Contents

[<命令行中的数据科学> 2](#_Toc460960114)

[1. Linux内置命令 3](#_Toc460960115)

[2. csvkit 6](#_Toc460960116)

[3. JSON处理器，包括分片、过滤、转换等等 7](#_Toc460960117)

[4. scrape – 用XPath和CSS选择器进行HTML信息提取的工具 8](#_Toc460960118)

[5. XML2JSON xml <-> json 8](#_Toc460960119)

[6. json2csv 8](#_Toc460960120)

[python使用 9](#_Toc460960121)

[包管理器pip 9](#_Toc460960122)

[模块查询 9](#_Toc460960123)

[virtual environment 9](#_Toc460960124)

[Requirements files 10](#_Toc460960125)

[Python脚本编程和运行方式 11](#_Toc460960126)

[Interactive Mode Programming 11](#_Toc460960127)

[Script Mode Programming 11](#_Toc460960128)

[like shell 11](#_Toc460960129)

[语法 11](#_Toc460960130)

[Standard Data Types 11](#_Toc460960131)

[Types of Operator 12](#_Toc460960132)

[函数 12](#_Toc460960133)

[模块 14](#_Toc460960134)

[Packages in Python 14](#_Toc460960135)

[输入输出 15](#_Toc460960136)

[惯用法 16](#_Toc460960137)

[函数式编程 17](#_Toc460960138)

[常用模块 17](#_Toc460960139)

[$import sys, os, glob, tempfile, shutil, base64, logging, smptplib 17](#_Toc460960140)

[日期和时间 20](#_Toc460960141)

[数学 20](#_Toc460960142)

[迭代器函数 22](#_Toc460960143)

[$import re 23](#_Toc460960144)

[文件解析 24](#_Toc460960145)

[import pprint 24](#_Toc460960146)

[copy and paste 24](#_Toc460960147)

[$import json 25](#_Toc460960148)

[$import csv 25](#_Toc460960149)

[Beautiful Soup --HTML and XML 解析 26](#_Toc460960150)

[PDF解析 26](#_Toc460960151)

[Excel解析 27](#_Toc460960152)

[Word解析 28](#_Toc460960153)

[多线程，定时任务和GUI自动化 28](#_Toc460960154)

[多线程 28](#_Toc460960155)

[启动进程 29](#_Toc460960156)

[定时任务 29](#_Toc460960157)

[GUI自动化 30](#_Toc460960158)

[数据库 31](#_Toc460960159)

[Sqlite3 31](#_Toc460960160)

[网络数据采集 32](#_Toc460960161)

[网络请求和响应 --requests 32](#_Toc460960162)

[解析HTML and XML --Beautiful Soup 34](#_Toc460960163)

[模拟浏览器操作 37](#_Toc460960164)

[web framework 38](#_Toc460960165)

[build-in web server to setup a web server 38](#_Toc460960166)

[Flask 39](#_Toc460960167)

[R 39](#_Toc460960168)

[install R and RStudio 39](#_Toc460960169)

[<R语言初学者指南> 40](#_Toc460960170)

[R语法 40](#_Toc460960171)

[数据获取 42](#_Toc460960172)

[数据预处理 （一般针对数据框） 42](#_Toc460960173)

[数据分析 43](#_Toc460960174)

[数据可视化 43](#_Toc460960175)

[<R and Ruby数据分析之旅> 44](#_Toc460960176)

[Spark 45](#_Toc460960177)

[《Spark快速大数据分析》 45](#_Toc460960178)

[install spark 46](#_Toc460960179)

[命令行（交互式分析数据） 47](#_Toc460960180)

[脚本（独立应用） 47](#_Toc460960181)

[RDD编程 48](#_Toc460960182)

# <命令行中的数据科学>

数据科学是一个宽广的领域，与许多领域都有交叉，例如程序设计、数据可视化以及机器学习。

数据科学含5个步骤：数据获取、数据清洗、数据探索、数据建模和数据解释

数据科学是OSEMN（和 awesome 相同发音），它包括获取(Obtaining)、整理(Scrubbing)、探索(Exploring)、建模(Modeling)和翻译(iNterpreting)数据。

数据获取

从其他地方（如网页或服务器）下载数据

从数据库或API（如MySQL and Twitter）中查询数据

从其他文件（如HTML文件或电子表格）中提取数据

自己生成数据（例如读取传感器或进行调查）

获取数据的格式通常是纯文本、CSV、JSON或HTML/XML

数据清洗

在所获取的数据中，缺失值、不一致、错误、怪异字符或冗余列屡见不鲜。常见的清洗操作：行过滤，列抽取、值替换、单词提取、缺失值处理、数据格式转换。任何数据项目中80%的工作都是数据清洗

数据探索

查看数据，从数据中推导统计量，创建有趣的可视化

数据建模

若想解释数据或预测将要发生的事情，应为数据建立统计模型。技术包括聚类、分类、回归以及降维

数据解释：从数据中得出结论、评估结果的含义

1. Linux内置命令
2. sudo apt-get install tree

tree -L 2 只查看当前第N级的目录和文件

1. seq first increment last 产生序列

seq -f "Line %g" 10

1. Bash特性：括号扩展，会将{0..100..2}变成一列由空格分隔的数字：0 2 4 … 98 100

for i in {0..100..2}

do

echo "$i^2" | bc

done | head

$echo $RANDOM 内置伪随机数，返回0~32 767之间

1. find

$find ~ -name "\*.txt" -print #在$HOME中查.txt文件并显示

$find /home -size +512k 查大于512k的文件

1. linux命令parallel实现多进程并行计算

$sudo apt-get install parallel

并行处理命令和管道

parallel -j 2 "sh a.sh" "sh b.sh"; 进程a.sh与进程b.sh并行运行

sh c.sh 等以上进程结束，才运行进程c.sh

1. 输入的字符进行替换、压缩和删除

tr(选项)(参数)

选项：

-c：取代所有不属于第一字符集的字符

-d：删除所有属于第一字符集的字符

-s：把连续重复的字符以单独一个字符表示

-t：先删除第一字符集较第二字符集多出的字符。

参数：

字符集1：指定要转换或删除的原字符集

字符集2：指定要转换成的目标字符集。

$ echo "thissss is a text linnnnnnne." | tr -s ' sn' 压缩输入中重复的字符

this is a text line.

$ tr '[:lower:]' '[:upper:]' 小写 -> 大写

$ echo $PATH | tr : '\n' | sort 显示搜索路径

$ echo ‘hello\_world!’ | tr ‘ ‘ ‘\_’

hello\_world!

1. $ sort -nrk 3 -t: sort.txt

eee:40:5.4

eee:60:5.1

ddd:20:4.2

ccc:50:3.3

bbb:10:2.5

aaa:30:1.6

AAA:BB:CC

# -n是按照数字大小排序，-r是以相反顺序，-k是指定需要爱排序的栏位，-t指定栏位分隔符为冒号

sort -u file.txt 删除重复行

1. uniq

uniq命令用于报告或忽略文件中的重复行，一般与sort命令结合使用。

uniq(选项)(参数)

选项

-c：在每列旁边显示该行重复出现的次数

-d：仅显示重复出现的行列

-f<栏位>：忽略比较指定的栏位

-s<字符位置>：忽略比较指定的字符

-u：仅显示出一次的行列

-w<字符位置>：指定要比较的字符。

参数

输入文件：指定要去除的重复行文件。如果不指定此项，则从标准读取数据；

输出文件：指定要去除重复行后的内容要写入的输出文件。如果不指定此选项，则将内容显示到标准输出设备（显示终端）。

uniq -u file.txt 只显示单一行：

sort file.txt | uniq -c 统计各行在文件中出现的次数：

sort file.txt | uniq -d 在文件中找出重复的行：

1. grep

grep（global search regular expression(RE) and print out the line，全面搜索正则表达式并把行打印出来）是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来

grep -E "[1-9]+"

grep match\_pattern file\_name

#只在目录中所有的.php和.html文件中递归搜索字符"main()"

grep "main()" . -r --include \*.{php,html}

#在搜索结果中排除所有README文件

grep "main()" . -r --exclude "README"

#在搜索结果中排除filelist文件列表里的文件

grep "main()" . -r --exclude-from filelist

1. awk '{pattern + action}' {filenames}

是一个强大的文本分析工具，相对于grep的查找，sed的编辑，awk在其对数据分析并生成报告时，显得尤为强大。简单来说awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。

命令行方式

awk [-F field-separator] 'commands' input-file(s)

