**Python课堂笔记**

# 学前准备：

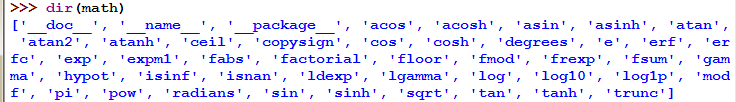
目标：能够独立写框架、独立做接口测试、独立做性能测试、独立带项目。

警惕：不低谷自己的能力、不高估自己的毅力！

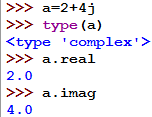
注意：如果有大量的缩进的代码，建议使用文件的模式来执行； 在没有大量缩进代码的情况下，使用交互模式。

## 常用函数：

dir()查看模块中的方法；help()查看帮助文档；random内置模块【随机数】；randrange//random模块中的方法，返回一个随机数；integer；

复数：实部a.real；虚部a.imag；



执行py文件输出中文显示乱码：a.保存文件编码格式为utf-8；b.中文字符前加u； c.文件开头声明#encoding=utf-8

math.fabs()//取绝对值



## 编码：

以key-value的方式对中文字符进行编码gbk、gb2312；utf-8、utf-16编码格式主要用于存储文件；uinicoce为每一种语言每一个字符设定了统一并且唯一的二进制编码。

## 执行pip命令的两个问题：

1. pip执行报错：

在C:\Python27\Lib\site-packages路径下新建sitecustomize.py文件，输入以下内容保存：

import sys

reload(sys)

sys.setdefaultencoding('utf-8')

2. pip install xxx安装python包报错：UnicodeDecodeErro:'utf-8' codec cant't decode byte 0xb6 in position 7:invalid start byte

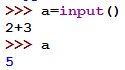
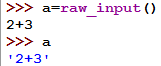
将问题1中的 sys.setdefaultencoding('utf-8')改为 sys.setdefaultencoding('gbk')保存

## 下载安装python模块

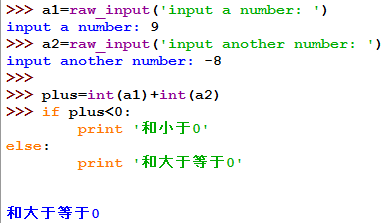
python setup.py install ；使用pip安装pip install xxx；卸载模块：pip uninstall xxx; pip list查看已安装包; 指定安装版本：pip install xlrd==0.9.3

单元测试框架nose；selenium

**raw\_input()、input()：**raw\_input()输入源字符，input()会将输入的内容进行运算：

**练习：**



# 其他

## Coockie与session

1. cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。Session是基于cookie的（session机制可能需要借助于cookie机制来达到保存标识的目的），如果将cookie删除session就不能用了。
2. cookie的内容主要包括：名字，值，过期时间，路径和域。路径与域一起构成cookie的作用范围。
3. **若不声明过期时间**，则表示这个cookie的生命周期为浏览器会话期间，这种叫做会话cookie，一般不保存在硬盘中，而是保存在内存中。**设定了过期时间的cookie**保存在硬盘上，关闭浏览器后重新打开依然生效，超过过期时间后失效。
4. session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。

# Python基础

## While循环

**While--else 当没有break时，循环完成后执行else语句。**

#encoding=utf-8

# print u"中国"

'''输入得分：如果>=90,打印优'''

i=0

while i<3:

i+=1

score=int(raw\_input('请输入你的分数： '.decode('utf-8').encode('gbk')))

if score>=90 and score<=100:

print u'优'

elif score<90 and score>=80:

print u'良'

elif score<80 and score>=70:

print u'中'

elif score<70 and score>60:

print u'及格'

elif score==60:

print u'跳出循环'

break

elif score<0 or score>100:

print u'输入不合法'

else:

print u'不及格'

#可以用try-except语句来确保输入的为数字。

s=raw\_input('input a number: ')

try:

s=float(s)

except Exception ,e:

print 'not a number'

## For循环

#循环输入3个数字，输出3个数字的和

summ=0

for i in range(3):

n=raw\_input('input a number: ')

try:

n=float(n)

summ=summ+n

print summ

except Exception,e:

print e

for i in range(1,11):

print i

for i in range(2,11):

if i%2==1:

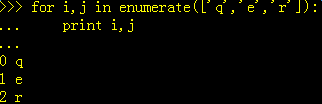
print i

## enumerate（枚举类型）

可以自定义索引开始的位置：enumerat([4,5,6],2)

for i,j in enumerate([‘s’,’d’,’e’]):

print i,j



## Reversed

list\_example = [i for i in range(5)]

iter\_example = (i for i in range(5)) # 迭代器

set\_example = {i for i in range(5)} # 集合

>>> **reversed**('hello')

<reversed object at 0x0000000001E26358>返回可迭代对象

*>>> for i in reversed('hellow'):*

*... print i,*

*...*

*w o l l e h*

*flask 简易代码：*

*from flask import Flask*

*app=Flask(\_\_name\_\_)*

*@app.route('/')*

*def hello():*

*return "hello world!"*

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

*app.run()*

**还可以通过实现类里的\_\_reversed\_\_方法，将类反向迭代：**

*class Countdown(object):*

*def \_\_init\_\_(self,start):*

*self.start=start*

*#正向迭代*

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

*n=self.start*

*while n>0:*

*yield n*

*n-=1*

*#反向迭代*

*def \_\_reversed\_\_(self):*

*n=1*

*while n<=self.start:*

*yield n*

*n+=1*

*for i in reversed(Countdown(4)):*

*print i,*

*print #1 2 3 4*

*for i in Countdown(4):*

*print i, #4 3 2 1*

# 功能测试

## 时间管理：

1. 制定中长期目标，分解到每年、每月、没周、每天
2. 坚持和落地
3. 排好优先级
4. 时间管理书籍-《暗时间》
5. 每天两页书、50行代码，每天发送学习笔记（至少一行代码；微信群惩罚监督）

## 如何学习：

找导师，指明学习方向、答疑解惑

看书—找书（推荐、书评），做读书笔记

知识积累：做笔记

目标1000页笔记

## 面试

### 作为测试人员自己的优势

1. 定位bug能力，对产品的理解和把握
2. 能够设计有效测试用例
3. 经验（见得多了以后，基本上会出问题的地方看一眼就知道了）、思路、管理能力
4. 碰到问题主动的去思考并尽量自己分析解决
5. 对产品有更深的理解，熟悉整个软件流程，生命周期。
6. 业务比以前熟练，编写测试用例熟练，还有就是测试的方法多了一些。
7. 上手快，沟通协调能力提高，能提出自己的看法
8. 掌握了技术：数据库、代码
9. 有了自己的一个测试方法
10. 测试的效率高了、根据自己的经验给产品或者开发提出修改意见、可以发现一些需求的不合理
11. 对整个项目测试有提前思考；
12. 测试流程，测试思路的积累
13. 拿到一个测试任务心里有底，知道大概需要多少时间就能搞定
14. 测试流程的熟练度、沟通技巧提升、测试用例设计思路、行业知识的积累
15. 对系统和测试对象更了解
16. 知道出现问题大概的原因和找谁解决
17. 知道容易出现问题的地方

### 印象深刻的bug：

要说出bug的严重程度（选择叫严重的bug）、分析bug的原因、总结如何避免下次出现类似的问题（制定测试规范）、如何发现的bug（可以阐述发现的bug的插曲）

### 开发不改的bug：

与领导沟通，是否同意，最后邮件说明（什么原因、与谁确认可以接收怎么的结果）

### 如何提升质量：

[公众号文章](http://mp.weixin.qq.com/s?__biz=MjM5OTI2MTQ3OA==&mid=2652177995&idx=1&sn=8316683786749c06f0be60e3c3209174&chksm=bcdf82028ba80b14b9a07e83cc743042e22d6da616214be2ad6443ad07041570729a128cf47b&mpshare=1&scene=1&srcid=1015oADYqlpDEUjCHpiyrSHz&from=groupmessage&isappinstalled=0#wechat_redirect)

### 没有测试用例如何测试：

探索式测试

### 如何开展测试

了解需求、拆分模块、测试方法、设计测试用例

### 没有需求，如何测试

口头沟通询问、书面整理需求、参考行业标准

### 专业术语

Smtp协议、https、md5、ajax、memcache、base64编码、jni

[Testman1980@126.com](mailto:Testman1980@126.com) wulaoshi1978

### 电梯测试

系统的分类：界面测试、功能测试、兼容测试、易用性、可靠性、可维护性、稳定性、负载/压力测试、竞品测试、配置测试、文档测试、算法测试（业务逻辑）

## 测试框架

### UI测试

1. 界面是否美观
2. 元素的大小
3. 界面元素的对齐方式是否统一
4. 字体属性、大小是否统一
5. 界面的链接及触发动作
6. 元素内容检查，显示正确、易懂、友好
7. 对所有输入框的判断测试

### 链接测试

1. 死链（404、403、503状态；过多的“死链接”会产生许多多余代码，会影响到网站的加载速度，和每次提示“404页面”一样，网站过长的加载速度，会影响一个用户的体验程度）
2. 链接的打开方式（在当前页打开，或者在新标签页打开）

### 功能测试

1. 使用所有默认值进行测试
2. 遍历产品文档、帮助文档中描述的所有内容
3. 所有界面的出现是否符合逻辑
4. 输入的判断
5. 根据需求文档的流程图遍历所有流程图路径
6. 各种控件的测试
7. 异常测试
8. 敏感词检查
9. 代码覆盖（if、elif、switch各语句的逻辑点的遍历）

### 易用性

1. 界面控件可通过tab键遍历，顺序合理
2. 操作步骤、操作习惯
3. 界面布局合理，功能易于查找、使用
4. 有足够提示信息，提示信息准确

### 稳定性

7\*24小时运行，验证是否存在内存泄露、crash等严重问题

### 容错测试

1. 输入系统不允许的数据，检查结果
2. 移除依赖模块或子系统，验证对当前系统的影响
3. 配置文件删除或配置错误
4. 数据库注入错误数据

### 性能测试

1. 前端（客户端）、后端（服务端）性能测试
2. 并发测试
3. 压力测试
4. 容灾备份恢复
5. 数据的预处理

### 可维护性

1. 易于升级
2. 安装、卸载
3. 回滚至上一版本
4. 易于备份数据

### 容量测试

1. 数据库存储容量
2. 文件存储容量
3. 本地cache文件存储
4. 网络cdn存储
5. 网络带宽容量

### 基准测试

与行业领先产品的功能对比测试

### 安全和访问权限测试

1. 验证所有权限是否生效
2. 检查需要登陆后才可见的内容
3. url是否包含敏感信息
4. 对权限的控制
5. Cookie等敏感信息是否加密处理

### 故障转移恢复测试

负载均衡、分担流量

### 配置测试

1. 不同的操作系统
2. 不同硬件配置
3. 不同分辨率
4. 不同网络
5. 不同浏览器
6. 不同的放火墙，放病毒软件

### 发布测试

说明书、帮助文档、API文档、宣传材料

### 文档审核

需求文档、概要设计、帮助文档、详细设计、说明书

### 多语言测试

各国语言（中英文、日韩）、在a语言下运行b语言软件，界面检查

### 参数输入考虑

1. 参数值1个、2个、多个、null、输入前后包含空格
2. 数字类型：正数、负数、整数、小数、0、0.0、+0.0、-0.0、对数、复数、指数、科学计数法
3. 文字类型：空格、半角、全角、中英文字符、系统保留字、编程保留字、数据库保留字

### 文字测试

1. 文字内容与需求文档是否一致
2. 多语言环境下的显示
3. 语言风格、文字排版、错误提示

### 图片测试

图片的格式、大小、显示尺寸、属性、路径、图片链接、对界面其他元素的影响

### 文件

文件类型、大小、超大文件、超小文件、只读文件、被占用状态、无权限

文件不存在、被重命名、磁盘已满、文件名称（超长文本、特殊字符）、网络文件

### 文件目录

目录名称、目录下的文件数量、层级目录、网络文件夹、隐藏文件、名称包含特殊字符

### 网络测试

流量、不同运营商

### 竞品测试

同类产品、是否冲突

### 接口测试

正确性、容错、压力、稳定性

### 算法

逻辑处理

### 兼容测试

系统版本、分辨率、系统配置、浏览器、网络、防火墙

### 单元测试

### 随机测试

## 如何设计测试用例（方法）

### 等价类划分

输入等价、有效等价、无效等价

### 边界值分析

取值范围，边界内、外、中

### 因果图：

### 正交实验设计：

### 功能图分析（白盒测试）

代码覆盖率，语句覆盖（50%-70%）、判定覆盖、条件覆盖、条件/组合覆盖、路径覆盖

### 错误推测

经验

### 需求文档转化

需求文档、示意图、流程图转化用例；需求评审、用例评审、架构评审

### 随机测试：

### 对象属性分析

## Bug预防

### Web测试常见问题

分辨率、浏览器兼容、link问题（当前页面打开、新窗口打开）、快捷键/焦点、前进/回退/刷新、语言、文字缩略折行、图片显示和链接、重复提交、输入判断、多个IE同时访问、安全问题（NoCache参数）、sql注入、Cookie中敏感信息加密（cookie与session的区别，session是基于Cookie的）、资源链接的释放（keepalive的设置）、系统上线log配置、程序实现与接口文档描述不一致、参数容错（太多的容错提示会影响性能）

### 思路

分析问题🡪找到具体问题—>解决问题—总结问题—建立预防方法🡪规范流程和制度🡪监督执行。

## 探索式测试

见文档

# 接口测试

**Httprequster**火狐插件、或者chrome的**postman**插件（需要在chrome应用商店下载安装）

示例：根据接口文档，http请求，查看返回结果（一般为json格式）

注册、登陆

{"code": "00", "userid": 39}

{"token": "175b9913b998ad0ee2ad1e8a3be0ab0b", "code": "00", "userid": 39, "login\_time": "2016-10-15 17:13:22"}

查询

{"data": [{"update\_time": null, "title": "JavaScript", "content": "this is js", "articleId": 59, "owner": 39, "posted\_on": "2016-10-15 17:35:15"}, {"update\_time": null, "title": "interface", "content": "python port test1", "articleId": 41, "owner": 39, "posted\_on": "2016-10-15 17:28:35"}], "code": "00", "userid": 39}

## 测试方法

## 作业

Bug预防

**书籍推荐：**google软件测试之道、http权威指南、暗时间、异类、探索式测试

MD5加密不可逆，生成16位 或32位加密数据

# 手机测试（android测试）

Android4.1新特性：动画帧速提高至60fps，无障碍操作（支持自动化测试uiautomator工具）

Android4.4全新的非java虚拟机运行环境ART（android runtime）与dalvik共存。

Android5.0全新界面MeterialDesign，支持64位处理器，全面转换为ART编译，性能提升。

Android6.0扁平化界面，性能提升、移动支付、完整的权限管理（多用户登陆）

1.0 instrumentation框架，接口测试（**缺点**：控件获取不到、必须源码程序）

2.2 robotium第三方工具，可捕获更多控件，可直接对apk文件测试。**缺点**：需要测试程序与被测程序有相同签名。

2.3 monkeyrunner工具，python语言，接口少、不利于完整的自动化测试。

4.1 input text/keyevent更新，支持滑动、拖拽更多操作

4.4 uiautomator工具，生成可执行的jar包、appium

## 功能测试

遵循需求，参加需求评审

### 启动和停止：

1. 首次启动：欢迎界面（5s内）、启动时间、信息拉取（从server端拉取文件，创建文件log、日志上报【用户行为统计】）
2. 二次启动：启动时间（与首次启动的对比）、启动入口（登陆后的信息展示）、
3. 退出应用：退出方式（手动、被动）、收拾残局（进程、service）、再次启动

### 程序功能模块：

1. 文本输入：正常输入、非正常输入、边界值、特殊字符
2. 时间触发：满足需求、点击非点击区域、检查debug后门（是否关闭用于测试、开发的调试信息）

### 权限安全：

1. 需要用户确认的权限、权限默认关闭
2. 联网权限被管家、安全类软件禁止（是否被黑名单）
3. 权限敏感，如通讯录

### 文件存储

1. App使用过程中临时文件的存储路径、命名方式
2. 涉及下载操作产生的文件存储方式（下载文件后找不到）
3. 存储的文件被锁、占用
4. 内置外置SD卡路径（相对路径、绝对路径）
5. App分别安装在SD卡、手机存储空间
6. 异常情况：磁盘空间不足、磁盘无权限（读写等）

### 网络与流量

1. 网络提示是否正确（明确提示），搜索提示（无结果、无网络），区分网络提示与错误提示。
2. 网络信号，弱网环境下应用表现。（在弱网环境下的不同处理方式是否合理）
3. 不同运行商网络：电信、联通、移动、2/3/4G
4. 网络中断、网络恢复场景的逻辑处理（类似弱网环境的处理方式，“在线视频”）
5. 首次启动的流量是否符合预期（流量值控制在预期内）
6. 统计、异常上报对流量的影响
7. App中图片大小、尺寸对流量的影响
8. 基于流量安全特殊业务，“仅wifi下，离线、视频自动播放”

### 接口容错

1. 请求网络层错误：http response返回非200的状态
2. 请求业务层错误：接口返回内容为空、超长、字段类型不匹配
3. 对客户端的容错提示

### 中断测试

1. 锁屏中断：两种锁屏方式，锁屏恢复后检查（对不同的app有不同的处理方式）
2. 前后台中断：程序界面，home键前后台切换
3. 加载中断：页面接口请求、界面框架加载时，home键、back、快速切换操作进行中断
4. 系统异常中断：关机、断电、来电

### 机型适配

1. 分辨率：UI结构、对话框、屏幕大小
2. OS版本适配：
3. CPU硬件配置：

### 系统配置

1. 进程管理：省电管理、后台进程驻留管理
2. 显示管理：字体大小、字体类型
3. 语言环境：中英文显示
4. 横竖屏配置：是否支持横竖屏自适应处理

### 升级、覆盖安装

1. 逐步升级（顺序升级，中间没有跳跃）：用户数据、设置、状态保留；
2. 跳级：隔开版本覆盖安装；
3. 降级：覆盖安装更低版本
4. 卸载安装：安装目录清理，SD卡存储数据不被清理
5. 省流量升级：增量升级
6. 给用户正确升级提示信息
7. 强制升级，不升级不可用

### 性能测试

1. 核心操作的性能指标：cpu/内存、响应时间、

### 兼容测试

1. 通用软件：输入法
2. 安全软件：如各种管家
3. 竞品软件、同类软件、是否冲突（腾讯管家、360管家）

## 专项测试

1. 内存、电量、流量、cpu、fps、弱网络、流畅度、加载速度、成功率、响应时间、稳定性、规划时间、内容推荐、竞品、支持格式
2. 可使用第三方工具GT测试，gt.qq.com

### Cpu

Dumpsys cpuinfo | grep <packagename>

### 内存测试

Dumpsys meminfo [packagename]

### 流量测试

1. 从文件中获取流量值：/proc/uid\_stat/<uid>/tcp\_snd

/proc/uid\_stat/<uid>/tcp\_rcv

1. 流量抓包：tcpdum –p –s 0 –vv –w filename

### FPS

Service cal SurfaceFlinger 1013

Adb shell dumpsys SurfaceFlinger –latency <window\_activity>

SurfaceFlinger 是一个系统服务，管理android帧缓存区。测试需要root权限。

### 电量

Cat sys/class/power\_supply/battery/uevent 查看文件前后的电量变化。（current\_now）

5.0之后的手机电量测试可直接查看应用的耗电情况：adb shell dumpsys batterystats>xx.txt

Adb shell dumpsys batterystats>com.package.name>xx.txt

### 稳定性测试

Monkey测试，随机操作，模拟用户按键、触摸、手势输入操作。

### 弱网络

网络质量差，数据传输丢包、延迟。

可使用fiddler工具模拟弱网环境。

Network delay monitor

1. debug.choreographer.skipwarning
2. aapt d badaging xxx.apk
3. adb shell am start –n com.xxx.xxx/.mainActivity
4. am force-stop packagename
5. am instrument
6. am monitor
7. pm list packages
8. pm clear packagename 清空data/目录下数据库、缓存内容
9. pm permission-groups
10. dumpsys input\_method 输入法
11. dumpsys window
12. dumpsys media.audio\_flinger
13. dumpsys
14. tcpdump –s 0 –vv –w filename
15. input text 内容
16. input tap 123 456 /点击坐标值
17. input swipe xxx /滑动坐标

fling swipe

按info排序，查看get、set请求右键tcp

M\_simulateMode设置值—开启simulateMode

## Bug根因分析

1. 界面适配：手机分辨率
2. 系统配置，
3. 交互适配

# Python特点

[www.seleniumhq.org](http://www.seleniumhq.org) 官网找教程Document

python框架：django—web开发框架；appium—自动化；webdriver；sikuli；

1. 程序=数据结构+算法
2. Css-Cascading Style Sheets
3. Javascript与java：前后台
4. <http://www.w3school.com.cn/> 教程学习！
5. 编码：gb2312、gbk、utf-8、unicode、Big5
6. ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）。
7. HyperText Markup Language--超级文本标记语言。

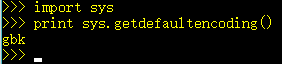
* 除法函数：divmod()

>>> isinstance('r',unicode)

False

>>> isinstance('d',str)

True

1. is判断两个对象的id是否相等。
2. 其他编码方式转换为unicode---解码decode；从unicode到其他编码方式---编码encode。如：gbk—decode--unicode—encode—utf-8；若没有指定编解码方式，默认使用系统的编码方式。
3. Gbk与gb2312编码方式基本重合。Gb2312简体中文字符集编码；Gbk是gb2312的扩展，兼容gb2312，也包括繁体和日文假名。
4. 中文decode会出错
5. 

import sys

print sys.getdefaultencoding()

reload(sys)

sys.setdefaultencoding(‘utf-8’)#改变系统默认编码方式为utf-8

1. round()
2. 交互模式调用repr()；print方法调用 str()；还原字符串 eval()

# 函数

## 参数

必备参数、可选参数、可变参数、

推荐：

1 函数内变量别和全局变量重名

2 使用全局变量，需要使用global 在函数内声明为全局变量

## Zill函数

zfill(int)在字符前不全0



32位和64位操作系统是指：CPU一次处理数据的能力是32位还是64位。

## 取反运算

**取反：**~9的计算步骤：

转二进制：0 1001

计算补码：0 1001

按位取反：1 0110

\_\_\_\_\_

转为原码：

按位取反：1 1001

末位加一：1 1010

符号位为1是负数，即-10

**Break只是跳出本次循环，**如下break只会跳出j的循环。

for i in range(10,51):

for j in range(1,6):

if i%10==j:

break

**Pass** 可退出多重循环，实际中使用return语句即可。

class getoutofloop(Exception): pass

try:

for i in range(5):

for j in range(5):

for k in range(5):

if i == j == k == 3:

raise getoutofloop()

else:

print i, '----', j, '----', k

except getoutofloop:

pass

**Return** 当函数执行到return语句时，函数执行完毕。

**递归算法3个原则：**

1、 一个递归算法必须有一个基本状态，达到这个基本状态时递归函数就停止递归。

2、 递归必须改变其状态并向基本状态移动。

3、 递归函数必须调用自身（调用的参数值不一样）。

# With…as…

基本思想是with所求值的对象必须有一个\_\_enter\_\_() \_\_exit\_\_()方法

紧跟with后面的语句被求值后，返回对象的\_\_enter\_\_()方法被调用，这个方法的返回值将被赋值给as后面的变量。当with后面的代码块全部被执行完之后，将调用前面返回对象的\_\_exit\_\_()方法。

