, index=False 关闭序号列, header=False 不写入表头

```
from pandas import DataFrame

data = {
    '姓名': ['Neo', 'Tom', 'Jerry'],
    '年龄': [11, 12, 13],
    '性别': ['Y', 'Y', 'N']
}
df = DataFrame(data)
df.to_excel('neo.xlsx', sheet_name='我的工作表',
            startrow=2, index=False, header=False)
```

写入多个工作表

```
import pandas as pd

data = pd.DataFrame(
    {
        '姓名': ['Neo', 'Tom', 'Jerry'],
        '年龄': [11, 12, 13],
        '性别': ['Y', 'Y', 'N']
    }
)
excel = pd.ExcelWriter("sheet.xlsx")
data.to_excel(excel, sheet_name="Sheet1", index=False)
data.to_excel(excel, sheet_name="Sheet2", index=False)
data.to_excel(excel, sheet_name="Sheet3", index=False)
excel.save()
excel.close()
```

向Excel中追加工作表

```
# 追加工作表，首先创建 Excel 文件
with pd.ExcelWriter("sheet1.xlsx") as writer:
    data.to_excel(writer, sheet_name="Sheet1", index=False)
    data.to_excel(writer, sheet_name="Sheet2", index=False)
    data.to_excel(writer, sheet_name="Sheet3", index=False)


append = pd.DataFrame(
    {
        '姓名': ['Neo', 'Tom', 'Jerry'],
        '年龄': [11, 12, 13],
        '性别': ['Y', 'Y', 'N']
    }
)
# 默认ExcelWriter是覆盖模式，需要使用 mode='a' 参数，才能Excel文件中增加工作表
with pd.ExcelWriter("sheet1.xlsx", mode='a',
engine='openpyxl') as writer:
    append.to_excel(writer, sheet_name="Sheet4", index=False)
    append.to_excel(writer, sheet_name="Sheet5", index=False)
```

读取 Excel 文件

read_excel() 参数详解：

```
加载函数为read_excel(),其具体参数如下。

read_excel(io, sheetname=0, header=0, skiprows=None,
skip_footer=0, index_col=None,names=None, parse_cols=None,
parse_dates=False,date_parser=None,na_values=None,thousands=Non
e, convert_float=True, has_index_names=None,
converters=None,dtype=None, true_values=None,
false_values=None, engine=None, squeeze=False, **kwds)

常用参数解析：
io : string, path object ; excel 路径。
sheetname : string, int, mixed list of strings/ints, or None,
default 0 返回多表使用sheetname=[0,1],若sheetname=None是返回全表
header : int, list of ints, default 0 指定列表头，默认0，即取第一
行，数据没有表头设定
skiprows : list-like,Rows to skip at the beginning，跳过指定行数的
数据
skip_footer : int,default 0, 省略从尾部数的行数据，默认是 0
index_col : int, list of ints, default None 指定列为索引列
names : array-like, default None, 指定列的名字。
```

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import pandas as pd
# 默认读取第一个工作表
df = pd.read_excel("团购2021.xlsx")
data = df.head()  # 默认读取前5行的数据
```

```
print("当前默认工作表：\n{0}".format(data))  # 格式化输出
```

读取指定列

```
import pandas as pd

df = pd.read_excel("../netkiller.xlsx")
print(df.head())
print("=" * 50)
# 读取B～E列的数据
df = pd.read_excel("../netkiller.xlsx", index_col=False,
usecols="B:E")
print(df)
```

工作表

显示所有工作表

获取Excel文件中的工作表

```
import pandas as pd
xls = pd.ExcelFile("团购2021.xlsx")
sheet_names = xls.sheet_names
print(sheet_names)
```

打开默认工作表

```
```

```
import pandas as pd
df = pd.read_excel("团购2021.xlsx", None)
print(df.keys())
for k,v in df.items():
    print(k)
```

打开工作表

打开指定工作表

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import pandas as pd

file_path = r'团购.xlsx'
df = pd.read_excel(file_path, sheet_name="3月2日", header=1)
```

打开 Excel 并返回指定列数据

```
import pandas as pd
sheet = pd.read_excel(io="new.xlsx", usecols=['name'])
print(sheet)
```

打开多个工作表

```
import pandas as pd
sheet = pd.read_excel(io="测试数据.xlsx", sheet_name=[0, 1])
print(sheet[0])
```

例 12.1. Pandas 打开工作表的四种方法

```
import pandas as pd

# 打开一张工作表
df = pd.read_excel("sheet1.xlsx", sheet_name="Sheet1")
print(df)

# 指定并打开多张工作表
df = pd.read_excel("sheet1.xlsx", sheet_name=["Sheet1",
"Sheet2", "Sheet3"])
for sheet, data in df.items():
    print("=" * 20, sheet, "=" * 20)
    print(data)

# 使用数字索引打开多张工作表
df = pd.read_excel("sheet1.xlsx", sheet_name=[0, 1, 2])
for sheet, data in df.items():
    print("=" * 20, sheet, "=" * 20)
    print(data)

# 使用 ExcelFile 对象打开多个工作表
sheet = {}
with pd.ExcelFile("sheet1.xlsx") as xls:
    sheet["Sheet1"] = pd.read_excel(
        xls, "Sheet1", index_col=None, na_values=["NA"])
    sheet["Sheet2"] = pd.read_excel(xls, "Sheet2",
index_col=1)

for sheet, data in sheet.items():
    print("=" * 20, sheet, "=" * 20)
    print(data)
```

合并工作表

```
import pandas as pd
sheet = pd.read_excel("测试.xlsx", sheet_name=[1, 0])
st = pd.concat(sheet,ignore_index = True)
```

```
import pandas as pd
sheet = pd.read_excel("sheet1.xlsx", sheet_name=[1, 0])
df = pd.concat(sheet, ignore_index=True)
print(df)
df.to_excel('merge.xlsx', index=False)
```

打开工作表，指定返回列数据

```
import pandas as pd

df = pd.read_excel("../netkiller.xlsx", index_col=False,
usecols="B,C:E")
print("=" * 20, "读取B,C:E列的数据", "=" * 20)
print(df.head())

# 读取B～E列的数据
df = pd.read_excel("../netkiller.xlsx", index_col=False,
usecols="B:E")
print("=" * 20, "读取B～E列的数据", "=" * 20)
print(df)

df = pd.read_excel("netkiller.xlsx", index_col=False,
usecols=[1, 2, 3])
print("=" * 20, "读取[1, 2, 3]列的数据", "=" * 20)
print(df)
```

跳过不需要的数据

```
import pandas as pd
from pandas import DataFrame
xlsx = 'skip.xlsx'
data = {
    '姓名': ['张三', '李四', '王五', '牛七', '马八', '赵九'],
    '年龄': [11, 12, 13, 14, 15, 17],
    '性别': ['Y', 'Y', 'N', 'Y', 'N', 'Y']
}
df = DataFrame(data)
df.to_excel(xlsx, index=True, index_label='序号')

df = pd.read_excel(xlsx, skiprows=[1, 3, 4])
print("跳过数据 [1, 3, 4]: \n{0}".format(df))

df = pd.read_excel(xlsx, skiprows=3)
print("跳过前3条数据: \n{0}".format(df))

df = pd.read_excel(xlsx, skipfooter=2)
print("从尾部剪掉1条数据: \n{0}".format(df))
```

```
跳过数据 [1, 3, 4]:
   序号  姓名  年龄 性别
0   1  李四  12  Y
1   4  马八  15  N
2   5  赵九  17  Y
跳过 top10 数据:
   2  王五  13  N
0  3  牛七  14  Y
1  4  马八  15  N
2  5  赵九  17  Y
从尾部剪掉1条数据:
   序号  姓名  年龄 性别
0   0  张三  11  Y
1   1  李四  12  Y
```

```
2    2   王五   13   N
3    3   牛七   14   Y
```

数据操作

打印头部/尾部数据

仅查看数据示例时常用

```
print(df.head())
print(df.tail())
```

打印列标题

```
print(sheet.columns)
```

打印行

```
print(sheet.index)
```

描述数据

```
print(sheet.describe())
```

修改 Excel 数据

```
import pandas as pd
from pandas import DataFrame

file_path = r'new.xlsx'
df = pd.read_excel(file_path)

df['gender'][df['gender'] == 'N'] = 'Female'
df['gender'][df['gender'] == 'Y'] = 'Male'

print(df)

DataFrame(df).to_excel(
    file_path, sheet_name='Sheet1', index=False, header=True)
```

新增行/列

```
import pandas as pd
from pandas import DataFrame

file_path = r'new.xlsx'
df = pd.read_excel(file_path)

# 新增一列
df['ctime'] = None

# 新增一行
df.loc[4] = [3, 'Alice', 20, 'Female',  '2021-5-11']

print(df)
```

```
DataFrame(df).to_excel(
    file_path, sheet_name='Sheet2', index=False, header=True)
```

数据筛选

```
import pandas as pd
sheet = pd.read_excel("工资表.xlsx", usecols=['工资'])
high_salary = sheet[sheet['工资'] > 10000]
middle_salary = sheet[(sheet['工资'] >= 8000) & (sheet['工资']
<=10000)]
low_salary = sheet[sheet['工资'] < 8000]
```

```
high_salary = sheet[(sheet['工资'] >= 8000) & (sheet['工资']
<=10000)][['姓名','工资']]
```

数据排序

Excel 设置项

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', 10)
pd.set_option('max_rows', 20)
# 禁用科学计数法
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

大数据写入Excel问题

df.to_excel 导出大数据会出现 Excle 表格损坏或没有数据的情况

```
df = pd.read_sql_query(text(sqlOrder), engine.connect())
df.to_excel("{yesterday}.xlsx".format(yesterday=yesterday),
sheet_name='Sheet1', index=False)
```

解决方案

```
        df = pd.read_sql_query(text(query), connect)
    with pd.ExcelWriter(xlsxfile) as writer:
        df.to_excel(writer,
sheet_name=platform_name.replace(':',''), index=False)
```

2.3. Pandas 读写 CSV 文件

将数据保存到CSV文件

```
import pandas as pd

months=
[202001,202002,202003,202004,202005,202006,202007,202008,2020
09,202010,202011,202012]

for month in months:
    print(month)
```

```
    weather =
pd.read_html(f'http://www.tianqihoubao.com/lishi/wanzhou/mont
h/{month}.html', encoding='gb18030', header=0)[0]
    print(weather)
    weather.to_csv(f'/tmp/{month}天气预报数据.csv', mode='a+',
index=False, header=False)
```

写入表头列名

header=True

```
df.to_csv(f'/tmp/{month}天气预报数据.csv', mode='a+',
index=False, header=True)
```

分隔符

```
dt.to_csv('C:/Users/think/Desktop/Result.csv',sep='?')
dt.to_csv('C:/Users/think/Desktop/Result.csv',sep=':')
```

格式化

```
# 格式 float_format: Format string for floating point numbers
# 保留两位小数
dt.to_csv('/tmp/neo.csv',float_format='%.2f')
```

指定列输出

```
# cols: Columns to write (default None)
dt.to_csv('C:/Users/think/Desktop/Result.csv',columns=
['name'])
```

留行索引

```
# index:  whether to write row (index) names (default True)
dt.to_csv('/tmp/neo.csv',index=0)
```

替换空值

```
# na_rep: A string representation of a missing value (default
‘’)
dt.to_csv('/tmp/neo.csv',na_rep='NA')
```

替换NaN(dropna,fillna,isnull)

```
import pandas as pd
import numpy as np

a = np.arange(25, dtype=float).reshape((5, 5))
# print(len(a))
```

```
for i in range(len(a)):
    a[i, :i] = np.nan
a[3, 0] = 25.0
df = pd.DataFrame(data=a, columns=list('ABCDE'))
print('-'*20, '原始数据', '-'*20)
print(df)

print('-'*20, '填充0', '-'*20)
print(df.fillna(value=0))

print('-'*20, '向后填充', '-'*20)
print(df.fillna(method='pad'))

print('-'*20, '向前填充', '-'*20)
print(df.fillna(method='backfill'))

print('-'*20, '用字典填充', '-'*20)
values = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4}
print(df.fillna(value=values))

print('-'*20, '只替换第1个NaN值', '-'*20)
print(df.fillna(method='pad', limit=1))
```

输出结果

```
-------------------- 原始数据 --------------------
      A    B     C     D     E
0   0.0  1.0   2.0   3.0   4.0
1   NaN  6.0   7.0   8.0   9.0
2   NaN  NaN  12.0  13.0  14.0
3  25.0  NaN   NaN  18.0  19.0
4   NaN  NaN   NaN   NaN  24.0
-------------------- 填充0 --------------------
      A    B     C     D     E
0   0.0  1.0   2.0   3.0   4.0
1   0.0  6.0   7.0   8.0   9.0
2   0.0  0.0  12.0  13.0  14.0
3  25.0  0.0   0.0  18.0  19.0
4   0.0  0.0   0.0   0.0  24.0
```

```
-------------------- 向后填充 --------------------
      A    B     C     D     E
0   0.0  1.0   2.0   3.0   4.0
1   0.0  6.0   7.0   8.0   9.0
2   0.0  6.0  12.0  13.0  14.0
3  25.0  6.0  12.0  18.0  19.0
4  25.0  6.0  12.0  18.0  24.0
-------------------- 向前填充 --------------------
      A    B     C     D     E
0   0.0  1.0   2.0   3.0   4.0
1  25.0  6.0   7.0   8.0   9.0
2  25.0  NaN  12.0  13.0  14.0
3  25.0  NaN   NaN  18.0  19.0
4   NaN  NaN   NaN   NaN  24.0
-------------------- 用字典填充 --------------------
      A    B     C     D     E
0   0.0  1.0   2.0   3.0   4.0
1   0.0  6.0   7.0   8.0   9.0
2   0.0  1.0  12.0  13.0  14.0
3  25.0  1.0   2.0  18.0  19.0
4   0.0  1.0   2.0   3.0  24.0
-------------------- 只替换第1个NaN值 --------------------
      A    B     C     D     E
0   0.0  1.0   2.0   3.0   4.0
1   0.0  6.0   7.0   8.0   9.0
2   NaN  6.0  12.0  13.0  14.0
3  25.0  NaN  12.0  18.0  19.0
4  25.0  NaN   NaN  18.0  24.0
```

2.4. Pandas SQL

建立数据库链接

sqlite3

```
import pandas as pd
from pandas import DataFrame
import sqlite3
```

```
con = sqlite3.connect(":memory:")

data = DataFrame({
    '姓名': ['张三', '李四', '王五'],
    '年龄': [11, 12, 13],
    '性别': ['Y', 'Y', 'N']
})

data.to_sql("data", con)
table = pd.read_sql_query("SELECT * FROM data", con)
print(table)
```

SQLAlchemy

安装依赖库

```
neo@MacBook-Pro-Neo ~ % pip install sqlalchemy
```

创建链接引擎参考实例

```
from sqlalchemy import create_engine

engine =
create_engine("postgresql://scott:tiger@localhost:5432/mydata
base")

engine =
create_engine("mysql+mysqldb://scott:tiger@localhost/foo")

engine =
create_engine("oracle://scott:tiger@127.0.0.1:1521/sidname")

engine = create_engine("mssql+pyodbc://mydsn")
```

```
# sqlite://<nohostname>/<path>
# where <path> is relative:
engine = create_engine("sqlite:///foo.db")

# or absolute, starting with a slash:
engine = create_engine("sqlite:////absolute/path/to/foo.db")
```

DataFrame数据写入到数据库

分批写入数据

DataFrame 结果集很大，写入时间过程很长，容易产生错误，这时可以使用 chunksize 切割数据，分批写入。

```
from sqlalchemy import create_engine
import pandas as pd
from pandas import DataFrame

engine = create_engine("sqlite:///:memory:")

data = DataFrame({
    '姓名': ['张三', '李四', '王五'],
    '年龄': [11, 12, 13],
    '性别': ['Y', 'Y', 'N']
})

data.to_sql("data", engine, chunksize=1000)

table = pd.read_sql_query("SELECT * FROM data", engine)
print(table)
```

3. 数据帧(DataFrame)

3.1. 什么是 DataFrame

通俗的讲数据帧(DataFrame)是二维数据矩阵，即数据由行和列组成的表格，类似数据库中的表和 Excel 电子表格。

Pandas 数据分析过程，首先从各种媒体中加载数据，然后将数据放入 DataFrame 处理，最后输出，输出包括输出到各种媒体和可视化图表。

DataFrame 都能做哪些数据处理呢?

矩阵运算，排序，筛选，过滤，分组，以及各种函数（例如求和）等等，这些操作很类似 Excel 对表格的操作。

DataFrame 是 Pandas 中最重要的对象，把它搞定，也就是熟练掌握 Pandas 数据分析了。

3.2. 如何创建 DataFrame

pandas.DataFrame 参数

```
pandas.DataFrame( data, index, columns, dtype, copy)
参数如下:

参数              描述
------------------------
data     数据采取各种形式，如:ndarray，series，map，lists，dict，
constant和另一个DataFrame。
index    对于行标签，要用于结果帧的索引是可选缺省值np.arrange(n)，如果
没有传递索引值。
columns 对于列标签，可选的默认语法是 - np.arange(n)。 这只有在没有索
引传递的情况下才是这样。
dtype    每列的数据类型。
copy     用于复制数据，默认值为False，
```

创建 DataFrame

创建 DataFrame 有以下几种方式:

- 列表
- 字典
- 序列
- Numpy
- ndarrays
- 复制另一个数据帧(DataFrame)

创建一个空的基本数据帧

```
import pandas as pd
df = pd.DataFrame()
print(df)
```

下面集中演示

```
import pandas as pd


def line(msg):
    print('-' * 20, msg, '-'*20)


df = pd.DataFrame()
line('空数据')
print(df)

data = [1, 2, 3, 4, 5]
```

```
df = pd.DataFrame(data)
line('从列表创建数据帧')
print(df)

data = [['张三', 26], ['李四', 25], ['王五', 23]]
df = pd.DataFrame(data, columns=['Name', 'Age'])
print(df)

data = [['苹果', 26], ['鸭梨', 25], ['橘子', 23.45]]
df = pd.DataFrame(data, columns=['Name', 'Age'], dtype=float)
line('指定数据类型为浮点')
print(df)

data = {'Name': ['张三', '李四', '王五', '牛七', '马八'],
        'Age': [23, 24, 21, 25, 27]}
line('从字典创建数据帧')
df = pd.DataFrame(data)
print(df)


data = [{'苹果': 1, '鸭梨': 2}, {'苹果': 5, '鸭梨': 10, '葡萄':
20}, {'橘子': 30}]
df = pd.DataFrame(data)
line('列表+字典的数据结构')
print(df)
```

输出结果

```
--------------------- 空数据 ---------------------
Empty DataFrame
Columns: []
Index: []
--------------------- 从列表创建数据帧 ---------------------
   0
0  1
1  2
2  3
3  4
4  5
  Name  Age
```

```
0   张三   26
1   李四   25
2   王五   23
-------------------- 指定数据类型为浮点 --------------------
  Name    Age
0   苹果  26.00
1   鸭梨  25.00
2   橘子  23.45
-------------------- 从字典创建数据帧 --------------------
  Name  Age
0   张三   23
1   李四   24
2   王五   21
3   牛七   25
4   马八   27
-------------------- 列表+字典的数据结构 --------------------
    苹果    鸭梨    葡萄    橘子
0  1.0   2.0   NaN   NaN
1  5.0  10.0  20.0   NaN
2  NaN   NaN   NaN  30.0
```

使用 Series 创建 DataFrame

```
import pandas as pd

data = pd.Series([1, 2, 3])
df = pd.DataFrame(data)
print(df)

data = {'序号': pd.Series([1, 2, 3]),
        '姓名': ['张三', '李四', '王五']}

df = pd.DataFrame(data)
print(df)
```

3.3. 行与列操作 index/columns

可以把 DataFrame 理解为一个二维矩阵（多数情况下以二维存在），左侧行名保存在index, 头部列名保存在columns。

下面举例，我们构建一个 3×3 的DataFrame矩阵，程序如下：

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(9).reshape(3, 3))
print(df)
```

输出结果

```
   0  1  2
0  0  1  2
1  3  4  5
2  6  7  8
```

第一行 0 1 2 是 columes

第一列 0 1 2 是 index

这里的 index 和 columes 是自动产生的。

方法一，指定 index / columes 名称

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(9).reshape(3, 3), index=[
```

```
                    'row1', 'row2', 'row3'], columns=['col1',
'col2', 'col3'])
print(df)
```

输出结果

```
      col1  col2  col3
row1     0     1     2
row2     3     4     5
row3     6     7     8
```

row1 ~ row3 是 index 用于访问行，col1 ~ col3 是 columns 用于访问列

方法二，指定 index / columes 名称

```
from pandas import DataFrame, Index
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(9).reshape(3, 3))
# print(df)

df.index = Index(['第一行', '第二行', '第三行'], dtype='object')
df.columns = Index(['第一列', '第二列', '第三列'],
dtype='object')

print(df)
```

运行结果

```
     第一列   第二列   第三列
第一行     0     1     2
第二行     3     4     5
第三行     6     7     8
```

获取 columns 名称

```
import numpy as np
import pandas as pd

df = pd.DataFrame([['Snow', 'M', 22], ['Tyrion', 'M', 32],
['Sansa', 'F', 18], [
                   'Arya', 'F', 14]], columns=['Name',
'Gender', 'Age'])

print(df)

col_name = df.columns.tolist()
print(col_name)
```

插入列

```
import numpy as np
import pandas as pd

df = pd.DataFrame([['Snow', 'M', 22], ['Tyrion', 'M', 32],
['Sansa', 'F', 18], [
                   'Arya', 'F', 14]], columns=['Name',
'Gender', 'Age'])

print(df)
```

```
col_name = df.columns.tolist()
print(col_name)
col_name.insert(3, 'City')
# 对原行/列索引重新构建索引值
df = df.reindex(columns=col_name)
# 给City列赋值
df['City'] = ['China', 'Japan', 'Koren', 'Hongkong']
print(df)
```

迭代行

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'c1': [10, 11, 12], 'c2': [100, 110,
120]})

for index, row in df.iterrows():
    print(row['c1'], row['c2'])
```

3.4. 数据访问

```
from pandas import DataFrame, Index
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(9).reshape(3, 3))
# print(df)

df.index = Index(['第一行', '第二行', '第三行'], dtype='object')
df.columns = Index(['第一列', '第二列', '第三列'],
dtype='object')
```

```
print("="*20, "原始数据", "=" * 20)
print(df)
print("="*20, "访问行", "=" * 20)
print(df[0:2])
print("="*20, "访问列", "=" * 20)
print(df.第一列)
print(df['第二列'])
```

输出结果

```
==================== 原始数据 ====================
     第一列  第二列  第三列
第一行    0    1    2
第二行    3    4    5
第三行    6    7    8
==================== 访问行 ====================
     第一列  第二列  第三列
第一行    0    1    2
第二行    3    4    5
==================== 访问列 ====================
第一行    0
第二行    3
第三行    6
Name: 第一列, dtype: int64
第一行    1
第二行    4
第三行    7
Name: 第二列, dtype: int64
```

head() 与 tail()

head() 从头部取数据

tail() 从尾部取数据

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(30).reshape(10, 3), index=list(
    range(10)), columns=['col1', 'col2', 'col3'])
print('-' * 20, "原始数据", '-' * 20)
print(df)
print('-' * 20, "head() 取出前5条数据", '-' * 20)
print(df.head())
print('-' * 20, "head() 取出前3条数据", '-' * 20)
print(df.head(3))
print('-' * 20, "tail() 取出尾部5条数据", '-' * 20)
print(df.tail())
print('-' * 20, "tail() 取出尾部3条数据", '-' * 20)
print(df.tail(3))
```

输出结果

```
-------------------- 原始数据 --------------------
   col1  col2  col3
0     0     1     2
1     3     4     5
2     6     7     8
3     9    10    11
4    12    13    14
5    15    16    17
6    18    19    20
7    21    22    23
8    24    25    26
9    27    28    29
-------------------- head() 取出前5条数据 --------------------
   col1  col2  col3
0     0     1     2
1     3     4     5
2     6     7     8
3     9    10    11
4    12    13    14
```

```
-------------------- head() 取出前3条数据 --------------------
   col1  col2  col3
0     0     1     2
1     3     4     5
2     6     7     8
-------------------- tail() 取出尾部5条数据 --------------------
   col1  col2  col3
5    15    16    17
6    18    19    20
7    21    22    23
8    24    25    26
9    27    28    29
-------------------- tail() 取出尾部3条数据 --------------------
   col1  col2  col3
7    21    22    23
8    24    25    26
9    27    28    29
```

iloc 访问数据

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(9).reshape(3, 3), index=[
                   'row1', 'row2', 'row3'], columns=['col1',
'col2', 'col3'])

print("="*20, "原始数据", "=" * 20)
print(df)
print("="*20, "访问一行", "=" * 20)
print(df.iloc[0])
print(df.iloc[0, :])
print("="*20, "访问指定行", "=" * 20)
print(df.iloc[[0, 2]])
print(df.iloc[[0, 2], :])
print("="*20, "访问连续多行", "=" * 20)
print(df.iloc[0:3])
print(df.iloc[0:3, :])
print("="*20, "访问一列", "=" * 20)
```

```
print(df.iloc[:, 0])
print("="*20, "访问指定列", "=" * 20)
print(df.iloc[:, [0, 2]])
print("="*20, "访问连续列", "=" * 20)
print(df.iloc[:, 0:3])

df = pd.DataFrame(np.arange(25).reshape(5, 5), index=[
                   'row1', 'row2', 'row3', 'row4', 'row5'],
columns=['col1', 'col2', 'col3', 'col4', 'col5'])
print("="*20, "访问中间区域", "=" * 20)
print(df.iloc[1:4:, 1:4])
```

运行结果

```
===================== 原始数据 ====================
      col1  col2  col3
row1     0     1     2
row2     3     4     5
row3     6     7     8
==================== 访问一行 ====================
col1    0
col2    1
col3    2
Name: row1, dtype: int64
col1    0
col2    1
col3    2
Name: row1, dtype: int64
==================== 访问指定行 ====================
      col1  col2  col3
row1     0     1     2
row3     6     7     8
      col1  col2  col3
row1     0     1     2
row3     6     7     8
==================== 访问连续多行 ====================
      col1  col2  col3
row1     0     1     2
row2     3     4     5
```

```
row3     6     7     8
      col1  col2  col3
row1     0     1     2
row2     3     4     5
row3     6     7     8
==================== 访问一列 ====================
row1    0
row2    3
row3    6
Name: col1, dtype: int64
==================== 访问指定列 ====================
      col1  col3
row1     0     2
row2     3     5
row3     6     8
==================== 访问连续列 ====================
      col1  col2  col3
row1     0     1     2
row2     3     4     5
row3     6     7     8
==================== 访问中间区域 ====================
      col2  col3  col4
row2     6     7     8
row3    11    12    13
row4    16    17    18
```

loc 访问数据

```
from pandas import DataFrame, Index
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(25).reshape(5, 5), index=[
                  'row1', 'row2', 'row3', 'row4', 'row5'],
columns=['col1', 'col2', 'col3', 'col4', 'col5'])

print("="*20, "原始数据", "=" * 20)
print(df)
print("="*20, "访问一行", "=" * 20)
```

```
print(df.loc['row1'])
print(df.loc['row1', :])
print("="*20, "访问指定行", "=" * 20)
print(df.loc[['row1', 'row2']])
print(df.loc[['row1', 'row2'], :])
print("="*20, "访问连续多行数据", "=" * 20)
print(df.loc['row1':'row3'])
print(df.loc['row1':'row3', :])

print("="*20, "访问一列", "=" * 20)
print(df.loc[:, 'col1'])
print("="*20, "访问指定列", "=" * 20)
print(df.loc[:, ['col1', 'col2']])
print("="*20, "访问连续多列数据", "=" * 20)
print(df.loc[:, 'col1':'col3'])
print("="*20, "访问数据区域", "=" * 20)
print(df.loc['row2':'row4':, 'col2':'col4'])
```

运行结果

```
==================== 原始数据 ====================
      col1  col2  col3  col4  col5
row1     0     1     2     3     4
row2     5     6     7     8     9
row3    10    11    12    13    14
row4    15    16    17    18    19
row5    20    21    22    23    24
==================== 访问一行 ====================
col1    0
col2    1
col3    2
col4    3
col5    4
Name: row1, dtype: int64
col1    0
col2    1
col3    2
col4    3
col5    4
```

```
Name: row1, dtype: int64
==================== 访问指定行 ====================
      col1  col2  col3  col4  col5
row1     0     1     2     3     4
row2     5     6     7     8     9
      col1  col2  col3  col4  col5
row1     0     1     2     3     4
row2     5     6     7     8     9
==================== 访问连续多行数据 ====================
      col1  col2  col3  col4  col5
row1     0     1     2     3     4
row2     5     6     7     8     9
row3    10    11    12    13    14
      col1  col2  col3  col4  col5
row1     0     1     2     3     4
row2     5     6     7     8     9
row3    10    11    12    13    14
==================== 访问一列 ====================
row1     0
row2     5
row3    10
row4    15
row5    20
Name: col1, dtype: int64
==================== 访问指定列 ====================
      col1  col2
row1     0     1
row2     5     6
row3    10    11
row4    15    16
row5    20    21
==================== 访问连续多列数据 ====================
      col1  col2  col3
row1     0     1     2
row2     5     6     7
row3    10    11    12
row4    15    16    17
row5    20    21    22
==================== 访问数据区域 ====================
      col2  col3  col4
row2     6     7     8
row3    11    12    13
row4    16    17    18
```

Axis（轴）

二维数据拥有两个轴：

axis = 0：沿着行的方向垂直往下。

axis = 1：沿着列的方向水平延伸。

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(9).reshape(3, 3), index=[
                  'row1', 'row2', 'row3'], columns=['col1',
'col2', 'col3'])

print("=" * 20, "原始数据", "=" * 20)
print(df)
print("=" * 20, "axis=0", "=" * 20)
print(df.mean(axis=0))
print("=" * 20, "axis=1", "=" * 20)
print(df.mean(axis=1))
print("=" * 20, "删除行 row1, axis=0", "=" * 20)
print(df.drop('row1', axis=0))
print("=" * 20, "删除列 col1, axis=1", "=" * 20)
print(df.drop(['col1'], axis=1))
```

```
==================== 原始数据 ====================
      col1  col2  col3
row1     0     1     2
row2     3     4     5
row3     6     7     8
==================== axis=0 ====================
```

```
col1    3.0
col2    4.0
col3    5.0
dtype: float64
==================== axis=1 ====================
row1    1.0
row2    4.0
row3    7.0
dtype: float64
==================== 删除行 row1, axis=0 ====================
      col1  col2  col3
row2     3     4     5
row3     6     7     8
==================== 删除列 col1, axis=1 ====================
      col2  col3
row1     1     2
row2     4     5
row3     7     8
```

axis=0 按列计算的均值

axis=1 按行计算的均值

3.5. 添加操作

添加列

```
import pandas as pd

data = {'序号': pd.Series([1, 2, 3]),
        '姓名': ['张三', '李四', '王五']}

df = pd.DataFrame(data)
df['年龄'] = [23, 25, 26]
print(df)
```

输出结果

```
   序号   姓名   年龄
0   1   张三   23
1   2   李四   25
2   3   王五   26
```

追加数据

向 DataFrame 尾部追加数据

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns=['col1', 'col2'])
df1 = pd.DataFrame([[5, 6], [7, 8]], columns=['col1',
'col2'])
df2 = pd.DataFrame([[9, 10], [11, 12]], columns=['col1',
'col2'])

df = df.append(df1)
df = df.append(df2)
print(df)
```

输出结果

```
   col1  col2
0     1     2
1     3     4
0     5     6
1     7     8
0     9    10
```

```
1    11    12
```

3.6. 删除操作

删除列数据

```
import pandas as pd

data = {'序号': pd.Series([1, 2, 3]),
        '姓名': ['张三', '李四', '王五'],
        '年龄': [23, 25, 26]
        }

df = pd.DataFrame(data)
print('-' * 20, "原始数据", '-' * 20)
print(df)
del df['年龄']
print('-' * 20, "del 删除数据", '-' * 20)
print(df)
print('-' * 20, "pop 删除数据", '-' * 20)
df.pop('序号')
print(df)
```

输出结果

```
-------------------- 原始数据 --------------------
   序号  姓名  年龄
0   1  张三  23
1   2  李四  25
2   3  王五  26
-------------------- del 删除数据 --------------------
   序号  姓名
0   1  张三
1   2  李四
```

```
2   3  王五
-------------------- pop 删除数据 --------------------
   姓名
0  张三
1  李四
2  王五
```

使用 pandas.drop() 方法删除行与列

```
import pandas as pd

data = {'序号': pd.Series([1, 2, 3]),
        '姓名': ['张三', '李四', '王五'],
        '年龄': [23, 25, 26]
        }

df = pd.DataFrame(data)
print('-' * 20, "原始数据", '-' * 20)
print(df)
print('-' * 20, "drop(0) 删除第一行", '-' * 20)
print(df.drop(0))
print('-' * 20, "df.drop(index=1, axis=0) 删除一行", '-' * 20)
print(df.drop(index=1, axis=0))
print('-' * 20, "df.drop(index=[1, 2]) 删除行", '-' * 20)
print(df.drop(index=[1, 2]))

print('-' * 20, "df.drop('序号', axis=1) 删除序号列", '-' * 20)
print(df.drop('序号', axis=1))
```

多行与多列的删除方法

print(df.drop(index=['Bob', 'Dave', 'Frank'], columns=['Name', 'Age']))
print(df.drop(index=df.index[[1, 3, 5]], columns=df.columns[[1, 2]]))

df_org = df.copy()
df_org.drop(index=['Bob', 'Dave', 'Frank'], columns=
['Name', 'Age'], inplace=True)

print(df.drop(columns='Name'))
print(df.drop(columns=['Name', 'Age']))
print(df.drop(['Name', 'Age'], axis=1))

print(df.columns[[1, 2]])
print(df.drop(df.columns[[1, 2]], axis=1))
print(df.drop(columns=df.columns[[1, 2]]))

3.7. 插入数据

插入一列数据

在 Dataframe 的指定列后面插入数据

Dataframe.insert(loc, column, value, allow_duplicates=False):

参数介绍:
loc: int型，表示第几列；若在第一列插入数据，则 loc=0
column: 给插入的列取名，如 column='新的一列'
value：数据，array，series
allow_duplicates: 是否允许列名重复，选择Ture表示允许新的列名与已存在的列名重复。

```
import numpy as np
import pandas as pd
```

```
data = pd.DataFrame(np.arange(16).reshape(4, 4),
columns=list('ABCD'))
print(data)
print('-' * 20, '在最左侧插入一列', '-' * 20)
data.insert(loc=0, column='新增一列', value='1')
print(data)
print('-' * 20, '在第六列后面增加一列', '-' * 20)
data.insert(loc=2, column='新增列', value=2)
print(data)
print('-' * 20, '插入重复列名称', '-' * 20)
data.insert(loc=6, column='D', value=3,
allow_duplicates=True)
print(data)
```

