# 目录

[1. 目录 1](#_Toc25953174)

[2. python基础知识 9](#_Toc25953175)

[2.1. 基础与解答 9](#_Toc25953176)

[2.2. 迭代器 10](#_Toc25953177)

[2.2.1. 基础 10](#_Toc25953178)

[2.2.2. 举例 12](#_Toc25953179)

[2.2.3. \_\_iter\_\_ 类想被用于for ... in循环 12](#_Toc25953180)

[2.2.4. functools.wraps 13](#_Toc25953181)

[2.3. 生成器 13](#_Toc25953182)

[2.4. 描述器 descriptor 15](#_Toc25953183)

[2.5. 装饰器 16](#_Toc25953184)

[2.5.1. 无参数 16](#_Toc25953185)

[2.5.2. 带参数 17](#_Toc25953186)

[2.5.3. 参数数量不确定 17](#_Toc25953187)

[2.5.4. 让装饰器带参数 18](#_Toc25953188)

[2.5.5. 让装饰器带 类 参数 18](#_Toc25953189)

[2.6. TCP编程 19](#_Toc25953190)

[2.6.1. 客户端 19](#_Toc25953191)

[2.6.2. 服务器 21](#_Toc25953192)

[2.7. UDP编程 23](#_Toc25953193)

[2.8. 动态变量 24](#_Toc25953194)

[2.9. 进程和线程 25](#_Toc25953195)

[2.9.1. ThreadLocal 25](#_Toc25953196)

[2.9.2. 多进程 27](#_Toc25953197)

[2.9.3. 多线程 28](#_Toc25953198)

[2.10. WebSocket协议分析 28](#_Toc25953199)

[2.10.1. 协议概览 28](#_Toc25953200)

[2.10.2. 数据帧 29](#_Toc25953201)

[2.10.3. 消息分片 31](#_Toc25953202)

[2.11. python后台框架 31](#_Toc25953203)

[2.12. python其他 32](#_Toc25953204)

[3. python函数 33](#_Toc25953205)

[3.1. hasattr() 33](#_Toc25953206)

[3.2. getattr() 33](#_Toc25953207)

[3.3. setattr() 33](#_Toc25953208)

[3.4. delattr() 33](#_Toc25953209)

[3.5. dir() 34](#_Toc25953210)

[3.6. repr() 34](#_Toc25953211)

[3.7. 闭包 34](#_Toc25953212)

[3.8. 递归函数 35](#_Toc25953213)

[4. python模块编写 36](#_Toc25953214)

[4.1. setup.py文件的编写 36](#_Toc25953215)

[4.2. 模块包中\_\_init\_\_.py作用 36](#_Toc25953216)

[4.3. 模块中导入的模块 37](#_Toc25953217)

[4.4. 常用模块列表 38](#_Toc25953218)

[4.5. 问题解决 41](#_Toc25953219)

[4.6. 细节 41](#_Toc25953220)

[5. python模块 42](#_Toc25953221)

[5.1. 常用模块介绍 42](#_Toc25953222)

[5.2. 有关多线程多进程 45](#_Toc25953223)

[5.2.1. Queue模块 45](#_Toc25953224)

[5.2.2. threading模块 49](#_Toc25953225)

[5.2.3. multiprocessing模块 59](#_Toc25953226)

[5.2.4. gevent 异步 并发 60](#_Toc25953227)

[5.2.5. thread 多线程控制 60](#_Toc25953228)

[5.3. 有关高级数据结构 61](#_Toc25953229)

[5.3.1. collections 提供额外的数据类型 61](#_Toc25953230)

[5.3.2. heapq 堆队列 有序的list 排序算法用 63](#_Toc25953231)

[5.3.3. bisect 能够提供保持list元素序列的支持 66](#_Toc25953232)

[5.3.4. array 类似list类型 66](#_Toc25953233)

[5.3.5. itertools 创建迭代器 67](#_Toc25953234)

[5.4. 有关数据处理 70](#_Toc25953235)

[5.4.1. StringIO和BytesIO 字符串像文件一样读取 70](#_Toc25953236)

[5.4.2. HTMLParser模块 解析html 72](#_Toc25953237)

[5.4.3. pprint模块 输出的格式化 73](#_Toc25953238)

[5.4.4. random模块 生成随机数 74](#_Toc25953239)

[5.4.5. math模块 75](#_Toc25953240)

[5.4.6. pickle模块 保存对象到文件 数据序列和反序列化 75](#_Toc25953241)

[5.4.7. struct模块 解决str和其他二进制数据类型的转换 77](#_Toc25953242)

[5.4.8. inspect模块 对是否是模块，框架，函数等进行类型检查。 78](#_Toc25953243)

[5.4.9. fileinput 模块 遍历文本文件 79](#_Toc25953244)

[5.4.10. uniout模块 打印可读的字符，而不是转义的字符串。 79](#_Toc25953245)

[5.4.11. xpinyin 一个用于把汉字转换为拼音的库。 80](#_Toc25953246)

[5.4.12. unidecode模块 Unicode 文本的 ASCII 转换形式 80](#_Toc25953247)

[5.4.13. pangu模块 在中日韩语字符和数字字母之间添加空格 81](#_Toc25953248)

[5.4.14. ftfy模块 让Unicode文本更完整更连贯。 81](#_Toc25953249)

[5.4.15. schema 数据校验 81](#_Toc25953250)

[5.4.16. PyH模块 生成HTML 82](#_Toc25953251)

[5.5. 有关嵌入式数据库 83](#_Toc25953252)

[5.5.1. pickledb 一个简单，轻量级键值储存数据库 83](#_Toc25953253)

[5.5.2. tinydb 纯python编写的文档型数据库 83](#_Toc25953254)

[5.5.3. anydbm, shelve 纯python数据库模块 84](#_Toc25953255)

[5.5.4. sqlite3模块 85](#_Toc25953256)

[5.6. 有关ORM 87](#_Toc25953257)

[5.6.1. peewee 87](#_Toc25953258)

[5.6.2. SQLAlchemy 91](#_Toc25953259)

[5.7. 有关文件比较 94](#_Toc25953260)

[5.7.1. filecmp模块 比较文件及文件夹的内容 94](#_Toc25953261)

[5.7.2. difflib模块 95](#_Toc25953262)

[5.7.3. Levenshtein模块 96](#_Toc25953263)

[5.7.4. fuzzywuzzy 模糊字符串匹配 97](#_Toc25953264)

[5.7.5. distance模块 97](#_Toc25953265)

[5.7.6. 比较两字符串相似度 97](#_Toc25953266)

[5.8. 有关自然语言处理 98](#_Toc25953267)

[5.8.1. jieba模块 中文分词 98](#_Toc25953268)

[5.8.2. Snownlp模块 用来处理中文文本的库 100](#_Toc25953269)

[5.8.3. tgrocery模块 101](#_Toc25953270)

[5.9. 有关网络及网络协议 102](#_Toc25953271)

[5.9.1. select模块 102](#_Toc25953272)

[5.9.2. socket模块 102](#_Toc25953273)

[5.9.3. furl模块 url处理 103](#_Toc25953274)

[5.10. 有关命令行参数 105](#_Toc25953275)

[5.10.1. sys.argv 105](#_Toc25953276)

[5.10.2. argparse模块 105](#_Toc25953277)

[5.10.3. getopt模块 比如-i -o 107](#_Toc25953278)

[5.10.4. docopt模块 108](#_Toc25953279)

[5.10.5. optparse模块 110](#_Toc25953280)

[5.11. 有关优化 110](#_Toc25953281)

[5.11.1. 简单计时器 装饰器方式 111](#_Toc25953282)

[5.11.2. 使用上下文管理器进行及时 111](#_Toc25953283)

[5.11.3. cProfile模块 111](#_Toc25953284)

[5.11.4. LineProfiler模块 113](#_Toc25953285)

[5.11.5. memory\_profiler模块 计算内存 114](#_Toc25953286)

[5.11.6. timeit模块 测量小代码片段的执行时间 114](#_Toc25953287)

[5.11.7. traceback模块 输出具体的异常结果 114](#_Toc25953288)

[5.11.8. pdb模块 115](#_Toc25953289)

[5.11.9. trace模块 115](#_Toc25953290)

[5.12. 有关日志 116](#_Toc25953291)

[5.12.1. logging模块 116](#_Toc25953292)

[5.12.2. logbook模块 – Logging 库的替代品 119](#_Toc25953293)

[5.13. 有关数据挖掘 119](#_Toc25953294)

[5.13.1. 机器学习 120](#_Toc25953295)

[5.13.2. 算法介绍 120](#_Toc25953296)

[5.13.3. 图graph 122](#_Toc25953297)

[5.13.4. scipy 125](#_Toc25953298)

[5.14. 有关配置文件 125](#_Toc25953299)

[5.14.1. ConfigParser模块 配置文件 125](#_Toc25953300)

[5.14.2. python-decouple模块 将设置和代码完全隔离 126](#_Toc25953301)

[5.15. 有关加密 127](#_Toc25953302)

[5.15.1. hashlib 摘要算法 存储密码 不可逆 127](#_Toc25953303)

[5.15.2. hmac 不可逆 130](#_Toc25953304)

[5.15.3. base64 可逆 130](#_Toc25953305)

[5.15.4. uuid模块 131](#_Toc25953306)

[5.15.5. Pycrypto模块 加解密算法 132](#_Toc25953307)

[5.16. 有关模板 133](#_Toc25953308)

[5.16.1. mako 133](#_Toc25953309)

[5.16.2. jinja2 134](#_Toc25953310)

[5.17. 有关流程图 134](#_Toc25953311)

[5.17.1. dot形式 graphviz 134](#_Toc25953312)

[5.17.2. pygraphviz模块 画流程图 137](#_Toc25953313)

[5.18. \_\_future\_\_模块 把下一个新版本的特性导入到当前版本 139](#_Toc25953314)

[5.19. 其他 140](#_Toc25953315)

[5.19.1. PySpider 140](#_Toc25953316)

[5.19.2. selenium 140](#_Toc25953317)

[5.19.3. jira模块 154](#_Toc25953318)

[5.19.4. hadoop 158](#_Toc25953319)

[5.19.5. zookeeper 165](#_Toc25953320)

[5.19.6. cookie处理 167](#_Toc25953321)

[5.19.7. NetworkX 网络拓扑计算 169](#_Toc25953322)

[5.19.8. watchdog --监控文件系统变化 170](#_Toc25953323)

[5.19.9. pyhook模块 监控鼠标和键盘 170](#_Toc25953324)

[5.19.10. PyUserInput 控制鼠标和键盘的模块 172](#_Toc25953325)

[5.19.11. xhtml2pdf模块 html生成pdf 173](#_Toc25953326)

[5.19.12. functools 173](#_Toc25953327)

[5.19.13. jPype 174](#_Toc25953328)

[6. flask框架 175](#_Toc25953329)

[6.1. 启动 175](#_Toc25953330)

[6.2. 安装 175](#_Toc25953331)

[6.3. 简单例子 176](#_Toc25953332)

[6.4. http方法 post 177](#_Toc25953333)

[6.5. 模板 178](#_Toc25953334)

[6.5.1. 模板用法 178](#_Toc25953335)

[6.5.2. 模板实例 179](#_Toc25953336)

[6.5.3. 传递多个参数给模版（template） 180](#_Toc25953337)

[6.5.4. css 180](#_Toc25953338)

[6.5.5. html 180](#_Toc25953339)

[6.6. 配置文件 181](#_Toc25953340)

[6.6.1. 基础 181](#_Toc25953341)

[6.6.2. 更新配置项 182](#_Toc25953342)

[6.7. 日志 182](#_Toc25953343)

[6.7.1. 显示在eclipse的console上 182](#_Toc25953344)

[6.7.2. 将日志写入文件中 182](#_Toc25953345)

[6.8. 与mongodb 183](#_Toc25953346)

[6.9. 与cmd交互 184](#_Toc25953347)

[6.10. 其他规则 184](#_Toc25953348)

[6.11. 问题解决 185](#_Toc25953349)

[7. web.py框架 185](#_Toc25953350)

[7.1. 基础 185](#_Toc25953351)

[7.2. 模板 186](#_Toc25953352)

[7.3. 细节 186](#_Toc25953353)

[8. bottle框架 186](#_Toc25953354)

[8.1. 基础 186](#_Toc25953355)

[8.2. 自带模板语法 187](#_Toc25953356)

[8.3. 举例 188](#_Toc25953357)

[8.4. 其他 188](#_Toc25953358)

[8.5. 使用beaker模块支持session 189](#_Toc25953359)

[8.6. 源码收获 190](#_Toc25953360)

[8.6.1. 框架 190](#_Toc25953361)

[8.6.2. 实际 191](#_Toc25953362)

[9. tornado框架 192](#_Toc25953363)

[9.1. 基础 192](#_Toc25953364)

[9.2. 模板 194](#_Toc25953365)

[9.3. cookie 194](#_Toc25953366)

[9.4. 使用Tornado的XSRF保护 195](#_Toc25953367)

[9.5. 异步web服务 196](#_Toc25953368)

[9.6. 其他 198](#_Toc25953369)

[9.7. windows上查看端口占用情况并清除 198](#_Toc25953370)

[9.8. 遇到的问题 199](#_Toc25953371)

[9.9. linux安装和启动 199](#_Toc25953372)

[10. MongoDB 199](#_Toc25953373)

[10.1. windows下MongoDB的安装及配置 199](#_Toc25953374)

[10.2. linux下安装mongodb 200](#_Toc25953375)

[10.2.1. 步骤 200](#_Toc25953376)

[10.2.2. 具体细节 200](#_Toc25953377)

[10.3. 使用mongodb命令 201](#_Toc25953378)

[10.4. 高级应用 201](#_Toc25953379)

[10.5. MongoVUE工具 202](#_Toc25953380)

[10.6. PyMongo 与mongo库交互 203](#_Toc25953381)

[11. redis 205](#_Toc25953382)

[11.1. redis基础 205](#_Toc25953383)

[11.2. redis命令 客户端 206](#_Toc25953384)

[11.3. redis 五种数据类型的使用场景 206](#_Toc25953385)

[11.3.1. String 206](#_Toc25953386)

[11.3.2. Hash 207](#_Toc25953387)

[11.3.3. list 208](#_Toc25953388)

[11.3.4. set 208](#_Toc25953389)

[11.3.5. Sort Set zset 209](#_Toc25953390)

[11.3.6. 消息订阅 Pub/Sub 209](#_Toc25953391)

[11.3.7. transaction 210](#_Toc25953392)

[11.4. python redis模块 210](#_Toc25953393)

[11.4.1. 基础 210](#_Toc25953394)

[11.4.2. 消息订阅 211](#_Toc25953395)

[11.4.3. pipeline 211](#_Toc25953396)

[11.5. linux下redis安装 212](#_Toc25953397)

[11.5.1. 整个流程 212](#_Toc25953398)

[11.5.2. 错误解决 215](#_Toc25953399)

[12. scrapy 215](#_Toc25953400)

[12.1. eclipse运行scrapy 215](#_Toc25953401)

[12.2. 简单例子 216](#_Toc25953402)

[12.2.1. 新建项目（Project） 216](#_Toc25953403)

[12.2.2. 明确目标（Item） 217](#_Toc25953404)

[12.2.3. 制作爬虫（Spider） 218](#_Toc25953405)

[12.3. scrapy+flask+mongodb方案 223](#_Toc25953406)

[12.3.1. 第一步：定义Model层 223](#_Toc25953407)

[12.3.2. 第二步：编写爬虫 223](#_Toc25953408)

[12.3.3. 第三步：将我们的MeiziItem存入数据库中 224](#_Toc25953409)

[12.3.4. 管道管理 224](#_Toc25953410)

[12.3.5. 建立flask 225](#_Toc25953411)

[12.4. CrawlSpider 自动爬取 226](#_Toc25953412)

[12.5. pymongo模块操作mongodb 226](#_Toc25953413)

[12.6. log日志 228](#_Toc25953414)

[12.7. 解决方案 228](#_Toc25953415)

[13. Xpath 229](#_Toc25953416)

[13.1. 基础 229](#_Toc25953417)

[13.2. 谓语（Predicates） 229](#_Toc25953418)

[13.3. 选取未知节点 230](#_Toc25953419)

[13.4. 选取若干路径 230](#_Toc25953420)

[13.5. XPath 运算符 231](#_Toc25953421)

[13.6. Xpath例子 231](#_Toc25953422)

[14. Tkinter 231](#_Toc25953423)

[14.1. 控件 232](#_Toc25953424)

[14.2. 控件实例 232](#_Toc25953425)

[14.3. 控件细节 237](#_Toc25953426)

[14.3.1. 键盘鼠标触发命令 bind 237](#_Toc25953427)

[14.3.2. 有关菜单menu 238](#_Toc25953428)

[14.3.3. 有关Entry 240](#_Toc25953429)

[14.3.4. 有关Radiobutton 240](#_Toc25953430)

[14.3.5. 有关Text 241](#_Toc25953431)

[14.3.6. 有关Checkbutton 242](#_Toc25953432)

[14.3.7. 有关button 244](#_Toc25953433)

[14.3.8. 滚动条 244](#_Toc25953434)

[14.3.9. 打开与保存 244](#_Toc25953435)

[14.3.10. 有关窗口显示 246](#_Toc25953436)

[14.3.11. 其他 247](#_Toc25953437)

[14.4. 使用class类来组织 251](#_Toc25953438)

[14.5. grid布局 251](#_Toc25953439)

[14.5.1. 基础 251](#_Toc25953440)

[14.5.2. 细节 252](#_Toc25953441)

[14.5.3. 有关迷你模式 253](#_Toc25953442)

[14.6. 例子 253](#_Toc25953443)

[14.6.1. 显示web上的图片 253](#_Toc25953444)

[14.6.2. 图片与文字交互 254](#_Toc25953445)

[14.6.3. 将英尺转换为mi 255](#_Toc25953446)

[14.6.4. 简易计算器 256](#_Toc25953447)

[14.7. 代码模板 257](#_Toc25953448)

[14.8. 遇到的难题与问题 260](#_Toc25953449)

[15. PyQt4 263](#_Toc25953450)

[15.1. 基础 263](#_Toc25953451)

[15.2. 组件 263](#_Toc25953452)

[15.2.1. 其他 263](#_Toc25953453)

[15.2.2. 有关Label 264](#_Toc25953454)

[15.2.3. 有关button 264](#_Toc25953455)

[15.3. 事件与信号 264](#_Toc25953456)

[15.4. 布局管理 266](#_Toc25953457)

[15.4.1. grid布局 266](#_Toc25953458)

[15.4.2. 其他 267](#_Toc25953459)

[15.5. 菜单和工具栏 267](#_Toc25953460)

[15.6. 输入框/对话框 268](#_Toc25953461)

[15.6.1. QInputDialog输入对话框 268](#_Toc25953462)

[15.6.2. QColorDialog 颜色对话框 269](#_Toc25953463)

[15.6.3. QFontDialog 字体对话框 269](#_Toc25953464)

[15.6.4. QFileDialog 打开文件 270](#_Toc25953465)

[15.6.5. QMessageBox 271](#_Toc25953466)

[15.7. 字体和背景 271](#_Toc25953467)

[15.8. 窗口 272](#_Toc25953468)

[15.9. 细节 275](#_Toc25953469)

[16. 组成exe 276](#_Toc25953470)

[16.1. py2exe模块 276](#_Toc25953471)

[16.1.1. 基础 276](#_Toc25953472)

[16.1.2. python编译的所有文件打包到一个exe中 277](#_Toc25953473)

[16.1.3. 遇到的问题 278](#_Toc25953474)

[16.2. PySide模块 279](#_Toc25953475)

[16.2.1. 基础 279](#_Toc25953476)

[16.2.2. Qt Designer 280](#_Toc25953477)

[16.3. pywin32 281](#_Toc25953478)

[16.4. wxpython 282](#_Toc25953479)

[17. 爬虫wiki文档平台 282](#_Toc25953480)

[17.1. wiki\_scrapy文件 282](#_Toc25953481)

[17.2. pipelines文件 连接mongodb mysql 282](#_Toc25953482)

[17.3. setting文件 284](#_Toc25953483)

[17.4. items文件 284](#_Toc25953484)

[17.5. mongo查询语句 284](#_Toc25953485)

[17.6. mysql数据库表设置 284](#_Toc25953486)

[17.7. 创建爬取组下的所有页面链接 285](#_Toc25953487)

[17.8. flask 285](#_Toc25953488)

[17.8.1. html 285](#_Toc25953489)

[17.8.2. 细节 286](#_Toc25953490)

[17.8.3. 全文 286](#_Toc25953491)

[17.9. 问题解决 288](#_Toc25953492)

[18. 二维码查看器 288](#_Toc25953493)

[18.1. 客户端 288](#_Toc25953494)

[18.1.1. qrcode模块 288](#_Toc25953495)

[18.1.2. 获取mac地址 289](#_Toc25953496)

[18.1.3. 问题 289](#_Toc25953497)

[18.2. 服务端 289](#_Toc25953498)

[18.2.1. 基础 289](#_Toc25953499)

[18.2.2. 问题 291](#_Toc25953500)

[18.3. 手机html 292](#_Toc25953501)

[19. loglogic模块 292](#_Toc25953502)

[19.1. grep命令 292](#_Toc25953503)

[19.2. 数据库 sqlite3 293](#_Toc25953504)

[19.3. ORM操作 293](#_Toc25953505)

[19.4. 生成graph 294](#_Toc25953506)

[19.5. 与外界交互 查看数据库信息 295](#_Toc25953507)

[19.6. 细节技巧 296](#_Toc25953508)

[20. logre 模块 296](#_Toc25953509)

[20.1. 单库操作模块 296](#_Toc25953510)

[20.2. 同步模块 298](#_Toc25953511)

[20.3. 寻找数据库表中各字段频率最高的值并保存 299](#_Toc25953512)

[20.4. 细节及bug解决 300](#_Toc25953513)

[21. linux 300](#_Toc25953514)

[21.1. linux命令 300](#_Toc25953515)

[21.2. 防火墙 iptables 和 SELinux 303](#_Toc25953516)

[21.3. tomcat nginx 303](#_Toc25953517)

[21.3.1. 基础 304](#_Toc25953518)

[21.3.2. 安装nginx 304](#_Toc25953519)

[21.3.3. 安装tomcat 305](#_Toc25953520)

[21.3.4. 安装auto-add-tomcat 305](#_Toc25953521)

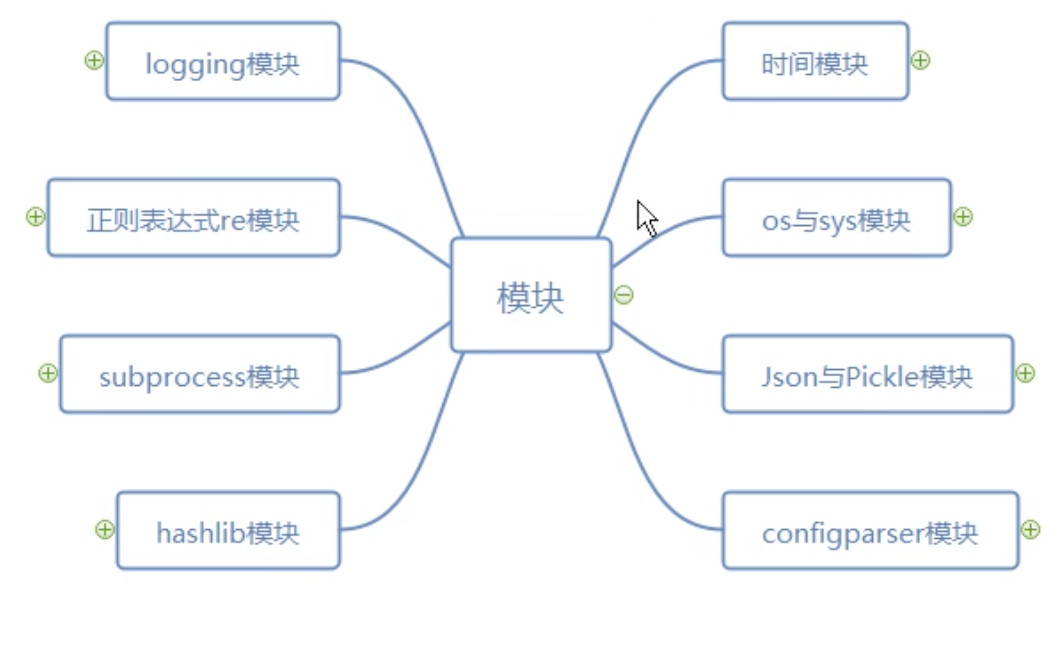
[21.4. linux安装环境 305](#_Toc25953522)

[21.4.1. 安装模块包 305](#_Toc25953523)

[21.4.2. 运行问题 307](#_Toc25953524)

[21.4.3. 其他 307](#_Toc25953525)

# python面试题知识点:



# python基础知识

## 基础与解答

1. NameError: global name is not defined?

class App:

def \_\_init\_\_(self, root):

self.cutoff=0

self.back\_next\_index=0

def back\_case():

self.back\_next\_index-=1

self.go=0

def gisy():

print self.go

self.\_\_init\_\_(root)

self.back\_case()

def hil(self):

print self.cutoff

python 不会自动去当前行代码所在的class搜索变量或函数，所以需要加上self.才行

同一class下不同方法下的变量调用也要加self. 否则识别不出。

同一函数下的不同子函数中变量也要加self

调用函数时要加()

1. AttributeError: 'NoneType' object has no attribute 'cache\_textbox'

不带参数值的return语句返回None。return语句退出函数，选择性地向调用方返回一个表达式。

1. return返回的不会打印在控制台上，必须再次使用print

def hello(self):

return 'gisy'

app=App(root)

print app.hello()

1. 如果一个函数的参数中含有默认参数，则这个默认参数后的所有参数都必须是默认参数 ，

否则会抛出：SyntaxError: non-default argument follows default argument的异常。

def run(name,age=10,sex):

print name ,age ,sex

SyntaxError: non-default argument follows default argument gg 23 boy

1. 函数调用 不能前面参数有keyword，后面没有

比如self.center\_window(window=self.root,700,850)

SyntaxError: non-keyword arg after keyword arg

run('gg',name='pp')#duplicate value for argument 在一次调用中不能同时使用位置和keyword绑定

1. 将字典、列表 格式化成 树形、带缩进的字符串

import json;

#demoDictList is the value we want format to output。demoDictList不能是字符串，否则无法格式化

jsonDumpsIndentStr = json.dumps(demoDictList, indent=1);

print "jsonDumpsIndentStr=",jsonDumpsIndentStr;

1. 将json格式化成树形、带缩进的json

因为json已经是字符串格式，必须先转换为python对象，也就是list或dictionary格式。然后在转换成带缩进的json字符串

data = tablib.Dataset(\*self.case\_cashe, headers=columnlist)

jsonList=json.loads(data.json)

print type(jsonList) # <type 'list'>

# indent = 1代表缩进1个字符；ensure\_ascii = False代表汉字字符可以正常显示汉字，如果为 True,则显示汉字的Unicode编码

jsonDumpsIndentStr = json.dumps(jsonList,ensure\_ascii = False, indent = 1)

print jsonDumpsIndentStr

filetypes = [ ("json Files", '\*.txt')]

dirname=os.path.dirname(self.original\_name)

basename=os.path.basename(self.original\_name)

default\_jsonname=os.path.splitext(basename)[0]+'.txt'

jsonname = tkFileDialog.asksaveasfilename(filetypes=filetypes,initialdir =dirname,initialfile =default\_jsonname)

with open(jsonname, 'w+') as f:

f.write(jsonDumpsIndentStr)

1. 参数列list的分拆、 参数字典dict的分拆

range(3, 6) # normal call with separate arguments

[3, 4, 5]

>>> args = [3, 6]

>>> range(\*args) # call with arguments unpacked from a list

[3, 4, 5]

gisy(\*\*args)

1. 可变参数

参数被包装进一个元组。在这些可变个数的参数之前，可以有零到多个普通的参数：

def run(name,\*args):

1. 函数中变量有可能之前还未定义，如何解决？

def f(a, L=[]):

L.append(a)

return L

## 迭代器

### 基础

我们已经知道，可以直接作用于for循环的数据类型有以下几种：

一类是集合数据类型，如list、tuple、dict、set、str等；

一类是generator，包括生成器和带yield的generator function。

这些可以直接作用于for循环的对象统称为可迭代对象：Iterable。

可以使用isinstance()判断一个对象是否是**Iterable**对象：

>>> from collections import Iterable

>>> isinstance([], Iterable)

True

>>> isinstance({}, Iterable)

True

>>> isinstance('abc', Iterable)

True

>>> isinstance((x for x in range(10)), Iterable)

True

>>> isinstance(100, Iterable)

False

而生成器不但可以作用于for循环，还可以被next()函数不断调用并返回下一个值，直到最后抛出StopIteration错误表示无法继续返回下一个值了。

可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator。

可以使用isinstance()判断一个对象是否是Iterator对象：

>>> from collections import Iterator

>>> isinstance((x for x in range(10)), Iterator)

True

生成器都是Iterator对象，但list、dict、str虽然是Iterable，却不是Iterator。

把list、dict、str等Iterable变成Iterator可以使用iter()函数：

凡是可作用于for循环的对象都是Iterable类型；

凡是可作用于next()函数的对象都是Iterator类型，它们表示一个惰性计算的序列；

集合数据类型如list、dict、str等是Iterable但不是Iterator，不过可以通过iter()函数获得一个Iterator对象。

Python的for循环本质上就是通过不断调用next()函数实现的，例如：

for x in [1, 2, 3, 4, 5]:

pass

实际上完全等价于：

# 首先获得Iterator对象:

it = iter([1, 2, 3, 4, 5])

# 循环:

while True:

try:

# 获得下一个值:

x = next(it)

except StopIteration:

# 遇到StopIteration就退出循环

break

### 举例

\_\_iter\_\_():迭代器，生成迭代对象时调用，返回值必须是对象自己,然后for可以循环调用next方法

next():每一次for循环都调用该方法（必须存在）

class A(object):

def \_\_init\_\_(self,num):

self.num = num

self.start\_num = -1

def \_\_iter\_\_(self):

'''''

@summary: 迭代器，生成迭代对象时调用，返回值必须是对象自己,然后for可以循环调用next方法

'''

print "\_\_iter\_\_"

return self

def next(self):

'''''

@summary: 每一次for循环都调用该方法（必须存在）

'''

self.start\_num += 1

if self.start\_num >= self.num:

raise StopIteration()

return self.start\_num

if \_\_name\_\_ == "\_\_main\_\_":

for i in A(10):print i,

### \_\_iter\_\_ 类想被用于for ... in循环

如果一个类想被用于for ... in循环，类似list或tuple那样，就必须实现一个\_\_iter\_\_()方法，该方法返回一个迭代对象，然后，Python的for循环就会不断调用该迭代对象的next()方法拿到循环的下一个值，直到遇到StopIteration错误时退出循环。

我们以斐波那契数列为例，写一个Fib类，可以作用于for循环：

class Fib(object):

def \_\_init\_\_(self):

self.a, self.b = 0, 1 # 初始化两个计数器a，b

def \_\_iter\_\_(self):

return self # 实例本身就是迭代对象，故返回自己

def next(self):

self.a, self.b = self.b, self.a + self.b # 计算下一个值

if self.a > 100000: # 退出循环的条件

raise StopIteration();

return self.a # 返回下一个值

### functools.wraps

wraps() 函数把用 partial() 把 update\_wrapper() 给封装了一下。

不需要编写wrapper.\_\_name\_\_ = func.\_\_name\_\_这样的代码，Python内置的functools.wraps就是干这个事的。这个函数可以将被封装函数的名称、模块、文档拷贝给封装函数。

import functools

def log(func):

@functools.wraps(func)

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

print 'call %s():' % func.\_\_name\_\_

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

return wrapper

或者针对带参数的decorator：

import functools

def log(text):

def decorator(func):

@functools.wraps(func)

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

print '%s %s():' % (text, func.\_\_name\_\_)

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

return wrapper

return decorator

import functools是导入functools模块。现在，只需记住在定义wrapper()的前面加上@functools.wraps(func)即可。

## 生成器

yield 的作用就是把一个函数变成一个 generator，带有 yield 的函数不再是一个普通函数，Python 解释器会将其视为一个 generator，调用 fab(5) 不会执行 fab 函数，而是返回一个 iterable 对象

def fab(max):

n, a, b = 0, 0, 1

while n < max:

yield b

# print b

a, b = b, a + b

n = n + 1

在一个 generator function 中，如果没有 return，则默认执行至函数完毕，如果在执行过程中 return，则直接抛出 StopIteration 终止迭代。

判断一个函数是否是一个特殊的 generator 函数？可以利用 isgeneratorfunction 判断

from inspect import isgeneratorfunction

isgeneratorfunction(fab)

要注意区分 fab 和 fab(5)，fab 是一个 generator function，而 fab(5) 是调用 fab 返回的一个 generator，好比类的定义和类的实例的区别

另一个 yield 的例子来源于文件读取。如果直接对文件对象调用 read() 方法，会导致不可预测的内存占用。好的方法是利用固定长度的缓冲区来不断读取文件内容。通过 yield，我们不再需要编写读文件的迭代类，就可以轻松实现文件读取

def read\_file(fpath):

BLOCK\_SIZE=1024

with open(fpath,'rb') as f:

while True:

block=f.read(BLOCK\_SIZE)

if block:

yield block

else:

return

for n in read\_file('file.txt'):

print n

处理任意层的嵌套

>>> def fla(aa):

try:

for bb in aa:

for cc in fla(bb):

yield cc

except TypeError:

yield aa

>>> list(fla([[[1],2],3,4,[5,[6,7]],8])) #注意括号层次比较多

[1, 2, 3, 4, 5, 6, 7, 8]

试着将传入的对象和一个字符串拼接，看看会不会出现TypeError，**这是检查一个对象是不是类似于字符串最简单快速的方法。**

def flatten(nested):

try:

#不要迭代类似字符串的对象

try:nested + ''

except TypeError: pass

else: raise TypeError

for sublist in nested:

for element in flatten(sublist):

yield element

except TypeError:

yield nested

>>> list(flatten(['foo',['bar',['baz']]]))

['foo', 'bar', 'baz']

生成器不保留迭代过后的结果

In [12]: gen = (i for i in range(5))

In [13]: 2 in gen

Out[13]: True

In [15]: 1 in gen

Out[15]: False # 1为什么不在gen里面了? 因为调用1->2,这个时候1已经不在迭代器里面了,被按需生成过了

In [20]: gen = (i for i in range(5))

In [21]: a\_list = list(gen) # 可以转化成列表，当然a\_tuple = tuple(gen) 也可以

In [22]: 2 in a\_list

Out[22]: True

In [24]: 1 in a\_list # 就算循环过,值还在

Out[24]: True

## 描述器 descriptor

所谓描述器，即实现了描述符协议，即\_\_get\_\_, \_\_set\_\_, 和 \_\_delete\_\_方法的对象。

class WebFramework(object):

def \_\_init\_\_(self, name='Flask'):

self.name = name

def \_\_get\_\_(self, instance, owner):

return self.name

def \_\_set\_\_(self, instance, value):

self.name = value

class PythonSite(object):

webframework = WebFramework() # WebFramework()为描述器

In [1]: PythonSite.webframework

Out[1]: 'Flask'

In [2]: PythonSite.webframework = 'Tornado'

In [3]: PythonSite.webframework

Out[3]: 'Tornado'

定义了一个类WebFramework，它实现了描述符协议\_\_get\_\_和\_\_set\_\_，该对象（类也是对象，一切都是对象）即成为了一个描述器。同时实现\_\_get\_\_和\_\_set\_\_的称之为资料描述器（data descriptor）。仅仅实现\_\_get\_\_的则为非描述器。两者的差别是相对于实例的字典的优先级。

如果实例字典中有与描述器同名的属性，如果描述器是资料描述器，优先使用资料描述器，如果是非资料描述器，优先使用字典中的属性。

描述器的调用比较简单，对象属性的时候会调用

In [15]: webframework = WebFramework()

In [16]: webframework.\_\_get\_\_(webframework, WebFramework)

Out[16]: 'Flask'

class Descriptor:

def \_\_get\_\_(self, instance, owner):

print(self, instance, owner)

def \_\_set\_\_(self, instance, value):

print(self, instance, value)

def \_\_delete\_\_(self, instance):

print(self, instance)

class Some:

x = Descriptor() # Descriptor()为描述器

在上例中，如果這麼執行：

s = Some()

s.x

s.x = 10

del s.x

其實相當於這麼作：

s = Some()

Some.\_\_dict\_\_['x'].\_\_get\_\_(s, Some);

Some.\_\_dict\_\_['x'].\_\_set\_\_(s, 10);

Some.\_\_dict\_\_['x'].\_\_delete\_\_(s);

如果這麼作的話：

Some.x

則相當於這麼作：

Some.\_\_dict\_\_['x'].\_\_get\_\_(None, Some)

## 装饰器

这种在代码运行期间动态增加功能的方式，称之为“装饰器”（Decorator）。

### 无参数

'''使用内嵌包装函数来确保每次新函数都被调用，

内嵌包装函数的形参和返回值与原函数相同，装饰函数返回内嵌包装函数对象'''

def deco(func):

def \_deco():

print("before myfunc() called.")

func()

print(" after myfunc() called.")

# 不需要返回func，实际上应返回原函数的返回值

return \_deco

@deco

def myfunc():

print(" myfunc() called.")

return 'ok'

myfunc()

myfunc()

### 带参数

内嵌包装函数的形参和返回值与原函数相同，装饰函数返回内嵌包装函数对象'''

def deco(func):

def \_deco(a, b):

print("before myfunc() called.")

ret = func(a, b)

print(" after myfunc() called. result: %s" % ret)

return ret

return \_deco

@deco

def myfunc(a, b):

print(" myfunc(%s,%s) called." % (a, b))

return a + b

myfunc(1, 2)

myfunc(3, 4)

### 参数数量不确定

参数用(\*args, \*\*kwargs)，自动适应变参和命名参数'''

def deco(func):

def \_deco(\*args, \*\*kwargs):

print("before %s called." % func.\_\_name\_\_)

ret = func(\*args, \*\*kwargs)

print(" after %s called. result: %s" % (func.\_\_name\_\_, ret))

return ret

return \_deco

@deco

def myfunc(a, b):

print(" myfunc(%s,%s) called." % (a, b))

return a+b

@deco

def myfunc2(a, b, c):

print(" myfunc2(%s,%s,%s) called." % (a, b, c))

return a+b+c

myfunc(1, 2)

myfunc(3, 4)

myfunc2(1, 2, 3)

myfunc2(3, 4, 5)

在类中实现

class dataInit(object):

def gisy(func):

def \_gisy(\*args, \*\*kwargs):

print 'hello nihaoj;'

print args,kwargs

func(\*args, \*\*kwargs)

return \_gisy

@gisy

def \_\_setitem\_\_(self, key, value):

self.orderidlist[key]=value

### 让装饰器带参数

def deco(arg):

def \_deco(func):

def \_\_deco():

print("before %s called [%s]." % (func.\_\_name\_\_, arg))

func()

print(" after %s called [%s]." % (func.\_\_name\_\_, arg))

return \_\_deco

return \_deco

@deco("mymodule")

def myfunc():

print(" myfunc() called.")

@deco("module2") #装饰器带参数

def myfunc2():

print(" myfunc2() called.")

myfunc()

myfunc2()

### 让装饰器带 类 参数

class locker:

def \_\_init\_\_(self):

print("locker.\_\_init\_\_() should be not called.")

@staticmethod

def acquire():

print("locker.acquire() called.（这是静态方法）")

@staticmethod

def release():

print(" locker.release() called.（不需要对象实例）")

def deco(cls):

'''cls 必须实现acquire和release静态方法'''

def \_deco(func):

def \_\_deco():

print("before %s called [%s]." % (func.\_\_name\_\_, cls))

cls.acquire() #实现locker的acquire方法

try:

return func()

finally:

cls.release() #实现locker的release方法

return \_\_deco

return \_deco

@deco(locker) #locker为类

def myfunc():

print(" myfunc() called.")

myfunc()

myfunc()

## TCP编程

Socket是网络编程的一个抽象概念。通常我们用一个Socket表示“打开了一个网络链接”，而打开一个Socket需要知道目标计算机的IP地址和端口号，再指定协议类型即可。

### 客户端

大多数连接都是可靠的TCP连接。创建TCP连接时，主动发起连接的叫客户端，被动响应连接的叫服务器。

举个例子，当我们在浏览器中访问新浪时，我们自己的计算机就是客户端，浏览器会主动向新浪的服务器发起连接。如果一切顺利，新浪的服务器接受了我们的连接，一个TCP连接就建立起来的，后面的通信就是发送网页内容了。

所以，我们要创建一个基于TCP连接的Socket，可以这样做：

# 导入socket库:

import socket

# 创建一个socket:

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# 建立连接:

s.connect(('www.sina.com.cn', 80))

创建Socket时，AF\_INET指定使用IPv4协议，如果要用更先进的IPv6，就指定为AF\_INET6。SOCK\_STREAM指定使用面向流的TCP协议，这样，一个Socket对象就创建成功，但是还没有建立连接。

客户端要主动发起TCP连接，必须知道服务器的IP地址和端口号。新浪网站的IP地址可以用域名www.sina.com.cn自动转换到IP地址，但是怎么知道新浪服务器的端口号呢？

答案是作为服务器，提供什么样的服务，端口号就必须固定下来。由于我们想要访问网页，因此新浪提供网页服务的服务器必须把端口号固定在80端口，因为80端口是Web服务的标准端口。其他服务都有对应的标准端口号，例如SMTP服务是25端口，FTP服务是21端口，等等。端口号小于1024的是Internet标准服务的端口，端口号大于1024的，可以任意使用。

因此，我们连接新浪服务器的代码如下：

s.connect(('www.sina.com.cn', 80))

注意参数是一个tuple，包含地址和端口号。

建立TCP连接后，我们就可以向新浪服务器发送请求，要求返回首页的内容：

# 发送数据:

s.send('GET / HTTP/1.1\r\nHost: www.sina.com.cn\r\nConnection: close\r\n\r\n')

TCP连接创建的是双向通道，双方都可以同时给对方发数据。但是谁先发谁后发，怎么协调，要根据具体的协议来决定。例如，HTTP协议规定客户端必须先发请求给服务器，服务器收到后才发数据给客户端。

发送的文本格式必须符合HTTP标准，如果格式没问题，接下来就可以接收新浪服务器返回的数据了：

# 接收数据:

buffer = []

while True:

# 每次最多接收1k字节:

d = s.recv(1024)

if d:

buffer.append(d)

else:

break

data = ''.join(buffer)

接收数据时，调用recv(max)方法，一次最多接收指定的字节数，因此，在一个while循环中反复接收，直到recv()返回空数据，表示接收完毕，退出循环。

当我们接收完数据后，调用close()方法关闭Socket，这样，一次完整的网络通信就结束了：

# 关闭连接:

s.close()

接收到的数据包括HTTP头和网页本身，我们只需要把HTTP头和网页分离一下，把HTTP头打印出来，网页内容保存到文件：

header, html = data.split('\r\n\r\n', 1)

print header

# 把接收的数据写入文件:

with open('sina.html', 'wb') as f:

f.write(html)

现在，只需要在浏览器中打开这个sina.html文件，就可以看到新浪的首页了。

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

import socket

s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.connect((*'127.0.0.1'*,9999))

print s.recv(1024)

for data in ['micheal','tracy','sarah']:

s.send(data)

print s.recv(1024)

s.send(*'exit'*)

s.close()

### 服务器

和客户端编程相比，服务器编程就要复杂一些。

服务器进程首先要绑定一个端口并监听来自其他客户端的连接。如果某个客户端连接过来了，服务器就与该客户端建立Socket连接，随后的通信就靠这个Socket连接了。

所以，服务器会打开固定端口（比如80）监听，每来一个客户端连接，就创建该Socket连接。由于服务器会有大量来自客户端的连接，所以，服务器要能够区分一个Socket连接是和哪个客户端绑定的。一个Socket依赖4项：服务器地址、服务器端口、客户端地址、客户端端口来唯一确定一个Socket。

但是服务器还需要同时响应多个客户端的请求，所以，每个连接都需要一个新的进程或者新的线程来处理，否则，服务器一次就只能服务一个客户端了。

我们来编写一个简单的服务器程序，它接收客户端连接，把客户端发过来的字符串加上Hello再发回去。

首先，创建一个基于IPv4和TCP协议的Socket：

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

然后，我们要绑定监听的地址和端口。服务器可能有多块网卡，可以绑定到某一块网卡的IP地址上，也可以用0.0.0.0绑定到所有的网络地址，还可以用127.0.0.1绑定到本机地址。127.0.0.1是一个特殊的IP地址，表示本机地址，如果绑定到这个地址，客户端必须同时在本机运行才能连接，也就是说，外部的计算机无法连接进来。

端口号需要预先指定。因为我们写的这个服务不是标准服务，所以用9999这个端口号。请注意，小于1024的端口号必须要有管理员权限才能绑定：

# 监听端口:

s.bind(('127.0.0.1', 9999))

紧接着，调用listen()方法开始监听端口，传入的参数指定等待连接的最大数量：

s.listen(5)

print 'Waiting for connection...'

接下来，服务器程序通过一个永久循环来接受来自客户端的连接，accept()会等待并返回一个客户端的连接:

while True:

# 接受一个新连接:

sock, addr = s.accept()

# 创建新线程来处理TCP连接:

t = threading.Thread(target=tcplink, args=(sock, addr))

t.start()

每个连接都必须创建新线程（或进程）来处理，否则，单线程在处理连接的过程中，无法接受其他客户端的连接：

def tcplink(sock, addr):

print 'Accept new connection from %s:%s...' % addr

sock.send('Welcome!')

while True:

data = sock.recv(1024)

time.sleep(1)

if data == 'exit' or not data:

break

sock.send('Hello, %s!' % data)

sock.close()

print 'Connection from %s:%s closed.' % addr

连接建立后，服务器首先发一条欢迎消息，然后等待客户端数据，并加上Hello再发送给客户端。如果客户端发送了exit字符串，就直接关闭连接。

要测试这个服务器程序，我们还需要编写一个客户端程序：

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# 建立连接:

s.connect(('127.0.0.1', 9999))

# 接收欢迎消息:

print s.recv(1024)

for data in ['Michael', 'Tracy', 'Sarah']:

# 发送数据:

s.send(data)

print s.recv(1024)

s.send('exit')

s.close()

我们需要打开两个命令行窗口，一个运行服务器程序，另一个运行客户端程序，就可以看到效果了：

client-server

需要注意的是，客户端程序运行完毕就退出了，而服务器程序会永远运行下去，必须按Ctrl+C退出程序。

小结

用TCP协议进行Socket编程在Python中十分简单，对于客户端，要主动连接服务器的IP和指定端口，对于服务器，要首先监听指定端口，然后，对每一个新的连接，创建一个线程或进程来处理。通常，服务器程序会无限运行下去。

同一个端口，被一个Socket绑定了以后，就不能被别的Socket绑定了。

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

import socket,threading,time

def **tcplink**(sock,addr):

print 'Accept new connection from %s:%s...' %addr

sock.send(*'Welcome!'*)

while True:

data=sock.recv(1024)

time.sleep(1)

if data==*'exit'* or not data:

break

sock.send(*'Hello,%s!'* %data)

sock.close()

print 'Connection from %s:%s closed.' %addr

s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.bind((*'127.0.0.1'*,9999))

s.listen(5)

print 'Waiting for connection'

while True:

sock,addr=s.accept()

t= threading.Thread(target=tcplink,args=(sock,addr))

t.start()

## UDP编程

UDP编程

TCP是建立可靠连接，并且通信双方都可以以流的形式发送数据。相对TCP，UDP则是面向无连接的协议。

使用UDP协议时，不需要建立连接，只需要知道对方的IP地址和端口号，就可以直接发数据包。但是，能不能到达就不知道了。

虽然用UDP传输数据不可靠，但它的优点是和TCP比，速度快，对于不要求可靠到达的数据，就可以使用UDP协议。

我们来看看如何通过UDP协议传输数据。和TCP类似，使用UDP的通信双方也分为客户端和服务器。服务器首先需要绑定端口：

s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# 绑定端口:

s.bind(('127.0.0.1', 9999))

创建Socket时，SOCK\_DGRAM指定了这个Socket的类型是UDP。绑定端口和TCP一样，但是不需要调用listen()方法，而是直接接收来自任何客户端的数据：

print 'Bind UDP on 9999...'

while True:

# 接收数据:

data, addr = s.recvfrom(1024)

print 'Received from %s:%s.' % addr

s.sendto('Hello, %s!' % data, addr)

recvfrom()方法返回数据和客户端的地址与端口，这样，服务器收到数据后，直接调用sendto()就可以把数据用UDP发给客户端。

注意这里省掉了多线程，因为这个例子很简单。

客户端使用UDP时，首先仍然创建基于UDP的Socket，然后，不需要调用connect()，直接通过sendto()给服务器发数据：

s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

for data in ['Michael', 'Tracy', 'Sarah']:

# 发送数据:

s.sendto(data, ('127.0.0.1', 9999))

# 接收数据:

print s.recv(1024)

s.close()

从服务器接收数据仍然调用recv()方法。

仍然用两个命令行分别启动服务器和客户端测试，结果如下：

client-server

小结

UDP的使用与TCP类似，但是不需要建立连接。此外，服务器绑定UDP端口和TCP端口互不冲突，也就是说，UDP的9999端口与TCP的9999端口可以各自绑定。

## 动态变量

locals()为局部变量 globals()为全局变量

python内部并没有读locals()来寻找变量名，通过locals来更改变量值也是不可行的

如果你只是读取全局变量的值，你就不需要全局关键字global。你只有在想要改变全局变量名引用的对象时有使用global关键字的必要。

columnlist=['name','summary','preconditions','step\_number','actions','expectedresults']

for i in range(6):

globals()['CheckVar'+str(i)]=IntVar()

globals()['check'+str(i)]=Checkbutton(top, text =columnlist[i], variable = globals()['CheckVar'+str(i)],onvalue = 1, offvalue = 0)

globals()['check'+str(i)].grid(row=1, column=i+1,sticky='e')

globals()['check'+str(i)].select()

print globals()

def toggle\_fun():

for i in range(6):

globals()['check'+str(i)].toggle()

UnboundLocalError: local variable referenced before assignment

这是由于python对变量的搜索机制引起的：如果有一个函数内部的变量赋值 ，该变量被认为是本地的，所以如果有修改变量的值就会变成局部变量。

解决方案：

caseorder=0

def print\_content():

global caseorder

caseorder+=1

## 进程和线程

### ThreadLocal

在多线程环境下，每个线程都有自己的数据。一个线程使用自己的局部变量比使用全局变量好，因为局部变量只有线程自己能看见，不会影响其他线程，而全局变量的修改必须加锁。

但是局部变量也有问题，就是在函数调用的时候，传递起来很麻烦：

def process\_student(name):

std = Student(name)

# std是局部变量，但是每个函数都要用它，因此必须传进去：

do\_task\_1(std)

do\_task\_2(std)

def do\_task\_1(std):

do\_subtask\_1(std)

do\_subtask\_2(std)

def do\_task\_2(std):

do\_subtask\_2(std)

do\_subtask\_2(std)

每个函数一层一层调用都这么传参数那还得了？用全局变量？也不行，因为每个线程处理不同的Student对象，不能共享。

如果用一个全局dict存放所有的Student对象，然后以thread自身作为key获得线程对应的Student对象如何？

global\_dict = {}

def std\_thread(name):

std = Student(name)

# 把std放到全局变量global\_dict中：

global\_dict[threading.current\_thread()] = std

do\_task\_1()

do\_task\_2()

def do\_task\_1():

# 不传入std，而是根据当前线程查找：

std = global\_dict[threading.current\_thread()]

...

def do\_task\_2():

# 任何函数都可以查找出当前线程的std变量：

std = global\_dict[threading.current\_thread()]

...

这种方式理论上是可行的，它最大的优点是消除了std对象在每层函数中的传递问题，但是，每个函数获取std的代码有点丑。

有没有更简单的方式？

ThreadLocal应运而生，不用查找dict，ThreadLocal帮你自动做这件事：

import threading

# 创建全局ThreadLocal对象:

local\_school = threading.local()

def process\_student():

print 'Hello, %s (in %s)' % (local\_school.student, threading.current\_thread().name)

def process\_thread(name):

# 绑定ThreadLocal的student:

local\_school.student = name

process\_student()

t1 = threading.Thread(target= process\_thread, args=('Alice',), name='Thread-A')

t2 = threading.Thread(target= process\_thread, args=('Bob',), name='Thread-B')

t1.start()

t2.start()

t1.join()

t2.join()

执行结果：

Hello, Alice (in Thread-A)

Hello, Bob (in Thread-B)

全局变量local\_school就是一个ThreadLocal对象，每个Thread对它都可以读写student属性，但互不影响。你可以把local\_school看成全局变量，但每个属性如local\_school.student都是线程的局部变量，可以任意读写而互不干扰，也不用管理锁的问题，ThreadLocal内部会处理。

可以理解为全局变量local\_school是一个dict，不但可以用local\_school.student，还可以绑定其他变量，如local\_school.teacher等等。

ThreadLocal最常用的地方就是为每个线程绑定一个数据库连接，HTTP请求，用户身份信息等，这样一个线程的所有调用到的处理函数都可以非常方便地访问这些资源。

### 多进程

#### Pool

如果要启动大量的子进程，可以用进程池的方式批量创建子进程：

from multiprocessing import Pool

import os, time, random

def long\_time\_task(name):

print 'Run task %s (%s)...' % (name, os.getpid())

start = time.time()

time.sleep(random.random() \* 3)

end = time.time()

print 'Task %s runs %0.2f seconds.' % (name, (end - start))

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

print 'Parent process %s.' % os.getpid()

p = Pool()

for i in range(5):

p.apply\_async(long\_time\_task, args=(i,))

print 'Waiting for all subprocesses done...'

p.close()

p.join()

print 'All subprocesses done.'

代码解读：

对Pool对象调用join()方法会等待所有子进程执行完毕，调用join()之前必须先调用close()，调用close()之后就不能继续添加新的Process了。

请注意输出的结果，task 0，1，2，3是立刻执行的，而task 4要等待前面某个task完成后才执行，这是因为Pool的默认大小在我的电脑上是4，因此，最多同时执行4个进程。这是Pool有意设计的限制，并不是操作系统的限制。如果改成：

p = Pool(5)

就可以同时跑5个进程。

由于Pool的默认大小是CPU的核数，如果你不幸拥有8核CPU，你要提交至少9个子进程才能看到上面的等待效果。

#### 进程间通信

Process之间肯定是需要通信的，操作系统提供了很多机制来实现进程间的通信。Python的multiprocessing模块包装了底层的机制，提供了Queue、Pipes等多种方式来交换数据。

我们以Queue为例，在父进程中创建两个子进程，一个往Queue里写数据，一个从Queue里读数据：

from multiprocessing import Process, Queue

import os, time, random

# 写数据进程执行的代码:

def write(q):

for value in ['A', 'B', 'C']:

print 'Put %s to queue...' % value

q.put(value)

time.sleep(random.random())

# 读数据进程执行的代码:

def read(q):

while True:

value = q.get(True)

print 'Get %s from queue.' % value

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

# 父进程创建Queue，并传给各个子进程：

q = Queue()

pw = Process(target=write, args=(q,))

pr = Process(target=read, args=(q,))

# 启动子进程pw，写入:

pw.start()

# 启动子进程pr，读取:

pr.start()

# 等待pw结束:

pw.join()

# pr进程里是死循环，无法等待其结束，只能强行终止:

pr.terminate()

### 多线程

多线程编程，模型复杂，容易发生冲突，必须用锁加以隔离，同时，又要小心死锁的发生。

Python解释器由于设计时有GIL全局锁，导致了多线程无法利用多核。多线程的并发在Python中就是一个美丽的梦。

balance = 0

lock = threading.Lock()

def run\_thread(n):

for i in range(100000):

# 先要获取锁:

lock.acquire()

try:

# 放心地改吧:

change\_it(n)

finally:

# 改完了一定要释放锁:

lock.release()

## WebSocket协议分析

### 协议概览

协议包含两部分：握手，数据传输。

客户端的握手如下：

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Origin: http://example.com

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

服务端的握手如下：

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

Sec-WebSocket-Protocol: chat

客户端和服务端都发送了握手，并且成功，数据传输即可开始。

1.3 发起握手

发起握手是为了兼容基于HTTP的服务端程序，这样一个端口可以同时处理HTTP客户端和WebSocket客户端

因此WebSocket客户端握手是一个HTTP Upgrade请求：

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Origin: http://example.com

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

握手中的域的顺序是任意的。

这个请求是对服务器的端口发送的，一般的话，会预先在服务器将一个socket 绑定到一个端口上，客户端和服务器端在这个预定的端口上通信(我这里绑定的就是 4000 端口，默认情况下，websocke 使用 80 端口)。

然后，在服务器端的socket监听到这个packet 之后就生成一个新的 socket，将发送过来的数据中的 Sec-WebSocket-Key 解析出来，然后按照把“Sec-WebSocket-Ke”加上一个魔幻字符串“258EAFA5-E914-47DA-95CA-C5AB0DC85B11”。使用SHA-1加密，之后进行BASE-64编码，将结果做为“Sec-WebSocket-Accept”头的值，返回给客户端。

客户端收到这个之后，就会将 通信协议 upgrade 到 websocket 协议。

然后就会在这个持久的通道下进行通信，包括浏览器的询问，服务器的push，双方是在一个全双工的状态下相互通信。 切换后的websocket 协议中的 数据传输帧的格式(此时不再使用html协议)

### 数据帧

5.1 概述

WebScoket协议中，数据以帧序列的形式传输。

考虑到数据安全性，客户端向服务器传输的数据帧必须进行掩码处理。服务器若接收到未经过掩码处理的数据帧，则必须主动关闭连接。

服务器向客户端传输的数据帧一定不能进行掩码处理。客户端若接收到经过掩码处理的数据帧，则必须主动关闭连接。

针对上情况，发现错误的一方可向对方发送close帧（状态码是1002，表示协议错误），以关闭连接。

5.2 帧协议

WebSocket数据帧结构如下图所示：

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-------+-+-------------+-------------------------------+

|F|R|R|R| opcode|M| Payload len | Extended payload length |

|I|S|S|S| (4) |A| (7) | (16/64) |

|N|V|V|V| |S| | (if payload len==126/127) |

| |1|2|3| |K| | |

+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +

| Extended payload length continued, if payload len == 127 |

+ - - - - - - - - - - - - - - - +-------------------------------+

| |Masking-key, if MASK set to 1 |

+-------------------------------+-------------------------------+

| Masking-key (continued) | Payload Data |

+-------------------------------- - - - - - - - - - - - - - - - +

: Payload Data continued ... :

+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +

| Payload Data continued ... |

+---------------------------------------------------------------+

FIN：1位

表示这是消息的最后一帧（结束帧），一个消息由一个或多个数据帧构成。若消息由一帧构成，起始帧即结束帧。

RSV1，RSV2，RSV3：各1位

MUST be 0 unless an extension is negotiated that defines meanings for non-zero values. If a nonzero value is received and none of the negotiated extensions defines the meaning of such a nonzero value, the receiving endpoint MUST \_Fail the WebSocket Connection\_.

这里我翻译不好，大致意思是如果未定义扩展，各位是0；如果定义了扩展，即为非0值。如果接收的帧此处非0，扩展中却没有该值的定义，那么关闭连接。

OPCODE：4位

解释PayloadData，如果接收到未知的opcode，接收端必须关闭连接。

0x0表示附加数据帧

0x1表示文本数据帧

0x2表示二进制数据帧

0x3-7暂时无定义，为以后的非控制帧保留

0x8表示连接关闭

0x9表示ping

0xA表示pong

0xB-F暂时无定义，为以后的控制帧保留

MASK：1位

用于标识PayloadData是否经过掩码处理。如果是1，Masking-key域的数据即是掩码密钥，用于解码PayloadData。客户端发出的数据帧需要进行掩码处理，所以此位是1。

Payload length：7位，7+16位，7+64位

PayloadData的长度（以字节为单位）。

如果其值在0-125，则是payload的真实长度。

如果值是126，则后面2个字节形成的16位无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

如果值是127，则后面8个字节形成的64位无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

长度表示遵循一个原则，用最少的字节表示长度（我理解是尽量减少不必要的传输）。举例说，payload真实长度是124，在0-125之间，必须用前7位表示；不允许长度1是126或127，然后长度2是124，这样违反原则。

Payload长度是ExtensionData长度与ApplicationData长度之和。ExtensionData长度可能是0，这种情况下，Payload长度即是ApplicationData长度。

WebSocket协议规定数据通过帧序列传输。

客户端必须对其发送到服务器的所有帧进行掩码处理。

服务器一旦收到无掩码帧，将关闭连接。服务器可能发送一个状态码是1002（表示协议错误）的Close帧。

而服务器发送客户端的数据帧不做掩码处理，一旦客户端发现经过掩码处理的帧，将关闭连接。客户端可能使用状态码1002。

### 消息分片

分片目的是发送长度未知的消息。如果不分片发送，即一帧，就需要缓存整个消息，计算其长度，构建frame并发送；使用分片的话，可使用一个大小合适的buffer，用消息内容填充buffer，填满即发送出去。

分片规则：

1.一个未分片的消息只有一帧（FIN为1，opcode非0）

2.一个分片的消息由起始帧（FIN为0，opcode非0），若干（0个或多个）帧（FIN为0，opcode为0），结束帧（FIN为1，opcode为0）。

3.控制帧可以出现在分片消息中间，但控制帧本身不允许分片。

4.分片消息必须按次序逐帧发送。

5.如果未协商扩展的情况下，两个分片消息的帧之间不允许交错。

6.能够处理存在于分片消息帧之间的控制帧

7.发送端为非控制消息构建长度任意的分片

8.client和server兼容接收分片消息与非分片消息

9.控制帧不允许分片，中间媒介不允许改变分片结构（即为控制帧分片）

10.如果使用保留位，中间媒介不知道其值表示的含义，那么中间媒介不允许改变消息的分片结构

11.如果协商扩展，中间媒介不知道，那么中间媒介不允许改变消息的分片结构，同样地，如果中间媒介不了解一个连接的握手信息，也不允许改变该连接的消息的分片结构

12.由于上述规则，一个消息的所有分片是同一数据类型（由第一个分片的opcode定义）的数据。因为控制帧不允许分片，所以一个消息的所有分片的数据类型是文本、二进制、opcode保留类型中的一种。

需要注意的是，如果控制帧不允许夹杂在一个消息的分片之间，延迟会较大，比如说当前正在传输一个较大的消息，此时的ping必须等待消息传输完成，才能发送出去，会导致较大的延迟。为了避免类似问题，需要允许控制帧夹杂在消息分片之间。

## python后台框架

Pandas、Numpy、Scipy 等等。撑起了 Python 在数学和数据分析领域的地位。

分布式服务、异步任务：Gearman、redis + celery

Diesel：基于Greenlet的事件I/O框架 Diesel提供一个整洁的API来编写网络客户端和服务器。支持TCP和UDP。

Cubes：轻量级Python OLAP框架 Cubes是一个轻量级Python框架，包含OLAP、多维数据分析和浏览聚合数据（aggregated data）等工具。

Pulsar是一个事件驱动的并发框架，有了pulsar，你可以写出在不同进程或线程中运行一个或多个活动的异步服务器.仅限于Python 3.5 and above

Falcon：构建云API和网络应用后端的高性能Python框架。它鼓励使用REST架构风格，尽可能以最少的力气做最多的事情。

Dpark：Python版的Spark。DPark是Spark的Python克隆，是一个Python实现的分布式计算框架，可以非常方便地实现大规模数据处理和迭代计算。DPark由豆瓣实现，目前豆瓣内部的绝大多数数据分析都使用DPark完成，正日趋完善。

Zerorpc：基于ZeroMQ的高性能分布式RPC框架。Zerorpc是一个基于ZeroMQ和MessagePack开发的远程过程调用协议（RPC）实现。和 Zerorpc 一起使用的 Service API 被称为 zeroservice。Zerorpc 可以通过编程或命令行方式调用。

Twisted

Twisted是一款事件驱动的网络编程框架，支持TCP，UDP，SSL / TLS，IP多点传播和大量的协议（包括HTTP，XMPP，NNTP，IMAP，SSH，IRC和FTP等等）。

## python其他

1. xrange

函数说明：用法与range完全相同，所不同的是生成的不是一个数组，而是一个生成器。

xrange则不会直接生成一个list，而是每次调用返回其中的一个值：

a = xrange(0,100)

print type(a)

print a

print a[0], a[1]

输出结果：

<type 'xrange'>

xrange(100) 所以xrange做循环的性能比range好，尤其是返回很大的时候，尽量用xrange吧，除非你是要返回一个列表。

0 1

1. 引用模块多重方案

try:

import cPickle as pickle

except ImportError:

import pickle

try:

import hashlib

sha1 = hashlib.sha1

except ImportError:

import sha

sha1 = sha.new

1. 当编译这则代码时，Python碰到给X赋值的语句时认为在这个函数中的任何地方X会被视作一个本地变量名。但是之后当真正运行这个函数时，执行print语句的时候，赋值语句还没有发生，这样Python便会报告一个“未定义变量名”的错误。

X = 99

def func():

print X # 这个时候还不存在

X = 88 # 在整个def中将X视作本地变量

func( ) # 出错了！

如果你真的是想要输出这个全局的X，你需要将它在一个全局语句中声明它，或者通过包络模块的名字来引用它。c

# python函数

## hasattr()

hasattr用于确定一个对象是否具有某个属性。

语法：

hasattr(object, name) -> bool

判断object中是否有name属性，返回一个布尔值。

## getattr()

Getattr用于返回一个对象属性，或者方法

class A:

def \_\_init\_\_(self):

self.name = 'zhangjing'

　　 #self.age='24'

def method(self):

print"method print"

Instance = A()

print getattr(Instance , 'name, 'not find') #如果Instance 对象中有属性name则打印self.name的值，否则打印'not find'

print getattr(Instance , 'age', 'not find') #如果Instance 对象中有属性age则打印self.age的值，否则打印'not find'

print getattr(a, 'method', 'default')

#如果有方法method，否则打印其地址，否则打印default

print getattr(a, 'method', 'default')()

#如果有方法method，运行函数并打印None否则打印default

注：使用getattr可以轻松实现工厂模式。

## setattr()

setattr( object, name, value)

这是相对应的getattr()。参数是一个对象,一个字符串和一个任意值。字符串可能会列出一个现有的属性或一个新的属性。这个函数将值赋给属性的。该对象允许它提供。

例如,setattr(x,“foobar”,123)相当于x.foobar = 123。

## delattr()

delattr( object, name)

与setattr()相关的一组函数。参数是由一个对象(记住python中一切皆是对象)和一个字符串组成的。string参数必须是对象属性名之一。该函数删除该obj的一个由string指定的属性。

delattr(x, 'foobar')=del x.foobar

## dir()

如果要获得一个对象的所有属性和方法，可以使用dir()函数，它返回一个包含字符串的list，比如，获得一个str对象的所有属性和方法：

>>> dir('ABC')

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_formatter\_field\_name\_split', '\_formatter\_parser', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

## repr()

obj==eval(repr(obj)) 这个等式是成立的。这两个函数接受一个对象作为其参数，返回适当的字符串。

事实上repr()和``做一样的事情，返回一个对象的“官方”字符串表示。其结果绝大多数情况下（不是所有）可以通过求值运算（内建函数eval()）重新得到该对象。

## 闭包

可以不返回求和的结果，而是返回求和的函数

def lazy\_sum(\*args):

def sum():

ax = 0

for n in args:

ax = ax + n

return ax

return sum

当我们调用lazy\_sum()时，返回的并不是求和结果，而是求和函数：

>>> f = lazy\_sum(1, 3, 5, 7, 9)

>>> f

<function sum at 0x10452f668>

调用函数f时，才真正计算求和的结果：

>>> f()

25

返回闭包时牢记的一点就是：返回函数不要引用任何循环变量，或者后续会发生变化的变量。

一定要引用循环变量怎么办？

方法是再创建一个函数，用该函数的参数绑定循环变量当前的值，无论该循环变量后续如何更改，已绑定到函数参数的值不变：

>>> def count():

... fs = []

... for i in range(1, 4):

... def f(j):

... def g():

... return j\*j

... return g

... fs.append(f(i))

... return fs

## 递归函数

在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数。

于是，fact(n)用递归的方式写出来就是：

def fact(n):

if n==1:

return 1

return n \* fact(n - 1)

使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。可以试试fact(1000)：

>>> fact(1000)

RuntimeError: maximum recursion depth exceeded

尾递归是指，**在函数返回的时候，调用自身本身，并且，return语句不能包含表达式。**这样，编译器或者解释器就可以把尾递归做优化，使递归本身无论调用多少次，都只占用一个栈帧，不会出现栈溢出的情况。

上面的fact(n)函数由于return n \* fact(n - 1)引入了乘法表达式，所以就不是尾递归了。要改成尾递归方式，需要多一点代码，主要是要把每一步的乘积传入到递归函数中：

def fact(n):

return fact\_iter(n, 1)

def fact\_iter(num, product):

if num == 1:

return product

return fact\_iter(num - 1, num \* product)

可以看到，return fact\_iter(num - 1, num \* product)仅返回递归函数本身，num - 1和num \* product在函数调用前就会被计算，不影响函数调用。

小结

使用递归函数的优点是逻辑简单清晰，缺点是过深的调用会导致栈溢出。

针对尾递归优化的语言可以通过尾递归防止栈溢出。尾递归事实上和循环是等价的，没有循环语句的编程语言只能通过尾递归实现循环。

# python模块编写

查看模块如何编写 最好方法就是进入python模块包中查看C:\Python27\Lib ，而不是去查看模块的安装包

根据大多数模块的语言命令规则，py文件都是小写，比如logwhitelist.py class 函数等 用大写字母分割class logWhiteList

变量用下划线分割 log\_whitelist

## setup.py文件的编写

setup.py中主要执行一个 setup函数，该函数中大部分是描述性东西，最主要的是packages参数，列出所有的package，可以用自带的find\_packages来动态获取package。所以setup.py文件的编写实际是很简单的。

不要增加scripts=["scripts/test.py"],因为安装不上。双引号内不要有空格。

from setuptools import setup, find\_packages

setup(

name = "foo" ,

version = "1.0" ,

packages = find\_packages(),

)

setup函数还有一些参数：

1. packages 告诉Distutils需要处理那些包（包含\_\_init\_\_.py的文件夹） packages = [ 'paramiko' ],列出打包涉及的文件及文件夹
2. package\_dir 告诉Distutils哪些目录下的文件被映射到哪个源码包。一个例子：package\_dir = {'': 'lib'}，表示“root package”中的模块都在lib目录中。
3. ext\_modules 是一个包含Extension实例的列表，Extension的定义也有一些参数。
4. ext\_package 定义extension的相对路径
5. requires 定义依赖哪些模块
6. provides 定义可以为哪些模块提供依赖
7. scripts 指定python源码文件，可以从命令行执行。在安装时指定--install-script scripts=["scripts/test.py"],
8. package\_data 通常包含与包实现相关的一些数据文件或类似于readme的文件。如果没有提供模板，会被添加到MANIFEST文件中。
9. data\_files 指定其他的一些文件（如配置文件）
10. install\_requires = ['six'] 安装过程中会自动安装six，如果安装six模块失败，则安装的该模块也不成功。

然后，运行python setup.py sdist为模块创建一个源码包

在当前目录下，会创建dist目录，里面有个文件名为foo-1.0.tar.gz，这个就是可以分发的包。使用者拿到这个包后，解压，到foo-1.0目录下执行：python setup.py install，那么，foo.py就会被拷贝到python类路径下，可以被导入使用。

对于Windows，可以执行python setup.py bdist\_wininst生成3个文件夹。其中最主要的是一个exe文件，在dist文件夹中。 windows中建议用这个安装 最后只需要将exe这个执行包给别人就行。

最后点击该exe文件安装该模块后在C:\Python27\Lib\site-packages 生成 updatehost-1.0-py2.7.egg

若要生成RPM包，执行python setup.py bdist\_rpm，但系统必须有rpm命令的支持。

## 模块包中\_\_init\_\_.py作用

1. 只要import redis 则\_\_init\_\_.py 中变量就可直接调用。

比如 redis.\_\_version\_\_

1. \_\_init\_\_.py中导入该模块包的其他py文件的类，主要目的是：使这些类可以直接调用，而无需再调用所在的py文件

比如 from redis.client import Redis

这样 在其他系统调用redis模块中的Redis类，无需from redis.client import Redis 可以直接 from redis import Redis

1. \_\_all\_\_变量为了使用者可以了解该模块有多少方法可供调用

比如 redis.\_\_all\_\_ 可显示出。

from redis.client import Redis, StrictRedis

from redis.connection import (

BlockingConnectionPool,

ConnectionPool,

Connection,

SSLConnection,

UnixDomainSocketConnection

)

from redis.utils import from\_url

from redis.exceptions import (

AuthenticationError,

BusyLoadingError,

ConnectionError,

DataError,

InvalidResponse,

PubSubError,

ReadOnlyError,

RedisError,

ResponseError,

TimeoutError,

WatchError

)

\_\_version\_\_ = '2.10.5'

VERSION = tuple(map(int, \_\_version\_\_.split('.')))