等价于try……finally……

*#coding=utf-8*

*#with as 用法*

*import os*

*class Demo(object):*

*def \_\_enter\_\_(self):*

*print 'now in \_\_enter\_\_'*

*return 'Fun'*

*def \_\_exit\_\_(self,type,value,trace):*

*print 'now in \_\_exit\_\_'*

*with Demo() as demo:*

*print demo*

*class File(object):*

*def \_\_init\_\_(self,file,mode='r'):*

*self.file=file*

*self.mode=mode*

*def \_\_enter\_\_(self):*

*self.f=open(self.file,self.mode)*

*return self.f*

*def \_\_exit\_\_(self,type,value,trace):*

*self.f.close()*

*return Ture*

*# 返回 True 代表不抛出错误*

*# 否则错误会被 with 语句抛出*

*with File(os.path.join(os.getcwd(),'tmp.py')) as f:*

*for line in f:*

*print line*

*break*

# python树结构

*#coding=utf-8*

***from collections import defaultdict***

*#定义一个可形成树的方法*

*tree=lambda: defaultdict(tree)*

*#实例化一个树*

*db=tree()*

*db['user']['name']='jack'*

*import json*

*print json.dumps(db)*

*#{"user": {"name": "jack"}}*

# 拆箱

example = [1, 2, 3, 4]

a, \*b, c = example

# a -> 1

# b -> [2, 3]

# c -> 4

# 反转字典

**方法1：**

ex\_dict={'a':1,'b':2,'c':3}

tmp=zip(ex\_dict.values(),ex\_dict.keys())

print tmp #[(1, 'a'), (3, 'c'), (2, 'b')]

result=dict(tmp)

print result #{1: 'a', 2: 'b', 3: 'c'}

**方法2：**

example = {

'a': 1,

'b': 2,

'c': 3

}

elpmaxa = {key: value for value, key in example.items()}

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

# 列表

列表切片总结：

1. 对于任何范围[start:end]，切片后的序列为包括start在内，但不包括end的序列，适用于正向索引和负向索引。--前闭后开

2. 使用seq[index]方式，index不能超过最大索引；但是seq[start:x]方式，x可以出现超过最大索引的情况，实际只获取到最大索引的元素。

3. 如果开始索引或结束索引没有指定，则会分别以序列的第一个或最后一个索引值为默认值。

4. 对于任何范围[start:end]，需满足start<end，否则会返回空的序列。

5. 默认索引为正序索引。

**reverse()、sort()没有返回值；需要返回值可用sorted()**

>>> lis.sort()

>>> lis

[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> lis.reverse()

>>> lis

[8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> sorted(lis)

[0, 1, 2, 3, 4, 5, 6, 7, 8]

## Sort、sorted函数

**自定义cmp、key、reverse函数**

list1 = [(-1,5,3),(-5,3,6,3),(1,1,2,4,5,6),(2,9),(-2,10)]

def compare(a,b):

if abs(a)>abs(b):

return 1

elif abs(a)==abs(b):

return 0

else:

return -1

def L(tup) :

return tup[0]

list1.sort(cmp=compare,key = L,reverse = False)

print list1

>>> lis.sort(cmp=lambda x,y:1 if abs(x)>abs(y) else -1,key=lambda x:x[0])

>>> lis

[(1, 2, 1, 5, 4, 2), (-1, 5, 3), (-5, 2, 1)]

## 复制

>>> a

[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> b=a[:]

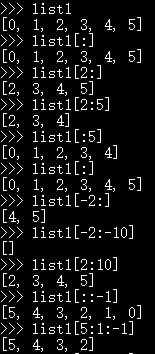
>>> a.remove(5)

>>> a

[0, 1, 2, 3, 4, 6, 7, 8]

>>> b

[0, 1, 2, 3, 4, 5, 6, 7, 8]



## 生成器

>>> g=(x+y for x in '12' for y in '0123456789')

>>> g

<generator object <genexpr> at 0x022FB4E0>

>>> g.next()

'10'

>>> for x in g:

... print g.next()

...

生成器的方法：

['gi\_frame', 'gi\_running', 'next', 'send', 'throw']

**使用生成器**

*#coding=utf-8*

*def get\_a\_index(strr):*

*for index,letter in enumerate(strr):*

*if letter=='a':*

*yield index*

*# 生成器在获取完之后如果继续通过 next() 取值，则会触发 StopIteration 错误*

*# 但通过 for 循环遍历时会自动捕获到这个错误*

*indexs=get\_a\_index('this ia a test to awo')*

*for i in indexs:*

*print i,*

*#若还是要一个列表，可使用list方法*

*result=list(indexs)*

*print result*

## 可迭代对象

在自定义类中实现\_\_iter\_\_方法，但只能通过for 循环迭代，要使用next，需要用iter函数

*#coding=utf-8*

*class LoopIter(object):*

*def \_\_init\_\_(self,data):*

*self.data=data*

*#在\_\_iter\_\_中yield结果*

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

*for index,letter in enumerate(self.data):*

*if letter=='a':*

*yield index*

*#可重复迭代*

*indexs=LoopIter('a test mama, a big dog')*

*for i in indexs:*

*print i,*

*for j in indexs:*

*print j,*

*#使用next需要使用iter方法*

*indexs=iter(indexs)*

*print next(indexs)*

## copy 、deepcopy

import copy

a = [1, 2, 3, 4, ['a', 'b']] #原始对象

b = a #赋值，传对象的引用

c = copy.copy(a) #对象拷贝，浅拷贝 ， 不会拷贝[‘a’,’b’]

d = copy.deepcopy(a) #对象拷贝，深拷贝 , 会拷贝[‘a’,’b’]

a.append(5) #修改对象a

a[4].append('c') #修改对象a中的['a', 'b']数组对象

print 'a = ', a

print 'b = ', b

print 'c = ', c

print 'd = ', d

# 元组

不可变、安全

1. 主动表达，不逃避，不懒惰
2. 积极学习，主动分享
3. 为人正直
4. 让更多人认识自己，扩大影响力
5. 给与他人更多帮助

# 测试分类

单元测试的收益最大

接口测试的收益七次

其次

UI测试的收益最差

UI 缺点：执行速度太慢，界面经常变，测试代码要经常维护，成本太高

测试开发：开发测试工具、测试框架、测试系统、测试脚本、设计测试方案、性能测试

安全测试

问你的质量体系的构建，和优化

这个应该是QA(测试)的责任

测试开发：开发测试工具、测试框架、测试系统、测试脚本、设计测试方案、性能测试

安全测试

问你的质量体系的构建，和优化

这个应该是QA(测试)的责任

ngix反向代理

# 字符串

字符串类型：str、 unicode，可通过isinstance()

Print 函数会调用\_\_str\_\_() ; 不加print会调用\_\_repr\_\_()方法

R’\n\t’ 原字符输出raw---不转义直接输入（可用于文件路径\\）

字符基本操作：+ \* [:]

格式化输出：>>> '%d%%' %45 ----'45%' %s ,%c ,%d ,%.2f ,%.2e ,%o ,%x

Lower() ,upper() ,swapcase() ,title() ,capitalize() ,strip() ,split() ,join() ,find() ,zfill() ,ljust() ,center()

## S.translate()函数

S.translate(table[,deletechars])

# 使用上面的函数产后的翻译表，把S进行翻译，并把deletechars中有的字符删掉。需要注意的是，如果S为unicode字符串，那么就不支持 deletechars参数，可以使用把某个字符翻译为None的方式实现相同的功能

Import string

Map=string.maketrans(‘123’,’abc’)

S=’1234abc’

Print s.translate(map)

# 文件操作

## 相对路径vs绝对路径

相对路径（网页css、js文件路径）：方便移植、修改，在不同服务器上部署文件。

## 读写模式

w—写；r—读；w+--清空后读写；r+--不清空读写（游标在文件开头，会替换）；wb—二进制；a—追加（游标在文件末尾）

r+模式：会将乱码写入文件，读取必须从文件起始位读。

## seek

seek(offset[,where]) seek(1,0) //0表示绝对位置，1表示偏移量；seek(2,1) //2表示偏移量，1表示当前位置；seek(-2,2) //-2表示偏移量往前2个字符，2表示文件结尾位置。

## 文件对象方法：

[ 'close', 'closed', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines', 'next', 'read', 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell', 'truncate', 'write', 'writelines', 'xreadlines']

* 在对文件中的内容编解码时可通过chardet包chardet.detect()方法查看当前文件的编码方式，对应的解码如jdecode('utf-8’)，再编码为可显示的形式如encode(‘gbk’,’ignore’)，cmd的交互模式的输出方式为中文gbk，其中ignore不可省，否则会报错。

## CSV文件读写

*#coding=utf-8*

*import csv*

*headers = ['Symbol', 'Price', 'Date', 'Time', 'Change', 'Volume']*

*rows = [{'Symbol':'AA', 'Price':39.48, 'Date':'6/11/2007',*

*'Time':'9:36am', 'Change':-0.18, 'Volume':181800},*

*{'Symbol':'AIG', 'Price': 71.38, 'Date':'6/11/2007',*

*'Time':'9:36am', 'Change':-0.15, 'Volume': 195500},*

*{'Symbol':'AXP', 'Price': 62.58, 'Date':'6/11/2007',*

*'Time':'9:36am', 'Change':-0.46, 'Volume': 935000},*

*]*

*rows1 = [('AA', 39.48, '6/11/2007', '9:36am', -0.18, 181800),*

*('AIG', 71.38, '6/11/2007', '9:36am', -0.15, 195500),*

*('AXP', 62.58, '6/11/2007', '9:36am', -0.46, 935000),*

*]*

*with open('stocks.csv','w') as f:*

*f\_csv=csv.DictWriter(f,headers)*

*f\_csv.writeheader()*

*f\_csv.writerows(rows)*

*with open('stocks.csv','w') as f:*

*f\_csv=csv.writer(f)*

*f\_csv.writerow(headers)*

*f\_csv.writerows(rows1)*

*#coding=utf-8*

*import csv*

*with open('stocks.csv') as f:*

*f\_csv=csv.DictReader(f)*

*for row in f\_csv:*

*print row['Change']*

*from collections import namedtuple*

*with open('stocks.csv') as f:*

*f\_csv=csv.reader(f)*

*headings=next(f\_csv)*

*Row=namedtuple('Row',headings)*

*for r in f\_csv:*

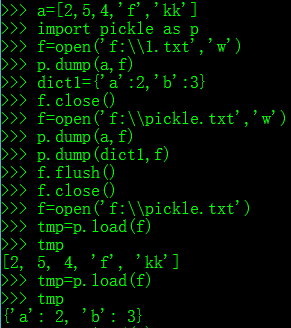
*row=Row(\*r)*

*print row.Change*

# 序列化

将对象存入文件中保存

Import cPickle as cp



pickle.dumps()方法把任意对象序列化成一个str，然后，就可以把这个str写入文件。或者用另一个方法pickle.dump()直接把对象序列化后写入一个file-like Object：

## 对象的实例化：json

JSON(JavaScript Object Notation, JS 对象标记) 是一种轻量级的数据交换格式。

**Json.dump(obj,fp) json.dumps(obj)**

*>>> json.dumps((1,2,3))*

*'[1, 2, 3]'*

*>>> json.dumps({1:'w','e':'ff'})*

*'{"1": "w", "e": "ff"}'*

*>>> json.dumps([1,2,3])*

*'[1, 2, 3]'*

*>>> json.dumps('hello')*

*'"hello"'*

*>>>*

*import json*

*def is\_josn(arg):*

*try:*

*json.loads(arg)*

*except Exception,e:*

*return False*

*else:*

*return True*

json值可以是：数字、true、false、字符串、数组 []、对象 {}、null

*>>> json.loads('{"2":null}')*

*{u'2': None}*

*>>> json.loads('{"2":{"3":"erp"}}')*

*{u'2': {u'3': u'erp'}}*

*>>> json.loads('{"2":23.9}')*

*{u'2': 23.9}*

*>>> json.loads('{"2":[1,2.7]}')*

*{u'2': [1, 2.7]}*

*#file*

*>>> with open('f.txt','w') as f:*

*... json.dump('{2:"ww"}',f)*

*>>> with open('f.txt') as f:*

*... json.load(f)*

*u'{2:"ww"}'*

*import json*

*class Employee(object):*

*def \_\_init\_\_(self, name, age, sex, tel):*

*self.name = name*

*self.age = age*

*self.sex = sex*

*self.tel = tel*

*emp = Employee('Lily', 24, 'female', '18223423423')*

*print emp.\_\_dict\_\_*

*print(json.dumps(emp, default = lambda Employee: Employee.\_\_dict\_\_))*

*print(json.dumps(emp, default = lambda emp: emp.\_\_dict\_\_))*

*#{"age": 24, "tel": "18223423423", "name": "Lily", "sex": "female"}相同输出*

*例2：*

*class Employee(object):*

*def \_\_init\_\_(self, name, age, sex, tel):*

*self.name = name*

*self.age = age*

*self.sex = sex*

*self.tel = tel*

*emp = Employee('Lily', 24, 'female', '18223423423')*

*def jsonToClass(emp):*

*return Employee(emp['name'], emp['age'], emp['sex'], emp['tel'])*

*json\_str = '{"name": "Lucy", "age": 21, "sex": "female", "tel": "15834560985"}'*

*e = json.loads(json\_str,* ***object\_hook*** *= jsonToClass)*

*print e #<\_\_main\_\_.Employee object at 0x00000000022C58D0>*

*print e.name #Lucy*

无法把Student类实例序列化为JSON，是因为默认情况下，dumps()方法不知道如何将Student实例变为一个JSON的{}对象，通常class的实例都有一个\_\_dict\_\_属性，它就是一个dict，用来存储实例变量。因此可以这样写：json.dumps(s,default=lambda x:x.\_\_dict\_\_)

class Student(object):

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

self.name=name

self.age=age

self.score=score

def study(self):

print 'i am studying'

xm=Student('xm',78,99.5)

print json.dumps(xm,default=lambda x:x.\_\_dict\_\_)

def dict2obj(dict\_str):

return Student(dict\_str['name'],dict\_str['age'],dict\_str['score'])

dict\_str='{"name":"jack","age":33,"score":89.9}'

jack=json.loads(dict\_str,object\_hook=dict2obj)

print "jack's name is ",jack.name

jack.study()

## XML

解析的三种方法：SAX【省内存】 DOM【占内存】 ElementTree【支持xpath】

### Dom方式

**From xml.dom.minidom import parse**

**Booklist=parse(filename)**

**Dir(booklist):**

['appendChild', 'attributes', 'childNodes', 'cloneNode', 'firstChild', 'getAttribute', 'getAttributeNS', 'getAttributeNode', 'getAttributeNodeNS', 'getElementsByTagName', 'getElementsByTagNameNS', 'getInterface', 'getUserData', 'hasAttribute', 'hasAttributeNS', 'hasAttributes', 'hasChildNodes', 'insertBefore', 'isSameNode', 'isSupported', 'lastChild', 'localName', 'namespaceURI', 'nextSibling', 'nodeName', 'nodeType', 'nodeValue', 'normalize', 'ownerDocument', 'parentNode', 'prefix', 'previousSibling', 'removeAttribute', 'removeAttributeNS', 'removeAttributeNode', 'removeAttributeNodeNS', 'removeChild', 'replaceChild', 'schemaType', 'setAttribute', 'setAttributeNS', 'setAttributeNode', 'setAttributeNodeNS', 'setIdAttribute', 'setIdAttributeNS', 'setIdAttributeNode', 'setUserData', 'tagName', 'toprettyxml', 'toxml', 'unlink', 'writexml']

**Books=Booklist.getElementsByTagName(‘book’)**

**Dir(books)**

['abort', 'actualEncoding', 'appendChild', 'async', 'attributes', 'childNodes', 'cloneNode', 'createAttribute', 'createAttributeNS', 'createCDATASection', 'createComment', 'createDocumentFragment', 'createElement', 'createElementNS', 'createProcessingInstruction', 'createTextNode', 'doctype', 'documentElement', 'documentURI', 'encoding', 'errorHandler', 'firstChild', 'getElementById', 'getElementsByTagName', 'getElementsByTagNameNS', 'getInterface', 'getUserData', 'hasChildNodes', 'implementation', 'importNode', 'insertBefore', 'isSameNode', 'isSupported', 'lastChild', 'load', 'loadXML', 'localName', 'namespaceURI', 'nextSibling', 'nodeName', 'nodeType', 'nodeValue', 'normalize', 'ownerDocument', 'parentNode', 'prefix', 'previousSibling', 'removeChild', 'renameNode', 'replaceChild', 'saveXML', 'setUserData', 'standalone', 'strictErrorChecking', 'toprettyxml', 'toxml', 'unlink', 'version', 'writexml']

>>> print booklist.getElementsByTagName('author')[1].toxml()

<author>李四</author>

>>> booklist.getElementsByTagName('book')[1].getElementsByTagName('title')

from xml.dom.minidom import parse

import MySQLdb

#解析xml

parser=parse(r'C:\Users\Administrator\Desktop\file\manager.xml')

database=parser.documentElement

host=database.getElementsByTagName('host')[0].childNodes[0].data

user=database.getElementsByTagName('user')[0].childNodes[0].data

password=database.getElementsByTagName('passwd')[0].childNodes[0].data

db=database.getElementsByTagName('db')[0].childNodes[0].data

#连接数据库

try:

con=MySQLdb.connect(host=host,

port=3306,

user=user,

passwd="",

db=db)

except Exception,e:

print 'error'

finally:

con.close()

#写入excel

from openpyxl import Workbook

wb=Workbook()

ws=wb.active

ws.append(['host','user','password','db'])

ws.append([host,user,password,db])

wb.save('xml.xlsx')

**创建XML文件**

*#coding=utf-8*

*import xml.dom.minidom*

*import codecs*

*#创建空文件*

*doc=xml.dom.minidom.Document()*

*#创建根节点*

*root=doc.createElement('database')*

*#设置属性*

*root.setAttribute('author',u'光荣')*

*doc.appendChild(root)*

*#创建子节点*

*conf=doc.createElement('config')*

*host=doc.createElement('host')*

*host.appendChild(doc.createTextNode('127.0.0.1'))*

*port=doc.createElement('port')*

*port.appendChild(doc.createTextNode('3306'))*

*user=doc.createElement('user')*

*user.appendChild(doc.createTextNode('root'))*

*#添加子节点*

*conf.appendChild(host)*

*conf.appendChild(port)*

*conf.appendChild(user)*

*root.appendChild(conf)*

*#写入文件*

*f=codecs.open('database.xml','w','utf-8')*

*doc.writexml(f,indent='',addindent='\t',newl='\n',encoding='utf-8')*

*f.close()*

*例2：*

*#coding=utf-8*

*import xml.dom.minidom*

*#在内存中创建一个空的文档*

*doc = xml.dom.minidom.Document()*

*#创建一个根节点Managers对象*

*root = doc.createElement('Managers')*

*#设置根节点的属性*

*root.setAttribute('company', 'xx科技')*

*root.setAttribute('address', '科技软件园')*

*#将根节点添加到文档对象中*

*doc.appendChild(root)*

*managerList = [{'name' : 'joy', 'age' : 27, 'sex' : '女'},*

*{'name' : 'tom', 'age' : 30, 'sex' : '男'},*

*{'name' : 'ruby', 'age' : 29, 'sex' : '女'} ]*

*for i in managerList :*

*nodeManager = doc.createElement('Manager')*

*nodeName = doc.createElement('name')*

*#给叶子节点name设置一个文本节点，用于显示文本内容*

*nodeName.appendChild(doc.createTextNode(str(i['name'])))*

*nodeAge = doc.createElement("age")*

*nodeAge.appendChild(doc.createTextNode(str(i["age"])))*

*nodeSex = doc.createElement("sex")*

*nodeSex.appendChild(doc.createTextNode(str(i["sex"])))*

*#将各叶子节点添加到父节点Manager中，*

*#最后将Manager添加到根节点Managers中*

*nodeManager.appendChild(nodeName)*

*nodeManager.appendChild(nodeAge)*

*nodeManager.appendChild(nodeSex)*

*root.appendChild(nodeManager)*

*#开始写xml文档*

*fp =codecs. open('Manager.xml', 'w', ’utf-8’)*

*doc.writexml(fp, indent='\t', addindent='\t', newl='\n', encoding="utf-8")*

*fp.close()*

### Etree方式

*try:*

*import xml.etree.cElementTree as ET*

*except ImportError:*

*import xml.etree.ElementTree as ET*

*#coding=utf-8*

*import sys*

*try:*

*import xml.etree.cElementTree as ET*

*except ImportError:*

*import xml.etree.ElementTree as ET*

*tree = ET.ElementTree(file='movies.xml')*

*root=tree.getroot()*

*print root.tag*

*print root.attrib*

*for child\_of\_root in root:*

*print child\_of\_root.tag*

*print "\*\*\*\*\*\*\*\*", child\_of\_root.attrib*

*print "\*"\*50*

*print root[0].tag*

*print root[0].text*

*print root[0][0].tag*

*print root[0][0].text*

*print "\*"\*50*

*for elem in tree.iter(): #遍历所有元素，所有标签*

*print elem.tag, elem.attrib*

*print "\*"\*50*

*for elem in tree.iterfind('movie/type'):#查找movie下的所有type标签*

*print elem.tag, elem.attrib*

*for i in tree.findall("movie"):*

*print "movie:",i,i.tag,i[0].tag,i[0].text*

*for i in tree.findall("movie/format"):*

*print "movie:",i,i.tag,i.tag,i.text*

*print "\*"\*50*

*for elem in tree.iter('tag=stars'):#查找标签为star的元素*

*print elem.tag, elem.attrib*

*print "\*"\*50*

*for elem in tree.iterfind('\*[@title="Ishtar"]'): #查找属性为title="Ishtar"的元素*

*print elem.tag, elem.attrib*

*print "-"\*50*

*root = tree.getroot() #获取<collection>元素*

*print "root:",root[0].tag #打印第一级movie元素的标签，为movie*

*print "subnode:",root[0][0].tag #打印第一级movie元素下的第一个子元素标签type*

*print "subnode:",root[0][1].tag #打印第一级movie元素下的第二个子元素标签format*

*print "subnode:",root[0][2].tag #打印第一级movie元素下的第三个子元素标签year*

*print "subnode:",root[0][3].tag #打印第一级movie元素下的第四个子元素标签rating*

*print "subnode:",root[0][4].tag #打印第一级movie元素下的第五个子元素标签stars*

*del root[0][4] #删除第一级movie元素下的第四个子元素*

*del root[0][3] #删除第一级movie元素下的第三个子元素*

*del root[0][2] #删除第一级movie元素下的第二个子元素*

*del root[0][1] #删除第一级movie元素下的第一个子元素*

*del root[3] #删除第四个movie元素*

*del root[2] #删除第三个movie元素*

*for subelem in root:*

*print subelem.tag, subelem.attrib #打印第一个movie和第二个movie元素的标签和属性*

*tree.write(sys.stdout) #将xml文件的内容写到屏幕上*

*#tree.write("d:\\movies.xml") #将变更的xml文件写入到文件中*

*例2*

*#coding=utf-8*

*from xml.etree.ElementTree import parse*

*from urllib import urlopen*

*from openpyxl import Workbook*

*content=urlopen('http://planet.python.org/rss20.xml')*

*doc=parse(content)*

*wb=Workbook()*

*ws=wb.active*

*ws.append(['title','date','link'])*

*for item in doc.iterfind('channel/item'):*

*title=item.findtext('title')*

*date=item.findtext('pubDate')*

*link=item.findtext('link')*

*ws.append([title,date,link])*

*wb.save('python.xlsx')*

## 包

Linecache、chardet

# Excel操作

from openpyxl import Workbook

wb=Workbook()

ws=wb.active

ws.append(['host','user','password','db'])