```
import numpy as np
import pandas as pd

data = pd.DataFrame(np.arange(16).reshape(4, 4),
columns=list('ABCD'))
print(data)
print('-' * 20, '在最左侧插入一列', '-' * 20)
data.insert(loc=0, column='新增一列', value='1')
print(data)
print('-' * 20, '在第六列后面增加一列', '-' * 20)
data.insert(loc=2, column='新增列', value=2)
print(data)
print('-' * 20, '插入重复列名称', '-' * 20)
data.insert(loc=6, column='D', value=3, allow_duplicates=True)
print(data)
```

3.8. 替换操作

修改/替换 index 和 columns 标签名

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(9).reshape(3, 3), index=[
                   'r1', 'r2', 'r3'], columns=['c1', 'c2',
'c3'])
print("=" * 20, "原始数据", "=" * 20)
print(df)

print('-' * 20, '重新定义行标签', '-' * 20)
df.index = ['a', 'b', 'c']
print(df)

print('-' * 20, '修改行标签', '-' * 20)
df.rename(index={'a': '第一行', 'b': '第二行', 'c': '第三行'},
inplace=True)
print(df)

print('-' * 20, '重新定义列标签', '-' * 20)
df.columns = ['A', 'B', 'C']
print(df)

print('-' * 20, '修改列标签', '-' * 20)
df.rename(columns={'A': '第一列', 'B': '第二列', 'C': '第三列'},
inplace=True)
print(df)
```

输出结果

```
==================== 原始数据 ====================
    c1  c2  c3
r1   0   1   2
r2   3   4   5
r3   6   7   8
-------------------- 重新定义行标签 --------------------
   c1  c2  c3
a   0   1   2
```

```
b   3   4   5
c   6   7   8
-------------------- 修改行标签 --------------------
     c1  c2  c3
第一行   0   1   2
第二行   3   4   5
第三行   6   7   8
-------------------- 重新定义列标签 --------------------
     A  B  C
第一行  0  1  2
第二行  3  4  5
第三行  6  7  8
-------------------- 修改列标签 --------------------
     第一列  第二列  第三列
第一行    0    1    2
第二行    3    4    5
第三行    6    7    8
```

```
import numpy as np
import pandas as pd

data = {
    '姓名': ['张三', '李四', '王五'],
    '年龄': [11, 12, 13],
    '性别': ['Y', 'Y', 'N']
}
df = pd.DataFrame(data)

print("=" * 20, "原始数据", "=" * 20)
print(df)

print("=" * 20, "列替换", "=" * 20)
df.loc[df['性别'] == 'Y', '性别'] = '男'
df.loc[df['性别'] == 'N', '性别'] = '女'
print(df)
print("=" * 20, "行替换", "=" * 20)
df.loc[0, df.loc[0, :] == 11] = 20
print(df)
```

```
==================== 原始数据 ====================
   姓名  年龄 性别
0  张三  11  Y
1  李四  12  Y
2  王五  13  N
==================== 列替换 ====================
   姓名  年龄 性别
0  张三  11  男
1  李四  12  男
2  王五  13  女
==================== 行替换 ====================
   姓名  年龄 性别
0  张三  20  男
1  李四  12  男
2  王五  13  女
```

3.9. 筛选

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import pandas as pd

file_path = r'团购.xlsx'
df = pd.read_excel(file_path, sheet_name="3月2日",
                   header=1, usecols=['房号', '客户名', '合计'])
data = df[2:-3]
print(data)
print(data[data['合计'] > 0])
```

```
       房号  客户名    合计
```

```
2     7-6A  李松   25
3     7-8A  王雪   68
4     7-9A  梁梅  100
5    7-10A  宋森   21
6    9-27B  蓝天   20
7    9-28B  赵岚    0
8    9-30B  皮思   50
9    1-20B  龚风  256
10   1-24A  阮纵    0
```

筛选后

```
      房号 客户名   合计
2    7-6A  李松   25
3    7-8A  王雪   68
4    7-9A  梁梅  100
5   7-10A  宋森   21
6   9-27B  蓝天   20
8   9-30B  皮思   50
9   1-20B  龚风  256
```

3.10. 排序

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import numpy as np
import pandas as pd

df = pd.DataFrame({'数值': np.random.randint(1, 10, size=8)})
print(("-" * 10) + "随机数" + ("-" * 10))
print(df)
print(("-" * 10) + "排序后" + ("-" * 10))
print(df.sort_values('数值'))
print(("-" * 10) + "降序排序" + ("-" * 10))
print(df.sort_values('数值', ascending=False))
```

运行结果

```
----------随机数----------
    数值
0     5
1     9
2     1
3     6
4     2
5     5
6     3
7     4
----------排序后----------
    数值
2     1
4     2
6     3
7     4
0     5
5     5
3     6
1     9
----------降序排序----------
    数值
1     9
3     6
0     5
5     5
7     4
6     3
4     2
2     1
```

多列排序

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import numpy as np
import pandas as pd

df = pd.DataFrame({'A': np.random.randint(1, 10, size=8),
'B': np.random.randint(
    1, 10, size=8), 'C': np.random.randint(1, 10, size=8)})
print(("-" * 10) + "随机数" + ("-" * 10))
print(df)
print(("-" * 10) + "B列排序" + ("-" * 10))
print(df.sort_values('B'))
print(("-" * 10) + "A降序,B升序" + ("-" * 10))
print(df.sort_values(['A', 'B'], ascending=[False, True]))
print(("-" * 10) + "横向排序(行排序)" + ("-" * 10))
print(df.sort_values(by=0, axis=1, ascending=False))
```

运行结果

```
----------随机数----------
   A  B  C
0  3  2  7
1  4  8  1
2  3  8  6
3  2  4  2
4  7  9  2
5  7  9  4
6  4  1  8
7  9  4  4
----------B列排序----------
   A  B  C
6  4  1  8
0  3  2  7
3  2  4  2
7  9  4  4
1  4  8  1
```

```
2   3   8   6
4   7   9   2
5   7   9   4
----------A降序,B升序----------
   A   B   C
7   9   4   4
4   7   9   2
5   7   9   4
6   4   1   8
1   4   8   1
0   3   2   7
2   3   8   6
3   2   4   2
----------横向排序(行排序)----------
   C   A   B
0   7   3   2
1   1   4   8
2   6   3   8
3   2   2   4
4   2   7   9
5   4   7   9
6   8   4   1
7   4   9   4
```

标签排序

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import numpy as np
import pandas as pd

df = pd.DataFrame({'B': np.random.randint(1, 10, size=5),
'A': np.random.randint(
    1, 10, size=5), 'C': np.random.randint(1, 10, size=5)},
index=np.random.randint(1, 5, size=5))
print(("-" * 10) + "随机数" + ("-" * 10))
print(df)
print(("-" * 10) + "行标签排序后" + ("-" * 10))
print(df.sort_index())
print(("-" * 10) + "列标签排序后" + ("-" * 10))
```

```
print(df.sort_index(axis=1))
```

运行结果

```
neo@MacBook-Pro-Neo ~/workspace/python/pandas % python3.9
/Users/neo/workspace/python/pandas/dataframe/index.py
----------随机数----------
   B  A  C
4  9  2  1
4  1  1  5
4  1  4  5
3  8  4  7
2  3  1  6
----------行标签排序后----------
   B  A  C
2  3  1  6
3  8  4  7
4  9  2  1
4  1  1  5
4  1  4  5
----------列标签排序后----------
   A  B  C
4  2  9  1
4  1  1  5
4  4  1  5
3  4  8  7
2  1  3  6
```

3.11. 分类聚合

```
import pandas as pd

data = {'序号': list(range(6)),
        '姓名': ['张三', '李四', '王五', '赵六', '牛七', '马八'],
```

```
        '年龄': [23, 25, 26, 25, 25, 27],
        '生日': ['2001-12-01', '2001-12-05', '2001-10-01',
'2001-1-5', '2002-2-15', '2001-08-01']
        }

df = pd.DataFrame(data)
print('-' * 20, "原始数据", '-' * 20)
print(df)
print('-' * 20, "按年龄分组", '-' * 20)
ages = df.groupby(['年龄'])
for k, v in ages:
    print("{0} 岁年龄组: ".format(k))
    print(v)
    print()
print('-' * 20, "按年龄分组并计算数量", '-' * 20)
ages = df.groupby(['年龄'])['年龄'].count()
print(ages)
```

输出结果

```
-------------------- 原始数据 --------------------
   序号  姓名  年龄          生日
0   0  张三  23  2001-12-01
1   1  李四  25  2001-12-05
2   2  王五  26  2001-10-01
3   3  赵六  25    2001-1-5
4   4  牛七  25   2002-2-15
5   5  马八  27  2001-08-01
-------------------- 按年龄分组 --------------------
23 岁年龄组:
   序号  姓名  年龄          生日
0   0  张三  23  2001-12-01

25 岁年龄组:
   序号  姓名  年龄          生日
1   1  李四  25  2001-12-05
3   3  赵六  25    2001-1-5
4   4  牛七  25   2002-2-15

26 岁年龄组:
```

```
   序号  姓名  年龄          生日
2   2  王五  26  2001-10-01

27 岁年龄组:
   序号  姓名  年龄          生日
5   5  马八  27  2001-08-01

-------------------- 按年龄分组并计算数量 --------------------
年龄
23    1
25    3
26    1
27    1
Name: 年龄, dtype: int64
```

3.12. 数据去重

df.drop_duplicates(['姓名'],inplace=True)

```
import pandas as pd

data = {'序号': list(range(6)),
        '姓名': ['张三', '李四', '张三', '赵六', '张三', '马八'],
        '年龄': [23, 25, 26, 25, 25, 27],
        '生日': ['2001-12-01', '2001-12-05', '2001-10-01',
'2001-1-5', '2002-2-15', '2001-08-01']
        }

df = pd.DataFrame(data)
print('-' * 20, "原始数据", '-' * 20)
print(df)
print('-' * 20, "重复数据", '-' * 20)
print(df[df.姓名.duplicated()])
print('-' * 20, "取出重复的姓名", '-' * 20)
df.drop_duplicates(['姓名'], inplace=True)
print(df)
```

```
-------------------- 原始数据 --------------------
   序号  姓名  年龄          生日
0   0  张三  23  2001-12-01
1   1  李四  25  2001-12-05
2   2  张三  26  2001-10-01
3   3  赵六  25    2001-1-5
4   4  张三  25   2002-2-15
5   5  马八  27  2001-08-01
-------------------- 重复数据 --------------------
   序号  姓名  年龄          生日
2   2  张三  26  2001-10-01
4   4  张三  25   2002-2-15
-------------------- 取出重复的姓名 --------------------
   序号  姓名  年龄          生日
0   0  张三  23  2001-12-01
1   1  李四  25  2001-12-05
3   3  赵六  25    2001-1-5
5   5  马八  27  2001-08-01
```

3.13. 数据格式化

日期格式化

df['创建时间'] = df['创建时间'].dt.strftime('%Y-%m-%d')

```
import pandas as pd

data = {'序号': pd.Series([1, 2, 3]),
        '姓名': ['张三', '李四', '王五'],
        '年龄': [23, 25, 26],
        '生日': ['2001-12-01', '2001-12-05', '2001-10-01']
        }
```

```
df = pd.DataFrame(data)
df.生日 = pd.to_datetime(df.生日)
print(df)
df['生日'] = df['生日'].dt.strftime('%Y-%m-%d %H:%M:%S')
print(df)
```

浮点格式化

```
import pandas as pd

data = {'序号': pd.Series([1, 2, 3]),
        '姓名': ['张三', '李四', '王五'],
        '生活费': [123.23, 125.113, 226.999],
        '生日': ['2001/12/01', '2001-12-05', '2001-10-01']
        }

df = pd.DataFrame(data)

pd.options.display.float_format = '{:.2f}'.format
df.生日 = pd.to_datetime(df.生日)
print(df)
df['生日'] = df['生日'].dt.strftime('%Y-%m-%d %H:%M:%S')
print(df)
```

```
import pandas as pd

df = pd.DataFrame([['10.0', 6, 7, 8],
                   ['1.0', 9, 12, 14],
                   ['5.0', 8, 10, 6]], columns=['A', 'B',
'C', 'D'])

df['A'] = pd.to_numeric(df['A'], errors='coerce')
print(df)
```

```
print('-' * 40)
print(df.info())
```

小数位数

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame([
        (.1, .044),
        (.21, .32),
        (.01, .6),
        (.001, .0006),
        (.66, .03),
        (.21, .183)
        ],columns=['dogs', 'cats'])
>>> df
    dogs    cats
0  0.100  0.0440
1  0.210  0.3200
2  0.010  0.6000
3  0.001  0.0006
4  0.660  0.0300
5  0.210  0.1830

# 统一保持2位小数
>>> df.round(2)
   dogs  cats
0  0.10  0.04
1  0.21  0.32
2  0.01  0.60
3  0.00  0.00
4  0.66  0.03
5  0.21  0.18

# 统一保持一位小数
>>> df.round(1)
   dogs  cats
0   0.1   0.0
1   0.2   0.3
```

```
2   0.0   0.6
3   0.0   0.0
4   0.7   0.0
5   0.2   0.2

>>> df.round(0)
   dogs  cats
0   0.0   0.0
1   0.0   0.0
2   0.0   1.0
3   0.0   0.0
4   1.0   0.0
5   0.0   0.0
```

3.14. 迭代输出

```
for index, row in df.iterrows():
    print(index, row.to_dict())
```

4. 序列

Series：是一种类似于一维数组的对象，是由一组数据和一组与之相关的数据标签(即索引)组成。

Series 创建方式有两种:

- 通过以为数组方式创建
- 通过以为列表方式创建
- 通过字典的方式创建（此时key变成索引，value变成了值）

Series 参数:

- Series (Series)是能够保存任何类型的数据(整数，字符串，浮点数，Python对象等)的一维标记数组。轴标签统称为索引。
- data 数据
- index 索引 索引值必须是唯一的和散列的，与数据的长度相同。默认np.arange(n)如果没有索引被传递。
- dtype 输出的数据类型 如果没有，将推断数据类型
- copy 复制数据 默认为false

4.1. 创建 Series 对象

```
import pandas as pd

print('-' * 20, '通过数组创建 Series', '-' * 20)
ser = pd.Series(['a', 'b', 'c', 'd', 'e'])
print(ser)

print('-' * 20, '通过列表创建 Series', '-' * 20)
ser = pd.Series(list(range(5)), index=['a', 'b', 'c', 'd',
'e'])
print(ser)

print('-' * 20, '通过字典创建 Series', '-' * 20)
```

```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['a', 'b', 'c'])
print(ser)

print('-' * 20, '指定常量创建 Series', '-' * 20)
ser = pd.Series(5, index=[1, 2, 3, 4, 5])
print(ser)
```

4.2. Series 运算

```
import pandas as pd

s1 = pd.Series(data=[1, 2, 3, 4, 5], index=["a", "b", "c",
"d", "e"])
s2 = pd.Series(data=[1, 2, 3, 4, 5], index=["a", "b", "c",
"e", "f"])
ser = s1 + s2
print(ser)
```

```
a    2.0
b    4.0
c    6.0
d    NaN
e    9.0
f    NaN
dtype: float64
```

4.3. Series 常用方法

head() /tail()

```
import pandas as pd
import string
data = [chr(x) for x in range(ord('a'), ord('z') + 1)]
res = pd.Series(data, index=list(string.ascii_lowercase))
# 原始数据
print(res)
# 查看前三个
print(res.head(3))
# 查看后两个
print(res.tail(2))
```

isnull() / notnull()

```
print('-' * 20, '筛选出非空的正常数据', '-' * 20)
print(ser[ser.notnull()])
print('-' * 20, '筛选出空的数据', '-' * 20)
print(ser[ser.isnull()])
```

去重复数据

```
import pandas as pd
s = pd.Series(data=[1, 1, 3, 2, 3, 4, 5, 6, 5, 6, 7, 6, 9, 7,
8, 10])
print(s.unique())
```

输出结果

```
[ 1  3  2  4  5  6  7  9  8 10]
```

5. 数据可视化

数据可视化是借助于图形化手段，对数字罗列的数字进行分析，使分析结果可视化，清晰有效地展现数据背后意义、直观地传达出信息内容、从而实现视觉对话，这是表格或电子表格无法做到的。

常用的图表包括：线形图、柱状图、条形图、面积图、饼图、点图、仪表盘、走势图外，还有和弦图、圈饼图、雷达图、金字塔、漏斗图、K线图、关系图、网络图、玫瑰图、帕累托图、数学公式图、预测曲线图、正态分布图、迷你图、行政地图、GIS地图等各种展现形式。

数据可视化需要四个步骤：

1. 数据准备：从数据库,Excle,CSV文件，HTML表格等等
2. 数据加载：pd.read_sql/pd.read_csv/pd.read_excel 将数据加载到DataFrame中
3. 数据清洗：删除，排序，筛选，分组聚合，运算
4. 数据可视化：使用 matplotlib 生成图表

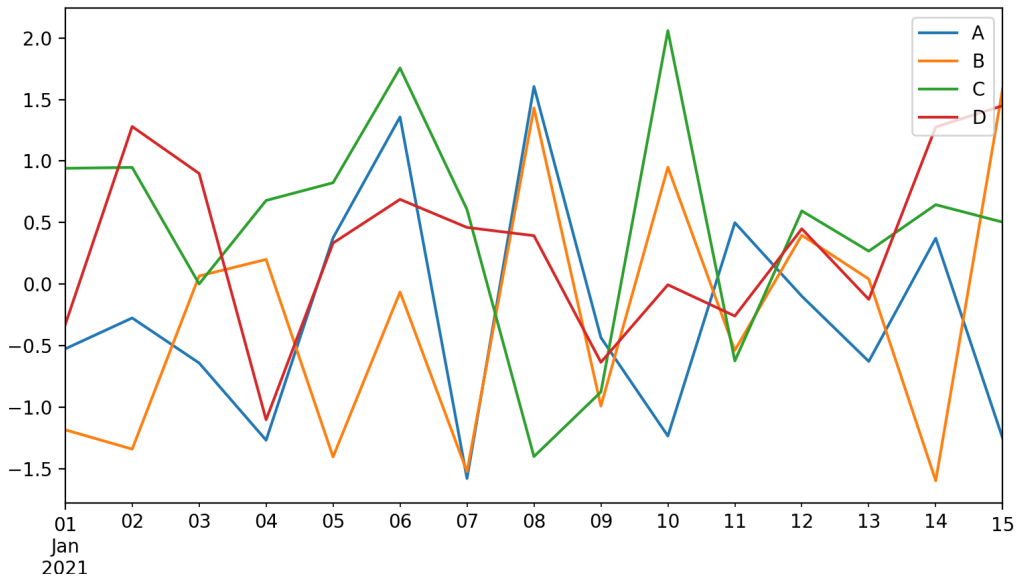
5.1. 演示代码

折线图

折线图的用途主要用于展示数据随着时间变化的趋势。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame(np.random.randn(15, 4),
index=pd.date_range(
    '2021/01/01', periods=15), columns=list('ABCD'))
```

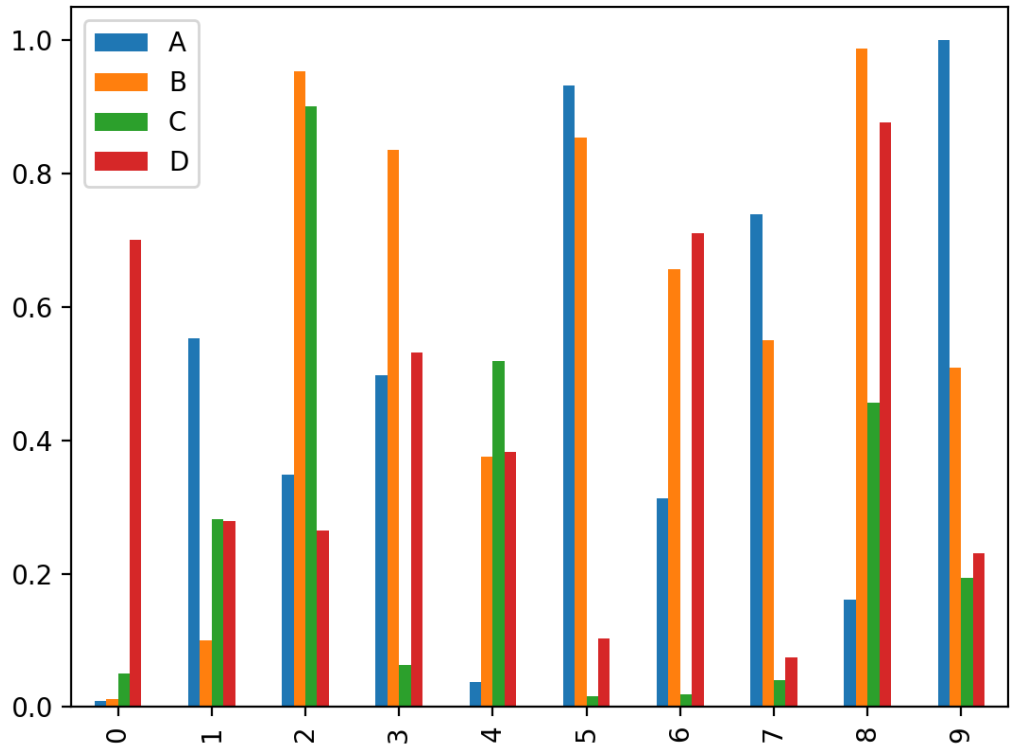
```
df.plot()
plt.show()
```



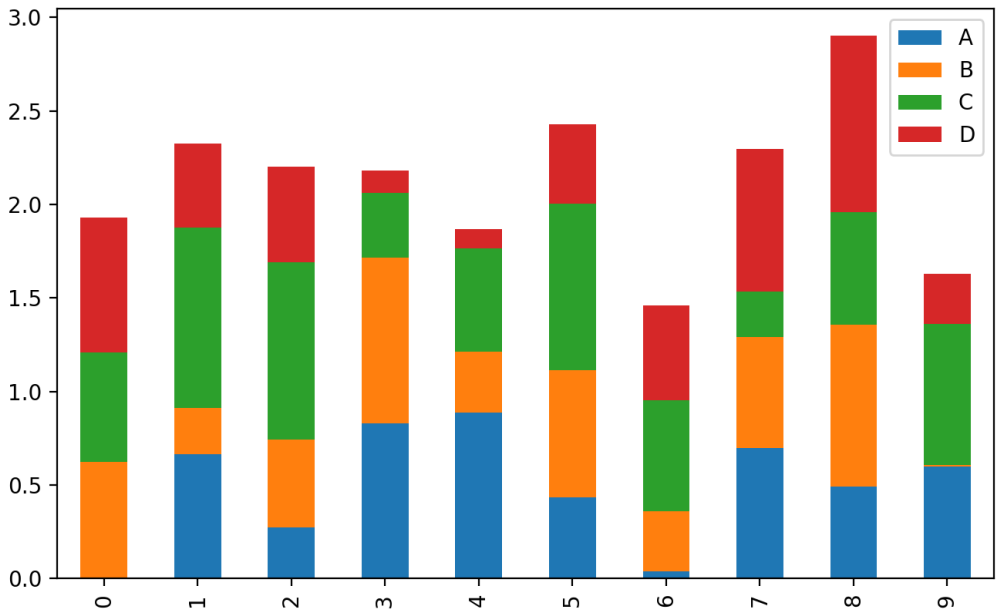
条形图

条形图主要用于表示离散型数据资料，即计数数据

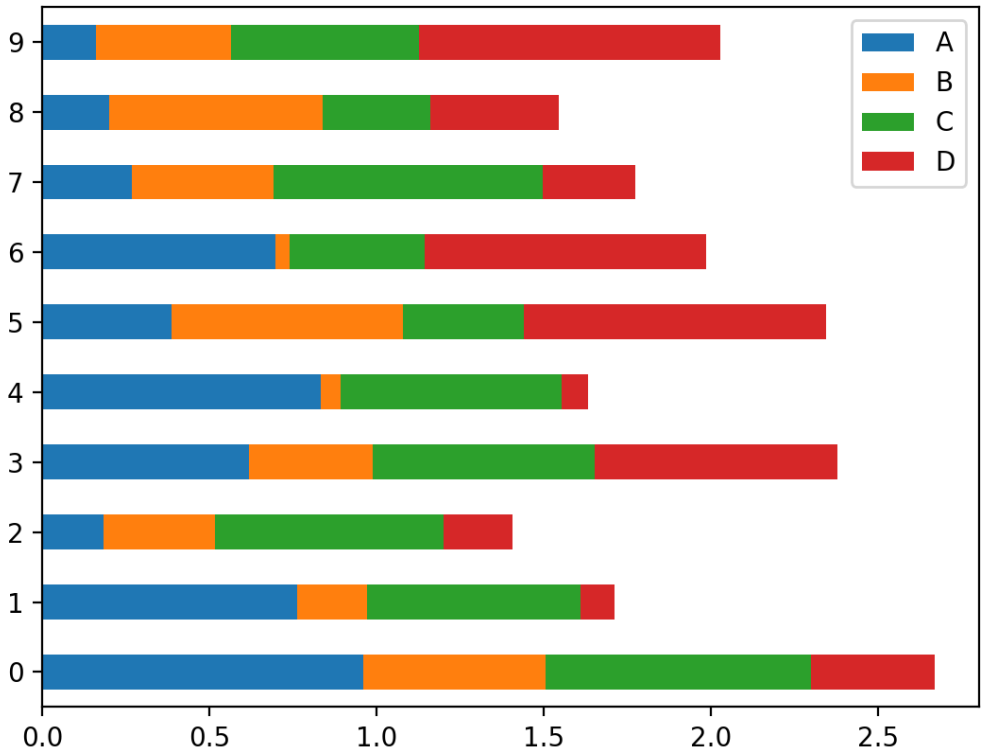
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame(np.random.rand(10, 4), columns=['A', 'B',
'C', 'D'])
df.plot.bar()
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.DataFrame(np.random.rand(10, 4), columns=['A', 'B',
'C', 'D'])
df.plot.bar(stacked=True)
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.DataFrame(np.random.rand(10, 4), columns=['A', 'B',
'C', 'D'])

df.plot.barh(stacked=True)
plt.show()
```

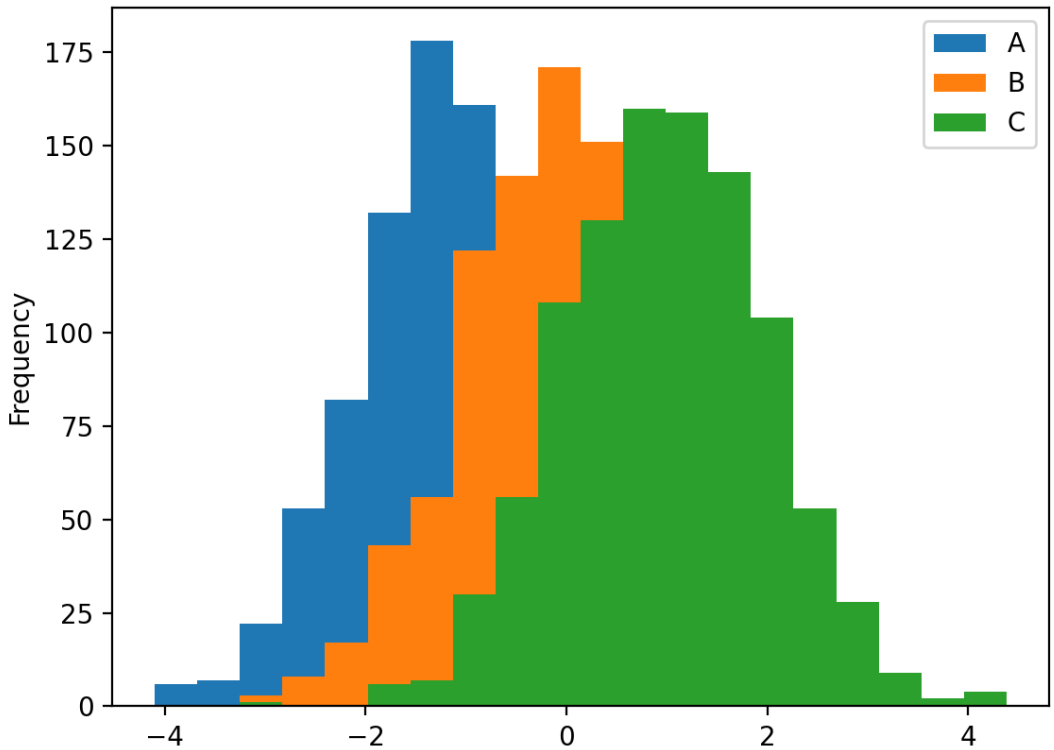


直方图

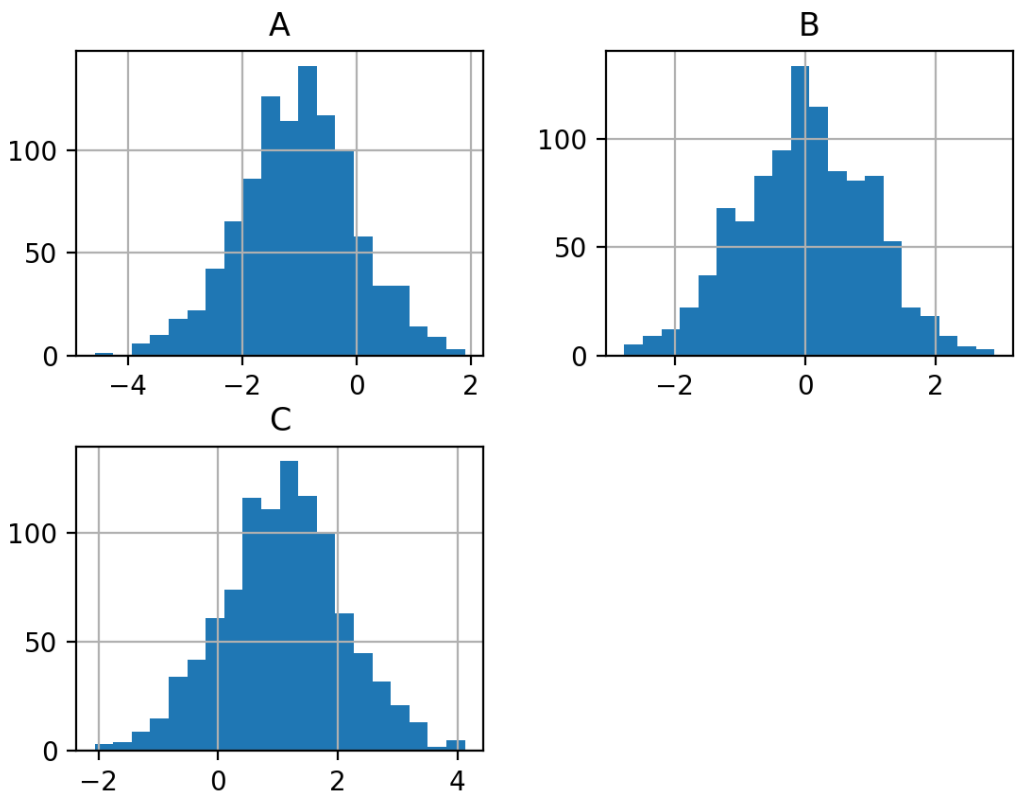
直方图(Histogram)，又称质量分布图，是一种统计报告图，由一系列高度不等的纵向条纹或线段表示数据分布的情况。 一般用横轴表示数据类型，纵轴表示分布情况。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame({'A': np.random.randn(1000)-1,
                   'B': np.random.randn(1000),
                   'C': np.random.randn(1000)+1},
                  columns=['A', 'B', 'C'])
```

```
df.plot.hist(bins=20)
plt.show()
```



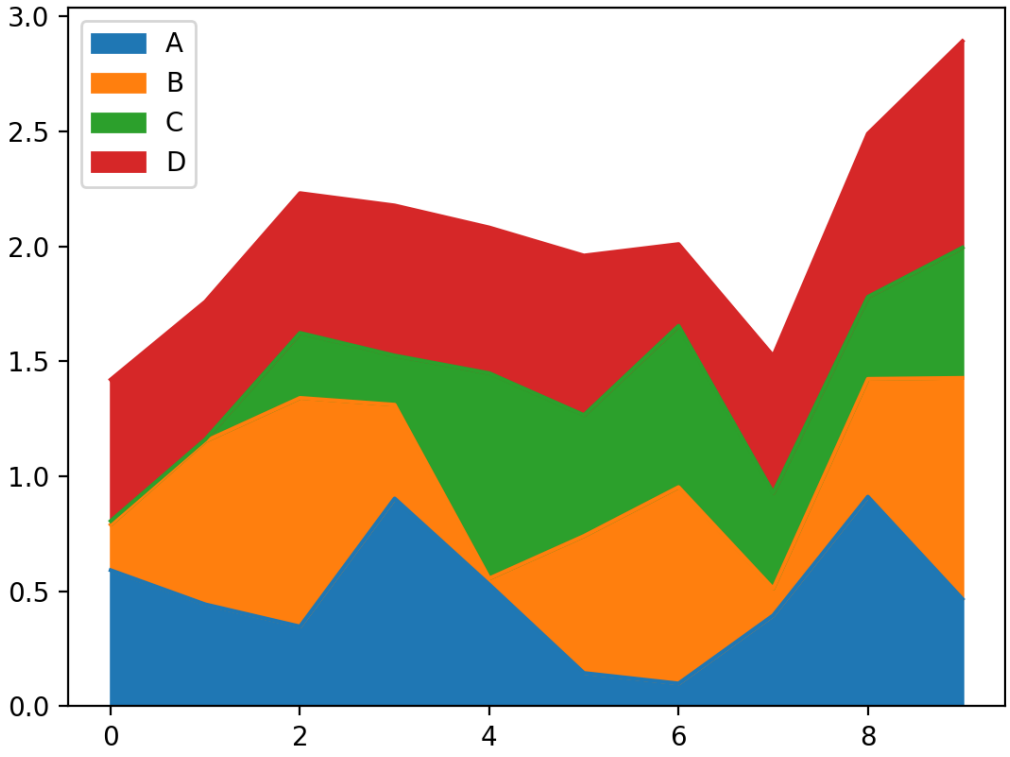

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = np.random.randn(1000)

df = pd.DataFrame({'A': data, 'B': data, 'C': data}, columns=
['A', 'B', 'C'])

df.hist(bins=20)
plt.show()
```



区域图

面积图又称区域图，面积图强调数量随时间而变化的程度，也可用于引起人们对总值趋势的注意

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.DataFrame(np.random.rand(10, 4), columns=['A', 'B',
'C', 'D'])
df.plot.area()
plt.show()
```

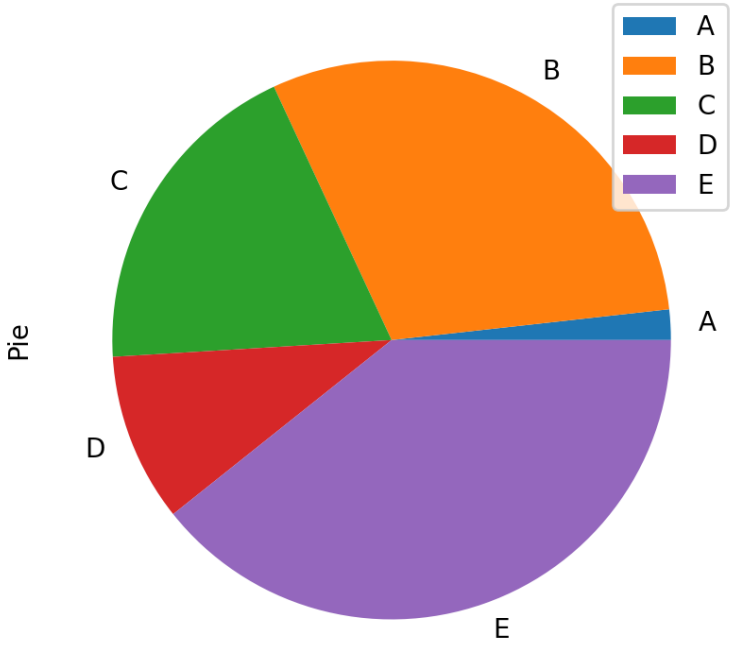


饼形图

饼形图主要用于展示数据总额的百分比。缺点是，当多个数据值都小于饼图的 5% 时，区分各个扇区将十分困难。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.DataFrame(np.random.rand(5),
                  index=['A', 'B', 'C', 'D', 'E'], columns=
['Pie'])
```