其中，commands 是真正awk命令，[-F域分隔符]是可选的。 input-file(s) 是待处理的文件。

在awk中，文件的每一行中，由域分隔符分开的每一项称为一个域。通常，在不指名-F域分隔符的情况下，默认的域分隔符是空格。

$</etc/passwd | awk -F ':' 'BEGIN {print “start”} {print $1} END {print “end”}'

awk工作流程是这样的：读入有'\n'换行符分割的一条记录，然后将记录按指定的域分隔符划分域，填充域，$0则表示所有域,$1表示第一个域,$n表示第n个域。默认域分隔符是"空白键" 或 "[tab]键",所以$1表示登录用户，$3表示登录用户ip,以此类推。

1. sed

sed编辑器可以基于输入到命令行的或是存储在命令文本文件中的命令来处理数据流中的数据。它每次从输入中读取一行，用提供的编辑器命令匹配数据、按命令中指定的方式修改流中的数据，然后将生成的数据输出到STDOUT。

set options script file

$echo "This is a test" | sed 's/test/big test/' 替换

$sed '3i\This is an inserted line.' data7 插入行到数据流第3行前

$sed '3a\This is an inserted line.' data7 插入行到数据流第3行后

$sed '3c\This is a changed line of text.' data7 指定行整行替换

$sed 'y/123/789/' data8 转换命令

$sed -e 's/brown/breen/; s/dog/cat/' data1 执行多个命令于文件中的每行数据上

$sed 's!/bin/bash!/bin/csh!' /etc/passwd 感叹号使路径名不需要转义

1. printf命令

printf "%-5s %-10s %-4s\n" NO Name Mark

printf "%-5s %-10s %-4.2f\n" 01 Tom 90.3456

1. cut

cut的工作就是“剪”，具体的说就是在文件中负责剪切数据用的。cut是以每一行为一个处理对象的

cut命令主要是接受三个定位方法：

第一，字节（bytes），用选项-b

第二，字符（characters），用选项-c

第三，域（fields），用选项-f

$ head -n5 /etc/passwd |cut -d : -f 1,3-5

root:0:0:root

bin:1:1:bin

-d指定域分隔符，-f 指定要剪出哪几个域

1. curl -s http//www.../\*.txt -o data/finn.txt

curl -u username:password ftp://host/file

curl -L http://t.co/ 重定向

1. csvkit

<https://csvkit.readthedocs.io/en/0.9.0/>

$pip install csvkit

* Input

**in2csv**将csv,dbf, fixed, geojson, json, ndjson, xls, xlsx -> csv

$in2csv data/imdb.xlsx > data/imdb.csv

$head data/imdb.csv | cvscut -c Title,Year,Rating | csvlook

**sql2csv**利用python SQLAlchemy软件包，只需要使用一个工具就可以在许多不同的数据库上执行查询，包括MySQL, Oracle, PostgreSQL, SQLite, SQL Server and Sybase.

Execute an SQL query on a database and output the result to a CSV file.

#Load sample data into a table using csvsql and then query it using sql2csv:

$ csvsql --db "sqlite:///dummy.db" --table "test" --insert examples/dummy.csv

$ sql2csv --db "sqlite:///dummy.db" --query "select \* from test"

* Processing

**csvclean**

cleans a CSV file of common syntax errors. Outputs [basename]\_out.csv and [basename]\_err.csv, the former containing all valid rows and the latter containing all error rows along with line numbers and descriptions

$ csvclean examples/bad.csv

$ csvclean -n examples/bad.csv

**csvcut**

Filters and truncates CSV files. Like unix “cut” command, but for tabular data:

#Print the indices and names of all columns:

$ csvcut -n \*.csv

$ csvcut -c 1,3 \*.csv

$ csvcut -c TOTAL,"State Name" \*.csv

**csvgrep**

Filter tabular data to only those rows where certain columns contain a given value or match a regular expression:

Like the unix "grep" command, but for tabular data.

$ csvgrep -c 1 -r "^I" \*.csv

**csvjoin**

Merges two or more CSV tables together using a method analogous to SQL JOIN operation. By default it performs an inner join, but full outer, left outer, and right outer are also available via flags.

$csvjoin -c "ColumnKey,Column Key" --outer file1.csv file2.csv

This command says you have two files to outer join, file1.csv and file2.csv. The key column in file1.csv is ColumnKey, the key column in file2.csv is Column Key.

**csvsort**

sort CSV files. Like unix “sort” command, but for tabular data:

$ <countries.csv csvsort -c 2

**csvstack**

stack up the rows from multiple CSV files, optionally adding a grouping value to each row:

$ csvstack -g 2009,2010 State1.csv State2.csv

* Output (and Analysis)

**csvformat**

convert a CSV file to a custom output format.

#convert “standard” CSV file to a pipe-delimited one

$ csvformat -D "|" examples/dummy.csv

**csvjson**

converts a CSV file into JSON or GeoJSON (depending on flags):

$ <examples/realdata/FY09\_EDU\_Recipients\_by\_State.csv csvjson -i 4

**csvlook**

render a CSV file in the console as a fixed-width table.

$ csvlook examples/testfixed\_converted.csv

**csvpy**

loads a CSV file into a csvkit.CSVKitReader object and then drops into a Python shell so the user can inspect the data however they see fit

$ csvpy --dict examples/dummy.csv -v

**csvsql**

generate SQL statements for a CSV file or create execute those statements

directly on a database.

generate a SQL CREATE TABLE statement for a CSV file.

$ createdb test

$ csvsql --db postgresql:///test --table fy09 --insert FY09\_EDU\_Recipients\_by\_State.csv

use CSVSQL to “directly” query one or more CSV files. Please note that this will create an in-memory SQL database, so it won’t be very fast:

$ csvsql --query "select m.usda\_id, avg(i.sepal\_length) as mean\_sepal\_length from iris as i join irismeta as m on (i.species = m.species) group by m.species" examples/iris.csv examples/irismeta.csv

$ <data/imdb-lists.csv csvsql --query "SELECT title trim, Year, Rating FROM stdin"

**csvstat**

Prints descriptive statistics for all columns in a CSV file. Will intelligently determine the type of each column and then print analysis relevant to that type (ranges for dates, mean and median for integers, etc.):

$ csvstat examples/realdata/FY09\_EDU\_Recipients\_by\_State.csv

1. JSON处理器，包括分片、过滤、转换等等

<https://stedolan.github.io/jq/manual/>

$apt-get install jq

$<data.json python -m json.tool 格式化json

$<data.json jq . 格式化打印并高亮

$jq '.\_id' u.json 查看某个属性

$jq '.\_id, .description' u.json 查看某2个属性

jq '.versions["1.0.3"].keywords' u.json 看看一个数组

jq '.versions["1.0.3"].keywords[1]' u.json 查看一个数组的第1个值

jq '.versions["1.0.3"] | .name, .author.name' u.json 过滤属性

1. scrape – 用XPath和CSS选择器进行HTML信息提取的工具

<https://github.com/jeroenjanssens/data-science-at-the-command-line>

将python脚本scrape -> scrape命令

$ pip install cssselect

$ curl -sL 'http://en.wikipedia.org/wiki/List\_of\_countries\_and\_territories\_by\_border/area\_ratio' > wiki.html

$ < data/wiki.html grep wikitable -A 21

$ < data/wiki.html scrape -be 'table.wikitable > tr:not(:first-child)' > table.html

$ xml2json.py -t xml2json -o data/table.json data/table.html --strip\_text

$<table.json jq -c '.html.body.tr[] | {country: .td[1], border: .td[2], surface: .td[3]}' >countries.json

$ <data/countries.json json2csv -p -k border,surface >countries.csv

1. XML2JSON xml <-> json

<https://github.com/hay/xml2json>

$ chmod u+x xml2json

$ xml2json.py -t xml2json -o data/table.json data/table.html --strip\_text

$ xml2json.py -t json2xml -o data/table1.xml data/table.json

1. json2csv

converts a stream of newline separated json data to csv format.

https://github.com/jehiah/json2csv

$sudo apt-get install golang

$gedit ~/.bashrc

export GOPATH=$HOME/go

export PATH=$PATH:$GOROOT/bin:$GOPATH/bin

$go get github.com/jehiah/json2csv

$ <data/countries.json json2csv -p -k border,surface >data/countries.csv

data/countries.json

{"surface":"0.44","border":"3.2","country":"Vatican City"}

{"surface":"2","border":"4.4","country":"Monaco"}

border,surface

3.2,0.44

4.4,2

# python使用

## 包管理器pip

pip类似RedHat的yum, Ubuntu的apt-get, Ruby的gem

PyPI is a repository for open-source third-party Python packages. It's similar to RubyGems in the Ruby world, PHP's Packagist, CPAN for Perl, and NPM for Node.js.