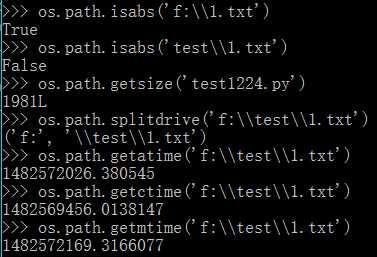
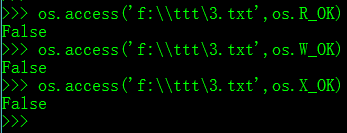
ws.append([host,user,password,db])

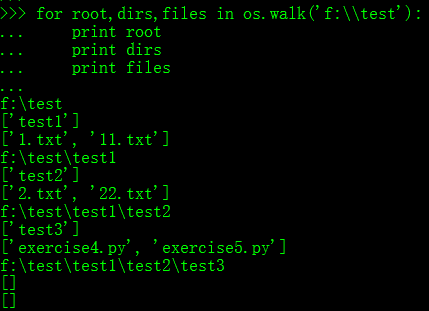
wb.save('xml.xlsx')

# 目录操作

Os.chdir(path) ,os.mkdir(filename) ,os.makedirs(‘f:\\t\\t\\d’) ,os.pardir ,os.curdir os.rename(oldname,newname) ,os.stat(filename) //文件属性 ,os.system() //系统命令 ,os.remove(filename) ,os.rmdir(path) ,os.removedirs(path) ,os.access() ,os.popen() ,os.walk()

os.path.getsize(filename) ,os.path.isdir(dir) ,os.path.exists(filename) ,os.path.isfile(file) ,os.path.abspath(path) ,os.path.join(path,file) ,os.path.split(filename) ,os.path.splitext(filename) ,os.path.basename(filename) ,os.path.dirname(filename) ,os.path.splitdrive(path)





## mysql中命令：

truncate是把占用的空间给释放掉

drop table是删除表

delete只删除了表中的数据，但是之前数据占用的空间没有被释放

# 异常处理

IndexError ,NameError ,ValueError , ZeroDivisionError , SyntaxError , KeyError ,IOError ,AttributeError ,TypeError ,AssertionError ,NotImplementedError ,StandarError

try：

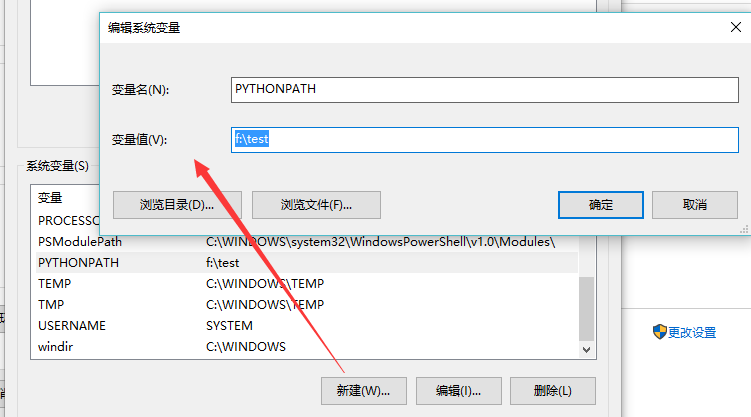
Int(raw\_input(‘input a number: ’))

except Exception,e:

print e

# 模块

* 导入顺序：同级目录下🡪python path(os.sys.path)🡪site-package



* 多次导入，以最后导入的函数生效。
* Python文件中if \_\_name\_\_==’\_\_main\_\_’ 之上的语句也会执行
* \_\_init\_\_.py中\_\_all\_\_=[‘’] //在from xx import \* 时导入哪些模块（导入all中的模块）;若本包为a，也可以导入包：from xx import \* ，可直接使用本包a调用其他模块（若导入出错，reload(a)）

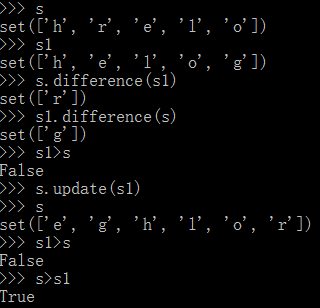
# 包

* 必须有\_\_init\_\_.py文件，包的标识
* 引用：import glory.tmp glory.tmp.sum(4,5)
* 包中包含模块（py文件），模块中内容为不同的函数

# 集合

[**'add'**, **'clear'**, **'copy'**, 'difference', 'difference\_update', **'discard'**, 'intersection', 'intersection\_update', 'isdisjoint', 'issubset', 'issuperset', **'pop'**, **'remove'**, 'symmetric\_difference', 'symmetric\_difference\_update', 'union', **'update'**]

* Discard方法：删除元素，若元素不存在，没有任何操作，不会报异常，remove方法会报异常。



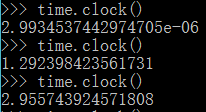
# 时间模块

Time.time //时间戳 time.localtime() //时间元组

Time.mktime(时间元组) //传入时间元组—>返回时间戳

Time.struct\_time([2017,1,7,16,59,45,5,7,0]) //传入9位数的序列🡪返回时间元组。

Time.lock() //需初始化，类似秒表功能，累计计时



time.strftime(‘%Y-%m-%d %H:%M:%S’,time.localtime()) //格式化时间输出

time.strptime() time.strptime(stime,"%Y-%m-%d %H:%M:%S")//返回时间元组

## datetime

* **datetime模块：**

['date', 'datetime', 'datetime\_CAPI', 'time', 'timedelta', 'tzinfo']

* **Datetime.date模块**:

['ctime', 'day', 'fromordinal', 'fromtimestamp', 'isocalendar', 'isoformat', 'isoweekday', 'max', 'min', 'month', 'replace', 'resolution', 'strftime', 'timetuple', 'today', 'toordinal', 'weekday', 'year']

* **Datetime.datetime模块：**

['astimezone', 'combine', 'ctime', 'date', 'day', 'dst', 'fromordinal', 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday', 'max', 'microsecond', 'min', 'minute', 'month', 'now', 'replace', 'resolution', 'second', 'strftime', 'strptime', 'time', 'timetuple', 'timetz', 'today', 'toordinal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow', 'utcoffset', 'utctimetuple', 'weekday','year']

* **Datetime.time模块**

['dst', 'hour', 'isoformat', 'max', 'microsecond', 'min', 'minute', 'replace', 'resolution', 'second', 'strftime', 'tzinfo', 'tzname', 'utcoffset']

* **Datetime.timedelta（时间差，比如推算3天后或10小时前）**

['days', 'max', 'microseconds', 'min', 'resolution', 'seconds', 'total\_seconds']

Datetime.date.today()-datetime.timedelta(days=3)

* **Datetime.tzinfo**

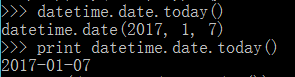
['dst', 'fromutc', 'tzname', 'utcoffset']

## Calendar

Calendar可直接打印出月日历、年日历输出格式：calendar.month(3,2,1,1) calendar.calendar(2017,3,2,1)

时间模块需要转为字符串输出：print datetime.date.today() //会自动调用类的str方法输出，否则为时间类型。

now=datetime.datetime.now()





Now.replace(year=2012,month=12)

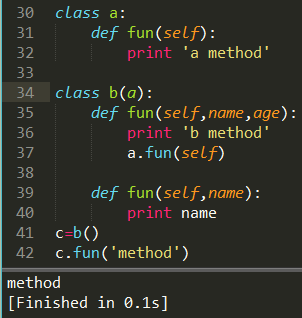
Now.timetuple()

# 面向对象

## 概念部分

* 对函数进行分类封装，使开发更方便。
* 类是用来描述具有相同属性和方法的对象的集合，对象是类的实例。
* 类变量在整个实例化的对象中是公用的，一般由类直接调用（也可通过实例调用）
* 实例变量是定义在方法中的变量，只可以通过实例调用（一般为self.xxx）
* 类分两种：经典类（类名称后没有括号）和新式类（需写明继承自object）
* 类中的所有普通方都必须带self参数，类方法（classmethod）静态方法（staticmethod）除外。
* Self参数代表实例本身，当使用实例变量或调用方法时（在类中）需要使用self.xxx，实例变量一般定义在\_\_init\_\_()方法中，init方法为类的构造方法，在实例化的过程中自动调用。
* 函数式编程的应用场景-🡪各函数之间独立且无公用数据；面向对象编程的应用场景-🡪各函数之间共用一组数据。
* 对函数式编程、面向对象编程的选择：面向对象一般用于1.多函数需使用共同的值如操作数据库；2.多个事务之间存在共性，需要创建多个事务。
* 调用被封装的内容可以有两种情况：1.通过对象直接调用被封装的内容；2.通过self间接调用被封装的内容self.name self.say\_hello()。
* 类成员有字段、属性、方法三种：字段（普通字段、静态字段）、方法（普通方法【必须有self参数】、类方法【必须有cls参数，代表类本身】、静态方法【无默认参数】）、属性（普通属性property）
* 对于类方法、静态方法需加装饰器说明：@classmethod @staticmethod
* 属性的定义：需加@property装饰器，只有一个self参数。
* 新式类中的属性定义方式：@property @方法名.setter @方法名.deleter ，当属性被赋值时会自动调用setter方法，当属性被删除时会自动调用deleter方法
* 可用过实例化对象动态的进行添加、修改、删除属性操作：p=Person() p.name=’wang’ ; p.name=’Li’ ; del p.name
* 也可通过函数进行：getattr(obj,name) ; hasattr(obj,name) ; setattr(obj,name,value) ; delattr(obj,name) ,实际使用过程中需对getattr delattr 方法做捕获异常处理。
* Python的内置类属性：\_\_name\_\_ , \_\_dict\_\_ , \_\_module\_\_ , \_\_bases\_\_ , \_\_doc\_\_ , \_\_file\_\_
* 类变量在类中的所有实例之间共享，可通过类名直接调用，也可通过实例调用，效果一样。
* Python的隐藏数据：以\_\_开头的变量名，不能在类的外部使用或直接访问。但可以通过对象名.\_类名\_\_变量名 访问。p.\_Person\_\_inname
* 私有方法：以双下划线开始的方法名，内部调用self.\_\_func()

## 继承

* 子类中若没有init构造方法，会自动调用父类的构造方法，若有init构造方法则会调用自己本身的构造方法，其他普通方法会自动继承。
* 继承构造方法的两种形式：baseclass.\_\_init\_\_(self,name,age) ; super(subclass,self).\_\_init\_\_(name,age)【super只能在新式类中使用】
* 若子类没有实现构造方法\_\_init\_\_ 则会直接调用父类的构造方法。
* 子类调用父类的方法的两种方式:在子类中调用 parent.parentMethod(self) self参数必须写; self.parentMethod()
* Python中的super函数只能在构造方法中使用。
* 类之间关系判断：isinstance(实例,类) issubclass(子类,父类)
* 多重继承：若子类没有实现构造方法，则以第一个父类为中心。若实现了构造方法，继承具体的父类只能使用这种方式：superclass.\_\_init\_\_(self,name,age),因为必须指明是继承哪个父类的方法。
* 经典类的多继承情况，按照**深度优先**方式查找；新式类的多继承情况，按照**广度优先**方式查找。MRO（method resolution order）[方法解析顺序](http://python.jobbole.com/85685/)。 **类名.\_\_mro\_\_** 查看查找顺序
* Python中的方法重写：方法名一样就会重写（覆盖之前的同名方法），以最后一个方法为准，无论参数个数。
* **Property**：TypeError: property() takes at most **4 arguments** (5 given) name = property(getName,setName,delName,'name property docs') 映射对应关系，顺序不可变，说明文档：Person.name.\_\_doc\_\_

**描述符**

*class CheckInteger(object):*

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

*self.name=name*

*def \_\_get\_\_(self,instance,cls):*

*if instance is None:*

*return self*

*else:return instance.\_\_dict\_\_[self.name]*

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

*if not isinstance(value,int):*

*raise TypeError('expected an int')*

*instance.\_\_dict\_\_[self.name]=value*

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

*del instance.\_\_dict\_\_[self.name]*

*class Point(object):*

*x=CheckInteger('x')*

*y=CheckInteger('y')*

*def \_\_init\_\_(self,x,y):*

*self.x=x*

*self.y=y*

*point=Point(1,2)*

*print point.x*

*point.x=3*

*print point.x*

*point.x='e' #raise TypeError('expected an int')*

*print Point.x #<\_\_main\_\_.CheckInteger object at 0x000000000221AD68>*

## 多态

* 多态可以对不同的对象使用同样的操作，但他们可能会以多种形态呈现出结果。传入对象（实例）

## 类的内置方法

Print 会自动调用str()方法，str()方法会自动调用类中的\_\_str\_\_()方法，

* \_\_getitem\_\_() , \_\_setitem\_\_() , \_\_delitem\_\_() 类的3个方法在obj[‘k’]---调用
* \_\_getslice\_\_ , \_\_setslice\_\_ , \_\_delslice\_\_用于切片操作
* \_\_iter\_\_() 用于迭代器，将对象转为可迭代对象
* \_\_init\_\_ 构造函数（创建实例时调用）、\_\_del\_\_ 析构函数（实例销毁时被调用）、\_\_repr\_\_ 转化为供解释器读取的形式（repr()方法时调用）、\_\_str\_\_ 转为适合人阅读的形式（str()方法时调用）、\_\_cmp\_\_ 对象比较（cmp调用）、\_\_call\_\_ 对象后面加括号触发执行
* \_\_new\_\_类准备将自身实例化时调用，始终是类的静态方法，执行顺序先于\_\_init\_\_方法，格式：\_\_new\_\_(cls,\*args,\*\*kw) cls代表要实例化的类，必须要有返回值，返回实例化出来的实例return object.\_\_new\_\_(cls,\*args,\*\*kw)
* 单例：一个类只能实例化一个对象，不允许有多个实例

### \_\_slots\_\_

定义类的属性，告诉python不要使用字典而是固定的集合来分配空间

*#coding=utf-8*

*class Test(object):*

*#用列表罗列出所有属性*

*\_\_slots\_\_=['name','age']*

*def \_\_init\_\_(self,name='python',age=10):*

*self.name=name*

*self.age=age*

*test=Test()*

*print test.name*

*test.new\_key='new\_key'*

*#AttributeError: 'Test' object has no attribute 'new\_key'*

### \_\_call\_\_

让类可以直接调用

*#coding=utf-8*

*class AddNum(object):*

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

*self.num=0*

*def \_\_call\_\_(self,num=1):*

*self.num+=num*

*add=AddNum()*

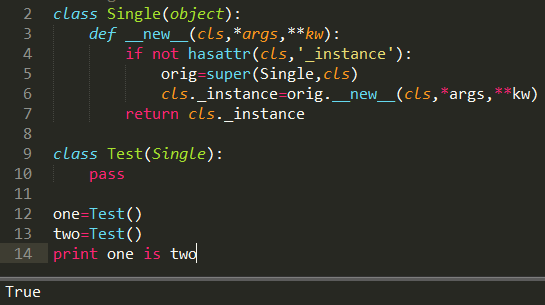
*print add.num #0*

*add()*

*print add.num #1*

*add(4)*

*print add.num #5*



*>>> a,b,c=map(str,'12-23-45'.split('-'))*

*>>> a*

*'12'*

## 类方法、静态方法

*#coding=utf-8*

*class Date(object):*

*def \_\_init\_\_(self,year=0,month=0,day=0):*

*self.year=year*

*self.month=month*

*self.day=day*

*if not Date.is\_month\_valid(self.month):*

*print '{} is a invalid month number'.format(month)*

*@classmethod*

*def from\_string(cls,string):*

*year,month,day=map(str,string.split('-'))*

*date=cls(year,month,day)*

*return date*

*@property*

*def time(self):*

*return "{year}-{month}-{day}".format(year=self.year,*

*month=self.month,*

*day=self.day)*

*@staticmethod*

*def is\_month\_valid(month):*

*return int(month)<=12 and int(month)>=1*

*date=Date('2017','04','23')*

*print date.time*

*# 可通过新增classmethod类方法，在不改变现有接口和方法的情况下，*

*# 可以通过传入2016-11-09这样的字符串来创建一个Date实例*

*date1=Date.from\_string('2017-13-24')*

*print date1.time*

## Python垃圾回收

采用引用计数的方式，若某对象引用为0，则系统会自动销毁对象。

查看对象的引用次数：**sys.getrefcount(‘hello’)**

垃圾回收同样可处理循环引用的情况，（两个对象相互引用，没有其他变量引用他们）。处理机制：引用计数器+循环垃圾收集器，当对象不再被使用时\_\_del\_\_方法运行。

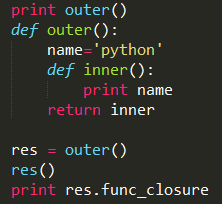
## 类的特殊成员

* \_\_doc\_\_文档字符串 , \_\_module\_\_ , \_\_class\_\_ , \_\_dict\_\_ , \_\_metaclass\_\_

## 闭包

内部函数中用到的外部变量会被打包进闭包，可在其他任意地方调用。返回函数对象，还没有调用。Print func.func\_closure 可查看闭包中的内容。**必须包含函数本身之外的变量**

定义：如果一个函数定义在另一个函数的作用域内，并且引用了外层的变量，则该函数称为闭包。



## 装饰器

### 用途

* 分析记录日志：在函数运行前后打印日志
* 验证运行时检查：对函数的返回值做判断，是否满足条件
* 创建框架：Flask 就用到装饰器将不同url路由到不同处理http请求函数
* 代码复用。

### 代码示例

*#coding=utf-8*

*def decorator\_fun(fun):*

*def new\_fun(\*args,\*\*kwargs):*

*print 'current fun: ',fun.\_\_name\_\_*

*print 'position arguments: ',args*

*print 'key arguments: ',kwargs*

*result=fun(\*args,\*\*kwargs)*

*print result*

*return result*

*return new\_fun*

*@decorator\_fun*

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

*return sum(args)+sum(kwargs.values())*

*add(1,2,3,l=4,k=5)*

*#带参数的装饰器*

*#coding=utf-8*

*From functools import wraps*

*def read\_file(filename='log.txt'):*

*def decorator\_fun(fun):*

*@wraps(fun) #为了不改变原函数的内容，fun.\_\_name\_\_ @wraps相当于装饰器的装饰器*

*def new\_fun(\*args,\*\*kwargs):*

*result=fun(\*args,\*\*kwargs)*

*with open(filename,'w') as f:*

*f.write(str(result)+'\n')*

*f.write('fun name : {name}\noposition args : {arg}\nkey args : {kw}'*

*.format(name=fun.\_\_name\_\_, arg=args,kw=kwargs))*

*return result*

*return new\_fun*

*return decorator\_fun*

*@read\_file()*

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

*return sum(args)+sum(kwargs.values())*

*add(1,2,3,k=5,l=6)*

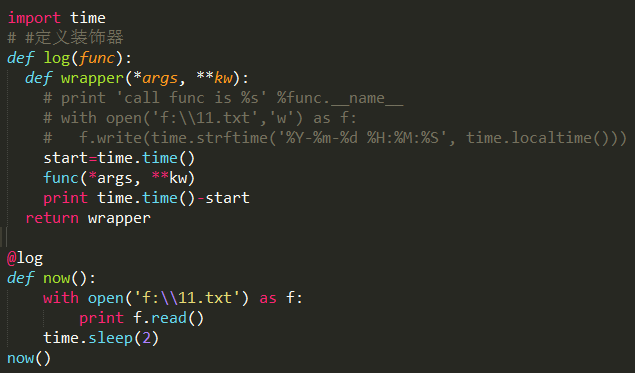
@g

@f

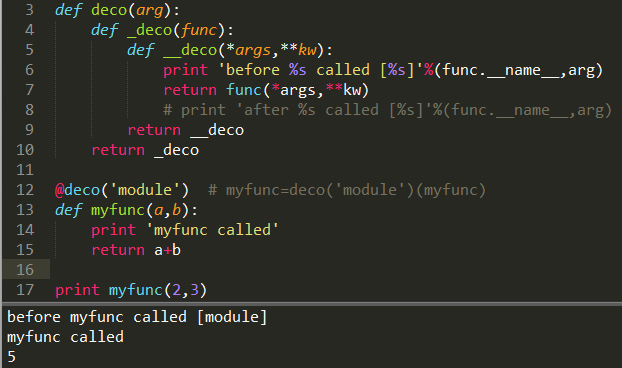
Def foo():

Pass

等价于 foo=g(f(foo))



**带参数的装饰器：**



**执行顺序**：先执行wrapper函数，再执行被装饰的函数。

* 作用：在代码运行期间不改变原函数定义的基础上，动态给函数增加功能。（插入日志、性能测试、事务处理等）可代码复用。

# 正则表达式

regular expression

## 应用：

* 判断某文本内容是否在文本中

Import re

Pattern=re.compile(r’\w\d\s2’)

Print pattern.match(‘s1 22’).group()

Print re.search(r’\s\w\d’,’ f3’).group()

Compile()指定匹配规则

Group()输出匹配到的结果

Search()只匹配第一个符合条件的字符串

Match()必须从第一个字符开始匹配，若第一个字符不符合则匹配失败

## 匹配字符：

\w--------------字母或数字

\d---------------数字

\s---------------空白字符

\W--------------取反—非字母非数字

\b---------------匹配的单词的边界

## 匹配数量：

\*----------------0个或多个

？---------------0个或1个

+---------------1个或多个（至少一个）

{3,6}-----------匹配前一个字符的3到6次，或指定次数{5}

[]-------或，匹配其中一个字符

^在[]中---非；在外---已某字符开始的字符串

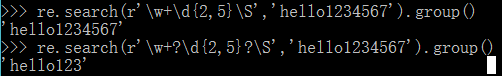
$----已某字符结尾的字符串





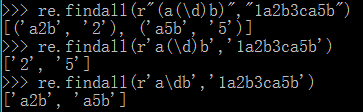


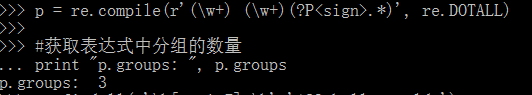
**正则匹配的贪婪性：匹配最多次，一直匹配到不符合为止，加？可以尽可能少的匹配**