```
df.plot.pie(subplots=True)
plt.show()
```

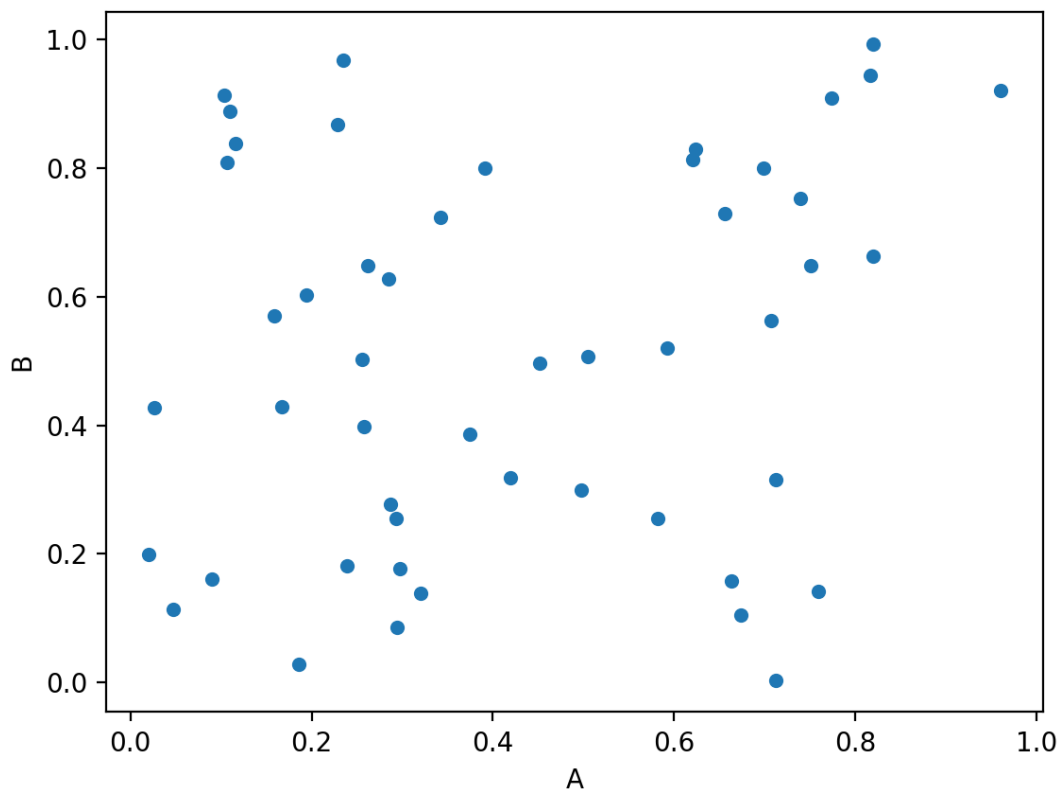


XY散点图

散点图是指在回归分析中，数据点在直角坐标系平面上的分布图，散点图表示因变量随自变量而变化的大致趋势，据此可以选择合适的函数对数据点进行拟合。

用两组数据构成多个坐标点，考察坐标点的分布，判断两变量之间是否存在某种关联或总结坐标点的分布模式。散点图将序列显示为一组点。值由点在图表中的位置表示。类别由图表中的不同标记表示。散点图通常用于比较跨类别的聚合数据。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.DataFrame(np.random.rand(50, 4), columns=['A', 'B',
'C', 'D'])
df.plot.scatter(x='A', y='B')
plt.show()
```



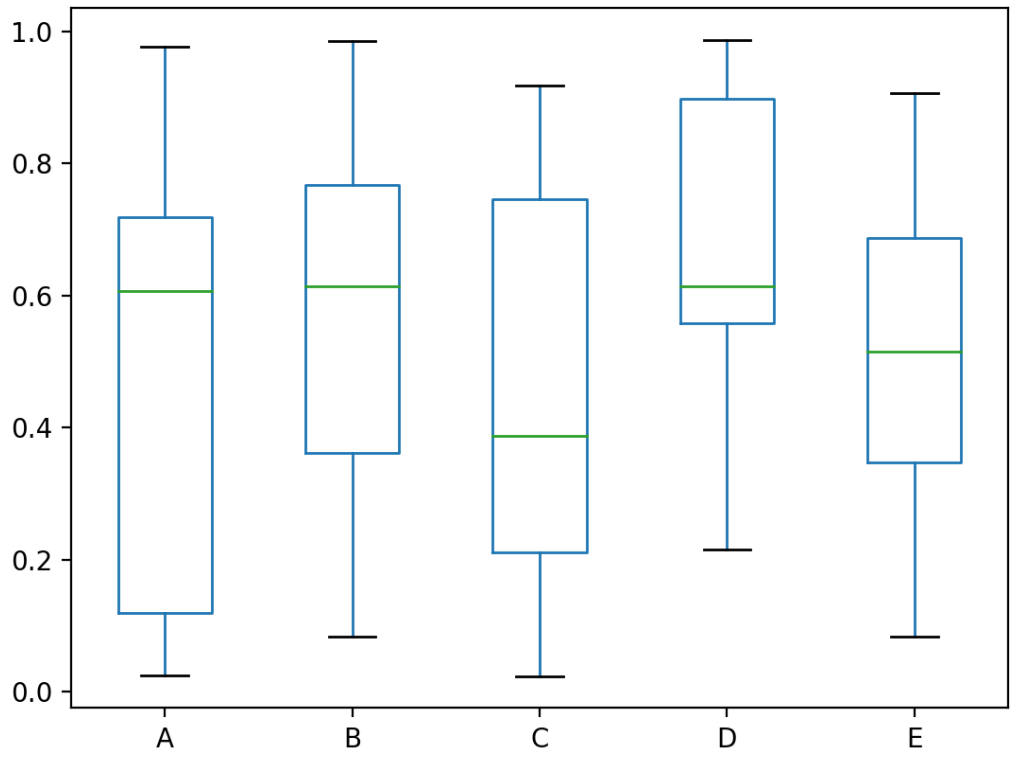
箱形图

箱形图（英文：Box plot），又称为盒须图、盒式图、盒状图或箱线图，是一种用作显示一组数据分散情况资料的统计图。因型状如箱

子而得名。在各种领域也经常被使用，常见于品质管理。不过作法相对较繁琐。

箱形图于1977年由美国著名统计学家约翰·图基（John Tukey）发明。它能显示出一组数据的最大值、最小值、中位数、及上下四分位数。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B',
'C', 'D', 'E'])
df.plot.box()
plt.show()
```



核密度估计图（Kernel Density Estimation, KDE）

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.Series(np.random.normal(loc=0, scale=5, size=10000))
df.plot(kind='kde')
df.hist(density=True)
plt.grid()
plt.show()
```

5.2. 中文支持

查看系统支持的字体

```
from matplotlib import font_manager
a = sorted([f.name for f in
font_manager.fontManager.ttflist])
for i in a:
    print(i)
```

```
.Aqua Kana
.Arabic UI Display
.Arabic UI Text
.Keyboard
.New York
.New York
.SF Compact
.SF Compact
.SF Compact Rounded
.SF NS Mono
.SF NS Mono
.SF NS Rounded
Academy Engraved LET
Adobe Arabic
Adobe Arabic
Adobe Arabic
Adobe Arabic
Adobe Caslon Pro
Adobe Caslon Pro
Adobe Caslon Pro
Adobe Caslon Pro
Adobe Caslon Pro
Adobe Caslon Pro
Adobe Devanagari
Adobe Devanagari
Adobe Devanagari
```

Adobe Devanagari
Adobe Fan Heiti Std
Adobe Fangsong Std
Adobe Garamond Pro
Adobe Garamond Pro
Adobe Garamond Pro
Adobe Garamond Pro
Adobe Gothic Std
Adobe Hebrew
Adobe Hebrew
Adobe Hebrew
Adobe Hebrew
Adobe Heiti Std
Adobe Kaiti Std
Adobe Ming Std
Adobe Myungjo Std
Adobe Naskh
Adobe Song Std
Al Bayan
Al Nile
Al Tarikh
American Typewriter
Andale Mono
Apple Braille
Apple Braille
Apple Braille
Apple Braille
Apple Braille
Apple Chancery
Apple SD Gothic Neo
Apple Symbols
AppleGothic
AppleMyungjo
Arial
Arial
Arial
Arial
Arial Black
Arial Hebrew
Arial Narrow
Arial Narrow
Arial Narrow
Arial Narrow
Arial Rounded MT Bold
Arial Unicode MS

Arial Unicode MS
Athelas
Avenir
Avenir Next
Avenir Next Condensed
Ayuthaya
Baghdad
Bangla MN
Bangla Sangam MN
Baskerville
Beirut
Big Caslon
Birch Std
Blackoak Std
Bodoni 72
Bodoni 72 Oldstyle
Bodoni 72 Smallcaps
Bodoni Ornaments
Bradley Hand
Brush Script MT
Brush Script Std
Chalkboard
Chalkboard SE
Chalkduster
Chaparral Pro
Chaparral Pro
Chaparral Pro
Chaparral Pro
Chaparral Pro
Charlemagne Std
Charter
Cochin
Comic Sans MS
Comic Sans MS
Cooper Std
Cooper Std
Copperplate
Corsiva Hebrew
Courier New
Courier New
Courier New
Courier New
DIN Alternate
DIN Condensed
Damascus

DecoType Naskh
DejaVu Sans
DejaVu Sans
DejaVu Sans
DejaVu Sans
DejaVu Sans Display
DejaVu Sans Mono
DejaVu Sans Mono
DejaVu Sans Mono
DejaVu Sans Mono
DejaVu Serif
DejaVu Serif
DejaVu Serif
DejaVu Serif
DejaVu Serif Display
Devanagari MT
Devanagari Sangam MN
Didot
Diwan Kufi
Diwan Thuluth
Euphemia UCAS
Farah
Farisi
Futura
Galvji
Geeza Pro
Georgia
Georgia
Georgia
Georgia
Giddyup Std
Gill Sans
Gujarati MT
Gujarati Sangam MN
Gurmukhi MN
Gurmukhi MT
Gurmukhi Sangam MN
Heiti TC
Heiti TC
Helvetica
Helvetica Neue
Herculanum
Hiragino Maru Gothic Pro
Hiragino Mincho ProN
Hiragino Sans

Hiragino Sans
Hiragino Sans
Hiragino Sans
Hiragino Sans
Hiragino Sans
Hiragino Sans
Hiragino Sans
Hiragino Sans
Hiragino Sans
Hiragino Sans GB
Hobo Std
Hoefler Text
Hoefler Text
ITF Devanagari
Impact
InaiMathi
Iowan Old Style
Kailasa
Kannada MN
Kannada Sangam MN
Kefa
Khmer MN
Khmer Sangam MN
Kohinoor Bangla
Kohinoor Devanagari
Kohinoor Gujarati
Kohinoor Telugu
Kokonor
Kozuka Gothic Pr6N
Kozuka Gothic Pr6N
Kozuka Gothic Pr6N
Kozuka Gothic Pr6N
Kozuka Gothic Pr6N
Kozuka Gothic Pr6N
Kozuka Gothic Pro
Kozuka Gothic Pro
Kozuka Gothic Pro
Kozuka Gothic Pro
Kozuka Gothic Pro
Kozuka Gothic Pro
Kozuka Mincho Pr6N
Kozuka Mincho Pr6N
Kozuka Mincho Pr6N
Kozuka Mincho Pr6N
Kozuka Mincho Pr6N

Kozuka Mincho Pr6N
Kozuka Mincho Pro
Kozuka Mincho Pro
Kozuka Mincho Pro
Kozuka Mincho Pro
Kozuka Mincho Pro
Kozuka Mincho Pro
Krungthep
KufiStandardGK
Lao MN
Lao Sangam MN
Letter Gothic Std
Letter Gothic Std
Letter Gothic Std
Letter Gothic Std
Lithos Pro
Lithos Pro
Lucida Grande
Luminari
Malayalam MN
Malayalam Sangam MN
Marion
Marker Felt
Menlo
Mesquite Std
Microsoft Sans Serif
Minion Pro
Minion Pro
Minion Pro
Minion Pro
Minion Pro
Minion Pro
Minion Pro
Minion Pro
Minion Pro
Minion Pro
Mishafi
Mishafi Gold
Mshtakan
Mukta Mahee
Muna
Myanmar MN
Myanmar Sangam MN
Myriad Arabic
Myriad Arabic

Myriad Arabic
Myriad Arabic
Myriad Hebrew
Myriad Hebrew
Myriad Hebrew
Myriad Hebrew
Myriad Pro
Myriad Pro
Myriad Pro
Myriad Pro
Myriad Pro
Myriad Pro
Myriad Pro
Myriad Pro
Myriad Pro
Myriad Pro
Nadeem
New Peninim MT
Noteworthy
Noto Nastaliq Urdu
Noto Sans Adlam
Noto Sans Armenian
Noto Sans Avestan
Noto Sans Bamum
Noto Sans Bassa Vah
Noto Sans Batak
Noto Sans Bhaiksuki
Noto Sans Brahmi
Noto Sans Buginese
Noto Sans Buhid
Noto Sans Carian
Noto Sans Caucasian Albanian
Noto Sans Chakma
Noto Sans Cham
Noto Sans Coptic
Noto Sans Cuneiform
Noto Sans Cypriot
Noto Sans Duployan
Noto Sans Egyptian Hieroglyphs
Noto Sans Elbasan
Noto Sans Glagolitic
Noto Sans Gothic
Noto Sans Hanifi Rohingya
Noto Sans Hanunoo
Noto Sans Hatran

Noto Sans Imperial Aramaic
Noto Sans Inscriptional Pahlavi
Noto Sans Inscriptional Parthian
Noto Sans Javanese
Noto Sans Kaithi
Noto Sans Kannada
Noto Sans Kayah Li
Noto Sans Kharoshthi
Noto Sans Khojki
Noto Sans Khudawadi
Noto Sans Lepcha
Noto Sans Limbu
Noto Sans Linear A
Noto Sans Linear B
Noto Sans Lisu
Noto Sans Lycian
Noto Sans Lydian
Noto Sans Mahajani
Noto Sans Mandaic
Noto Sans Manichaean
Noto Sans Marchen
Noto Sans Meetei Mayek
Noto Sans Mende Kikakui
Noto Sans Meroitic
Noto Sans Miao
Noto Sans Modi
Noto Sans Mongolian
Noto Sans Mro
Noto Sans Multani
Noto Sans Myanmar
Noto Sans NKo
Noto Sans Nabataean
Noto Sans New Tai Lue
Noto Sans Newa
Noto Sans Ogham
Noto Sans Ol Chiki
Noto Sans Old Hungarian
Noto Sans Old Italic
Noto Sans Old North Arabian
Noto Sans Old Permic
Noto Sans Old Persian
Noto Sans Old South Arabian
Noto Sans Old Turkic
Noto Sans Oriya
Noto Sans Osage

Noto Sans Osmanya
Noto Sans Pahawh Hmong
Noto Sans Palmyrene
Noto Sans Pau Cin Hau
Noto Sans PhagsPa
Noto Sans Phoenician
Noto Sans Psalter Pahlavi
Noto Sans Rejang
Noto Sans Runic
Noto Sans Samaritan
Noto Sans Saurashtra
Noto Sans Sharada
Noto Sans Shavian
Noto Sans Siddham
Noto Sans Sora Sompeng
Noto Sans Sundanese
Noto Sans Syloti Nagri
Noto Sans Syriac
Noto Sans Tagalog
Noto Sans Tagbanwa
Noto Sans Tai Le
Noto Sans Tai Tham
Noto Sans Tai Viet
Noto Sans Takri
Noto Sans Thaana
Noto Sans Tifinagh
Noto Sans Tirhuta
Noto Sans Ugaritic
Noto Sans Vai
Noto Sans Wancho
Noto Sans Warang Citi
Noto Sans Yi
Noto Serif Ahom
Noto Serif Balinese
Noto Serif Myanmar
Nueva Std
Nueva Std
Nueva Std
Nueva Std
Nueva Std
Nueva Std
OCR A Std
Optima
Orator Std
Orator Std

Oriya MN
Oriya Sangam MN
PT Mono
PT Sans
PT Serif
PT Serif Caption
Palatino
Papyrus
Party LET
Phosphate
PingFang HK
Plantagenet Cherokee
Poplar Std
Prestige Elite Std
Raanana
Rockwell
Rosewood Std
SF Compact
SF Compact
SF Compact Display
SF Compact Display
SF Compact Display
SF Compact Display
SF Compact Display
SF Compact Display
SF Compact Display
SF Compact Display
SF Compact Rounded
SF Compact Rounded
SF Compact Rounded
SF Compact Rounded
SF Compact Rounded
SF Compact Rounded
SF Compact Rounded
SF Compact Rounded
SF Compact Rounded
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text

```
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Compact Text
SF Pro
SF Pro
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Display
SF Pro Rounded
SF Pro Rounded
SF Pro Rounded
SF Pro Rounded
SF Pro Rounded
SF Pro Rounded
SF Pro Rounded
SF Pro Rounded
SF Pro Rounded
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
```

```
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
SF Pro Text
STHeiti
STIXGeneral
STIXGeneral
STIXGeneral
STIXGeneral
STIXGeneral
STIXGeneral
STIXGeneral
STIXIntegralsD
STIXIntegralsD
STIXIntegralsSm
STIXIntegralsSm
STIXIntegralsUp
STIXIntegralsUp
STIXIntegralsUpD
STIXIntegralsUpD
STIXIntegralsUpSm
STIXIntegralsUpSm
STIXNonUnicode
STIXNonUnicode
STIXNonUnicode
STIXNonUnicode
STIXNonUnicode
STIXNonUnicode
STIXNonUnicode
STIXNonUnicode
STIXSizeFiveSym
STIXSizeFiveSym
STIXSizeFourSym
STIXSizeFourSym
STIXSizeFourSym
STIXSizeFourSym
```

STIXSizeOneSym
STIXSizeOneSym
STIXSizeOneSym
STIXSizeOneSym
STIXSizeThreeSym
STIXSizeThreeSym
STIXSizeThreeSym
STIXSizeThreeSym
STIXSizeTwoSym
STIXSizeTwoSym
STIXSizeTwoSym
STIXSizeTwoSym
STIXVariants
STIXVariants
Sana
Sathu
Savoye LET
Seravek
Shree Devanagari 714
SignPainter
Silom
Sinhala MN
Sinhala Sangam MN
Skia
Snell Roundhand
Songti SC
Source Code Pro
Source Code Pro
Source Code Pro
Source Code Pro
Source Code Pro
Source Code Pro
Stencil Std
Sukhumvit Set
Superclarendon
Symbol
System Font
System Font
Tahoma
Tahoma
Tamil MN
Tamil Sangam MN
Tekton Pro
Tekton Pro
Tekton Pro

```
Tekton Pro
Telugu MN
Telugu Sangam MN
Thonburi
Times
Times New Roman
Times New Roman
Times New Roman
Times New Roman
Trajan Pro
Trajan Pro
Trattatello
Trebuchet MS
Trebuchet MS
Trebuchet MS
Trebuchet MS
Verdana
Verdana
Verdana
Verdana
Waseem
Webdings
Wingdings
Wingdings 2
Wingdings 3
Zapf Dingbats
Zapfino
cmb10
cmex10
cmmi10
cmr10
cmss10
cmsy10
cmtt10
```

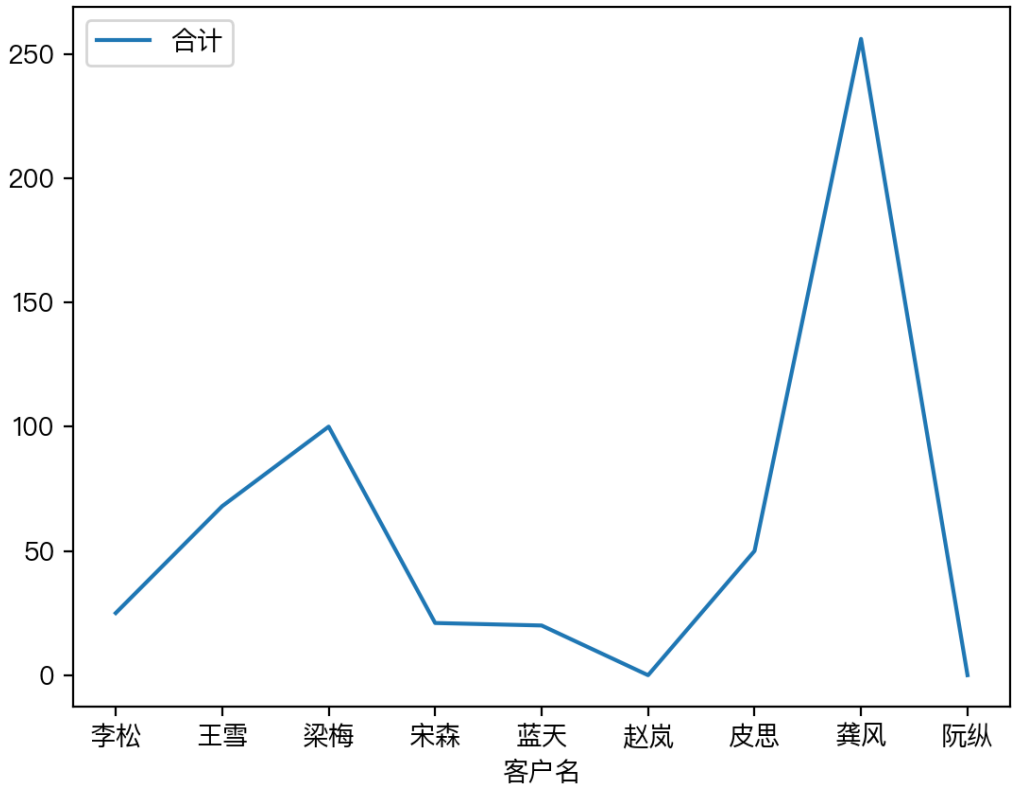
设置字体

从字体支持列表中选择你需要的字体

```
import matplotlib as mpl
mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False
```

中文演示代码

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False


sheet = pd.read_excel('团购.xlsx', sheet_name="3月2日",
                      header=1, usecols=['房号', '客户名', '合
计'])
df = sheet[2:-3]
print(df)
df.plot.line(x='客户名', y='合计')
plt.show()
```



5.3. 开启网格

plt.grid()

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False

sheet = pd.read_excel('团购.xlsx', sheet_name="3月2日",
                      header=1, usecols=['房号', '客户名', '合
计'])
df = sheet[2:-3]
```

```
# print(df)
df.plot.line(x='客户名', y='合计')
plt.title('销售业绩表')
plt.xlabel('客户')
plt.ylabel('消费金额')
plt.grid()
plt.show()
```

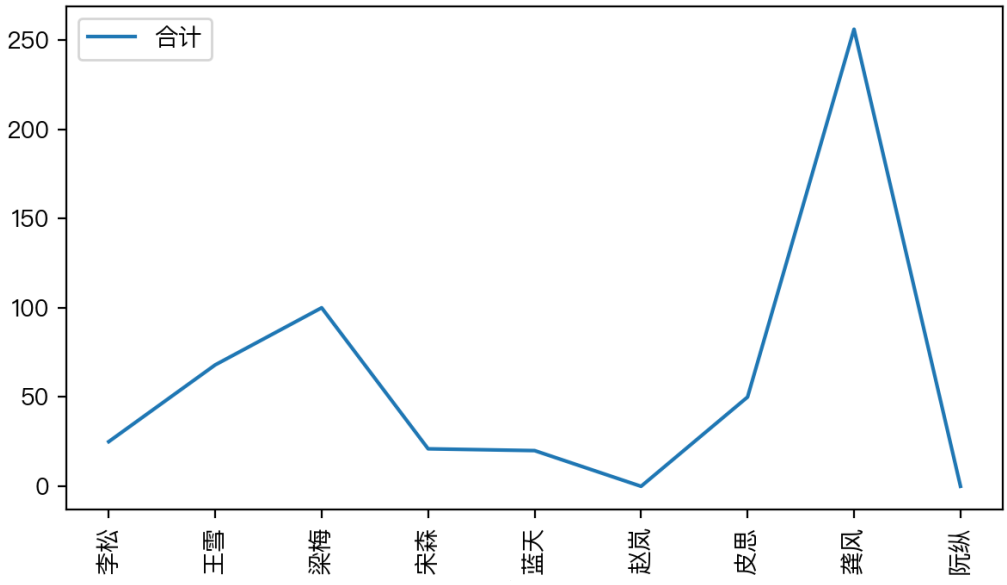
5.4. 坐标轴

xlabel():设置x坐标轴名称
ylabel():设置y坐标轴名称
xlim(): 设置x坐标轴范围
ylim(): 设置y坐标轴范围
xticks():设置x轴刻度
yticks():设置y轴刻度

轴标签旋转

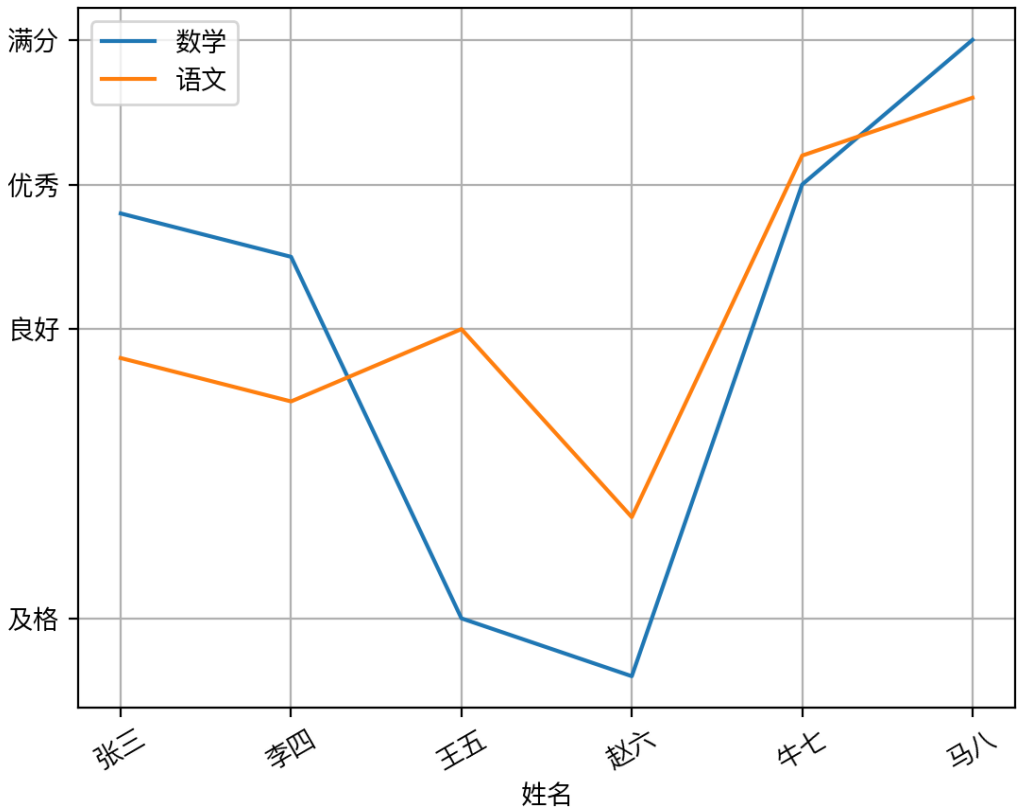
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False


sheet = pd.read_excel('团购.xlsx', sheet_name="3月2日",
                      header=1, usecols=['房号', '客户名', '合
计'])
df = sheet[2:-3]
print(df)
```

```
df.plot.line(x='客户名', y='合计')
plt.xticks(rotation=90)  # X 轴标签旋转90度
plt.show()
```




```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False

data = {'序号': list(range(6)),
        '姓名': ['张三', '李四', '王五', '赵六', '牛七', '马八'],
        '年龄': [23, 25, 26, 25, 25, 27],
        '生日': ['2001-12-01', '2001-12-05', '2001-10-01',
'2001-1-5', '2002-2-15', '2001-08-01'],
        '数学': [88, 85, 60, 56, 90, 100],
        '语文': [78, 75, 80, 67, 92, 96]
        }
```

```
df = pd.DataFrame(data)

df.plot.line(x='姓名', y=['数学', '语文'])
plt.xticks(rotation=30)
plt.yticks([60, 80, 90, 100], ['及格', '良好',
            '优秀', '满分'])
plt.grid()
plt.show()
```



标题/X标签/Y标签

```
plt.title('销售业绩表')
```

```
plt.xlabel('客户')
plt.ylabel('消费金额')
```

设置X/Y坐标范围

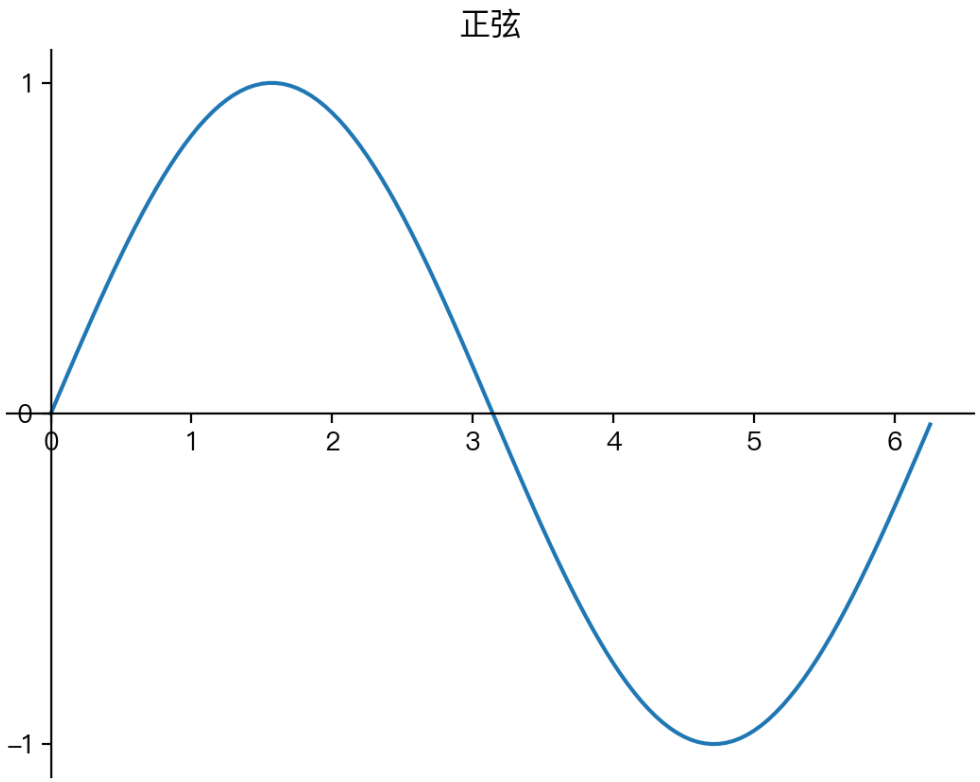
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-3, 3, 50)
y1 = 2*x + 1
y2 = x**2

plt.figure()
plt.plot(x, y2)
plt.plot(x, y1, color='red', linewidth=1.0, linestyle='--')

plt.xlim((-1, 2))
plt.ylim((-2, 3))
plt.show()
```

5.5. 边框设置

设置上边和右边无边框

```
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
```

```
#! /usr/bin/env python
# coding=utf-8
import matplotlib.pyplot as plt
import numpy as np
import math
import matplotlib as mpl
mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False

x = np.arange(0, math.pi*2, 0.05)
fig = plt.figure()
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])  # main axes
y = np.sin(x)
ax.plot(x, y)
ax.set_title('正弦')
ax.set_yticks([-1, 0, 1])

ax = plt.gca()
# 设置上边和右边无边框
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
# 设置x坐标的位置
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data', 0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data', 0))

plt.show()
```



5.6. plot 设置

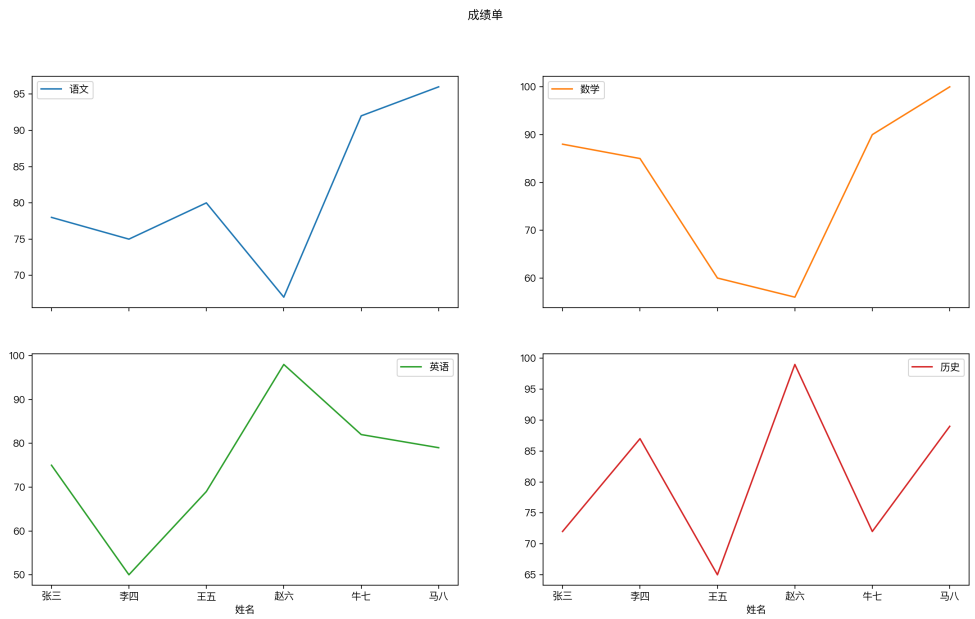
df.plot 参数

df.plot(x, y, kind, figsize, title, grid, legend, style)

x x 横坐标变量
y Y 纵坐标变量
kind 可视化图的种类，如line,hist, bar, barh, pie, kde, scatter
figsize 画布尺寸
title 标题
grid 是否显示格子线条
legend 是否显示图例

style 图的风格

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False

data = {'序号': list(range(6)),
        '姓名': ['张三', '李四', '王五', '赵六', '牛七', '马八'],
        '语文': [78, 75, 80, 67, 92, 96],
        '数学': [88, 85, 60, 56, 90, 100],
        '英语': [75, 50, 69, 98, 82, 79],
        '历史': [72, 87, 65, 99, 72, 89]
        }

df = pd.DataFrame(data)
print(df)
df.plot.line(x='姓名', y=['语文', '数学', '英语', '历史'],  # 4个
变量可视化
             legend=True,  # 显示图例
             subplots=True,  # 多子图并存
             layout=(2, 2),  # 图形排列2行2列
             figsize=(20, 10),  # 图形尺寸
             title='成绩单'  # 标题
             )

plt.show()
```



隐藏图例

legend=False

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

mpl.rcParams['font.sans-serif'] = ['PingFang HK']
mpl.rcParams['font.serif'] = ['PingFang HK']
plt.rcParams['axes.unicode_minus'] = False

data = {'序号': list(range(6)),
        '姓名': ['张三', '李四', '王五', '赵六', '牛七', '马八'],
        '年龄': [23, 25, 26, 25, 25, 27],
        '生日': ['2001-12-01', '2001-12-05', '2001-10-01',
'2001-1-5', '2001-2-15', '2001-08-01'],
        '数学': [88, 85, 60, 56, 90, 100],
        '语文': [78, 75, 80, 67, 92, 96]
        }
```

```
df = pd.DataFrame(data)
df.生日 = pd.to_datetime(df.生日)
df.plot.line(x='姓名', y='生日', legend=False)
plt.grid()
plt.show()
```

5.7. 保存为图片

```
plt.savefig('1.svg')

plt.savefig('2.png',dpi=400,bbox_inches='tight')
```

5.8. matplotlib 绘图风格

```
import matplotlib.pyplot as plt
print(plt.style.available)
```

输出风格

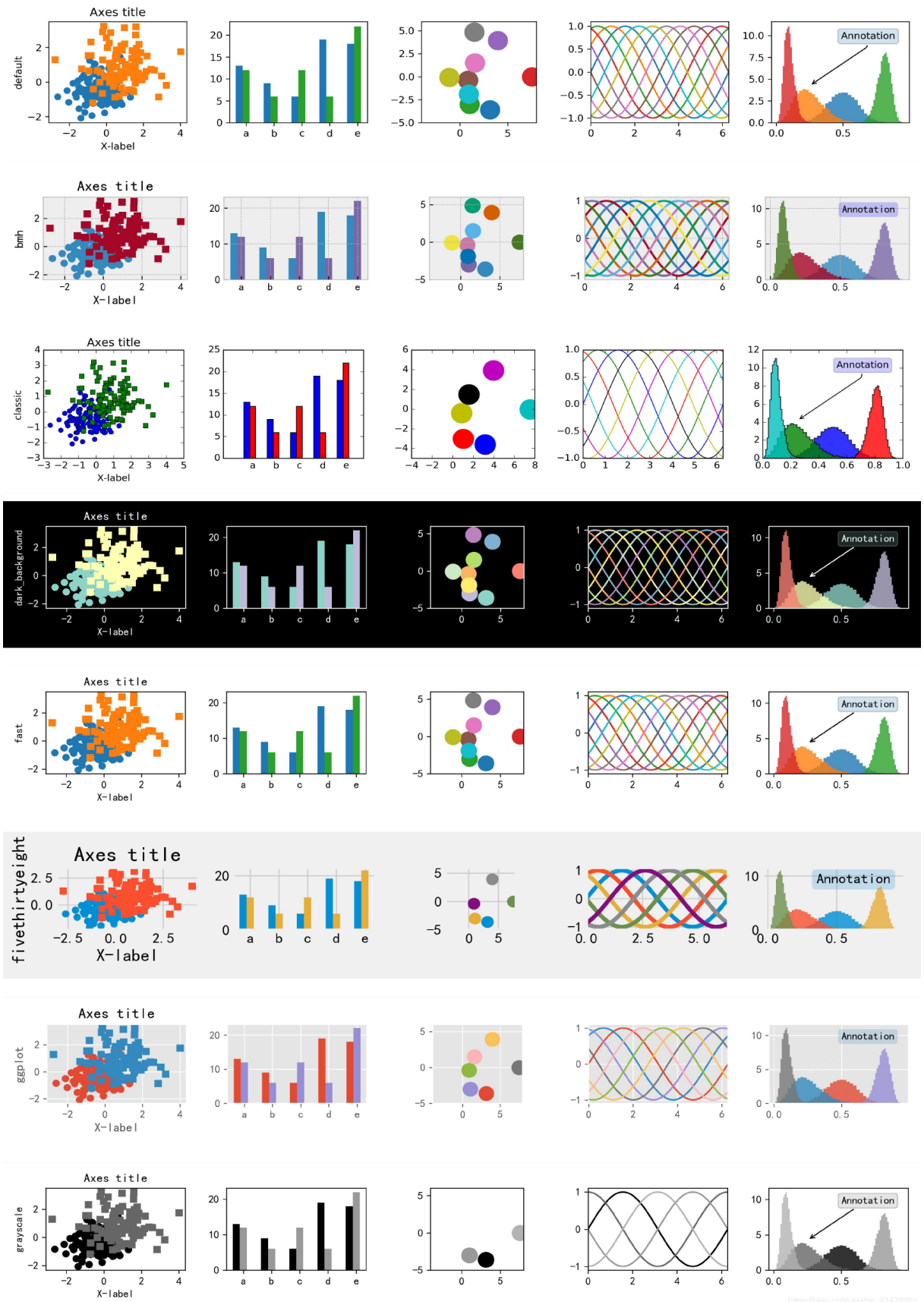
```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic',
'dark_background', 'fast', 'fivethirtyeight', 'ggplot',
'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind',
'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid',
'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-
paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk',
'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid',
```

```
'tableau-colorblind10']
```

应用一个风格

```
plt.style.use('ggplot')
```

风格参考


Axes title
default
X-label
Annotation
Axes title
bmh
X-label
Annotation
Axes title
classic
X-label
Annotation
Axes title
dark_background
X-label
Annotation
Axes title
fast
X-label
Annotation
Axes title
fivethirtyeight
X-label
Annotation
Axes title
ggplot
X-label
Annotation
Axes title
grayscale
X-label
Annotation

6. Pandas 实用函数

6.1. 日期范围

```
import pandas as pd
date = pd.date_range('2021-1-1', '2021-11-30', freq='15d')
print(date)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame(np.random.randn(15, 4),
index=pd.date_range(
    '2021/01/01', periods=15), columns=list('ABCD'))

df.plot()
plt.show()
```

6.2.