\_\_all\_\_ = [

'Redis', 'StrictRedis', 'ConnectionPool', 'BlockingConnectionPool',

'Connection', 'SSLConnection', 'UnixDomainSocketConnection', 'from\_url',

'AuthenticationError', 'BusyLoadingError', 'ConnectionError', 'DataError',

'InvalidResponse', 'PubSubError', 'ReadOnlyError', 'RedisError',

'ResponseError', 'TimeoutError', 'WatchError'

]

## 模块中导入的模块

import pika, pyyaml , gevent, haigha , greenlet , requests , six

import select ,urlparse, pika ,yaml

import socket

import struct

import base64, cgi, email.utils, functools, hmac, imp, itertools, mimetypes,\

os, re, subprocess, sys, tempfile, threading, time, warnings

from optparse import OptionParser

from datetime import date as datedate, datetime, timedelta

from tempfile import TemporaryFile

from traceback import format\_exc, print\_exc

from inspect import getargspec

from unicodedata import normalize

import pickle

from io import BytesIO

from logging import getLogger

from math import modf

import Queue

from SimpleHTTPServer import SimpleHTTPRequestHandler

import webapi as web

import net

import utils

import SimpleHTTPServer, SocketServer, BaseHTTPServer, urlparse

import errno

import traceback

import glob

import tokenize

import types

import compiler

Pillow greenlet 以及其基础之上的eventlet gevent 微线程并发库

## 常用模块列表

standard libs:

itertools http://docs.python.org/2/library/itertools.html

functools http://docs.python.org/2/library/functools.html 学好python有必要掌握上面这两个库吧，

re 正则

subprocess http://docs.python.org/2/library/subprocess.html 调用shell命令的神器

pdb 调试

traceback 调试

pprint 漂亮的输出

logging 日志

threading和multiprocessing 多线程

urllib/urllib2/httplib http库，httplib底层一点，推荐第三方的库requests

os/sys 系统，环境相关

Queue 队列

pickle/cPickle 序列化工具

hashlib md5, sha等hash算法

cvs

json/simplejson python的json库，据so上的讨论和benchmark，simplejson的性能要高于json

timeit 计算代码运行的时间等等

cProfile python性能测量模块

glob 类似与listfile，可以用来查找文件

atexit 有一个注册函数，可用于正好在脚本退出运行前执行一些代码

dis python 反汇编，当对某条语句不理解原理时，可以用dis.dis 函数来查看代码对应的python 解释器指令等等。

3th libs:

paramiko https://github.com/paramiko/paramiko ssh python 库

selenium 浏览器自动化测试工具selenium的python 接口

lxml http://lxml.de/ python 解析html,xml 的神器

mechanize <https://pypi.python.org/pypi/mechanize/> 模拟自然的浏览器浏览网站 爬虫相关

pycurl https://pypi.python.org/pypi/pycurl cURL library module for Python

Fabric <http://docs.fabfile.org/en/1.8/> ssh命令模块 streamlining the use of SSH for application deployment or systems administration tasks.

xmltodict https://github.com/martinblech/xmltodict xml 转 dict，真心好用

urllib3 和 requests: 当然其实requests就够了 Requests: HTTP for Humans

flask http://flask.pocoo.org/python web 微框架

ipdb 调试神器，同时推荐ipython！结合ipython使用

redis redis python接口

pymongo mongodbpython接口

PIL http://www.pythonware.com/products/pil/ python图像处理

mako http://www.makotemplates.org/ python模版引擎

numpy ， scipy 科学计算

matplotlib 画图

scrapy 爬虫

django/tornado/web.py/web2py/uliweb/flask/twisted/bottle/cherrypy.等等 python web框架/服务器

sh 1.08 — sh v1.08 documentation 用来运行shell 模块的 极佳选择

Docopt 忘了optparse和argparse吧，使用docstring来构建优雅的、高可读性、复杂（如果你有这个需要）的命令行界面。在我看来这是2013年诞生的最好的库。

Requests 或曰，为人类准备的HTTP，以更pythonic的方式处理HTTP请求。比urllib2好用得多得多得多。PyPI下载已经超过5,000,000次，这可不是没有理由的哦:)

lxml联合了libxml2和libxslt。如果你要处理XML或HTML，lxml是最好的选择，真的。

Bottle是一个快速、简单、轻量的WSGI微型web框架。利用Bottle构建小型站点和API的时间以秒计算。这个框架只有一个py文件，你可以把它放进任何目录。

sh是一个成熟的Python子进程接口，允许你像调用函数一样调用任何程序。超级好用。

Structlog是一个先进的日志处理器。他可以和任何现存的日志记录工具相集成，并包装了Python标准库。你可以构建定制的记录工具，根据你的需要增加上下文，保证你的日志一致、可读。

Watchdog是一个跨平台的Python库和shell工具，可以监视文件系统事件。超级好用，容易上手。

Delorean用它处理日期和时间非常方便。设置时区，截取到秒、分、小时，甚至使用特定步骤从一个日期进到另一个日期。浏览下文档，里面有很多例子。

核心模块

\_ \_builtin\_ \_ 模块, exceptions 模块, os 模块, os.path 模块, stat 模块, string 模块, re 模块, math 模块, cmath 模块, operator 模块, copy 模块, sys 模块, atexit 模块, time 模块, types 模块, gc 模块

更多标准模块

fileinput 模块, shutil 模块, tempfile 模块, StringIO 模块, cStringIO 模块, mmap 模块, UserDict 模块, UserList 模块, UserString 模块, traceback 模块, errno 模块, getopt 模块, getpass 模块, glob 模块, fnmatch 模块, random 模块, whrandom 模块, md5 模块, sha 模块, crypt 模块, rotor 模块, zlib 模块, code 模块,

线程和进程

threading 模块, Queue 模块, thread 模块, commands 模块, pipes 模块, popen2 模块, signal 模块

数据表示

array 模块, struct 模块, xdrlib 模块, marshal 模块, pickle 模块, cPickle 模块, copy\_reg 模块, pprint 模块, repr 模块, base64 模块, binhex 模块, quopri 模块, uu 模块, binascii 模块

文件格式

xmllib 模块, xml.parsers.expat 模块, sgmllib 模块, htmllib 模块, htmlentitydefs 模块, formatter 模块, ConfigParser 模块, netrc 模块, shlex 模块, zipfile 模块, gzip 模块

邮件和新闻消息处理

rfc822 模块, mimetools 模块, MimeWriter 模块, mailbox 模块, mailcap 模块, mimetypes 模块, packmail 模块, mimify 模块, multifile 模块,

网络协议

socket 模块, select 模块, asyncore 模块, asynchat 模块, urllib 模块, urlparse 模块, cookie 模块, robotparser 模块, ftplib 模块, gopherlib 模块, httplib 模块, poplib 模块, imaplib 模块, smtplib 模块, telnetlib 模块, nntplib 模块, SocketServer 模块, BaseHTTPServer 模块, SimpleHTTPServer 模块, CGIHTTPServer 模块

cgi 模块, webbrowser 模块

国际化

locale 模块, unicodedata 模块, ucnhash 模块,

多媒体相关模块

imghdr 模块, sndhdr 模块, whatsound 模块, aifc 模块, sunau 模块, sunaudio 模块, wave 模块

audiodev 模块, winsound 模块

数据储存

anydbm 模块, whichdb 模块, shelve 模块, dbhash 模块, dbm 模块, dumbdbm 模块, gdbm 模块

工具和实用程序

dis 模块, pdb 模块, bdb 模块, profile 模块, pstats 模块, tabnanny 模块

其他模块

fcntl 模块, pwd 模块, grp 模块, nis 模块, curses 模块, termios 模块, tty 模块, resource 模块, syslog 模块, msvcrt 模块, nt 模块, \_winreg 模块, posix 模块

执行支持模块

dospath 模块, macpath 模块, ntpath 模块, posixpath 模块, strop 模块, imp 模块, new 模块, pre 模块, sre 模块, py\_compile 模块, compileall 模块, ihooks 模块, linecache 模块, macurl2path 模块, nturl2path 模块, tokenize 模块, keyword 模块, parser 模块, symbol 模块, token 模块,

其他模块

pyclbr 模块, filecmp 模块, cmd 模块, rexec 模块, Bastion 模块, readline 模块, rlcompleter 模块, statvfs 模块, calendar 模块, sched 模块, statcache 模块, grep 模块, dircache 模块, dircmp 模块, cmp 模块, cmpcache 模块, util 模块, soundex 模块, timing 模块, posixfile 模块, bisect 模块, knee 模块, tzparse 模块, regex 模块, regsub 模块, reconvert 模块, regex\_syntax 模块, find 模块

## 问题解决

1. 安装模块时报错 error: Unable to find vcvarsall.bat

If you have Visual Studio 2010 installed, execute

SET VS90COMNTOOLS=%VS100COMNTOOLS%

or with Visual Studio 2012 installed (Visual Studio Version 11)

SET VS90COMNTOOLS=%VS110COMNTOOLS%

or with Visual Studio 2013 installed (Visual Studio Version 12)

SET VS90COMNTOOLS=%VS120COMNTOOLS%

安装VS后该重启的重启，clean一下之前安装Python扩展失败的残留文件，然后 直接下载 pil pillow greenlet eventlet等源码，解压后python setup.py build发现都可以编译了。接下来就换成 python setup.py install安装吧。

1. 编写类时，需要导入参数，一般这么做

def \_\_init\_\_(self, resolver):

self.resolver = resolver #导入的参数在\_init\_\_转为self前缀变量，这样才能被该类中其他的方法下使用

def build(self, name):

if isinstance(name, basestring): #一般导入参数会判断传入的参数是否正确，然后才进入逻辑判断

type = self.resolver.find(name)

1. 在\_\_init\_\_.py中一般会自定义exceptions

class MethodNotFound(Exception):

def \_\_init\_\_(self, name):

Exception.\_\_init\_\_(self, "Method not found: '%s'" % name)

1. 可以在class入参中默认值

def entry(self, name, key=None, cmd=None, focus=False):

## 细节

1. class类名为多个

def Connect(\*args, \*\*kwargs):

"""Factory function for connections.Connection."""

from MySQLdb.connections import Connection

return Connection(\*args, \*\*kwargs)

connect = Connection = Connect

1. 可以专门起一个py文件，专门存变量
2. 外作用域显得非常强大。给外作用域中的任何变量使用有特色的名字是明智的做法，例如，使用IN\_ALL\_CAPS 这个常量名，这样可以避免不必要的bug
3. 测试是否为None

if x is None:print x

测试是否为空

if x:print x

# python模块

<http://python.usyiyi.cn/python_278/library/index.html> 标准库中文介绍

python setup.py build && python setup.py install

import redisir(redis) 查看redis中有多少方法 注意请不要加引号

dir(redis.Redis) 查看redis中Redis方法中有多少方法可以调用

1. 打包代码为模块安装包 python setup.py install
2. 安装whl格式时： pip install mock-1.3.0-py2.py3-none-any.whl
3. 查看redis模块的帮助 help(redis)

client connection为py文件 StrictRedis为client.py文件中的class类

1. 导入该类语句 from redis.client import Redis

CLASSES

\_\_builtin\_\_.object

redis.client.StrictRedis

redis.client.Redis

redis.connection.Connection

redis.connection.SSLConnection

1. 导入多个类

from redis.exceptions import (

AuthenticationError,

BusyLoadingError,

ConnectionError)

## 常用模块介绍

python除了关键字（keywords）和内置的类型和函数（builtins）,更多的功能是通过libraries（即modules）来提供的。

以下为第三方模块，其他为标准模块

cmmands、configParserwinreg、fcntl、queue、socketserver、xmlrpc、htmlparser

常用的libraries（modules）如下：

1）python运行时服务

\* copy： copy模块提供了对复合（compound）对象（list，tuple，dict，custom class）进行浅拷贝和深拷贝的功能。

\* pickle： pickle模块被用来序列化python的对象到bytes流，从而适合存储到文件，网络传输，或数据库存储。（pickle的过程也被称serializing,marshalling或者flattening，pickle同时可以用来将bytes流反序列化为python的对象）。

\* sys：sys模块包含了跟python解析器和环境相关的变量和函数。

\* 其他： atexit，gc，inspect，marshal，traceback，types，warnings，weakref。

2）数学

\* decimal：python中的float使用双精度的二进制浮点编码来表示的，这种编码导致了小数不能被精确的表示，例如0.1实际上内存中为0.100000000000000001，还有3\*0.1 == 0.3 为False. decimal就是为了解决类似的问题的，拥有更高的精确度，能表示更大范围的数字，更精确地四舍五入。

\* math：math模块定义了标准的数学方法，例如cos(x),sin(x)等。

\* random：random模块提供了各种方法用来产生随机数。

\* 其他：fractions，numbers。

3）数据结构，算法和代码简化

\* array： array代表数组，类似与list，与list不同的是只能存储相同类型的对象。

\* bisect： bisect是一个有序的list，其中内部使用二分法（bitsection）来实现大部分操作。

\* collections：collections模块包含了一些有用的容器的高性能实现，各种容器的抽象基类，和创建name-tuple对象的函数。例如包含了容器deque，defaultdict，namedtuple等。

\* heapq：heapq是一个使用heap实现的带有优先级的queue。

\* itertools：itertools包含了函数用来创建有效的iterators。所有的函数都返回iterators或者函数包含iterators（例如generators 和generators expression）。

\* operator： operator提供了访问python内置的操作和解析器提供的特殊方法，例如 x+y 为 add（x，y），x+=y为iadd（x，y），a % b 为mod（a，b）等等。

\* 其他：abc，contextlib，functools。

4) string 和 text 处理

\*codecs：codecs模块被用来处理不同的字符编码与unicode text io的转化。

\* re：re模块用来对字符串进行正则表达式的匹配和替换。

\* string：string模块包含大量有用的常量和函数用来处理字符串。也包含了新字符串格式的类。

\* struct：struct模块被用来在python和二进制结构间实现转化。

\* unicodedata：unicodedata模块提供访问unicode字符数据库

5) python数据库访问

\* 关系型数据库拥有共同的规范Python Database API Specification V2.0，MySQL，Oracle等都实现了此规范，然后增加自己的扩展。

\* sqlite3: sqlite3 模块提供了SQLite数据库访问的接口。SQLite数据库是以一个文件或内存的形式存在的自包含的关系型数据库。

\* DBM-style 数据库模块：python提供了打了的modules来支持UNIX DBM-style数据库文件。dbm模块用来读取标准的UNIX-dbm数据库文件，gdbm用来读取GNU dbm数据库文件，dbhash用来读取Berkeley DB数据库文件。所有的这些模块提供了一个对象实现了基于字符串的持久化的字典，他与字典dict非常相似，但是他的keys和values都必须是字符串。

\* shelve：shelve模块使用特殊的“shelf”对象来支持持久化对象。这个对象的行为与dict相似，但是所有的他存储的对象都使用基于hashtable的数据库（dbhash，dbm，gdbm）存储在硬盘。与dbm模块的区别是所存储的对象不仅是字符串，而且可以是任意的与pickle兼容的对象。

6）文件和目录处理

\* bz2：bz2模块用来处理以bzip2压缩算法压缩的文件。

\* filecmp：filecmp模块提供了函数来比较文件和目录。

\* fnmatch：fnmatch模块提供了使用UNIX shell-style的通配符来匹配文件名。这个模块只是用来匹配，使用glob可以获得匹配的文件列表。

\* glob：glob模块返回了某个目录下与指定的UNIX shell通配符匹配的所有文件。

\* gzip：gzip模块提供了类GzipFile，用来执行与GNUgzip程序兼容的文件的读写。

\* shutil： shutil模块用来执行更高级别的文件操作，例如拷贝，删除，改名。shutil操作之针对一般的文件，不支持pipes，block devices等文件类型。

\* tarfile： tarfile模块用来维护tar存档文件。tar没有压缩的功能。

\* tempfile：tempfile模块用来产生临时文件和文件名。

\* zipfile： zipfile模块用来处理zip格式的文件。

\* zlib，zlib模块提供了对zlib库的压缩功能的访问。

7）操作系统的服务

\* cmmands： commands模块被用来执行简单的系统命令，命令以字符串的形式传入，且同时以字符串的形式返回命令的输出。但是此模块只在UNIX系统上可用。

\* configParser，configParser模块用来读写windows的ini格式的配置文件。

\* datetime，datetime模块提供了各种类型来表示和处理日期和时间。

\* errno， 定义了所有的errorcode对应的符号名字。

\* io，io模块实现了各种IO形式和内置的open()函数。

\* logging， logging模块灵活方便地对应用程序记录events，errors，warnings，和debuging 信息。这些log信息可以被收集，过滤，写到文件或系统log，甚至通过网络发送到远程的机器上。

\*mmap，mmap模块提供了内存映射文件对象的支持，使用内存映射文件与使用一般的文件或byte字符串相似。

\*msvcrt，mscrt只可以在windows系统使用，用来访问Visual C运行时库的很多有用的功能。

\*optparse，optparse模块更高级别来处理UNIX style的命令行选项sys.argv。

\* os，os模块对通用的操作系统服务提供了可移植的（portable）的接口。os可以认为是nt和posix的抽象。nt提供windows的服务接口，posix提供UNIX（linux，mac）的服务接口。

\* os.path，os.path模块以可移植的方式来处理路径相关的操作。

\* signal，signal模块用来实现信号（signal）处理，往往跟同步有关。

\* subprocess，subprocess模块包含了函数和对象来统一创建新进程，控制新进程的输入输出流，处理进程的返回。

\* time，time模块提供了各种时间相关的函数。常用的time.sleep().

\* winreg, winreg模块用来操作windows注册表。

\* 其他：fcntl。

8）线程和并行

\* multiprocessing，multiprocessing模块提供通过subprocess来加载多个任务，通信，共享数据，执行各种同步操作。

\* threading，threading模块提供了thread类很很多的同步方法来实现多线程编程。

\* queue，queue模块实现了各种多生产者，多消费者队列，被用来实现多线程程序的信息安全交换。

\* 其他：Coroutines and Microthreading。

9）网络编程和套接字（sockets）

\* asynchat，asynchat模块通过封装asyncore来简化了应用程序的网络异步处理。

\* ssl，ssl模块被用来使用secure sockets layer（SSL）包装socket对象，从而使得实现数据加密和终端认证。python使用openssl来实现此模块。

\* socketserver，socketserver模块提供了类型简化了TCP，UDP和UNIX领域的socket server的实现。

\* 其他：asyncore，select。

10）internet应用程序编程

\* ftplib，ftplib模块实现了ftp的client端协议。此模块很少使用，因为urllib提供了更高级的接口。

\* http包，包含了http client和server的实现和cookies管理的模块。

\* smtplib，smtplib包含了smtp client的底层接口，用来使用smtp协议发送邮件。

\* urllib，urllib包提供了高级的接口来实现与http server，ftp server和本地文件交互的client。

\* xmlrpc，xmlrpc模块被用类实现XML-RPC client。

11）web 编程

\* cgi，cgi模块用来实现cgi脚本，cgi程序一般地被webserver执行，用来处理用户在form中的输入，或生成一些动态的内容。当与cgi脚本有管的request被提交，webserver将cgi作为子进程执行，cgi程序通过sys.stdin或环境变量来获得输入，通过sys.stdout来输出。

\* webbrowser，webbrowser模块提供了平台独立的工具函数来使用web browser打开文档。

\* 其他：wsgiref/WSGI (Python Web Server Gateway Interface).

12) internet 数据处理和编码

\* base64，base64模块提供了base64，base32，base16编码方式，用来实现二进制与文本间的编码和解码。base64通常用来对编码二进制数据，从而嵌入到邮件或http协议中。

\* binascii，binascii模块提供了低级的接口来实现二进制和各种ASCII编码的转化。

\* csv，csv模块用来读写comma-separated values（CSV）文件。

\* email，email包提供了大量的函数和对象来使用MIME标准来表示，解析和维护email消息。

\* hashlib，hashlib模块实现了各种secure hash和message digest algorithms，例如MD5和SHA1。

\* htmlparser（html.parser），此模块定义了HTMLParser来解析HTML和XHTML文档。使用此类，需要定义自己的类且继承于HTMLParser。

\* json，json模块被用类序列化或饭序列化Javascript object notation（JSON）对象。

\* xml,xml包提供了各种处理xml的方法。

Sphinx – Python 文档生成器。

Pycco – 文学编程（literate-programming）风格的文档生成器。

## 有关多线程多进程

### Queue模块

Python中，队列是线程间最常用的交换数据的形式。Queue模块是提供队列操作的模块，虽然简单易用，但是不小心的话，还是会出现一些意外。

这些队列都实现了锁原语，能够在多线程中直接使用。可以使用队列来实现线程间的同步。

Queue模块实现了多生产者多消费者队列, 尤其适合多线程编程.Queue类中实现了所有需要的锁原语(这句话非常重要), Queue模块实现了三种类型队列:

FIFO(先进先出)队列, 第一加入队列的任务, 被第一个取出

LIFO(后进先出)队列,最后加入队列的任务, 被第一个取出

PriorityQueue(优先级)队列, 保持队列数据有序, 最小值被先取出

import Queue

#类

Queue.Queue(maxsize = 0) #构造一个FIFO队列,maxsize设置队列大小的上界, 如果插入数据时, 达到上界会发生阻塞, 直到队列可以放入数据. 当maxsize小于或者等于0, 表示不限制队列的大小(默认)

Queue.LifoQueue(maxsize = 0) #构造一LIFO队列,maxsize设置队列大小的上界, 如果插入数据时, 达到上界会发生阻塞, 直到队列可以放入数据. 当maxsize小于或者等于0, 表示不限制队列的大小(默认)

Queue.PriorityQueue(maxsize = 0) #构造一个优先级队列,,maxsize设置队列大小的上界, 如果插入数据时, 达到上界会发生阻塞, 直到队列可以放入数据. 当maxsize小于或者等于0, 表示不限制队列的大小(默认). 优先级队列中, 最小值被最先取出

创建一个“队列”对象

import Queue

q = Queue.Queue(maxsize = 10)

Queue.Queue类即是一个队列的同步实现。队列长度可为无限或者有限。可通过Queue的构造函数的可选参数maxsize来设定队列长度。如果maxsize小于1就表示队列长度无限。

将一个值放入队列中

q.put(10)

调用队列对象的put()方法在队尾插入一个项目。put()有两个参数，第一个item为必需的，为插入项目的值；第二个block为可选参数，默认为1。如果队列当前为满且block为1，put()方法就使调用线程暂停,直到空出一个数据单元。如果block为0，put方法将引发Full异常。

将一个值从队列中取出

q.get()

调用队列对象的get()方法从队头删除并返回一个项目。可选参数为block，默认为True。如果队列为空且block为True，get()就使调用线程暂停，直至有项目可用。如果队列为空且block为False，队列将引发Empty异常。

此包中的常用方法(q = Queue.Queue()):

q.qsize() 返回队列的大小

q.empty() 如果队列为空，返回True,反之False

q.full() 如果队列满了，返回True,反之False

q.full 与 maxsize 大小对应

q.get([block[, timeout]]) 获取队列，timeout等待时间

q.get\_nowait() 相当q.get(False)

非阻塞 q.put(item) 写入队列，timeout等待时间

q.put\_nowait(item) 相当q.put(item, False)

q.task\_done() 在完成一项工作之后，q.task\_done() 函数向任务已经完成的队列发送一个信号

q.join() 实际上意味着等到队列为空，再执行别的操作

在主线程中使用了 queue.join()，导致主线程阻塞。queue.task\_done() 表示完成一个 task，并递减没有完成的队列数，当队列全部完成时候，没有task可执行，因此需要发送一个信号，通知被阻塞的主线程，继续运行。

Queue.put(item[, block[, timeout]]) #向队列中放入元素, 如果可选参数block为True并且timeout参数为None(默认), 为阻塞型put(). 如果timeout是正数, 会阻塞timeout时间并引发Queue.Full异常. 如果block为False为非阻塞put

Queue.put\_nowait(item) #等价于put(itme, False)

Queue.get([block[, timeout]]) #移除列队元素并将元素返回, block = True为阻塞函数, block = False为非阻塞函数. 可能返回Queue.Empty异常

Queue.get\_nowait() #等价于get(False)

Queue.task\_done() #在完成一项工作之后，Queue.task\_done()函数向任务已经完成的队列发送一个信号

Queue.join() #实际上意味着等到队列为空，再执行别的操作

Queue和线程结合的小程序

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

import threading

import time

import Queue

SHARE\_Q=Queue.Queue()

\_WORKER\_THREAD\_NUM=3

class MyThread(threading.Thread):

def \_\_init\_\_(self,func):

super(MyThread,self).\_\_init\_\_()

self.func=func

def run(self):

self.func()

def worker():

global SHARE\_Q

while not SHARE\_Q.empty():

item=SHARE\_Q.get()

print 'Processing:',item

time.sleep(1)

def main():

global SHARE\_Q

threads=[]

for task in xrange(5):

SHARE\_Q.put(task)

for i in xrange(\_WORKER\_THREAD\_NUM):

thread=MyThread(worker)

thread.start()

threads.append(thread)

for thread in threads:

thread.join()

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

main()

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

import Queue

import threading

import time

import random

q=Queue.Queue()

NUM\_WORKERS=3

class MyThread(threading.Thread):

def \_\_init\_\_(self,input,worktype):

self.\_jobq=input

self.\_work\_type=worktype

threading.Thread.\_\_init\_\_(self)

def run(self):

while True:

if self.\_jobq.qsize()>0:

job=self.\_jobq.get()

worktype=self.\_work\_type

self.\_process\_job(job,worktype)

else:

break

def \_process\_job(self,job,worktype):

doJob(job)

def doJob(job):

time.sleep(random.random()\*3)

print 'doing',job

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

print 'begin...'

for i in range(NUM\_WORKERS\*2):

q.put(i)

print 'job size',q.qsize()

for x in range(NUM\_WORKERS):

MyThread(q,x).start()

#### FIFO

from queue import Queue

q = Queue()

for i in range(3):

q.put(i)

while not q.empty():

print(q.get())

#### LIFO

from queue import LifoQueue

q = LifoQueue()

for i in range(3):

q.put(i)

while not q.empty():

print(q.get())

#### PriorityQueue

from queue import PriorityQueue

class Job(object):

def \_\_init\_\_(self, priority, description):

self.priority = priority

self.description = description

print('New job:', description)

return

def \_\_lt\_\_(self, other):

return self.priority < other.priority

q = PriorityQueue()

q.put(Job(5, 'Mid-level job'))

q.put(Job(10, 'Low-level job'))

q.put(Job(1, 'Important job'))

while not q.empty():

next\_job = q.get()

print('Processing job', next\_job.description)

### threading模块

#### 介绍

threading提供了一个比thread模块更高层的API来提供线程的并发性。这些线程并发运行并共享内存。

thread：多线程的底层支持模块，一般不建议使用；

threading：对thread进行了封装，将一些线程的操作对象化

t = threading.Thread(target=worker)

t.start()

Thread 线程类，这是我们用的最多的一个类，你可以指定线程函数执行或者继承自它都可以实现子线程功能；

Timer与Thread类似，但要等待一段时间后才开始运行；

Lock 锁原语，这个我们可以对全局变量互斥时使用；

RLock 可重入锁，使单线程可以再次获得已经获得的锁；

Condition 条件变量，能让一个线程停下来，等待其他线程满足某个“条件”；

Event 通用的条件变量。多个线程可以等待某个事件发生，在事件发生后，所有的线程都被激活；

Semaphore为等待锁的线程提供一个类似“等候室”的结构；

BoundedSemaphore 与semaphore类似，但不允许超过初始值；

Queue：实现了多生产者（Producer）、多消费者（Consumer）的队列，支持锁原语，能够在多个线程之间提供很好的同步支持。

其中Thread类

是你主要的线程类，可以创建进程实例。该类提供的函数包括：

getName(self) 返回线程的名字

isAlive(self) 布尔标志，表示这个线程是否还在运行中

isDaemon(self) 返回线程的daemon标志

join(self, timeout=None) 程序挂起，直到线程结束，如果给出timeout，则最多阻塞timeout秒

run(self) 定义线程的功能函数

setDaemon(self, daemonic) 把线程的daemon标志设为daemonic

setName(self, name) 设置线程的名字

start(self) 开始线程执行

其中Queue提供的类

Queue队列

LifoQueue后入先出（LIFO）队列

PriorityQueue 优先队列

#### 将函数传递进Thread对象

import threading

import time

def worker():

print "worker"

time.sleep(1)

return

for i in xrange(5):

t = threading.Thread(target=worker)

t.start()

import threading

def thread\_fun(num):

for n in range(0, int(num)):

print " I come from %s, num: %s" %( threading.currentThread().getName(), n)

def main(thread\_num):

thread\_list = list();

# 先创建线程对象

for i in range(0, thread\_num):

thread\_name = "thread\_%s" %i

thread\_list.append(threading.Thread(**target = thread\_fun**, name = thread\_name, args = (20,)))

# 启动所有线程

for thread in thread\_list:

thread.start()

# 主线程中等待所有子线程退出

for thread in thread\_list:

thread.join()

#### 模板 用Threading模块创建线程

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

import threading

import time

import thread

exitFlag=0

class myThread(threading.Thread):#继承父类threading.Thread

def \_\_init\_\_(self,threadID,name,counter):

threading.Thread.\_\_init\_\_(self)

self.threadID=threadID

self.name=name

self.counter=counter

def run(self): #把要执行的代码写到run函数里面 线程在创建后会直接运行run函数

print 'starting'+self.name

print\_time(self.name,self.counter,5)

print 'Exiting'+self.name

def print\_time(threadName,delay,counter):

while counter:

if exitFlag:

thread.exit()

time.sleep(delay)

print "%s: %s" % (threadName, time.ctime(time.time()))

counter-=1

thread1=myThread(1,'Thread-1',1)

thread2=myThread(2,'Thread-2',2)

thread1.start()

thread2.start()

print 'Exiting Main Thread'

简单版

import threading

class MyThread(threading.Thread):

def \_\_init\_\_(self):

threading.Thread.\_\_init\_\_(self);

def run(self):

print "I am %s" %self.name

if \_\_name\_\_ == "\_\_main\_\_":

for thread in range(0, 5):

t = MyThread()

t.start()

#### 重要 threading中处理主进程和子线程的关系

如何使多线程都运行完后才运行后面的主进程的函数？

**使用join函数后，主进程会在调用join的地方等待子线程结束，然后才接着往下执行。**

join使用实例如下：

import time

import random

import threading

class worker(threading.Thread):

def \_\_init\_\_(self):

threading.Thread.\_\_init\_\_(self)

def run(self):

t = random.randint(1,10)

time.sleep(t)

print "This is " + self.getName() + ";I sleep %d second."%(t)

tsk = []

for i in xrange(0,5):

time.sleep(0.1)

thread = worker()

thread.start()

tsk.append(thread)

**for tt in tsk:**

**tt.join()**

print "This is the end of main thread."

运行结果如下：

# python testjoin.py

This is Thread-3;I sleep 2 second.

This is Thread-1;I sleep 4 second.

This is Thread-2;I sleep 7 second.

This is Thread-4;I sleep 7 second.

This is Thread-5;I sleep 7 second.

This is the end of main thread.

这里创建了5个子线程，每个线程随机等待1-10秒后打印退出；主线程分别等待5个子线程结束。最后结果是先显示各个子线程，再显示主进程的结果。

2. 如果使用的setDaemon函数，则与join相反，主进程结束的时候不会等待子线程。

setDaemon函数使用实例：

import time

import random

import threading

class worker(threading.Thread):

def \_\_init\_\_(self):

threading.Thread.\_\_init\_\_(self)

def run(self):

t = random.randint(1,10)

time.sleep(t)

print "This is " + self.getName() + ";I sleep %d second."%(t)

tsk = []

for i in xrange(0,5):

time.sleep(0.1)

thread = worker()

thread.setDaemon(True)

thread.start()

tsk.append(thread)

print "This is the end of main thread."

这里设置主进程为守护进程，当主进程结束的时候，子线程被中止

运行结果如下：

#python testsetDaemon.py

This is the end of main thread.

3、如果没有使用join和setDaemon函数，则主进程在创建子线程后，直接运行后面的代码，主程序一直挂起，直到子线程结束才能结束。

import time

import random

import threading

class worker(threading.Thread):

def \_\_init\_\_(self):

threading.Thread.\_\_init\_\_(self)

def run(self):

t = random.randint(1,10)

time.sleep(t)

print "This is " + self.getName() + ";I sleep %d second."%(t)

tsk = []

for i in xrange(0,5):

time.sleep(0.1)

thread = worker()

thread.start()

tsk.append(thread)

print "This is the end of main thread."

运行结果如下：

# python testthread.py

This is the end of main thread.

This is Thread-4;I sleep 1 second.

This is Thread-3;I sleep 7 second.

This is Thread-5;I sleep 7 second.

This is Thread-1;I sleep 10 second.

This is Thread-2;I sleep 10 second.

#### 线程同步

如果多个线程共同对某个数据修改，则可能出现不可预料的结果，为了保证数据的正确性，需要对多个线程进行同步。

这两个对象都有acquire方法和release方法，对于那些需要每次只允许一个线程操作的数据，可以将其操作放到acquire和release方法之间。

锁有两种状态——锁定和未锁定。每当一个线程比如"set"要访问**共享数据**时，必须先获得锁定；如果已经有别的线程比如"print"获得锁定了，那么就让线程"set"暂停，也就是同步阻塞；等到线程"print"访问完毕，释放锁以后，再让线程"set"继续。

threadLock = threading.Lock()

def run(self):

print "Starting " + self.name

# 获得锁，成功获得锁定后返回True

# 可选的timeout参数不填时将一直阻塞直到获得锁定

# 否则超时后将返回False

threadLock.acquire()

print\_time(self.name, self.counter, 3)

# 释放锁

threadLock.release()

#### threading.Thread与Queue

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

import Queue

import threading

import time

exitFlag=0

class myThread(threading.Thread):

def \_\_init\_\_(self,threadID,name,q):

threading.Thread.\_\_init\_\_(self)

self.threadID=threadID

self.name=name

self.q=q

def run(self):

print 'Starting'+self.name

process\_data(self.name,self.q)

print'Exiting'+self.name

def process\_data(threadName,q):

while not exitFlag:

queueLock.acquire()

if not workQueue.empty():

data=q.get()

queueLock.release()

print "%s processing %s" % (threadName, data)

else:

queueLock.release()

time.sleep(1)

threadList= ["Thread-1", "Thread-2", "Thread-3"]

nameList=["One", "Two", "Three", "Four", "Five"]

queueLock=threading.Lock()

workQueue=Queue.Queue(10)

threads=[]

threadID=1

# 创建新线程

for tName in threadList:

thread=myThread(threadID,tName,workQueue)

thread.start()

threads.append(thread)

threadID+=1

# 填充队列

queueLock.acquire()

for word in nameList:

workQueue.put(word)

queueLock.release()

# 等待队列清空

while not workQueue.empty():

pass

# 通知线程是时候退出

exitFlag=1

# 等待所有线程完成

for t in threads:

t.join()

print 'exiting main thread'

#### 继承threading.Thread类，重写它的run方法

有两种方式来创建线程：一种是通过继承Thread类，重写它的run方法；另一种是创建一个threading.Thread对象，在它的初始化函数（\_\_init\_\_）中将可调用对象作为参数传入。

#coding=gbk

import threading, time, random

count = 0

class Counter(threading.Thread):

def \_\_init\_\_(self, lock, threadName):

'''@summary: 初始化对象。

@param lock: 琐对象。

@param threadName: 线程名称。

'''

super(Counter, self).\_\_init\_\_(name = threadName) #注意：一定要显式的调用父类的初始

化函数。

self.lock = lock

def run(self):

'''@summary: 重写父类run方法，在线程启动后执行该方法内的代码。

'''

global count

self.lock.acquire()

for i in xrange(10000):

count = count + 1

self.lock.release()

lock = threading.Lock()

for i in range(5):

Counter(lock, "thread-" + str(i)).start()

time.sleep(2) #确保线程都执行完毕

print count

在代码中，我们创建了一个Counter类，它继承了threading.Thread。初始化函数接收两个参数，一个是琐对象，另一个是线程的名称。在Counter中，重写了从父类继承的run方法，run方法将一个全局变量逐一的增加10000。在接下来的代码中，创建了五个Counter对象，分别调用其start方法。最后打印结果。这里要说明一下run方法 和start方法: 它们都是从Thread继承而来的，run()方法将在线程开启后执行，可以把相关的逻辑写到run方法中（通常把run方法称为活动[Activity]。）；start()方法用于启动线程。

#### 创建一个threading.Thread对象

在这段代码中，我们定义了方法doAdd，它将全局变量count 逐一的增加10000。然后创建了5个Thread对象，把函数对象doAdd 作为参数传给它的初始化函数，再调用Thread对象的start方法，线程启动后将执行doAdd函数。

import threading, time, random

count = 0

lock = threading.Lock()

def doAdd():

'''@summary: 将全局变量count 逐一的增加10000。

'''

global count, lock

lock.acquire()

for i in xrange(10000): #加锁后才能进行，否则会乱掉

count = count + 1

lock.release()

for i in range(5):

threading.Thread(target = doAdd, args = (), name = 'thread-' + str(i)).start()

time.sleep(2) #确保线程都执行完毕

print count

#### 其他类

threading.Timer

threading.Timer是threading.Thread的子类，可以在指定时间间隔后执行某个操作。

def hello():

print "hello, world"

t = Timer(3, hello)

t.start() # 3秒钟之后执行hello函数。

Thread.join([timeout])

调用Thread.join将会使主调线程堵塞，直到被调用线程运行结束或超时。参数timeout是一个数值类型，表示超时时间，如果未提供该参数，那么主调线程将一直堵塞到被调线程结束。

import threading, time

def doWaiting():

print 'start waiting:', time.strftime('%H:%M:%S')

time.sleep(3)

print 'stop waiting', time.strftime('%H:%M:%S')

thread1 = threading.Thread(target = doWaiting)

thread1.start()

time.sleep(1) #确保线程thread1已经启动

print 'start join'

thread1.join() #将一直堵塞，直到thread1运行结束。

print 'end join'

threading.activeCount()的使用，此方法返回当前进程中线程的个数。返回的个数中包含主线程。

#!/usr/bin/python

#current's number of threads

import threading

import time

def worker():

print "test"

time.sleep(1)

for i in xrange(5):

t = threading.Thread(target=worker)

t.start()

print "current has %d threads" % (threading.activeCount() - 1)

threading.enumerate()的使用。此方法返回当前运行中的Thread对象列表。

#!/usr/bin/python

#test the variable threading.enumerate()

import threading

import time

def worker():

print "test"

time.sleep(2)

threads = []

for i in xrange(5):

t = threading.Thread(target=worker)

threads.append(t)

t.start()

for item in threading.enumerate():

print item

print

for item in threads:

print item

threading.setDaemon()的使用。设置后台进程。

可以看出worker（）方法中的打印操作并没有显示出来，说明已经成为后台进程。

#!/usr/bin/python

#create a daemon

import threading

import time

def worker():

time.sleep(3)

print "worker"

t=threading.Thread(target=worker)

t.setDaemon(True)

t.start()

print "haha"

threading.Event机制

threading.Event机制类似于一个线程向其它多个线程发号施令的模式，其它线程都会持有一个threading.Event的对象，这些线程都会等待这个事件的“发生”，如果此事件一直不发生，那么这些线程将会阻塞，直至事件的“发生”。

对此，我们可以考虑一种应用场景（仅仅作为说明），例如，我们有多个线程从Redis队列中读取数据来处理，这些线程都要尝试去连接Redis的服务，一般情况下，如果Redis连接不成功，在各个线程的代码中，都会去尝试重新连接。如果我们想要在启动时确保Redis服务正常，才让那些工作线程去连接Redis服务器，那么我们就可以采用threading.Event机制来协调各个工作线程的连接操作：主线程中会去尝试连接Redis服务，如果正常的话，触发事件，各工作线程会尝试连接Redis服务。

threading.Event的wait方法还接受一个超时参数，默认情况下如果事件一致没有发生，wait方法会一直阻塞下去，而加入这个超时参数之后，如果阻塞时间超过这个参数设定的值之后，wait方法会返回。对应于上面的应用场景，如果Redis服务器一致没有启动，我们希望子线程能够打印一些日志来不断地提醒我们当前没有一个可以连接的Redis服务，我们就可以通过设置这个超时参数来达成这样的目的：

与前面的无限阻塞版本唯一的不同就是，我们在工作线程中加入了一个while循环，直到redis\_ready事件触发之后才会结束循环，wait方法调用会在1秒的超时后返回，这样，我们就可以看到各个工作线程在系统启动的时候等待redis\_ready的同时，会记录一些状态信息。

import threading

import time

import logging

logging.basicConfig(level=logging.DEBUG, format='(%(threadName)-10s) %(message)s',)

def worker(event):

while not event.is\_set():

logging.debug('Waiting for redis ready...')

event.wait(1)

logging.debug('redis ready, and connect to redis server and do some work [%s]', time.ctime())

time.sleep(1)

readis\_ready = threading.Event()

t1 = threading.Thread(target=worker, args=(readis\_ready,), name='t1')

t1.start()

t2 = threading.Thread(target=worker, args=(readis\_ready,), name='t2')

t2.start()

logging.debug('first of all, check redis server, make sure it is OK, and then trigger the redis ready event')

time.sleep(3) # simulate the check progress

readis\_ready.set()

Python通过两个标准库thread和threading提供对线程的支持。thread提供了低级别的、原始的线程以及一个简单的锁。

#### 死锁及解决方案

死锁概念

所谓死锁： 是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。 由于资源占用是互斥的，当某个进程提出申请资源后，使得有关进程在无外力协助下，永远分配不到必需的资源而无法继续运行，这就产生了一种特殊现象死锁。

在Python中为了支持在同一线程中多次请求同一资源，python提供了“可重入锁”：threading.RLock。这个RLock内部维护着一个Lock和一个counter变量，counter记录了acquire的次数，从而使得资源可以被多次require。直到一个线程所有的acquire都被release，其他的线程才能获得资源。上面的例子如果使用RLock代替Lock，则不会发生死锁：

代码只需将上述的：

mutex = threading.Lock()

替换成：

mutex = threading.RLock()

#### threading包装普通函数

class sqlThread(threading.Thread):

def \_\_init\_\_(self,orderid,deleteSql,insertSql,para):

super(sqlThread, self).\_\_init\_\_()

self.orderid=orderid

self.deleteSql=deleteSql

self.insertSql=insertSql

self.para=para

def run(self):

#普通函数

h=sqlhandle(orderid=self.orderid,deleteSql=self.deleteSql,insertSql=self.insertSql,para=self.para)

h()

for i in range(5):

orderid=orderStartnum+i

insertparaOrder=(sortNumVar,tagVar)

t=sqlThread(orderid=orderid,deleteSql=deleteOrderSql,insertSql=insertOrderSql,para=insertparaOrder)

t.start()

### multiprocessing模块

multiprocessing已过时，只适用于python2.6

multiprocessing提供了threading包中没有的IPC(比如Pipe和Queue)，效率上更高。应优先考虑Pipe和Queue，避免使用Lock/Event/Semaphore/Condition等同步方式 (因为它们占据的不是用户进程的资源)。

multiprocessing包是Python中的多进程管理包。与threading.Thread类似，它可以利用multiprocessing.Process对象来创建一个进程。该进程可以运行在Python程序内部编写的函数。该Process对象与Thread对象的用法相同，也有start(), run(), join()的方法。此外multiprocessing包中也有Lock/Event/Semaphore/Condition类 (这些对象可以像多线程那样，通过参数传递给各个进程)，用以同步进程，其用法与threading包中的同名类一致。所以，multiprocessing的很大一部份与threading使用同一套API，只不过换到了多进程的情境。

Pipe可以是单向(half-duplex)，也可以是双向(duplex)。我们通过mutiprocessing.Pipe(duplex=False)创建单向管道 (默认为双向)。一个进程从PIPE一端输入对象，然后被PIPE另一端的进程接收，单向管道只允许管道一端的进程输入，而双向管道则允许从两端输入。

Queue与Pipe相类似，都是先进先出的结构。但Queue允许多个进程放入，多个进程从队列取出对象。Queue使用mutiprocessing.Queue(maxsize)创建，maxsize表示队列中可以存放对象的最大数量。

### gevent 异步 并发

<http://xlambda.com/gevent-tutorial/>

当一个greenlet遇到IO操作时，比如访问网络，就自动切换到其他的greenlet，等到IO操作完成，再在适当的时候切换回来继续执行。由于IO操作非常耗时，经常使程序处于等待状态，有了gevent为我们自动切换协程，就保证总有greenlet在运行，而不是等待IO。

由于切换是在IO操作时自动完成，所以gevent需要修改Python自带的一些标准库，这一过程在启动时通过monkey patch完成：

from gevent import monkey; monkey.patch\_socket()

import gevent

def f(n):

for i in range(n):

print gevent.getcurrent(), i

g1 = gevent.spawn(f, 5)

g2 = gevent.spawn(f, 5)

g3 = gevent.spawn(f, 5)

g1.join()

g2.join()

g3.join()

### thread 多线程控制

这个模块为多线程提供了一个底层 、原始的操作 (也可以成为light-weight processes 或者 tasks) — 多个控制线程共享全局数据空间。为了多线程同步，提供了简单的锁(也称呼为 mutexes 后者 binary semaphores) 。当然， threading 模块在thread模块之上提供了一个更容易使用、更高层次的线程API。

thread模块是可选的。它支持在Windows，linux，SGI IRIX，Solaris 2.x，以及有POSIX线程（也称为：pthread）的系统上运行。对于某些缺乏 thread 模块的操作系统，dummy\_thread 模块可以作为一种替代。dummy\_thread模块提供了和thread模块同样的接口，可以在使用中直接替换。

#### 线程 模板

Python中使用线程有两种方式：函数或者用类来包装线程对象。

函数式：调用thread模块中的start\_new\_thread()函数来产生新线程。语法如下:

thread.start\_new\_thread ( function, args[, kwargs] )

thread 模块提供的其他方法：

threading.currentThread(): 返回当前的线程变量。

threading.enumerate(): 返回一个包含正在运行的线程的list。正在运行指线程启动后、结束前，不包括启动前和终止后的线程。

threading.activeCount(): 返回正在运行的线程数量，与len(threading.enumerate())有相同的结果。

除了使用方法外，线程模块同样提供了Thread类来处理线程，Thread类提供了以下方法:

run(): 用以表示线程活动的方法。

start():启动线程活动。

join([time]): 等待至线程中止。这阻塞调用线程直至线程的join() 方法被调用中止-正常退出或者抛出未处理的异常-或者是可选的超时发生。

isAlive(): 返回线程是否活动的。

getName(): 返回线程名。

setName(): 设置线程名。

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

import thread

import time

def print\_time(threadName,delay):

count=0

while count<5:

time.sleep(delay)

count+=1

print "%s: %s" % ( threadName, time.ctime(time.time()) )

try:

thread.start\_new\_thread(print\_time,("Thread-1", 2, ) )

thread.start\_new\_thread(print\_time,('Thread-2',4,))

except:

print'Error: unable to start thread'

while 1:

pass

线程的结束一般依靠线程函数的自然结束；也可以在线程函数中调用thread.exit()，他抛出SystemExit exception，达到退出线程的目的。

## 有关高级数据结构

单向链表（单链表）是链表的一种，其特点是链表的链接方向是单向的，对链表的访问要通过顺序读取从头部开始。

### collections 提供额外的数据类型

Python拥有一些内置的数据类型，比如str, int, list, tuple, dict等， collections模块在这些内置数据类型的基础上，提供了几个额外的数据类型：

1.namedtuple(): 生成可以使用名字来访问元素内容的tuple子类

2.deque: 双端队列，可以快速的从另外一侧追加和推出对象

3.Counter: 计数器，主要用来计数

4.OrderedDict: 有序字典

5.defaultdict: 带有默认值的字典

#### namedtuple 使用名字访问元素内容的tuple子类

我们知道tuple可以表示不变集合，例如，一个点的二维坐标就可以表示成：

>>> p = (1, 2)

但是，看到(1, 2)，很难看出这个tuple是用来表示一个坐标的。

定义一个class又小题大做了，这时，namedtuple就派上了用场：

>>> from collections import namedtuple

>>> Point = namedtuple('Point', ['x', 'y'])

>>> p = Point(1, 2)

>>> p.x

1

>>> p.y

2

namedtuple是一个函数，它用来创建一个自定义的tuple对象，并且规定了tuple元素的个数，并可以用属性而不是索引来引用tuple的某个元素。

这样一来，我们用namedtuple可以很方便地定义一种数据类型，它具备tuple的不变性，又可以根据属性来引用，使用十分方便。

可以验证创建的Point对象是tuple的一种子类：

>>> isinstance(p, Point)

True

>>> isinstance(p, tuple)

True

类似的，如果要用坐标和半径表示一个圆，也可以用namedtuple定义：

# namedtuple('名称', [属性list]):

Circle = namedtuple('Circle', ['x', 'y', 'r'])

#### deque 双端队列list

使用list存储数据时，按索引访问元素很快，但是插入和删除元素就很慢了，因为list是线性存储，数据量大的时候，插入和删除效率很低。

deque是为了高效实现插入和删除操作的双向列表，适合用于队列和栈：

>>> from collections import deque

>>> q = deque(['a', 'b', 'c'])

>>> q.append('x')

>>> q.appendleft('y')

>>> q

deque(['y', 'a', 'b', 'c', 'x'])

deque除了实现list的append()和pop()外，还支持appendleft()和popleft()，这样就可以非常高效地往头部添加或删除元素。

#### Counter 计数器，主要用来计数

Counter是一个简单的计数器，例如，统计字符出现的个数：

>>> from collections import Counter

>>> c = Counter()

>>> for ch in 'programming':

... c[ch] = c[ch] + 1

...

>>> c

Counter({'g': 2, 'm': 2, 'r': 2, 'a': 1, 'i': 1, 'o': 1, 'n': 1, 'p': 1})

Counter实际上也是dict的一个子类，上面的结果可以看出，字符'g'、'm'、'r'各出现了两次，其他字符各出现了一次。

#### OrderedDict 有序字典

使用dict时，Key是无序的。在对dict做迭代时，我们无法确定Key的顺序。

如果要保持Key的顺序，可以用OrderedDict：

>>> from collections import OrderedDict

>>> d = dict([('a', 1), ('b', 2), ('c', 3)])

>>> d # dict的Key是无序的

{'a': 1, 'c': 3, 'b': 2}

>>> od = OrderedDict([('a', 1), ('b', 2), ('c', 3)])

>>> od # OrderedDict的Key是有序的

OrderedDict([('a', 1), ('b', 2), ('c', 3)])

注意，OrderedDict的Key会按照插入的顺序排列，不是Key本身排序：

>>> od = OrderedDict()

>>> od['z'] = 1

>>> od['y'] = 2

>>> od['x'] = 3

>>> od.keys() # 按照插入的Key的顺序返回

['z', 'y', 'x']

for i in od:print i #i为key

#### defaultdict 带有默认值的字典

使用dict时，如果引用的Key不存在，就会抛出KeyError。如果希望key不存在时，返回一个默认值，就可以用defaultdict：

>>> from collections import defaultdict

>>> dd = defaultdict(lambda: 'N/A')

>>> dd['key1'] = 'abc'

>>> dd['key1'] # key1存在

'abc'

>>> dd['key2'] # key2不存在，返回默认值

'N/A'

注意默认值是调用函数返回的，而函数在创建defaultdict对象时传入。

除了在Key不存在时返回默认值，defaultdict的其他行为跟dict是完全一样的。

### heapq 堆队列 有序的list 排序算法用

heapq模块实现了python中的堆排序，并提供了有关方法。

让用Python实现排序算法有了简单快捷的方式。

import heapq

heap = []

for value in [20, 10, 30, 50, 40]:

heapq.heappush(heap, value)

while heap:

print heapq.heappop(heap)

#### 实现堆排序

#! /usr/bin/evn python

#coding:utf-8

from heapq import \*

def heapsort(iterable):

h = []

for value in iterable:

heappush(h,value)

return [heappop(h) for i in range(len(h))]

if \_\_name\_\_=="\_\_main\_\_":

print heapsort([1,3,5,9,2])

heapq模块提供了如下几个函数：

heapq.heappush(heap, item) 把item添加到heap中（heap是一个列表）

heapq.heappop(heap) 把堆顶元素弹出，返回的就是堆顶

heapq.heappushpop(heap, item) 先把item加入到堆中，然后再pop，比heappush()再heappop()要快得多

heapq.heapreplace(heap, item) 先pop，然后再把item加入到堆中，比heappop()再heappush()要快得多

heapq.heapify(x) 将列表x进行堆调整，默认的是小顶堆

heapq.merge(\*iterables) 将多个列表合并，并进行堆调整，返回的是合并后的列表的迭代器

heapq.nlargest(n, iterable, key=None) 返回最大的n个元素（Top-K问题）

heapq.nsmallest(n, iterable, key=None) 返回最小的n个元素（Top-K问题）

#### heappush()

heapq.heappush(heap, item):将item压入到堆数组heap中。如果不进行此步操作，后面的heappop()失效

#### heappop()

heapq.heappop(heap):从堆数组heap中取出最小的值，并返回。

>>> h=[] #定义一个list

>>> from heapq import \* #引入heapq模块

>>> h

[]

>>> heappush(h,5) #向堆中依次增加数值

>>> heappush(h,2)

>>> heappush(h,3)

>>> heappush(h,9)

>>> h #h的值

[2, 5, 3, 9]

>>> heappop(h) #从h中删除最小的，并返回该值

2

>>> h

[3, 5, 9]

>>> h.append(1) #注意，如果不是压入堆中，而是通过append追加一个数值

>>> h #堆的函数并不能操作这个增加的数值，或者说它堆对来讲是不存在的

[3, 5, 9, 1]

>>> heappop(h) #从h中能够找到的最小值是3,而不是1

3

>>> heappush(h,2) #这时，不仅将2压入到堆内，而且1也进入了堆。

>>> h

[1, 2, 9, 5]

>>> heappop(h) #操作对象已经包含了1

1

#### heapq.heappushpop(heap, item)

是上述heappush和heappop的合体，同时完成两者的功能.注意：相当于先操作了heappush(heap,item),然后操作heappop(heap)

>>> h

[1, 2, 9, 5]

>>> heappop(h)

1

>>> heappushpop(h,4) #增加4同时删除最小值2并返回该最小值，与下列操作等同：

2 #heappush(h,4),heappop(h)

>>> h

[4, 5, 9]

#### heapq.heapify(x)

x必须是list，此函数将list变成堆，实时操作。从而能够在任何情况下使用堆的函数。

>>> a=[3,6,1]

>>> heapify(a) #将a变成堆之后，可以对其操作

>>> heappop(a)

1

>>> b=[4,2,5] #b不是堆，如果对其进行操作，显示结果如下

>>> heappop(b) #按照顺序，删除第一个数值并返回,不会从中挑选出最小的

4

>>> heapify(b) #变成堆之后，再操作

>>> heappop(b)

2

#### heapq.heapreplace(heap, item)

是heappop(heap)和heappush(heap,item)的联合操作。注意，与heappushpop(heap,item)的区别在于，顺序不同，这里是先进行删除，后压入堆

>>> a=[]

>>> heapreplace(a,3) #如果list空，则报错

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: index out of range

>>> heappush(a,3)

>>> a

[3]

>>> heapreplace(a,2) #先执行删除（heappop(a)->3),再执行加入（heappush(a,2))

3

>>> a

[2]

>>> heappush(a,5)

>>> heappush(a,9)

>>> heappush(a,4)

>>> a

[2, 4, 9, 5]

>>> heapreplace(a,6) #先从堆a中找出最小值并返回，然后加入6

2

>>> a

[4, 5, 9, 6]

>>> heapreplace(a,1) #1是后来加入的，在1加入之前，a中的最小值是4

4

>>> a

[1, 5, 9, 6]

#### heapq.merge(\*iterables)

举例：

>>> a=[2,4,6]

>>> b=[1,3,5]

>>> c=merge(a,b)

>>> list(c)

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

在归并排序中详细演示了本函数的使用方法。

#### heapq.nlargest(n, iterable[, key]),heapq.nsmallest(n, iterable[, key])

获取列表中最大、最小的几个值。

>>> a

[2, 4, 6]

>>> nlargest(2,a)

[6, 4]

### bisect 能够提供保持list元素序列的支持

bisect模块能够提供保持list元素序列的支持。它使用了二分法完成大部分的工作。它在向一个list插入元素的同时维持list是有序的。在某些情况下，这比重复的对一个list进行排序更为高效，并且对于一个较大的list来说，对每步操作维持其有序也比对其排序要高效。

import bisect

a = [(0, 100), (150, 220), (500, 1000)]

bisect.insort\_right(a, (250,400))

print a # [(0, 100), (150, 220), (250, 400), (500, 1000)]

#我们可以使用bisect()函数来寻找插入点：

# bisect(sequence, item) => index 返回元素应该的插入点，但序列并不被修改。

print bisect.bisect(a, (550, 1200)) # 5

### array 类似list类型

array模块定义了一个很像list的新对象类型，不同之处在于它限定了这个类型只能装一种类型的元素。array元素的类型是在创建并使用的时候确定的。

import array

a = array.array(&quot;i&quot;, [1,2,3,4,5])

b = array.array(a.typecode, (2\*x for x in a))

import array

a = array.array(&quot;i&quot;, [1,2,3,4,5])

for i, x in enumerate(a):

a[i] = 2\*x

### itertools 创建迭代器

Python的内建模块itertools提供了非常有用的用于操作迭代对象的函数。

#### 无限迭代器 count() cycle() repeat()

首先，我们看看itertools提供的几个“无限”迭代器：

##### count()

**count()会创建一个无限的迭代器，所以代码会打印出自然数序列，根本停不下来，只能按Ctrl+C退出。**

>>> import itertools

>>> natuals = itertools.count(1)

>>> for n in natuals:

... print n

##### cycle()

**cycle()会把传入的一个序列无限重复下去：**

>>> import itertools

>>> cs = itertools.cycle('ABC') # 注意字符串也是序列的一种

>>> for c in cs:

... print c

...

'A'

'B'

'C'

'A'

'B'

'C'

...

同样停不下来。

##### repeat()

**repeat()负责把一个元素无限重复下去，不过如果提供第二个参数就可以限定重复次数：**

>>> ns = itertools.repeat('A', 10)

>>> for n in ns:

... print n

...

打印10次'A'

无限序列只有在for迭代时才会无限地迭代下去，如果只是创建了一个迭代对象，它不会事先把无限个元素生成出来，事实上也不可能在内存中创建无限多个元素。

##### takewhile()

无限序列虽然可以无限迭代下去，但是通常我们会通过**takewhile()**等函数根据条件判断来截取出一个有限的序列：

>>> natuals = itertools.count(1)

>>> ns = itertools.takewhile(lambda x: x <= 10, natuals)

>>> for n in ns:

... print n

...

打印出1到10

#### 迭代器操作函数

itertools提供的几个迭代器操作函数更加有用：

##### itertools.chain()

**itertools.**chain()可以把一组迭代对象串联起来，形成一个更大的迭代器：

for c in **itertools.**chain('ABC', 'XYZ'):

print c

# 迭代效果：'A' 'B' 'C' 'X' 'Y' 'Z'

itertools.chain.from\_iterable(iterables)

import itertools

test=itertools.chain.from\_iterable('ABCDEF')

test.next()

一个备用链构造函数，其中的iterables是一个迭代变量，生成迭代序列，此操作的结果与以下生成器代码片段生成的结果相同：

def f(iterables):

for x in iterables:

for y in x:

yield y

test = f('ABCDEF')

print test.next()

##### groupby()

groupby()把迭代器中相邻的重复元素挑出来放在一起：

>>> for key, group in itertools.groupby('AAABBBCCAAA'):

... print key, list(group) # 为什么这里要用list()函数呢？

...

A ['A', 'A', 'A']

B ['B', 'B', 'B']

C ['C', 'C']

A ['A', 'A', 'A']

实际上挑选规则是通过函数完成的，只要作用于函数的两个元素返回的值相等，这两个元素就被认为是在一组的，而函数返回值作为组的key。如果我们要忽略大小写分组，就可以让元素'A'和'a'都返回相同的key：

>>> for key, group in itertools.groupby('AaaBBbcCAAa', lambda c: c.upper()):

... print key, list(group)

...

A ['A', 'a', 'a']

B ['B', 'B', 'b']

C ['c', 'C']

A ['A', 'A', 'a']

##### imap()

imap()和map()的区别在于，imap()可以作用于无穷序列，**并且，如果两个序列的长度不一致，以短的那个为准。**

for x in itertools.imap(lambda x, y: x \* y, [10, 20, 30], itertools.count(1)):

print x

# 10 40 90

注意imap()返回一个迭代对象，而map()返回list。

>>> r = map(lambda x: x\*x, [1, 2, 3])

>>> r # r已经计算出来了 [1, 4, 9]

当你调用imap()时，并没有进行任何计算：

>>> r = itertools.imap(lambda x: x\*x, [1, 2, 3])

>>> r <itertools.imap object at 0x103d3ff90>

# r只是一个迭代对象

必须用for循环对r进行迭代，才会在每次循环过程中计算出下一个元素：

for x in r:print x

这说明imap()实现了“惰性计算”，也就是在需要获得结果的时候才计算。类似imap()这样能够实现惰性计算的函数就可以处理无限序列：

##### ifilter()

不用多说了，ifilter()就是filter()的惰性实现。

##### combinations()

combinations(iterable, r)

创建一个迭代器，返回iterable中所有长度为r的**子序列**，返回的子序列中的项按输入iterable中的顺序排序:

import itertools

test=itertools.combinations([1,2,3,4],2)

for el in test:print el

(1, 2)(1, 3)(2, 3)

##### dropwhile()

dropwhile(predicate, iterable)

创建一个迭代器，只要函数predicate(item)为True，就丢弃iterable中的项，如果predicate返回False，就会生成iterable中的项和所有后续项。

##### izip()

返回一个合并了多个迭代器为一个元组的迭代器. 它类似于内置函数zip(), 只是它返回的是一个迭代器而不是一个列表

import itertools

for i in itertools.izip([1, 2, 3], ['a', 'b', 'c']):print i

(1, 'a')(2, 'b')(3, 'c')

## 有关数据处理

### StringIO和BytesIO 字符串像文件一样读取

很多时候，数据读写不一定是文件，也可以在内存中读写。

#### StringIO

顾名思义就是在内存中读写str。

因为文件对象和StringIO大部分的方法都是一样的，比如read, readline, readlines, write, writelines都是有的，这样，**StringIO就可以非常方便的作为"内存文件对象"。**

StringIO只能写入unicode字符串，不能写入str字符串。

可以在程序开头from \_\_future\_\_ import unicode\_literals，或者字符串前加u

要把str写入StringIO，我们需要先创建一个StringIO，然后，像文件一样写入即可：

from io import StringIO

f = StringIO()

f.write(u'hello ')

f.write(u'world!')

print f.getvalue() #获取全部数据，输出为str类型

# 每次使用read()\readline()\readlines()读取前，必须seek(0),否则指针会在最后

# 定位到开头

f.seek(0)

print f.readline()

f.seek(0)

print f.read()

print s.len

注意StringIO不能使用with语句

f = StringIO()

with open(f , "w+") as gisy:gisy.writelines('gisy')

报错TypeError: coercing to Unicode: need string or buffer, \_io.StringIO found

如果使用read方法获取其中的数据，必须通过seek先设置"文件指针"的位置。

s.seek(3)

# 从第三个位置读取两个字符，将输出DE

print s.read(2)

必须使用u为前缀，代表还是Unicode，而不是str类型

s.read([n])

参数n限定读取长度，int类型；缺省状态为从当前读写位置读取对象s中存储的所有数据。读取结束后，读写位置被移动。

s.readline([length])

参数length限定读取的结束位置，int类型，缺省状态为None：从当前读写位置读取至下一个以“\n”为结束符的当前行。读写位置被移动。

s.readlines([sizehint])

参数sizehint为int类型，缺省状态为读取所有行并作为列表返回，除此之外从当前读写位置读取至下一个以“\n”为结束符的当前行。读写位置被移动。

s.write(s)

从读写位置将参数s写入给对象s。参数s为str或unicode类型。读写位置被移动。

s.writelines(list)

从读写位置将list写入给对象s。参数list为一个列表，列表的成员为str或unicode类型。读写位置被移动。

s.getvalue()

此函数没有参数，返回对象s中的所有数据。

s.truncate([size])

从读写位置起切断数据，参数size限定裁剪长度，缺省值为None。

s.tell()

返回当前读写位置。

s.seek(pos[,mode])

移动当前读写位置至pos处，可选参数mode为0时将读写位置移动至pos处，为1时将读写位置从当前位置起向后移动pos个长度，为2时将读写位置置于末尾处再向后移动pos个长度；默认为0。

s.close()

释放缓冲区，执行此函数后，数据将被释放，也不可再进行操作。

s.isatty()

此函数总是返回0。不论StringIO对象是否已被close()。

s.flush()

刷新内部缓冲区。

#### 注意点

#### cStringIO

Python标准模块中还提供了一个cStringIO模块，它的行为与StringIO基本一致，但运行效率方面比StringIO更好。因此，import的时候会写成下面的方式，但使用 cStringIO模块时，有几个注意点：

1. cStringIO.StringIO不能作为基类被继承；

2. 创建cStringIO.StringIO对象时，如果初始化函数提供了初始化数据，新生成的对象是只读的。

try:

from cStringIO import StringIO

except ImportError:

from StringIO import StringIO

s = StringIO('A')

s.write('abc')

# 会提示：AttributeError: 'cStringIO.StringI' object has no attribute 'write'

print s.getvalue()

#### 文件转换为StringIO

try:

from cStringIO import StringIO

except:

from StringIO import StringIO

import pprint

with open('40600251298.txt', "r+") as gisy:

a=gisy.readlines()

f = StringIO()

f.writelines(a)

#print f.getvalue()

f.seek(0)

pprint.pprint(f.readlines())

#### BytesIO

StringIO操作的只能是str，如果要操作二进制数据，就需要使用BytesIO。

BytesIO实现了在内存中读写bytes，我们创建一个BytesIO，然后写入一些bytes：

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

from io import BytesIO

f=BytesIO()

f.write(u'中文'.encode('utf-8'))

print (f.getvalue())

### HTMLParser模块 解析html

如果我们要编写一个搜索引擎，第一步是用爬虫把目标网站的页面抓下来，第二步就是解析该HTML页面，看看里面的内容到底是新闻、图片还是视频。

假设第一步已经完成了，第二步应该如何解析HTML呢？

HTML本质上是XML的子集，但是HTML的语法没有XML那么严格，所以不能用标准的DOM或SAX来解析HTML。

好在Python提供了HTMLParser来非常方便地解析HTML，只需简单几行代码：

from HTMLParser import HTMLParser

from htmlentitydefs import name2codepoint

class MyHTMLParser(HTMLParser):

def handle\_starttag(self, tag, attrs):

print('<%s>' % tag)

def handle\_endtag(self, tag):

print('</%s>' % tag)

def handle\_startendtag(self, tag, attrs):

print('<%s/>' % tag)

def handle\_data(self, data):

print('data')

def handle\_comment(self, data):

print('<!-- -->')

def handle\_entityref(self, name):

print('&%s;' % name)

def handle\_charref(self, name):

print('&#%s;' % name)

parser = MyHTMLParser()

parser.feed('<html><head></head><body><p>Some <a href=\"#\">html</a> tutorial...<br>END</p></body></html>')

#feed()方法可以多次调用，也就是不一定一次把整个HTML字符串都塞进去，可以一部分一部分塞进去。

特殊字符有两种，一种是英文表示的&nbsp;，一种是数字表示的&#1234;，这两种字符都可以通过Parser解析出来。

### pprint模块 输出的格式化

pprint.pprint(obj)

用于打印 Python 数据结构. 当你在命令行下打印特定数据结构时你会发现它很有用(输出格式比较整齐, 便于阅读).

import sys

import pprint

pprint.pprint(sys.path)

import pprint

data = (

"this is a string", [1, 2, 3, 4], ("more tuples",

1.0, 2.3, 4.5), "this is yet another string"

)

pprint.pprint(data)

####

('this is a string',

[1, 2, 3, 4],

('more tuples', 1.0, 2.3, 4.5),

'this is yet another string')

import pprint

matrix = [ [1,2,3], [4,5,6], [7,8,9] ]

a = pprint.PrettyPrinter(width=20)

1. pprint(matrix)

import sys

import pprint

string=pprint.pformat(sys.path)

print string

import pprint

stuff = ['spam','eggs','lumberjack','knight','ni']

stuff.insert(0,stuff)

pprint.pprint(stuff)

import pprint

tup = ('spam',('eggs',('lumberjack',)))

stuff = ['a'\*5, tup, ['b'\*5, 'c'\*5],['c'\*5, 'd'\*5]]

pprint.pprint(stuff,depth=2) #显示深度

# ['aaaaa', ('spam', (...)), ['bbbbb', 'ccccc'], ['ccccc', 'ddddd']]

### random模块 生成随机数

random.random

random.random()用于生成一个0到1的随机符点数: 0 <= n < 1.0

random.uniform

random.uniform(a, b)，用于生成一个指定范围内的随机符点数，两个参数其中一个是上限，一个是下限。如果a > b，则生成的随机数n: a <= n <= b。如果 a <b， 则 b <= n <= a

random.uniform(1, 10)

# 5.4221167969800881

random.randint

random.randint(a, b)，用于生成一个指定范围内的整数。其中参数a是下限，参数b是上限，生成的随机数n: a <= n <= b

# print random.randint(20, 10) # 该语句是错误的。下限必须小于上限

random.randrange

random.randrange([start], stop[, step])，从指定范围内，按指定基数递增的集合中 获取一个随机数。如：random.randrange(10, 100, 2)，结果相当于从[10, 12, 14, 16, ... 96, 98]序列中获取一个随机数。random.randrange(10, 100, 2)在结果上与 random.choice(range(10, 100, 2) 等效

random.choice

random.choice从序列中获取一个随机元素。其函数原型为：random.choice(sequence)。参数sequence表示一个有序类型。这里要说明 一下：sequence在python不是一种特定的类型，而是泛指一系列的类型。list, tuple, 字符串都属于sequence。

print random.choice("学习Python")

print random.choice(["JGood", "is", "a", "handsome", "boy"])

print random.choice(("Tuple", "List", "Dict"))

random.shuffle

random.shuffle(x[, random])，用于将一个列表中的元素打乱

p = ["Python", "is", "powerful", "simple", "and so on..."]

random.shuffle(p)

print p

# ['powerful', 'simple', 'is', 'Python', 'and so on...']

random.sample

random.sample(sequence, k)，从指定序列中随机获取指定长度的片断。sample函数不会修改原有序列

list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

slice = random.sample(list, 5) # 从list中随机获取5个元素，作为一个片断返回

print slice

print list # 原有序列并没有改变

### math模块

math模块实现了许多对浮点数的数学运算函数. 这些函数一般是对平台 C 库中同名函数的简单封装, 所以一般情况下, 不同平台下计算的结果可能稍微地有所不同, 有时候甚至有很大出入

from math import \*

函数（方法） 示例 说明

acos(x) 求x的反余弦（结果是弧度） acos(2.0)等于0.0

asin(x) 求x的反正弦（结果是弧度） asin(0.0)等于0.0

atan(x) 求x的反正切（结果是弧度） atan(0.0)等于0.0

ceil(x) 为x取整，结果是不小于x的最小整数 ceil(9.2)等于10.0 ceil(-9.8)等于-9.0

cos(x) 求x的余弦（x是弧度） cos(0.0)等于1.0

exp(x) 求幂函数e` exp(1.0)等于2.71828 exp(2.0)等于7.38906

fabs(x) 求x的绝对值 fabs(5.1)等于5.1 fabs(-5.1)等于5.1

floor(x) 为x取整，结果是不大于x的最大整数 floor(9.2)等于9.0 floor(-9.8)等于-10.0

fmod(x,y) 求x/y的余数，结果是浮点数 fmod(9.8,4.0)等于1.8

hypot(x,y) 求直角三角的斜边长度，直边长度为x和y：Sqrt(x2-y2) hypot(3.0,4.0)等于5.0

log10(x) 求x的对数（以10为底） log10(10.0)等于1.0 log10(100.0)等于2.0

pow(x,y) 求x的y次方（xy） pow(2.7,7.0)等于128.0 pow(9.0,0.5)等于3.0

sin(x) 求x的正弦（x是弧度） sin(0.0)等于0.0

sqrt(x) 求x的平方根 sqrt(900.0)等于30.0 sqrt(9.0)等于3.0

tan(x) 求x的正切（x是弧度） tan(0.0)等于0.0

import math math.pi 3.1415926535897931

math.sin(math.pi / 2) 1.0

math.tan(math.pi / 4) 0.99999999999999989

### pickle模块 保存对象到文件 数据序列和反序列化

python的pickle模块实现了基本的数据序列和反序列化。

通过pickle 我们能保存列表，字典等任何对象。

通过pickle模块的序列化操作我们能够将程序中运行的对象信息保存到文件中去，永久存储；通过pickle模块的反序列化操作，我们能够从文件中创建上一次程序保存的对象。

在Python中最基础的实现序列化的两个模块是**cPickle和pickle**模块, 功能基本一样, 前者是C写的更快, 后者是python写的会慢点. 一般优先使用cPickle.

dumps()函数执行和dump() 函数相同的序列化，但是与dump不同的dumps并不将转换后的字符串写入文件，而是将所得到的转换后的数据以字符串的形式返回。

loads()函数执行和load()函数一样的反序列化。 loads接受一个字符串参数，将字符串解码成为python的数据类型，函数loads和dumps进行的是互逆的操作。

pickle的dump和load相当于java的序列化和反序列化操作

设你有一个字典，你想存储它，稍后再取出来。你可以把它的内容写入一个文件，小心翼翼地确保使用了正确地格式，要把它读取出来，你可以使用 exec() 或处理文件输入。但是这种方法并不可靠：如果你使用纯文本来存储重要数据，数据很容易以多种方式被破坏或者修改，导致你的程序崩溃，更糟糕的情况下，还可能在你的计算机上运行恶意代码。因此，我们要pickle它

#### dump 写入文件(序列化)

#使用pickle模块将数据对象保存到文件

import pickle

data1 = {'a': [1, 2.0, 3, 4+6j],

'b': ('string', u'Unicode string'),

'c': None}

selfref\_list = [1, 2, 3]

**output = open('data.pkl', 'wb')**

# Pickle dictionary using protocol 0.

**pickle.dump(data1, output)**

pickle.dump(selfref\_list, output, 2)

output.close()

主要有两个函数:

dump(data,file\_handle[, protocol]) : 将数据序列化到文件.

dumps(data[, protocol]) : 将数据序列化成字符串.

file\_handle这个对象可以是一个**以写模式打开的文件对象或者一个 StringIO 对象**，或者其他自定义的满足条件的对象。

protocol是数据流处理策略:

0: ascii串保存, 默认形式, 方便人读取

1: 旧式兼容性较强2进制形式

2: 支持新式类的2进制模式,Python2.3开始引入.

a=[1,2,3]

# dumps() to string

cPickle.dumps(a)

'(lp1\nI1\naI2\naI3\na.'

# dump() to file

f=open('data.txt','wb')

cPickle.dump(a,f)

f.close

如果protocol不为0，则打开方式必须是二进制模式，就是'wb'或'ab',否则报错

当然最好都是 设为二进制模式

TypeError: ord() expected a character, but string of length 0 found

#### load 从文件读取出来(反序列化)

主要有两个函数:

load(file) : 将序列化数据从文件读入返回数据.

loads(string) : 将字符串的序列化数据读入并返回数据.

# loads() from string to data

d=cPickle.loads('(lp1\nI1\naI2\naI3\na.')