## 方法

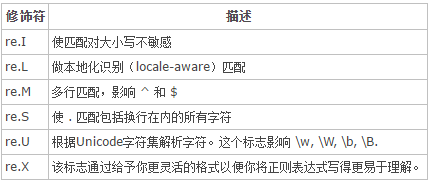
['findall', 'finditer', 'flags', 'groupindex', 'groups', 'match', 'pattern', 'scanner', 'search', 'split', 'sub', 'subn']





## 属性

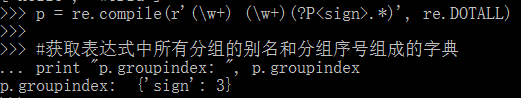
['DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE', 'M', 'MULTILINE', 'S', 'Scanner', 'T', 'TEMPLATE', 'U', 'UNICODE', 'VERBOSE', 'X',]



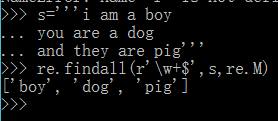


**忽略大小写匹配：**

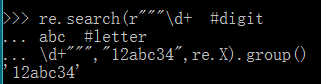




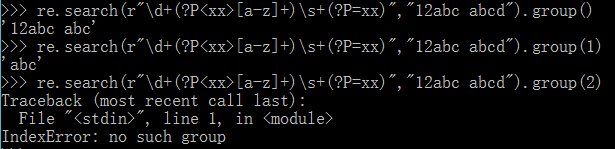
**Re.M :匹配多行内容**



**Re.X 注释**



## 命名分组：?P<>



## 前向肯定、否定；后向肯定、否定

s = 'aaa111aaa , bbb222 , 333ccc'

# 指定前后否定断言，不满足前三个字母和后三个字母的条件

print re.findall( r'(?<![a-z]{3})\d+(?![a-z]+)', s) [1,22,33]

# 操作excel文件

* xlsx格式文件。导入openpyxl模块
* 获取一个文件对象wb=Workbook(guess\_types=True) 可选参数
* 获取sheet页：ws=wb.active //默认获取第一个sheet；创建sheet：ws1=wb.create\_sheet(u’练习’) ws2=wb.create\_sheet(‘test’,0) ；修改sheet的名字：ws1.title=u’修改’；修改sheet背景色：ws1.sheet\_properties.tabColor=’FFFFFF’；通过名字获取sheet：ws3=wb[u’练习’]、或者ws3=wb.get\_sheet\_by\_name(u’练习’)
* 写入内容：可根据单元格的坐标直接赋值ws['A1’]=’hello’ ；写入一行内容ws.append([12,’hello’,datetime.datetime.now()])
* 保存文件：wb.save(filename)
* 单元格操作：获取单元格对象cell=ws[‘A1’] 属性value、coordinate;设定单元格对象的值ws.cell(row=4,column=1,value=u’单元’) 【注意设置行列从1开始】 cell.value=u’修改’ ；属性，最大行ws.max\_row 最大列ws.max\_column ；获取整行ws.rows[0] 某行某列ws.rows[0][0]
* get\_column\_letter(column) //获取列所对应的字母【openpyxl.utils】
* 添加图片from openpyxl.drawing.image import Image 获取图片对象img=Image(‘1.png’) 🡪ws.add\_image(img,’A1’) 【注意每次添加图片都要重新获取图片img=Image(‘1.png’)】
* 获取一个已有的excel文件wb=load\_workbook(filename)
* 可在单元格中直接输入公式：ws[‘A1’]=’=SUM(1,2)’
* ws[‘A1’].number\_format //类型为General
* 样式：thin=Side(border\_style=’thin’,color=’FFFFFF’) [thin ,thick ,double] ,Border(top=thin,right=thin ,left=thin ,bottom=thin) ,Font(b=True,color=’000000’) ,Alignment(horizontal=’center’,vertical=’center’) ,PatternFill(‘solid’, fgColor=’000000’) ,Font(name=u’宋体’, color=colors.RED ,size=18 ,bold=True) [from openpyxl.styles import colors,Side,Border….]

# 网络编程

## 客户端/服务器架构： c/s架构

* 服务器存在的唯一目的就是等待客户的请求，给这些客户服务，然后再等待其他的请求。
* 硬件服务器：如打印机、文件服务器（磁盘）
* 软件服务器：Web服务器、数据库服务器、窗口服务器

## 套接字

* 网络化的应用程序在开始任何通讯之前都必须要创建套接字。
* Unix套接字（基于文件型），其家族名为AF\_UNIX ，AF\_LOCAL会代替AF\_UNIX，但python自己仍然使用AF\_UNIX
* 套接字分两种：基于文件型 & 基于网络型
* 基于网络型的套接字，家族名AF\_INET 是使用最广泛的一个。Python2.5中加入了一种Linux套接字的支持AF\_NETLINK，让用户代码与内核代码之间的IPC可以使用标准的BSD套接字接口。
* Python只支持AF\_UNIX , AF\_INET , AF\_NETLINK，在网络编程中用AF\_INET套接字。
* 一个因特网地址由网络通信所必须的主机与端口号组成。合法的端口号范围：0~65535，其中小于1024的端口为系统保留端口。
* 套接字类型：面向连接的套接字在通讯前一定要建立一条连接，可提供顺序的、可靠的、不会重复的数据传输，传输过程会进行数据的拆分、重组---TCP协议，要创建TCP连接套接字类型必须为SOCK\_STREAM
* 无连接类型套接字：实现这种连接的主要协议为UDP，套接字类型SOCK\_DGRAM

# 数据库

分库、分表

主从同步（微博 读写）

缓存cache 到内存（reddis）---

负载均衡（不同的请求，分配到不同的服务器）；硬件均衡、软件均衡（nginx、反向代理、正向代理）

# XML

* Extensive Markup Language
* 用处：配置文件
* DOM：Document Object Model

## Python xml模块

### 方法一：

from xml.parsers.expat import ParserCreate

**parser=ParserCreate()**

***dir(parser)***

*handler = BaiduWeatherSaxHandler()*

*parser.returns\_unicode = True*

*parser.StartElementHandler = handler.start\_element*

*parser.EndElementHandler = handler.end\_element*

*parser.CharacterDataHandler = handler.char\_data*

*parser.Parse(xml)*

['AttlistDeclHandler', **'CharacterDataHandler'**, 'CommentHandler', 'CurrentByteIndex', 'CurrentColumnNumber', 'CurrentLineNumber', 'DefaultHandler', 'DefaultHandlerExpand', 'ElementDeclHandler', 'EndCdataSectionHandler', 'EndDoctypeDeclHandler', **'EndElementHandler'**, 'EndNamespaceDeclHandler', 'EntityDeclHandler', 'ErrorByteIndex', 'ErrorCode', 'ErrorColumnNumber', 'ErrorLineNumber', 'ExternalEntityParserCreate', 'ExternalEntityRefHandler', 'GetBase', 'GetInputContext', 'NotStandaloneHandler', 'NotationDeclHandler', **'Parse'**, 'ParseFile', 'ProcessingInstructionHandler', 'SetBase', 'SetParamEntityParsing', 'SkippedEntityHandler', 'StartCdataSectionHandler', 'StartDoctypeDeclHandler', **'StartElementHandler'**, 'StartNamespaceDeclHandler', 'UnparsedEntityDeclHandler', 'UseForeignDTD', 'XmlDeclHandler', 'buffer\_size', 'buffer\_text', 'buffer\_used', 'intern', 'namespace\_prefixes', 'ordered\_attributes', **'returns\_unicode'**, 'specified\_attributes']

### 方法二：

*# 创建一个 XMLReader*

*parser = xml.sax.make\_parser()*

*# turn off namepsaces关闭命名空间*

*parser.setFeature(xml.sax.handler.feature\_namespaces, 0)*

*# 重写 ContextHandler*

*Handler = MovieHandler()*

*parser.setContentHandler( Handler )*

*parser.parse("movies.xml")*

class MovieHandler( xml.sax.ContentHandler ):

#继承xml.sax.ContentHandler类，重写父类的三个方法，tag为XML标签，attrs为标签的属性，#content为所显示的文本字段

def startElement(self,tag,attrs):

if tag==”xx”:

pass

def endElement(self,tag):

pass

def characters(self,content):

if self.currentData==’xx’:

print content

# HTMLParser

from HTMLParser import HTMLParser

#自定义类继承HTMLParser，重写父类的方法[根据实际情况，重写需要的方法]；parser.feed(pyhtml)解析HTML

*def handle\_starttag(self, tag, attrs):*

*print('<%s>' % tag,attrs)*

*def handle\_endtag(self, tag):*

*print('</%s>' % tag)*

*def handle\_startendtag(self, tag, attrs):*

*print('<%s/>' % tag)*

*def handle\_data(self, data):*

*print('data:',data)*

*def handle\_comment(self, data):*

*print('<!-- -->')*

*def handle\_entityref(self, name):*

*print('&%s;' % name)*

*def handle\_charref(self, name):*

*print('&#%s;' % name)*

# 服务器端测试

功能测试：

容错：

稳定性：

性能测试：

灾备：

恢复：

算法、配置：时间复杂度、耗时

# 职场软技能

## 考核目标

1. bug数
2. 上线质量
3. 考勤
4. 主动性：多接活、帮助别人、多加班（比领导晚走一点就好）
5. 技术提升：（组内分享）自动化、性能、编程、技术攻关
6. 引入新的工具，优化流程
7. 测试的进度和团队协作

## 项目管理

* 项目：项目是为创造独特的产品、服务或成果而进行的临时性工作。
* 产品：产品是“一组将输入转化为输出的相互关联或相互作用的活动”的结果，即“过程”的结果。
* 项目管理、部门管理
* PMP：Project Management Professional 项目管理专业人员
* WBS：work breakdown structure 工作分解结构

了解项目的生命周期(开发周期、测试周期)

项目计划（了解需求、概要设计、详细设计、预估测试方案，有多少测试模块，每个模块有多少工作量，测试用例编写评审、测试工具、人力分配）

执行测试，第一轮：详细测试所有用例；第二轮：测试bug回归；第三轮：验证bug是否解决

灰度发布、预发布出测试报告

## 面试

* 中小企业：要求直接上手工作，独立负责
* 大公司：年轻人，重培养，看重潜力（快速学习能力）
* 学历、经历
* 独立完成自己的项目
* 编程、Linux、数据库、http
* 接口自动化（自己的作品）、如何做性能测试
* **面试前一定要做好充分准备：编程算法能力、复习笔记、书；面试题准备、（自己比别人的优势在哪，根据具体的工作经历有条理的阐述）；功能测试3剑客；每个点半小时！**

# 自动化测试

Google测试之道、软件测试架构

在线监控：生产环境，快速发现问题，通知开发相关人员（邮件、短信）

局部自动化：写小脚本，收集脚本复用。

获取操作元素---操作---获取结果---结果比对

UI自动化测试框架、接口自动化、性能自动化测试

评估项目产品质量：千行代码bug率，重大缺陷、上线质量

## 自动化测试需要的条件

软件需求变动不频繁

项目周期足够长

自动化测试脚本可重复利用

## chrome webdrive

ChromeDriver目录加入到环境变量

vim ~/.bash\_profile

export PATH=$PATH:ChromeDriver目录(注意，是存放的目录，不是这个文件)

:wq

source ~/.bash\_profile

## xpath

// 从目录树的任意位置查找

//book 标签查找

//book[1] 标签的第一个元素

//book[last()] 最后一个元素

//book[@category=’cooking’] 根据标签的属性查找 ，@--属性

// book[@category=’cooking’ and @cover=’paperback’] 表达式多属性查找；or !

//book/title[contains(text(),'Potter')] 模糊匹配—包含文本Potter，等价contains(.,’Potter’)

//book/year[text()='2005'] 或者大于、小于

/bookstore/book/title[1] 绝对位置查找

/bookstore/book[1]/title/following-sibling::\* 轴某元素的后一级、

### XPath轴(XPath Axes)可定义某个相对于当前节点的节点集：

1、child 选取当前节点的所有子元素

2、parent 选取当前节点的父节点

3、descendant 选取当前节点的所有后代元素（子、孙等）

4、ancestor 选取当前节点的所有先辈（父、祖父等）

5、descendant-or-self 选取当前节点的所有后代元素（子、孙等）以及当前节点本身

6、ancestor-or-self 选取当前节点的所有先辈（父、祖父等）以及当前节点本身

7、preceding-sibling 选取当前节点之前的所有同级节点

8、following-sibling 选取当前节点之后的所有同级节点

9、preceding 选取文档中当前节点的开始标签之前的所有节点

10、following 选取文档中当前节点的结束标签之后的所有节点

11、self 选取当前节点

12、attribute 选取当前节点的所有属性

13、namespace 选取当前节点的所有命名空间节点

frame中的元素必须先进入frame中才可定位到switch\_to\_frame()

# webdriver API

driver=webdriver.Firefox(executable\_path=’’)

## browser方法

['add\_cookie', 'application\_cache', 'back', 'capabilities', 'close', 'command\_executor', 'create\_options', 'create\_web\_element', 'current\_url', 'current\_window\_handle', 'delete\_all\_cookies', 'delete\_cookie', 'desired\_capabilities', 'error\_handler', 'execute', 'execute\_async\_script', 'execute\_script', 'file\_detector', 'file\_detector\_context', 'find\_element', 'find\_element\_by\_class\_name', 'find\_element\_by\_css\_selector', 'find\_element\_by\_id', 'find\_element\_by\_link\_text', 'find\_element\_by\_name', 'find\_element\_by\_partial\_link\_text', 'find\_element\_by\_tag\_name', 'find\_element\_by\_xpath', 'find\_elements', 'find\_elements\_by\_class\_name', 'find\_elements\_by\_css\_selector', 'find\_elements\_by\_id', 'find\_elements\_by\_link\_text', 'find\_elements\_by\_name', 'find\_elements\_by\_partial\_link\_text', 'find\_elements\_by\_tag\_name', 'find\_elements\_by\_xpath', 'forward', 'get', 'get\_cookie', 'get\_cookies', 'get\_log', 'get\_screenshot\_as\_base64', 'get\_screenshot\_as\_file', 'get\_screenshot\_as\_png', 'get\_window\_position', 'get\_window\_size', 'implicitly\_wait', 'launch\_app', 'log\_types', 'maximize\_window', 'mobile', 'name', 'orientation', 'page\_source', 'quit', 'refresh', 'save\_screenshot', 'service', 'session\_id', 'set\_page\_load\_timeout', 'set\_script\_timeout', 'set\_window\_position', 'set\_window\_size', 'start\_client', 'start\_session', 'stop\_client', 'switch\_to', 'switch\_to\_active\_element', 'switch\_to\_alert', 'switch\_to\_default\_content', 'switch\_to\_frame', 'switch\_to\_window', 'title', 'w3c', 'window\_handles']

driver.back() //后退

driver.forward() //前进

driver.close() //关闭窗口

driver.get(url) //打开网页

driver.get\_screenshot\_as\_file(filepath) //截图

driver.get\_cookies() //获取所有cookie字典

driver.maximize\_window() //最大化窗口

driver.get\_window\_size() //获取窗口大小

driver.get\_window\_position() //获取窗口位置

driver.set\_window\_position(x,y) //设置窗口位置

driver.set\_window\_size(x,y) //设置窗口大小

driver.execute\_script(“js code”) //可执行js脚本

[driver.execute\_script('document.getElementById("kw").value="python"')]

driver.refresh() //刷新

driver.get\_elements\_by\_tag\_name(‘’)

driver.get\_element\_by\_xpath()

driver.find\_element\_by\_link\_text(‘’)

driver.find\_element\_by\_partial\_link\_text(‘’)

driver.find\_elements\_by\_tag\_name(‘’)

driver.switch\_to() //切换

driver.switch\_to\_alert() //切换到弹框

driver.switch\_to\_frame() //

driver.switch\_to\_window(‘2147483662’) //切换tab页面 driver.switch\_to\_window(driver.window\_handles[1])

driver.implicitly\_wait(3) //

driver.set\_page\_load\_timeout(5) //设置网页加载超时时间，需捕获异常处理

driver.set\_script\_timeout //设置脚本执行超时时间

## 属性

driver.page\_source //网页源码

driver.title //网页标题

driver.name //浏览器的名字，Firefox

driver.desired\_capabilities //浏览器的设置属性

driver.current\_window\_handle //获取当前窗口句柄

driver.window\_handles //返回一个包含所有窗口标识的list

driver.service //获取一个服务对象

## element方法

element=driver.find\_element\_by\_id(‘’)

element.find\_element\_by\_xx(‘’)

element.clear() //清空

element.click() //点击

element.send\_keys()

element.submit() //

element.is\_displayed() //是否可见

element.is\_enabled() //返回True,是否可点击 ;js控制

>>> browser.execute\_script("document.getElementById('kw').disabled=true;")

element.is\_selected() //返回布尔型，是否已选择

element.value\_of\_css\_property(‘height’) //width 获取元素的宽高,css属性

element.screenshot(‘1.png’) //保存元素截图

element.get\_attribute(‘value’) //HTML中的属性value，其他属性有【id name class type href】

element.get\_property(‘value’) //HTML中的属性value

## 属性

element.text //指标签中的文本内容<a href=’’>text</a>

element.location //返回元素坐标

element.size //返回元素的大小

element.parent //返回<class 'selenium.webdriver.firefox.webdriver.WebDriver'>

element.tag\_name //元素的标签

## driver.switch\_to

['active\_element', 'alert', 'default\_content', 'frame', 'parent\_frame', 'window']

## 需要用到的包

from selenium import webdriver

### ActionChains

//鼠标操作【单击、双击、拖动、移动到】

from selenium.webdriver import ActionChains //

['click', 'click\_and\_hold', 'context\_click', 'double\_click', 'drag\_and\_drop', 'drag\_and\_drop\_by\_offset', 'key\_down', 'key\_up', 'move\_by\_offset', 'move\_to\_element', 'move\_to\_element\_with\_offset', 'perform', 'release', 'send\_keys', 'send\_keys\_to\_element']

ActionChains can be used in a chain pattern::

|

| menu = driver.find\_element\_by\_css\_selector(".nav")

| hidden\_submenu = driver.find\_element\_by\_css\_selector(".nav #submenu1")

|

| ActionChains(driver).move\_to\_element(menu).click(hidden\_submenu).perform()

|

| Or actions can be queued up one by one, then performed.::

|

| menu = driver.find\_element\_by\_css\_selector(".nav")

| hidden\_submenu = driver.find\_element\_by\_css\_selector(".nav #submenu1")

|

| actions = ActionChains(driver)

| actions.move\_to\_element(menu)

| actions.click(hidden\_submenu)

| actions.perform()

ActionChains(driver).key\_down(Keys.CONTROL).send\_keys('c').key\_up(Keys.CONTROL).perform()

### Select

//操作下拉选项框

from selenium.webdriver.support.ui import Select //

['all\_selected\_options', 'deselect\_all', 'deselect\_by\_index', 'deselect\_by\_value', 'deselect\_by\_visible\_text', 'first\_selected\_option', 'options', 'select\_by\_index', 'select\_by\_value', 'select\_by\_visible\_text']

### Keys

//操作系统按键

from selenium.webdriver.common.keys import Keys //

['ADD', 'ALT', 'ARROW\_DOWN', 'ARROW\_LEFT', 'ARROW\_RIGHT', 'ARROW\_UP', 'BACKSPACE', 'BACK\_SPACE', 'CANCEL', 'CLEAR', 'COMMAND', 'CONTROL', 'DECIMAL', 'DELETE', 'DIVIDE', 'DOWN', 'END', 'ENTER', 'EQUALS', 'ESCAPE', 'F1', 'F10', 'F11', 'F12', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'HELP', 'HOME', 'INSERT', 'LEFT', 'LEFT\_ALT', 'LEFT\_CONTROL', 'LEFT\_SHIFT', 'META', 'MULTIPLY', 'NULL', 'NUMPAD0', 'NUMPAD1', 'NUMPAD2', 'NUMPAD3', 'NUMPAD4', 'NUMPAD5', 'NUMPAD6', 'NUMPAD7', 'NUMPAD8', 'NUMPAD9', 'PAGE\_DOWN', 'PAGE\_UP', 'PAUSE', 'RETURN', 'RIGHT', 'SEMICOLON', 'SEPARATOR', 'SHIFT', 'SPACE', 'SUBTRACT', 'TAB', 'UP']

### WebDriverWait

//显示等待

from selenium.webdriver.support.ui import WebDriverWait

class WebDriverWait

\_\_init\_\_(self, driver, timeout, poll\_frequency=0.5, ignored\_exceptions=None)

**['until', 'until\_not']**

*until(self, method, message='') unbound selenium.webdriver.support.wait.WebDriverWait method*

*Calls the method provided with the driver as an argument until the return value is not False.*

**usage：**

*# 通过显示等待方式获取页面元素*

*element = WebDriverWait(driver, 10).until(lambda x: \*

*x.find\_element(locatorMethod, locatorExpression)) #util中必须为可调用的函数x---driver*

*#等待直到某字符出现在页面源码中*

*WebDriverWait(browser,10).until(lambda x:u'百度' in x.page\_source)*

### By

//通过何种方式定位页面元素

from selenium.webdriver.common.by import By

['CLASS\_NAME', 'CSS\_SELECTOR', 'ID', 'LINK\_TEXT', 'NAME', 'PARTIAL\_LINK\_TEXT', 'TAG\_NAME', 'XPATH']

### EC

//期望某种条件出现，与显示等待合用

from selenium.webdriver.support import expected\_conditions as EC

**与WebDriverWait合用，WebDriverWait.until(EC.element\_to\_be\_selected(element))**

['NoAlertPresentException', 'NoSuchElementException', 'NoSuchFrameException', 'StaleElementReferenceException', 'WebDriverException', 'alert\_is\_present', 'element\_located\_selection\_state\_to\_be', 'element\_located\_to\_be\_selected', 'element\_selection\_state\_to\_be', '**element\_to\_be\_clickable**', '**element\_to\_be\_selected**', 'frame\_to\_be\_available\_and\_switch\_to\_it', 'invisibility\_of\_element\_located', 'new\_window\_is\_opened', 'number\_of\_windows\_to\_be', 'presence\_of\_all\_elements\_located', 'presence\_of\_element\_located', 'staleness\_of', 'text\_to\_be\_present\_in\_element', 'text\_to\_be\_present\_in\_element\_value', 'title\_contains', 'title\_is', 'visibility\_of', 'visibility\_of\_any\_elements\_located', 'visibility\_of\_element\_located']

from selenium.common.exceptions import TimeoutException, NoSuchElementException

### Option

from selenium.webdriver.chrome.options import Options

//设置浏览器的启动选项

*添加对浏览器的设置，【禁用插件、禁止证书弹出、默认启动最大化】*

*chrome\_optins=Option()*

*# 向Options实例中添加禁用扩展插件的设置参数项*

*chrome\_options.add\_argument("--disable-extensions")*

*# 添加屏蔽--ignore-certificate-errors提示信息的设置参数项*

*chrome\_options.add\_experimental\_option("excludeSwitches", ["ignore-certificate-errors"])*