7. FAQ

7.1. xlrd.biffh.XLRDError: Excel xlsx file; not supported

```
neo@MacBook-Pro-Neo ~ % pip show xlrd
Name: xlrd
Version: 2.0.1
Summary: Library for developers to extract data from Microsoft
Excel (tm) .xls spreadsheet files
Home-page: http://www.python-excel.org/
Author: Chris Withers
Author-email: chris@withers.org
License: BSD
Location:
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6
/site-packages
Requires:
Required-by:
```

降级 xlrd 到 1.2.0 可以解决

```
pip uninstall xlrd
pip install xlrd==1.2.0
```

7.2. Missing optional dependency 'xlrd'

提示错误

```
ImportError: Missing optional dependency 'xlrd'. Install xlrd
>= 1.0.0 for Excel support Use pip or conda to install xlrd.
```

解决方案

```
neo@MacBook-Pro-Neo ~/workspace/netkiller-project % pip3
install openpyxl
Collecting openpyxl
  Downloading openpyxl-3.0.7-py2.py3-none-any.whl (243 kB)
     |████████████████████████████████| 243 kB 2.2 MB/s
Collecting et-xmlfile
  Using cached et_xmlfile-1.0.1.tar.gz (8.4 kB)
Building wheels for collected packages: et-xmlfile
  Building wheel for et-xmlfile (setup.py) ... done
  Created wheel for et-xmlfile: filename=et_xmlfile-1.0.1-py3-
none-any.whl size=8913
sha256=a4c964230c25014b55de927ac9284d184e6c729edf6f414cba97f3d9
4d3023aa
  Stored in directory:
/Users/neo/Library/Caches/pip/wheels/bb/9b/ff/8fdf015c95a572242
37bf88fbb9e45237cbe8177213bc13732
Successfully built et-xmlfile
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.0.1 openpyxl-3.0.7
```

第 13 章 股票

1. easyquotation - 快速获取新浪/腾讯的全市场行情

https://github.com/shidenggui/easyquotation

1.1. 安装

安装

```
pip install easyquotation
```

升级

```
pip install easyquotation --upgrade
```

下载源码，然后安装

```
git clone https://github.com/shidenggui/easyquotation.git
cd easyquotation
python setup.py install
```

1.2. 演示

```
import easyquotation
quotation = easyquotation.use('sina') # 新浪 ['sina'] 腾讯 ['tencent', 'qq']
```

2. akshare

```
import akshare as ak
import pandas as pd

# 获取股票数据
stock_zh_a_spot_df = ak.stock_zh_a_spot()

# 打印结果
print(stock_zh_a_spot_df)

# 保存到excel
stock_zh_a_spot_df.to_excel("stock.xlsx")
```

第 14 章 数据可视化

1. matplotlib

1.1. 直方图

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("data1.xlsx", "Sheet1")

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(df['年龄'], bins=7)
plt.title('Age distribution')
plt.xlabel('Age')
plt.ylabel('Employee')
plt.show()
```

1.2. 显示中文

```
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.font_manager import FontProperties

title =
FontProperties(fname=r"/System/Library/Fonts/PingFang.ttc",
size=14)
font =
FontProperties(fname=r"/System/Library/Fonts/PingFang.ttc",
size=10)
```

```
df = pd.read_excel("data.xlsx", "Sheet1")

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(df['年龄'], bins=7)
plt.title('年龄分布图', fontproperties=title)
plt.xlabel('年龄', fontproperties=font)
plt.ylabel('员工数量', fontproperties=font)
plt.show()
```

2. pyecharts

https://pyecharts.org/

安装

```
$ pip install pyecharts
```

部分 IV. 人工智能 AI

第 15 章 AI 相关

1. tokenizers

1.1. Normalization

文本清洗，Normalization 对原始文本 sentence 执行一系列清洗操作，如：删除空格、去除重音字符、小写化

```
from tokenizers import normalizers
from tokenizers.normalizers import NFD, StripAccents

normalizer = normalizers.Sequence([NFD(), StripAccents()])
text = normalizer.normalize_str("Héllò hôw are ü?")
print(text)
# "Hello how are u?"
```

1.2. Pre-Tokenization

拆分文本，并标记文本的位置

```
from tokenizers.pre_tokenizers import Whitespace
from tokenizers import pre_tokenizers
from tokenizers.pre_tokenizers import Digits

#Whitespace使用正则表达式\w+|[^\w\s]+，即以word开头，以空格或非word
结尾来拆分token，返回数据  List[Tuple[str, Offsets]]:
pre_tokenizer = Whitespace()
data1 = pre_tokenizer.pre_tokenize_str("What's your nickname?
My nickname is netkiller.")
print(data1)
```

```
pre_tokenizer = pre_tokenizers.Sequence([Whitespace(),
Digits(individual_digits=True)])
data2 =
pre_tokenizer.pre_tokenize_str("https://www.netkiller.cn")
print(data2)
```

输出结果

```
[('What', (0, 4)), ("'", (4, 5)), ('s', (5, 6)), ('your', (7,
11)), ('nickname', (12, 20)), ('?', (20, 21)), ('My', (22,
24)), ('nickname', (25, 33)), ('is', (34, 36)), ('netkiller',
(37, 46)), ('.', (46, 47))]
[('https', (0, 5)), ('://', (5, 8)), ('www', (8, 11)), ('.',
(11, 12)), ('netkiller', (12, 21)), ('.', (21, 22)), ('cn',
(22, 24))]
```

2. transformers

2.1. 安装 transformers

安装 transformers

```
pip install transformers
```

2.2. 加载本地模型

```
#!/usr/bin/python
# -*-coding: utf-8-*-
from transformers import BertTokenizer, BertModel

model = "./transformers/bert-base-chinese"
tokenizer = BertTokenizer.from_pretrained(model)
print(tokenizer.tokenize("I have a good time, thank you."))
```

```
neo@MacBook-Pro-M2 ~/w/test> ll transformers/bert-base-chinese/
total 804576
-rw-r--r--  1 neo  staff   624B 10 23 17:00 config.json
-rw-r--r--  1 neo  staff   392M 10 23 17:05 model.safetensors
-rw-r--r--  1 neo  staff   263K 10 23 17:05 tokenizer.json
-rw-r--r--  1 neo  staff    29B 10 23 17:05
tokenizer_config.json
-rw-r--r--@ 1 neo  staff   107K 10 23 16:13 vocab.txt
```

运行结果

```
neo@MacBook-Pro-M2 ~/w/test> python3
/Users/neo/workspace/test/test.py
['[UNK]', 'have', 'a', 'good', 'time', ',', 'than', '##k',
'you', '.']
```

2.3. 自动下载模型

```
#!/usr/bin/python
# -*-coding: utf-8-*-
from transformers import AutoTokenizer, AutoModel

modelPath = "hfl/chinese-macbert-base"
tokenizer = AutoTokenizer.from_pretrained(modelPath)
print(tokenizer.tokenize("I have a good time, thank you."))
```

2.4. 编码

```
#!/usr/bin/python
# -*-coding: utf-8-*-
from transformers import BertTokenizer, BertModel

model_path = "./transformers/bert-base-chinese"
tokenizer = BertTokenizer.from_pretrained(model_path)

print(tokenizer.encode('你好世界'))
print(tokenizer.encode_plus('我不喜欢这世界','你好中国'))
print(tokenizer.convert_ids_to_tokens(tokenizer.encode('我爱
你')))
```

2.5. 计算向量

```
from transformers import AutoTokenizer, AutoModel

cache_dir = "/tmp/transformers"
# model = "hfl/chinese-macbert-base"
pretrained_model_name_or_path = "bert-base-chinese"

tokenizer =
AutoTokenizer.from_pretrained(pretrained_model_name_or_path,
cache_dir=cache_dir)  # , force_download=True
model =
AutoModel.from_pretrained(pretrained_model_name_or_path,
cache_dir=cache_dir)

sentences = ['https://www.netkiller.cn']

inputs = tokenizer(sentences, return_tensors="pt")
outputs = model(**inputs)
array = outputs.pooler_output.tolist()
print(array)
```

2.6. FAQ

隐藏警告

```
- This IS expected if you are initializing BertModel from the
checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification
model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from
the checkpoint of a model that you expect to be exactly
```

```
identical (initializing a BertForSequenceClassification model
from a BertForSequenceClassification model).
```

解决方案

```
from transformers import logging
logging.set_verbosity_warning()

或:

from transformers import logging
logging.set_verbosity_error()
```

打开 config.json 文件，查看 architectures 配置项，讲 BertModel 更换为 BertForMaskedLM 即可

```
{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
```

```
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "type_vocab_size": 2,
  "vocab_size": 21128
}
```

```
Some weights of the model checkpoint at ./transformers/bert-
base-chinese were not used when initializing BertForMaskedLM:
['cls.seq_relationship.bias', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertForMaskedLM from
the checkpoint of a model trained on another task or with
another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForMaskedLM
from the checkpoint of a model that you expect to be exactly
identical (initializing a BertForSequenceClassification model
from a BertForSequenceClassification model).
```

添加 from_tf=True 参数

```
#!/usr/bin/python
# -*-coding: utf-8-*-
from transformers import BertTokenizer,
BertModel,BertConfig,BertForMaskedLM
pretrained_model_name_or_path = "./transformers/bert-base-
chinese"
tokenizer =
BertTokenizer.from_pretrained(pretrained_model_name_or_path)
model =
BertForMaskedLM.from_pretrained(pretrained_model_name_or_path
, from_tf=True)
```

```
All TF 2.0 model weights were used when initializing
BertForMaskedLM.

All the weights of BertForMaskedLM were initialized from the TF
2.0 model.
If your task is similar to the task the model of the checkpoint
was trained on, you can already use BertForMaskedLM for
predictions without further training.
```

第 16 章 OCR

OCR(Optical Character Recognition)：光学字符识别，是指电子设备（例如扫描仪或数码相机）检查纸上打印的字符，通过检测暗、亮的模式确定其形状，然后用字符识别方法将形状翻译成计算机文字的过程。

OCR 的应用场景:

- 扫描件的文字识别：纸质扫描件，PDF文件，纸书籍转电子书
- 卡证文字识别：名片，身份证，银行卡
- 车牌设别：多用在停车场和交警执法
- 票据识别：报销凭证，税务票据，医疗票据
- 教育领域：公式识别，自动判卷，自动答题

OCR 原理

```
扫描/拍照 > 图像输入 > 二值化 > 去噪声 > 倾斜校正 > 版面分析 > 字符切割 > 字符识别 > 版面恢复 > 文字校对
```

扫描或拍照：使用扫描仪将纸质资料扫描成图像文件，手机拍照也能达到同样效果，然后将图像文件交给OCR软件进行处理。

首先把彩色图像编程黑白图像（黑色文字，白色背景），识别系统不关心颜色。

通过算法去除图像中的影响因素，噪点，污染，这部叫去噪声，类似Ps中的色阶，降噪，模糊一类的操作，让图像中的文字更突出。

然后矫正倾斜的图像，分析文本段落，切割字符，识别字符。

将识别的字符，重新排版，生成 Text 文本文件。

较为高级OCR系统，带有AI语法分析，能够实现拼写检查，语法校对等等。

1. EasyOCR

https://www.jaided.ai

1.1. 安装 EasyOCR

```
pip install easyocr
```

EasyOCR 源码

https://github.com/JaidedAI/EasyOCR

安装最新版

```
pip install git+git://github.com/jaidedai/easyocr.git
```

1.2. 操作演示

```
import easyocr

reader = easyocr.Reader(['ch_sim','en'])
result = reader.readtext('test.jpg')

print(result)

for text in result:
```

```
    word = text[1]
    print(word)
```

1.3. 命令行运行

```
$ easyocr -l ch_sim en -f chinese.jpg --detail=1 --gpu=True
```

1.4. 函数

Reader()

禁用 GPU

```
reader = easyocr.Reader(['ch_sim','en'], gpu = False)
```

readtext() 函数

进输出识别结果

```
reader.readtext('chinese.jpg', detail = 0)
```

输出结果

```
['愚园路', '西', '东', '315', '309', 'Yuyuan Rd.', 'W', 'E']
```

1.5. urllib.error.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:833)>

解决方法，加入下面代码

```
import ssl
ssl._create_default_https_context =
ssl._create_unverified_context
```

或运行

```
pip install --upgrade certifi
```

2. Tesseract

Tesseract：开源的OCR识别引擎，初期Tesseract引擎由HP实验室研发，后来贡献给了开源软件业，后由Google进行改进、修改bug、优化，重新发布。

2.1. 安装 Tesseract

```
neo@MacBook-Pro-Neo ~/workspace/python/ocr % brew install
tesseract
neo@MacBook-Pro-Neo ~/workspace/python/ocr % brew install
tesseract-lang
neo@MacBook-Pro-Neo ~/workspace/python/ocr % pip3 install
pytesseract
```

2.2. 演示 Tesseract

```
#!/usr/bin/python3
# -*- coding:utf-8 -*-

from PIL import Image
import pytesseract

# 英文识别
english =
pytesseract.image_to_string(Image.open("english.png"))
print(english)

print('-' * 50)

# 简体中文识别
chinese =
pytesseract.image_to_string(Image.open("chinese.png"),
```

```
lang='chi_sim')
print(chinese)
```

2.3.

```

```

```

```

第 17 章 语音处理

1. TTS(Text To Speech) 文本转语音

pyttsx3 - 语音朗读

TTS(Text To Speech) 译为从文本到语音，TTS是人工智能AI的一个模组，是人机对话的一部分，即让机器能够说话。

TTS是语音合成技术应用的一种，首先采集语音波形，然后进行优化处理，最后存储在数据库中，合成语音是提取波形转换成自然语音输出。

TTS 有哪些应用场景

1. TTS 能帮助有视觉障碍的人阅读计算机上的信息
2. 懒人听书，很多人没有时间读书，我们可以通过TTS将书中的内容朗读出来
3. 与声音识别程序一起使用，实现人机交互，例如客服系统的对话脚本
4. 不方便视觉交互场景，例如驾驶汽车，我们可以将短信朗读出来，来电电话号码朗读出来
5. 公交车报站

1.1. 安装 pyttsx3

```
pip install pyttsx3
```

Linux

```
[root@gitlab ~]# dnf install espeak-ng
```

```
libespeak.so.1: cannot open shared object file: No such file or
directory
```

1.2. 演示

```
#coding=utf-8
import pyttsx3
pyttsx3.speak("Hello World!")
```

1.3. 方法详解

say() 方法

speak() 实际上是下面代码的封装

```
#coding=utf-8
import pyttsx3
engine = pyttsx3.init()
engine.say("Hello World!")
engine.runAndWait()
```

save_to_file()

```
engine.save_to_file(text, 'test.mp3')
```

调整人声类型

男性（voices[0].id）、女性（voices[1].id）

```
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[0].id)
```

调整语速

一般范围一般在0~500之间

```
rate = engine.getProperty('rate')
engine.setProperty('rate', 200)
```

调整声量

范围在0~1之间

```
volume = engine.getProperty('volume')
```

```
engine.setProperty('volume',0.8)
```

查看语音引擎

```
voices = engine.getProperty('voices')
for item in voices:
    print(item)
```

1.4. 例子

```
import pyttsx3
engine = pyttsx3.init() # object creation

""" RATE"""
rate = engine.getProperty('rate')   # getting details of
current speaking rate
print (rate)                        #printing current voice
rate
engine.setProperty('rate', 125)     # setting up new voice
rate


"""VOLUME"""
volume = engine.getProperty('volume')   #getting to know
current volume level (min=0 and max=1)
print (volume)                          #printing current
volume level
engine.setProperty('volume',1.0)    # setting up volume level
between 0 and 1

"""VOICE"""
voices = engine.getProperty('voices')       #getting details
of current voice
```

```
#engine.setProperty('voice', voices[0].id)  #changing index,
changes voices. o for male
engine.setProperty('voice', voices[1].id)   #changing index,
changes voices. 1 for female

engine.say("Hello World!")
engine.say('My current speaking rate is ' + str(rate))
engine.runAndWait()
engine.stop()

"""Saving Voice to a file"""
# On linux make sure that 'espeak' and 'ffmpeg' are installed
engine.save_to_file('Hello World', 'test.mp3')
engine.runAndWait()
```

2. STT(Speech To Text) 语音转文本

2.1. SpeechRecognition

https://github.com/Uberi/speech_recognition

安装

```
pip install SpeechRecognition
```

麦克风相关

```
brew install portaudio
pip install pyaudio
```

运行下面命令授权访问麦克风

```
neo@MacBook-Pro-Neo ~ % python3 -m speech_recognition
```

查看麦克风列表

```
import speech_recognition as sr

for index, name in
```

```
enumerate(sr.Microphone.list_microphone_names()):
    print("Microphone with name \"{1}\" found for
`Microphone(device_index={0})`".format(index, name))
```

输出结果

```
neo@MacBook-Pro-Neo ~/workspace/python/speech % python3
microphone.py
Microphone with name "Built-in Microphone" found for
`Microphone(device_index=0)`
Microphone with name "Built-in Output" found for
`Microphone(device_index=1)`
```

指定麦克风设备

```
import speech_recognition as sr
print(sr.__version__) # just to print the version not
required
r = sr.Recognizer()
mic = sr.Microphone(device_index=1) #my device index is 1,
you have to put your device index
```

噪声抑制

```
import speech_recognition as sr
print(sr.__version__) # just to print the version not
required
r = sr.Recognizer()
my_mic = sr.Microphone(device_index=1) #my device index is 1,
you have to put your device index
```

```
with my_mic as source:
    print("Say now!!!!")
    r.adjust_for_ambient_noise(source) #reduce noise
    audio = r.listen(source) #take voice input from the
microphone
print(r.recognize_google(audio)) #to print voice into text
```

PocketSphinx 文件转文本

PocketSphinx默认仅支持英文识别，中文需要下载语言模型文件，Mandarin 为中文普通话。

```
brew install swig
brew install pocketsphinx
pip install PocketSphinx
```

从文件识别

```
import speech_recognition as sr

# obtain audio from the file
recognizer = sr.Recognizer()
audioFile = sr.AudioFile(r"english.wav")
with audioFile as source:
    audio = recognizer.record(source)
# recognize speech using Sphinx
try:
    print("Sphinx thinks you said: " +
recognizer.recognize_sphinx(audio))
except sr.UnknownValueError:
    print("Sphinx could not understand audio")
except sr.RequestError as e:
    print("Sphinx error; {0}".format(e))
```

从麦克风识别

```
#!/usr/bin/env python3

import speech_recognition as sr

print(sr.__version__)

for index, name in
enumerate(sr.Microphone.list_microphone_names()):
    print("Microphone with name \"{1}\" found for
`Microphone(device_index={0})`".format(index, name))

# obtain audio from the microphone
r = sr.Recognizer()
with sr.Microphone() as source:
    print("Say something!")
    audio = r.listen(source)

# recognize speech using Sphinx
try:
    print("Sphinx thinks you said: " +
r.recognize_sphinx(audio))
except sr.UnknownValueError:
    print("Sphinx could not understand audio")
except sr.RequestError as e:
    print("Sphinx error; {0}".format(e))
```

Google Cloud Speech API

使用谷歌产品先要会使用科学上网，你懂得！

```
import speech_recognition as sr

r = sr.Recognizer()
with sr.Microphone() as source:
    print("Say something!")
    audio = r.listen(source)
try:
    text = r.recognize_google(audio)
    print("You said: " + text)
except sr.UnknownValueError:
    print("Google Speech Recognition could not understand
audio")
except sr.RequestError as e:
    print("Could not request results from Google Speech
Recognition service" + format(e))
```

指定默认语言

```
text = r.recognize_google(audio, language='zh-CN', show_all=
True)
text = r.recognize_google(audio_data, language="es-ES")
```

IBM Speech to Text

使用IBM的服务需要一个云账号 IBM Cloud，如你你没有请先注册一个账号，然后创建 Speech To Text 服务。

测试 Speech to Text 是否正常工作

```
neo@MacBook-Pro-Neo ~/workspace/python/speech % wget
```

```
https://watson-developer-cloud.github.io/doc-tutorial-
downloads/speech-to-text/audio-file.flac

neo@MacBook-Pro-Neo ~/workspace/python/speech % curl -X POST -u
"apikey:eXuTdDOg_l7Ljp5bV8NpFsswVq58ebf2Kr-K5dpp5SZK" \
--header "Content-Type: audio/flac" \
--data-binary audio-file.flac \
"https://api.au-syd.speech-to-
text.watson.cloud.ibm.com/instances/8a7df79c-c8fe-4e31-8000-
c44bbd025b22/v1/recognize"
```

```
#!/usr/bin/env python3

import speech_recognition as sr
import ssl

ssl._create_default_https_context =
ssl._create_unverified_context

# obtain path to "english.wav" in the same folder as this
script
from os import path
# AUDIO_FILE =
path.join(path.dirname(path.realpath(__file__)),
"english.wav")
# AUDIO_FILE =
path.join(path.dirname(path.realpath(__file__)),
"french.aiff")
AUDIO_FILE = path.join(path.dirname(path.realpath(__file__)),
"chinese.flac")
print(AUDIO_FILE)

# use the audio file as the audio source
r = sr.Recognizer()
with sr.AudioFile(AUDIO_FILE) as source:
    audio = r.record(source)  # read the entire audio file


try:
    print("IBM Speech to Text thinks you said " +
```

```
r.recognize_ibm(audio, username="netkiller@msn.com",
password="******"))
except sr.UnknownValueError:
    print("IBM Speech to Text could not understand audio")
except sr.RequestError as e:
    print("Could not request results from IBM Speech to Text
service; {0}".format(e))
```

2.2. DeepSpeech

https://deepspeech.readthedocs.io/en/latest/index.html

```
# Install DeepSpeech
pip3 install deepspeech

# Download pre-trained English model files
curl -LO
https://github.com/mozilla/DeepSpeech/releases/download/v0.9.3/
deepspeech-0.9.3-models.pbmm
curl -LO
https://github.com/mozilla/DeepSpeech/releases/download/v0.9.3/
deepspeech-0.9.3-models.scorer

# Download example audio files
curl -LO
https://github.com/mozilla/DeepSpeech/releases/download/v0.9.3/
audio-0.9.3.tar.gz
tar xvf audio-0.9.3.tar.gz

# Transcribe an audio file
deepspeech --model deepspeech-0.9.3-models.pbmm --scorer
deepspeech-0.9.3-models.scorer --audio audio/2830-3980-0043.wav
```

3. Baidu AipSpeech

```
pip install baidu-aip
```

4. AI文字转语音模型Bark

https://github.com/suno-ai/bark

5. Automatic Speech Recognition

5.1. kaldi

```
docker run -it kaldiasr/kaldi:latest bash
docker run -it --runtime=nvidia kaldiasr/kaldi:gpu-latest bash
```

```
docker run -it kaldiasr/kaldi:latest bash
```

5.2. OpenAI Whisper

https://github.com/openai/whisper

```
import openai
audio_file= open("/path/to/file/audio.mp3", "rb")
transcript = openai.Audio.transcribe("whisper-1", audio_file)
```

第 18 章 视频

1. 摄像头

```
pip install opencv_python
```

```
import cv2

# 设置摄像头 0是默认的摄像头 如果你有多个摄像头的话呢，可以设置
1,2,3....
cap = cv2.VideoCapture(1) # 苹果电脑 Facetime 摄像头是 1

# 进入无限循环
while True:
    ret, frame = cap.read()      # 摄像头拍摄图像（单位是帧）
    cv2.imshow('frame', frame)  # 将frame中的图像显示出来
    c = cv2.waitKey(1)                    # 判断退出条件 当按
下'Q'键的退出程序
    if c == ord('q'):
        break

# 释放摄像头
cap.release()
# 销毁窗口
cv2.DestroyAllWindows()
```

拍照

```
import cv2
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW) # 打开摄像头
```

```
while True:
  # get a frame
  ret, frame = cap.read()
  frame = cv2.flip(frame, 1) # 摄像头是和人对立的，将图像左右调换回来正常显示
  # show a frame
  cv2.imshow("capture", frame) # 生成摄像头窗口

  if cv2.waitKey(1) & 0xFF == ord('q'): # 如果按下q 就截图保存并退出
    cv2.imwrite("test.png", frame) # 保存路径
    break

cap.release()
cv2.destroyAllWindows()
```

2. MoviePy

```
pip install moviepy
```

2.1. 合成视频

```
clip1 = VideoFileClip("test1.mp4")
clip2 = VideoFileClip("test2.mp4")
clip3 = VideoFileClip("test3.mp4")

video = CompositeVideoClip([clip1,clip2,clip3],size=(1920,
1080))
video.write_videofile("video.mp4")
```

```
from moviepy.editor import VideoFileClip,
concatenate_videoclips
# 重新设置视频大小
clip1 = VideoFileClip("chip1.mp4").resize((1920, 1080))
# 剪切指定时间片段（单位秒）
clip2 = VideoFileClip("chip2.mp4").subclip(50,60)
# 剪切持续时间（单位秒）
clip3 = VideoFileClip("chip3.mp4").set_duration(5)
# 按顺序拼接视频
final_clip = concatenate_videoclips([clip1,clip2,clip3])
final_clip.write_videofile("video.mp4")
```

2.2. 提取视频中的音频

```
#coding=utf-8
import moviepy.editor as mov

video = mov.VideoFileClip('test.mp4')
audio = video.audio
audio.write_audiofile('test.wav')
```

```
neo@MacBook-Pro-Neo ~/workspace/python/video % python3 audio.py
MoviePy - Writing audio in test.wav
MoviePy - Done.

neo@MacBook-Pro-Neo ~/workspace/python/video % ls *.wav
test.wav
```

2.3. 加字幕

```
brew install ImageMagick
```

查看可用字体

```
from moviepy.editor import TextClip
```

```
print ( TextClip.list("font") )
```

运行结果

```
neo@MacBook-Pro-Neo ~/workspace/python/video % /usr/bin/python3
/Users/neo/workspace/python/video/font.py
['AvantGarde-Book', 'AvantGarde-BookOblique', 'AvantGarde-
Demi', 'AvantGarde-DemiOblique', 'Bookman-Demi', 'Bookman-
DemiItalic', 'Bookman-Light', 'Bookman-LightItalic', 'Courier',
'Courier-Bold', 'Courier-BoldOblique', 'Courier-Oblique',
'fixed', 'Helvetica', 'Helvetica-Bold', 'Helvetica-
BoldOblique', 'Helvetica-Narrow', 'Helvetica-Narrow-Bold',
'Helvetica-Narrow-BoldOblique', 'Helvetica-Narrow-Oblique',
'Helvetica-Oblique', 'NewCenturySchlbk-Bold',
'NewCenturySchlbk-BoldItalic', 'NewCenturySchlbk-Italic',
'NewCenturySchlbk-Roman', 'Palatino-Bold', 'Palatino-
BoldItalic', 'Palatino-Italic', 'Palatino-Roman', 'Symbol',
'Times-Bold', 'Times-BoldItalic', 'Times-Italic', 'Times-
Roman']
```

指定字体路径

```
from moviepy.editor import TextClip
font_path = './font/heimi.ttf'
txtClip = TextClip('《Netkiller Python 手札》',color='white',
font=font_path,kerning = 5, fontsize=100)
```

苹果电脑的字体安装在 /System/Library/Fonts/ 目录中。

```
from moviepy.editor import *
```

```
video = VideoFileClip("test.mp4").subclip(50,60)

# Make the text. Many more options are available.
txt_clip = ( TextClip("《Netkiller Python 手
札》",font="/System/Library/Fonts/PingFang.ttc",
fontsize=70,color='white')
             .set_position('center')
             .set_duration(10) )

result = CompositeVideoClip([video, txt_clip]) # Overlay text
on video
result.write_videofile("test.webm",fps=25) # Many options...
```

2.4. 音量大小调节

```
VolumeMultiple = 0.8
clip = clip.volumex(VolumeMultiple)
```

2.5. 音频合成剪辑

```
# 对白
dialogue1 = AudioFileClip("voice/tom.mp3")
dialogue2 = AudioFileClip("voice/jerry.mp3")

# 制作混合音轨
concat = concatenate_audioclips([dialogue1, dialogue2])
compo = CompositeAudioClip([dialogue1.volumex(1.2), # 音量设置
大小
                            dialogue2.set_start(5)])# 音频起始
时间
```

2.6. 视频中设置背景音乐

```
bgm = AudioFileClip("music/paino.mp3")
videoclip = videoclip.set_audio(bgm)
```

2.7. 画面截图

```
# 保存视频第一帧的画面
clip.save_frame("Screenshot.jpeg")

# 保存视频第10秒的画面
clip.save_frame("Screenshot.png", t=10)
```

2.8.

第 19 章 人脸识别

人脸识别，是基于人的脸部特征信息进行身份识别的一种生物识别技术。用照相机、摄像机或摄像头采集含有人脸的图像或视频流，并自动在图像中检测和跟踪人脸，进而对检测到的人脸进行脸部识别的一系列相关技术，通常也叫做人像识别、面部识别。

https://github.com/ageitgey/face_recognition

face_recognition，这个是一个开源的人脸识别库，该模块是基于业内领先的C++开源库dlib中的深度学习模型，用Labeled Faces in the Wild人脸数据集进行测试，有高达99.38%的准确率。

1. 安装

需要用到 dlib,face_recognition,opencv 这三个模块。

```
pip3 install dlib
pip3 install opencv-python
pip3 install face_recognition
```

2. 命令行工具

2.1. face_detection - 在单张图片或一个图片文件夹中定位人脸位置

首先，准备一个已知名字的人脸图片文件夹，每个人一张图，图片的文件名对应人的名字

然后，你需要准备第二个图片文件夹，文件夹内放置你希望识别的图片

最后，适用face_recognition命令行工具，认出是谁的人脸。

```
$ face_recognition ./know_people/ ./unknown_pictures/
```

2.2. face_detection - 在单张图片或一个图片文件夹中定位人脸位置

输出结果是人脸坐标位置

```
neo@MacBook-Pro-Neo ~/workspace/python/face % face_detection ./
./6.jpeg,251,423,509,165
./5.jpeg,85,365,157,293
./face.jpeg,198,680,508,370
./2.jpg,321,1023,692,651
./1.jpg,164,818,474,508
```

每一行对应图片中的一张脸，输出坐标代表着这张脸的上、右、下、左像素点。

3. 定位头像位置

```
import face_recognition as face

image = face.load_image_file("face.jpeg")
face_locations = face.face_locations(image)

print(face_locations)
```

```
neo@MacBook-Pro-Neo ~/workspace/python/face % python3.9
/Users/neo/workspace/python/face/face_locations.py
[(231, 676, 498, 409)]
```

使用深度学习模型达到更加精准的人脸定位

```
import face_recognition

image = face_recognition.load_image_file("my_picture.jpg")
face_locations = face_recognition.face_locations(image,
model="cnn")
```

例 19.1. 找出图片中头像

```
import PIL.Image as img
import face_recognition as face
```

```
file = "face.jpeg"
image = face.load_image_file(file)
face_locations = face.face_locations(image)

print(face_locations)

if face_locations:
    (top, right, bottom, left) = face_locations[0]
    im = img.open(file)
    box = (left, top, right, bottom)
    head = im.crop(box)
    head.save('head.jpg')
```

4. 人脸比较

```
import face_recognition

picture_of_me = face_recognition.load_image_file("me.jpeg")
my_face_encoding =
face_recognition.face_encodings(picture_of_me)[0]

# my_face_encoding now contains a universal 'encoding' of my
facial features that can be compared to any other picture of
a face!

unknown_picture =
face_recognition.load_image_file("unknown.jpeg")
unknown_face_encoding =
face_recognition.face_encodings(unknown_picture)[0]

# Now we can see the two face encodings are of the same
person with `compare_faces`!

results = face_recognition.compare_faces(
    [my_face_encoding], unknown_face_encoding)

if results[0] == True:
    print("It's a picture of me!")
else:
    print("It's not a picture of me!")

results = face_recognition.compare_faces([my_face_encoding],
my_face_encoding)

if results[0] == True:
    print("It's a picture of me!")
else:
    print("It's not a picture of me!")
```

5. 摄像头识别人脸

```
import face_recognition
import cv2
import numpy as np

# Get a reference to webcam #0 (the default one)
video_capture = cv2.VideoCapture(1)

# Load a sample picture and learn how to recognize it.
image = face_recognition.load_image_file("me.jpg")
known_face_encodings = face_recognition.face_encodings(image)
[0]

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Convert the image from BGR color (which OpenCV uses) to
RGB color (which face_recognition uses)
    rgb_frame = frame[:, :, ::-1]

    # Find all the faces and face enqcodings in the frame of
video
    face_locations =
face_recognition.face_locations(rgb_frame)
    face_encodings =
face_recognition.face_encodings(rgb_frame, face_locations)

    # Loop through each face in this frame of video
    for (top, right, bottom, left), face_encoding in
zip(face_locations, face_encodings):
        # See if the face is a match for the known face(s)
        matches = face_recognition.compare_faces(
            [known_face_encodings], face_encoding)

        name = "Unknown"

        if matches[0] == True:
            print("It's a picture of me!")
```

```
        else:
            print("It's not a picture of me!")

