利用Python进行数据分析(末尾)

其他Python笔记(末尾)

WxPython窗口编程笔记(末尾)

目 录

[利用Python进行数据分析 7](#_Toc523144087)

[Ch03: IPython 7](#_Toc523144088)

[Ch04： Numpy 7](#_Toc523144089)

[Ch05-1： Pandas 7](#_Toc523144090)

[Seires 7](#_Toc523144091)

[DataFrame 8](#_Toc523144092)

[索引对象 8](#_Toc523144093)

[索引、选取和过滤 8](#_Toc523144094)

[算术运算和数据对齐 9](#_Toc523144095)

[DataFrame与Series之间的运算 9](#_Toc523144096)

[函数应用和映射 9](#_Toc523144097)

[排序和排名 9](#_Toc523144098)

[插入和删除 9](#_Toc523144099)

[汇总和计算描述统计 10](#_Toc523144100)

[相关系数与协方差 10](#_Toc523144101)

[唯一值、值计数以及成员资格P157 10](#_Toc523144102)

[处理缺失数据 10](#_Toc523144103)

[滤除缺失数据 10](#_Toc523144104)

[填充缺失数据 10](#_Toc523144105)

[层次化索引 10](#_Toc523144106)

[重排分级顺序 11](#_Toc523144107)

[根据级别汇总统计 11](#_Toc523144108)

[使用DataFrame的列 P169 11](#_Toc523144109)

[整数索引 11](#_Toc523144110)

[CH06 数据加载、存储与文件格式 11](#_Toc523144111)

[读写文本格式的数据 11](#_Toc523144112)

[逐块读取文本文件 11](#_Toc523144113)

[将数据写出到文本格式 12](#_Toc523144114)

[手工处理分隔符格式 12](#_Toc523144115)

[JSON数据(JavaScript Object Notation) 12](#_Toc523144116)

[XML和HTML:Web信息收集 12](#_Toc523144117)

[利用lxml.objectify解析XML 12](#_Toc523144118)

[二进制数据格式 12](#_Toc523144119)

[使用HDF5格式（not working） 13](#_Toc523144120)

[使用HTML和Web APi 13](#_Toc523144121)

[使用数据库 13](#_Toc523144122)

[存取MongoDB中的数据 13](#_Toc523144123)

[CH07 数据规整化：清理、转换、合并、重塑 13](#_Toc523144124)

[数据库风格的DataFrame合并 13](#_Toc523144125)

[索引上的合并 13](#_Toc523144126)

[轴向连接P206 13](#_Toc523144127)

[合并重叠数据P210 13](#_Toc523144128)

[重塑和轴向旋转 13](#_Toc523144129)

[重塑层次化索引（索引旋转和还原） 13](#_Toc523144130)

[将“长格式”旋转为“短格式”P213 14](#_Toc523144131)

[数据转换 14](#_Toc523144132)

[移除重复数据 14](#_Toc523144133)

[利用函数或映射进行数据转换P217 14](#_Toc523144134)

[替换值 14](#_Toc523144135)

[重命名轴索引 14](#_Toc523144136)

[离散化和面元划分 14](#_Toc523144137)

[检测和过滤异常值（孤立点或离群值,outlier） 15](#_Toc523144138)

[排列和随机采样 15](#_Toc523144139)

[计算指标/哑变量 15](#_Toc523144140)

[字符串操作 16](#_Toc523144141)

[正则表达式(regex) 16](#_Toc523144142)

[pandas中矢量化的字符串函数P234 17](#_Toc523144143)

[例子：USDA食品数据库 17](#_Toc523144144)

[CH08绘图和可视化 17](#_Toc523144145)

[Figure 17](#_Toc523144146)

[Subplot 17](#_Toc523144147)

[颜色,标记和线性 17](#_Toc523144148)

[刻度、标签和图例 18](#_Toc523144149)

[注解及在subplot上绘图 18](#_Toc523144150)

[图形绘制 18](#_Toc523144151)

[将图表保存到文件 18](#_Toc523144152)

[matplotlib配置P255 19](#_Toc523144153)

[pandas的绘图函数 19](#_Toc523144154)

[线型图 19](#_Toc523144155)

[柱状图 19](#_Toc523144156)

[直方图和密度图 19](#_Toc523144157)

[散步图 19](#_Toc523144158)

[绘制地图：图形化显式海地地震危机数据 20](#_Toc523144159)

[Python图形化工具生态系统 20](#_Toc523144160)

[CH09 数据聚合与分组运算 20](#_Toc523144161)

[GroupBy技术(P264) 20](#_Toc523144162)

[对分组进行迭代 20](#_Toc523144163)

[选取一个或一组列 20](#_Toc523144164)

[通过字典或Series进行分组 20](#_Toc523144165)

[通过函数进行分组 20](#_Toc523144166)

[根据索引级别分组 21](#_Toc523144167)

[数据聚合 21](#_Toc523144168)

[面向列的多函数应用 21](#_Toc523144169)

[以‘无索引’的形式返回聚合数据 21](#_Toc523144170)

[分组级运算和转换 21](#_Toc523144171)

[apply: 一般性的“拆分-应用-合并” 22](#_Toc523144172)

[禁止分组键 22](#_Toc523144173)

[分位数和桶分析 22](#_Toc523144174)

[示例：用特定分组的值填充缺失值 22](#_Toc523144175)

[示例：随机采样和排列 23](#_Toc523144176)

[示例：分组加权平均数和相关系数 23](#_Toc523144177)

[示例：面向分组的线性回归 23](#_Toc523144178)

[透视表 23](#_Toc523144179)

[交叉表P290 24](#_Toc523144180)

[示例：2012选举委员会数据库 26](#_Toc523144181)

[对出资额分组P297 28](#_Toc523144182)

[根据州统计赞助信息 28](#_Toc523144183)

[Chapter 10 时间序列 28](#_Toc523144184)

[日期和时间数据类型及工具 28](#_Toc523144185)

[字符串和datetime的相互转换:str or strftime 28](#_Toc523144186)

[时间序列基础 29](#_Toc523144187)

[索引、选取u、子集构造 29](#_Toc523144188)

[带有重复索引的时间序列 29](#_Toc523144189)

[日期的范围、频率以及移动 30](#_Toc523144190)

[生成日期范围 30](#_Toc523144191)

[频率和日期偏移量 30](#_Toc523144192)

[移动（超前和滞后）数据 30](#_Toc523144193)

[通过偏移量对日期进行移位 30](#_Toc523144194)

[时区处理 31](#_Toc523144195)

[本地化和转换 31](#_Toc523144196)

[操作时区意识型Timestamp对象 31](#_Toc523144197)

[不同时区之间的运算 32](#_Toc523144198)

[时期及算术运算P322 32](#_Toc523144199)

[时期的频率转换 32](#_Toc523144200)

[按季度计算的时期频率P324 32](#_Toc523144201)

[将timestamp转换为Period(及反向过程) 33](#_Toc523144202)

[通过过数组创建PeriodIndex 33](#_Toc523144203)

[重采样及频率转换 33](#_Toc523144204)

[降采样 33](#_Toc523144205)

[OHLC重采样 34](#_Toc523144206)

[通过groupby进行重采样 34](#_Toc523144207)

[升采样和插值 34](#_Toc523144208)

[通过时期进行重采样 34](#_Toc523144209)

[时间序列绘图 34](#_Toc523144210)

[移动窗口函数 35](#_Toc523144211)

[指数加权函数 35](#_Toc523144212)

[二元移动窗口函数 36](#_Toc523144213)

[用户定义的移动窗口函数 36](#_Toc523144214)

[性能和内存使用方面的注意事项 36](#_Toc523144215)

[第11章 金融和经济数据应用 36](#_Toc523144216)

[时间序列及截面对齐 36](#_Toc523144217)

[频率不同的时间序列的运算 37](#_Toc523144218)

[使用Period 37](#_Toc523144219)

[时间和“最当前”数据选取 37](#_Toc523144220)

[拼接多个数据源 38](#_Toc523144221)

[收益指数和累计收益 38](#_Toc523144222)

[分组变换和分析 39](#_Toc523144223)

[分组因子暴露 39](#_Toc523144224)

[十分位和四分位分析 40](#_Toc523144225)

[更多示例应用 41](#_Toc523144226)

[信号前沿分析：简化截面动量投资组合 41](#_Toc523144227)

[期货合约转仓 42](#_Toc523144228)

[期货合约转仓(自，成功) 43](#_Toc523144229)

[移动相关系数与线性回归 44](#_Toc523144230)

[CH10 Numpy高级应用 44](#_Toc523144231)

[ndarray对象的内部机理 44](#_Toc523144232)

[Numpy数据类型体系 44](#_Toc523144233)

[高级数组操作 44](#_Toc523144234)

[数组重塑 44](#_Toc523144235)

[C和Fortran顺序 45](#_Toc523144236)

[数组的合并和拆分 45](#_Toc523144237)

[堆叠辅助类：r\_和c\_ 45](#_Toc523144238)

[元素的重复操作：tile和repeat 45](#_Toc523144239)

[花式索引及其等价函数: take/put 46](#_Toc523144240)

[广播 46](#_Toc523144241)

[沿其他轴向广播 46](#_Toc523144242)

[通过广播设置数组的值 47](#_Toc523144243)

[ufuc高级应用 47](#_Toc523144244)

[ufunc实例方法 47](#_Toc523144245)

[自定义ufunc 48](#_Toc523144246)

[结构化和记录式数组 48](#_Toc523144247)

[为什么要用结构化数组 49](#_Toc523144248)

[间接排序: argsort和lexsort 49](#_Toc523144249)

[其他排序方法 50](#_Toc523144250)

[numpy.searchsorted在有序数组中查找元素 50](#_Toc523144251)

[Numpy的matrix类 50](#_Toc523144252)

[高级数组输入输出 51](#_Toc523144253)

[内存映像文件 51](#_Toc523144254)

[HDF5及其他数组存储方式 51](#_Toc523144255)

[性能建议 51](#_Toc523144256)

[连续内存的重要性 52](#_Toc523144257)

[其他加速手段: Cython, f2py, C 52](#_Toc523144258)

[例子：数据处理和画图 52](#_Toc523144259)

[例子：mmap使用测试程序 55](#_Toc523144260)

[a 56](#_Toc523144261)

[a 56](#_Toc523144262)

[其他PYTHON笔记 57](#_Toc523144263)

[Python异常处理 57](#_Toc523144264)

[获取日期和时间 59](#_Toc523144265)

[Pandas学习 59](#_Toc523144266)

[工厂函数和内建函数区别 62](#_Toc523144267)

[python3.52 八进制 62](#_Toc523144268)

[enumerate用法 62](#_Toc523144269)

[ini解析 62](#_Toc523144270)

[Multi-files Open 63](#_Toc523144271)

[feedparser 64](#_Toc523144272)

[threading例程 65](#_Toc523144273)

[socket Client例程 71](#_Toc523144274)

[socket Server例程 73](#_Toc523144275)

[logger例程 75](#_Toc523144276)

[SQL知识教程 75](#_Toc523144277)

[SQL快速参考 91](#_Toc523144278)

[a 93](#_Toc523144279)

[a 93](#_Toc523144280)

[a 93](#_Toc523144281)

[a 93](#_Toc523144282)

[a 93](#_Toc523144283)

[a 93](#_Toc523144284)

[a 93](#_Toc523144285)

[a 93](#_Toc523144286)

[WXPYTHON窗口编程说明 94](#_Toc523144287)

[开始 94](#_Toc523144288)

[一个简单的文本编辑器 94](#_Toc523144289)

[添加一个菜单栏 94](#_Toc523144290)

[事件处理 95](#_Toc523144291)

[对话框 97](#_Toc523144292)

[可能的扩展 97](#_Toc523144293)

[可视化控件的布局 98](#_Toc523144294)

[Validator/验证器 101](#_Toc523144295)

[加强布局-使用Sizer 105](#_Toc523144296)

[一个可以运行的例子 108](#_Toc523144297)

## 利用Python进行数据分析

### Ch03: IPython

1. 每次使用some\_lib都重新load，否则会使用some\_lib的一个引用

Import some\_lib

Reload (some\_lib)

如果不行，from imp import reload, or 重启Python

1. 测试代码的执行时间: %time和%timeit
2. (P83)基本性能分析: %prun和%run –p
3. %run module.py #运行脚本
4. %run –d module.py #调试脚本
5. b line\_numer #在line\_number增加断点
6. clear #清楚全部断点(如不清除，再次-d可能会出错)
7. h #help
8. help command
9. c #continue
10. q #quit
11. b num #break at line num
12. b path/to/file.py: num #break file
13. s #step into
14. n #next
15. u(p)/d(own) #在函数调用栈中向上或向下移动
16. a(rgs) #显式当前函数的参数
17. debug statement
18. l(ist) statement #显式当前行
19. w(here) #打印当前完整栈跟踪
20. !var #show variable value

### Ch04： Numpy

1. numpy.array(list), 定义数组。为数组推荐适合类型。数据类型保存在arr.dtype里
2. arr.ndim, 数组一维度2
3. arr.shape, 数组二维度(2, 4)
4. numpy.zeros((1, 2))和numpy.ones((2, 3))可以新建全0或全1数组。
5. numpy.empty((1, 3, 4))，返回未初始化数组
6. nummpy.arange(15)，产生array([0, 1, …,14])
7. float\_arr = int\_arr.astype(np.float64)，显式的把int型转换成float型，支持字符串转换。(P98)
8. numpy.random.randn(7,4)，生成7x4的矩阵
9. 数组的切片不会生成新数组；但花式索引会生成新数组

### Ch05-1： Pandas

### Seires

1. obj = Series([list])生成带标号（索引）的Series
2. obj.values, obj.index，求值或索引。可以对index属性赋值obj.index = [‘a’, ‘b’,’c’]
3. obj2= Series([4,7,-2,2], index=[‘a’,’b’, ‘f’, ‘d’])，指定索引
4. Series数组可以进行Numpy数组运算
5. obj3=Series(dict)，可以通过字典创建Series
6. obj4 = Series(o\_dict, index = o\_list)，生成以o\_list为索引的数组，值在o\_dict里查找；如果没找到则为NaN。
7. pd.isnull(obj4)和pd.notnull(obj4)可以检测缺失数据。也可用方法调用: obj4.isnull(), obj4.notnull()
8. Series最重要的功能：在算数运算中自动对齐不同索引数据。
9. Series对象本身和索引都有name属性

### DataFrame

1. DataFrame有四个重要的属性：index：行索引。columns：列索引。values：值的二维数组。name：名字。
2. DataFrame是表格型的数据结构，既有行索引，也有列索引。可以看成由Series组成的字典（共同用一个索引），即多个Series组成DataFrame。
3. DataFrame(data, columns=[‘year’,’state’,’pop’])，可以按照year, state, pop顺序排列
4. 可以通过字典frame[‘year’]或属性frame.year的方式获取列数值
5. DataFrame的行数据可以通过索引字段ix获取，比如frame.ix[‘three’]
6. 可以赋值frame[‘debt’]=1/list/，对于空字段自动补NaN。’debt’是列的索引。对不存在的列赋值，会新建一个列
7. del frame2[‘year’]用于删除列
8. 通过索引返回的列是数据的视图，全部修改都会反映到源DataFrame上。
9. frame.columns, 返回表格列索引。
10. 把‘字典的嵌套’传给DataFrame，解释成外层字典的键作为列索引，内层键作为行索引。
11. frame.T转置。
12. DataFrame构造器数据：Page134

### 索引对象

1. Series和DataFrame的索引对象Index是不可修改的(immutable)，这样才可以在多个数据结构间共享。
2. pd.Index(Series/List/DataFrame)可以得到对应的index值，不可修改。
3. pandas中主要的Index对象 P136
4. Index的方法和属性。P137
5. obj.reindex=([‘a’,’b’,’c’,’d’,’e’],fill\_value=0)可以重新产生索引，空值填0
6. obj.reindex(range(6),method=’ffill’)可以把Index的数据前向填充，适合时间序列。
7. reindex的插值method选项：ffill/pad：前向搬运；bfill/backfill：后向搬运
8. frame.reindex(index=[list…], method=’ffill’, columns=[list]), 行列同时重新索引，但ffill只能对行进行（即轴0）

### 索引、选取和过滤

1. Series索引类似Numpy数组，不过Series的索引值不只是整数。
   1. Obj\_Series[[‘b’,’c’]]，注意两个方括号，可以取b和c两个索引的值
   2. Obj[obj>2]，取obj>2的值
   3. Obj[‘b’:’c’]=5，利用索引的切片进行赋值或运算
2. DataFrame索引是获取一个或多个列值
   1. frame[0:1], frame.ix[‘line\_index’]进行行索引和切片
   2. frame[‘col\_index1’, ‘col\_index2’,…]进行列索引，列不能切片
3. 通过切片或布尔来选取行：
   1. frame[:2], frame[2:6]
   2. frame[frame[‘col\_index’]>5]，选取col\_index大于5的所有行
4. ix方法选取行
   1. frame.ix([row\_index1, row\_index2 ],[col\_index1, col\_index2])取行列两个列表组成的dataframe。
   2. .ix[3]，选取单行时，会把数据转成列模式显式
   3. .ix[[row\_index1, row\_index2 ],[col\_index1, col\_index2]]，会按原格式选取。
5. DataFrame的索引选项P144

### 算术运算和数据对齐

1. Pandas可以对不同索引的对象进行算数运算，如果存在不同索引，则结果的索引就是该索引对的并集。（P144）
2. 在算数方法中填充值，dfe.add(df2,fill\_value=0)。利用DataFrame里的add方法(修改填充参数)
3. Df1.reindex(columns=df2.columns, fill\_value=0)。根据另一个数组(Series, DataFrame)的列索引修改本数组，新增值填0

### DataFrame与Series之间的运算

1. DataFrame运算的广播(broadcasting)，会把某一行的运算‘广播’到所有行
2. DataFrame和Series之间的算术运算会将Sereis的索引匹配到DataFrame的列，然后沿着行一直向下广播。
3. 如果希望沿着列广播，则需使用算术运算方法：frame.sub(series3, axis=0)

### 函数应用和映射

1. DataFrame的apply方法可以将函数应用到由各列或行形成的一维数组上。
   1. f = lambda x:x.max()-x.min()
   2. frame.apply(f) #应用到列
   3. frame.apply(f, axis=1) #应用到行
2. 元素级的Python函数调用
   1. format = lambda x: ‘%.2f’ % x
   2. frame.applymap(format)

### 排序和排名

1. DataFrame调用sort\_index方法，返回一个按index排序的新对象
   1. frame.sort\_index() #行index排序
   2. frame.sort\_index(axis=1, ascending=False) #列index排序，降序
   3. frame.sort\_values(by=[‘a’,’b’])，先对a列排序，再对b列排序
2. Series调用order()方法，返回根据值排序的对象
3. DataFrame对多个列排序
   1. frame.sort\_index(by=[‘a’,’b’])，先对a列排序，再对b列排序
4. Series的rank方法可以产生排名
   1. obj.rank(ascending = False, method=’max’, axis=1)，降序排名，对行排名， max=使用分组的最大排名，解释见P152

### 插入和删除

1. columnA=df.pop(‘index\_columnA’), pop方法可以把一列取出赋值给其他对象，并从df中删除index\_columnA
2. df[‘new\_column’]=[1,2,’lisa’,20170912]，在末尾增加一个新列，列名为’new\_column’
3. df.loc[2]=[3,'Lisa',12,20170902]，在末尾增加一个新行
4. df.insert(1,’date’,date)，在第1列插入一个新列date
5. 没有找到DataFram插入行的函数和方法；所以先切割，再拼接。
   1. above=df.loc[:2]
   2. below=df.loc[3:]
   3. insertDF=pd.DataFrame([1,2,’lisa’,20170304],columns=[‘rank’,’time’,’name’,’date’])
   4. newDF=pd.concat([above,insertDF,below],ignore\_index=True)

带有重复值的轴索引

1. Obj = Series(range(5),index=[‘a’,’b’,’a’,’c’,’b’,’b’])。索引可以重复。
2. Obj.index.is\_unique，is\_unique属性告诉他的索引值是否唯一
3. 对于DataFrame也是一样

### 汇总和计算描述统计

1. 基于没有缺失数据而构建的一组方法。
2. df.sum()，按列求和
3. df.mean(axis=1,skipna=False)，按行求和，不要跳过NA(有NA的就记为NaN)
4. df.idxmax(), df.idmin()，最大值或最小值的索引
5. df.cumsum(),???
6. df.describe()，多个汇总统计，针对数值和非数值型数据。

### 相关系数与协方差

1. DataFrame的corr和cov方法返回DataFrame形式的相关系数或协方差矩阵
   1. returns.corr(),returns.cov()
   2. framesA.corrwith(framesB)，计算framesA与framesB的相关系数

### 唯一值、值计数以及成员资格P157

1. series\_obj.unique()，unique()生成唯一值数组
2. seriesA.sort()，对序列排序
3. seriesA.value\_counts()，统计Series各值出现频率;
4. pd.value\_counts(obj\_series.values, sort=False)，value\_counts()还是pd的顶级方法，可以直接调用
5. mask = seriesA.isin([‘b’,’c’])，生成True/False的Series，然后筛选seriesA[mask]

### 处理缺失数据

1. data\_series = Series([‘a’,np.nan,None,’sksldf’]), np.nan和None都会被当作NaN(Not a Number)处理
2. data\_series.isnull()，生成True/False Series。
3. .dropna(),.fillna(),.isnull(),notnull(), P160

### 滤除缺失数据

1. from numpy import nan as NA,引入NaN
2. seriesA.dropna()滤除NaN数据，或seriesA[seriesA.dotnull()]
3. dataframeA.dropna()滤除包含NaN的行
4. dataframeA.dropna(how=’all’)只滤除全NaN的行
5. dataframeA.dropna(axis=1,how=’all’)只滤除全NaN的列
6. df.dropna(thresh=3),保留3列不为NaN的数据行

### 填充缺失数据

1. from numpy import nan as NA,引入NaN
2. df.fillna(0)，NaN值填0
3. df.fillna({1:0.5,3:-1})，列index1填0.5，索引为3的列填-1
4. fillna会返回新对象；
5. .ix is deprecated. Please use .loc for label based indexing or .iloc for positional indexing
6. df2 = df.fillna(0, inplace = True), inplace直接在df上修改。df2为空值，无效。
7. df.fillna(method = ‘ffill’,limit =2),对reindex有效的插值方法对fillna也适用。’ffill’拷贝前值；’limit’拷贝前值的数量。
8. Seriesdata.fillna(Seriesdata.mean())，把均值赋给NA值
9. fillna的参数P164

### 层次化索引

1. 以低维度形式处理高维度数据
2. Series行轴可以分层（没有列轴）data =Series(np.random.randn(10),
3. index=[['a','a','a','b','b','b','c','c','d','d'],
4. [1,2,3,1,2,3,1,2,2,3]])
5. DataFrame可以行轴和列轴都可以有分层索引,每个层都可以有名字，如
   1. frame.index.names=[‘key1’,’key2’]
   2. frame.columns.names = ['state','color']

### 重排分级顺序

1. frame.swaplevel(‘key1’,’key2’)，swaplevel接受两个级别编号或名称，返回一个互换级别的新对象（数据不变化）。或frame.swaplevel(0,1)
2. frame.sortlevel(0)，以第0级别编号（名称）为索引，对行编号。

### 根据级别汇总统计

1. frame.sum(level=‘key2’)，根据行级别来进行求和。
2. frame.sum(level=’color’,axis=1)，根据列级别求和

### 使用DataFrame的列 P169

1. frame.set\_index([‘c’,’d’])，会将DataFrame的一个或多个列（‘c’列和‘d’列）当作行索引来用，或者将行索引变为列
2. frame.set\_index([‘c’,’d’],drop=False)，drop会把c和d列保留
3. frame.reset\_index()，reset\_index()方法会把‘层次化索引’转移到列里。

### 整数索引

1. ser=Series(np.arange(3)); ser[-1]会报错。一般整数索引会有歧义。
2. ser2=Series(np.arange(3),index=[‘a’,’b’,’c’])；ser[-1]正确。

面板数据

1. panel数据结构是三维的DataFrame，pangel的每一项（类似DataFrame的列）都是一个DataFrame。
2. DataFrame有to\_panel()方法和to\_frame()方法

### CH06 数据加载、存储与文件格式

### 读写文本格式的数据

1. read\_csv, read\_table, read\_fwf, read\_clipboard.P174
2. 文件读取函数的选项一般包含：
   1. 索引
   2. 类型推断(type inference)和数据转换：用户定义值转换、缺失数据标记
   3. 日期解析：包括组合功能。
   4. 迭代：支持对大文件逐块迭代
   5. 不规整数据处理：跳过一些行、页脚、注释或其他不重要的东西
3. df=pd.read\_csv('ch06/ex1.csv',header=None)，读取’,’分隔的csv数据；header=None针对没有列名的文件
4. df2=pd.read\_table('ch06/ex1.csv',sep=',’)，sep设定分隔符，分隔符可以用正则表达式
5. df=pd.read\_csv('ch06/ex1.csv',names=[‘a’,’b’,’c’,’d’,’message’])，指定列名
6. pd.read\_csv('ch06/ex1.csv',index\_col=’message’)，指定行索引为’message’列的内容；也可用通过index\_col = [‘key1’,’key2’]来指定层次化索引
7. 可以用df=pd.read\_csv('ch06/ex1.csv',skiprows=[0,2,3])来跳过第0，2，3行
8. pd.isnull(dataframe1)，函数可以返回是否有NA的真值表，格式为DataFrame
9. result=pd.read\_csv(‘ch06/ex5.csv’,na\_values=[‘null’])，允许接受NA值
10. pd.read\_csv(‘ch06/ex5.csv’,na\_values=obj\_dict)，可以 用一个字典为各列指定不同的NA标记值，P178
11. len(df),返回datafram的行数.
12. read\_csv/read\_table函数参数列表：P178

### 逐块读取文本文件

1. df=pd.read\_csv('ch06/ex6.csv'，nrows=5)，nrows=5指定读取的行数
2. chunker=pd.read\_csv(‘ch06/ex6.csv’,chunksize=1000)，这样可以利用chunker进行迭代

tot = Series([])

for piece in chunker:

tot = tot.add(piece[‘key’].value\_counts(),fill\_value=0)

tot = tot.order(ascending = False) #.order错误

### 将数据写出到文本格式

1. dataframe1.to\_csv(‘ch06/out.csv’,sep=’|’,na\_rep=‘null’, header=False, index=True, cols=[‘a’,’b’,’c’]), to\_csv()方法输出数据, sep参数指定分隔符；na\_rep参数指定空字符；header指定是否输出表头；index指定是否输出索引;cols指定输出列的顺序
2. series1.to\_csv(‘ch06/tseries.csv’), series也可以用to\_csv输出。
3. df2=Series.from\_csv('ch06/tseries.csv',parse\_dates=True)，使用from\_csv读取series数据

### 手工处理分隔符格式

1. 对于单字符分隔符可以使用python内置的csv模块，将任意打开的文件或文件型对象传给csv.reader：

#ex1

import csv

f=open(‘ch06/ex7.csv’)

reader = csv.reader(f)

#ex2

import csv

lines = list(csv.reader(open('ch06/ex7.csv')))

header, values = lines[0], lines[1:]

data\_dict = {h: v for h, v in zip(header, zip(\*values))}????