Install easy\_install

$ sudo apt-get install python-setuptools python-dev build-essential

$ sudo easy\_install pip

or

$apt-get -y install python-pip

Python has primitive package manager called easy\_install, which is installed automatically when you install Python itself

$pip --help

$pip list 列出已安装包

$pip install 安装包

$pip install --upgrade 升级包

$pip uninstall 卸载包

$pip show 显示包详细信息

$pip search 搜索包，类似yum里的search

## 模块查询

help(‘模块’) 查看模块的帮助信息

dir(‘模块’) 列出模块的类，函数，变量等

help(‘模块.函数’) 查看模块内函数的具体信息

## virtual environment

virtualenv is a tool for creating isolated Python environments containing their own copy of python, pip, and their own place to keep libraries installed from PyPI.

simply a directory that contains a complete copy of everything needed to run a Python program, including a copy of the python binary itself, a copy of the entire Python standard library, a copy of the pip installer, and (crucially) a copy of the site-packages directory mentioned above.

$sudo pip install virtualenv

若有代理：

$export https\_proxy=http://web-proxy.mydomain.com

$sudo -E pip install virtualenv

Usually pip and virtualenv are the only two packages you ever need to install globally.In fact, virtualenv comes with a copy of pip which gets copied into every new environment you create, so virtualenv is really all you need.

$ cd ~/code/myproject/

$ virtualenv env

Start by changing directory into the root of your project directory, and then use the virtualenv command-line tool to create a new environment

It's a common convention to call this directory env, and to put it inside your project directory (so, say you keep your code at ~/code/projectname/, the environment will be at ~/code/projectname/env/ - each project gets its own env).

Note: if you're using a version control system like git, you shouldn't commit the env directory. Add it to your .gitignore file (or similar).

projectname

env

bin

include

lib

helloworld.py

helloworld.py

import requestsprint requests.get('http://www.baidu.com')

$env/bin/pip install requests

$env/bin/python helloworld.py

可以简约如下：

$source env/bin/activate 激活虚拟环境

虚拟环境Python解释器的路径就被添加进PATH中，但这种改变不是永久性的，它只会影响当前的命令行会话

(env)$ pip install flask

(env)$ python

>>> ...正常python命令行操作

(env)$ deactivate 退出虚拟环境

## Requirements files

Requirements files give you a way to create an environment: a set of packages that work together.

requirements make explicit, repeatable installation of packages.

requirements.txt

MyApp

Framework==0.9.4

Library>=0.2

$pip install -r requirements.txt

Output installed packages in requirements format

$pip freeze > requirements.txt

## Python脚本编程和运行方式

### Interactive Mode Programming

$ python

>>> print "Hello, Python!"

>>> exit()

### Script Mode Programming

test.py

print "Hello, Python"

$ python test.py

### like shell

#!/usr/bin/python

print "Hello, Python!"

$ chmod +x test.py # This is to make file executable

$./test.py

## 语法

### Standard Data Types

Numbers

String

List

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

Tuple

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated

Dictionary

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

### Types of Operator

Arithmetic Operators： +， -， \*， 、， %， \*\*， //

Comparison (Relational) Operators： ==, !=, <>, >, <, >=, <=

Assignment Operators： =, +=, -=, \*=, /=, %=, \*\*=, //=

Logical Operators (Bitwise Operators)： &, |, ^, ~, <<, >>

Membership Operators： in, not in

Identity Operators： is, is not

### 函数

#### 函数定义

def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

sum = lambda arg1, arg2: arg1 + arg2;

#### 函数传参

Pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example −

def changeme( mylist ):

mylist.append([1,2,3,4]);

print "Values inside the function: ", mylist

return

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

argument is being passed by reference and the reference is being overwritten inside the called function.

def changeme( mylist ):

mylist = [1,2,3,4]; # This would assig new reference in mylist

print "Values inside the function: ", mylist

return

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

#### 函数参数

Required arguments

Keyword arguments：

def printinfo( name, age ):

...

printinfo( age=50, name="miki" )

Default arguments

def printinfo( name, age = 35 ):

...

printinfo( name="miki" )

Variable-length arguments

def printinfo( arg1, \*vartuple ):

...

printinfo( 70, 60, 50 )

结果：arg1=70, vartuple=(60, 50)

An asterisk (\*) is placed before the variable name that holds the values of all nonkeyword variable arguments.

### 模块

#### 模块定义

A module can define functions, classes and variables.

support.py

def print\_func( par ):

print "Hello : ", par

return

#### 模块调用

# Import module support

import support

# Now you can call defined function that module as follows

support.print\_func("Zara")

The from...import Statement

The from...import \* Statement:

#### 模块搜索

1. The current directory.
2. If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
3. If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.

### Packages in Python

Phone/Isdn.py file having function Isdn()

Phone/G3.py file having function G3()

Phone/\_\_init\_\_.py

from Pots import Pots

from Isdn import Isdn

from G3 import G3

To make all of your functions available when you've imported Phone, you need to put explicit import statements in \_\_init\_\_.py as above

# Now import your Phone Package.

import Phone

Phone.Pots()

Phone.Isdn()

Phone.G3()

## 输入输出

#### STDOUT

#!/usr/bin/python

print """Hello, Python!

I am qzlin,

please welcome me"""

$./helloworld.py | grep py

#### STDIN

#!/usr/bin/python

import sys

for line in sys.stdin:

print line

$./helloworld.py | ./stream-read.py

argv

import sys

print 'Then command line arguments are: '

for (i, v) in enumerate(sys.argv):

print i, "\t", v

$./argsv-ex.py arg1 arg2 arg3

1. ./argv-ex.py
2. arg1
3. arg2
4. arg3

#### 文件迭代器

当文件对象用于for loop时，Python会自动调用readline方法

#文件内容如果可以很小，一次性读取所有内容:

f = open(filename)

for line in f.readlines():

process(line)

f.close()

#文件内容很大，流式处理,一行一行读取

import fileinput

for line in fileinput.input(filename):

process(line)

漂亮打印python数据结构

$import pprint

pprint.pprint(sys.path) 用来打印python数据结构

格式化json

json输出 | python -m json.tool

## 惯用法

’/’.join((‘’, ‘usr’, ‘bin’, ‘env’)) => ‘/usr/bin/env’

list2 = [x\*x for x in range(10) if x%3==0]

list1 = [(x,y) for x in range(3) for y in range(3)]

sequence unpacking

**first, second, \*rest = sequence**

等同于first =sequence[0], second=sequence[1], rest=sequence[2:];

Sequence unpacking可以用来传递任意数量的位置参数

map unpacking

args = (600, 900)

kwargs = dict(copies=2, collate=False)

**def print\_args(\*args, \*\*kwargs):**

可以用来传递任意数量的关键字参数

args 是传入位置参数的tuple

kwargs是传入关键字参数的dict

**locals() & glocal()**

它囊括了函数执行到该时间点时所定义的一切变量, 返回的字典对所有局部变量的名称与值进行映射.常配合mapping unpacking，用来实例化字符串模板

str.format(\*\*locals()).