        # Or instead, use the known face with the smallest
distance to the new face
        face_distances = face_recognition.face_distance(
            [known_face_encodings], face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = 'netkiller'

        # Draw a box around the face
        cv2.rectangle(frame, (left, top), (right, bottom),
(0, 0, 255), 2)

        # Draw a label with a name below the face
        cv2.rectangle(frame, (left, bottom - 35),
                      (right, bottom), (0, 0, 255),
cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6),
                    font, 1.0, (255, 255, 255), 1)

    # Display the resulting image
    cv2.imshow('Video', frame)

    # Hit 'q' on the keyboard to quit!
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
```

第 20 章 自然语言处理

自然语言处理（Natural Language Processing）

中文分词(Chinese Word Segmentation)，英文是由单词组成，并使用空格来分开每个单词，而中文以字为单位，由字组成词，字于词的含有可能完全不同，因此，中文分词NPL相比英文分词要复杂的多。

中文分词技术主要使用场景有哪些:

- 搜索优化，关键词提取
- 语义分析
- 非结构化文本媒体内容，如社交信息
- 文本聚类，根据内容生成自动分类
- 文章自动标签
- 情感分析
- 词性提取

1. 全文索引（full-text index）

全文索引 (也称全文检索)是目前搜索引擎使用的一种关键技术。

全文索引是利用分词技术，首先将一段文本中的关键词提取出来，同时获得该词在文本中出现的位置。然后对提取的关键词做索引。

当用户查询关键词，并且命中的时候，返回查询结果。同时记录关键词的查询频率，再进行词频优化，以便下次命中率更高。

2. 人机对话

在前面的视频中我们介绍了“TTS(Text To Speech) 文本转语音”和“STT(Speech To Text) 语音转文本”，在人工智能领域仅仅实现TTS和STT还无法做到人机对话，因为机器无法理解我们说的话，所以无法执行我们下达的指令。

机器是怎样听懂我们所说的话？答案是提取一句话中的重点部分。

例如：小爱同学请播放音乐。重点是：“播放音乐”四个字，其他都可以忽略。

进一步分析，其中“播放”是动词，“音乐”是名词。

3. 情感分析

情感分析是自然语言处理的一个分支，通过分词技术，提取文本中带有情感色彩（褒义贬义/正向负向）的主观性文本进行分析，以确定该文本的观点、喜好、情感倾向。

例如：

“这本书读起来让人爱不释手” 归为正向，“这本书很难看”归为负向。

“这款产品很不错”归为正向，“这款产能缺陷很多”归为负向。

应用场景：

商品评论内容分析，舆情监控

4. 常用的 Python 分词库

Python 分词库：

- jieba（结巴分词）
- THULAC（清华大学自然语言处理与社会人文计算实验室）
- pkuseg（北京大学语言计算与机器学习研究组）
- SnowNLP
- pynlpir
- CoreNLP
- pyltp

5. 结巴中文分词

https://github.com/fxsjy/jieba

安装

```
pip install jieba
pip install paddlepaddle
```

5.1. 分词演示

```
# encoding=utf-8
import jieba
import paddle
paddle.enable_static()
jieba.enable_paddle()  # 启动paddle模式。
strs = ["我来到北京清华大学", "乒乓球拍卖完了", "中国科学技术大学"]
for str in strs:
    seg_list = jieba.cut(str, use_paddle=True)  # 使用paddle模式
    print("Paddle Mode: " + '/'.join(list(seg_list)))

seg_list = jieba.cut("我来到北京清华大学", cut_all=True)
print("Full Mode: " + "/ ".join(seg_list))  # 全模式

seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
print("Default Mode: " + "/ ".join(seg_list))  # 精确模式

seg_list = jieba.cut("他来到了网易杭研大厦")  # 默认是精确模式
print(", ".join(seg_list))

seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所，后在日本京都大学深造")  # 搜索引擎模式
print(", ".join(seg_list))
```

5.2. 日志设置

```
import jieba
import logging
logger = logging.getLogger()
# 配置 logger 禁止输出无用的信息
jieba.default_logger = logger

text = "他来到了网易杭研大厦"

words = jieba.cut(text)
print(", ".join(words))
print("-" * 50)
# 将 “杭研大厦”, “他来到了” 词频优先
jieba.suggest_freq('杭研大厦', True)
jieba.suggest_freq('他来到了', True)
words = jieba.cut(text)
print(", ".join(words))
```

5.3. 返回 generator

默认 cut 返回 generator

```
# encoding=utf-8
import jieba
import paddle

segs = jieba.cut("转载请与作者联系，同时请务必标明文章原始出处和作者信息及本声明。")
print(type(segs))
print(", ".join(segs))
```

5.4. 返回 list

```
# encoding=utf-8
import jieba
import paddle

segs = jieba.lcut("转载请与作者联系，同时请务必标明文章原始出处和作者信息及本声明。")
print(type(segs))
print(", ".join(segs))
```

5.5. 精准模式与全模式比较

```
# encoding=utf-8
import jieba
import paddle
text = "转载请与作者联系，同时请务必标明文章原始出处和作者信息及本声明。"
segs = jieba.cut(text)  # 精准模式
print(", ".join(segs))
print("=" * 50)
segs = jieba.cut(text, cut_all=True)  # 全模式
print(", ".join(segs))
```

输出结果

```
neo@MacBook-Pro-Neo ~/workspace/python/jieba % python3.9
```

```
/Users/neo/workspace/python/jieba/lcut.py
Building prefix dict from the default dictionary ...
Loading model from cache
/var/folders/2f/jfnljdpn1t1dj_f61z2s8bwm0000gn/T/jieba.cache
Loading model cost 0.787 seconds.
Prefix dict has been built successfully.
转载，请，与，作者，联系，，，同时，请，务必，标明，文章，原始，出处，
和，作者，信息，及本，声明，。
==================================================
转载，请，与，作者，联系，，，同时，请，务必，标明，明文，文章，原始，
出处，和，作者，信息，及，本，声明，。
```

5.6. 精准模式与搜索引擎模式比较

```
# encoding=utf-8
import jieba
import paddle
text = "小明硕士毕业于中国科学院计算所，后在日本京都大学深造"
segs = jieba.cut(text)  # 精准模式
print(", ".join(segs))
print("=" * 50)
searchs = jieba.cut_for_search(text)  # 搜索引擎模式
print(", ".join(searchs))
```

输出结果

```
neo@MacBook-Pro-Neo ~/workspace/python/jieba % python3.9
/Users/neo/workspace/python/jieba/search.py
Building prefix dict from the default dictionary ...
Loading model from cache
/var/folders/2f/jfnljdpn1t1dj_f61z2s8bwm0000gn/T/jieba.cache
Loading model cost 0.807 seconds.
Prefix dict has been built successfully.
小明，硕士，毕业，于，中国科学院，计算所，，，后，在，日本京都大学，深造
```

```
==================================================
小明，硕士，毕业，于，中国，科学，学院，科学院，中国科学院，计算，计算
所，，，后，在，日本，京都，大学，日本京都大学，深造
```

5.7. 词性标注

标签	含义	标签	含义	标签	含义	标签	含义
n	普通名词	f	方位名词	s	处所名词	t	时间
nr	人名	ns	地名	nt	机构名	nw	作品名
nz	其他专名	v	普通动词	vd	动副词	vn	名动词
a	形容词	ad	副形词	an	名形词	d	副词
m	数量词	q	量词	r	代词	p	介词
c	连词	u	助词	xc	其他虚词	w	标点符号
PER	人名	LOC	地名	ORG	机构名	TIME	时间

```
import jieba
import jieba.posseg as pseg
import paddle
words = pseg.cut("我爱北京天安门")  # jieba默认模式
for word, flag in words:
    print('%s %s' % (word, flag))

print("="*40)

paddle.enable_static()
jieba.enable_paddle()  # 启动paddle模式。
words = pseg.cut("我爱北京天安门", use_paddle=True)  # paddle模式
for word, flag in words:
```

```
    print('%s %s' % (word, flag))
```

```
neo@MacBook-Pro-Neo ~/workspace/python % python3.9
/Users/neo/workspace/python/jieba/seg.py
Building prefix dict from the default dictionary ...
Loading model from cache
/var/folders/2f/jfnljdpn1t1dj_f61z2s8bwm0000gn/T/jieba.cache
Loading model cost 0.753 seconds.
Prefix dict has been built successfully.
我 r
爱 v
北京 ns
天安门 ns
==========================================
Paddle enabled successfully......
我 r
爱 v
北京 LOC
天安门 LOC
```

5.8. 词典管理

添加/删除词语

```
import jieba

text = "他来到了网易杭研大厦"

words = jieba.cut(text, HMM=False)
print(", ".join(words))
print("-" * 50)
jieba.add_word('杭研大厦')  # 将 “杭研大厦” 添加到词典
jieba.add_word('来到了')  # 将 “来到了” 添加到词典
```

```
words = jieba.cut(text, HMM=False)
print(", ".join(words))
print("-" * 50)
jieba.del_word('深圳')  # 将 “深圳” 从词典中删除
words = jieba.cut("我爱深圳", HMM=False)
print(", ".join(words))
```

用户词典

自定义词典

```
import jieba

jieba.load_userdict('dict.txt') # 载入用户词典
seg_list = jieba.cut("他来到了网易杭研大厦", HMM=False)
print(", ".join(seg_list))
```

自定义词库

分词系统默认使用自带的词库，load_userdict 是在默认词库的基础上做加法操作。

set_dictionary 是设置默认基础词库。

```
# encoding=utf-8
import jieba
jieba.set_dictionary('dict.txt')
seg_list = jieba.cut("转载请与作者联系，同时请务必标明文章原始出处和作者信息及本声明。")
print(seg_list)
```

```
print(", ".join(seg_list))
```

dict.txt

```
请务必 12 n
作者信息 10 n
```

5.9. 抽取文本标签

提取标签

```
方法参数:
jieba.analyse.extract_tags(sentence, topK=5, withWeight=True,
allowPOS=())
参数说明:
sentence 需要提取的字符串，必须是str类型，不能是list
topK 提取前多少个关键字
withWeight 是否返回每个关键词的权重
allowPOS是允许的提取的词性，默认为allowPOS='ns', 'n', 'vn', 'v'，提
取地名、名词、动名词、动词
```

```
file = open('article.txt', 'r', encoding='utf-8')
contents = file.read()
print(jieba.analyse.extract_tags(sentence=contents, topK=20,
allowPOS=('ns', 'n')))
```

基于 TextRank 算法的关键词抽取

```
import jieba.analyse
import jieba
import os

os.chdir('jieba')
file = open('article.txt', 'r', encoding='utf-8')
contents = file.read()
print(jieba.analyse.textrank(sentence=contents, topK=20,
allowPOS=('ns', 'n')))
```

5.10. 返回词语在原文的起止位置

```
import jieba
import logging
logger = logging.getLogger()
jieba.default_logger = logger
text = u'大和服装饰品有限公司'
result = jieba.tokenize(text)
for tk in result:
    print("word %s\t\t start: %d \t\t end:%d" % (tk[0],
tk[1], tk[2]))
print("-" * 50)
result = jieba.tokenize(text, mode='search')  # 搜索模式
for tk in result:
    print("word %s\t\t start: %d \t\t end:%d" % (tk[0],
tk[1], tk[2]))
```

6. wordcloud

https://github.com/amueller/word_cloud

```
pip install wordcloud
```

演示

```
import wordcloud

w = wordcloud.WordCloud()
w.generate(
    "Netkiller Neo Linux Nginx SSH Ubuntu CentOS MySQL
PostgreSQL Java Python")
w.to_file("wordcloud.png")
```

更多演示代码 https://amueller.github.io/word_cloud/auto_examples/index.html#example-gallery

6.1. wordcloud_cli

```
neo@MacBook-Pro-Neo ~ % wordcloud_cli -h
usage: wordcloud_cli [-h] [--text file] [--regexp regexp] [--
stopwords file] [--imagefile file]
                     [--fontfile path] [--mask file] [--
colormask file] [--contour_width width]
                     [--contour_color color] [--
relative_scaling rs] [--margin width]
```

```
                      [--width width] [--height height] [--
color color] [--background color]
                      [--no_collocations] [--include_numbers]
[--min_word_length min_word_length]
                      [--prefer_horizontal ratio] [--scale
scale] [--colormap map] [--mode mode]
                      [--max_words N] [--min_font_size size]
[--max_font_size size]
                      [--font_step step] [--random_state seed]
[--no_normalize_plurals] [--repeat]
                      [--version]

A simple command line interface for wordcloud module.

optional arguments:
  -h, --help            show this help message and exit
  --text file           specify file of words to build the
word cloud (default: stdin)
  --regexp regexp       override the regular expression
defining what constitutes a word
  --stopwords file      specify file of stopwords (containing
one word per line) to remove from
                        the given text after parsing
  --imagefile file      file the completed PNG image should
be written to (default: stdout)
  --fontfile path       path to font file you wish to use
(default: DroidSansMono)
  --mask file           mask to use for the image form
  --colormask file      color mask to use for image coloring
  --contour_width width
                        if greater than 0, draw mask contour
(default: 0)
  --contour_color color
                        use given color as mask contour color
- accepts any value from
                        PIL.ImageColor.getcolor
  --relative_scaling rs
                        scaling of words by frequency (0 - 1)
  --margin width        spacing to leave around words
  --width width         define output image width
  --height height       define output image height
  --color color         use given color as coloring for the
image - accepts any value from
                        PIL.ImageColor.getcolor
  --background color    use given color as background color
```

```
for the image - accepts any value from
                        PIL.ImageColor.getcolor
  --no_collocations     do not add collocations (bigrams) to
word cloud (default: add unigrams and
                        bigrams)
  --include_numbers     include numbers in wordcloud?
  --min_word_length min_word_length
                        only include words with more than X
letters
  --prefer_horizontal ratio
                        ratio of times to try horizontal
fitting as opposed to vertical
  --scale scale         scaling between computation and
drawing
  --colormap map        matplotlib colormap name
  --mode mode           use RGB or RGBA for transparent
background
  --max_words N         maximum number of words
  --min_font_size size  smallest font size to use
  --max_font_size size  maximum font size for the largest
word
  --font_step step      step size for the font
  --random_state seed   random seed
  --no_normalize_plurals
                        whether to remove trailing 's' from
words
  --repeat              whether to repeat words and phrases
  --version             show program's version number and
exit
```

6.2. WordCloud 对象配置参数

```
w = wordcloud.WordCloud(<参数>)

参数     描述
width           指定词云对象生成图片的宽度，默认400像素
height          指定词云对象生成图片的高度，默认200像素
min_font_size   指定词云中字体的最小字号，默认4号
max_font_size   指定词云中字体的最大字号，根据高度自动调节
```

```
font_step        指定词云中字体字号的步进间隔，默认为1
font_path        指定字体文件的路径，默认None
max_words        指定词云显示的最大单词数量，默认200
stop_words       指定词云的排除词列表，即不显示的单词列表
mask             指定词云形状，默认为长方形，需要引用imread()函数
background_color         指定词云图片的背景颜色，默认为黑色
```

```
import wordcloud

w = wordcloud.WordCloud(background_color="white")
w.generate(
    "Netkiller Neo Linux Nginx SSH Ubuntu CentOS MySQL
PostgreSQL Java Python")
w.to_file("wordcloud.png")
```

6.3. 与分词共用

```
# encoding=utf-8
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import jieba
import jieba.analyse

# background = plt.imread('background.jpg')  # 遮罩图
content = open('text.txt', 'r').read()  # 生成词云的文档

# seg_list = jieba.lcut(f)  # 默认是精确模式
tags = jieba.analyse.extract_tags(content, topK=50)
text = ", ".join(list(tags))
print(text)

wordcloud = WordCloud(
    background_color='white',  # 背景颜色，根据图片背景设置，默认为
```

```
黑色
    # mask = background, #笼罩图
    font_path='/Library/Fonts/AdobeSongStd-Light.otf',  # 若有
中文需要设置才会显示中文
    width=1024,  # 宽度
    height=768,  # 高度
    margin=5  # 边缘空白
).generate(text)

plt.imshow(wordcloud)
plt.axis('off')
plt.show()
plt.close()

# 保存图片
wordcloud.to_file('wordcloud.jpg')
```

6.4. 遮罩图

```
# encoding=utf-8
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from wordcloud import WordCloud
import jieba
import jieba.analyse

content = open('text.txt', 'r').read()
tags = jieba.analyse.extract_tags(content, topK=50)
text = ", ".join(list(tags))
# print(text)

# mask = np.array(Image.open("stormtrooper_mask.png"))
mask = np.array(Image.open("background.png"))

wordcloud = WordCloud(
    background_color='white',
    font_path='/Library/Fonts/AdobeSongStd-Light.otf',
```

```
    mask=mask,  # 遮罩图
    width=1024,  # 宽度
    height=768,  # 高度
    margin=5  # 边缘空白
).generate(text)
# 保存图片
wordcloud.to_file('wordcloud.jpg')

plt.imshow(wordcloud)
plt.axis('off')
plt.show()
plt.close()
```

7. Transformers 自然语言处理

8. 汉字转拼音

8.1.

```
from xpinyin import Pinyin

p = Pinyin()
o = p.get_pinyin("阅读圣经第五百零三节")
print(o)
```

8.2. pypinyin

```
from pypinyin import lazy_pinyin, load_phrases_dict,
load_single_dict, Style

hans = "打开圣经第一卷"

print(lazy_pinyin(hans))

load_single_dict({ord("零"): "零", ord("一"): "一", ord("二"):
"二", ord("三"): "三", ord("四"): "四", ord("五"): "五",
ord("六"): "六", ord("七"): "七", ord("八"): "八", ord("九"):
"九", ord("十"): "十", ord("百"): "百", ord("千"): "千",
ord("万"): "万", ord("亿"): "亿"})

load_phrases_dict(
    {
        "一卷": [["一"], ["juan"]],
        "三节": [["三"], ["jie"]],
        "一百": [["一"], ["百"]],
    }
)
```

```
print("=" * 50)
print(lazy_pinyin("播放圣经第一卷"))
print(lazy_pinyin("打开圣经第二卷"))
print(lazy_pinyin("打开圣经第一章"))
print(lazy_pinyin("打开圣经第一亿元"))
print(lazy_pinyin("阅读圣经第一百零六章"))
print(lazy_pinyin("阅读圣经第五百零六章"))
print(lazy_pinyin("阅读圣经第五百零五节"))
```

第 21 章 OpenAI

1. ChatGPT

1.1. gpt-3.5-turbo

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
##############################################
# Home  : https://www.netkiller.cn
# Author: Neo <netkiller@msn.com>
# Upgrade: 2023-06-25
##############################################
import os
import openai

# openai.api_key = os.getenv("OPENAI_API_KEY")
openai.api_key = "sk-
DNsMaVmxxIm3Xp7ZTZrZw2mcGgDF1nev5OT3BlbkFJ8wb3Y8"

completion = openai.ChatCompletion.create(
    model="gpt-3.5-turbo", messages=[{"role": "user",
"content": "谁是netkiller? "}]
)
print(openai.Model.list())
print(completion)
response = completion.choices[0].message.content
print(response)
```

1.2. 流式输出

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
##############################################
# Home  : https://www.netkiller.cn
# Author: Neo <netkiller@msn.com>
# Upgrade: 2023-10-07
##############################################
import openai

# openai.api_key = os.getenv("OPENAI_API_KEY")
openai.api_key = "sk-
UB5SdJmlbkFJnPC9GjYuY0sAEHzyuotulWBFgT3BQ70vTnKr"

question = "netkiller 写了那些电子书?"

response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[{"role": "user", "content": question}],
    stream=True,
)

print(f"问题: {question}")

for chunk in response:
    content = chunk["choices"][0].get("delta",
{}).get("content")
    if content is not None:
        print(content, end="")

print()
```

2. Embedding

```
import openai

openai.api_key = "sk-
UB5SdJmEHzykFJnPuotulWBFgT3BY0sAQ70vTnKrlbC9GjYu"

response = openai.Embedding.create(
    input="Your text string goes here",
    model="text-embedding-ada-002"
)
embeddings = response['data'][0]['embedding']
print(embeddings)
print(len(embeddings))
```

第 22 章 LangChain

1. 拆分文档

载入 README.md 文本

```
from langchain.document_loaders import TextLoader
loader = TextLoader("./README.md")
docs = loader.load()
print(docs)
```

2. 拆分文档

2.1. 拆分文本

```
from langchain.document_loaders import TextLoader

loader = TextLoader("./README.md")
docs = loader.load()
# print(docs)

from langchain.text_splitter import CharacterTextSplitter

text_splitter = CharacterTextSplitter(
    separator="\n\n",
    chunk_size=200,
    chunk_overlap=200,
    length_function=len,
)
split_docs = text_splitter.split_documents(docs)

for text in split_docs:
    print(text, end="\n\n")

# print(split_docs)
```

2.2. 拆分代码

```
from langchain.text_splitter import
RecursiveCharacterTextSplitter, Language

PYTHON_CODE = """
def hello_langchain():
    print("Hello, Langchain!")
```

```
# Call the function
hello_langchain()
"""
python_splitter =
RecursiveCharacterTextSplitter.from_language(language=Languag
e.PYTHON, chunk_size=50, chunk_overlap=0)
python_docs = python_splitter.create_documents([PYTHON_CODE])
print(python_docs)
```

2.3. 拆分 Markdown 文档

```
from langchain.text_splitter import
MarkdownHeaderTextSplitter
markdown_document = "# Chapter 1\n\n    ## Section 1\n\nHi
this is the 1st section\n\nWelcome\n\n ### Module 1 \n\n Hi
this is the first module \n\n ## Section 2\n\n Hi this is the
2nd section"
headers_to_split_on = [
("#", "Header 1"),
("##", "Header 2"),
("###", "Header 3"),
]
splitter =
MarkdownHeaderTextSplitter(headers_to_split_on=headers_to_spl
it_on)
splits = splitter.split_text(markdown_document)
print(splits)
```

2.4. 按token拆分

例如 OpenAI 的 token 有字数限制。在API调用时不应超过 token 限制，使用 from_tiktoken_encoder 可以解决这个问题。

```
```

```
import openai

openai.api_key = "sk-
UB5STnKrdJmEHT3BlbkFJnPC9GjYuY0sAQzyuotulWBFg70v"

from langchain.document_loaders import TextLoader

loader = TextLoader("./README.md")
docs = loader.load()
# print(docs)
from langchain.text_splitter import CharacterTextSplitter
text_splitter = CharacterTextSplitter.from_tiktoken_encoder(
chunk_size=100, chunk_overlap=0
)
split_docs = text_splitter.split_documents(docs)
print(split_docs)
```

3. ChatGPT

```
pip install openai gradio langchain llama-index
```

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple openai
gradio langchain llama-index
```

4. 相似度搜索

```
import openai

openai.api_key = "sk-
UB5SdJFgT3BlbkFJnzyuotulWB0sAQ70vTnKrPC9GjYuYmEH"

from langchain.document_loaders import TextLoader

loader = TextLoader("./README.md")
docs = loader.load()

from langchain.document_loaders import TextLoader
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma
text_splitter=CharacterTextSplitter(chunk_size=1000,
chunk_overlap=0)
documents = text_splitter.split_documents(docs)
db = Chroma.from_documents(documents, OpenAIEmbeddings())

query = "谁是 netkiller? "
docs = db.similarity_search(query)
print(docs[0].page_content)
```

第 23 章 自动化运维

1. 日志中心

1.1. 什么是日志中心

日志中心是针对日志类数据一站式管理，日志中心分成两部分，分别是采集端和收集端。

采集器从各服务器采集数据，然后发送给接收端，接收端接到数据后，将日志保存到磁盘上。用户能在日志中心服务器上查询和分析日志，帮助运维、运营提升效率。

当年我开发此程序的时候还没有出现如今流行的ELK方案，接下来就让我为大家介绍如何使用 Python 开发一个日志采集和接受程序。

1.2. 工作原理

日志采集端：有两种工作方式：一种是文件采集，另一种是标准输出采集。文件采集又分为，静态日志文件采集，将整个文件给接收端。动态文件采集，仅采集添加到日志文件尾部的更新内容，即每次只读取日志追加的内容。

日志接收端： 启动后进入后台，开启UDP端口，等候采集端发送内容。

1.3. 安装

pip 安装

```
[root@localhost ~]# pip3 install netkiller-logging
```

源码安装

```
[root@localhost ~]# git clone
https://github.com/netkiller/logging.git
[root@localhost ~]# cd logging
[root@localhost ~]# python3 setup.py sdist
[root@localhost ~]# python3 setup.py install
```

1.4. 命令

日志采集端

日志采集端名师是 rlog

```
[root@localhost ~]# whereis rlog
rlog: /usr/local/bin/rlog

[root@localhost ~]# rlog
Usage: rlog [options] filename

Options:
-h, --help            show this help message and exit
-H localhost, --host=localhost
                                                  push log to
remote host
-p 1214, --port=1214  port
--sleep=0.05          with -s, sleep  for  approximately S
seconds between
                                                  iterations
-d, --daemon          run as daemon
-f, --full            Full text
--stdin               cat file | prog ...
-e /tmp/rlog.log, --errlog=/tmp/rlog.log
```

```
                                                        error log
--postion             save postion of log file

Homepage: http://netkiller.github.io    Author: Neo
<netkiller@msn.com>
```

日志接收端

日志接收端命令式 collection

```
[root@localhost ~]# collection
Usage: collection [options] module

Options:
  -h, --help            show this help message and exit
  -p 1214, --port=1214  port
  -l /tmp/test.log, --logfile=/tmp/test.log
                                                        log file
  --list                show module message
  -d, --daemon          run as daemon

  Homepage: http://netkiller.github.io  Author: Neo
<netkiller@msn.com>
```

1.5. 操作演示

从标准输出采集日志

日志接收端命令:

```
collection -p 1214
```

启动日志接收程序，使用UDP 1214 端口接收日志。

日志采集端命令：

```
echo Helloworld | rlog -H 127.0.0.1 —stdin
```

运行日志采集程序，将标准输出内容发送到日志接收端。

发送日志文件

将日志文件发送给日志接收端

日志接收端命令：

```
collection -p 1214
```

这里仅仅演示，所以我们在终端输出日志。

日志采集端命令：

```
rlog -H 127.0.0.1 -f /var/log/your.log
```

参数 -f 是指发送整个文件，否则只会发送采集程序启动后，追加的日志。

接收日志并保存到文件

日志接收端命令:

```
collection -p 1214 -l /tmp/test.log
```

参数 -l 表示将接收到的日志写入文件，使用 -l 参数后，终端屏幕不再输出内容

日志采集端命令:

```
rlog -H 127.0.0.1 -f /var/log/your.log
```

参数 -f 是指发送整个文件，否则只会发送采集程序启动后，追加的日志。

发送动态日志文件

所谓动态日志文件是指，正在运行的程序随时向日志追加内容，例如WEB服务器nginx

日志接收端命令:

```
collection -d -p 1214 -l /tmp/test.log
```

参数 -d 表示后台运行，-l 将接收到的日志写入文件

日志采集端命令:

```
rlog -d -H 127.0.0.1 /var/log/your.log
```

参数 -d 表示rlog启动后在后台运行，注意不要使用 -f 参数

2. Python 开发防火墙

2.1. 我为什么要开发一个防火墙软件?

起因是这样的，10年前，我们的IDC部署大量的CentOS，当时是6.x版本，后面升级到7.x版本。对于新版CentOS 大家都不太适应它携带的 firewalld 防火墙，另一个原因是IDC的服务器几乎只做 INPUT（进入）规则，极少服务器有OUTPUT（访问）和FORWARD（转发）规则。firewalld 更适合做企业防火墙，并不适合做IDC防火墙，把简单的事情复杂化了。

使用下面命令查看区域设置:

```
[root@localhost ~]# firewall-cmd --get-zones
block dmz drop external home internal nm-shared public trusted
work
```

当我尝试删除的时候失败，于是我便使用Python开发了一款防火墙，用来替代 firewalld。

```
[root@localhost ~]# for zone in $(firewall-cmd --get-zones);do
firewall-cmd --delete-zone=$zone --permanent; done
Error: BUILTIN_ZONE: 'block' is built-in zone
Error: BUILTIN_ZONE: 'dmz' is built-in zone
Error: BUILTIN_ZONE: 'drop' is built-in zone
Error: BUILTIN_ZONE: 'external' is built-in zone
```

防火墙设计与想法

面向哪些用户：运维人员和开发人员。

运维人员：运维人员可以像使用其他软件一样使用这款防火墙软件。容易安装，容易学习，容易使用，解决运维面临的痛点。

开发人员：能通过开发包，做防火墙的二次开发。

2.2. 安装 Python 防火墙

准备一台 CentOS 8 (CentOS Stream)，在 root 用户下运行下面命令安装

pip 安装

```
[root@localhost firewall]# pip3 install netkiller-firewall
WARNING: Running pip install with root privileges is generally
not a good idea. Try `pip3 install --user` instead.
Collecting netkiller-firewall
  Using cached
https://files.pythonhosted.org/packages/d2/82/f0d7cc6646447e256
0702415606b9aa668b0dc7536e24944a2d0823db7ff/netkiller_firewall-
0.0.1-py3-none-any.whl
Installing collected packages: netkiller-firewall
Successfully installed netkiller-firewall-0.0.1
[root@localhost firewall]#
```

源码安装

```
[root@localhost ~]# cd /usr/local/src/
[root@localhost ~]# dnf install -y git python38
[root@localhost ~]# git clone
https://github.com/netkiller/firewall.git
[root@localhost ~]# cd firewall
[root@localhost ~]# bash install.sh
```

如何使用该防火墙

```
启用防火墙:
systemctl enable firewall

启动防火墙:
systemctl start firewall

停止防火墙:
systemctl stop firewall

查看iptables规则:
iptables -L
```

2.3. 切换防火墙规则

修改 /etc/sysconfig/firewall 文件设置/切换默认规则

```
[root@localhost ~]# vim /etc/sysconfig/firewall
##############################################
LIBEXEC=/srv/firewall/libexec
RULE=www
##############################################
```

LIBEXEC 是规则库的位置

RULE 是使用规则库 LIBEXEC 中的那个规则

2.4. 规则库

规则库的目录是 /srv/firewall/libexec/ 默认提供几个常用的规则：Web 服务器，邮件服务器，数据库服务器。

```
[root@localhost ~]# cd /srv/firewall/libexec/
[root@localhost ~]# ls
db.py   smtp.py www.py
```

默认防火墙规则 /srv/firewall/libexec/www.py

```
        $ sudo cat /srv/firewall/libexec/www.py
        #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        from firewall import *

        ########################################
        # Web Application
        ########################################

        www = Firewall()
        www.flush()
        www.policy(www.INPUT,www.ACCEPT)
        www.policy(www.OUTPUT,www.ACCEPT)
        www.policy(www.FORWARD,www.ACCEPT)
        www.input().state(('RELATED','ESTABLISHED')).accept()
        www.input().protocol('icmp').accept()
        www.input().interface('-i','lo').accept()

www.input().protocol('tcp').dport('22').state('NEW').accept()

www.input().protocol('tcp').dport(('443','80')).state('NEW').ac
cept()

www.output().protocol('tcp').dport(('20','21')).reject()


#www.input().protocol('tcp').inbound('eth0').dport('80').recent
('HTTP',2,20).drop()

#www.input().protocol('tcp').inbound('eth0').dport('80').connli
mit(30).drop()
```

```
#www.input().protocol('tcp').inbound('eth0').dport('80').recent
('HTTP').accept()
        # DDOS

#www.input().proto('tcp').dport("80").string('XXDD0S').drop()
        www.input().reject('--reject-with icmp-host-
prohibited')
        www.forward().reject('--reject-with icmp-host-
prohibited')

        def start():
                www.start()
        def stop():
                www.stop()
        def restart():
                www.stop()
                www.start()
        def show():
                www.show()
        def status():
                www.status()
        def main():
                show()
                return( 0 )

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

3. 监视文件系统

3.1. watchdog

watchdog 提供了指定目录/文件的变化监控，对于指定目录内的操作，被视为一次事件。如添加删除文件或目录、重命名文件或目录、修改文件内容等，每种变化都会触发一次事件，事件是用户定义的业务逻辑代码。

安装

```
pip install watchdog
```

演示程序

```
import sys
import time
import logging
from watchdog.observers import Observer
from watchdog.events import LoggingEventHandler

if __name__ == "__main__":

    path = r'/tmp'

    logging.basicConfig(level=logging.INFO,
                        format='%(asctime)s - %(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S')
    event_handler = LoggingEventHandler()
    observer = Observer()
    observer.schedule(event_handler, path, recursive=True)
    observer.start()
    try:
```