1. 修改csv.Dialect的一个子类，可以重新定义csv分隔符
   1. reader = csv.reader(file, dialect=mydialect)
   2. reader = csv.reader(file, delimiter = ‘|’),分隔符=‘|’，根据属性定义csv.reader格式
2. csv语支选项P183

with open(‘mydata.csv’,w) as f:

writer = csv.writer(f, dialect = my\_dialect)

writer.writerow((‘one,’two’,’three’))

writer.writerow((‘1,’2’,’3’))

1. csv.writer接收一个已打开可写的文件对象，以及csv.reader相同的语支和格式化选项

### JSON数据(JavaScript Object Notation)

1. import json，json标准库可以将json字符串转换成python格式
   1. result=json.loads(obj\_json)，将json字符串转换成python格式
   2. asjson=json.dumps(result),将python对象转换成json格式
2. 一般使用方法：向DataFrame构造器传入一组JSON对象，然后选取对象子集
   1. siblings = DataFrame(result[‘siblings’],columns=[‘name’,’age’])
3. panda团队正在开发to\_json和from\_json，

### XML和HTML:Web信息收集

1. from lxml.html import parse, from urllib2 import urlopen, 利用lxml库
2. lxml有多个编程接口，首先用lxml.html处理HTML,然后再用lxml.objectify做一些XML处理

### 利用lxml.objectify解析XML

from lxml import objectify

### 二进制数据格式

1. pandas对象都有一个用于降数据以pickle形式保存在磁盘上的save方法：
   1. df.save(‘ch06/frame\_pickle’)，保存二进制文件
   2. df2=pd.load(‘ch06/frame\_pickle’)，读取二进制文件

### 使用HDF5格式（not working）

python的HDF5有两个接口，h5py提供HDF5 API访问接口，PyTables抽象了HDF5的细节提供多种灵活的数据容器、表索引、查询功能以及核外计算技术

### 使用HTML和Web APi

a

### 使用数据库

a

### 存取MongoDB中的数据

a

### CH07 数据规整化：清理、转换、合并、重塑

### 数据库风格的DataFrame合并

1. pd.merge(df1,df2,on=’key’)，合并df1和df2，on用来指定键key值。
2. pd.merge(df1,df2, how=’outer’).how可以指定连接的方式。inner=交集，outer=并集，left=左连接，right=右连接
3. pd.merge(df1,df2,on=[‘key1’,’key2’],how=’outer’)，outer并集，把两个键值作为元组合并df
4. merge函数的参数P202

### 索引上的合并

1. pd.merge(df1,df2,left\_on=’key’,right\_index=True),right\_index指定把左边df的索引作为键值。
2. df1.join(df2,how=’outer’,on=’key’),join在连接键上左连接
3. df1.join([df2,df3],how=’outer’),把df1，df2，df3连接起来

### 轴向连接P206

1. np.concatenate([arr1,arr2],axis=1), numpy的concatenate轴向连接，把arr1和arr2连接起来
2. pd.concat([s1,s2,s3]),pandas的concat方法可以把三个Series的索引和值依次连接起来，产生一个新Series
3. pd.concat([s1,s2,s3],axis=1),加入axis=1可以把索引连接起来，并生成三个列。产生一个新的DataFrame
4. pd.concat([s1,s4],axis=1,join=’inner’)，join=‘inner’可以得到两个Series的交集，产生Dataframe
5. pd.concat([s1,s4],axis=1,join\_axes=[[‘a’,’c’,’b’,’e’]]),join\_axes指定在其他轴上指定的索引，没有指定的，该索引对应的数据被忽略。
6. result = pd.concat([s1,s2,s3],keys=['one','two','three'])，keys可以指定层次化索引，分别对应三个Series
7. pd.concat([df1,df2],axis=1,keys=[‘level1’,’level2’]，names=[‘upper’,’lower’]), 同样的逻辑对DataFrame对象也是一样，names用来管理层次化索引
8. pd.concat({'level1':df1,'level2':df2},axis=1)，如果传入的值是字典，则字典的键会被当作keys选项的值，相当于层次化索引
9. pd.concat([df1,df2],ignore\_index=True)，可以i忽略跟当前分析工作无关的DataFrame行索引
10. concat函数参数P209: objs, axis, join, join\_axes, keys,levels, names, verify\_integrity, ignore\_index

### 合并重叠数据P210

1. np.where(pd.isnull(seriesA),seriesB,seriesA)，Numpy的where函数，用于表达矢量化的if-else
2. seriesB[:-2].combine\_first(seriesA[2:]), Series的combine\_first方法：对NaN值用SeriesA的值填充，非NaN值不变。用于把一个SeriesA的非NaN值部分赋值到另一个SeriesB的NaN位置。DataFrame也有同样的combine\_first方法。

### 重塑和轴向旋转

reshape,重塑运算；pivot,轴向旋转运算

### 重塑层次化索引（索引旋转和还原）

1. result=df.stack()，将数据列“旋转”为行(原来的行，还是行，形成层次化索引)
2. result.unstack()，默认将数据内层的行“旋转”为列；result.unstack(0),可以对第一层级的索引进行unstack。；result.unstack(‘state’)，可以对nammes=’state’的index名进行旋转。
3. df2.unstack().stack(), stack默认滤除缺失数据，因此该运算时可逆的。
4. df2.unstack().stack(dropna=False)，dropna参数为False，则不滤除NaN数据。
5. 再对DataFrame进行unstack操作时，作为旋转轴的级别将会成为结果中的最低级别。

### 将“长格式”旋转为“短格式”P213

1. pivoted\_data=longdata.pivot(‘date’,’item’,’value’),pivoted.head()就变成通用的DataFrame数据了。第一个参数’date’作为行的index，第二个参数’item’作为列的index。第三个参数作为填充DataFrame的数据列的列名。（参考P214的例子）
2. pivot其实是一个快捷方式：用set\_index创建层次化索引，再用unstack重塑。

### 数据转换

a

### 移除重复数据

1. df.duplicated(),duplicated方法表示各行是否是重复行。返回布尔型Series
2. df.drop\_duplicates([‘k1’]，take\_last=True),drop\_duplicates方法移除重复行；参数[‘k1’]表示只对k1列进行重复项判断；take\_last=True保留最后一个重复数据（默认保留第一个出现的值，移除后面出现的值）

### 利用函数或映射进行数据转换P217

1. df[‘animal’]=df[‘food’].map(str.lower).map(meat\_to\_animal),df[‘animal’]增加一个index为animal的列；map(str.lower)转换为小写字母；map(meat\_to\_animal),其中meat\_to\_animal是一个{food:animal}字典，map可以把对应的animal映射值赋给df[‘animal’]

### 替换值

1. seriesA.replace(-999,np.nan),replace是Series替换的一个方法。将-999替换为np.nan;-999可以用一个list代替，则list里的值被np.nan替换。产生一个新的数据。
2. seriesA.replace([A,B],[C,D])，分别使用C代替A，D代替B
3. seriesA.replace({A：C，B：D})，分别使用C代替A，D代替B

### 重命名轴索引

1. df.index = df.index.map(str.upper),将df的index变为大写。
2. df.rename(index=str.title, columns=str.upper),重新生成一个数据，对index和columns进行操作。
3. df.rename(index={‘AA’:’BB’},columns={‘CC’:’DD’},inplace=True)，重新生成一个df数据，将index AA替换成BB，列名CC替换成DD；inplace=True修改原数据，False新生成一个数据。

### 离散化和面元划分

1. cut函数可以进行数据拆分，cats对象是一个特殊的Categorical对象。包含cats.labes数据进行标号的属性；cats.levels不同分类名称的数组；cats.label=array([0,0,0,1,0,0,2]); cats.levels=Index((18,25], (25,35], (35,60], (60,100],);pd.value\_counts(cats)，返回每组对应数量;cut的参数right=False代表不包含右侧。

ages=[20,22,25,21,23,37]

bins=[18,25,35,100]

cats=pd.cut(ages,bins,right=False)

ages=[20,22,25,21,23,37]

bins=[18,25,35,100]

cats=pd.cut(ages,bins,right=False)

1. 如果向cut传入的是面元的数量，而不是确切的面元边界，他会根据数据的最小值和最大值计算等长面元。
   1. dataArray=np.random.rand(20)
   2. pd.cut(data,4,precision=2)
2. qcut类似cut函数，根据样本分位数对数据进行面元划分。
   1. dataarray=np.random.randn(1000) #正态分布
   2. cats=pd.qcut(dataarray,4) #按4分位进行切割
   3. pd.value\_counts(cat) #每组数据250个。
3. pd.qcut(dataarray,[0,0.1,0.5,0.9,1.]),qcut自定义分位数（0到1之间的数值，包含端点）

### 检测和过滤异常值（孤立点或离群值,outlier）

1. np.random.seed(12345)，设置产生随机的种子
2. df.describe()，统计df数据。
3. col[np.abs(col)>3],找出某列绝对值大于3的数据
4. data[(np.abs(data)>3).any(1)]=np.sign(data)\*3，找出全部绝对值大于3的数据, np.sign()返回数据的符号。

### 排列和随机采样

1. sampler=np.random.permutation(5),产生一组0-4的随机数组
2. df.take(sampler), 按照sampler的顺序，调整行index的顺序。
3. df.take(np.random.permutation(len(df))[:3])，取df的随机3行数据
4. 通过替换的方式产生样本，最快的方式是通过np.random.randint得到一组随机整数。

bag\_arr=np.array([5,7,-1,6,4])

sampler=np.random.randint(0,len(bag\_arr),size=10) #size数量

draws=bag\_arr.take(sampler)

### 计算指标/哑变量

In [41]: df=DataFrame({'key':['b','b','a','c','a','b'], 'data1':range(6)})

In [42]: df1=pd.get\_dummies(df['key'])

In [43]: df

Out[43]:

data1 key

0 0 b

1 1 b

2 2 a

3 3 c

4 4 a

5 5 b

In [44]: df1

Out[44]:

a b c

0 0 1 0

1 0 1 0

2 1 0 0

3 0 0 1

4 1 0 0

5 0 1 0

1. pd有一个get\_dummies函数，可以实现转换。在‘key’对应a的位置置1，其他置零。
2. dummies= pd.get\_dummies(df['key'],prefix='new'),将上面的列名’a’增加前缀’new\_a’
3. df\_with\_dummy = df[['data1']].join(dummies)修改DataFrame列名。df[[‘data1’]]是一个DataFrame，df['data1']是一个series，join()方法可以连接两个DataFrame（前面的需要时DataFrame格式数据）
4. genre\_iter = (set(x.split('|')) for x in movies.genres),df.col\_index可以直接访问对应的列，生成一个迭代器；x.split(‘|’)拆分。python的set是一个无序不重复元素集，基本功能包括关系测试和消除重复元素. 集合对象还支持并、交、差、对称差等。sets 支持 x in set、 len(set)、和 for x in set。作为一个无序的集合，sets不记录元素位置或者插入点。因此，sets不支持 indexing, slicing, 或其它类序列（sequence-like）的操作。
5. genres = sorted (set.union(\*genre\_iter)), sorted()函数对数据排序；set.union()？？？
6. for i, gen in enumerate(movies.genres):dummies.loc[i, gen.split('|')] =1, enumerate()函数遍历序列，返回对应序列序号和值。

#这种方式构建得惰成员指标变量运行速度慢

mnames = ['movie\_id','title','genres']

movies = pd.read\_table('ch02/movielens/movies.dat', sep='::', header=None, names=mnames)

genre\_iter = (set(x.split('|')) for x in movies.genres)

genres = sorted (set.union(\*genre\_iter))

dummies=DataFrame(np.zeros((len(movies), len(genres))),columns=genres)

for i, gen in enumerate(movies.genres):

dummies.loc[i, gen.split('|')] =1

movies\_windic = movies.join(dummies.add\_prefix('Genre\_'))

1. 一个对统计应用有用得秘诀是：结合get\_dummies和cut之类的离散化函数

values = np.random.rand(10)

bins = [0,0.2, 0.4, 0.6, 0.8, 1]

pd.get\_dummies(pd.cut(values,bins))

### 字符串操作

1. str.split(‘,’),split()方法返回分割后的字符串，保存再list里，包含空格。
2. pieces=[x.strip() for x in val.split(‘,’)]，删除空格，保存再list里。
3. ‘：：‘.join(pieces)，字符串的join方法传入一个列表或元组，可以把列表或元组用字符串分开。
4. ‘abc’in str, in关键字可以字串定位，判断是否包含该字符串
5. str.index(‘,’),字符串index方法也可以定位字符串。如果找不到字符串，index会引发一个异常。
6. str.find(':'), 字符串find方法也可以定位字符串。找到返回1，找不到返回-1
7. str.count(‘,’)，count()返回字符串出现的次数
8. str.replace(‘a’,’aaa’)，replace替换字符串.
9. 可以使用正则表达式
10. Python内置字符串方法P229: count,endswith,startwith,join,index,find, replace, strip, rstrip, lstrip, split, lower, upper, ljust, rjust

### 正则表达式(regex)

1. import re，内置的re模块匹配正则表达式
2. re模块分三类：模式匹配，替换以及拆分
3. '\s+'，匹配一个或多个空白符。
4. r'c:\x'，r可以避免'\'转义
5. 如果打算对许多字符串应用同一条正则表达式，强烈建议通过re.compile创建regex对象，可以节省CPU时间

pattern = r'[A-Z0-9.\_%+-]+@[ A-Z0-9.-]+\.[A-Z]{2,4}'

regex = re.compile(pattern, flags=re.IGNORECASE) # re.IGNORECASE忽略大小写

regex.findall(text\_obj) #findall()返回列表，包含所有电子邮件

m=regex.search(text\_obj) #search()返回第一个匹配值的起始和结束的位置

text[m.start(): m.end()] #使用.start()和.end()调用

regex.match(text) #从开始匹配text，

1. pattern = r'([A-Z0-9.\_%+-]+)@([ A-Z0-9.-]+)\.([A-Z]{2,4})',带有分组功能，可以产生分组。{}代表重复次数
2. regex=re.compile(r(?P<username>[A-Z0-9.\_%+-]+)@(?p<DOMAIN>[A-Z0-9.-]+)\.(?P<suffix>[A-Z]{2,4}), flags=re.IGNORECASE|re.VERBOSE), m.groupdict()可以生成字典式分组。
3. 正则表达式方法P233：findall, finditer,match,search,split,sub,subn

### pandas中矢量化的字符串函数P234

1. seriesA.str.contains('gmail'),Series数据的str属性可以访问contains方法，可以返回是否包含字符串。
2. matches=seriesData.str.match(pattern, flags=re.IGNORECASE)，矢量化元素获取操作。matches.str.get(1),matches.str[0]
3. sereiesData.str[:5]，字符串截取。
4. 矢量化字符串方法P235：cat,contains, endswith, startswith, findall, get, join, len, lower, upper, match, pad, center, repeat, replace, slice, split, strip, rstrip, lstrip

### 例子：USDA食品数据库

1. import json,导入ujson库
2. db = json.load(open('ch07/foods-2011-10-03.json')),用json.load打开并导入数据。
3. info\_keys= ['group','id'], info=DataFrame(db,columns=info\_keys)，在将字典列表转换未DataFrame时，可以只抽取其中的一部分字段。columns参数只导入info\_keys对应的数据
4. pd.value\_counts(info.group)[:10]，value\_counts可以查看食物类别分布。info.group是DF中的一列
5. 将全部营养数据做一些分析，最简单的方法是将所有食物的营养成分整合到一个大表中

for rec in db:

fnuts = DataFrame(rec['nutrients']) #拷贝一列

fnuts['id'] = rec['id'] #增加一列

nutrients.append(fnuts) #增加到列表里

nutrients = pd.concat(nutrients, ignore\_index= True) #连接两个列表，忽略index

(NEED MORE WORK HERE)

### CH08绘图和可视化

import matplotlib.pyplot as plt

ipython –pylab or %gui魔术命令

### Figure

1. fig=plt.figure(),创建一个新的figure。
2. 不能通过空Figure绘图，必须通过add\_subplot创建一个或多个subplot才行。
3. ax1 = fig.add\_subplot(2,2,1) #图像是2x2，选择第一个subplot，如果第三个参数是3，就选择第三个窗口
4. plt.plot(np.random.randn(50).cumsum(),'k--') #randn(50)生成50个随机数；randn.cumsum()生成一条随机数的累计曲线，'k--'是划线的属性:绘制黑色虚线图。
5. ax1.hist(randn(100),bins=20, color='k', alpha=0.3) #.hist画柱状图方法
6. ax2.scatter(np.arange(30), np.arange(30)+3\*randn(30)) #.scatter画散点图

### Subplot

1. fig.axes = plt.subplots(2,3) #画子图的简便方法
2. plt.subplots的选项：nrows, ncols, sharex, sharey, subplot\_kw, \*\*fig\_kw

fig, axes = plt.subplots(2, 2, sharex= True, sharey=True) #初始化对象

for i in range(2):

for j in range(2):

axes[i,j].hist(randn(500), bins=50, color ='k', alpha=0.5) #axes[i,j]指定窗口, .hist方法设定柱状图参数

plt.subplots\_adjust(wspace=0,hspace=0) #subplots\_adjust调整子图间距

### 颜色,标记和线性

1. ax.plot(x,y,'g--') #根据x和y绘制绿色虚线
2. ax.plot(x, y, linestyle='--',color='g') #根据x和y绘制绿色虚线
3. plt.plot(randn(30).cumsum(), 'ko--') #'ko'，强调实际数据点。或marker='o'

data = randn(30).cumsum()

plt.plot(data, 'k--', label = 'Default') #label参数设定标签

plt.plot(data, 'k-',drawstyle='steps-post',label='steps-post') #'k-'代表实线，drawstyle划线风格

plt.legend(loc='best')

### 刻度、标签和图例

fig = plt.figure() #初始化对象

ax=fig.add\_subplot(1,1,1) #添加plot面板

ax.plot(randn(1000).cumsum(), 'k--',label='one') #画图

ax.plot(randn(1000).cumsum(), 'k--',label='two')

#ax.plot(randn(1000).cumsum(), 'k--',label='two')

ticks = ax.set\_xticks([0, 250, 500, 750, 1000]) # set\_xticks 设置x轴刻度间距（点数）

labels= ax.set\_xticklabels(['one', 'two', 'three','four','five'], rotation=30,fontsize='small') #x轴刻度名称

ax.set\_title('my plot') #设置表头

ax.set\_xlabel('stages') #设置水平标签

ax.legend(loc='best') #legend画图例'one','two'. 或plt.legend()

### 注解及在subplot上绘图

1. 注解可以通过text,arrow,annotae等函数进行添加
2. ax.text(x,y,'hello world',family='monospace',fontsize=10)

from datetime import datetime

fig=plt.figure()

ax = fig.add\_subplot(1,1,1)

data=pd.read\_csv('ch08/spx.csv',index\_col=0,parse\_dates=True)

spx=data['SPX']

spx.plot(ax=ax, style='k-')

crisis\_data= [(datetime(2007,10,11),'peak of bull market'), #定义坐标点和注释

(datetime(2008,3,12),'bear stearns fails'),

(datetime(2008,9,15),'lehman bankruptcy')]

for date, label in crisis\_data:

ax.annotate(label, xy=(date, spx.asof(date)+50),

xytext=(date,spx.asof(date)+200),

arrowprops=dict(facecolor='black'),

horizontalalignment='left',verticalalignment='top') # annotate添加注释

ax.set\_xlim(['1/1/2007', '1/1/2011']) #x轴局部放大

ax.set\_ylim([600,1800]) #y轴局部放大

ax.set\_title('Important dates in 2008-2009 financial crisis') #设置表头

### 图形绘制

1. 矩形和圆形可以在matplotlib.pyplot中找到，但完整的集合在matplotlib.patches.
2. 在表中添加图形，需要创建一个块对象shp，然后通过ax.add\_patch(shp)添加到subplot中

fig = plt.figure()

ax = fig.add\_subplot(1,1,1)

rect=plt.Rectangle((0.2, 0.75), 0.4, 0.15,color='k',alpha=0.3)

circ = plt.Circle((0.7, 0.2), 0.15,color='b',alpha=0.3)

pgon = plt.Polygon([[0.15,0.15],[0.35,0.4],[0.2,0.6]],color='g',alpha=0.5)

ax.add\_patch(rect)

ax.add\_patch(circ)

ax.add\_patch(pgon)

### 将图表保存到文件

1. plt.savefig('figpath.svg')
2. plt.savefig('figpath.png')
3. plt.savefig('figpath.png', dpi=400, bbox\_inches='tight'), savefig并非一定写入磁盘，也可以写入任何文件型对象，比如StringIO.

from io import StringIO

buffer = StringIO

plt.savefig(buffer)

plot\_data = buffer.getvalue() #对web上提供动态生成的图片是很实用的

1. Figure.saveifg的选项P255: fname, dpi, facecolor, edgecolor, format, bbox\_inches

### matplotlib配置P255

1. plt.rc('figure',figsize=(10,10)) #rc函数可以配置plot全局参数。第一个参数是定义的对象，比如'figure', 'axes', 'xtick', 'ytick','grid','legend'。
2. 可以把选项写成字典。

font\_options = {'family': 'monospace', 'weight':'bold'}

plt.rc('font',\*\*font\_options)

### pandas的绘图函数

pandas绘图更简单

### 线型图

1. Series和DataFrame都有一个用于生成图表的plot方法。默认是线型图。
   1. s=Series(np.random.randn(10).cumsum(), index=np.arange(0,100,10))
   2. s.plot() #Series和Dataframe自带plot方法
   3. plt.savefig('splot.png') #输出到文件
2. df.plot() #DataFrame的plot方法会在一个subplot中为每一列绘制一条线，并创建图例。
3. Series.plot方法参数：label, ax, style, alpha, kind, logy, use\_index, rot, xticks, yticks, ylim, xlim, grid.
4. 专用与DataFrame的plot参数P258：subplots, sharex, sharey, figsize, title, legend, sort\_columns

### 柱状图

1. df.plot(kind='bar',...) #kind='bar'绘制柱状图，kind='barh'绘制水平柱状图。 对于DataFrame，柱状图会将每一行的值分为一组。
2. 柱状图有一个非常不错的用法：利用value\_counts图形化显式Series中各值出现的频率。比如：
   1. s.value\_counts().plot(kind='bar')

### 直方图和密度图

1. seriesA.hist(bins=50) #series的hist方法是画直方图。直方图是一种可以对频率进行离散化显式的柱状图。
2. seriesA.plot(kind='kde') #kind='kbe'，生成概率密度图（标准混合正态分布KDE）

comp1 = np.random.normal(0, 1, size=200)

comp2 = np.random.normal(10, 4, size=200)

values = Series(np.concatenate([comp1, comp2]))

values.hist(bins=100, alpha=0.3, color='k', normed=True)

values.plot(kind='kde', style='k--')

### 散步图

1. 散步图（scatter plot）是观察两个一维数据序列之间的关系的有效手段。matplotlib的scatter方法是主要方法。

plt.scatter(trans\_data['m1'],trans\_data['unemp'])

pd.scatter\_matrix(trans\_data, diagonal='kde', color='k',alpha=0.3)

### 绘制地图：图形化显式海地地震危机数据

P271

### Python图形化工具生态系统

1. Chaco绘图工具包
2. mayavi基于开源C++图形库VTK的3D图形工具包

### CH09 数据聚合与分组运算

### GroupBy技术(P264)

1. means=df['data1'].groupby([df['key1'], df['key2']]).mean() #groupby第一个参数是分组参照列
2. means.unstack() #把层次化索引分为行索引和列索引
3. 分组键可以是任何长度适当的数组

### 对分组进行迭代

1. groupby对象支持迭代，可以产生一组二元元组（由分组名和数据块组成）

for name, group in df.groupby('key1'):

print(name)

print (group)

1. 对于多重键的情况，元组第一个元素是由键值组成的元组

for (k1,k2), group in df.groupby(['key1','key2']):

print(k1,k2)

print (group)

1. pieces=dict(list(df.groupby('key1'))) #把分组后的数据转换成list，然后生成字典
2. grouped=df.groupby(df.dtypes, axis=1) #groupby默认是在axis=0上进行分组的，通过设置

### 选取一个或一组列

1. df.groupby('key1')['data1'] #对于DataFrame产生的GroupBy对象，如果用一个(单个字符)或一组（字符串数组）列名对其进行索引，就能实现选取部分列进行聚合的目的，即df['data1'].groupby(df['key1'])
   1. 例: df.groupby(['key1','key2'])[['data2']].mean() #得到Dataframe格式的结果

### 通过字典或Series进行分组

除了数组以外，分组信息还可以以其他形式存在，比如字典：

people = DataFrame(np.random.randn(5,5),

columns=['a','b','c','d','e'],

index = ['joe','steve','wes','jim','travis'])

people.loc[2:3,['b','c']]=np.nan

mapping = {'a':'red', 'b':'red', 'c':'blue','d':'blue','e':'red','f':'orange'}

by\_column = people.groupby(mapping, axis=1)

by\_column.sum()

也可以用Series作为分组键

map\_series = Seies(mapping)

people.groupby(map\_series, axis=1)

### 通过函数进行分组

people.groupby(len，Key\_list, Key\_dict)).sum() #groupy(函数名，Key\_list, Key\_dict)，可以同时应用函数、字典，列表，列名用作分组键。

### 根据索引级别分组

通过level关键字传入层次化索引级别编号或名称，即可实线根据索引级别进行聚合

columns = pd.MultiIndex.from\_arrays([['us','us','us','jp','jp'],

[1,3,5,1,3]], names=['cty','tenor']) #层次化索引

hier\_df = DataFrame(np.random.randn(4,5), columns=columns)

hier\_df

hier\_df.groupby(level='cty', axis=1).count() #level关键字指定分组键

### 数据聚合

1. 数据聚合只从数组产生标量值的数据转换过程，例如.mean(),.sum(),.quantile(0.9)等函数
2. 如果要实现自己的聚合函数，只需将其传入aggregate或agg方法即可：

def peak\_to\_peak(arr):

return arr.max() - arr.min()

df = DataFrame({'key1': ['a','a','b','b','a'],

'key2': ['one','two','one','two','one'],

'data1':np.random.randn(5),

'data2': np.random.randn(5)})

grouped = df.groupby('key1')

grouped.agg(peak\_to\_peak) #agg(函数名)，实现调用自己的函数

1. 有些方法也可以用在group里，比如grouped.describe()，严格来说并非聚合运算
2. 经过优化的groupby方法P284: count, sum, mean, median, std, var, min, max, prod, first, last

### 面向列的多函数应用

tips = pd.read\_csv('ch08/tips.csv')

tips['tip\_pct'] = tips['tip']/tips['total\_bill']

grouped = tips.groupby(['sex','smoker'])

grouped\_pct = grouped['tip\_pct']

#应用多个函数

grouped\_pct.agg(['mean','std',peak\_to\_peak])

#对函数产生的列重新命名

grouped\_pct.agg([('foo','mean'),('bar',np.std)])

#对所有列应用相同的多个函数

functions = ['count', 'mean','max']

result = grouped['tip\_pct','total\_bill'].agg(functions)

#对不同列应用不同的函数

grouped.agg({'tip':np.max, 'size':['sum',',mean']})

### 以‘无索引’的形式返回聚合数据

a

### 分组级运算和转换

一般的方法：

k1\_means = df.groupby('key1').mean().add\_prefix('mean\_') #先生成新的df

pd.merge(df, k1\_means, left\_on='key', right\_index=True) #合并两个df

灵活一些的方法：

key = ['one', 'two', 'one', 'two','one']

people.groupby(key).mean()

people.groupby(key).transform(np.mean) #根据groupby的结果运行mean()

def demean( arr ):

return arr - arr.mean()

demeaned = people.groupby(key).transform(demean) #transfer也可以接函数

demeaned.groupby(key).mean()

### apply: 一般性的“拆分-应用-合并”

apply会将待处理的对象拆分成多个片段，然后对个片段调用传入的函数，最后尝试将各片段组合在一起。

tips = pd.read\_csv('ch08/tips.csv')

tips['tip\_pct'] = tips['tip']/tips['total\_bill']

def top(df,n=5,column='tip\_pct'):

return df.sort\_values(by=column)[-n:] #sort\_values()排序

top(tips, n=6)

tips.groupby('smoker').apply(top) #apply(top)会在'smoker'分组的两个索引下分别求top,得到两组排序值

tips.groupby(['smoker','day']).apply(top,n=1,column='total\_bill') #apply可以对top函数赋初值

### 禁止分组键

分组键会跟原始对象的索引共同构成结果对象中的层次化索引，将group\_keys=False传入groupby即可禁止该效果

tips.groupby('smoker',group\_keys=False).apply(top) #不会增加'smoker'索引

### 分位数和桶分析

pandas有一些能根据指定面元或样本分位数将数据拆分成多块的工具(cut和qcut)，结合groupby可以实现对数据的桶(bucket)和分位数（quantile）分析

frame = DataFrame({'data1': np.random.randn(1000), 'data2':np.random.randn(1000)}) #生成2维DF数组

factor = pd.cut(frame.data1, 4) #pd.cut把data1数据分为长度相等的4组，得到每组的区间。

factor2 = pd.qcut(frame.data1, 4) #pd.qcut把data1数据按分位数分成4组，得到每组的区间。

def get\_stats(group):

return {'min': group.min(), 'max':group.max(), 'count':group.count(), 'mean':group.mean()}

grouped = frame.data2.groupby(factor) #将data2按data1的分组排列, 为下面求均值做准备

grouped.apply(get\_stats).unstack() #apply(func)按groupby分组运行函数get\_stats, #unstack()把min,max,count,mean转换成列名

### 示例：用特定分组的值填充缺失值

1. 对于缺失数据的清理工作，有时会用dropna将其清楚，有时希望用固定值填充，有时用数据集本身的值（前值 or 后值）填充NA值。需要使用fillna这个工具了，下面的例子中，我们用平均值填充。

s=Series(np.random.randn(6))

s[::2] = np.nan

s.fillna(s.mean()) #填充mean值

1. 需要对不同分组填充不同的值，使用apply+fillna组合。

fill\_mean = lambda g: g.fillna(g.mean()) #利用lambda实现对自身数据集运算mean

data.groupby(group\_key).apply(fill\_mean) #apply+lambda函数实现对自身调用

1. 第三种方法在代码中预定义各组的天充值，由于分组具有一个name属性，所以可以用lamdba调用一下：

fill\_values = {'east':0.5, 'west':-1}

fill\_func = lambda g : g.fillna(fill\_values[g.name]) #??????

data.groupby(group\_key).apply(fill\_func)

### 示例：随机采样和排列

1. np.random.permutation(N)希望从大数据集中随机抽取样本.

suits = ['h','s','c','d']

card\_val = (list(range(1,11))+[10]\*3)\*4

base\_names = ['a']+list(range(2,11))+['j','q','k']

cards=[]

for suit in suits:

cards.extend(str(num)+suit for num in base\_names) # extend() 函数用于在列表末尾一次性追加另一个序列中的多个值

deck = Series(card\_val, index=cards)

def draw( deck, n=5 ):

return deck.take(np.random.permutation(len(deck))[:n]) #随机抽取，df.take(sampler)按sampler的顺序调整df的顺序

draw(deck)

1. 从每种花色中随机抽取两张：

get\_suit = lambda card: card[-1]

deck.groupby(get\_suit).apply(draw,n=2)

1. 关于extend和append:

.append() 向列表尾部追加一个新元素，列表只占一个索引位，在原有列表上增加

.extend() 向列表尾部追加一个列表，将列表中的每个元素都追加进来，在原有列表上增加

.+ 直接用+号看上去与用extend()一样的效果，但是实际上是生成了一个新的列表存这两个列表的和，只能用在两个列表相加上

.+= 效果与extend()一样，向原列表追加一个新元素，在原有列表上增加

### 示例：分组加权平均数和相关系数

根据groupby的“拆分-应用-合并”范式i，DataFrame的列与列之间或两个Series之间的运算（比如分组加权平均）成为一种标准作业。

df = DataFrame ({'category': ['a','a','a','a','b','b','b','b'], 'data':np.random.randn(8),

'weights':np.random.randn(8)})

grouped = df.groupby('category') #根据'category'分组

get\_wavg = lambda g: np.average(g['data'], weights=g['weights']) #lambda匿名函数, get\_wavg函数名

grouped.apply(get\_wavg) #根据分组分别求均值(data\*weights)

计算一个股票的日收益率（百分比）与SPX之间年度相关系数组成的DataFrame

#page297

close\_px=pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

rets=close\_px.pct\_change().dropna() #pct\_change()：Series也有这个函数，这个函数用来计算同colnums两个相邻的数字之间的变化率，并且丢掉NA数据

spx\_corr = lambda x: x.corrwith(x['SPX']) #匿名函数,计算相关系数，

by\_year = rets.groupby(lambda x: x.year) #按年分组

by\_year.apply(spx\_corr) #对分组数据运行相关函数

by\_year.apply(lambda g: g['AAPL'].corr(g['MSFT'])) #匿名函数名.apply(匿名函数名)

### 示例：面向分组的线性回归

P287利用regress函数(statsmodels库)，对各数据块执行普通最小二乘法回归。

### 透视表

透视表（pivot\_table）：根据一个或多个键对数据进行聚合，并根据行和列上的分组键将数据分配到各个矩形区域中。

1. 小费数据集，根据sex和smoker计算分组平均数，并将sex和smoker放到行上：

tips = pd.read\_csv('ch08/tips.csv')

tips['tip\_pct'] = tips['tip']/tips['total\_bill']

tips.pivot\_table(index=['sex','smoker']) #grouped = tips.groupby(['sex','smoker'])