*# 添加浏览器最大化的设置参数项，已启动就最大化*

*chrome\_options.add\_argument('--start-maximized')*

*# 设置Chrome浏览器禁用PDF和Flash插件,把图片也关掉了。*

*profile = {"plugins.plugins\_disabled": ['Chrome PDF Viewer'],*

*"plugins.plugins\_disabled": ['Adobe Flash Player'],*

*"profile.managed\_default\_content\_settings.images":2}*

*chrome\_options.add\_experimental\_option("prefs", profile)*

['add\_argument', 'add\_encoded\_extension', 'add\_experimental\_option', 'add\_extension', 'arguments', 'binary\_location', 'debugger\_address', 'experimental\_options', 'extensions', 'to\_capabilities']

### DesiredCapabilities

*caps = DesiredCapabilities.INTERNETEXPLORER*

*# 将忽略IE保护模式的参数设置为True*

*caps['ignoreProtectedModeSettings'] = True*

from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities

['ANDROID', 'CHROME', 'EDGE', 'FIREFOX', 'HTMLUNIT', 'HTMLUNITWITHJS', 'INTERNETEXPLORER', 'IPAD', 'IPHONE', 'OPERA', 'PHANTOMJS', 'SAFARI']

### FirefoxProfile

webdriver.FirefoxProfile

webdriver.firefox.firefox\_profile.FirefoxProfile //**这两个有什么不同？**

*Mac 安装路径：~/Library/Application Support/Google*

*Windows 安装路径：C:\Users\liudongjie\AppData\Roaming\Mozilla\Firefox\Profiles\g6m1cswj.default*

*//设置浏览器启动配置文件*

*profilePath=r’ C:\Users\liudongjie\AppData\Roaming\Mozilla\Firefox\Profiles\g6m1cswj.default’*

*profile = webdriver.firefox.firefox\_profile.FirefoxProfile(profilePath)*

*# 将添加了新配置文件的Firefox浏览器首页设为百度主页，*

*profile.set\_preference("browser.startup.homepage", "http://www.baidu.com")*

*# 设置启动浏览器的同时主页不为空白页*

*profile.set\_preference("browser.startup.page", 1)*

*# 自动打开firebug*

*profile.set\_preference("extensions.firebug.allPagesActivation", "on")*

*# 启用firebug网络面板功能*

*profile.set\_preference("extensions.firebug.net.enableSites", True)*

*# 启用firebug Cookies面板功能*

*profile.set\_preference("extensions.firebug.cookies.enableSites", True)*

['accept\_untrusted\_certs', 'add\_extension', 'assume\_untrusted\_cert\_issuer', 'encoded', 'native\_events\_enabled', 'path', 'port', 'set\_preference', 'set\_proxy', 'update\_preferences']

### ChromeOption

*//模拟Android4.0.2*

*options.add\_argument(*

*'--user-agent=Mozilla/5.0 (Linux; U; Android 4.0.2; en-us; Galaxy Nexus Build/ICL53F) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30')*

*//模拟iPhone*

*options.add\_argument(*

*'--user-agent=Mozilla/5.0 (iPhone; CPU iPhone OS 5\_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A334 Safari/7534.48.3')*

*//模拟iPad*

*options.add\_argument(*

*'--user-agent=Mozilla/5.0 (iPad; CPU OS 5\_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A334 Safari/7534.48.3')*

*//设置下载文件默认路径*

*# 创建Chrome浏览器配置对象实例*

*chromeOptions = webdriver.ChromeOptions()*

*# 设定下载文件的保存目录为C盘的iDownload目录，如果该目录不存在，将会自动创建*

*prefs = {"download.default\_directory": "/Users/ralphliu/Document/learn-python/python-code/iDownload"}*

*# 将自定义设置添加到Chrome配置对象实例中*

*chromeOptions.add\_experimental\_option("prefs", prefs)*

*chrome=webdriver.Chrome(executable=’’ ,chrome\_options=chromeOptions)*

*#Firefox下载文件前的设置*

*# 创建一个FirefoxProfile实例，用于存放自定义配置*

*profile = webdriver.FirefoxProfile()*

*# 指定下载路径，默认只会自动创建一级目录，如果指定了*

*# 多级不存在的目录，将会下载到默认路径*

*profile.set\_preference('browser.download.dir', 'd:\\iDownload')*

*# 将browser.download.folderList设置为2，表示将文件下载到指定路径*

*# 设置成2表示使用自定义下载路径；*

*# 设置成0表示下载到桌面；设置成1表示下载到默认路径*

*profile.set\_preference('browser.download.folderList', 2)*

*# browser.helperApps.alwaysAsk.force对于未知的 MIME 类型文件会弹出窗口*

*# 让用户处理，默认值为true，设定为False表示不会记录打开未知 MIME 类型*

*# 文件的方式*

*profile.set\_preference("browser.helperApps.alwaysAsk.force", False)*

*# 在开始下载时是否显示下载管理器*

*profile.set\_preference('browser.download.manager.showWhenStarting',\*

*False)*

*# 设定为 False 会把下载框进行隐藏*

*profile.set\_preference("browser.download.manager.useWindow", False)*

*# 默认值为 true，设定为 False 表示不获取焦点*

*profile.set\_preference("browser.download.manager. focusWhenStarting",\*

*False)*

*# 下载.exe文件弹出警告，默认值是 true，设定为False 则不会弹出警告框*

*profile.set\_preference("browser.download.manager.alertOnEXEOpen",\*

*False)*

*# browser.helperApps.neverAsk.openFile表示直接打开下载文件，不显示确认框*

*# 默认值为空字符串，下行代码行设定了多种文件的 MIME类型，*

*# 例如application/exe，表示.exe类型的文件，*

*# application/excel表示 Excel 类型的文件*

*profile.set\_preference("browser.helperApps.neverAsk.openFile", \*

*"application/pdf")*

*# 对所给出文件类型不再弹出框进行询问，直接保存到本地磁盘*

*profile.set\_preference('browser.helperApps.neverAsk.saveToDisk', \*

*'application/zip, application/octet-stream')*

*# browser.download.manager.showAlertOnComplete设定下载文件结束后是否显示下*

*#载完成提示框，默认为true，设定为False表示下载完成后不显示下载完成提示框*

*profile.set\_preference("browser.download.manager. showAlertOnComplete",\*

*False);*

*# browser.download.manager.closeWhenDone设定下载结束后是否自动*

*# 关闭下载框，默认值为true，设定为False 表示不关闭下载管理器*

*profile.set\_preference("browser.download.manager.closeWhenDone",\*

*False)*

*self.driver = webdriver.Firefox(executable\_path=“ ",firefox\_profile = profile)*

## JS语句

*execute\_script(self, script, \*args)*

*:Args:*

*- script: The JavaScript to execute.*

*- \\*args: Any applicable arguments for your JavaScript.*

*:Useage: //videoPlayer为return语句中传入的参数*

*videoSrc = self.driver.execute\_script ("return arguments[0].currentSrc;", videoPlayer)*

//画一个红色的图案

var c = document.getElementById('myCanvas');

var cxt=c.getContext('2d');

cxt.fillStyle='#FF0000';

cxt.fillRect(0,0,150,150);

//获取视频播放地址

return arguments[0].currentSrc;

//获取视频播放时长

return arguments[0].duration;

//播放视频

return arguments[0].play();

//暂停播放

return arguments[0].pause();

//定位正文输入框，输入文本【<b>邮件的正文内容<b>】<b>表示加粗显示

document.getElementsByTagName('body')[0].innerHTML='<b>邮件的正文内容<b>;'

//设置元素样式

arguments[0].setAttribute('style',arguments[1]); ,element, ”background:green;border:2px solid red;”

# 通过执行Javascript来stop加载，然后继续执行后续动作

self.driver.execute\_script('window.stop()')

# 构造JavaScript查找百度首页的搜索输入框的代码字符串

searchInputBoxJS = "document.getElementById('query').value='光荣之路';"

# 构造JavaScript查找百度首页的搜索按钮的代码字符串

searchButtonJS = "document.getElementById('stb').click()"

#滚动页面到底部

self.driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

self.driver.execute\_script("window.scrollTo(0, 0);") #滚动到顶部

#每次滚动页面100像素

self.driver.execute\_script('window.scrollBy(0,100);')

#将被遮挡的元素滚动到可见屏幕上

# scrollIntoView(true)表示将元素滚到屏幕中间

# scrollIntoView(false)表示将元素滚动屏幕底部

self.driver.execute\_script("document.getElementsByTagName('a')[500].scrollIntoView(true);")

#通过js给元素属性赋值，elementObj.attributeName=value

driver.execute\_script("arguments[0].%s=arguments[1]" %attributeName,elementObj,value) #第一种方式

driver.execute\_script("arguments[0].setAttribute(arguments[1],arguments[2])",elementObj, attributeName, value) #第二种方式

#删除元素属性

driver.execute\_script("arguments[0].removeAttribute(arguments[1])", elementObj,attributeName)

#给元素赋值

document.getElementById('kw').value='hello';

#获取元素的值

return document.getElementById('query').value;

# Python logging模块

Import logging

## 对象：

['BASIC\_FORMAT', 'BufferingFormatter', 'CRITICAL', 'DEBUG', 'ERROR', 'FATAL', 'FileHandler', 'Filter', 'Filterer', 'Formatter', 'Handler', 'INFO', 'LogRecord', 'Logger', 'LoggerAdapter', 'Manager', 'NOTSET', 'NullHandler', 'PlaceHolder', 'RootLogger', 'StreamHandler', 'WARN', 'WARNING']

## 方法：

['addLevelName', 'atexit', 'basicConfig', 'cStringIO', 'captureWarnings', 'codecs', 'collections', 'critical', 'currentframe', 'debug', 'disable', 'error', 'exception', 'fatal', 'getLevelName', 'getLogger', 'getLoggerClass', 'info', 'log', 'logMultiprocessing', 'logProcesses', 'logThreads', 'makeLogRecord', 'os', 'raiseExceptions', 'root', 'setLoggerClass', 'shutdown', 'sys', 'thread', 'threading', 'time', 'traceback', 'warn', 'warning', 'warnings', 'weakref']

## logging.Logger

['addFilter', 'addHandler', 'callHandlers', 'critical', 'debug', 'error', 'exception', 'fatal', 'filter', 'findCaller', 'getChild', 'getEffectiveLevel', 'handle', 'info', 'isEnabledFor', 'log', 'makeRecord', 'manager', 'removeFilter', 'removeHandler', 'root', 'setLevel', 'warn', 'warning']

## logging.config

['BaseConfigurator', 'ConvertingDict', 'ConvertingList', 'ConvertingMixin', 'ConvertingTuple', 'DEFAULT\_LOGGING\_CONFIG\_PORT', 'DictConfigurator', 'IDENTIFIER', 'RESET\_ERROR', 'StreamRequestHandler', 'ThreadingTCPServer']

**方法：**

[cStringIO', 'dictConfig', 'dictConfigClass', 'errno', 'fileConfig', 'io', 'listen', 'logging', 'os', 're', 'socket', 'stopListening', 'struct', 'sys', 'thread', 'threading', 'traceback', 'types', 'valid\_ident']

# ConfigParser

## dir(ConfigParser.ConfigParser)

['OPTCRE', 'OPTCRE\_NV', 'SECTCRE', 'add\_section', 'defaults', 'get', 'getboolean', 'getfloat', 'getint', 'has\_option', 'has\_section', 'items', 'options', 'optionxform', 'read', 'readfp', 'remove\_option', 'remove\_section', 'sections', 'set', 'write']

#coding=utf-8

import ConfigParser

cf=ConfigParser.ConfigParser()

cf.read('uiObjMapPath.ini') #读取配置文件

print cf.sections() #获取所有的section

print cf.options('db') #获取section下的所有配置项

print cf.items('db') #获取section下的配置项和对应的值

num= cf.getint('sogou','num') #返回int类型的值

print type(num)

cf.remove\_option('gloryroad',cf.options('gloryroad')[0]) #移除section下某一项配置

cf.remove\_section('web') #移除某一项section ，移除操作完成后需要write

cf.write(open('uiObjMapPath.ini','w'))

cf.set('sogou','num',5678) #修改配置项

cf.write(open('uiObjMapPath.ini','w')) #

cf.add\_section('db') #添加配置项

cf.set('db','host','127.0.0.1') #每次修改、添加、移除操作完成后都需要写操作write

cf.set('db','user','ralph')

cf.write(open('uiObjMapPath.ini','w'))

print cf.has\_section('sogou')

print cf.has\_option('sogou','num')

## 获取文件路径

os.path.dirname(os.path.abspath(\_\_file\_\_)) #\_\_file\_\_ 只在类中使用，表示获取文件的当前路径

# autoit

脚本解释：

Send("c:\test.txt")表示使用键盘输入“c:\test.txt”。

Send("{ENTER}")表示按ENTER键。

调用两次Enter键，主要是解决某些操作系统默认的输入法是中文输入法，输入“c:\test.txt”以后，

必须按一下Enter键才能将输入的内容写入路径输入框中，再按一次Enter键，就等价于单击文件打开窗体的“打开”按钮。

（3）将AutoIt脚本保存为文件名为“test.au3”的文件并存放在D盘驱动器中。

（4）“开始”→“所有程序”→“AutoIt v3” →“Compile script to.exe(x64)”（根据自己的操作系统位数选择正确的位数），

调出将AutoIt脚本转换成exe文件的界面，如图11-11所示。

图11- 11

（5）在Source路径框中选择上面保存的AutoIt脚本“test.au3”文件，Destination处选择.exe单选项，并在接下来的输入框中设置好生成exe文件的保存路径，其他默认即可。

（6）单击Convert按钮，将会把AutoIt脚本“test.au3”文件转换成“test.exe”可执行文件

# from PIL import Image

## 应用：

*用于对两张图片通过像素比对的算法，获取文件像素个数大小一一比对，计算出比对结果的相似度*

## 对象：

['ADAPTIVE', 'AFFINE', 'ANTIALIAS', 'BICUBIC', 'BILINEAR', 'CONTAINER', 'CUBIC', 'DEBUG', 'EXTENSION', 'EXTENT', 'FLIP\_LEFT\_RIGHT', 'FLIP\_TOP\_BOTTOM', 'FLOYDSTEINBERG', 'ID', 'Image', 'ImageMode', 'ImagePalette', 'ImagePointHandler', 'ImageTransformHandler', 'IntType', 'LINEAR', 'MESH', 'MIME', 'MODES', 'NEAREST', 'NONE', 'NORMAL', 'OPEN', 'ORDERED', 'PERSPECTIVE', 'QUAD', 'RASTERIZE', 'ROTATE\_180', 'ROTATE\_270', 'ROTATE\_90', 'SAVE', 'SEQUENCE', 'StringType', 'TupleType', 'UnicodeStringType', 'VERSION', 'WEB']

## 方法：

['blend', 'byteorder', 'composite', 'core', 'eval', 'fromarray', 'frombuffer', 'fromstring', 'getmodebandnames', 'getmodebands', 'getmodebase', 'getmodetype', 'init', 'isDirectory', 'isImageType', 'isNumberType', 'isSequenceType', 'isStringType', 'isTupleType', 'merge', 'new', 'open', 'os', 'preinit', 'register\_extension', 'register\_mime', 'register\_open', 'register\_save', 'string', 'sys', 'warnings']

## png=Image.open(file)

[category', 'convert', 'copy', 'crop', 'decoderconfig', 'decodermaxblock', 'draft', 'filename', 'filter', 'format', 'format\_description', 'fp', 'fromstring', 'getbands', 'getbbox', 'getcolors', 'getdata', 'getextrema', 'getim', 'getpalette', 'getpixel', 'getprojection', 'histogram', 'im', 'info', 'load', 'load\_end', 'load\_prepare', 'load\_read', 'mode', 'offset', 'palette', 'paste', 'png', 'point', 'putalpha', 'putdata', 'putpalette', 'putpixel', 'quantize', 'readonly', 'resize', 'rotate', 'save', 'seek', 'show', 'size', 'split', 'tell', 'text', 'thumbnail', 'tile', 'tobitmap', 'tostring', 'transform', 'transpose', 'verify']

# unittest

## 跳过测试用执行

@unittest.skip("skipping") # 无条件忽略该测试方法

# 如果变量a > 5，则忽略该测试方法

@unittest.skipIf(a > 5, "condition is not satisfied!")

# 除非执行测试用例的平台是Linux平台，否则忽略该测试方法

@unittest.skipUnless(sys.platform.startswith("linux"), "requires Linux")

## 导入测试用例

### unittest.TestLoader导入方式

*This class is responsible for loading tests according to various criteria*

*| and returning them wrapped in a TestSuite*

*| discover(self, start\_dir, pattern='test\*.py', top\_level\_dir=None)*

*| Find and return all test modules from the specified start*

*| directory, recursing into subdirectories to find them. Only test files*

*| that match the pattern will be loaded. (Using shell style pattern*

*| matching.)*

*|*

*| All test modules must be importable from the top level of the project.*

*| If the start directory is not the top level directory then the top*

*| level directory must be specified separately.*

*|*

*| If a test package name (directory with '\_\_init\_\_.py') matches the*

*| pattern then the package will be checked for a 'load\_tests' function. If*

*| this exists then it will be called with loader, tests, pattern.*

*|*

*| If load\_tests exists then discovery does \*not\* recurse into the package,*

*| load\_tests is responsible for loading all tests in the package.*

*|*

*| The pattern is deliberately not stored as a loader attribute so that*

*| packages can continue discovery themselves. top\_level\_dir is stored so*

*| load\_tests does not need to pass this argument in to loader.discover().*

['discover', 'getTestCaseNames', 'loadTestsFromModule', 'loadTestsFromName', 'loadTestsFromNames', 'loadTestsFromTestCase', 'sortTestMethodsUsing', 'suiteClass', 'testMethodPrefix']

### unittest.TestSuite执行测试

*A test suite is a composite test consisting of a number of TestCases.*

*| For use, create an instance of TestSuite, then add test case instances.*

*| When all tests have been added, the suite can be passed to a test*

*| runner, such as TextTestRunner. It will run the individual test cases*

*| in the order in which they were added, aggregating the results. When*

*| subclassing, do not forget to call the base class constructor.*

*\_\_init\_\_(self, tests=())*

['addTest', 'addTests', 'countTestCases', 'debug', 'run']

### unittest.TextTestRunner

*class TextTestRunner(\_\_builtin\_\_.object)*

*| A test runner class that displays results in textual form.*

*|*

*| It prints out the names of tests as they are run, errors as they*

*| occur, and a summary of the results at the end of the test run.*

*| Methods defined here:*

*| \_\_init\_\_(self, stream=<open file '<stderr>', mode 'w'>, descriptions=True, verbosity=1, failfast=False, buffer=False, resultclass=None)*

*|*

*| run(self, test)*

*| Run the given test case or test suite.*

*| ----------------------------------------------------------------------*

# 根据给定的测试类，获取其中的所有以“test”开头的测试方法，并返回一个测试套件

suite1 = unittest.TestLoader().loadTestsFromTestCase(

TestSequenceFunctions)

suite2 = unittest.TestLoader().loadTestsFromTestCase(

TestDictValueFormatFunctions)

# 将多个测试类加载到测试套件中

suite = unittest.TestSuite([suite2, suite1]) #通过调整suit2和suite1的顺序，可以设定执行顺序

# 设置verbosity = 2，可以打印出更详细的执行信息

unittest.TextTestRunner(verbosity = 2).run(suite)

## 单步执行

进入单元测试用例所在的目录：

* 执行某个模块的所有测试用例：

python -m unittest Test\_Calc\_by\_specific\_order

* 执行某个模块的：

python -m unittest Test\_Calc\_by\_specific\_order.MyTest

* 执行某个模块的某个测试方法：

python -m unittest Test\_Calc\_by\_specific\_order.MyTest.test\_add

* 执行某个目录的全部测试用例：(注意所有测试方法名称前须有test关键字)进入test目录：

执行python -m unittest discover -v

或者指定某个目录来执行所有单元测试用例：python -m unittest discover -v -s e:\单元测试\test

## unittest.TestCase

['addCleanup', 'addTypeEqualityFunc', 'assertAlmostEqual', 'assertAlmostEquals', 'assertDictContainsSubset', 'assertDictEqual', 'assertEqual', 'assertEquals', 'assertFalse', 'assertGreater', 'assertGreaterEqual', 'assertIn', 'assertIs', 'assertIsInstance', 'assertIsNone', 'assertIsNot', 'assertIsNotNone', 'assertItemsEqual', 'assertLess', 'assertLessEqual', 'assertListEqual', 'assertMultiLineEqual', 'assertNotAlmostEqual', 'assertNotAlmostEquals', 'assertNotEqual', 'assertNotEquals', 'assertNotIn', 'assertNotIsInstance', 'assertNotRegexpMatches', 'assertRaises', 'assertRaisesRegexp', 'assertRegexpMatches', 'assertSequenceEqual', 'assertSetEqual', 'assertTrue', 'assertTupleEqual', 'assert\_', 'countTestCases', 'debug', 'defaultTestResult', 'doCleanups', 'fail', 'failIf', 'failIfAlmostEqual', 'failIfEqual', 'failUnless', 'failUnlessAlmostEqual', 'failUnlessEqual', 'failUnlessRaises', 'failureException', 'id', 'longMessage', 'maxDiff', 'run', 'setUp', 'setUpClass', 'shortDescription', 'skipTest', 'tearDown', 'tearDownClass']

# 代码覆盖率统计：

安装coverage：pip install coverage

coverage run xx.py #运行程序

coverage report –m #查看报告

coverage html #在同级目录下生成HTML格式的测试报告

# Mysql

## 解释

* 外键为某个表中的一列，它包含另一个表 的主键值，定义了两个表之间的关系。

MySQL在运行时关联指定的每个表以处理联结。 这种处理可能是非常耗费资源的，因此应该仔细，不要联结 不必要的表。联结的表越多，性能下降越厉害

自联结：自联结通常作为外部语句用来替代 从相同表中检索数据时使用的子查询语句。虽然最终的结果是 相同的，但有时候处理联结远比处理子查询快得多。

*SELECT p1.prod\_id,p1.prod\_name*

*FROM products AS p1 , products AS p2*

*WHERE p1.vend\_id=p2.vend\_id*

*AND p2.prod\_id=’DTNTR’*

mysql -uroot -proot -P3306 -h127.0.0.1 --prompt \u@\h \d登录数据库，--prompt修改提示符

exit; QUIT; \Q; 退出MySQL

SHOW DATABASES; 显示所有的数据库

SHOW TABLES FROM XX; 显示某数据库中的表

SHOW COLUMNS FROM XX; DESC XX 显示表中的列

CREATE DATABASE NAME; 创建数据库

DROP DATABASE NAME; 删除数据库

USE TEST; 进入数据库test

SELECT VERSION(); 查询当前服务器版本

SELECT NOW(); 查询当前时间

SELECT DATABASE();查询当前使用的数据库

SELECT USER();查询当前用户

**表操作：**

1. ALTER TABLE tbl\_name ADD [COLUMN] col\_name column\_definition [FIRST | AFTER col\_name]; 增加列
2. ALTER TABLE tbl\_name DROP [COLUMN] col\_name; 删除列
3. ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...); 添加主键
4. ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [index\_name] [index\_type] (index\_col\_name,...)； 添加唯一约束
5. ALTER TABLE tbl\_name ALTER [COLUMN] col\_name {SET DEFAULT literal | DROP DEFAULT} 添加删除默认值