字符串模板形参需要的实参从locals()返回的dict传入

集合

遍历

for (i, v) in enumerate(arr): list

…

for (key, value) in scientists.iteritems(): dict

…

for name, age in zip(names, ages): 并行迭代多个序列

…

## 函数式编程

filter(function, sequence)

map(function, sequence)

reduce(function, sequence, starting\_value)

filter(lambda x: x%2!=0 and x%3!=0, range(2, 25))

map(lambda x: x\*\*2, range(10))

reduce(lambda x,y: x+y, range(1, 100))

## 常用模块

高级文本处理NLTK程序包

大量数值型数据，Pandas程序包

### $import sys, os, glob, tempfile, shutil, base64, logging, smptplib

**$import sys**

处理Python运行时配置以及资源，从而可以与前当程序之外的系统环境交互

sys.stdin 标准输入流；sys.stdout 标准输出流；sys.stderr 标准错误流。

sys.path 查找模块所在目录的目录名列表。

sys.argv 命令行的参数列表，第一项脚本名称，第二项第一个命令行参数

sys.platform 返回当前系统平台，如：win32、Linux等。

$import os

Python的系统编程的操作模块，可以处理文件和目录

os.sep 更改操作系统中的路径分隔符。

os.getcwd()获取当前路径，这个在Python代码中比较常用。

os.listdir() 列出当前目录下的所有文件和文件夹。

os.remove() 方法可以删除指定的文件。

os.system() 方法用来运行shell命令。

os.chdir() 改变当前目录，到指定目录中。

os.makedirs() 创建新文件夹

**$import os.path**

os.path：多平台上处理文件

os.path.split('/one/two/three') 解析路径

parts = ('/', 'one', 'two', 'three')

os.path.join(\*parts) /one/two/three

os.path.isfile(file)

os.path.isdir(file)

os.path.exists(file)

**import glob**

glob.glob('dir/\*') 匹配目录dir中的所有路径名（文件或目录）

glob.glob('dir/file?.txt')

glob.glob('dir/\*[0-9].\*') 字符区间

**import tempfile**

tempfile模块提供了多个函数来安全地创建临时文件系统资源

临时文件系统对象，文件关闭时会自动删除

TemporaryFile() 打开并返回一个未命名的文件

temp = tempfile.TemporaryFile()

NamedTemporaryFile()打开并返回一个命名文件

with tempfile.NamedTemporaryFile() as temp:

mkdtemp()创建一个临时目录，并返回其目录名

dir\_name = tempfile.mkdtemp()

tempfile.gettempdir() 获取临时文件位置

查找过程：

环境变量TMPDIR -> 环境变量TEMP -> 环境变量TMP -> window(C:\TMP, ...), Linux(/tmp, /var/tmp, usr/tmp) -> 当前工作目录

高级文件操作

**import shutil**

shutil.copyfile('main.py', 'main.py.copy')

shutil.copy('main.py', 'script') 类似于unix的cp,若目标是目录，则copy文件于该目录下

shutil.move('main.py', 'main.txt') 类似于unix的mv,改名或移动文件

shutil.copytree(src, dst) 复制整个文件夹

shutil.rmtree(path) 删除整个文件夹

import send2trash

send2trash.send2trash('bacon.txt') 删除至回收站

**$import base64**

info = “qizhonglin”

info\_b64encode = base64.b64encode(info)

info\_copy = base64.b64decode(info\_b64encode )

**$import logging**

日志信息保存到一个自己设定格式的文件中

loggin.basicConfig(level=loggin.DEBUG, format=' %(asctime)s - %(levelname)s - %(message)s')

#logging.diable(logging.CRITICAL) 禁用接下来的日志

logging.debug('Start of program')

**$import smptplib**

**$import email.utils**

$from email.mime.text import MIMEText

SMTP（简单邮件传输协议）规定电子邮件应该如何格式化、加密、在邮件服务器之间传递，以及在你点击发送后，计算机要处理的所有其他细节

处理Email，发送、接收、抄送、下载邮件内容等操作

msg = MIMEText('这是消息的主体部分.')

msg['To'] = email.utils.formataddr(('admin', 'admin@example.com'))

msg['From'] = email.utils.formataddr(('Author', 'author@example.com'))

msg['Subject'] = 'Simple test message'

smtpObj = smtplib.SMTP(‘smtp.gmail.com’, 587) 连接到SMTP服务器

try:

smtpObj.ehlo() 向SMTP电子邮件服务器打招呼

smtpObj.starttls() 开始TLS加密

smtpObj.login(‘author@example.com’, ‘password’) 登录到SMTP服务器

smtpObj.sendmail('author@example.com', ['admin@example.com'], msg.as\_string())

finally:

smtpObj.quit()

IMAP（因特网消息访问协议）规定了如何与电子邮件服务提供商的服务器通信，取回发送到你的电子邮件地址的电子邮件

//用IMAP获取和删除电子邮件

import imapclient, pyzmail

imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True) 连接IMAP服务器

imapObj.login('author@gmail.com', 'password') 登录

imapObj.select\_folder('INBOX', readonly=True) 搜索电子邮件

UIDs = imapObj.search(['SINCE 05-Jul-2014']) 执行搜索

rawMessages = imapObj.fetch([UIds], ['BODY[]', 'FLAGS']) 取邮件

//解析邮件内容

message = pyzmail.PyzMessage.factory(rawMessages[UIDs[i]['BODY[]'])

message.get\_subject()

message.get\_addresses('from')

message.get\_addresses('to')

message.get\_addresses('cc')

message.html\_part.get\_payload().decode(message.html\_part.charset)

### 日期和时间

$import time

time.time() 返回从纪元以来的秒数

time.ctime() 返回可读的时间

time.clock() 返回处理器时钟时间，用于性能测试、基准测试等

time.gmtime() 以UTC格式返回当前时间

time.localtime() 返回应用了当前时区的当前时间

time.sleep(5) 程序暂停5秒，阻塞直到5秒时间过去，不要一次调用time.sleep(30)来暂停30秒，而是使用for循环执行30次time.sleep(1)调用

for i in range(30):

time.sleep(1)

解析和格式化时间

now = time.ctime()

parsed = time.strptime(now)

print ' tm\_year : ', parsed.tm\_year

time.strftime("%a %b ...", parsed)

$import datetime

t = datetime.time() 时间

t.hour, t.minute, ...

today = datetime.date.today() 日期

today.tm\_year, today.tu\_mon, today.tm\_hour, ...

时间运算

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

tommorrow = toda + datetime.timedelta(days=1)

today > yesterday

today < tommorrow

$import calendar

c = calendar.TextCalendar(calendar.SUNDAY)

c.prmonth(2011, 7) 返回一个月的格式化文本输出

print c.formatyear(2011, 2, 1, 1, 3) 返回一年的格式化文本输出

### 数学

$import decimal

Decimal类实现定点和浮点运算，构造函数取一个整数或字符串作为参数,重载了算术运算符

decimal.Decimal('3.14') + decimal.Decimal(str(f))

$import fractions

Fraction类基于numbers模块中Rational定义的API，实现了有理数的数值运算,重载了算术运算符

fractions.Fraction(1, 2) ==1/2

fractions.Fraction('1/2')

fractions.Fraction('0.5')

$import math

pring "pi = %.30f" % math.pi 取小数点后30位的值

math.trunc()

math.floor()

math.ceil()

math.modf() 返回tuple,输入值的小数和整数

math.fabs()

math.pow(x, y)

math.log()

math.exp()

math.radians() <-> math.degrees()

math.hypot(deltaX, deltaY)

$import random

生成随机浮点数、整数、字符串，甚至帮助你随机选择列表序列中的一个元素，打乱一组数据等。

random.seed(1)

random() 返回0<=n<1之间的随机实数n；

random.uniform(a, b)设定浮点数的范围，一个是上限，一个是下限。

random.randint(-100, 100)随机生一个整数int类型，可以指定这个整数的范围

random.randrange(0, 101, 5)区间选择值

random.choice(seq) 从序列seq中返回随机的元素；

random.getrandbits(n) 以长整型形式返回n个随机位；

random.shuffle(seq[, random]) 原地指定seq序列；

random.sample(seq, n) 从序列seq中选择n个随机且独立的元素；

非均分分布

random.normalvariate()

random.gauss()

random.lognormvariate()

random.triangular()

random.expovariate()

random.paretovariate()

random.vonmisesvariate()

random.betavariate()

random.gammavariate()

random.weibullvariate()

### 迭代器函数

与使用列表的代码相比，基于迭代器的算法可以提供更好的内存使用特性。在真正需要数据之前，并不从迭代器生成数据，由于这个原因，不需要将所有数据都同时存储在内存中，这种“懒”处理模型可以减少内存使用，相应地还可以减少交换以及大数据集的其他副作用，从而改善性能

$from itertools import \*

for i in chain([1, 2], ['a', 'b']) //chain合并迭代器成列表迭代器

print i

for i in izip([1,2], ['a', 'b']): //izip，合并迭代器成元组迭代器

print i //(1, 'a')

//imap函数返回迭代器，对输入迭代器中的值调用一个函数并返回结果

for i in imap(lambda x,y: (x, y, x\*y), xrange(5), xrange(5, 10):

print '%d \* %d = %d' % i //0 \* 5 = 0

//starmap函数类似imap,不过使用\*语法分解迭代器中的元素作为映射函数的函数

values = [(0, 5), (1, 6)]

for i in starmap(lambda x,y: (x, y, x\*y), values):

print '%d \* %d = %d' % i //0 \* 5 = 0

count(start)返回迭代器，无限地生成连续整数

islice(迭代器，start, end, step)

fro i in islice(count(100), 0, 100, 10):

print i

结果是100, 110, 120, ...