```
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

Observer

```
# 设置超时时间
watchdog.observers.Observer(timeout=30)

# 监控指定路径path，该路径触发任何事件都会调用event_handler来处理，如果path是目录，recursive=True 开启会递归模式，监控该目录下的所有变化。
observer.schedule(event_handler, path, recursive=False)

# 添加一个事件处理器到watch中
observer.add_handler_for_watch(event_handler, watch)

# 从watch中移除一个事件处理器
observer.remove_handler_for_watch(event_handler, watch)

# 移除一个watch及这个watch上的所有事件处理器
observer.unschedule(watch)

# 移除所有watch及关联的事件处理器
observer.unschedule_all()
# 等同于observer.unschedule_all()
observer.on_thread_stop()

# 启动 observer 线程
observer.start()

# 停止observer线程
observer.stop()
```

Watchdog 几种 Observer 类型:

- InotifyObserver，Linux系统默认使用的观察目录的调度事件，效率比较高。
- PollingObserver，与平台无关，轮询目录以检测文件的更改，效率比较低。
- WindowsApiObserver，Windows系统默认使用的观察目录的调度事件，效率比较高。
- FSEventsObserver，macOS 系统默认使用的调度事件
- KqueueObserver，FreeBSD 系统默认使用

默认 Observer 会判断操作系统类型，选择最佳的方式。下面👇是 Observer 的源码

```
# Linux系统
if platform.is_linux():
    try:
        from .inotify import InotifyObserver as Observer
    except UnsupportedLibc:
        from .polling import PollingObserver as Observer
# darwin系统
elif platform.is_darwin():
    # FIXME: catching too broad. Error prone
    try:
        from .fsevents import FSEventsObserver as Observer
    except:
        try:
            from .kqueue import KqueueObserver as Observer
            warnings.warn("Failed to import fsevents. Fall
back to kqueue")
        except:
            from .polling import PollingObserver as Observer
            warnings.warn("Failed to import fsevents and
kqueue. Fall back to polling.")
# bsd系统
elif platform.is_bsd():
    from .kqueue import KqueueObserver as Observer
# windows系统
elif platform.is_windows():
    # TODO: find a reliable way of checking Windows version
and import
```

```
    # polling explicitly for Windows XP
    try:
        from .read_directory_changes import
WindowsApiObserver as Observer
    except:
        from .polling import PollingObserver as Observer
        warnings.warn("Failed to import
read_directory_changes. Fall back to polling.")

else:
    from .polling import PollingObserver as Observer
```

手工选择 Observer，注意下面代码 observer = PollingObserver()

```
import sys
import time
import logging
# from watchdog.observers import Observer
from watchdog.observers.polling import PollingObserver
from watchdog.events import LoggingEventHandler

if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO,
                        format='%(asctime)s - %(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S')
    path = sys.argv[1] if len(sys.argv) > 1 else '.'
    event_handler = LoggingEventHandler()
    observer = PollingObserver()
    observer.schedule(event_handler, path, recursive=True)
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

创建/删除/修改/移动

定义创建/删除/修改/移动四种事件

```
from watchdog.observers import Observer
from watchdog.events import *
import time


class DemoFileSystemEventHandler(FileSystemEventHandler):

    def __init__(self):
        FileSystemEventHandler.__init__(self)

    def on_moved(self, event):
        if event.is_directory:
            print("directory moved from {0} to {1}".format(
                event.src_path, event.dest_path))
        else:
            print("file moved from {0} to {1}".format(
                event.src_path, event.dest_path))

    def on_created(self, event):
        if event.is_directory:
            print("directory created:
{0}".format(event.src_path))
        else:
            print("file created:{0}".format(event.src_path))

    def on_deleted(self, event):
        if event.is_directory:
            print("directory deleted:
{0}".format(event.src_path))
        else:
            print("file deleted:{0}".format(event.src_path))

    def on_modified(self, event):
        if event.is_directory:
            print("directory modified:
{0}".format(event.src_path))
        else:
```

```
            print("file modified:{0}".format(event.src_path))


if __name__ == "__main__":

    path = r'/tmp'

    observer = Observer()
    event_handler = DemoFileSystemEventHandler()
    observer.schedule(event_handler, path, recursive=True)
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

on_any_event 任何事件都会触发

```
import time
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler


class MyHandler(FileSystemEventHandler):
    def on_any_event(self, event):
        if event.is_directory:
            print('Directory', event.event_type,
event.src_path)
        else:
            print('File', event.event_type, event.src_path)


if __name__ == "__main__":

    monitor_path = '/tmp'
```

```
    event_handler = MyHandler()
    observer = Observer()
    observer.schedule(event_handler, path=monitor_path,
recursive=True)
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

多事件绑定

```
import time
import logging
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
from watchdog.events import LoggingEventHandler


class MyEventHandler(FileSystemEventHandler):
    def on_any_event(self, event):
        if event.is_directory:
            print('Directory', event.event_type,
event.src_path)
        else:
            print('File', event.event_type, event.src_path)


if __name__ == "__main__":

    watch_path = '/tmp'

    logging.basicConfig(level=logging.INFO,
                        format='%(asctime)s - %(message)s',
                        datefmt='%Y-%m-%d %H:%M:%S')
    loggingEventHandler = LoggingEventHandler()
```

```
    myEventHandler = MyEventHandler()

    observer = Observer()

    watch1 = observer.schedule(
        loggingEventHandler, path=watch_path, recursive=True)
    watch2 = observer.schedule(
        myEventHandler, path=watch_path, recursive=True)

    observer.add_handler_for_watch(loggingEventHandler,
watch1)
    observer.add_handler_for_watch(myEventHandler, watch2)

    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

自动备份程序

事情的 起因 - 经过 - 结果

做视频剪辑有一段时间了，苹果笔记本电脑硬盘比较小，视频剪辑空间不够，所以必须外挂一个剪辑盘。

我使用了由两块 SSD 组成的 RAID 硬盘盒子，起初是追求速度快，使用 RAID 0 模式，后期为了安全 切换到了 RAID 1 模式。这样硬件的安全就解决了，再也不用担心硬盘损坏数据丢失了。

某一天，就如日常剪辑一样，当我试图打开 Final cut pro 资源库的时候，悲剧了！！！

Fcpx 的工程文件打不开了，闪退后提示错误，这次翻车丢失了《Netkiller Architect 手札》6个视频，不过还好，那些视频都是初期做

品，录制和剪辑都不算好，本就有重做的想法。

这次经历让我不在相信 fcpx 的资源库管理能力，鸡蛋放在一个篮子里及其危险的，《Netkiller Python 手札》我把一章内容放在一个资源库中，每节一个事件。

之前也偶尔备份，手工复制，有时比较懒，就没有备份。于是我便写了一个程序，来自动备份视频剪辑盘。

实现功能：

- 插入U设备，发现移动硬盘
- 剪辑盘和备份盘没有其他程序占用
- 开始备份
- 备份完成后自动弹出移动设备

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
#
==================================================================
====================
# 视频剪辑盘自动备份程序
# 作者: netkiller | netkiller@msn.com |
http://www.netkiller.cn
#
==================================================================
====================

import time
import os
import signal
import sys
import atexit
import subprocess
# from atexit import register
from watchdog.observers import Observer
from watchdog.events import *
```

```
class FileEventHandler(FileSystemEventHandler):
    src_path = ''
    dst_path = ''
    watchpath = {}

    def __init__(self, logger, watchpath):
        FileSystemEventHandler.__init__(self)
        self.watchpath = watchpath
        self.logger = logger

    def execute(self, cmd):
        fr = os.popen(cmd)
        # text = fr.read()
        text = fr.readlines()
        fr.close()
        # print(len(text), text)
        return(text)

    def isBusy(self, path):
        cmd = "lsof {}".format(path)
        # print(cmd)
        if len(self.execute(cmd)) == 0:
            self.logger.info("目录空闲: {0}".format(path))
            return False
        else:
            self.logger.info("目录繁忙: {0}".format(path))
            return True

    def mirror(self):
        self.logger.info("开始备份")
        cmd = "/usr/bin/rsync -auzv --delete --
exclude=.Spotlight-V100 --exclude=.fseventsd {0} {1}".format(
            self.src_path, self.dst_path)
        # self.logger.info(cmd)

        sp = subprocess.Popen(cmd, shell=True,
stdout=subprocess.PIPE)
        for line in iter(sp.stdout.readline, b''):

self.logger.info(line.decode('UTF8').replace("\n", ''))
        sp.stdout.close()
        sp.wait()
        self.logger.info("备份完成")

    def umount(self):
```

```
        if not self.isBusy(self.src_path):
            self.logger.info("卸载移动硬盘")
            os.system("diskutil eject {0}".format(
                os.path.basename(self.src_path)))
        else:
            self.logger.info("移动硬盘 {0} 繁
忙".format(self.src_path))

    def backup(self):
        if not os.path.exists(self.dst_path):
            os.mkdir(self.dst_path)
        if not self.isBusy(self.src_path):
            if not self.isBusy(self.dst_path):
                self.mirror()
                time.sleep(5)
                self.umount()

    def on_created(self, event):
        if event.is_directory:
            self.logger.info("发现移动硬盘:
{0}".format(event.src_path))
            # self.path = self.watchpath[event.src_path]
            self.src_path = event.src_path
            self.dst_path = self.watchpath[event.src_path]
            self.backup()

    def on_deleted(self, event):
        if event.is_directory:
            self.logger.info("卸载移动硬盘:
{0}".format(event.src_path))


class Backup():

    pidfile = "/tmp/netkiller.pid"
    logdir = "/tmp"
    logfile = logdir+"/netkiller.log"
    logger = None
    loop = True
    job = True

    watchpath = {'/Volumes/Video': '/tmp/Backup',
                 '/Volumes/Photo': '/tmp/Backup'}

    observer = Observer()
```

```
    def __init__(self):
        logging.basicConfig(level=logging.NOTSET,
                            format='%(asctime)s %
(levelname)-8s %(message)s',
                            datefmt='%Y-%m-%d %H:%M:%S',
                            filename=self.logfile,
                            filemode='a')
        self.logger = logging.getLogger()

    # 保存进程ID
    def savepid(self, pid):
        with open(self.pidfile, 'w') as f:
            f.write(str(os.getpid()))
        # 注册退出函数，进程退出时自动移除pidfile文件
        atexit.register(os.remove, self.pidfile)

    # 从pidfile中读取进程ID
    def getpid(self):
        pid = 0
        try:
            with open(self.pidfile, 'r') as f:
                pid = int(f.readline())
        except FileNotFoundError as identifier:
            print(identifier)
        return pid

    # 信号处理
    def signalHandler(self, signum, frame):
        # global loop, job
        # print("SIGN ",str(signum));
        if signum == signal.SIGHUP:
            # 优雅重启
            self.job = False
            self.logger.info("优雅重启完毕")
        elif signum == signal.SIGINT:
            # 正常退出
            self.loop = False
            self.job = False
            self.observer.stop()
            self.logger.info("正常退出")
        elif signum == signal.SIGUSR1:
            # 日志切割
            pass
```

```
    def daemonize(self):
        # global job
        self.logger.info("启动守护进程")
        signal.signal(signal.SIGHUP, self.signalHandler)
        signal.signal(signal.SIGINT, self.signalHandler)
        signal.signal(signal.SIGUSR1, self.signalHandler)
        signal.alarm(5)

        pid = os.fork()
        sys.stdout.flush()
        sys.stderr.flush()
        if pid:
            sys.exit(0)
        # print(os.getpid())
        self.savepid(str(os.getpid()))
        while self.loop:
            self.logger.info("Start!!!")
            self.main()
            self.logger.info("Exit!!!")

    def start(self):
        if os.path.isfile(self.pidfile):
            print("程序已经启动")
            sys.exit(1)
        else:
            self.daemonize()

    def stop(self):
        try:
            os.kill(self.getpid(), signal.SIGINT)
        except ProcessLookupError as identifier:
            print(identifier)

    def reloads(self):
        try:
            os.kill(self.getpid(), signal.SIGHUP)
        except ProcessLookupError as identifier:
            print(identifier)

    def logrotate(self):
        try:
            os.kill(self.getpid(), signal.SIGUSR1)
        except ProcessLookupError as identifier:
            print(identifier)
```

```
    def main(self):
        # 业务逻辑
        event_handler = FileEventHandler(self.logger,
self.watchpath)
        for src, dst in self.watchpath.items():
            self.logger.info("监控 {0} => {1}".format(src,
dst))
            self.observer.schedule(
                event_handler, path=src, recursive=False)
        self.logger.info("启动 Watchdog")
        self.observer.start()
        self.observer.join()
        # 业务逻辑


def usage():
    print(sys.argv[0] + " start | stop | restart | reload |
log")


if __name__ == "__main__":
    # print(sys.argv)
    backup = Backup()
    if len(sys.argv) > 1:
        arg = sys.argv[1]
        if arg == "start":
            backup.start()
        elif arg == "stop":
            backup.stop()
        elif arg == "restart":
            backup.stop()
            backup.start()
        elif arg == "reload":
            backup.reloads()
        else:
            usage()
    else:
        usage()
```

3.2. pyinotify

4. 容器

Flask 是一个轻量级Web应用框架，简单易用，可以很快速地创建web应用。我们用它来创建一个demo应用。

如果还没有安装Flask库，可以使用下面命令安装:

```
$ pip install flask
```

创建 flask 项目

安装成功后，新建一个命名为flask的目录

```
cd workspace/python
mkdir flask
cd flask
```

然后在该目录下创建app.py文件。

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
  return """
  <h1>《Netkiller Python 手札》</h1>
  <p>这是一个运行在 Docker 容器中的演示程序.</p>
  """
```

```
if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0')
```

运行项目

```
➜  flask git:(master) ✗ python3 app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a
production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a
production deployment.
 * Running on http://192.168.3.2:5000/ (Press CTRL+C to quit)
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 211-304-394
```

然后在浏览器中访问 http://localhost:5000/ 确认 flask 可以正常运行。

```
➜  python git:(master) ✗ curl http://localhost:5000

  <h1>《Netkiller Python 手札》</h1>
  <p>这是一个运行在 Docker 容器中的演示程序.</p>
```

4.1. 在 Docker 容器中运行 Python 项目

要在Docker上运行应用程序，首先必须使用 Dockerfile 脚本构建一个容器，而且必须包含使用的所有依赖项，包括 python 和 依赖库。

新建一个 requirements.txt 文件，包含所有依赖的 python 包，我们的例子中只有用到了一个 Flask。

```
flask==2.0.1
```

创建 Dockerfile 文件用来构建映像

```
FROM python

LABEL org.opencontainers.image.authors="netkiller@msn.com"

WORKDIR /

COPY ./requirements.txt /requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

COPY . /

ENTRYPOINT [ "python" ]

CMD [ "app.py" ]
```

构建镜像

```
docker build -t flask:0.0.1 .
```

启动容器

```
docker run --name flask -p 5000:5000 flask:0.0.1
```

容器启动运行后，测试无误，上传docker镜像到仓库

```
➜  flask git:(master) docker login --username netkiller
Password:
Login Succeeded
```

```
➜  flask git:(master) docker tag flask:0.0.1
netkiller/flask:latest
```

```
➜  flask git:(master) docker push netkiller/flask:latest
The push refers to repository [docker.io/netkiller/flask]
3591098640e3: Pushed
86c12f640351: Pushed
f4559c1df7ec: Pushed
cd6b2a9ae627: Pushed
84c97f2e3099: Pushed
b0cb6a43f300: Pushed
4b4c002ee6ca: Pushed
cdc9dae211b4: Pushed
7095af798ace: Pushed
fe6a4fdbedc0: Pushed
e4d0e810d54a: Pushed
4e006334a6fd: Pushed
latest: digest:
sha256:c3be7315046aa8abe6851475658ea1b23ab1b44d411b0a5be650f38d
2b197bc3 size: 2842
```

4.2. 在 kubernetes 中部署项目

安装 kubernetes 测试环境

```
$ brew install minikube
$ brew install kubectl
```

启动 minikube

```
minikube start --memory 2048mb --cpus 2 --cache-images=true --
driver=hyperkit \
--image-mirror-country=cn --registry-
mirror="https://registry.docker-
cn.com,https://docker.mirrors.ustc.edu.cn" --insecure-
registry="127.0.0.1:5000,192.168.3.0/24,192.168.64.0/24,172.17.
0.0/16,10.10.0.0/24"
```

创建密钥

```
kubectl create secret docker-registry docker-hub \
--docker-server=https://index.docker.io/v2/ \
--docker-username=netkiller \
```

```
--docker-password=passw0rd \
--docker-email=netkiller@msn.com
```

查看密钥，确保 docker-hub 被创建

```
→  Python git:(master) ✗ kubectl get secret
NAME                  TYPE
DATA   AGE
default-token-lcmzw   kubernetes.io/service-account-token   3
2m24s
docker-hub            kubernetes.io/dockerconfigjson        1
35s
```

flask.yaml 文件

```
apiVersion: v1
kind: Service
metadata:
  name: flask
  namespace: default
  labels:
    app: flask
spec:
  type: NodePort
  ports:
    - port: 5000
      nodePort: 31000
  selector:
    app: flask
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask
```

```
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask
  template:
    metadata:
      labels:
        app: flask
    spec:
      containers:
        - name: flask
          image: netkiller/flask:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5000
      imagePullSecrets:
        - name: docker-hub
```

部署

```
➜  kubernetes git:(master) ✗ kubectl create -f flask.yaml
service/flask created
deployment.apps/flask created
➜  kubernetes git:(master) ✗ minikube service list
|-------------|------------|--------------|--------------------
-------|
|  NAMESPACE  |    NAME    | TARGET PORT  |            URL
|
|-------------|------------|--------------|--------------------
-------|
| default     | flask      |         5000 |
http://192.168.64.9:31000 |
| default     | kubernetes | No node port |
| kube-system | kube-dns   | No node port |
|-------------|------------|--------------|--------------------
-------|
```

测试

```
➜  kubernetes git:(master) ✗ curl http://192.168.64.9:31000

  <h1>《Netkiller Python 手札》</h1>
  <p>这是一个运行在 Docker 容器中的演示程序.</p>
```

销毁

```
➜  kubernetes git:(master) ✗ kubectl delete -f flask.yaml
service "flask" deleted
deployment.apps "flask" deleted
```

第 24 章 办公自动化

1. Python 处理 PDF 文件

1.1. Word 转 PDF

安装

```
pip install docx2pdf
```

macOS 还需要安装

```
brew install aljohri/-/docx2pdf
```

命令行演示

查看帮助信息 docx2pdf --help

```
neo@MacBook-Pro-Neo ~ % docx2pdf
usage: docx2pdf [-h] [--keep-active] [--version] input [output]

Example Usage:

Convert single docx file in-place from myfile.docx to
myfile.pdf:
    docx2pdf myfile.docx

Batch convert docx folder in-place. Output PDFs will go in the
```

```
same folder:
    docx2pdf myfolder/

Convert single docx file with explicit output filepath:
    docx2pdf input.docx output.docx

Convert single docx file and output to a different explicit
folder:
    docx2pdf input.docx output_dir/

Batch convert docx folder. Output PDFs will go to a different
explicit folder:
    docx2pdf input_dir/ output_dir/

positional arguments:
  input           input file or folder. batch converts entire
folder or convert single file
  output          output file or folder

optional arguments:
  -h, --help      show this help message and exit
  --keep-active   prevent closing word after conversion
  --version       display version and exit
```

代码演示

```
from docx2pdf import convert

convert("input.docx")
convert("input.docx", "output.pdf")
convert("my_docx_folder/")
```

如果只是转换一个文档，我们就没有必要用Python了。下面的程序是批量转换指定目录下的 Word 文档。

```
import os
import glob
from pathlib import Path

# 指定转换目录
path = os.getcwd() + '/'

p = Path(path)
files=list(p.glob("**/*.docx"))

for file in files:
    convert(file,f"{file}.pdf")
```

1.2. 提取 PDF 文件中的文字和表格

Plumb a PDF for detailed information about each char, rectangle, and line.

安装 pdfplumber

```
neo@MacBook-Pro-Neo ~/workspace/python % pip install pdfplumber
```

查看 pdfplumber 是否安装成功

```
neo@MacBook-Pro-Neo ~/workspace/python % pip show pdfplumber
Name: pdfplumber
Version: 0.5.26
Summary: Plumb a PDF for detailed information about each char,
rectangle, and line.
Home-page: https://github.com/jsvine/pdfplumber
Author: Jeremy Singer-Vine
```

```
Author-email: jsvine@gmail.com
License: UNKNOWN
Location:
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6
/site-packages
Requires: pdfminer.six, Pillow, Wand
Required-by:
```

获取PDF文档信息

```
import os,pdfplumber
import pandas as pd

file = os.path.expanduser("~/tmp/每日开放式基金净值表.pdf")

with pdfplumber.open(file) as pdf:
    print(pdf.metadata)
```

输出结果

```
{'Producer': 'macOS 版本11.2.1（版号20D74） Quartz PDFContext',
'CreationDate': "D:20210227145013Z00'00'", 'ModDate':
"D:20210227145013Z00'00'"}
```

获取PDF总页数

```
import os,pdfplumber
import pandas as pd
```

```
file = os.path.expanduser("~/tmp/每日开放式基金净值表.pdf")

with pdfplumber.open(file) as pdf:
    print(len(pdf.pages))
```

查看PDF页面信息

```
import os,pdfplumber
import pandas as pd

file = os.path.expanduser("~/tmp/每日开放式基金净值表.pdf")

with pdfplumber.open(file) as pdf:

    first_page = pdf.pages[0]

    # 查看当前页码
    print('页码：',first_page.page_number)

    # 查看当前页宽
    print('页宽：',first_page.width)

    # 查看当前页高
    print('页高：',first_page.height)
```

输出结果

```
neo@MacBook-Pro-Neo ~/workspace/python % python3 pdf.py
页码：  1
页宽：  1324
页高：  7638
```

提取文本内容

```
import os,pdfplumber
import pandas as pd

file = os.path.expanduser("~/tmp/每日开放式基金净值表.pdf")

with pdfplumber.open(file) as pdf:

    first_page = pdf.pages[0]

    # 读取文本
    text = first_page.extract_text()
    print(text)
```

提取pdf中的表格数据

```
import os,pdfplumber
import pandas as pd

file = os.path.expanduser("~/tmp/每日开放式基金净值表.pdf")

with pdfplumber.open(file) as pdf:
    first_page = pdf.pages[0]
    table = first_page.extract_table()
    print(table)
```

```
import os,pdfplumber
import pandas as pd
```

```
file = os.path.expanduser("~/tmp/每日开放式基金净值表.pdf")

with pdfplumber.open(file) as pdf:

    first_page = pdf.pages[0]
    table = first_page.extract_table()
    # print(table)
    for t in table:
        print(t)
```

```
neo@MacBook-Pro-Neo ~/workspace/python % python3 pdf.py
['关注', '比较', '序号', '基金代码', '基金简称', '2021-02-26', None,
'2021-02-25', None, '日增长值', '日增长率', '申购状态', '赎回状态',
'手续费']
[None, None, None, None, None, '单位净值', '累计净值', '单位净值',
'累计净值', None, None, None, None, None]
['', '', '1', '501030', '汇添富中证环境治理指数A 估值图 基金吧',
'0.5501', '0.5501', '0.5421', '0.5421', '0.0080', '1.48%', '开
放', '开放', '0.12%']
['', '', '2', '501031', '汇添富中证环境治理指数C 估值图 基金吧',
'0.5471', '0.5471', '0.5392', '0.5392', '0.0079', '1.47%', '开
放', '开放', '0.00%']
['', '', '3', '164908', '交银中证环境治理(LOF) 估值图 基金吧',
'0.4890', '0.4890', '0.4820', '0.4820', '0.0070', '1.45%', '开
放', '开放', '0.12%']
['', '', '4', '004005', '东方民丰回报赢安混合A 估值图 基金吧',
'1.0564', '1.0709', '1.0438', '1.0583', '0.0126', '1.21%', '开
放', '开放', '0.06%']
['', '', '5', '004006', '东方民丰回报赢安混合C 估值图 基金吧',
'1.0463', '1.0593', '1.0338', '1.0468', '0.0125', '1.21%', '开
放', '开放', '0.00%']
```

保存数据到 Excel

```
neo@MacBook-Pro-Neo ~/workspace/python % pip install openpyxl
```

```
import os,pdfplumber

import pandas as pd

file = os.path.expanduser("~/tmp/每日开放式基金净值表.pdf")

# 读取pdf文件，保存为excel
with pdfplumber.open(file) as pdf:
    first_page = pdf.pages[0]

    # 自动读取表格信息，返回列表
    table = first_page.extract_table()
    # print(table)

    save = pd.DataFrame(table[2:],columns=table[0])
    # 保存excel
    save.to_excel('output.xlsx')
```

1.3. PyPDF2

2. Word 文字处理

Python-docx 是一款针对于word文档处理的一个模块，它可以创建word文档，遍历word文档，以及修改word文档。

主要应用场景是文档生成，文档转换，文档分析等等。

举例一、招聘网站自动生成Word文档简历，使用 python-docx 将用户输入的简历内容，自动拼装成 word 简历，并下载。

举例二、Word 文档差异对比，例如合同修改后，产生两个版本的 word 文档，我们不知道那一行，或者那一个字做了修改，人工核对费时费力，我们就可以写一个程序，逐行逐字核对，并将差异文字显示出来。

2.1. 安装

```
pip3 install python-docx
```

2.2. 创建空白文档

```
from docx import Document
document = Document()
document.save('new.docx')
```

2.3. 添加标题

```
from docx import Document

# 创建文档对象
document = Document()

# 标题
document.add_heading('标题一', 0)
document.add_heading('标题二', 1)
document.add_heading('标题三', 2)
document.add_heading('标题四', 3)
document.add_heading('标题五', 4)
document.add_heading('标题六', 5)
document.add_heading('标题七', 6)
document.add_heading('标题八', 7)
document.add_heading('标题九', 8)
document.add_heading('标题十', 9)

# 保存 word 文档
document.save('heading.docx')
```

设置对齐

```
from docx import Document
from docx.enum.text import WD_ALIGN_PARAGRAPH,
WD_LINE_SPACING

document = Document()
head = document.add_heading('Netkiller Python 手札', level=0)
head.alignment = WD_ALIGN_PARAGRAPH.CENTER  # 居中

document.add_heading('Python 入门', 1)
document.add_heading('安装 Python 运行环境', 2)
document.add_heading('Python 语法', 1)
document.add_heading('if 判断', 2)
document.add_heading('for 循环', 2)

document.save('Netkiller Python 手札.docx')
```

设置字体大小

```
from docx.enum.text import WD_ALIGN_PARAGRAPH,WD_LINE_SPACING
head = document.add_heading('level=1') #添加一级标题
run = head.add_run('这是标题')
run.font.size = Pt(24)
head.alignment = WD_ALIGN_PARAGRAPH.CENTER #居中
```

2.4. 添加段落

```
from docx import Document

# 创建文档对象
document = Document()

# 添加标题
document.add_heading('什么是多维度架构', 0)
document.add_heading('架构师的大局观', 1)
# 添加段落
document.add_paragraph(
    '我们从小的教育就是如何拆分问题、解决问题，这样做显然会使复杂的问题变得更容易些。但是这带来一个新问题，我们丧失了如何从宏观角度看问题，分析问题，解决问题，对更大的整体的内在领悟能力。这导致了我们对现有问题提出的解决方案，但无法预计实施该方案后产生的各种后果，为此我们付出了巨大代价。')
document.add_paragraph('而我们试图考虑大局的时候，总要在脑子里重新排序，组合哪些拆分出来问题，给它们编组列单。'
                       '习惯性认为解决了所有微观领域的问题，那么宏观上问题就得到了解决。'
                       '然而，这种做法是徒劳无益的，就好比试图通过重新拼起来的碎镜子来观察真实的影像。')
document.add_paragraph('所以在一段时间后，我们便干脆完全放弃了对整体的关注。')
```

```
document.add_heading('面临的问题', 2)

document.add_paragraph(
    '当今的社会，几乎所有的企业情况都是岗位职责清晰，分工明确，员工是企业机器上的一颗螺丝钉，我们在招聘下属的时候也仅仅是用他的一技之长。项目一旦立项，我们就根据项目需求针对性性的招聘，短短半年团队就会膨胀数倍，但效率并不是成正比增长。另一个问题是这个庞大的团队合作起来并不尽人意。结果是 80 % 协调的时间，20 % 实际工作时间。')

# 保存文档
document.save('paragraph.docx')
```

插入段落

```
paragraph = document.add_paragraph('明天去上班。')
在此段落之前插入一个段落，如下：
pre_paragraph = paragraph.insert_paragraph_before('今天休息。')
```

段落对齐

```
paragraph = document.add_paragraph("段落对齐方式")
paragraph_format = paragraph.paragraph_format
paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

段落缩进设置

```
from docx.shared import Inches

paragraph = document.add_paragraph("段落缩进")
```

```
paragraph_format = paragraph.paragraph_format
paragraph_format.left_indent = Inches(0.5)
```

首行缩进：

```
paragraph_format.first_line_indent = Inches(-0.25)
```

2.5. 列表

```
from docx import Document

# 创建文档
document = Document()

# 添加标题
document.add_heading('Linux', 0)
document.add_heading('桌面发行版', 1)

# 添加列表
document.add_heading('Desktop: ', 2)
document.add_paragraph('Ubuntu', style='List Bullet')
document.add_paragraph('Debian', style='List Bullet')
document.add_paragraph('Fedora', style='List Bullet')

# 保存文档
document.save('list.docx')
```

编号

```
document.add_paragraph(
    'first item in ordered list', style='List Number'
)
```

编号

```
from docx.enum.style import WD_STYLE_TYPE
from docx import Document

document = Document()
styles = document.styles

# 遍历所有表样式
for s in styles:
    if s.type == WD_STYLE_TYPE.TABLE:
        document.add_paragraph("表格样式 :  " + s.name)
        table = document.add_table(3, 3, style=s)
        heading_cells = table.rows[0].cells
        heading_cells[0].text = '第一列内容'
        heading_cells[1].text = '第二列内容'
        heading_cells[2].text = '第三列内容'
        document.add_paragraph("\n")

document.save('所有表格样式演示.docx')
```

2.6. 表格

```
from docx import Document

document = Document()
```

```
document.add_heading('表格演示', 1)

# 表格
table = document.add_table(rows=3, cols=2, style='Table
Grid')

# 表头
thead = table.rows[0].cells
thead[0].text = '姓名'
thead[1].text = '年龄'

# 表体
tbody = table.rows[1].cells
tbody[0].text = 'Neo'
tbody[1].text = '22'

tbody = table.rows[2].cells
tbody[0].text = 'Jam'
tbody[1].text = '33'

document.add_paragraph('2020年4月8日 netkiller 制表')
# 保存
document.save('table.docx')
```

2.7. 添加图片

```
document.add_picture('demo.png')
```

指定图片宽度，高度会自适应

```
# 图片
document.add_picture('pic.jpg', width=Inches(1))
```

指定图片宽度和高度

```
from docx.shared import Inches
document.add_picture('demo.png', width=Inches(1.0),
height=Inches(1.0))
```

2.8. 强制分页

```
# 分页
# document.add_page_break()
```

2.9. 样式

对齐

```
# LEFT       =>   左对齐
# CENTER     =>   文字居中
# RIGHT      =>   右对齐
# JUSTIFY    =>   文本两端对齐
```

标题对齐

```
style = document.styles['Normal']
# 标题
h0 = document.add_heading('标题0', 0)
# 居中
h0.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
```

段落对齐

```
from docx.enum.text import WD_ALIGN_PARAGRAPH
paragraph = document.add_paragraph("Netkiller Python 手札")
paragraph_format = paragraph.paragraph_format
paragraph_format.alignment = WD_ALIGN_PARAGRAPH.CENTER
```

首行缩进

```
# 首行缩进两个字符
paragraph_format = style.paragraph_format
paragraph_format.first_line_indent = Cm(0.74)
```

段落间距

设置值用 Pt 单位是 磅 ， 如下:

```
# 段前
paragraph_format.space_before = Pt(18)
# 段后
paragraph_format.space_after = Pt(12)
```

行间距

设置值用 Pt 单位是 磅 ，如下:

当行距为 最小值 和 固定值 时，设置值单位为 磅 ，需要用 Pt ;

```
#SINGLE          =>  单倍行距（默认）
#ONE_POINT_FIVE  =>  1.5倍行距
#DOUBLE2         =>  倍行距
#AT_LEAST        =>  最小值
#EXACTLY         =>  固定值
#MULTIPLE        =>  多倍行距
```

当行距为 多倍行距 时，设置值为数值，如下:

```
from docx.shared import Length
paragraph.line_spacing_rule = WD_LINE_SPACING.EXACTLY   #固定
值
paragraph_format.line_spacing = Pt(18)
# 固定值18磅
paragraph.line_spacing_rule = WD_LINE_SPACING.MULTIPLE  #多倍
行距
paragraph_format.line_spacing = 1.75
# 1.75倍行间距
```

粗体，斜体

```
# 段落
p1 = document.add_paragraph('样式演示')
# 字体加粗
p1.add_run('字体加粗').bold = True
# 斜体
p1.add_run('设置斜体').italic = True
```

字体大小

```
p2 = document.add_paragraph('正常尺寸')
run = p2.add_run('设置字体大小\12Pt ')
# 设置字体大小
run.font.size = Pt(12)
```

查看段落样式

查看paragraph有哪些样式

```
from docx import Document
from docx.enum.style import WD_STYLE_TYPE
document = Document()
for i in document.styles:
        if i.type == WD_STYLE_TYPE.PARAGRAPH:
                print(i.name)
>>>
Normal
Heading 1
Heading 2
Heading 3
Heading 4
Heading 5
Heading 6
```

```
Heading 7
Heading 8
Heading 9
No Spacing
Title
Subtitle
List Paragraph
Body Text
Body Text 2
Body Text 3
List
List 2
List 3
List Bullet
List Bullet 2
List Bullet 3
List Number
List Number 2
List Number 3
List Continue
List Continue 2
List Continue 3
macro
Quote
Caption
Intense Quote
TOC Heading
```

文档样式

查看文档有哪些样式

```
from docx import Document
from docx.enum.style import WD_STYLE_TYPE
document = Document()
for i in document.styles:
        if i.type == WD_STYLE_TYPE.CHARACTER:
                print(i.name)
```

```
>>>
Default Paragraph Font
Heading 1 Char
Heading 2 Char
Heading 3 Char
Title Char
Subtitle Char
Body Text Char
Body Text 2 Char
Body Text 3 Char
Macro Text Char
Quote Char
Heading 4 Char
Heading 5 Char
Heading 6 Char
Heading 7 Char
Heading 8 Char
Heading 9 Char
Strong
Emphasis
Intense Quote Char
Subtle Emphasis
Intense Emphasis
Subtle Reference
Intense Reference
Book Title
```

自动分页设置

```
#widow_control       =>  孤行控制，防止在页面顶端单独打印段落末行或在页面底端单独打印段落首行。
#keep_with_next      =>  与下段同页，防止在选中段落与后面一段间插入分页符。
#page_break_before   =>  段前分页，防止在段落中出现分页符。
#keep_together       =>  段中不分页，在选中段落前插入分页符。

paragraph_format.keep_with_next = True
```

样式演示

```
from docx import Document
from docx.shared import Inches
from docx.enum.text import WD_PARAGRAPH_ALIGNMENT
from docx.shared import Cm, Pt

# 创建文档
document = Document()
style = document.styles['Normal']
document.add_heading('标题剧中', 1)
# 首行缩进两个字符
paragraph_format = style.paragraph_format
paragraph_format.first_line_indent = Cm(0.74)
# 段落
p1 = document.add_paragraph(
    '我们从小的教育就是如何拆分问题、解决问题，这样做显然会使复杂的问题变得更容易些。')
# 加粗
p1.add_run('但是这带来一个新问题，我们丧失了如何从宏观角度看问题，分析问题，解决问题，对更大的整体的内在领悟能力。').bold = True
# 斜体
p1.add_run('这导致了我们对现有问题提出的解决方案，但无法预计实施该方案后产生的各种后果，为此我们付出了巨大代价。').italic = True