**关键词：**

NOT NULL ,UNSIGNE ,AUTO\_INCREMENT ,PRIMARY KEY ,UNIQUE KEY ,DEFAULT ,SET ,CHARSET ,FOREIGN KEY ,InnoDB ,REFERENCES ,

**字符型：**

CHAR(20) VARCHAR(20) INT(10) TINYINT ENUM(XX,XX,XX) SET(XX,XX,XX) FLOAT(8,2) DATA DATATIME

**创建表：**

CREATE TABLE `computer` (

`ID` int(11) NOT NULL AUTO\_INCREMENT,

`品牌` varchar(20) NOT NULL,

`CPU` enum('2g','4g','8g') DEFAULT NULL,

`尺寸` int(11) DEFAULT '0',

`价格` varchar(30) DEFAULT NULL,

`生产日期` date DEFAULT NULL,

`重量` varchar(10,2) DEFAULT NULL,

PRIMARY KEY (`ID`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

UPDATE zhuanti SET zhuanti\_link = (REPLACE(zhuanti\_link, 'https://www.9drug.com', 'http://pc.9drug.test'))

create table cump (id int(10) not null auto\_increment,

品牌 varchar(20) not null,

cpu enum('2G','4G','8G','16G') DEFAULT '2G',

尺寸 varchar(20) default '13.3',

primary key (id)) engine=InnoDB auto\_increment=1 default charset=utf8;

show variables like 'autocommit';

select count(id) c,cpu,品牌 from cump group by cpu having c=1;

左连接: 左表所有与右表满足条件的

右连接: 右表所有与左表满足条件的

例子，相信你一看就明白，不需要多说

A表(a1,b1,c1) B表(a2,b2)

a1 b1 c1 a2 b2

01 数学 95 01 张三

02 语文 90 02 李四

03 英语 80 04 王五

select A.\*,B.\* from A

left join B on(A.a1=B.a2)

结果是:

a1 b1 c1 a2 b2

01 数学 95 01 张三

02 语文 90 02 李四

03 英语 80 NULL NULL

select A.\*,B.\* from A

right join B on(A.a1=B.a2)

结果是:

a1 b1 c1 a2 b2

01 数学 95 01 张三

02 语文 90 02 李四

NULL NULL NULL 04 王五

表studentinfo：

create table studentInfo(

ID int not null auto\_increment comment "不为空的自增长的主键ID",

student\_id varchar(20) not null,

name varchar(30) not null,

sex char(4),

tel varchar(13) unique not null,

AdmissionDate datetime default '0000:00:00 00:00:00',

status tinyint(2) default 0,

primary key (ID),

unique student\_id(student\_id)

)engine=innodb character set utf8 comment "学生信息表";

insert studentinfo(student\_id,name,tel) values('1','战三','13311111111');

insert grade(course) values('语文');

表grade：

create table grade(

ID int auto\_increment not null,

stuID varchar(20),

course varchar(20) not null,

score tinyint(4) default 0,

primary key (ID),

key idx\_stuid(stuID),

CONSTRAINT FK\_ID FOREIGN KEY(stuID) REFERENCES studentInfo(student\_id)

)engine=innodb character set utf8 comment "学生成绩表";

内连接：select student\_id, course from studentinfo s,grade g where s.student\_id=g.stuId;

右连接：select s.student\_id,g.course from studentinfo s right join grade g on s.student\_id=g.stuid;

左连接：select s.student\_id,g.course from studentinfo s left join grade g on s.student\_id=g.stuid;

## 为知笔记部分

数据库操作：

SELECT VERSION(); 查询当前服务器版本

SELECT NOW(); 查询当前日期时间

SELECT USER(); 查询当前用户

SELECT DATABASE();

mysql -uroot -proot -P3306 -h127.0.0.1 --prompt \u@\h \d 登陆mysql，修改提示符

exit; quit; \q; 退出

增：CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name [DEFAULT] CHARACTER SET [=] charset\_name

删：DROP DATABASE db\_name;

改：ALTER DATABASE db\_name CHARACTER SET charset\_name;

查：显示所有的数据库 ：SHOW DATABASES;

字符型

1、CHAR(M)：定长类型，比如CHAR(5),如果我们只赋值abc，那么系统会自动在后面补两个空格0——255

2、VARCHAR(M)：变长类型，L+1个字节，其中L<=M且0<=M0——65535

3、ENUM（'value1','value2'...）枚举值，提供几个选项，选择其中一个，如：性别（'男','女','保密'）

4、SET('value1','value2'...) 最多64个成员，可以任选里面其中几个，任意排列组合

1、数据表（或表）是数据库最重要的组成部分之一，是其他对象的基础

2、表是一个二维表，行称为【记录】，列称为【字段】

二、创建数据表

1、首先打开数据库（数据库要存在）: USE 数据库名称

2、创建数据表：CREATE TABLE [IF NOT EXISTS] table\_name(column\_name(列名称) data\_type（列类型）,.... )

eg：CREATE TABLE tb1(username VARCHAR(20),userage TINYINT UNSIGNED,salary FLOAT(8,2) UNSIGNED,);

SHOW TABLES FROM XXX;

SHOW COLUMNS FROM table\_name;

INSERT [INTO] tb1\_name [(col\_name,...)] VALUES(val,...);

SELECT expr,...FROM tb1\_name

CREATE TABLE tb2(username VARCHAR(20) NOT NULL,age TINYINT UNSIGNE NULL);

自动编号必须与主键一起使用，每一个表只能有一个主键：AUTO\_INCREMENT PRIMARY KEY,

UNIQUE KEY 唯一约束、唯一约束可以保证记录的唯一性、唯一约束的字段可以为空值（NULL）、每张数据表可以存在多个唯一约束。

DEFAULT默认值：当插入记录时，如果没有明确为字段赋值，则自动赋予默认值。

CREATE TABLE tb6(

id SMALLINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(20) NOT NULL UNIQUE KEY,

sex ENUM('1','2','3') DEFAULT '3');

FOREIGN KEY：外键约束，外键约束的要求：<br>

1.父表和子表必须使用相同的存储引擎，而且禁止使用临时表<br>

2.数据表的存储引擎只能为InnoDB<br>

3.外键列和参照列必须具有相似的数据类型。其中数字的长度或是否有符号位必须相同；而字符的长度则可以不同（总之：数据类型必须相同，字符类型可以相似）<br>

4.外键列和参照列必须创建索引。如果外键列不存在索引，MySQL将自动创建索引

在 MY.ini 文件中编辑默认的存储引擎：default-storage-engine=INNODB；

外键约束

1. CASCADE: 从父表删除或更新且自动删除或更新子表中匹配的行。eg：FOREIGN KEY (pid) REFERENCES provinces (id) ON DELETE CASCADE

2. SET NULL: 从父表删除或更新行，并设置子表中的外键列为NULL。如果使用该选项，必须保证子表列没有指定NOT NULL。

3. RESTRICT: 拒绝对父表的删除或者更新操作。

4. NO ACTION: 标准SQL的关键字，在MySQL中与RESTRICT相同。

实际开发中，我们很少使用物理的外键约束，而是使用逻辑约束；物理的外键约束只有innoDB这种存储引擎才会支持，MYISAM这种引擎就不支持物理的外键约束。反过来说，当我们使用到的引擎为MYISAM时，只能使用逻辑外键（即两个表的设计的逻辑关系）。

1.表级约束与列级约束

（1）对一个数据列建立的约束，称为列级约束

（2）对多个数据列建立的约束，称为表级约束

（3）列级约束既可以在列定义时声明，也可以在列定以后声明

（4）表级约束只能在列定义后声明

往数据表中添加一列：ALTER TABLE tbl\_name ADD [COLUMN] col\_name column\_definition [FIRST | AFTER col\_name]

添加多列：ALTER TABLE tbl\_name ADD [COLUMN] (col\_name column\_definition,...)

删除列：ALTER TABLE tbl\_name DROP [COLUMN] col\_name

添加主键：ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...)

e.g:ALTER TABLE users ADD CONSTRAINT PK\_users\_id PRIMARY KEY (id);

2.添加唯一约束

ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [index\_name] [index\_type] (index\_col\_name,...)

e.g:ALTER TABLE users ADD UNIQUE (username);

3.添加外键约束

ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] FOREIGN KEY [index\_name](index\_col\_name,...) reference\_definition

e.g:ALTER TABLE users ADD FOREIGN KEY (pid) REFERENCES provinces (id)

4.添加/删除默认约束 DEFAULT

ALTER TABLE tbl\_name ALTER [COLUMN] col\_name {SET DEFAULT literal | DROP DEFAULT}

e.g:

ALTER TABLE users ADD age TINYINT UNSIGNED NULL;

ALTER TABLE users ALTER age SET DEFAULT 15;

ALTER TABLE users ALTER age DROP DEFAULT;

1.修改列定义

ALTER TABLE tbl\_name MODIFY [COLUMN] col\_name column\_definition [FIRST |AFTER col\_name];

ALTER TABLE users2 MODIFY id SMALLINT UNSIGNED NOT NULL FIRST; //将id字段的位置提到第一列

SHOW COLUMNS FROM users2;

ALTER TABLE users2 MODIFY id TINYINT UNSIGNED NOT NULL; //修改数据类型,需注意数据丢失的问题

2.修改列名称

ALTER TABLE tbl\_name CHANGE [COLUMN] col\_name new\_col\_name column\_definition [FIRST|AFTER col\_name];

ALTER TABLE users2 CHANGE pid p\_id TINYINT UNSIGNED; //修改列名称

数据表更名：ALTER TABLE tbl\_name RENAME [TO/AS] new\_tbl\_name RENAME TABLE tbl\_name TO new\_tbl\_name [, tbl\_name2 TO new\_tbl\_name2] ...

RENAME TABLE users5 TO users2;

表级约束 ：foreign key unique key primary key 列级约束：NOT NULL , DEFAULT

插入记录：INSERT [INTO] table\_name [(column\_name,...)] {VALUES|VALUE} ({expr|DEFAULT},...),(...),...; 当字段有自动增加的属性后，在插入表的所有字段的数据时，可以填写 NULL 或 DEFAULT 这两个关键字，表示按照原先默认自动增长的属性。可同时插入多条记录。

INSERT tb\_name SET column\_name={expr|DEFAULT} 的方法是可以进行子查询，但是只能插入一条记录.EG: INSERT users SET username='Ben',password='456';

第三种：insert table\_name [(colname...)] select.....

UPDATE [LOW\_PRIORITY] [IGNORE] table\_reference SET col\_name1={exp1|DEFAULT}[,col\_name2=...][WHERE where\_condition]; eg：指定位置： UPDATE users set age = age+10 WHERE id % 2=0; 取id为偶数的位置，不指定位置将对表中所有列修改。

DELETE FROM users WHERE id=6;

[group by{col\_name|position} [ASC|DESC],...]

select \* from users group by sex;

select \* from users group by 1;(这里的一表示查询的第一个字段，这里查询所有，第一个字段就是id，也就是会按照字段进行分组)

利用GROUP BY 分组 添加分组条件 [HAVING where\_condition]

要么为一个聚合函数，要么出现在SELECT 条件中。

[HAVING when where\_condition]

SELECT sex FROM users GROUP BY 1 HAVING age > 35; //有错

SELECT sex, age FROM users GROUP BY 1 HAVING age > 35;

SELECT sex FROM users GROUP BY 1 HAVING count(id) >= 2;

（这条指令，指按照sex分组，分成了两组sex=0和sex=NULL，条件id数大于2的留下显示，显然sex=0个数有8>2故留下，而sex=NULL个数为1，不满足条件，故不显示。如果条件改成count(id)>=1,则会显示sex为0和NULL两个分组。）

聚合函数永远只有一个返回结果

count是记录个数的，

order by

对查询结果进行排序，默认是升序

order by{col\_name}

select \* from users order by id desc; //对一个字段排序

select \* from users order by age,id desc; //两个字段同时排序

desc是降序

SELECT \* FROM users LIMIT 2; // 一个数字限制查询结果数量为 2 条

SELECT \* FROM users LIMIT 2,3 ;从第三个开始（第一个为0），返回三条。

第三种insert，数据从一个表插入到另一个表：

INSERT test SELECT username FROM users WHERE age >=30; // 字段不匹配提示

INSERT test（username） SELECT username FROM users WHERE age >=30;

这是INSERT的第三种方法，与前两种相比，这种方法是使用了子选择，这样就相当于将A表中满足条件的数据导入了B表，无需手动写了。

SELECT \* FROM tdb\_goods\G; // 网格形式输出查询结果。 SET NAMES GBK; 设置读取信息的编码格式。

子查询de注意点：

（1）子查询指嵌套在查询内部，且必须始终出现在圆括号内。

（2）子查询可以包含多个关键字或条件，如：

DISTINCT / GROUP BY / ORDER BY / LIMIT / 函数等。

（3）子查询的外层查询可以是：SELECT, INSERT, UPDATE , SET 或DO

（4）子查询可以返回标量、一行、一列或子查询。

（1）SELECT AVG( price ) FROM goods;

（2）SELECT ROUND（ AVG(price),2 ）; // 它是个聚合函数——AVG求平均值进行四舍五入，保留 2 位小数

（3）SELECT id,name,price FROM goods WHERE price >= (SELECT ROUND(AVG(price),2) FROM goods); // 子查询方式

（4）在子查询时的关键字ANY / SOME / ALL

select goods\_name,goods\_price from tdb\_goods where goods\_price >= ANY (select goods\_price from tdb\_goods where goods\_cate='超级本');

IN NOT IN EXISTS eg：SELECT goods\_id,goods\_name,goods\_price FROM tdb\_goods WHERE goods\_price IN (SELECT goods\_price FROM tdb\_goods WHERE goods\_cate = '超级本')

INNER JOIN,内连接

在MySQL中，JOIN, CROSS JOIN 和 INNER JOIN 是等价的。

LEFT [OUTER] JOIN ,左外连接

RIGHT [OUTER] JOIN,右外连接

eg：update tdb\_goods inner join tdb\_goods\_cates on goods\_cate=cate\_name set goods\_cate=cate\_id;

tdb\_goods：想要更改的表名

inner join： 内连接

tdb\_goods\_cates： 关联的附表

goods\_cate=cate\_name 两个表对应列的关系

goods\_cate=cate\_id; 设置 值

创建表的同时向里面写入查找到的数据：

语法： create table if not exists tbl\_test1 (

id int unsigned primary key auto\_increment,

brand\_name varchar(40) not null) select brand\_name from tbl\_test2 group by brand\_name;

同时修改多个表的数据类型：

语法： alter table tbl\_test3 change brand\_name brand\_id int unsigned not null, change goods\_cate cate\_id int unsigned not null;

INNER JOIN eg：SELECT \* FROM tabA JOIN tabB ON tabA.name = tabB.name;表示返回都含有的name值对应的字段，on表示连接条件

LEFT JOIN：显示左表全部和左右符合连接条件的记录，RIGHT JOIN：显示左右符合连接条件的记录和右表全部记录

SELECT goods\_id,goods\_name,b.cate\_name,c.brand\_name,goods\_price

FROM products AS a INNER JOIN products\_cate AS b ON a.goods\_cate = b.cate\_id

INNER JOIN products\_brand AS c ON a.brand\_name = c.brand\_id; //多表连接

外连接：

以左外连接为例：

A LEFT JOIN B join\_condition

数据表B的结果集依赖于数据表A

数据表A的结果集根据左连接条件依赖所有数据表(B表除外)

左外连接条件决定如何检索数据表B(在没有指定WHERE条件的情况下)

如果数据表A的某条记录符合WHERE条件，但是在数据表B不存在符合连接条件的记录，将生成一个所有列为空的额外的B行

内连接：

使用内连接查找的记录在连接数据表中不存在，并且在WHERE子句中尝试一下操作：column\_name IS NULL 。如果 column\_name 被指定为 NOT NULL，MySQL将在找到符合连接着条件的记录后停止搜索更多的行（查找冲突）

无限分类：即在同一张表中既有父类，又有子类

通过在分类表中再增加多一个字段标识其属于哪一个父类的 ID 来实现

可以通过对同一张数据表的自身连接来进行查询，需要对表标识别名

查找显示父级id对应的名称

select s.type\_id ,s.type\_name,p.type\_name As parent\_id from tdb\_goods\_types s left join tdb\_goods\_types p on s.parent\_id=p.type\_id;

查找子级对应的名称

select p.type\_id ,p.type\_name,s.type\_name from tdb\_goods\_types p left join tdb\_goods\_types s on p.type\_id=s.parent\_id;

查找有多少子级

select p.type\_id ,p.type\_name,COUNT(s.type\_name) child\_count from tdb\_goods\_types p left join tdb\_goods\_types s on p.type\_id=s.parent\_id GROUP BY p.type\_name ORDER BY p.type\_id;

多表删除：DELETE t1 FROM tdb\_goods AS t1 LEFT JOIN (SELECT goods\_id,goods\_name FROM tdb\_goods GROUP BY goods\_name HAVING count(goods\_name) >= 2 ) AS t2 ON t1.goods\_name = t2.goods\_name WHERE t1.goods\_id > t2.goods\_id;

字符函数：（1）CONCAT函数 字符连接

SELECT CONCAT('IMOOC','-','MySQL');

SELECT CONCAT (first\_name,last\_name) AS fullname FROM test;

（2）CONCAT\_WS() 使用指定的分隔符进行字符连接，（第一个位置指定分隔符，后面的 为分割的内容）

SELECT CONCAT\_WS('%','asdsa','asdasda'); //第一个是指定的分隔符

（3）FORMAT() 数字格式化

例如：SELECT FORMAT(123560.75,2); 123,560.75

（4）LOWER() 转化小写

（5）UPPER() 转换大写

SELECT UPPER('mysql');

（6）LEFT() 获取左侧字符

SELECT LEFT ('MYSQL',2); MY

SELECT LOWER(LEFT('mYsql',2)); //函数嵌套

（7）RIGHT()获取右侧字符

（8）LENGTH()取得字符串长度

LENGTH('MYSQL’);

（9）TRIM() 删除前导、后续空格或者指定字符:

SELECT TRIM(' MySQL ');

SELECT TRIM(LEADING '?' FROM '??MYSQL???'); //删除前导？

SELECT TRIM(TRAILING '?' FROM '??MYSQL???'); //删除后续？

SELECT TRIM(BOTH '?' FROM '??MYSQL???'); //前后均删除，但不能删除中间的?

（10）REPLACE() 替换 :

SELECT REPLACE('MY???SQL','?',''); //将问号换成空白，即去掉？

SELECT REPLACE('MY???SQL','?','~~'); //可以将n个？换成m个组合符号

（11）SUBSTRING() 字符串截取

SELECT SUBSTRING('mYSQL',1,2); //从字符串中的第一个位置开始截取2个字符mY

SELECT SUBSTRING('mYSQL',3); //SQL 从第3的位置开始截取到结束

SELECT SUBSTRING('mYSQL',-2); //QL 从倒数第2位开始截取到结束

数值运算符：CEIL()--进一取正

FLOOR()--舍一取正

DIV--整数除法

MOD --取余

POWER() --幂运算

ROUND() --取余（四舍五入）

TRUNCATE() --截断（不四舍五入）

1、[NOT]BETWEEN...AND...[不]在范围之内，结果返回1或0。例如:15 BETWEEN 10 AND 16:1

2、[NOT]IN()列出值选项。例如SELECT 10 IN(5,10,25):1

3、IS [NOT] NULL：只有NULL IS [NOT] NULL返回1，其他情况（包括'',0）都返回0

SELECT DATABASES(); SELECT USER(); SELECT VERSION();

聚合函数：AVG() 平均值、COUNT() 计数、MAX() 最大值、MIN() 最小值、SUM() 求和

select md5（'admin'）;给admin加密。如果mysql中的信息是为了web开发做准备，那么尽量使用md5.

PASSWORD md5（'admin'）;进行密码的计算，修改当前或者其他客户端

改密码:SET PASSWORD=PASSWORD('dimitar');

CREATE FUNCTION function\_name

RETURNS

{STRING|INTEGER|REAL|DECIMAL}

routine\_body - 函数体

CREATE FUNCTION f1() RETURNS VARCHAR(30)//创建一个f1()函数

RETURN DATE\_FORMAT(NOW(),'%Y年 %m月 %d日 %H点：%i分：%s秒');//返回值

SELECT f1();//调用自定义函数f1();

CREATE FUNCTION f2(num1 SMALLINT UNSIGNED , num2 SMALLINT UNSIGNED)

RETURNS FLOAT(10,2) UNSIGNED

RETURN (num1 + num2)/2; //带参函数

delimiter // 切换结束符//

CREATE FUNCTION ADD\_USER(p\_id SMALLINT,username VARCHAR(20))

RETURNS INT UNSIGNED

BEGIN

INSERT user(p\_id,username) VALUES(p\_id,username);

RETURN LAST\_INSERT\_ID();

END

//MYSQL存储过程简介

存储过程是SQL语句和控制语句的预编译集合，以一个名称存储并作为一个单元处理。

存储过程存储在数据库内，可由应用程序调用执行。允许用户声明变量，进行过程控制及接受参数，且可存在多个返回值。存储过程是预编译的，当第一次调用时，会进行编译和分析，后面再调用则不用。

优点：

1.增强SQL语句的功能和灵活性

2.实现较快的执行速度

3.减少了网络流量

1、存储过程语法结构分析

CREATE

[DEFINER = {user|CURRENT\_USER}]

PROCEDURE sp\_name ([proc\_parameter[,...]]) //可以带0到多个参数

[characteristic ...] routine\_body

其中参数

proc\_parameter:

[IN | OUT | INOUT] param\_name type

IN, 表示该参数的值必须在调用存储过程时指定

OUT, 表示该参数的值可以被存储过程改变，并且可以返回

INOUT, 表示该参数的值调用时指定，并且可以被改变和返回

CREATE PROCEDURE sp1() SELECT VERSION(); CALL sp1;

创建带有IN类型参数的存储过程

DELIMITER // //定界符

CREATE PROCEDURE removeUserById(IN p\_id INT UNSIGNED) //int 是数据类型

BEGIN DELETE FROM users WHERE id = p\_id;

END

创建带有IN OUT类型参数的存储过程

DELIMITER //

CREATE PROCEDURE removerUserAndReturnUserName(IN showID INT UNSIGNED,OUT showName INT UNSIGNED)

BEGIN

DELETE FROM user WHERE id = showID;

SELECT count(ID) FROM user INTO showName;

END

//

SELECT count(ID) FROM user INTO showName; /\*\* 该语句中的 INTO 含义就是将 SELECT 语句结果的表达式返回到 showName 变量中 \*/

CALL removerUserAndReturnUserName(10,@nums); /\*\* @nums 所代表的就是用户变量，可用 SELECT @nums 输出 \*/

用 DECLARE 声明的变量是局部变量，局部变量只能存在于 BEGIN...END 之间，且声明时必须置于 BEGIN...END 的第一行

而通过 SELECT...INTO.../SET @id = 07 这种方法设置的变量我们称之为用户变量，只能存在于当前用户所使用的客户端有效。

CALL rmUserAndRtUserNums(27, @nums);

SELECT @nums; //@nums - 就是用户变量

DECLARE声明的变量都是在BEGIN与END之间，是局部变量

SET @i = 7; //通过@或SET设置的变量称为用户变量

CREATE PROCEDURE rmUserByAgeAndRtInfos(IN p\_age SMALLINT UNSIGNED, OUT delNums SMALLINT UNSIGNED, OUT leftNums SMALLINT UNSIGNED)

BEGIN

DELETE FROM users WHERE WHERE age = p\_age; //注意变量不同

SELECT ROW\_COUNT() INTO delNums;

SELECT COUNT(id) FROM users INTO leftNums;

END //带有多个OUT类型参数的PROCEDURE

存储过程与自定义函数的区别

A、存储过程实现的功能相对复杂，函数针对性较强

B、存储过程可以返回多个值，函数只能有一个返回值

C、存储过程一般独立执行，函数可以作为 sql 语句的组成部分来出现

存储引擎

（1）MySQL可以将数据以不同的技术存储在文件（内存）中，这种技术就成为存储引擎。每种存数引擎使用不同的存储机制、索引技巧、锁定水平，最终提供广泛且不同的功能。

（2）使用不同的存储引擎也可以说不同类型的表

（3）MySQL支持的存储引擎：MyISAM、InnoDB、Memory、CSV、Archive

并发控制：

（1）当多个连接对记录进行修改时保证数据的一致性和完整。

1. 共享锁(读锁) ： 在同一段时间段内，多个用户可以读取同一个资源，读取过程中数据不会发生任何变化

2. 排他锁（写锁） ：在任何时候只能有一个用户写入资源，当进入写锁时会阻塞其他的读锁或者写锁操作。

锁颗粒：

表锁：是一种开销最小的锁策略

行锁：是一种开销最大的锁策略，能提供最大的并发操作

事务处理：整个过程每一个单元全部完成才算事务处理成功，某一个单元失败事务就会回滚.