//tee()函数根据一个原输入迭代器返回多个独立的迭代器（默认为两个），

r = islice(count(), 5)

i1, i2 = tee(r)

print 'i1:' list(i1)

print 'i2:' list(i2)

//cycle()

for i, item in izip(xrange(7), cycle(['a', 'b', 'c']):

print (i, item)

item是从['a', 'b', 'c']循环获取

对应str \* 10生成列表[str str str...]

也可以采用迭代器方式

for i in repeat(str, 10):

print i

filter()生成列表

ifilter()生成迭代器

for i in ifilter(filter\_condition, [-1, 2, ...]):

print 'Yielding:', i

groupby()返回迭代器，每个迭代器键-值列表

根据一个属性对相关的值分组

data = [(0, 0), (1, 0), (0, 1), (1, 1)]

for k, g in groupby(data, operator.attrgetter('x')):

print list(g)

结果：

[(0, 0), (0, 1)]

[(1, 0), (1, 1)]

### $import re

1、匹配字符

. 匹配任意除换行符，也就是“\n”以外的任何字符。

\ 转义符，改变原来符号含义，后面会有演示。

[ ] 中括号用来创建一个字符集，第一个出现字符如果是^，表示反向匹配。

2、预定义字符集

\d 匹配数字，如：[0-9]

\D 与上面正好相反，匹配所有非数字字符。

\s 空白字符，如：空格，\t\r\n\f\v等。

\S 非空白字符。

\w 单词字符，如：大写A~Z，小写a~z，数字0~9。

\W 非上面这些字符。

[^abc] 匹配不在这些字符类的所有字符

^Hello 表明匹配必须发生在被查找文本开始处

Hello$ 表明被查找文本必须以这个正则表达式的模式结束

3、可选项与重复子模式

｜ 用管道匹配或关系 re.compile(r’Batman|Tina’)

\* 匹配前一个字符0次或无限次数。

+ 匹配前一个字符1次或无限次数。

? 匹配前一个字符0次或1次。

{m} 匹配前一个字符m次。

{m,n} 匹配前一个字符m至n次。

compile(模式字符串) 根据正则表达式字符串，创建模式的对象。

search(待查找字符串) 返回第一个匹配项（Match对象）。

split() 根据模式的匹配项来分割字符串。

findall(待查找字符串) 返回所有匹配项(文本列表,不是Match对象)，若有分组，返回元组的列表

sub(new, 待处理字符串) 用将所有匹配项用new替换掉

escape() 将字符串中所有特殊正则表达式字符转义

import re

phoneRegex = re.compile(r'\d{3}-\d{3}-\d{4}') 创建正则表达式对象

matchObj = phoneRegex.search(待查找字符串) 查找字符串,返回Match对象

matchObj.group() 返回实际匹配文本的字符串

利用括号分组,第一对括号是第一组，第二对括号是第2组

phoneRegex = re.compile(r'(\d{3})-(\d{3}-\d{4})')

matchObj = phoneRegex.search(待查找字符串)

matchObj.group(1) 返回第一个分组

matchObj.group(2) 返回第二个分组

matchObj.group() 返回整个匹配文本即\d{3}-\d{3}-\d{4}

areaCode, mainNumber = matchObj.groups() 返回所有分组的元组

Python的正则表达式默认是贪心的，这表示在有二义的情况下，它们会尽可能匹配最长的字符串

## 文件解析

### import pprint

pprint.pprint()美观打印python数据结构

pprint.pformat()返回python数据结构的字符串表示。一般用来保存变量和内容到\*.py文件中，该文件将成为你自己的模块，如果你需要使用存储在其中的变量，就可以导入它

from pprint import pformat

cats = [{'name': 'qzlin', 'desc': 'sleepy'}, {...}]

fileObj = open('myCats.py', 'w')

fileObj.write('cats = ' + pformat(cats) + '\n')

fileObj.close()

### copy and paste

import pyperclip

pyperclip.copy('Hello world')

pyperclip.paste()

### $import json

list = [“iplaypython”, [1,2,3], {“name”: “xiaoming”}] 列表

//dumps()将Python数据结构转换成json字符串

encoded\_json = json.dumps(list) 将列表进行json格式化编码

//loads()读取json字符串

decode\_json = json.loads(encoded\_json) 解码python json格式

data\_json = json.loads(open(‘data/input.json’).read())

print json.dumps(data\_json, indent=4) 漂亮打印json数据

### $import csv

读取写入操作csv数据文件

reader()读取csv文件数据的函数方法；

writer()写入csv文件数据的函数方法；

#!/usr/bin/python

import csv

def write\_csv(): with open('data/some.csv', 'wb') as f: writer = csv.writer(f, delimiter=' ', quotechar='|', quoting=csv.QUOTE\_MINIMAL) writer.writerow(['a', '1', '2']) writer.writerow(['b', '2', '3'])

def read\_csv(): with open('data/some.csv', 'rb') as f: reader = csv.reader(f, delimiter=' ', quotechar='|') for row in reader: print ','.join(row)

if \_\_name\_\_ == '\_\_main\_\_': write\_csv() read\_csv()

### Beautiful Soup --HTML and XML 解析

### PDF解析

方法一：pdf2htmlEX -> BeautifulSoup

方法二：PyPDF2

PyPDF2 is a pure-python PDF library capable of splitting, merging together, cropping, and transforming the pages of PDF files.PyPDF2 is a pure-python PDF library capable of splitting, merging together, cropping, and transforming the pages of PDF files

<https://www.binpress.com/tutorial/manipulating-pdfs-with-python/167>

PyPDF2没有办法从PDF文档中提取图像、图表或其他媒体，但它可以提取文本，并将文本返回为Python字符串

import PyPDF2

pdfFileObj = open('example.pdf', 'rb')

pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

//解密PDF

pdfReader.isEncrypted

pdfReader.decrypt('password')

pdfReader.numPages

pageObj = pdfReader.getPage(0)

//提取文本，可以会出错

pdfObj.extractText()

//拷贝页面，旋转页面，叠加页面，加密页面，

加水印

import os

from pyPdf import PdfFileWriter, PdfFileReader

from reportlab.pdfgen import canvas

def create\_watermark(content, pdf\_watermark):

c = canvas.Canvas(pdf\_watermark)

c.setFont('Courier', 10)

c.saveState()

c.translate(300, 15)

c.drawCentredString(0, 0, content)

c.restoreState()

c.save()

pdf\_watermark = PdfFileReader(file(pdf\_watermark, 'rb'))

return pdf\_watermark

def add\_watermark(pdf\_file, pdf\_watermark):

pdf\_output = PdfFileWriter()

pdf\_input = PdfFileReader(file(pdf\_file, 'rb'))

watermark = pdf\_watermark.getPage(0)

numPages = pdf\_input.getNumPages()

for i in xrange(numPages):

page = pdf\_input.getPage(i)

page.mergePage(watermark)

pdf\_output.addPage(page)

with open('output.pdf', 'wb') as f:

pdf\_output.write(f)

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

pdf\_watermark = create\_watermark('www.site-digger.com', 'watermark.pdf')

add\_watermark('test.pdf', pdf\_watermark)

### Excel解析

一个Excel电子表格文档称为一个工作簿。每个工作簿含多个工作表，用户当前查看的表，称为活动表

每个表都有一些列（地址是从A开始的字母）和一些行（地址是从1开始的数字）

import openpyxl

wb = openpyxl.load\_workbook('example.xlsx')

wb.get\_sheet\_names()

sheet = wb.get\_sheet\_name('Sheet3')

sheet.title

activeSheet = wb.get\_active\_sheet()

sheet['A1'].value

sheet.cell(row=1, column=2).value

for rowOfCellObjects in sheet['A1':'C3']:

for cellObj in rowOfCellObjects:

fun(cellObj.coordinate, cellObj.value)

wb.save('exmaple\_copy.xlsx')

from openpyxl.styles import Font, Style

styleOj = Style(font=Font(size=24, italic=True))

sheet['A'].style = styleObj

sheet['A1'] = 'Hello world'

wb.save('styled.xlsx')

设置行高列宽，合并拆分单元格，图表等。。。

### Word解析

在最高一层，Document对象表示整个文档，Document对象包含一个Paragraph对象的列表，表示文档中的段落，每个Paragraph对象都包含一个Run对象的列表。