# load() from file to data

df=cPickle.load('data.txt')

#使用pickle模块从文件中重构python对象 记住之前存入时存入几次，读取就要读取load几次 先存入的先读出

import pprint, pickle

pkl\_file = open('data.pkl', 'rb')

data1 = pickle.load(pkl\_file)

print data1

data2 = pickle.load(pkl\_file)

print data2

pkl\_file.close()

### struct模块 解决str和其他二进制数据类型的转换

有的时候需要用python处理二进制数据，比如，存取文件，socket操作时.这时候，可以使用python的struct模块来完成.可以用 struct来处理c语言中的结构体.

struct模块中最重要的三个函数是pack(), unpack(), calcsize()

pack(fmt, v1, v2, ...) 按照给定的格式(fmt)，把数据封装成字符串(实际上是类似于c结构体的字节流)

unpack(fmt, string) 按照给定的格式(fmt)解析字节流string，返回解析出来的tuple

calcsize(fmt) 计算给定的格式(fmt)占用多少字节的内存

准确地讲，Python没有专门处理字节的数据类型。但由于str既是字符串，又可以表示字节，所以，字节数组＝str。而在C语言中，我们可以很方便地用struct、union来处理字节，以及字节和int，float的转换。

在Python中，比方说要把一个32位无符号整数变成字节，也就是4个长度的str，你得配合位运算符这么写：

>>> n = 10240099

>>> b1 = chr((n & 0xff000000) >> 24)

>>> b2 = chr((n & 0xff0000) >> 16)

>>> b3 = chr((n & 0xff00) >> 8)

>>> b4 = chr(n & 0xff)

>>> s = b1 + b2 + b3 + b4

>>> s

'\x00\x9c@c'

非常麻烦。如果换成浮点数就无能为力了。

好在Python提供了一个struct模块来解决str和其他二进制数据类型的转换。

struct的pack函数把任意数据类型变成字符串：

>>> import struct

>>> struct.pack('>I', 10240099)

'\x00\x9c@c'

pack的第一个参数是处理指令，'>I'的意思是：

>表示字节顺序是big-endian，也就是网络序，I表示4字节无符号整数。

后面的参数个数要和处理指令一致。

unpack把str变成相应的数据类型：

>>> struct.unpack('>IH', '\xf0\xf0\xf0\xf0\x80\x80')

(4042322160, 32896)

根据>IH的说明，后面的str依次变为I：4字节无符号整数和H：2字节无符号整数。

所以，尽管Python不适合编写底层操作字节流的代码，但在对性能要求不高的地方，利用struct就方便多了。

struct模块定义的数据类型可以参考Python官方文档：

https://docs.python.org/2/library/struct.html#format-characters

Windows的位图文件（.bmp）是一种非常简单的文件格式，我们来用struct分析一下。

首先找一个bmp文件，没有的话用“画图”画一个。

读入前30个字节来分析：

>>> s = '\x42\x4d\x38\x8c\x0a\x00\x00\x00\x00\x00\x36\x00\x00\x00\x28\x00\x00\x00\x80\x02\x00\x00\x68\x01\x00\x00\x01\x00\x18\x00'

BMP格式采用小端方式存储数据，文件头的结构按顺序如下：

两个字节：'BM'表示Windows位图，'BA'表示OS/2位图；

一个4字节整数：表示位图大小；

一个4字节整数：保留位，始终为0；

一个4字节整数：实际图像的偏移量；

一个4字节整数：Header的字节数；

一个4字节整数：图像宽度；

一个4字节整数：图像高度；

一个2字节整数：始终为1；

一个2字节整数：颜色数。

所以，组合起来用unpack读取：

>>> struct.unpack('<ccIIIIIIHH', s)

('B', 'M', 691256, 0, 54, 40, 640, 360, 1, 24)

结果显示，'B'、'M'说明是Windows位图，位图大小为640x360，颜色数为24。

### inspect模块 对是否是模块，框架，函数等进行类型检查。

#### 基础

inspect模块主要提供了四种用处：

(1).对是否是模块，框架，函数等进行类型检查。

(2).获取源码

(3).获取类或函数的参数的信息

(4).解析堆栈

inspect.ismodule(object)： 是否为模块

inspect.isclass(object)：是否为类

inspect.ismethod(object)：是否为方法（bound method written in python）

inspect.isfunction(object)：是否为函数(python function, including lambda expression)

inspect.isgeneratorfunction(object)：是否为python生成器函数

inspect.isgenerator(object):是否为生成器

inspect.istraceback(object)： 是否为traceback

inspect.isframe(object)：是否为frame

inspect.iscode(object)：是否为code

inspect.isbuiltin(object)：是否为built-in函数或built-in方法

inspect.isroutine(object)：是否为用户自定义或者built-in函数或方法

inspect.isabstract(object)：是否为抽象基类

inspect.ismethoddescriptor(object)：是否为方法标识符

inspect.isdatadescriptor(object)：是否为数字标识符，数字标识符有\_\_get\_\_ 和\_\_set\_\_属性； 通常也有\_\_name\_\_和\_\_doc\_\_属性

inspect.isgetsetdescriptor(object)：是否为getset descriptor

inspect.ismemberdescriptor(object)：是否为member descriptor

#### 实例

当程序出现诡异的bug时，我们需要层层去跟踪，尤其是要理清顺序。

这时如果有个清晰的调用脉络调试就容易多了。

利用inspect模块的getframeinfo方法来实现这需求。

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

import inspect

def foo():

who=inspect.getframeinfo(inspect.currentframe().f\_back)[2]

print '{} call me'.format(who)

def a():

foo()

def b():

foo()

a()

b()

a()

结果输出：

a call me

b call me

a call me

inspect模块也提供了许多有用的函数，来获取活跃对象的信息。比方说，你可以查看一个对象的成员，只需运行：

import inspect

print(inspect.getmembers(str))

# Output: [('\_\_add\_\_', <slot wrapper '\_\_add\_\_' of ... ...

### fileinput 模块 遍历文本文件

通过fileinput来编历所有的D盘下的文本文件，将每一行的长度打印出来

import fileinput

from glob import glob

for line in fileinput.input(glob(r'd:/\*.txt')):

print fileinput.lineno(), u'文件:', fileinput.filename(), /

u'行号:', fileinput.filelineno(), u'长度:', len(line.strip('/n'))

fileinput.close()

input()接受要编历的所有文件路径的列表，通过filename()返回当前正在读取的文件的文件名，filelineno()返回当前读取的行的行号，而lineno()返回当前已经读取的行的数量（或者序号）。其实，模块内部通过FileInput类来实现文件的编历读取，input()在内部创建了该类的一个对象，当处理完数据行之后，通过fileinput.close()来关闭这个内部对象。

### uniout模块 打印可读的字符，而不是转义的字符串。

import uniout #只要导入uniout模块就行，print显示的不再是Unicode字符而是汉字

from pprint import pprint

langs = ['你好，世界！']

pprint(langs) ['你好，世界！']

print langs

### xpinyin 一个用于把汉字转换为拼音的库。

from xpinyin import Pinyin

p = Pinyin()

#默认分隔符为-

p.get\_pinyin(u"上海") #'shang-hai'

# 显示拼音符号

p.get\_pinyin(u"上海", show\_tone\_marks=True) #'shàng-hǎi'

#分隔符设置为无

p.get\_pinyin(u"上海", '') #'shanghai'

#只显示拼音首字母

p.get\_initials(u"上海") #'S-H'

**如果方法中传入变量，那么直接加前缀是不可以了。而是要将变量转为utf-8编码：**

wordvalue = '中国'

wordvalue= unicode(wordvalue,'utf-8')

或者wordvalue= wordvalue.decode('utf-8')

s = p.get\_initials(wordvalue, u'').lower() #'zg'

#把日志中文部分全部弄成拼音

class convertToPinyin(object):

def \_\_init\_\_(self,filename,outfilename=None):

import xpinyin

with open(filename,'r+') as gisy:

lines=gisy.readlines()

newlines=[]

p = xpinyin.Pinyin()

for index,item in enumerate(lines):

#如果方法中传入变量，那么直接加前缀是不可以了。而是要将变量转为utf-8编码

item= unicode(item,'utf-8')

line=p.get\_pinyin(item,'', show\_tone\_marks=True)

print line

newlines.append(line)

if not outfilename:

outfilename=filename

with open(outfilename,'w+') as gisy:

gisy.writelines(newlines)

### unidecode模块 Unicode 文本的 ASCII 转换形式

>>> from unidecode import unidecode

>>> unidecode(u'ko\u017eu\u0161\u010dek')

'kozuscek'

>>> unidecode(u'30 \U0001d5c4\U0001d5c6/\U0001d5c1')

'30 km/h'

>>> unidecode(u"\u5317\u4EB0")

'Bei Jing '

### pangu模块 在中日韩语字符和数字字母之间添加空格

import pangu

pangu.spacing('新八的構造成分有95%是眼鏡、3%是水、2%是垃圾')

# output: u'新八的構造成分有 95% 是眼鏡、3% 是水、2% 是垃圾'

pangu.spacing\_text("Mr.龍島主道：「Let's Party!各位高明博雅君子！」")

# output: u"Mr. 龍島主道：「Let's Party! 各位高明博雅君子！」"

spacing\_text() is an alias of spacing(). spacing\_text = spacing

### ftfy模块 让Unicode文本更完整更连贯。

from ftfy import fix\_text

>>> print(fix\_text(u'This text should be in â€œquotesâ€\x9d.'))

This text should be in "quotes".

>>> print(fix\_text(u'uÌˆnicode'))

ünicode

>>> print(fix\_text(u'Broken text&hellip; it&#x2019;s ﬂubberiﬁc!',

... normalization='NFKC'))

Broken text... it's flubberific!

>>> print(fix\_text(u'HTML entities &lt;3'))

HTML entities <3

>>> print(fix\_text(u'<em>HTML entities in HTML &lt;3</em>'))

<em>HTML entities in HTML &lt;3</em>

>>> print(fix\_text(u'\001\033[36;44mI&#x92;m blue, da ba dee da ba '

... 'doo&#133;\033[0m', normalization='NFKC'))

I'm blue, da ba dee da ba doo...

>>> print(fix\_text(u'ＬＯＵＤ　ＮＯＩＳＥＳ'))

LOUD NOISES

>>> print(fix\_text(u'ＬＯＵＤ　ＮＯＩＳＥＳ', fix\_character\_width=False))

ＬＯＵＤ　ＮＯＩＳＥＳ

报错UnicodeError: Hey wait, this isn't Unicode

### schema 数据校验

**from schema import Schema**

Schema(xxx).validate(data)

其中参数xxx可以是：

基本数据类型比如int/float/list/tuple/set/str等。

函数

list, tuple, set, frozenset, dict实例

某个重写了validate方法的类。比如Use。

如果验证通过,会返回验证通过的数据。

如果验证不通过,会抛出SchemaError异常。

基本类型

>>Schema(int).validate(123)

函数

>>Schema(os.path.exists).validate('/path/to/file')

先转换再验证

>>from schema import Use

>>Schema(Use(int)).validate('123')

如果Schema的参数是一个容器，list, tuple, set or frozenset，那么待验证的数据也必须是个对应的容器，且容器元素在枚举内。

>>Schema([1, 0]).validate([1, 1, 0, 1])

>>Schema((int, float)).validate((5, 7, 8, 'not int or float here'))

可以对字典的key和value验证

>>Schema({'name': str, 'age': lambda n: 18 < 99}).validate({'name': 'Sue', 'age': 28})

>>Schema({str: int, # 代表字符串的key的value类型必须是整型

int: None}) #整型的key的value必须是None

如果只想对部分键值验证，而不管其它的，可以用object。

>>Schema({'id': int,str: object}).validate({'id':10,'age':2,'name':'Rooney'})

可选、与、或

>>from schema import Optional,And, Or

>>Schema({'name': str, Optional('occupation'): str}).validate({'name': 'Sam'})

>>Schema({'age': And(int, lambda n: 0 < n < 99)}).validate({'age': 7})

>>Schema(And(Or(int, float), lambda x: x > 0)).validate(3.1415)

### PyH模块 生成HTML

from pyh import \*

page = PyH('My wonderful PyH page')

page.addCSS('myStylesheet1.css', 'myStylesheet2.css')

page.addJS('myJavascript1.js', 'myJavascript2.js')

page << h1('My big title', cl='center')

page << div(cl='myCSSclass1 myCSSclass2', id='myDiv1') << p('I love PyH!', id='myP1')

mydiv2 = page << div(id='myDiv2')

mydiv2 << h2('A smaller title') + p('Followed by a paragraph.')

page << div(id='myDiv3')

page.myDiv3.attributes['cl'] = 'myCSSclass3'

page.myDiv3 << p('Another paragraph')

page.printOut() #打印在console窗口

page.printOut('gisy.html') #输出到gisy.html

## 有关嵌入式数据库

### pickledb 一个简单，轻量级键值储存数据库

一个简单，轻量级键值储存数据库。python实现

import pickledb

db = pickledb.load('example.db', False)

db.set('key', 'value') #存，两个参数，前面为key，后面为value

db.get('key') #'value' 取key对应的值

db.getall() 获取所有keys

db.dump() #将内存的数据生成example.db文件保存 #True 必须最后加这个，否则无法存为文件

db.rem('key') #删除某个key和对应的值

db.append(key, more) '''Add more to a key's value'''

列表

db.lcreate(name) '''Create a list'''

db.ladd(name, value) '''Add a value to a list'''

db.lextend(name, seq) '''Extend a list with a sequence'''

db.lgetall(name) '''Return all values in a list'''

db.lget(name, pos) '''Return one value in a list'''

db.lrem(name) '''Remove a list and all of its values'''

db.lpop(name, pos) '''Remove one value in a list'''

db.llen(name) '''Returns the length of the list'''

db.lappend(name, pos, more) '''Add more to a value in a list'''

db.lgetall(1)[0]='b' 可以修改list中元素的值

### tinydb 纯python编写的文档型数据库

Example Code

from tinydb import TinyDB, where,Query

#建立数据库，以文件的完整路径为参数

db = TinyDB('db.json')

#插入数据

db.insert({'name': 'John', 'age': 22})

db.insert({'name': 'John', 'age': 37})

db.insert({'name': 'Bob', 'age': 42})

查询语句

User = Query() #建立查询实例

# Search for a field value

db.search(User.name == 'John')

#[{'name': 'John', 'age': 22}, {'name': 'John', 'age': 37}]

# Combine two queries with logical and

db.search((User.name == 'John') & (User.age <= 30))

#[{'name': 'John', 'age': 22}]

# Combine two queries with logical or

db.search((User.name == 'John') | (User.name == 'Bob'))

#[{'name': 'John', 'age': 22}, {'name': 'John', 'age': 37}, {'name': 'Bob', 'age': 42}]

>>> # More possible comparisons: != < > <= >=

>>> # More possible checks: where(...).matches(regex), where(...).test(your\_test\_func)

建表

from tinydb import TinyDB, where,Query

db = TinyDB('db.json')

table = db.table('gisy')

table.insert({'name': 'John', 'age': 22})

table.all()

User = Query()

print table.search(User.name == 'John')

使用中间件

from tinydb.storages import JSONStorage

from tinydb.middlewares import CachingMiddleware

db = TinyDB('/path/to/db.json', storage=CachingMiddleware(JSONStorage))

### anydbm, shelve 纯python数据库模块

# key与value必须是字符串,value不能是list等

import anydbm

try:

db = anydbm.open('db.dat', 'c')

db['string'] = "I like python."

db['key'] = 'value'

finally:

db.close()

db = anydbm.open('db.dat', 'r')

for item in db.items():

print item

for k, v in db.iteritems():

print k, v

print db['string']

db.close()

db = anydbm.open('db.dat', 'w')

db['key']='hello'

db.close()

anydbm.open(filename[, flag[, mode]])，filename是关联的文件路径，可选参数flag可以是: ‘r’: 只读, ‘w’: 可读写, ‘c’: 如果数据文件不存在，就创建，允许读写; ‘n’: 每次调用open()都重新创建一个空的文件。mode是unix下文件模式，如0666表示允许所有用户读写。

shelve模块是anydbm的增强版，它支持在”dict-like”对象中存储任何可以被pickle序列化的对象，但key也必须是字符串。同样的例子，与shelve来实现:

import shelve

db = shelve.open('db.dat', 'c')

db['string'] = "I like python."

db['list'] = ["I like python.",'grf']

db[dict] = {"I like python.":'grf'}

db['key'] = 'value'

db.close()

db = shelve.open('db.dat', 'r')

for item in db.items():

print item

for k, v in db.iteritems():

print k, v

print db['string']

db.close()

db = shelve.open('db.dat', 'w')

db['key']='hello'

db.close()

### sqlite3模块

#### 基础

# 导入SQLite驱动

import sqlite3

# 连接到SQLite数据库，数据库文件是test.db。如果文件不存在，会自动在当前目录创建:

conn = sqlite3.connect(*'test.db'*)

**#创建row类型，在创建cursor之前**

**conn.row\_factory = sqlite3.Row**

# 创建一个Cursor:

cursor = conn.cursor()

# 执行一条SQL语句，创建user表:

cursor.execute(*'create table user (id varchar(20) primary key, name varchar(20))'*)

# 继续执行一条SQL语句，插入一条记录:

cursor.execute(*'insert into user (id, name) values (\'1\', \'Michael\')'*)

cursor.execute(*'insert into user (id, name) values ("1","Michael")'*)

# 通过rowcount获得插入的行数:

cursor.rowcount

# 执行查询语句: 如果参数只是单个数值，也要采用t=(n,)的形式，因为元组是不可变的。

**cursor.execute('select \* from user where id=?', ('1',)**

# 获得查询结果集:

values = cursor.fetchall()

#获取元素值

for i in values:

print i.keys() #获取字段名

print i[1] #显示值的第二个字段值

#获取单条记录

r=cursor.fetchone()

#row类型 必须之前添加conn.row\_factory = sqlite3.Row

print len(r) #获取记录字段的数量

print r['id'] #显示字段下的值

#修改

**cursor**.execute("update catalog set name='Boy' where id = 0")

#删除

**cursor**.execute("delete from catalog where id = 1")

#该例程返回自数据库连接打开以来被修改、插入或删除的数据库总行数。

conn.total\_changes()

# 关闭Cursor:

cursor.close()

# 提交事务:

conn.commit()

# 关闭Connection:

conn.close()

#只要搞清楚Connection和Cursor对象，打开后一定记得关闭，就可以放心地使用。

返回查询的结果信息

result= **cursor**.fetchall()方法获取所有记录。如果只有1条，返回元祖如下：((a,b),)

results= **cursor**.fetchmany(5)获取前5条

result= **cursor**.fetchone() 获取一条 返回元祖如下：(a,b)

　 使用以上3个方法后，指针是会发生移动的。所以，若不重置指针，那么之后使用fetchall的信息将只会包含指针后面的行内容。

sqlite3没有滚动游标这个功能 **cursor**.scroll()

#### 注意点

sqlite可视化管理工具 SQLite Expert Personal 4.x DB Browser for SQLite <http://sqlitebrowser.org/>

程序创建表时，不能打开sqlite可视化管理工具，否则报错sqlite3.OperationalError: database is locked

使用Cursor对象执行insert，update，delete语句时，执行结果由rowcount返回影响的行数，就可以拿到执行结果。

使用Cursor对象执行select语句时，通过fetchall()可以拿到结果集。结果集是一个list，每个元素都是一个tuple，对应一行记录

如果SQL语句带有参数，那么需要把参数按照位置传递给execute()方法，有几个?占位符就必须对应几个参数，例如：

cursor.execute('select \* from user where name=? and pwd=?', ('abc', '123456'))

在调用connect函数的时候，指定库名称，如果指定的数据库存在就直接打开这个数据库，如果不存在就新创建一个再打开。

cx = sqlite3.connect("E:/test.db")

可以创建数据库在内存中。 con = sqlite3.connect(":memory:")

在 SQLite 中，ALTER TABLE 命令允许用户重命名表，或向现有表添加一个新的列。重命名列，删除一列，或从一个表中添加或删除约束都是不可能的。

CHECK 约束：CHECK 约束确保某列中的所有值满足一定条件。

SALARY REAL CHECK(SALARY > 0)

IS NOT NULL 运算符的用法 WHERE SALARY IS NOT NULL; WHERE SALARY IS NULL;

SELECT DISTINCT name FROM COMPANY;

外键约束CREATE TABLE Students (Id integer PRIMARY KEY, TeacherId integer, FOREIGN KEY(TeacherId) REFERENCES Teachers(id)

.db-journal文件是sqlite的一个临时的日志文件，主要用于sqlite事务回滚机制，在事务开始时产生，在事务结束时删除；当程序发生崩溃或者系统断电时该文件将留在磁盘上，以便下次程序运行时进行事务回滚。

#### 数据类型

SQLite 存储类

每个存储在 SQLite 数据库中的值都具有以下存储类之一：

存储类 描述

NULL 值是一个 NULL 值。

INTEGER 值是一个带符号的整数，根据值的大小存储在 1、2、3、4、6 或 8 字节中。

REAL 值是一个浮点值，存储为 8 字节的 IEEE 浮点数字。

TEXT 值是一个文本字符串，使用数据库编码（UTF-8、UTF-16BE 或 UTF-16LE）存储。

BLOB 值是一个 blob 数据，完全根据它的输入存储。

SQLite 的存储类稍微比数据类型更普遍。INTEGER 存储类，例如，包含 6 种不同的不同长度的整数数据类型。

SQLite 没有单独的 Boolean 存储类。相反，布尔值被存储为整数 0（false）和 1（true）。

SQLite 没有一个单独的用于存储日期和/或时间的存储类，但 SQLite 能够把日期和时间存储为 TEXT、REAL 或 INTEGER 值。

TEXT 格式为 "YYYY-MM-DD HH:MM:SS.SSS" 的日期。

REAL 从公元前 4714 年 11 月 24 日格林尼治时间的正午开始算起的天数。

INTEGER 从 1970-01-01 00:00:00 UTC 算起的秒数。

您可以以任何上述格式来存储日期和时间，并且可以使用内置的日期和时间函数来自由转换不同格式。

#### 提升SQLite数据插入效率低、速度慢的方法

第一种 显式开启事务

开启事务只需在上述代码的前后各加一句开启与提交事务的命令即可：

sqlite3\_exec(db,"begin;",0,0,0);

…… #执行插入数据库命令行

sqlite3\_exec(db,"commit;",0,0,0);

所谓”事务“就是指一组SQL命令，这些命令要么一起执行，要么都不被执行。在SQLite中，每调用一次sqlite3\_exec()函数，就会隐式地开启了一个事务，如果插入一条数据，就调用该函数一次，事务就会被反复地开启、关闭，会增大IO量。如果在插入数据前显式开启事务，插入后再一起提交，则会大大提高IO效率，进而加数据快插入速度。

第二种 写同步(synchronous)

简要说来，full写入速度最慢，但保证数据是安全的，不受断电、系统崩溃等影响，而off可以加速数据库的一些操作，但如果系统崩溃或断电，则数据库可能会损毁。

在打开数据库之后，直接插入以下:

sqlite3\_exec(db,"PRAGMA synchronous = OFF; ",0,0,0);

## 有关ORM

ORM框架的作用就是把数据库表的一行记录与一个对象互相做自动转换。

可以直接用orm进行建表

正确使用ORM的前提是了解关系数据库的原理。主要的ORM含有

SQLObject Storm Django’s ORM peewee SQLAlchemy

### peewee

小型orm 支持postgresql、mysql和sqlite。

#### 模板

为了创建数据库模型映射，我们实现了一个Person 类 和一个Address类 来映射对应的数据库表。

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

from peewee import \*

from datetime import date

mysql\_db=MySQLDatabase('test',host='192.168.166.17',user='poc\_rw',

passwd='1qaz2wsx1qaz2wsx',port=3306,charset="utf8") #数据库名必须放第一个

#一个属性对应一个字段

class Person(Model):

name = CharField() #CharField 为抽象数据类型 相当于 varchar

birthday = DateField() #DateField 相当于 date

is\_relative = BooleanField() #BooleanField 相当于 bool

class Meta: # 所用数据库为db，必须有这个，对应建立的数据库实例mysql\_db

database =mysql\_db

#表格模型 Pet

class Pet(Model):

owner = ForeignKeyField(Person, related\_name='pets') #外连接的声明(和Person关联) 外键

name = CharField()

animal\_type = CharField()

class Meta:

database =mysql\_db

mysql\_db.connect() #连接数据库

print mysql\_db.get\_conn() #获取数据库连接

#最后要mysql\_db.close()

mysql\_db.create\_tables([Person, Pet]) #建表，只需用执行一次，讲完表后续无须执行

#插入数据 当你调用 save 方法时，将返回生效的记录行数。

uncle\_bob = Person(name='Bob', birthday=date(1967, 1, 28), is\_relative=True)

uncle\_bob.save() #如果方法为自定义类的实例，需要再执行.save()方法

#插入数据 #如果是类.create方法，则无须save()方法，即可插入表

herb\_fido = Pet.create(owner=herb, name='Fido', animal\_type='dog')

herb\_mittens = Pet.create(owner=herb, name='Mittens', animal\_type='cat')

#插入多条数据

data = [{'username':'sla'},{'username':'saf'},{'username':'djangs'}]

**with loglogicdb.atomic():**

Logtrunk.insert\_many(data).execute()

#修改数据，为了更新一行数据，只需修改 model 实例然后调用 save 方法保存变更

herb\_fido.name = 'Marry'

herb\_fido.save()

#Update更新操作

q = User.update(active=False).where(User.registration\_expired == True)

q.execute()

print q.execute() #这样可以查看被执行的行数数量

q = Logicflow.update(flowjson=flowitem).where(Logicflow.id==flowid)

#update括号内不需要加表名了，直接字段名 flowjson

#Insert插入操作

q = User.insert(username='admin', active=True, registration\_expired=False)

q.execute()

Delete删除操作

q = User.delete().where(User.active == False)

q.execute()

#查询

#获取单条记录 使用 SelectQuery.get() 从数据库中获取单条记录。

grandma = Person.select().where(Person.name == 'Grandma L.').get()

print grandma.name #这样才能读取出字段信息

#查询结果为单条、多条记录都可使用

query =Logtrunk.select().where(Logtrunk.log ==item)

for i in query:print i.log #只能循环，一行行输出

#简便的 Model.get() 来实现相同的功能

grandma = Person.get(Person.name == 'Grandma L.')

print grandma.log

print grandma.\_\_dict\_\_ #查看该对象下所有属性

#等于实例查询 uncle\_bob为之前插入数据时创建的实例

for pet in Pet.select().where(Pet.owner == uncle\_bob).order\_by(Pet.name):

print pet.name

#降序查询

for person in Person.select().order\_by(Person.birthday.desc()):

print person.name, person.birthday

#查询Person表中person所拥有的pet数量及名字和类型 person.pets.count() 计数

for person in Person.select():

print person.name, person.pets.count(), 'pets'

for pet in person.pets:

print ' ', pet.name, pet.animal\_type

#or形式查询

d1940 = date(1940, 1, 1)

d1960 = date(1960, 1, 1)

query = (Person.select().where((Person.birthday < d1940) | (Person.birthday > d1960)))

for person in query:

print person.name, person.birthday

#按照expression查询person名开头为小写或大写 G 的person

expression = (fn.Lower(fn.Substr(Person.name, 1, 1)) == 'g')

for person in Person.select().where(expression):

print person.name

#所有其他的 SQL 子句也都是可用的，比如：

#group\_by()\having()\limit() 和 offset()

#查看删除数据行数 delete\_instance() 的返回值是被从数据库中移除的数据的总行数

herb\_mittens.delete\_instance()

#变量赋值

bob\_fido = herb\_fido

关闭数据库

mysql\_db.close()

#### 表达式

Peewee表达 相当于SQL

fn.Count(Tweet.id).alias('count') Count(t1."id") AS count

fn.Lower(fn.Substr(User.username, 1, 1)) Lower(Substr(t1."username", 1, 1))

fn.Rand().alias('random') Rand() AS random

fn.Stddev(Employee.salary).alias('sdv') Stddev(t1."salary") AS sdv

fn.Max(Tweet.created\_date) max(Tweet.created\_date)

#### 细节

1. 查询

gisy =Logtrunk.select().where(Logtrunk.log =='df')

gisy.get() #如果查询没找到数据，get()则抛出异常 LogtrunkDoesNotExist

for i in gisy:print i.id,i.log #这种方式没找到，不会报错

1. 查询两字段，并组合成字典

logdict={}

print loglist

for i in logtrunkResult:

logdict.setdefault(i.id,i.value)

或者这样：

allflow=[i.flowjson for i in all]

idflow=[i.id for i in all]

allflowdict={key:value for key,value in zip(idlist,allflow)}

查不到数据就会抛出异常：instance matching query does not exist

#### 连接数据库

from peewee import \*

mysql\_db = MySQLDatabase('test',host='192.168.166.17',user='poc\_rw',passwd='1qaz2wsx1qaz2wsx',port=3306,charset="utf8")

loglogicdb = SqliteDatabase(*'loglogic.db'*, \*\*{})

mysql\_db.connect() #连接数据库

print mysql\_db.get\_conn() #获取数据库连接

不过，get\_conn()不止是获取数据库连接：如果没有连接存在，新建一个。

所以我们可以使用get\_conn()来实现单例数据库连接.

可以看到，两次使用建立connect()函数建立的并不是一个连接

#### 建表

orm的理念是一个表对应于一个类, 字段对应于类的属性

from peewee import \*

mysql\_db = MySQLDatabase("mydb", user = "root", passwd = "123456", charset = "utf8")

mysql\_db.connect() #连接数据库

class MySQLModel(Model):

class Meta:

database = mysql\_db

class User(MySQLModel): #类的小写即表名

username = CharField() #字段声明

email = CharField()

User.create\_table() #建表, 其实create\_table是一个类方法，但也可以被其实例调用

#### 作用于已存在的数据库

如果你已经拥有了一个数据库，你可以使用 pwiz, a model generator. 自动生成 peewee models。 比如，我有一个名字叫 gisy 的 postgresql 数据库，我就可以这样运行:

python -m pwiz -e postgresql gisy > blog\_models.py

python -m pwiz -e sqlite3 loglogic.db > gisymodels.py 如果是sqlite3数据库记得将数据库名称后缀.db也加上

#### 插入多条记录 最快方法

在执行语句前加入**with loglogicdb.atomic(): 可以使速度变快数百倍**

loglogicdb = SqliteDatabase('loglogic.db', \*\*{})

第一种

paralist=[]

for item in lines:

parameters={}

parameters.setdefault('log',item)

paralist.append(parameters)

**with loglogicdb.atomic():**

Logtrunk.insert\_many(paralist).execute()

第二种

a=time.time()

**with loglogicdb.atomic():**

for index,item in enumerate(lines):

try:

logtrunk\_save= Logtrunk.create(log=item)

except peewee.IntegrityError:

continue

print time.time()-a

### SQLAlchemy

大型orm 企业级 API，使得代码有健壮性和适应性，灵活的设计，使得能轻松写复杂查询

#### 基础

第一步，导入SQLAlchemy，并初始化DBSession：

from sqlalchemy import Column, String, create\_engine

from sqlalchemy.orm import sessionmaker

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base() # 创建对象的基类:

class User(Base): # 定义User对象:

\_\_tablename\_\_ = 'user' # 表的名字:

id = Column(String(20), primary\_key=True) # 表的结构:

name = Column(String(20))

# 初始化数据库连接:

engine = create\_engine('mysql+mysqlconnector://root:password@localhost:3306/test')

DBSession = sessionmaker(bind=engine) # 创建DBSession类型:

create\_engine()用来初始化数据库连接。SQLAlchemy用一个字符串表示连接信息：

'数据库类型+数据库驱动名称://用户名:口令@机器地址:端口号/数据库名'

添加和查询

session = DBSession() # 创建session对象:

new\_user = User(id='5', name='Bob') # 创建新User对象:

session.add(new\_user) # 添加到session:

session.commit()# 提交即保存到数据库

# 创建Query查询，filter是where条件，最后调用one()返回唯一行，如果调用all()则返回所有行:

user = session.query(User).filter(User.id=='5').one()

print user.name

session.close()# 关闭session:

可见，关键是获取session，然后把对象添加到session，最后提交并关闭。DBSession对象可视为当前数据库连接。

复杂创建表

class User(Base):

\_\_tablename\_\_ = "users"

id = Column("id", Integer, primary\_key=True)

name = Column("name", String)

fullname = Column("fullname", String)

password = Column("password", String)

addresses = relation("Address", order\_by="Address.id", backref="user")

def \_\_init\_\_(self, id=None, name=None, fullname=None, password=None, addresses=[]):

self.id = id

self.name = name

self.fullname = fullname

self.password = password

self.addresses = addresses

def \_\_repr\_\_(self):

return "<User '{name}' '{fullname}' '{password}' {addresses}>".format(name=self.name, fullname=self.fullname, password=self.password, addresses=self.addresses)

class Address(Base):

\_\_tablename\_\_ = "address"

id = Column(Integer, primary\_key=True)

email\_address = Column(String, nullable=False)

user\_id = Column(Integer, ForeignKey("users.id"))

#user = relation("User", backref="addresses", order\_by="Address.id")

def \_\_init\_\_(self, email\_address=None):

self.email\_address = email\_address

def \_\_repr\_\_(self):

return "<Address ({email\_address}) user={user}>".format(email\_address=self.email\_address, user=self.user.name)

**提供两个对象之间的一对多、多对多等功能**

如果一个User拥有多个Book，就可以定义一对多关系如下：

class User(Base):

\_\_tablename\_\_ = 'user'

id = Column(String(20), primary\_key=True)

name = Column(String(20))

# 一对多:

books = relationship('Book')

class Book(Base):

\_\_tablename\_\_ = 'book'

id = Column(String(20), primary\_key=True)

name = Column(String(20))

# “多”的一方的book表是通过外键关联到user表的:

user\_id = Column(String(20), ForeignKey('user.id'))

当我们查询一个User对象时，该对象的books属性将返回一个包含若干个Book对象的list。

#### 查询语句

#简单查询

print(session.query(User).all())

print(session.query(User.name, User.fullname).all())

print(session.query(User, User.name).all())

#带条件查询

print(session.query(User).filter\_by(name='user1').all())

print(session.query(User).filter(User.name == "user").all())

print(session.query(User).filter(User.name.like("user%")).all())

#多条件查询

print(session.query(User).filter(and\_(User.name.like("user%"), User.fullname.like("first%"))).all())

print(session.query(User).filter(or\_(User.name.like("user%"), User.password != None)).all())

#sql过滤

print(session.query(User).filter("id>:id").params(id=1).all())

#关联查询

print(session.query(User, Address).filter(User.id == Address.user\_id).all())

print(session.query(User).join(User.addresses).all())

print(session.query(User).outerjoin(User.addresses).all())

#聚合查询

print(session.query(User.name, func.count('\*').label("user\_count")).group\_by(User.name).all())

print(session.query(User.name, func.sum(User.id).label("user\_id\_sum")).group\_by(User.name).all())

#子查询

stmt = session.query(Address.user\_id, func.count('\*').label("address\_count")).group\_by(Address.user\_id).subquery()

print(session.query(User, stmt.c.address\_count).outerjoin((stmt, User.id == stmt.c.user\_id)).order\_by(User.id).all())

#exists

print(session.query(User).filter(exists().where(Address.user\_id == User.id)))

print(session.query(User).filter(User.addresses.any()))

## 有关文件比较

### filecmp模块 比较文件及文件夹的内容

filecmp模块用于比较文件及文件夹的文件属性及内容，它是一个轻量级的工具，使用非常简单。

filecmp定义了两个函数，用于方便地比较文件与文件夹：

filecmp.cmp(f1, f2[, shallow])： 默认shallow=1

比较文件名为f1和f2的文件，相同返回True，不相同返回False

参数f1, f2指定要比较的文件的路径。可选参数shallow指定比较文件时是否需要考虑文件本身的属性（通过os.stat函数可以获得文件属性）。

filecmp.cmp('/tmp/file1','/tmp/file3' ,shallow=0)

True

f1=r'D:\eclipse\_workspace\gisydata\40600251299.txt'

f2=r'D:\eclipse\_workspace\gisydata\41480458581.txt'

print filecmp.cmp(f1, f2 ,shallow=0)

如果参数shallow为True，只判断os.stat()函数返回内容是否相同，如果相同就返回True，否则再比较文件内容是否相同，如果相同也返回True。使用shallow参数可以快速地比较文件是否有修改过。

也就是如果文件内容相同，但是文件修改时间不同，就认为为false,不相同

filecmp.cmpfiles(dir1, dir2, common[, shallow])：

比较两个文件夹内指定文件是否相等。

参数dir1, dir2指定要比较的文件夹，参数common指定要比较的文件名列表。函数返回包含3个list元素的元组，分别表示匹配、不匹配以及错误的文件列表。错误的文件指的是不存在的文件，或文件被琐定不可读，或没权限读文件，或者由于其他原因访问不了该文件。

对于大目录树的递归比较或者完成更完整的分析，dircmp类会更有用。

比较两个目录文件内容filecmp.dircmp()

>>> filecmp.dircmp('/tmp/dir1','/tmp/dir2')

<filecmp.dircmp instance at 0x28584c2c>

>>> filecmp.dircmp('/tmp/dir1','/tmp/dir2').same\_files

['file.txt', 'file2', 'file1']

>>> filecmp.dircmp('/tmp/dir1','/tmp/dir2').diff\_files

['file3']

filecmp模块中定义了一个dircmp类，用于比较文件夹，通过该类比较两个文件夹，可以获取一些详细的比较结果（如只在A文件夹存在的文件列表），并支持子文件夹的递归比较。

dircmp提供了三个方法用于报告比较的结果：

report()：只比较指定文件夹中的内容（文件与文件夹）

report\_partial\_closure()：比较文件夹及第一级子文件夹的内容

report\_full\_closure()：递归比较所有的文件夹的内容

dircmp还提供了下面这些属性用于获取比较的详细结果：

left\_list：左边文件夹中的文件与文件夹列表；

right\_list：右边文件夹中的文件与文件夹列表；

common：两边文件夹中都存在的文件或文件夹；

left\_only：只在左边文件夹中存在的文件或文件夹；

right\_only：只在右边文件夹中存在的文件或文件夹；

common\_dirs：两边文件夹都存在的子文件夹；

common\_files：两边文件夹都存在的子文件；

common\_funny：两边文件夹都存在的子文件夹；

same\_files：匹配的文件；

diff\_files：不匹配的文件；

funny\_files：两边文件夹中都存在，但无法比较的文件；

subdirs：我没看明白这个属性的意思，python手册中的解释如下：A dictionary mapping names in common\_dirs to dircmp object

filecmp.clear\_cache()

清除文件比较缓冲区。当比较的文件不断地快速修改时，就可以使用这个函数来更新文件的信息

### difflib模块

difflib是python提供的比较序列(string list)差异的模块。实现了三个类：

1>SequenceMatcher 任意类型序列的比较 (可以比较字符串)

2>Differ 对字符串进行比较

3>HtmlDiff 将比较结果输出为html格式.

#### SequenceMatcher 任意类型序列的比较 (可以比较字符串)

import difflib

from pprint import pprint

a = 'pythonclub.org is wonderful'

b = 'Pythonclub.org also wonderful'

s = difflib.SequenceMatcher(None, a, b)

# SequeceMatcher(None,a,b)创建序列比较对象，将以a作为参考标准进行

pprint(s.get\_matching\_blocks())

# [Match(a=1, b=1, size=14), Match(a=16, b=17, size=1), Match(a=17, b=19, size=10), Match(a=27, b=29, size=0)]

for tag, i1, i2, j1, j2 in **s.get\_opcodes()**:

print ("%7s a[%d:%d] (%s) b[%d:%d] (%s)" % (tag, i1, i2, a[i1:i2], j1, j2, b[j1:j2]))

# a,b表示待比较的两个序列，生成序列比较对象后，调用该对象的get\_opcodes()方法，将返回一个元组

(tag,i1,i2,j1,j2).tag表示序列分片的比较结果.i1,i2表示序列a的索引，j1,j2表示序列b的索引.

get\_opcodes()返回元组(tag,i1,i2,j1,j2)的含义

%7s 表示该变量打印的区域统一为7个字符，不足则左边空格补齐。

s.get\_opcodes():

replace a[0:1] (p) b[0:1] (P)

equal a[1:15] (ythonclub.org ) b[1:15] (ythonclub.org )

replace a[15:16] (i) b[15:17] (al)

equal a[16:17] (s) b[17:18] (s)

insert a[17:17] () b[18:19] (o)

equal a[17:27] ( wonderful) b[19:29] ( wonderful)

#### Differ 对字符串进行比较

import difflib

diff= difflib.Differ().compare("start up","starT u4p")

print list(diff)

[' s', ' t', ' a', ' r', '- t', '+ T', ' ', ' u', '+ 4', ' p']

#### HtmlDiff 将比较结果输出为html格式

import difflib

s=difflib.HtmlDiff.make\_file(difflib.HtmlDiff(),'start up','storT up')

f=open("gisy.html","a")

f.write(s)

f.close

difflib.HtmlDiff.make\_file(difflib.HtmlDiff(),'start up','storT up')

等于

a= difflib.HtmlDiff()

a. make\_file('start up','storT up')

### Levenshtein模块

#### 基础

编辑距离，又称Levenshtein距离，是指两个字串之间，由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括：将一个字符替换成另一个字符，插入一个字符，删除一个字符。

python有一个官方扩展包（在pypi里面，即python package index），叫做python-Levenshtein，这个包不仅可以计算编辑距离，还能计算hamming（汉明）距离，Jaro-Winkler距离

import Levenshtein

#### 方法

Levenshtein.hamming(str1, str2)计算汉明距离。

要求str1和str2必须长度一致。是描述两个等长字串之间对应位置上不同字符的个数 如果不相等，报错ValueError: hamming expected two strings of the same length

Levenshtein.distance(str1, str2)计算编辑距离（也成Levenshtein距离）。

是描述由一个字串转化成另一个字串最少的操作次数，在其中的操作包括插入、删除、替换。

Levenshtein.ratio(str1, str2) 计算莱文斯坦比。

计算公式 r = (sum - ldist) / sum, 其中sum是指str1 和 str2 字串的长度总和，ldist是类编辑距离  
这里的类编辑距离不是2中所说的编辑距离，2中三种操作中每个操作+1，而在此处，删除、插入依然+1，但是替换+2

Levenshtein.jaro(s1, s2) 计算jaro距离

Levenshtein.jaro\_winkler(s1, s2) 计算Jaro–Winkler距离

### fuzzywuzzy 模糊字符串匹配

Fuzzy string matching like a boss. It uses Levenshtein Distance to calculate the differences between sequences

首先导入相关模块

from fuzzywuzzy import fuzz

from fuzzywuzzy import process

简单比

fuzz.ratio("this is a test", "this is a test!") 97

部分比

fuzz.partial\_ratio("this is a test", "this is a test!") 100

单词排序比

fuzz.ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear") 91

fuzz.token\_sort\_ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear") 100

单词集合比

fuzz.token\_sort\_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear") 84

fuzz.token\_set\_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear") 100

提取类似元素 Process

choices = ["Atlanta Falcons", "New York Jets", "New York Giants", "Dallas Cowboys"]

process.extract("new york jets", choices, limit=2)

[('New York Jets', 100), ('New York Giants', 78)]

process.extractOne("cowboys", choices)

("Dallas Cowboys", 90)

### distance模块

该模块可供计算Levenshtein, Hamming, Jaccard, and Sorensen distance

import distance

t1 = ("de", "ci", "si", "ve")

t2 = ("de", "ri", "si", "ve")

distance.levenshtein(t1, t2) #1

distance.hamming("fat", "cat", normalized=True) 0.3333333333333333

distance.nlevenshtein("abc", "acd", method=1) # shortest alignment 0.6666666666666666

distance.nlevenshtein("abc", "acd", method=2) # longest alignment 0.5

sor= 1 - distance.sorensen(a,b) 余弦相似性

jar= 1 - distance.jaccard(a,b)

Jaccard距离用来度量两个集合之间的差异性，它是Jaccard的相似系数的补集，被定义为1减去Jaccard相似系数。

**Jaccard相似指数用来度量两个集合之间的相似性，它被定义为两个集合交集的元素个数除以并集的元素个数。**

distance.jaccard("decide", "resize")

0.7142857142857143

### 比较两字符串相似度

汉明距离 两种

import Levenshtein

Levenshtein.ratio('hello world', 'hello')

Result: 0.625

import difflib

difflib.SequenceMatcher(None, 'hello world', 'hello').ratio() 第一个参数一定要是None

Result: 0.625

fuzz.partial\_ratio(sline.lower(),item.lower())>90

def similar(seq1, seq2):

return difflib.SequenceMatcher(None,a=seq1.lower(), b=seq2.lower()).ratio() > 0.9

print similar(a, b) #True

print similar('Hello, world', 'Hi, world') #False

汉明距离

两个等长字符串之间的汉明距离（英语：Hamming distance）是两个字符串对应位置的不同字符的个数。换句话说，它就是将一个字符串变换成另外一个字符串所需要替换的字符个数。

汉明重量是字符串相对于同样长度的零字符串的汉明距离，也就是说，它是字符串中非零的元素个数：对于二进制字符串来说，就是1的个数，所以11101的汉明重量是4。

## 有关自然语言处理

用来处理人类语言的库。

NLTK – 一个先进的平台，用以构建处理人类语言数据的 Python 程序。

jieba – 中文分词工具。

langid.py – 独立的语言识别系统。

Pattern – Python 网络信息挖掘模块。

SnowNLP – 一个用来处理中文文本的库。

TextBlob – 为进行普通自然语言处理任务提供一致的 API。

TextGrocery –tgrocery模块 一简单高效的短文本分类工具，基于 LibLinear 和 Jieba。

ctypes 载入 dll 产生的 WindowsError: [Error 126] 错误原因

导致这样的问题有两个最主要的原因

1 系统无法定位到你的DLL 动态库

2 你的DLL 动态库依赖于其他其他DLL 动态库无法被系统找到。

### jieba模块 中文分词

支持三种分词模式：

精确模式，试图将句子最精确地切开，适合文本分析；

全模式，把句子中所有的可以成词的词语都扫描出来, 速度非常快，但是不能解决歧义；

搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

支持繁体分词

支持自定义词典

#### 分词

import jieba

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 seg\_list

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))

【全模式】: 我/ 来到/ 北京/ 清华/ 清华大学/ 华大/ 大学

【精确模式】: 我/ 来到/ 北京/ 清华大学

【新词识别】：他, 来到, 了, 网易, 杭研, 大厦 (此处，“杭研”并没有在词典中，但是也被Viterbi算法识别出来了)

【搜索引擎模式】： 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

jieba.cut 方法接受三个输入参数: 需要分词的字符串；cut\_all 参数用来控制是否采用全模式；HMM 参数用来控制是否使用 HMM 模型

jieba.cut\_for\_search 方法接受两个参数：需要分词的字符串；是否使用 HMM 模型。该方法适合用于搜索引擎构建倒排索引的分词，粒度比较细。

待分词的字符串可以是 unicode 或 UTF-8 字符串、GBK 字符串。注意：不建议直接输入 GBK 字符串，可能无法预料地错误解码成 UTF-8

jieba.cut 以及 jieba.cut\_for\_search 返回的结构都是一个可迭代的 generator，可以使用 for 循环来获得分词后得到的每一个词语(unicode)，或者用", ".join(seg\_list)

jieba.lcut 以及 jieba.lcut\_for\_search 直接返回 list

jieba.Tokenizer(dictionary=DEFAULT\_DICT) 新建自定义分词器，可用于同时使用不同词典。jieba.dt 为默认分词器，所有全局分词相关函数都是该分词器的映射。

#中文分词

class SeparateWords(object):

def \_\_init\_\_(self,filename):

import jieba

with open(filename,'r+') as gisy:

lines=gisy.readlines()

newlines=[]

for index,item in enumerate(lines):

seg\_list = jieba.cut(item) # 默认是精确模式

line="/".join(seg\_list)

newlines.append(line)

with open('10.txt','w+') as gisy:

gisy.writelines(newlines)

#### 添加自定义词典

载入词典

开发者可以指定自己自定义的词典，以便包含 jieba 词库里没有的词。虽然 jieba 有新词识别能力，但是自行添加新词可以保证更高的正确率

用法： jieba.load\_userdict(file\_name) # file\_name 为文件类对象或自定义词典的路径

词典格式和 dict.txt 一样，一个词占一行；每一行分三部分：词语、词频（可省略）、词性（可省略），用空格隔开，顺序不可颠倒。file\_name 若为路径或二进制方式打开的文件，则文件必须为 UTF-8 编码。

词频省略时使用自动计算的能保证分出该词的词频

jieba.load\_userdict("word.txt")

调整词典

使用 add\_word(word, freq=None, tag=None) 和 del\_word(word) 可在程序中动态修改词典。

使用 suggest\_freq(segment, tune=True) 可调节单个词语的词频，使其能（或不能）被分出来。

注意：自动计算的词频在使用 HMM 新词发现功能时可能无效。

print('/'.join(jieba.cut('如果放到post中将出错。', HMM=False)))

jieba.suggest\_freq(('中', '将'), True) #中 将两者将分开

print('/'.join(jieba.cut('如果放到post中将出错。', HMM=False)))

Tokenize：返回词语在原文的起止位置

from \_\_future\_\_ import print\_function, unicode\_literals

import jieba

result = jieba.tokenize(u'京东商城有限公司')

for tk in result:

print("word %s\t\t start: %d \t\t end:%d" % (tk[0],tk[1],tk[2]))

词性标注

ieba.posseg.POSTokenizer(tokenizer=None) 新建自定义分词器，tokenizer 参数可指定内部使用的 jieba.Tokenizer 分词器。jieba.posseg.dt 为默认词性标注分词器。

import jieba.posseg as pseg

words = pseg.cut("我爱北京天安门")

for word, flag in words:print('%s %s' % (word, flag))

### Snownlp模块 用来处理中文文本的库

SnowNLP是一个python写的类库，可以方便的处理中文文本内容。

注意本程序都是处理的unicode编码，所以使用时请自行decode成unicode。

SnowNLP则实现了词性标标准，情感分析，文本分类，转换成拼音，繁体转简体，文本关键词提取（Textrank算法），文本摘要提取（Textrank算法），tf，idf，Tokenization（分割成句子），文本相似。

为实现上面的功能用到了不少的算法和模型，比如textrank、贝叶斯分类器、隐含马尔科夫模型等。对于学习研究中文分词和贝叶斯分类有一定的帮助。

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

import uniout

from snownlp import SnowNLP

s = SnowNLP(u'这个东西真心很赞')

print s.words # [u'这个', u'东西', u'真心',u'很', u'赞']

print s.tags # [(u'这个', u'r'), (u'东西', u'n'), (u'真心', u'd'), (u'很', u'd'), (u'赞', u'Vg')]

print s.sentiments # 0.9769663402895832 positive的概率 情感分析

print s.pinyin # [u'zhe', u'ge', u'dong', u'xi', u'zhen', u'xin', u'hen', u'zan']

s = SnowNLP(u'「繁體字」「繁體中文」的叫法在臺灣亦很常見。')

print s.han #「繁体字」「繁体中文」的叫法在台湾亦很常见。

#只能实现繁体转简体，SnowNLP无法实现简体转繁体

text = u'''

自然语言处理是计算机科学领域与人工智能领域中的一个重要方向。

它研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法。

自然语言处理是一门融语言学、计算机科学、数学于一体的科学。

因此，这一领域的研究将涉及自然语言，即人们日常使用的语言，

所以它与语言学的研究有着密切的联系，但又有重要的区别。

自然语言处理并不是一般地研究自然语言，

而在于研制能有效地实现自然语言通信的计算机系统，

特别是其中的软件系统。因而它是计算机科学的一部分。

'''

s = SnowNLP(text)

print s.keywords(3) # [u'语言', u'自然', u'计算机']

print s.summary(3) # [u'因而它是计算机科学的一部分', u'自然语言处理是计算机科学领域与人工智能领域中的一个重要方向', u'自然语言处理是一门融语言学、计算机科学、数学于一体的科学']

print s.sentences #输出全部文本分成句子后的列表

s = SnowNLP([[u'这篇', u'文章'],

[u'那篇', u'论文'],

[u'这个']])

print s.tf # [{u'这篇': 1, u'文章': 1}, {u'论文': 1, u'那篇': 1}, {u'这个': 1}]

print s.idf # {u'论文': 0.5108256237659907, u'这个': 0.5108256237659907, u'那篇': 0.5108256237659907, u'这篇': 0.5108256237659907, u'文章': 0.5108256237659907}

print s.sim([u'文章']) # [0.4686473612532025, 0, 0]

#运用SnowNLP来获取key 和摘要

class extractSnowNLPSummary(object):

def \_\_init\_\_(self,filename):

from snownlp import SnowNLP

with open(filename,'r+') as gisy:

lines=gisy.read()

s = SnowNLP(lines)

print s.keywords(3)

print s.summary(3)

### tgrocery模块

简单高效的短文本分类工具，基于 LibLinear 和 Jieba

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

from tgrocery import Grocery

# 新开张一个杂货铺（别忘了取名）

grocery = Grocery('sample')

# 训练文本可以用列表传入

train\_src = [

('education', '名师指导托福语法技巧：名词的复数形式'),

('education', '中国高考成绩海外认可 是“狼来了”吗？'),

('sports', '图文：法网孟菲尔斯苦战进16强 孟菲尔斯怒吼'),

('sports', '四川丹棱举行全国长距登山挑战赛 近万人参与')

]

grocery.train(train\_src)

# 也可以用文件传入（默认以tab为分隔符，也支持自定义）

grocery.train('2.txt')

# 保存模型

grocery.save()

# 加载模型（名字和保存的一样）

new\_grocery = Grocery('sample')

new\_grocery.load()

# 预测

new\_grocery.predict('考生必读：新托福写作考试评分标准')

# 测试

test\_src = [

('education', '福建春季公务员考试报名18日截止 2月6日考试'),

('sports', '意甲首轮补赛交战记录:米兰客场8战不败国米10年连胜'),

]

new\_grocery.test(test\_src)

# 同样可支持文件传入

new\_grocery.test('4.txt')

# 自定义分词模块（必须是一个函数）

custom\_grocery = Grocery('custom', custom\_tokenize=list)

## 有关网络及网络协议

### select模块

当一个服务器需要与多个客户端进行通信时，可以使用多进程或者多线程的服务器，也可以使用select模块，它可以实现异步通信。

select 和 poll 都是比较低级的函数, 用起来比较麻烦, 如果使用异步socket编程,可以使用twisted

iwtd,owtd,ewtd 分别为需要异步处理的读socket队列, 写socket队列(一般不用), 和错误socket队列, 返回事件的读写和错误socket队列

il,ol,el = select(iwtd,owtd,ewtd[,timeout])

for sock in il:

#read the sock

### socket模块

Python 提供了两个基本的 socket 模块。

第一个是 Socket，它提供了标准的 BSD Sockets API。

第二个是 SocketServer， 它提供了服务器中心类，可以简化网络服务器的开发。

服务端

import socket,select

server=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

server.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1)

server.bind(('',10000))

server.listen(5)

inputs=[server]

while 1:

rs,ws,es=select.select(inputs,[],[],1)

for r in rs:

if r is server:

clientsock,clientaddr=r.accept()

inputs.append(clientsock)

else:

data=r.recv(1024)

if not data:

inputs.remove(r)

else:

print data

客户端

import socket

host='127.0.0.1'

port=10000

s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.connect((host,port))

s.send('hello from client')

s.close()

服务端一次发送的数据量很大很大,超出了 recv(指定的bufsize) 那么一次接收有问题,改成了循环接收方式

import socket,time

host = "127.0.0.1"

port = 9009

s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.connect((host,port))

s.send("SELECT \* FROM tt")

while True:

a = s.recv(1024)

if not len(a):break

print a

s.close()

### furl模块 url处理

Python 的furl模块是一个url操作模块

from furl import furl

f = furl('http://www.baidu.com/?bid=12331')

打印参数

print f.args

{'bid': '12331'}

增加参数

f.args['haha']='123'

print f.args

{'bid': '12331', 'haha': '123'}

修改参数

f.args['haha']='124'

print f.args

{'bid': '12331', 'haha': '124'}

删除参数

del f.args['haha']

print f.args

{'bid': '12331'}

3.或者使用内联方法

增加参数

furl('http://www.baidu.com/?bid=12331').add({'haha':'123'}).url

'http://www.baidu.com/?bid=12331&haha=123'

设置参数（只保留设置的参数）

furl('http://www.baidu.com/?bid=12331').set({'haha':'123'}).url

'http://www.baidu.com/?haha=123'

移除参数

furl('http://www.baidu.com/?bid=12331').remove(['bid']).url

'http://www.baidu.com/'

4.路径操作

获取路径

f=furl('http://www.baidu.com/hha/sh/?bid=12331')

f.path

Path('/hha/sh/')

设置路径

f.path='ss/ss/ss/ss'

print f.url

http://www.baidu.com/ss/ss/ss/ss?bid=12331

f.path='ss ss blank'

f.url

'http://www.baidu.com/ss%20ss%20blank?bid=12331'

也可以这样

f.set(host=u'ドメイン.テスト', path=u'джк', query=u'☃=☺')

f.url('http://xn--eckwd4c7c.xn--zckzah/%D0%B4%D0%B6%D0%BA??=?')

f.fragment.args={"one":'argument'}

f.url

'http://www.baidu.com#one=argument'

或者使用这种形式

f = furl('http://www.google.com/search?q=query#1')

f.copy().remove(path=True).set(host='taco.com').join('/pumps.html').add(fragment\_path='party').url

'http://taco.com/pumps.html#party'

## 有关命令行参数

Python 有两个内建的模块用于处理命令行参数：

一个是 getopt，《Deep in python》一书中也有提到，只能简单处理 命令行参数；

另一个是 optparse，它功能强大，而且易于使用，可以方便地生成标准的、符合Unix/Posix 规范的命令行说明。

<http://lingxiankong.github.io/blog/2014/01/14/command-line-parser/>

### sys.argv

最简单、最原始的方法就是手动解析了

import sys

def TestSys():

for arg in sys.argv[1:]:

print (arg)

TestSys()

cmd上输入 python loglogic/test.py gisy

输出gisy

### argparse模块

#### 基础

argparse模块使得编写用户友好的命令行接口非常容易。程序只需定义好它要求的参数，然后argparse将负责如何从sys.argv中解析出这些参数。argparse模块还会自动生成帮助和使用信息,并且当用户赋给程序非法的参数时产生错误信息。

其实ArgumentParser函数有很多可选参数，prog、usage、description、epilog分别定义解析器的名称、使用说明、描述、最后的结尾描述。

首先导入该模块；然后创建一个解析对象；然后向该对象中添加你要关注的命令行参数和选项，每一个add\_argument方法对应一个你要关注的参数或选项；最后调用parse\_args()方法进行解析；解析成功之后即可使用

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("square", help="display a square of a given number",

type=int)

args = parser.parse\_args() # parse\_args() 的返回值是一个命名空间，包含传递给命令的参数。

print args.**square**\*\*2 #用命令中的参数square

运行$ python prog.py 4 16

parser.add\_argument('src\_path'):添加必选参数src\_path 位置参数

必选参数，调用python脚本时，必须传进去。

parser.add\_argument('-f', '--foo') #选项参数 可选参数

#### 格式

# 可选参数前面多了--符号，-v是简写形式，参数名称square，store\_true说明碰到该参数时保存为true，否则就是false

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("-v", "--verbose",dest="square",help="increase output verbosity",

type=int, action="store\_true")

args=parser.parse\_args()

print args

#Namespace(a=False, b=None, c=23) #如果没赋值，默认为None,本次赋值的值只能在本次用，下次调用前全部清除。

args.**square 获取命令中参数值 args.a=** False

**type: 使用这个参数，转换输入参数的具体类型,这个参数可以关联到某个自定义的处理函数，**这种函数通常用来检查值的范围，以及合法性

parser.add\_argument('-f',type=file)

parser.add\_argument('-u',type=int)

type=float

**#choices: 这个参数用来检查输入参数的范围 #规定了可选参数的类型和取值范围**

parser.add\_argument("-v", "--verbosity", type=int, choices=[0, 1, 2],

help="increase output verbosity")

**# count表示遇到该参数几次，值就加几，默认值是0**

parser.add\_argument("-v", "--verbosity", action="count", default=0,

help="increase output verbosity")

**required:该选项必须出现 当某个选项指定需要在命令中出现的时候用这个参数**

parser.add\_argument('-u',required=True)

error: argument -a is required

**help: 使用这个参数描述选项作用**

parse\_args() 的返回值是一个命名空间，包含传递给命令的参数。该对象将参数保存其属性，因此如果你的参数 dest 是 "myoption"，那么你就可以args.myoption 来访问该值。

print parser.parse\_args(['-a', '-bval', '-c', '3'])

#### nargs: 当选项后接受多个或者0个参数时需要这个来指定

nargs: 当选项后接受多个或者0个参数时需要这个来指定

比如-u选项接受2个参数

parser.add\_argument('-u',**nargs=2**)

parser.parse\_args('-u a b'.split()) #加入输入的指令是 -u a b

#Namespace(**u=['a', 'b']**)

当选项接受1个或者不需要参数时指定nargs='?',当没有参数时，会从default中取值。**对于选项参数有一个额外的情况，就是出现选项而后面没有跟具体参数，那么会从const中取值。**

parser.add\_argument('-u',nargs='?',default='d')

parser.add\_argument('A',nargs='?',default='e')

parser.parse\_args('-u'.split())

#Namespace(A='e', u=None)

parser.parse\_args(''.split())

#Namespace(A='e', u='d')

parser.add\_argument('-u',nargs='?',default='d',const='s')

parser.parse\_args('-u'.split())

#Namespace(u='s')

而对于后面需要跟多个参数的情况（–foo a1 a2 a3…),则需要设置nargs=’\*’

nargs=’+'也和nargs=’\*'一样，但是有一个区别当’+'时少于1个参数（没有参数）位置参数会报错误

#### action参数动作

argparse内置6种动作可以在解析到一个参数时进行触发：

**store 保存参数值，可能会先将参数值转换成另一个数据类型。若没有显式指定动作，则默认为该动作。**

使用store**，则调用时必须跟随参数值**

parser.add\_argument('-c', action="store", dest="c", type=int) python gisy.py -c 4

**store\_true/store\_false 保存相应的布尔值。**这两个动作被用于实现布尔开关。使用**store\_true，则调用时不能跟随参数值**

parser.add\_argument('-t', action='**store\_true**',default=False,dest='b')

调用到这个-t，就会是true,否则为false python argparse\_action.py -t #parser.boolean\_switch True

**append 将值保存到一个列表中。若参数重复出现，则保存多个值。**

parser.add\_argument('-a', action='append',dest='collection',default=[],help='Add repeated values to a list')

python loglogic/test.py -a one -a two -a three # parser.collection ['one', 'two', 'three']

**store\_const 保存一个被定义为参数规格一部分的值，而不是一个来自参数解析而来的值。**这通常用于实现非布尔值的命令行标记。

**append\_const 将一个定义在参数规格中的值保存到一个列表中。**

**version 打印关于程序的版本信息，然后退出**

parser.add\_argument('--version', action='version', version='%(prog)s 1.0')

### getopt模块 比如-i -o

python中使用命令行选项：

例如我们需要一个convert.py脚本。它的作用是处理一个文件,并将处理后的结果输出到另一个文件中。

要求该脚本满足以下条件：

1.通过-i -o选项来区别参数是输入文件还是输出文件.

>>> python convert.py -i inputfile -o outputfile

2.当不知道convert.py需要哪些参数时，用-h打印出帮助信息

>>> python convert.py -h

getopt函数原形:

getopt.getopt(args, options[, long\_options])

convert.py

import sys, getopt

opts, args = getopt.getopt(sys.argv[1:], "hi:o:")

input\_file=""

output\_file=""

for op, value in opts:

if op == "-i":

input\_file = value

elif op == "-o":

output\_file = value

elif op == "-h":

usage()

sys.exit()

代码解释：

a) sys.argv[1:]为要处理的参数列表，sys.argv[0]为脚本名，所以用sys.argv[1:]过滤掉脚本名。

b) "hi:o:": 当一个选项只是表示开关状态时，即后面不带附加参数时，在分析串中写入选项字符。当选项后面是带一个附加参数时，在分析串中写入选项字符同时后面加一个":"号。所以"hi:o:"就表示"h"是一个开关选项；"i:"和"o:"则表示后面应该带一个参数。

c) 调用getopt函数。函数返回两个列表：opts和args。opts为分析出的格式信息。args为不属于格式信息的剩余的命令行参数。opts是一个两元组的列表。每个元素为：(选项串,附加参数)。如果没有附加参数则为空串''。

getopt函数的第三个参数[, long\_options]为可选的长选项参数，上面例子中的都为短选项(如-i -o)

长选项格式举例:

--version

--file=error.txt

**让一个脚本同时支持短选项和长选项**

getopt.getopt(sys.argv[1:], "hi:o:", ["version", "file="])

程序上：

elif op=='--version':

print 'ok'

### docopt模块

它是根据你自己写的help messages（文档描述），自动为你生成parser。使用之前需要下载相应的库

# naval\_fate.py

"""Naval Fate.

Usage:

naval\_fate.py ship new <name>...

naval\_fate.py ship <name> move <x> <y> [--speed=<kn>]

naval\_fate.py ship shoot <x> <y>

naval\_fate.py mine (set|remove) <x> <y> [--moored | --drifting]

naval\_fate.py (-h | --help)

naval\_fate.py --version

Options:

-h --help Show this screen.

--version Show version.

--speed=<kn> Speed in knots [default: 10].

--moored Moored (anchored) mine.

--drifting Drifting mine.

"""

from docopt import docopt

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

arguments = docopt(\_\_doc\_\_, version='Naval Fate 2.0')

print(arguments)

Naval Fate是app名称，naval\_fate是命令行命令，ship, new, move这些是可选的执行命令(commands), , , 这些是位置参数(positional arguments)，-h, --help, --speed=这些是选项参数(options)，

例子里面用

"[]"描述可选元素(optional)

"()"描述必要元素(required)

"|" 描述互斥元素(mutually exclusive)

"..."描述重复元素(repeating)

这些参数，前面加上naval\_fate就形成了可用的命令，这些命令在Usage中介绍。

Usage下面的Options里罗列了选项(options)及其描述，它具体描述了

选项是否有长/短形式，如-h, --help

选项后面是否带参数，如--speed=

选项是否有默认值，如[default: 10]

Usage和options里的内容就组成了帮助信息，当用户输入-h或--help参数时，命令行就会输出帮助信息。

docopt会抽取帮助信息里的内容，然后对命令行传入的参数进行解析。

在python脚本中：

首先从docopt模块中调用docopt函数

from docopt import docopt

之后用docopt函数解析参数

docopt(doc, argv=None, help=True, version=None, options\_first=False)

docopt需要1个必要的位置参数和4个可选的关键字参数

doc可以是模块的docstring(doc)或其他包含帮助信息的字符串。它被用于当作参数解析的规则。

argv是一个参数列表，默认值为sys.argv[1:]，你也可以提供其他列表如['--verbose', '-o', 'hai.txt']。

help选择当遇到-h和--help参数时是否输出帮助信息并退出程序，默认值为True。

version描述程序版本，默认值为None，如果给version传入了值，那么在解析--version选项时，程序会输出版本信息并退出。

options\_first默认值为False。如果它设为True的话会禁止混合使用选项(options)和位置参数(positional argument)。例如，在第一个位置参数后面，其他参数都被当作位置参数，而不是选项。

docopt()的返回值是一个字典

所有选项、位置参数和命令的名称作为键(key)

传入的参数作为值(value)

可以看出，

docopt(\_\_doc\_\_)这个函数根据帮助文档的说明解析命令行参数，然后将结果返回为一个字典。

当用户使用不在Usage之内的命令，输出帮助文档

$ python naval\_fate.py ship shoot 42 36

得到输出

{'--drifting': False,

'--help': False,

'--moored': False,

'--speed': '10',

'--version': False,

'<name>': [],

'<x>': '**42**',

'<y>': '**36**',

'mine': False,

'move': False,

'new': False,

'remove': False,

'set': False,

'ship': True,

'shoot': True}

### optparse模块

optaprse自2.7版开始弃用:弃用optparse模块,不会进一步开发,将继续开发argparse模块作为替代。

from optparse import OptionParser

parser = OptionParser()

parser.add\_option("-f", "--file", dest="filename",

help="write report to FILE", metavar="FILE")

parser.add\_option("-q", "--quiet",

action="store\_false", dest="verbose", default=True,

help="don't print status messages to stdout")

(options, args) = parser.parse\_args()

print options.verbose

print args

parse\_args() 方法提供了一个 default 参数用于设置默认值。

action 是 parse\_args() 方法的参数之一，它指示 optparse 当解析到一个命令行参数时该如何处理。actions 有一组固定的值可供选择，默认是’store ‘，表示将命令行参数值保存在 options 对象里。

如果没有指定 dest 参数，将用命令行的参数名来对 options 对象的值进行存取。

add\_option方法使用

parser.add\_option([short\_option\_str,long\_option\_str],attr\_option\_pair,...)

short\_option\_str和long\_option\_str，至少要有一个，可以有多个

attr\_option\_pair就是一组键值对attr=value, 有几个常用的，

action 读取对应option的value后怎么处理

dest 用哪个key引用(dict)或者理解为存到哪个变量中

nargs 表示该option有几个值，默认是1，则对饮的dest存的就是tuple

type 读取值得类型

choices 给定该option的值得选择范围

metavar 指定，option value的占位符

default 默认值

help 解释信息

## 有关优化

你应该用更简单的工具来对测试用例进行根本的检查，并且用更慢但能显示更多细节的line\_profiler来深入到函数内部。

如果你仍然觉得这太慢，而是用一些你自己的秘密武器，如比较属性访问技术或调整平衡检查技术。你也可以用如下的方法：

1．忍受缓慢或者缓存它们

2．重新思考整个实现

3．更多使用优化的数据结构

4．写一个C扩展

### 简单计时器 装饰器方式

计时器很简单，这是一个最灵活的记录执行时间的方法。

在函数前面加上装饰器@timefunc

import time

def timefunc(f):

def f\_timer(\*args, \*\*kwargs):

start = time.time()

result = f(\*args, \*\*kwargs)

end = time.time()

print f.\_\_name\_\_, 'took', end - start, 'time'

return result

return f\_timer

def get\_number():

for x in xrange(5000000):

yield x

@timefunc

def expensive\_function():

for x in get\_number():

i = x ^ x ^ x

return 'some result!'

result = expensive\_function()

# expensive\_function took 0.47200012207 time

### 使用上下文管理器进行及时

class **Timer**(object):

def **\_\_init\_\_**(*self*, verbose=True):

*self*.verbose = verbose

def **\_\_enter\_\_**(*self*):

*self*.start = time.time()

return *self*

def **\_\_exit\_\_**(*self*, \*args):

*self*.end = time.time()

*self*.secs = *self*.end - *self*.start

*self*.msecs = *self*.secs \* 1000

if *self*.verbose:

print *'elapsed time: %f ms'* % *self*.msecs

with Timer() as t:

result = expensive\_function()

需要用Python的with关键字和Timer上下文管理器包装想要计时的代码块。它将会在你的代码块开始执行的时候启动计时器，在你的代码块结束的时候停止计时器。

### cProfile模块

#### 基础

import cProfile

# 直接把分析结果打印到控制台

cProfile.run("test()")

# 把分析结果保存到文件中

cProfile.run("test()", filename="result.out")

# 增加排序方式

cProfile.run("test()", filename="result.out", sort="cumulative")

使用命令行运行的方法基本一致，Bash代码如下:

# 直接把分析结果打印到控制台

python -m cProfile test.py

# 把分析结果保存到文件中

python -m cProfile -o result.out test.py

# 增加排序方式

python -m cProfile -o result.out -s cumulative test.py

#### pstats 分析工具 显示内容

使用cProfile分析的结果可以输出到指定的文件中，但是文件内容是以二进制的方式保存的，用文本编辑器打开时乱码。所以，Python提供了一个pstats模块，用来分析cProfile输出的文件内容。它支持多种形式的报表输出

# 创建Stats对象

p = pstats.Stats("result.out")

# strip\_dirs(): 去掉无关的路径信息

# sort\_stats(): 排序，支持的方式和上述的一致

# print\_stats(): 打印分析结果，可以指定打印前几行

# 和直接运行cProfile.run("test()")的结果是一样的

p.strip\_dirs().sort\_stats(-1).print\_stats()

# 按照函数名排序，只打印前3行函数的信息, 参数还可为小数,表示前百分之几的函数信息

p.strip\_dirs().sort\_stats("name").print\_stats(3)

# 按照运行时间和函数名进行排序

p.strip\_dirs().sort\_stats("cumulative", "name").print\_stats(0.5)

# 如果想知道有哪些函数调用了sum\_num

p.print\_callers(0.5, "sum\_num")

# 查看test()函数中调用了哪些函数

p.print\_callees("test")

#### 装饰器方式

import cProfile

def do\_cprofile(func):

def profiled\_func(\*args, \*\*kwargs):

profile = cProfile.Profile()

try:

profile.enable()

result = func(\*args, \*\*kwargs)

profile.disable()

return result

finally:

profile.print\_stats()

return profiled\_func

def get\_number():

for x in xrange(5000000):

yield x

@do\_cprofile

def expensive\_function():

for x in get\_number():

i = x ^ x ^ x

return 'some result!'

# perform profiling

result = expensive\_function()

#### 结果

5000003 function calls in 0.916 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

5000001 0.292 0.000 0.292 0.000 test.py:15(get\_number)

1 0.624 0.624 0.916 0.916 test.py:19(expensive\_function)

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

输出如上图，主要有：

ncalls: 函数被call的次数

tottime：函数总的耗时，但是不包括其子函数的耗时

percall：tottime平均到每次调用的耗时

cumtime：函数总的耗时，包括了其子函数的耗时（递归函数也不例外）

percall：cumtime平均到每次调用的耗时

filename:lineno(function) ：每个函数各自的信息

### LineProfiler模块

在这个例子里，我们会用非常棒的line\_profiler库。为了容易使用，我们会再次用装饰器包装一下，这种简单的方法也可以防止把它放在生产代码里。

try:

from line\_profiler import LineProfiler

def do\_profile(follow=[]):

def inner(func):

def profiled\_func(\*args, \*\*kwargs):

try:

profiler = LineProfiler()

profiler.add\_function(func)

for f in follow:

profiler.add\_function(f)

profiler.enable\_by\_count()

return func(\*args, \*\*kwargs)

finally:

profiler.print\_stats()

return profiled\_func

return inner

except ImportError:

def do\_profile(follow=[]):

"Helpful if you accidentally leave in production!"

def inner(func):

def nothing(\*args, \*\*kwargs):

return func(\*args, \*\*kwargs)

return nothing

return inner

def get\_number():

for x in xrange(5000000):

yield x

@do\_profile(follow=[get\_number])

def expensive\_function():

for x in get\_number():

i = x ^ x ^ x

return 'some result!'

result = expensive\_function()

### memory\_profiler模块 计算内存

这里建议同时安装psutil模块，因为它可以大大改善memory\_profiler的性能

在感兴趣的函数上面装饰@profile装饰器：

@profile

def primes(n):

….

想要观察你的函数使用了多少内存，像下面这样执行:

python -m memory\_profiler primes.py

### timeit模块 测量小代码片段的执行时间

import timeit

#执行命令

t2 = timeit.Timer('x=range(1000)')

#显示时间

t2.timeit()

10.620039563513103

timeit.timeit('"-".join(str(n) for n in range(100))', number=10000)

0.8187260627746582

### traceback模块 输出具体的异常结果

try:

1/0

except Exception,e:

print e

输出结果是integer division or modulo by zero，只知道是报了这个错，但是却不知道在哪个文件哪个函数哪一行报的错。

下面使用traceback模块

import traceback

try:

1/0

except Exception,e:

traceback.print\_exc()

输出结果是

Traceback (most recent call last):

File "test\_traceback.py", line 3, in <module>

1/0

ZeroDivisionError: integer division or modulo by zero

这样非常直观有利于调试。

traceback.print\_exc()跟traceback.format\_exc()有什么区别呢？

format\_exc()返回字符串，print\_exc()则直接给打印出来。

即traceback.print\_exc()与print traceback.format\_exc()效果是一样的。

print\_exc()还可以接受file参数，直接写入到一个文件。比如

traceback.print\_exc(file=open('tb.txt','w+')) 写入到tb.txt文件去。

### pdb模块

它实际上是一个交互式源代码调试器。

你需要下面的两行代码来使用此模块。

import pdb

pdb.set\_trace() #添加断点，程序运行到该行暂停

import pdb

import sys

def add(num1=0, num2=0):

return int(num1) + int(num2)

def sub(num1=0, num2=0):

return int(num1) - int(num2)

def main():

#Assuming our inputs are valid numbers

print sys.argv

pdb.set\_trace() # <--断点添加在这

addition = add(sys.argv[1], sys.argv[2])

print addition

subtraction = sub(sys.argv[1], sys.argv[2])

print subtraction

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

main()

操作

在你的调试器提示中，输入n运行到下一行 n

打印->p (Pdb) p sys.argv

我们可以使用“s”进入下一行程序所调用的函数 s

你可以通过单击c使PDB跳到末尾或者直到下一个断点，因为程序中我们并没有其他的断点了，所有程序将会执行完成。 c

动态添加断点- > b b 18 在第18行加入断点

使用“l”将会显示出此刻你在代码中的位置。 L的小写

在代码的任何地方如果你想结束调试，可以使用“q”，那么正在执行的程序将会终止

动态分配变量 比如修改 addition变量 addition = &

### trace模块

我们可以使用–trace选项来查看程序运行时哪条语句正在被执行.在cmd中输入命令启动py文件

$ python -m trace --trace trace\_example/main.py

输出结构的第一部分表明了trace的一个安装操作. 剩下来的输出显示了每个函数的入口信息, 包括函数位于哪个模块, 然后是原脚本文件中的语句行

trace还可以收集函数间调用关系. 使用–listfuncs可以在结果中输出简单的函数调用关系:

$ python -m trace --listfuncs trace\_example/main.py

从命令行中运行trace并使用–count选项可以产生代码信息报告, 因此可以看到哪些行是被执行的, 哪些被跳过了. 因为你的程序通常是多个文件组成, 那就会为每个文件产生独立的报表. 默认下, 报表文件在和模块的同一目录下被创建, 并以模块名命名,后缀名为.cover

$ python -m trace --count trace\_example/main.py

可以使用–trackcalls获得更多信息, 比如说谁调用了函数.