主要作用：保证数据库的完整性

事务的特性：原子性、一致性、隔离性、持久性

简称：A(Atomic)C(Consistency)I(Isolation)D(Durable)

使用最多的：MyISAM，InnoDB

CSV存储引擎：以逗号为分隔符，不支持索引；

BlackHole：黑洞引擎，写入的数据都会消失，一般用于做数据复制的中继；

MyISAM：适用于事务的处理不多的情况；

InnoDB：适用于事务处理比较多，需要有外键支持的情况。

修改存储引擎：

修改MySQL配置文件实现 default-storage-engine = engine

CREATE TABLE table\_name(\...) ENGINE = engine;

ALTER TABLE tbl\_name ENGINE [=] engine\_name;

图形化管理工具：PHPMyAdmin （Web界面）、Navicat （平台方式运行）、MySQL Workbench （平台方式运行）

# HTTP协议

* *我们在传输数据时，可以只使用(传输层)TCP/IP协议，但是那样的话，如果没有应用层，便无法识别数据内容。如果想要使传输的数据有意义，则必须使用到应用层协议。应用层协议有很多，比如HTTP、FTP、TELNET等，也可以自己定义应用层协议。WEB使用HTTP协议作应用层协议，以封装HTTP文本信息，然后使用TCP/IP做传输层协议将它发到网络上*
* *socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口(API)；TCP/IP只是一个协议栈，就像操作系统的运行机制一样，必须要具体实现，同时还要提供对外的操作接口。*

*这个就像操作系统会提供标准的编程接口，比如win32编程接口一样，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口。*

* *传输层的TCP是基于网络层的IP协议的，而应用层的HTTP协议又是基于传输层的TCP协议的，而Socket本身不算是协议，它只是提供了一个针对TCP或者UDP编程的接口。*
* *TCP的三次握手：****第一次握手****：客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认;****第二次握手****：服务器收到syn包，必须确认客户的SYN(ack=j+1)，同时自己也发送一个SYN包(syn=k)，即SYN+ACK包，此时服务器进入SYN\_RECV状态;****第三次握手****：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。*

应用层，基于TCP协议—可靠传输

B/S C/S –浏览器/服务器、客户端（PC软件or手机app）/服务器

资源：静态资源（不变的文件）、动态资源（新闻），静态资源可缓存访问速度快，效率高

媒体类型content-type：

URI-URL统计资源定位符 URN

Get 、Post的区别

报文：请求+响应，组成：起始行+header+body

get 从服务器获取资源

post 向服务器提交信息

option 返回服务器支持的请求方法

head 获取报文，不返回文档主体

put 创建资源

pv--page view 不排重，页面刷新一次计数一次

uv--user vistor 不同用户名，不同ip

cdn--content delivery network 内容分发网络，缓存经常访问的文件，就近

获取资源，付费提供商【蓝汛、网宿】；静态内容缓存js文件，解决网络延迟

Nginx--web application

dns--Domain Name Service

dns劫持--域名劫持

DNS劫持又称域名劫持，是指在劫持的网络范围内拦截域名解析的请求，分析请求的域名，把审查范围以外的请求放行，否则返回假的IP地址或者什么都不做使请求失去响应，其效果就是对特定的网络不能反应或访问的是假网址。

TCP数据包：分段发送、源端口、目的端口、源ip、目的ip

计算机端口65535个端口：http--80；https--430；smtp--25；pop--110；mysql--3306

连接数配置：apache--100左右；tomcat；mysql

连接池：节省时间，节省连接创建的时间，使用完释放到连接池

socket协议：编写server客户端

tcp建立连接；tcp udp的区别、应用

http事务时延

## cookie、session

<https://www.zhihu.com/question/19786827>

sessionID是服务器发送给客户端的用于标识同一用户，比如购物车、结算类似需要验证是哪一个用户对服务器的请求，sessionID在服务器可存入内存、数据库、文件等等，客户端则会存入cookie中，在每次请求中都会把sessionID发送到服务器，若客户端禁用了cookie功能，则会通过一种重写URL的技术来识别用户（在每次请求的URL后追加sid=xxx的参数）。Cookie可以记录用户的信息（如登录账号密码，在下次访问同一网页时自动填入账号密码）

* session在服务器，cookie在客户端
* session默认被存在服务器的一个文件里（不是内存）
* session的运行依赖sessionID 而sessionID是存在cookie中的，也就是说，如果浏览器禁用了cookie 同时session也会失效（但可通过在URL中传递sessionID的方式实现）
* session可放在文件、数据库、或内存中
* 用户验证这种场合一般会用session

# 电子邮件

## 一封电子邮件的旅程

*MUA-- Mail User Agent——邮件用户代理*

*MTA-- Mail Transfer Agent——邮件传输代理*

*MDA-- Mail Delivery Agent——邮件投递代理*

发件人 -> MUA -> MTA -> MTA -> 若干个MTA -> MDA <- MUA <- 收件人

## 编写程序

### 模块

from email.mime.text import MIMEText

import smtplib

msg=MIMEText(‘send email by python…’ , ’utf-8’)

**dir(msg):**

*['attach', 'defects', 'del\_param', 'epilogue', 'get', 'get\_all', 'get\_boundary', 'get\_charset', 'get\_charsets', 'get\_content\_charset', 'get\_content\_maintype', 'get\_content\_subtype', 'get\_content\_type', 'get\_default\_type', 'get\_filename', 'get\_param', 'get\_params', 'get\_payload', 'get\_unixfrom', 'has\_key', 'is\_multipart', 'items', 'keys', 'preamble', 'replace\_header', 'set\_boundary', 'set\_charset', 'set\_default\_type', 'set\_param', 'set\_payload', 'set\_type', 'set\_unixfrom', 'values', 'walk']*

**server=smtplib.SMTP(‘smtp.163.com’ , 25)**

*dir(smtplib.SMTP)*

*['close', 'connect', 'data', 'debuglevel', 'default\_port', 'docmd', 'does\_esmtp', 'ehlo', 'ehlo\_msg', 'ehlo\_or\_helo\_if\_needed', 'ehlo\_resp', 'expn', 'file', 'getreply', 'has\_extn', 'helo', 'helo\_resp', 'help', 'login', 'mail', 'noop', 'putcmd', 'quit', 'rcpt', 'rset', 'send', 'sendmail', 'set\_debuglevel', 'starttls', 'verify', 'vrfy']*

### 发邮件时

MUA和MTA使用的协议就是SMTP：Simple Mail Transfer Protocol，后面的MTA到另一个MTA也是用SMTP协议。

构造一个邮件对象就是一个Messag对象，如果构造一个MIMEText对象，就表示一个文本邮件对象，如果构造一个MIMEImage对象，就表示一个作为附件的图片，要把多个对象组合起来，就用MIMEMultipart对象，而MIMEBase可以表示任何对象。它们的继承关系如下：

***Message***

***+- MIMEBase***

***+- MIMEMultipart***

***+- MIMENonMultipart***

***+- MIMEMessage***

***+- MIMEText***

***+- MIMEImage***

*#coding=utf-8*

*from email import encoders*

*from email.header import Header*

*from email.mime.text import MIMEText*

*from email.mime.multipart import MIMEMultipart*

*from email.mime.multipart import MIMEBase*

*from email.utils import parseaddr,formataddr*

*import smtplib*

*def \_format\_addr(s):*

*name,addr=parseaddr(s)*

*return formataddr((Header(name,'utf-8').encode(),*

*addr.encode('utf-8') if isinstance(addr,unicode) else addr))*

*from\_addr='ldjwyyx@163.com'*

*passwd='wy1989-10-26'*

*to\_addr='865479851@qq.com'*

*smtp\_server='smtp.163.com'*

*body='''*

*<html>*

*<body>*

*<h1>Hello</h1>*

*<p>send by <a href="http://www.python.org">Python</a>...</p>*

*<p><img src="cid:0"></p> <!-- 将附件的图片添加至正文显示 -->*

*</body></html>*

*'''*

*msg=MIMEMultipart('alternative')*

*#同事支持HTML和plain格式，由于部分古老设备不支持HTML，则可以选择接收plain格式*

*msg.attach(MIMEText('hello this is your info...','plain','utf-8'))*

*#添加正文部分*

*msg.attach(MIMEText(body,'html','utf-8'))*

*msg['From']=\_format\_addr(u'你大哥<%s>'%from\_addr)*

*msg['To']=\_format\_addr(u'小弟<%s>'%to\_addr)*

*msg['Subject']=Header(u'来自SMTP的问候。。。','utf-8').encode()*

*#发送附件*

*with open('/Users/ralphliu/Desktop/1.png','rb') as f:*

*mime=MIMEBase('image','png',filename='1.png')*

*mime.add\_header('Content-Disposition','attachment',filename='1.png')*

*mime.add\_header('Content-ID','<0>')*

*mime.add\_header('X-Attachment-Id','0')*

*mime.set\_payload(f.read())*

*encoders.encode\_base64(mime)*

*msg.attach(mime) #添加附件*

*server=smtplib.SMTP(smtp\_server,25)*

*server.starttls() #加密传输，如Gmail的SMTP是587，使用加密传输*

*server.set\_debuglevel(1)*

*server.login(from\_addr,passwd)*

*server.sendmail(from\_addr,[to\_addr],msg.as\_string())*

*server.quit()*

### 收邮件时

MUA和MDA使用的协议有两种：POP：Post Office Protocol，目前版本是3，俗称POP3；IMAP：Internet Message Access Protocol，目前版本是4，优点是不但能取邮件，还可以直接操作MDA上存储的邮件，比如从收件箱移到垃圾箱。

收取邮件就是编写一个MUA作为客户端，从MDA把邮件获取到用户的电脑或者手机上。收取邮件最常用的协议是POP协议，目前版本号是3，俗称POP3

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

*import poplib*

*import email*

*from email.parser import Parser*

*from email.header import decode\_header*

*from email.utils import parseaddr*

*def guess\_charset(msg):*

*charset = msg.get\_charset()*

*if charset is None:*

*content\_type = msg.get('Content-Type', '').lower()*

*pos = content\_type.find('charset=')*

*if pos >= 0:*

*charset = content\_type[pos + 8:].strip()*

*return charset*

*def decode\_str(s):*

*value, charset = decode\_header(s)[0]*

*if charset:*

*value = value.decode(charset)*

*return value*

*def print\_info(msg, indent=0):*

*if indent == 0:*

*for header in ['From', 'To', 'Subject']:*

*value = msg.get(header, '')*

*if value:*

*if header=='Subject':*

*value = decode\_str(value)*

*else:*

*hdr, addr = parseaddr(value)*

*name = decode\_str(hdr)*

*value = u'%s <%s>' % (name, addr)*

*print('%s%s: %s' % (' ' \* indent, header, value))*

*if (msg.is\_multipart()):*

*parts = msg.get\_payload()*

*for n, part in enumerate(parts):*

*print('%spart %s' % (' ' \* indent, n))*

*print('%s--------------------' % (' ' \* indent))*

*print\_info(part, indent + 1)*

*else:*

*content\_type = msg.get\_content\_type()*

*if content\_type=='text/plain' or content\_type=='text/html':*

*content = msg.get\_payload(decode=True)*

*charset = guess\_charset(msg)*

*if charset:*

*content = content.decode(charset)*

*print('%sText: %s' % (' ' \* indent, content + '...'))*

*else:*

*print('%sAttachment: %s' % (' ' \* indent, content\_type))*

*email = 'liudongjie2015@126.com'*

*password = 'liudongjie126'*

*pop3\_server = 'pop.126.com'*

*server = poplib.POP3(pop3\_server)*

*#server.set\_debuglevel(1)*

*print(server.getwelcome())*

*# 认证:*

*server.user(email)*

*server.pass\_(password)*

*print('Messages: %s. Size: %s' % server.stat())*

*resp, mails, octets = server.list()*

*# 获取最新一封邮件, 注意索引号从1开始:*

*resp, lines, octets = server.retr(len(mails))*

*# 解析邮件:*

*msg = Parser().parsestr('\r\n'.join(lines))*

*# 打印邮件内容:*

*print\_info(msg)*

*# 慎重:将直接从服务器删除邮件:*

*# server.dele(len(mails))*

*# 关闭连接:*

*server.quit()*

# 操作数据库

## SQLite

SQLite支持常见的SQL语句及数据类型，轻量级、可嵌入的数据管理无需安装，会生成一个db文件。占位符为？

Import sqlite3

*>>> conn=sqlite3.connect('test.db')*

*>>> cursor=conn.cursor()*

*>>> cursor.execute('create table user(id varchar(20) primary key,name varchar(20))')*

*>>> cursor.execute("select \* from user")*

*<sqlite3.Cursor object at 0x10e120b90>*

*>>> values=cursor.fetchall()*

*>>> values*

*[(u'1', u'jack')]*

*>>> cursor.close()*

*>>> conn.commit()*

*>>> conn.close()*

## MySQL

show variables like '%char%'; 可查看数据库设置项，占位符--%s

在Python中，最有名的ORM--Object-Relational Mapping框架是SQLAlchemy。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

Pip install sqlalchemy

*#coding=utf-8*

*#step1 导入SQLalchemy，并且初始化*

*from sqlalchemy import Column,String,create\_engine*

*from sqlalchemy.orm import sessionmaker*

*from sqlalchemy.ext.declarative import declarative\_base*

*Base=declarative\_base()*

*class User(Base):*

*\_\_tablename\_\_='user'*

*id=Column(String(20),primary\_key=True)*

*name=Column(String(20))*

*#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*

*engine=create\_engine('mysql+mysqldb://root:root@localhost:3306/tmp')*

*DBSession=sessionmaker(bind=engine)*

*#创建session对象*

*session=DBSession()*

*new\_user=User(id='5',name='bob')*

*# session.add(new\_user)*

*# session.commit()*

*user=session.query(User).filter(User.id=='5').one()*

*print 'type:',type(user)*

*print 'name:',user.name*

*session.close()*

*#一对多联表查询*

*class User(Base):*

*\_\_tablename\_\_='user':*

*id=Column(String(20),primary\_key=True)*

*name=Column(String(20))*

*books=relationship('Book')*

*class Book(Base):*

*\_\_tablename\_\_='book'*

*id=Column(String(20),primary\_key=True)*

*name=Column(String(20))*

*user\_id=Column(String(20),ForeignKey('user.id'))*

# web开发

## WSGI接口：Web Server Gateway Interface

Python内置了一个WSGI服务器，这个模块叫wsgiref，它是用纯Python编写的WSGI服务器的参考实现。所谓“参考实现”是指该实现完全符合WSGI标准，但是不考虑任何运行效率，仅供开发和测试使用。

打开浏览器地址栏输入localhost:8000/ralph -🡪 Hello, ralph

*#coding=utf-8*

*def application(environ,start\_response):*

*start\_response('200 ok',[('Content-Type','text/html')])*

*return '<h1>Hello, %s</h1>'%(environ['PATH\_INFO'][1:] or 'web')*

*from wsgiref.simple\_server import make\_server*

*httpd=make\_server('',8000,application)*

*print 'Serving HTTP on port 8000'*

*httpd.serve\_forever()*

无论多么复杂的Web应用程序，入口都是一个WSGI处理函数。HTTP请求的所有输入信息都可以通过environ获得，HTTP响应的输出都可以通过start\_response()加上函数返回值作为Body。

## Web框架

Flask框架：pip install flask

*#coding=utf-8*

*from flask import Flask*

*from flask import request*

*app=Flask(\_\_name\_\_)*

*@app.route('/',methods=['GET','POST'])*

*def home():*

*return '<h1>Home</h1>'*

*@app.route('/signin',methods=['GET'])*

*def signin\_form():*

*return '''<form action="/signin" method="post">*

*<p><input name="username"></p>*

*<p><input name="password" type="password"></p>*

*<p><button type="submit">Sign In</button></p>*

*</form> '''*

*@app.route('/signin',methods=['POST'])*

*def signin():*

*if request.form['username']=='admin' and request.form['password']=='1234':*

*return '<h3>Hello, admin</h3>'*

*return '<h3>Bad username or password</h3>'*

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

*app.run()*

### MVC模型

* Flask通过render\_template()函数来实现模板的渲染。和Web框架类似，Python的模板也有很多种。Flask默认支持的模板是jinja2。在view中HTML文件中使用{{ username }}
* 在Jinja2模板中，我们用{{ name }}表示一个需要替换的变量。很多时候，还需要循环、条件判断等指令语句，在Jinja2中，用{% ... %}表示指令.**比如循环输出页码：**

*{% for i in page\_list %}*

*<a href="/page/{{ i }}">{{ i }}</a>*

*{% endfor %}*

* 有了MVC，我们就分离了Python代码和HTML代码。HTML代码全部放到模板里。

*#coding=utf-8*

*from flask import Flask,request,render\_template*

*app=Flask(\_\_name\_\_)*

*@app.route('/',methods=['GET','POST'])*

*def home():*

*return render\_template('home.html')*

*@app.route('/signin',methods=['GET'])*

*def signin\_form():*

*return render\_template('form.html')*

*@app.route('/signin',methods=['POST'])*

*def signin():*

*username=request.form['username']*

*password=request.form['password']*

*if username=='admin' and password=='1234':*

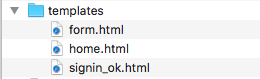
*return render\_template('signin\_ok.html',username=username)*

*return render\_template('form.html',message='bad username or password',username=username)*

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

*app.run()*

app.py 与templates目录在同级目录下



# 协程coroutine

* 协程，又称微线程。
* 子程序，或者称为函数，在所有语言中都是层级调用，子程序调用总是一个入口，一次返回，调用顺序是明确的。而协程的调用和子程序不同。
* 协程看上去也是子程序，但执行过程中，在子程序内部可中断，然后转而执行别的子程序，在适当的时候再返回来接着执行。
* 协程执行有点像多线程，但协程的特点在于是一个线程执行。最大的优势就是协程极高的执行效率。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。
* 第二大优势就是不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多
* 传统的生产者-消费者模型是一个线程写消息，一个线程取消息，通过锁机制控制队列和等待，但一不小心就可能死锁。如果改用协程，生产者生产消息后，直接通过yield跳转到消费者开始执行，待消费者执行完毕后，切换回生产者继续生产，效率极高：

*#coding=utf-8*

*import time*

*def cosumer():*

*r=''*

*while True:*

*n=yield r*

*if not n:*

*return*

*print '[consumer] consuming %s...'%n*

*time.sleep(1)*

*r='200 ok'*

*def produce(c):*

*c.next()*

*n=0*

*while n<5:*

*n+=1*

*print '[producer] producing %s...'%n*

*r=c.send(n)*

*print '[producer] consumer return %s'%r*

*c.close()*

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

*c=cosumer()*

*produce(c)*

## yield用法

*>>> def echo(value=None):*

*... while 1:*

*... value = (yield value)*

*... print("The value is", value)*

*... if value:*

*... value += 1*

*...*

*>>> g = echo(1)*

*>>> next(g)*

*1*

*>>> g.send(2)*

*The value is 2*

*3*

*>>> g.send(5)*

*The value is 5*

*6*

*>>> next(g)*

*The value is None*

第一次调用next()方法，执行到yield value表达式，保存上下文环境暂停返回1。第二次调用send(value)方法，从value = yield开始，打印，再次遇到yield value暂停返回。

## Gevent

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：

*#coding=utf-8*

*from gevent import monkey*

*monkey.patch\_all()*

*import gevent*

*import urllib2*

*def fun(url):*

*print 'GET: %s'%url*

*resp=urllib2.urlopen(url)*

*data=resp.read()*

*print '%d bytes received from %s'%(len(data),url)*

gevent.joinall([gevent.spawn(fun,'https://www.baidu.com'),

gevent.spawn(fun,'https://www.qq.com'),

gevent.spawn(fun,'https://github.com')])

使用gevent，可以获得极高的并发性能，但gevent只能在Unix/Linux下运行，在Windows下不保证正常安装和运行。

由于gevent是基于IO切换的协程，所以最神奇的是，我们编写的Web App代码，不需要引入gevent的包，也不需要改任何代码，仅仅在部署的时候，用一个支持gevent的WSGI服务器，立刻就获得了数倍的性能提升。

# HTML、css

Iframe 内联框架，嵌入到页面中<iframe src="https://www.baidu.com" width="900" height="400" frameborder="0"></iframe>

## HTML颜色

颜色值、颜色名、web安全色，三种都可以。Rgb(255,0,0)

## 字符实体

&nbsp --空格

## css

* 层叠样式优先级：内联样式（在HTML元素内部）>内部样式表（在head标签中）>外部样式表>浏览器缺省样式
* 最后加载的样式会覆盖之前的样式

# mac Apache

# Virtual hosts //设置虚拟主机【包括访问路径和域名地址，在hosts文件中添加】

配置文件：httpd.conf

DocumentRoot //设置访问路径

Apache安装目录：/private/etc/apache2/

网页访问路径：/Library/WebServer/Documents 将要发布的文件放在路径下即可【重启Apache】

启动/停止/重启命令：sudo apachectl start/stop/restart

示例：默认的两个虚拟主机要注释掉# ，重新添加即可

<VirtualHost \*:80>

DocumentRoot "/Library/WebServer/Documents/web"