Word文档中的文本不仅仅是字符串，它包含与之相关的字体、大小、颜色和其他样式信息

pip install python-docx

import docx

doc = docx.Document('demo.docx')

paragraphList = doc.paragraphs

paragraphList[0].text

runList = paragraphList[0].runs

runList[0].text

fullText = []

for para in doc.paragraphs:

fullText.append(para.text)

'\n'.join(fullText)

设置Paragraph and Run对象的样式

对于Word文档，有3种类型的样式：段落样式可以应用于Paragraph对象，字符样式可以应用于Run对象，链接样式可以应用于这两种对象

具体见<Python编程快速上手>p256

paraObj = doc.add\_paragraph('...')

paraObj.add\_run('...')

doc.add\_picture('zo.png', width=docx.shared.Inches(1), height=docx.shared.Cm(4))

在处理PDF and Word文档时有很多限制，因为这些格式的本章是很好地展示给人看，而不是让软件易于解析。而Json and CSV文件是设计给计算机使用

## 多线程，定时任务和GUI自动化

### 多线程

计算机的时钟可以调度程序，在特定的时间和日期运行，或定期运行

利用subprocess and threading模块，可以编程按时启动其他程序

不必让所有的代码等待，直到time.sleep()函数完成，你可以使用Python的threading模块，在单独的线程中执行延迟或安排的代码。这个单独的线程将因为time.sleep()调用而暂停。同时，程序可以在原来的线程中做其他工作

import threading, time

print('Start of program')

def takeANap():

time.sleep(5)

print('Wake up!')

threadObj = threading.Thread(target=takeANap)

threadObj.start()

print('End of program.')

向线程的目标函数传递参数

threadObj = threading.Thread(target=print, args=['cats', 'dogs', 'frogs'], kwargs={'sep': '&'})

threadObj.start()

常规参数可以作为一个列表，传递给threading.Thread()中的args关键字参数。关键字参数可以作为一个字典，传递给threading.Thread()中的kwargs关键字参数

### 启动进程

import subprocess

subprocess.Popen([r'c:\Windows\notepad.exe', r'c:\hello.txt'])

向Popen()传递一个列表，作为唯一的参数。该列表中的第一个字符串是要启动的程序的可执行文件名，所有后续的字符串将是该程序启动时，传递给该程序的命令行参数。实际上，这个列表将作为被启动程序的sys.argv的值

### 定时任务

cron job被用于安排那些需要被周期性执行的命令。利用它，你可以配置某些命令或者脚本，让它们在某个设定的时间内周期性地运行。cron 是 Linux 或者类 Unix 系统中最为实用的工具之一。cron 服务（守护进程）在系统后台运行，并且会持续地检查 /etc/crontab 文件和 /etc/cron.\*/目录。它同样也会检查 /var/spool/cron/ 目录。

crontab 是用来安装、卸载或者列出定时任务列表的命令。cron 配置文件则用于驱动 Vixie Cron 的 cron(8) 守护进程。每个用户都可以拥有自己的 crontab 文件

编辑定时任务

$ crontab -e

m h dom mon dow /path/to/command arg1 arg2

m字段：分钟 (0-59)

h字段：小时 (0-23)

dom字段：日期 (1-31)

mon字段：月份 (1-12 [12 代表 December])

dow字段：一周当中的某天 (0-7 [7 或 0 代表星期天])

/path/to/command - 计划执行的脚本或命令的名称

简单的 crontab 示例：

### 每隔 5 分钟运行一次 backupscript 脚本 ##

\*/5 \* \* \* \* /root/backupscript.sh

### 每天的凌晨 1 点运行 backupscript 脚本 ##

0 1 \* \* \* /root/backupscript.sh

### 每月的第一个凌晨 3:15 运行 backupscript 脚本 ##

15 3 1 \* \* /root/backupscript.sh

每隔1分钟运行一次/home/qlin/.../thread-ex.py,并且将output输出到out-e

\*/1 \* \* \* \* /home/qzlin/…/threading-ex.py >> /home/qzlin/…/out-threading-ex.txt

## GUI自动化

<python编程快速上手>p352

Pyautogui模块包含一些函数，可以模拟鼠标移动、按键和滚动鼠标滚轮

$pip install python-xlib

$apt-get install scrot python-tk python-dev

$pip install pyautogui

import pyautogui

pyautogui.PAUSE = 1 每个PyAutoGUI函数调用后，暂停一秒

自动防故障功能，鼠标移到屏幕左上角，导致pyautogui.FailSafeException异常

pyautogui.FAILSAFE = True

pyautogui.moveTo(x, y, duration=0.25) 移至(x,y),需0.25秒

pyautogui.moveRel(deltaX, deltaY, duration=0.25) 相对于当前位置

try:

while True:

x, y = pyautogui.position()

positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)

print(positionStr, end='')

print('\b' \* len(positionStr), end='', flush=True)

except KeyboardInterrupt:

print('\nDone.')

pyautogui.click(x, y)

pyautogui.dragRel(deltaX, deltaY, duration=0.25)

滚动鼠标：单位的意义在每个操作系统和应用上不一样，滚动发生在鼠标的当前位置。传递正整数表示向上滚动，传递负整数表示向下滚动

pyautogui.scroll(200)

屏幕快照

im = pyautogui.screenshot()

im.getpixel((x, y))

pyautogui.pixelMathesColor(50, 200, (130, 135, 144))

打字

pyautogui.click(x, y)

pyautogui.typewrite('Hello world')

pyautogui.hotkey('ctrl', 'c')

## 数据库

### Sqlite3

SQLite是一个进程中关系数据库，SQLite设计为嵌入在应用中，而不是作为一个单独的数据库服务器程序

$sudo apt-get install sqlite3 安装sqlite3

**todo\_schema.sql**

**sqlite-generate-db.py**

import os

import sqlite3

db\_filename = 'todo.db'

schema\_filename = 'todo\_schema.sql'

db\_is\_new = not os.path.exists(db\_filename)

with sqlite3.connect(db\_filename) as conn:

if db\_is\_new:

print 'Creating schema'

with open(schema\_filename, 'rt') as f:

schema = f.read()

conn.executescript(schema)

else:

print 'Database exists'

**sqlite-query.py**

import sqlite3

db\_filename = 'todo.db'

with sqlite3.connect(db\_filename) as conn:

conn.row\_factory = sqlite3.Row

cursor = conn.cursor()

query = """

select id, priority, details, status, deadline

from task

where project = :project\_name

order by deadline

"""

cursor.execute(query, {'project\_name': 'pymotw'})

for row in cursor.fetchall():

print '%2d {%d} %-25s [%-8s] (%s)' % (row['id'], row['priority'], row['details'], row['status'], row['deadline'])

事务性

with sqlite3.connect(db\_filename) as conn:

try:

cursor = conn.cursor()

cursor.execute("""...""")

cursor.execute("""...""")

execpt Exception, err:

conn.rollback() 回滚

else:

conn.commit() 保存

## 网络数据采集

### 网络请求和响应 --requests

$pip install requests

Requests模块是“人类“用的http模块，类似的模块有urllib，urllib2，httplib，httplib2

#### 7个http动作 --发送请求, 返回Response 对象

r = requests.get('https://github.com/timeline.json')

r = requests.post("http://httpbin.org/post")

r = requests.put("http://httpbin.org/put")

r = requests.delete("http://httpbin.org/delete")

r = requests.head("http://httpbin.org/get")

r = requests.options("http://httpbin.org/get")

#### 为URL传递参数

payload = {'key1': 'value1', 'key2': 'value2'}

r = requests.get("http://httpbin.org/get", params=payload)

r.url == http://httpbin.org/get?key2=value2&key1=value1

#### 定制请求头

url = 'https://api.github.com/some/endpoint'

payload = {'some': 'data'}

headers = {'content-type': 'application/json'}

r = requests.post(url, data=json.dumps(payload), headers=headers)

表单形式的POST请求

payload = {'key1': 'value1', 'key2': 'value2'}

r = requests.post("http://httpbin.org/post", data=payload)

POST一个多部分编码(Multipart-Encoded)的文件

url = 'http://httpbin.org/post'

files = {'file': open('report.xls', 'rb')}

r = requests.post(url, files=files)

Basic Authentication

requests.get('https://api.github.com/user', auth=('user', 'pass'))

OAuth 1 Authentication

auth = OAuth1('YOUR\_APP\_KEY', 'YOUR\_APP\_SECRET',

'USER\_OAUTH\_TOKEN', 'USER\_OAUTH\_TOKEN\_SECRET')

requests.get(url, auth=auth)

Proxies

proxies = {

'http': 'http://165.225.96.34:10015',

'https': ' http://165.225.96.34:10015',

}

requests.get('http://example.org', proxies=proxies)

#### 响应对象

r.headers 响应头

r.status\_code == requests.codes.ok 响应状态码

r.cookies Cookies

r.text 响应内容

r.content 二进制响应内容

from PIL import Image

from StringIO import StringIO

i = Image.open(StringIO(r.content))

r.json JSON响应内容

r.raw 原始响应内容

#### 保存响应内容

with open(filename, 'wb') as fd:

for chunk in r.iter\_content(chunk\_size):

fd.write(chunk)

### 解析HTML and XML --Beautiful Soup

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

$pip install beautifulsoup4

$pip install lxml HTML解析器 速度快

$pip install html5lib 解析方式与浏览器相同

from bs4 import BeautifulSoup

#### 创建BeautifulSoup对象

soup = BeautifulSoup(open("index.html"))

soup = BeautifulSoup("<html>data</html>")

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种: Tag , NavigableString , BeautifulSoup , Comment .