$ python -m trace --listfuncs --trackcalls trace\_example/main.py

## 有关日志

### logging模块

#### console /file同时记录日志

logging库提供了多个组件：Logger、Handler、Filter、Formatter。Logger对象提供应用程序可直接使用的接口，Handler发送日志到适当的目的地，Filter提供了过滤日志信息的方法，Formatter指定日志显示格式。

NOTSET < DEBUG < INFO < WARNING < ERROR < CRITICAL

一个模块级别的函数是logging.getLogger([name])（返回一个logger对象，如果没有指定名字将返回root logger）

# 创建一个logger

logger1=logging.getLogger('mylogger')

logger1.setLevel(logging.DEBUG)

# 创建一个handler，用于写入日志文件

fh=logging.FileHandler('test.log'**,encoding='UTF-8'**)

# 再创建一个handler，用于输出到控制台

ch=logging.StreamHandler()

# 定义handler的输出格式formatter

formatter=logging.Formatter('%(asctime)s-%(name)s-%(levelname)s-%(message)s')

fh.setFormatter(formatter)

ch.setFormatter(formatter)

#定义一个filter

filter=logging.Filter('mylogger.child1.child2')

#fh.addFilter(filter)

**#给logger添加handler**

logger1.addFilter(filter) #添加过滤器

logger1.addHandler(fh)

logger1.addHandler(ch)

# 记录一条日志

logger1.debug('logger1 debug message')

logger1.info('logger1 info message')

logger1.warning('logger1 warning message')

logger1.error('logger1 error message')

logger1.critical('logger1 critical message')

Filter

限制只有满足过滤规则的日志才会输出。

比如我们定义了filter = logging.Filter('a.b.c'),并将这个Filter添加到了一个Handler上，则使用该Handler的Logger中只有名字带a.b.c前缀的Logger才能输出其日志。

#### 模板

logger=logging.getLogger() #设置实例

logger.setLevel(logging.DEBUG) #设置最低级别

fileLog = logging.FileHandler('server.log', encoding='UTF-8') #设置文件handler

consoleLog = logging.StreamHandler() #设置控制台handler

logging\_format = logging.Formatter(

'%(asctime)s %(levelname)s %(filename)s (%(funcName)s:%(lineno)s) - %(message)s' ) #设置输出文本格式

fileLog.setFormatter(logging\_format) #设置文件handler的文本格式

consoleLog.setFormatter(logging\_format) #设置控制台handler的文本格式

logger.addHandler(fileLog) #将设置好的handler添加到实例中

logger.addHandler(consoleLog) #将设置好的handler添加到实例中

#添加一行日志

logger.info('logger1 info message') #在程序中添加日志，将该行日志设置为info级别

logger.info(str(urlSt)+str(urlUd)) #括号内如果两个变量相加，必须先转换为字符串形式

logger.info('好了 %s' %str(a))

logger.info(urlSt) #如果只有单个遍历，无需转换为str; list等都可直接显示；但是多个必须转换str

默认情况下python的logging模块将日志打印到了标准输出中，且只显示了大于等于WARNING级别的日志，这说明默认的日志级别设置为WARNING（日志级别等级CRITICAL > ERROR > WARNING > INFO > DEBUG > NOTSET），默认的日志格式为日志级别：Logger名称：用户输出消息。

#### 细节

Logger.setLevel(lel):指定最低的日志级别，低于lel的级别将被忽略。debug是最低的内置级别，critical为最高

Logger.addFilter(filt)、Logger.removeFilter(filt):添加或删除指定的filter

Logger.addHandler(hdlr)、Logger.removeHandler(hdlr)：增加或删除指定的handler

通过handler对象可以把日志内容写到不同的地方。比如简单的StreamHandler就是把日志写到类似文件的地方。python提供了十几种实用handler，比较常用有：

StreamHandler: 输出到控制台

FileHandler: 输出到文件

Logger.debug()、Logger.info()、Logger.warning()、Logger.error()、Logger.critical()：可以设置的日志级别

logger = logging.getLogger()返回一个默认的Logger也即root Logger，并应用默认的日志级别、Handler和Formatter设置。

只要logging.getLogger（name）中名称参数name相同则返回的Logger实例就是同一个，且仅有一个，**也即name与Logger实例一一对应。**

运行时报错：AttributeError: 'module' object has no attribute 'handlers'

原来导入logging模块后并没有自动导入其子模块handlers

import logging.handlers

而logging是一个拥有子模块的包，这些子模块没有被自动载入。所以在访问签需要明确的导入logging.handlers子模块。

#### RotatingFileHandler、TimedRotatingFileHandler自动备份历史日志并删除

有多中可用的Handler：

logging.StreamHandler 可以向类似与sys.stdout或者sys.stderr的任何文件对象(file object)输出信息

logging.FileHandler 用于向一个文件输出日志信息

**logging.handlers.RotatingFileHandler**

类似于上面的FileHandler，但是它可以管理文件大小。当文件达到一定大小之后，它会自动将当前日志文件改名，然后创建一个新的同名日志文件继续输出

RotatingFileHandler( filename[, mode[, maxBytes[, backupCount]]])

(filename, mode='a', maxBytes=0, backupCount=0, encoding=None, delay=0)

filelog=logging.handlers.RotatingFileHandler('server.log', encoding='UTF-8',maxBytes=480, backupCount=2)

其中filename和mode两个参数和FileHandler一样。

maxBytes用于指定日志文件的最大文件大小。如果maxBytes为0，意味着日志文件可以无限大，这时上面描述的重命名过程就不会发生。maxBytes为字节 1kb=1024字节

backupCount用于指定保留的备份文件的个数。比如，如果指定为2，也就是备份文件最多为2个，当上面描述的重命名过程发生时，原有的chat.log.2并不会被更名，而是被删除。

mode是文件的打开方式。默认是’a'，即添加到文件末尾。

**logging.handlers.TimedRotatingFileHandler**

和RotatingFileHandler类似，不过，它没有通过判断文件大小来决定何时重新创建日志文件，而是间隔一定时间就自动创建新的日志文件

TimedRotatingFileHandler( filename [,when [,interval [,backupCount]]])

其中filename参数和backupCount参数和RotatingFileHandler具有相同的意义。

interval是时间间隔。

when参数是一个字符串。表示时间间隔的单位，不区分大小写。它有以下取值：

S 秒 M 分 H小时 D 天 W 每星期（interval==0时代表星期一） midnight 每天凌晨

import logging

import logging.handlers

**fileLog = logging.handlers.TimedRotatingFileHandler('server.log', encoding='UTF-8',when='s',interval=10,backupCount=3)**

#### Formatter

#定义Formatter

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

#为Handler添加Formatter

fh.setFormatter(formatter)

ch.setFormatter(formatter)

%(name)s Logger的名字

%(levelno)s 数字形式的日志级别

%(levelname)s 文本形式的日志级别

%(pathname)s 调用日志输出函数的模块的完整路径名，可能没有

%(filename)s 调用日志输出函数的模块的文件名

%(module)s 调用日志输出函数的模块名

%(funcName)s 调用日志输出函数的函数名

%(lineno)d 调用日志输出函数的语句所在的代码行

%(created)f 当前时间，用UNIX标准的表示时间的浮 点数表示

%(relativeCreated)d 输出日志信息时的，自Logger创建以 来的毫秒数

%(asctime)s 字符串形式的当前时间。默认格式是 “2003-07-08 16:49:45,896”。逗号后面的是毫秒

%(thread)d 线程ID。可能没有

%(threadName)s 线程名。可能没有

%(process)d 进程ID。可能没有

**%(message)s用户输出的消息**

#### basicConfig模式

import logging

logging.basicConfig(level=logging.DEBUG,

format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',

datefmt='%Y-%m-%d %H:%M:%S',

filename='test.log',

filemode='a')

#添加日志

logging.debug('debug message')

logging.info('info message')

logging.warning('warning message')

logging.error('error message')

logging.critical('critical message')

可见在logging.basicConfig()函数中可通过具体参数来更改logging模块默认行为，可用参数有

filename：用指定的文件名创建FiledHandler（后边会具体讲解handler的概念），这样日志会被存储在指定的文件中。

filemode：文件打开方式，在指定了filename时使用这个参数，默认值为“a”还可指定为“w”。

format：指定handler使用的日志显示格式。

datefmt：指定日期时间格式。

level：设置rootlogger（后边会讲解具体概念）的日志级别

stream：用指定的stream创建StreamHandler。可以指定输出到sys.stderr,sys.stdout或者文件，默认为sys.stderr。若同时列出了filename和stream两个参数，则stream参数会被忽略。

### logbook模块 – Logging 库的替代品

它设计支持复杂和简单的应用，目的是让记录日志变得有趣：

from logbook import Logger

log = Logger('A Fancy Name')

log.warn('Logbook is too awesome for most applications')

log.error("Can't touch this")

## 有关数据挖掘

1. matplotlib， 图形化 2. pandas，数据挖掘的关键， 提供各种挖掘分析的算法 3. numpy， 提供基本的统计 scipy， 提供各种数学公式

numpy 准确地说提供了一个在python中做科学计算的基础库，侠义地讲它重在数值计算，甚至可以说是用于多维数组处理的库；而 scipy 则是基于numpy，提供了一个在python中做科学计算的工具集，也就是说它是更上一个层次的库，主要包含一下模块：

statistics

optimization

numerical integration

linear algebra

Fourier transforms

signal processing

image processing

ODE solvers

special functions

### 机器学习

机器学习模块： 一般都需要先安装numpy和SciPy

Crab – 灵活、快速的推荐引擎。

gensim – 人性化的话题建模库。

hebel – GPU 加速的深度学习库。

NuPIC – 智能计算 Numenta 平台。

pattern – Python 网络挖掘模块。

PyBrain – 另一个 Python 机器学习库。

Pylearn2 – 一个基于 Theano 的机器学习库。

python-recsys – 一个用来实现推荐系统的 Python 库。

scikit-learn – 基于 SciPy 构建的机器学习 Python 模块。

pydeep – Python 深度学习库。

vowpalporpoise – 轻量级 Vowpal Wabbit 的 Python 封装。

skflow – 一个 TensorFlow 的简化接口(模仿 scikit-learn)。

Python 有很多为机器学习提供便利的开源库。通常它们被称为 Python 科学库（scientific Python libraries），用以执行基本的数据科学任务：

▪ numpy – 主要用于 N 维数组

▪ pandas – Python 数据分析库，包含 dataframe 等结构

▪ matplotlib – 2D 绘图库，产出质量足以进行印刷的图

▪ scikit-learn – 数据分析、数据挖掘任务使用的机器学习算法

学习以上这些内容可以使用：

▪ Scipy Lecture Notes 作者 Gaël Varoquaux， Emmanuelle Gouillart， Olav Vahtras

下面这个 pandas 教程也不错，贴近主题：

▪ 10 Minutes to Pandas

http://pandas.pydata.org/pandas-docs/stable/10min.html

在后面的教程中你会看到其他一些 packages ，比如包括 Seaborn ，一个基于 matplotlib 的可视化库。前面提到的 packages （再次承认具有一定主观色彩）是许多 Python 机器学习任务的核心工具。不过，理解它们也可以让你在之后的教程中更好理解其他相关 packages。

<http://www.numpy.org/>

<http://pandas.pydata.org/>

<http://scikit-learn.org/stable/>

### 算法介绍

记忆化是一种提高程序运行速度的优化技术。通过储存大计算量函数的返回值，当这个结果再次被需要时将其从缓存提取，而不用再次计算来节省计算时间。 记忆化是一种典型的时间存储平衡方案。

动态规划（英语：Dynamic programming，简称DP）是一种在数学、管理科学、计算机科学、经济学和生物信息学中使用的，通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。

动态规划常常适用于有重叠子问题[1]和最优子结构性质的问题，动态规划方法所耗时间往往远少于朴素解法。

动态规划背后的基本思想非常简单。大致上，若要解一个给定问题，我们需要解其不同部分（即子问题），再合并子问题的解以得出原问题的解。

通常许多子问题非常相似，为此动态规划法试图仅仅解决每个子问题一次，从而减少计算量：一旦某个给定子问题的解已经算出，则将其记忆化（en:memoization）存储，以便下次需要同一个子问题解之时直接查表。这种做法在重复子问题的数目关于输入的规模呈指数增长时特别有用。

常见的数据结构

数组（Array）

堆栈（Stack）

队列（Queue）

链表（Linked List）

树（Tree）

图（Graph）

堆（Heap）

散列表（Hash）

堆栈（英语：stack），也可直接称栈（港澳台作堆叠），在计算机科学中，是一种特殊的串列形式的数据结构，它的特殊之处在于只能允许在链接串列或阵列的一端（称为堆叠顶端指标，英语：top）进行加入资料（英语：push）和输出资料（英语：pop）的运算。由于堆叠数据结构只允许在一端进行操作，因而按照后进先出（LIFO, Last In First Out）的原理运作。

树（英语：tree）是一种抽象资料型别（ADT）或是实作这种抽象资料型别的数据结构，用来模拟具树状结构性质的资料集合。它是由n（n>=1）个有限节点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下的特点：

每个节点有零个或多个子节点；

没有父节点的节点称为根节点；

每一个非根节点有且只有一个父节点；

除了根节点外，每个子节点可以分为多个不相交的子树；

在数学上，一个图（Graph）是表示物件与物件之间的关系的方法，是图论的基本研究对象。一个图看起来是由一些小圆点（称为顶点或结点）和连结这些圆点的直线或曲线（称为边）组成的。

散列表（Hash table，也叫哈希表），是根据键（Key）而直接访问在内存存储位置的数据结构。也就是说，它通过计算一个关于键值的函数，将所需查询的数据映射到表中一个位置来访问记录，这加快了查找速度。这个映射函数称做散列函数，存放记录的数组称做散列表。

一个通俗的例子是，为了查找电话簿中某人的号码，可以创建一个按照人名首字母顺序排列的表（即建立人名 {\displaystyle x} x到首字母 {\displaystyle F(x)} F(x)的一个函数关系），在首字母为W的表中查找“王”姓的电话号码，显然比直接查找就要快得多。这里使用人名作为关键字，“取首字母”是这个例子中散列函数的函数法则 {\displaystyle F()} F()，存放首字母的表对应散列表。关键字和函数法则理论上可以任意确定。

### 图graph

图的遍历方法有深度优先搜索法和广度（宽度）优先搜索法。

深度优先搜索法是树的先根遍历的推广，它的基本思想是：从图G的某个顶点v0出发，访问v0，然后选择一个与v0相邻且没被访问过的顶点vi访问，再从vi出发选择一个与vi相邻且未被访问的顶点vj进行访问，依次继续。如果当前被访问过的顶点的所有邻接顶点都已被访问，则退回到已被访问的顶点序列中最后一个拥有未被访问的相邻顶点的顶点w，从w出发按同样的方法向前遍历，直到图中所有顶点都被访问。

广度优先搜索是树的按层次遍历的推广，它的基本思想是：首先访问初始点vi，并将其标记为已访问过，接着访问vi的所有未被访问过的邻接点vi1,vi2,…, vi t，并均标记已访问过，然后再按照vi1,vi2,…, vi t的次序，访问每一个顶点的所有未被访问过的邻接点，并均标记为已访问过，依次类推，直到图中所有和初始点vi有路径相通的顶点都被访问过为止。

图的存储表示

数组（邻接矩阵）存储表示（有向或无向）

邻接表存储表示

前向星存储表示

有向图的十字链表存储表示

无向图的邻接多重表存储表示

邻接矩阵是表示一个图的常用存储表示。它用两个数组分别存储数据元素（顶点）的信息和数据元素之间的关系（边或弧）的信息。

距离矩阵可算是邻接矩阵的扩充。

如果是无向图，那么每条边由两个结点组成，分别代表边的两个端点；如果是有向图，那么每条边是一个结点对，分别代表边的始点和终点。一般来说，邻接表是无向的。邻接表结构的困难之一是无法明确在什么地方保存相关边的长度或花销。

十字链表

对于有向图来说，邻接表是有缺陷的。关心了出度问题，想了解入度就必须要遍历整个图才知道，反之，逆邻接表解决了入度却不了解出度情况。十字链表（Orthogonal List）是有向图的一种存储方法，它实际上是邻接表与逆邻接表的结合，即把每一条边的边结点分别组织到以弧尾顶点为头结点的链表和以弧头顶点为头顶点的链表中。

查找最短路径

graph = {'A': ['B', 'C'],

'B': ['C', 'D'],

'C': ['D'],

'D': ['C'],

'E': ['F'],

'F': ['C']}

def find\_shortest\_path(graph, start, end, path=[]):

path = path + [start]

if start == end:

return path

if not graph.has\_key(start):

return None

shortest = None

for node in graph[start]:

if node not in path:

newpath = find\_shortest\_path(graph, node, end, path)

if newpath:

if not shortest or len(newpath) < len(shortest):

shortest = newpath

else:

return start #如果找不到路径，返回卡在哪个节点

return shortest

print find\_shortest\_path(graph,'A','B')

Python的图邻接矩阵法做为存储结构，0表示没有边，1表示有边。

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

class Graph:

def \_\_init\_\_(self,maps=[],nodenum=0,edgenum=0):

self.map=maps

self.nodenum=len(maps)

self.edgenum=edgenum

def isOutRange(self,x):

try:

if x>=self.nodenum or x<=0:

raise IndexError

except IndexError:

print('节点下标出界')

def GetNodenum(self):

self.nodenum=len(self.map)

return self.nodenum

def GetEdgenum(self):

GetNodenum()

self.edgenum=0

for i in range(self.nodenum):

for j in range(self.nodenum):

if self.map[i][j] is 1:

self.edgenum=self.edgenum+1

return self.edgenum

def InsertNode(self):

for i in range(self.nodenum):

self.map[i].append(0)

self.nodenum=self.nodenum+1

ls=[0]\*self.nodenum

self.map.append(ls)

def DeleteNode(self,x):

for i in range(self.nodenum):

if self.map[i][x] is 1:

self.map[i][x]=0

self.edgenum=self.edgenum-1

if self.map[x][i] is 1:

self.map[x][i]=0

self.edgenum=self.edgenum-1

def AddEdge(self,x,y):

if self.map[x][y] is 0:

self.map[x][y]=1

self.edgenum=self.edgenum+1

def RemoveEdge(self,x,y):

if self.map[x][y] is 0:

self.map[x][y]=1

self.edgenum=self.edgenum+1

def BreadthFirstSearch(self):

def BFS(self,i):

print(i)

visited[i]=1

for k in range(self.nodenum):

if self.map[i][k]==1 and visited[k]==0:

BFS(self,k)

visited=[0]\*self.nodenum

for i in range(self.nodenum):

if visited[i] is 0:

BFS(self,i)

def DepthFirstSearch(self):

def DFS(self,i,queue):

queue.append(i)

print(i)

visited[i]=1

if len(queue)!=0:

w=queue.pop()

for k in range(self.nodenum):

if self.map[w][k] is 1 and visited[k] is 0:

DFS(self,k,queue)

visited=[0]\*self.nodenum

queue=[]

for i in range(self.nodenum):

if visited[i] is 0:

DFS(self,i ,queue)

def DoTest():

maps=[

[-1, 1, 0, 0],

[0, -1, 0, 0],

[0, 0, -1, 1],

[1, 0, 0, -1]

]

G=Graph(maps)

G.InsertNode()

G.AddEdge(1,4)

print("广度优先遍历")

G.BreadthFirstSearch()

print("深度优先遍历")

G.DepthFirstSearch()

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

DoTest()

### scipy

阶乘

import math

math.factorial(6) #720

import scipy.misc

scipy.misc.factorial(6) #array(720.0)

## 有关配置文件

### ConfigParser模块 配置文件

configParser 模块用于操作配置文件.

ini配置文件的格式:

可以包含一个或多个节（section），每个节可以有多个参数（键=值），即每个section由若干个键(key)组成。每个key可以赋相应的值。可以用冒号，也可以用等号。注释用#号

比如：

[book]

title:ConfigParser模块教程

time:2012-09-20 22:04:55

[size]

size:1024

gisy:wkb

##================EPT-VMS====================

[com.jd.vms.service.EptVendorService]

ip=192.168.156.64

port = 22000

alias=ept-vms

protocol = jsf

config=ConfigParser.ConfigParser() 创建ConfigParser实例

config.read(filename) 读取配置文件

比如 gisy\_path='conf/config.ini' config.read(gisy\_path)

config.sections() 返回配置文件中节序列

config.options(section) 返回某个节中的所有键的序列 ['port', 'alias']

config.items("size") 得到指定section的所有键值对 [(size,1024),(gisy,wkb)]

config.get(section,option) 返回section节中，option键的键值，返回为string类型

config.getint(section,option) 得到section节中，option键的键值，，返回为int类型

config.add\_section(str) 添加一个配置文件节点(str) 只会在缓存中添加，不会添加到文件中

config.set(section,option,val) 设置section节点中，键名为option的值(val)

config.write( open(filename, 'a') ) 将当前config缓存写入配置文件filename

from ConfigParser import ConfigParser

config=ConfigParser()

d=config.read("C:\Program Files\Tencent\QQGame\config\Original\QQGameRpc\_Login.ini")

for section in config.sections():

print section

for option in config.options(section):

print *""*,option,*"="*,config.get(section,option)

import ConfigParser

conf\_dbpath='conf/interface\_config.ini'

#根据配置文件中节点名称获取配置信息

def getconfdb(section):

cf = ConfigParser.ConfigParser()

cf.read(conf\_dbpath)

option = cf.options(section)

parameters = [cf.get(section,p) for p in option]

paramedict = dict(zip(option,parameters))

return paramedict

### python-decouple模块 将设置和代码完全隔离

pip install python-decouple

使用方法：

在你的settings.py.文件里面,你可以这么写:

1. 导入config对象：

from decouple import config

2. 获取配置参数

SECRET\_KEY = config('SECRET\_KEY')

DEBUG = config('DEBUG', default=False, cast=bool)

EMAIL\_HOST = config('EMAIL\_HOST', default='localhost')

EMAIL\_PORT = config('EMAIL\_PORT', default=25, cast=int)

有些时候你想不通过.ini或者.env文件改变一个参数，从3.0开始，decouple将遵从unix的方式，环境变量优先级高于配置文件变量；你可以覆盖配置文件的变量：DEBUG=True

一个简单的例子：

.evn文件

user\_info=root

port=3306

ipadd=127.0.0.1

switch=on

pyton解析文件：

import decouple

config = decouple.AutoConfig()

ipadd = config('ipadd')

print ipadd

Decouple支持两种类型：.ini文件和.env 文件

INI文件示例：

创建一个settings.ini文件，下一步在你的文件中可以写入如下格式的配置，例如：

DEBUG=True

TEMPLATE\_DEBUG=%(DEBUG)s

DATABASE\_URL=mysql://myuser:mypassword@myhost/mydatabase

PERCENTILE=90%%

注意：因为ConfigParser模块支持字符串填充的原因，百分号%应该由两个百分号，也就是%%代替

在你的程序的根目录中创建一个.env文件，例如：

DEBUG=True

TEMPLATE\_DEBUG=True

DATABASE\_URL=mysql://myuser:mypassword@myhost/mydatabase

PERCENTILE=90%

## 有关加密

### hashlib 摘要算法 存储密码 不可逆

#### 用法

import hashlib

m=hashlib.md5()

m.update(*'Gisy'*)

print m.digest() #二进制

print m.hexdigest() #十六进制

如果只对一个字符串，也可以这样

print hashlib.new("md5", "Nobody inspects the spammish repetition").hexdigest()

hashlib是个专门提供hash算法的库，现在里面包括md5, sha1, sha224, sha256, sha384, sha512，使用非常简单、方便。

md5经常用来做用户密码的存储。而sha1则经常用作数字签名。

import hashlib

a = "a test string"

print hashlib.md5(a).hexdigest()

print hashlib.sha1(a).hexdigest()

print hashlib.sha224(a).hexdigest()

print hashlib.sha256(a).hexdigest()

print hashlib.sha384(a).hexdigest()

print hashlib.sha512(a).hexdigest()

raw\_input() 将所有输入作为字符串看待，返回字符串类型。而 input() 在对待纯数字输入时具有自己的特性，它返回所输入的数字的类型（ int, float ）

import uuid

import hashlib

def encrypt\_password(password):

m=hashlib.md5()

m.update(password)

return m.hexdigest()

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

pw=raw\_input('please input your password:')

print ("The password stored in the database is:" + encrypt\_password(pw))

我们以常见的摘要算法MD5为例，计算出一个字符串的MD5值：

import hashlib

md5 = hashlib.md5()

md5.update('how to use md5 in python hashlib?')

print md5.hexdigest()

计算结果如下：

d26a53750bc40b38b65a520292f69306

#如果有中文的话，使用gb2312编码

>>> myMd5 = hashlib.md5('python 是一门语言'.encode('gb2312'))

>>> myMd5.hexdigest()

'6c0f33c5f4b96f1aa771bf432ba53002'

如果数据量很大，可以分块多次调用update()，最后计算的结果是一样的：

md5 = hashlib.md5()

md5.update('how to use md5 in ')

md5.update('python hashlib?')

print md5.hexdigest()

#### 原理解释

摘要算法在很多地方都有广泛的应用。要注意摘要算法不是加密算法，不能用于加密（因为无法通过摘要反推明文），只能用于防篡改，但是它的单向计算特性决定了可以在不存储明文口令的情况下验证用户口令。

hashlib模块中md5，sha1，sha224，sha256，sha384，sha512等算法

摘要算法又称哈希算法、散列算法。它通过一个函数，把任意长度的数据转换为一个长度固定的数据串（通常用16进制的字符串表示）。

举个例子，你写了一篇文章，内容是一个字符串'how to use python hashlib - by Michael'，并附上这篇文章的摘要是'2d73d4f15c0db7f5ecb321b6a65e5d6d'。如果有人篡改了你的文章，并发表为'how to use python hashlib - by Bob'，你可以一下子指出Bob篡改了你的文章，因为根据'how to use python hashlib - by Bob'计算出的摘要不同于原始文章的摘要。

可见，摘要算法就是通过摘要函数f()对任意长度的数据data计算出固定长度的摘要digest，目的是为了发现原始数据是否被人篡改过。

试试改动一个字母，看看计算的结果是否完全不同。

MD5是最常见的摘要算法，速度很快，生成结果是固定的128 bit字节，通常用一个32位的16进制字符串表示。

另一种常见的摘要算法是SHA1，调用SHA1和调用MD5完全类似：

import hashlib

sha1 = hashlib.sha1()

sha1.update('how to use sha1 in ')

sha1.update('python hashlib?')

print sha1.hexdigest()

SHA1的结果是160 bit字节，通常用一个40位的16进制字符串表示。

比SHA1更安全的算法是SHA256和SHA512，不过越安全的算法越慢，而且摘要长度更长。

有没有可能两个不同的数据通过某个摘要算法得到了相同的摘要？完全有可能，因为任何摘要算法都是把无限多的数据集合映射到一个有限的集合中。这种情况称为碰撞，比如Bob试图根据你的摘要反推出一篇文章'how to learn hashlib in python - by Bob'，并且这篇文章的摘要恰好和你的文章完全一致，这种情况也并非不可能出现，但是非常非常困难。

摘要算法应用 **存储密码**

任何允许用户登录的网站都会存储用户登录的用户名和口令。如何存储用户名和口令呢？方法是存到数据库表中。

如果以明文保存用户口令，如果数据库泄露，所有用户的口令就落入黑客的手里。此外，网站运维人员是可以访问数据库的，也就是能获取到所有用户的口令。

正确的保存口令的方式是不存储用户的明文口令，而是存储用户口令的摘要，比如MD5：

username | password

---------+---------------------------------

michael | e10adc3949ba59abbe56e057f20f883e

bob | 878ef96e86145580c38c87f0410ad153

当用户登录时，首先计算用户输入的明文口令的MD5，然后和数据库存储的MD5对比，如果一致，说明口令输入正确，如果不一致，口令肯定错误。

使用加盐和用户名使每个用户的密码不同，使黑客更无法破解

由于常用口令的MD5值很容易被计算出来，所以，要确保存储的用户口令不是那些已经被计算出来的常用口令的MD5，这一方法通过对原始口令加一个复杂字符串来实现，俗称“加盐”：

get\_md5(password + 'the-Salt')

经过Salt处理的MD5口令，只要Salt不被黑客知道，即使用户输入简单口令，也很难通过MD5反推明文口令。

如果假定用户无法修改登录名，就可以通过把登录名作为Salt的一部分来计算MD5，从而实现相同口令的用户也存储不同的MD5。

get\_md5(password + username + 'the-Salt')

### hmac 不可逆

hmac模块实现了hmac算法，需要一个key来进行加密

这可用于在电子邮件或其它非二进制环境中安全交换数据。

hmac用法如下：

import hmac

myhmac = hmac.new(b'mykey')

myhmac.update(b'mymessage')

print myhmac.hexdigest() 'd811630c4e62c6ef90d1bfe540212aaf'

print myhmac.digest()

a=myhmac.copy()

HMAC =hmac.new(key[, msg[, digestmod]])

返回一个新的 hmac 对象。如果提供了 msg ，该方法调用 update(msg) 。 digestmod 是 HMAC 对象将要使用的摘要构造器或模块。 缺省为 hashlib.md5。

HMAC.update(msg)

用 string 参数 msg 来更新 HMAC 对象。重复调用该方法等价于一次调用该方法，但传入所有参数按调用的顺序连接的值。即 m.update(a); m.update(b) 等价于 m.update(a + b)

HMAC.digest()

返回到目前为止传递给 update() 方法的字符串的摘要（字节串）。这个字串和传给构造器的 digest 的 digest\_size 一样长，可能会包括非 ASCII 字符，包括 NUL 字节。

HMAC.hexdigest()

就像 digest() 一样，除了返回一个 digest() 的结果两倍长的只包含十六进制数的摘要。可以应用于一些非二进制的环境中。

HMAC.copy()

返回 HMAC 对象的拷贝，可以有效地计算某些具有相同子串的字符串的摘要。

hmac.compare\_digest(a, b)

返回 a == b. 该函数通过禁止基于内容的短路（short circuiting）行为来阻止定时分析（timing analysis），从而适用于密码学。 参数中的 a 和 b 必须是相同类型的，要么是 unicode 要么是一个类字节对象（bytes-like object）。

　注意：

　　如果 a 和 b 的长度不同，或者一个错误发生，定时攻击理论上可以得知 a 和 b 的长度和类型，但是不能获得它们的值。

### base64 可逆

广泛应用于MIME协议，作为电子邮件的传输编码，生成的编码可逆，后一两位可能有“=”，生成的编码都是ascii字符。

Base64是一种任意二进制到文本字符串的编码方法，常用于在URL、Cookie、网页中传输少量二进制数据。

import base64

s = '我是字符串'

a = base64.b64encode(s)

print a #ztLKx9fWt/u0rg==

print base64.b64decode(a) #我是字符串

Python base64模块真正用的上的方法只有8个，分别是encode, decode, encodestring, decodestring, b64encode,b64decode, urlsafe\_b64decode,urlsafe\_b64encode。他们8个可以两两分为4组，

encode,decode一组，专门用来编码和 解码文件的,也可以对StringIO里的数据做编解码；

encodestring,decodestring一组，专门用来编码和解码字符串；

b64encode和b64decode一组，用来编码和解码字符串，并且有一个替换符号字符的功能。

这个功能是这样的：因为base64编码后的字符除 了英文字母和数字外还有三个字符 + / =, 其中=只是为了补全编码后的字符数为4的整数，而+和/在一些情况下需要被替换的，b64encode和b64decode正是提供了这样的功能。至于什 么情况下+和/需要被替换，最常见的就是对url进行base64编码的时候。

urlsafe\_b64encode和urlsafe\_b64decode 一组，这个就是用来专门对url进行base64编解码的，实际上也是调用的前一组函数。

由于标准的Base64编码后可能出现字符+和/，在URL中就不能直接作为参数，所以又有一种"url safe"的base64编码，其实就是把字符+和/分别变成-和\_：

b = base64.urlsafe\_b64encode(a) # 进行url的字符串编码

### uuid模块

UUID是128位的全局唯一标识符，通常由32字节的字符串表示。它可以保证时间和空间的唯一性

print hex(uuid.getnode()) #根据系统的mac地址打印 ,但是多个mac就有问题

u = uuid.uuid1()

print 'bytes :', repr(u.bytes)

print 'hex :', u.hex #uuid变16进制 <type 'str'>

print 'int :', u.int #uuid变10进制 <type 'long'>

print 'urn :', u.urn

print u"uuid1 生成基于计算机主机ID和当前时间的UUID"

print uuid.uuid1() # UUID('a8098c1a-f86e-11da-bd1a-00112444be1e')

print u"\nuuid3 基于命名空间和一个字符的MD5加密的UUID"

print uuid.uuid3(uuid.NAMESPACE\_DNS, 'python.org') #UUID('6fa459ea-ee8a-3ca4-894e-db77e160355e')

print u"\nuuid4 随机生成一个UUID"

print uuid.uuid4() #'16fd2706-8baf-433b-82eb-8c7fada847da'

print u"\nuuid5 基于命名空间和一个字符的SHA-1加密的UUID"

uuid.uuid5(uuid.NAMESPACE\_DNS, 'python.org') #UUID('886313e1-3b8a-5372-9b90-0c9aee199e5d')

print u"\n根据十六进制字符生成UUID"

x = uuid.UUID('{00010203-0405-0607-0809-0a0b0c0d0e0f}')

print u"转换成十六进制的UUID表现字符"

print str(x) # '00010203-0405-0607-0809-0a0b0c0d0e0f'

首先，Python中没有基于DCE的，所以uuid2可以忽略；

其次，uuid4存在概率性重复，由无映射性，最好不用；

再次，若在Global的分布式计算环境下，最好用uuid1；

最后，若有名字的唯一性要求，最好用uuid3或uuid5。

### Pycrypto模块 加解密算法

Pycrypto提供了比较完善的加密算法。RSA广泛用于加密与解密，还有数字签名通信领域。使用Publick/Private秘钥算法中，加密主要用对方的公钥，解密用自己的私钥。签名用自己的私钥，验签用对方的公钥。

RSA是一种公钥密码算法，RSA的密文是对代码明文的数字的 E 次方求mod N 的结果。也就是将明文和自己做E次乘法，然后再将其结果除以 N 求余数，余数就是密文。RSA是一个简洁的加密算法。E 和 N 的组合就是公钥（public key）。

对于RSA的解密，即密文的数字的 D 次方求mod N 即可，即密文和自己做 D 次乘法，再对结果除以 N 求余数即可得到明文。D 和 N 的组合就是私钥（private key）。

#### 生成秘钥对

from Crypto import Random

from Crypto.Hash import SHA

from Crypto.Cipher import PKCS1\_v1\_5 as Cipher\_pkcs1\_v1\_5

from Crypto.Signature import PKCS1\_v1\_5 as Signature\_pkcs1\_v1\_5

from Crypto.PublicKey import RSA

# 伪随机数生成器

random\_generator = Random.new().read

# rsa算法生成实例

rsa = RSA.generate(1024, random\_generator)

# master的秘钥对的生成

private\_pem = rsa.exportKey()

with open('master-private.pem', 'w') as f:

f.write(private\_pem)

public\_pem = rsa.publickey().exportKey()

with open('master-public.pem', 'w') as f:

f.write(public\_pem)

# ghost的秘钥对的生成

private\_pem = rsa.exportKey()

with open('ghost-private.pem', 'w') as f:

f.write(private\_pem)

public\_pem = rsa.publickey().exportKey()

with open('ghost-public.pem', 'w') as f:

f.write(public\_pem)

#### 加密与解密

通常通信的时候，发送者使用接受者的公钥加密，接受者使用接受者私钥解密。

Master给Ghost通信，需要加密内容，那么Ghost会生成一个秘钥对，Ghost的公钥ghost-public.pem和私钥ghost-private.pem 。Ghost 把公钥公开给发送者，任何人都可以用来加密，然后Master使用ghost-public.pem进行加密，然后把内容发给Ghost，Ghost再使用ghost-private.pem进行解密。

# Master使用Ghost的公钥对内容进行rsa 加密

message = 'hello ghost, this is a plian text'

with open('ghost-public.pem') as f:

key = f.read()

rsakey = RSA.importKey(key)

cipher = Cipher\_pkcs1\_v1\_5.new(rsakey)

cipher\_text = base64.b64encode(cipher.encrypt(message))

print cipher\_text

# Ghost使用自己的私钥对内容进行rsa 解密

with open('ghost-private.pem') as f:

key = f.read()

rsakey = RSA.importKey(key)

cipher = Cipher\_pkcs1\_v1\_5.new(rsakey)

text = cipher.decrypt(base64.b64decode(encrypt\_text), random\_generator)

assert text == message, 'decrypt falied'

#### 签名与验签

当然，对于窃听者，有时候也可以对伪造Master给Ghost发送内容。为此出现了数字签名。也就是Master给Ghost发送消息的时候，先对消息进行签名，表明自己的身份，并且这个签名无法伪造。具体过程即Master使用自己的私钥对内容签名，然后Ghost使用Master的公钥进行验签。

签名

# Master 使用自己的公钥对内容进行签名

with open('master-private.pem') as f:

key = f.read()

rsakey = RSA.importKey(key)

signer = Signature\_pkcs1\_v1\_5.new(rsakey)

digest = SHA.new()

digest.update(message)

sign = signer.sign(digest)

signature = base64.b64encode(sign)

验签

with open('master-public.pem') as f:

key = f.read()

rsakey = RSA.importKey(key)

verifier = Signature\_pkcs1\_v1\_5.new(rsakey)

digest = SHA.new()

# Assumes the data is base64 encoded to begin with

digest.update(message)

is\_verify = signer.verify(digest, base64.b64decode(signature))

is\_verify为 true 代表是master发送的,将signature与解密后的message生成的digest进行比对

## 有关模板

python独立模板有mako和jinja2

### mako

from mako.template import Template

mytemplate = Template(filename='/docs/mytmpl.txt')

print mytemplate.render()

可以用 <% %> 标签引入任意Python代码块：

this is a template

<%

x = db.get\_resource('foo')

y = [z.element for z in x if x.frobnizzle==5]

%>

% for elem in y:

element: ${elem}

% endfor

### jinja2

<http://docs.jinkan.org/docs/jinja2/templates.html>

flask框架的模板就是使用jinja2

在任何框架下如何导入jinja2?

from flask import Flask,render\_template,request,current\_app

from jinja2 import Template,Environment, FileSystemLoader

import jinja2,os

app= Flask(\_\_name\_\_,instance\_relative\_config=True)

@app.route("/",methods=["GET","POST"])

def fenbiao():

if request.method=='POST':

orderid=request.form['orderid']

remainder=int(orderid) % 32

app.logger.debug("订单号%s的后缀是%d",orderid,remainder)

return **jinja2.Environment(loader=jinja2.FileSystemLoader('./templates')**

**).get\_template('fenbiao.html').render(orderid=orderid,remainder=remainder)**

else:

return jinja2.Environment(loader=jinja2.FileSystemLoader('./templates')

).get\_template('fenbiao.html').render()

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

app.run(host="127.0.0.1",port=5000, debug=True)

## 有关流程图

### dot形式 graphviz

#### 生成图片方式

先生成dot文件，然后在cmd命令栏中输入命令

dot gisy.dot –Tpng –o gisy.png

根据生成的不同格式选择不同的输出方式

dot gisy.dot –Tbmp –o gisy.bmp

#### dot格式

graph（无向图）或者digraph（有向图）表示图，然后｛｝中的内容是对图的描述，注释风格和C类似（“//”用于单行注释，/\*\*/用于多行注释）。如一个无向图：

graph graph1 { //无向图graph1

a -- b }

无向图必须使用graph 之间连线用 -- 有向图必须使用digraph前缀， 之间连线用->

graph [pad="0.5", nodesep="1", ranksep="2"];

记得label=后面的字符串要加双引号, 否则报错 pygraphviz.agraph.DotError；有个例外，单个单词可以不加

可以将文本为只有1行，如下

digraph graphname {graph [encoding="UTF-8",rankdir=LR];1[label="Teacher df"];2[label="Pupil yujg"];1-> 2[fontcolor=darkgreen,label=Instructions];}

##### 带标签的简单有向图

digraph graphname{

T [label="Teacher"] // node T

P [label="Pupil"] // node P

T->P [label="Instructions", fontcolor=darkgreen] // edge T->P

}

##### 带有复杂属性设置的有向图

digraph summary{

start [label="Start with a Node"]

next [label="Choose your shape", shape=box]

warning [label="Don't go overboard", color=Blue, fontcolor=Red,fontsize=24,style=filled, fillcolor=green,shape=octagon]

end [label="Draw your graph!", shape=box, style=filled, fillcolor=yellow]

start->next

start->warning

next->end [label="Getting Better...", fontcolor=darkblue]

}

##### 节约时间的技巧

digraph hierarchy {

nodesep=1.0 // increases the separation between nodes

node [color=Red,fontname=Courier,shape=box] //All nodes will this shape and colour

edge [color=Blue, style=dashed] //All the lines look like this

Headteacher->{Deputy1 Deputy2 BusinessManager}

Deputy1->{Teacher1 Teacher2}

BusinessManager->ITManager

{rank=same;ITManager Teacher1 Teacher2} // Put them on the same level

}

##### 设置graph整体格式

digraph graphname {

graph [encoding="UTF-8",

rankdir=LR

];

1 [label=Teacher];

2 [label=Pupil];

1 -> 2 [fontcolor=darkgreen,

label=Instructions];

2 -> 2 [ label = "a " ]; #可以前后节点为同一个

}

##### 显示中文/单行显示

digraph graphname {graph [encoding="UTF-8",rankdir=LR];node [fontname = "Microsoft YaHei"];edge [fontname = "Microsoft YaHei"];1[label="你好"];2[label="Pupil yujg"];1-> 2[fontcolor=darkgreen,label=Instructions];}

##### 一个节点有多个部分

digraph structs {

node[shape=record]

struct1 [label="<f0> left|<f1> mid\ dle|<f2> right"];

struct2 [label="{<f0> one|<f1> two\n\n\n}" shape=Mrecord];

struct3 [label="hello\nworld |{ b |{c|<here> d|e}| f}| g | h"];

struct1:f1 -> struct2:f0;

struct1:f0 -> struct3:f1;

}

#### dot属性

图像属性

label="My Graph"; # 给图像设置标签

rankdir=LR; # 将图片由原来的从上到下布局变成从左到右布局

{rank=same; a, b, c } # 将一组元素放到同一个level

splines="line"; # 让边框变为直线，没有曲线和锐角

K=0.6; # 用来在布局中影响spring属性，spring属性可以用于将节点往外推，这个在twopi和sfdp布局中很有用。

交点属性

[label="Some Label"] # 给交点打标签

[color="red"] # 给交点上色

[fillcolor="blue"] # 设置交点的填充色

边的属性

[label="Some Label"] # 给边设置标签 (设置路径权重的时候很有用)

[color="red"] # 给交点上色 (标示路径的时候很有用)

[penwidth=2.0] # 给边适配厚度，标示路径的时候很有用。

尺寸, 背景颜色

fixedsize=true;

size="1,1";

resolution=72;

bgcolor="#C6CFD532";

### pygraphviz模块 画流程图

可用该模块画流程图 所需环境： Win7， python2.7

#### 简单举例

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

import pygraphviz

G = pygraphviz.AGraph(strict=False,

directed=True, # 有向图

encoding='UTF-8', # 为了可以显示中文

rankdir='LR', # 从左到右，默认为 TB

)

G.add\_node(*'a'*)

G.add\_node(*'b'*)

G.add\_edge(*'b'*, *'c'*) #**不需要**先加载相关节点，也能加载连接线

#批量加载节点

nodelist = [*'f'*, *'g'*, *'h'*]

G.add\_nodes\_from(nodelist)

# attributes

G.graph\_attr[*'label'*] = *"simple nodes and edge"*

G.node\_attr[*'shape'*] = *'circle'*

G.edge\_attr[*'color'*] = *'red'*

print G #输出生成的源码

s = G.to\_string() #将源码写入到变量s中

print s

G.write(*"first.dot"*) #在该文件写入graph生成的源码

G.layout(prog=*'dot'*)

G.draw(*'first.png'*)

#### 安装

一、下载相关

1、pygraphviz 1.3.1（zip），网址：https://pypi.Python.org/pypi/pygraphviz/

2、graphviz-2.38.msi软件，网址：http://www.graphviz.org/Download\_windows.php

3、【关键】pygraphviz-1.3.1-cp27-none-win\_amd64.whl，网址：http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygraphviz

★这个根据自己的python版本和系统版本下载相应的whl文件。

二、运行：

1、安装graphviz-2.38.msi软件，然后为其配置环境变量，例如将以下路径加入path：（注意分号，会的应该都懂）

;C:\Program Files (x86)\Graphviz2.38\bin;

2、解压pygraphviz 1.3.1（zip）；

3、将pygraphviz-1.3.1-cp27-none-win32.whl这个文件拖入到上面解压后的文件夹；

4、打开CMD，首先在CMD进入到上述文件夹目录，然后再运行命令：如果python为32位

pip install pygraphviz-1.3.1-cp27-none-win32.whl

为64位使用

pip install pygraphviz-1.3.1-cp27-none-win\_amd64.whl

然后就直接提示安装成功了，中间不需要在文件里指定软件include和lib路径。

如果运行提示ValueError: Program dot not found in path. 请尝试将eclipse重启，这样eclipse才会加载新的环境变量

#### 加载dot文件

使用dot文件创建实例

G=pgv.AGraph("gisy.dot")

最常见就是使用strict directed关键词创建实例

G=pgv.AGraph(strict=False,directed=True)、

可以两者合并

G = pygraphviz.AGraph("gisy.dot",strict=False,

directed=True, # 有向图

encoding='UTF-8', # 为了可以显示中文

rankdir='LR', # 从左到右，默认为 TB

)

import pygraphviz as pgv

G = pgv.AGraph("gisy.dot")

print G

G.write("first.dot")

G.layout(prog='dot')

G.draw('first.png')

**可以加载字符串**

import pygraphviz

G = pygraphviz.AGraph('digraph graphname {graph [encoding="UTF-8",rankdir=LR];node [fontname = "Microsoft YaHei"];edge [fontname = "Microsoft YaHei"];1[label="你好"];2[label="Pupil yujg"];1-> 2[fontcolor=darkgreen,label=Instructions];}',)

G.add\_node('a') #也可继续添加

G.write("first.dot") #在该文件写入graph生成的源码

G.layout(prog='dot')

G.draw('gisy.png')

#### 属性

添加属性3种方式

G.edge\_attr['color']='red'

G.add\_edge('b','c',color='blue')

e=G.get\_edge('b','c')

e.attr['color']='green'

Node

创建节点的时候可以传入很多参数，具体可以去查阅 attrs 文档。这里介绍一些简单常用的参数。

label：node 内的文字。

color：node 的颜色。

shape：node 的类型。

image：node 的图片。

style：node 的样式。

注意要使用 image 属性的话，要令 label=''，否则节点名会作为默认的 label 遮盖在图片上。

在添加结点时，可以在括号中添加结点的属性

g.add\_node('node0', label='节点 0', color='lightgrey', style='filled')

g.add\_node('node1', label='节点1', shape='box', color='red', style='dotted')

g.add\_node('node2', shape='plaintext', label='', image='./head.jpg')

Edge

常用的连接属性有：

color：连接线的颜色。

label：连接线的注释。

style：连接线的样式。

arrowhead：连接线的箭头样式。

dir：连接线的方向。

g.add\_edge('node0', 'node1', color='red', arrowhead='diamond')

g.add\_edge('node1', 'node2', color='green', style='dotted')

g.add\_edge('node2', 'node0', label='我转', color='green:red;0.5', dir='both')

layout

G.layout(prog=*'dot'*) 此为设置layout 流程图方向的

layout='dot' 最普遍

layout='neato'

layout='twopi'

layout='circo'

layout='fdp'

dot 默认布局方式，主要用于有向图

neato 基于spring-model(又称force-based)算法

twopi 径向布局

circo 圆环布局

fdp 用于无向图

#### 其他

Subgraph

声明一个 cluster 的语法是： graph.add\_subgraph(['node2', 'node3'], name='cluster\_nodes')

需要注意的是，cluster 的名字一定要以 cluster\_开头。

可以通过在 add\_edge 里使用参数 lhead 来指向 cluster，不过需要先在 graph 里声明参数 compound=True。

g.add\_edge('node1', 'node3', lhead='cluster\_nodes')

## \_\_future\_\_模块 把下一个新版本的特性导入到当前版本

Python提供了\_\_future\_\_模块，把下一个新版本的特性导入到当前版本，于是我们就可以在当前版本中测试一些新版本的特性。

如果使用这个语句，则该语句必须是模块或程序的第一个语句。

\_\_future\_\_.all\_feature\_names

['nested\_scopes','generators','division','absolute\_import',

'with\_statement','print\_function','unicode\_literals']

1. from \_\_future\_\_ import unicode\_literals 这个是对字符串使用unicode字符

2.x里的字符串用'xxx'表示str，Unicode字符串用u'xxx'表示unicode，而在3.x中，所有字符串都被视为unicode，因此，写u'xxx'和'xxx'是完全一致的，而在2.x中以'xxx'表示的str就必须写成b'xxx'，以此表示“二进制字符串”。

2. from \_\_future\_\_ import division 在Python 2.7的代码中直接使用Python 3.x的除法。 所有的除法都是精确除法，地板除 取整的用//表示

3 / 5 0.6

3. from \_\_future\_\_ import with\_statement 使在2.6之前版本下可以使用with语句

4. from \_\_future\_\_ import print\_function 新的print是一个函数，如果导入此特性，之前的print语句就不能用了。

print('test')  
5. from \_\_future\_\_ import generators 生成器，对应yield的语法，python2.7.5中默认使用

1. from \_\_future\_\_ import nested\_scopes 这个是修改嵌套函数或lambda函数中变量的搜索顺序，从`当前函数命名空间->模块命名空间`的顺序更改为了`当前函数命名空间->父函数命名空间->模块命名空间`,python2.7.5中默认使用
2. from \_\_future\_\_ import absolute\_import

相对导入：在不指明 package 名的情况下导入自己这个 package 的模块，比如一个 package 下有 a.py 和 b.py 两个文件，在 a.py 里 from . import b 即是相对导入 b.py。

绝对导入：指明顶层 package 名。比如 import a，Python 会在 sys.path 里寻找所有名为 a 的顶层模块。

from \_\_future\_\_ import absolute\_import: 在 3.0 以前的旧版本中启用相对导入等特性所必须的 future 语句。

## 其他

### PySpider

PySpider 是一个我个人认为非常方便并且功能强大的爬虫框架，支持多线程爬取、JS动态解析，提供了可操作界面、出错重试、定时爬取等等的功能，使用非常人性化。

1. 安装 pip install pyspider
2. 运行 pyspider 并 启动它的所有组件。 cmd 命令行输入 pyspider all 然后浏览器访问 <http://localhost:5000>

### selenium

#### 基础

1. 安装Chrome driver

下载解压，你会得到一个chromedriver.exe文件，需要把这家伙放到chrome的安装目录下...\Google\Chrome\Application\ ,然后设置path环境变量，chrome的安装目录（C:\Program Files\Google\Chrome\Application），然后再调用运行

1. 要想调用键盘按键操作需要引入keys包：

from selenium.webdriver.common.keys import Keys

通过send\_keys()调用按键：

send\_keys(Keys.TAB) # TAB

send\_keys(Keys.ENTER) # 回车

1. 输入中文问题

顺便解决了一个困扰我到现的一个输入中文件的问题。selenium2 python在send\_keys()中输入中文一直报错，其实前面加个小u 就解决了：

send\_keys(u"输入中文")

1. from selenium.webdriver.common.action\_chains import ActionChains #引入ActionChains鼠标操作类

ActionChains(browser).move\_to\_element(article).perform()#将鼠标移动到这里，但是这里不好用

ActionChains(browser).context\_click(article).perform()

1. 下载selenium的jar。

启动使用java -jar selenium-server-standalone-2.53.0.jar

1. 怎么下拉页面到底部？

driver**.**execute\_script("window.scrollTo(0, document.body.scrollHeight);")

import requests

content\_type = requests.head('http://www.python.org').headers['content-type']

print(content\_type)

1. 对当前页面进行截图保存

from selenium import webdriver

driver=webdriver.Firefox()

driver.get('http://www.python.org')

driver.save\_screenshot('screenshot.png')

driver.quit()

#### 可以获取常用的值

size 获取元素的尺寸

text 获取元素的文本

get\_attribute(name) 获取属性值

location 获取元素坐标，先找到要获取的元素，再调用该方法

page\_source 返回页面源码

driver.title 返回页面标题

current\_url 获取当前页面的URL

is\_displayed() 设置该元素是否可见

is\_enabled() 判断元素是否被使用

is\_selected() 判断元素是否被选中

tag\_name 返回元素的tagName

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

import time

driver = webdriver.PhantomJS(executable\_path="G:\phantomjs-1.9.1-windows\phantomjs.exe")

driver.get("http://www.baidu.com/")

size = driver.find\_element\_by\_name("wd").size

print size

#尺寸: {'width': 500, 'height': 22}

news = driver.find\_element\_by\_xpath("//div[@id='u1']/a[1]").text

print news

#文本: 新闻

href = driver.find\_element\_by\_xpath("//div[@id='u1']/a[2]").get\_attribute('href')

name = driver.find\_element\_by\_xpath("//div[@id='u1']/a[2]").get\_attribute('name')

print href,name

#属性值: http://www.hao123.com/ tj\_trhao123

location = driver.find\_element\_by\_xpath("//div[@id='u1']/a[3]").location

print location

#坐标: {'y': 19, 'x': 498}

print driver.current\_url

#当前链接: https://www.baidu.com/

print driver.title

#标题: 百度一下， 你就知道

result = location = driver.find\_element\_by\_id("su").is\_displayed()

print result

#是否可见: True

#### 页面操作

##### 页面交互

仅仅抓取页面没有多大卵用，我们真正要做的是做到和页面交互，比如点击，输入等等。那么前提就是要找到页面中的元素。WebDriver提供了各种方法来寻找元素。例如下面有一个表单输入框。

<input type="text" name="passwd" id="passwd-id" />

我们可以这样获取它

element = driver.find\_element\_by\_id("passwd-id")

element = driver.find\_element\_by\_name("passwd")

element = driver.find\_elements\_by\_tag\_name("input")

element = driver.find\_element\_by\_xpath("//input[@id='passwd-id']")

你还可以通过它的文本链接来获取，但是要小心，文本必须完全匹配才可以，所以这并不是一个很好的匹配方式。

而且你在用 xpath 的时候还需要注意的是，如果有多个元素匹配了 xpath，它只会返回第一个匹配的元素。如果没有找到，那么会抛出 NoSuchElementException 的异常。

获取了元素之后，下一步当然就是**向文本输入内容**了，可以利用下面的方法

element.send\_keys("some text")

同样你还可以利用 Keys 这个类来模拟点击某个按键。

element.send\_keys("and some", Keys.ARROW\_DOWN)

你可以对任何获取到到元素使用 send\_keys 方法，就像你在 GMail 里面点击发送键一样。不过这样会导致的结果就是输入的文本不会自动清除。所以输入的文本都会在原来的基础上继续输入。你可以用下面的方法来清除输入文本的内容。

element.clear()

这样输入的文本会被清除。

##### 填充表单

我们已经知道了怎样向文本框中输入文字，但是其它的表单元素呢？例如下拉选项卡的的处理可以如下

element = driver.find\_element\_by\_xpath("//select[@name='name']")

all\_options = element.find\_elements\_by\_tag\_name("option")

for option in all\_options:

print("Value is: %s" % option.get\_attribute("value"))

option.click()

首先获取了第一个 select 元素，也就是下拉选项卡。然后轮流设置了 select 选项卡中的每一个 option 选项。你可以看到，这并不是一个非常有效的方法。

其实 WebDriver 中提供了一个叫 Select 的方法，可以帮助我们完成这些事情。

from selenium.webdriver.support.ui import Select

select = Select(driver.find\_element\_by\_name('name'))

select.select\_by\_index(index)

select.select\_by\_visible\_text("text")

select.select\_by\_value(value)

如你所见，它可以根据索引来选择，可以根据值来选择，可以根据文字来选择。是十分方便的。

全部取消选择怎么办呢？很简单

select = Select(driver.find\_element\_by\_id('id'))

select.deselect\_all()

这样便可以取消所有的选择。

另外我们还可以通过下面的方法获取所有的已选选项。

select = Select(driver.find\_element\_by\_xpath("xpath"))

all\_selected\_options = select.all\_selected\_options

获取所有可选选项是

options = select.options

点击单选框/多选框按钮：

driver.find\_element\_by\_id("os\_cookie").click()

清空单选框/多选框按钮：

driver.find\_element\_by\_id("os\_cookie").clear()

如果你把表单都填好了，最后肯定要提交表单对吧。怎么提交呢？很简单

driver.find\_element\_by\_id("submit").click()

这样就相当于模拟点击了 submit 按钮，做到表单提交。

当然你也可以单独提交某个元素

element.submit()

方法，WebDriver 会在表单中寻找它所在的表单，如果发现这个元素并没有被表单所包围，那么程序会抛出 NoSuchElementException 的异常。

##### 元素拖拽

要完成元素的拖拽，首先你需要指定被拖动的元素和拖动目标元素，然后利用 ActionChains 类来实现。

element = driver.find\_element\_by\_name("source")

target = driver.find\_element\_by\_name("target")

from selenium.webdriver import ActionChains

action\_chains = ActionChains(driver)

action\_chains.drag\_and\_drop(element, target).perform()

这样就实现了元素从 source 拖动到 target 的操作。

##### 页面切换

一个浏览器肯定会有很多窗口，所以我们肯定要有方法来实现窗口的切换。切换窗口的方法如下

driver.switch\_to\_window("windowName")

另外你可以使用 window\_handles 方法来获取每个窗口的操作对象。例如

for handle in driver.window\_handles:

driver.switch\_to\_window(handle)

另外切换 frame 的方法如下

driver.switch\_to\_frame("frameName.0.child")

这样焦点会切换到一个 name 为 child 的 frame 上。

##### 鼠标操作

from selenium.webdriver import ActionChains

context\_click(elem) 右击鼠标点击元素elem，另存为等行为

double\_click(elem) 双击鼠标点击元素elem，地图web可实现放大功能

drag\_and\_drop(source,target) 拖动鼠标，源元素按下左键移动至目标元素释放

move\_to\_element(elem) 鼠标移动到一个元素上

click\_and\_hold(elem)按下鼠标左键并不松开在一个元素上

perform() 在通过调用该函数执行ActionChains中存储行为 可以在最后统一执行

release(elem)松开按住的鼠标

key\_down(key) 按住并一直按住某个键盘键

key\_up(key) 松开某个键盘键

#鼠标定位到子元素上

ActionChains(driver).move\_to\_element(menu).perform()

#定位到要右击的元素

qqq =driver.find\_element\_by\_xpath("/html/body/div/div[2]/div[2]/div/div[3]/table/tbody/tr/td[2]")

#对定位到的元素执行鼠标右键操作

ActionChains(driver).context\_click(qqq).perform()

#定位到要双击的元素

qqq =driver.find\_element\_by\_xpath("xxx")

#对定位到的元素执行鼠标双击操作

ActionChains(driver).double\_click(qqq).perform()

#定位元素的原位置

element = driver.find\_element\_by\_name("source")

#定位元素要移动到的目标位置

target = driver.find\_element\_by\_name("target")

#执行元素的移动操作

ActionChains(driver).drag\_and\_drop(element, target).perform()

#选中某块区域并右键显示快捷菜单

action.double\_click().key\_down(Keys.SHIFT).context\_click(target).key\_up(Keys.SHIFT)

---------------

import time

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.common.action\_chains import ActionChains

driver = webdriver.Firefox()

driver.get("http://www.baidu.com")

#鼠标移动至图片上 右键保存图片

elem\_pic = driver.find\_element\_by\_xpath("//div[@id='lg']/img")

print elem\_pic.get\_attribute("src")

action = ActionChains(driver).move\_to\_element(elem\_pic)

action.context\_click(elem\_pic)

#重点:当右键鼠标点击键盘光标向下则移动至右键菜单第一个选项

action.send\_keys(Keys.ARROW\_DOWN)

time.sleep(3)

action.send\_keys('v') #另存为

action.perform()

#获取另存为对话框(失败)

alert.switch\_to\_alert()

alert.accept()

##### 窗口大小控制

driver.maximize\_window() #将浏览器最大化显示

driver.set\_window\_size(480, 800) #参数数字为像素点 设置浏览器宽480、高800显示

##### 弹窗处理

当你出发了某个事件之后，页面出现了弹窗提示，那么你怎样来处理这个提示或者获取提示信息呢？

alert = driver.switch\_to\_alert()

通过上述方法可以获取弹窗对象。

##### 历史记录

那么怎样来操作页面的前进和后退功能呢？

driver.forward()

driver.back()

##### cookies处理

为页面添加 Cookies，用法如下

# Go to the correct domain

driver.get("http://www.example.com")

# Now set the cookie. This one's valid for the entire domain

cookie = {‘name’ : ‘foo’, ‘value’ : ‘bar’}

driver.add\_cookie(cookie)

获取页面 Cookies，用法如下

# Go to the correct domain

driver.get("http://www.example.com")

# And now output all the available cookies for the current URL

driver.get\_cookies()

以上便是 Cookies 的处理，同样是非常简单的。

##### 键盘操作

send\_keys(Keys.ENTER) 按下回车键

send\_keys(Keys.TAB) 按下Tab制表键

send\_keys(Keys.SPACE) 按下空格键space

send\_keys(Kyes.ESCAPE) 按下回退键Esc

send\_keys(Keys.BACK\_SPACE) 按下删除键BackSpace

send\_keys(Keys.SHIFT) 按下shift键

send\_keys(Keys.CONTROL) 按下Ctrl键

send\_keys(Keys.ARROW\_DOWN) 按下鼠标光标向下按键

send\_keys(Keys.CONTROL,'a') 组合键全选Ctrl+A

send\_keys(Keys.CONTROL,'c') 组合键复制Ctrl+C

send\_keys(Keys.CONTROL,'x') 组合键剪切Ctrl+X

send\_keys(Keys.CONTROL,'v') 组合键粘贴Ctrl+V

------

#coding=utf-8

import time

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

driver = webdriver.Firefox()

driver.get("http://www.baidu.com")

#输入框输入内容

elem = driver.find\_element\_by\_id("kw")

elem.send\_keys("Eastmount CSDN")

time.sleep(3)

#删除一个字符CSDN 回退键

elem.send\_keys(Keys.BACK\_SPACE)

elem.send\_keys(Keys.BACK\_SPACE)

elem.send\_keys(Keys.BACK\_SPACE)

elem.send\_keys(Keys.BACK\_SPACE)

time.sleep(3)

#输入空格+"博客"

elem.send\_keys(Keys.SPACE)

elem.send\_keys(u"博客")

time.sleep(3)

#ctrl+a 全选输入框内容

elem.send\_keys(Keys.CONTROL,'a')

time.sleep(3)

#ctrl+x 剪切输入框内容

elem.send\_keys(Keys.CONTROL,'x')

time.sleep(3)

#输入框重新输入搜索

elem.send\_keys(Keys.CONTROL,'v')

time.sleep(3)

#通过回车键替代点击操作

driver.find\_element\_by\_id("su").send\_keys(Keys.ENTER)

time.sleep(3)

driver.quit()

#### 元素选取

关于元素的选取，有如下的API

单个元素选取

find\_element\_by\_id

find\_element\_by\_name

find\_element\_by\_xpath

find\_element\_by\_link\_text 链接的文字说明

find\_element\_by\_partial\_link\_text 链接的文字说明中部分

find\_element\_by\_tag\_name

find\_element\_by\_class\_name

find\_element\_by\_css\_selector

多个元素选取

find\_elements\_by\_name

find\_elements\_by\_xpath

find\_elements\_by\_link\_text

find\_elements\_by\_partial\_link\_text

find\_elements\_by\_tag\_name

find\_elements\_by\_class\_name

find\_elements\_by\_css\_selector

driver.find\_element\_by\_link\_text(u'设置')

driver.find\_element\_by\_partial\_link\_text (u'设')

另外还可以利用 By 类来确定哪种选择方式

from selenium.webdriver.common.by import By

driver.find\_element(By.XPATH, '//button[text()="Some text"]')

driver.find\_elements(By.XPATH, '//button')

driver.find\_element\_by\_css\_selector(".nav")

By 类的一些属性如下

ID = "id"

XPATH = "xpath"

LINK\_TEXT = "link text"

PARTIAL\_LINK\_TEXT = "partial link text"

NAME = "name"

TAG\_NAME = "tag name"

CLASS\_NAME = "class name"

CSS\_SELECTOR = "css selector"

##### css selector

如果button上有class属性的，如:

<button id="ext-eng=1026" class="x-right-button"...>

可以这样写：

css=button.x-right-button

.代表class

如果class里带的空格，用.来代替空格如：

<button class="x-btn-text module\_picker\_icon">...

可以这样写：

css=button.x-btn-text.module\_picker\_icon

如果想用别的属性值定位，用方括号【属性名=属性值】的方式，如：

<abbr>Abc<abbr/>

css=abbr[title="Abc"]

<button id="ext-eng-1026" class="x-right-button">OK</button>

css=button.x-right-button:contains("OK")

:contains是个Pseudo-class，用冒号开头，括号里是内容

#### 页面等待

这是非常重要的一部分，现在的网页越来越多采用了 Ajax 技术，这样程序便不能确定何时某个元素完全加载出来了。这会让元素定位困难而且会提高产生 ElementNotVisibleException 的概率。

所以 Selenium 提供了两种等待方式，一种是隐式等待，一种是显式等待。

隐式等待是等待特定的时间，显式等待是指定某一条件直到这个条件成立时继续执行。

##### 显式等待

显式等待指定某个条件，然后设置最长等待时间。如果在这个时间还没有找到元素，那么便会抛出异常了。

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

driver = webdriver.Chrome()

driver.get("http://somedomain/url\_that\_delays\_loading")

try:

element = WebDriverWait(driver, 10).until(

EC.presence\_of\_element\_located((By.ID, "myDynamicElement"))

**)**

finally:

driver.quit()

10是要等待的秒数.如果没有满足until()方法中的条件,就会始终在这里wait 15秒,依然找不到,就抛出异常.

程序默认会 500ms 调用一次来查看元素是否已经生成，如果本来元素就是存在的，那么会立即返回。

#找到id 为dropdown1的父元素

WebDriverWait(dr, 10).until(lambda the\_driver: the\_driver.find\_element\_by\_id('dropdown1').is\_displayed())

下面是一些内置的等待条件，你可以直接调用这些条件，而不用自己写某些等待条件了。

title\_is

title\_contains

presence\_of\_element\_located

visibility\_of\_element\_located

visibility\_of

presence\_of\_all\_elements\_located

text\_to\_be\_present\_in\_element

text\_to\_be\_present\_in\_element\_value

frame\_to\_be\_available\_and\_switch\_to\_it

invisibility\_of\_element\_located

element\_to\_be\_clickable – it is Displayed and Enabled.

staleness\_of

element\_to\_be\_selected

element\_located\_to\_be\_selected

element\_selection\_state\_to\_be

element\_located\_selection\_state\_to\_be

alert\_is\_present

----------

from selenium.webdriver.support import expected\_conditions as EC

wait = WebDriverWait(driver, 10)

element = wait.until(EC.element\_to\_be\_clickable((By.ID,'someid')))

##### 隐式等待

隐式等待比较简单，就是简单地设置一个等待时间，单位为秒。

from selenium import webdriver

driver = webdriver.Chrome()

driver.implicitly\_wait(10) # seconds

driver.get("http://somedomain/url\_that\_delays\_loading")

myDynamicElement = driver.find\_element\_by\_id("myDynamicElement")

当然如果不设置，默认等待时间为0。

#### 常用命令

selenium常用命令

1. Open(url)

url可以是相对的，也可以是绝对的

如绝对的url：open(http://www.taobao.com/)

相对的url：open("/search?q=’aa’");

2. Click(locator)

click可以单击一个链接，按钮或者单选框复选框等。

Click系列的有：click、clickAt、clickAndWait、clickAtAndWait

clickAt(locator，coordstring): coordstring是需要单击的坐标

3. doubleClick(locator)

双击操作，用户与click一致

4. type(locator，value)

可以为指定位置输入指定值，也可以为单选框和复选框按钮赋值，value是选项的值而不是文本内容

5. typeKeys(locator,value)

模拟用户输入的行为，与type不同的是，type是强行赋值，而typeKeys是完全模拟用户操作，一个键一个键敲上去的。当输入框中原来就有值时，使用type会用新值替换旧值，而typeKeys则是在原来值上进行追加。如，原来值为aaa，现在要输入bbb，使用type则最终值为bbb，使用typeKeys则最终值为aaabbb.

6. focus(locator) 将焦点移动到指定的元素上，如果是一个可输入的元素，则将聚焦到输入框

7. select(locator，option)

选择下列框中的选项，option默认为标签label

如selenium.select (“sizeselect”,”label=small”)，id=sizeselect

8. selenium.goBack() 后退

9. highlight(locator) 指定元素高亮

10. refresh() 刷新当前页面

11. AssertText(locator，pattern)

验证某个元素的文本值是否与预期值一致，locator为元素定位，pattern为预期值

12. AssertAttribute(attributelocator，pattern)

验证某个元素的某个属性值与预期值是否一致，attributelocator为属性定位，pattern为预期值

13.assertEquals(value,selenium.getAttribute(xpath+"@value"))

验证某个元素的属性值或者是文本值是否与预期值一致

14. assertInPara(remarkpara1, getRemark(engine\_type));

一般用于验证url和remark串

15. mouseOver(locator)

将鼠标停留在某个元素上，locator为元素定位。

16. mouseDown(locator)/mouseUp(locator)

用户在某个元素上按下或者释放鼠标。Locator为元素定位

17. keyDown(locator，keysequence)/ keyUp(locator，keysequence)

用户按下或者释放某一个按钮，Locator为元素定位，keysequence为按键对应的ASCII码

18. indexOf

判断两个字符串是否存在包含关系

若x=”abcd”;y=”bc”;则有X.indexOf(y)==0

若不存在包含关系，则X.indexOf(y)==-1

19. System.out.println(value);

可以向屏幕打印出value值，这个value值可以是元素的属性值或者文本值等

20. selenium.getCookie();

可以取到当前页面的cookie值

21. selenium.deleteCookie(name，path)

删除cookie值中的某个参数，如下：

selenium.deleteCookie("nk","path=/, domain=.taobao.com, recurse=true");

删除nk参数，path的格式必须指定上面三个字段，path=/表示当前路径，domain=.taobao.com为改cookie所属的域，recurse=true

22. assertValue(locator,pattern)

验证某个可输入元素是否被赋了某个值

23. selenium.close()

关闭浏览器。

24. contextMenu(locator)

打开关联的菜单，与右键作用差不多，但是对弹出的菜单selenium没办法操作。

25. assertEquals(“aa”,selenium.getAttribute(xpath+"@value"));

取出value值来与某个已知值对比：

#### 断言

如果断言成功（如果为真）那么不执行任何操作！

如果断言不成功，那么会触发AssertionError

比如 assert "wangkaibin" not in driver.page\_source 判断页面中是否有 wangkaibin 如果没有 console抛出异常

assert 1 == 0,'one does not equal zero' 抛出异常 AssertionError: one does not equal zero

#### 例子

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

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

import time

browser=webdriver.Chrome()

browser.get('http://www.yahoo.com')

time.sleep(2)

assert *'Yahoo'* in browser.title

elem=browser.find\_element\_by\_name(*'p'*)

elem.send\_keys(*'seleniumhq'*+Keys.RETURN)

browser.quit()

--------

from selenium import webdriver

from selenium.common.exceptions import TimeoutException

from selenium.webdriver.support.ui import WebDriverWait # available since 2.4.0

import time

# Create a new instance of the browser driver

driver = webdriver.Firefox() ##可以替换为IE(), Firefox()

# go to the google home page

driver.get("http://www.google.com")

# find the element that's name attribute is q (the google search box)

inputElement = driver.find\_element\_by\_name(*"q"*)

# type in the search

inputElement.send\_keys(*"Cheese!"*)

# submit the form. (although google automatically searches now without submitting)

inputElement.submit()

# the page is ajaxy so the title is originally this:

print driver.title

try:

# we have to wait for the page to refresh, the last thing that seems to be updated is the title

WebDriverWait(driver, 10).until(lambda driver : driver.title.lower().startswith(*"cheese!"*))

# You should see "cheese! - Google Search"

print driver.title

finally:

driver.quit()

----------

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

import unittest

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

class PythonOrgSearch(unittest.TestCase):

def setUp(self):

self.driver = webdriver.Firefox()

def test\_search\_in\_python\_org(self):

driver = self.driver

driver.get("http://www.python.org")

self.assertIn("Python", driver.title)

elem = driver.find\_element\_by\_name("q")

elem.send\_keys("pycon")

elem.send\_keys(Keys.RETURN)

assert "No results found." not in driver.page\_source

def tearDown(self):

self.driver.close()

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

----------

from selenium import webdriver

import time

driver = webdriver.Firefox()

driver.get('http://detail.tmall.com/item.htm?id=12577759834 ')

time.sleep(10)

pageSource = driver.page\_source

print pageSource

driver.close()

------

#coding=utf-8

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

import os,time

driver = webdriver.Firefox()

driver.get("http://www.baidu.com")

#输入框输入内容

driver.find\_element\_by\_id(*"kw"*).send\_keys(*"selenium"*)

time.sleep(3)

#ctrl+a 全选输入框内容

driver.find\_element\_by\_id(*"kw"*).send\_keys(Keys.CONTROL,*'a'*)

time.sleep(3)

#ctrl+x 剪切输入框内容

driver.find\_element\_by\_id(*"kw"*).send\_keys(Keys.CONTROL,*'x'*)

time.sleep(3)

#输入框重新输入内容，搜索

driver.find\_element\_by\_id(*"kw"*).send\_keys(u*"王"*)

driver.find\_element\_by\_id(*"su"*).click()

time.sleep(3)

driver.quit()

------------

读取用例数

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

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

import time

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.support.ui import Select

from selenium.webdriver import ActionChains

from selenium.webdriver.common.by import By

driver=webdriver.Chrome()

driver.get('http://jira.360buy.com/confluence/login.action')

element1=driver.find\_element\_by\_id('os\_username')

element1.send\_keys('zhangda')

element2=driver.find\_element\_by\_id('os\_password')

element2.send\_keys('666666')

driver.find\_element\_by\_id("os\_cookie").click()

driver.find\_element\_by\_id("loginButton").click()

#driver.maximize\_window()

driver.find\_element\_by\_xpath('//td/a[@href="/confluence/display/GYLTEST"]').click()

time.sleep(2)

driver.find\_element\_by\_id('plusminus28591859-0').click()

time.sleep(2)

driver.find\_element\_by\_xpath('//a[@href="/confluence/pages/viewpage.action?pageId=28594884"]').click()

driver.find\_element\_by\_xpath('//input[@name="run\_1\_start\_time"]').send\_keys("2016-01-01")

driver.find\_element\_by\_xpath('//input[@name="run\_1\_end\_time"]').send\_keys("2016-02-01")

driver.find\_element\_by\_xpath('//button[@type="submit"]').click()

element= driver.find\_element\_by\_xpath("//th[@title='testlinkID']")

target = driver.find\_element\_by\_xpath('//div[@class="wiki-content"]/table/tbody/tr[last()]/td[last()]')

action = ActionChains(driver)

action.move\_to\_element(element)

action.double\_click().key\_down(Keys.SHIFT).click(target).key\_up(Keys.SHIFT)

action.send\_keys(Keys.CONTROL, 'c')

action.perform() #需要最后才加perform()

### jira模块

#### 实际运用

使用此模块与jira进行交互。

安装

最简单的安装 jira-python 的方法是通过 pip ：

$ pip install jira

1. 为何运行自己编写的py文件时，报错ImportError: No module named client?

请注意不要将自己编写的py文件设置为 jira.py

因为你的文件里就是要from jira.client import JIRA 。设置会造成程序无法读取jira模块。

1. jira.exceptions.JIRAError: JiraError HTTP 400

text: 无效的备注

url: http://jira.jd.com/rest/api/2/issue/289402/comment/187825

response headers = {'X-AUSERNAME': 'wangkaibin', 'X-AREQUESTID': '684x3045227x5', 'X-Content-Type-Options': 'nosniff', 'Content-Encoding': 'gzip', 'Transfer-Encoding': 'chunked', 'X-Seraph-LoginReason': 'OK', 'Vary': 'User-Agent', 'X-ASESSIONID': '1haeq09', 'Server': 'JDWS/1.0.0', 'Connection': 'close', 'Cache-Control': 'no-cache, no-store, no-transform', 'Date': 'Fri, 01 Apr 2016 03:23:02 GMT', 'Content-Type': 'application/json;charset=UTF-8'}

response text = {"errorMessages":[],"errors":{"comment":"无效的备注"}}

1. 查看class对象有什么方法 dir() 比如dir(gisy)
2. 中英文对照表
   1. dashboard 面板
   2. issue 需求或bug
   3. filter 过滤器

resources import Resource, Issue, Comment, Project, Attachment, Component, Dashboard, Filter, Votes, Watchers,

Worklog, IssueLink, IssueLinkType, IssueType, Priority, Version, Role, Resolution, SecurityLevel, Status, User,

CustomFieldOption, RemoteLink

dashboard {u'self': u'http://jira.jd.com/rest/api/2/dashboard/11910', u'view': u'http://jira.jd.com/secure/Dashboard.jspa?selectPageId=11910', u'id': u'11910', u'name': u'ofc\u6d4b\u8bd5'}

project 'assigneeType', 'avatarUrls', 'components', 'delete', 'description', 'expand', 'find', 'id', 'issueTypes', 'key', 'lead', 'name', 'raw', 'roles', 'self', 'update', 'url', 'versions']

1. 过滤器网址 全组bug统计-ofc
   1. <http://jira.jd.com/issues/?filter=13725&jql=project%20in%20(LSXTB%2C%20NEWLSXTB)%20AND%20issuetype%20%3D%20Bug%20AND%20created%20%3C%202016-04-01%20AND%20created%20%3E%3D%202016-01-01%20AND%20reporter%20in%20(bjguoyurui%2C%20lixionghu%2C%20zhangjinming%2C%20bjduanyingying%2C%20liufeng53%2C%20bjkeqinqin%2C%20bjzhangda%2C%20wangkaibin%2C%20wanglijiang5%2C%20fanghuan3%2C%20heyujiao3%2C%20liuhua106%2C%20zhouxuemei)%20ORDER%20BY%20resolution%20ASC>
   2. 全组项目需求-ofc
   3. <http://jira.jd.com/issues/?filter=13731> 已保存过滤器的网址
   4. <http://jira.jd.com/issues/?filter=13731&jql=project%20in%20(LSXTB%2C%20NEWLSXTB)%20AND%20issuetype%20%3D%20%E9%80%81%E6%B5%8B%20AND%20created%20%3C%202016-04-01%20AND%20created%20%3E%3D2016-01-01%20AND%20assignee%20in%20(bjguoyurui%2C%20wangkaibin%2C%20lixionghu%2C%20zhangjinming%2C%20wanglijiang5%2C%20bjkeqinqin%2C%20bjzhangda%2C%20heyujiao3%2C%20bjduanyingying%2C%20fanghuan3%2C%20zhouxuemei)> 修改过的过滤器网址
   5. 转移项目需求
   6. <http://jira.jd.com/issues/?filter=14662&jql=project%20in%20(LSXTB%2C%20NEWLSXTB)%20AND%20issuetype%20%3D%20%E9%80%81%E6%B5%8B%20AND%20created%20%3E%3D2016-01-01%20AND%20reporter%20in%20(lulanhua%2C%20changhaichao%2C%20yinhang5%2C%20jinyan%2C%20huaxuzhong%2C%20zhangzhiwei17%2C%20bjfengzhuo)>
   7. ofc测试面板 <http://jira.jd.com/secure/Dashboard.jspa?selectPageId=11910>

#### 代码分析

* 1. 创建issue 也就是需求或者bug

new\_issue **=** jira**.**create\_issue(project**=**'PROJ\_key\_or\_id', summary**=**'New issue from jira-python',

description**=**'Look into this one', issuetype**=**{'name': 'Bug'})

* 1. Or you can use a dict:

issue\_dict **=** {

'project': {'id': 123},

'summary': 'New issue from jira-python',

'description': 'Look into this one',

'issuetype': {'name': 'Bug'},

}

new\_issue **=** jira**.**create\_issue(fields**=**issue\_dict)

Project, summary, description and issue type are always required when creating issues.