ServerName localhost

ErrorLog "/private/var/log/apache2/localhost-error\_log"

CustomLog "/private/var/log/apache2/localhost-access\_log" common

</VirtualHost>

<VirtualHost \*:80>

DocumentRoot "/Users/ralphliu/work"

ServerName mysites

ErrorLog "/private/var/log/apache2/sites-error\_log"

CustomLog "/private/var/log/apache2/sites-access\_log" common

<Directory />

Options Indexes FollowSymLinks MultiViews

AllowOverride None

Order deny,allow

Allow from all

</Directory>

</VirtualHost>

# 面试

## 逻辑题

1、A、B两人分别在两座岛上。B生病了，A有B所需要的药。C有一艘小船和一个可以上锁的箱子。C愿意在A和B之间运东西，但东西只能放在箱子里。只要箱子没被上锁，C都会偷走箱子里的东西，不管箱子里有什么。如果A和B各自有一把锁和只能开自己那把锁的钥匙，A应该如何把东西安全递交给B？

*答案：A把药放进箱子，用自己的锁把箱子锁上。B拿到箱子后，再在箱子上加一把自己的锁。箱子运回A后，A取下自己的锁。箱子再运到B手中时，B取下自己的锁，获得药物。*

2、有一个软件公司，1/2的人是系统分析员，2/5的人是软件工程师，有1/4的人两者都是，问有多少人两者都不是？

*答案： 1 – 1/2 – 2/5 + 1/4＝ 0.35*

3、有25匹马，速度都不同，但每匹马的速度都是定值。现在只有5条赛道，无法计时，即每赛一场最多只能知道5匹马的相对快慢。问最少赛几场可以找出25匹马中速度最快的前3名？（百度2008年面试题）

*答案：每匹马都至少要有一次参赛的机会，所以25匹马分成5组，一开始的这5场比赛是免不了的。接下来要找冠军也很容易，每一组的冠军在一起赛一场就行了（第6场）。最后就是要找第2和第3名。我们按照第6场比赛中得到的名次依次把它们在前5场比赛中所在的组命名为A、B、C、D、E。即：A组的冠军是第6场的第1名，B组的冠军是第6场的第2名……每一组的5匹马按照他们已经赛出的成绩从快到慢编号：*

*A组：1，2，3，4，5*

*B组：1，2，3，4，5*

*C组：1，2，3，4，5*

*D组：1，2，3，4，5*

*E组：1，2，3，4，5*

*从现在所得到的信息，我们可以知道哪些马已经被排除在3名以外。只要已经能确定有3匹或3匹以上的马比这匹马快，那么它就已经被淘汰了。可以看到，只有上表中粗体蓝色的那5匹马才有可能为2、3名的。即：A组的2、3名；B组的1、2名，C组的第1名。取这5匹马进行第7场比赛，第7场比赛的前两名就是25匹马中的2、3名。故一共最少要赛7场。*

4、考虑一个双人游戏。游戏在一个圆桌上进行。每个游戏者都有足够多的硬币。他们需要在桌子上轮流放置硬币，每次必需且只能放置一枚硬币，要求硬币完全置于桌面内（不能有一部分悬在桌子外面），并且不能与原来放过的硬币重叠。谁没有地方放置新的硬币，谁就输了。游戏的先行者还是后行者有必胜策略？这种策略是什么？

*答案：先行者在桌子中心放置一枚硬币，以后的硬币总是放在与后行者刚才放的地方相对称的位置。这样，只要后行者能放，先行者一定也有地方放。先行者必胜。*

5、一个矩形蛋糕，蛋糕内部有一块矩形的空洞。只用一刀，如何将蛋糕切成大小相等的两块？

*答案：注意到平分矩形面积的线都经过矩形的中心。过大矩形和空心矩形各自的中心画一条线，这条线显然把两个矩形都分成了一半，它们的差当然也是相等的。*

# 多进程

## 进程概念

**进程是**具有一定独立功能的程序、它是系统进行资源分配和调度的一个独立单位，重点在系统调度和单独的单位，也就是说进程是可以独立运行的一段程序。为了使参与并发执行的每个程序都能独立运行，在操作系统中必须为之分配一个专门的数据结构，称为进程控制块PCB，系统利用PCB来描述进程的基本情况和活动过程，进而控制和管理进程。**定义**：由程序段、相关数据段和PCB三部分构成的进程实体。

**典型的定义有**：进程是程序的一次执行；进程是一个程序及其数据在处理机上顺序执行时所发生的活动；进程是具有独立功能的程序在一个数据结合上运行的过程，它是系统进行资源分配和调度的一个独立单元。

每一个进程都有自己的地址空间，一般情况下，包括文本区域【存代码】、数据区域【变量和进程执行期间使用的动态分配的内存】、堆栈【活动过程调用的指令和本地变量】。

**线程是**进程的一个实体，是CPU调度和分派的基本单位，他是比进程更小的能独立运行的基本单位，线程自己基本上不拥有系统资源。在运行时，只是暂用一些计数器、寄存器和栈 。

1. 计算机的核心是CPU，它承担了所有的计算任务，就像一座工厂，时刻在运行。
2. 假定工厂的电力有限，一次只能供给一个车间使用，即单个CPU一次只能运行一个任务。
3. 进程就好比工厂的车间，它代表CPU所能处理的单个任务，任一时刻，CPU总是运行一个进程，其他进程处于非运行状态。
4. 一个车间可以有很多工人，他们协同完成一个任务，线程就好比车间里的工人，一个进程可以包括多个线程。
5. 车间的空间是工人们共享的，比如许多房间是每个工人都可以进出的，这象征一个进程的内存空间是共享的每个线程都可以使用这些共享内存。
6. 可是每间房间的大小不同，有些房间最多只能容纳一个人，比如厕所，当里面有人时其他人就不能进去，这代表一个线程使用某些共享内存时，其他线程必须等他结束，才能使用这一块内存。这时就需要用到锁，上锁时，其他线程必须等待锁打开再进去，防止多个线程同时读写某一块内存区域。
7. 还有一些房间，可同时容纳n个人，即如果人数>n ，多出来的人只能在外面等着，好比某些内存区域，只能供给固定数目的线程使用。这时的解决方法就是在门口挂n把钥匙，进去的人就取一把钥匙，出来再把钥匙挂回原处，后来的人发现钥匙架空了，就必须在门口排队等着，这种做法叫信号量semaphore，用来保证多个线程不会互相冲突。

进程特征：

动态性：进程是动态产生，动态消亡的

并发性：任何进程都可以同其他进程一起并发执行。

独立性：进程是一个能独立运行的基本单元，同时也是系统分配资源和调度的独立单元。

异步性：由于进程间的相互制约，使进程具有执行的间断性，即进程按各自独立的、不可预知的速度 执行。

进程的调度算法：FIFO先进先出 ，RR时间片算法，HPF最高优先级算法

进程状态：创建、就绪、运行、阻塞、结束

队列与堆栈：

进程终止：可引入进程终止的事件，正常结束；异常结束；外界干预。

## CPU

CPU由运算器、控制器、寄存器组成

## 进程与线程区别

在CPU看来所有的任务都是一个一个的轮流执行的，具体的轮流方法就是：先加载程序A的上下文（除CPU以外所有构成这个程序的执行环境），然后开始执行A，保存程序A的上下文，调入下一个要执行的程序B的程序上下文，然后开始执行B,保存程序B的上下文。。。。

进程就是包换上下文切换的程序执行时间总和 = CPU加载上下文+CPU执行+CPU保存上下文

CPU在执行的时候没有进行上下文切换的。这里的a，b，c就是线程，也就是说线程是共享了进程的上下文环境的更为细小的CPU时间段。 进程和线程都是一个时间段的描述，是CPU工作时间段的描述，不过是颗粒大小不同。

1．一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程（通常说的主线程）。

2．资源分配给进程，同一进程的所有线程共享该进程的所有资源。

3、线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

4、处理机分给线程，即真正在处理机上运行的是线程。

5、线程是指进程内的一个执行单元，也是进程内的可调度实体。

6、调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位。

7、并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可以并发执行。

8、拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源。

## Python进程

### Python的GIL

[完整介绍](http://cenalulu.github.io/python/gil-in-python/)

GIL全称Global Interpreter Lock；一个防止多线程并发执行机器码的一个Mutex【互斥锁】；首先需要明确的一点是GIL并不是Python的特性，它是在实现Python解析器(CPython)时所引入的一个概念。

为了更有效的利用多核处理器的性能，就出现了多线程的编程方式，而随之带来的就是线程间数据一致性和状态同步的困难。

### Import multiprocessing

所涉及到的对象方法：

['Array', 'AuthenticationError', 'BoundedSemaphore', 'BufferTooShort', 'Condition', 'Event', 'JoinableQueue', 'Lock', 'Manager', 'Pipe', **'Pool'**, **'Process'**, 'ProcessError', **'Queue'**, 'RLock', 'RawArray', 'RawValue', 'SUBDEBUG', 'SUBWARNING', 'Semaphore', 'TimeoutError', 'Value', 'active\_children', 'allow\_connection\_pickling', 'cpu\_count', 'current\_process', 'freeze\_support', 'get\_logger', 'log\_to\_stderr', 'os', 'process', 'sys', 'util']

*#coding=utf-8*

*import os*

*print os.getpid()*

*pid=os.fork()*

*if pid==0:*

*print os.getpid()*

*print 'i am a child process'*

*else:*

*print 'i am a parent process'*

*#* *5677*

*#i am a parent process*

*#5681*

*#i am a child process*

***创建多进程***

*#coding=utf-8*

*import multiprocessing*

*def do(n) :*

*name = multiprocessing.current\_process().name #获取当前进程名*

*print name,'starting'*

*print "worker ", n*

*return*

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

*numList = []*

*for i in xrange(5) :*

*p = multiprocessing.Process(name='ralph'+str(i),target=do, args=(i,))*

*numList.append(p)*

*p.start()*

*p.join() #* *#表示等待子进程结束以后再 继续往下运行，通常用于进程间的同步*

*print "Process end."*

*print numList*

*# 测试单进程和多进程执行效率*

*#coding: utf-8*

*import multiprocessing*

*import time*

*def m1(x):*

*time.sleep(0.01)*

*return x \* x*

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

*pool = multiprocessing.Pool(multiprocessing.cpu\_count()) #获取计算机CPU核数*

*i\_list = range(1000)*

*time1=time.time()*

*pool.map(m1, i\_list)*

*time2=time.time()*

*print 'time elapse:',time2-time1*

*time1=time.time()*

*map(m1, i\_list)*

*time2=time.time()*

*print 'time elapse:',time2-time1*

### 使用进程池批量生成多个进程执行程序

进程池的方法：'apply', 'apply\_async', 'close', 'imap', 'imap\_unordered', 'join', 'map', 'map\_async', 'terminate'

*#coding=utf-8*

*from multiprocessing import Pool*

*def f(x):*

*return x \* x*

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

*pool = Pool(processes = 4) # start 4 worker processes*

*result = pool.apply\_async(f, [10]) # 异步执行 # prints "100"*

*print result.get(timeout = 1)*

*print pool.map(f, range(10)) # prints "[0, 1, 4,..., 81]"*

*#encoding=utf-8*

*import time*

*from multiprocessing import Pool*

*def run(fn):*

*#fn: 函数参数是数据列表的一个元素 time.sleep(1)*

*time.sleep(1)*

*return fn \* fn*

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

*testFL = [1,2,3,4,5,6]*

*print 'Single process execution sequence:' #顺序执行(也就是串行执行，单 进程)*

*s = time.time()*

*for fn in testFL:*

*run(fn)*

*e1 = time.time()*

*print u"顺序执行时间:", int(e1 - s)*

*print 'concurrent:' #创建多个进程，并行执行*

*pool = Pool(5) #创建拥有5个进程数量的进程池 #testFL:要处理的数据列表，run:处理testFL列表中数据的函数*

*rl =pool.map(run, testFL)*

*pool.close()#关闭进程池，不再接受新的进程 pool.join()#主进程阻塞等待子进程的退出*

*e2 = time.time()*

*print u"并行执行时间:", int(e2 - e1)*

*print rl*

*Single process execution sequence:*

*顺序执行时间: 6*

*concurrent:*

*并行执行时间: 2*

*[1, 4, 9, 16, 25, 36]*

### 同步与异步

### 队列

from multiprocessing import Queue

[‘cancel\_join\_thread', 'close', **'empty'**, 'full', **'get'**, 'get\_nowait', 'join\_thread', **'put'**, 'put\_nowait', 'qsize']

from multiprocessing import JoinableQueue

['cancel\_join\_thread', 'close', 'empty', 'full', 'get', 'get\_nowait', **'join'**, 'join\_thread', 'put', 'put\_nowait', 'qsize', '**task\_done**']

multiprocessing.Queue类似于queue.Queue， 一般用来多个进程间交互信息。Queue是进程和线 程安全的。它实现了queue.Queue的大部分方法， 但task\_done()和join()没有实现。 multiprocessing.JoinableQueue是 multiprocessing.Queue的子类，增加了 task\_done()方法和join()方法。

**task\_done()**:用来告诉queue一个task完成。一 般在调用get()时获得一个task，在task结束后调用 task\_done()来通知Queue当前task完成。

**join()**:阻塞直到queue中的所有的task都被处理 (即task\_done方法被调用)

*#encoding=utf-8*

*from multiprocessing import Process, Queue*

*def offer(queue):*

*# 入队列*

*queue.put("Hello World")*

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

*# 创建一个队列实例*

*q = Queue()*

*p = Process(target = offer, args = (q,))*

*p.start()*

*print q.get() # 出队列*

*p.join()*

*进程间通信*

*#encoding=utf-8*

*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 not q.empty():*

*# if not q.empty():*

*print 'Get %s from queue.' % q.get(True)*

*time.sleep(1) # 目的是等待写队列完成*

*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.join()*

*# encoding=utf-8*

*# 进程同步，使用Queue& JoinableQueue*

*import multiprocessing*

*import time*

*class Consumer(multiprocessing.Process):*

*# 派生进程*

*def \_\_init\_\_(self, task\_queue, result\_queue):*

*multiprocessing.Process.\_\_init\_\_(self )*

*self.task\_queue = task\_queue*

*self.result\_queue = result\_queue*

*# 重写原进程的run方法*

*def run(self):*

*proc\_name = self.name*

*while True:*

*next\_task = self.task\_queue.get()*

*if next\_task is None:*

*# Poison pill means shutdown*

*print ('%s: Exiting' % proc\_name)*

*self.task\_queue.task\_done()*

*break*

*print ('%s: %s' % (proc\_name, next\_task))*

*answer = next\_task() # \_\_call\_\_()*

*self.task\_queue.task\_done()*

*self.result\_queue.put(answer)*

*return*

*class Task(object):*

*def \_\_init\_\_(self, a, b):*

*self.a = a*

*self.b = b*

*def \_\_call\_\_(self):*

*time.sleep(0.1) # pretend to take some time to do the work*

*return '%s \* %s = %s' % (self.a, self.b, self.a \* self.b)*

*def \_\_str\_\_(self):*

*return '%s \* %s' % (self.a, self.b)*

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

*# Establish communication queues*

*tasks = multiprocessing.JoinableQueue()*

*results = multiprocessing.Queue()*

*# Start consumers*

*num\_consumers = multiprocessing.cpu\_count()*

*print ('Creating %d consumers' %num\_consumers)*

*# 创建cup核数量数量个的子进程*

*consumers = [ Consumer(tasks, results) for i in range(num\_consumers)]*

*# 依次启动子进程*

*for w in consumers:*

*w.start()*

*# Enqueue jobs*

*num\_jobs = 10*

*for i in range(num\_jobs):*

*# print i*

*tasks.put(Task(i, i))*

*# Add a poison pill for each consumer*

*for i in range(num\_consumers):*

*# print i*

*tasks.put(None)*

*# Wait for all of the tasks to finish*

*tasks.join()*

*# Start printing results*

*while num\_jobs:*

*result = results.get()*

*print 'Result: %s' %result*

*num\_jobs -= 1*

### 管道pip

>>> p=multiprocessing.Pipe()

>>> p

(<read-write Connection, handle 6>, <read-write Connection, handle 7>)

>>> type(p)

<type 'tuple'>

>>> type(p[0])

<type '\_multiprocessing.Connection'>

>>> p=mp.Pipe()

>>> p[0].send('ll')

>>> p[1].recv()

'll'

>>> p[1].send('gg')

>>> p[0].recv()

'gg'

双向：p[0]发，p[1]收

['close', 'closed', 'fileno', 'poll', 'readable', 'recv', 'recv\_bytes', 'recv\_bytes\_into', 'send', 'send\_bytes', 'writable']

### 多进程共享数据

*from multiprocessing import Process, Value, Array*

*def f(n):*

*n.value+=1*

*num=Value('d')*

*for i in range(3):*

*p=Process(target=f,args=(num,))*

*p.start()*

*p.join()*

*print num.value*

*#加锁同步*

*import time*

*from multiprocessing import Process, Value, Lock*

*class Counter(object):*

*def \_\_init\_\_(self, initval = 0):*

*self.val = Value('i', initval)*

*self.lock = Lock()*

*def increment(self):*

*with self.lock:*

*self.val.value += 1 # 共享变量自加1*

*def value(self):*

*with self.lock:*

*return self.val.value*

*def func(counter):*

*for i in range(50):*

*time.sleep(0.01)*

*counter.increment()*

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

*counter = Counter(0)*

*procs = [Process(target =func, args = (counter,)) for i in range(10)]*

*for p in procs:*

*p.start()*

*for p in procs:*

*p.join()*

*print counter.value()*

*# 多进程共享字符串变量*

*#encoding=utf-8*

*from multiprocessing import Process, Manager, Value*

*from ctypes import c\_char\_p*

*def greet(shareStr):*

*shareStr.value = shareStr.value + ", World!"*

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

*manager = Manager()*

*shareStr = manager.Value(c\_char\_p, "Hello")*

*for i in range(4):*

*process = Process(target = greet, args = (shareStr,))*

*process.start()*

*process.join()*

*print shareStr.value*

*Manager()函数返回一个管理对象 ，它控制了一个服务端进程，用来 保持Python对象，并允许其它进程 使用代理来管理这些对象。 Manager()返回的管理者，支持类 型包括，list, dict, Namespace, Lock,RLock, Semaphore, BoundedSemaphore,Condition, Event, Queue, Value and Array。 managers比使用共享内存对象更灵 活，因为它支持任意对象类型。 同样的，单个的manager可以通过 网络在不同机器上进程间共享。但 是，会比共享内存慢。*

*# 多进程共享不同类型的数据结果对象*

*#encoding=utf-8*

*from multiprocessing import Process, Manager*

*def f( shareDict, shareList ):*

*shareDict[1] = '1'*

*shareDict['2'] = 2*

*shareDict[0.25] = None*

*shareList.reverse() # 翻转列表*

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

*manager = Manager()*

*shareDict = manager.dict() # 创建共享的字典类型*

*shareList = manager.list( range( 10 ) ) # 创 建共享的列表类型*

*p = Process( target = f, args = ( shareDict,shareList ) )*

*p.start()*

*p.join()*

*print shareDict*

*print shareList*

*# 进程间共享实例对象*

*import time, os*

*import random*

*from multiprocessing import Pool, Value, Lock, Manager*

*from multiprocessing.managers import BaseManager*

*class MyManager(BaseManager):*

*pass*

*def Manager():*

*m = MyManager()*

*m.start()*

*return m*

*class Counter(object):*

*def \_\_init\_\_(self, initval=0):*

*self.val = Value('i', initval)*

*self.lock = Lock()*

*def increment(self):*

*with self.lock:*

*self.val.value += 1*

*def value(self):*

*with self.lock:*

*return self.val.value*

*# 将Counter类注册到Manager管理类中*

*MyManager.register('Counter', Counter)*

*def long\_time\_task(name,counter):*

*time.sleep(0.2)*

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

*start = time.time()*

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

*for i in range(50):*

*time.sleep(0.01)*

*counter.increment()*

*end = time.time()*

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

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

*manager = Manager()*

*# 创建共享Counter类实例对象的变量，Counter类的初始值0*

*counter = manager.Counter(0)*

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

*p = Pool()*

*for i in range(5):*

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

*print 'Waiting for all subprocesses done...'*

*p.close()*

*p.join()*

*print 'All subprocesses done.'*

*print counter.value()*

*# 共享实例模板*

*from multiprocessing import Pool, Value, Lock, Manager*

*import os, time, random*

*def long\_time\_task(name,requestCount,countList):*

*requestCount.value = requestCount.value + 1*

*print u"计数:", requestCount.value*

*countList.append(requestCount.value)*

*time.sleep(0.2)*

*print 'Run task %s (%s)...\n' % (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\_\_':*

*manager = Manager()*

*requestCount = manager.Value('i',0)*

*countList = manager.list([])*

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

*p = Pool()*

*for i in range(5):*

*p.apply\_async(long\_time\_task, args = (str(i),requestCount,countList))*

*print 'Waiting for all subprocesses done...'*

*p.close()*

*p.join()*

*print 'All subprocesses done.'*

*print requestCount.value*

*print countList*

*# 进程日志*

*import multiprocessing*

*import logging*

*import sys*

*def worker():*

*print 'I am working....'*

*sys.stdout.flush()*

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

*# 设置日志输出到控制台*

*multiprocessing.log\_to\_stderr()*

*logger = multiprocessing.get\_logger()*

*# 设置输出日志的级别*

*logger.setLevel(logging.INFO)*

*p = multiprocessing.Process(target = worker)*

*p.start()*

*p.join()*

*守护进程就是不阻挡主进程退出,会随着主进程退出而退出，如果要等待守护进程退 出，需要加上join函数。*

*#守护进程*

*import multiprocessing*

*import time, logging*

*import sys*

*def daemon():*

*p = multiprocessing.current\_process()*

*print 'Starting:', p.name, p.pid*

*sys.stdout.flush() # 将缓冲区数据写入终端 #*

*time.sleep(2)*

*print 'Exiting :', p.name, p.pid*

*sys.stdout.flush()*

*def non\_daemon():*

*p = multiprocessing.current\_process()*

*print 'Starting:', p.name, p.pid*

*sys.stdout.flush()*

*print 'Exiting :', p.name, p.pid*

*sys.stdout.flush()*

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

*# 设置日志输出到控制台*

*multiprocessing.log\_to\_stderr()*

*logger = multiprocessing.get\_logger() # 设置输出日志的级别*

*logger.setLevel(logging.DEBUG)*

*d = multiprocessing.Process(name='daemon', target=daemon)*

*d.daemon = True*

*n = multiprocessing.Process(name='non-daemon', target=non\_daemon)*

*n.daemon = False*

*d.start()*

*time.sleep(1)*

*n.start()*

*# d.join(1)*

*# n.join()*

*print 'd.is\_alive()', d.is\_alive()*

*print "n.is\_alive()", n.is\_alive()*

*print "main Process end!"*