指定文档解析器

如果仅是想要解析HTML文档,只要用文档创建 BeautifulSoup 对象就可以了，Beautiful Soup会自动选择一个解析器来解析文档.但是还可以通过参数指定使用那种解析器来解析当前文档.

BeautifulSoup 第一个参数应该是要被解析的文档字符串或是文件句柄,第二个参数用来标识怎样解析文档.如果第二个参数为空,那么Beautiful Soup根据当前系统安装的库自动选择解析器,解析器的优先数序: lxml, html5lib, Python标准库.在下面两种条件下解析器优先顺序会变化:

要解析的文档是什么类型: 目前支持, “html”, “xml”, 和 “html5”

指定使用哪种解析器: 目前支持, “lxml”, “html5lib”, 和 “html.parser”

编码

任何HTML或XML文档都有自己的编码方式,比如ASCII 或 UTF-8,但是使用Beautiful Soup解析后,文档都被转换成了Unicode:

Beautiful Soup用了编码自动检测子库来识别当前文档编码并转换成Unicode编码

编码自动检测功能大部分时候都能猜对编码格式,但有时候也会出错.有时候即使猜测正确,也是在逐个字节的遍历整个文档后才猜对的,这样很慢.如果预先知道文档编码,可以设置编码参数来减少自动检查编码出错的概率并且提高文档解析速度.在创建 BeautifulSoup 对象的时候设置 from\_encoding 参数.

#### 标签Tag对象

Tag 对象与XML或HTML原生文档中的tag相同:

soup = BeautifulSoup('<b class="boldest">Extremely bold</b>')

tag = soup.b

tag中最重要的属性: name和attributes

str(tag) tag -> str

tag.name == 'b'

tag.attrs 字典

tag['class']

tag.string == u'Extremely bold'

unicode\_string = unicode(tag.string)

tag的 .contents tag的子节点 （列表）

tag.contents[0]

tag.descendants tag的所有子孙节点 （列表）

tag.strings 子孙字符串 （列表）

tag.stripped\_strings 去除多余空白的子孙字符串 （列表）

tag.parent 元素的父节点

tag.parents 元素的所有父辈节点 (列表）

tag.next\_sibling 和 .previous\_sibling 属性来查询兄弟节点

tag.next\_siblings 和 .previous\_siblings 属性可以对当前节点的兄弟节点迭代输出

tag.get\_text() 方法,这个方法获取到tag中包含的所有文版内容包括子孙tag中的内容

#### Tag查询

find\_all( name , attrs , recursive , text , \*\*kwargs )

soup.a 第一个<a>标签:

soup.find\_all('a') 所有的<a>标签

soup.find\_all(["a", "b"]) 所有<a>标签和<b>标签

soup.find\_all(id='link2')

soup.find\_all("a", attrs={"class": "sister"})

soup.find\_all(text=re.compile("Dormouse"))

soup.select("p nth-of-type(3)") 基于css选择器方法，返回Tag列表

Tag创建与替换

创建一个tag最好的方法是调用工厂方法 BeautifulSoup.new\_tag() :

tag.append(new\_tag)

PageElement.replace\_with() 方法移除文档树中的某段内容,并用新tag或文本节点替代它

PageElement.wrap() 方法可以对指定的tag元素进行包装

#### Tag美观打印

prettify() 方法将Beautiful Soup的文档树格式化后以Unicode编码输出,每个XML/HTML标签都独占一行

#### 常用场景：

从文档中找到所有<a>标签的链接

for link in soup.find\_all('a'):

print(link.get('href'))

# http://example.com/elsie

# http://example.com/lacie

# http://example.com/tillie

从文档中获取所有文字内容

print(soup.get\_text())

# The Dormouse's story

#

# ...

下载所有XKCD漫画

import requests, os, bs4

url ='http://xkcd.com'

os.makedirs('xkcd', exist\_ok=True)

while not url.endswith('#'):

res = requests.get(url)

res.raise\_for\_status() #如果下载文件出错，抛出异常

soup = bs4.BeautifulSoup(res.text)

comicElem = soup.select('#comic img')

if comicElem == []:

print('Could not find comic image.')

else:

comicUrl = 'http:' + comicElem[0].get('src')

res = requests.get(comicUrl)

res.raise\_for\_status()

即使页面是纯文本的，你也需要写入二进制数据，而不是文本数据，目的是为了保存该文本中的"unicode编码"

imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)), 'wb')

for chunk in res.iter\_content(100\*1000):

imageFile.write(chunk)

imageFile.close()

#Get the Prev button's url

prevLink = soup.select('a[rel="prev"]')[0]

url = 'http://xkcd.com' + prevLink.get('href')

用requests的响应text创建BeautifulSoup会有中英文乱码问题，建议使用如下

import urllib2, bs4

html = urllib2.urlopen(url).read()

soup = bs4.BeautifulSoup(html, 'lxml')

### 模拟浏览器操作

#### import webbrowser

webbrowser.open('http://inventwithpython.com/') 启动浏览器，打开url

selenium模块让Python直接控制浏览器，实际点击链接，填写登录信息, 与Requests and Beautiful Soup相比，Selenium允许你用高级得多的方式与网页交互。但因为它启动了Web浏览器，假如你只是想从网络上下载一些文件，会有点慢，并且难以在后台运行

具体见<Python编程快速上手—让繁琐工作自动化>p210

#### from selenium import webdriver

browser = webdriver.Firefox()

browser.get('http://inventwithpython.com')

查找

browser.find\_elements\_by\_css\_selector(selector)

如果页面上没有元素匹配该方法要查找的元素，selenium模块就会抛出NoSuchElement异常

点击

linkElem = browser.find\_element\_by\_link\_text('Read It Online')

linkElem.click()

填写并提交表单

browser.get('http://gmail.com')

emailElem = browser.find\_element\_by\_id('Email')

passwordElem = browser.find\_element\_by\_id('Passwd')

emailElem.send\_keys('lin.qizhong@gmail.com')

passwordElem.send\_keys('...')

passwordElem.submit() 在任何元素上调用submit()方法，等同于该元素所在表单的submit按钮

from selenium.webdriver.common.keys import Keys

htmlElem = browser.find\_element\_by\_tag\_name('html')

htmlElem.send\_keys(Keys.END)

htmlElem.send\_keys(Keys.HOME) 按下home键，从而页面滚动至顶

## web framework

### build-in web server to setup a web server

$python -m SimpleHTTPServer 8080 默认是8000

$curl localhost:8080

相当于ls .

$curl localhost:8080/static/data.txt

相当于cat static/data.txt

### Flask

具体见Python.docx

# R

统计计算和统计制图

CRAN: Comprehensive R Archive Network

## install R and RStudio

/etc/apt/sources.list

deb https://mirrors.tuna.tsinghua.edu.cn/CRAN/bin/linux/ubuntu trusty/

sudo apt-get update

sudo apt-get install r-base

//install RStudio

apt-get install libjpeg62

wget http://download1.rstudio.org/rstudio-0.98.1062-amd64.deb

sudo dpkg -i \*.deb

rm \*.deb

$which R R命令行方式

$R

$which Rscript R脚本运行方式

helloworld.r

#!/usr/bin/Rscript

print("Hello World")

$chmod u+x helloworld.r

$./helloworld.r

$which rstudio R IDE

$rstudio 启动RStudio环境

## <R语言初学者指南>

### R语法

>help(”boxplot“)

>library(MASS) 载入包

>q() 退出

>getwd()

>setwd(file="C:/.../") 改变当前目录

>rm(list = ls(all=TRUE) ) 删除当前所有变量

一般来说，R中的变量名最好使用大写字母开头，这样可以避免将它和一些内部函数名混淆

在R中使用内部函数前最好查看一下相应的帮助文件，以确保你知道这个函数如何处理缺失值

#表示注释

<-表示赋值

向量

Wing <- c(59, 55, 53.5, 55) c()表示连接(Concatenate)

Wing[1]

Wing[1:3]

Id <- rep(c(1, 2, 3, 4), each=8) 生成8个1,然后是8个2,...