# 段落
p2 = document.add_paragraph('所以在一段时间后，')
# 字体大小
p2.add_run('我们便干脆完全放弃了对整体的关注。').font.size = Pt(40)

document.save('test.docx')
```

2.10. 演示例子

官方演示例子

```
from docx import Document
from docx.shared import Inches

document = Document()

document.add_heading('Document Title', 0)

p = document.add_paragraph('A plain paragraph having some ')
p.add_run('bold').bold = True
p.add_run(' and some ')
p.add_run('italic.').italic = True

document.add_heading('Heading, level 1', level=1)
document.add_paragraph('Intense quote', style='Intense
Quote')

document.add_paragraph(
    'first item in unordered list', style='List Bullet'
)
document.add_paragraph(
    'first item in ordered list', style='List Number'
)

document.add_picture('monty-truth.png', width=Inches(1.25))

records = (
    (3, '101', 'Spam'),
    (7, '422', 'Eggs'),
    (4, '631', 'Spam, spam, eggs, and spam')
)

table = document.add_table(rows=1, cols=3)
hdr_cells = table.rows[0].cells
hdr_cells[0].text = 'Qty'
hdr_cells[1].text = 'Id'
hdr_cells[2].text = 'Desc'
for qty, id, desc in records:
    row_cells = table.add_row().cells
    row_cells[0].text = str(qty)
```

```
    row_cells[1].text = id
    row_cells[2].text = desc

document.add_page_break()

document.save('demo.docx')
```

完整的演示例子

```
from docx import Document
from docx.enum.text import WD_ALIGN_PARAGRAPH,
WD_LINE_SPACING

document = Document()
head = document.add_heading('Netkiller Python 手札', level=0)
head.alignment = WD_ALIGN_PARAGRAPH.CENTER  # 居中
document.add_picture('netkiller.png')
document.add_heading('Python 入门', 1)
document.add_paragraph('Python 是脚本语言的一种，可以运行在
Linux,Mac,Window操作系统之上。')
document.add_heading('安装 Python 运行环境', 2)
document.add_paragraph('前往 python.org 网站，下载安装包，根据提示
安装。')
document.add_paragraph(
    'Linux : https://www.python.org/downloads/release/python-
395/', style='List Bullet')
document.add_paragraph(
    'Windows ：https://www.python.org/ftp/python/3.9.5/python-
3.9.5-amd64.exe', style='List Bullet')
document.add_paragraph(
    'MacOS : https://www.python.org/ftp/python/3.9.5/python-
3.9.5-macosx10.9.pkg', style='List Bullet')
document.add_heading('Python 语法', 1)
document.add_heading('if 判断', 2)
document.add_heading('for 循环', 2)

document.add_heading('文档修订历史记录', 1)
# 表格
table = document.add_table(rows=3, cols=3, style='Table
```

```
Grid')

# 表头
thead = table.rows[0].cells
thead[0].text = '版本'
thead[1].text = '作者'
thead[2].text = '描述'

# 表体
tbody = table.rows[1].cells
tbody[0].text = 'v1.0'
tbody[1].text = 'Neo'
tbody[2].text = '创建文档'

tbody = table.rows[2].cells
tbody[0].text = 'V2.0'
tbody[1].text = 'Jam'
tbody[2].text = '添加图片'

document.save('Netkiller Python 手札.docx')
```

2.11. 另存操作

```
from docx import Document
document = Document('原始文档.docx')
document.save('修改后的文档.docx')
```

2.12. 读取 Word 文档

```
from docx import Document

# 打开已存在文档
document = Document('test.docx')
```

```
# 读取标题、段落、列表内容
paras = [ paragraph.text for paragraph in
document.paragraphs]
for p in paras:
    print(p)

# 读取表格内容
tables = [table for table in document.tables]
for t in tables:
    for row in t.rows:
        for cell in row.cells:
            print(cell.text, end=' ')
        print()
```

风格筛选

```
from docx import Document

document = Document('Netkiller Python 手札.docx')

# 读取文中所有段落内容
for p in document.paragraphs:
    print("paragraphs: ", p.text)

# 读取文中所有标题一
for p in document.paragraphs:
    if p.style.name == 'Heading 1':
        print("Heading 1: ", p.text)

# 读取文中所有标题二
for p in document.paragraphs:
    if p.style.name == 'Heading 2':
        print("Heading 2: ", p.text)

# 读取文中正文
for p in document.paragraphs:
    if p.style.name == 'Normal':
        print("Normal: ", p.text)
```

运行结果

```
neo@MacBook-Pro-Neo ~/workspace/python % python3.9
/Users/neo/workspace/python/office/netkiller.py
paragraphs: Netkiller Python 手札
paragraphs: Python 入门
paragraphs: Python 是脚本语言的一种，可以运行在Linux,Mac,Window操作
系统之上。
paragraphs: 安装 Python 运行环境
paragraphs: 前往 python.org 网站，下载安装包，根据提示安装。
paragraphs: Linux :
https://www.python.org/downloads/release/python-395/
paragraphs: Windows :
https://www.python.org/ftp/python/3.9.5/python-3.9.5-amd64.exe
paragraphs: MacOS :
https://www.python.org/ftp/python/3.9.5/python-3.9.5-
macosx10.9.pkg
paragraphs: Python 语法
paragraphs: if 判断
paragraphs: for 循环
Heading 1: Python 入门
Heading 1: Python 语法
Heading 2: 安装 Python 运行环境
Heading 2: if 判断
Heading 2: for 循环
Normal: Python 是脚本语言的一种，可以运行在Linux,Mac,Window操作系统之
上。
Normal: 前往 python.org 网站，下载安装包，根据提示安装。
```

2.13. Word 模版合并

docx-mailmerge

安装 docx-mailmerge

3. Python 处理 Excel

3.1. openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files

https://openpyxl.readthedocs.io/en/stable/

创建空文档

```
from openpyxl import Workbook
workbook = Workbook()
sheet = workbook.active
workbook.save('netkiller.xlsx')
```

工作表

默认工作表

sheet = workbook.active 会创建一个工作表，默认名字是 Sheet 修改方法是 sheet.title = 'netkiller'

```
from openpyxl import Workbook
workbook = Workbook()
sheet = workbook.active
# 重命名工作表
sheet.title = 'netkiller'
workbook.save('openpyxl.xlsx')
```

创建新工作表

```
from openpyxl import Workbook

# 默认工作表
workbook = Workbook()
worksheet = workbook.active
worksheet.title = 'Windows'

# 创建新工作表，后面的参数0表示表示在工作表中的位置，0是第一位。
worksheet1 = workbook.create_sheet("MacOS", 0)
worksheet2 = workbook.create_sheet("Linux")
workbook.save('worksheet.xlsx')
```

遍历工作表

```
from openpyxl import Workbook, load_workbook

workbook = load_workbook('worksheet.xlsx')

print('输出文件所有工作表名：', workbook.sheetnames)

# 遍历查看当前 Excel 文档所有工作表名称
for sheet in workbook:
    print(sheet.title)
```

删除工作表

```
from openpyxl import Workbook, load_workbook

workbook = load_workbook('worksheet.xlsx')
```

```
print('列出当前文档所有工作表名: ', workbook.sheetnames)
# 打开默认工作表
sheet = workbook.active
# 删除工作表
workbook.remove(sheet)
# 遍历查看当前 Excel 文档所有工作表名称
for sheet in workbook:
    print(sheet.title)
```

单元格

单元格填充数据

```
from openpyxl import Workbook
workbook = Workbook()
worksheet = workbook.active
worksheet.title = '员工表'

# 向单元格写入数据
worksheet['A1'] = '姓名'
worksheet['B1'] = '性别'
worksheet.cell(row=2, column=1, value='景峯')
worksheet.cell(row=2, column=2, value='男')
worksheet.cell(row=3, column=1, value='小明')
worksheet.cell(row=3, column=2, value='男')
worksheet.cell(row=3, column=1, value='小梅')
worksheet.cell(row=3, column=2, value='女')

workbook.save('cell.xlsx')
```

追加数据

```
```

```
import datetime
import random
from openpyxl import Workbook
wb = Workbook()

# grab the active worksheet
ws = wb.active

# Data can be assigned directly to cells
ws['A1'] = '数据测试表'

# Rows can also be appended
for i in range(10):
    ws.append([random.randint(1, 10), random.randint(
        1, 10), random.randint(1, 10)])

# Python types will automatically be converted
ws['A12'] = "创建日期"
ws['B12'] = datetime.datetime.now()

# Save the file
wb.save("sample.xlsx")
```

获取工作表行数和列数

```
from openpyxl import Workbook
workbook = Workbook()
worksheet = workbook.active
worksheet.title = '员工表'

# 向单元格写入数据
worksheet['A1'] = '姓名'
worksheet['B1'] = '性别'
worksheet.cell(row=2, column=1, value='景峯')
worksheet.cell(row=2, column=2, value='男')
worksheet.cell(row=3, column=1, value='小明')
worksheet.cell(row=3, column=2, value='男')
worksheet.cell(row=3, column=1, value='小梅')
```

```
worksheet.cell(row=3, column=2, value='女')

workbook.save('cell.xlsx')

# 获取表有多少列
print('输出文件所有工作表名：', worksheet.max_column)

# 获取表中最多有多少行
print('输出文件所有工作表名：', worksheet.max_row)
```

读取单元格

```
# 读取
col0 = worksheet['A1']
col1 = worksheet.cell(row=1, column=2)
# print(c, c1)
print(col0.value, col1.value)
```

读取行列数据

```
import datetime
import random
from openpyxl import Workbook
wb = Workbook()

ws = wb.active

# Data can be assigned directly to cells
ws['A1'] = '序列号'
ws['B1'] = '随机数列A'
ws['C1'] = '随机数列B'
ws['D1'] = '随机数列C'
```

```
# Rows can also be appended
for i in range(10):
    ws.append([i+1, random.randint(1, 10), random.randint(
        1, 10), random.randint(1, 10)])

# Python types will automatically be converted
ws['A12'] = "合计"
ws['B12'] = "=SUM(B2:B11)"
ws['C12'] = "=SUM(C2:C11)"
ws['D12'] = "=SUM(D2:D11)"


# 获取指定单元格数据
a3 = ws['A1']
print(a3.value)

print(("-" * 10) + "获取A列数据" + ("-" * 10))

# 获取指定一列数据
columns = ws['A']
for i in columns:
    print(i.value)

print(("-" * 10) + "获取第3行数据" + ("-" * 10))

# 获取一行数据
rows = ws[3]
for i in rows:
    print(i.value, sep=',')
print()
print(("-" * 10) + "iter_rows方法演示" + ("-" * 10))

# 获取数据库
for row in ws.iter_rows(min_row=1, max_col=2,
max_row=ws.max_row-1):
    for cell in row:
        print(cell.value, end="\t")
    print()

print(("-" * 10) + "遍历工作表" + ("-" * 10))
for row in ws.values:
    for value in row:
        print(value, end="\t")
    print()
```

```
# Save the file
# wb.save("formulae.xlsx")
```

修改单元格

```
# 修改
worksheet['A1'] = '参加人'
worksheet.cell(2, 2).value = 'netkiller'
```

单元格合并/取消合并

```
from openpyxl.workbook import Workbook

wb = Workbook()
ws = wb.active

ws.merge_cells('A2:D2')
ws.unmerge_cells('A2:D2')

# or equivalently
ws.merge_cells(start_row=2, start_column=1, end_row=4,
end_column=4)
ws.unmerge_cells(start_row=2, start_column=1, end_row=4,
end_column=4)
```

单元格格式化

日期格式化

```
import datetime
from openpyxl import Workbook
wb = Workbook()
ws = wb.active
# set date using a Python datetime
ws['A1'] = datetime.datetime(2010, 7, 21)

print(ws['A1'].number_format)
'yyyy-mm-dd h:mm:ss'
```

使用公式

```
import datetime
import random
from openpyxl import Workbook
wb = Workbook()

ws = wb.active

# Data can be assigned directly to cells
ws['A1'] = '序列号'
ws['B1'] = '随机数列A'
ws['C1'] = '随机数列B'
ws['D1'] = '随机数列C'
# Rows can also be appended
for i in range(10):
    ws.append([i+1, random.randint(1, 10), random.randint(
        1, 10), random.randint(1, 10)])

# Python types will automatically be converted
ws['A12'] = "合计"
ws['B12'] = "=SUM(B2:B11)"
ws['C12'] = "=SUM(C2:C11)"
ws['D12'] = "=SUM(D2:D11)"

# Save the file
wb.save("formulae.xlsx")
```

插入图片

```
from openpyxl import Workbook
from openpyxl.drawing.image import Image

wb = Workbook()
ws = wb.active
ws['A1'] = 'You should see three logos below'
# create an image
img = Image('logo.png')
# add to worksheet and anchor next to cells
ws.add_image(img, 'A1')
wb.save('logo.xlsx')
```

行高和列宽

```
# 将整个表的行高设置为 50，列宽设置为 30；
sheet.row_dimensions.height = 50
sheet.column_dimensions.width = 30
```

```
# 设置第 1 行的高度
sheet.row_dimensions[1].height = 50
# 设置 B 列的宽度
sheet.column_dimensions["B"].width = 20
```

行列隐藏

```
import openpyxl
wb = openpyxl.Workbook()
ws = wb.create_sheet()
ws.column_dimensions.group('A','D', hidden=True)
ws.row_dimensions.group(1,10, hidden=True)
wb.save('group.xlsx')
```

样式设置

sheet选项卡背景色

修改sheet选项卡背景色，默认为白色

```
ws.sheet_properties.tabColor = "FFA500"
```

字体

```
"""
name:            字体名称
color:           颜色通常是RGB或aRGB十六进制值
b(bold):         加粗（bool）
i(italic):       倾斜(bool)
shadow:                  阴影（bool）
underline:       下划线（‘doubleAccounting’, ‘single’, ‘double’,
```

```
‘singleAccounting’)
charset:        字符集(int)
strike:         删除线(bool)
"""
```

```
from openpyxl import Workbook
from openpyxl.styles import Font

workbook = Workbook()
worksheet = workbook.active

fontStyle = Font(size="18")
worksheet.cell(row=1, column=1, value='《Netkiller Python 手
札》').font = fontStyle
worksheet.merge_cells('A1:E1')

worksheet.cell(row=2, column=1, value='作者：netkiller').font
= Font(
    name='黑体', color='FF4433', size=15, b=False)

workbook.save('font.xlsx')
```

单元格背景色

```
from openpyxl.styles import PatternFill, GradientFill
from openpyxl import Workbook

workbook = Workbook()
worksheet = workbook.active

# 填充单元格颜色
worksheet["A1"].fill = PatternFill(fill_type="solid",
fgColor="99ccff")
worksheet.merge_cells('A1:E1')
```

```
# 填充单元格渐变色
worksheet["A3"].fill = GradientFill(stop=("FFFFFF", "99ccff",
"000000"))
worksheet.merge_cells('A3:E3')

workbook.save(filename="fill.xlsx")
```

设置单元格样式

```
from openpyxl import Workbook
from openpyxl.styles import Font, Alignment, PatternFill
workbook = Workbook()
worksheet = workbook.active
worksheet.title = '员工表'

# 在第一行插入新的一行作为表头
worksheet.insert_rows(1)
# 设置文本标题
worksheet['A1'] = '《Netkiller Python 手札》'
# 水平跨列合并单元格
worksheet.merge_cells('A1:E1')
# 设置样式
style = worksheet['A1']
# 设置字体
font = Font(name='黑体', color='555555', size=15, b=True)

style.font = font

# 设置文本对齐
align = Alignment(horizontal='center', vertical='center')
"""
horizontal:水平对齐('centerContinuous', 'general',
'distributed',
                    'left', 'fill', 'center', 'justify',
'right')
vertical:垂直对齐（'distributed', 'top', 'center', 'justify',
'bottom'）
```

```
"""
style.alignment = align

# 设置图案填充
fill = PatternFill('solid', fgColor='FFAABB')
style.fill = fill
workbook.save('style.xlsx')
```

综合应用

```
from openpyxl.styles import Side, Border, Alignment, Font,
PatternFill, NamedStyle, colors
from openpyxl import Workbook
from openpyxl.utils import get_column_letter
import random

workbook = Workbook()
worksheet = workbook.active

worksheet['A1'] = '数据测试表'
worksheet.merge_cells('A1:E1')


worksheet['A2'] = '序号'
worksheet['B2'] = 'A列'
worksheet['C2'] = 'B列'
worksheet['D2'] = 'C列'
worksheet['E2'] = '合计'

# Rows can also be appended
for i in range(1, 10):
    worksheet.append([i, random.randint(1, 10),
random.randint(
        1, 10), random.randint(1, 10),
"=SUM(B{}:D{})".format(i+2, i+2)])

# Python types will automatically be converted
worksheet['A12'] = "合计"
worksheet['B12'] = "=SUM(B2:B11)"
```

```
worksheet['C12'] = "=SUM(C2:C11)"
worksheet['D12'] = "=SUM(D2:D11)"
worksheet['E12'] = "=SUM(E2:E11)"
# worksheet['B12'] = datetime.datetime.now()


# 字体
fontTitle = Font(name='黑体', size=18, b=True)
fontHeader = Font(name='黑体', size=11, b=True)

# 边框
thin = Side(style='thin', color='000000')  # 细边框
medium = Side(style='medium', color=colors.BLACK)  # 粗边框

titleBorder = Border(top=medium, bottom=medium, left=medium,
right=medium)
headerBorder = Border(top=medium, bottom=thin, left=thin,
right=thin)
bodyBorder = Border(top=thin, bottom=thin, left=thin,
right=thin)

# 最外边粗边框
tableTopBorder = Border(top=medium)
tableRightBorder = Border(right=medium)
tableBottomBorder = Border(bottom=medium)
tableLeftBorder = Border(left=medium)

# 单元格填充颜色
titlePatternFill = PatternFill('solid', fgColor='CFCFCF')
headerPatternFill = PatternFill('solid', fgColor='EEEEEE')

# 对齐
alignment = Alignment(horizontal='center', vertical='center')

# 将样式打包命名
titleNamedStyle = NamedStyle(name='titleNamedStyle',
font=fontTitle, fill=titlePatternFill,
                             alignment=alignment)  #
border=titleBorder,
# 表头样式
headerNamedStyle = NamedStyle(
    name='headerNamedStyle', font=fontHeader,
fill=headerPatternFill, alignment=alignment,
border=headerBorder)
headerLeftNamedStyle = NamedStyle(
```

```
    name='headerLeftNamedStyle', font=fontHeader,
fill=headerPatternFill, alignment=alignment,
border=tableLeftBorder+tableTopBorder)
headerRightNamedStyle = NamedStyle(
    name='headerRightNamedStyle', font=fontHeader,
fill=headerPatternFill, alignment=alignment,
border=tableRightBorder+tableTopBorder)

bodyNamedStyle = NamedStyle(
    name='bodyNamedStyle', font=fontHeader,
border=bodyBorder, alignment=alignment)

# 读取数据表格范围
rows = worksheet.max_row
cols = worksheet.max_column

worksheet['A1'].style = titleNamedStyle

# for cell in worksheet[1]:
# cell.border = tableTopBorder

n = 0
for cell in worksheet[2]:
    if n == 0:
        cell.style = headerLeftNamedStyle
    elif n == cols-1:
        cell.style = headerRightNamedStyle
    else:
        cell.style = headerNamedStyle
    n += 1

for r in range(3, rows+1):
    for c in range(1, cols+1):
        if c == 1:
            worksheet.cell(r, c).border = tableLeftBorder +
bodyBorder
        elif c == worksheet.max_column:
            worksheet.cell(r, c).border = tableRightBorder +
bodyBorder
        else:
            worksheet.cell(r, c).style = bodyNamedStyle

n = 0
for cell in worksheet[rows]:
    if n == 0:
```

```
        cell.border = tableLeftBorder + tableBottomBorder +
bodyBorder
    elif n == cols-1:
        cell.border = tableRightBorder + tableBottomBorder +
bodyBorder
    else:
        cell.border = tableBottomBorder + bodyBorder
    n += 1

# for cell in worksheet['A']:
#     cell.border = tableLeftBorder + bodyBorder
# for cell in
worksheet[get_column_letter(worksheet.max_column)]:
#     cell.border = tableRightBorder + bodyBorder

workbook.save("borders.xlsx")
```

工具

数字列转标签

```
from openpyxl.utils import get_column_letter

letter = column = get_column_letter(3)

print(letter)
```

第3列对应标签是C

```
neo@MacBook-Pro-Neo ~/workspace/python % python3.9
/Users/neo/workspace/python/office/openpyxl/utils.py
C
```

3.2. xlrd/xlwt/xlutils

https://github.com/python-excel

读 Excel

安装 xlrd

```
neo@MacBook-Pro-Neo ~/workspace/python % pip install xlrd
```

演示程序

```
import xlrd

workbook = xlrd.open_workbook(filename='test.xls')  # 打开文件
print("获取所有工作表: {}".format(workbook.sheet_names()))
# 获取所有工作表
print("工作表数量 {0}".format(workbook.nsheets))

sheet1 = workbook.sheet_by_index(0)        # 通过索引获取表格
sheet2 = workbook.sheet_by_name('工资表')   # 通过名字获取表格
print(sheet1, sheet2)
print("工作表名 {}, 行数 {}, 列数 {}".format(sheet1.name,
sheet1.nrows, sheet1.ncols))

rows = sheet1.row_values(2)                 # 获取行内容
cols = sheet1.col_values(3)                 # 获取列内容
print("第二行内容 {}".format(rows))
print("第三列内容 {}".format(cols))

# 获取表格里的内容，三种方式
print(sheet1.cell(1, 0).value)
```

```
print(sheet1.cell_value(1, 0))
print(sheet1.row(1)[0].value)
```

写 Excel

```
neo@MacBook-Pro-Neo ~/workspace/python % pip install xlwt
```

添加工作表

```
import xlwt
# 创建workbook对象
workbook = xlwt.Workbook()
# 创建工作表 sheet
sheet1 = workbook.add_sheet('sheet1', cell_overwrite_ok=True)
sheet2 = workbook.add_sheet('sheet2', cell_overwrite_ok=True)

# 向sheet工作表中写入数据
sheet1.write(0, 0, '姓名')
sheet1.write(0, 1, '性别')
sheet1.write(0, 2, '年龄')

sheet1.write(1, 0, '景峯')
sheet1.write(1, 1, '男')
sheet1.write(1, 2, '35')

# 第二张工作表
sheet2.write(0, 0, '姓名')
sheet2.write(0, 1, '性别')
sheet2.write(0, 2, '年龄')

sheet2.write(1, 0, '景峯')
sheet2.write(1, 1, '男')
sheet2.write(1, 2, '35')
```

```
# 保存该excel文件,文件同名会被覆盖
workbook.save('class.xlsx')
```

设置编码

```
excel = xlwt.Workbook(encoding='utf-8')
```

设置列宽

```
sheet.col(0).width = 25 * 256
```

设置行高

```
sheet.row(0).height_mismatch = True
sheet.row(0).height = 20 * 20
```

合并单元格

```
import xlwt
```

```
workbook = xlwt.Workbook()
worksheet = workbook.add_sheet('My Sheet')
# 参数详解： write_merge(列, 行, 合并列数, 合并行数, '文本', 样式)
worksheet.write_merge(0, 0, 0, 3, '《Netkiller Python 手札》')
worksheet.write_merge(1, 2, 0, 3, '作者：netkiller')
workbook.save('netkiller.xls')
```

```
import xlwt
# 创建workbook对象
workbook = xlwt.Workbook()
# 创建工作表 sheet
sheet1 = workbook.add_sheet('sheet1', cell_overwrite_ok=True)

# 水平合并行单元格
sheet1.write_merge(0, 0, 0, 3, '班级学生名单')

# 向sheet工作表中写入数据
sheet1.write(1, 0, '姓名')
sheet1.write(1, 1, '性别')
sheet1.write(1, 2, '年龄')
sheet1.write(1, 2, '疫苗接种')

sheet1.write(2, 0, '景峯')
sheet1.write(2, 1, '男')
sheet1.write(2, 2, '35')

sheet1.write(3, 0, '小明')
sheet1.write(3, 1, '男')
sheet1.write(3, 2, '35')

# 垂直合并列单元格
sheet1.write_merge(2, 3, 3, 3, '已接种')

# 保存该excel文件,文件同名会被覆盖
workbook.save('class.xlsx')
```

运行公式

```
import xlwt
workbook = xlwt.Workbook()
worksheet = workbook.add_sheet('My Sheet')
worksheet.write(0, 0, 5) # Outputs 5
worksheet.write(0, 1, 2) # Outputs 2
worksheet.write(1, 0, xlwt.Formula('A1*B1'))            #
Should output "10"(A1[5] * A2[2])
worksheet.write(1, 1, xlwt.Formula('SUM(A1,B1)'))       #
Should output"7" (A1[5] + A2[2])
workbook.save('Workbook.xls')
```

超链接

```
import xlwt
workbook = xlwt.Workbook()
worksheet = workbook.add_sheet('My Sheet')
worksheet.write(0,
0,xlwt.Formula('HYPERLINK("http://www.netkiller.cn";"Netkille
r Python 手札")'))
workbook.save('Excel.xls')
```

样式设置

单元格对齐

```
import xlwt
# 创建workbook对象
workbook = xlwt.Workbook()
# 创建工作表 sheet
sheet1 = workbook.add_sheet('sheet1', cell_overwrite_ok=True)
```

```
title = xlwt.XFStyle()
alignment = xlwt.Alignment()
# 垂直对齐
alignment.horz = alignment.HORZ_CENTER
# 水平对齐
alignment.vert = alignment.VERT_CENTER
# 换行
alignment.wrap = alignment.WRAP_AT_RIGHT
# 设置边框

title.alignment = alignment

# valign = xlwt.XFStyle()

# 水平合并行单元格
sheet1.write_merge(0, 0, 0, 3, '班级学生名单', title)

# 向sheet工作表中写入数据
sheet1.write(1, 0, '姓名')
sheet1.write(1, 1, '性别')
sheet1.write(1, 2, '年龄')
sheet1.write(1, 2, '疫苗接种')

sheet1.write(2, 0, '景峯')
sheet1.write(2, 1, '男')
sheet1.write(2, 2, '35')

sheet1.write(3, 0, '小明')
sheet1.write(3, 1, '男')
sheet1.write(3, 2, '35')

# 垂直合并列单元格
sheet1.write_merge(2, 3, 3, 3, '已接种')

# 保存该excel文件,文件同名会被覆盖
workbook.save('class.xlsx')
```

字体和颜色

```
```

```
import xlwt
# 创建workbook对象
workbook = xlwt.Workbook()
# 创建工作表 sheet
sheet1 = workbook.add_sheet('sheet1', cell_overwrite_ok=True)

title = xlwt.XFStyle()

font = xlwt.Font()
# 字体类型
font.name = 'name Times New Roman'
# 字体颜色
font.colour_index = 4
# 字体大小，11为字号，20为衡量单位
font.height = 20 * 11
# 字体加粗
font.bold = False
# 下划线
font.underline = True
# 斜体字
font.italic = True


alignment = xlwt.Alignment()
# 垂直对齐
alignment.horz = alignment.HORZ_CENTER
# 水平对齐
alignment.vert = alignment.VERT_CENTER
# 换行
alignment.wrap = alignment.WRAP_AT_RIGHT
# 设置边框

title.alignment = alignment
title.font = font

valign = xlwt.XFStyle()
va = xlwt.Alignment()
va.vert = alignment.VERT_CENTER
valign.alignment = va

# 水平合并行单元格
sheet1.write_merge(0, 0, 0, 3, '班级学生名单', title)

# 向sheet工作表中写入数据
sheet1.write(1, 0, '姓名')
```

```
sheet1.write(1, 1, '性别')
sheet1.write(1, 2, '年龄')
sheet1.write(1, 3, '疫苗接种')

sheet1.write(2, 0, '景峯')
sheet1.write(2, 1, '男')
sheet1.write(2, 2, '35')

sheet1.write(3, 0, '小明')
sheet1.write(3, 1, '男')
sheet1.write(3, 2, '35')

# 垂直合并列单元格
sheet1.write_merge(2, 3, 3, 3, '已接种', valign)

# 保存该excel文件,文件同名会被覆盖
workbook.save('class.xlsx')
```

设置边框

```
import xlwt
# 创建workbook对象
workbook = xlwt.Workbook()
# 创建工作表 sheet
sheet1 = workbook.add_sheet('sheet1', cell_overwrite_ok=True)

borders = xlwt.Borders()

# 细实线:1，小粗实线:2，细虚线:3，中细虚线:4，大粗实线:5，双线:6，细点虚线:7
# 大粗虚线:8，细点划线:9，粗点划线:10，细双点划线:11，粗双点划线:12，斜点划线:13
borders.left = 1
borders.right = 1
borders.top = 1
borders.bottom = 1
# borders.left_colour = 1
# borders.right_colour = 1
```

```
# borders.top_colour = 1
# borders.bottom_colour = 1

style_borders = xlwt.XFStyle()
style_borders.borders = borders

title = xlwt.XFStyle()
alignment = xlwt.Alignment()
# 垂直对齐
alignment.horz = alignment.HORZ_CENTER
# 水平对齐
alignment.vert = alignment.VERT_CENTER
# 换行
alignment.wrap = alignment.WRAP_AT_RIGHT
# 设置边框

title.alignment = alignment

valign = xlwt.XFStyle()
va = xlwt.Alignment()
va.vert = alignment.VERT_CENTER
valign.alignment = va
valign.borders = borders

# 水平合并行单元格
sheet1.write_merge(0, 0, 0, 3, '班级学生名单', title)

# 向sheet工作表中写入数据
sheet1.write(1, 0, '姓名', style_borders)
sheet1.write(1, 1, '性别', style_borders)
sheet1.write(1, 2, '年龄', style_borders)
sheet1.write(1, 3, '疫苗接种', style_borders)

sheet1.write(2, 0, '景峯', style_borders)
sheet1.write(2, 1, '男', style_borders)
sheet1.write(2, 2, '35', style_borders)

sheet1.write(3, 0, '小明', style_borders)
sheet1.write(3, 1, '男', style_borders)
sheet1.write(3, 2, '35', style_borders)

# 垂直合并列单元格
sheet1.write_merge(2, 3, 3, 3, '已接种', valign)

# 保存该excel文件,文件同名会被覆盖
```

```
workbook.save('class.xlsx')
```

设置背景色

```
import xlwt
workbook = xlwt.Workbook()
worksheet = workbook.add_sheet('My Sheet')
pattern = xlwt.Pattern()  # Create the Pattern
# May be: NO_PATTERN,SOLID_PATTERN, or 0x00 through 0x12
pattern.pattern = xlwt.Pattern.SOLID_PATTERN
# May be: 8 through 63. 0 = Black,1 = White, 2 = Red, 3 =
Green, 4 = Blue, 5 = Yellow, 6 = Magenta, 7= Cyan, 16 =
Maroon, 17 = Dark Green, 18 = Dark Blue, 19 = DarkYellow ,
almost brown), 20 = Dark Magenta, 21 = Teal, 22 = LightGray,
23 = Dark Gray, the list goes on...
pattern.pattern_fore_colour = 5

style = xlwt.XFStyle()  # Create Style
style.pattern = pattern  # Add Borders to Style
worksheet.write(0, 0, 'Cell Contents', style)
workbook.save('Workbook.xls')
```

单元格格式化

```
import xlwt
from datetime import datetime
workbook = xlwt.Workbook()
sheet1 = workbook.add_sheet('工资表', cell_overwrite_ok=True)
columes = ["姓名", "年龄", "出生日期", "工资", '报销']
datas = [
    ["张三", 13, '2019-02-12', 16800, 15779.24],
    ["李四", 12, '2018-03-15', 17200, -24.225]
]
```

```
format = xlwt.XFStyle()
format.num_format_str = 'yyyy/mm/dd'

number = xlwt.XFStyle()


sheet1.write_merge(0, 0, 0, 4, '工资表')  # 合并行单元格
# 写第一行
for i in range(0, len(columes)):
    sheet1.write(1, i, columes[i])

line = 2
# 写第一列
for data in datas:
    for col in range(len(data)):
        cell = data[col]
        if col == 2:
            date = datetime.strptime(cell, '%Y-%m-%d').date()
# %H:%M:%S
            # print(date)
            sheet1.write(line, col, date, format)
        elif col == 3:
            number.num_format_str = '#,##;[RED]-#,##'
            sheet1.write(line, col, cell, number)
        elif col == 4:
            number.num_format_str = '#,##0.00;[RED]-#,##0.00'
            sheet1.write(line, col, cell, number)
        else:
            sheet1.write(line, col, cell)
    line = line+1

workbook.save('test.xls')
```

xlutils

安装 xlutils

```
pip install xlutils
```

```
import xlrd
import xlutils.copy

# 打开一个xls文件
xls = xlrd.open_workbook('test.xls')
workbook = xlutils.copy.copy(xls)

# 添加新sheet表
workbook.add_sheet('sheet2', cell_overwrite_ok=True)

# 获取sheet对象，通过sheet_by_index()获取的sheet对象没有write()方法
sheet = workbook.get_sheet(0)

# 修改数据
sheet.write(2, 0, '王二小')

# 保存时同名会覆盖，达到修改excel文件的目的
workbook.save('test.xls')
```

3.3. xlwings

```
import matplotlib.pyplot as plt
import xlwings as xw

fig = plt.figure()
plt.plot([1, 2, 3])

sheet = xw.Book().sheets[0]
sheet.pictures.add(fig, name='Plot', update=True)
```

第 25 章 OpenCV

1. 安装 OpenCV

```
pip3 install opencv-python
```

2. 显示图片

```
import cv2 as cv

src = cv.imread("me.jpeg")
cv.namedWindow("Picture", cv.WINDOW_AUTOSIZE)
cv.imshow("Picture", src)
cv.waitKey(0)
# 关闭所有窗口
cv.destroyAllWindows()
```

3. 摄像头捕捉图像

```
import cv2 as cv

capture = cv.VideoCapture(1)
while (True):
    # ret为返回值，frame为视频的每一帧
    ret, frame = capture.read()
    cv.imshow("video", frame)
    c = cv.waitKey(50)
    # 按了esc候可以退出
    if c == 27:
        break
```

4. imread()

```
import cv2 as cv

image = cv.imread("me.jpeg")

print(type(image))  # 类别
print(image.shape)  # 高，宽，通道数目
print(image.size)   # 像素数据，上面三个属性的乘积
print(image.dtype)
```

第 26 章 图形开发

1. SVG 图形库

1.1. 安装

```
pip3 install drawsvg -i
https://pypi.tuna.tsinghua.edu.cn/simple
pip3 install cairosvg -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

1.2. 绘制多边形

```
lines = draw.Lines(
    # 坐标
    5, 5,
    # 横线
    200, 5,
    # 竖线
    200, 40,
    # 斜线
    200 - 10, 20,
    # 横线2
    5 + 10, 20,
    # 斜线
    5, 40,
    # 闭合竖线
    5,5,
    fill='none', stroke='black')
```