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

from jira import JIRA

# 默认情况下，客户端从 Atlassian 的插件 SDK 中启用了实例去连接 JIRA

#options = {'server' : 'http://jira.jd.com'}

#jira=JIRA(options)

jira=JIRA(options={*'server'* : *'http://jira.jd.com'*},basic\_auth=(*'wangkaibin'*,*'Gisywang.8261'*)) # 一个账号 / 密码的元祖

# 获取所有的匿名用户可查看的项目。

projects = jira.project(*'LSXTB'*)

#print projects.key,dir(projects)

# 排序可用的项目键，然后返回第二个，第三个和第四个键。

#keys = sorted([project.key for project in projects])[2:5]

#print keys

#面板

#gisy=jira.dashboards(filter='my', startAt=0, maxResults=20)

gisy=jira.dashboard(11910)

#print gisy.view,'-----' ,dir(gisy)

#print gisy.raw,gisy.\_load,gisy.find

# 获取一个 issue

issue = jira.issue (*'NEWLSXTB-1857'*)

#print issue.fields.summary #获取标题

#print issue.fields.comment.comments.sort #获取备注

print issue.fields.workratio

print issue.fields.watches

print issue.fields.status #问题状态

print issue.fields.progress

#print issue.id,issue.raw #raw显示问题所有信息

#print issue.update,issue.expand , issue.find , issue.key , issue.self

print issue.fields.attachment,issue.fields.assignee ,issue.fields.attachment ,issue.fields.components

#在jira的备注地方添加了备注

#comment = jira.add\_comment ('NEWLSXTB-1857','new comment' )

comments\_a=issue.fields.comment.comments

print comments\_a

comments\_b= jira.comments(issue ) # comments\_b ==comments\_a

#如果你知道 issueID, 可以获取个别评论：

comment = jira.comment(*'NEWLSXTB-1857'*,*'187814'* )

print comment

# 从这个 issue 上获取所有 By Atlassians 的评论。

import re

atl\_comments=[comment for comment in issue.fields.comment.comments]

#if re.search(r'@atlassian.com$' , comment.author.emailAddress)]

print atl\_comments

# 改变 issue 的 summary( 摘要也就是标题 ) and description( 描述也就是需求的开发填写说明 ).

issue.update(summary = "I'mdifferent!" , description = 'Changed the summary to be different.' )

# 找出所有关于 admin 的 issue 的报告（代理人或者是开发者是 admin ）

issues = jira.search\_issues ( *'assignee=wangkaibin'* )

print issues

# 找出前三个（报告人是） admin 的 issue

from collections import Counter

top\_three = Counter([issue.fields.project.name for issue in issues]).most\_common(3)

print top\_three

#利用 JQL 来快速找到你想要的 issues ：

issues\_in\_proj = jira . search\_issues( 'project in (LSXTB, NEWLSXTB) AND issuetype = Bug AND created < 2016-04-01 AND created >= 2016-01-01 AND reporter in (bjguoyurui, lixionghu, zhangjinming, bjduanyingying, liufeng53, bjkeqinqin, bjzhangda, wangkaibin, wanglijiang5, fanghuan3, heyujiao3, liuhua106, zhouxuemei)' )

print issues\_in\_proj,*'总数是'*,issues\_in\_proj.\_\_len\_\_()

print jira.filter(14662)

print issue.fields

#all\_proj\_issues\_but\_mine = jira . search\_issues ( 'project=PROJ andassignee != currentUser()' )

## 我应该在本周完成的 5 个 issue （优先级排序）

#oh\_crap = jira.search\_issues( 'assignee = currentUser() and due < endOfWeek() orderby priority desc' , maxResults = 5 )

## Summaries of my last 3 reported issues 我的最后三个 issue 报告的总结

#print [issue . fields . summary for issue in jira . search\_issues ( 'reporter =currentUser() order by created desc' , maxResults = 3 )]

### hadoop

156.100 autotest

#### 基础

1. Hadoop，Spark和Storm是目前最重要的三大分布式计算系统，Hadoop常用于离线的复杂的大数据处理，Spark常用于离线的快速的大数据处理，而Storm常用于在线的实时的大数据处理。
2. Hadoop采用MapReduce分布式计算框架，并根据GFS开发了HDFS分布式文件系统，根据BigTable开发了HBase数据存储系统。

HDFS默认会将文件分割成block，64M为1个block。然后将block按键值对存储在HDFS上，并将键值对的映射存到内存中。如果小文件太多，那内存的负担会很重。

1. Apache Hadoop系统其实就像一个操作系统。主要包含HDFS －相当于Linux下面的ext3,ext4，和Yarn － 相当于Linux下面的进程调度和内存分配模块。
2. 安装包后缀名区别

hadoop-2.6.0-src.tar.gz是源码压缩文件。可以用eclipse导入研究源码，或者Maven构建编译打包。

hadoop-2.6.0.tar.gz是已经官方发布版压缩包，可以直接使用。不过官网下载的hadoop发布版本只适合x86环境，若要x64的则需要Maven重新构建。

\*.mds 是描述文件，记录压缩包的MD5，SHA1等信息。

1. 查看安装包是32位还是64位

进入目录hadoop-2.7.2/lib/native

使用file命令 file libhadoop.so.1.0.0

显示 libhadoop.so.1.0.0: ELF **64-bit** LSB shared object, x86-64, version 1 (SYSV), dynamically linked, not stripped

可得为64位安装包

1. hadoop2.7只支持jdk1.7及更高版本，不在支持jdk1.6
2. Apache Hadoop的生态系统大部分都是用Java来实现的，也是为Java准备的。python可以用的Hadoop框架:

Hadoop流\mrjob\dumbo\hadoopy\pydoop

1. 单机模式是Hadoop的默认模。当配置文件为空时，Hadoop完全运行在本地。因为不需要与其他节点交互，单机模式就不使用HDFS，也不加载任何Hadoop的守护进程。该模式主要用于开发调试MapReduce程序的应用逻辑。slave配置localhost
2. 伪分布模式. Hadoop守护进程运行在本地机器上，模拟一个小规模的的集群。该模式在单机模式之上增加了代码调试功能，允许你检查内存使用情况，HDFS输入输出，以及其他的守护进程交互。
3. 全分布模式. Hadoop守护进程运行在一个集群上。slave配置文件配置从节点
4. hadoop支持什么开发语言 有2种方式来实现Map/Reduce java的方式

Hadoop Streaming， SHELL/Python/ruby等各种支持 标准输入输出的语言

1. mapper和reducer会从标准输入中读取用户数据，一行一行处理后发送给标准输出。Streaming工具会创建MapReduce作业，发送给各个tasktracker，同时监控整个作业的执行过程。 任何语言，只要是方便接收标准输入输出就可以做mapreduce
2. **HDFS**也是按照Master和Slave的结构。分NameNode、SecondaryNameNode、DataNode这几个角色。

NameNode：是Master节点，是大领导。管理数据块映射；处理客户端的读写请求；配置副本策略；管理HDFS的名称空间；

SecondaryNameNode：是一个小弟，分担大哥namenode的工作量；是NameNode的冷备份；合并fsimage和fsedits然后再发给namenode。

DataNode：Slave节点，奴隶，干活的。负责存储client发来的数据块block；执行数据块的读写操作。

热备份：b是a的热备份，如果a坏掉。那么b马上运行代替a的工作。

冷备份：b是a的冷备份，如果a坏掉。那么b不能马上代替a工作。但是b上存储a的一些信息，减少a坏掉之后的损失。

fsimage:元数据镜像文件（文件系统的目录树。）

edits：元数据的操作日志（针对文件系统做的修改操作记录）

namenode内存中存储的是=fsimage+edits。

SecondaryNameNode负责定时默认1小时，从namenode上，获取fsimage和edits来进行合并，然后再发送给namenode。减少namenode的工作量。

1. Map-Reduce主要包括两个步骤：Map和Reduce

每一步都有key-value对作为输入和输出：

map阶段的key-value对的格式是由输入的格式所决定的，如果是默认的TextInputFormat，则每行作为一个记录进程处理，其中key为此行的开头相对于文件的起始位置，value就是此行的字符文本

map阶段的输出的key-value对的格式必须同reduce阶段的输入key-value对的格式相对应

1. 编写Map-Reduce程序，一般需要实现两个函数：mapper中的map函数和reducer中的reduce函数。
2. 欲配置JobConf，需要大致了解Hadoop运行job的基本原理：

Hadoop将Job分成task进行处理，共两种task：map task和reduce task

Hadoop有两类的节点控制job的运行：JobTracker和TaskTracker

JobTracker协调整个job的运行，将task分配到不同的TaskTracker上

TaskTracker负责运行task，并将结果返回给JobTracker

Hadoop将输入数据分成固定大小的块，我们称之input split

Hadoop为每一个input split创建一个task，在此task中依次处理此split中的一个个记录(record)

Hadoop会尽量让输入数据块所在的DataNode和task所执行的DataNode(每个DataNode上都有一个TaskTracker)为同一个，可以提高运行效率，所以input split的大小也一般是HDFS的block的大小。

Reduce task的输入一般为Map Task的输出，Reduce Task的输出为整个job的输出，保存在HDFS上。

在reduce中，相同key的所有的记录一定会到同一个TaskTracker上面运行，然而不同的key可以在不同的TaskTracker上面运行，我们称之为partition

1. Map-Reduce的处理过程主要涉及以下四个部分：

客户端Client：用于提交Map-reduce任务job

JobTracker：协调整个job的运行，其为一个Java进程，其main class为JobTracker

TaskTracker：运行此job的task，处理input split，其为一个Java进程，其main class为TaskTracker

HDFS：hadoop分布式文件系统，用于在各个进程间共享Job相关的文件

1. 首先看一下Hadoop解决了什么问题，Hadoop就是解决了大数据（大到一台计算机无法进行存储，一台计算机无法在要求的时间内进行处理）的可靠存储和处理。

HDFS，在由普通PC组成的集群上提供高可靠的文件存储，通过将块保存多个副本的办法解决服务器或硬盘坏掉的问题。

MapReduce，通过简单的Mapper和Reducer的抽象提供一个编程模型，可以在一个由几十台上百台的PC组成的不可靠集群上并发地，分布式地处理大量的数据集，而把并发、分布式（如机器间通信）和故障恢复等计算细节隐藏起来。而Mapper和Reducer的抽象，又是各种各样的复杂数据处理都可以分解为的基本元素。这样，复杂的数据处理可以分解为由多个Job（包含一个Mapper和一个Reducer）组成的有向无环图（DAG）,然后每个Mapper和Reducer放到Hadoop集群上执行，就可以得出结果。

#### 更新java版本

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html> java最新安装包

1. 查看并修改使jdk-7u5-linux-i586.rpm具有执行权限(我用root安装，默认有执行权限)

chmod +x jdk-7u5-linux-i586.rpm

1. 卸载旧的版本

1、若之前没有装过，卸载系统自带的jdk版本：

查看自带的jdk：

#rpm -qa|grep gcj

看到如下信息：

libgcj-4.1.2-44.el5

java-1.4.2-gcj-compat-1.4.2.0-40jpp.115

使用rpm -e --nodeps 命令删除上面查找的内容：

#rpm -e –nodeps java-1.4.2-gcj-compat-1.4.2.0-40jpp.115

2、若之前自己装过，卸载rpm安装的jdk版本

查看安装的jdk：

#rpm -qa|grep jdk

看到如下信息：

jdk-1.6.0\_22-fcs

卸载：

#rpm -e --nodeps jdk-1.6.0\_22-fcs

1. 安装jdk

进入安装包所在的文件夹 rpm -ivh jdk-8u73-linux-x64.rpm

1. 增加JAVA\_HOME环境变量

vi /etc/profile

在最后面修改：

export JAVA\_HOME=/usr/java/jdk1.7.0\_05

export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

export PATH=$PATH:$JAVA\_HOME/bin

export JAVA\_HOME CLASSPATH PATH

保存退出

注释：

JAVA\_HOME指明JDK安装路径，就是刚才安装时所选择的路径，此路径下包括lib，bin，jre等文件夹（此变量最好设置，因为以后运行tomcat，Eclipse等都需要依靠此变量）。

PATH使得系统可以在任何路径下识别java命令，设为：%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin 将上边设置的JAVA变量添到PATH这个环境变量里 并且包含$PATH 否则其他PATH被覆盖

CLASSPATH为java加载类(class or lib)路径，只有类在classpath中，java命令才能识别，设为：.:%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar (要加.表示当前路径)

设置 JAVA\_OPTS 环境变量来将额外的选项传递给 Java 虚拟机。

1. 测试安装情况：

新建Test.java

public class Test

{

public static void main(String[] args)

{

System.out.println("Hello,Welcome to Linux World!");

}

}

在终端用cd命令进入Test.java目录,然后输入

javac Test.java

java Test

若输出

Hello,Welcome to Linux World!

则表明配置成功!

1. java -version 查看java的版本

#### 安装hadoop

##### 正常安装流程

<http://my.oschina.net/fengjihu/blog/530869>

1. 在以下文件添加相关内容

etc/hadoop/core-site.xml

hdfs-site.xml

mapred-site.xml.template 修改为 mapred-site.xml

yarn-site.xml

1. 如果是单机模式，不进行本步骤。如果是多级集群模式，则修改etc/hadoop/slaves

删除localhost，把所有的datanode添加到这个文件中。

增加： node1 node2 指的必须是 从服务器的真实的主机名。如果从服务器主机名为gisy，则hosts添加gisy

node1

node2

1. 修改/etc/hosts文件

增加如下内容： 主机名需根据实际进行配置

192.168.63.227 NameNode

192.168.63.202 node1

192.168.63.203 node2

1. 添加/usr/local/hadoop/bin到环境变量
2. 格式化namenode hdfs namenode -format 格式化（namenode节点执行）
3. 启动hadoop ./start-dfs.sh;./start-yarn.sh 停止：./stop-all.sh （namenode节点执行）
4. 如果有文件拒绝执行，则在hadoop的上一级文件夹中执行 chmod -R +x hadoop
5. 在所有节点上执行：hadoop dfsadmin -report
6. 在namenode节点上执行命令：jps

9088 SecondaryNameNode

9254 ResourceManager

1158 jar

7592 GetConf

6538 NodeManager

9644 Jps

8893 NameNode

1. 在datanode节点上执行jps 从节点也要手动启动，以便打开SecondaryNameNode

20098 Jps

22856 Bootstrap

19818 SecondaryNameNode

19326 NodeManager

1. 访问hadoop的web页面：<http://namenode:8088> <http://namenode:50070> HDFS的管理情况
2. 在Hadoop启动以后，Namenode是通过SSH（Secure Shell）来启动和停止各个datanode上的各种守护进程的，这就须要在节点之间执行指令的时候是不须要输入密码的形式，故我们须要配置SSH运用无密码公钥认证的形式。

##### 错误解决

1. Hadoop安装完后，启动时报Error: JAVA\_HOME is not set and could not be found.

解决办法：修改/etc/hadoop/hadoop-env.sh 和 yarn-env.sh 中JAVA\_HOME。

应当使用绝对路径。

export JAVA\_HOME=$JAVA\_HOME //错误，不能这么改

export JAVA\_HOME=/usr/java/jdk1.6.0\_45 //正确，应该这么改

1. 报错Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2. hadoop: command not found 进入hadoop安装目录的bin目录下执行 ./hadoop fs -ls 或者将hadoop安装目录的bin目录配置到path变量中
3. 启动后打不开 192.168.156.100:50070 ，是因为没有在启动前进行格式化namenode

##### 安装时添加内容

注意：NameNode为该机主机名 如果该机主机名为gisy，则NameNode替换为gisy.

<file:///usr/local/hadoop> 指的是该机安装hadoop的路径 如果为其他，替换为其他路径

etc/hadoop/core-site.xml

在<configuration>中间插入

<property>

<name>fs.defaultFS</name>

<value>hdfs://NameNode:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>file:///usr/local/hadoop/tmp</value>

</property>

<property>

<name>io.file.buffer.size</name>

<value>131702</value>

</property>

hdfs-site.xml

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///usr/local/hadoop/hdfs/name</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///usr/local/hadoop/hdfs/data</value>

</property>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.webhdfs.enabled</name>

<value>true</value>

</property>

mapred-site.xml

在<configuration>中间插入：

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

<final>true</final>

</property>

<property>

<name>mapreduce.jobtracker.http.address</name>

<value>NameNode:50030</value>

</property>

<property>

<name>mapreduce.jobhistory.address</name>

<value>NameNode:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>NameNode:19888</value>

</property>

<property>

<name>mapred.job.tracker</name>

<value>http://NameNode:9001</value>

</property>

yarn-site.xml

<property>

<name>yarn.resourcemanager.hostname</name>

<value>NameNode</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>NameNode:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>NameNode:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>NameNode:8031</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address</name>

<value>NameNode:8033</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address</name>

<value>NameNode:8088</value>

</property>

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>2048</value>

</property>

<property>

<name>yarn.nodemanager.resource.cpu-vcores</name>

<value>1</value>

</property>

##### 设置ssh免密码登录

配置ssh免密码登录（三个节点NameNode主节点、stock.ofc.jd.com从节点）

主节点配置：

ssh-keygen -t rsa -P "" (使用rsa加密方式生成密钥)回车后，会提示三次输入信息，我们直接回车即可。

进入文件夹cd /root/.ssh

将生成的公钥id\_rsa.pub 内容追加到authorized\_keys（执行命令：cat id\_rsa.pub >> authorized\_keys）

从节点配置：

以同样的方式生成秘钥ssh-keygen -t rsa -P ""

然后s1将生成的id\_rsa.pub公钥追加到m1的authorized\_keys中。直接将内容复制到authorized\_keys文件的尾部即可

最后将生成的包含三个节点的秘钥的authorized\_keys 复制到stock.ofc.jd.com的.ssh目录下

必须要保证各个节点的相关文件的权限，不能是777：

~/.ssh需要是700权限

authorized\_keys需要是644权限

chmod -R 700 ~/.ssh

chmod 644 ~/.ssh/authorized\_keys

验证ssh免密码登录

1. 输入命令ssh NameNode(主机名)，或者ssh localhost (验证本机) 如果配置成功，直接显示

Last login: Wed Mar 2 16:03:23 2016 from localhost

1. 在主节点输入ssh stock.ofc.jd.com(从节点主机名) ，则命令行console进入从节点服务器.

##### Hadoop中HDFS常用命令

hadoop fs -mkdir /tmp/input 在HDFS上新建文件夹

hadoop fs -put input1.txt /tmp/input 把本地文件input1.txt传到HDFS的/tmp/input目录下

hadoop fs -get input1.txt /tmp/input/input1.txt 把HDFS文件拉到本地

hadoop fs -ls /tmp/output 列出HDFS的某目录

hadoop fs -cat /tmp/ouput/output1.txt 查看HDFS上的文件

hadoop fs -rmr /home/less/hadoop/tmp/output 删除HDFS上的目录

hadoop dfsadmin -report 查看HDFS状态，比如有哪些datanode，每个datanode的情况

hadoop dfsadmin -safemode leave 离开安全模式

hadoop dfsadmin -safemode enter 进入安全模式

**简化hadoop命令**，在用户根目录下 vi .bashrc

在最后添加：

alias fs='hadoop fs'

alias fsa='hadoop dfsadmin'

执行source .bashrc即可生效

##### 安装ant

Error: Could not find or load main class org.apache.tools.ant.launch.Launcher

错误原因：错误很明显没有找到org.apache.tools.ant.launch.Launcher这个类

解决方法：修改profile文件中的CLASSPATH，将ant-launcher.jar的完整路径加进去

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib: /usr/share/ant/lib/ant-launcher.jar

注意不要添加/usr/share/ant/lib 为PATH环境变量，否则依旧报错

### zookeeper

最近刚好在研究zookeeper，下面综合考虑了现有的常见python客户端，讨论一下：

zkpython

它是比较早的一个客户端，它依赖zookeeper的C语言库，因此它必须是在linux下使用。

这里有几篇关于它的使用方法的文章，大家可以感受一下：

在python中使用zookeeper管理你的应用集群: http://www.zlovezl.cn/articles/40/

zookeeper客户端zkpython使用文档(一)<http://justfansty.blog.sohu.com/217953183.htmlhttp://justfansty.blog.sohu.com/218331818.html>

zklock

zklock是使用zkpython来完成的一个分布式锁实现。由于它依赖了zkpython，所以它也必须是在linux下才能运行。

关于分布式的锁服务，其实也可以采用 memcached\_lock 。它的好处在于它可以在windows下运行。

zc.zk

它是zookeeper官方认定的一个python实现（可见：https://cwiki.apache.org/ZOOKEEPER/zkclientbindings.html）。必须使用zookeeper的C库或者安装 zc-zookeeper-static 3.4.4 ，然后才能安装它。因此，它也必须是在linux下才能运行的。

kazoo

kazoo也是zookeeper官方认定的一个强大的纯python实现（为什么说是纯呢？就是因为它无需要依赖zookeeper的C库）。它不仅有zookeeper的所有原语实现，并且，他还实现了 分布式锁等高级语义。

由于作者认为zookeeper都是在linux下运行的，所以kazoo不支持windows。

因此，可见，目前所有zookeeper python客户端都只能支持linux下运行的。

ZooKeeper是Hadoop和Hbase的重要组件。

#### 安装

1. 进入Zookeeper 的启动脚本在 bin 目录下，Linux 下的启动脚本是 zkServer.sh。
2. conf 目录下**zoo\_sample.cfg 改名为 zoo.cfg,** 因为 Zookeeper 在启动时会找这个文件作为默认配置文件。
3. 进入Zookeeper 的启动脚本在 bin 目录下

cd /usr/local/etc/zookeeper-3.4.8/bin **chmod +x ./zkServer.sh** ./zkServer.sh start

1. 将/usr/local/etc/zookeeper-3.4.8/bin添加到环境变量path中。这样可以不用进入bin文件目录下，就能执行命令

比如直接执行 zkCli.sh -server 192.168.153.122:218

#### 配置文件中各个配置项

tickTime=2000

tickTime：这个时间是作为 Zookeeper 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳。

dataDir=D:/devtools/zookeeper-3.2.2/build

dataDir：顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里。

clientPort=2181

clientPort：这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。

#### zk的日常命令

1. 查看服务器状态 ./zkServer.sh status

Using config: /usr/local/etc/zookeeper-3.4.8/bin/../conf/zoo.cfg

Mode: standalone

1. 查看命令帮助 ./zkServer.sh help

Using config: /usr/local/etc/zookeeper-3.4.8/bin/../conf/zoo.cfg

Usage: ./zkServer.sh {start|start-foreground|stop|restart|status|upgrade|print-cmd}

1. 打开客户端 ./zkCli.sh
2. 打开客户端并连接另一台服务器的zookeeper服务端 ./zkCli.sh -server 192.168.153.122:2181
3. zk客户端命令 命令行工具的一些简单操作如下：
   1. 显示根目录下、文件： ls / 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容
   2. 显示根目录下、文件： ls2 / 查看当前节点数据并能看到更新次数等数据
   3. 创建文件，并设置初始内容： create /zk "test" 创建一个新的 znode节点“ zk ”以及与它关联的字符串
   4. 获取文件内容： get /zk 确认 znode 是否包含我们所创建的字符串

比如获取zookeeper的某PATH信息 get /config/transfer-java/ots-mainConfig/development

* 1. 修改文件内容： set /zk "zkbak" 对 zk 所关联的字符串进行设置
  2. 删除文件： delete /zk 将刚才创建的 znode 删除
  3. 退出客户端： quit
  4. 帮助命令： help

#### ZooKeeper 常用四字命令

直接在linux的console输入 这些命令 不是在zk的客户端命令行上。

ZooKeeper 支持某些特定的四字命令字母与其的交互。它们大多是查询命令，用来获取 ZooKeeper 服务的当前状态及相关信息。用户在客户端可以通过 telnet 或 nc 向 ZooKeeper 提交相应的命令

1. 可以通过命令：echo stat|nc 127.0.0.1 2181 来查看哪个节点被选择作为follower或者leader

2. 使用echo ruok|nc 127.0.0.1 2181 测试是否启动了该Server，若回复imok表示已经启动。

3. echo dump| nc 127.0.0.1 2181 ,列出未经处理的会话和临时节点。

4. echo kill | nc 127.0.0.1 2181 ,关掉server

5. echo conf | nc 127.0.0.1 2181 ,输出相关服务配置的详细信息。

6. echo cons | nc 127.0.0.1 2181 ,列出所有连接到服务器的客户端的完全的连接 / 会话的详细信息。

7. echo envi |nc 127.0.0.1 2181 ,输出关于服务环境的详细信息（区别于 conf 命令）。

8. echo reqs | nc 127.0.0.1 2181 ,列出未经处理的请求。

9. echo wchs | nc 127.0.0.1 2181 ,列出服务器 watch 的详细信息。

10. echo wchc | nc 127.0.0.1 2181 ,通过 session 列出服务器 watch 的详细信息，它的输出是一个与 watch 相关的会话的列表。

11. echo wchp | nc 127.0.0.1 2181 ,通过路径列出服务器 watch 的详细信息。它输出一个与 session 相关的路径。

#### 问题解决

1. echo ruok|nc localhost 2181 时发现命令无法被识别 -bash: nc: command not found

安装方法：

下载后得到一个rpm包，如：netcat-0.7.1-1.i386.rpm

进入该目录，执行命令：rpm -ihv netcat-0.7.1-1.i386.rpm ，进行安装

安装成功后可以执行命令： nc --help

1. rpm包安装过程中依赖问题“libc.so.6 is needed by XXX”解决方法

yum list glibc\* 搜索后发现，本机上有些库已经安装了。比如libc.so.6，该库对应的软件包名称为glibc。

从Red Hat Enterprise Linux 6开始，默认只安装主架构所需要的包，而不安装兼容架构的包，也就是说，64位系统默认只安装64位的软件包。为避免这种问题，在64位系统中，要同时安装64位的包和32位的兼容包。

所以安装32位兼容包 yum install glibc.i686

完成

1. 安装telnet

yum list telnet\* Available Packages telnet.i686 telnet-server.i686

telnet.i686 表示客户端， telnet-server.i686 表示服务端。我们需要的是客户端，安装 telnet.i686 即可。

yum install telnet.i686

#### zookeeper 的测试配置

{

//'zkConfigs': '10.10.225.38:2182,10.10.225.38:2181,10.10.225.38:2183',

'zkConfigs': '192.168.153.121:2181,192.168.153.122:2181,192.168.153.123:2181',

'zkCheckInterval': 30000,

'environment':'development',

'localPath':'/export/Domains/redis.worker.ws.jd.com/server1/logs/development'

}

192.168.153.111 zkconfigs.360buy.com

### cookie处理

1.Opener

当你获取一个URL你使用一个opener(一个urllib2.OpenerDirector的实例)。在前面，我们都是使用的默认的opener，也就是urlopen。它是一个特殊的opener，可以理解成opener的一个特殊实例，传入的参数仅仅是url，data，timeout。

如果我们需要用到Cookie，只用这个opener是不能达到目的的，所以我们需要创建更一般的opener来实现对Cookie的设置。

2.Cookielib

cookielib模块的主要作用是提供可存储cookie的对象，以便于与urllib2模块配合使用来访问Internet资源。Cookielib模块非常强大，我们可以利用本模块的CookieJar类的对象来捕获cookie并在后续连接请求时重新发送，比如可以实现模拟登录功能。该模块主要的对象有CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。

它们的关系：CookieJar —-派生—->FileCookieJar —-派生—–>MozillaCookieJar和LWPCookieJar

#### 获取Cookie保存到变量

首先，我们先利用CookieJar对象实现获取cookie的功能，存储到变量中

import urllib2

import cookielib

#声明一个CookieJar对象实例来保存cookie

cookie = cookielib.CookieJar()

#利用urllib2库的HTTPCookieProcessor对象来创建cookie处理器

handler=urllib2.HTTPCookieProcessor(cookie)

#通过handler来构建opener

opener = urllib2.build\_opener(handler)

#此处的open方法同urllib2的urlopen方法，也可以传入request

response = opener.open('http://www.baidu.com')

for item in cookie:

print 'Name = '+item.name

print 'Value = '+item.value

#### 保存Cookie到文件

在上面的方法中，我们将cookie保存到了cookie这个变量中，如果我们想将cookie保存到文件中该怎么做呢？这时，我们就要用到FileCookieJar这个对象了，在这里我们使用它的子类MozillaCookieJar来实现Cookie的保存

import cookielib

import urllib2

#设置保存cookie的文件，同级目录下的cookie.txt

filename = 'cookie.txt'

#声明一个MozillaCookieJar对象实例来保存cookie，之后写入文件

cookie = cookielib.MozillaCookieJar(filename)

#利用urllib2库的HTTPCookieProcessor对象来创建cookie处理器

handler = urllib2.HTTPCookieProcessor(cookie)

#通过handler来构建opener

opener = urllib2.build\_opener(handler)

#创建一个请求，原理同urllib2的urlopen

response = opener.open("http://www.baidu.com")

#保存cookie到文件

cookie.save(ignore\_discard=True, ignore\_expires=True)

#### 从文件中获取Cookie并访问

那么我们已经做到把Cookie保存到文件中了，如果以后想使用，可以利用下面的方法来读取cookie并访问网站，感受一下

import cookielib

import urllib2

#创建MozillaCookieJar实例对象

cookie = cookielib.MozillaCookieJar()

#从文件中读取cookie内容到变量

cookie.load('cookie.txt', ignore\_discard=True, ignore\_expires=True)

#创建请求的request

req = urllib2.Request("http://www.baidu.com")

#利用urllib2的build\_opener方法创建一个opener

opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookie))

response = opener.open(req)

print response.read()

#### 例子

##### 登录jira

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

import urllib2

import urllib

import cookielib

filename=*'cookie.txt'*#声明一个MozillaCookieJar对象实例来保存cookie，之后写入文件

cookie=cookielib.MozillaCookieJar(filename)

opener=urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookie))

postdata=urllib.urlencode(

{'os\_username':'wangkaibin' ,

'os\_password':'Gisywang.8261'})

loginUrl='http://jira.jd.com/login.jsp'

#模拟登录，并把cookie保存到变量

result=opener.open(loginUrl,postdata)

cookie.save(ignore\_discard=True,ignore\_expires=True)

gradeUrl='http://jira.jd.com/browse/LSXTB-6497'

#请求访问成绩查询网址

result=opener.open(gradeUrl)

print result.read()

### NetworkX 网络拓扑计算

NetworkX提供了4种常见网络的建模方法，分别是：规则图，ER随机图，WS小世界网络和BA无标度网络。

规则图差不多是最没有复杂性的一类图，random\_graphs.random\_regular\_graph(d, n)方法可以生成一个含有n个节点，每个节点有d个邻居节点的规则图。

下面一段示例代码，生成了包含20个节点、每个节点有3个邻居的规则图：

import networkx as nx

import matplotlib.pyplot as plt

# regular graphy

# generate a regular graph which has 20 nodes & each node has 3 neghbour nodes.

RG = nx.random\_graphs.random\_regular\_graph(3, 20)

# the spectral layout

pos = nx.spectral\_layout(RG)

# draw the regular graphy

nx.draw(RG, pos, with\_labels = False, node\_size = 30)

plt.show()

### watchdog --监控文件系统变化

watchdog用来监控指定目录/文件的变化，如添加删除文件或目录、修改文件内容、重命名文件或目录等，每种变化都会产生一个事件，且有一个特定的事件类与之对应，然后再通过事件处理类来处理对应的事件，怎么样处理事件完全可以自定义，只需继承事件处理类的基类并重写对应实例方法。

使用LoggingEventHandler()事件处理器，当指定目录发生任何变化都会打印消息到终端。

import sys

import time

import logging

from watchdog.observers import Observer

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=Observer()

observer.schedule(event\_handler, path, recursive=True)

observer.start()

try:

while True:

time.sleep(1)

except KeyboardInterrupt:

observer.stop()

observer.join()

### pyhook模块 监控鼠标和键盘

PyHook是一个基于Python的“钩子”库，主要用于监听当前电脑上鼠标和键盘的事件。这个库依赖于另一个Python库PyWin32，**PyWin32只能运行在Windows平台，所以PyHook也只能运行在Windows平台。**

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

import pythoncom

import pyHook

def onMouseEvent(event):

# 监听鼠标事件

print "MessageName:",event.MessageName

print "Message:", event.Message

print "Time:", event.Time

print "Window:", event.Window

print "WindowName:", event.WindowName

print "Position:", event.Position

print "Wheel:", event.Wheel

print "Injected:", event.Injected

print"---"

# 返回 True 以便将事件传给其它处理程序

# 注意，这儿如果返回 False ，则鼠标事件将被全部拦截

# 也就是说你的鼠标看起来会僵在那儿，似乎失去响应了

return True

def onKeyboardEvent(event):

# 监听键盘事件

print "MessageName:", event.MessageName

print "Message:", event.Message

print "Time:", event.Time

print "Window:", event.Window

print "WindowName:", event.WindowName

print "Ascii:", event.Ascii, chr(event.Ascii)

print "Key:", event.Key

print "KeyID:", event.KeyID

print "ScanCode:", event.ScanCode

print "Extended:", event.Extended

print "Injected:", event.Injected

print "Alt", event.Alt

print "Transition", event.Transition

print "---"

# 同鼠标事件监听函数的返回值

return True

def main():

# 创建一个“钩子”管理对象

hm = pyHook.HookManager()

# 监听所有键盘事件

hm.KeyDown = onKeyboardEvent

# 设置键盘“钩子”

hm.HookKeyboard()

# 监听所有鼠标事件

hm.MouseAll = onMouseEvent

# 设置鼠标“钩子”

hm.HookMouse()

# 进入循环，如不手动关闭，程序将一直处于监听状态

pythoncom.PumpMessages()

if \_\_name\_\_ == "\_\_main\_\_":

main()

### PyUserInput 控制鼠标和键盘的模块

在安装完PyUserInput后，**pymouse**和**pykeyboard**模块就被安装到你的Python路径下。

#建立一个鼠标和键盘对象：

from pymouse import PyMouse

from pykeyboard import PyKeyboard

m = PyMouse()

k = PyKeyboard()

#接下来是一个示例，完成点击屏幕中央并键入“Hello, World!”的功能：

x\_dim, y\_dim = m.screen\_size()

m.click(x\_dim/2, y\_dim/2, 1)

k.type\_string('Hello, World!')

#PyKeyboard还有很多种方式来发送键盘键入：

# pressing a key

k.press\_key('H')

# which you then follow with a release of the key

k.release\_key('H')

# or you can 'tap' a key which does both

k.tap\_key('e')

# note that that tap\_key does support a way of repeating keystrokes with a interval time between each

k.tap\_key('l',n=2,interval=5)

# and you can send a string if needed too

k.type\_string('o World!')

#并且它还支持很多特殊按键：

#Create an Alt+Tab combo

k.press\_key(k.alt\_key)

k.tap\_key(k.tab\_key)

k.release\_key(k.alt\_key)

k.tap\_key(k.function\_keys[5]) # Tap F5

k.tap\_key(k.numpad\_keys['Home']) # Tap 'Home' on the numpad

k.tap\_key(k.numpad\_keys[5], n=3) # Tap 5 on the numpad, thrice

#注意，你也可以使用press\_keys方法将多个键一起发送（例如，使用某些组合键）：

# Mac example

k.press\_keys(['Command','shift','3'])

# Windows example

k.press\_keys([k.windows\_l\_key,'d'])

### xhtml2pdf模块 html生成pdf

import xhtml2pdf.pisa as pisa

pdf = pisa.CreatePDF(open('templates/fenbiao.html','rb'),open('test.pdf','wb'))

if not pdf.err:

print "pdf is build"

pisa.startViewer('test.pdf')

### functools

#### functools.partial 调用函数的参数固定

作用就是，把一个函数的某些参数给固定住（也就是设置默认值），返回一个新的函数，调用这个新函数会更简单。

创建偏函数时，实际上可以接收函数对象、\*args和\*\*kw这3个参数，当传入：

int2 = functools.partial(int, base=2)

实际上固定了int()函数的关键字参数base，也就是：

int2('10010')

相当于：

kw = { base: 2 }

int('10010', \*\*kw)

当传入：

max2 = functools.partial(max, 10)

实际上会把10作为\*args的一部分自动加到左边

相当于：

args = (10, 5, 6, 7)

max(\*args)

结果为10。

注意到上面的新的int2函数，仅仅是把base参数重新设定默认值为2，但也可以在函数调用时传入其他值：

>>> int2('1000000', base=10)

1000000

#### functools.update\_wrapper

被封装的函数的 module, name, doc 和 dict 复制到封装的函数中去

update\_wrapper(living, fun) fun的属性赋予living

#### functools.wraps

wraps() 函数把用 partial() 把 update\_wrapper() 给封装了一下

from functools import wraps

def my\_decorator(f):

*@wraps*(f) #把f的属性赋予wrapper

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

print 'Calling decorated function'

return f(\*args,\*\*kwds)

return wrapper

@my\_decorator

def **example**():

print 'Called example function'

example()

print example.\_\_name\_\_

print example.\_\_doc\_\_

### jPype

#### 基础

JPype是一个能够让 Python 代码方便地调用 Java 代码的工具，从而克服了 python 在某些领域（如服务器端编程）中的不足。

Pype与Jython(JPython后继者)的区别？

1）运行环境不同：jython运行在jvm上，而JPype的实际运行环境仍然是python runtime，只是在运行期间启动了一个嵌入的jvm；

2）使用者不同：jython是给java程序玩的，JPype是给python程序员玩的。

1.启动JVM

JPype 提供的 startJVM() 函数的作用是启动 JAVA 虚拟机，所以在后续的任何 JAVA 代码被调用前，必须先调用此方法启动 JAVA 虚拟机。

jpype.startJVM() 的定义

startJVM(jvm, \*args)

jpype.startJVM() 的参数

参数 1： jvm, 描述你系统中 jvm.dll 文件所在的路径，如“ C:\Program Files\IBM\Java50\jre\bin\j9vm\jvm.dll ”。可以通过调用 jpype.getDefaultJVMPath() 得到默认的 JVM 路径。

参数 2： args, 为可选参数，会被 JPype 直接传递给 JVM 作为 Java 虚拟机的启动参数。此处适合所有合法的 JVM 启动参数，例如：

-agentlib:libname[=options]

-classpath classpath

-verbose

-Xint

2.关闭JVM

当使用完 JVM 后，可以通过 **jpype.shutdownJVM()** 来关闭 JVM，该函数没有输入参数。当 python 程序退出时，JVM 会自动关闭。

3.引用第三方Java扩展包

很多时候，在 python 项目中需要调用第三方的 Java 扩展包，这也是 JPype 的一个重要用途。

通过在 JVM 启动参数增加：-Djava.class.path=ext\_classpath，实现在 python 代码中调用已有的 Java 扩展包。

4.访问JAVA的系统属性

有时，某些 Java 应用需要设置或者获取 JVM 中的系统属性。

在 JVM 启动时设置系统变量示例：

在 JVM 的启动参数中加入如下参数：

-Dproperty=value

#### 举例

1. 直接调用java api

import jpype

jpype.startJVM("C:\\Program Files (x86)\\Java\\jre1.8.0\_91\\bin\\client\\jvm.dll", "-ea")

jpype.java.lang.System.out.println("hello World")

jpype.shutdownJVM()

#### java基础

Java源程序的后缀，编译后的文件后缀，application程序打包后的文件后缀分别是\*.java \*.class \*.jar

1、创建 jar 包

jar cf HelloWord.jar Hello.java

2、创建 jar 包（显示过程： v 参数）

jar cvf HelloWord.jar Hello.java

3、列出 jar 包的内容

jar tvf HelloWord.jar

4、解压 jar 包 [windows平台上的解压软件(WinRAR)也行]

jar xf HelloWord.jar

4、解压 jar 包 [windows平台上的解压软件(WinRAR)也行;显示过程]

jar xvf HelloWord.jar

5、添加文件到 jar 包

jar uf HelloWord.jar Hello123.java

# flask框架

## 启动

1. 启动flask项目 cd /usr/local/games/wikidoc\_flask python wikidoc\_flask.py
2. 后台启动 nohup python wikidoc\_flask.py &
3. 清除linux中已存在的端口。即关闭已存在端口对应的服务。端口和服务是对应的，如果要关闭端口就要停止掉对应服务。

查找端口对应的服务pid号。 netstat -anlp| grep 端口号 **netstat -anlp| grep 5000**

在上述命令的最后一列会给出端口对应应用的pid号 **kill -9 pid** 即可关闭服务，并关掉端口。

## 安装

1. 安装flask： pip install flask

及其他扩展

pip install flask-login

pip install flask-openid

pip install flask-mail

pip install flask-sqlalchemy

pip install sqlalchemy-migrate

pip install flask-whooshalchemy

pip install flask-wtf

pip install flask-babel

pip install guess\_language

pip install flipflop

pip install coverage

1. 每次修改完代码后，直接保存。如果flask之前启动，则会自动重启，应用修改过的代码。

@app.route('/post/<int:post\_id>')

def show\_post(post\_id):

# show the post with the given id, the id is an integer

return 'Post %d' % post\_id

## 简单例子

创建flask项目：

1. eclipse中新建python项目。在python项目文件夹中新建hello.py
2. 在python项目文件夹中新建templates文件夹。templates文件夹下放置html文件,比如hello.html
3. 在函数前加上域名调用行，[如@app.route('/hello/')](mailto:如@app.route('/hello/'))
4. py文件最后是ip port 设置

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

app.run(host="127.0.0.1",port=5000, debug=True)

运行flask项目：

直接在py文件右键运行 python run 而不是在项目名上运行

hello.py

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

from flask import Flask,render\_template

app= Flask(\_\_name\_\_)

@app.route('/hello/')

@app.route('/hello/<name>')

def **hello**(name=None):

if name == None:

name = *"gisy256"*

templateDate = {*'name'* : name};

return render\_template(*"hello.html"*, \*\*templateDate);

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

app.run(host=*"127.0.0.1"*,port=5000, debug=True)

hello.html

<html>

<head>

<title>Flask Test</title>

</head>

<body>

<h2>Hello {{name}}</h2>

</body>

</html>

剖析上述代码：

　　　　1. 首先，我们导入了类 Flask 。这个类的实例化将会是我们的 WSGI 应用。第一个参数是应用模块的名称。 如果你使用的是单一的模块（就如本例），第一个参数应该使用 \_\_name\_\_。因为取决于如果它以单独应用启动或作为模块导入， 名称将会不同 （ '\_\_main\_\_' 对应于实际导入的名称）。

　　　　2. 接着，我们创建一个该类的实例。我们传递给它模块或包的名称。这样 Flask 才会知道去哪里寻找模板、静态文件等等。

　　　　3. 之后，我们使用装饰器 route() 告诉 Flask 哪个 URL 才能触发我们的函数。这也就是路由，博文之后会详细学习。

　　　　4. 之后 ，定义一个函数，该函数名也是用来给特定函数生成 URLs，并且返回我们想要显示在用户浏览器上的信息。

　　　　5. 最后，我们用函数 run() 启动本地服务器来运行我们的应用。if \_\_name\_\_ == '\_\_main\_\_': 确保服务器只会在该脚本被 Python 解释器直接执行的时候才会运行，而不是作为模块导入的时候。

## http方法 post

@app.route("/postUser", methods=["GET","POST"])

def hello\_user():

if request.method == 'POST':

return "Post User"

else:

return request.args.get('abc')

@app.route('/ postUser, methods=['POST'])是指url/postUser只接受POST方法。如果get访问就会报错：The method is not allowed for the requested URL.

也可以根据需要修改methods参数，例如@app.route('/register', methods=['GET', 'POST']) # 接受GET和POST方法

报错ValueError: View function did not return a response，是没有返回值。比如没有

return render\_template("flask.html",content=c)

表单提交

search=request.form['search'] 'search'取得是name名

<label for="keyword">搜索文档标题</label>

<input type="search" style='margin-right:50px' id='keyword' **name='search'** />

Request 参数

需要在开头添加 from flask import Flask, request 来进行request参数的传值

@app.route("/query\_user")

def query\_user():

userId = **request.args.get**("id")

return "query user: {0}".format(userId)

post方式使用下面这个获取参数

orderid=request.form['orderid'] 取得是html中name值

有两个提交submit按钮

<form action="{{ url\_for('index') }}" method="post">

<input type="submit" name="add" value="Like">

<input type="submit" name="remove" value="Dislike">

</form>

Use request.form's .get method:

if request.form.get('add', None) == "Like":

# Like happened

elif request.form.get('remove', None) == "Dislike":

# Dislike happened

两个submit按钮使用相同的name值

<input type="submit" name="action" value="Like">

<input type="submit" name="action" value="Dislike">

# and in your code

if request.form["action"] == "Like":

# etc.

两个form表单提交

<form action="" method="post">

<input type="submit" name='action' value="查询" class='button' />

</form>

<form action="" method="post">

<input type="submit" name='action' value="检索wiki平台文档" class='button' />

</form>

if request.method=='POST':

if request.form["action"] == "查询":

return render\_template('wikidoc.html',result=wikidoc,num=wikidoc\_count,writer=writer,search=search,startdate=startdate,enddate=enddate)

else:

return render\_template('wikidoc.html')

## 模板

Jinja2模板

### 模板用法

For

遍历序列中的每项。例如，要显示一个由 users` 变量提供的用户列表:

<h1>Members</h1>

<ul>

{% for user in users %}

<li>{{ user.username|e }}</li>

{% endfor %}

</ul>

因为模板中的变量保留它们的对象属性，可以迭代像 dict 的容器:

<dl>

{% for key, value in my\_dict.iteritems() %}

<dt>{{ key|e }}</dt>

<dd>{{ value|e }}</dd>

{% endfor %}

</dl>

{% if %}标签接受and，or或者not来测试多个变量值或者否定一个给定的变量，{% if athlete\_list and coach\_list %}

此外，你也可以手动剥离模板中的空白。当你在块（比如一个 for 标签、一段注释或变 量表达式）的开始或结束放置一个减号（ - ），可以移除块前或块后的空白

{% for item in seq -%}

{{ item }}

{%- endfor %}

这会产出中间不带空白的所有元素。如果 seq 是 1 到 9 的数字的列表， 输出会是 123456789 。

**模板%与括号}之间不能有空格**{%elif startdate and enddate% } 否则报错TemplateSyntaxError: unexpected '}'

### 模板实例

<!doctype html>

<title>Hello from Flask</title>

{% if name %}

<h1>Hello {{ name }}!</h1>

{% else %}

<h1>Hello World!</h1>

{% endif %}

<title>{% block title %}{% endblock %}</title>

<ul>

{% for user in users %}

<li><a href="{{ user.url }}">{{ user.username }}</a></li>

{% endfor %}

</ul>

**调用css** 直接在html 模板中引用该路径，如下：

<link href="/static/css/demo.css" rel="stylesheet" type="text/css" />

新建一 flask.html ， 同样放置于 templates 文件夹中，body 内容参考如下：

<body>

<h1>{{ content }}</h1>

</body>

同上例，新建一路由，代码如下：

@app.route("/flask")

def template2():

c = "flask"

return render\_template("flask.html",content=c)

### 传递多个参数给模版（template）

可以向模板（template）传递多个参数或者把全部的本地参数传递给template：

1. 传递多个参数给template，直接将参数放在render\_template()函数里面，参数间用逗号隔开：

@app.route('/')

def index():

content = '.....'

user='Micheal'

return render\_template('index.html', var1=content, var2=user)

template中可以直接使用{{var1}}和{{var2}}直接操作变量。

2. 传递全部的本地变量给template，使用\*\*locals()：

@app.route('/')

def index():

content = '.....'

user='Micheal'

return render\_template('index.html', **\*\*locals()**)

template中可以直接使用{{content}}和{{user}}直接操作变量。

### css

那在flask中，如何处理静态文件？ 做法很简单，只要在项目下创建一个名为 static 的文件夹，与templates互为兄弟文件夹。

在static文件夹下再创建css文件夹。css文件夹下存放css文件，比如demo.css

　你可以直接在html 模板中引用该路径，如下：

<link href="/static/css/demo.css" rel="stylesheet" type="text/css" />

url\_for 构造路径

当然，也可使用 url\_for 构造，代码如 url\_for("static",filename="css/demo.css")

在html模板中使用<link href="{{demoCSS}} " rel="stylesheet" type="text/css" />

在py文件中：

@app.route("/")

def template1():

demoCSS=url\_for("static",filename="css/demo.css")

print demoCSS

return render\_template("index.html",demoCSS=demoCSS)

导入js css

app = Flask(\_\_name\_\_, static\_url\_path='/static') 默认就是/static,可以省略

<link href="{{ url\_for('static', filename='css/bootstrap.min.css') }}" rel="stylesheet">

<script src="{{ url\_for('static', filename='js/bootstrap.min.js') }}"></script>

### html

**多选框** 单选框 选中为on 不选中为null label for 对应的是input的id

<label for="name">按writer排序</label>

<input type="checkbox" name="name" id="nameorder" />

如果value有值为'test'，则选中获得该值'test'

<input type="radio" name="nameorder" id="nameorder" value="test" />

<span style='margin-left:40px'>

<input type="radio" id="nameorder" class="" name='order' value='nameorder' />

<label for="nameorder">按作者排序</label>

<input type="radio" id="dateorder" class="" name='order' value='dateorder' checked='checked'/>

<label for="dateorder">按日期排序</label>

</span>

如何处理多选框checkbox 没被选中？checkboxField 为name

if request.form.get("checkboxField#N", False):

# Do something because the box is checked

比如：

fieldsort='date'

if request.form.get('order',False):

fieldsort='writer'

## 配置文件

### 基础

写配置项时，注意必须全部大写，否则无法读取该配置项。报错KeyError

如 **URLGET**="cd D:\eclipse\_workspace\wikidoc\_url\ &&scrapy crawl wikiurl"

需要在你的根目录下放置一个config.py文件，并在app.py或yourapp/\_\_init\_\_.py中加载它。

比如config.py下编写

DEBUG = True # 启动Flask的Debug模式

BCRYPT\_LEVEL = 13 # 配置Flask-Bcrypt拓展

MAIL\_FROM\_EMAIL = "robert@example.com" # 设置邮件来源

config.py的每一行中应该是某一个变量的赋值语句。一旦config.py在稍后被加载，这个配置变量可以通过app.config字典来获取，比如app.config["DEBUG"]。

app.logger.debug(app.config["BCRYPT\_LEVEL"])

#如果config没有该节点，会报错KeyError

第2个配置-instance文件

可以设置第2个配置文件。instance文件夹可以帮助我们隐藏不愿为人所知的配置变量。Flask提供了一个叫instance文件夹的特性。 instance文件夹是根目录的一个子文件夹，包括了一个特定于当前应用实例的配置文件。

要想加载定义在instance文件夹中的配置变量，你可以使用app.config.from\_pyfile()

如果在调用Flask()创建应用时设置了**instance\_relative\_config=True**，app.config.from\_pyfile()将查看在instance文件夹的特殊文件。如果不设置，则app.config.from\_pyfile()将找不到instance下的配置文件。

app = Flask(\_\_name\_\_,instance\_relative\_config=True)

app.config.from\_object('config') #主配置

app.config.from\_pyfile('config.py') #第2个配置

这样如果在第2个配置和主配置都有节点'DEBUG',则最后调用的配置的该节点值会覆盖之前调用的配置节点值，也就是第2个配置的节点起作用。如果只有1个配置有该节点，则该节点使用该配置的节点值。

app.logger.debug(app.config["BCRYPT\_LEVEL"])

### 更新配置项

导入配置项

app= Flask(\_\_name\_\_)

app.config.from\_object('config')

使用配置项

app.config["DEBUG"]

app.logger.debug(app.config["BCRYPT\_LEVEL"])

你可以使用 dict.update() 方法来一次性更新多个键:

app.config.update(

DEBUG=True,

SECRET\_KEY='...'

)

## 日志

### 显示在eclipse的console上

flask框架下的print是无法显示在console上的

app.logger.debug(orderid)

app.logger.debug('A value for debugging')

app.logger.warning('A warning occurred (%d apples)', 42)

app.logger.error('An error occurred')

### 将日志写入文件中

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

handler=logging.FileHandler('flask.log') #即创建一个log handler，比较常用的有FileHandler和RotatingHandler

app.logger.addHandler(handler) #即将此handler加入到此app中

app.run(host="0.0.0.0",port=5000, debug=True)

例子

from flask import Flask

import logging

app = Flask(\_\_name\_\_)

@app.route("/fenbiao",methods=["GET","POST"])

def fenbiao():

if request.method=='POST':

orderid=request.form['orderid']

remainder=int(orderid) % 32

app.logger.debug(["订单号%s的后缀是%d",**orderid,remainder])**

return render\_template('fenbiao.html',orderid=orderid,remainder=remainder)

else:

return render\_template('fenbiao.html')

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

app.debug = True

handler = logging.FileHandler('flask.log', encoding='UTF-8')

handler.setLevel(logging.DEBUG)

logging\_format = logging.Formatter(

'%(asctime)s %(levelname)s %(filename)s (%(funcName)s:%(lineno)s) - %(message)s' )

handler.setFormatter(logging\_format)

app.logger.addHandler(handler)

app.run()

logging的级别主要有NOTSET、DEBUG、INFO、WARNING、ERROR和CRITICAL

显示两个以上变量，请使用list格式，如 app.logger.debug([startdate,type(startdate)])

FileHandler - 在文件系统上记录日志

RotatingFileHandler - 在文件系统上记录日志， 并且当消息达到一定数目时，会滚动记录

NTEventLogHandler - 记录到 Windows 系统中的系统事件日志。如果你在 Windows 上做开发，这就是你想要用的。

SysLogHandler - 发送日志到 Unix 的系统日志

复杂日志格式

格式 描述

%(levelname)s 消息文本的记录等级 ('DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL').

%(pathname)s 发起日志记录调用的源文件的完整路径（如果可用）

%(filename)s 路径中的文件名部分

%(module)s 模块（文件名的名称部分）

%(funcName)s 包含日志调用的函数名

%(lineno)d 日志记录调用所在的源文件行的行号（如果可用）

%(asctime)s LogRecord 创建时的人类可读的时间。默认情况下，格式为 "2003-07-08 16:49:45,896" （逗号后的数字时间的毫秒部分）。这可以通过继承 :class:~logging.Formatter，并重载 formatTime() 改变。

%(message)s 记录的消息，视为 msg % args 这项即调用如app.logger.info('info log')中的参数'info log'，即message

## 与mongodb

<https://api.mongodb.com/python/current/api/pymongo/cursor.html>

import pymongo

flask连接mongodb

if request.method=='GET':

client=pymongo.MongoClient('127.0.0.1',27017)

db=client.wikidoc

collection=db.wiki

wikidoc=collection.find()

wikidoc\_count= wikidoc.count() #查询wikidoc返回记录的数量。

return render\_template('wikidoc.html',result=wikidoc,num=wikidoc\_count)

count()为统计数量的方法，我们先使用dir(wikidoc)查询wikidoc有那些方法，发现有count方法，则调用wikidoc.count()

结果按date字段进行排序，wikidoc=collection.find().sort('date', pymongo.DESCENDING)

降序DESCENDING 升序ASCENDING

if request.method=='POST':

search=request.form['search']

writer=request.form['writer']

wikidoc=collection.find().sort('date', pymongo.DESCENDING)

if writer:

wikidoc=collection.find({ "writer" : writer }).sort('date', pymongo.DESCENDING)

if search:

rex='^.\*'+search+'.\*'

app.logger.debug(rex)

rexExp = re.compile(rex)

wikidoc=collection.find({ "name" : rexExp}).sort('date', pymongo.DESCENDING)

if search and writer:

wikidoc=collection.find({ "name" : rexExp,'writer':writer }).sort('date', pymongo.DESCENDING)

wikidoc\_count=wikidoc.count()

app.logger.debug(search)

app.logger.debug(writer)

return render\_template('wikidoc.html',result=wikidoc,num=wikidoc\_count)

使用正则查询mongodb请看上面。

查询大于小于条件

wikidoc=collection.find({ "date" : {"$gt":startdate,"$lt":enddate}})

$lt：小于，$lte：小于等于，$gt：大于，$gte：大于等于

{ "date" : {"$gte":'2016/03',"$lte":'2016/04'}} 如果大于小于的日期只有月份，比如2016/03<=date<=2016/04

则代表2016/03为2016/03/01 2016/04代表2016/04/01

## 与cmd交互

1. 使用os.system()

os.system('cd d:/ && dir')

1. 使用os.popen。该方法不但执行命令还返回执行后的信息对象

tmp = os.popen('ls \*.py').readlines()

print tmp 好处在于：将返回的结果赋于一变量，便于程序的处理。

1. 当执行命令的参数或者返回中包含了中文文字，那么建议使用subprocess

使用模块subprocess

doc\_get=subprocess.Popen('cd D:\eclipse\_workspace\wikidoc\_url\ &&scrapy crawl wikidoc', shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

for i in doc\_get.stdout.readlines():

app.logger.debug(i)

## 其他规则

变量规则

url后如果添加str作为变量

@app.route('/user/<username>')

def show\_user\_profiile(username):

# show the user profile for that user

return 'User: {0}'.format(username)

url后如果添加 int float 需要如下转换

@app.route('/userid/<int:user\_id>')

def show\_userId(user\_id):

# show the user\_id, the id is an integer

return 'User ID: {0}'.format(user\_id)

**定制错误页面**，可以使用 errorhandler() 装饰器:

from flask import render\_template

@app.errorhandler(404)

def page\_not\_found(error):

return render\_template('page\_not\_found.html'), 404

## 问题解决

1. jinja2.exceptions.TemplateNotFound 没有找到template
2. 如果显示无法访问该网页，则重启py主文件试试。
3. 确保你的应用不叫做 flask.py， 因为这会与 Flask 本身冲突。
4. 在本机测试时记得写成 app.run(host="127.0.0.1",port=5000, debug=True)
   1. 而不是 app.run(host="0.0.0.0",port=5000, debug=True)
5. UnicodeDecodeError: 'ascii' codec can't decode byte 0xe7 in position 0: ordinal not in range(128)

解决方案：在代码文件开头

import sys

reload(sys)

sys.setdefaultencoding('utf-8')

# web.py框架

## 基础

urls是url映射规则（类似于servlet中的映射），class类hellofunction是链接请求响应。

如果要停止该程序Ctrl+C就可以退出。默认程序运行在8080端口.

如果运行在其他端口，使用python hello.py 20220

注意web.py并不具备部署网站的能力，因此对于web.py程序只能在本地访问，如果要进行部署必须要使用apache或者nginx。

当前版本0.37

简单的例子

import web

urls = ('/hello', 'hellofuction',

)

class hellofunction(object):

def GET(self):

return 'hello world'

if \_\_name\_\_ == "\_\_main\_\_":

app = web.application(urls, globals())

app.run()

## 模板

在第一行下面加上：

render = web.template.render('templates/')

这名话的意思是告诉 web.py 在 templates 目录中寻找模版文件

然后把 index.GET改成: 告诉 web.py 在你的模板目录下查找模板文件。修改 index.GET ：

name = 'Bob'

return render.index(name)

$def with (name)

$if name:

I just wanted to say <em>hello</em> to $name.

$else:

<em>Hello</em>, world!

注意：现在代码里需要有 4 个空格的缩进。

正如你看到的，除了顶部有 def with 的声明（saying what the template gets called with）以及每行代码开头都有 $ 外，这个模版有点像 Python 文件。目前，template.py 要求 $def 声明在文件的第一行。而且请注意，web.py 会自动转义在这儿使用的任何变量，这样，如果为了某种需要，name 被赋成一个包含 HTML 的值，它将得到正确地转义并作为平文本显示出来。如果你想关掉这个功能，用 $:name 代替 $name。

## 细节

1. 数据库交互

web.config.db\_parameters = dict(dbn='postgres', user='username', pw='password', db='dbname')

（按照你自已的设置调整这些参数——特别是 username，password 和 dbname，MySQL 的用户需要把 dbn 的值改成 mysql。）

# bottle框架

尽量不用bottle框架 太简洁

taskkill /f /t /im python.exe 如果调试过程中网页为500，在cmd上清除端口

## 基础

@bottle.route('/') 没有url后缀就写这样

这个映射装饰器有可选的关键字 method 默认是 method='GET'。

还有可能是 POST，PUT，DELETE，HEAD 或者监听其他的 HTTP 请求方法。

默认 default\_app() 工作

Bottle 创建一个 bottle.Bottle() 对象和装饰器，调用 bottle.run() 运行。

bottle.default\_app() 是默认。当然你可以创建自己的 bottle.Bottle() 实例

from bottle import Bottle, run

mybottle = Bottle()

@mybottle.route('/', method='GET')

def index():

return 'default\_app'

run(app=mybottle,host='localhost', port=8080)

from bottle import route, request

@route('/form/submit', method='POST')

def form\_submit():

form\_data = request.POST

do\_something(form\_data)

return "Done"

你可以提取 URL 的部分来建立动态变量名的映射。

@route('/hello/:name')

def hello(name):

return "Hello %s!" % name

默认情况下，一个 :placeholder 会一直匹配到下一个斜线。

需要修改的话，可以把正则字符加入到 #s 之间：

@route('/get\_object/:id#[0-9]+#')

def get(id):

return "Object ID: %d" % int(id)

或者使用完整的正则匹配组来实现：

@route('/get\_object/(?P<id>[0-9]+)')

def get(id):

return "Object ID: %d" % int(id)

正如你看到的，URL 参数仍然是字符串，即使你正则里面是数字。

你必须显式的进行类型强制转换。

## 自带模板语法

bottle的模板很挫，如果想要更进一步的效果，请选择第三方库jinja2或mako作为模板

Bottle 使用自带的小巧的模板。

你可以使用调用 template(template\_name, \*\*template\_arguments) 并返回结果。

@route('/hello/:name')

def hello(name):

return template('hello\_template', username=name)

模板搜索路径

模板是根据 bottle.TEMPLATE\_PATH 列表变量去搜索。

默认路径包含 ['./%s.tpl', './views/%s.tpl']。

模板缓存

模板在编译后在内存中缓存。

修改模板不会更新缓存，直到你清除缓存。

调用 bottle.TEMPLATES.clear()。

模板语法

模板语法是围绕 Python 很薄的一层。

主要目的就是确保正确的缩进块。

下面是一些模板语法的列子:

%...Python 代码开始。不必处理缩进问题。Bottle 会为你做这些。

%end 关闭一些语句 %if ...，%for ... 或者其他。关闭块是必须的。

{{...}} 打印出 Python 语句的结果。

%include template\_name optional\_arguments 包括其他模板。

每一行返回为文本。

模板语法

模板语法是非常精巧的，其工作原理基本可以说成是：将模板文件中的代码进行正确的缩进处理，以至你不再需要担心块缩进问题：

%if name == 'World':

<h1> Hello {{name}} </h1>

<p> This is a test.</p>

%else:

<h1>Hello {{name.title()}}</h1>

<p>How are you?</p>

%end

## 举例

import jinja2,os

from bottle import Bottle, run,request

app= Bottle()

*@app.route*(*"/"*,methods=[*"GET"*,*"POST"*] ,name=*'some\_method'*)

def **fenbiao**():

if request.method==*'POST'*:

orderid = request.forms.get(*"orderid"*)

remainder=int(orderid) % 32

app.logger.debug(*"订单号%s的后缀是%d"*,orderid,remainder)

return jinja2.Environment(loader=jinja2.FileSystemLoader(*'./templates'*)

).get\_template(*'fenbiao.html'*).render(orderid=orderid,remainder=remainder)

elif request.method == *'GET'*:

return jinja2.Environment(loader=jinja2.FileSystemLoader(*'./templates'*)

).get\_template(*'fenbiao.html'*).render()

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

run(app=app,host=*"127.0.0.1"*,port=8000)

## 其他

返回文件流和 JSON

WSGI 规范不能处理文件对象或字符串。

Bottle 自动转换字符串类型为 iter 对象。

下面的例子可以在 Bottle 下运行，但是不能运行在纯 WSGI 环境下。

@route('/get\_string')

def get\_string():

return "This is not a list of strings, but a single string"

@route('/file')

def get\_file():

return open('some/file.txt','r')

字典类型也是允许的。

会转换成 json 格式，自动返回 Content-Type: application/json。

@route('/api/status')

def api\_status():

return {'status':'online', 'servertime':time.time()}

你可以关闭这个特性 :bottle.default\_app().autojson = False

@validate() 装饰器

Bottle 提供一个方便的装饰器 validate() 来校验多个参数。

它可以通过关键字和过滤器来对每一个 URL 参数进行处理然后返回请求。

from bottle import route, validate

# /test/validate/1/2.3/4,5,6,7

@route('/test/validate/:i/:f/:csv')

@validate(i=int, f=float, csv=lambda x: map(int, x.split(',')))

def validate\_test(i, f, csv):

return "Int: %d, Float:%f, List:%s" % (i, f, repr(csv))

创造错误的压面

@bottle.error(404)

def error404(error):

return '404 error,nothing here,sorry'

## 使用beaker模块支持session

beaker – 一个缓存和会话库，可以用在 web 应用和独立 Python脚本和应用上。

import bottle

from bottle import request, run

from beaker.middleware import SessionMiddleware

app = bottle.default\_app()

@app.get('/')

def hello():

s = request.environ.get('beaker.session')

print s

s['user'] = "wt"

s.save()

return "hello"

@app.get('/login')

def hello():

s = request.environ.get('beaker.session')

print s

return "hello"

session\_opts = {

'session.type': 'file',

'session.cookie\_expires': 100,

'session.data\_dir': './session',

'session.auto': True

}

app = SessionMiddleware(app, session\_opts)

run(app = app, host='219.246.178.240', port='9092', reloader = True)

## 源码收获

### 框架

1. 路由（route）：这是 web 框架最关键的功能，把请求转发到对应的处理函数
2. 运行（run）：提供 wsgi server，能够监听所有请求并传递给后面的应用（就是 bottle 应用）
3. 模板（template）：动态生成 HTML 文件，发送给客户端

除了上面的三个模块，还有两个隐藏的东西没有在代码中出现：

1. 请求（request）：对 HTTP 请求的封装
2. 响应（response)：对 HTTP 响应的封装

下面会逐个分析这五个部分的内容，在那之前我们先看看整个流程的流转过程：

1. wsgi 启动，加载 bottle 应用，监听在设置的端口
2. 请求进来之后，被 wsgi server 封装好，调用 Bottle 的 \_\_call\_\_ 函数，交给 bottle 处理
3. bottle 根据传过来的 environ 字典，初始化 request 和 response 对象
4. bottle 获取传过来的 url 值，匹配之前根据代码中装饰器生成的路由器对象，找到处理函数
5. 调用处理函数
6. 处理函数会使用默认的模板引擎，替换所有的变量，返回处理结果
7. 把处理函数返回的值封装成 wsgi 兼容的对象
8. 把封装好的 response 返回给 wsgi server

框架概要

0000-0140: 模块载入, 兼容性调整

0140-0200: 逻辑无关的工具函数和工具类定义

0200-0240: 异常定义. 需要注意的是, 不需要消息体的HTTP响应, 例如HTTP重定向之类, 在bottle中也被处理成一种异常.

0240-0520: URL映射相关逻辑, 包括若干个路由异常的定义.

0520:0860: 主Bottle类的定义.

0860-1440: 装HTTP请求和响应的类的定义. 微型框架啥都可以省省, 但是这个如果再省, 就不能被称为是框架了.

1440-1570: 各种插件.

1570-1800: 各种数据结构.

1800-2050: 乱七八糟的小函数.

2050-2280: 框架虽小, 兼容的服务器倒真不少...

2280-2450: 应用控制, 也挺乱的, 两个用来载入app, 一个起server, 还有一个用来自动重启server(这个都有啊喂, 你真是微型框架咩).

2450-2830: 模板渲染及处理, 兼容的模板系统也不少.

2830-EOF : 变量定义及一些实例化, 以及起内置服务器的main函数.