#只想聚合tip\_pct和size，且根据day进行分组。

tips.pivot\_table(['tip\_pct', 'size'], index=['sex','day'],columns='smoker') #index指定行索引，columns指定列索引。['tip\_pct','size']聚合的对象

1. 传入margins=True添加分项小计。添加标签为ALL的行和列，对应单个级别中所有的分组统计：

tips.pivot\_table(['tip\_pct', 'size'], index=['sex','day'],columns='smoker',margins=True) #All为平均数

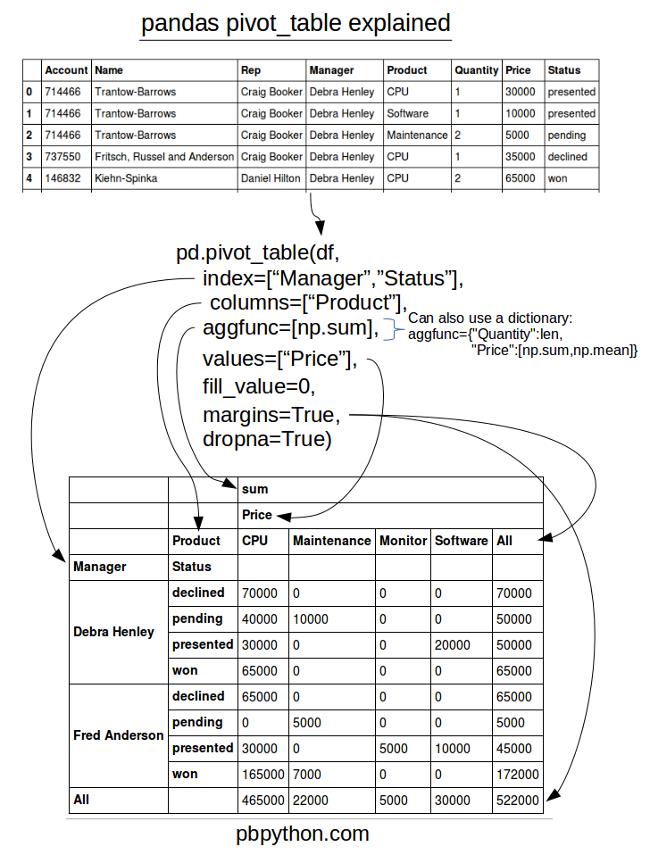
1. 变量“columns（列）”是可选的，***它提供一种额外的方法来分割你所关心的实际值。然而，聚合函数aggfunc最后是被应用到了变量“values”中你所列举的项目上.它会把Value值再按照columns.value分类。***
2. 使用其他聚合函数，将其传给aggfunc即可

tips.pivot\_table('tip\_pct', index=['sex','smoker'],columns='day',aggfunc=len, margins=True)

1. 如果存在空的组合(NA)，可以通过设置fill\_value

tips.pivot\_table('size', index=['time','sex','smoker'],columns='day',aggfunc=sum, fill\_value=0, margins=True)

1. 一张图表：



1. pivot\_table参数：values, index, columns,aggfunc,fill\_value,margins

### 交叉表P290

是一种用于计算分组频率的特殊透视表。

pd.crosstab(df.Gender, df.Handedness, margins=True) #crosstab用于统计Gender列(行索引)和Handedness列（列索引）

pd.crosstab([df.time, df.day], tips.smoker, margins=True) #crosstab前两个参数可以是数组、Series或数组列表

### 示例：2012选举委员会数据库

fec = pd.read\_csv('ch09/P00000001-All.csv')

tmp = fec.ix[12345] #df.ix[n], get the data of line n.

unique\_cands = fec.cand\_nm.unique() #unique() gets a unique list of cand\_nm, like list(set(cand\_nm))

parties = {'Bachmann, Michelle':'Republican',

'Cain, Herman':'Republican',

'Gingrich, Newt':'Republican',

'Huntsman, Jon':'Republican',

'Johnson, Gary Earl':'Republican',

'McCotter, Thaddeus G':'Republican',

'Obama, Barack':'Democrat',

'Paul, Ron':'Republican',

'Pawlenty, Timothy':'Republican',

'Perry, Rick':'Republican',

"Roemer, Charles E. 'Buddy' III":'Republican',

'Romney, Mitt':'Republican',

'Santorum, Rick':'Republican'}

fec.cand\_nm[123456:123461]

fec.cand\_nm[123456:123461].map(parties) #map(dict)根据字典索引映射，返回字典值

fec['party'] = fec.cand\_nm.map(parties) #add a new columns

fec['party'].value\_counts() #value\_counts返回数量, caculate num of every value in column 'party'

(fec.contb\_receipt\_amt>0).value\_counts() #统计cntb\_receipt\_amt大于0的数量, format: (dataframe.column\_name>value).value\_counts()

fec=fec[fec.contb\_receipt\_amt>0] #把cntb\_receipt\_amt>0的数据赋给fec, remove data lines with contb\_receipt\_amt value is less than 0

fec\_mrbo = fec[fec.cand\_nm.isin(['Obama, Barack','Romney, Mitt'])] #df.cand\_nm.isin([list])把某一包含在[list]里的列赋值到一个新df数据里

fec.contbr\_occupation.value\_counts()[:10] #统计每种职业的数量, value\_counts() is the method of Series, not a Dataframe

#####对各党派总出资额最高的职业

#category similar information to the exact same information in column contbr\_occupation

occ\_mapping = {'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',

'INFORMATION REQUESTED' : 'NOT PROVIDED',

'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',

'C.E.O.' : 'CEO'

}

#if the value of contbr\_occupation not exsited in occ\_mapping.key(), just return its self (no change); else retun occ\_mapping.values()

#dic.get(key, defaut value)

f = lambda x : occ\_mapping.get(x, x)

fec.contbr\_occupation = fec.contbr\_occupation.map(f) #same as before. substitue the value if key is same in occ\_mapping

emp\_mapping = {

'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',

'INFORMATION REQUESTED' : 'NOT PROVIDED',

'SELF' : 'SELF-EMPLOYED',

'SELF EMPLOYED' : 'SELF-EMPLOYED'

}

f = lambda x :emp\_mapping.get(x,x)

fec.contbr\_employer = fec.contbr\_employer.map(f)

#use pivot\_table to remove lines where sum<2000000

by\_occupation = fec.pivot\_table('contb\_receipt\_amt', index = 'contbr\_occupation', columns = 'party', aggfunc = 'sum')

over\_2mm = by\_occupation[by\_occupation.sum(1)>2000000]

over\_2mm

over\_2mm.plot(kind = 'barh') #plot a bar type figure

#####对各党派总出资额最高的职业

#category similar information to the exact same information in column contbr\_occupation

occ\_mapping = {'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',

'INFORMATION REQUESTED' : 'NOT PROVIDED',

'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',

'C.E.O.' : 'CEO'

}

#if the value of contbr\_occupation not exsited in occ\_mapping.key(), just return its self (no change); else retun occ\_mapping.values()

#dic.get(key, defaut value)

f = lambda x : occ\_mapping.get(x, x)

fec.contbr\_occupation = fec.contbr\_occupation.map(f) #same as before. substitue the value if key is same in occ\_mapping

emp\_mapping = {

'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',

'INFORMATION REQUESTED' : 'NOT PROVIDED',

'SELF' : 'SELF-EMPLOYED',

'SELF EMPLOYED' : 'SELF-EMPLOYED'

}

f = lambda x :emp\_mapping.get(x,x)

fec.contbr\_employer = fec.contbr\_employer.map(f)

#use pivot\_table to remove lines where sum<2000000

by\_occupation = fec.pivot\_table('contb\_receipt\_amt', index = 'contbr\_occupation', columns = 'party', aggfunc = 'sum')

over\_2mm = by\_occupation[by\_occupation.sum(1)>2000000]

over\_2mm

over\_2mm.plot(kind = 'barh') #plot a bar type figure

#####对obama和Romney总出资额最高的职业和企业

#####先对候选人分组，再用求取最大值方法

def get\_top\_amounts(group, key, n=5):

totals = group.groupby(key)['contb\_receipt\_amt'].sum()

#order totals in reverse

#return totals.order(ascending = False)[n:]

return totals.sort\_values(ascending = False)[n:]

grouped = fec\_mrbo.groupby('cand\_nm') #grouped by 'cand\_nm', it is an iteration,

grouped.apply(get\_top\_amounts, 'contbr\_occupation', 7) #group 'grouped' by '\_occupation'.

grouped.apply(get\_top\_amounts, 'contbr\_employer', n=10)

### 对出资额分组P297

利用cut函数根据出资额的大小将数据离散化到多个面元中：

bins = np.array([0, 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000]) #generate labels

labels = pd.cut(fec\_mrbo.contb\_receipt\_amt, bins) #use cut to group contb\_receipt\_amt

labels

#根据候选人姓名以及面元标签对数据进行分组

grouped = fec\_mrbo.groupby(['cand\_nm',labels])

grouped.size().unstack(0) #unstack(0): put values of column 0 onto new column A and B...

bucket\_sums = grouped.contb\_receipt\_amt.sum().unstack(0) #sum of contb\_receipt\_amt

normed\_sums = bucket\_sums.div(bucket\_sums.sum(axis=1), axis = 0) #bucket\_sums.sum(axis=1), sum the row by row; div(\*\*, axis=0): div col by col

normed\_sums[:-2].plot(kind = 'barh', stacked = True)

### 根据州统计赞助信息

#根据候选人和州对数据进行聚合

grouped = fec\_mrbo.groupby(['cand\_nm', 'contbr\_st'])

totals = grouped.contb\_receipt\_amt.sum().unstack(0).fillna(0)

totals = totals[totals.sum(1) > 100000]

totals[:10]

#对totals的每行除以总赞助额比例，得到百分比

percent = totals.div(totals.sum(1), axis=0) #totals.sum(1)对行求和，sum(0)对列求和，axis=0对行做除法

percent2 = totals.div(totals.sum(0), axis=1) #totals.sum(1)对列求和，sum(0)对列求和, axis=1对列做除法

#matplotlib及basemap工具包可以画地图

from mpl\_toolkits.basemap import Basemap,cm

TODO

### Chapter 10 时间序列

pandas提供时间切片/切块、聚合、对定期/不定期的时间序列重采样。

### 日期和时间数据类型及工具

包含datetime,time以及calendar模块。datetime.datetime是用的最多的数据类型

from datetime import datetime

now = datetime.now()

now.year, now.month, now.day

delta = datetime(2011,1,7)- datetime(2008,6,24,8,15) #计算时间差

delta.days #换算天

delta.seconds

from datetime import timedelta

start = datetime(2011, 1, 7)

start + timedelta(12)

start - 2 \* timedelta(12)

datetime的数据类型: **date, time,datetime,timedelta**

### 字符串和datetime的相互转换:str or strftime

pandas通常是用于处理成组日期，不管是DataFrame的轴索引或列，to\_dateime方法可以解析多种不同的日期表示形式。对标准日期的解析非常快

#处理缺失(None, 空字符串)

idx = pd.to\_datetime(datestrs + [None])

idx[2]

pd.isnull(idx)

### 时间序列基础

from datetime import datetime

dates = [datetime(2011,1,2), datetime(2011,1,5), datetime(2011,1,7),datetime(2011,1,8),datetime(2011,1,10),datetime(2011,1,12)]

ts = Series(np.random.randn(6), index = dates)

type(ts)

ts.index

#不同索引的时间序列之间的算术运算会自动按日期对齐

ts+ts[::2]

#pandas用Numpy的datetime64数据类型以n秒形式存储时间戳

ts.index.dtype

stamp = ts.index[0]

stamp

### 索引、选取u、子集构造

stamp = ts.index[0]

stamp

#TimeSeries is a subclass of Series

#数据索引上与Series一致

stamp = ts.index[2]

data = ts[stamp]

#其他方法得到数据

d1 = ts['1/10/2011']

d2 = ts['20110110']

#对于较长序列只需传入年和月就可以选取切片

longer\_ts = Series(np.random.randn(1000), index = pd.date\_range('1/1/2000', periods = 1000)) #pd.date\_range('开始时间', period=长度)

longer\_ts['2001'] #切片2001年数据

longer\_ts['2001-05'] #切片2001年-5月数据

ts[datetime(2011,1,7):] #通过日期进行切片的方式只对规则Series有效

ts['1/6/2011':'1/11/2011'] #通过时间段切片，可以传入字符串日期、datetime、Timestamp

ts.truncate(after = '1/9/2011') #.truncate一个实例方法截取时间段

#以上的操作对DataFrame也有效

dates = pd.date\_range('1/1/2000', periods = 100, freq='W-WED')

long\_df = DataFrame(np.random.randn(100,4), index = dates, columns=['Colorado', 'Texas', 'New York', 'Ohio'])

long\_df.ix['5-2001']

### 带有重复索引的时间序列

dates = pd.DatetimeIndex(['1/1/2000', '1/2/2000', '1/2/2000', '1/2/2000', '1/3/2000'])

dup\_ts = Series(np.arange(5), index=dates)

dup\_ts

dup\_ts.index.is\_unique #is\_unique属性表明时间索引是否唯一

#对非唯一时间戳的数据进行聚合，一个办法是使用groupby，并传入level=0(索引的唯一一层)

grouped = dup\_ts.groupby(level = 0)

grouped.mean()

grouped.count()

### 日期的范围、频率以及移动

pandas有一整套标准时间序列频率以及用于重采样、频率推断、生成固定频率日期范围的工具。例如把时间序列转换为一个具有固定频率（每日）的时间序列，只需调用resample.

ts

ts.resample('D',how=mean)

### 生成日期范围

index = pd.date\_range('4/1/2012', '6/1/2012') #pd.date\_range可以生成时间序列

index

pd.date\_range(start = '4/1/2012', periods = 20) #pd.date\_range可以生成时间序列

pd.date\_range(end= '6/1/2012', periods = 20) #pd.date\_range可以生成时间序列

pd.date\_range('1/1/2000','12/1/2000',freq='BM') #freq='BM', bottom of month

pd.date\_range('5/2 2012 12:56:31',periods=5)

pd.date\_range('5/2/2012 12:56:31', periods=5,normalize=True) #normalize规范化到午夜时间戳

### 频率和日期偏移量

from pandas.tseries.offsets import Hour, Minute

hour = Hour()

four\_hours = Hour(4)

pd.date\_range('1/1/2000','1/3/2000 23:59',freq='4h') #need to do "from pandas.tseries.offsets import Hour, Minute"???

Hour(2)+Minute(30) #caculate hour+minute

pd.date\_range('1/1/2000','1/3/2000 23:59',freq='4h30min') #freq='4h30min' 会被解析成4.5hours

rng = pd.date\_range('1/1/2012','9/1/2012',freq='WOM-3FRI') #每个月第三个星期5

#时间序列的基础频率：D,B,H,T/min,L/ms,U,M,BM,MS,BMS,W-MON/TUE...,WOM-1MON...,Q-JAN,BQ-JAN...,QS-JAN...,BOS-JAN...,A-JAN,BA-JAN,AS-JAN,BAS-JAN..

### 移动（超前和滞后）数据

ts = Series(np.random.randn(4), index = pd.date\_range('1/1/2000', periods=4, freq='M'))

ts.shift(2) #shift(num)数据下移num行

ts.shift(-2) #shift(num)数据上移num行

#shift计算时间序列中百分比变化

ts/ts.shift(1)-1 #移位不修改索引，所以部分数据被丢弃

ts.shift(2, freq='M') #freq='M'移动索引，不丢失数据。

ts.shift(3, freq='D')

ts.shift(1, freq='3D')

### 通过偏移量对日期进行移位

from pandas.tseries.offsets import Day, MonthEnd

now = datetime(2011, 11,17)

now+3 \* Day()

now + MonthEnd() #当月月底

now + MonthEnd(2) #两个月后的月底

#通过锚点的偏移量的rollforward和rollback方法，可显式地将日期向前或后滚动

offset = MonthEnd()

offset.rollforward(now)

#结合groupby使用日期偏移量滚动: 生成(groupby)每个月的平均值

ts = Series(np.random.randn(20), index=pd.date\_range('1/15/2000',periods=20,freq='4d'))

In [123]: ts

Out[123]:

2000-01-15 0.536603

2000-01-19 2.474292

2000-01-23 0.349214

2000-01-27 0.295661

2000-01-31 -0.383701

2000-02-04 0.230991

2000-02-08 1.725376

2000-02-12 1.555938

2000-02-16 -0.306516

2000-02-20 0.320364

2000-02-24 1.968424

2000-02-28 0.119211

2000-03-03 0.142022

2000-03-07 0.558954

2000-03-11 -0.727436

2000-03-15 -1.037621

2000-03-19 0.269401

2000-03-23 0.008612

2000-03-27 -0.952699

2000-03-31 1.560577

Freq: 4D, dtype: float64

ts.groupby(offset.rollforward).mean()

ts.resample('M',how='mean') #使用resample对月进行统计会更简单

### 时区处理

#pandas包装了pytz(第三方库，区分时区)的功能

import pytz

pytz.common\_timezones[-5:]

tz = pytz.timezone('US/Eastern')

### 本地化和转换

#时间序列被本地化到某个特定时区，就可以用tz\_convert将其转换到别的时区

ts\_utc.tz\_convert('US/Eastern') #tz\_convert转换到时区

ts\_eastern= ts.tz\_localize('US/Eastern')

ts\_eastern.tz\_convert('Europe/Berlin')

ts.index.tz\_localize('Asia/Shanghai')

#警告，对单纯时间戳的本地化操作还会检查夏令时转变期附近容易混淆或不存在的时间

### 操作时区意识型Timestamp对象

与时间序列和日期差不多，Timestamp对象

from pandas.tseries.offsets import Hour

stamp = pd.Timestamp('2012-03-12 01:30', tz = 'US/Eastern')

stamp+Hour()

stamp = pd.Timestamp('2012-11-04 00:30', tz='US/Eastern')

stamp

stamp + 2 \* Hour()

### 不同时区之间的运算

#如果两个时间序列的时区不同，合并的结果时UTC。

rng = pd.date\_range('3/7/2012 9:30', periods=10, freq='B') #freq='B', 周一至周五

ts = Series(np.random.randn(len(rng)),index=rng)

ts1 = ts[:7].tz\_localize('Europe/London')

ts2 = ts1[2:].tz\_convert('Europe/Moscow')

result = ts1 + ts2 #result把ts2的时区先转换为UTC，然后再相加

result.index

### 时期及算术运算P322

rng = pd.period\_range('1/1/2000','6/30/2000',freq='M')

rng #rng为PeriodIndex类保存了一组Period，可以在任何pandas数据结构中被用做轴索引

Series(np.random.randn(6), index = rng)

#PeriodIndex类构造函数允许直接使用一组字符串

values = ['2001Q3', '2002Q2', '2003Q1']

index = pd.PeriodIndex(values, freq='Q-DEC')

### 时期的频率转换

#Period和PeriodIndex对象都可以通过其asfreq方法被转换成别的频率: 划分时间段

#如果我们有一个年度时期，可以将其转换为当年年初或年末的一个月度时期

p = pd.Period('2007', freq='A-DEC')

p.asfreq('M', how='start')

#Out[34]: Period('2007-01', 'M')

p.asfreq('M', how='end')

#Out[36]: Period('2007-12', 'M')

p = pd.Period('2007-08', 'M')

p.asfreq('A-JUN')

#Out[37]: Period('2008', 'A-JUN'), 08月在按六月划分的周期

rng = pd.period\_range('2006','2009', freq='A-DEC')

ts = Series(np.random.randn(len(rng)), index=rng)

ts.asfreq('M', how='start')

ts.asfreq('B', how='end')

### 按季度计算的时期频率P324

#季度性数据涉及'财年末'概念，通常是12个月的最后某一个日历日或工作日。2012Q4根据财年末的不同会有不同的含义

#pandas支持12中可能的季度性数据,即Q-JAN到Q-DEC

p = pd.Period('2012Q4', freq='Q-JAN') #Q-JAN说明财年计算日为1月，因此2012Q4是从11年11月到12年1月

p.asfreq('D', 'start')

p.asfreq('D','end')

p4pm = (p.asfreq('B', 'e')-1).asfreq('T','s')+16 \* 60 #获取该季度倒数('e')第二个(-1)工作日('B')下午4点(16 \* 60)的时间戳

p4pm.to\_timestamp() #Period转换为时间戳

rng = pd.period\_range('2011Q3','2012Q4',freq='Q-JAN') #period\_range生成季度型范围

ts = Series(np.arange(len(rng)), index=rng)

new\_rng = (rng.asfreq('B','e') -1).asfreq('T','s') + 16 \* 60 #获取该季度倒数第二个工作日下午4点时间戳

ts.index = new\_rng.to\_timestamp() #改变index

### 将timestamp转换为Period(及反向过程)

#to\_period可以将时间戳索引转换为时期索引

rng = pd.date\_range('1/1/2000', periods=3, freq='M')

ts = Series(randn(3),index=rng)

pts = ts.to\_period()

ts

pts

#由于时期是非重叠时间区间，因此一个时间戳智能属于一个时期。新Period的频率默认是从时间戳推断来的

rng = pd.date\_range('1/29/2000', periods=6, freq='D')

ts2 = Series(randn(6), index=rng)

ts2.to\_period('M')

#使用to\_timestamp()可以转换为时间戳

pts = ts.to\_period()

pts

pts.to\_timestamp(how='end') #how='end', 转换为最后一天

### 通过过数组创建PeriodIndex

#年度和季度分别放在不同的列

data = pd.read\_csv('ch08/macrodata.csv')

data.year

data.quarter

#把year和quarter以及一个频率传入PeriodIndex，就可以合并成DF的一个索引

index = pd.PeriodIndex(year=data.year, quarter=data.quarter, freq='Q-DEC')

index

data.index = index #添加周期index

data.infl

### 重采样及频率转换

#重采样(resampling)指的是将时间序列从一个频率转换到另一个频率的处理过程。

#将高频数据聚合到低频率称为"降采样", downsampling, 将低频数据聚合到高频率称为"升采样", upsampling

#pandas对象都带有一个resample方法，是各种频率采样的主力函数

rng = pd.date\_range('1/1/2000', periods=100, freq='D')

ts = Series(randn(len(rng)), index=rng)

ts.resample('M', how='mean', kind='period')

#resample是一个高效灵活的处理时间序列的函数，其参数包括：freq, how='mean/ohlc,np.max,min', axis=0, fill\_method=None, closed='right', label='right',loffset=None, limit=None, kind=None, convention=None

### 降采样

使用resample进行降采样时，需要考虑两样东西：

1. 各区间哪边是闭合的
   * + 1. 如何标记各个聚合面元，用区间的开头还是末尾

#降采样

#1分钟数据

rng = pd.date\_range ('1/1/2000', periods=12, freq='T')

ts = Series(np.arange(12), index=rng)

#通过求和的方式将数据聚合到"5分钟"，close左边界闭合,lable左边界作为标记

ts.resample('5min',how='sum', closed='left', label='left')

#loffset对索引移位，从索引右边界减去1s，或使用shift方法

ts.resample('5min', how='sum', loffset='-1s')

### OHLC重采样

ts.resample('5min',how='ohlc') #how='ohlc'即可计算开盘，

### 通过groupby进行重采样

#.groupby()根据月份或星期进行分组，只需传入一个能够访问时间序列的索引上的这些字段的函数即可

rng = pd.date\_range('1/1/2000', periods=100, freq='D')

ts =Series(np.arange(100), index=rng)

ts.groupby(lambda x: x.month).mean()

ts.groupby(lambda x: x.weekday).mean()

### 升采样和插值

frame= DataFrame(np.random.randn(2,4), index=pd.date\_range('1/1/2000', periods=2, freq='W-WED'),

columns=['colorado', 'texas', 'new york', 'ohio'])

df\_daily=frame.resample('D').ffill() #'D', 按day进行升采样, .ffill()方法插值(forward fill)

df\_daily2=frame.resample('D').ffill(limit=2) #只前向插值两个数据

frame.resample('W-THU').ffill() #修改时间索引到其他周期

### 通过时期进行重采样

frame = DataFrame(np.random.randn(24,4), index=pd.period\_range('1-2000', '12-2001', freq='M'), columns=['colorado', 'texas', 'new york', 'ohio'])

frame[:5]

annual\_frame = frame.resample('A-DEC').mean() #按年求均值

#升采样，生成季度型数据

annual\_frame.resample('Q-DEC').ffill()

annual\_frame.resample('Q-DEC', convention='start').ffill()

#由于时期指的是时间区间，所以升采样和降采样的规则比较严格

#降采样中，目标频率必须是源频率的子时期(subperiod)

#升采样中，目标频率必须是源频率的超时期(superperiod)

#如果不满足，就会引发异常， 由Q-MAR定义的时间区间智能升采样为A-MAR, A-JUN, A-SEP, A-DEC

annual\_frame.resample('Q-MAR').ffill()

### 时间序列绘图

#pandas时间序列的绘图功能在日期格式化方面比matplotlib原生的要好

close\_px\_all = pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

close\_px = close\_px\_all[['AAPL', 'MSFT', 'XOM']]

close\_px = close\_px.resample('B').ffill()

close\_px

#对某一列绘图，使用.plot()方法

close\_px['AAPL'].plot()

#对DataFrame里2009年的数据，.loc代表截取行(axis=0),['2009']表示截取时间段儿

close\_px.loc['2009'].plot()

#苹果股票在2011年1月到3月的股价

close\_px['AAPL'].loc['01-2011':'03-2011'].plot()

#转换成季度数据

appl\_q = close\_px['AAPL'].resample('Q-DEC').ffill()

appl\_q.loc['2009':].plot()

### 移动窗口函数

包括指数加权移动平均（会自动排除缺失值）

#rolling\_mean接受TimeSeries或DataFrame以及一个window

close\_px.AAPL.plot()

pd.rolling\_mean(close\_px.AAPL, 250).plot() #pd.rolling\_mean被Series.rolling(center=False,window=250).mean()代替

#apple股票的250日均线

#对DataFrame调用rolling方法会对每一列生成一组数据,.plot()对每一列划线

close\_px.rolling(center=False, window=250).mean().plot()

close\_px.AAPL.rolling(center=False, window=250).mean().plot() #not recommended

#苹果公司250日每日回报标准差

app\_std250 = close\_px.AAPL.rolling(center=False, window=250,min\_periods=10).std()

app\_std250.plot()

#通过rolling\_mean定义扩展平均

expanding\_mean = lambda x : x.rolling(len(x), min\_periods=1).mean()

expanding\_mean\_2 = lambda x: rolling\_mean(x, len(x), min\_periods=1)

#plot(logy=True), 画log坐标

app\_std250\_log = close\_px.AAPL.rolling(center=False, window=250,min\_periods=10).mean().plot(logy=True)

#log坐标的60日均线

pd.rolling\_mean(close\_px, 60).plot(logy=True)

#rolling\_mean()这些方法将来会被取消，使用pd.rolling.mean()代替

#移动窗口和指数加权函数:

#rolling.count(), 返回各窗口非NA观测值数量

#sum, mean, median, var, std,移动窗口和,平均数，中位数，方差(分母为n-1)，标准差(分母为n-1)

#skew, kurt, 偏度（三阶距）和峰度（四阶据）

#min, max, 移动窗口的最小值，最大值

#quantile,移动窗口指定百分位数/样本分位数位置的值？？？？

#corr, cov, 相关系数和协方差

#apply, 对窗口应用普通数组函数

#指数加权函数

#ewma, ewmvar, ewmstd, ewmcorr，指数加权平均，移动方差，标准差，相关系数和协方差

### 指数加权函数

#定义两幅图片

fig, axes = plt.subplots(nrows=2, ncols=1, sharex=True, sharey=True, figsize=(12,7))

aapl\_px = close\_px.AAPL['2005': '2009']

ma60 = aapl\_px.rolling(window=60, min\_periods=50).mean()

#ewma60 = pd.ewma(aapl\_px, span=60)

ewma60 = aapl\_px.ewm(adjust=True, min\_periods=0, span=120, ignore\_na=False).mean()

aapl\_px.plot(style='k-', ax=axes[0])

ma60.plot(style='k--', ax=axes[0])

aapl\_px.plot(style='k-', ax=axes[1])

ewma60.plot(style='k--', ax=axes[1])

axes[0].set\_title('Simple MA')

axes[1].set\_title('Exponentiallyweighted MA')

### 二元移动窗口函数

#在两个时间序列上执行统计运算（相关系数和协方差），比如计算某只股票对某个指数的关系系数

spx\_px = close\_px\_all['SPX']

spx\_rets = spx\_px / spx\_px.shift(1)-1

returns = close\_px.pct\_change()

#corr = pd.rolling\_corr(returns.AAPL, spx\_rets, 125, min\_periods=100)

#AAPL 6个月回报与标准普尔500指数的相关系数

corr = returns.AAPL.rolling(window=125,min\_periods=100).corr(spx\_rets) #.corr(other Series)

corr.plot()

#所有股票与标普500的相关系数

corr = returns.rolling(window=125,min\_periods=100).corr(spx\_rets) #.corr(other Series)

corr.plot()

### 用户定义的移动窗口函数

#rolling.apply()可以在移动窗口上应用自定义数组函数，唯一要求是：该函数能从数组的多个片段中产生单个值（约简）

from scipy.stats import percentileofscore

score\_at\_2percent = lambda x: percentileofscore(x, 0.02)

#result = pd.rolling\_apply(returns.AAPL, 250, score\_at\_2percent)

result = returns.AAPL.rolling(window=250, center=False).apply(score\_at\_2percent)

result.plot()

### 性能和内存使用方面的注意事项

rng = pd.date\_range('1/1/2000', periods=10000000, freq='10ms')

ts = Series(np.random.randn(len(rng)), index=rng)

#ts.resample('15min', how='ohlc')

#resample(, how='ohlc') is deprecated. use "ts.resample('15min').ohlc()"

ts.resample('15min').ohlc()

#%timeit ts.resample('15min', how='ohlc')

%timeit ts.resample('15min').ohlc()

#运行时间跟聚合结果的相对大小有一定关系，越高频的聚合耗费时间越多

rng = pd.date\_range('1/1/2000', periods=10000000, freq='1s')

ts = Series(np.random.randn(len(rng)), index=rng)