1.3. SVG 事件

```
<!DOCTYPE html>
<html>
<body>
<svg width="500" height="150">

<rect x="10" y="10" width="100" height="40"
   style="stroke: black; fill: silver; fill-opacity: .4;"
   onmouseover="this.style.stroke = 'blue'; this.style['stroke-
width'] = 5;"
   onmouseout="this.style.stroke = 'green'; this.style['stroke-
width'] = 1;"
   onclick="this.style['width'] = 300;" />

</svg>
</body>
</html>
```

```
<!DOCTYPE html>
<html>

<body>
   <input id="aa" type="text" value="200"
onclick="svg.style['width']=this.value;" />
   <svg width="500" height="150">


      <rect id='svg' x="10" y="10" width="100" height="40"
style="stroke: black; fill: silver; fill-opacity: .4;"
         onmouseover="this.style.stroke = 'blue';
this.style['stroke-width'] = 5;"
         onmouseout="this.style.stroke = 'green';
this.style['stroke-width'] = 1;"
         onclick="this.style['width'] = 300; aa.value='300'" />

   </svg>
</body>
```

```
</html>
```

2. PIL

```
from PIL import Image
from PIL import ImageChops
def compare_images(one, two, diff):
  image_one = Image.open(one)
  image_two = Image.open(two)
  try:
    diff = ImageChops.difference(image_one, image_two)
    if diff.getbbox() is None:
      print("【+】We are the same!")
    else:
      diff.save(diff)
  except ValueError as e:
    text = ("Pastes another image into this image."
        "The box argument is either a 2-tuple giving the
upper left corner, a 4-tuple defining the left, upper, "
        "right, and lower pixel coordinate, or None (same as
(0, 0)). If a 4-tuple is given, the size of the pasted "
        "image must match the size of the region.")
    print("【{0}】{1}".format(e,text))
if __name__ == '__main__':
  compare_images('1.png', '2.png', '我们不一样.png')
```

3. 二维码

3.1. qrcode

安装

pip install qrcode

```
neo@MacBook-Pro-Neo ~ % pip install qrcode
Collecting qrcode
  Downloading qrcode-6.1-py2.py3-none-any.whl (31 kB)
Requirement already satisfied: six in
/usr/local/lib/python3.9/site-packages (from qrcode) (1.15.0)
Installing collected packages: qrcode
Successfully installed qrcode-6.1
```

常规二维码

```
import qrcode

img = qrcode.make('《Netkiller Python 手札》')
img.save("text.png")

img = qrcode.make('http://www.netkiller.cn')
img.save("url.png")
```

设置颜色

颜色设置背景为白色，前景为绿色

```
import qrcode as qrcode
qr = qrcode.QRCode(version=1, box_size=10, border=2)
# 向二维码添加数据
qr.add_data("http://www.netkiller.cn")
qr.make(fit=True)

# 更改QR的背景为白色和绘画颜色为绿色
img = qr.make_image(fill_color="green", back_color="white")
img.save('green.png')
img.show()  # 显示二维码
```

QRCode 参数

version: 范围为1到40整数（最小值是1，表示12×12的矩阵），如果想让程序自动生成，将值设置为 None 并使用 fit=True 参数即可。
error_correction: 二维码的纠错范围，可以选择4个常量:
1. ERROR_CORRECT_L 7%以下的错误会被纠正
2. ERROR_CORRECT_M (default) 15%以下的错误会被纠正
3. ERROR_CORRECT_Q 25 %以下的错误会被纠正
4. ERROR_CORRECT_H. 30%以下的错误会被纠正
boxsize: 每个点（方块）中的像素个数
border: 二维码图像外围边框宽度，默认为4

qr - script to create QR codes at the command line

```
NAME
       qr - script to create QR codes at the command line

SYNOPSIS
       qr  [--help]  [--factory=FACTORY]  [--optimize=OPTIMIZE]
```

```
[--error-correction=LEVEL]
       [data]

DESCRIPTION
       This script uses the python qrcode module. It can take
data from stdin or from  the
       commandline  and  generate a QR code.  Normally it will
output the QR code as ascii
       art to the terminal. If the output is piped to a file,
it  will  output  the  image
       (default type of PNG).

OPTIONS
        -h, --help
           Show a help message.


        --factory=FACTORY
           Full  python  path to the image factory class to
create the image with. You can
           use the  following  shortcuts  to  the  built-in
image  factory  classes:  pil
           (default), pymaging, svg, svg-fragment, svg-path.


        --optimize=OPTIMIZE
           Optimize  the  data by looking for chunks of at
least this many characters that
           could use a more efficient encoding method. Use 0 to
turn off  chunk  optimiza-
           tion.


        --error-correction=LEVEL
           The  error  correction  level  to  use. Choices are
L (7%), M (15%, default), Q
           (25%), and H (30%).


        data
           The data from which the QR code will be generated.


SEE ALSO
       https://github.com/lincolnloop/python-qrcode/
```

使用方法

```
neo@MacBook-Pro-Neo ~ % qr "Some text" > test.png
```

3.2. MyQR

安装依赖包

```
pip install myqr
```

```
from MyQR import myqr
# myqr.run 参数:
# words: 文本/链接, 或者你想说的话(不支持中文, 很不友好)
# picture: 背景图
# colorsize: True 表示生成彩图
# save_name: 生成的二维码文件名
myqr.run(words="http://www.netkiller.com",
         picture="db.png", colorized=True,
         save_name="netkiller.png")
```

3.3. 从图片识别二维码

安装依赖包

```
pip install pyzbar
```

pyzbar 是调用 zbar 共享库，所以还需要安装 zbar

```
brew install zbar
```

```
import numpy as np
from PIL import Image
import pyzbar.pyzbar as pyzbar

# 读取文件，转成数组
im = np.array(Image.open("netkiller.png"))
print(pyzbar.decode(im))
# 返回的信息
print('-' * 50)
# 读取内容
print(pyzbar.decode(im)[0].data.decode("utf-8"))
```

输出信息

```
[Decoded(data=b'http://www.netkiller.com', type='QRCODE',
rect=Rect(left=36, top=36, width=297, height=297), polygon=
[Point(x=36, y=36), Point(x=36, y=332), Point(x=333, y=333),
Point(x=332, y=36)])]
--------------------------------------------------
http://www.netkiller.com
```

3.4. 从摄像头识别二维码

```
import cv2
import pyzbar.pyzbar as pyzbar


def decodeQrcode(image):
    barcodes = pyzbar.decode(image)
    for barcode in barcodes:
        # 提取二维码数据
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type
        # 打印调试信息
        print("[INFO] Found {}: {}".format(barcodeType,
barcodeData))

        # 取出二维码在图像中的位置
        (x, y, w, h) = barcode.rect
        cv2.rectangle(image, (x, y), (x + w, y + h),
                      (255, 255, 255), cv2.BORDER_DEFAULT)

        # 框出图像中的二维码
        text = "{} ({})".format(barcodeData, barcodeType)
        cv2.putText(image, text, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX,
                    .5, (225, 225, 225), 2)
    return image


if __name__ == '__main__':
    # 默认摄像头是0，如果读取不到，改为1试试
    camera = cv2.VideoCapture(1)
    while True:
        # 从摄像头读取当前帧
        ret, frame = camera.read()
        # 转为灰度图像，转递给解码函数
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        im = decodeQrcode(gray)
        cv2.waitKey(5)
        cv2.imshow("camera", im)
        # 设置退出按键，按下q跳出本次循环
        if cv2.waitKey(5) & 0xFF == ord('q'):
            break
    camera.release()
```

```
    cv2.destroyAllWindows()
```

4. graphviz

4.1. 安装 graphviz 环境

安装 graphviz 命令

```
brew install graphviz
```

安装 graphviz 包

```
pip install graphviz
```

4.2. 例子

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from graphviz import Digraph

A = [('Yellow', 'ink'), ('blue', 'ink'), ('pink', 'ink')]

e = Digraph()
e.attr(rankdir='LR')
for a in A:
        e.node(a[0])
        e.edge(a[0], a[1])

e.view()
```

第 27 章 3rdparty toolkit

1. ZeroRPC

http://zerorpc.dotcloud.com/

Zeromq 是基于zeromq、gevent和 msgpack开发的分布式RPC框架.

2. 表情符号 emoji

```
import emoji
result = emoji.emojize('Python is :thumbs_up:')
print(result)
# 'Python is  &#x27;
<br/># You can also reverse this:
result = emoji.demojize('Python is  ')
print(result)<br/># 'Python is :thumbs_up:&#x27;
```

第 28 章 图形开发

1. SVG 图形库

安装

```
pip3 install drawsvg -i
https://pypi.tuna.tsinghua.edu.cn/simple
pip3 install cairosvg -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

绘制多边形

```
lines = draw.Lines(
    # 坐标
    5, 5,
    # 横线
    200, 5,
    # 竖线
    200, 40,
    # 斜线
    200 - 10, 20,
    # 横线2
    5 + 10, 20,
    # 斜线
    5, 40,
    # 闭合竖线
    5,5,
    fill='none', stroke='black')
```

SVG 事件

```
<!DOCTYPE html>
<html>
<body>
<svg width="500" height="150">

<rect x="10" y="10" width="100" height="40"
   style="stroke: black; fill: silver; fill-opacity: .4;"
   onmouseover="this.style.stroke = 'blue'; this.style['stroke-
width'] = 5;"
   onmouseout="this.style.stroke = 'green'; this.style['stroke-
width'] = 1;"
   onclick="this.style['width'] = 300;" />

</svg>
</body>
</html>
```

```
<!DOCTYPE html>
<html>

<body>
   <input id="aa" type="text" value="200"
onclick="svg.style['width']=this.value;" />
   <svg width="500" height="150">


      <rect id='svg' x="10" y="10" width="100" height="40"
style="stroke: black; fill: silver; fill-opacity: .4;"
         onmouseover="this.style.stroke = 'blue';
this.style['stroke-width'] = 5;"
         onmouseout="this.style.stroke = 'green';
this.style['stroke-width'] = 1;"
         onclick="this.style['width'] = 300; aa.value='300'" />

   </svg>
</body>

</html>
```

2. PIL

```
from PIL import Image
from PIL import ImageChops
def compare_images(one, two, diff):
  image_one = Image.open(one)
  image_two = Image.open(two)
  try:
    diff = ImageChops.difference(image_one, image_two)
    if diff.getbbox() is None:
      print("【+】We are the same!")
    else:
      diff.save(diff)
  except ValueError as e:
    text = ("Pastes another image into this image."
        "The box argument is either a 2-tuple giving the
upper left corner, a 4-tuple defining the left, upper, "
        "right, and lower pixel coordinate, or None (same as
(0, 0)). If a 4-tuple is given, the size of the pasted "
        "image must match the size of the region.")
    print("【{0}】{1}".format(e,text))
if __name__ == '__main__':
  compare_images('1.png', '2.png', '我们不一样.png')
```

3. 二维码

qrcode

安装

pip install qrcode

```
neo@MacBook-Pro-Neo ~ % pip install qrcode
Collecting qrcode
  Downloading qrcode-6.1-py2.py3-none-any.whl (31 kB)
Requirement already satisfied: six in
/usr/local/lib/python3.9/site-packages (from qrcode) (1.15.0)
Installing collected packages: qrcode
Successfully installed qrcode-6.1
```

常规二维码

```
import qrcode

img = qrcode.make('《Netkiller Python 手札》')
img.save("text.png")

img = qrcode.make('http://www.netkiller.cn')
img.save("url.png")
```

设置颜色

颜色设置背景为白色，前景为绿色

```
import qrcode as qrcode
qr = qrcode.QRCode(version=1, box_size=10, border=2)
# 向二维码添加数据
qr.add_data("http://www.netkiller.cn")
qr.make(fit=True)

# 更改QR的背景为白色和绘画颜色为绿色
img = qr.make_image(fill_color="green", back_color="white")
img.save('green.png')
img.show()  # 显示二维码
```

QRCode 参数

version: 范围为1到40整数（最小值是1，表示12×12的矩阵），如果想让程序自动生成，将值设置为 None 并使用 fit=True 参数即可。
error_correction: 二维码的纠错范围，可以选择4个常量:
1. ERROR_CORRECT_L 7%以下的错误会被纠正
2. ERROR_CORRECT_M (default) 15%以下的错误会被纠正
3. ERROR_CORRECT_Q 25 %以下的错误会被纠正
4. ERROR_CORRECT_H. 30%以下的错误会被纠正
boxsize: 每个点（方块）中的像素个数
border: 二维码图像外围边框宽度，默认为4

qr - script to create QR codes at the command line

```
NAME
       qr - script to create QR codes at the command line

SYNOPSIS
       qr  [--help]  [--factory=FACTORY]  [--optimize=OPTIMIZE]
[--error-correction=LEVEL]
```

```
       [data]

DESCRIPTION
       This script uses the python qrcode module. It can take
data from stdin or from  the
       commandline  and  generate a QR code.  Normally it will
output the QR code as ascii
       art to the terminal. If the output is piped to a file,
it  will  output  the  image
       (default type of PNG).

OPTIONS
        -h, --help
           Show a help message.


        --factory=FACTORY
           Full  python  path to the image factory class to
create the image with. You can
           use the  following  shortcuts  to  the  built-in
image  factory  classes:  pil
           (default), pymaging, svg, svg-fragment, svg-path.


        --optimize=OPTIMIZE
           Optimize  the  data by looking for chunks of at
least this many characters that
           could use a more efficient encoding method. Use 0 to
turn off  chunk  optimiza-
           tion.


        --error-correction=LEVEL
           The  error  correction  level  to  use. Choices are
L (7%), M (15%, default), Q
           (25%), and H (30%).


        data
           The data from which the QR code will be generated.


SEE ALSO
       https://github.com/lincolnloop/python-qrcode/
```

使用方法

```
neo@MacBook-Pro-Neo ~ % qr "Some text" > test.png
```

MyQR

安装依赖包

```
pip install myqr
```

```
from MyQR import myqr
# myqr.run 参数:
# words: 文本/链接, 或者你想说的话(不支持中文, 很不友好)
# picture: 背景图
# colorsize: True 表示生成彩图
# save_name: 生成的二维码文件名
myqr.run(words="http://www.netkiller.com",
         picture="db.png", colorized=True,
         save_name="netkiller.png")
```

从图片识别二维码

安装依赖包

```
pip install pyzbar
```

pyzbar 是调用 zbar 共享库，所以还需要安装 zbar

```
brew install zbar
```

```
import numpy as np
from PIL import Image
import pyzbar.pyzbar as pyzbar

# 读取文件，转成数组
im = np.array(Image.open("netkiller.png"))
print(pyzbar.decode(im))
# 返回的信息
print('-' * 50)
# 读取内容
print(pyzbar.decode(im)[0].data.decode("utf-8"))
```

输出信息

```
[Decoded(data=b'http://www.netkiller.com', type='QRCODE',
rect=Rect(left=36, top=36, width=297, height=297), polygon=
[Point(x=36, y=36), Point(x=36, y=332), Point(x=333, y=333),
Point(x=332, y=36)])]
--------------------------------------------------
http://www.netkiller.com
```

从摄像头识别二维码

```
import cv2
import pyzbar.pyzbar as pyzbar


def decodeQrcode(image):
    barcodes = pyzbar.decode(image)
    for barcode in barcodes:
        # 提取二维码数据
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type
        # 打印调试信息
        print("[INFO] Found {}: {}".format(barcodeType,
barcodeData))

        # 取出二维码在图像中的位置
        (x, y, w, h) = barcode.rect
        cv2.rectangle(image, (x, y), (x + w, y + h),
                      (255, 255, 255), cv2.BORDER_DEFAULT)

        # 框出图像中的二维码
        text = "{} ({})".format(barcodeData, barcodeType)
        cv2.putText(image, text, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX,
                    .5, (225, 225, 225), 2)
    return image


if __name__ == '__main__':
    # 默认摄像头是0，如果读取不到，改为1试试
    camera = cv2.VideoCapture(1)
    while True:
        # 从摄像头读取当前帧
        ret, frame = camera.read()
        # 转为灰度图像，转递给解码函数
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        im = decodeQrcode(gray)
        cv2.waitKey(5)
        cv2.imshow("camera", im)
        # 设置退出按键，按下q跳出本次循环
```

```
        if cv2.waitKey(5) & 0xFF == ord('q'):
            break
    camera.release()
    cv2.destroyAllWindows()
```

4. graphviz

安装 graphviz 环境

安装 graphviz 命令

```
brew install graphviz
```

安装 graphviz 包

```
pip install graphviz
```

例子

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from graphviz import Digraph

A = [('Yellow', 'ink'), ('blue', 'ink'), ('pink', 'ink')]

e = Digraph()
e.attr(rankdir='LR')
for a in A:
        e.node(a[0])
        e.edge(a[0], a[1])

e.view()
```

3. Markdown

3.1. 安装

第 29 章 实用代码

1. 随机生成姓名

为什么要生成姓名?

在做测试的时候，例如压力测试和性能测试，我们要批量造一批数据

新系统上线，平台刚刚起步，内容不能空空如野，所以需要做一些数据

此前我们完全使用随机字符串，这样的数据太假，也不好看。

所以我们需要做一个程序自动生成姓名。

随机生成姓名的原理

首先，我们需要百家姓，下载一个百家姓表，然后做成数组

然后，随机从数组内抽取一个姓氏

接着，产生名字，中国人多是两个字或三个字的名字。名字可以从GB2312字库中提取

最后，拼接姓和名字

源码

例 29.1. 随机生成姓名

```
import random
```

```
def lastname():
    name = ["赵", "钱", "孙", "李", "周", "吴", "郑", "王",
"冯", "陈", "褚", "卫", "蒋", "沈", "韩", "杨", "朱", "秦", "尤",
             "许", "何", "吕", "施", "张", "孔", "曹", "严",
"华", "金", "魏", "陶", "姜", "戚", "谢", "邹", "喻", "柏", "水",
             "窦", "章", "云", "苏", "潘", "葛", "奚", "范",
"彭", "郎", "鲁", "韦", "昌", "马", "苗", "凤", "花", "方", "俞",
             "任", "袁", "柳", "酆", "鲍", "史", "唐", "费",
"廉", "岑", "薛", "雷", "贺", "倪", "汤", "滕", "殷", "罗", "毕",
             "郝", "邬", "安", "常", "乐", "于", "时", "傅",
"皮", "卞", "齐", "康", "伍", "余", "元", "卜", "顾", "孟", "平",
             "黄", "和", "穆", "萧", "尹", "姚", "邵", "湛",
"汪", "祁", "毛", "禹", "狄", "米", "贝", "明", "臧", "计", "伏",
             "成", "戴", "谈", "宋", "茅", "庞", "熊", "纪",
"舒", "屈", "项", "祝", "董", "梁", "杜", "阮", "蓝", "闵", "席",
             "季", "麻", "强", "贾", "路", "娄", "危", "江",
"童", "颜", "郭", "梅", "盛", "林", "刁", "钟", "徐", "邱", "骆",
             "高", "夏", "蔡", "田", "樊", "胡", "凌", "霍",
"虞", "万", "支", "柯", "昝", "管", "卢", "莫", "经", "房", "裘",
             "缪", "干", "解", "应", "宗", "丁", "宣", "贲",
"邓", "郁", "单", "杭", "洪", "包", "诸", "左", "石", "崔", "吉",
             "钮", "龚", "程", "嵇", "邢", "滑", "裴", "陆",
"荣", "翁", "荀", "羊", "于", "惠", "甄", "曲", "家", "封", "芮",
             "羿", "储", "靳", "汲", "邴", "糜", "松", "井",
"段", "富", "巫", "乌", "焦", "巴", "弓", "牧", "隗", "山", "谷",
             "车", "侯", "宓", "蓬", "全", "郗", "班", "仰",
"秋", "仲", "伊", "宫", "宁", "仇", "栾", "暴", "甘", "钭", "厉",
             "戎", "祖", "武", "符", "刘", "景", "詹", "束",
"龙", "叶", "幸", "司", "韶", "郜", "黎", "蓟", "溥", "印", "宿",
             "白", "怀", "蒲", "邰", "从", "鄂", "索", "咸",
"籍", "赖", "卓", "蔺", "屠", "蒙", "池", "乔", "阴", "郁", "胥",
             "能", "苍", "双", "闻", "莘", "党", "翟", "谭",
"贡", "劳", "逄", "姬", "申", "扶", "堵", "冉", "宰", "郦", "雍",
             "却", "璩", "桑", "桂", "濮", "牛", "寿", "通",
"边", "扈", "燕", "冀", "浦", "尚", "农", "温", "别", "庄", "晏",
             "柴", "瞿", "阎", "充", "慕", "连", "茹", "习",
"宦", "艾", "鱼", "容", "向", "古", "易", "慎", "戈", "廖", "庾",
             "终", "暨", "居", "衡", "步", "都", "耿", "满",
"弘", "匡", "国", "文", "寇", "广", "禄", "阙", "东", "欧", "殳",
             "沃", "利", "蔚", "越", "夔", "隆", "师", "巩",
"厍", "聂", "晁", "勾", "敖", "融", "冷", "訾", "辛", "阚", "那",
             "简", "饶", "空", "曾", "毋", "沙", "乜", "养",
"鞠", "须", "丰", "巢", "关", "蒯", "相", "查", "后", "荆", "红",
             "游", "郏", "竺", "权", "逯", "盖", "益", "桓",
```

```
"公", "仉", "督", "岳", "帅", "缑", "亢", "况", "郈", "有", "琴",
             "归", "海", "晋", "楚", "闫", "法", "汝", "鄢",
"涂", "钦", "商", "牟", "佘", "佴", "伯", "赏", "墨", "哈", "谯",
             "篁", "年", "爱", "阳", "佟", "言", "福", "南",
"火", "铁", "迟", "漆", "官", "冼", "真", "展", "繁", "檀", "祭",
             "密", "敬", "揭", "舜", "楼", "疏", "冒", "浑",
"挚", "胶", "随", "高", "皋", "原", "种", "练", "弥", "仓", "眭",
             "蹇", "覃", "阿", "门", "恽", "来", "綦", "召",
"仪", "风", "介", "巨", "木", "京", "狐", "郇", "虎", "枚", "抗",
             "达", "杞", "苌", "折", "麦", "庆", "过", "竹",
"端", "鲜", "皇", "亓", "老", "是", "秘", "畅", "邝", "还", "宾",
             "闾", "辜", "纵", "龠", "万俟", "司马", "上官", "欧
阳", "夏侯", "诸葛", "闻人", "东方", "赫连", "皇甫", "羊舌",
             "尉迟", "公羊", "澹台", "公冶", "宗正", "濮阳", "淳
于", "单于", "太叔", "申屠", "公孙", "仲孙", "轩辕", "令狐",
             "钟离", "宇文", "长孙", "慕容", "鲜于", "闾丘", "司
徒", "司空", "兀官", "司寇", "南门", "呼延", "子车", "颛孙",
             "端木", "巫马", "公西", "漆雕", "车正", "壤驷", "公
良", "拓跋", "夹谷", "宰父", "谷梁", "段干", "百里", "东郭",
             "微生", "梁丘", "左丘", "东门", "西门", "南宫", "第
五", "公仪", "公乘", "太史", "仲长", "叔孙", "屈突", "尔朱",
             "东乡", "相里", "胡母", "司城", "张廖", "雍门", "毋
丘", "贺兰", "綦毋", "屋庐", "独孤", "南郭", "北宫", "王孙"]
    return (random.choice(name))


def firstname():
    # name = GBK2312()+GBK2312()
    name = GBK2312()
    if random.choice([True, False]) == True:
        name = name + GBK2312()
    return(name)


def Unicode():
    character = random.randint(0x4e00, 0x9fbf)
    return chr(character)


def GBK2312():
    head = random.randint(0xb0, 0xf7)
    body = random.randint(0xa1, 0xfe)
    val = f'{head:x} {body:x}'
    character = bytes.fromhex(val).decode('gb2312')
    return character
```

```
if __name__ == '__main__':
    for n in range(10):
        print(lastname()+firstname())
```

词库生成姓名

由于随机生成姓名比较生硬，再次之上经过优化，增加了自定义词库。我们90后00后常用取名用字添加到词库中。

例如：

梓，馨，宇，轩，函……

'泓遥','辰祥','祁文','诗梦','秋云',‘卿妤'……

例 29.2. 词库生成姓名

```
import random


def lastname():
    name = ["赵", "钱", "孙", "李", "周", "吴", "郑", "王",
"冯", "陈", "褚", "卫", "蒋", "沈", "韩", "杨", "朱", "秦", "尤",
             "许", "何", "吕", "施", "张", "孔", "曹", "严",
"华", "金", "魏", "陶", "姜", "戚", "谢", "邹", "喻", "柏", "水",
             "窦", "章", "云", "苏", "潘", "葛", "奚", "范",
"彭", "郎", "鲁", "韦", "昌", "马", "苗", "凤", "花", "方", "俞",
             "任", "袁", "柳", "酆", "鲍", "史", "唐", "费",
"廉", "岑", "薛", "雷", "贺", "倪", "汤", "滕", "殷", "罗", "毕",
             "郝", "邬", "安", "常", "乐", "于", "时", "傅",
"皮", "卞", "齐", "康", "伍", "余", "元", "卜", "顾", "孟", "平",
             "黄", "和", "穆", "萧", "尹", "姚", "邵", "湛",
"汪", "祁", "毛", "禹", "狄", "米", "贝", "明", "臧", "计", "伏",
             "成", "戴", "谈", "宋", "茅", "庞", "熊", "纪",
```

"舒", "屈", "项", "祝", "董", "梁", "杜", "阮", "蓝", "闵", "席",
"季", "麻", "强", "贾", "路", "娄", "危", "江",
"童", "颜", "郭", "梅", "盛", "林", "刁", "钟", "徐", "邱", "骆",
"高", "夏", "蔡", "田", "樊", "胡", "凌", "霍",
"虞", "万", "支", "柯", "昝", "管", "卢", "莫", "经", "房", "裘",
"缪", "干", "解", "应", "宗", "丁", "宣", "贲",
"邓", "郁", "单", "杭", "洪", "包", "诸", "左", "石", "崔", "吉",
"钮", "龚", "程", "嵇", "邢", "滑", "裴", "陆",
"荣", "翁", "荀", "羊", "于", "惠", "甄", "曲", "家", "封", "芮",
"羿", "储", "靳", "汲", "邴", "糜", "松", "井",
"段", "富", "巫", "乌", "焦", "巴", "弓", "牧", "隗", "山", "谷",
"车", "侯", "宓", "蓬", "全", "郗", "班", "仰",
"秋", "仲", "伊", "宫", "宁", "仇", "栾", "暴", "甘", "钭", "厉",
"戎", "祖", "武", "符", "刘", "景", "詹", "束",
"龙", "叶", "幸", "司", "韶", "郜", "黎", "蓟", "薄", "印", "宿",
"白", "怀", "蒲", "邰", "从", "鄂", "索", "咸",
"籍", "赖", "卓", "蔺", "屠", "蒙", "池", "乔", "阴", "郁", "胥",
"能", "苍", "双", "闻", "莘", "党", "翟", "谭",
"贡", "劳", "逄", "姬", "申", "扶", "堵", "冉", "宰", "郦", "雍",
"却", "璩", "桑", "桂", "濮", "牛", "寿", "通",
"边", "扈", "燕", "冀", "浦", "尚", "农", "温", "别", "庄", "晏",
"柴", "瞿", "阎", "充", "慕", "连", "茹", "习",
"宦", "艾", "鱼", "容", "向", "古", "易", "慎", "戈", "廖", "庾",
"终", "暨", "居", "衡", "步", "都", "耿", "满",
"弘", "匡", "国", "文", "寇", "广", "禄", "阙", "东", "欧", "殳",
"沃", "利", "蔚", "越", "夔", "隆", "师", "巩",
"厍", "聂", "晁", "勾", "敖", "融", "冷", "訾", "辛", "阚", "那",
"简", "饶", "空", "曾", "毋", "沙", "乜", "养",
"鞠", "须", "丰", "巢", "关", "蒯", "相", "查", "后", "荆", "红",
"游", "郏", "竺", "权", "逯", "盖", "益", "桓",
"公", "仉", "督", "岳", "帅", "缑", "亢", "况", "郈", "有", "琴",
"归", "海", "晋", "楚", "闫", "法", "汝", "鄢",
"涂", "钦", "商", "牟", "佘", "佴", "伯", "赏", "墨", "哈", "谯",
"篁", "年", "爱", "阳", "佟", "言", "福", "南",
"火", "铁", "迟", "漆", "官", "冼", "真", "展", "繁", "檀", "祭",
"密", "敬", "揭", "舜", "楼", "疏", "冒", "浑",
"挚", "胶", "随", "高", "皋", "原", "种", "练", "弥", "仓", "眭",
"蹇", "覃", "阿", "门", "恽", "来", "綦", "召",
"仪", "风", "介", "巨", "木", "京", "狐", "郇", "虎", "枚", "抗",
"达", "杞", "苌", "折", "麦", "庆", "过", "竹",
"端", "鲜", "皇", "亓", "老", "是", "秘", "畅", "邝", "还", "宾",
"闾", "辜", "纵", "俞", "万俟", "司马", "上官", "欧
阳", "夏侯", "诸葛", "闻人", "东方", "赫连", "皇甫", "羊舌",
"尉迟", "公羊", "澹台", "公冶", "宗正", "濮阳", "淳
于", "单于", "太叔", "申屠", "公孙", "仲孙", "轩辕", "令狐",

```
            "钟离", "宇文", "长孙", "慕容", "鲜于", "闾丘", "司
徒", "司空", "兀官", "司寇", "南门", "呼延", "子车", "颛孙",
            "端木", "巫马", "公西", "漆雕", "车正", "壤驷", "公
良", "拓跋", "夹谷", "宰父", "谷梁", "段干", "百里", "东郭",
            "微生", "梁丘", "左丘", "东门", "西门", "南宫", "第
五", "公仪", "公乘", "太史", "仲长", "叔孙", "屈突", "尔朱",
            "东乡", "相里", "胡母", "司城", "张廖", "雍门", "毋
丘", "贺兰", "綦毋", "屋庐", "独孤", "南郭", "北宫", "王孙"]
    return (random.choice(name))


def firstname():
    # name = GBK2312()+GBK2312()
    name = GBK2312()
    if random.choice([True, False]) == True:
        name = name + GBK2312()
    return(name)


def Unicode():
    character = random.randint(0x4e00, 0x9fbf)
    return chr(character)


def GBK2312():
    head = random.randint(0xb0, 0xf7)
    body = random.randint(0xa1, 0xfe)
    val = f'{head:x} {body:x}'
    character = bytes.fromhex(val).decode('gb2312')
    return character


def popular():
    single = [
        '浩', '权', '博', '博', '皓', '琛', '钦', '然',
        '会', '群', '子', '栩', '童', '倩', '睿', '炜',
        '茵', '嘉', '祺', '顺', '雨', '欣', '清', '华',
        '广', '兴', '海', '尘', '沐', '智', '勝', '雅',
        '熙', '虎', '跃', '啸', '靖', '芸', '夏', '箐',
        '启', '谊', '佳', '溪', '羲', '杰', '炜', '殷',
        '润', '潼', '宇', '涵', '闰', '菲', '琳', '晋',
        '玉', '婷', '贻', '淇', '思', '杰', '雅', '琪',
        '家', '伟', '嘉', '娟', '坤', '鸿', '林', '卓',
        '皓', '宇', '懿', '翀', '懿', '辰', '亦', '益',
        '宾', '景', '晴', '昱', '民', '奕', '保', '杰']
```

```
    double = ['语溪', '瑶薇', '倩汐', '万媛', '兮榆', '恬忆',
              '若洛', '梓萱', '绿茜', '嫣倩', '含媛', '雨妤',
              '甜悠', '雅珞', '云陶', '攸竹', '皎盈', '盈盈',
              '隅静', '歌彤', '翔轩', '茂俊', '子棋', '宸斯',
              '以菱', '梓姝', '梦儿', '可浩', '逸然', '泰析',
              '桓萁', '瀚宇', '海佑', '洁永', '以添', '梓艺',
              '代灏', '宇锦', '添宏', '歆玥', '芳华', '静松',
              '慧娉', '琳玉', '鸿培', '明国', '鹏澄', '文轩',
              '萧宏', '若栋', '俊柯', '寒沙', '宏弈', '坤颐',
              '景明', '诺琪', '妍琼', '羿静', '文婧', '佳可',
              '雪春', '惠思', '芳雯', '薇蔷', '清妍', '慧灵',
              '粟淮', '锦宥', '彬彬', '宣淑', '海秀', '雨灵',
              '苕溪', '樱蝶', '月婵', '青青', '安涛', '泓佳',
              '华姝', '彤颖', '希珍', '渊杰', '仁傲', '钰仁',
              '钲洋', '荣程', '予名', '宇均', '昱晖', '昌琼',
              '铭立', '仁凡', '佑棋', '羽乐', '胤鹏', '博飞',
              '淇若', '柯宏', '鉴添', '莫涵', '舒闵', '宏旭',
              '泓遥', '辰祥', '祁文', '诗梦', '秋云', '卿妤'
              ]
    name = lastname()
    if random.choice([True, False]) == True:
        name += random.choice(double)
    else:
        name += random.choice(single)
    return(name)


if __name__ == '__main__':
    for n in range(10):
        print(popular())
```

2. 心知天气

https://www.seniverse.com/dashboard

```
import requests, time, hmac, hashlib, base64
from urllib import parse

UID = "Pi62K5jkMsbsrfvBW"  # 用户ID
KEY = "SU8HCiR5slL8RvBMj"  # API key

# UID = "U785B76FC9"  # 用户ID
# KEY = "4r9bergjetiv1tsd"  # API key

LOCATION = "Huayin"  # 所查询的位置，可以使用城市拼音、v3 ID、经纬度等
API = "https://api.seniverse.com/v3/weather/now.json"  # API URL，可替换为其他 URL
UNIT = "c"  # 单位
LANGUAGE = "zh-Hans"  # 查询结果的返回语言


def getJsonpUrl():
    ts = int(time.time())  # 当前时间戳
    text = "ts={ts}&ttl=300&uid={uid}".format(ts=ts, uid=UID)
    key = bytes(KEY, "UTF-8")
    raw = bytes(text, "UTF-8")

    # 使用 HMAC-SHA1 方式，以 API 密钥（key）对上一步生成的参数字符串（raw）进行加密
    digest = hmac.new(key, raw, hashlib.sha1).digest()
    # 将上一步生成的加密结果用 base64 编码，并做一个 urlencode，得到签名sig
    signature = base64.b64encode(digest)
    params = {"location": LOCATION, "language": LANGUAGE, "unit": UNIT, "ts": ts, "ttl": "300", "uid": UID, "sig": signature}

    response = requests.get(API, params=params, timeout=10)

    if response.status_code == 200:
```

```
        result = response.json()["results"].pop()
    else:
        result = {}
    return result


def fetchWeather():
    params = {"key": KEY, "location": LOCATION, "language":
LANGUAGE, "unit": UNIT}
    response = requests.get(API, params=params, timeout=10)
    if response.status_code == 200:
        result = response.json()["results"].pop()
    else:
        result = {}
    return result


if __name__ == "__main__":
    print(fetchWeather())
    print("-" * 50)
    print(getJsonpUrl())
```

第 30 章 FAQ

1. ImportError: No module named 'zlib'

问题：编译安装的python3.4.2使用中提示 ImportError: No module named 'zlib'

原因：这个原因是你没有安装zlib

解决方法：安装 zlib-devel 后重新编译安装

```
yum install zlib-devel

./configure --prefix=/srv/python-3.4.2
make -j8
make install
```

2. UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb2 in position 679: invalid start byte

解决方法指定编码encoding='utf-8', errors='ignore'

```
with open(cfgfile+'.in','r', encoding='utf-8', errors='ignore')
as f:
     for line in f.readlines():
        pring(line)
```

3. ERROR: Could not install packages due to an EnvironmentError: [Errno 28] No space left on device

```
pip install -b /your_path netkiller-devops
```