### 实际

def makelist(data):

'''包装对象成为list'''

if isinstance(data, (tuple, list, set, dict)): return list(data)

elif data: return [data] # 对除去上述以外的元素生成list

else: return []

class CachedProperty(object):

'''每个实例只在第一次get的时候计算的property的值，之后存在instance的\_\_dict\_\_

中（当访问对象的property时，如果\_\_dict\_\_中有，则先返回\_\_dict\_\_中记录的值，如

果没有，则执行被访问property的\_\_get\_\_的方法。）'''

def \_\_init\_\_(self, func):

self.func = func

def \_\_get\_\_(self, obj, cls):

if obj is None: return self

value = obj.\_\_dict\_\_[self.func.\_\_name\_\_] = self.func(obj)

return value

class lazy\_attribute(object):

'''会在第一次调用的时候，把计算的结果设为owner class的attribute'''

def \_\_init\_\_(self, func):

functools.update\_wrapper(self, func, updated=[])

self.getter = func

def \_\_get\_\_(self, obj, cls):

value = self.getter(cls)

setattr(cls, self.\_\_name\_\_, value)

return value

try: from simplejson import dumps as json\_dumps, loads as json\_lds

except ImportError: # pragma: no cover

try: from json import dumps as json\_dumps, loads as json\_lds

except ImportError:

try: from django.utils.simplejson import dumps as json\_dumps, loads as json\_lds

except ImportError:

def json\_dumps(data):

raise ImportError("JSON support requires Python 2.6 or simplejson.")

json\_lds = json\_dumps

测试python版本号

import sys

py = sys.version\_info

print py

py3k = py >= (3, 0, 0)

py25 = py < (2, 6, 0)

py31 = (3, 1, 0) <= py < (3, 2, 0)

py27=(2, 6, 0) <= py < (2, 8, 0)

print py3k False

print py27 True

# tornado框架

## 基础

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

import tornado.httpserver

import tornado.ioloop

import tornado.options

import tornado.web

import os

from tornado.options import define,options

define(*'port'*,default=8008,help=*'run on the given port'*,type=int)

class **IndexHandler**(tornado.web.RequestHandler):

def **get**(*self*):

*self*.render(*'fenbiao.html'*,remainder=None)

def **post**(*self*):

orderid=*self*.get\_argument(*'orderid'*)

remainder=int(orderid) % 32

*self*.render(*'fenbiao.html'*,orderid=orderid,remainder=remainder)

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

tornado.options.parse\_command\_line()

app=tornado.web.Application(

handlers=[(r*'/'*,IndexHandler)],

template\_path=os.path.join(os.path.dirname(\_\_file\_\_),*"templates"*),

static\_path=os.path.join(os.path.dirname(\_\_file\_\_),*'static'*),

debug=True

)

http\_server=tornado.httpserver.HTTPServer(app)

http\_server.listen(options.port)

tornado.ioloop.IOLoop.instance().start()

1. 获取方法

talk=self.get\_argument('talk') get post都是用这获取值

Tornado的RequestHandler类有一系列有用的内建方法，包括get\_argument，我们在这里从一个查询字符串中取得参数greeting的值。（如果这个参数没有出现在查询字符串中，Tornado将使用get\_argument的第二个参数作为默认值。）

1. 状态码

404 Not Found

Tornado会在HTTP请求的路径无法匹配任何RequestHandler类相对应的模式时返回404（Not Found）响应码。

400 Bad Request

如果你调用了一个没有默认值的get\_argument函数，并且没有发现给定名称的参数，Tornado将自动返回一个400（Bad Request）响应码。

405 Method Not Allowed

如果传入的请求使用了RequestHandler中没有定义的HTTP方法（比如，一个POST请求，但是处理函数中只有定义了get方法），Tornado将返回一个405（Methos Not Allowed）响应码。

500 Internal Server Error

当程序遇到任何不能让其退出的错误时，Tornado将返回500（Internal Server Error）响应码。你代码中任何没有捕获的异常也会导致500响应码。

200 OK

如果响应成功，并且没有其他返回码被设置，Tornado将默认返回一个200（OK）响应码。

1. 指定url与函数关系

app = tornado.web.Application(handlers=[(r"/", IndexHandler)])

这里的参数handlers非常重要，值得我们更加深入的研究。它应该是一个元组组成的列表，其中每个元组的第一个元素是一个用于匹配的正则表达式，第二个元素是一个RequestHanlder类。在hello.py中，我们只指定了一个正则表达式-RequestHanlder对，但你可以按你的需要指定任意多个。

app = tornado.web.Application(handlers=[

(r"/reverse/(\w+)", ReverseHandler),

(r"/wrap", WrapHandler)

])

在上面的代码中，Application类在"handlers"参数中实例化了两个RequestHandler类对象。第一个引导Tornado传递路径匹配下面的正则表达式的请求：

/reverse/(\w+)

正则表达式告诉Tornado匹配任何以字符串/reverse/开始并紧跟着一个或多个字母的路径。括号的含义是让Tornado保存匹配括号里面表达式的字符串，并将其作为请求方法的一个参数传递给RequestHandler类。让我们检查ReverseHandler的定义来看看它是如何工作的：

class ReverseHandler(tornado.web.RequestHandler):

def get(self, input):

self.write(input[::-1])

你可以看到这里的get方法有一个额外的参数input。这个参数将包含匹配处理函数正则表达式第一个括号里的字符串。（如果正则表达式中有一系列额外的括号，匹配的字符串将被按照在正则表达式中出现的顺序作为额外的参数传递进来。）

1. 设置静态路径 js/css

你可以通过向Application类的构造函数传递一个名为static\_path的参数来告诉Tornado从文件系统的一个特定位置提供静态文件。Alpha Munger中的相关代码片段如下：

app = tornado.web.Application(

handlers=[(r'/', IndexHandler), (r'/poem', MungedPageHandler)],

template\_path=os.path.join(os.path.dirname(\_\_file\_\_), "templates"),

static\_path=os.path.join(os.path.dirname(\_\_file\_\_), "static"),

debug=True

)

在这里，我们设置了一个当前应用目录下名为static的子目录作为static\_path的参数。现在应用将以读取static目录下的filename.ext来响应诸如/static/filename.ext的请求，并在响应的主体中返回。

## 模板

1. 继承父模板，并且使用块

{% extends "main.html" %}

{% block header %}

<h1>{{ header\_text }}</h1>

{% end %}

1. 调用css和js

<link rel="stylesheet" href="{{ static\_url("transfer.css") }}">

<script src="{{ static\_url("js/script.js") }}"></script>

1. 如何穿插调用两个不同的list中的值？

<div >

{%for i in range(len(excelContentList)) %}

<p>{{topRow[i]}}</p>

<p><span>{{excelContentList[i]}}</span></p>

{% end %}

</div>

1. 如何在一个form中设置两个不同的submit按钮？

程序根据submit按钮的name来进行区分。

<input type="submit" name="list" value="Submit and go to list">

<input type="submit" name="detail" value="Submit and go to detail">

if 'list' in request.POST:

return redirect('list\_url')

else:

return redirect('detail\_url')

## cookie

Tornado的set\_secure\_cookie()和get\_secure\_cookie()函数发送和取得浏览器的cookies，以防范浏览器中的恶意修改。为了使用这些函数，你必须在应用的构造函数中指定cookie\_secret参数。

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

import tornado.httpserver

import tornado.ioloop

import tornado.web

import tornado.options

from tornado.options import define,options

define('port',default=8000,type=int)

class MainHandler(tornado.web.RequestHandler):

def get(self):

cookie=self.get\_secure\_cookie('count')#获取cookie

count=int(cookie)+1 if cookie else 1 #如果cookie存在加1，不存在设为1

countString='1 time' if count==1 else'%d times'%count

self.set\_secure\_cookie('count',str(count)) #设置新的cookie

self.write(

'<html><head><title>Cookie Counter</title></head>'

'<body><h1>You’ve viewed this page %s times.</h1>' % countString +

'</body></html>'

)

if \_\_name\_\_ == "\_\_main\_\_":

tornado.options.parse\_command\_line()

settings = {"cookie\_secret": "bZJc2sWbQLKos6GkHn/VB9oXwQt8S0R0kRvJ5/xJ89E="}

#设置cookie秘钥

application = tornado.web.Application([ (r'/', MainHandler)],

\*\*settings)

http\_server = tornado.httpserver.HTTPServer(application)

http\_server.listen(options.port)

tornado.ioloop.IOLoop.instance().start()

## 使用Tornado的XSRF保护

你可以通过在应用的构造函数中包含xsrf\_cookies参数来开启XSRF保护：

settings = {

"cookie\_secret": "bZJc2sWbQLKos6GkHn/VB9oXwQt8S0R0kRvJ5/xJ89E=",

"xsrf\_cookies": True

}

application = tornado.web.Application([ (r'/', MainHandler), (r'/purchase', PurchaseHandler),], \*\*settings)

当这个应用标识被设置时，Tornado将拒绝请求参数中不包含正确的\_xsrf值的POST、PUT和DELETE请求。Tornado将会在幕后处理\_xsrf cookies，但你必须在你的HTML表单中包含XSRF令牌以确保授权合法请求。要做到这一点，只需要在你的模板中包含一个xsrf\_form\_html调用即可：

<form action="/purchase" method="POST">

{% raw xsrf\_form\_html() %}

<input type="text" name="title" />

<input type="text" name="quantity" />

<input type="submit" value="Check Out" />

</form>

## 异步web服务

class IndexHandler(tornado.web.RequestHandler):

@tornado.web.asynchronous

@tornado.gen.engine

def get(self):

query = self.get\_argument('q')

client = tornado.httpclient.AsyncHTTPClient()

response = yield tornado.gen.Task(client.fetch,

"http://search.twitter.com/search.json?" + \

urllib.urlencode({"q": query, "result\_type": "recent", "rpp": 100}))

body = json.loads(response.body)

result\_count = len(body['results'])

now = datetime.datetime.utcnow()

raw\_oldest\_tweet\_at = body['results'][-1]['created\_at']

oldest\_tweet\_at = datetime.datetime.strptime(raw\_oldest\_tweet\_at,

"%a, %d %b %Y %H:%M:%S +0000")

seconds\_diff = time.mktime(now.timetuple()) - \

time.mktime(oldest\_tweet\_at.timetuple())

tweets\_per\_second = float(result\_count) / seconds\_diff

self.write("""

<div style="text-align: center">

<div style="font-size: 72px">%s</div>

<div style="font-size: 144px">%.02f</div>

<div style="font-size: 24px">tweets per second</div>

</div>""" % (query, tweets\_per\_second))

self.finish()

if \_\_name\_\_ == "\_\_main\_\_":

tornado.options.parse\_command\_line()

app = tornado.web.Application(handlers=[(r"/", IndexHandler)])

http\_server = tornado.httpserver.HTTPServer(app)

http\_server.listen(options.port)

tornado.ioloop.IOLoop.instance().start()

正如你所看到的，这个代码和前面两个版本的代码非常相似。主要的不同点是我们如何调用Asynchronous对象的fetch方法。下面是相关的代码部分：

client = tornado.httpclient.AsyncHTTPClient()

response = yield tornado.gen.Task(client.fetch,

"http://search.twitter.com/search.json?" + \

urllib.urlencode({"q": query, "result\_type": "recent", "rpp": 100}))

body = json.loads(response.body)

我们使用Python的yield关键字以及tornado.gen.Task对象的一个实例，将我们想要的调用和传给该调用函数的参数传递给那个函数。这里，yield的使用返回程序对Tornado的控制，允许在HTTP请求进行中执行其他任务。当HTTP请求完成时，RequestHandler方法在其停止的地方恢复。这种构建的美在于它在请求处理程序中返回HTTP响应，而不是回调函数中。因此，代码更易理解：所有请求相关的逻辑位于同一个位置。而HTTP请求依然是异步执行的，所以我们使用tornado.gen可以达到和使用回调函数的异步请求版本相同的性能，

import tornado.httpserver

import tornado.ioloop

import tornado.options

import tornado.web

import tornado.httpclient

import urllib

import json

import datetime

import time

from tornado.options import define, options

define("port", default=8000, help="run on the given port", type=int)

class IndexHandler(tornado.web.RequestHandler):

@tornado.web.asynchronous

def get(self):

query = self.get\_argument('q')

client = tornado.httpclient.AsyncHTTPClient()

client.fetch("http://search.twitter.com/search.json?" + \

urllib.urlencode({"q": query, "result\_type": "recent", "rpp": 100}),

callback=self.on\_response)

def on\_response(self, response):

body = json.loads(response.body)

result\_count = len(body['results'])

now = datetime.datetime.utcnow()

raw\_oldest\_tweet\_at = body['results'][-1]['created\_at']

oldest\_tweet\_at = datetime.datetime.strptime(raw\_oldest\_tweet\_at,

"%a, %d %b %Y %H:%M:%S +0000")

seconds\_diff = time.mktime(now.timetuple()) - \

time.mktime(oldest\_tweet\_at.timetuple())

tweets\_per\_second = float(result\_count) / seconds\_diff

self.write("""

<div style="text-align: center">

<div style="font-size: 72px">%s</div>

<div style="font-size: 144px">%.02f</div>

<div style="font-size: 24px">tweets per second</div>

</div>""" % (self.get\_argument('q'), tweets\_per\_second))

self.finish()

if \_\_name\_\_ == "\_\_main\_\_":

tornado.options.parse\_command\_line()

app = tornado.web.Application(handlers=[(r"/", IndexHandler)])

http\_server = tornado.httpserver.HTTPServer(app)

http\_server.listen(options.port)

tornado.ioloop.IOLoop.instance().start()

AsyncHTTPClient的fetch方法并不返回调用的结果。取而代之的是它指定了一个callback参数；你指定的方法或函数将在HTTP请求完成时被调用，并使用HTTPResponse作为其参数。

client = tornado.httpclient.AsyncHTTPClient()

client.fetch("http://search.twitter.com/search.json?" + »

urllib.urlencode({"q": query, "result\_type": "recent", "rpp": 100}),

callback=self.on\_response)

在这个例子中，我们指定on\_response方法作为回调函数。我们之前使用期望的输出转化Twitter搜索API请求到网页中的所有逻辑被搬到了on\_response函数中。还需要注意的是@tornado.web.asynchronous装饰器的使用（在get方法的定义之前）以及在回调方法结尾处调用的self.finish()

记住当你使用@tornado.web.asynchonous装饰器时，Tornado永远不会自己关闭连接。你必须在你的RequestHandler对象中调用finish方法来显式地告诉Tornado关闭连接。（否则，请求将可能挂起，浏览器可能不会显示我们已经发送给客户端的数据。

## 其他

1. 禁用自动转义

禁用自动转义，一种方法是在Application构造函数中传递autoescape=None，另一种方法是在每页的基础上修改自动转义行为，如下所示：

{% autoescape None %}

{{ mailLink }}

这些autoescape块不需要结束标签，并且可以设置xhtml\_escape来开启自动转义（默认行为），或None来关闭。

然而，在理想的情况下，你希望保持自动转义开启以便继续防护你的网站。因此，你可以使用{% raw %}指令来输出不转义的内容。

{% raw mailLink %}

## windows上查看端口占用情况并清除

查看所有的端口占用情况

C:\>netstat -ano

查看指定端口的占用情况

C:\>netstat -aon|findstr "9050"

协议 本地地址 外部地址 状态 PID

TCP 127.0.0.1:9050 0.0.0.0:0 LISTENING 2016

查看PID对应的进程

C:\>tasklist|findstr "2016"

映像名称 PID 会话名 会话# 内存使用

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

python.exe 2016 Console 0 16,064 K

结束该进程

C:\>taskkill /f /t /im python.exe

使windows程序后台运行：

在执行的命令前加上start /b，比如start /b run.bat。就相当于Linux下的run.sh &。

处于局域网中的计算机IP地址，是由路由器或集线器分配的局域网IP地址，这串IP地址字符通常以192.168.0.\*的形式出现，它是本机在局域网中的IP地址，并非公网地址

## 遇到的问题

1. 报错socket.error: [Errno 10048] 那是因为端口号被占用。可能之前启动过另外一个程序占用了该端口

解决办法：点击eclipse-console窗口中的terminate all

如果想重新启动当前所处的文件程序，点击 relauch with the same configuration

1. 如何使tornado程序在2分钟后自动停止？

print "Your web server will self destruct in 2 minutes"

time.sleep(120)

stopTornado()

1. tornado修改后不用重新启动，保存后程序自己会关闭并重启

## linux安装和启动

cd /usr/local/games/excelPhoneServer

nohup python server.py & 后台运行

必须加上nohup,才能在关闭linux终端后，仍然会后台运行，而不是挂起

清除已存在的端口号

查找端口对应的服务pid号。 netstat -anlp| grep 端口号 netstat -anlp| grep 2016

在上述命令的最后一列会给出端口对应应用的pid号 kill -9 pid 即可关闭服务，并关掉端口。

# MongoDB

## windows下MongoDB的安装及配置

安装包 mongodb-win32-x86\_64-2008plus-2.6.7.zip

地址：<http://www.360sdn.com/opensource-datasource/2015/0403/5584.html>

1. 将zip文件解压放到盘符的根目录（如E：或D：），为了方便建议文件夹命名尽量简短如（E:\mongodb）
2. 创建数据库文件的存放位置，比如d:/mongodb/data/db。启动mongodb服务之前需要必须创建数据库文件的存放文件夹，否则命令不会自动创建，而且不能启动成功。
3. 打开cmd（windows键+r输入cmd）命令行，进入E:\mongodb\bin目录,输入如下的命令启动mongodb服务：
   1. E:/mongodb/bin>mongod --dbpath E:\mongodb\data\db
4. mongodb默认连接端口27017，可以打开http://localhost:27017查看。 显示It looks like you are trying to access MongoDB over HTTP on the native driver port.代表成功。
5. 在e:\mongodb\data下新建文件夹log（存放日志文件）并且新建文件mongodb.log
   1. 在e:\mongodb新建文件mongo.config
6. 用记事本打开mongo.config输入：
   1. dbpath=E:\mongodb\data\db
   2. logpath=E:\mongodb\data\log\mongodb.log
7. 启动mongo数据库服务端 mongod --config E:\mongodb\mongo.config 代表取mongo.config配置中的值。
8. 当mongod.exe被关闭时，mongo.exe 就无法连接到数据库了，因此每次想使用mongodb数据库都要开启mongod.exe程序，所以比较麻烦，此时我们可以将MongoDB安装为windows服务

还是运行cmd，进入bin文件夹，执行下列命令

　> e:\mongodb\bin>mongod --dbpath "e:\mongodb\data\db" --logpath "e:\mongodb\data\log\MongoDB.log" --install --serviceName "MongoDB"

　这里MongoDB.log就是开始建立的日志文件，--serviceName "MongoDB" 服务名为MongoDB

　接着启动mongodb服务

　> e:\mongodb\bin>NET START MongoDB

打开任务管理器，可以看到进程已经启动

1. 关闭服务和删除进程

　> e:\mongodb\bin>NET stop MongoDB (关闭服务)

　> e:\mongodb\bin>mongod --dbpath "e:\mongodb\data\db" --logpath "e:\mongodb\data\log\MongoDB.log" --remove --serviceName "MongoDB" (删除，注意不是--install了）

## linux下安装mongodb

### 步骤

mongodb-linux-x86\_64-2.6.10 文件名要改为mongodb-linux-x86\_64-2.6.10.tgz 然后解压

1. 创建数据库文件的存放位置，比如/usr/local/mongodb/data/db
2. 在/usr/local/mongodb/data下新建文件夹log（存放日志文件）并且新建文件mongodb.log
3. 在/usr/local/mongodb新建文件mongo.config
4. 添加环境变量 /usr/local/mongodb/bin 到/etc/profile文件
5. 用记事本打开mongo.config输入：

dbpath=/usr/local/mongodb/data/db

logpath=/usr/local/mongodb/data/log

port=27017

fork=true

nohttpinterface=true

1. 启动 cd /usr/local/mongodb/bin

chmod -R 755 /usr/local/mongodb/bin 修改bin文件夹下所有内容的执行权限为755

mongod --dbpath /usr/local/mongodb/data/db --logpath /usr/local/mongodb/data/log/MongoDB.log --fork

--fork 选项将会通知 mongod 在后台运行 输入该命令后，直接关闭标签页

1. 进入mongodb的shell模式 /usr/local/mongodb/bin/mongo 或者如果添加了环境变量，直接输入 mongo

#查看数据库列表

> show dbs

local 0.078GB

1. 停止mongodb 直接ctrl+c 而不是ctrl+s

### 具体细节

1. 如果使用kill -9 pid的方式关闭mongodb，并且mongodb不能正常启动，则需删除mongod.lock文件：

# rm -f /usr/local/mongodb/data/db/mongod.lock

1. MongoDB 提供了简单的 HTTP 用户界面。 如果你想启用该功能，需要在启动的时候指定参数 --rest 。

mongod --dbpath /usr/local/mongodb/data/db --logpath /usr/local/mongodb/data/log/MongoDB.log --rest

然后进入网址：<http://192.168.156.105:28017/>

1. 如果让mongo随linux自动启动，在/etc/rc.local添加如下即可：

mongod --dbpath /usr/local/mongodb/data/db --logpath /usr/local/mongodb/data/log/MongoDB.log --rest

1. 查看mongo后台运行 ps -ef |grep mongo

## 使用mongodb命令

1.常用的命令

show dbs 显示数据库列表 刚创建的数据库mydb 在该命令下是不显示的。因为显示的数据库至少有一个文件。

use dbname 进入dbname数据库，大小写敏感，没有这个数据库也不要紧

db 要检查当前选择的数据库使用命令

show collections 显示数据库中的集合，相当于表格

2.创建&新增

db.users.save({"name":"lecaf"}) 创建了名为users的集合，并新增了一条{"name":"lecaf"}的数据

db.users.insert({"name":"ghost", "age":10}) 在users集合中插入一条新数据，，如果没有users这个集合，mongodb会自动创建

save()和insert()也存在着些许区别：若新增的数据主键已经存在，insert()会不做操作并提示错误，而save() 则更改原来的内容为新内容。

存在数据：{ \_id : 1, " name " : " n1 "} ，\_id是主键

insert({ \_id : 1, " name " : " n2 " }) 会提示错误

save({ \_id : 1, " name " : " n2 " }) 会把 n1 改为 n2 ，有update的作用。

3.删除

db.users.remove() 删除users集合下所有数据

db.users.remove({"name": "lecaf"}) 删除users集合下name=lecaf的数据

db.users.drop()或db.runCommand({"drop","users"}) 删除集合users

db.runCommand({"dropDatabase": 1}) 删除当前数据库

4.查找

db.users.find() 查找users集合中所有数据 users为collection名

db.users.findOne() 查找users集合中的第一条数据

5.修改

db.users.update({"name":"lecaf"}, {"age":10})

db.users.update({"name":"lecaf"}, {"age":10,'name':'lecaf'})

修改name=lecaf的数据为age=10，第一个参数是查找条件，第二个参数是修改内容，除了主键，其他内容会被第二个参数的内容替换，主键不能修改

## 高级应用

1.条件查找

db.collection.find({ "key" : value }) 查找key=value的数据

db.collection.find({ "key" : { $gt: value } }) key > value

db.collection.find({ "key" : { $lt: value } }) key < value

db.collection.find({ "key" : { $gte: value } }) key >= value

db.collection.find({ "key" : { $lte: value } }) key <= value

db.collection.find({ "key" : { $gt: value1 , $lt: value2 } }) value1 < key <value2

db.collection.find({ "key" : { $ne: value } }) key <> value

db.collection.find({ "key" : { $mod : [ 10 , 1 ] } }) 取模运算，条件相当于key % 10 == 1 即key除以10余数为1的

db.collection.find({ "key" : { $nin: [ 1, 2, 3 ] } }) 不属于，条件相当于key的值不属于[ 1, 2, 3 ]中任何一个

db.collection.find({ "key" : { $in: [ 1, 2, 3 ] } }) 属于，条件相当于key等于[ 1, 2, 3 ]中任何一个

db.collection.find({ "key" : { $size: 1 } }) $size 数量、尺寸，条件相当于key的值的数量是1（key必须是数组，一个值的情况不能算是数量为1的数组）

db.collection.find({ "key" : { $exists : true|false } }) $exists 字段存在，true返回存在字段key的数据，false返回不存在字度key的数据

db.collection.find({ "key": /^val.\*val$/i }) 正则，类似like；“i”忽略大小写，“m”支持多行

db.collection.find({ $or : [{a : 1}, {b : 2} ] }) $or或 （注意：MongoDB 1.5.3后版本可用），符合条件a=1的或者符合条件b=2的数据都会查询出来

db.collection.find({ "key": value , $or : [{ a : 1 } , { b : 2 }] }) 符合条件key=value ，同时符合其他两个条件中任意一个的数据

db.collection.find({ "key.subkey" :value }) 内嵌对象中的值匹配，注意："key.subkey"必须加引号

db.collection.find({ "key": { $not : /^val.\*val$/i } }) 这是一个与其他查询条件组合使用的操作符，不会单独使用。上述查询条件得到的结果集加上$not之后就能获得相反的集合。

db.collection.find({ "field" : { $gt: value1, $lt: value2 } } ) // value1 < field < value

wikidoc=collection.find({ "name" : rexExp,'writer':'wangkaibin' }) 多种条件一起查找

2.排序

db.collection.find().sort({ "key1" : -1 ,"key2" : 1 }) 这里的1代表升序，-1代表降序

3.其他

db.collection.find().limit(5) 控制返回结果数量，如果参数是0，则当作没有约束，limit()将不起作用

db.collection.find().skip(5) 控制返回结果跳过多少数量，如果参数是0，则当作没有约束，skip()将不起作用，或者说跳过了0条

db.collection.find().skip(5).limit(5) 可用来做分页，跳过5条数据再取5条数据

db.collection.find().count(true) count()返回结果集的条数

db.collection.find().skip(5).limit(5).count(true) 在加入skip()和limit()这两个操作时，要获得实际返回的结果数，需要一个参数true，否则返回的是符合查询条件的结果总数

## MongoVUE工具

1. 连接本地创建的mongodb，server:127.0.0.1 port:27017 username password无需填写
2. 创建collections，相当于表格table。 右键数据库,选择add collection 如果collection的某个为gisy,则查询该集合下所有数据：db.gisy.find()
3. collections中一个集合如wiki下的一个key相当于mysql中一条记录 collections下集合相当于表table
4. 删除表记录右键该条记录，remove.
5. 删除该表中所有记录，collection--remove all
6. find窗口-collection下
   1. 区域为基本查询{find}

一组键值，key为要查询的字段，value为要查询字段的值，如有多个用逗号隔开

语法 {“要查询的字段”：“要查询字段的值”，“第二个”…..}

比如{ "writer" : 'wangkaibin' }

* 1. 字段查询

{'date':1,'url':1}，查询之后只显示date, url两列

字段的值是1表示：查出id与该字段；值为0表示：查出表中所有字段，不包括该字段

* 1. 排序

方法使用一组键值对做参数，key是document里的key的名字，value是1升序或者-1降序。

使用date进行排序，小于0表示降序，大于等于0表示升序

{"date":-1}

$lt：小于，$lte：小于等于，$gt：大于，$gte：大于等于

* 1. 结果集的筛选

在4区域会看到，形如

limit函数限制返回的结果集的上限，如设置为1000那么就返回前面1000行

skip函数跳过前x个结果，返回剩余的，此处为0，就不跳过，如果设置为1，那么就跳过查询到的第一条数据，返回后面的

点击”find2”

基本和1相同，唯一的区别是有个where语句，和这个里面是写js的，有js基础的兴趣的可以点下面的网站看看

http://blog.163.com/wm\_at163/blog/static/132173490201252610424458/

1. update更新数据

左边1为查询，就是要修改的地方，先查出来，然后再右边修改，比如这边要名为111的人，将其年龄改为111岁那么

左边写{“name”：“111”}右边写{"name":"111","age":111}，

左边写完可以点击下面的find看看文档内关于这个名字的信息，

点击update1进行修改，

当此处选择的时候，点击update1会先查看这条数据，如果没有就插入，此时，左边不需要写

1. insert插入数据

此处插入的数据，为bjosn格式，键值对

“key”:value

Key为要插入的字段名称，vlaue为对应的值，如果有多个值，可以用[]括起来

## PyMongo 与mongo库交互

引用PyMongo

>>> import pymongo

创建连接Connection

>>> import pymongo

>>> conn = pymongo.Connection('localhost',27017)

或

>>> from pymongo import Connection

>>> conn = Connection('localhost',27017)

创建Connection时，指定host及port参数

>>> import pymongo

>>> conn = pymongo.Connection(host='127.0.0.1',port=27017)

连接数据库

>>> db = conn.ChatRoom

或

>>> db = conn['ChatRoom']

连接聚集

>>> account = db.Account

或

>>> account = db["Account"]

查看全部聚集名称

>>> db.collection\_names()

查看聚集的一条记录

>>> db.Account.find\_one()

>>> db.Account.find\_one({"UserName":"keyword"})

查看聚集的字段

>>> db.Account.find\_one({},{"UserName":1,"Email":1})

{u'UserName': u'libing', u'\_id': ObjectId('4ded95c3b7780a774a099b7c'), u'Email': u'libing@35.cn'}

>>> db.Account.find\_one({},{"UserName":1,"Email":1,"\_id":0})

{u'UserName': u'libing', u'Email': u'libing@35.cn'}

查看聚集的多条记录

>>> for item in db.Account.find():

item

>>> for item in db.Account.find({"UserName":"libing"}):

item["UserName"]

查看聚集的记录统计

>>> db.Account.find().count()

>>> db.Account.find({"UserName":"keyword"}).count()

聚集查询结果排序

>>> db.Account.find().sort("UserName") --默认为升序

>>> db.Account.find().sort("UserName",pymongo.ASCENDING) --升序

>>> db.Account.find().sort("UserName",pymongo.DESCENDING) --降序

聚集查询结果多列排序

>>> db.Account.find().sort([("UserName",pymongo.ASCENDING),("Email",pymongo.DESCENDING)])

添加记录

>>> db.Account.insert({"AccountID":21,"UserName":"libing"})

修改记录

>>> db.Account.update({"UserName":"libing"},{"$set":{"Email":"libing@126.com","Password":"123"}})

删除记录

>>> db.Account.remove() -- 全部删除

>>> db.Test.remove({"UserName":"keyword"})

# redis

Redis是一个Key-Value存储系统。它支持存储的value类型有：string(字符串),list(链表), set(无序集合), zset(sorted set有序集合)和hash，也可以把redis看成一个数据结构服务器。这些数据类型都支持pust/pop、add/remove及取交集、并集和差集运算，Redis支持各种不同方式的排序。数据都是缓存在内存中的，它也可以周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并实现了master-slave(主从)同步。

Redis 是一个高性能的key-value数据库。

## redis基础

E:\04redis\32bit

1. 启动redis服务端即redis-server.exe

敲入命令：redis-server.exe redis.conf 或者 直接点击该exe

启动cmd窗口要一直开着，关闭后则Redis服务关闭

1. 启动redis客户端

一个服务端可以连接多个客户端。

另开一个cmd 命令窗口：

敲入命令：redis-cli.exe -h 10.1.89.10 -p 6379（这里的ip是你本机的ip cmd:ipconfig一下就行了）

或者 redis-cli.exe -h 127.0.0.1 -p 6379，其中 127.0.0.1是本地ip，6379是redis服务端的默认端口。

或者直接点击redis-cli.exe

连接成功。服务端页面显示1 clients connected (0 slaves), 1188240 bytes in use

1. 存储一个key为test，value为hello word的字符串，然后获取key值。

set test 'hello word'

get test

1. 配置文件

可为redis服务启动指定配置文件，配置文件 redis.conf 在Redis根目录下。

#修改daemonize为yes，即默认以后台程序方式运行（还记得前面手动使用&号强制后台运行吗）。

daemonize no

#可修改默认监听端口

port 6379

#修改生成默认日志文件位置

logfile "/home/futeng/logs/redis.log"

#配置持久化文件存放位置

dir /home/futeng/data/redisData

1. 文件介绍

redis-benchmark.exe #基准测试

redis-check-aof.exe # aof

redischeck-dump.exe # dump

redis-cli.exe # 客户端

redis-server.exe # 服务器

redis.windows.conf # 配置文件

## redis命令 客户端

<http://www.cnblogs.com/stephen-liu74/archive/2012/03/26/2356951.html> 命令查询网页

1. 打印一个特定的信息 message ，测试时使用。

redis> ECHO "Hello Moto"

"Hello Moto"

1. 返回当前数据库中的key的数目： dbsize
2. 获取服务器的信息和统计 info
3. 客户端退出连接 quit
4. 连接redis数据库(0-15) 比如选择5号数据库 select 5
5. 测试连接是否存活 ping
6. 返回key对应的value的类型 type mm mm为key 由于mm键在数据库中不存在，因此该命令返回none。
7. 清空当前打开的数据库 flushdb
8. 返回数据库中的任意键。randomkey
9. 返回(nil) 代表没有数据 比如输入randomkey 数据库无key，则返回(nil)
10. 添加String类型的模拟数据 set mykey 2
11. 添加Set类型的模拟数据 sadd mysetkey 1 2 3
12. 添加Hash类型的模拟数据 hset mmtest username "stephen"
13. 根据参数中的模式，获取当前数据库中符合该模式的所有key，从输出可以看出，该命令在执行时并不区分与Key关联的Value类型。 keys my\*
14. 删除了两个Keys。del mykey mykey2
15. 查看Key是否还存在 exists mykey (integer) 0表示不存在 (integer) 1 表示存在
16. 将当前数据库中的mysetkey键移入到ID为1的数据库中 move mysetkey 1 结果(integer) 1表示已经移动成功。
17. 打开ID为1的数据库。select 1
18. 重新打开ID为0的缺省数据库。select 0
19. 将mykey改名为mykey1 rename mykey mykey1
20. 获取key对应的value值 get mykey 结果(nil)表示无该key
21. rename mykey mykey1 为指定指定的键重新命名，如果参数中的两个Keys的命令相同，或者是源Key不存在，该命令都会返回相关的错误信息(error) ERR no such key。如果newKey已经存在，则直接覆盖。
22. renamenx oldkey newkey 如果新值不存在，则将参数中的原值修改为新值。其它条件和RENAME一致。
23. 将该键的超时设置为100秒。expire mykey 100

该命令为参数中指定的Key设定超时的秒数，在超过该时间后，Key被自动的删除。如果该Key在超时之前被修改，与该键关联的超时将被移除。 (integer) 1 1表示超时被设置，0则表示Key不存在，或不能被设置。

1. 通过ttl命令查看一下还剩下多少秒。ttl mykey 返回所剩描述，如果该键不存在或没有超时设置，则返回-1。

(integer) 97

1. 立刻执行persist命令，该存在超时的键变成持久化的键，即将该Key的超时去掉。 persist mykey (integer) 1
2. 删除key和value del mykey
3. 重新更新该键的超时时间为20秒，从返回值可以看出该命令执行成功。expire mykey 20 (integer) 1
4. 立刻更新该键的值，以使其超时无效。set mykey "world"
5. Redis Expireat命令是用来以Unix时间戳格式设置键的到期时间。 EXPIREAT yiibai 1293840000
6. auth superman 输入密码(superman) 是用来向服务器验证给定的密码。 如果密码与在配置文件中的口令相匹配，则服务器会返回OK状态码，并开始接受命令。否则，将返回一个错误，并且客户需要尝试新的密码。

## redis 五种数据类型的使用场景

### String

常用命令：

除了get、set、incr、decr mget等操作外，Redis还提供了下面一些操作：

获取字符串长度

往字符串append内容

设置和获取字符串的某一段内容

设置及获取字符串的某一位（bit）

批量设置一系列字符串的内容

应用场景：

String是最常用的一种数据类型，普通的key/value存储都可以归为此类，value其实不仅是String，

也可以是数字：比如想知道什么时候封锁一个IP地址(访问超过几次)。INCRBY命令让这些变得很容易，通过原子递增保持计数。

实现方式：

m,decr等操作时会转成数值型进行计算，此时redisObject的encoding字段为int。

### Hash

常用命令：

hget,hset,hgetall 等。

应用场景：

我们简单举个实例来描述下Hash的应用场景，比如我们要存储一个用户信息对象数据，包含以下信息：

用户ID，为查找的key，

存储的value用户对象包含姓名name，年龄age，生日birthday 等信息，

如果用普通的key/value结构来存储，主要有以下2种存储方式：

第一种方式将用户ID作为查找key,把其他信息封装成一个对象以序列化的方式存储，

如：set u001 "李三,18,20010101"

这种方式的缺点是，增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护，引入CAS等复杂问题。

第二种方法是这个用户信息对象有多少成员就存成多少个key-value对儿，用用户ID+对应属性的名称作为唯一标识来取得对应属性的值，

如：mset user:001:name "李三 "user:001:age18 user:001:birthday "20010101"

虽然省去了序列化开销和并发问题，但是用户ID为重复存储，如果存在大量这样的数据，内存浪费还是非常可观的。

那么Redis提供的Hash很好的解决了这个问题，Redis的Hash实际是内部存储的Value为一个HashMap，

并提供了直接存取这个Map成员的接口，

如：hmset user:001 name "李三" age 18 birthday "20010101"

也就是说，Key仍然是用户ID,value是一个Map，这个Map的key是成员的属性名，value是属性值，

这样对数据的修改和存取都可以直接通过其内部Map的Key(Redis里称内部Map的key为field), 也就是通过

key(用户ID) + field(属性标签) 操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题。

这里同时需要注意，Redis提供了接口(hgetall)可以直接取到全部的属性数据,但是如果内部Map的成员很多，那么涉及到遍历整个内部Map的操作，由于Redis单线程模型的缘故，这个遍历操作可能会比较耗时，而另其它客户端的请求完全不响应，这点需要格外注意。

实现方式：

上面已经说到Redis Hash对应Value内部实际就是一个HashMap，实际这里会有2种不同实现，这个Hash的成员比较少时Redis为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的HashMap结构，对应的value redisObject的encoding为zipmap,当成员数量增大时会自动转成真正的HashMap,此时encoding为ht。

### list

常用命令：

lpush,rpush,lpop,rpop,lrange,BLPOP(阻塞版)等。

应用场景：

Redis list的应用场景非常多，也是Redis最重要的数据结构之一。

我们可以轻松地实现最新消息排行等功能。

Lists的另一个应用就是消息队列，可以利用Lists的PUSH操作，将任务存在Lists中，然后工作线程再用POP操作将任务取出进行执行。

实现方式：

Redis list的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

RPOPLPUSH source destination

命令 RPOPLPUSH 在一个原子时间内，执行以下两个动作：

将列表 source 中的最后一个元素(尾元素)弹出，并返回给客户端。

将 source 弹出的元素插入到列表 destination ，作为 destination 列表的的头元素。

如果 source 和 destination 相同，则列表中的表尾元素被移动到表头，并返回该元素，可以把这种特殊情况视作列表的旋转(rotation)操作。

一个典型的例子就是服务器的监控程序：它们需要在尽可能短的时间内，并行地检查一组网站，确保它们的可访问性。

redis.lpush "downstream\_ips", "192.168.0.10"

redis.lpush "downstream\_ips", "192.168.0.11"

redis.lpush "downstream\_ips", "192.168.0.12"

redis.lpush "downstream\_ips", "192.168.0.13"

Then:

next\_ip = redis.rpoplpush "downstream\_ips", "downstream\_ips"

BLPOP

假设现在有 job 、 command 和 request 三个列表，其中 job 不存在， command 和 request 都持有非空列表。考虑以下命令：

BLPOP job command request 30 #阻塞30秒，0的话就是无限期阻塞,job列表为空,被跳过,紧接着command 列表的第一个元素被弹出。

1) "command" # 弹出元素所属的列表

2) "update system..." # 弹出元素所属的值

为什么要阻塞版本的pop呢，主要是为了避免轮询。举个简单的例子如果我们用list来实现一个工作队列。执行任务的thread可以调用阻塞版本的pop去获取任务这样就可以避免轮询去检查是否有任务存在。当任务来时候工作线程可以立即返回，也可以避免轮询带来的延迟。

### set

常用命令：

sadd,srem,spop,sdiff ,smembers,sunion 等。

应用场景：

Redis set对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。

比如在微博应用中，每个人的好友存在一个集合（set）中，这样求两个人的共同好友的操作，可能就只需要用求交集命令即可。

Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实

实现方式：

set 的内部实现是一个 value永远为null的HashMap，实际就是通过计算hash的方式来快速排重的，这也是set能提供判断一个成员是否在集合内的原因。

### Sort Set zset

常用命令：

zadd,zrange,zrem,zcard等

使用场景：

以某个条件为权重，比如按顶的次数排序.

ZREVRANGE命令可以用来按照得分来获取前100名的用户，ZRANK可以用来获取用户排名，非常直接而且操作容易。

Redis sorted set的使用场景与set类似，区别是set不是自动有序的，而sorted set可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。

比如:twitter 的public timeline可以以发表时间作为score来存储，这样获取时就是自动按时间排好序的。

比如:全班同学成绩的SortedSets，value可以是同学的学号，而score就可以是其考试得分，这样数据插入集合的，就已经进行了天然的排序。

另外还可以用Sorted Sets来做带权重的队列，比如普通消息的score为1，重要消息的score为2，然后工作线程可以选择按score的倒序来获取工作任务。让重要的任务优先执行。

需要精准设定过期时间的应用

比如你可以把上面说到的sorted set的score值设置成过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据了，不仅是清除Redis中的过期数据，你完全可以把Redis里这个过期时间当成是对数据库中数据的索引，用Redis来找出哪些数据需要过期删除，然后再精准地从数据库中删除相应的记录。

实现方式：

Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

### 消息订阅 Pub/Sub

Pub/Sub 从字面上理解就是发布（Publish）与订阅（Subscribe），在Redis中，你可以设定对某一个key值进行消息发布及消息订阅，

当一个key值上进行了消息发布后，所有订阅它的客户端都会收到相应的消息。这一功能最明显的用法就是用作实时消息系统，比如普通的即时聊天，群聊等功能。

客户端1：subscribe rain

客户端2：PUBLISH rain "my love!!!" 消息标题为 rain 消息体为"my love!!!"

(integer) 2 代表有几个客户端订阅了这个消息

### transaction

谁说NoSQL都不支持事务，虽然Redis的Transactions提供的并不是严格的ACID的事务（比如一串用EXEC提交执行的命令，在执行中服务器宕机，那么会有一部分命令执行了，剩下的没执行），但是这个Transactions还是提供了基本的命令打包执行的功能（在服务器不出问题的情况下，可以保证一连串的命令是顺序在一起执行的，中间有会有其它客户端命令插进来执行）。

Redis还提供了一个Watch功能，你可以对一个key进行Watch，然后再执行Transactions，在这过程中，如果这个Watched的值进行了修改，那么这个Transactions会发现并拒绝执行。

Session 1

(1)第1步

redis 127.0.0.1:6379> get age

"10"

redis 127.0.0.1:6379> watch age

OK

redis 127.0.0.1:6379> multi

OK

redis 127.0.0.1:6379>

Session 2

(2)第2步

redis 127.0.0.1:6379> set age 30

OK

redis 127.0.0.1:6379> get age

"30"

redis 127.0.0.1:6379>

Session 1

(3)第3步

redis 127.0.0.1:6379> set age 20

QUEUED

redis 127.0.0.1:6379> exec

(nil)

redis 127.0.0.1:6379> get age

"30"

redis 127.0.0.1:6379>

第一步，Session 1 还没有来得及对age的值进行修改

　　第二步，Session 2 已经将age的值设为30

第三步，Session 1 希望将age的值设为20，但结果一执行返回是nil，说明执行失败，之后我们再取一下age的值是30，这是由于Session 1中对age加了乐观锁导致的。

## python redis模块

### 基础

调用redis数据库

import redis

r = redis.StrictRedis(host='localhost', port=6379, db=0)

r.set('foo', 'bar') #True

r.get('foo') #'bar'

import redis

r = redis.Redis(host='localhost', port=6379, db=0) #如果设置了密码，就加上password=密码

r.set('foo', 'bar') #或者写成 r['foo'] = 'bar'

r.get('foo')

r.delete('foo') #True

r.dbsize() #库里有多少key，多少条数据 0

r['test']='OK!'

r.save() #强行把数据库保存到硬盘。保存时阻塞 True

r.flushdb() #删除当前数据库的所有数据 True

r.exists('chang') #看是否存在这个键值 False

r.keys() # 列出所有键值。（这时候已经存了4个了）['aaa', 'test', 'bbb', 'key1']

### 消息订阅

客户端 发送消息

>>> import redis

>>> r = redis.Redis(host='localhost',port=6379,db=0);

>>> r.publish('foo', 'hello world');

服务端

import redis

rc = redis.Redis(host='127.0.0.1')

ps = rc.pubsub()

ps.subscribe(['foo', 'bar']) //订阅两个频道，分别是foo，或bar

for item in ps.listen():

if item['type'] == 'message':

print item['data']

客户端 接受消息

import redis

rc = redis.Redis(host='127.0.0.1')

ps = rc.pubsub()

ps.subscribe(['foo', 'bar']) //订阅两个频道

### pipeline

管道（pipeline）是redis在提供单个请求中缓冲多条服务器命令的基类的子类。用于同时提交多条命令(批量执行)

p = r.pipeline() --创建一个管道

p.set('hello','redis')

p.sadd('faz','baz')

p.incr('num')

p.execute() # #会一次性提交所有的命令到服务器

[True, 1, 1]

r.get('hello') #这个命令是单独的，用来跟上面进行对比

'redis'

管道的命令可以写在一起，如：

p.set('hello','redis').sadd('faz','baz').incr('num').execute()

## linux下redis安装

### 整个流程

安装前准备，安装gcc

先用 gcc -v命令检测本机是否安装gcc，如果没有则用下面命令安装：

yum install cpp

yum install binutils

yum install glibc

yum install glibc-kernheaders

yum install glibc-common

yum install glibc-devel

yum install gcc

yum install make

安装包放在/usr/local/etc

http://www.cnblogs.com/\_popc/p/3684835.html

第一部分：安装redis

希望将redis安装到此目录

/usr/local/redis

希望将安装包下载到此目录

/usr/local/src

那么安装过程指令如下：

$ mkdir /usr/local/redis

$ cd /usr/local/src

$ wget http://redis.googlecode.com/files/redis-2.6.14.tar.gz

$ tar xzf redis-2.6.14.tar.gz

$ ln -s redis-2.6.14 redis #建立一个链接

$ cd redis

$ make PREFIX=/usr/local/redis install #安装到指定目录中

注意上面的最后一行，我们通过PREFIX指定了安装的目录。如果make失败，一般是你们系统中还未安装gcc,那么可以通过yum安装：

yum install gcc

安装完成后，继续执行make.

在安装redis成功后，你将可以在/usr/local/redis看到一个bin的目录，里面包括了以下文件：

redis-benchmark redis-check-aof redis-check-dump redis-cli redis-server

第二部分：将redis做成一个服务

1.复制脚本到/etc/rc.d/init.d目录

ps: /etc/rc.d/init.d/目录下的脚本就类似与windows中的注册表，在系统启动的时候某些指定脚本将被执行

按以上步骤安装Redis时，其服务脚本位于：

/usr/local/src/redis/utils/redis\_init\_script

必须将其复制到/etc/rc.d/init.d的目录下：

cp /usr/local/src/redis/utils/redis\_init\_script /etc/rc.d/init.d/redis

将redis\_init\_script复制到/etc/rc.d/init.d/，同时易名为redis。

如果这时添加注册服务：

chkconfig --add redis

将报以下错误：

redis服务不支持chkconfig

为此，我们需要更改redis脚本。

2.更改redis脚本

打开使用vi打开脚本，查看脚本信息：

vim /etc/rc.d/init.d/redis

看到的内容如下(下内容是更改好的信息)：

#!/bin/sh

#chkconfig: 2345 80 90

# Simple Redis init.d script conceived to work on Linux systems

# as it does use of the /proc filesystem.

REDISPORT=6379

EXEC=/usr/local/redis/bin/redis-server

CLIEXEC=/usr/local/redis/bin/redis-cli

PIDFILE=/var/run/redis\_${REDISPORT}.pid

CONF="/etc/redis/${REDISPORT}.conf"

case "$1" in

start)

if [ -f $PIDFILE ]

then

echo "$PIDFILE exists, process is already running or crashed"

else

echo "Starting Redis server..."

$EXEC $CONF &

fi

;;

stop)

if [ ! -f $PIDFILE ]

then

echo "$PIDFILE does not exist, process is not running"

else

PID=$(cat $PIDFILE)

echo "Stopping ..."

$CLIEXEC -p $REDISPORT shutdown

while [ -x /proc/${PID} ]

do

echo "Waiting for Redis to shutdown ..."

sleep 1

done

echo "Redis stopped"

fi

;;

\*)

echo "Please use start or stop as first argument"

;;

esac

和原配置文件相比：

1.原文件是没有以下第2行的内容的，

#chkconfig: 2345 80 90

2.原文件EXEC、CLIEXEC参数，也是有所更改。

EXEC=/usr/local/redis/bin/redis-server

CLIEXEC=/usr/local/redis/bin/redis-cli

3.redis开启的命令，以后台运行的方式执行。

$EXEC $CONF &

ps:注意后面的那个“&”，即是将服务转到后面运行的意思，否则启动服务时，Redis服务将 占据在前台，占用了主用户界面，造成其它的命令执行不了。

3.将redis配置文件拷贝到/etc/redis/${REDISPORT}.conf

mkdir /etc/redis

cp /usr/local/src/redis/redis.conf /etc/redis/6379.conf

这样，redis服务脚本指定的CONF就存在了。默认情况下，Redis未启用认证，可以通过开启6379.conf的requirepass 指定一个验证密码。

以上操作完成后，即可注册redis服务：

chkconfig --add redis

4.启动redis服务

service redis start

第三，将Redis的命令所在目录添加到系统参数PATH中

修改profile文件：

vi /etc/profile

在最后行追加:

export PATH="$PATH:/usr/local/redis/bin"

然后马上应用这个文件：

. /etc/profile

这样就可以直接调用redis-cli的命令了，如下所示：

$ redis-cli

redis 127.0.0.1:6379> auth superman

OK

redis 127.0.0.1:6379> ping

PONG

redis 127.0.0.1:6379>

至此，redis 就成功安装了。

### 错误解决

1.sh: ./mkreleasehdr.sh: Permission denied

权限不够，在src目录下执行：chmod 777 /mkreleasehdr.sh

2.error: jemalloc/jemalloc.h: No such file or directory

执行：make MALLOC=libc

# scrapy

scrapy startproject tutorial 创建项目

scrapy crawl wikidoc 在该项目路径下执行

## eclipse运行scrapy

配置Debug Configurations

1、进入run——>Debug Configurations-——>Main

name中输入这个配置的名称；

Projecct选择你的Scrapy项目

Main Module 中填入scrapy库下的cmdline.py文件

C:\Python27\Lib\site-packages\scrapy\cmdline.py

2、进入run——>Debug Configurations-——>Arguments

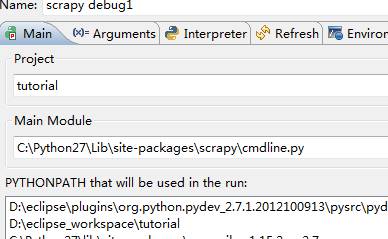
在Program arguments中输入 crawl example，需要注意这个example是你的爬虫的名称

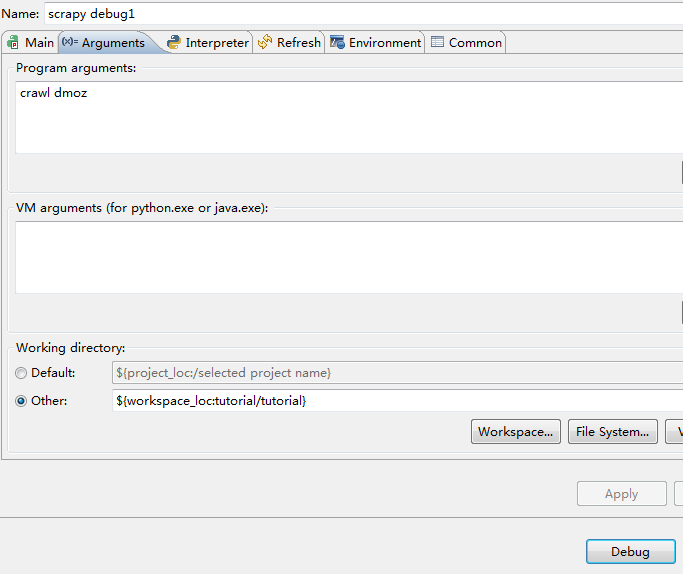
比如crawl dmoz

这个名字并不是你的项目名称，而是在你所写的类似于xxxSpider()类中的name的值。

在Working directory中选择other 选择你的爬虫工作目录

${workspace\_loc:tutorial/tutorial}





配置完成，点击Debug按钮，如果能正常显示类似于下面的信息则说明配置成功。

2015-07-04 00:10:10 [scrapy] INFO: Closing spider (finished)

2015-07-04 00:10:10 [scrapy] INFO: Dumping Scrapy stats:

唯一麻烦一点的就是每次debug都需要通过run进行，而不能使用鼠标右键运行。

至此eclipse + pydev开发环境debug Scrapy就全部完成了。

下次运行该项目就可直接点击菜单栏 run-run 即可

## 简单例子

我们使用dmoz.org这个网站来作为小抓抓一展身手的对象。

首先先要回答一个问题。

问：把网站装进爬虫里，总共分几步？

答案很简单，四步：

新建项目 (Project)：新建一个新的爬虫项目

明确目标（Items）：明确你想要抓取的目标

制作爬虫（Spider）：制作爬虫开始爬取网页

存储内容（Pipeline）：设计管道存储爬取内容

好的，基本流程既然确定了，那接下来就一步一步的完成就可以了。

### 新建项目（Project）

在空目录下按住Shift键右击，选择“在此处打开命令窗口”，输入一下命令：

scrapy startproject tutorial

其中，tutorial为项目名称。

可以看到将会创建一个tutorial文件夹，目录结构如下：

tutorial/

scrapy.cfg

tutorial/

\_\_init\_\_.py

items.py

pipelines.py

settings.py

spiders/

\_\_init\_\_.py

...

下面来简单介绍一下各个文件的作用：

scrapy.cfg：项目的配置文件

tutorial/：项目的Python模块，将会从这里引用代码

tutorial/items.py：项目的items文件

tutorial/pipelines.py：项目的pipelines文件

tutorial/settings.py：项目的设置文件

tutorial/spiders/：存储爬虫的目录

然后在eclipse中new一个python项目，name取为tutorial

### 明确目标（Item）

在Scrapy中，items是用来加载抓取内容的容器，有点像Python中的Dic，也就是字典，但是提供了一些额外的保护减少错误。

一般来说，item可以用scrapy.item.Item类来创建，并且用scrapy.item.Field对象来定义属性（可以理解成类似于ORM的映射关系）。

接下来，我们开始来构建item模型（model）。

首先，我们想要的内容有：

名称（name）

链接（url）

描述（description）

修改tutorial目录下的items.py文件，在原本的class后面添加我们自己的class。

因为要抓dmoz.org网站的内容，所以我们可以将其命名为DmozItem：

# Define here the models for your scraped items

#

# See documentation in:

# http://doc.scrapy.org/en/latest/topics/items.html

from scrapy.item import Item, Field

class TutorialItem(Item):

# define the fields for your item here like:

# name = Field()

pass

class DmozItem(Item):

title = Field()

link = Field()

desc = Field()

刚开始看起来可能会有些看不懂，但是定义这些item能让你用其他组件的时候知道你的 items到底是什么。

可以把Item简单的理解成封装好的类对象。

### 制作爬虫（Spider）

制作爬虫，总体分两步：先爬再取。

也就是说，首先你要获取整个网页的所有内容，然后再取出其中对你有用的部分。

#### 爬

Spider是用户自己编写的类，用来从一个域（或域组）中抓取信息。

他们定义了用于下载的URL列表、跟踪链接的方案、解析网页内容的方式，以此来提取items。

要建立一个Spider，你必须用scrapy.spider.BaseSpider创建一个子类，并确定三个强制的属性：

name：爬虫的识别名称，必须是唯一的，在不同的爬虫中你必须定义不同的名字。

start\_urls：爬取的URL列表。爬虫从这里开始抓取数据，所以，第一次下载的数据将会从这些urls开始。其他子URL将会从这些起始URL中继承性生成。

parse()：解析的方法，调用的时候传入从每一个URL传回的Response对象作为唯一参数，负责解析并匹配抓取的数据(解析为item)，跟踪更多的URL。

这里可以参考宽度爬虫教程中提及的思想来帮助理解，教程传送：[Java] 知乎下巴第5集：使用HttpClient工具包和宽度爬虫。

也就是把Url存储下来并依此为起点逐步扩散开去，抓取所有符合条件的网页Url存储起来继续爬取。

下面我们来写第一只爬虫，命名为dmoz\_spider.py，保存在tutorial\spiders目录下。

dmoz\_spider.py代码如下：

from scrapy.spider import Spider

class DmozSpider(Spider):

name = "dmoz"

allowed\_domains = ["dmoz.org"]

start\_urls = [

"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",

"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"

]

def parse(self, response):

filename = response.url.split("/")[-2]

open(filename, 'wb').write(response.body)

allow\_domains是搜索的域名范围，也就是爬虫的约束区域，规定爬虫只爬取这个域名下的网页。

从parse函数可以看出，将链接的最后两个地址取出作为文件名进行存储。

然后运行一下看看，在tutorial目录下按住shift右击，在此处打开命令窗口，输入：

scrapy crawl dmoz

报错了：

UnicodeDecodeError: 'ascii' codec can't decode byte 0xb0 in position 1: ordinal not in range(128)

运行第一个Scrapy项目就报错，真是命运多舛。

应该是出了编码问题，谷歌了一下找到了解决方案：

在python的Lib\site-packages文件夹下新建一个sitecustomize.py：

import sys

sys.setdefaultencoding('gb2312')

最后一句INFO: Closing spider (finished)表明爬虫已经成功运行并且自行关闭了。

包含 [dmoz]的行 ，那对应着我们的爬虫运行的结果。

可以看到start\_urls中定义的每个URL都有日志行。

还记得我们的start\_urls吗？

http://www.dmoz.org/Computers/Programming/Languages/Python/Books

http://www.dmoz.org/Computers/Programming/Languages/Python/Resources

因为这些URL是起始页面，所以他们没有引用(referrers)，所以在它们的每行末尾你会看到 (referer: <None>)。

在parse 方法的作用下，两个文件被创建：分别是 Books 和 Resources，这两个文件中有URL的页面内容。

那么在刚刚的电闪雷鸣之中到底发生了什么呢？

首先，Scrapy为爬虫的 start\_urls属性中的每个URL创建了一个 scrapy.http.Request 对象 ，并将爬虫的parse 方法指定为回调函数。

然后，这些 Request被调度并执行，之后通过parse()方法返回scrapy.http.Response对象，并反馈给爬虫。

#### 取

爬取整个网页完毕，接下来的就是的取过程了。

光存储一整个网页还是不够用的。

在基础的爬虫里，这一步可以用正则表达式来抓。

在Scrapy里，使用一种叫做 XPath selectors的机制，它基于 XPath表达式。

这是一些XPath表达式的例子和他们的含义

/html/head/title: 选择HTML文档<head>元素下面的<title> 标签。

/html/head/title/text(): 选择前面提到的<title> 元素下面的文本内容

//td: 选择所有 <td> 元素

//div[@class="mine"]: 选择所有包含 class="mine" 属性的div 标签元素

以上只是几个使用XPath的简单例子，但是实际上XPath非常强大。

为了方便使用XPaths，Scrapy提供XPathSelector 类，有两种可以选择，HtmlXPathSelector(HTML数据解析)和XmlXPathSelector(XML数据解析)。

必须通过一个 Response 对象对他们进行实例化操作。

你会发现Selector对象展示了文档的节点结构。因此，第一个实例化的selector必与根节点或者是整个目录有关 。

在Scrapy里面，Selectors 有四种基础的方法（点击查看API文档）：

xpath()：返回一系列的selectors，每一个select表示一个xpath参数表达式选择的节点

css()：返回一系列的selectors，每一个select表示一个css参数表达式选择的节点

extract()：返回一个unicode字符串，为选中的数据

re()：返回一串一个unicode字符串，为使用正则表达式抓取出来的内容

#### xpath实验

下面我们在Shell里面尝试一下Selector的用法。

实验的网址：http://www.dmoz.org/Computers/Programming/Languages/Python/Books/

熟悉完了实验的小白鼠，接下来就是用Shell爬取网页了。

进入到项目的顶层目录，也就是第一层tutorial文件夹下，在cmd中输入：

scrapy shell <http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>

在Shell载入后，你将获得response回应，存储在本地变量 response中。

所以如果你输入response.body，你将会看到response的body部分，也就是抓取到的页面内容：

或者输入response.headers 来查看它的 header部分：

现在就像是一大堆沙子握在手里，里面藏着我们想要的金子，所以下一步，就是用筛子摇两下，把杂质出去，选出关键的内容。

selector就是这样一个筛子。

在旧的版本中，Shell实例化两种selectors，一个是解析HTML的 hxs 变量，一个是解析XML 的 xxs 变量。

而现在的Shell为我们准备好的selector对象，sel，可以根据返回的数据类型自动选择最佳的解析方案(XML or HTML)。

然后我们来捣弄一下！~

要彻底搞清楚这个问题，首先先要知道，抓到的页面到底是个什么样子。

比如，我们要抓取网页的标题，也就是<title>这个标签：

sel.xpath('//title')

这样就能把这个标签取出来了，用extract()和text()还可以进一步做处理。

备注：简单的罗列一下有用的xpath路径表达式：

表达式 描述

nodename 选取此节点的所有子节点。

/ 从根节点选取。

// 从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。

. 选取当前节点。

.. 选取当前节点的父节点。

@ 选取属性。

全部的实验结果如下，In[i]表示第i次实验的输入，Out[i]表示第i次结果的输出

我们可以用如下代码来抓取这个<li>标签：

sel.xpath('//ul/li')

从<li>标签中，可以这样获取网站的描述：

sel.xpath('//ul/li/text()').extract()

可以这样获取网站的标题：

sel.xpath('//ul/li/a/text()').extract()

可以这样获取网站的超链接：

sel.xpath('//ul/li/a/@href').extract()

当然，前面的这些例子是直接获取属性的方法。

我们注意到xpath返回了一个对象列表，

那么我们也可以直接调用这个列表中对象的属性挖掘更深的节点

sites = sel.xpath('//ul/li')

for site in sites:

title = site.xpath('a/text()').extract()

link = site.xpath('a/@href').extract()

desc = site.xpath('text()').extract()

print title, link, desc

#### xpath实战

我们用shell做了这么久的实战，最后我们可以把前面学习到的内容应用到dmoz\_spider这个爬虫中。

在原爬虫的parse函数中做如下修改：

from scrapy.spider import Spider

from scrapy.selector import Selector

class DmozSpider(Spider):

name = "dmoz"

allowed\_domains = ["dmoz.org"]

start\_urls = [

"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",

"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"

]

def parse(self, response):

sel = Selector(response)

sites = sel.xpath('//ul/li')

for site in sites:

title = site.xpath('a/text()').extract()

link = site.xpath('a/@href').extract()

desc = site.xpath('text()').extract()

print title

注意，我们从scrapy.selector中导入了Selector类，并且实例化了一个新的Selector对象。这样我们就可以像Shell中一样操作xpath了。

我们来试着输入一下命令运行爬虫（在tutorial根目录里面）：

scrapy crawl dmoz

看来是我们的xpath语句有点问题，没有仅仅把我们需要的项目名称抓取出来，也抓了一些无辜的但是xpath语法相同的元素。

审查元素我们发现我们需要的<ul>具有class='directory-url'的属性，

那么只要把xpath语句改成sel.xpath('//ul[@class="directory-url"]/li')即可

将xpath语句做如下调整：

from scrapy.spider import Spider

from scrapy.selector import Selector

class DmozSpider(Spider):

name = "dmoz"

allowed\_domains = ["dmoz.org"]

start\_urls = [

"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",

"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"

]

def parse(self, response):

sel = Selector(response)

sites = sel.xpath('//ul[@class="directory-url"]/li')

for site in sites:

title = site.xpath('a/text()').extract()

link = site.xpath('a/@href').extract()

desc = site.xpath('text()').extract()

print title

成功抓出了所有的标题，绝对没有滥杀无辜：

#### 使用Item

接下来我们来看一看如何使用Item。

前面我们说过，Item 对象是自定义的python字典，可以使用标准字典语法获取某个属性的值：

>>> item = DmozItem()

>>> item['title'] = 'Example title'

>>> item['title']

'Example title'

作为一只爬虫，Spiders希望能将其抓取的数据存放到Item对象中。为了返回我们抓取数据，spider的最终代码应当是这样:

from scrapy.spider import Spider

from scrapy.selector import Selector

from tutorial.items import DmozItem

class DmozSpider(Spider):

name = "dmoz"

allowed\_domains = ["dmoz.org"]

start\_urls = [

"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",

"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"

]

def parse(self, response):

sel = Selector(response)

sites = sel.xpath('//ul[@class="directory-url"]/li')

items = []

for site in sites:

item = DmozItem()

item['title'] = site.xpath('a/text()').extract()

item['link'] = site.xpath('a/@href').extract()

item['desc'] = site.xpath('text()').extract()

items.append(item)

return items

#### 存储内容（Pipeline）

保存信息的最简单的方法是通过Feed exports，主要有四种：JSON，JSON lines，CSV，XML。

我们将结果用最常用的JSON导出，命令如下：

scrapy crawl dmoz -o items.json -t json

-o 后面是导出文件名，-t 后面是导出类型。

然后来看一下导出的结果，用文本编辑器打开json文件即可（为了方便显示，在item中删去了除了title之外的属性）：

## scrapy+flask+mongodb方案

### 第一步：定义Model层

打开items.py我们修改后的代码是这样的。

from scrapy.item import Item, Field

class MeiziItem(Item):

# define the fields for your item here like:

# name = scrapy.Field()

datasrc = Field()

title = Field()

dataid = Field()

startcount = Field()

我们定义了一个class叫做MeiziItem，继承自scrapy的类Item，然后为我们的类定义了4个属性，datasrc用来保存等下拉去的image-url，title用来保存图片的title，dataid用来记录图片id防止重复图片，starcount是喜欢次数

### 第二步：编写爬虫

打开spiders文件夹，新建一个dbmeizi\_scrapy.py文件。在文件里写如下内容.

from scrapy import Spider

from scrapy.selector import Selector

from dbmeizi.items import MeiziItem

class dbmeiziSpider(Spider):

name = "dbmeiziSpider"

allowed\_domains =["dbmeizi.com"]

start\_urls = [

"http://www.dbmeizi.com",

]

def parse(self, response):

liResults = Selector(response).xpath('//li[@class="span3"]')

for li in liResults:

for img in li.xpath('.//img'):

item = MeiziItem()

item['title'] = img.xpath('@data-title').extract()

item['dataid'] = img.xpath('@data-id').extract()

item['datasrc'] = img.xpath('@data-src').extract()

item['startcount'] = 0

yield item

首先我们定义了一个类名为dbmeiziSpider的类，继承自Spider类，然后这个类有3个基础的属性，name表示这个爬虫的名字，等一下我们在命令行状态启动爬虫的时候，爬虫的名字就是name规定的。

allowed\_domains意思就是指在dbmeizi.com这个域名爬东西。

start\_urls是一个数组，里面用来保存需要爬的页面，目前我们只需要爬首页。所以只有一个地址。

然后def parse就是定义了一个parse方法（肯定是override的，我觉得父类里肯定有一个同名方法），然后在这里进行解析工作，这个方法有一个response参数，你可以把response想象成，scrapy这个框架在把start\_urls里的页面下载了，然后response里全部都是html代码和css代码。

然后， liResults = Selector(response).xpath('//li[@class="span3"]') 这句话什么意思呢？

这句话的意思就是从html里的最开始一直到最后，搜索所有class="span3"的li标签。

我们获得了li这个标签后，再通过 for img in li.xpath('.//img'): 这个循环取出所有img标签，最后一个for循环里新建我们之前创建的MeiziItem，然后赋值。

parse()：是spider的一个方法。被调用时，每个初始URL完成下载后生成的 Response 对象将会作为唯一的参数传递给该函数。该方法负责解析返回的数据(response data)，提取数据(生成item)以及生成需要进一步处理的URL的 Request 对象。

yield的功能类似于return，但是不同之处在于它返回的是生成器。

如果一个函数包含yield关键字，这个函数就会变为一个生成器。

生成器并不会一次返回所有结果，而是每次遇到yield关键字后返回相应结果，并保留函数当前的运行状态，等待下一次的调用。

### 第三步：将我们的MeiziItem存入数据库中

打开settings.py，我们修改成如下样子。

BOT\_NAME = 'dbmeizi'

SPIDER\_MODULES = ['dbmeizi.spiders']

NEWSPIDER\_MODULE = 'dbmeizi.spiders'

ITEM\_PIPELINES = ['dbmeizi.pipelines.MongoDBPipeline',]

MONGODB\_SERVER = "localhost"

MONGODB\_PORT = 27017

MONGODB\_DB = "dbmeizi"

MONGODB\_COLLECTION = "meizi"

请注意ITEM\_PIPELINES那个常数，这个常数是用来规定我们获取爬虫爬下来的数据并将之转为model类型之后应该做什么事。很简单，这个常数告诉我们转为model类型之后区pipelines.py里找MongoDBPipeline这个class。这个class会替我们完成接下来的工作。

下面几个MONGODB打头的常数用来规定MONGODB的一些基本信息，比如server是哪个啊，端口是哪个之类的

### 管道管理

我们建立了爬虫去抓取和解析HTML，而且已经设置了数据库配置。现在要在pipelines.py中通过一个管道连接两个部分。

连接数据库 首先，让我们定义一个函数去连接数据库：

在settings.py里修改之后，我们打开pipelines.py。修改如下。

import pymongo

from scrapy.conf import settings

from scrapy.exceptions import DropItem

from scrapy import log

class MongoDBPipeline(object):

def \_\_init\_\_(self):

connection = pymongo.MongoClient(

settings['MONGODB\_SERVER'],

settings['MONGODB\_PORT']

)

db = connection[settings['MONGODB\_DB']]

self.collection = db[settings['MONGODB\_COLLECTION']]

def process\_item(self, item, spider):

valid = True

for data in item:

if not data:

valid = False

raise DropItem("Missing {0}!".format(data))

if valid:

self.collection.insert(dict(item))

log.msg("Meizi added to MongoDB database!",

level=log.DEBUG, spider=spider)

return item

我们定义了一个类，名字叫做MongoDBPipeline，注意，这个名字是与我们在settings里的ITEM\_PIPELINES这个常数里面规定的时一致的。注意别写错了。

\_\_init\_\_就是python中的构造方法，这个方法里我们主要是建立于mongodb的链接，所以用上了pymongo这个框架。然后将一个collection作为类的一个属性，在mongodb中，一个collection你可以想象成mysql中的一个table。

在process\_item这个方法中，我们主要是将item存入数据库。注意self.collection.insert(dict(item))这句话，就是将我们的MeiziItem存入数据库。

### 建立flask

from flask import Flask, request

import json

from bson import json\_util

from bson.objectid import ObjectId

import pymongo

app = Flask(\_\_name\_\_)

mongoClient = pymongo.MongoClient('localhost', 27017)

db = mongoClient['dbmeizi']

#定义了一个函数，用来把mongodb里的数据转换为json格式。用来返回给我们的ios客户端。

def toJson(data):

return json.dumps(data, default=json\_util.default)

@app.route('/meizi/', methods=['GET'])

def findmeizi():

if request.method == 'GET':

lim = int(request.args.get('limit', 10))

off = int(request.args.get('offset'),0)

results = db['meizi'].find().skip(off).limit(lim)

json\_results= []

for result in results:

json\_results.append(result)

**return toJson**(json\_results)

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

app.run(host="0.0.0.0",port=5000,debug=True)

利用我们的db（就是刚才利用pymongo获取的mongodb实例），取出‘meizi’这个collection，skip（off）的意思就是跳过前面多少行，limit（lim）表示从数据库取出多少个值。

整句话的意思就是，从meizi这个collection里跳过前off个值，取后面的lim个值。

现在取到的数据都在results变量里，我们遍历results，放入json\_results这个数组里，然后把数组转换成json格式返回给客户端。

app = Flask(\_\_name\_\_)这句话就是利用Flask的构造方法生成一个Flask实例，name是什么？简单来说，你创建的任何python文件（.py），都会有一个内置属性，叫做\_\_name\_\_，他有两个用途，如果你在命令行状态下直接运行`python .py`的时候，这个时候这个python文件里的\_\_name\_\_就是\_\_main\_\_,如果你是在别的python文件里import \*.py，那么这个name的东西就是这个Python文件的文件名。so,这个东西常常用来判断，你是在import还是直接在命令行里运行这个文件。

网址：<http://127.0.0.1:5000/?limit=2&offset=2>

## CrawlSpider 自动爬取

它是Spider的派生类，首先在说下Spider，它是所有爬虫的基类，对于它的设计原则是只爬取start\_url列表中的网页，而从爬取的网页中获取link并继续爬取的工作CrawlSpider类更适合。

它与Spider类的最大不同是多了一个rules参数，其作用是定义提取动作。在rules中包含一个或多个Rule对象，Rule类与CrawlSpider类都位于scrapy.contrib.spiders模块中。

class wikidocSpider(Spider): class wikidocSpider(CrawlSpider):

from scrapy.spiders import CrawlSpider,Rule

from scrapy.contrib.spiders import CrawlSpider,Rule 这个scrapy.contrib.spiders已弃用

from scrapy.contrib.linkextractors.sgml import SgmlLinkExtractor

通过SmglLinkExtractor提取希望获取的链接。

allow：满足括号中“正则表达式”的值会被提取，如果为空，则全部匹配。

deny：与这个正则表达式(或正则表达式列表)不匹配的URL一定不提取。

allow\_domains：会被提取的链接的domains。

deny\_domains：一定不会被提取链接的domains。

restrict\_xpaths：使用xpath表达式，和allow共同作用过滤链接。

follow：指定了根据该规则从response提取的链接是否需要跟进。当callback为None,默认值为true。

搜索起始链接下面符合allow中正则表达式的链接，并跟进解析，如果follow = False，则只会解析起始链接中找到的符合要求的链接

process\_links：主要用来过滤由link\_extractor获取到的链接。

process\_request：主要用来过滤在rule中提取到的request。

rules = [

Rule(SgmlLinkExtractor(allow=('/u012150179/article/details'),

restrict\_xpaths=('//li[@class="next\_article"]')),

callback='parse\_item',

follow=True)

]

## pymongo模块操作mongodb

from scrapy.conf import settings

import pymongo

client=pymongo.MongoClient('127.0.0.1',27017)

db=client.wikidoc

collection=db.wikiurl

只有在pipelines下调用from scrapy.conf import settings才有效。其他文件无法实现

def \_\_init\_\_(self):

connection=pymongo.MongoClient(

settings['MONGODB\_SERVER'],

settings['MONGODB\_PORT']

)

db=connection[settings['MONGODB\_DB']]

self.collection=db[settings['MONGODB\_COLLECTION']]

#//插入数据,\_id自动创建

post = {"id": "1",

"date": time.strftime('%Y-%m-%d %H:%M:%S')}

posts = db.posts

posts.insert(post) #把post数据插入posts聚合(表)中，返回一个ObjectId('...')

#//批量插入(一个列表里面包含了2个字典),\_id自动创建

new\_posts = [{"id": "2",

"date": time.strftime('%Y-%m-%d %H:%M:%S')},

{"id": "3",

"date": time.strftime('%Y-%m-%d %H:%M:%S')}]

posts = db.posts

posts.insert(new\_posts) #把new\_posts数据插入posts聚合(表)中，返回2个ObjectId('...')

#//删除数据

db.posts.remove() #删除posts聚合(表)中所有数据

db.posts.remove({'id':1}) #删除posts聚合(表)中id为1的数据

#//更新数据

db.posts.update({'id':1},{"$set":{"text":"cscscascs"}}) #更新一个value

db.posts.update({'id':1},{"$set":{"text":"cscscascs","title":"test title"}}) #更新多个value

#//查询数据

db.collection\_names() #查询所有聚合名称

db.posts.count() #统计posts聚合中的数据数量

db.posts.find() #查询posts中所有内容

db.posts.find\_one({"author":"Mike"}) #根据条件查询posts中数据

db.posts.find({"author":"Mike"}).sort('author') #--默认为升序

db.posts.find({"author":"Mike"}).sort('author',pymongo.ASCENDING) #升序

db.posts.find({"author":"Mike"}).sort('author',pymongo.DESCENDING) #降序

.find\_one() 显示满足条件的第一个 collection，find() 的结果则是一个满足条件的对象数组

指定大于小于等于等条件进行查询：

db.posts.find({“age”: {“$lt”: 30}})

#//插入数据成功

#//利用{'author':'Mike'} 测试查询正常

>>> posts.find\_one({'author':'Mike'})

{u'\_id': ObjectId('53bd5a5fe138235f74b67563'), u'author': u'Mike',....}

#//如何利用ObjectId来查询？

>>> from bson import ObjectId

>>> posts.find\_one({'\_id':ObjectId('53bd5a5fe138235f74b67563')})

{ u'\_id': ObjectId('53bd5a5fe138235f74b67563'), u'author': u'Mike',....}

#//原来ObjectId是一个对象，而不是一个字符串

## log日志

scrapy.log.start(logfile=None, loglevel=None, logstdout=None)

启动日志处理机制。此方法必须在实际记录任何日志信息之前被调用，否则，在调用之前已经记录的信息会被遗失掉。

Parameters:

logfile(str) - 用来日志输出到文件路径。如果忽略此参数，则会使用LOG\_FILE设置。如果两个都没有设置，那么日志将会被输出到标准错误输出。

loglevel - 最低的日志级别。可用的值有CRITICAL，ERROR，WARNING，INFO和DEBUG。

logstdout(boolean) - 如果为True，所有你应用程序到标准输出（和错误信息）都会被记录为日志。比如说，你"print 'hello'"，它将会被记录为日志。如果忽略此参数，将会使用LOG\_STDOUT设置。

scrapy.log.msg(message, level=INFO, spider=None)

记录日志信息。

Parameters:

message(str) - 要记录到日志信息。

level - 记录此日志信息到日志级别。看LOG LEVEL。

spider(Spider对象) - 记录此日志信息到spider。这个参数应该总是在记录和spider相关到信息时被使用。

scrapy可以直接使用print语句到console上。

## 解决方案

1. exceptions.ImportError: No module named win32api

安装pywin32-218.win-amd64-py2.7.exe

1. WARNING: c:\python27\lib\site-packages\scrapy\pipelines\\_\_init\_\_.py:21: ScrapyDeprecationWarning: ITEM\_PIPELINES defined as a list or a set is deprecated, switch to a dict

解决：setting文件中ITEM\_PIPELINES=['wikidoc.pipelines.MongoDBPipeline']改为

ITEM\_PIPELINES={'wikidoc.pipelines.MongoDBPipeline':300,}

1. WARNING: D:\eclipse\_workspace\wikidoc\wikidoc\pipelines.py:11: ScrapyDeprecationWarning: Module `scrapy.log` has been deprecated, Scrapy now relies on the builtin Python library for logging. Read the updated logging entry in the documentation to learn more.

pipelines文件中 使用 from scrapy import log 会提示warning,但是不影响最后结果

1. DEBUG: Crawled (403) <GET http://www.douban.com/misc/sorry?original-url=http%3A%2F%2Fmovie.douban.com%2Fsubject%2F1849031%2Fcomments%3Fsort%3Dtime> (referer: <http://movie.douban.com/subject/1849031/>)
2. 您可以使用 shell 来启动Scrapy终端:

scrapy shell <url> <url> 是您要爬取的网页的地址。Scrapy终端是一个交互终端，供您在未启动spider的情况下尝试及调试您的爬取代码。 其本意是用来测试提取数据的代码

# Xpath

## 基础

**使用Chrome的开发者工具**，可以很容易找到一个特定的Xpath。简单地检查一个特定的HTML元素，复制XPath，然后修改（如有需要）。

copy Xpath

开发者工具同时为用户提供在JavaScript控制台测试XPath选择器的功能，使用$x，如$x("//img"):

$x('//div[@class="summary"]/h3/a[@class="question-hyperlink"]/@href')

下面列出了最有用的路径表达式：

表达式 描述

nodename 选取此节点的所有子节点。

/ 从根节点选取。

// 从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。

. 选取当前节点。

.. 选取当前节点的父节点。

@ 选取属性。

实例

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

路径表达式 结果

bookstore 选取 bookstore 元素的所有子节点。

/bookstore 选取根元素 bookstore。注释：假如路径起始于正斜杠( / )，则此路径始终代表到某元素的绝对路径！

bookstore/book 选取属于 bookstore 的子元素的所有 book 元素。

//book 选取所有 book 子元素，而不管它们在文档中的位置。

bookstore//book 选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。

//@lang 选取名为 lang 的所有属性。如<p lang=''></p>

/html/head/title: 选择HTML文档中 <head> 标签内的 <title> 元素

/html/head/title/text(): 选择 <title> 元素内的文本

## 谓语（Predicates）

谓语用来查找某个特定的节点或者包含某个指定的值的节点。

谓语被嵌在方括号中。

实例

在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

路径表达式 结果

/bookstore/book[1] 选取属于 bookstore 子元素的第一个 book 元素。

/bookstore/book[last()] 选取属于 bookstore 子元素的最后一个 book 元素。

/bookstore/book[last()-1] 选取属于 bookstore 子元素的倒数第二个 book 元素。

/bookstore/book[position()<3] 选取最前面的两个属于 bookstore 元素的子元素的 book 元素。

//title[@lang] 选取所有拥有名为 lang 的属性的 title 元素。

//title[@lang='eng'] 选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。

/bookstore/book[price>35.00] 选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。

/bookstore/book[price>35.00]/title 选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

查询两个条件时：//td[text()=1365][@class="confluenceTd"]

比如选择列表最后一行的最后一个单元格：

driver.find\_element\_by\_xpath('//div[@class="wiki-content"]/table/tbody/tr[last()]/td[last()]')

## 选取未知节点

XPath 通配符可用来选取未知的 XML 元素。

通配符 描述

\* 匹配任何元素节点。

@\* 匹配任何属性节点。

node() 匹配任何类型的节点。

实例

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式 结果

/bookstore/\* 选取 bookstore 元素的所有子元素。

//\* 选取文档中的所有元素。

//title[@\*] 选取所有带有属性的 title 元素。

## 选取若干路径

通过在路径表达式中使用**“|”**运算符，您可以选取若干个路径。

实例

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式 结果

//book/title | //book/price 选取 book 元素的所有 title 和 price 元素。

//title | //price 选取文档中的所有 title 和 price 元素。

/bookstore/book/title | //price 选取属于 bookstore 元素的 book 元素的所有 title 元素，以及文档中所有的 price 元素。

## XPath 运算符

下面列出了可用在 XPath 表达式中的运算符：

运算符 描述 实例 返回值

| 计算两个节点集 //book | //cd 返回所有拥有 book 和 cd 元素的节点集

+ 加法 6 + 4 10

– 减法 6 – 4 2

\* 乘法 6 \* 4 24

div 除法 8 div 4 2

= 等于 price=9.80 如果 price 是 9.80，则返回 true。如果 price 是 9.90，则返回 false。

!= 不等于 price!=9.80 如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。

< 小于 price<9.80 如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。

<= 小于或等于 price<=9.80 如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。

> 大于 price>9.80 如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。

>= 大于或等于 price>=9.80 如果 price 是 9.90，则返回 true。如果 price 是 9.70，则返回 false。

or 或 price=9.80 or price=9.70 如果 price 是 9.80，则返回 true。如果 price 是 9.50，则返回 false。

and 与 price>9.00 and price<9.90 如果 price 是 9.80，则返回 true。如果 price 是 8.50，则返回 false。

mod 计算除法的余数 5 mod 2 1

## Xpath例子

1. 获取倒数第二个元素的内容

result = html.xpath('//li[last()-1]/a')

print result[0].text

1. 获取 class 为 bold 的标签名

result = html.xpath('//\*[@class="bold"]')

print result[0].tag

# Tkinter

导入Tkinter 时DLL load failed: %1 解决方案：卸载64位python,安装32位python

查看Tkinter是否安装正常

import \_tkinter # with underscore, and lowercase 't'

import Tkinter # no underscore, uppercase 'T' for versions prior to V3.0 python2.7

Tkinter.\_test() # note underscore in \_test and uppercase 'T' for versions prior to V3.0

Tkinter 是Python的标准GUI库。Python使用Tkinter可以快速的创建GUI应用程序。

由于Tkinter是内置到python的安装包中、只要安装好Python之后就能import Tkinter库、而且IDLE也是用Tkinter编写而成、对于简单的图形界面Tkinter还是能应付自如。

创建一个GUI程序

1、导入Tkinter模块

2、创建控件

3、指定这个控件的master， 即这个控件属于哪一个

4、告诉GM(geometry manager)有一个控件产生了。

实例:

# -\*- coding: UTF-8 -\*-

import Tkinter

top = Tkinter.Tk()

# 进入消息循环

top.mainloop()

## 控件

每个控件都得pack() 比如canvas.pack()，否则无法显示 grid情况下不需要

如果你想显示多行文本可以编辑，那么你应该使用文本部件. Text

如果你想显示一个或多个行文本不能由用户修改，那么你应该使用标签部件.Label

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tkinter 组件

Button 按钮控件；在程序中显示按钮。

**Canvas** 画布控件；显示图形元素如线条或文本

Checkbutton 多选框控件；用于在程序中提供多项选择框

Entry 输入控件；用于显示简单的文本内容

**Frame** 框架控件；在屏幕上显示一个矩形区域，**多用来作为容器**

**Label** 标签控件；可以显示文本和位图

**Listbox** 列表框控件；在Listbox窗口小部件是用来显示一个字符串列表给用户

Menubutton 菜单按钮控件，由于显示菜单项。

**Menu** 菜单控件；显示菜单栏,下拉菜单和弹出菜单

**Message** 消息控件；用来显示多行文本，与label比较类似

Radiobutton 单选按钮控件；显示一个单选的按钮状态

**Scale** 范围控件；显示一个数值刻度，为输出限定范围的数字区间

Scrollbar 滚动条控件，当内容超过可视化区域时使用，如列表框。.