%timeit ts.resample('15s').ohlc()

#383 ms ± 20.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

### 第11章 金融和经济数据应用

### 时间序列及截面对齐

rng = pd.date\_range('09/06/2011', periods=6,freq = 'D')

close\_px\_all = pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

prices = close\_px\_all.loc['09/06/2011': '09/12/2011']

rng2 = pd.date\_range('09/06/2011', periods=4,freq = 'D')

volume = DataFrame((np.random.randn(4,3)\*1000).round(0)\*\*2, index=rng2, columns=['AAPL','MSFT','SPX'])

prices\*volume #pandas会自动按行索引和列名进行矩阵运算

vwap = (prices \* volume).sum() / volume.sum()

vwap.dropna() #去掉NaN数据

#手动对齐，使用align()方法

prices.align(volume, join='inner')

s1 = Series(range(3), index=['a','b','c'])

s2 = Series(range(4), index=['d','b','c','e'])

s3 = Series(range(3), index=['f','a','c'])

DataFrame({'one':s1, 'two':s2, 'three':s3})

#编辑索引

DataFrame({'one':s1, 'two':s2, 'three':s3}, index=list('face'))

### 频率不同的时间序列的运算

#频率转换和重对齐的两大主要工具是resample和reindex方法

#resample用于将数据转到固定频率

#reindex用于使数据符合一个新索引，他们都支持插值(前向填充)逻辑

ts1 = Series(np.random.randn(3), index=pd.date\_range('2012-6-13', periods=3, freq='W-WED'))

ts1.resample('B').mean() #补充周期为工作日，注意mean()

ts1.resample('B').ffill() #对数据前向填充

#reindex使用

dates = pd.DatetimeIndex(['2012-6-12', '2012-6-17', '2012-6-18', '2012-6-21', '2012-6-22', '2012-6-29'])

ts2 = Series(np.random.rand(6), index=dates)

#ts1和ts2的序列相差很多。一个办法使把所有序列升采样，第二个办法使用reindex

ts1.reindex(ts2.index, method='ffill') #reindex(ts2.index, method='ffill')

ts2+ts1.reindex(ts2.index, method='ffill')

### 使用Period

gdp = Series([1.78, 1.94, 2.08, 2.01, 2.15, 2.31, 2.46], index=pd.period\_range('1984Q2', periods=7, freq='Q-SEP'))

infl = Series([0.025, 0.045, 0.037, 0.04], index=pd.period\_range('1982', periods=4, freq='A-DEC'))

infl\_q = infl.asfreq('Q-SEP', how='end') #把年数据换成季度数据

infl\_q.reindex(gdp.index, method='ffill') #重新并入序列

### 时间和“最当前”数据选取

#假设有一个很长的盘中市场数据时间序列，现在希望抽取每天特定时间的价格数据。如果数据不规整，如何处理？

#生成一个交易日内日期范围和时间序列

rng = pd.date\_range('2012-06-01 09:30', '2012-06-01 15:59', freq='T')

#生成5天时间点(9:30~15:59之间的值)

rng =rng.append([rng + pd.offsets.BDay(i) for i in range(1,4)])

ts = Series(np.arange(len(rng), dtype=float), index=rng)

ts

from datetime import time

#利用datetime.time对象进行索引即可抽取这些时间点的值

ts[time(10,0)] #抽取位于10分钟点的数据

#ts.at\_time(time(10,0))

#选取两个Time对象之间的值

ts.between\_time(time(10,0), time(10,1))

#取得上午10点之前最后出现的值

#将时间序列大部分内容设置为NA

indexer = np.sort(np.random.permutation(len(ts))[700:])

irr\_ts = ts.copy()

irr\_ts[indexer] = np.nan

irr\_ts['2012-06-01 09:50':'2012-06-01 10:00']

#将一组Timestamp传入asof方法，可以得到这些时间点处（或之前最近）的有效值（非NA）

selection = pd.date\_range('2012-06-01 10:00', periods=4, freq='B')

irr\_ts.asof(selection)

### 拼接多个数据源

#在一个特定的时间点从一个数据源切换到另一个数据源

data1 = DataFrame(np.ones((6,3),dtype=float), columns=['a','b','c'], index=pd.date\_range('6/12/2012',periods=6))

data2 = DataFrame(np.ones((6,3),dtype=float)\*2, columns=['a','b','c'], index=pd.date\_range('6/13/2012',periods=6))

spliced = pd.concat([data1.loc[:'2012-06-14'],data2.loc['2012-06-15':]])

spliced

#用另一个时间序列对当前时间序列的缺失值“打补丁”

data2 = DataFrame(np.ones((6,4),dtype=float)\*2, columns=['a','b','c','d'], index=pd.date\_range('6/13/2012',periods=6))

spliced = pd.concat([data1.loc[:'2012-06-14'],data2.loc['2012-06-15':]])

spliced

#combine\_first可以引入合并点之前的数据，这样也就扩展了'd'项的历史

spliced\_filled = spliced.combine\_first(data2)

#将数据中的符号(国家、资产代码等)替换为实际数据

#DF.update方法可以实现就地更新，如果只想填充空洞，必须传入overwrite=False才行

spliced.update(data2, overwrite=False)

cp\_spliced = spliced.copy()

cp\_spliced[['a','c']] = data1[['a','c']] #替换'a','c'列数据

cp\_spliced

### 收益指数和累计收益

#收益指某资产价格的百分比变化

#以苹果2011-2012股价为例

#Not Working...

#import pandas.io.data as web

#from pandas\_datareader import wb as web

#price = web.get\_data\_yahoo('AAPL', '2011-01-01')['Adj close']

#price[-5:]

close\_px\_all = pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

close\_px = close\_px\_all[['AAPL', 'MSFT', 'XOM']]

close\_px = close\_px.resample('B').ffill()

price = close\_px.copy()

price['2011-10-03']/price['2011-3-01']-1

#计算收益百分比(currt-last)/last

returns = price.pct\_change()

ret\_index = (1+returns).cumprod() #计算累计收益率,第一个值是NaN

ret\_index[0] = 1

#计算指定时期的累计收益

#m\_returns = ret\_index.resample('BM', how='last').pct\_change()

m\_returns = ret\_index.resample('BM').last().pct\_change()

m\_returns['2011']

#或者

m\_rets = (1+returns).resample('M', kind='period').prod()-1

m\_rets['2011']

#如果知道了股息的派发日和支付率，可以将他们计入到每日总收益中，如下所示：

returns[dividend\_dates]+=dividend\_pcts

### 分组变换和分析

#首先，随机生成1000个股票代码

import random; random.seed(0)

import string

N=1000

def rands(n):

choices = string.ascii\_uppercase

return ''.join([random.choice(choices) for \_ in range(n)])

tickers = np.array([rands(5) for \_ in range(N)]) #\_ 代表执行range(N)次

#创建一个含有3列的DataFrame，承载假想数据。只选择部分股票组成该投资组合

M=500

df = DataFrame({'Momentum': np.random.randn(M)/200+0.03,'value':np.random.randn(M)/200+0.08,

'ShortInterest': np.random.randn(M)/200 - 0.02}, index = tickers[:M])

#为这些股票创建行业分类。只选了两个行业，并将映射关系保存在Series中

ind\_names = np.array(['FINANCIAL', 'TECH'])

sampler = np.random.randint(0,len(ind\_names),N)

industries = Series(ind\_names[sampler],index=tickers, name='industry')

by\_industry = df.groupby(industries) #对工业进行分组聚合和变换

by\_industry.describe()

#对这些按行业分组的投资组合进行各种变换，我们可以编写自定义变换函数。

#例如行内标准化处理，它广泛用于股票投资组合的构建过程

#行业内标准化处理

def zscore(group):

return (group - group.mean())/group.std()

def\_stand = by\_industry.apply(zscore)

def\_stand.groupby(industries).agg(['mean','std'])

#使用内置变换函数(rank)的用法会更简洁

#行业内降序排名

ind\_rank = by\_industry.rank(ascending=False)

ind\_rank.groupby(industries).agg(['min','max'])

#在股票投资组合的定量分析中，“排名和标准化”是一种很常见的变换运算组合。通过将rank和zscore链接在一起，即可以完成整个变换过程

#行业内排名和标准化

by\_industry.apply(lambda x : zscore(x.rank()))

### 分组因子暴露

#因子分析(factor analysis)是投资组合定量管理的一种技术。投资组合的持有量和性能(收益与损失)可以被分解为一个或多个表示投资组合权重的因子(风险因子就是其中之一)

#例如某只股票的价格与标准普尔500指数的协动性被称作贝塔(beta)风险系数

#下面以一个人为构成的投资组合为例，进行讲解。

#它由三个随机生成的因子(因子载荷)和一些权重构成

from numpy.random import rand

fac1, fac2,fac3=np.random.rand(3, 1000)

import random; random.seed(0)

import string

N=1000

def rands(n):

choices = string.ascii\_uppercase

return ''.join([random.choice(choices) for \_ in range(n)])

tickers = np.array([rands(5) for \_ in range(N)]) #\_ 代表执行range(N)次

ticker\_subset = tickers.take(np.random.permutation(N)[:1000])

#因子加权以及噪声

port = Series(0.7 \* fac1 - 1.2 \* fac2 + 0.3 \* fac3 + rand(1000), index=ticker\_subset)

factors = DataFrame({'f1':fac1, 'f2':fac2, 'f3':fac3}, index=ticker\_subset)

#各因子与投资组合之间的矢量相关性

factors.corrwith(port)

#计算因子暴露的标准方式是：最小二乘回归。

#使用pandas.ols（将factors作为解释变量），可计算整个投资组合的暴露：

pd.ols(y=port, x=factors).beta #not working

#通过groupby计算各行业的暴露量

def beta\_exposure(chunk, factors=None):

return pd.ols(y=chunk, x=factors).beta

#根据行业进行分组，应用beta\_exposure函数

by\_ind = port.groupby(industries)

exposures = by\_ind.apply(beta\_exposure, factors = factors)

exposures.unstack()

### 十分位和四分位分析

#股票投资组合的性能可以根据个股的市盈率被划入四分位

#例子：利用跟随策略或动量交易策略通过交易所交易基金买卖标准普尔500指数

#import pandas.io.data as web #out of date

#from pandas\_datareader import data, wb

close\_px\_all = pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

close\_px = close\_px\_all[['AAPL', 'MSFT', 'XOM']]

close\_px = close\_px.resample('B').ffill()

price = close\_px.copy()

px = price['AAPL']

returns = px.pct\_change()

def to\_index(rets):

index = (1+rets).cumprod()

#first\_loc = max(index.notnull().argmax() - 1, 0) #Set NaN to 1, not working

#index.values[first\_loc] = 1

return index

def trend\_signal(rets, lookback, lag):

signal = pd.rolling\_sum(rets, lookback, min\_periods=lookback - 5)

return signal.shift(lag)

#创建和测试一种根据每周五动量信号进行交易的策略

signal = trend\_signal(returns, 100, 3)

trade\_friday = signal.resample('W-FRI').resample('B').ffill()

#trade\_friday = signal.resample('W-FRI').resample('B').bfill()

trade\_rets = trade\_friday.shift(1) \* returns

#将该策略的收益率转换为收益指数

total\_rets = (1+trade\_rets).cumprod()

total\_rets.plot()

#to\_index(trade\_rets).plot()

#如果希望策略性能按不同大小的交易期波幅进行划分。年度标准差是计算波幅的一种简单方法。我们可以通过计算夏普比率来观察不同波动机制下的风险收益率

vol = pd.rolling\_std(returns, 250, min\_periods=200) \* np.sqrt(250)

def sharpe(rets, ann=250):

return rets.mean() / rets.std() \* np.sqrt(ann)

trade\_rets.groupby(pd.qcut(vol,4)).agg(sharpe)

Out[215]:

AAPL

(0.205, 0.32] 0.376464

(0.32, 0.378] 1.311624

(0.378, 0.408] 0.739462

(0.408, 0.581] 0.680181

Name: AAPL, dtype: float64

#结果说明：该策略再波幅较低时较好

### 更多示例应用

### 信号前沿分析：简化截面动量投资组合

#构建投资组合，计算特定回顾期内的动量，然后按降序排列并标准化

#变换函数

def calc\_mom(price, lookback, lag):

mom\_ret = price.shift(lag).pct\_change(lookback)

ranks = mom\_ret.rank(axis=1, ascending=False)

demeaned = ranks - ranks.mean(axis=1)

return demeaned / demeaned.std(axis=1)

#对策略事后检验的函数：通过指定回顾期和持有期，计算投资组合整体的夏普比率

compound = lambda x: (1+x).prod()-1

daily\_sr = lambda x: x.mean() / x.std()

def strat\_sr(prices, lb, hold):

#cal weight of investment profiles

freq = '%dB' %hold

port = calc\_mom(prices, lb, lag=1)

daily\_rets = prices.pct\_change()

#interest of investment

#port = port.shift(1).resample(freq, how='first')

port = port.shift(1).resample(freq).first()

#returns = daily\_rets.resample(freq, how=compound)

returns = daily\_rets.resample(freq).apply(compound)

port\_rets = (port \* returns).sum(axis=1)

return daily\_sr(port\_rets) \* np.sqrt(252 / hold)

strat\_sr(px, 70, 30)

# NOT WORKING, NEET TO CHECK LATER

#为了便于观察，将结果图形化。下面这个函数会利用matplotlib生成一张带有图例的热图(heatmap):

import matplotlib.pyplot as plt

def heatmap(df, cmap=plt.cm.gray\_r):

fig = plt.figure()

ax = fig.add\_subplot(111)

axim = ax.imshow(df.columns.name)

ax.set\_xlabel(df.columns.name)

ax.set\_xtickabels(list(df.columns))

ax.set\_ylabel(df.index.name)

ax.set\_yticks(np.arange(len(df.index)))

sx.set\_ytickabels(list(df.index))

plt.colorbar(axim)

heatmap(ddf)

### 期货合约转仓

#该何时以及如何快速卖出到期合约并买入下期合约

#首先用SPY交易所交易基金的部分价格作为标准普尔500指数的代理

import pandas.io.data as web

names = ['AAPL', 'GOOG', 'MSFT', 'DELL', 'GS', 'MS', 'BAC', 'C']

close\_px\_all = pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

close\_px = close\_px\_all[['AAPL']]

px = close\_px

#px = web.get\_data\_yahoo('SPY')['Adj Close'] \* 10

from datetime import datetime

expiry = {'ESU2': datetime(2010, 9, 21), 'ESZ2': datetime(2010,12,21)}

expiry = Series(expiry).sort\_index()

#利用随机漫步和一些噪声来模拟这两份合约的未来走势

np.random.seed(12347)

N=len(px)

walk = (np.random.randint(0, 200, size=N)-100) \* 0.25

perturb = (np.random.randint(0,20,size=N)-10) \* 0.25

walk = walk.cumsum()

# NOT WORKING

rng = pd.date\_range(px.index[0], periods=len(px)+N, freq='B')

near = np.concatenate([px.values, px.values[-1]+walk])

far = np.concatenate([px.values, px.values[-1]+walk+perturb])

prices = DataFrame({'ESU2':near, 'ESZ2':far}, index=rng)

prices.tail()

### 期货合约转仓(自，成功)

names = ['AAPL', 'GOOG', 'MSFT', 'DELL', 'GS', 'MS', 'BAC', 'C']

close\_px\_all = pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

close\_px = close\_px\_all[['AAPL', 'MSFT', 'XOM']]

#close\_px = close\_px\_all[['AAPL']]

px = close\_px

px['ESU2']= close\_px['AAPL']\*0.2+close\_px['MSFT']\*0.8

N = len(px)

walk = (np.random.randint(0, 200, size=N)-100) \* 0.25

perturb = (np.random.randint(0,20,size=N)-8) \* 0.25

walk = walk.cumsum()

walk\_tmp = walk+perturb

px['ESZ2']= px['ESU2']+perturb

del px['AAPL']

del px['MSFT']

del px['XOM']

prices = px

#将多个时间序列合并为单个连续序列的一个办法是构造一个加权矩阵。活动合约的权重为1，直到期满；

#下面的函数可以计算一个加权矩阵（权重根据到期前期数减少而线性衰减）

def get\_roll\_weights(start, expiry, items, roll\_periods=5):

#start: caculate the first day of weight matrix

#expiry: Series composed of "contract code"->'end date'

#item: name of contrat

dates = pd.date\_range(start, expiry[-1], freq='B')

weights = DataFrame(np.zeros((len(dates), len(items))), index=dates, columns=items)

prev\_date = weights.index[0]

#for i, (item, ex\_date) in enumerate(expiry.iteritems())):

for i, (item, ex\_date) in enumerate(expiry.items()): #.items() used for py3, .iteritems() used for py2

if i<len(expiry)-1:

weights.loc[prev\_date:ex\_date - pd.offsets.BDay(), item] = 1

roll\_rng = pd.date\_range(end=ex\_date -pd.offsets.BDay(), periods=roll\_periods+1,freq='B')

decay\_weights = np.linspace(0, 1, roll\_periods+1)

weights.loc[roll\_rng, item] = 1- decay\_weights

weights.loc[roll\_rng, expiry.index[i+1]] = decay\_weights

else:

weights.loc[prev\_date:, item] =1

prev\_date = ex\_date

return weights

weights = get\_roll\_weights('6/1/2010', expiry, prices.columns)

weights.loc['2010-09-12': '2010-09-21']

prices\_cut = prices.loc['2010-06-01':'2010-12-21']

rolled\_returns = (prices\_cut.pct\_change() \* weights).sum(1)

return\_index = (1+rolled\_returns).cumprod() #cumulated returns

return\_index.plot()

### 移动相关系数与线性回归

#移动窗口和指数加权时间序列函数就是用于处理动态模型的工具

#相关系数是观察两个资产时间序列变换协动性的一种手段。pandas的rolling\_corr函数可以根据两个收益序列计算出移动窗口相关系数。

#首先，从yahoo加载一些价格序列，并计算每日收益

#Not working

aapl = web.get\_data\_yahoo('AAPL', '2000-01-01')['Adj Close']

msft = web.get\_data\_yahoo('AAPL', '2000-01-01')['Adj Close']

names = ['AAPL', 'GOOG', 'MSFT', 'DELL', 'GS', 'MS', 'BAC', 'C']

close\_px\_all = pd.read\_csv('ch09/stock\_px.csv', parse\_dates=True, index\_col=0)

#close\_px = close\_px\_all[['AAPL', 'MSFT']]

aapl = close\_px\_all['AAPL']

msft = close\_px\_all['MSFT']

#每日收益率

aapl\_rets = aapl.pct\_change()

msft\_rets = msft.pct\_change()

#计算一年期移动相关系数并绘制图表

pd.rolling\_corr(aapl\_rets, msft\_rets, 250).plot()

#最小二乘回归提供了另一种对一个变量与一个或多个其他预测变量之间动态关系的建模方法

from statsmodels.api import add\_constant

model = pd.stats.api.ols(y=aapl\_rets, x={'MSFT': msft\_rets}, window=250) #not working, removed, change to statsmodels

model.beta

有关统计学和计量经济学的复杂模型更多信息请参考statsmodels

### CH10 Numpy高级应用

### ndarray对象的内部机理

#ndarray内部由以下内容组成

1. 指向数组的指针

2. 数据类型或dtype

3. 表示数组形状的元组shape

4. 跨度元组(stride)，其中的整数指的是为了前进到当前维度下一个元素需要跨过的字节数。例如3x4x5的 float64数组，其跨度stride=(160, 40, 8)

np.ones((10,5)).shape

np.ones((3,4,5), dtype=np.float64).strides

### Numpy数据类型体系

#使用dtype的超类(np.integer和np.floating)，可以跟np.issubtype函数结合，检查数组中包含的是整数、浮点数、字符串或python对象

ints = np.ones(10,dtype=np.uint16)

floats = np.ones(10,dtype=np.float32)

np.issubdtype(ints.dtype, np.integer)

np.issubdtype(floats.dtype, np.floating)

#调用dtype的mro方法查询其所有父类

np.float64.mro()

### 高级数组操作

### 数组重塑

#通过向数组的实例方法reshape传入一个表示新形状的元组即可实现'无需复制任何数据，改变数组形状'

arr = np.arange(8)

arr

arr.reshape((4,2))

#多维数组也可以被重塑

arr.reshape((4,2)).reshape((2,4))

#参数的形状其中一维可以是-1：表示维度的大小由数据本身推断而来

arr = np.arange(15)

arr.reshape((5,-1))

#由于数组的shape属性是一个元素，因此可以被传入reshape

other\_arr = np.ones((3,5))

arr.reshape(other\_arr.shape)

#把数组散开称为扁平化(flattening or raveling)

arr = np.arange(15).reshape((5,3))

arr

arr.ravel()

arr.flatten()

### C和Fortran顺序

C顺序：按行依次排列, 先经过更高的维度(axis 1优先0处理)

Fortran顺序：按列依次排列，后经过更高的维度(axis 1落后axis0处理)

#reshape和reval这样的函数可以接受一个表示数组存储数据顺序的参数'C' or 'F' (或者'A'和'K')

arr = np.arange(12).reshape((3, 4))

arr.ravel()

arr.ravel('F')

### 数组的合并和拆分

arr1 = np.array([[1,2,3],[4,5,6]])

arr2 = np.array([[7,8,9],[10,11,12]])

np.concatenate([arr1, arr2], axis=0)

np.concatenate([arr1,arr2], axis=1)

#np提供一些简便的方法:

np.vstack((arr1,arr2)) #= np.concatenate([arr1, arr2], axis=0)

np.hstack((arr1,arr2)) #= np.concatenate([arr1,arr2], axis=1)

#split将一个数组延指定轴拆为多个数组

from numpy.random import randn

arr =randn(5,2)

first,second, third = np.split(arr, [1,3])

#数组连接函数: concatenate, vstack, row\_stack, hstack, column\_stack, dstack, split, hsplit, vsplit, dsplit

### 堆叠辅助类：r\_和c\_

arr = np.arange(6)

arr1 = arr.reshape((3,2))

arr2 = randn(3,2)

np.r\_[arr1, arr2] #np.r\_[]行堆叠

np.c\_[np.r\_[arr1,arr2],arr] #np.c\_[]列堆叠

### 元素的重复操作：tile和repeat

arr = np.arange(3)

arr.repeat(3)

arr.repeat([2,3,4])

arr = randn(2,2)

arr.repeat(2, axis=0)

#如果没有设置轴向，则数组会被扁平化。

#也可以传入一组整数

arr.repeat([2,3], axis=0)

arr.repeat([2,3], axis=1)

#title的功能是沿指定轴向堆叠数组副本。

arr

np.tile(arr, 2)

#tile第二个参数是重复的数量，也可以是元组

np.tile(arr, (2,1))

### 花式索引及其等价函数: take/put

arr =np.arange(10) \* 100

inds = [7, 1, 2, 6]

arr[inds] #花式索引，抽取inds列表里对应位置的值

arr.take(inds)

#注意：put不接受axis参数，只会在数组扁平化版本(一维，C顺序)进行索引

#因此在使用其他轴索引设置元素时，最好使用花式索引

arr.put(inds, 42) #替换inds位置的值为42

inds=[2, 0, 2, 1]

arr = randn(2, 4)

arr.take(inds, axis=1)

#目前take和put的性能比花式索引好得多。但这可能是一个bug。

#当用整数数组选取大数组的子集时，还是注意这个问题

arr =randn(1000,50)

#500 lines random samples

inds = np.random.permutation(1000)[:500]

#caculate the running time

%timeit arr[inds]

%timeit arr.take(inds, axis=0)

### 广播

#将标量值跟数组合并，就会发生广播

arr = np.arange(5)

arr

arr \* 4

#通过减去平均值的方式对数组的每一列进行距平化处理

arr = randn(4, 3)

arr.mean(0)

demeaned = arr - arr.mean(0)

demeaned

demeaned.mean(0) #0 means axis=0, try demeaned(1), and demeaned()

row\_means =arr.mean(1)

row\_means.reshape((4,1))

demeaned = arr - row\_means.reshape((4,1))

demeaned.mean(1)

### 沿其他轴向广播

#通过特殊的np.newaxis属性以及全切片插入新轴

#generate an all zeros Array

arr = np.zeros((4, 4))

#添加新轴

arr\_3d = arr[:, np.newaxis, :]

arr\_3d.shape

arr\_1d = np.random.normal(size=3)

arr\_1d[:,np.newaxis]

#如果有三维数组，并希望对轴2进行距平化：

arr = randn(3, 4, 5)

depth\_means = arr.mean(2)

depth\_means

aa=depth\_means[:,:,np.newaxis] #aa.shape=(3,4,1)

demeaned = arr - depth\_means[:,:,np.newaxis]

demeaned.mean(2)

#在对指定轴进行距平化时，可以通过索引的一些技巧既不牺牲性能又增加通用性：

def mean\_axis(arr, axis=0):

means = arr.mean(axis)

#以下类似于N维的[:,:,np.newaxis]

indexer = [slice(None)] \* arr.ndim

indexer[axis] = np.newaxis

return arr - means[indexer]

demeaned2 = mean\_axis(arr, 2)

demeaned2.mean(2)

### 通过广播设置数组的值

#算术运算的广播原则也适用于通过索引机制设置数组值的操作

arr = np.zeros((4, 3))

arr[:] = 5

#复杂点儿的例子

col = np.array([1.28, -0.42, 0.44, 1.6])

arr[:]=col[:,np.newaxis]

arr[:2] = [[-1.37], [0.509]]

### ufuc高级应用

ufunc是unit function，通用函数实际上还有一些高级用法，可以丢开循环写出更为简洁的代码

### ufunc实例方法

#Numpy的各个二元ufunc都有执行特定矢量化运算的特殊方法。

#reduce接受一个数组参数，并通过二元运算聚合

arr = np.arange(10)

#累加arr

np.add.reduce(arr)

#sum方法

arr.sum()

#用np.logical\_and检查数组各行的值是否有序:

arr = randn(5, 5)

arr[::2].sort(1) #arr[::step]取arr[0], [2], [4]

arr[:,:-1]<arr[:,1:]

np.logical\_and.reduce(arr[:,:-1]<arr[:,1:], axis=1)

#等价

np.all(arr[:,:-1]<arr[:,1:], axis=1)

#accumulate跟reduce的关系就像cumsum跟sum的关系，它产生一个跟原生数组大小相同的中间累计数组

arr = np.arange(15).reshape((3, 5))

np.add.accumulate(arr, axis=1)

#outer用于计算两个数组的叉积

arr = np.arange(3).repeat([1,2,2]) #array([0, 1, 1, 2, 2])

np.multiply.outer(arr, np.arange(5))

#outer输出维度是两个输入数据维度之和

result = np.subtract.outer(randn(3,4),randn(5))

result.shape #(3,4,5)

#reduceat用于计算"局部约简"，即对数据各切片进行聚合的groupby运算. 其灵活性不如pandas的groupby，但在适当情况下速度快

#reduceat接受一组指示如何对值进行拆分和聚合的"面元边界"

arr = np.arange(10)

np.add.reduceat(arr, [0, 5, 8]) #求arr[0:5], arr[5:8], arr[8:]的和，结果: array([10, 18, 17], dtype=int32)

arr = np.multiply.outer(np.arange(4), np.arange(5))

arr

np.add.reduceat(arr, [0,2,4], axis=1)

#ufun的方法: reduce(x), accumulate(x), reduceat(x, bins), outer(x, y)

### 自定义ufunc

#有两个工具可以让你将自定义函数像ufunc那样使用

#numpy.frompyfunc接受一个python函数以及两个分别表示输入输出参数数量的整数

def add\_elements(x, y):

return x + y

add\_them = np.frompyfunc(add\_elements, 2, 1)

add\_them(np.arange(8), np.arange(8))

#使用frompyfunc创建的函数总是返回python对象数组

#numpy.vectorize可以更灵活

add\_them = np.vectorize(add\_elements, otypes=[np.float64])

add\_them(np.arange(8), np.arange(8))

#np.vertorize缺点是运行速度比Numpy自带的基于C的ufunc慢

arr = randn(10000)

%timeit add\_them(arr,arr)

%timeit np.add(arr,arr)

### 结构化和记录式数组

#数组内的数据具有不通类型和结构，类似struct

dtype = [('x', np.float64),('y', np.int32)]

sarr = np.array([(1.5, 6),(np.pi, -2)], dtype=dtype)

sarr

#定义结构化dtype的方式有很多，典型的是元组列表。各元组的格式为(field\_name, field\_data\_type)

#数组的元素成了元组式对象

sarr[0]

sarr[0]['x'] #1.5

sarr[0]['y'] #6

sarr['x'] #(1.5, 3.14)

#嵌套dtype和多维字段

#定义结构化dtype是，可以设置一个形状(整数 or 元组)

dtype = [('x', np.int64, 3),('y',np.int32),('z', np.float64)]

arr = np.zeros(4, dtype=dtype)

arr

arr[0]['x']

arr['x'] #得到二维数组

#dtype可以嵌套dtype

dtype = [('x',[('a','f8'),('b','f4')]),('y', np.int32)]

data = np.array([((1,2),5),((3,4),6)], dtype=dtype)

data['x']

data['y']

data['x']['a']

#numpy的可变形字段和嵌套是一种强大的功能。pandas的DataFrame不直接支持此功能，但pd的分层索引机制跟这个功能差不多

### 为什么要用结构化数组

#结构化数组提供非常快速高效的磁盘数据读写(内存映像)、网络传输等功能

#结构化数组的另一个常见用法是：将数据文件写成定长记录字节流，只要知道文件的格式(记录大小、元素的顺序、字节数以及数据类型)，就可以用np.fromfile将数据读入内存

#结构化数组操作：numpy.lib.recfunctions. 有一些用于增减字段或执行基本连接运算的工具。这些工具一般都需要创建一个新数组以便对dtype修改(比如添加或删除一列)

#更多有关排序的话题

#ndarray.sort()是就地排序，不产生新数组

arr = randn(6)

arr.sort()

arr = randn(3, 5)

arr

arr[:, 0].sort() #sort first column values in-place

arr

#numpy.sort()函数会创建一个排序副本，它所接受的参数(kind...)跟ndarray.sort()一样

arr = randn(5)

arr

np.sort(arr) #func

arr

arr = randn(3, 5)

arr.sort(axis=1) #sort by axis

arr

#两个排序方法都不可以被设置为降序，用values[::-1]可以返回一个反序的列表

arr[:,::-1]

### 间接排序: argsort和lexsort

#数据分析中，需要根据一个或多个键对数据集排序。

values = np.array([5, 0, 1, 3, 2])

indexer = values.argsort() #indexer返回排序后索引顺序; array([1, 2, 4, 3, 0], dtype=int32)

values[indexer]

arr = randn(3, 5)

arr[0] = values

arr

arr[:, arr[0].argsort()] #sort a single row(arr[0]) of nxm array

first\_name = np.array(['Bob', 'Jane', 'Steve', 'Bill', 'Barbara'])

last\_name = np.array(['Jones', 'Arnold', 'Arnold', 'Jones', 'Walters'])

sorter = np.lexsort((first\_name, last\_name))

for i in zip(last\_name[sorter], first\_name[sorter]):

print(i)

### 其他排序方法

#稳定的排序算法会保持等价元素的相对位置

values = np.array(['2:first','2:second', '1:first', '1:second', '1:third'])

key = np.array([2, 3, 1,1, 1])

indexer = key.argsort(kind = 'mergesort')

indexer

values.take(indexer)

#mergesort(合并排序)是唯一的稳定排序; 平均性能比quicksort要差

#数组排序算法: kind= 'quicksort', 'mergesort', 'heapsort'

### numpy.searchsorted在有序数组中查找元素

#searchsorted是一个二分法查找数组方法，只要把值插入到返回的那个位置就能维持数组的有序性

arr = np.array([0, 1, 7, 12, 15])

arr.searchsorted(9) #3

arr= np.array([0, 0, 0, 1,1,1,1])

arr.searchsorted([0,1])

arr.searchsorted([0,1], side='right') #insert from the 'right'

#searchsorted另一个用法

#假设我们有一个数据数组，还有一个表示'面元边界'的数组，希望用面元数组把数据数组拆分开

data = np.floor(np.random.uniform(0, 10000, size=50))

bins = np.array([0, 100, 1000, 5000, 10000])

#为了得到各数据点所属区间编号(1表示[0, 100))，直接使用searchsorted

labels = bins.searchsorted(data)

#通过pandas的groupby使用该结果即可对原数据集进行拆分

Series(data).groupby(labels).mean()

#或者使用Numpy的digitize函数

labels = np.digitize(data, bins)

### Numpy的matrix类

#Numpy的线性代数语法繁琐， 矩阵操作需要用numpy.dot，numpy的索引语义也不同。所以不方便把matlab代码移植到python。

#例如：x[1,:], x[:, 1]在python中会产生一维数组；但在matlab中还是二维数组

x = randn(4,5) #(4x5)

x[:, 0] #1行

y = x[:, :1] #1列

#x'\*y表示如下:

np.dot(x.T,y) #.T转置

#Numpy提供matrix类，单行或列以二维形式返回，\*直接代表矩阵乘法

xm = np.matrix(x)

ym = xm[:, 0]

(ym.T \* xm)[:,:4] \* ym

#matrix特殊属性I，返回矩阵的逆

xm=xm[:,:4] #方阵

xm.I \* xm

#不建议用numpy.matrix替代正规的ndarray,因为应用面比较窄，对于个别带大量线性代数运算的函数可以将函数参数转为matrix类型，然后返回之前用np.asarray(不会复制任何数据)将其转换会正规的ndarray

return np.asarray(xm)

### 高级数组输入输出

#np.save和np.load可用于读写磁盘上以二进制存储的数组。

#numpy中还可以使用内存映像(memory map)，使你处理在内存中放不下的数据集

### 内存映像文件

#np.save和np.load可用于读写磁盘上以二进制存储的数组。

#numpy中还可以使用内存映像(memory map)，使你处理在内存中放不下的数据集

#Numpy实现了一个类似ndarray的memmap对象，允许将大文件分成小段进行读写，而不是一次性将整个数组读入内存。memmap也拥有跟普通数组一样的方法, 因此基本上只要能用于ndarray的算法也能用于memmap

#使用函数np.memmap并传入一个文件路径、数据类型、形状以及文件模式，即可创建一个新的memmap

#10000 is not working in win10, try 100 instead.

mmap = np.memmap('mymmap', dtype='float64', mode='w+', shape=(10000, 10000))

#对memmap切片会返回磁盘上的数据视图

section = mmap[:5]

#将数据赋值给这些视图：数据会先被缓存在内存中， 调用flush即可写入磁盘

section[:] = np.random.randn(5, 10000)

mmap.flush()

mmap

del mmap

#只要某个内存映像超出了作用域，它就会被垃圾回收器回收。之前对其所做的任何修改都会被写入磁盘

#当打开一个已经存在的内存映像时，仍然需要指明数据类型和形状，因为磁盘上是二进制数而已。

mmap = np.memmap('mymmap', dtype='float64', mode='r+',shape=(10000, 10000))

mmap

### HDF5及其他数组存储方式

### 性能建议

#PyTables和h5py这两个Python项目可以将Numpy的数组数据存储为高效且可压缩的HD5格式(层次化数据格式)

#可以将几百GB的数据存储为HDF5格式

#PyTables提供一些结构化数组的高级查询功能，能添加索引以提升查询速度

#性能建议

1. 将Python循环和条件逻辑转换为数组运算和布尔数组运算

2. 尽量使用广播

3. 避免复制数据，尽量使用数组视图(切片)

4. 利用ufunc及其各种方法

5. 如果Numpy无法满足性能要求，可以考虑用C或Cython(不需要太多精力就可以达到C那样的性能)。

### 连续内存的重要性

#数组的内存布局对计算速度造成极大的影响。例如对C顺序存储的数组行求和速度最快

#通过ndarray的flags属性可以查看数组存储的信息

arr\_c = np.ones((1000,1000), order='C')

arr\_f = np.ones((1000,1000), order='F')

arr\_c.flags

arr\_f.flags.f\_contiguous

#如果对两个数组求和，理论上，arr\_c会比较快，因为arr\_c在内存里连续

%timeit arr\_c.sum(1) #slow

%timeit arr\_f.sum(1) #fast

#如果数组内存顺序不符合要求，使用copy，并传入'C'或'F'即可解决

arr\_fc = arr\_f.copy('C')

#在构造数组视图时，其结果并不一定是连续的

arr\_c[:50].flags.contiguous #true

arr\_c[:,:50].flags #False

### 其他加速手段: Cython, f2py, C

#Cython代码运行速度很快，可以把Cython看成有静态类型并能嵌入C函数的Python

#例子: Cython对一维数组所有元素求和

from numpy cimport ndarray, float64\_t

def sum\_elements(ndarray[float64\_t] arr):

cdef Py\_ssize\_t i, n = len(arr)

cdef float64\_t result =0

for i in range(n):

result + = arr[i]

return result

#Cython处理这段代码时，先将其翻译成C代码，然后编译并创建一个Python扩展。

#Cython只比编写纯Python代码多花一点时间，还能跟Numpy紧密结合。

#一般的工作流程是：1. 得到能在Python中运行的算法，2.然后将其翻译成Cython(添加类型定义，并完成一些其他必要的工作)

#其他有关Numpy的高性能代码编写手段还有f2py以及利用C语言编写Python扩展

### 例子：数据处理和画图

import os

import sys

import re

import numpy as np

from numpy.random import randn

import pandas as pd

from pandas import Series, DataFrame

from pandas\_datareader import data as web

from numpy import nan as NA

#import re

#import json

import matplotlib.pyplot as plt

from numpy import nan as NA

from datetime import datetime,time

obj\_str1 = '5\_StaticPB3\_'

obj\_str2 = '5\_2P8\_'

source\_str1 = 'Main Rx Pwr='

source\_str2 = '5\_2P8\_'

NN=-1

def generate\_2Y\_figure(df):

fig = plt.figure()

ax1 = fig.add\_subplot(111)

ax1.plot(df['Main Rx Pwr for 1x'],'k-',color='red',label="Main Rx Pwr for 1x");

ax1.legend(loc=1)

ax1.set\_ylabel('Main Rx Pwr for 1x')

ax2 = ax1.twinx() # this is the important function

ax2.plot(df['Pilot\_Call'],'k-',color='blue',label = 'Pilot\_Call')

ax2.legend(loc=2)

#ax2.set\_xlim([50, 55]); #set x lim

ax2.set\_ylabel('Pilot\_Call')

#ax2.set\_xlabel('Main Rx Pwr for 1x')

plt.show()

def drawAxPlot(df,ax, ylabelName, ax\_color, labelstr, ax\_index, linetype='k.'):

ax.grid(True)

ax.set\_ylabel(ylabelName)

ax.plot(df, linetype,color=ax\_color,label=labelstr)

ax.set\_xlim(ax\_index[0], ax\_index[-1]) #set x

ax.legend(loc='best') #set position of label

#def pwr\_data\_analysis(file\_dir):

for root, dirs, files in os.walk(sys.path[0]):

pass

for file\_name in files:

if '.log' in file\_name and '.zip' not in file\_name:

pwr\_data=[]

time\_stamp=[]

pwr\_time\_idx=[]

pilot\_call\_idx=[]

pilot\_call\_time\_idx=[]

pilot\_idle\_idx=[]

pilot\_idle\_time\_idx=[]

current\_pilot=52.5

#cell\_dsp\_data={'pwr\_data':[], 'pilot\_call\_idx':[]}

try:

with open(file\_name,'r') as rdfile:

with open('result\_'+file\_name.split('.')[0],'w') as wrfile:

line = rdfile.readline()

#while line:

try:

for line in rdfile:

m = re.match(r'(\d+:\d+:\d+:\d+)>.\*',line)

if m is not None:

stamp = m.group(1)

mytime = time(int(stamp.split(':')[0]),int(stamp.split(':')[1]),int(stamp.split(':')[2]),1000\*int(stamp.split(':')[3])) #change ms to us

t = re.match(r'.\*PILOT\_PN\.?[01]?=(\d+)',line)

if t is not None:

pass

#current\_pilot=float(t.group(1))

pilot\_idle\_idx.append(float(t.group(1)))

pilot\_idle\_time\_idx.append(mytime)

p = re.match(r'.\*PilotPN\.0=(\d+)',line) #PilotPN.0=51

if p is not None:

if NN<0: #record pilot data with delay NN

current\_pilot=float(p.group(1))

pilot\_call\_idx.append(current\_pilot)

pilot\_call\_time\_idx.append(mytime)

else:

NN-=1

if source\_str1 in line:

n = re.match(r'.\*Main Rx Pwr=(-?\d+.\d+),.\*',line)

#wrfile.write(line)

if n is not None:

#wrfile.write(n.group(1))

pwr\_data.append(float(n.group(1)))

#time\_stamp.append(stamp)

pwr\_time\_idx.append(mytime)

#cell\_dsp\_data['pilot\_call\_idx'].append(current\_pilot)

#cell\_dsp\_data['pwr\_data'].append(float(n.group(1)))

#line = rdfile.readline()

except StopIteration as e:

print ('end of ', file\_name)

#print(pwr\_data)

pwr\_data\_series=Series(pwr\_data, index=pwr\_time\_idx)

pilot\_call\_data\_series=Series(pilot\_call\_idx, index=pilot\_call\_time\_idx)

pilot\_idle\_data\_series=Series(pilot\_idle\_idx, index=pilot\_idle\_time\_idx)

#pwr\_data\_series=Series(pwr\_data, index=time\_stamp) #wrong, X axis is wrong

pwr\_df=DataFrame(pwr\_data\_series, columns=['Main Rx Pwr for 1x'])

pilot\_call\_df=DataFrame(pilot\_call\_data\_series, columns=['Pilot\_Call'])

pilot\_idle\_df=DataFrame(pilot\_idle\_data\_series, columns=['Pilot\_idle'])

#remove duplicated index data

if not pwr\_df.index.is\_unique:

pwr\_df=pwr\_df.groupby(level=0).mean()

if not pilot\_call\_df.index.is\_unique:

pilot\_call\_df=pilot\_call\_df.groupby(level=0).mean()

if not pilot\_idle\_df.index.is\_unique:

pilot\_idle\_df=pilot\_idle\_df.groupby(level=0).mean()

#df = DataFrame(cell\_dsp\_data, index=pwr\_time\_idx)

pwr\_df.to\_csv(file\_name.replace('ETS\_INFRA\_BEIZTE\_','\_').replace('.log','\_pwr')+'.csv')

pilot\_call\_df.to\_csv(file\_name.replace('ETS\_INFRA\_BEIZTE\_','\_').replace('.log','\_pilot\_call')+'.csv')

pilot\_idle\_df.to\_csv(file\_name.replace('ETS\_INFRA\_BEIZTE\_','\_').replace('.log','\_pilot\_idle')+'.csv')

#remove unreasonable data

pwr\_df[(pwr\_df<-100).any(1)]=NA

pilot\_call\_df[(pilot\_call\_df>100).any(1)]=NA #remove 460

pilot\_call\_df[(pilot\_call\_df<30).any(1)]=NA #remove 460

pilot\_idle\_df[(pilot\_idle\_df>100).any(1)]=NA #remove 460

pilot\_idle\_df[(pilot\_idle\_df<30).any(1)]=NA #remove 460

#merge two DF to align time

cell\_dsp\_data=pd.merge(pwr\_df, pilot\_call\_df, how='outer',left\_index=True,right\_index=True)

cell\_dsp\_data=pd.merge(cell\_dsp\_data, pilot\_idle\_df, how='outer',left\_index=True,right\_index=True)

#setup size of plt figure

plt.rcParams["figure.figsize"]=[16.8, 9]

fig=plt.figure()

ax1=fig.add\_subplot(3,1,1)

ax2=fig.add\_subplot(3,1,2)

ax3=fig.add\_subplot(3,1,3)

xindex=cell\_dsp\_data.index

drawAxPlot(df=cell\_dsp\_data['Main Rx Pwr for 1x'], ax=ax1, ylabelName='Main Rx Pwr for 1x', ax\_color='blue', linetype='k-', ax\_index=xindex, labelstr='Main Rx Pwr for 1x')

drawAxPlot(df=cell\_dsp\_data['Pilot\_Call'], ax=ax2, ylabelName='Pilot PN\_Call', ax\_color='red', linetype='k.', ax\_index=xindex, labelstr='Pilot PN Call')

drawAxPlot(df=cell\_dsp\_data['Pilot\_idle'], ax=ax3, ylabelName='Pilot PN\_Idle', ax\_color='green', linetype='k.', ax\_index=xindex, labelstr='Pilot PN idle')

fig.autofmt\_xdate() #update format

plt.savefig(file\_name.replace('ETS\_INFRA\_BEIZTE\_','\_').replace('.log','\_')+'.png') #save to png

plt.show() #show picture

generate\_2Y\_figure(cell\_dsp\_data)

#sys.exit(0)

except Exception as e:

#except StopIteration as e:

print('Error Msg:', e)

sys.exit(0)

#pwr\_data\_analysis(sys.path[0])

def dataClean(df):

return df

### 例子：mmap使用测试程序

#win os 32bit 2\*\*32bit max(2Gbit – 4Gbit); win os 64bit, 2\*\*64bit

import itertools

import mmap

import os

files = []

mmaps = []

file\_names= []

mmap\_cap=0

bytes\_per\_mmap = 100 \* 1024 \*\* 2

try:

for i in itertools.count(1):

file\_name = r"c:\%d.tst" % i

file\_names.append(file\_name)

f = open(file\_name, "w+b")

files.append(f)

mm = mmap.mmap(f.fileno(), bytes\_per\_mmap)

mmaps.append(mm)

mmap\_cap += bytes\_per\_mmap

print ("Created %d writeable mmaps containing %d MB" % (i, mmap\_cap/(1024\*\*2)) )

#Clean up

finally:

print ("Removing mmaps..." )

for mm, f, file\_name in zip(mmaps, files, file\_names):

mm.close()

f.close()

os.remove(file\_name)

print ("Done..." )

### a

a

### 利用Python进行数据分析(末尾)

## 其他PYTHON笔记

### Python异常处理

异常处理的语句结构

try:

<statements> #运行try语句块，并试图捕获异常

except <name1>:

<statements> #如果name1异常发现，那么执行该语句块。

except (name2, name3):

<statements> #如果元组内的任意异常发生，那么捕获它

except <name4> as <variable>:

<statements> #如果name4异常发生，那么进入该语句块，并把异常实例命名为variable

except:

<statements> #发生了以上所有列出的异常之外的异常

else:

<statements> #如果没有异常发生，那么执行该语句块

finally:

<statement> #无论是否有异常发生，均会执行该语句块。

说明

else和finally是可选的，可能会有0个或多个except，但是，如果出现一个else的话，必须有至少一个except。

不管你如何指定异常，异常总是通过实例对象来识别，并且大多数时候在任意给定的时刻激活。一旦异常在程序中某处由一条except子句捕获，它就死掉了，除非由另一个raise语句或错误重新引发它。

raise语句

raise语句用来手动抛出一个异常，有下面几种调用格式：

raise #可以在raise语句之前创建该实例或者在raise语句中创建。

raise #Python会隐式地创建类的实例

raise name(value) #抛出异常的同时，提供额外信息value

raise # 把最近一次产生的异常重新抛出来

raise exception from E

例如：

抛出带有额外信息的ValueError: raise ValueError('we can only accept positive values')

当使用from的时候，第二个表达式指定了另一个异常类或实例，它会附加到引发异常的\_\_cause\_\_属性。如果引发的异常没有捕获，Python把异常也作为标准出错消息的一部分打印出来：

比如下面的代码：

try:

1/0

except Exception as E:

raise TypeError('bad input') from E

执行的结果如下：

Traceback (most recent call last):

File "hh.py", line 2, in <module>

1/0

ZeroDivisionError: division by zero

The above exception was the direct cause of the following exception:

Traceback (most recent call last):

File "hh.py", line 4, in <module>

raise TypeError('bad input') from E

TypeError: bad input

assert语句

assert主要用来做断言，通常用在单元测试中较多，到时候再做介绍。

with...as语句

with语句支持更丰富的基于对象的协议，可以为代码块定义支持进入和离开动作。

with语句对应的环境管理协议要求如下：

环境管理器必须有\_\_enter\_\_和\_\_exit\_\_方法。

\_\_enter\_\_方法会在初始化的时候运行，如果存在ass子在，\_\_enter\_\_函数的返回值会赋值给as子句中的变量，否则，直接丢弃。

代码块中嵌套的代码会执行。

如果with代码块引发异常，\_\_exit\_\_(type,value,traceback)方法就会被调用（带有异常细节）。这些也是由 sys.exc\_info返回的相同值.如果此方法返回值为假，则异常会重新引发。否则，异常会终止。正常 情况下异常是应该被重新引发，这样的话才能传递到with语句之外。

如果with代码块没有引发异常，\_\_exit\_\_方法依然会被调用，其type、value以及traceback参数都会以None传递。

下面为一个简单的自定义的上下文管理类。

class Block:

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

print('entering to the block')

return self

def prt(self, args):

print('this is the block we do %s' % args)

def \_\_exit\_\_(self,exc\_type, exc\_value, exc\_tb):

if exc\_type is None:

print('exit normally without exception')

else:

print('found exception: %s, and detailed info is %s' % (exc\_type, exc\_value))

return False

with Block() as b:

b.prt('actual work!')

raise ValueError('wrong')

如果注销到上面的raise语句，那么会正常退出。

在没有注销掉该raise语句的情况下，运行结果如下：

entering to the block

this is the block we do actual work!

found exception: <class 'ValueError'>, and detailed info is wrong

Traceback (most recent call last):

File "hh.py", line 18, in <module>

raise ValueError('wrong')

ValueError: wrong

### 获取日期和时间

import time

In [86]: time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(time.time()))

Out[86]: '2017-08-16 16:12:51'

----------分割线------------

time.strftime里面有很多参数，可以让你能够更随意的输出自己想要的东西：

下面是time.strftime的参数：

strftime(format[, tuple]) -> string

将指定的struct\_time(默认为当前时间)，根据指定的格式化字符串输出

python中时间日期格式化符号：

%y 两位数的年份表示（00-99）

%Y 四位数的年份表示（000-9999）

%m 月份（01-12）

%d 月内中的一天（0-31）

%H 24小时制小时数（0-23）

%I 12小时制小时数（01-12）

%M 分钟数（00=59）

%S 秒（00-59）

%a 本地简化星期名称

%A 本地完整星期名称

%b 本地简化的月份名称

%B 本地完整的月份名称

%c 本地相应的日期表示和时间表示

%j 年内的一天（001-366）

%p 本地A.M.或P.M.的等价符

%U 一年中的星期数（00-53）星期天为星期的开始

%w 星期（0-6），星期天为星期的开始

%W 一年中的星期数（00-53）星期一为星期的开始

%x 本地相应的日期表示

%X 本地相应的时间表示

%Z 当前时区的名称

%% %号本身

### Pandas学习

DataFrame类：

DataFrame有四个重要的属性：

index：行索引。

columns：列索引。

values：值的二维数组。

name：名字。

这个类是Pandas最重要的类之一。

构建方法，DataFrame(sequence)，通过序列构建，序列中的每个元素是一个字典。

frame=DateFrame构建完之后，假设frame中有'name','age','addr'三个属性，可以使用fame['name']查看属性列内容，也可以fame.name这样直接查看。

frame按照'属性提取出来的每个列是一个Series类。

DataFrame类可以使用布尔型索引。

groupby(str|array...)函数：可以使用frame中对应属性的str或者和frame行数相同的array作为参数还可以使用一个会返回和frame长度相同list的函数作为参数，如果使用函数做分组参数，这个用做分组的函数传入的参数将会是fame的index，参数个数任意。使用了groupby函数之后配合,size()函数就可以对groupby结果进行统计。

groupby后可以使用：

size()：就是count

sum()：分组求和

apply(func，axis=0)：在分组上单独使用函数func返回frame，不groupby用在DataFrame会默认将func用在每个列上，如果axis=1表示将func用在行上。

reindex(index,column,method)：用来重新命名索引，和插值。

size()：会返回一个frame，这个frame是groupby后的结果。

sum(n).argsort()：如果frame中的值是数字，可以使用sum函数计算frame中摸个属性，各个因子分别求和，并返回一个Series，这个Series可以做为frame.take的参数，拿到frame中对应的行。

pivot\_table(操作str1,index=str2,columns=str3,aggfunc=str4)透视图函数：

str1：是给函数str4作为参数的部分。

str2：是返回frame的行名。

str3：是返回frame的列名。

str4：是集合函数名，有'mean','sum'这些，按照str2，str3分组。

使用透视图函数之后，可以使用.sum()这类型函数，使用后会按照index和columns的分组求和。

order\_index(by,ascending):

返回一个根据by排序，asceding=True表示升序，False表示降序的frame

concat(list)：将一个列表的frame行数加起来。

ix[index]：就是行索引，DataFrame的普通下标是列索引。

take(index)：作用和ix差不多，都是查询行，但是ix传入行号，take传入行索引。

unstack()：将行信息变成列信息。

apply(func，axis=0)和applymap(func)：apply用在DataFrame会默认将func用在每个列上，如果axis=1表示将func用在行上。applymap表示func用在每个元素上。

combine\_first(frame2)：combine\_first会把frame中的空值用frame1中对应位置的数据进行填充。Series方法也有相同的方法。

stack()函数，可以将DataFrame的列转化成行，原来的列索引成为行的层次索引。（stack和unstack方法是两个互逆的方法，可以用来进行Series和DataFrame之间的转换）

duplicated()：返回一个布尔型Series，表示各行是否重复。

drop\_duplicates()：返回一个移除了重复行后的DataFrame

pct\_change()：Series也有这个函数，这个函数用来计算同colnums两个相邻的数字之间的变化率。

corr()：计算相关系数矩阵。

cov()：计算协方差系数矩阵。

corrwith(Series|list,axis=0)：axis=0时计算frame的每列和参数的相关系数。

Series类：

两个重要的属性：

value：存放series值的一个数组。

index：Series的下标索引。

name：就是Series的名字

index有一个name属性。

可以通过Series(list,index=None)的形式来创建一个Series类，index表示的是用下标访问对应的数据。

也可以直接使用字典创建Series(dict)

value\_counts()：该方法可以用来统计series类中各因子出现的次数，返回一个带统计结果的series。

fillna(str)：给series中的空值赋值。

plot()函数：可以用来给带统计结果的函数画图。但是要配合matplotlib使用

notnull()：返回一个判断series位置是否空值的布尔型索引。

sum()：如果是数字型Series，可以求和。

cumsum()：如果是数字型Series，可以返回一个累加的Series。

searchsorted()：在数字Series中定位一个数字的位置，这个数字不完全相同，但接近。

map(func)：将Series中的元素，每个都当做func的参数使用一遍，返回执行结果组成的Series

unique()：类似于sql中的distinct

isnull()/notnull()：返回一个布尔型索引

order()：对值进行排序。

order\_value()：对索引进行排序。

unstack()方法：将Series的层次索引转换成列索引，变成一个DataFrame。

replace()：可以用list或dict作为参数，替换需要替换的值

str属性：Series.str后会将Series单做一个字符串的集合，这个集合能够使用字符串的操作，例如：

data=Series(['abc','bcd','cde'])

data.str[1]

输出;

1 b

2 c

3 d

read\_table()函数：读dat文件。

import pandas as pd

mnames=['movie\_id','title','genres']

movies=pd.read\_table(r'C:\Users\Administrator\Desktop\python for data analysis data\pydata-book-master\ch02\movielens\movies.dat',sep='::',header=None,names=mnames)

read\_csv()函数：可以将frame文件直接读成frame。

movies=pd.read\_csv(r'names\job1880.txt',names=column)

read\_csv函数有一个sep参数，设置分隔符，可以给这个参数传入正则表达式。

skiprows参数，参数是一个list，表示读取文件的时候，跳过list中的几行，第一行为0

read\_excel()函数

可以直接读取excel文件为DataFrame

merge(frame1,frame2)：

根据两个frame列的名字自动合并，返回一个frame。

此函数可以通过on，left\_on，right\_on三个属性来设置怎么frame1和frame2通过什么属性来进行连接。

concat函数：

可以将DataFrame或者Series按照axis的方向堆积起来。

cut函数和qcut函数：

可以将一些离散值分箱，cut函数用的是数值区间将数值分箱，qcut用的是分位数。

cut用在长度相等的桶，qcut用在大小相等的桶。

to\_datetime(str):

解析常用的时间格式。

date\_range函数：

产生时间序列。

### 工厂函数和内建函数区别

工厂函数的运行结果是返回一个类的实例，工厂函数看上去有点像函数，实质上他们是类，当你调用它们时，实际上是生成了该类型的一个实例，就像工厂生产货物一样。类型工厂函数,指的是不通过类而是通过函数来创建对象；内建函数返回的是一个值

内建函数：比如str类型的len() ，要求把该类型作为参数，放在内建函数中调用

内建方法：如list的的pop操作，需要以list对象调用该方法，li.pop()

工厂函数：创建一个返回值为某个对象类型的内建函数，如list()返回一个list对象,open('/etc/conf.md')返回一个文件对象

### python3.52 八进制

a= 0o23

python3.52取消了长整型

### enumerate用法

今日发现一个新函数 enumerate 。一般情况下对一个列表或数组既要遍历索引又要遍历元素时，会这样写：

for i in range (0 , len (list )):

print i , list [ i ]

但是这种方法有些累赘，使用 内置enumerrate函数会有更加直接，优美的做法，先看看enumerate的定义：

def enumerate (collection ):

'Generates an indexed series: (0,coll[0]), (1,coll[1]) ...'

i = 0

it = iter (collection )

while 1 :

yield (i , it . next ())

i += 1

enumerate会将数组或列表组成一个索引序列。使我们再获取索引和索引内容的时候更加方便如下：

for index ， text in enumerate (list )):

print index , text在 cookbook里介绍，如果你要计算文件的行数，可以这样写：

count = len (open (thefilepath , ‘ rU ’ ). readlines ())

前面这种方法简单，但是可能比 较慢，当文件比较大时甚至不能工作，下面这种循环读取的方法更合适些。

Count = - 1

For count , line in enumerate (open (thefilepath , ‘ rU ’ ))：

Pass

Count += 1

### ini解析

'write a .ini file, which is case-insensitive, [DEFAULT] save default values for all other sections'

import configparser

config = configparser.ConfigParser()

config['DEFAULT'] = {'ServerAliveInterval': '45', 'Compression': 'yes', 'CompressionLevel': '9'}

config['bitbucket.org'] = {}

config['bitbucket.org']['User'] = 'hg'

config['topsecret.server.com'] = {}

topsecret = config['topsecret.server.com']

topsecret['Port'] = '50022' # mutates the parser

topsecret['ForwardX11'] = 'no' # same here

config['DEFAULT']['ForwardX11'] = 'yes'

with open('example.ini', 'w') as configfile:

config.write(configfile)

import configparser

config = configparser.ConfigParser()

config.sections()

config.read('example.ini')

print(config.sections())

print('DEFAULT' in config)

print('bitbucket.org' in config)

print('bytebong.com' in config)

print('-'\*5)

print(config['bitbucket.org']['User'])

print(config['bitbucket.org']['compression'])

print(config['DEFAULT']['Compression'])

print('-'\*5)

topsecret = config['topsecret.server.com']

print(topsecret['ForwardX11'])

print(topsecret['Port'])

for key in config['bitbucket.org']:

print(key)

print(config['bitbucket.org']['ForwardX11'])

### Multi-files Open

#Multiple files open

row = 0 #assume left\_top is (1,1)

col = 1

result = 0 #0=equal, 1=different charactor, 2=different length of line, 3=different lengh of row

#length = 0

from contextlib import ExitStack

with ExitStack() as stack:

files = [stack.enter\_context(open(fname)) for fname in ['a.txt', 'b.txt']]

#for i, j, k in zip(files[0], files[1], files[2]):

for linea, lineb in zip(files[0], files[1]): #readline from muli-files

row+=1 #increase row

col = 1

#length = max(len(linea), len(lineb))

for ca, cb in zip(linea, lineb):

col += 1

if ca!=cb: #if ca='\n', ca='a', return

result = 1

print(result, row, col)

return result, row, col

#break #find un-equal charactor

print(result, row, col)

return result, row, col

### feedparser

Python之feedparser

参考链接：http://blog.csdn.net/lanchunhui/article/details/51020566

feedparser是一个Python的Feed解析库，可以处理RSS ，CDF，Atom 。使用它我们可从任何 RSS 或 Atom 订阅源得到标题、链接和文章的条目了。