**Text** 文本控件；用于显示多行文本

**Toplevel** 容器控件；用来提供一个单独的对话框，和Frame比较类似

**Spinbox** 输入控件；与Entry类似，但是可以指定输入范围值

PanedWindow PanedWindow是一个窗口布局管理的插件，可以包含一个或者多个子控件。

LabelFrame labelframe 是一个简单的容器控件。常用与复杂的窗口布局。

**tkMessageBox** 用于显示你应用程序的消息框。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

标准属性

标准属性也就是所有控件的共同属性，如大小，字体和颜色等等。

属性 描述

Dimension 控件大小； Color 控件颜色； Font 控件字体； Anchor 锚点；

Relief 控件样式； Bitmap 位图； Cursor 光标；

relief 设置浮雕效果

relief SUNKEN, RAISED, GROOVE, and RIDGE.

颜色 "white", "black", "red", "green", "blue", "cyan", "yellow", and "magenta"

或者比如 bg="#b61d1d"

focuswid.config(selectbackground='red') 控件中选中的item的背景色为红色

## 控件实例

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

from Tkinter import \*

def btn\_click():

b2['text'] = 'clicked'

evalue = e.get()

print 'btn Click and Entry value is %s' % evalue

def btn\_click\_bind(event):

print 'enter b2'

display.set('enter b2')

# Toplevel

def show\_toplevel():

top = Toplevel()

top.title('2号窗口')

Label(top, text='这是2号窗口').pack()

def menu\_click():

print 'I am menu'

# 弹出菜单

def pop(event):

submenu.post(event.x\_root, event.y\_root)

# 获取鼠标左键点击的坐标

def get\_clickpoint(event):

print event.x, event.y

root = Tk() **#定义一个窗体**

root.title('1号窗口') **#窗口名字**

root.geometry('1000x500') **#定义窗体的大小，是1000X500像素**

#显示图片gif

bm = PhotoImage(file ='jd.gif')

l = Label(root, fg='red', bg='LightBlue',text='gisy',compound = 'bottom',image=bm,width=1000, height=200)

l.image=bm

l.pack()

# 弹出Toplevel窗口

b3 = Button(root, text = 'showToplevel', command=show\_toplevel)

b3.pack()

#Spinbox 数字输入框

w = Spinbox(root, from\_=0, to=10) #限制0到10的数字输入

w.pack()

skipNum= w.get() #获取到的数字是 str格式

#Scale 可以有get() set(value) 两个方法

def sel():

selection = "Value = " + str(var1.get())

label1.config(text = selection)

var1 = DoubleVar()

scale = Scale( root, variable = var1 )

scale.pack(anchor=CENTER)

button = Button(root, text="Get Scale Value", command=sel)

button.pack(anchor=CENTER)

label1 = Label(root)

label1.pack()

#Label set config

var = StringVar()

label = Label( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")

label.config(bg='black', fg='yellow')

label.config(font=labelfont)

label.config(height=3, width=20)

label.pack()

#显示图片

from PIL import Image, ImageTk

image\_frame = Frame(root)

image\_file = Image.open("hello.jpg")

im = ImageTk.PhotoImage(image\_file)

image\_label = Label(image\_frame,image = im).pack(side = LEFT,padx = 5)

image\_frame.pack()

# tkMessageBox 跳出一个消息框

# showinfo() showwarning() showerror () askquestion() askokcancel()

#askyesno () askretrycancel ()

import tkMessageBox

def hello():

tkMessageBox.showinfo("Say Hello", "Hello World")

B1 =Button(text = "Say Hello", command = hello)

B1.pack()

#Frame 与button button放在frame内

frame = Frame(root)

frame.pack()

bottomframe = Frame(root)

bottomframe.pack( side = BOTTOM )

redbutton = Button(**frame**, text="Red", fg="red") #

redbutton.pack( side = LEFT)

blackbutton = Button(bottomframe, text="Black", fg="black")

blackbutton.pack( side = BOTTOM)

# LabelFrame

labelframe = LabelFrame(root, text="This is a LabelFrame")

labelframe.pack(fill="both", expand="yes")

left = Label(labelframe, text="Inside the LabelFrame")

left.pack()

# Message 消息框

var = StringVar()

label = Message( root, **textvariable**=var, relief=RAISED )

var.set("Hey!? How are you doing?")

label.pack()

#如果想消息框没有边框，就不要加relief=RAISED

var = StringVar(value='gisy')

#文本框Text

text = Text(root)

text.insert(INSERT, "Hello.....")

text.insert(END, "Bye Bye.....")

text.pack()

#将结果显示在输入框中

def btn(event):

#print 'enter b2'

display.set('enter b2')

# 输入框

e = Entry(root,text=display)

e.grid(row=0, column=1)

e.bind("<FocusOut>",btn)

e.pack()

# Canvas 画布

C =Canvas(root, bg="blue", height=250, width=300)

coord = 10, 50, 240, 210

arc = C.create\_arc(coord, start=0, extent=150, fill="red")

filename = PhotoImage(file = "jd.gif")

image = C.create\_image(50, 50, anchor=NE, image=filename)

C.pack()

canvas = Canvas(root, width=300, height=200, bg='LightBlue')

canvas.pack()

# Frame 显示3种颜色方块

for x in ['red', 'blue', 'yellow']:

Frame(height = 20, width = 20, bg = x).pack()

#列表Listbox

li = ['C','python','php','html','SQL','java']

listb = Listbox(root) # 创建两个列表组件

for item in li: # 第一个小部件插入数据

listb.insert(0,item)

listb.pack() # 将小部件放置到主窗口中

#root代表整个框，如果聚焦点在该项目，则有效 root = Tk()

root.mainloop() #循环

-------------------

# PanedWindow

from Tkinter import \*

m1 = PanedWindow()

m1.pack(fill=BOTH, expand=1)

left = Label(m1, text="left pane")

m1.add(left)

m2 = PanedWindow(m1, orient=VERTICAL)

m1.add(m2)

top = Label(m2, text="top pane")

m2.add(top)

bottom = Label(m2, text="bottom pane")

m2.add(bottom)

mainloop()

## 控件细节

### 键盘鼠标触发命令 bind

#### 基础

使用bind 方式关联按钮和函数

**函数设置时有event入参为** defbtn\_click\_bind(event)

当鼠标和键盘输入触发的命令，必须有event. 其他点击按钮触发命令等无须event

root.bind('<Button-3>', pop) #按鼠标右键代表 <Button-3>.按鼠标右键就调用pop函数

root.bind('<Button-1>', get\_clickpoint) #按鼠标左键代表 <Button-1>

root.bind('<Button-2>', pop) #按鼠标滚轮代表 <Button-2>

root.bind('<Button>', pop) #按鼠标触发

root.bind('<ButtonRelease>', pop) #松开鼠标按钮 触发

<ButtonRelease-1>, <ButtonRelease-2>, and <ButtonRelease-3>

root.bind('<Motion>', pop) #按住鼠标按钮并移动 触发 <B1-Motion>, <B2-Motion> and <B3-Motion>

root.bind('<Key>', btn\_click\_bind) #按下键盘任何键，触发函数

root.bind('<Double-Button>', btn\_click\_bind) #鼠标双击触发 可细分为<Double-Button-1>, <Double-Button-2>, and <Double-Button-3>

b2.bind("<Enter>", btn\_click\_bind) #代表鼠标进入b2这个控件范围内，并不是代表按下enter键。进入1次，触发1次。

b2.bind("<Leave>", btn\_click\_bind) #代表鼠标离开b2这个控件范围内

e.bind("<FocusIn>", btn\_click\_bind) #键盘焦点进入该控件e中，或者该控件的子控件中

e.bind("<FocusOut>", btn\_click\_bind) #键盘焦点离开该控件e，或者该控件的子控件中

e.bind("<Return>", btn\_click\_bind) #用户在该控件e下按下enter键按钮,比如输入框e

textbox.bind("**<Control-Key-a>**", select\_all) #ctrl +a 两者都行

textbox.bind("**<Control-a>**", hello)

#The special keys are Cancel (the Break key), BackSpace, Tab, Return(the Enter key), Shift\_L (any Shift key), Control\_L (any Control key), Alt\_L (any Alt key), Pause, Caps\_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num\_Lock, and Scroll\_Lock.

e.bind("<**Shift-Up**>", btn\_click\_bind) #用户按住shift和向上键，也可以类似使用Alt, Shift, and Control.

root.bind("<Configure>", btn\_click\_bind) #当root的width height变化，触发函数

除了特殊键，剩下的键也都可用于直接绑定事件，包括数字键，但要注意 1 与 <1> 是不同的，前者表示按键1,后者表示鼠标左键。除此之外，还有两个特殊情况：空格键使用 <Space>，小于号键使用 <less>

root.bind('<Control-slash>', quit) 按下 Ctrl-/

Control-backslash 对应 Ctrl-\

escape 对应 Escape

#### 细节

1. 取消bind命令

textbox.bind("**<Control-Key-a>**", select\_all)

textbox.unbind("**<Control-Key-a>**") 只需要输入前面的字符串即可

1. 属性

root.bind("<Control-Key-a>", ctrl\_a)

def ctrl\_a(event):

caller = event.widget 获取调用的控件

event.x, event.y说明：当前鼠标指针的位置，以象素为单位。

event. x\_root,event. y\_root 说明：当前鼠标相对于屏幕左上方的位置，以象素为单位。

event.char 说明：字符代码（仅键盘事件），是一个字符串。

event.keycode 说明：键的代码（仅键盘事件）

event.keysym 说明：键的符号（仅键盘事件）

event.num 说明：鼠标按钮的数字（仅鼠标按钮事件）

event.width,event.height 说明：窗口部件的新尺寸，以象素为单位（仅Configure事件）

1. <<Paste>> <<Copy>> <<Cut>> 代表命令可以直接使用lambda e:'<<Copy>>'当做命令
2. 使文本框只读模式，但是可以复制、全选

textbox.bind("<Key>", lambda e: "break")

textbox.bind("<Control-Key-a>", self.ctrl\_a)

textbox.bind("<Control-Key-A>", self.ctrl\_a)

textbox.bind("<Control-Key-c>", lambda e:'<<Copy>>')

textbox.bind("<Control-Key-C>", lambda e:'<<Copy>>')

1. 键盘操作命令与点击按钮命令 如何可以重复使用某些代码

def \_\_save(self):

…

def \_\_invoke\_save(self, event):

self.\_\_save()

1. #获取每次聚焦的控件，来产生针对该控件的命令

#键盘操作ctrl+a

def ctrl\_a(event):

caller = event.widget

caller .tag\_add('sel', '1.0', 'end')

root.bind("<Control-Key-a>", ctrl\_a)

root.bind("<Control-Key-A>", ctrl\_a)

1. 如何实现ctrl+z的初步功能，就是恢复输入框原值？

def ctrl\_z(self,event):

self.caller = event.widget

print self.caller

for h in range(6):

if **globals()[self.columnlist[h]]**==self.caller: #动态变量必须放相等号前面，否则系统报错

self.caller.delete('1.0',END)

self.caller.insert('1.0',self.caseCache[self.back\_next\_index][h])

### 有关菜单menu

#### 基础

#导航菜单标签显示Menu

（用add\_command方法创建，最后用add\_cascade方法加入到上级菜单中去），为每个下拉选项都绑定一个函数

xmenu = Menu(root)

submenu = Menu(xmenu, tearoff = 0)

for item in ['java', 'cpp', 'c', 'php']:

xmenu.add\_command(label = item, command = menu\_click)

for item in ['think in java', 'java web', 'android']:

submenu.add\_command(label = item, command = menu\_click)

submenu.add\_separator() #分割线

xmenu.add\_cascade(label = 'progame', menu = submenu)

root.config(menu= xmenu) #显示菜单 没有该行则无法显示

root['menu'] = xmenu #或者该行

#增加右键菜单

rightMenu = Menu(root,**tearoff=0**)

rightMenu.add\_command(label = '复制' , command = ctrl\_c)

rightMenu.add\_separator()

rightMenu.add\_command(label = '粘贴' , command = ctrl\_v)

rightMenu.add\_separator()

rightMenu.add\_command(label = '全选' , command = ctrl\_a)

def right\_click(event):

global rightMenu

rightMenu.tk\_popup(event.x\_root, event.y\_root)

#### 主菜单上复制粘贴功能键

self.editmenu.add\_command(label ='全选', command =lambda:self.root.focus\_get().tag\_add('sel', '1.0', 'end'))

self.editmenu.add\_command(label ='剪切', command=lambda:self.root.focus\_get().event\_generate('<<Cut>>'))

self.editmenu.add\_command(label ='复制', command =lambda:self.root.focus\_get().event\_generate("<<Copy>>"))

self.editmenu.add\_command(label ='粘贴', command =lambda:self.root.focus\_get().event\_generate("<<Paste>>"))

或者

self.editmenu.add\_command(label ='剪切', accelerator="Ctrl+X",

command=lambda:self.root.focus\_get().event\_generate(' <Control-x>'))

#### 右键菜单

#右键菜单

self.rightMenu = Menu(self.root,tearoff=0)

self.rightMenu.add\_command(label = '复制' ,accelerator='Ctrl+C',

command=lambda:self.focuswid.event\_generate("<<Copy>>"))

self.rightMenu.add\_separator()

self.rightMenu.add\_command(label = '粘贴' ,accelerator='Ctrl+V',

command=lambda:self.focuswid.event\_generate("<<Paste>>"))

self.rightMenu.add\_separator()

self.rightMenu.add\_command(label = '全选' ,accelerator='Ctrl+A',

command=lambda:self.focuswid.tag\_add('sel', '1.0', 'end'))

self.caseText.bind('<Button-3>',self.text\_right\_click)

#右键菜单函数

def right\_click(self,event):

self.rightMenu.tk\_popup(event.x\_root, event.y\_root)

self.focuswid=event.widget

### 有关Entry

e = Entry(root, text = 'input your name')

e.pack()

e.get() #获取输入框内容 #获取到是 str格式

e.insert ( index, 'gisy' )

e.delete ( first, last=None )

### 有关Radiobutton

1. Radiobutton

from Tkinter import \*

def sel():

selection = "You selected the option " + str(var.get())

label.config(text = selection)

root = Tk()

var = IntVar()

R1 = Radiobutton(root, text="Option 1", variable=var, value=1,command=sel)

R2 = Radiobutton(root, text="Option 2", variable=var, value=2,command=sel)

R3 = Radiobutton(root, text="Option 3", variable=var, value=3,command=sel)

R1.pack( anchor = W )

R2.pack( anchor = W )

R3.pack( anchor = W)

label = Label(root)

label.pack()

root.mainloop()

1. Radiobutton 与Text交互

#切换显示summary /显示全部节点

def switch\_dispaly():

with open(temp\_txtname, "a+") as casedoc: #重新读取文档

caseread2=casedoc.readlines()

selection = var.get()

caselist.delete('1.0',END)

if selection:

for k in range(len(caseread2)):

caselist.insert(INSERT, caseread2[k])

else:

for k in range(len(caseread2)):

m = re.match(u"\xe6\x91\x98\xe8\xa6\x81\xef\xbc.\*", caseread2[k])

if m:

caselist.insert(INSERT, caseread2[k])

var = IntVar()

#用例浏览frame 滚动条

frame3=Frame(root)

frame3.grid(row=5, column=0,columnspan=2,rowspan=10,sticky='e')

R1 = Radiobutton(frame3, text="显示summary", variable=var, value=0,command=switch\_dispaly)

R2 = Radiobutton(frame3, text="显示全部节点", variable=var, value=1,command=switch\_dispaly)

R1.select()

R1.grid(row=2, column=0, sticky=W)

R2.grid(row=2, column=0, sticky=E)

caseLabel=Label(frame3,text='用例预览')

caseLabel.grid(row=0, column=0)

caselist = Text(frame3,width=80,height=20)

caselist.grid(row=3, column=0, sticky=N+E+S+W)

yroll=Scrollbar(frame3, orient=VERTICAL) #文本框-竖向滚动条

caselist.config(yscrollcommand=yroll.set)

yroll.config(command=caselist.yview)

yroll.grid(row=3,column=1, sticky=N+S) #上下拉长

#将用例显示在文本框中

with open(temp\_txtname, "a+") as casedoc:

caseread=casedoc.readlines()

n=1

print caseread

for k in range(len(caseread)):

m = re.match(u"\xe6\x91\x98\xe8\xa6\x81\xef\xbc.\*", caseread[k])

if m:

caselist.insert(INSERT, caseread[k])

n=n+1

caselist.bind('<Button-3>',list\_right\_click)

### 有关Text

1. Text 方法

from Tkinter import \*

root = Tk()

text = Text(root)

text.insert(INSERT, "Hello.....")

text.insert(END, "Bye Bye.....")

text.pack()

text.tag\_add("here", "1.0", "1.4")

text.tag\_add("start", "1.8", "1.13")

text.tag\_config("here", background="yellow", foreground="blue")

text.tag\_config("start", background="black", foreground="green")

root.mainloop()

"1.4" Text的index是这种str表示，不能是int "1.4"表示第一行第4个 也可以是float类型

整数可以转换为float类型 float(1)

# tag\_add 划界"here"为 tagname，就是这次划出的区域名

tag\_config 给划出的区域上属性

INSERT:光标的插入点

CURRENT:鼠标的当前位置所对应的字符位置

END:这个Textbuffer的最后一个字符

1. Text获取插入文本方法

end-1c 代表减去1个字符 ，结尾必须用'end-1c',如果用.get("1.0",END)最后会多出一个空白行

def print\_content():

input = summary.get("1.0",'end-1c')

tkMessageBox.showinfo ("Say Hello",input )

#在第一行插入字符串'你好'

summary.insert('1.0','你好')

1. Text 获取行号 标红

text.index('current') #获取鼠标指针所在行号列号 5.1

text.index('insert') #获取当前光标所在行号列号 6.2

line =caselist.index('current') #获取鼠标指针所在行号列号

#caselist.tag\_remove("current\_line", 1.0, "end") #如果只想存在一行描红的行，取消该行注释

caselist.tag\_add("current\_line", "current linestart", "current lineend+1c")

caselist.tag\_config("current\_line", background="#b61d1d")

#如果是描红当前光标所在行 current都改为insert

1. 如何使Text框不能进行手动修改，但是可以程序插入行，并且人可以进行选择文本？

self.caseText.bind("<Key>", lambda e: "break")

1. 从文本框中获取文本，打印，但是每行之间都有空行？

将获取到文本字符串在首尾去除换行符

for line\_num in range(1,200):

print self.caseText.get(float(line\_num),float(line\_num+1))**.strip('\n')**

1. 如何标记与取消标记某行？

def markRed(self):

self.caseText.tag\_add("insert\_line", "insert linestart", "insert lineend+1c")

self.caseText.tag\_config("insert\_line", background="#FFCCCC")

def cancel\_markRed(self):

self.caseText.**tag\_remove**("insert\_line", "insert linestart", "insert lineend+1c")

tag\_delete(tagname) 删除某个tag

### 有关Checkbutton

1. Checkbutton

from Tkinter import \*

import tkMessageBox

import Tkinter

top = Tkinter.Tk()

CheckVar1 = IntVar()

CheckVar2 = IntVar()

C1 = Checkbutton(top, text = "Music", variable = CheckVar1, \

onvalue = 1, offvalue = 0, height=5, \

width = 20)

C2 = Checkbutton(top, text = "Video", variable = CheckVar2, \

onvalue = 1, offvalue = 0, height=5, \

width = 20)

C1.pack()

C2.pack()

top.mainloop()

Checkbutton方法

check1.deselect() 取消勾选

check2.invoke() 反转

check3.select() 勾选

check4.flash() 在active和normal状态下切换颜色，引起人注意

check5.toggle() 反转 勾选变成不勾选 不勾选变勾选

1. 如何设置全选/取消全选按钮？

函数可以嵌套函数

def config():

top = Toplevel()

top.title('设置')

def save2():

check1.toggle()

toggle=Button(top, text='全部反转', command=save2,font=configfont)

toggle.grid(row=3, column=1,sticky='w')

1. 如何使勾选框默认勾选

注意：必须 top.mainloop() 循环显示

top = Toplevel()

CheckVar1 = IntVar()

check1 = Checkbutton(top, text = "name", variable = CheckVar1,onvalue = 1, offvalue = 0)

check1.grid(row=1, column=1,sticky='e')

check1.select()

top.mainloop()

#获取勾选框的值

v1=CheckVar1.get()

print v1

### 有关button

**# 按钮Button** command 指定按钮调用的函数 调用btn\_click函数 不能使用btn\_click() 函数设置时没有入参为def btn\_click()

command后面不需要加括号，因为是储存该函数，点击按钮后才调用，而不是刚创建按钮，系统就调用该函数了

b = Button(root, text='clickme', command=btn\_click)

b['width'] = 10

b['height'] = 2

b.pack()

### 滚动条

1. Scrollbar 滚动条 pack布局

from Tkinter import \*

root = Tk()

scrollbar = Scrollbar(root)

scrollbar.pack( side = RIGHT, fill=Y )

mylist = Listbox(root, yscrollcommand = scrollbar.set )

for line in range(100):

mylist.insert(END, "This is line number " + str(line))

mylist.pack( side = LEFT, fill = BOTH )

scrollbar.config( command = mylist.yview )

mainloop()

1. Scrollbar 滚动条 grid布局

#用例浏览frame

frame3=Frame(root)

frame3.grid(row=5, column=0,columnspan=2,rowspan=10,sticky='e')

caseLabel=Label(frame3,text='用例预览')

caseLabel.grid(row=0, column=0)

caselist = **Listbox**(**frame3**,width=80,height=4)

caselist.grid(row=1, column=0, sticky=N+E+S+W)

caselist.columnconfigure(3, weight=1)

yroll=**Scrollbar**(**frame3**, orient=VERTICAL) #文本框-竖向滚动条 滚动条放在frame内，而不是listbox内

caselist.config(**yscrollcommand**=yroll.set) #listbox移动触发 滚动条移动

yroll.config(**command**=caselist.yview) #滚动条移动触发listbox移动

yroll.grid(row=1,column=1, sticky=N+S) #上下拉长

### 打开与保存

defaultextension = '.xls' 默认文件的扩展名 记得加这个，如果用户忘记输入后缀名，系统自动添加上

filetypes = [("Excel Files", '\*.xls', 'TEXT')), ("Python Files", '\*.py', 'TEXT'), ...] 设置文件类型下拉菜单里的的选项 选择某个选择，扩展名就会变成所选择的。

initialdir = 'D:\\' 对话框中默认的路径

initialfile = 'myfile.txt' 对话框中初始化显示的文件名

parent = root 父对话框(由哪个窗口弹出就在哪个上端)

title = 'hello' 弹出对话框的标题

打开一文件 askopenfile 获取一文件名用以打开 askopenfilename

保存一文件 asksaveasfile 获取一文件名保存 asksaveasfilename

打开一文件夹askdirectory

import tkFileDialog

#保存

def save\_as(self):

filetypes = [

("All Files", '\*'),

("Excel Files", '\*.xls', 'TEXT') ]

text\_value =self.caselist.get('1.0', END).strip()

if text\_value:

fobj = tkFileDialog.asksaveasfile(filetypes=filetypes ,defaultextension='.xls',initialdir ='D:\\')

print fobj

if fobj:

fobj.write(text\_value + '\n')

#打开函数

def open(self):

filetypes = [

("All Files", '\*'),

("Python Files", '\*.py', 'TEXT'),

("Text Files", '\*.txt', 'TEXT'),

("Excel Files", '\*.xls', 'TEXT'),

("Config Files", '\*.conf', 'TEXT')]

fobj = tkFileDialog.askopenfile(filetypes=filetypes)

if fobj:

self.caselist.delete('1.0', END)

self.caselist.insert('1.0', fobj.read())

self.namevar.set(fobj.name)

重命名

#重命名，在同一路径下保存新文件，删除旧文件

def rename(self):

filetypes = [ ("All Files", '\*'),

("Text Files", '\*.txt'),

("Excel 2004 Files", '\*.xls'),

("Excel 2007+ Files", '\*.xlsx')

]

dirname=os.path.dirname(self.original\_name)

basename=os.path.basename(self.original\_name)

extension=os.path.splitext(basename)[1]

self.anewName = tkFileDialog.asksaveasfilename(filetypes=filetypes,initialdir =dirname,

initialfile =basename,defaultextension =extension)

os.rename(self.original\_name, self.anewName)

self.get\_name(self.anewName)

### 有关窗口显示

1. 如何使窗口总在最前？

root.wm\_attributes('-topmost', 1)

#取消置顶

root.attributes('-topmost', 0)

1. 使窗口打开时所处的位置？

+%d+%d 控制位置

class App(object):

def \_\_init\_\_(self, root):

self.center\_window()

def center\_window(self,width=700, height=820,window=root):

screen\_width = root.winfo\_screenwidth()

screen\_height = root.winfo\_screenheight()

x = (screen\_width/2) - (width/2)

y = (screen\_height/2) - (height/2)

window.geometry('%dx%d+%d+%d' % (width, height, x, y))

#增加一个入参变量 window=root 如果不输入入参window,则默认root，我们创建子窗口也可调用center\_window来定位

self.center\_window(width=560,height=160,window=self.top)

widthxheight±x±y +x 代表窗口左边框离左边屏幕像素 -x窗口右边框离右边屏幕像素 +y 离上边屏幕 -y离屏幕底边

1. 隐藏、最小化、恢复等操作窗口命令

root.withdraw() 隐藏

root.iconify() 最小化窗口

root.deiconify() 恢复窗口、取消隐藏

1. 使跳出的窗口处于active状态？

self.top.focus\_set()

1. 如何使只有一个Toplevel？

set=Button(frame2, text='设置', command=**self.call\_config\_function**, relief=GROOVE, padx=3)

self.top=None

def call\_config\_function(self):

if not self.top:

self.config()

else:

try:

#cmp(self.top.state(),'normal')

self.top.deiconify() #尝试恢复子窗口，如果成功，不会报错

except:

print Exception

self.config() #如果不存在，会报错，捕捉报错，调用打开新窗口

def config(self):

self.top = Toplevel()

1. 如何判断一个窗口仍存活着？

if 'normal' == root.state():

print 'running'

1. 如何使窗口为无边框？

root.overrideredirect(1)

### 其他

1. 如何使label与输入框并列，并控制两者大小？

使用grid格局

import tkFont

ft = tkFont.Font(family = 'Microsoft YaHei',size =12,weight = tkFont.BOLD)

nameLabel =Label(root,text='name',font=ft)

nameLabel.grid(row=0, column=1,sticky='e')

name = Text(root,font=ft,height=1,width=50)

name.grid(row=0, column=2,sticky='w')

1. 如何使输入框为多行模式？

不使用entry ，改为Text.height为高度，单位为行，width为宽度，width=50表示一行容纳50个字

summary = Text(root,font=ft,height=2,width=50)

summary.grid(row=1, column=2,sticky='w')

1. 如何区分按钮重要次要？

使用relief=GROOVE 可以使次要按钮不突出 padx=10代表字体与按钮边界间隔为10

lastcase=Button(root, text='上一条', command='',relief=GROOVE)

lastcase.grid(row=6, column=1)

submit=Button(root, text='提交', command='',font=ft,bg='#b61d1d',fg='#fff',padx=10)

submit.grid(row=6, column=2,sticky='e')

1. Tkinter有自己的变量类，通过使用类里的get()和set()实例方法来操作变量

修改属性

# coding=utf-8

import Tkinter as tk

root = tk.Tk()

tk.Label(root,text="Hello World").pack()

var = tk.StringVar(value="Hi, what's up")

text\_input = tk.Entry(root,textvariable=var)

text\_input.pack()

def print\_content():

text\_output['text'] = var.get() # 将var的值赋给text\_output的text属性

#text\_output.config(text=var.get()) # 通过config更新message里要显示的信息

var.set('')

tk.Button(root,text="print",command=print\_content).pack()

root.bind('<Return>',lambda event:print\_content())

text\_output = tk.Message(root,text='')

text\_output.pack()

root.mainloop()

1. 修改控件属性

text\_output['text'] = var.get() # 将var的值赋给text\_output的text属性

#text\_output.config(text=var.get()) # 通过config更新message里要显示的信息

1. 代码分两行写，怎么相连？

用括号。

def print\_content():

input =('主题:'+name.get("1.0",'end-1c')+'\n摘要：'+summary.get("1.0",'end-1c')

+'\n前提条件：'+preconditions.get("1.0",'end-1c')+'\n步骤号：'+step\_number.get("1.0",'end-1c')

+'\n操作：'+actions.get("1.0",'end-1c')+'\n预期结果：'+expectedresults.get("1.0",'end-1c'))

tkMessageBox.showinfo ("Say Hello",input )

1. 如何打开网页？

tkinter没有控件来展示网页。但是可以设置点击按钮后，在浏览器打开网页

import webbrowser

webbrowser.open("page.html")

1. 分割线

cutoff = Label(root, text = '-' \* 100)

cutoff.grid(row=7, column=1,columnspan=2)

1. 设置各控件的间隔

grid布局时控制

lastcase.grid(row=1, column=1,padx=8)

ipadx：设置控件里面x方向空白区域大小；

ipady：设置控件里面y方向空白区域大小；

padx：设置控件周围x方向空白区域保留大小；

pady：设置控件周围y方向空白区域保留大小；

1. 关闭窗口

top = Toplevel()

top.destroy()

1. 如何再打开一个主窗口？root=Tk()

设置成使用class类来组织

root = Tk()

root.title('小需求用例生成器')

root.geometry('700x850')

class App:

def \_\_init\_\_(self, root):

def web():

root=Toplevel()

root.geometry('700x850')

app2=App(root)

lastcase=Button(root, text='刷新 ', command=web)

app=App(root)

root.mainloop()

1. 某控件获得焦点

name.focus\_force()

1. 右键菜单 复制 粘贴 全选 并对全部输入框有效？

def right\_click(event):

rightMenu.tk\_popup(event.x\_root, event.y\_root)

self.focuswid=event.widget

globals()[columnlist[h]].bind('<Button-3>',**right\_click**)

#右键菜单

rightMenu = Menu(root,tearoff=0)

rightMenu.add\_command(label = '复制' ,accelerator='Ctrl+C', command=lambda:self.focuswid.event\_generate("<<Copy>>"))

rightMenu.add\_separator()

rightMenu.add\_command(label = '粘贴' ,accelerator='Ctrl+V', command=lambda:self.focuswid.event\_generate("<<Paste>>"))

rightMenu.add\_separator()

rightMenu.add\_command(label = '全选' ,accelerator='Ctrl+A',command=lambda:self.focuswid.tag\_add('sel', '1.0', 'end'))

1. 报错TclError: wrong # args: should be

是因为insert的内容为None

比如 self.caselist2.insert(INSERT,self.case\_cashe[i][h]) 中有None情况发生

1. 复制文件名到剪切板上？

#复制文件名

def copy\_name(self,event):

r = Tk()

r.withdraw()#使该窗口隐藏

r.clipboard\_clear()

r.clipboard\_append(self.namevar.get())

r.destroy()

1. 如何修改窗口图标logo?

from Tkinter import \*

root = Tk()

root.iconbitmap(r'c:\Python32\DLLs\favicon.ico')

#如果图标在同一目录下，就不用绝对路径

root.iconbitmap('favicon.ico')  
使用该网页创造图标,其他图片改后缀为.ico是不行的。会报错\_tkinter.TclError: bitmap "favicon.ico" not defined

更改桌面上的应用图标就必须替换python.ico，但是这样所有python所编写软件都会更改图标

1. 如何禁用控件？

属性中state=DISABLED 是控件无法操作

B = Tkinter.Button(top, text ="Hello", command = helloCallBack,state=DISABLED)

1. 如何使某个选项无效？

self.xmenu = Menu(self.root)

self.filemenu = Menu(self.xmenu, tearoff = 0)

self.filemenu.add\_command(label = '用例预览文本保存为txt', command =self.save\_as\_txt)

self.xmenu.add\_cascade(label = '文件', menu = self.filemenu)

#使某个二级菜单无效

self.filemenu.entryconfig('用例预览文本保存为txt', state="disabled")

#使某个二级菜单有效

self. filemenu.entryconfig('用例预览文本保存为txt', state="normal")

#使某个一级菜单无效

self.xmenu.entryconfig('文件', state="disabled")

1. 读取配置txt文件不成功,备用措施有什么？

#读取conig配置文件

def read\_config(self):

try:

configtxt=open('config.txt', "r")

self.prec\_default=configtxt.readlines()

configtxt.close()

self.prec\_default[5] #作用就是检测是否有6位元素，如果没有报错，走except语句

except:

self.prec\_default=[1,1,1,1,1,1]

1. 判断当前光标在哪个控件中？

self.root.focus\_get()

使某个控件获得焦点 name.focus\_set()

1. 使在self.caseText控件中无法进行剪切操作

def editmenu\_cut(self):

editMenuFocusWidget=self.root.focus\_get()

print editMenuFocusWidget,'----',self.caseText

if editMenuFocusWidget != self.caseText:

editMenuFocusWidget.event\_generate('<<Cut>>')

## 使用class类来组织

#-\*- encoding=UTF-8 -\*-

from Tkinter import \*

class App:

def \_\_init\_\_(self, master):

#构造函数里传入一个父组件(master),创建一个Frame组件并显示

frame = Frame(master)

frame.pack()

#创建两个button，并作为frame的一部分

self.button = Button(frame, text="QUIT", fg="red", command=frame.quit)

self.button.pack(side=LEFT) #此处side为LEFT表示将其放置 到frame剩余空间的最左方

self.hi\_there = Button(frame, text="Hello", command=self.say\_hi)

self.hi\_there.pack(side=LEFT)

def say\_hi(self):

print "hi there, this is a class example!"

root = Tk()

app = App(root)

root.mainloop()

------------------------------------------

## grid布局

### 基础

self.cutoff.grid(row=4, column=0,columnspan=2,pady=2)

Grid(网格)布局管理器会将控件放置到一个二维的表格里。主控件被分割成一系列的行和列，表格中的每个单元(cell)都可以放置一个控件。

可以指定控件跨越一个或者多个网格。columnspan选项可以指定控件跨越多列显示，而rowspan选项同样可以指定控件跨越多行显示。

默认的空间会在网格中居中显示。你可以使用**sticky选项**去指定对齐方式，可以选择的值有：N/S/E/W，分别代表上/下/左/右。如果你想让label靠左显示，你可以设置stricky的值为W。

注意：不要试图在一个主窗口中混合使用pack和grid。比如以下就不能用.pack()

column： 控件放置位置的列数，从0开始算起，默认为0；

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

from Tkinter import \*

master = Tk()

var = IntVar()

Label(master, text="First").grid(sticky=E)

Label(master, text="Second").grid(sticky=E)

e1 = Entry(master)

e2 = Entry(master)

e1.grid(row=0, column=1)

e2.grid(row=1, column=1)

checkbutton = Checkbutton(master, text='Preserve aspect', variable=var)

checkbutton.grid(**columnspan=2**, sticky=W)

photo = PhotoImage(file='jd.gif')

label = Label(image=photo)

label.image = photo

label.grid(row=0, column=2, columnspan=2, rowspan=2, sticky=W+E+N+S, padx=20, pady=5)

# padx=20, pady=5 padx代表水平方向的padding pady代表垂直方向的padding

button1 = Button(master, text='Zoom in')

button1.grid(row=2, column=2)

button2 = Button(master, text='Zoom out')

button2.grid(row=2, column=3)

mainloop()

### 细节

1. 如何只增加左侧的间隔，而不增加右侧的间隔？

self.submit\_frame.grid(row=0, column=5,padx=(180,0))

1. 如何使控件隐藏？

如果该控件使用.pack()布局，则使用forget() 或者pack\_forget()。

如果grid布局，则.grid\_forget()

place布局则.place\_forget()

if prec\_default[h]=='0\n':

globals()[columnlist[h]].grid\_forget()

globals()[columnlist[h]+'Label'].grid\_forget()

globals()[columnlist[h]+'Label'].grid\_remove()

"""Unmap this widget but remember the grid options."""

1. 如何中途修改grid的属性？

self.nameframe.grid\_configure(pady=0)

1. 控件隐藏后，如何在出现？

self.nameframe.grid\_forget()

self.nameframe.grid\_configure(pady=0)

再使用一次.grid\_configure语句

### 有关迷你模式

1. 如何使窗口改为迷你模式miniMode？

最主要设置一个标记，迷你模式为1，非迷你为0

self.isMiniMode=0

def switch\_mini\_model(self):

self.isMiniMode=1

然后很多函数就可以根据这个标记，来判断是否调用一些逻辑. 迷你和普通模式下控件显示不同。

if self.isMiniMode==1:

self.ordinal\_numeral\_mini\_display()

else:

self.ordinal\_numeral\_display()

1. 如何取消迷你模式minimode?

用下面语句将之前隐藏的控件重新显示，不过要记得加row=4, column=0,否则，系统默认将此控件放在最后。

self.cutoff.grid\_configure(row=4, column=0)

-------

if self.isMiniMode==1:

...

else:

self.center\_window(self.root,650,800)

self.nameframe.grid\_configure(row=1, column=0,pady=(10,0))

self.mainFont = tkFont.Font(family = 'Microsoft YaHei',size =12,weight = tkFont.NORMAL)

self.ordinal\_numeral\_display()

self.mainframe.destroy()

self.main\_input\_display()

self.controlFrame.destroy()

self.control\_frame\_display()

self.filemenu.entryconfig('用例预览文本保存为txt', state="normal")

self.cutoff.grid\_configure(row=4, column=0)

self.selectRadioFrame.grid\_configure(row=5, column=0)

self.cacheTextboxFrame.grid\_configure(row=6, column=0)

## 例子

### 显示web上的图片

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

import io

# allows for image formats other than gif

from PIL import Image, ImageTk

try:

# Python2

import Tkinter as tk

from urllib2 import urlopen

except ImportError:

# Python3

import tkinter as tk

from urllib.request import urlopen

root = tk.Tk()

# find yourself a picture on an internet web page you like

# (右键--复制图片地址)

#url = "http://www.google.com/intl/en/images/logo.gif"

url = "http://i50.tinypic.com/34g8vo5.jpg"

image\_bytes = urlopen(url).read()

# internal data file 将二进制数据写入内存

data\_stream = io.BytesIO(image\_bytes)

# open as a PIL image object 读取内存为一图片

pil\_image = Image.open(data\_stream)

# optionally show image info

# get the size of the image

w, h = pil\_image.size

# split off image file name 格式化窗口标题

fname = url.split('/')[-1]

sf = "{} ({}x{})".format(fname, w, h)

root.title(sf)

# convert PIL image object to Tkinter PhotoImage object

tk\_image = ImageTk.PhotoImage(pil\_image)

# put the image on a typical widget

label = tk.Label(root, image=tk\_image, bg='brown')

label.pack(padx=5, pady=5)

root.mainloop()

BytesIO 我们在读写文件时也有时利用二进制数据进行读取。

### 图片与文字交互

前景色和背景色 fg bg

有六种颜色可以使用 Red Green Blue Yellow LightBlue

置宽度与高度 width: 宽度 height: 高度 left： 图像居左 right: 图像居右

top： 图像居上 bottom：图像居下 center：文字覆盖在图像上

按钮点击jpg图片： 显示图片

因为create\_image\_label()结束后，图像就自动销毁了，因为其内部的image\_file/im/image\_label都是局部变量，仅在函数内存活。所以要使用全局变量。

from Tkinter import \*

from PIL import Image, ImageTk

root = Tk()

image\_frame = Frame(root)

image\_file = im = image\_label = None

def create\_image\_label():

global image\_file, im, image\_label

image\_file = Image.open("hello.jpg")

im = ImageTk.PhotoImage(image\_file)

image\_label = Label(image\_frame,image = im)

image\_label.grid(row = 3, column = 0, sticky = NW, pady = 8, padx = 20)

button = Button(image\_frame,text='猛击这里',anchor = 'center',command = create\_image\_label)

button.grid(row = 2, column = 0, sticky = NW, pady = 8, padx = 20)

image\_frame.pack()

root.mainloop()

点击按钮更换图片

#coding=utf-8

from Tkinter import \*

def change():

label.configure(image = bm2)

top = Tk()

bm = PhotoImage(file = "/home/fangxu/图片/4.png")

bm2 = PhotoImage(file = "/home/fangxu/图片/5.png")

label = Label(top, image = bm)

label.pack()

button = Button(top, text = "changepicture", command = change)

button.pack()

top.mainloop()

### 将英尺转换为mi

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

from Tkinter import \*

def calculate(\*args):

try:

value = float(feet.get())

meters.set((0.3048 \* value \* 10000.0 + 0.5)/10000.0)

except ValueError:

pass

root =Tk()

root.title("Feet to Meters")

mainframe = Frame(root)

mainframe.grid(column=0, row=0, sticky=(N, W, E, S))

mainframe.columnconfigure(0, weight=1)

mainframe.rowconfigure(0, weight=1)

feet = StringVar()

meters = StringVar()

feet\_entry = Entry(mainframe, width=7, textvariable=feet)

feet\_entry.grid(column=2, row=1, sticky=(W, E))

Label(mainframe, textvariable=meters).grid(column=2, row=2, sticky=(W, E))

Button(mainframe, text="Calculate", command=calculate).grid(column=3, row=3, sticky=W)

Label(mainframe, text="feet").grid(column=3, row=1, sticky=W)

Label(mainframe, text="is equivalent to").grid(column=1, row=2, sticky=E)

Label(mainframe, text="meters").grid(column=3, row=2, sticky=W)

for child in mainframe.winfo\_children(): child.grid\_configure(padx=5, pady=5)

feet\_entry.focus()

root.bind('<Return>', calculate)

root.mainloop()

# for child…第一行，检查框架内的所有子控件，并且在每个控件周围添加一小点距离，所以它们看起来没那么拥挤。当我们第一次放置控件到界面上时，我们可以添加这些选项到每一个"grid"调用，但这仅仅是一种漂亮的快捷方式。

第二行，告诉Tk把注意力转移到我们的输入框上，也就是说，在开始的时候，光标会默认在输入框区域内，所以用户在输入的时候，不需要先单击它。

第三行，告诉Tk如果用户按下了回车键，就会调用计算程序段，作用和点击计算按钮时一样

### 简易计算器

主要涉及了tkinter的使用，函数定义和调用，匿名函数的使用，类成员函数定义等python基础知识

#-\*- encoding=UTF-8 -\*-

from Tkinter import \*

#创建横条型框架

def frame(root,side):

w=Frame(root)

w.pack(side=side,expand=YES,fill=BOTH)

return w

#创建按钮

def button(root,side,text,command=None):

w=Button(root,text=text,command=command)

w.pack(side=side,expand=YES,fill=BOTH)

return w

#继承了Frame类，初始化程序页面布局

class Calculator(Frame):

def \_\_init\_\_(self):

Frame.\_\_init\_\_(self)

self.pack(expand=YES,fill=BOTH)

self.master.title('Simple Calculater')

display=StringVar()

#添加输入框

Entry(self,relief=SUNKEN,

textvariable=display).pack(side=TOP,expand=YES,fill=BOTH)

#添加横条型框架以及里面按钮

for key in ('123','456','789','-0.'):

keyF=frame(self,TOP)

for char in key:

button(keyF,LEFT,char,lambda w=display,c=char:w.set(w.get()+c))

#添加操作符按钮

opsF=frame(self,TOP)

for char in '+-\*/=':

if char == '=':

btn=button(opsF,LEFT,char)

btn.bind('<ButtonRelease -1>',lambda e,s=self,w=display:s.calc(w),'+')

else:

btn=button(opsF,LEFT,char,lambda w=display,s='%s' %char:w.set(w.get()+s))

#添加清除按钮

clearF=frame(self,BOTTOM)

button(clearF,LEFT,'clear',lambda w =display:w.set(''))

#调用eval函数计算表达式的值

def calc(self,display):

try:

display.set(eval(display.get()))

except:

display.set('ERROR')

#程序入口

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

print('ok')

Calculator().mainloop()

## 代码模板

import os.path

from Tkinter import \*

from tkFileDialog import \*

from tkFont import Font

try:

from ttk import Button, Scrollbar

except ImportError:

pass

class **Edit\_save**(object):

def **\_\_init\_\_**(*self*):

*self*.root = Tk()

*self*.root.title(*'EditSave'*)

*self*.root.geometry(*'+120+50'*)

*self*.font\_en = Font(*self*.root, size=12)

*self*.font\_text = Font(*self*.root,family=*"Helvetica"*,

size=12, weight=*'normal'*)

*self*.menubar = Menu(*self*.root, bg=*'purple'*)

*self*.filemenu = Menu(*self*.menubar)

*self*.filemenu.add\_command(label=*'Open'*, accelerator=*'Ctrl+o'*,

command=*self*.\_\_open, underline=0)

*self*.filemenu.add\_command(label=*'Save'*, accelerator=*'Ctrl+s'*,

command=*self*.\_\_save, underline=0)

*self*.filemenu.add\_command(label=*'Save As'*, accelerator=*'Ctrl+Shift+s'*,

command=*self*.\_\_save\_as, underline=5)

*self*.filemenu.add\_separator()

*self*.filemenu.add\_command(label=*'Quit'*, accelerator=*'Alt+F4'*,

command=*self*.root.destroy, underline=0)

*self*.menubar.add\_cascade(label=*'File'*, underline=0, menu=*self*.filemenu)

*self*.helpmenu = Menu(*self*.menubar)

*self*.menubar.add\_cascade(label=*'Help'*, underline=0, menu=*self*.helpmenu)

*self*.editmenu = Menu(*self*.menubar, tearoff=False)

*self*.editmenu.add\_command(label=*'Refresh'*, accelerator=*'F5'*,

command=*self*.root.update)

*self*.editmenu.add\_command(label=*'Copy'*, accelerator=*'Ctrl+C'*,

command=lambda : *self*.text.event\_generate(*'<<Copy>>'*))

*self*.editmenu.add\_command(label=*'Paste'*, accelerator=*'Ctrl+V'*,

command=lambda : *self*.text.event\_generate(*'<<Paste>>'*))

*self*.editmenu.add\_command(label=*'Cut'*, accelerator=*'Ctrl+X'*,

command=lambda : *self*.text.event\_generate(*'<<Cut>>'*))

# self.editmenu.add\_command(label='Undo', accelerator='Ctrl+Z',

# command=lambda : self.text.event\_generate('<<Undo>>'))

# self.editmenu.add\_command(label='Redo', accelerator='Ctrl+Y',

# command=lambda : self.text.event\_generate('<<Redo>>'))

*self*.editmenu.add\_command(label=*'select\_all'*, accelerator=*'Ctrl+A'*,

command=lambda : *self*.text.tag\_add(*'sel'*, *'1.0'*, *'end'*))

*self*.fm\_base = Frame(*self*.root)

*self*.fm\_up = Frame(*self*.fm\_base)

*self*.var = StringVar()

*self*.en = Entry(*self*.fm\_up, font=*self*.font\_en,

textvariable=*self*.var, width=20)

*self*.bt\_open = Button(*self*.fm\_up, text=*'Open'*,)

*self*.bt\_quit = Button(*self*.fm\_up, text=*'Quit'*, underline=0)

*self*.bt\_quit.pack(side=RIGHT, padx=10)

*self*.bt\_open.pack(side=RIGHT, padx=5)

*self*.en.pack(side=RIGHT, pady=5)

*self*.bt\_open.config(command=*self*.\_\_open)

*self*.bt\_quit.config(command=*self*.root.destroy)

*self*.fm\_up.pack(fill=X)

*self*.fm\_base.pack(fill=X)

*self*.fm\_down = Frame(*self*.root)

*self*.text = Text(*self*.fm\_down, font=*self*.font\_text,

width=50, height=20)

*self*.text.pack(side=LEFT, fill=BOTH, expand=True)

*self*.scb = Scrollbar(*self*.fm\_down)

*self*.scb.pack(side=LEFT, fill=Y)

*self*.text.config(yscrollcommand=*self*.scb.set)

*self*.scb.config(command=*self*.text.yview)

*self*.fm\_down.pack(fill=BOTH, expand=True)

*self*.root.bind(*'<Control-o>'*, *self*.\_\_invoke\_open)

*self*.root.bind(*'<Control-O>'*, *self*.\_\_invoke\_open)

*self*.root.bind(*'<Control-s>'*, *self*.\_\_invoke\_save)

*self*.root.bind(*'<Control-S>'*, *self*.\_\_invoke\_save\_as)

*self*.root.bind(*'<Button-3>'*, *self*.\_\_popupmenu)

*self*.root.bind(*'Alt-F4'*, lambda event: *self*.root.destroy)

*self*.root.bind(*'<F5>'*, lambda event: *self*.root.update)

*self*.root.bind(*'<Control-C>'*,

lambda event: *self*.text.event\_generate(*'<<Copy>>'*))

*self*.root.bind(*'<Control-V>'*,

lambda event: *self*.text.event\_generate(*'<<Paste>>'*))

*self*.root.bind(*'<Control-X>'*,

lambda event: *self*.text.event\_generate(*'<<Cut>>'*))

for i in *'a'*, *'A'*:

*self*.root.bind(*'<Control-%s>'* % i,

lambda event: *self*.text.tag\_add(*'sel'*, *'1.0'*, *'end'*))

# for i in 'z', 'Z':

# self.root.bind('<Control-%s>' % i,

# lambda event: self.text.event\_generate('<<Undo>>'))

# for i in 'y', 'Y':

# self.root.bind('<Control-%s>' % i,

# lambda event: self.text.event\_generate('<<Redo>>'))

*self*.root.config(menu=*self*.menubar)

*self*.root.mainloop()

def **\_\_open**(*self*):

filetypes = [

(*"All Files"*, *'\*'*),

("Python Files", '\*.py', 'TEXT'),

(*"Text Files"*, *'\*.txt'*, *'TEXT'*),

("Config Files", '\*.conf', 'TEXT')]

fobj = askopenfile(filetypes=filetypes)

if fobj:

*self*.text.delete(*'1.0'*, END)

*self*.text.insert(*'1.0'*, fobj.read())

*self*.en.delete(0, END)

*self*.en.insert(0, fobj.name)

def **\_\_save**(*self*):

value = *self*.var.get().strip()

if value:

f = open(value, *'w'*)

f.write(*self*.text.get(*'1.0'*, END).strip() + *'\n'*)

f.close()

else:

*self*.\_\_save\_as()

def **\_\_save\_as**(*self*):

text\_value = *self*.text.get(*'1.0'*, END).strip()

if text\_value:

fobj = asksaveasfile()

if fobj:

fobj.write(text\_value + *'\n'*)

def **\_\_invoke\_open**(*self*, event):

*self*.\_\_open()

def **\_\_invoke\_save**(*self*, event):

*self*.\_\_save()

def **\_\_invoke\_save\_as**(*self*, event):

*self*.\_\_save\_as()

def **\_\_popupmenu**(*self*, event):

*self*.editmenu.post(event.x\_root, event.y\_root)

if \_\_name\_\_ == *"\_\_main\_\_"*:

Edit\_save()

## 遇到的难题与问题

1. 如何导入特定规则的.txt文本，并且比如name字段如果多行文本，也可导入多行？

self.case\_cashe=[] #清除缓存

with open(txtname, "a+") as casedoc: #读取文档 如果txtname为None,这里报错，所以就不会更新文件名为空。

txtReadlines=casedoc.readlines()

#print txt\_readlines

eachCase=['']\*6

fieldContentList=[]

**isBegin=0**  #判断是否开始匹配到相关字段

for txtRow in txtReadlines:

for field in self.columnlist[1:]:

behindFieldMatch = re.match(field,txtRow)

if behindFieldMatch:

break

firstFieldMatch = re.match('name',txtRow)

ordinalNumMatch = re.match('CASE|case',txtRow)

num=self.columnlist.index(field) #匹配到的字段的索引

if behindFieldMatch:

fieldContentStr=''.join(fieldContentList)

eachCase[num-1]=fieldContentStr #上一个字段的多个行 加到每条用例list

fieldContentList=[]

txtRow=txtRow.strip(field+':').lstrip()**.strip('\n') #每个字段之后都要去掉换行符**

fieldContentList.append(txtRow)

isBegin=1

elif firstFieldMatch:

fieldContentStr=''.join(fieldContentList)

eachCase[5]=fieldContentStr #将第6个字段的值加到每条用例list中

if isBegin:

self.case\_cashe.append(eachCase) #碰到该用例的开头，就把上一用例的内容给增加到缓存中

fieldContentList=[]

txtRow=txtRow.strip('name:').lstrip().strip('\n')

fieldContentList.append(txtRow)

eachCase=['']\*6

isBegin=1

else:

if txtRow!='\n' and txtRow and isBegin and not ordinalNumMatch:

txtRow=txtRow.lstrip().strip('\n')

fieldContentList.append(**'\n'**+txtRow) #到此步，代表该字段多行，就加上换行符

1. self.caseText.insert(INSERT,self.case\_cashe[i][h]+'\n') 报错

TypeError: unsupported operand type(s) for +: 'float' and 'str' 因为self.case\_cashe[i][h]有可能不是str,而是float

1. 取消打开文件，会返回none,如何使显示的文件名不受影响？

filetypes = [ ("Excel 2004 Files", '\*.xls', 'TEXT')]

self.xlsname = tkFileDialog.askopenfilename(filetypes=filetypes,title ='导入.xls格式的excel')

except:

self.xlsname=self.FilePathName

self.get\_opened\_filename(self.xlsname)

1. 打开保存窗口，默认的文件名和路径怎么设置？

dirname=os.path.dirname(self.FilePathName)

basename=os.path.basename(self.FilePathName)

extension=os.path.splitext(basename)[1]

fileTypeVariable='\*'+extension

filetypes = [(extension+'文件',**fileTypeVariable**)]

self.anewName = tkFileDialog.asksaveasfilename(filetypes=filetypes,initialdir =dirname,

initialfile =basename,defaultextension =extension)

1. 如何将list转为xml格式？

#导出xml格式

def export\_xml(self):

filetypes = [ ("xml Files", '\*.xml')]

if self.FilePathName=='N/A' or not self.FilePathName:

dirname=os.path.dirname(os.getcwd()) #os.getcwd()获得当前工作目录

default\_jsonname='新建xml格式用例.xml'

else:

dirname=os.path.dirname(self.FilePathName)

basename=os.path.basename(self.FilePathName)

default\_jsonname=os.path.splitext(basename)[0]+'.xml'

extension=os.path.splitext(basename)[1]

print extension

xmlname = tkFileDialog.asksaveasfilename(filetypes=filetypes,initialdir =dirname,initialfile =default\_jsonname,title ='导出为xml格式',defaultextension ='.xml')

doc = minidom.Document()

#generate the DTC

allCasexml = doc.createElement("Casexml xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"")

doc.appendChild(allCasexml)

for eachCaseIndex in range(len(self.caseCache)):

eachCasexml= doc.createElement('Case'+str(eachCaseIndex))

allCasexml.appendChild(eachCasexml)

for filedIndex in range(6):

eachNode = doc.createElement(self.columnlist[filedIndex])

eachCasexml.appendChild(eachNode)

eachNode.appendChild(doc.createTextNode(str(self.caseCache[eachCaseIndex][filedIndex])))

f=open(xmlname, 'w+')

doc.writexml(f, "", " ", "\n") #第2个控制整体缩进，第3个控制每个子节点缩进，第4个控制每个节点后换行

f.close()

# PyQt4

## 基础

1. 面向过程

import sys

from PyQt4 import QtGui

app = QtGui.QApplication(sys.argv)

widget = QtGui.QWidget()

widget.resize(250, 150)

widget.setWindowTitle('PyQt')

widget.show()

sys.exit(app.exec\_())

下面，简单说一下上面这段代码的含义。

第2行引入的QtGui是PyQt中最基本的模块，包含了PyQt的绘图组件及其相关类。

第4行使用QApplication创建了一个application。每个PyQt程序中均要包含一个application对象。

第5行使用QWidget创建了一个widget，并在第6行中使用resize设置了它的大小。第7行的setWindowsTitle设置了窗口标题的文字。第8行的show使这个widget显示出来。

最后一行是这个程序的主循环，事件处理从本行语句开始。至于为什么是exec\_而不是exec呢？是因为exec是Python的关键字。

1. 面向对象

import sys

from PyQt4 import QtCore,QtGui

from PyQt4.QtCore import Qt

class MainWindow(QtGui.QWidget):

def \_\_init\_\_(self):

QtGui.QWidget.\_\_init\_\_(self)

#初始化position

self.m\_DragPosition=self.pos()

self.resize(300,400)

self.setWindowFlags(Qt.FramelessWindowHint | Qt.WindowStaysOnTopHint)

self.setMouseTracking(True)

if \_\_name\_\_=="\_\_main\_\_":

mapp=QtGui.QApplication(sys.argv)

mw=MainWindow()

mw.show()

sys.exit(mapp.exec\_())

## 组件

### 其他

review = QtGui.QLabel('Review')

authorEdit = QtGui.QLineEdit()

reviewEdit = QtGui.QTextEdit()

cb = QtGui.QCheckBox('Show title')

lcd = QtGui.QLCDNumber() #lcd数字

frame设置

bottomframe=QtGui.QFrame()

frame.setFrameShape(QtGui.QFrame.Box) #frame形状

frame.setFrameShadow(QtGui.QFrame.Sunken) #frame边框

edit = QtGui.QLineEdit(self) # 创建 QLineEdit 。

如果单行编辑器中的文本发生变化，调用 onChanged() 方法。

self.connect(edit, QtCore.SIGNAL('textChanged(QString)'),self.onChanged)

在 onChanged() 方法中，我们把输入的文字设置到标签中。并调用 adjustSize() 方法调整标签的尺寸为文本的长度。

def onChanged(self, text):

self.label.setText(text)

self.label.adjustSize()

### 有关Label

l1 = QLabel()

l1.setText("Hello World")

l1.setAlignment(Qt.AlignCenter)

### 有关button

1. pushbutton设置

grid2 = QtGui.QGridLayout(bottomframe) #创建新的grid，控制bottomframe内的格局

aboutButton = QtGui.QPushButton(parent=bottomframe,text=u'关于') #按钮放置在bottomframe内

aboutButton.setFixedSize(25,25) #使按钮的大小固定不能变

aboutButton.setStyleSheet("QPushButton:hover{color:#b61d1d}"+

"QPushButton{font-family:'Microsoft YaHei';font-size:12px;background-color:#fff;color:#666;border:0px }")

grid2.addWidget(aboutButton,0,0) #设置aboutButton控件的grid的位置

self.connect(aboutButton, QtCore.SIGNAL('clicked()'),self.aboutMessageBox) #设置按下按钮调用函数

bottomframe.setLayout(grid2) #使新的grid起作用在bottomframe起作用

1. 修改按钮的文本？

aboutButton.setText("B") 字符串将会覆盖之前的文本'关于'

1. 如何点击pushbutton关联closeEvent函数？

self.closeButton.clicked.connect(self.close)或者

self.connect(closeButton, QtCore.SIGNAL('clicked()'),self.close)

## 事件与信号

1. 信号和槽

self.connect(slider, QtCore.SIGNAL(‘valueChanged(int)’), lcd, QtCore.SLOT(‘display(int)’))

这里我们连接滑块的 valueChanged() 信号到LCD数字的 display() 槽。

connect 方法有4个参数， sender 是发送信号的对象， signal 是发射的信号， receiver 是接收信号的对象， 最后， slog 是对信号反应的方法。

#注意：如果调用的是函数，就不需要接收信号的对象，第3个参数就填 函数

self.connect(button1, QtCore.SIGNAL('clicked()'),self.buttonClicked)

1. 按下按钮调用函数

button = QtGui.QPushButton('Say hello!')

button.clicked.connect(self.sayHello)

或者

self.connect(button, QtCore.SIGNAL('clicked()'),self.sayHello)

1. 键盘操作

在我们的例子中，我们重新实现了 keyPressEvent() 处理。

def **keyPressEvent**(self, event): #当按下键盘时，程序自动调用该函数

if event.key() == QtCore.Qt.Key\_Escape:

self.close()

如果我们按下了escape按钮，程序将退出。

1. 知道哪个组件发出的信号，PyQt4有 sender() 方法。

由两个按钮，在 buttonClicked() 方法中我们通过调用 sender() 方法确定点击了哪个按钮。

#注意：如果调用的是函数，就不需要接收信号的对象，第3个参数就填 函数

self.connect(button1, QtCore.SIGNAL('clicked()'),self.buttonClicked)

self.connect(button2, QtCore.SIGNAL('clicked()'),self.buttonClicked)

两个按钮都连接了同一个信号。

def buttonClicked(self):

sender = self.sender()

self.statusBar().showMessage(sender.text() + ' was pressed')

通过调用 sender() 方法我们确定信号来源。在程序的状态栏，我们显示按下的按钮的标签。

1. 自建信号，并连接槽

我们创建一个名为 closeEmitApp() 的新信号，在鼠标的按下中发射该信号。

def mousePressEvent(self, event): #鼠标按下时，程序自动调用该函数

self.emit(QtCore.SIGNAL('closeEmitApp()'))

通过 emit() 方法发射信号。

self.connect(self, QtCore.SIGNAL('closeEmitApp()'),QtCore.SLOT('close()'))

这里我们把手工创建的 closeEmitApp() 信号和 close() 槽连接。

1. 自建调用的函数

def setLabelText(self):

self.label.setText('ABC')

self.connect(self.btnClearText, QtCore.SIGNAL('clicked()'), self.setLabelText)

1. 自建的调用函数含有参数

from PyQt4.QtCore import QObject, pyqtSlot

class Foo(QObject):

@pyqtSlot(int, str)

def foo(self, arg1, arg2):

""" C++: void foo(int, QString) """

@pyqtSlot(int, name='bar')

def foo(self, arg1):

""" C++: void bar(int) """

@pyqtSlot(int, QObject)

def foo(self, arg1):

""" C++: int foo(int, QObject \*) """

## 布局管理

### grid布局

最常用的布局类是网格布局，网格布局把空间划分成行和列。我们使用 QGridLayout 类来创建网格布局。

grid = QtGui.QGridLayout()

grid.addWidget(QtGui.QLabel(''), 0, 2)

self.setLayout(grid)

调用 addWidget() 方法来把窗口组件加到网格中，参数是部件（ widget ），行（ row ）和列（ column ）数字。

部件可以在表格中跨越多列或多行

grid = QtGui.QGridLayout()

grid.setSpacing(10)

#我们创建了一个表格布局，并且设置组件间的间隔为10。

review = QtGui.QLabel('Review')

grid.addWidget(review, 3, 0)

reviewEdit = QtGui.QTextEdit()

grid.addWidget(reviewEdit, 3, 1, 5, 2) #5代表跨5行 2代表跨2列 如果不设置默认是1，代表占1行/列

self.setLayout(grid)

如何设置frame并在frame中grid？

bottomframe=QtGui.QFrame()

frame.setFrameShape(QtGui.QFrame.Box)

frame.setFrameShadow(QtGui.QFrame.Sunken)

grid.addWidget(bottomframe, 4,0,1,3)

self.setLayout(grid)

grid2 = QtGui.QGridLayout(bottomframe) #设置在frame中

aboutButton = QtGui.QPushButton(parent=bottomframe,text=u'关于')

grid2.addWidget(aboutButton,0,0)

bottomframe.setLayout(grid2)

如何删除控件？

如下才能完全删除控件，使控件不存在

self.gridControl.removeWidget(self.openFileButton)

self.openFileButton.deleteLater()

del self.openFileButton

就比如删除2维码，2维码是存放在label中，label即使隐藏，2维码也还是在显示，所以必须彻底删除。

self.gridMain.removeWidget(self.imageDisplay)

self.imageDisplay.deleteLater()

del self.imageDisplay

### 其他

1. 框布局

使用布局类管理布局更灵活、更实用。这是在窗体上摆放组件的首选方式。基本的布局类是 QHBoxLayout 和QVBoxLayout ，它们可以横向和纵向排列窗口组件。

假设我们想要摆放两个按钮到右下角。为了创建这样一个布局，我们需要一个水平框和一个垂直框。我们通过增加**延展因素** 来创建必要的间隔。

def initUI(self):

okButton = QtGui.QPushButton("OK")

cancelButton = QtGui.QPushButton("Cancel")

hbox = QtGui.QHBoxLayout()

hbox.addStretch(1)

hbox.addWidget(okButton)

hbox.addWidget(cancelButton)

vbox = QtGui.QVBoxLayout()

vbox.addStretch(1)

vbox.addLayout(hbox)

self.setLayout(vbox)

self.setWindowTitle('box layout')

self.resize(300, 150)

1. 绝对定位

label2 = QtGui.QLabel('tutorials for programmers', self)

label2.move(35, 40)

我们简单的调用 move() 方法来定位组件。在我们的 QLabel 例子中，我们用x和y坐标来定位。坐标系统从左上角开始，x值从左到右增长，y值从上到下增长。

## 菜单和工具栏

创建一个菜单栏、工具栏和状态栏。

状态栏是用来显示状态信息的窗口组件。

import sys

from PyQt4 import QtGui, QtCore

class MainWindow(QtGui.QMainWindow):

def \_\_init\_\_(self):

QtGui.QMainWindow.\_\_init\_\_(self)

self.resize(350, 250)

self.setWindowTitle('mainwindow')

#这里我们创建了一个文本编辑控件，把它设置成 QMainWinow 的中心组件。中心组件将会占据所有留下的空间。

textEdit = QtGui.QTextEdit()

self.setCentralWidget(textEdit)

exit = QtGui.QAction(QtGui.QIcon('icons/exit.png'), 'Exit', self)

exit.setShortcut('Ctrl+Q')

exit.setStatusTip('Exit application')

#这里我们连接行为的 triggered() 信号到预定义的 close() 槽。

self.connect(exit, QtCore.SIGNAL('triggered()'), QtCore.SLOT('close()'))

#调用 QMainWindow 的 statusBar() 方法来得到状态栏， showMessage() 将消息显示在状态栏上。

self.statusBar().showMessage('Ready')

#首先我们使用QMainWindow类的 menuBar() 方法创建一个菜单栏，然后使用 addMenu()方法增加一个菜单项，最后我们插入上面创建好的对象到文件菜单。

menubar = self.menuBar()

file = menubar.addMenu('&File')

file.addAction(exit)

toolbar = self.addToolBar('Exit')

toolbar.addAction(exit)

app = QtGui.QApplication(sys.argv)

main = MainWindow()

main.show()

sys.exit(app.exec\_())

## 输入框/对话框

### QInputDialog输入对话框

QInputDialog 提供一个简单的对话框，以便从用户获取单个值。输入值可以是一个字符串，一个数字或者列表的一项。

import sys

from PyQt4 import QtGui

from PyQt4 import QtCore

class Example(QtGui.QWidget):

def \_\_init\_\_(self):

super(Example, self).\_\_init\_\_()

self.initUI()

def initUI(self):

self.button = QtGui.QPushButton('Dialog', self)

self.button.setFocusPolicy(QtCore.Qt.NoFocus)

self.button.move(20, 20)

self.connect(self.button, QtCore.SIGNAL('clicked()'),

self.showDialog)

self.setFocus()

self.label = QtGui.QLineEdit(self)

self.label.move(130, 22)

self.setWindowTitle('InputDialog')

self.setGeometry(300, 300, 350, 80)

def showDialog(self):

text, ok = QtGui.QInputDialog.getText(self, 'Input Dialog',

'Enter your name:')

if ok:

self.label.setText(str(text))

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

app = QtGui.QApplication(sys.argv)

ex = Example()

ex.show()

app.exec\_()

这个例子有一个按钮和一个单行编辑器，按钮显示输入对话框来获取值，输入的值将会显示在单行编辑器中。

text, ok = QtGui.QInputDialog.getText(self, 'Input Dialog',

'Enter your name:')

这行代码显示输入对话框，第一个字符串是对话框标题，第二个是对话框里面的消息。对话框返回输入的文本那一个布尔值。如果我们点击 ok 按钮，布尔值是 True ，否则为 False 。

### QColorDialog 颜色对话框

self.connect(self.button, QtCore.SIGNAL('clicked()'),self.showDialog)

def showDialog(self):

col = QtGui.QColorDialog.getColor()

if col.isValid():

self.widget.setStyleSheet("QWidget { background-color: %s }"

% col.name())

颜色对话框为定制颜色提供一个对话框组件

color = QtGui.QColorDialog.getColor()

这行代码将会弹出一个 QColorDialog 。

if col.isValid():

self.widget.setStyleSheet("QWidget { background-color: %s }"

% col.name())

检查颜色是否有效，如果点击了取消按钮，将返回无效的颜色。如果颜色有些，我们使用样式修改背景颜色。

### QFontDialog 字体对话框

def showDialog(self):

font, ok = QtGui.QFontDialog.getFont()

if ok:

self.label.setFont(font)

### QFileDialog 打开文件

QFileDialog 允许用户选择文件或文件夹，可选择文件来打开和保存。

filename = QtGui.QFileDialog.getOpenFileName(self, ‘Open file’,'/home')

打开文件后filename为<class 'PyQt4.QtCore.QString'>

转为普通的字符串需要为filename=str(filename)

import sys

from PyQt4 import QtGui

from PyQt4 import QtCore

class Example(QtGui.QMainWindow):

def \_\_init\_\_(self):

super(Example, self).\_\_init\_\_()

self.initUI()

def initUI(self):

self.textEdit = QtGui.QTextEdit()

self.setCentralWidget(self.textEdit)

self.statusBar()

self.setFocus()

openFile = QtGui.QAction(QtGui.QIcon('open.png'), 'Open', self)

openFile.setShortcut('Ctrl+O')

openFile.setStatusTip('Open new File')

self.connect(openFile, QtCore.SIGNAL('triggered()'), self.showDialog)

menubar = self.menuBar()

fileMenu = menubar.addMenu('&File')

fileMenu.addAction(openFile)

self.setGeometry(300, 300, 350, 300)

self.setWindowTitle('OpenFile')

def showDialog(self):

filename = QtGui.QFileDialog.getOpenFileName(self,'Open file','/home')

fname = open(filename)

data = fname.read()

self.textEdit.setText(data)

app = QtGui.QApplication(sys.argv)

ex = Example()

ex.show()

app.exec\_()

这个例子显示一个菜单，中间放置一个文本编辑框，还有一个状态栏。状态机仅为了设计目的显示。菜单项显示 QFileDialog 来选择文件，文件的内容加载进文本编辑器。

class Example(QtGui.QMainWindow):

def \_\_init\_\_(self):

super(Example, self).\_\_init\_\_()

这个例子建立在 QMainWindow 组件上，因为我们需要在中间设置文本编辑器。

filename = QtGui.QFileDialog.getOpenFileName(self, ‘Open file’,'/home')

我们弹出 QFileDialog ， getOpenFileName 方法的第一个字符串是标题，第二个字符串指定对话框的工作目录，文件过滤默认设置 All files(\*) 。

fname = open(filename)

data = fname.read()

self.textEdit.setText(data)

读取选择的文件，并把文件内容放入文本编辑器。

### QMessageBox

自己设置QMessageBox的选项内容

msgBox = QtGui.QMessageBox()

msgBox.setText('What to do?')

msgBox.addButton(QtGui.QPushButton('Accept'), QtGui.QMessageBox.YesRole)

msgBox.addButton(QtGui.QPushButton('Reject'), QtGui.QMessageBox.NoRole)

msgBox.addButton(QtGui.QPushButton('Cancel'), QtGui.QMessageBox.RejectRole)

reply = QtGui.QMessageBox.**question**(self,'Message',"Are you sure to quit?",

QtGui.QMessageBox.Yes,QtGui.QMessageBox.No)

question 显示询问图标

warning 显示警告图标

critical 显示故障图标

单纯显示内容，不进行询问

QtGui.QMessageBox.about(self,"About", "About this application")

## 字体和背景

from PyQt4.QtGui import QPalette,QPixmap,QFont,QMainWindow,QLabel,QApplication

from PyQt4.QtCore import Qt

1. 设置labe的字体居中

self.label.setAlignment(Qt.AlignCenter)

此居中为上下左右居中

AlignRight

1. 设置label的背景颜色和字体颜色：

pe = QPalette()

pe.setColor(QPalette.WindowText,Qt.red)#设置字体颜色

self.label.setAutoFillBackground(True)#设置背景充满，为设置背景颜色的必要条件

pe.setColor(QPalette.Window,Qt.blue)#设置背景颜色

#pe.setColor(QPalette.Background,Qt.blue)<span style="font-family: Arial, Helvetica, sans-serif;">#设置背景颜色，和上面一行的效果一样

self.label.setPalette(pe)

颜色如果用rgb表示的话，则rgb的第四个数可以表示透明度，0为透明，255为不透明。

1. 设置字体的类型大小和加粗：

self.label.setFont(QFont("Roman times",10,QFont.Bold))

1. 设置字体大小和类型分别设置

font = QFont()

font.setFamily("Helvetica")

fontHeight = rect.height()/1

font.setPixelSize(fontHeight)

font.setBold(True)

self.label.setFont(font)

1. 样式表

某种控件统一设置 不同的字符串之间用加号连接上

**self.setStyleSheet**("QLabel{background:white;}"+

"QLabel{color:rgb(100,100,100,250);font-size:15px;font-weight:bold;font-family:Roman times;}"

+"QLabel:hover{color:rgb(100,100,100,120);}")