RSS(Really Simple Syndication,简易信息聚合)是一种描述和同步网站内容的格式你可以认为是一种定制个性化推送信息的服务。它能够解决你漫无目的的浏览网页的问题。它不会过时，信息越是过剩，它的意义也越加彰显。网络中充斥着大量的信息垃圾，每天摄入了太多自己根本不关心的信息。让自己关注的信息主动来找自己，且这些信息都是用户自己所需要的，这就是RSS的意义。

parse() 方法

feedparser 最为核心的函数自然是 parse() 解析 URL 地址的函数。

我们知道，每个RSS和Atom订阅源都包含一个标题（d.feed.title）和一组文章条目(d.entries)

通常每个文章条目都有一段摘要（d.entries[i].summary）,或者是包含了条目中实际文本的描述性标签（d.entries[i].description）

>>>import feedparser

>>>d=feedparser.parse('http://feed.cnblogs.com/blog/sitehome/rss')

d.feed

feed 对应的值也是一个字典

>>>d['feed']['title']

'博客园\_首页'

>>>d.feed.title #通过属性的方式访问

'博客园\_首页'

>>>d.feed.subtitle

'代码改变世界'

>>>d.feed.link

'uuid:1b90fd0c-6080-4ea5-86b1-b87c64b95d69;id=4466'

d.entries

该属性类型为列表，表示一组文章的条目

>>>type(d.entries) #类型为列表

<class 'list'>

>>>len(d.entries) #一共20篇文章

20

>>>[e.title for e in d.entries][:5] #列出前5篇文章的标题

['僵尸进程 - 乌龟运维', '深入浅出 spring-data-elasticsearch - 基本案例详解（三 - 泥瓦匠BYSocket', 'js继承 - huanglei-', 'ionic 使用了 crosswalkwebview 所产生的bug 及 解决方案 - FEer\_llx', '关于并发你真的了解吗？（二） - 心灬无痕']

>>>d.entries[0].summary #第一篇文章的摘要 和d.entries[0].description功能一样

'在UNIX系统中，僵尸进程是指完成执行（通过exit系统调用，或运行时发生致命错误或收到终止信号所致）但在操作系统的进程表中仍 然有一个表项（进程控制块PCB），处于”终止状态“的进程。这发生于子进程需要保留表项以允许其父进程读取子进程的exit status：一旦退出态通过wait系统调用读取，僵尸'

### threading例程

# !/usr/bin/env python

'''This is the test excecises for chapter 18 of CORE\_PYTHON\_PROGRAMMING

Threading

'''

import sys

import os

from datetime import datetime

from time import ctime, sleep

import threading

from random import randint

from queue import Queue

import logging

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s: %(message)s')

handler = logging.FileHandler('trace.log',mode='a')

handler.setLevel(logging.DEBUG)

handler.setFormatter(formatter)

# create a logging format

formatterconsole = logging.Formatter('%(message)s')

handlerconsole = logging.StreamHandler()

handlerconsole.setFormatter(formatterconsole)

handlerconsole.setLevel(logging.DEBUG)

if logger.hasHandlers() is False:

logger.addHandler(handler)

logger.addHandler(handlerconsole)

logger.info("HasHandlers() is False, add two new handlers")

else:

logger.info("HasHandlers() is True, previous handlers existed, don't add new ones")

# Global variable definition here

debug =True

# Class definition here

class **MyThread** (threading.Thread):

def \_\_init\_\_(self, func, name='', \*kargs):

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

self.name = name

self.func = func

self.args = kargs

logger.debug('kargs=%s',kargs)

def getResult(self):

pass

return self.res

def run(self):

logger.debug ('starting %s at %s', self.name, ctime())

#self.res = apply(self.func, self.args)

self.res = self.func(\*self.args)

logger.debug("%s %s %s", self.name, 'finished at:', ctime())

def fib(x):

#logger.debug('x=%s',x)

sleep(0.0005)

if x<2:

return 1

#return (fib(x[0]-2)+fib(x[0]-1))

return (fib(x-2)+fib(x-1))

def fac(x):

logger.debug('x=%s',x)

sleep(0.1)

if x<2:

return 1

#return (x[0]\*fac(x-1))

return (x\*fac(x-1))

def mysum(x):

logger.debug('x=%s',x)

sleep(0.1)

if x<2:

return 1

#return (x[0]+mysum(x[0]-1))

return (x+mysum(x-1))

funcs = [fib, fac, mysum]

n=12

def main():

nfuncs = range(len(funcs))

logger.debug ("%s, %s",'\*'\*8, 'single thread')

for i in nfuncs:

logger.debug("%s, %s, %s, %s",'starting', funcs[i].\_\_name\_\_, 'at:', ctime())

logger.debug("%s result is %s",funcs[i].\_\_name\_\_, funcs[i](n))

logger.debug("%s, %s, %s", funcs[i].\_\_name\_\_, 'finished at:', ctime())

logger.debug("%s",'\n\*\*\* multiple threads')

threads = []

for i in nfuncs:

t = MyThread(funcs[i], funcs[i].\_\_name\_\_, n)

threads.append(t)

for i in nfuncs:

threads[i].start()

for i in nfuncs:

threads[i].join()

for i in nfuncs:

pass

logger.debug("%s",threads[i].getResult())

logger.debug('all done')

def writeQ(queue):

logger.debug('producing object for Q...')

queue.put('xxx', 1)

logger.debug('size now %s', queue.qsize())

def readQ(queue):

val = queue.get(1)

logger.debug('consumed object from Q... size now %s', queue.qsize())

def writer(\*kargs):

(queue, loops) = kargs

for i in range(loops):

writeQ(queue)

sleep(randint(1,3))

def reader(\*kargs):

(queue, loops) = kargs

for i in range(loops):

logger.debug("read queue is %s",readQ(queue))

sleep(randint(2,5))

funcs\_q = [writer, reader]

nfuncs\_q = range(len(funcs\_q))

def main\_queue():

nloops = randint(5,10)

q = Queue(32)

threads = []

for i in nfuncs\_q:

t = MyThread(funcs\_q[i], funcs\_q[i].\_\_name\_\_,q, nloops)

threads.append(t)

for i in nfuncs\_q:

threads[i].start()

for i in nfuncs\_q:

threads[i].join()

logger.debug ('all done')

# Global variable

queue\_size = 128

reqQueue = Queue(queue\_size)

rplQueue = Queue(queue\_size)

from socket import \*

def tsTserv(\*kargs):

#from socket import \*

#HOST='192.168.1.102'

HOST=''

PORT=21569

BUFSIZ=1024

ADDR=(HOST,PORT)

tcpSerSock=socket(AF\_INET,SOCK\_STREAM)

tcpSerSock.bind(ADDR)

tcpSerSock.listen(5)

while True:

print('waiting for connection...')

tcpCliSock,addr =tcpSerSock.accept()

print('...connected from:',addr)

sleep(0.2)

while True:

data =tcpCliSock.recv(BUFSIZ).decode()

logger.debug('date= %s',int(data))

if not data:

break

#tcpCliSock.send(('[%s] %s' %(ctime(),data)).encode())

tcpCliSock.send((">").encode())

reqQueue.put(int(data),block=0)

sleep(0.2)

tcpCliSock.close()

tcpSerSock.close()

def userRequest(\*kargs):

""" Request inputs, maybe from multi-I/O,

maybe created in multi-threads for each customer,

All requests are put into reqQueue

"""

threadlock = kargs[0][0]

rtn = False

sleep(0.2)

while(True):

# if threadlock.acquire(blocking=True):

with threadlock:

#with threadlock: can replace threadlock.acqure().... threadlock.release()

cmd=input("\nPlease input an int:\n")

if cmd is 'q':

logger.debug(" input command is 'q', will quit this program. Bye!")

rtn = True

else:

try:

cmd = int(cmd)

except ValueError as e:

# threadlock.release()

continue

logger.debug("the user request is %s", cmd)

reqQueue.put(cmd,block=0)

if rtn:

# threadlock.release()

return 0

# threadlock.release()

sleep(0.2)

pass

def requestProcessor(\*kargs):

""" Process requests in reqQueue one by one, as long as reqQueue is not empty.

maybe run in multi-threads if processions are not influenced by each other.

Save the process result to rltQueue.

"""

threadlock = kargs[0][0]

print(requestProcessor.\_\_doc\_\_)

rtn = False

while(True):

# if threadlock.acquire(blocking=True):

with threadlock:

if not reqQueue.empty():

act = reqQueue.get(block=0)

if act is 'q':

rlt = 'q'

rtn = True

else:

#if threadlock.acquire(blocking=True):

rlt = fib(int(act)) +1 #key arithmatic, don't want to be disturbed by threading

# threadlock.release()

logger.debug("the procession result for request %s is %s", act, rlt)

rplQueue.put(rlt, block=0)

reqQueue.task\_done() #finish this task from request Queue

if rtn:

# threadlock.release()

return 0

# threadlock.release()

sleep(0.02)

pass

def replyFunc(\*kargs):

""" Save the the result in rltQueue if it's not empty.

Output the result by sending through TCP or saving into database.

"""

threadlock = kargs[0][0]

print(replyFunc.\_\_doc\_\_)

rtn = False

while(True):

if threadlock.acquire(blocking=True):

if not rplQueue.empty():

rply=rplQueue.get()

if rply is 'q':

rtn = True

logger.debug("the output reply is %s",rply)

rplQueue.task\_done() #finish this task from reply Queue

if rtn:

threadlock.release()

return 0

threadlock.release()

sleep(0.02)

pass

if 0:

func\_list = [tsTserv, requestProcessor, replyFunc]

else:

func\_list = [userRequest, requestProcessor, replyFunc]

def action\_thread():

num = len(func\_list)

threads = []

myArg=[]

threadlock = threading.RLock() # threadlock.acquire() can be called by multiple times

#threadlock = threading.Lock() # threadlock.acquire() can't be called by multiple times

myArg.append(threadlock)

for i in range(num):

t=MyThread(func\_list[i], func\_list[i].\_\_name\_\_, myArg)

t.daemon=True #set all thread as daemon(quit all threads if main thread ends)

threads.append(t)

for i in range(num):

threads[i].start()

for i in range(num):

threads[i].join()

pass

#removehandle()

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

#main()

#main\_queue()

try:

action\_thread()

except KeyboardInterrupt as e:

pass

### socket Client例程

# !/usr/bin/env python

'''This is the test excecises for chapter 16 of CORE\_PYTHON\_PROGRAMMING

Network Programming

'''

import sys

import os

from datetime import datetime

from time import ctime, sleep

import threading

from random import randint

from queue import Queue

import logging

from socket import \*

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s: %(message)s')

handler = logging.FileHandler('trace.log',mode='a')

handler.setLevel(logging.DEBUG)

handler.setFormatter(formatter)

# create a logging format

formatterconsole = logging.Formatter('%(message)s')

handlerconsole = logging.StreamHandler()

handlerconsole.setFormatter(formatterconsole)

handlerconsole.setLevel(logging.DEBUG)

if logger.hasHandlers() is False:

logger.addHandler(handler)

logger.addHandler(handlerconsole)

logger.info("HasHandlers() is False, add two new handlers")

else:

logger.info("HasHandlers() is True, previous handlers existed, don't add new ones")

# Global variable definition here

debug =True

HOST = '10.240.17.60'

PORT = 21567

ADDR = (HOST, PORT)

BUFSIZ = 1024

def tsTclinet():

#HOST = 'localhost' # or 'localhost'

tcpCliSock = socket(AF\_INET,SOCK\_STREAM)

tcpCliSock.connect(ADDR)

while True:

data = input('> ')

#print('data=',data);

if not data:

break

tcpCliSock.send(data.encode())

data = tcpCliSock.recv(BUFSIZ).decode()

if not data:

break

print(data)

tcpCliSock.close()

def tsUclinet():

udpCliSock = socket(AF\_INET,SOCK\_DGRAM)

while True:

data = input('> ')

#print('data=',data);

if not data:

break

udpCliSock.sendto(data.encode(), ADDR)

data, ADDR2 = udpCliSock.recvfrom(BUFSIZ) #.decode()

if not data:

break

print(data)

udpCliSock.close()

def tsTclientt():

while True:

tcpCliSock = socket(AF\_INET, SOCK\_STREAM)

tcpCliSock.connect(ADDR)

data = input('>')

if not data:

break

tcpCliSock.send(('%s\r\n' % data).encode('utf-8'))

data = tcpCliSock.recv(BUFSIZ)

if not data:

break

print(data.strip().decode('utf-8'))

tcpCliSock.close()

# Class definition here

class **MyThread** (threading.Thread):

def \_\_init\_\_(self, func, name='', \*kargs):

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

self.name = name

self.func = func

self.args = kargs

logger.debug('kargs=%s',kargs)

def getResult(self):

pass

return self.res

def run(self):

logger.debug ('starting %s at %s', self.name, ctime())

#self.res = apply(self.func, self.args)

self.res = self.func(\*self.args)

logger.debug("%s %s %s", self.name, 'finished at:', ctime())

from twisted.internet import protocol, reactor

class **TSClntProtocol**(protocol.Protocol):

def sendData(self):

data = input("> ")

if data:

logger.info( '...sending %s...', data)

#print(data)

self.transport.write(data.encode())

else:

self.transport.loseConnection()

def connectionMade(self):

self.sendData()

def dataReceived(self,data):

logger.debug("%s", data.decode())

self.sendData()

class **TSClntFactory**(protocol.ClientFactory):

protocol = TSClntProtocol

clientConnectionLost = clientConnectionFailed = lambda self, connector, reason: reactor.stop()

def twistedClient():

reactor.connectTCP(HOST, PORT, TSClntFactory())

reactor.run()

# function definition here

# test function here

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

#main()

#main\_queue()

#tsTclinet()

#tsUclinet()

# tsTclientt()

twistedClient()

### socket Server例程

# !/usr/bin/env python

'''This is the test excecises for chapter 16 of CORE\_PYTHON\_PROGRAMMING

Network Programming

'''

import sys

import os

from datetime import datetime

from time import ctime, sleep

import threading

from random import randint

from queue import Queue

import logging

from socket import \*

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s: %(message)s')

handler = logging.FileHandler('trace.log',mode='a')

handler.setLevel(logging.DEBUG)

handler.setFormatter(formatter)

# create a logging format

formatterconsole = logging.Formatter('%(message)s')

handlerconsole = logging.StreamHandler()

handlerconsole.setFormatter(formatterconsole)

handlerconsole.setLevel(logging.DEBUG)

if logger.hasHandlers() is False:

logger.addHandler(handler)

logger.addHandler(handlerconsole)

logger.info("HasHandlers() is False, add two new handlers")

else:

logger.info("HasHandlers() is True, previous handlers existed, don't add new ones")

# Global variable definition here

debug =True

HOST = ''

PORT = 21567

ADDR = (HOST, PORT)

BUFSIZ=1024

def tsTserv():

#from socket import \*

tcpSerSock=socket(AF\_INET,SOCK\_STREAM) #创服务器套接字

tcpSerSock.bind(ADDR) #套接字与地址绑定

tcpSerSock.listen(5) #监听连接,传入连接请求的最大数

while True:

print('waiting for connection...')

tcpCliSock,addr =tcpSerSock.accept()

print('...connected from:',addr)

while True:

data =tcpCliSock.recv(BUFSIZ).decode()

print('date=',data)

if not data:

break

tcpCliSock.send(('[%s] %s' %(ctime(),data)).encode())

tcpCliSock.close()

tcpSerSock.close()

def tsUserv():

udpSerSock = socket(AF\_INET,SOCK\_DGRAM)

udpSerSock.bind(ADDR)

while True:

print('waiting for message...')

data,addr=udpSerSock.recvfrom(BUFSIZ)

data=data.decode()

udpSerSock.sendto(('[%s] %s'%(ctime(),data)).encode(),addr)

print('...received from and returned to:',addr)

udpSerSock.close()

from socketserver import (TCPServer as TCP, StreamRequestHandler as SRH)

from socketserver import BaseServer as BSC

from time import ctime

# heritage SRH father class

class **MyRequestHandler**(SRH):

# overide handle() method

def handle(self):

# StreamRequestHandler类把输入和输出的套接字看做类似文件的对象

# readline(): read client info，write(): write data to client

#print('---Connected from: ' ,self.client\_address, self.wfile.write(('[%s]%s'%(ctime(), self.rfile.readline().decode('utf-8'))).encode('utf-8')))

logger.info('---Connected from: %s' ,self.client\_address)

data= self.rfile.readline().decode("utf-8")

if data.strip() is not '':

logger.debug('data received is %s', data)

result = data

self.wfile.write(('[%s]%s'%(ctime(), result)).encode('utf-8'))

def tsTservSS():

# 使用给定的地址和请求处理类创建TCP服务器

tcpServ = TCP(ADDR, MyRequestHandler)

logger.info('Waiting for connection...')

# 无限的等待并服务于客户端

tcpServ.serve\_forever()

from twisted.internet import protocol, reactor

from time import ctime

# Class definition here

class **TSServProtocol**(protocol.Protocol):

def connectionMade(self):

clnt = self.clnt = self.transport.getPeer().host

logger.info("...connected from: %s", clnt)

def dataReceived(self, data):

self.transport.write(("[%s] %s"%(ctime(), data.decode())).encode())

def twistedServer():

factory = protocol.Factory()

factory.protocol = TSServProtocol

logger.debug("waiting for connection...")

reactor.listenTCP(PORT, factory)

reactor.run()

# function definition here

# test function here

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

#main()

#main\_queue()

#tsTserv()

#tsUserv()

#tsTservSS()

twistedServer()

### logger例程

import logging

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s: %(message)s')

handler = logging.FileHandler('trace.log',mode='a')

handler.setLevel(logging.DEBUG)

handler.setFormatter(formatter)

# create a logging format

formatterconsole = logging.Formatter('%(message)s')

handlerconsole = logging.StreamHandler()

handlerconsole.setFormatter(formatterconsole)

handlerconsole.setLevel(logging.DEBUG)

if logger.hasHandlers() is False:

logger.addHandler(handler)

logger.addHandler(handlerconsole)

logger.info("HasHandlers() is False, add two new handlers")

else:

logger.info("HasHandlers() is True, previous handlers existed, don't add new ones")

## SQL知识教程

<http://www.w3school.com.cn/sql/index.asp>

**SQL 概要**

本教程已经向您讲解了用来访问和处理数据库系统的标准计算机语言。我们已经学习了如何使用 SQL 在数据库中执行查询、获取数据、插入新的记录、删除记录以及更新记录。SQL 是一种与数据库程序协同工作的标准语言，这些数据库程序包括 MS Access、DB2、Informix、MS SQL Server、Oracle、MySQL、Sybase 等等。

### SQL语句讲解

**SELECT 语句用于从表中选取数据。结果被存储在一个结果表中（称为结果集）**

SELECT [DISTINCT] 列名称1, 列名称1 FROM 表名称

SELECT \* FROM 表名称

SELECT \* FROM表名称WHERE列名称='Beijing'

SELECT \* FROM Persons WHERE FirstName='Thomas' AND LastName='Carter'

SELECT TOP 50 PERCENT \* FROM Persons

SELECT \* FROM Person WHERE City LIKE 'N\_[!ar]%'

**# ORDER BY 语句用于对结果集进行排序, DESC=逆序，ASC=正序**

SELECT Company, OrderNumber FROM Orders ORDER BY Company

SELECT Company, OrderNumber FROM Orders ORDER BY Company [DESC], OrderNumber [ASC]

#**INSERT INTO 语句用于向表格中插入新的行**

INSERT INTO table\_name [(列1, 列2,...)] VALUES (值1, 值2,....)

**# Update 语句用于修改表中的数据**

UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值

UPDATE Person SET Address = 'Zhongshan 23', City = 'Nanjing' WHERE LastName = 'Wilson'

**# DELETE 语句用于删除表中的行**

DELETE FROM Person WHERE LastName = 'Wilson'

DELETE \* FROM table\_name

**IN 操作符允许我们在 WHERE 子句中规定多个值**

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (value1,value2,...)

SELECT \* FROM Persons WHERE LastName IN ('Adams','Carter')

**操作符 BETWEEN ... AND 会选取介于两个值之间的数据范围。这些值可以是数值、文本或者日期**

SELECT column\_name(s)

FROM table\_name

WHERE column\_name

BETWEEN value1 AND value2

**有时为了得到完整的结果，我们需要从两个或更多的表中获取结果。我们就需要执行 join**。

数据库中的表可通过键将彼此联系起来。主键（Primary Key）是一个列，在这个列中的每一行的值都是唯一的。在表中，每个主键的值都是唯一的。这样做的目的是在不重复每个表中的所有数据的情况下，把表间的数据交叉捆绑在一起

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons, Orders

WHERE Persons.Id\_P = Orders.Id\_P

**除了上面的方法，我们也可以使用关键词 JOIN 来从两个表中获取数据**

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

**INNER** **JOIN** Orders

ON Persons.Id\_P = Orders.Id\_P

ORDER BY Persons.LastName

下面列出了您可以使用的 JOIN 类型，以及它们之间的差异。

JOIN: 如果表中有至少一个匹配，则返回行

LEFT JOIN: 即使右表中没有匹配，也从左表返回所有的行

RIGHT JOIN: 即使左表中没有匹配，也从右表返回所有的行

FULL JOIN: 只要其中一个表中存在匹配，就返回行

**在表中存在至少一个匹配时，INNER JOIN 关键字返回行**

SELECT column\_name(s)

FROM table\_name1

*INNER JOIN* table\_name2

*ON* table\_name1.column\_name=table\_name2.column\_name

注释：INNER JOIN 与 JOIN 是相同的

**UNION 操作符用于合并两个或多个 SELECT 语句的结果集。**

请注意，UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同

SELECT column\_name(s) FROM table\_name1

UNION

SELECT column\_name(s) FROM table\_name2

注释：这个命令无法列出在中国和美国的所有雇员。在上面的例子中，我们有两个名字相同的雇员，他们当中只有一个人被列出来了。UNION 命令只会选取不同的值

**UNION ALL** 命令和 UNION 命令几乎是等效的，不过 UNION ALL 命令会列出所有的值。

**SELECT INTO 语句从一个表中选取数据，然后把数据插入另一个表中。**

SELECT INTO 语句常用于创建表的备份复件或者用于对记录进行存档

SELECT \* INTO new\_table\_name [IN externaldatabase]

FROM old\_tablename

SELECT column\_name(s)

INTO new\_table\_name [IN externaldatabase]

FROM old\_tablename

SELECT LastName,Firstname

INTO Persons\_backup

FROM Persons

WHERE City='Beijing'

下面的例子会创建一个名为 "Persons\_Order\_Backup" 的新表，其中包含了从 Persons 和 Orders 两个表中取得的信息

SELECT Persons.LastName,Orders.OrderNo

INTO Persons\_Order\_Backup

FROM Persons

INNER JOIN Orders

ON Persons.Id\_P=Orders.Id\_P

**CREATE DATABASE 用于创建数据库**

CREATE DATABASE database\_name

CREATE DATABASE my\_dbf

**CREATE TABLE 语句用于创建数据库中的表**

CREATE TABLE IF NOT EXISTS表名称

(

列名称1 数据类型,

列名称2 数据类型,

列名称3 数据类型,

....

)

数据类型（data\_type）规定了列可容纳何种数据类型。下面的表格包含了SQL中最常用的数据类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| * integer(size) * int(size) * smallint(size) * tinyint(size) | 仅容纳整数。在括号内规定数字的最大位数。 |
| * decimal(size,d) * numeric(size,d) | 容纳带有小数的数字。  "size" 规定数字的最大位数。"d" 规定小数点右侧的最大位数。 |
| char(size) | 容纳固定长度的字符串（可容纳字母、数字以及特殊字符）。  在括号中规定字符串的长度。 |
| varchar(size) | 容纳可变长度的字符串（可容纳字母、数字以及特殊的字符）。  在括号中规定字符串的最大长度。 |
| date(yyyymmdd) | 容纳日期。 |

本例演示如何创建名为 "Person" 的表。该表包含 5 个列，列名分别是："Id\_P"、"LastName"、"FirstName"、"Address" 以及 "City"：Id\_P 列的数据类型是 int，包含整数。其余 4 列的数据类型是 varchar，最大长度为 255 个字符

CREATE TABLE Persons

(

Id\_P int,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

**约束用于限制加入表的数据的类型**

可以在创建表时规定约束（通过 CREATE TABLE 语句），或者在表创建之后也可以（通过 ALTER TABLE 语句）。

我们将主要探讨以下几种约束：

NOT NULL

UNIQUE

PRIMARY KEY

FOREIGN KEY

CHECK

DEFAULT

**NOT NULL 约束强制列不接受 NULL 值**

下面的 SQL 语句强制 "Id\_P" 列和 "LastName" 列不接受 NULL 值：

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

**UNIQUE 约束唯一标识数据库表中的每条记录**。

UNIQUE 和 PRIMARY KEY 约束均为列或列集合提供了唯一性的保证。

PRIMARY KEY 拥有自动定义的 UNIQUE 约束。

请注意，每个表可以有多个 UNIQUE 约束，但是每个表只能有一个 PRIMARY KEY 约束

CREATE TABLE Persons

(

Id\_P int NOT NULL UNIQUE, #SQL Server / Oracle / MS Access

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

UNIQUE (Id\_P) #MySQL

)

**如果需要命名 UNIQUE 约束，以及为多个列定义 UNIQUE 约束，请使用下面的 SQL 语法**

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT uc\_PersonID UNIQUE (Id\_P,LastName)

)

**当表已被创建时，如需在 "Id\_P" 列创建 UNIQUE 约束，请使用下列 SQL**

ALTER TABLE Persons

ADD UNIQUE (Id\_P)

**如需命名 UNIQUE 约束，并定义多个列的 UNIQUE 约束，请使用下面的 SQL 语法**

ALTER TABLE Persons

ADD CONSTRAINT uc\_PersonID UNIQUE (Id\_P,LastName)

**如需撤销 UNIQUE 约束，请使用下面的 SQL**

ALTER TABLE Persons

DROP INDEX uc\_PersonID #MySQL

DROP CONSTRAINT uc\_PersonID #SQL Server / Oracle / MS Access

**RIMARY KEY 约束唯一标识数据库表中的每条记录。**

主键必须包含唯一的值。

主键列不能包含 NULL 值。

每个表都应该有一个主键，并且每个表只能有一个主键

CREATE TABLE Persons

(

Id\_P int NOT NULL PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

PRIMARY KEY (Id\_P) #MySQL

)

其他与UNIQUE类似

**一个表中的 FOREIGN KEY 指向另一个表中的 PRIMARY KEY**

CREATE TABLE Orders

(

Id\_O int NOT NULL PRIMARY KEY,

OrderNo int NOT NULL,

Id\_P int FOREIGN KEY REFERENCES Persons(Id\_P)

FOREIGN KEY (Id\_P) REFERENCES Persons(Id\_P) #MySQL

)

CREATE TABLE Orders

(

Id\_O int NOT NULL,

OrderNo int NOT NULL,

Id\_P int,

PRIMARY KEY (Id\_O),

CONSTRAINT fk\_PerOrders FOREIGN KEY (Id\_P)

REFERENCES Persons(Id\_P)

)

**如果在 "Orders" 表已存在的情况下为 "Id\_P" 列创建 FOREIGN KEY 约束，请使用下面的 SQL**

ALTER TABLE Orders

ADD FOREIGN KEY (Id\_P)

REFERENCES Persons(Id\_P)

**撤销 FOREIGN KEY 约束**

ALTER TABLE Orders

DROP CONSTRAINT fk\_PerOrders

**CHECK 约束用于限制列中的值的范围。**

如果对单个列定义 CHECK 约束，那么该列只允许特定的值。

如果对一个表定义 CHECK 约束，那么此约束会在特定的列中对值进行限制

CREATE TABLE Persons

(

Id\_P int NOT NULL CHECK (Id\_P>0),

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

如果需要命名 CHECK 约束，以及为多个列定义 CHECK 约束

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT chk\_Person CHECK (Id\_P>0 AND City='Sandnes')

)

**DEFAULT 约束用于向列中插入默认值。**

如果没有规定其他的值，那么会将默认值添加到所有的新记录

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255) DEFAULT 'Sandnes'

)

CREATE TABLE Orders

(

Id\_O int NOT NULL,

OrderNo int NOT NULL,

Id\_P int,

OrderDate date DEFAULT GETDATE()

)

**撤销 DEFAULT 约束**

ALTER TABLE Persons

ALTER COLUMN City DROP DEFAULT

**CREATE INDEX 语句用于在表中创建索引。在不读取整个表的情况下，索引使数据库应用程序可以更快地查找数据**

您可以在表中创建索引，以便更加快速高效地查询数据。用户无法看到索引，它们只能被用来加速搜索/查询。

**注释：**更新一个包含索引的表需要比更新一个没有索引的表更多的时间，这是由于索引本身也需要更新。因此，理想的做法是仅仅在常常被搜索的列（以及表）上面创建索引。

创建一个简单的索引。允许使用重复的值：

CREATE INDEX index\_name

ON table\_name (column\_name)

在表上创建一个唯一的索引。唯一的索引意味着两个行不能拥有相同的索引值。

CREATE UNIQUE INDEX index\_name

ON table\_name (column\_name)

CREATE INDEX 实例

本例会创建一个简单的索引，名为 "PersonIndex"，在 Person 表的 LastName 列：

CREATE INDEX PersonIndex

ON Person (LastName)

如果您希望以降序索引某个列中的值，您可以在列名称之后添加保留字 DESC：

CREATE INDEX PersonIndex

ON Person (LastName DESC)

假如您希望索引不止一个列，您可以在括号中列出这些列的名称，用逗号隔开：

CREATE INDEX PersonIndex

ON Person (LastName, FirstName)

**DROP 语句，可以轻松地删除索引、表和数据库**

SQL DROP INDEX 语句

我们可以使用 DROP INDEX 命令删除表格中的索引。

用于 Microsoft SQLJet (以及 Microsoft Access) 的语法:

DROP INDEX index\_name ON table\_name

用于 MS SQL Server 的语法:

DROP INDEX table\_name.index\_name

用于 IBM DB2 和 Oracle 语法:

DROP INDEX index\_name

用于 MySQL 的语法:

ALTER TABLE table\_name DROP INDEX index\_name

SQL DROP TABLE 语句

DROP TABLE 语句用于删除表（表的结构、属性以及索引也会被删除）：

DROP TABLE 表名称

SQL DROP DATABASE 语句

DROP DATABASE 语句用于删除数据库：

DROP DATABASE 数据库名称

SQL TRUNCATE TABLE 语句

如果我们仅仅需要除去表内的数据，但并不删除表本身，那么我们该如何做呢？

请使用 TRUNCATE TABLE 命令（仅仅删除表格中的数据）：

TRUNCATE TABLE 表名称

**ALTER TABLE 语句用于在已有的表中添加、修改或删除列**

SQL ALTER TABLE 语法

如需在表中添加列，请使用下列语法:

ALTER TABLE table\_name

ADD column\_name datatype

要删除表中的列，请使用下列语法：

ALTER TABLE table\_name

DROP COLUMN column\_name

注释：某些数据库系统不允许这种在数据库表中删除列的方式 (DROP COLUMN column\_name)。

要改变表中列的数据类型，请使用下列语法：

ALTER TABLE table\_name

ALTER COLUMN column\_name datatype

原始的表 (用在例子中的)：

Persons 表:

Id LastName FirstName Address City

1 Adams John Oxford Street London

2 Bush George Fifth Avenue New York

3 Carter Thomas Changan Street Beijing

SQL ALTER TABLE 实例

现在，我们希望在表 "Persons" 中添加一个名为 "Birthday" 的新列。

我们使用下列 SQL 语句：

ALTER TABLE Persons

ADD Birthday date

请注意，新列 "Birthday" 的类型是 date，可以存放日期。数据类型规定列中可以存放的数据的类型。

新的 "Persons" 表类似这样：

Id LastName FirstName Address City Birthday

1 Adams John Oxford Street London

2 Bush George Fifth Avenue New York

3 Carter Thomas Changan Street Beijing

改变数据类型实例

现在我们希望改变 "Persons" 表中 "Birthday" 列的数据类型。

我们使用下列 SQL 语句：

ALTER TABLE Persons

ALTER COLUMN Birthday year

请注意，"Birthday" 列的数据类型是 year，可以存放 2 位或 4 位格式的年份。

DROP COLUMN 实例

接下来，我们删除 "Person" 表中的 "Birthday" 列：

ALTER TABLE Person

DROP COLUMN Birthday

Persons 表会成为这样:

Id LastName FirstName Address City

1 Adams John Oxford Street London

2 Bush George Fifth Avenue New York

3 Carter Thomas Changan Street Beijing

**AUTO INCREMENT 字段**

我们通常希望在每次插入新记录时，自动地创建主键字段的值。

我们可以在表中创建一个 auto-increment 字段

CREATE TABLE Persons

(

P\_Id int NOT NULL AUTO\_INCREMENT,

#P\_Id int PRIMARY KEY IDENTITY(20,10), #MS SQL 使用 IDENTITY 关键字来执行 auto-increment 任务

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY (P\_Id)

)

**什么是视图？在 SQL 中，视图是基于 SQL 语句的结果集的可视化的表。**

视图包含行和列，就像一个真实的表。视图中的字段就是来自一个或多个数据库中的真实的表中的字段。我们可以向视图添加 SQL 函数、WHERE 以及 JOIN 语句，我们也可以提交数据，就像这些来自于某个单一的表

SQL CREATE VIEW 语法

CREATE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

EXAMPLE:

CREATE VIEW [Current Product List] AS

SELECT ProductID,ProductName

FROM Products

WHERE Discontinued=No

我们可以查询上面这个视图：

SELECT \* FROM [Current Product List]

**更新视图**

CREATE OR REPLACE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

**MySQL Date 函数**

下面的表格列出了 MySQL 中最重要的内建日期函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [NOW()](http://www.w3school.com.cn/sql/func_now.asp) | 返回当前的日期和时间 |
| [CURDATE()](http://www.w3school.com.cn/sql/func_curdate.asp) | 返回当前的日期 |
| [CURTIME()](http://www.w3school.com.cn/sql/func_curtime.asp) | 返回当前的时间 |
| [DATE()](http://www.w3school.com.cn/sql/func_date.asp) | 提取日期或日期/时间表达式的日期部分 |
| [EXTRACT()](http://www.w3school.com.cn/sql/func_extract.asp) | 返回日期/时间按的单独部分 |
| [DATE\_ADD()](http://www.w3school.com.cn/sql/func_date_add.asp) | 给日期添加指定的时间间隔 |
| [DATE\_SUB()](http://www.w3school.com.cn/sql/func_date_sub.asp) | 从日期减去指定的时间间隔 |
| [DATEDIFF()](http://www.w3school.com.cn/sql/func_datediff_mysql.asp) | 返回两个日期之间的天数 |
| [DATE\_FORMAT()](http://www.w3school.com.cn/sql/func_date_format.asp) | 用不同的格式显示日期/时间 |

**SQL Server Date 函数**

下面的表格列出了 SQL Server 中最重要的内建日期函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [GETDATE()](http://www.w3school.com.cn/sql/func_getdate.asp) | 返回当前日期和时间 |
| [DATEPART()](http://www.w3school.com.cn/sql/func_datepart.asp) | 返回日期/时间的单独部分 |
| [DATEADD()](http://www.w3school.com.cn/sql/func_dateadd.asp) | 在日期中添加或减去指定的时间间隔 |
| [DATEDIFF()](http://www.w3school.com.cn/sql/func_datediff.asp) | 返回两个日期之间的时间 |
| [CONVERT()](http://www.w3school.com.cn/sql/func_convert.asp) | 用不同的格式显示日期/时间 |

**SQL Date 数据类型**

MySQL 使用下列数据类型在数据库中存储日期或日期/时间值：

* DATE - 格式 YYYY-MM-DD
* DATETIME - 格式: YYYY-MM-DD HH:MM:SS
* TIMESTAMP - 格式: YYYY-MM-DD HH:MM:SS
* YEAR - 格式 YYYY 或 YY

SQL Server 使用下列数据类型在数据库中存储日期或日期/时间值：

* DATE - 格式 YYYY-MM-DD
* DATETIME - 格式: YYYY-MM-DD HH:MM:SS
* SMALLDATETIME - 格式: YYYY-MM-DD HH:MM:SS
* TIMESTAMP - 格式: 唯一的数字

SELECT \* FROM Orders WHERE OrderDate='2008-12-26'

**SQL IS NULL**

我们如何仅仅选取在 "Address" 列中带有 NULL 值的记录呢？

我们必须使用 IS NULL 操作符：

SELECT LastName,FirstName,Address FROM Persons

WHERE Address IS NULL

(WHERE Address IS NOT NULL)

**ISNULL()、NVL()、IFNULL() 和 COALESCE() 函数**

SELECT ProductName,UnitPrice\*(UnitsInStock+ISNULL(UnitsOnOrder,0))

FROM Products

**Microsoft Access、MySQL 以及 SQL Server 所使用的数据类型和范围。**

**Microsoft Access 数据类型**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| Text | 用于文本或文本与数字的组合。最多 255 个字符。 |  |
| Memo | Memo 用于更大数量的文本。最多存储 65,536 个字符。  注释：无法对 memo 字段进行排序。不过它们是可搜索的。 |  |
| Byte | 允许 0 到 255 的数字。 | 1 字节 |
| Integer | 允许介于 -32,768 到 32,767 之间的数字。 | 2 字节 |
| Long | 允许介于 -2,147,483,648 与 2,147,483,647 之间的全部数字 | 4 字节 |
| Single | 单精度浮点。处理大多数小数。 | 4 字节 |
| Double | 双精度浮点。处理大多数小数。 | 8 字节 |
| Currency | 用于货币。支持 15 位的元，外加 4 位小数。  提示：您可以选择使用哪个国家的货币。 | 8 字节 |
| AutoNumber | AutoNumber 字段自动为每条记录分配数字，通常从 1 开始。 | 4 字节 |
| Date/Time | 用于日期和时间 | 8 字节 |
| Yes/No | 逻辑字段，可以显示为 Yes/No、True/False 或 On/Off。  在代码中，使用常量 True 和 False （等价于 1 和 0）  注释：Yes/No 字段中不允许 Null 值 | 1 比特 |
| Ole Object | 可以存储图片、音频、视频或其他 BLOBs (Binary Large OBjects) | 最多 1GB |
| Hyperlink | 包含指向其他文件的链接，包括网页。 |  |
| Lookup Wizard | 允许你创建一个可从下列列表中进行选择的选项列表。 | 4 字节 |

**MySQL 数据类型**

在 MySQL 中，有三种主要的类型：文本、数字和日期/时间类型。

**Text 类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| CHAR(size) | 保存固定长度的字符串（可包含字母、数字以及特殊字符）。在括号中指定字符串的长度。最多 255 个字符。 |
| VARCHAR(size) | 保存可变长度的字符串（可包含字母、数字以及特殊字符）。在括号中指定字符串的最大长度。最多 255 个字符。  注释：如果值的长度大于 255，则被转换为 TEXT 类型。 |
| TINYTEXT | 存放最大长度为 255 个字符的字符串。 |
| TEXT | 存放最大长度为 65,535 个字符的字符串。 |
| BLOB | 用于 BLOBs (Binary Large OBjects)。存放最多 65,535 字节的数据。 |
| MEDIUMTEXT | 存放最大长度为 16,777,215 个字符的字符串。 |
| MEDIUMBLOB | 用于 BLOBs (Binary Large OBjects)。存放最多 16,777,215 字节的数据。 |
| LONGTEXT | 存放最大长度为 4,294,967,295 个字符的字符串。 |
| LONGBLOB | 用于 BLOBs (Binary Large OBjects)。存放最多 4,294,967,295 字节的数据。 |
| ENUM(x,y,z,etc.) | 允许你输入可能值的列表。可以在 ENUM 列表中列出最大 65535 个值。如果列表中不存在插入的值，则插入空值。  注释：这些值是按照你输入的顺序存储的。  可以按照此格式输入可能的值：ENUM('X','Y','Z') |
| SET | 与 ENUM 类似，SET 最多只能包含 64 个列表项，不过 SET 可存储一个以上的值。 |

**Number 类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| TINYINT(size) | -128 到 127 常规。0 到 255 无符号\*。在括号中规定最大位数。 |
| SMALLINT(size) | -32768 到 32767 常规。0 到 65535 无符号\*。在括号中规定最大位数。 |
| MEDIUMINT(size) | -8388608 到 8388607 普通。0 to 16777215 无符号\*。在括号中规定最大位数。 |
| INT(size) | -2147483648 到 2147483647 常规。0 到 4294967295 无符号\*。在括号中规定最大位数。 |
| BIGINT(size) | -9223372036854775808 到 9223372036854775807 常规。0 到 18446744073709551615 无符号\*。在括号中规定最大位数。 |
| FLOAT(size,d) | 带有浮动小数点的小数字。在括号中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DOUBLE(size,d) | 带有浮动小数点的大数字。在括号中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DECIMAL(size,d) | 作为字符串存储的 DOUBLE 类型，允许固定的小数点。 |

\* 这些整数类型拥有额外的选项 UNSIGNED。通常，整数可以是负数或正数。如果添加 UNSIGNED 属性，那么范围将从 0 开始，而不是某个负数。

**Date 类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| DATE() | 日期。格式：YYYY-MM-DD  注释：支持的范围是从 '1000-01-01' 到 '9999-12-31' |
| DATETIME() | \*日期和时间的组合。格式：YYYY-MM-DD HH:MM:SS  注释：支持的范围是从 '1000-01-01 00:00:00' 到 '9999-12-31 23:59:59' |
| TIMESTAMP() | \*时间戳。TIMESTAMP 值使用 Unix 纪元('1970-01-01 00:00:00' UTC) 至今的描述来存储。格式：YYYY-MM-DD HH:MM:SS  注释：支持的范围是从 '1970-01-01 00:00:01' UTC 到 '2038-01-09 03:14:07' UTC |
| TIME() | 时间。格式：HH:MM:SS 注释：支持的范围是从 '-838:59:59' 到 '838:59:59' |
| YEAR() | 2 位或 4 位格式的年。  注释：4 位格式所允许的值：1901 到 2155。2 位格式所允许的值：70 到 69，表示从 1970 到 2069。 |

\* 即便 DATETIME 和 TIMESTAMP 返回相同的格式，它们的工作方式很不同。在 INSERT 或 UPDATE 查询中，TIMESTAMP 自动把自身设置为当前的日期和时间。TIMESTAMP 也接受不同的格式，比如 YYYYMMDDHHMMSS、YYMMDDHHMMSS、YYYYMMDD 或 YYMMDD。

**SQL Server 数据类型**

**Character 字符串：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| char(n) | 固定长度的字符串。最多 8,000 个字符。 | n |
| varchar(n) | 可变长度的字符串。最多 8,000 个字符。 |  |
| varchar(max) | 可变长度的字符串。最多 1,073,741,824 个字符。 |  |
| text | 可变长度的字符串。最多 2GB 字符数据。 |  |

**Unicode 字符串：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| nchar(n) | 固定长度的 Unicode 数据。最多 4,000 个字符。 |  |
| nvarchar(n) | 可变长度的 Unicode 数据。最多 4,000 个字符。 |  |
| nvarchar(max) | 可变长度的 Unicode 数据。最多 536,870,912 个字符。 |  |
| ntext | 可变长度的 Unicode 数据。最多 2GB 字符数据。 |  |

**Binary 类型：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| bit | 允许 0、1 或 NULL |  |
| binary(n) | 固定长度的二进制数据。最多 8,000 字节。 |  |
| varbinary(n) | 可变长度的二进制数据。最多 8,000 字节。 |  |
| varbinary(max) | 可变长度的二进制数据。最多 2GB 字节。 |  |
| image | 可变长度的二进制数据。最多 2GB。 |  |

**Number 类型：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| tinyint | 允许从 0 到 255 的所有数字。 | 1 字节 |
| smallint | 允许从 -32,768 到 32,767 的所有数字。 | 2 字节 |
| int | 允许从 -2,147,483,648 到 2,147,483,647 的所有数字。 | 4 字节 |
| bigint | 允许介于 -9,223,372,036,854,775,808 和 9,223,372,036,854,775,807 之间的所有数字。 | 8 字节 |
| decimal(p,s) | 固定精度和比例的数字。允许从 -10^38 +1 到 10^38 -1 之间的数字。  p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。  s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | 5-17 字节 |
| numeric(p,s) | 固定精度和比例的数字。允许从 -10^38 +1 到 10^38 -1 之间的数字。  p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。  s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | 5-17 字节 |
| smallmoney | 介于 -214,748.3648 和 214,748.3647 之间的货币数据。 | 4 字节 |
| money | 介于 -922,337,203,685,477.5808 和 922,337,203,685,477.5807 之间的货币数据。 | 8 字节 |
| float(n) | 从 -1.79E + 308 到 1.79E + 308 的浮动精度数字数据。 参数 n 指示该字段保存 4 字节还是 8 字节。float(24) 保存 4 字节，而 float(53) 保存 8 字节。n 的默认值是 53。 | 4 或 8 字节 |
| real | 从 -3.40E + 38 到 3.40E + 38 的浮动精度数字数据。 | 4 字节 |

**Date 类型：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| datetime | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 3.33 毫秒。 | 8 bytes |
| datetime2 | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 100 纳秒。 | 6-8 bytes |
| smalldatetime | 从 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度为 1 分钟。 | 4 bytes |
| date | 仅存储日期。从 0001 年 1 月 1 日 到 9999 年 12 月 31 日。 | 3 bytes |
| time | 仅存储时间。精度为 100 纳秒。 | 3-5 bytes |
| datetimeoffset | 与 datetime2 相同，外加时区偏移。 | 8-10 bytes |
| timestamp | 存储唯一的数字，每当创建或修改某行时，该数字会更新。timestamp 基于内部时钟，不对应真实时间。每个表只能有一个 timestamp 变量。 |  |

**其他数据类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| sql\_variant | 存储最多 8,000 字节不同数据类型的数据，除了 text、ntext 以及 timestamp。 |
| uniqueidentifier | 存储全局标识符 (GUID)。 |
| xml | 存储 XML 格式化数据。最多 2GB。 |
| cursor | 存储对用于数据库操作的指针的引用。 |
| table | 存储结果集，供稍后处理。 |

**SQL 服务器 - RDBMS**

*DBMS - 数据库管理系统（Database Management System）*

数据库管理系统是一种可以访问数据库中数据的计算机程序。DBMS 使我们有能力在数据库中提取、修改或者存贮信息。不同的 DBMS 提供不同的函数供查询、提交以及修改数据。

*RDBMS - 关系数据库管理系统（Relational Database Management System）*

关系数据库管理系统 (RDBMS) 也是一种数据库管理系统，其数据库是根据数据间的关系来组织和访问数据的。

20 世纪 70 年代初，IBM 公司发明了 RDBMS。RDBMS 是 SQL 的基础，也是所有现代数据库系统诸如 Oracle、SQL Server、IBM DB2、Sybase、MySQL 以及 Microsoft Access 的基础。

**SQL AVG() 实例**

SELECT AVG(OrderPrice) AS OrderAverage FROM Orders

SELECT Customer FROM Orders

WHERE OrderPrice>(SELECT AVG(OrderPrice) FROM Orders)

**COUNT() 函数返回匹配指定条件的行数**

SELECT COUNT(column\_name) FROM table\_name

SELECT COUNT(\*) FROM table\_name

SELECT COUNT(DISTINCT column\_name) FROM table\_name #返回指定列的不同值的数目

SELECT COUNT(Customer) AS CustomerNilsen FROM Orders WHERE Customer='Carter'

**FIRST() 函数返回指定的字段中第一个记录的值**

SELECT FIRST(column\_name) FROM table\_name

SELECT FIRST(OrderPrice) [AS FirstOrderPrice] FROM Orders

**LAST() 函数返回指定的字段中最后一个记录的值**

SELECT LAST(OrderPrice) [AS LastOrderPrice] FROM Orders

**MAX 函数返回一列中的最大值。NULL 值不包括在计算中**

SELECT MAX(OrderPrice) [AS LargestOrderPrice] FROM Orders

**MIN 函数返回一列中的最小值。NULL 值不包括在计算中**

SELECT MIN(OrderPrice) AS SmallestOrderPrice FROM Orders

**SUM 函数返回数值列的总数（总额）**

SELECT SUM(OrderPrice) AS OrderTotal FROM Orders

**GROUP BY 语句用于结合合计函数，根据一个或多个列对结果集进行分组**

SELECT column\_name, aggregate\_function(column\_name)

FROM table\_name

WHERE column\_name operator value

GROUP BY column\_name

example1:

SELECT Customer,SUM(OrderPrice) FROM Orders

GROUP BY Customer

example2:

SELECT Customer,OrderDate,SUM(OrderPrice) FROM Orders

GROUP BY Customer,OrderDate

**增加 HAVING 子句原因是，WHERE 关键字无法与合计函数一起使用**

SELECT column\_name, aggregate\_function(column\_name)

FROM table\_name

WHERE column\_name operator value

GROUP BY column\_name

HAVING aggregate\_function(column\_name) operator value

examle:

SELECT Customer,SUM(OrderPrice) FROM Orders

WHERE Customer='Bush' OR Customer='Adams'

GROUP BY Customer

HAVING SUM(OrderPrice)>1500

**UCASE 函数把字段的值转换为大写**

SELECT UCASE(column\_name) FROM table\_name

example:

SELECT UCASE(LastName) as LastName（大写）,FirstName(不变) FROM Persons

**LCASE 函数把字段的值转换为小写**

SELECT LCASE(column\_name) FROM table\_name

example:

SELECT LCASE(LastName) as LastName(小写),FirstName(不变) FROM Persons

**MID() 函数用于从文本字段中提取字符**

SELECT MID(column\_name,start[,length]) FROM table\_name

参数描述: column\_name必需。要提取字符的字段。start必需。规定开始位置（起始值是 1）。length可选。要返回的字符数。如果省略，则 MID() 函数返回剩余文本。

SELECT MID(City,1,3) as SmallCity FROM Persons

**LEN 函数返回文本字段中值的长度**

SELECT LEN(column\_name) FROM table\_name

example:

SELECT LEN(City) as LengthOfCity FROM Persons

**ROUND 函数用于把数值字段舍入为指定的小数位数**

SELECT ROUND(column\_name,decimals) FROM table\_name

SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice FROM Products

**NOW 函数返回当前的日期和时间**

SELECT NOW() FROM table\_name

SELECT ProductName, UnitPrice, Now() as PerDate FROM Products

**FORMAT 函数用于对字段的显示进行格式化**

SELECT FORMAT(column\_name,format) FROM table\_name

SELECT ProductName, UnitPrice, FORMAT(Now(),'YYYY-MM-DD') as PerDate

FROM Products

### SQL快速参考

**SQL 语句**

|  |  |
| --- | --- |
| **语句** | **语法** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE (add column) | ALTER TABLE table\_name  ADD column\_name datatype |
| ALTER TABLE (drop column) | ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias for column) | SELECT column\_name AS column\_alias FROM table\_name |
| AS (alias for table) | SELECT column\_name FROM table\_name  AS table\_alias |
| BETWEEN | SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE INDEX | CREATE INDEX index\_name ON table\_name (column\_name) |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, ....... ) |
| CREATE UNIQUE INDEX | CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| DELETE FROM | DELETE FROM table\_name  (**Note:**Deletes the entire table!!)  *or*  DELETE FROM table\_name WHERE condition |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name |
| DROP TABLE | DROP TABLE table\_name |
| GROUP BY | SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 |
| HAVING | SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 HAVING SUM(column\_name2) condition value |
| IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2,....)  *or*  INSERT INTO table\_name (column\_name1, column\_name2,...) VALUES (value1, value2,....) |
| LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern |
| ORDER BY | SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC] |
| SELECT | SELECT column\_name(s) FROM table\_name |
| SELECT \* | SELECT \* FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s) FROM table\_name |
| SELECT INTO (used to create backup copies of tables) | SELECT \* INTO new\_table\_name FROM original\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name FROM original\_table\_name |
| TRUNCATE TABLE (deletes only the data inside the table) | TRUNCATE TABLE table\_name |
| UPDATE | UPDATE table\_name SET column\_name=new\_value [, column\_name=new\_value] WHERE column\_name=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE condition |

### python读取sqlite3数据库的表格名称及列名称

def sqlite\_read():

import sqlite3

import pprint

"""python读取sqlite数据库文件

"""

conn = sqlite3.connect('example.db') # 链接数据库

cur = conn.cursor() # 创建游标cur来执行SQL语句

# 获取表名

cur.execute("SELECT name FROM sqlite\_master WHERE type='table'")

Tables = cur.fetchall() # Tables 为元组列表

print (Tables)

tbl\_name = Tables[0][0] # 获取第一个表名

# 获取表的列名

cur.execute("SELECT \* FROM {}".format(tbl\_name))

col\_name\_list = [tuple[0] for tuple in cur.description]

pprint.pprint(col\_name\_list)

# 获取表结构的所有信息

cur.execute("PRAGMA table\_info({})".format(tbl\_name))

pprint.pprint(cur.fetchall())

### 数据库-E-R图

E-R图也称实体-联系图**(Entity Relationship Diagram)**，提供了表示实体类型、属性和联系的方法，用来描述现实世界的概念模型。

它是描述现实世界概念结构模型的有效方法。是表示概念模型的一种方式，用**矩形表示实体**型，矩形框内写明实体名；用**椭圆表示实体的属性**，并用无向边将其与相应的实体型连接起来；用**菱形表示实体型之间的联系**，在菱形框内写明联系名，并用无向边分别与有关实体型连接起来，同时在**无向边旁标上联系的类型（1:1,1:n或m:n**）(???)。

总结来说，E-R图的四个组成的部分

矩形框: 表示实体，在矩形框中写上实体的名字

椭圆形框: 表示实体或联系的属性

菱形框: 表示联系，在框中记入联系名

连线:实体与属性之间；实体与联系之间；联系与属性之间用直线相连，（**对于一对一联系，要在两个实体连线方向各写1； 对于一对多联系，要在一的一方写1，多的一方写N；对于多对多关系，则要在两个实体连线方向各写N,M。**)???。

下面简单的介绍下基本的概念

**实体:** 客观存在并且可以相互区分的事物为实体。可以是具体的人，事，物或者抽象的概念

**属性:** 实体所具有的某一个特性就称为属性，一个实体可以由若干个属性来刻画。

**码:** 唯一标识实体的属性集

**域:** 属性的取值范围称为该属性的域

**实体型：**用实体名及其属性名集合来抽象和刻画同类实体称为实体型

**实体集：**同一类型实体的集合称为实体集

**联系：**现实世界中事物内部以及事物之间的联系在信息世界中反映为实体内部的联系和实体之间的联系。包括实体内部的联系及实体之间的联系

其中码分为超码，候选码，主码，外码

**超码：**一个或多个属性的集合，这些属性的组合可以使我们在一个实体集中唯一的标识一个实体

**候选码：**任意超码的真子集不能包括超码，则称其为候选码；超码包括候选码

**主码:** 其实主码一般指的就是主关键字。主关键字是表中的一个或多个字段，*它的值用于唯一地标识表中的某一条记录*，主关键字又可以称为主键。 *主键可以由一个字段，也可以由多个字段组成*，分别称为单字段主键或多字段主键。又称主码。

**外码:** 外码别称外键，具体来说就是比如说这个外码是相对于关系R来说的，但是如果相对于关系S来说的话是关系R中的主码，外码就是这个意思。

举个最经典的例子

**E-R图表示某个工厂物资管理的概念模型**

实体包括：

仓库： 仓库号、面积、电话号码；

零件 ：零件号、名称、规格、单价、描述；

供应商：供应商号、姓名、地址、电话号码、帐号；

项目：项目号、预算、开工日期；

职工：职工号、姓名、年龄、职称；

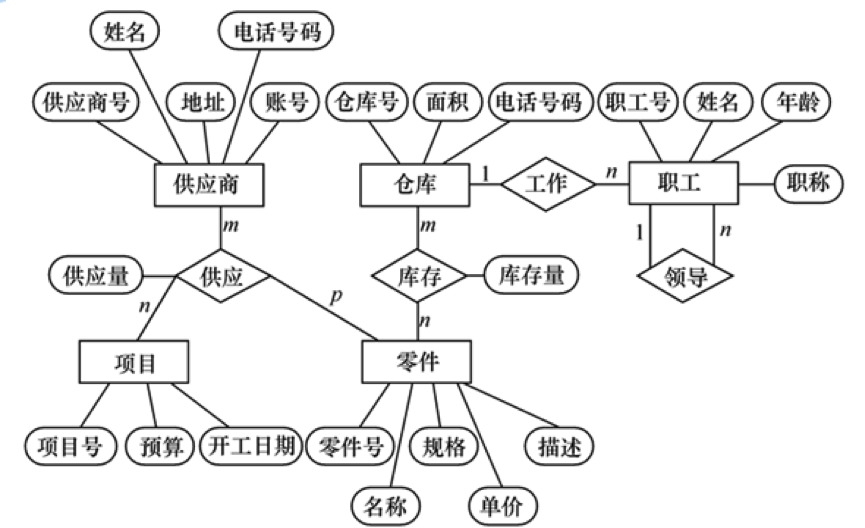
**实体之间的联系**

仓库------零件  
1、一个仓库可以存放多种零件，一种零件可以存放在多个仓库中。仓库和零件是多对多的关系。

2、用库存量来表示某种零件在某个仓库中的数量

仓库------职工  
1、一个仓库拥有多个职工，一个职工只能在一个仓库中工作，仓库和职工是一对多的关系

2、职工之间有领导和被领导的关系，因为管仓库肯定有个头，所以有领导和被领导的关系。

E-R图如下所示  


### a

a

### a

a

### a

a

### a

a

### a

a

### 其他Python笔记(末尾)

## WXPYTHON窗口编程说明

### 开始

import wx

app = wx.App(False) # Create a new app, don't redirect stdout/stderr to a window.

wx.Frame类是一个顶层窗口。它的用法是wx.Frame(Parent, Id, Title)。对于大对数的构造函数来说，都有这种通用的形式（一个父窗口名，后面紧随着它的Id）。在这个例子当中，我们使用None ，来说明没用父窗口，并且使用ID\_ANY，来拥有一个wxWidgets分配给我们的ID 号

frame = wx.Frame(None, wx.ID\_ANY, "Hello World") # A Frame is a top-level window.

frame.Show(True) #我们使一个窗口可见，通过这个函数。如果将参数改为False，你会发现程序真的在运行，但是我们看不到

app.MainLoop() #最后，我们开始应用程序的MainLoop函数，它用来处理各种事件

### 一个简单的文本编辑器

import wx

class MyFrame(wx.Frame):

""" We simply derive a new class of Frame. """

# 派生了wx.Frame类，并且重写了它的\_\_init\_\_方法。在这个方法中，我们声明了一个新的wx.TextCtrl实例，它是一个简单的文本编辑控件。注意：因为MyFrame类运行了self.Show()在它的\_\_init\_\_方法中，所以我们不需要再显式地调用frame.Show()

def \_\_init\_\_(self, parent, title):

wx.Frame.\_\_init\_\_(self, parent, title=title, size=(200,100))

self.control = wx.TextCtrl(self, style=wx.TE\_MULTILINE)

self.Show(True)

app = wx.App(False)

frame = MyFrame(None, 'Small editor')

app.MainLoop()

### 添加一个菜单栏

#注意那个wx.ID\_ABOUT和wx.ID\_EXIT。它们是wxWidgets提供的标准ID（查看全部的ID列表）。使用标准ID是一个好的习惯，如果它存在的话。这有助于让wxWidgets在不同的平台上使每一个控件的ID都看起来更加自然

import wx

class MainWindow(wx.Frame):

def \_\_init\_\_(self, parent, title):

wx.Frame.\_\_init\_\_(self, parent, title=title, size=(200,100))

self.control = wx.TextCtrl(self, style=wx.TE\_MULTILINE)

self.CreateStatusBar() # A Statusbar in the bottom of the window

# Setting up the menu.

#先定义第一个子层菜单内容

filemenu= wx.Menu()

# wx.ID\_ABOUT and wx.ID\_EXIT are standard IDs provided by wxWidgets.

filemenu.Append(wx.ID\_ABOUT, "&About"," Information about this program")

filemenu.AppendSeparator()

filemenu.Append(wx.ID\_EXIT,"E&xit"," Terminate the program")

#定义第二个子层菜单

filemenu2= wx.Menu()

filemenu2.Append(wx.ID\_ANY, "&haha"," kjsdfoaoidf")

filemenu2.AppendSeparator()

filemenu2.Append(wx.ID\_ANY,"r&un"," run another")

#定义顶层菜单

# Creating the menubar.

menuBar = wx.MenuBar()

menuBar.Append(filemenu,"&File") # Adding the "filemenu" to the MenuBar

menuBar.Append(filemenu2,"&Comment") # Adding the "filemenu2" to the MenuBar

self.SetMenuBar(menuBar) # Adding the MenuBar to the Frame content.