特定控件单独设置

CloseMinStyle="font-family:Consolas;font-size:40px;background-color:#fff;border: 0px solid #222222"

minimumButton.setStyleSheet(CloseMinStyle)

## 窗口

1. 设置最小化到托盘

import sys

from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):

def \_\_init\_\_(self):

super(Example, self).\_\_init\_\_()

self.initUI()

def initUI(self):

style = self.style()

# Set the window and tray icon to something

icon = style.standardIcon(QtGui.QStyle.SP\_MediaSeekForward) #取自pyqt模块中的图片

#设置自定义图片的话，icon ='icons/web.png'

self.tray\_icon = QtGui.QSystemTrayIcon()

self.tray\_icon.setIcon(QtGui.QIcon(icon))

self.setWindowIcon(QtGui.QIcon(icon))

# Restore the window when the tray icon is double clicked.

self.tray\_icon.activated.connect(self.restore\_window)

def event(self, event):

#如果窗口的状态改变，并且是最小化

if (event.type() == QtCore.QEvent.WindowStateChange and self.isMinimized()):

self.setWindowFlags(self.windowFlags() & ~QtCore.Qt.Tool)

self.tray\_icon.show()

return True

else:

return super(Example, self).event(event)

def closeEvent(self, event):

reply = QtGui.QMessageBox.question(

self,

'Message',"Are you sure to quit?",

QtGui.QMessageBox.Yes | QtGui.QMessageBox.No,

QtGui.QMessageBox.No)

if reply == QtGui.QMessageBox.Yes:

event.accept() #关闭窗口

else:

self.tray\_icon.show() #托盘图标显示

self.hide() #隐藏窗口

event.ignore() 不关闭窗口

def restore\_window(self, reason):

if reason == QtGui.QSystemTrayIcon.DoubleClick:

self.tray\_icon.hide() #托盘图标消失

self.showNormal() #恢复窗口

def main():

app = QtGui.QApplication(sys.argv)

ex = Example()

ex.show()

sys.exit(app.exec\_())

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

main()

1. 设置窗口大小 图标 标题

self.setGeometry(300, 300, 250, 150)

self.setWindowTitle('Icon')

self.setWindowIcon(QtGui.QIcon('icons/web.png'))

所有这三个类，均继承自 QtGui.QWidget ， setGeometry() 方法干了两件事，它定义了窗体在屏幕上的位置，并设置窗体的大小。开始的两个参数是窗体的x和y的位置，第三个是宽度，第四个是高度。最后一个方法设置了应用程序图标。通过这个，我们创建了一个 QIcon 对象， QIcon 接受我们想要显示的图标的路径。

1. 关闭时跳出窗口提醒是否关闭

reply = QtGui.QMessageBox.question(self, 'Message',"Are you sure to quit?", QtGui.QMessageBox.Yes, QtGui.QMessageBox.No)

我们显示一个两个按钮的消息框，Yes和No。第一个字符串显示在标题栏上，第二个字符串是信息文本，显示在对话框中，返回值存储在 relay 变量中。

if reply == QtGui.QMessageBox.Yes:

event.accept() #关闭窗口

else:

event.ignore() #不关闭窗口

这里我们测试返回值，如果点击Yes按钮，表明接受这个事件，将会关闭窗口组件并进回到应用程序的结尾，反之则忽略该关闭事件。

如果使跳出的窗口为3个选项，如下

reply = QtGui.QMessageBox.question(self,'Message',"Are you sure to quit?",

QtGui.QMessageBox.Yes | QtGui.QMessageBox.No | QtGui.QMessageBox.Cancel,

QtGui.QMessageBox.Yes)

1. 使窗口打开在屏幕正中间

self.resize(250, 150)

调整这个 QWidget 的大小为250像素宽和150像素高。

screen = QtGui.QDesktopWidget().screenGeometry()

计算屏幕分辨率。

size = self.geometry()

获得 QWidget 的尺寸。

self.move((screen.width()-size.width())/2, (screen.height()-size.height())/2)

1. 固定窗口大小

一种方法是设置它的最大大小和最小大小，并且使它们的值相等，都等于当前你设置的窗口尺寸。

setMinimumSize(370, 150);

setMaximumSize(370, 150);

一种方法是使用setFixedSize()

self.setFixedSize(300, 400)

1. 使窗体置顶 self.setWindowFlags(Qt.WindowStaysOnTopHint)
2. **使窗体没有边框**： your\_window.setWindowFlags(QtCore.Qt.FramelessWindowHint)
3. 使窗体取消置顶：

This should disable it:

window.setWindowFlags(window.windowFlags() & ~QtCore.Qt.WindowStaysOnTopHint)

This should enable it:

window.setWindowFlags(window.windowFlags() | QtCore.Qt.WindowStaysOnTopHint)

1. 支持在窗口任何位置按住左键并拖动？

class MainWindow(QtGui.QWidget): #继承QtGui.QWidget类

def \_\_init\_\_(self):

QtGui.QWidget.\_\_init\_\_(self) #继承QtGui.QWidget的\_\_init\_\_(self)函数并补充下面内容

#初始化position

self.m\_DragPosition=self.pos()

self.resize(300,400)

self.setWindowFlags(Qt.FramelessWindowHint | Qt.WindowStaysOnTopHint) #同时无边框和置顶

self.setMouseTracking(True)

#支持窗口拖动,重写两个方法

def mousePressEvent(self, event):

if event.button()==Qt.LeftButton:

self.m\_drag=True

self.m\_DragPosition=event.globalPos()-self.pos()

event.accept()

def mouseMoveEvent(self, QMouseEvent):

if QMouseEvent.buttons() and Qt.LeftButton:

self.move(QMouseEvent.globalPos()-self.m\_DragPosition)

QMouseEvent.accept()

def mouseReleaseEvent(self, QMouseEvent):

self.m\_drag=False

1. 使窗口最小化

self.showMinimized()

如果是点击按钮'-'最小化

minimumButton = QtGui.QPushButton('-')

minimumButton.setFixedSize(25,25)

minimumButton.setStyleSheet("QPushButton:hover{background-color:rgb(100,100,100,120)}"+CloseMinStyle)

grid.addWidget(minimumButton, 0, 1)

self.connect(minimumButton, QtCore.SIGNAL('clicked()'),self.showMinimized)

1. 退出后，后台继续保持进程

app.setQuitOnLastWindowClosed(True) #true 关闭， false 开启后台

1. 窗体透明度

self.setWindowOpacity(0.5) # 1表示100%显示， 0.5 透明度50%

## 细节

1. QtGui.QWidget QtGui.QMainWindow区别： 如果需要使用到菜单栏、状态栏、工具栏时要使用QtGui.QMainWindow
2. 显示图片

from PyQt4 import QtGui

led = QtGui.QLabel(window)

led.setPixmap(QtGui.QPixmap(os.getcwd() +"\\123.png"))

led.show()

1. 调整图片大小

pixmap = QtGui.QPixmap(path)

pixmap2 = pixmap.scaledToWidth(64) #使图片变为宽度为64像素

pixmap3 = pixmap.scaledToHeight(64) #使图片变为高度为64像素

pixmap4 = pixmap.scaled(64, 64, QtCore.Qt.KeepAspectRatio) 使图片变为宽度64、高度64的

1. 气泡帮助

self.setToolTip('This is a <b>QWidget</b> widget')

通过调用 setTooltip() 方法来创建提示，我们使用富文本格式。

QtGui.QToolTip.setFont(QtGui.QFont('OldEnglish', 10))

因为默认的 QToolTip 字体太难看，我们改变了它。

1. 关闭按钮

quit = QtGui.QPushButton('Close', self)

quit.setGeometry(10, 10, 60, 35)

创建一个按钮并放置在 QWidget 上，就像把 QWidget 放置在屏幕上一样。

self.connect(quit, QtCore.SIGNAL('clicked()'),QtGui.qApp, QtCore.SLOT('quit()'))

1. 设置编码

self.comboBox.addItem(\_fromUtf8(""))

1. PyQt4 names showing as undefined in eclipse, but it runs fine？

因为PyQt4内核是用c++编写的。不是.py文件，PyDev无法解析。

1. 将designer的ui文件转为py文件

python C:\Python27\Lib\site-packages\PyQt4\uic\pyuic.py test\_dialog.ui -o test.py -x

# 组成exe

1. 在windows上安装python 的模块后，导入模块时报 python ImportError: DLL load failed: %1 不是有效的 Win32 应用程序这个是因为你安装了64位的python，然后安装32位的模块，或者你安装了32位的python，然后安装64位的模块
2. GUI 图形用户页面 所以在电脑软件必须用到python GUI 首选Tkinter, tk本身就是为了快速GUI开发而生,且经过多年发展，已经十分成熟，而且Tkinter是python内置的事实上的标准GUI库。

还有pyqt wxpython Pyside pyqt的页面成熟好看。Tkinter的页面部件少，页面朴素

## py2exe模块

### 基础

使用

python setup.py py2exe 使用该模块将程序生成exe

版本 py2exe-0.6.9.win32-py2.7.exe py2exe-0.6.9.win64-py2.7.amd64.exe

如果你有一个名为helloworld.py的python脚本，你想把它转换为运行在windows上的可执行程序，并运行在没有安装python的windows系统上。

helloworld.py内容：

import win32api

import win32con

win32api.MessageBox(0, 'hi!', 'Python', win32con.MB\_OK)

setup.py编写

那么首先你应写一个用于发布程序的设置脚本例如setup.py，在其中的setup函数前插入语句import py2exe 。

setup.py示例如下:

# setup.py

from distutils.core import setup

import py2exe

setup(windows=[*"helloworld.py"*])

方括号中就是要编译的脚本名，前边的windows表示将其编译成GUI程序。如果要编译命令行界面的可执行文件，只要将windows改为console。如果需要将脚本编译成Windows服务，则可以使用service选项。

然后按下面的方法运行setup.py:

python setup.py py2exe

生成的目录结构

dist文件夹可以改名，exe名称也可改变，并不会影响

上面的命令执行后将产生一个名为dist的子目录，其中包含了helloworld.exe,python24.dll,library.zip这些文件。

如果你的helloworld.py脚本中用了已编译的C扩展模块，那么这些模块也会被拷贝在个子目录中，同样，所有的dll文件在运行时都是需要的，除了系统的dll文件。

dist子目录中的文件包含了你的程序所必须的东西，你应将这个子目录中的所有内容一起发布。

指定额外的文件一些应用程序在运行时需要额外的文件，诸如配置文件、字体、位图。

如果在安装脚本中用data\_files可选项指定了那些额外的文件，那么py2exe能将这些文件拷贝到dist子目录中。data\_files应包含一个元组(target-dir, files)列表，其中的files是这些额外的文件的列表。

from distutils.core import setup

import glob

import py2exe

setup(console=["helloworld.py"],

data\_files=[("bitmaps", ["bm/large.gif", "bm/small.gif"]),

("fonts", glob.glob("fonts\\\*.fnt"))])

说明：data\_files选项将创建一个子目录dist\bitmaps，其中包含两个.gif文件；一个子目录dist\fonts，其中包含了所有的.fnt文件。

默认情况下，py2exe在目录dist下创建以下这些必须的文件：

1、一个或多个exe文件。

2、python##.dll。

3、几个.pyd文件，它们是已编译的扩展名，它们是exe文件所需要的；加上其它的.dll文件，这些.dll是.pyd所需要的。

4、一个library.zip文件，它包含了已编译的纯的python模块如.pyc或.pyo

上面的mysetup.py创建了一个控制台的helloword.exe程序，如果你要创建一个图形用户界的程序，那么你只需要将mysetup.py中的console=["helloworld.py"]替换为windows=["helloworld.py"]既可。

py2exe一次能够创建多个exe文件，你需要将这些脚本文件的列表传递给console或windows的关键字参数。如果你有几个相关联的脚本，那么这是很有用的。

运行下面个命令，将显示py2exe命令的所有命令行标记。

python mysetup.py py2exe –help

### python编译的所有文件打包到一个exe中

就需要在setup()这个函数中，要设置2个参数：options中的bundle\_files和zipfile。

其中bundle\_files有效值为：

3 (默认)不打包。

2 打包，但不打包Python解释器。

1 打包，包括Python解释器。

zipfile的有效值为:

不填(默认) 生成一个library.zip文件

None 把所有东西打包进.exe文件中

#安装成windows服务的python脚本

#内容：

from distutils.core import setup

import py2exe

options = {"py2exe":{"compressed": 1, #压缩

"optimize": 2,

"bundle\_files": 1 #所有文件打包成一个exe文件

}}

setup( service=["PyWindowsService"], options=options, zipfile=None)

或者options直接写在里面

setup(console=['MessageBox.py'],

version='1.0',

description='for test',

name='fortest',

options = {"py2exe": {"compressed": 1,"optimize": 2,"ascii": 0,"bundle\_files": 1,

'packages': ['urllib2', 'urllib'], #如果某个模块不起作用，包含进来comma-separated list of packages to include

'includes': ['lxml.etree', 'lxml.\_elementpath', 'gzip', 'processcls'],

# comma-separated list of modules to include

'compressed' : True

}},

zipfile=None)

旁边还有一个w9xpopen.exe的文件，明显是用来在win9x下运行用的，这个文件可以直接删了。

### 遇到的问题

1. SyntaxError: invalid syntax

通过查找源代码，发现如print(" builtin: %s" % name) ，则可知这是python3.0以上的写法

python2.7是不加括号 print "builtin: %s" % name

1. tablib模块调用时报错 InvalidDimensions

data = tablib.Dataset(\*self.case\_cashe, headers=tuple(self.columnlist))

是因为导入的data的数据原因 \*self.case\_cashe数据有问题

1. UnicodeEncodeError: 'ascii' codec can't encode characters in position 16-25: ordinal not in range

f.write(data.html.encode("utf-8")) 变成"utf-8"格式再写入

# 使得 sys.getdefaultencoding() 的值为 'utf-8'

reload(sys) # reload 才能调用 setdefaultencoding 方法

sys.setdefaultencoding('utf-8') # 设置 'utf-8'

1. error: [Errno 2] No such file or directory: 'MSVCP90.dll'

setup(windows=["chapter13.py"],options = { "py2exe":{"dll\_excludes":["MSVCP90.dll"]}})

1. ImportError: No module named \_qt

setup(windows=[{"script" : "app.pyw"}], options={"py2exe" : {"includes" : ["sip", **"PyQt4.\_qt"**]}})

1. ImportError: No module named sip

python setup.py py2exe --includes sip

或者setup(windows=[{"script":"main.py"}], options={"py2exe":{"includes":["sip"]}})

1. 打包后，发现urllib2不起作用，但是不会报错，怎么解决？

from distutils.core import setup

import py2exe

setup(windows=[{"script":"client.py"}],options = { "py2exe":{"compressed": 1, #压缩

"optimize": 2,

"bundle\_files": 1, #所有文件打包成一个exe文件

"dll\_excludes":["MSVCP90.dll"],

"includes" :["sip"],

'packages': ['urllib2', 'urllib'],

}

}, zipfile=None)

## PySide模块

### 基础

PySide就是其Qt库的Python移植（原先是C++的）。

PySide的使用非常方便，下面是一个简单是示例，用纯命令行建立一个标题为Test Form的窗口：

import sys

from PySide.QtCore import \*

from PySide.QtGui import \*

# Define Form

class Form(QDialog):

def \_\_init\_\_(self,parent=None):

super(Form,self).\_\_init\_\_(parent)

self.setWindowTitle('Test Form')

# Main function

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

# Create Qt App

app=QApplication(sys.argv)

# Create window

NewForm=Form()

NewForm.show()

# exit

sys.exit(app.exec\_())

### Qt Designer

#### 各控件解释

C:\Python27\Lib\site-packages\PySide下 designer.exe

默认左边是控件栏，提供了很多空间类，我们可以直接拖放到widget中看到效果。每个空间都有自己的名称，提供不同的功能，比如常用的按钮、输入框、单选、文本框等等。

右边是对窗口及控件的各种调整、设置、添加资源（列如:图片）、动作。还可以直接编辑Qt引以为豪的信号槽（signal和slot）。

预览窗口：点窗体form--预览preview（Ctrl+R）

窗口布局Layouts

Vertical Layout 纵向布局

Horizontal Layout 横向布局

Grid Layout 栅格布局

Form Layout 在窗体布局中布局

把控件放在这些layouts中，就会按layout所规定的方式对齐

spacers 分别是横向和竖向占位符

containers容器

GroupBox好处就是用户可以比较清晰地了解程序的界面，我们可以把功能相关的控件放到一个GroupBox中。

ScrollArea控件用来显示子控件的内容的框架，如果子控件的尺寸超过了框架的大小，可以使用滚动条，方便查看整个子控件

ToolBox提供了一系列的页和隔间，就像Qt Creator中的工具箱一样。

TabWidget控件切换卡控件顶部或底部有一个标签选项栏，每个标签选项都有一个页面，选择哪个页面，只需单击对应的标签即可，或按指定ALT+字母快捷键组合即可。

Frame控件框架控件用来存放其他控件，也可用于装饰，它一般用来作为更加复杂容器的基础。也可以用在form中作为占用控件。

Widget控件在创建时是不可见的，他可以包含子控件，在删除该Widget时，子控件也一起删除。

DockWidget控件停靠窗体，可以作为一个顶层窗口漂浮在桌面，主要作为辅助窗体出现在界面中，可以在很多IDE中看到停靠窗体

#### 与程序交互

**把ui文件转换成为py文件：** 进入ui文件存放的目录的cmd，输入命令 pyside-uic TestGUI.ui -o TestGUI.py 顺利的话就可以在同一个目录下拿到TestGUI.py文件

实际上是生成了一个类，这个类包含了窗口的所有信息。

然后来调用这个类，并生成新窗口：

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

import sys

# Import Qt GUI component

from PySide.QtGui import \*

# Import GUI File

from TestGUI import Ui\_Form

# Self Function

def PrintHello():

print("Hello")

# Make main window class

class MainWindow(QMainWindow,Ui\_Form):

def \_\_init\_\_(self, parent=None):

super(MainWindow,self).\_\_init\_\_(parent)

self.setupUi(self)

# Connect button click event to PrintHello function

self.pushButton.clicked.connect(PrintHello)

# End of main window class

# Main Function

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

Program = QApplication(sys.argv)

Window=MainWindow()

Window.show()

Program.exec\_()

打包成exe格式

from distutils.core import setup

import py2exe

setup(windows=['TestProgramWithGUI.py','MessageBox.py'], #造两个exe的方法，两个exe互不干扰

version='1.0',

description='for test',

name='fortest',

options = {"py2exe": {"compressed": 1,"optimize": 2,"ascii": 0,"bundle\_files": 3}},

zipfile=None)

由于是窗体文件，setup里用的是windows选项

问题：

1. WIN10 计算机中丢失MSVCP120.dll

重新下载一个msvcp120.dll文件，复制到C:\Windows\System32目录下和 C:\Windows\SysWOW64目录

## pywin32

Python是没有自带访问windows系统API的库的，需要下载。库的名称叫pywin32，可以从网上直接下载。

介绍这个库里面最重要的两个模块：win32api和win32con。win32api顾名思义，就是用python对win32的本地api进行了封装；win32con个人理解为win32constant，即win32的常量定义。

没有pywin64.但是pywin32有64位版本的。pywin32-219.win-amd64-py2.7.exe

http://sourceforge.net/projects/pywin32/files%2Fpywin32/

import win32api

import win32con

win32api.MessageBox(win32con.NULL, u'Python 你好', u'你好', win32con.MB\_OK)

u'Python 你好' 为弹窗中内容， u'你好'为弹窗标题 如果含有中文，请在字符串前面加上u

## wxpython

ImportError: No module named wx 的解决方案是：

安装wxpython.

Go to C:\Python27\Lib\site-packages\ find the folder wx-<version>-msw or similar

Move the wx from the above folder to C:\Python27\Lib\site-packages\

# 爬虫wiki文档平台

## wiki\_scrapy文件

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

from scrapy import Spider

from scrapy.selector import Selector

from wikidoc.items import WikidocItem

import logging

class wikidocSpider(Spider):

name='wikidoc'

allowed\_domains=['scm-wiki.jd.com']

start\_urls=["http://scm-wiki.jd.com/bin/view/运营研发测试组/2015-Q4?viewer=attachments",]

def parse(self,response):

sel=Selector(response)

liResults=sel.xpath('//div[@id="\_attachments"]/div/div') #筛选出所符合的，组合为一list

#items=[]

for gisy in liResults: #对list中各个记录进行循环计算

item=WikidocItem()

item['name']=gisy.xpath('span[@class="name"]/a/text()').extract()[0]

item['date']=gisy.xpath('div[@class="meta"]/span[@class="date"]/text()').extract()[0]

item['writer']=

gisy.xpath('div[@class="meta"]/span[@class="publisher"]/span/a/text()').extract()[0]

item['url']=gisy.xpath('span[@class="name"]/a/@href').extract()[0]

#items.append(item)

yield item #用yield就不用注释的items列表

#return items

因为用extract这个方法得到的是一个数组，而我们的每一个字段实际上是一个string而非一个array，如果不取第一个值，那么存入mongodb之后，title这个key对应的value是一个数组，这会导致我们将mongodb里的数据转换成json之后需要在客户端再进行分解。很麻烦。

## pipelines文件 连接mongodb mysql

import pymongo

from scrapy.conf import settings

from scrapy.exceptions import DropItem

from scrapy import log

class MongoDBPipeline(object):

def \_\_init\_\_(self):

connection=pymongo.MongoClient(

settings['MONGODB\_SERVER'],

settings['MONGODB\_PORT']

)

db=connection[settings['MONGODB\_DB']]

self.collection=db[settings['MONGODB\_COLLECTION']]

def process\_item(self,item,spider):

valid=True

for data in item:

if not data:

valid=False

raise DropItem('Missing{0}!'.format(data))

if valid:

self.collection.update({'url':item['url']},dict(item),upsert=True)

log.msg("Question added to MongoDB database!",

level=log.DEBUG, spider=spider) #显示在console上

return item

#self.collection.update({'url': item['url']}, dict(item), upsert=True) 它意味着如果一个问题已经存在数据库中，我们将更新它的标题；否则我们将新问题插入数据库中。

#下面类是调用mysql

class MySQLPipeline(object):

def \_\_init\_\_(self):

self.dbpool=adbapi.ConnectionPool('MySQLdb',

host=settings['MYSQL\_HOST'],

db =settings['MYSQL\_DBNAME'],

user =settings['MYSQL\_USER'],

passwd =settings['MYSQL\_PASSWD'],

cursorclass = MySQLdb.cursors.DictCursor,

charset = 'utf8',

use\_unicode = True )

# pipeline dafault function #这个函数是pipeline默认调用的函数

def process\_item(self, item, spider):

query = self.dbpool.runInteraction(self.\_conditional\_insert, item)

return item

# insert the data to databases #把数据插入到数据库中

def \_conditional\_insert(self, tx, item):

sql = "insert into wiki(name,date,writer,url) values (%s,%s,%s,%s)"

tx.execute(sql,

(item['name'],item['date'],item['writer'], item['url']

))

如果报错\_mysql\_exceptions.OperationalError: (1136, "Column count doesn't match value count at row 1")，

请记得写成wiki(name,date,writer,url)，而不是wiki

## setting文件

BOT\_NAME = 'wikidoc'

SPIDER\_MODULES = ['wikidoc.spiders']

NEWSPIDER\_MODULE = 'wikidoc.spiders'

ITEM\_PIPELINES={'wikidoc.pipelines.MongoDBPipeline':300,

'wikidoc.pipelines.MySQLPipeline': 300,}

MONGODB\_SERVER='127.0.0.1'

MONGODB\_PORT=27017

MONGODB\_DB='wikidoc'

MONGODB\_COLLECTION='wiki'

DOWNLOAD\_DELAY = 1

# start MySQL database configure setting

MYSQL\_HOST = '192.168.166.17'

MYSQL\_DBNAME = 'test\_gisy'

MYSQL\_USER = 'test\_write'

MYSQL\_PASSWD ='test\_write123'

# end of MySQL database configure setting

## items文件

from scrapy.item import Item,Field

class WikidocItem(Item):

name=Field()

date=Field()

writer=Field()

url=Field()

## mongo查询语句

use wikidoc #先切换到所在数据库，再执行查询命令

db.wiki.find({ "writer" : "立江 王" }, { "date" : 1, "url" : 1 }).limit(50).sort({ "date" : -1 });

## mysql数据库表设置

wiki表 id为自增键，必须为int类型

CREATE TABLE `wiki` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(200) NOT NULL,

`date` varchar(100) NOT NULL,

`writer` varchar(50) NOT NULL,

`url` varchar(200) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `name` (`name`,`url`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

## 创建爬取组下的所有页面链接

wikiurl\_scrapy文件

from scrapy import Spider

from scrapy.selector import Selector

from wikidoc\_url.items import WikiurlItem

class wikidocSpider(Spider):

name='wikiurl'

allowed\_domains=[]

start\_urls=["http://scm-wiki.jd.com/bin/view/运营研发测试组/",]

#定义提取链接的规则，继续跟进，提取到的链接回调给parse函数作为参数

def parse(self,response):

sel=Selector(response)

liResults=sel.xpath('//\*[@id="gadget\_0"]/div/ul[1]/li') #筛选出所符合的，组合为一list

for gisy in liResults:

item=WikiurlItem()

item['name']=gisy.xpath('div/a/text()').extract()[0]

item['url']=gisy.xpath('div/a/@href').extract()[0]

yield item #每条记录组合成一个mongo记录，dict格式

## flask

### html

带有人名选项的选择列表

<select>

<option value ="桦 刘">刘桦</option>

<option value ="立江 王">王立江</option>

<option value="xuemei zhou">周雪梅</option>

<option value="老虎 例">栗雄虎</option>

<option value ="Jinming Zhang">张晋明</option>

<option value="柯 琴琴">柯琴琴</option>

<option value="Eko Guo">郭玉锐</option>

<option value="焕 方">方焕</option>

<option value ="he yujiao">贺玉娇</option>

<option value ="莹莹 段">段莹莹</option>

<option value ="峰 刘">刘峰</option>

<option value="wangkaibin">王凯斌</option>

</select>

### 细节

表单获取到的格式是unicode格式，转为str格式。其实没必要转换。

startdate=request.form['startdate']

startdate=str(startdate)

if startdate and enddate:

startdate='/'.join(startdate.split('-'))

enddate='/'.join(enddate.split('-'))

wikidoc=collection.find({ "date" : {"$gte":startdate,"$lte":enddate}})

### 全文

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

from flask import Flask,request,render\_template

import pymongo

import logging

import re,os,subprocess

import time

app=Flask(\_\_name\_\_)

app.config.from\_object('config')

@app.route('/', methods=['GET','POST'])

def wikidoc():

client=pymongo.MongoClient('127.0.0.1',27017)

db=client.wikidoc

collection=db.wiki

#获取最近抓取时间

fo=open('lasttime.txt', "r")

lasttime=fo.readline()

updatetime=lasttime

lasttime=time.mktime(time.strptime(lasttime,'%Y/%m/%d, %H:%M'))

app.logger.debug(lasttime)

fo.close()

if request.method=='GET':

wikidoc=collection.find().sort('date', pymongo.DESCENDING)

wikidoc\_count=wikidoc.count()

return render\_template('wikidoc.html',result=wikidoc,num=wikidoc\_count,updatetime=updatetime)

if request.method=='POST':

if request.form["action"] == "查询":

search=request.form['search']

writer=request.form['writer']

startdate=request.form['startdate']

enddate=request.form['enddate']

wikidoc=collection.find().sort('date', pymongo.DESCENDING)

if writer:

wikidoc=collection.find({ "writer" : writer }).sort('date', pymongo.DESCENDING)

app.logger.debug(writer)

if search:

rex='^.\*'+search+'.\*'

app.logger.debug(rex)

rexExp = re.compile(rex)

wikidoc=collection.find({ "name" : rexExp}).sort('date', pymongo.DESCENDING)

if search and writer:

wikidoc=collection.find({ "name" : rexExp,'writer':writer }).sort('date', pymongo.DESCENDING)

if startdate and enddate:

startdate='/'.join(startdate.split('-'))

enddate='/'.join(enddate.split('-'))

wikidoc=collection.find({ "date" : {"$gte":startdate,"$lte":enddate}}).sort('date', pymongo.DESCENDING)

app.logger.debug([startdate,enddate])

if startdate and enddate and writer:

wikidoc=collection.find({ "date" : {"$gte":startdate,"$lte":enddate},"writer" : writer}).sort('date', pymongo.DESCENDING)

if startdate and enddate and search:

wikidoc=collection.find({ "date" : {"$gte":startdate,"$lte":enddate}, "name" : rexExp}).sort('date', pymongo.DESCENDING)

if startdate and enddate and search and writer:

wikidoc=collection.find({ "date" : {"$gte":startdate,"$lte":enddate}, "name" : rexExp,"writer" : writer}).sort('date', pymongo.DESCENDING)

wikidoc\_count=wikidoc.count()

return render\_template('wikidoc.html',result=wikidoc,num=wikidoc\_count,writer=writer,search=search,startdate=startdate,enddate=enddate,updatetime=updatetime)

else:

interval=app.config["INTERVAL"]

nowtime=time.time()

if nowtime-interval\*60>lasttime:

#获取所有页面链接的爬虫执行

app.logger.debug(app.config["URLGET"])

url\_get=subprocess.Popen(app.config["URLGET"], shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

for i in url\_get.stdout.readlines():

app.logger.debug(i)

#通过爬取所有页面获取所有文档信息爬虫执行

doc\_get=subprocess.Popen(app.config["DOCGET"], shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

for i in doc\_get.stdout.readlines():

app.logger.debug(i)

updatetime=time.strftime("%Y/%m/%d, %H:%M")

fo=open('lasttime.txt', "w+")

fo.write(updatetime)

fo.close

message='重新抓取成功。'

else:

message='离上次抓取时间间隔未到'+str(interval)+'分钟，为了安全性，此时不准抓取。'

#从数据库获取最新信息

wikidoc=collection.find().sort('date', pymongo.DESCENDING)

wikidoc\_count=wikidoc.count()

return render\_template('wikidoc.html',result=wikidoc,num=wikidoc\_count,updatetime=updatetime,message=message)

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

handler=logging.FileHandler('flask.log',encoding='UTF-8')

handler.setLevel(logging.DEBUG)

logging\_format=logging.Formatter('%(asctime)s %(levelname)s %(filename)s (%(funcName)s:%(lineno)s) - %(message)s' )

handler.setFormatter(logging\_format)

app.logger.addHandler(handler)

app.run(host="localhost",port=5000, debug=True)

#subprocess.Popen 对linux也有效。相当于在linux页面上输入命令

## 问题解决

1. mysql的 unsigned 既为非负数，用此类型可以增加数据长度!

例如如果 tinyint最大是127，那 tinyint unsigned 最大 就可以到 127 \* 2

unsigned 属性只针对整型，而binary属性只用于char 和varchar。

#定义提取链接的规则，继续跟进，提取到的链接回调给parse函数作为参数

rules =

[ Rule(SgmlLinkExtractor(allow=(u'/bin/view/%E8%BF%90%E8%90%A5%E7%A0%94%E5%8F%91%E6%B5%8B%E8%AF%95%E7%BB%84'),

restrict\_xpaths=('//\*[@id="leftPanels"]/div/div/p/span/a')),

callback='parse',

follow=False)]

1. 查看对象里包含的方法：对象为wikidoc dir(wikidoc)
2. 模板中循环字典的value值。
   1. {%for i in gisy.values()%}
3. <td >{{i}}</td>
4. 点击下载，即可下载文件？ <td ><a href="{{gisy.url|e}}">下载</a></td>

# 二维码查看器

## 客户端

### qrcode模块

import qrcode

qr = qrcode.QRCode(

version=1,

error\_correction=qrcode.constants.ERROR\_CORRECT\_L,

box\_size=10,

border=4,

)

qr.add\_data('http://192.168.156.105:2015/pptctrl/?os=1&id=2e228109191b40a21d8c9adbd410887d&st=1466990460\_23078&lc=1659010aa0f3')

qr.make(fit=True)

img = qr.make\_image()

img.save('123.png')

version：值为1~40的整数，控制二维码的大小（最小值是1，是个12×12的矩阵）。 如果想让程序自动确定，将值设置为 None 并使用 fit 参数即可。

error\_correction：控制二维码的错误纠正功能。可取值下列4个常量。

　　ERROR\_CORRECT\_L：大约7%或更少的错误能被纠正。

　　ERROR\_CORRECT\_M（默认）：大约15%或更少的错误能被纠正。

　　ROR\_CORRECT\_H：大约30%或更少的错误能被纠正。

box\_size：控制二维码中每个小格子包含的像素数。

border：控制边框（二维码与图片边界的距离）包含的格子数（默认为4，是相关标准规定的最小值）

### 获取mac地址

获取mac地址

from uuid import getnode as get\_mac

mac = get\_mac()

print mac

### 问题

1. IOError: [Errno 13] Permission denied: 说明该文件存在，但是程序无法打开该文件，可能已经被其他的程序行打开
2. 如何post一个list? 无法直接post一个list，应该将list转为其他格式，比如xml或者json，然后post
3. urllib2.URLError: <urlopen error [Errno 10061] > 传入excel时报错，说明服务端的程序没有启动。
4. 在svn上无法显示二维码？

不要写成os.getcwd()+"\\gisy.png",这是多此一举，

pixmap=QtGui.QPixmap("gisy.png")

## 服务端

### 基础

1. 客户端传入json，服务端解析json？

客户端

def export\_json(self):

data = json.dumps(self.excelCache)

print data

print type(data)

req = urllib2.Request(self.urlAndPort)

urllib2.urlopen(req, urllib.urlencode({'data': data}))

服务端 tornado框架

def post(self):

#尝试获取data数据

data = self.get\_argument('data')

logger.info(data.encode("utf-8")) #将接受到的json串显示到日志中，进行编码后才能正常显示中文。str格式才能encode,list格式不能encode

excelContentList = json.loads(data)

self.render('excelPhoneWeb.html')

1. 如何判断server.log的大小，并且大于一定值，进行清空

gisy=os.path.getsize('server.log') #获取文件的大小 字节为单位

if gisy>2000000:

f=open('server.log','a')

f.truncate() #清空文件内容

f.close()

1. 在同一个form中使用next、up两个submit按钮？

def post(self):

rowIndex=self.get\_argument('rowindex',1)

upInst=self.get\_argument('up',None)

nextInst=self.get\_argument('next',None)

rowIndex=int(rowIndex)

if rowIndex<len(globals()['test\_excelContent']) :

if nextInst:

rowIndex=rowIndex+1

elif upInst and rowIndex>1:

rowIndex=rowIndex-1

self.render('excelPhoneWeb.html',excelContentList= globals()['test\_excelContent'][rowIndex-1],topRow=globals()['test\_topRow'],rowIndex=rowIndex)

1. 如何使button right aligned 居右侧？

<form action="" method="post" **style="float: right;"**>

<span style='font: bold 14px consolas,"microsoft yahei"'>第{{rowIndex}}条</span>

<input type="text" name='rowindex' value='{{rowIndex}}' style='display:none'/>

<input type="submit" name='up' value="up" class='button' />

<input type="submit" name='next' value="next" class='button' />

</form>

1. 使文本居中

p.message{padding-top:50%;padding-bottom:30%}

1. 去除多余的空字符串

#去除多余的none值

def remove\_invalid\_data(self):

#对sequence中的item依次执行function(item)，将执行结果为True的item组成一个List/String/Tuple(取决于sequence类型）返回

def nullString(x):

return x!='' and x!=None

self.excelTemp=[]

for row in self.excelCache:

row = filter(nullString, row)

self.excelTemp.append(row)

self.excelCache=self.excelTemp

1. 读取xls格式的excel时date显示为日期格式而不是float形式

import datetime, xlrd

book = xlrd.open\_workbook("test.xls")

sh = book.sheet\_by\_index(0)

a1 = sh.cell\_value(rowx=0, colx=0)

a1\_as\_datetime = datetime.datetime(\*xlrd.xldate\_as\_tuple(a1, book.datemode))

print 'datetime: %s' % a1\_as\_datetime

### 问题

1. tornado加入显示后台日志功能后，每次登录网页都会显示

2016-07-12 14:41:21,638 INFO web.py (log\_request:1946) - 304 GET / (127.0.0.1) 1.00ms 为什么？

因为在tornado/web.py文件中就已经设置该日志

1. 在导入excel时报错TypeError: datetime.datetime(2016, 6, 28, 0, 0) is not JSON serializable

解决方案：

if type(col.value) in (datetime.datetime, datetime.date):

col.value=str(col.value)

1. 报错 File "excelPhoneWeb\_html.generated.py", line 16, in \_tt\_execute

\_tt\_tmp = topRow[i] # excelPhoneWeb.html:34 (via excelPhoneWeb.html:30)

IndexError: list index out of range

那是因为第一行的元素比展示的那行元素少导致的。

解决方案： html中

{%for i in range(len(excelContentList)) %}

{% if i<len(topRow) %}

<p class='toprow'>{{topRow[i]}}</p>

{% end %}

<p class='rowcontent'>{{excelContentList[i]}}</p>

{% end %}

## 手机html

自适应手机屏幕

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width,initial-scale=1,maximum-scale=1,user-scalable=no">

<meta name="apple-mobile-web-app-capable" content="yes" />

<title>excel PhoneLook</title>

<link href="css/main.css" rel="stylesheet" type="text/css" />

</head>

<body style="overflow:auto">

#overflow 属性规定当内容溢出元素框时发生的事情。

hidden 内容会被修剪，并且其余内容是不可见的。

auto 如果内容被修剪，则浏览器会显示滚动条以便查看其余的内容。

# loglogic模块

1. 匹配模块 fuzz.partial\_ratio 和Levenshtein.ratio 都比较有效

前面满分100 后面满分 1

1. 日志进行筛选，提取出不变的log

import uniout

b1=list('订单AskuId = C, sid = 0 , dcId = 5')

b2=list('订单BskuId = B, sid = 0 , dcId = 8')

print b1,b2

b3 = [val for val in b1 if val in b2]

print ''.join(b3)

1. worker形式

import time

while True:

foo()

time.sleep(10)

## grep命令

1. 只显示匹配出的字符，而不是匹配出的行 grep -o 正则表达式 文件名 比如 grep -o 开始处理
2. 设置匹配的最大数量阈值 grep -m2, --max-count=NUM 至多匹配两个
3. grep 正则表达式 grep -E 'word1|word2' 或 egrep 'word1|word2'
4. 不区分大小写 -i, --ignore-case 不区分大小写
5. 显示匹配了多少次 -c, --count

在括号内的表达式中，在“ [: ”和“ :] ”中所附的字符类的名称：代表属于该类的所有字符的列表。标准字符类名称：

[[:alnum:]] – 字母数字字符。[ 0-9a-zA-Z]

[[:alpha:]] – 字母大小写[a-zA-Z]

[[:blank:]] – 空格和制表符。

[[:digit:]] – 数字： [0-9]

[[:lower:]] – 小写字母：[a-z]

[[:space:]] – 特殊字符：制表符，换行符，垂直制表符、换页，回车，和空间。

[[:upper:]] – 大写字母：[A-Z]

[[::]] 字符集

1. 抓取订单开始处理日志

tail -60000 tomcatlog/catalina.out |egrep -o '订单：\[[[:digit:]]\*\]开始处理'

## 数据库 sqlite3

conn = sqlite3.connect('loglogic.db')

cursor = conn.cursor()

cursor.execute('CREATE TABLE `crucial\_factor` ( `id` INTEGER, `factorname` TEXT NOT NULL, `inlogid` INTEGER, `behindlogid` INTEGER, PRIMARY KEY(`id`), FOREIGN KEY(`inlogid`) REFERENCES logtrunk(id), FOREIGN KEY(`behindlogid`) REFERENCES logtrunk(id) )')

cursor.execute('CREATE TABLE `logtrunk` ( `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `log` TEXT NOT NULL UNIQUE )')

cursor.execute('CREATE TABLE `logicflow` ( `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, `name` TEXT UNIQUE, `flowjson` TEXT NOT NULL UNIQUE )')

cursor.close()

conn.commit()

conn.close()

## ORM操作

from peewee import \*

loglogicdb = SqliteDatabase(*'loglogic.db'*, \*\*{})

class **BaseModel**(Model):

class **Meta**:

database = loglogicdb

class **Logtrunk**(BaseModel):

log = TextField(unique=True)

class **Meta**:

db\_table = *'logtrunk'*

class **CrucialFactor**(BaseModel):

behindlogid = ForeignKeyField(db\_column=*'behindlogid'*, null=True, rel\_model=Logtrunk, to\_field=*'id'*)

factorname = TextField()

inlogid = ForeignKeyField(db\_column=*'inlogid'*, null=True, rel\_model=Logtrunk, related\_name=*'logtrunk\_inlogid\_set'*, to\_field=*'id'*)

class **Meta**:

db\_table = *'crucial\_factor'*

class **Logicflow**(BaseModel):

flowjson = TextField(unique=True)

name = TextField(null=True, unique=True)

class **Meta**:

db\_table = *'logicflow'*

## 生成graph

from \_ormBypeewee import \*

import time,re,os

#import uniout

from StringIO import StringIO

#digraph graphname {graph [encoding="UTF-8",rankdir=LR];1[label=Teacher];2[label=Pupil];1 -> 2[fontcolor=darkgreen,label=Instructions];}

class **produceDot**(object):

def **\_\_init\_\_**(*self*):

#提取出数据库数据

#先获取flowjson中存在的node,只有在这存在的node才能显示在逻辑中

\_logicflow\_content=Logicflow.select()

\_logicflow\_flowjson=[i.flowjson for i in \_logicflow\_content]

edge\_flowjson=*''*

\_existflow\_node=set()

for i in \_logicflow\_flowjson:

\_singleflowjsonlist=i.split(*"-"*)

# print \_singleflowjsonlist

\_existflow\_node.update(\_singleflowjsonlist) #收集node

\_singlelogicflow=*''*

for index,item in enumerate(\_singleflowjsonlist):

if index==0:

edgecontent=item+*'->'*

elif index+1==len(\_singleflowjsonlist):

edgecontent=item+*';'*

else:

edgecontent=item+*';'*+item+*'->'*

\_singlelogicflow=\_singlelogicflow+edgecontent

edge\_flowjson=edge\_flowjson+\_singlelogicflow

print \_existflow\_node

\_logtrunk\_result=Logtrunk.select().order\_by(Logtrunk.id)

\_logtrunk\_alllist=[(i.id,i.log) for i in \_logtrunk\_result]

node\_logtrunk=*''*

for key,value in \_logtrunk\_alllist:

print type(key)

if str(key) in \_existflow\_node:

singlenode=str(key)+*'[label=\"'*+value+*'\"];'*

print singlenode

node\_logtrunk=node\_logtrunk+singlenode

dot=*"digraph logic {graph [encoding=\"UTF-8\"];node [fontname = \"Microsoft YaHei\",shape=box];edge [fontname = \"Microsoft YaHei\"]; %s %s}"* %(node\_logtrunk,edge\_flowjson)

with open(*'logicGraph.dot'*,*'w+'*) as gisy:

gisy.write(dot)

class **produceImage**(object):

def **\_\_init\_\_**(*self*):

import pygraphviz

G = pygraphviz.AGraph(*'logicGraph.dot'*,)

# print G #输出生成的源码

G.layout(prog=*'dot'*)

G.draw(*'first.png'*)

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

produceDot()

if os.name==*'nt'*:

produceImage()

## 与外界交互 查看数据库信息

在cmd调用该模块文件，输出数据库数据。报错peewee.OperationalError: no such table: logtrunk

那是因为cmd下的路径不是在D:\eclipse\_workspace\loglogic下，而程序设置数据库为

loglogicdb = SqliteDatabase('loglogic.db', \*\*{})

如果cmd路径在D:\eclipse\_workspace\，则D:\eclipse\_workspace\loglogic.db不存在，也就是找不到数据库了

from \_ormBypeewee import \*

import time,re

import difflib

import uniout

import argparse

import os,sys

parser = argparse.ArgumentParser(description='GetandUpdate loglogic dbdata ')

parser.add\_argument('-t','--logtrunk', action="store\_true",dest='select\_logtrunk',default=False)

parser.add\_argument('-f','--logicflow', action="store\_true", dest="select\_logicflow",default=False)

parser.add\_argument('-d','--difflib', nargs='?',action="store", dest="select\_difflib",default=None,const=0.9,type=float)

parser.add\_argument('-u','--update', nargs=2,action="store", dest="update\_difflib",default=None,type=int)

args= parser.parse\_args()

print args

class selectLogtrunk(object):

def \_\_init\_\_(self):

\_logtrunk\_result=Logtrunk.select().order\_by(Logtrunk.id)

\_logtrunk\_alllist=[(i.id,i.log) for i in \_logtrunk\_result]

print \_logtrunk\_alllist

class selectLogicflow(object):

def \_\_init\_\_(self):

\_logicflow\_content=Logicflow.select()

\_logicflow\_alllist=[(i.id,i.name,i.flowjson) for i in \_logicflow\_content]

print \_logicflow\_alllist

#查询logtrunk中相似的 格式为{(outid,reid):(outlog,relog),}

class selectDifflib(object):

def \_\_init\_\_(self,proportion=0.9):

\_logtrunk\_result=Logtrunk.select().order\_by(Logtrunk.id)

\_logtrunk\_loglist=[i.log for i in \_logtrunk\_result]

replaceDict={}

numid=1

for index,item in enumerate(\_logtrunk\_loglist):

if index+1<len(\_logtrunk\_loglist):

for i,t in enumerate(\_logtrunk\_loglist[index+1:]):

if difflib.SequenceMatcher(None,item,t).ratio()>proportion:

# tup=(item,t)

# replaceDict.setdefault(numid,tup)

numid+=1

out = Logtrunk.select().where(Logtrunk.log == t).get()

replace= Logtrunk.select().where(Logtrunk.log == item).get()

tup1=(out.id,replace.id)

tup2=(t,item)

replaceDict.setdefault(tup1,tup2)

print replaceDict

class updateDifflib(object):

def \_\_init\_\_(self,convertlist):

if isinstance(convertlist, (tuple, list, set)):

q = Logtrunk.delete().where(Logtrunk.id== convertlist[0])

q.execute()

\_logicflow\_content=Logicflow.select()

\_logicflow\_alllist=[(i.id,i.flowjson) for i in \_logicflow\_content]

for flowid,flowitem in \_logicflow\_alllist:

flowitem=re.sub(r'\b'+str(convertlist[0])+r'\b',str(convertlist[1]),flowitem)

print flowitem

q = Logicflow.update(flowjson=flowitem).where(Logicflow.id==flowid)

print q.execute()

if args.select\_logtrunk:

selectLogtrunk()

if args.select\_logicflow:

selectLogicflow()

if args.select\_difflib:

selectDifflib(args.select\_difflib)

if args.update\_difflib:

updateDifflib(args.update\_difflib)

## 细节技巧

1. 尽量使用如下格式 allflow=[i.flowjson for i in all]

而不是

allflow=[]

for i in all:allflow.append(i.flowjson)

# logre 模块

## 单库操作模块

from \_sqlconfig import \*

from \_basic import \*

import MySQLdb,sys,warnings,re

from \_loggingset import \*

warnings.filterwarnings('ignore', category=MySQLdb.Warning)

class dbsinglehandle(object):

def \_\_init\_\_(self):

def \_update():

valuelen=len(changevalue)

if dbdatalen==valuelen:

pass

elif dbdatalen>valuelen:

cur.execute("delete from {0} where {1}={2} limit {3}".format(tablename,handle\_keyword,keyidlist[index],dbdatalen-valuelen))

elif dbdatalen<valuelen:

cur.execute("update {0} set {1}={2} where {1} in ( select t.{1} from ( select \* from {0} ORDER BY RAND() limit {3}) as t)".format(tablename,handle\_keyword,keyidlist[index],valuelen-dbdatalen))

#更新

print keyidlist[index]

for changeitem in changevalue:

try:

\_\_updatesql="update {0} set {1} where {2}={3}".format(tablename,changeitem,handle\_keyword,keyidlist[index])

print \_\_updatesql

cur.execute(\_\_updatesql)

except MySQLdb.Error,e:

if e[0]==1064:

print '请检查\_sqlconfig的testcases语句是否正确',e[1]

def \_replace():

for changeitem in changevalue:

cur.execute("replace into {0} set {1},{2}={3}".format(tablename,changeitem,handle\_keyword,keyidlist[index]))

conn=MySQLdb.connect(host=dbHost1,user=dbUser1,passwd=dbPasswd1,db=dbDb1,port=dbPort1,charset="utf8")

cur=conn.cursor()

if iscanModifydata:

casenum=len(testcases)

cur.execute('SELECT {1} FROM {0} ORDER BY RAND() LIMIT {2}'.format(keytablename,handle\_keyword,casenum))

rawkeyid=cur.fetchall()

keyidlist=[str(h) for i in rawkeyid for h in i]

print tuple(keyidlist)

if ispriorityCreatedata:

#replace

for index,onecase in enumerate(testcases):

for tablename,changevalue in onecase.items():

cur.execute("delete from {0} where {1}={2} limit 50".format(tablename,handle\_keyword,keyidlist[index]))

if isinstance(changevalue, (tuple, list, set)):

\_replace()

elif isinstance(changevalue, dict):

changevalue=changevalue.values()

\_replace()

elif isinstance(changevalue, basestring):

changevalue=[changevalue]

\_replace()

else:

#update

for index,onecase in enumerate(testcases):

for tablename,changevalue in onecase.items():

cur.execute("select count(\*) from {0} where {1}={2}".format(tablename,handle\_keyword,keyidlist[index]))

dbdatalen=cur.fetchone()[0]

#需要查看现存的记录数 是否与testcase想要的记录数相同

if isinstance(changevalue, (tuple, list, set)):

\_update()

elif isinstance(changevalue, dict):

changevalue=changevalue.values()

\_update()

elif isinstance(changevalue, basestring):

changevalue=[changevalue]

\_update()

else:

#select

keyidselect=[]

for onecase in testcases:

selectsqllist=[]

for tablename,changevalue in onecase.items():

if isinstance(changevalue, (tuple, list, set)):

pass

elif isinstance(changevalue, dict):

changevalue=changevalue.values()

pass

elif isinstance(changevalue, basestring):

changevalue=[changevalue]

pass

for changeitem in changevalue:

selectvalue=re.sub(',',' and ',changeitem)

subsql="select {0} from {1} where {2}".format(handle\_keyword,tablename,selectvalue)

selectsqllist.append(subsql)

for index,item in enumerate(selectsqllist):

if index==0:

selectsql=item

else:

selectsql=selectsql+' and {0} in ({1}) '.format(handle\_keyword,item)

selectsql=re.sub(r'=null',' is null',selectsql)

cur.execute(selectsql)

subkeyid=cur.fetchone()[0]

keyidselect.append(str(subkeyid))

print selectsql

if not subkeyid:

print '数据库中不存在符合条件的数据。可修改\_sqlconfig文件的iscanModifydata为True,再次运行。',onecase

else:

print subkeyid

print keyidselect #通过select取到的关键词list

conn.commit()

conn.close()

cur.close()

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

dbsinglehandle()

## 同步模块

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

import MySQLdb,sys

import warnings

from \_basic import \*

from \_sqlconfig import \*

from \_loggingset import \*

warnings.filterwarnings(*'ignore'*, category=MySQLdb.Warning)

class **dbsync**(object):

def **\_\_init\_\_**(*self*,keyid):

conn=MySQLdb.connect(host=dbHost1,user=dbUser1,passwd=dbPasswd1,db=dbDb1,port=dbPort1,charset=*"utf8"*)

cur=conn.cursor()

for tablename in sync\_tables:

cur.execute(*"select \* from {0} where {1}={2}"*.format(tablename,sync\_keyword,keyid))

globals()[tablename+*'select'*]=cur.fetchall()

cur.execute(*"describe {0}"*.format(tablename))

globals()[tablename+*'structure'*]=cur.fetchall()

globals()[tablename+*'columnlist'*]=[colunmname[0] for colunmname in globals()[tablename+*'structure'*]]

globals()[tablename+*'columnformat'*]=*','*.join(globals()[tablename+*'columnlist'*])

globals()[tablename+*'valueformat'*]=(*'%s,'*\*len(globals()[tablename+*'columnlist'*])).strip(*','*)

conn.commit()

conn.close()

cur.close()

conn=MySQLdb.connect(host=dbHost2,user=dbUser2,passwd=dbPasswd2,db=dbDb2,port=dbPort2,charset=*"utf8"*)

cur=conn.cursor()

for tablename in sync\_tables:

cur.execute(*"delete from {0} where {1}={2} limit 50"*.format(tablename,sync\_keyword,keyid))

globals()[tablename+*'replace'*]=*"replace into {0}({1}) values({2})"*.format(tablename,globals()[tablename+*'columnformat'*],globals()[tablename+*'valueformat'*])

print globals()[tablename+*'replace'*]

print globals()[tablename+*'select'*]

cur.executemany(globals()[tablename+*'replace'*],globals()[tablename+*'select'*])

conn.commit()

conn.close()

cur.close()

#是数据初始化 使变得可抓取运行

def **setdatarun**(keyidlist):

if isinstance(keyidlist,(tuple, list, set)):

keyidparam=[[item] for item in keyidlist]

elif isinstance(keyidlist,(int,long,basestring)):

keyidparam=[[keyidlist]]

else:

return

conn=MySQLdb.connect(host=dbHost1,user=dbUser1,passwd=dbPasswd1,db=dbDb1,port=dbPort1,charset=*"utf8"*)

cur=conn.cursor()

cur.executemany(setdatarunsql,keyidparam)

conn.commit()

conn.close()

cur.close()

print dbHost1,*'更新成功'*

conn=MySQLdb.connect(host=dbHost2,user=dbUser2,passwd=dbPasswd2,db=dbDb2,port=dbPort2,charset=*"utf8"*)

cur=conn.cursor()

cur.executemany(setdatarunsql,keyidparam)

conn.commit()

conn.close()

cur.close()

print dbHost2,*'更新成功'*

if \_\_name\_\_==*"\_\_main\_\_"*:

keyidlist=[*'42746977532'*, *'42747063233'*, *'42747004832'*]

for i in keyidlist:

dbsync(i)

## 寻找数据库表中各字段频率最高的值并保存

#选取出各个选项的最多频率的值

if CreatedataUsemaxFrequency:

if not os.path.isdir('log'):os.mkdir('log')

#组建该数据库独有的表名

filename='log/'

for tablename in sync\_tables:

filename=filename+str(tablename)

filename=filename+'UsemaxFrequency.pkl'

try:

inputfile = open(filename, 'rb')

pickledict = pickle.load(inputfile)

logger.info('找到之前保存的各字段最高频率的值。如下：{0}'.format(str(pickledict)))

inputfile.close()

except:

logger.info('程序没有发现有保存的各字段最高频率的值的文件。开始计算。')

output = open(filename, 'wb')

pickledict={}

for tablename in sync\_tables:

defaultvaluedict={}

cur.execute("describe {0}".format(tablename))

structure=cur.fetchall()

columnlist=[]

for colunmname in structure:

if colunmname[0].lower()==handle\_keyword.lower():

continue

if colunmname[5]==u'auto\_increment':

continue

columnlist.append(colunmname[0])

logger.info('获取到表{0}除了handle\_keyword的各字段值.如下：{1}'.format(tablename,str(columnlist)))

for columnname in columnlist:

cur.execute('SELECT {1} FROM {0} where {2} in (select t.{2} from (select \* from {0} limit 100)as t) GROUP BY {1} ORDER BY count(\*) DESC LIMIT 1'.format(tablename,columnname ,handle\_keyword))

maxfreqword=cur.fetchone()[0]

defaultvaluedict.setdefault(columnname,maxfreqword)

defaultstring=''

for key,item in defaultvaluedict.items():

if item==None: #区别对待None 设置None,要对应为null

defaultstring=defaultstring+"{0}=null,".format(key)

else:

defaultstring=defaultstring+"{0}='{1}',".format(key,item)

defaultstring=defaultstring.strip(',')

pickledict.setdefault(tablename,defaultstring)

pickle.dump(pickledict, output)

logger.info('每个字段频率最高的值保存到文件中成功，以便下次使用。如下：{0}'.format(str(pickledict)))

output.close()

for tablename,defaultstr in pickledict.items():

globals()[tablename+'defaultstring']=defaultstr

## 细节及bug解决

1. 有两条以上changeitem，如何每次只更新一个记录，并且不重复？

必须有limit 确保每次只更新1次。 必须把之前所有的已更新过的条件设为否定条件放在where中进行排除掉

lastupdateconverse=''

allupdateconverse=''

for i,changeitem in enumerate(changevalue):

if not i:

crupdatesql="update {0} set {1} where {2}={3} limit 1".format(tablename,changeitem,handle\_keyword,keyidlist[index])

else:

crupdatesql="update {0} set {1} where {2}={3} and {4} limit 1".format(tablename,changeitem,handle\_keyword,keyidlist[index],allupdateconverse)

lastupdateconverse=re.sub('=','!=',changeitem)

lastupdateconverse=re.sub(',',' or ',lastupdateconverse)

if allupdateconverse:

allupdateconverse=allupdateconverse+' and ({0})'.format(str(lastupdateconverse))

else:

allupdateconverse=allupdateconverse+' ({0}) '.format(str(lastupdateconverse))

# linux

## linux命令

1. yum list glibc\* 显示所有已经安装和可以安装的以glibc字符串为开头的程序包
2. java -version 查看java的版本 java version "1.6.0\_25"

javac -version javac 1.6.0\_25

1. javac命令与java命令区别

javac 可以将java源文件编译为class字节码文件 如 javac HelloWorld.java

运行javac命令后，如果成功编译没有错误的话，会出现一个HelloWorld.class的文件。

java 可以运行class字节码文件 如 java HelloWorld 注意java命令后面不要加.class

1. -R : 对目前目录下的所有文件与子目录进行相同的权限变更 （常用） chmod -R +x ./bin
2. chmod 777 gisy.sh r=4,w=2,x=1。777就是rwxrwxrwx，意思是该登录用户（可以用命令id查看）、他所在的组和其他人都有最高权限。
3. ./指的是当前文件夹 ../指上一文件夹
4. yum apt-get区别

yum可以用于运作rpm包。rpm包主要应用在RedHat系列包括 Fedora等发行版的Linux系统上

apt-get可以用于运作deb包。deb包主要应用于Debian系列包括现在比较流行的Ubuntu等发行版上。

1. 不同的Linux之间copy文件常用有3种方法：

第一种就是ftp，也就是其中一台Linux安装ftp Server，这样可以另外一台使用ftp的client程序来进行文件的copy。

第二种方法就是采用samba服务，类似Windows文件copy 的方式来操作，比较简洁方便。

第三种就是利用scp命令来进行文件复制。

scp是有Security的文件copy，基于ssh登录。操作起来比较方便，比如要把当前一个文件copy到远程另外一台主机上，可以如下命令。

scp /home/daisy/full.tar.gz root@172.19.2.75:/home/root

然后会提示你输入另外那台172.19.2.75主机的root用户的登录密码，接着就开始copy了。

如果想反过来操作，把文件从远程主机copy到当前系统，也很简单。

1. 443端口即网页浏览端口，主要是用于HTTPS服务，是提供加密和通过安全端口传输的另一种HTTP。
2. uname －a （Linux查看版本当前操作系统内核信息）

Linux localhost.localdomain 2.6.32-358.el6.x86\_64 #1 SMP Fri Feb 22 00:31:26 UTC 2013 x86\_64 x86\_64 x86\_64 GNU/Linux

1. cat /proc/version （Linux查看当前操作系统版本信息）

Linux version 2.6.32-358.el6.x86\_64 (mockbuild@c6b8.bsys.dev.centos.org) (gcc version 4.4.7 20120313 (Red Hat 4.4.7-3) (GCC) ) #1 SMP Fri Feb 22 00:31:26 UTC 2013

1. cat /proc/cpuinfo （Linux查看cpu相关信息，包括型号、主频、内核信息等）
2. cat /etc/issue 或cat /etc/redhat-release（Linux查看版本当前操作系统发行版信息）

CentOS release 6.4 (Final)

1. 修改/etc/hosts之后正常情况应该是保存之后立即生效的

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4 这一条建议不要修改，因为很多应用程序会用到这个，比如sendmail，修改之后这些程序可能就无法正常运行。

1. 查看主机名 hostname
2. 通过关闭 UseDNS和GSSAPIAuthentication选项加速 SSH登录

编辑配置文件 /etc/ssh/sshd\_config

找到 UseDNS选项，如果没有注释，将其注释 #UseDNS yes

添加 UseDNS no

找到 GSSAPIAuthentication选项，如果没有注释，将其注释 #GSSAPIAuthentication yes

添加 GSSAPIAuthentication no

保存配置文件

重启 OpenSSH服务器 /etc/init.d/sshd restart

1. touch fileA 如果fileA存在，使用touch指令可更改这个文件或目录的日期时间为当前系统时间，包括存取时间和更改时间；如果fileA不存在，touch指令会在当前目录下新建一个空白文件fileA。
2. -bash scp command not found的解决方法

首先在一台可以运行scp命令的服务器上找到scp所在的包：

# which scp

/usr/bin/scp

# rpm -qf /usr/bin/scp

openssh-clients-4.3p2-72.el5

这么看来scp所在的包是openssh-clients了。

接着在无法运行scp命令的机器上安装：

# yum install openssh-clients

1. rpm命令列表
   1. rpm -qf /usr/bin/scp 查询一个已经安装的文件对应软件安装包的名称 文件名所在的绝对路径要指出
   2. rpm -ql openssh-clients-5.3p1-84.1.el6.x86\_64 查询已安装软件包都安装到何处
   3. rpm -qc openssh-clients-5.3p1-84.1.el6.x86\_64 查看已安装的软件包的配置文件
   4. rpm -q openssh-clients 询问是否安装了openssh-clients软件包
   5. rpm -qa 查看系统中所有已经安装的包，要加 -a 参数
   6. rpm -qR openssh-clients 查看已安装软件所依赖的软件包及文件
2. 显示该机ip hostname -i
3. 永久修改主机名

通过修改配置文件/etc/sysconfig/network，修改的内容如下

NETWORKING=yes

HOSTNAME=localhost.localdomain

通过修改此文件的内容，它能够实现永久修改linux的主机名，但是它不会立即生效，即有可能不在当前运行时间生效，即在从下次重启后才开始生效，至少是 不在当前session生效，需要用户退出以后才生效。通过修改此配置文件，再配合hostname命令，可实现立即永久修改linux的主机名

1. jps 查看当前java进程
2. linux系统修改系统时间与时区

linux系统时钟有两个，一个是硬件时钟，即BIOS时间，就是我们进行CMOS设置时看到的时间，另一个是系统时钟，是linux系统Kernel时间。当Linux启动时，系统Kernel会去读取硬件时钟的设置，然后系统时钟就会独立于硬件运作。

查看系统时间 date

将系统时间设置为14点20分50秒 date -s 14:20:50

将日期设置为2014年6月18日 date -s 06/18/14

查看系统硬件时钟bios时间 hwclock --show

系统时钟和硬件时钟同步：hwclock --systohc 即用系统时钟同步硬件时钟

查看系统当前时间戳 date "+%s"

查看现在 年月日 date +%Y%m%d

1. cat -n textfile1 > textfile2 把 textfile1 的档案内容加上行号后输入 textfile2 这个档案里

cat -b textfile1 textfile2 >> textfile3 把 textfile1 和 textfile2 的档案内容加上行号（空白行不加）之后将内容附加到 textfile3 里。

1. echo gisy 输入一行文本并显示在标准输出上 如界面最后输入文本 gisy
2. 解压 .tar tar xvf FileName.tar

.tar.gz 和 .tgz tar zxvf FileName.tar.gz

1. find命令

find -name april\* 在**当前目录**下查找以april开始的文件 不加路径默认只在当前目录下

find / -mmin -5 # 查找在系统中最后5分钟里修改过的文件

find /etc -name "ap\*" -o -name "may\*" 查找在/etc目录下 以ap或may开头的文件

find . [!] -type d -print 查找在当前目录中非目录或者目录文件 d指目录

find / -type f -name \*libpcre.so.\* 查找在根目录下 包含libpcre.so.名的普通文档 f指普通文档

1. 配置更改后使配置立即生效

kill -HUP pid

pid 是进程标识。如果想要更改配置而不需停止并重新启动服务，请使用该命令。在对配置文件作必要的更改后，发出该命令以动态更新服务配置。根据约定，当您发送一个挂起信号（信号 1 或 HUP）时，大多数服务器进程（所有常用的进程）都会进行复位操作并重新加载它们的配置文件。

1. 查看进程号 ps -ef | grep "nginx: master process" | grep -v "grep" | awk -F ' ' '{print $2}'

$0表示整个当前行 $1每行第一个字段 -F指定分隔符

awk是一个强大的文本分析工具，相对于grep的查找，sed的编辑，awk在其对数据分析并生成报告时，显得尤为强大。简单来说awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。

1. 新建文件夹

mkdir -m 777 test3 创建权限为777的目录

mkdir -p test2/test22 递归创建多个目录

## 防火墙 iptables 和 SELinux

防火墙开启，外部机器就无法访问该服务器。最好关闭，或者开启并且在iptables配置开启相关端口

1. service iptables status可以查看到iptables服务的当前状态。
2. 但是即使服务运行了,防火墙也不一定起作用,你还得看防火墙规则的设置 iptables -L
3. 启动和关闭防火墙的命令:

1) 重启后生效

开启： chkconfig iptables on

关闭： chkconfig iptables off

2) 即时生效，重启后失效

开启： service iptables start

关闭： service iptables stop

1. 在开启了防火墙时，做如下设置，开启相关端口， 修改/etc/sysconfig/iptables 文件，添加到默认的22端口这条规则下面：

-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT

1. 做DNS服务器，开启53端口；做FTP服务器，开启21端口；做邮件服务器，开启25,110端口；做web服务器，开启80端口；

为了能远程ssh登陆，开启22端口；做mysql服务器，开启3306端口

1. 允许所有已经建立的和相关的连接 -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

ESTABLISHED：已建立的链接状态。

RELATED：该封包为本机发出的封包有关。

1. 允许icmp包通过，也就是允许ping -A INPUT -p icmp -j ACCEPT
2. 允许loopback!(不然会导致DNS无法正常关闭) -A INPUT -i lo -j ACCEPT
3. /etc/init.d/iptables restart #最后重启防火墙使配置生效

SELinux是一种安全子系统。它能控制程序只能访问特定文件。

1. 关闭SELinux

永久方法 – 需要重启服务器

修改/etc/selinux/config文件中设置SELINUX=disabled ，然后重启服务器

临时方法 – 设置系统参数

使用命令 setenforce 0 可以临时关闭，但重启之后还是会变成原来的状态。

附：

setenforce 1 设置SELinux 成为enforcing模式

setenforce 0 设置SELinux 成为permissive模式

1. getenforce 这个命令可以查看到selinux的状态。 Disabled为关闭状态

## tomcat nginx

如何查看tomcat的版本

linux 下进入tomcat安装文件夹的bin目录下 执行./version.sh

windows下进入tomcat安装文件夹的bin目录下 执行./version.bat

如何查看nginx版本号

进入 /nginx/sbin

./nginx -V

查看文件所在---这个速度慢但准确 find -name libmysqlclient.so.18

./configure是源代码安装的第一步，主要的作用是对即将安装的软件进行配置，检查当前的环境是否满足要安装软件的依赖关系

export PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin

### 基础

1. JSP全名为Java Server Pages，中文名叫java服务器页面，其根本是一个简化的Servlet设计
2. Servlet（Server Applet），全称Java Servlet，未有中文译文。是用Java编写的服务器端程序。其主要功能在于交互式地浏览和修改数据，生成动态Web内容。
3. 当配置正确时，Apache 为HTML页面服务，而Tomcat 实际上运行JSP 页面和Servlet。
4. nginx和apache能共存。不需要联系的话用两个不同的端口。

### 安装nginx

1. 编译安装nginx

./configure --with-http\_stub\_status\_module --with-http\_ssl\_module #启动server状态页和https模块

–with-http\_stub\_status\_module 可以用来启用 Nginx 的 NginxStatus 功能，以监控 Nginx 的运行状态。

1. 执行完后会提示一个错误，说缺少PCRE library 这个是HTTP Rewrite 模块，也即是url静态化的包

安装pcre模块 #./configure #make #make install

1. 安装pcre成功后，继续安装nginx ./configure make && make install
2. nginx安装成功后的安装目录为/usr/local/nginx

在conf文件夹中新建proxy.conf，用于配置一些代理参数，内容如下：

#!nginx (-)

# proxy.conf

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr; #获取真实ip

#proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for; #获取代理者的真实ip

client\_max\_body\_size 10m;

client\_body\_buffer\_size 128k;

proxy\_connect\_timeout 90;

proxy\_send\_timeout 90;

proxy\_read\_timeout 90;

proxy\_buffer\_size 4k;

proxy\_buffers 4 32k;

proxy\_busy\_buffers\_size 64k;

proxy\_temp\_file\_write\_size 64k;

1. 检查配置文件是否正确 /usr/local/nginx/sbin/nginx -t

如果报错error while loading shared libraries: libpcre.so.1: cannot open shared object file

则执行 find / -type f -name \*libpcre.so.\*

返回 /usr/local/lib/libpcre.so.1.0.1

建立软链接 ln -s /usr/local/lib/libpcre.so.1.0.1 /lib64/libpcre.so.1

1. 启动nginx命令 /usr/local/nginx/sbin/nginx
2. 查看Nginx主进程号 ps -ef | grep "nginx: master process" | grep -v "grep" | awk -F ' ' '{print $2}'
3. 停止nginx的命令 /usr/local/nginx/sbin/nginx -s stop
4. nginx启动好后启动tomcat
5. nginx 更改端口号 修改 nginx.conf 文件实现。在 Linux 上该文件的路径为 /usr/local/nginx/conf/nginx.conf

server {

listen 80;

server\_name localhost;

……

}

80改成其他端口号 比如88

1. 不停止Nginx服务的情况下使变更的Nginx配置立即生效：

输入以下命令查看Nginx主进程号：

ps -ef | grep "nginx: master process" | grep -v "grep" | awk -F ' ' '{print $2}'

屏幕显示的即为Nginx主进程号，例如：

　　6302

这时，执行以下命令即可使修改过的Nginx配置文件生效： kill -HUP 6302

### 安装tomcat

1. 配置环境变量：编辑/etc下的profile文件，加上如下内容：

CATALINA\_HOME="/usr/local/tomcat"

export CATALINA\_HOME

1. 启动tomcat 进入 /usr/local/tomcat/bin ./catalina.sh start
2. 输入http://localhost:8080，如果看到猫的页面即tomcat和jdk安装成功
3. webapps文件夹主要用于web应用程序部署，比如你可以把你的应用程序包，如war文件拷到该目录下，容器会自动部署。

conf文件夹下主要是放置tomcat的服务器的相关配置文件

1. 修改网页存放目录 比如修改为/home/www为网站存放目录

设置conf文件夹下的server.xml文件，在Host name="localhost"处将appBase=的指向路径改为/home/www/web

1. 修改tomcat的监听端口

server.xml中

<Server port="8005" shutdown="SHUTDOWN">【停止tomcat时的端口】

<Connector port="8081" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" /> 【tomcat默认的监听端口是8080，现在改成8081】

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />【apache+tomcat模式时访问tomcat的端口】

### 安装auto-add-tomcat

1. 在/home目录下新建admin文件夹， mkdir -m 777 admin
2. 将auto-add-tomcat文件夹移到 admin目录下
3. 创建域名

cd /home/admin/auto-add-tomcat

./ add-nginx+tomcat.sh gisy.jd.com

## linux安装环境

### 安装模块包

linux安装模块包 本人统一暂时存放在/usr/local/etc

1. 手动下载模块安装方法： cd paramiko-1.7.7.1/ python setup.py build && python setup.py install
2. 安装whl格式时： pip install mock-1.3.0-py2.py3-none-any.whl
3. 当安装包没有setup.py，使用此命令安装chmod +x ./configure
   1. ./configure --prefix= /usr/local/python2.7/lib/python2.7/site-packages;make && make install
4. 安装egg文件使用easy\_install 不能使用pip. easy\_install setuptools-0.6c11-py2.7.egg
5. 安装rpm包 rpm -i 需要安装的包文件名
6. 重新编译python 在python的安装包中执行 ，先进入python安装包路径。

【./configure --prefix=/usr/local/python2.7;make && make install】不能缺少第一个命令，因为这样重新编译的python就无法存放到/usr/local/python2.7，也就无法完全覆盖之前python

1. 查看安装的模块需要依赖哪些其他模块，打开setup.py,查看install\_requires或者requires选项
2. 使用pip install flask 安装不了模块，报错：

after connection broken by 'NewConnectionError('<pip.\_vendor.requests.packages.urllib3.connection.VerifiedHTTPSConnection object

原因是服务器连接不上外网。请ping [www.baidu.com](http://www.baidu.com) 查看能否ping通。如果不通，只能本地下载模块包，再上传到服务器上。

1. 安装libxml2时报错的处理

查看python-devel包是否安装

[root@localhost libxml2-2.9.1]# rpm -q python-devel

package python-devel is not installed

安装python-devel软件包

[root@localhost libxml2-2.9.1]# yum -y install python-devel

1. 安装lxml时gcc -pthread -fno-strict-aliasing -g -O2 -DNDEBUG 会卡很久，这是正常现象。请耐心等待即可安装完成。、
2. 报错：error: command 'gcc' failed with exit status 1

解决方案：yum install python-devel mysql-devel zlib-devel openssl-devel

yum -y install mysql-devel libxml2 libxml2-dev libxslt\* zlib gcc openssl

1. 适合py27的setuptools模块名setuptools-1.0 ，更高档不适用，会报错ImportError: No module named \_ctypes 。而且版本必须"setuptools>=1.0"
2. Build the lexing/parsing tables 代表pycparser安装完成
3. 安装scrapy中难点：各模块依赖关系

service\_identity --》cryptography pyOpenSSL

pyOpenSSL--》cryptography 版本必须是pyOpenSSL-0.15.1

cryptography --》cffi "cffi>=1.4.1" ipaddress enum34

1. 安装libffi

rpm -qa | grep libffi

libffi-3.0.5-3.2.el6.x86\_64

sudo yum install -y libffi libffi-devel

1. 安装cffi模块报错：

No package 'libffi' found

Package libffi was not found in the pkg-config search path.

解决方案：

在该网站下载，因为：yum install -y libffi libffi-devel 只能找到libffi ，下载不到libffi-devel <http://rpm.pbone.net/index.php3/stat/4/idpl/27825600/dir/centos_6/com/libffi-devel-3.0.5-3.2.el6.x86_64.rpm.html>

libffi-devel-3.0.5-3.2.el6.x86\_64.rpm

libffi-3.0.5-3.2.el6.src.rpm

### 运行问题

1. 爬虫过程中报错 scrapy exceptions.ImportError: No module named \_sqlite3

可以使用如下命令重新安装以上三个包

yum groupinstall "Development tools"

yum install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel

安装完成后，重新编译安装python即可，过程不再赘述。

安装完成后，重新执行scrapy，执行成功！

1. 需要在linux服务器配上hosts才能登陆网页 192.168.200.88 scm-wiki.jd.com

### 其他

1. 防火墙

service iptables status可以查看到iptables服务的当前状态。

但是即使服务运行了,防火墙也不一定起作用,你还得看防火墙规则的设置 iptables -L

在此说一下关于启动和关闭防火墙的命令:

1) 重启后生效

开启： chkconfig iptables on

关闭： chkconfig iptables off

2) 即时生效，重启后失效

开启： service iptables start

关闭： service iptables stop

在开启了防火墙时，做如下设置，开启相关端口，修改 /etc/sysconfig/iptables 文件，添加以下内容：

-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT #允许80端口通过防火墙

-A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT #允许3306端口通过防火墙

1. 关闭SELinux

vim /etc/selinux/config # 改为 SELINUX=disabled

# 保存退出，重启服务器

init 6

禁用SeLinux

#永久禁用，需要重启生效。

sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config

# 临时禁用，不需要重启

setenforce 0

1. hostname

hostname 没有选项，显示主机名字

hostname –d 显示机器所属域名

hostname –f 显示完整的主机名和域名

hostname –i 显示当前机器的ip地址

1. ping

ping 将数据包发向用户指定地址。当包被接收。目标机器发送返回数据包. ping 主要有两个作用

1. 用来确认网络连接是畅通的。

2. 用来查看连接的速度信息。

如果你 ping www.yahoo.com 它将返回它的ip地址 。你可以通过 ctrl+C 来停止命令。