#显示窗口

self.Show(True)

app = wx.App(False)

frame = MainWindow(None, "Sample editor")

app.MainLoop()

### 事件处理

#在wxPython中，对事件的响应，称作事件处理。事件就是指发生在你的程序当中的某些事情（一个按钮被按下，文本输入，鼠标移动等等）。GUI编程的很大一部分是由事件的响应组成的。你可以使用Bind()方法，将一个控件和事件绑定到一起

class MainWindow(wx.Frame):

def \_\_init\_\_(self, parent, title):

wx.Frame.\_\_init\_\_(self,parent, title=title, size=(200,100))

...

menuItem = filemenu.Append(wx.ID\_ABOUT, "&About"," Information about this program")

self.Bind(wx.EVT\_MENU, self.OnAbout, menuItem)

这意味着从现在开始，当用户选择About菜单项的时候，self.OnAbout方法将会被执行。wx.EVT\_MENU是“选择菜单项”事件。wxWidgets也会处理许多其他的事件（查看全部列表）。self.OnAbout方法一般是这样定义的：

def OnAbout(self, event):

...

这里的event是从wx.Event派生出一个子类的实例。比如说按钮点击事件-wx.EVT\_BUTTON就是wx.Event的一个子类。

当事件发生的时候，这个方法被执行。默认情况下，这个方法将会处理事件，并且在回调函数结束后，事件终止。但是，你可以跳过一个事件通过event.Skip()方法。这将会导致事件直接跨过事件用户处理层。比如说这样：

当一个按钮点击事件发生时，OnButtonClick方法被调用。如果some\_condition是真，我们就执行do\_something()，否则我们让这个事件用系统默认方式所处理。现在让我们看看我们的程序：

def OnButtonClick(self, event):

if (some\_condition):

do\_something()

else:

event.Skip()

def OnEvent(self, event):

...

import os

import wx

class MainWindow(wx.Frame):

def \_\_init\_\_(self, parent, title):

wx.Frame.\_\_init\_\_(self, parent, title=title, size=(200,100))

self.control = wx.TextCtrl(self, style=wx.TE\_MULTILINE)

self.CreateStatusBar() # A StatusBar in the bottom of the window

# Setting up the menu.

filemenu= wx.Menu()

# wx.ID\_ABOUT and wx.ID\_EXIT are standard ids provided by wxWidgets.

menuAbout = filemenu.Append(wx.ID\_ABOUT, "&About"," Information about this program")

menuExit = filemenu.Append(wx.ID\_EXIT,"E&xit"," Terminate the program")

# Creating the menubar.

menuBar = wx.MenuBar()

menuBar.Append(filemenu,"&File") # Adding the "filemenu" to the MenuBar

self.SetMenuBar(menuBar) # Adding the MenuBar to the Frame content.

# Set events.

#self.Bind(ID, functio\_name, menu\_name)

self.Bind(wx.EVT\_MENU, self.OnAbout, menuAbout)

self.Bind(wx.EVT\_MENU, self.OnExit, menuExit)

self.Show(True)

def OnAbout(self,e):

# A message dialog box with an OK button. wx.OK is a standard ID in wxWidgets.

#在这个例子中，我们可以忽略ID，wxWidget会自动使用一个默认的ID（就像我们指定了wx.ID\_ANY一样）

#dlg = wx.MessageDialog( self, "A small text editor", "About Sample Editor")

dlg = wx.MessageDialog( self, "A small text editor", "About Sample Editor", wx.OK)

dlg.ShowModal() # Show it

dlg.Destroy() # finally destroy it when finished.

def OnExit(self,e):

self.Close(True) # Close the frame.

app = wx.App(False)

frame = MainWindow(None, "Sample editor")

app.MainLoop()

### 对话框

#首先，我们创建了对话框，通过调用合适的构造函数。然后，我们调用了ShowModal。通过它，打开了对话框。“Modal（模式/模态）”意味着在用户点击了确定按钮或者取消按钮之前，他不能在该程序中做任何事情。ShowModal的返回值是被按下的按钮的ID。如果用户点击了确定按钮，我们就读文件。

你现在应该可以向菜单中添加相应的内容，并且将它和OnOpen方法链接起来。如果你有问题，可以翻到下面的附录，去查看完整的代码

def OnOpen(self,e):

""" Open a file"""

self.dirname = ''

dlg = wx.FileDialog(self, "Choose a file", self.dirname, "", "\*.\*", wx.FD\_OPEN)

if dlg.ShowModal() == wx.ID\_OK:

self.filename = dlg.GetFilename()

self.dirname = dlg.GetDirectory()

f = open(os.path.join(self.dirname, self.filename), 'r')

self.control.SetValue(f.read())

f.close()

dlg.Destroy()

### 可能的扩展

当然，这个程序距离一个像样的编辑器还差的很远。但是添加其他的功能不会比我们已经完成的部分难。你也许会从这些和wxPython绑定的样例代码中得到灵感。

拖放

MDI

选项卡视图/多文件

查找/替换对话框

打印对话框（印刷）

Python中的宏命令（使用eval函数）

等等...

在窗口中工作

标题：

框架

窗口

控件/工具

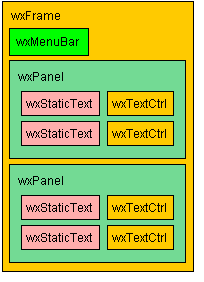
布局管理

验证器

在这段中，我们将学习如何使用wxPython处理窗口和它们的组件，包括构建输入框和使用各种各种的控件。我们将创建一个小的应用程序，用来显示一个标签。如果你是一个有GUI编程经验的开发者，这将是很简单的，你可以直接查看后面高级章节的Boa-Constructor子段落。

总览

### 可视化控件的布局

在一个框架当中，你将会使用很多wxPython的子类去充实框架里面的内容。这里是一些你将会在你的框架中使用到的常见的控件。

wx.MenuBar，在你的框架的顶部放一个菜单栏。

wx.Statusbar，在你的框架底部设置一个区域，来显示状态信息等等。

wx.ToolBar，在你的框架中放置一个工具栏

wx.Control的子类，这里面提供了一些控件的用户接口（比如说用来显示数据或者用户输入的可视化控件），常见的wx.Control对象包括wx.Button，wx.StaticText，wx.TextCtrl和wx.ComboBox。

wx.Panel，它是一个容器，可以用来包含你的许多wx.Control对象。将你的wx.Control对象放入一个wx.Panel中，意味着用户可以直接将一对控件从一个可视化器件移动到另一个可视化器件上。

所有的可视化控件（wx.Window对象和它们的子类）可以包含许多子控件。比如说，wx.Frame可以包含很多个wx.Panel对象，wx.Panel对象中又可以包含多个wx.Button，wx.StaticText和wx.TextCtrl对象，给大家一个完整的控件层次结构：HierarchyOfElements1注意：这个仅仅只是描述了可视化控件之间的相关关系，不是说明它们在框架中的布局情况。处理框架中的控件布局，你有以下几个选择：

你可以手动的指定每一个控件在父窗口中的像素坐标。由于字体大小的不同等等的问题，如果想要程序在不同的平台之间可以通用的话，这个选择一般不被考虑。

你可以使用wx.LayoutContains，它一般用起来很复杂。

你可以使用像Delphi一样的LayoutAnchors，它比wx.LayoutContains更加简单一点。

你可以使用一个wx.Sizer的子类。

在这个文档中将会集中使用wx.Sizers，因为是大家最常用的解决方案。

Sizer

Sizer用来解决在窗口和框架中可视化控件的放置问题。Sizer可以做下列事情：

为每个控件计算合适的大小。

通过一些包含规则放置每一个控件。

当一个框架的大小被改变后，自动的重新计算每个控件的大小并且改变其坐标。

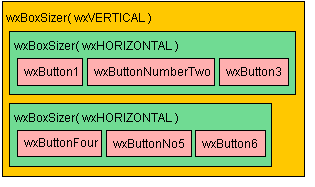
一些常见的sizers类包括：

wx.BoxSizer，以水平或垂直的方式将控件放置在一条线上。

wx.GridSizer，将控件以网状结构放置。

wx.FlexGridSizer，它和wx.GridSizer相似，但是它允许以更加灵活的方式放置可视化控件。

可以向Sizer添加一组wx.Window对象，可以通过调用sizer.Add(window, options...)，或者调用sizer.AddMany(...)这两个方法向sizer中添加控件。每一个Sizer只处理那些被添加进来的控件。Sizer可以被嵌套。也就是说，你可以将一个Sizer添加到另一个Sizer当中。举个例子来说有两排按钮（每一排按钮通过一个水平的wx.BoxSizer进行布局），它可以包含在另一个wx.BoxSizer中，将一排按钮放在另一排按钮的上面，就像这样：SizersExample1



注意：注意上面那个例子不是将6个按钮布局成2排3列，如果要那么做的话，那就应该使用wx.GridSizer。

在下面这个例子当中，我们使用2个嵌套的sizer，主Sizer是垂直布局，嵌套的Sizer是水平布局：

import wx

import os

class MainWindow(wx.Frame):

def \_\_init\_\_(self, parent, title):

self.dirname=''

# A "-1" in the size parameter instructs wxWidgets to use the default size.

# In this case, we select 200px width and the default height.

wx.Frame.\_\_init\_\_(self, parent, title=title, size=(200,-1))

self.control = wx.TextCtrl(self, style=wx.TE\_MULTILINE)

self.CreateStatusBar() # A Statusbar in the bottom of the window

# Setting up the menu.

filemenu= wx.Menu()

menuOpen = filemenu.Append(wx.ID\_OPEN, "&Open"," Open a file to edit")

menuAbout= filemenu.Append(wx.ID\_ABOUT, "&About"," Information about this program")

menuExit = filemenu.Append(wx.ID\_EXIT,"E&xit"," Terminate the program")

# Creating the menubar.

menuBar = wx.MenuBar()

menuBar.Append(filemenu,"&File") # Adding the "filemenu" to the MenuBar

self.SetMenuBar(menuBar) # Adding the MenuBar to the Frame content.

# Events.

self.Bind(wx.EVT\_MENU, self.OnOpen, menuOpen)

self.Bind(wx.EVT\_MENU, self.OnExit, menuExit)

self.Bind(wx.EVT\_MENU, self.OnAbout, menuAbout)

self.sizer2 = wx.BoxSizer(wx.HORIZONTAL)

self.buttons = []

for i in range(0, 6):

self.buttons.append(wx.Button(self, -1, "Button &"+str(i)))

self.sizer2.Add(self.buttons[i], 1, wx.EXPAND)

# Use some sizers to see layout options

self.sizer = wx.BoxSizer(wx.VERTICAL)

self.sizer.Add(self.control, 1, wx.EXPAND)

self.sizer.Add(self.sizer2, 0, wx.EXPAND)

#Layout sizers

self.SetSizer(self.sizer)

self.SetAutoLayout(1)

self.sizer.Fit(self)

self.Show()

def OnAbout(self,e):

# Create a message dialog box

dlg = wx.MessageDialog(self, " A sample editor \n in wxPython", "About Sample Editor", wx.OK)

dlg.ShowModal() # Shows it

dlg.Destroy() # finally destroy it when finished.

def OnExit(self,e):

self.Close(True) # Close the frame.

def OnOpen(self,e):

""" Open a file"""

dlg = wx.FileDialog(self, "Choose a file", self.dirname, "", "\*.\*", wx.OPEN)

if dlg.ShowModal() == wx.ID\_OK:

self.filename = dlg.GetFilename()

self.dirname = dlg.GetDirectory()

f = open(os.path.join(self.dirname, self.filename), 'r')

self.control.SetValue(f.read())

f.close()

dlg.Destroy()

app = wx.App(False)

frame = MainWindow(None, "Sample editor")

app.MainLoop()

sizer.Add这个方法有三个参数。第一个参数指定了放入sizer的控件名。第二个参数是一个宽度因子，它用来表示这个控件相对于其他控件的比例大小，比如说你有3个编辑框并且你希望它们的大小比例是3：2：1,那么当你添加这些控件的时候，你就可以将这些比例指定为这个参数。0意味着这个控件或者这个sizer将不会发生大小的变化。第3个参数通常使用wx.GROW(和wx.EXPAND一样)，它意味着这个控件当在需要的时候可以改变大小。如果你使用wx.SHAPED来替代wx.GROW，那么控件的纵横比将不会发生变化。

如果第二个参数是0,即控件将不会改变大小，第三个参数如果使用wx.ALIGN\_CENTER\_HORIZONTAL, wx.ALIGN\_CENTER\_VERICAL这两个参数替代wx.GROW或者wx.SHARPED，那么控件将会被定位在垂直或者水平方向的中间位置。如果使用wx.ALIGN\_CENTER那么控件将会被定位在程序窗口的正中间。

你可以在wx.ALIGN\_LEFT,wx.ALIGN\_TOP,wx.ALIGN\_RIGHT和wx.ALIGN\_BOTTOM中选择几个作为一个组合。默认的行为是wx.ALIGN\_LEFT | wx.ALIGN\_TOP。

大家可能对wx.Sizer和它的子类存在关于sizer和父窗口之间的差别的困惑。当你创建了一个对象放到了sizer里面，你并没有指定sizer的父窗口。Sizer只是一种窗口布局的方式，它本身并不是一个窗口，所以不需要指定sizer的父窗口。在上面的例子当中，六个按钮被创建的时候指定的父窗口是框架--不是sizer。如果你尝试创建一个可视化控件并且用sizer作为它的父窗口，那么你的程序将会崩溃。

当你建立了你的可视化控件之后并将它们添加到sizer（或者嵌套的sizer）里面，下一步就是告诉你的框架或者窗口去调用这个sizer，你可以完成这个，用以下三步：

[python] view plain copy

window.SetSizer(sizer)

window.SetAutoLayout(True)

sizer.Fit(window)

调用SetSizer()方法告诉你的窗口或框架去使用这个sizer。调用SetAutoLayout()方法告诉你的窗口使用sizer去为你的控件计算位置和大小。最后，调用sizer.Fit()告诉sizer去计算所有控件的初始位置和大小。如果你使用sizer进行布局，在第一次显示你窗口中所有的控件之前，这都是一个常见的步骤。

### Validator/验证器

当你创建了一个对话框或者一个其他的输入窗体，你可以使用wx.Validator来载入数据到你的输入窗体，验证输入的数据，再次从窗体中提取数据。wx.Validator也可以被用来截获键盘按键和其他的发生在输入区域框中的事件。如果要使用一个Validator，你应该创建一个你自己的wx.Validator的子类（wx.TextValidator和wx.GenericValidator都没有被wxPython实现）。通过调用myInputField.SetValidator(myValidator)来使你的这个子类和你的文字输入框联系起来。

注意：你的wx.Validator子类必须实现wx.Validator.Clone()方法。

一个可以运行的例子

我们的第一个程序包含一个panel（面板），panel中包含一个label（标签）

让我们开始一个例子。我们的程序将有一个框架，它有一个panel包含一个label：

[python] view plain copy

import wx

class ExampleFrame(wx.Frame):

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

wx.Frame.\_\_init\_\_(self, parent)

panel = wx.Panel(self)

self.quote = wx.StaticText(panel, label="Your quote: ", pos=(20, 30))

self.Show()

app = wx.App(False)

ExampleFrame(None)

app.MainLoop()

这段代码应该是很清晰的，并且应该没有任何问题。

注意：这里应该使用sizer，来替代指定控件坐标的部分。

注意这一行：

[python] view plain copy

self.quote = wx.StaticText(panel, label="Your quote: ", pos=(20, 30))

使用了panel作为我们的wx.StaticText的父窗口参数。我们的静态文本将出现在我们刚刚创建的面板上。wx.Point被用来做位置参数。还有一个可选参数wx.Size但是如果它在这里使用，将不会被对齐。

根据wxPython文档中的描述：

“面板（panel）是一个窗口，可以将其他的控件放到它的上面。它通常放置在框架中。在它父类（wx.Window）的基础上，它包含了最小的额外的功能。它的主要是用在功能和外观相似的对话框上，它要比任何的父窗口有更强的灵活性。”，事实上，它是一个简单的窗口，被用来做其他对象的背景。它作为一个控件，这些一般都是被大家所知道的。

标签（label）用来显示一些不能被用户所改变的文本。

添加更多的控件

你可以在wxPython的样例代码和帮助文档中发现全部控件的列表，但是我们在这里只展示几个最常用的控件：

wx.Button，最基础的控件：一个按钮上面显示文本，你可以点击它。比如说这里有一个“Clear”按钮：

[python] view plain copy

clearButton = wx.Button(self, wx.ID\_CLEAR, "Clear")

self.Bind(wx.EVT\_BUTTON, self.OnClear, clearButton)

wx.TextCtrl,文本框，这个控件可以让用户输入文本。它触发2个主要的事件。EVT\_TEXT，只要文本被改变了，它就会被调用。EVT\_CHAR，只要一个按键被按下，它就会被调用。

textField = wx.TextCtrl(self)

self.Bind(wx.EVT\_TEXT, self.OnChange, textField)

self.Bind(wx.EVT\_CHAR, self.OnKeyPress, textField)

比如说：如果用户按下了“Clear”按钮并且文字区域被清空，将会触发EVT\_TEXT事件，但是不会触发EVT\_CHAR事件。

wx.ComboBox, 组合框和文本框很相似，但是它除了包含wx.TextCtrl会触发的2个事件，它还有EVT\_COMBOBOX事件。

wx.CheckBox， 复选框是提供给用户选择真假的控件。

wx.RadioBox， 单选框可以让用户从一组选项中选择一个。

现在让我们看看这个Form1全部的代码：

[python] view plain copy

import wx

class ExamplePanel(wx.Panel):

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

wx.Panel.\_\_init\_\_(self, parent)

self.quote = wx.StaticText(self, label="Your quote :", pos=(20, 30))

# A multiline TextCtrl - This is here to show how the events work in this program, don't pay too much attention to it

self.logger = wx.TextCtrl(self, pos=(300,20), size=(200,300), style=wx.TE\_MULTILINE | wx.TE\_READONLY)

# A button

self.button =wx.Button(self, label="Save", pos=(200, 325))

self.Bind(wx.EVT\_BUTTON, self.OnClick,self.button)

# the edit control - one line version.

self.lblname = wx.StaticText(self, label="Your name :", pos=(20,60))

self.editname = wx.TextCtrl(self, value="Enter here your name", pos=(150, 60), size=(140,-1))

self.Bind(wx.EVT\_TEXT, self.EvtText, self.editname)

self.Bind(wx.EVT\_CHAR, self.EvtChar, self.editname)

# the combobox Control

self.sampleList = ['friends', 'advertising', 'web search', 'Yellow Pages']

self.lblhear = wx.StaticText(self, label="How did you hear from us ?", pos=(20, 90))

self.edithear = wx.ComboBox(self, pos=(150, 90), size=(95, -1), choices=self.sampleList, style=wx.CB\_DROPDOWN)

self.Bind(wx.EVT\_COMBOBOX, self.EvtComboBox, self.edithear)

self.Bind(wx.EVT\_TEXT, self.EvtText,self.edithear)

# Checkbox

self.insure = wx.CheckBox(self, label="Do you want Insured Shipment ?", pos=(20,180))

self.Bind(wx.EVT\_CHECKBOX, self.EvtCheckBox, self.insure)

# Radio Boxes

radioList = ['blue', 'red', 'yellow', 'orange', 'green', 'purple', 'navy blue', 'black', 'gray']

rb = wx.RadioBox(self, label="What color would you like ?", pos=(20, 210), choices=radioList, majorDimension=3,

style=wx.RA\_SPECIFY\_COLS)

self.Bind(wx.EVT\_RADIOBOX, self.EvtRadioBox, rb)

def EvtRadioBox(self, event):

self.logger.AppendText('EvtRadioBox: %d\n' % event.GetInt())

def EvtComboBox(self, event):

self.logger.AppendText('EvtComboBox: %s\n' % event.GetString())

def OnClick(self,event):

self.logger.AppendText(" Click on object with Id %d\n" %event.GetId())

def EvtText(self, event):

self.logger.AppendText('EvtText: %s\n' % event.GetString())

def EvtChar(self, event):

self.logger.AppendText('EvtChar: %d\n' % event.GetKeyCode())

event.Skip()

def EvtCheckBox(self, event):

self.logger.AppendText('EvtCheckBox: %d\n' % event.Checked())

app = wx.App(False)

frame = wx.Frame(None)

panel = ExamplePanel(frame)

frame.Show()

app.MainLoop()

我们的类现在变得更大了；它现在有了许多控件并且这些控件都是有响应的。我们添加了一个特别的wx.TextCtrl控件去显示控件发出的各种各样的事件。

标签页

有时候一个窗体变得太大了，以至于不能在一个单个的页面中显示。wx.NoteBook就是用来解决这个问题的：它允许用户通过点击相关的标签页，在多个页面中快速浏览。我们实现了这个，通过将窗体放入wx.NoteBook，而不是放入主框架中，并且通过使用AddPage方法将Form1添加到标签页中。

注意ExamplePanel的父窗口是NoteBook。这是很重要的。

[python] view plain copy

app = wx.App(False)

frame = wx.Frame(None, title="Demo with Notebook")

nb = wx.Notebook(frame)

nb.AddPage(ExamplePanel(nb), "Absolute Positioning")

nb.AddPage(ExamplePanel(nb), "Page Two")

nb.AddPage(ExamplePanel(nb), "Page Three")

frame.Show()

app.MainLoop()

### 加强布局-使用Sizer

使用一个绝对的坐标位置经常是不能让人满意的：如果窗口不是一个正确的大小，那么窗口最后的样子将会是很丑陋的。wxPython中有非常丰富的词汇来定义对象的位置。

wx.BoxSizer是一个最常用的并简单的布局对象，它只是将控件放在一个大概的位置。它的功能是很粗鲁地安排一系列的控件在一行或一列上，并且在需要的时候（比如说窗口的整体大小改变的时候）重新安排它们的位置。

wx.GridSizer和wx.FlexGridSizer是两个非常重要的布局工具。它们安排控件以表格的形式布局。

这里将上面的代码简单的重写了一下：

[python] view plain copy

class ExamplePanel(wx.Panel):

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

wx.Panel.\_\_init\_\_(self, parent)

# create some sizers

mainSizer = wx.BoxSizer(wx.VERTICAL)

grid = wx.GridBagSizer(hgap=5, vgap=5)

hSizer = wx.BoxSizer(wx.HORIZONTAL)

self.quote = wx.StaticText(self, label="Your quote: ")

grid.Add(self.quote, pos=(0,0))

# A multiline TextCtrl - This is here to show how the events work in this program, don't pay too much attention to it

self.logger = wx.TextCtrl(self, size=(200,300), style=wx.TE\_MULTILINE | wx.TE\_READONLY)

# A button

self.button =wx.Button(self, label="Save")

self.Bind(wx.EVT\_BUTTON, self.OnClick,self.button)

# the edit control - one line version.

self.lblname = wx.StaticText(self, label="Your name :")

grid.Add(self.lblname, pos=(1,0))

self.editname = wx.TextCtrl(self, value="Enter here your name", size=(140,-1))

grid.Add(self.editname, pos=(1,1))

self.Bind(wx.EVT\_TEXT, self.EvtText, self.editname)

self.Bind(wx.EVT\_CHAR, self.EvtChar, self.editname)

# the combobox Control

self.sampleList = ['friends', 'advertising', 'web search', 'Yellow Pages']

self.lblhear = wx.StaticText(self, label="How did you hear from us ?")

grid.Add(self.lblhear, pos=(3,0))

self.edithear = wx.ComboBox(self, size=(95, -1), choices=self.sampleList, style=wx.CB\_DROPDOWN)

grid.Add(self.edithear, pos=(3,1))

self.Bind(wx.EVT\_COMBOBOX, self.EvtComboBox, self.edithear)

self.Bind(wx.EVT\_TEXT, self.EvtText,self.edithear)

# add a spacer to the sizer

grid.Add((10, 40), pos=(2,0))

# Checkbox

self.insure = wx.CheckBox(self, label="Do you want Insured Shipment ?")

grid.Add(self.insure, pos=(4,0), span=(1,2), flag=wx.BOTTOM, border=5)

self.Bind(wx.EVT\_CHECKBOX, self.EvtCheckBox, self.insure)

# Radio Boxes

radioList = ['blue', 'red', 'yellow', 'orange', 'green', 'purple', 'navy blue', 'black', 'gray']

rb = wx.RadioBox(self, label="What color would you like ?", pos=(20, 210), choices=radioList, majorDimension=3,

style=wx.RA\_SPECIFY\_COLS)

grid.Add(rb, pos=(5,0), span=(1,2))

self.Bind(wx.EVT\_RADIOBOX, self.EvtRadioBox, rb)

hSizer.Add(grid, 0, wx.ALL, 5)

hSizer.Add(self.logger)

mainSizer.Add(hSizer, 0, wx.ALL, 5)

mainSizer.Add(self.button, 0, wx.CENTER)

self.SetSizerAndFit(mainSizer)

在这个例子当中使用GridBagSizer去放置控件。Grid对象的“pos”参数决定控件在grid中的位置。在这个实例中（0,0）是左上角，（3,5）指的是第三排第五列，span参数允许控件被强行扩展成多行多列。

在这段中，我们将介绍在窗口中画图的方法。我们也将会展示如何在主窗口中按下鼠标右键，出现弹出菜单。

### 一个可以运行的例子

使用wxPython

调试技术

当在你的程序中遇到一个不能被处理的异常时（bug！），程序被异常终止，那么使用追踪技术去定位问题代码是很有用的。wxPython程序也是一样的，但是它是扭曲的（twist）。wxPython中的追踪是重新定向标准输入输出。它是独立于你程序的GUI窗口的。如果一个异常发生在事件处理阶段，追踪信息会被显示出来，并且你的程序会尽力地继续执行。但是，如果异常发生在你程序的初始化阶段，追踪信息会被显示出来，然后你的程序会被异常终止，查看标准输入输出窗口（或者你重定向的位置），可以让读者尽快知道问题所在。你可以截获标准输入输出，通过wxPython提供的两个可选参数，在你实例化你的wx.App对象的时候，可以指定这两个参数。这个例子可以很好的说明：

[python] view plain copy

class MyApp (wx.App):

#...

#...

#...

myapp = MyApp() # functions normally. Stdio is redirected to its own window

myapp = MyApp(0) #does not redirect stdout. Tracebacks will show up at the console.

myapp = MyApp(1, 'filespec') #redirects stdout to the file 'filespec'

# NOTE: These are named parameters, so you can do this for improved readability:

myapp = MyApp(redirect = 1, filename = 'filespec') # will redirect stdout to 'filespec'

myapp = MyApp(redirect = 0) #stdio will stay at the console...

你也可以使用Widget Inspection Tool来帮你调试大多数的布局问题。

这里讨论代码调试问题，事件循环问题，等等。

PyCrust交互控制台

wxPython发布一个漂亮的PyCrust控制台。使用它，你可以以交互的方式测试你的布局。这里有一段简单的代码。

部署你的wxPython应用

下一步

事件

事件处理是wxPython很关键的一部分。我们知道所有的GUI系统都是依赖于事件去在各种应用程序之间分发消息。GUI程序的任务是当接收到一个特定的消息的时候，自己决定去做什么，并且调用事件处理。在面向对象编程以前，想要处理事件就意味着必须有一个“switch”操作，根据事件的类型来决定去做什么事。现在在面向对象编程中这就不再是那么简单的事情了。现在有2种事件处理的方式：

一种方法（像java）是依赖于事件处理器。事件处理器和一个特定的对象链接并且和一个回调函数/方法绑定。当对象接收到一个特定类型的事件，事件处理器就会触发回调函数。

另一种方法是预先给一个方法起一个名字，假定用这个方法处理某一种特定的事件。使用这种方法的话，如果你想要改变某个类对某个事件的响应，那么你就必须在原有的类的基础上进行派生，并且重载响应的方法。

wxPython结合了这两种方法。你可以定义事件处理器与派生类去实现新的行为。

所以“self.Bind(wx.EVT\_SOMETHING,ACallable)”的意思是：当EVT\_SOMETHING事件被发送到窗口的时候，它将会调用ACallable。ACallable可以是任何函数，通常程序员的选择是让它实现上一个类的功能。

第二个版本是“self.Bind(wx.EVT\_SOMETHING,ACallable,srcWin)”的意思是：当源窗口发出SOMETHING事件，这个事件到达窗口，然后调用ACallable。

但是很多事件只能被发出事件的窗口所捕获（这意味着第二个方式不能做任何事情），所以最好无论在什么情况下都使用第一种方法，第一种方法可以用在任何地方，除了菜单项，因为它没有Bind()方法。

在这里：创建一个自定义的事件处理程序

Scintilla

Scintilla是一个基本的组件，被用在wx.StyleTextCtrl类中，它可以使我们在wxPython中实现语法高亮的功能。

Boa-constructor

Boa-constructor是wxPython的一个集成开发环境

多线程

在这里：

有用的资源

http://wxPython.org

从这里开始，一个非常著名的网站，而且你也可以看看wxPython包中的样例代码。它都是非常有用的代码样例并且它几乎包含所有你能想到的主题。想要运行样例程序的话可以通过以下方法：

在windows中只需要选择开始菜单中的wxPython子菜单，然后只需要选择Run The Demo就可以了。

在Linux中找到样例代码的目录然后运行“python demo.py”

http://wxwidgets.org

你也可以尝试在wxWidgets的网站找到一些信息。wxPython的文档包含了wxWidget的所有内容，所以如果你开始看wxPython的文档的话，你可能会没有头绪。

http://wxpython.org/maillist.php

wxPython的邮箱列表，在这里你可以找到你想要的信息。但是在你提问之前，请先搜索档案库。

http://www.python.org

Python社区的参考手册网站。

http://starship.python.net/crew/theller/py2exe

这是一个工具，通过这个工具你可以将python脚本或者wxpython脚本转换成独立的windows中的可执行文件。这可以使你的程序被更广泛的使用。

### a

a

### a

a

### a

a

### a

a

### WxPython窗口编程说明(末尾)