Id <- rep(1:4, each=8)

Id <- rep(seq(from=1, to=4, by=1), each=8)

Head <- c(33, 22, 11, 44)

向量嵌套向量

Z <- cbind(Wing, Head) 以列的方式结合变量

Z[, 1] 第一列

Z[1:8, 1] 子数组

Z[2, ] 第二行

Size <- dim(Z) 返回维数(行，列）

Z2 <- rbind(Wing, Head) 以行的方式结合变量

矩阵

Dmat <- matrix(nrow=8, ncol=4) 矩阵

colnames(Dmat) <- c("Wing", "Tarsus", "Head", "Wt") 给矩阵加列名

Dmat[3:5, 2:3] 子矩阵

Dmat2 <- as.matrix(cbind(Wing, Head)) 生成矩阵

多维数组

a <- array(1:24, c(2, 3, 4))

4组两行三列

a[1, 2, 3] 访问第3组，第1行第2列元素

数组框

Dfrm <- data.frame(WC=Wing, TS=Tarsus, HD=Head, W=Wt, Wsq=sqrt(Wt)) 数据框

每一行表示一个样本的不同观察值

Dfrm$WC 访问列

Dfrm[ c(‘WC’, ‘W’) ] 访问数据框某几列

列表

Y <- list(x1 = c(1, 2,3),

x2 = c("a", "b"),

x3 = matrix(nrow=2, ncol=2)) 同python and javascript的list

Y$x1 访问元素

引入list函数的原因是因为几乎所有R中的函数（比如线性回归，广义线性模型，t-检验等等）的输出结果都是保存在列表中

条件指令

if (a) {

x < -1

} else {

x <- 2

}

ifelse (a, x <- 1, x < -2) 相当于a ? x=1 : x=2

循环

for (i in 1:27) {

YourFileName <- paste(i, ".jpg", sep="") 将变量连接为字符串

jpeg(file=YourFileName) 打开一个jpg文件

plot(x, y, ...)

dev.off() 关闭jpg文件

}

函数

函数中最后一行代码的结果将是返回信息, 也可以用return提前返回

Index.function <- function(Spec, Choice1="Sannon") {

if (Choice1 == "Richness") {

Index <- rowSums(Spec>0, na.rm=TRUE)

}

if (Choice1 == "Shannon") {

RS <- rowSums(Spec, na.rm=TRUE)

prop <- Spec / RS

Index <- -rowSums(prop\*log10(prop), na.rm=TRUE)

}

list(Index=Index, MyChoice=Choice1)

}

### 数据获取

以NA表示缺失值，一般最好以Excel中的第一列来识别样本单元，第一行作为变量名

Squid <- read.table(file="C:/RBook/squid.txt", header=TRUE) 将csv读入数据框中

write.table(Squid, file="Squid.txt", sep=",", quote=TRUE, append=FALSE, na="NA") 写csv

### 数据预处理 （一般针对数据框）

names(Squid) 返回变量名（列名）

str(Squid) 显示变量的属性

unique(Squid$Sex) 等价于levels(Squid$Sex)

attach(Squid)

Squid3 <- Squid[Sex==1 | Location==2, ] 过滤子集

Squid4 <- Squid[order(Sex), ] 排序

deattach(Squid)

合并数据框，Sample是两个数据框的相同变量

SquidMerged <- merge(Sq1, Sq2, by="Sample", all=TRUE)

变换列产生新列

Squid$fLocation <- factor(Squid$Location)

Squid$fSex <- factor(Squid$Sex, levels=c(1,2), labels=c("M", "F"))

在做任何统计分析之前，你应该将数据可视化

最后一部分data=Squid告诉R,变量在数据框 Squid中，从而lm()函数参数可以使用使用变量GSI,Location,...

有些函数没有data数据框，可以用Squid$GSI访问变量，若嫌麻烦，可以用attach and deattach限定上下文

M1 <- lm(GSI~factor(Location)+factor(Year), data=Squid)

attach(Squid）

...直接使用变量GSI, Location,...

deattach(Squid)

### 数据分析

tapply(y, x, FUN=mean) 根据x的不同水平对y使用FUN函数

sapply(y, FUN=mean)

lapply(y, FUN=mean) 对y的每一个变量使用FUN的函数

summary(y) 计算基本信息

y可以是一个变量，cbind命令的输出，或数据框

table(x,y) 计算列联表

### 数据可视化

plot(y, x) y对x的图形

不建议写成plot(Y~X, data=Veg)，因为Y~X符号是用来告诉R，Y作为X的函数。但并不是每个涉及变量的散点图都有一个因果关系

lines(x, y) 在已存在图形上添加线

order(x) 排序

M <- loess(y ~ x) 使用LOESS平滑

fitted(M) 得到拟合值

生成具有4个面板的图形窗口， mar选项指定每个图形周围空白的大小

op <- par(mfrow=c(2,2), mar=c(3,3,2,1))

pie(Cases) 饼图

pie3D(Cases)

par(op)

barplot(x) 条形图

boxplot(y) 盒形图

pairs(x) 生成多面板散点图

arrows(x1,y1, x2,y2) 绘制箭头

text(x,y, "hello") 在图形上添加文本

points(x, y) 在图形上添加点

legend("topleft", MyText, lty=c(1, 2, 3)) 添加图例

title(MyText) 添加标题

高维数据绘图

library(lattice)

xyplot(SAL ~ MyTime | factor(Station), ...) 散点图

~连接的变量分别表示y轴和x轴

|后条件变量是为了生成多面板

条件变量通常是一个因子，也可以是连续变量，当使用连续的变量作为条件变量时，它的每一个值在默认情况下都被理解为一个离散值，然而，此类变量通常具有很多的值，此时我们就需要将其分割为一些区间。

bwplot(SAL ~ factor(Month) | Area, ...) 盒形图

dotplot(factor(Month) ~ SAL | Station, ...) 点图

histogram(~ SAL | Station, ...) 直方图

barchart()

contourplot()

levelplot()

...

## <R and Ruby数据分析之旅>

$R

>install.packages('tree') 安装R程序包

>installed.packages() 查看已安装程序包

>library(tree) 使用程序包

因子factor

值可以分为名义型nominal, 顺序型ordinal 和连续型continuous

在R中，名义值和顺序值都用因子来表示

colors <- c('green', 'red', 'blue')

factor(colors) nominal

默认情况下，对于字符数据，因子各水平的排序是根据字母顺序创建的。但是，我们也可以在创建因子的时候指定各水平的顺序

results <- c('poor', 'average', 'good')

factor(results, order=TRUE, levels=results) ordinal

数据框

按列构造

league <- data.frame(team, home\_wins, home\_draws, home\_losses, away\_wins, away\_draws, away\_losses)

league[ c('team', 'home\_wins') ]

league$team

league$team[league$home\_wins > 8]

league$team[ league$away\_wins > league$home\_wins ]

等价于

with(league, team[away\_wins > home\_wins] ) with上下文管理

league <- merge(league, points, by='team') 合并数据框（添加列方式）

league[order(-league$pts, ]) 按字段pts降序

league <- rbind(league, league\_another) 合并数据框（添加行方式）

R包自带数据

install.packages('ggplot2')

library(ggplot2)

mpg

str(mpg)

summary(mpg)

从csv文件导入数据

epl <- read.csv('\*.csv')

从MySQL数据库导入数据

install.packages( c('DBI', 'RMySQL') )

con <- dbConnect(MySQL(), host='localhost', dbname='eps', user='root', password='root')

league <- dbGetQuery(con, 'select \* from league')

dbDisconnect(conn)

绘图

如果你将图表输出到文件，你可以使用pdf(), png(), jpeg(), bmp(), win.metafile()及其他函数来设置输出文件的格式。

pdf('some\_file\_name.pdf')

#do some chart plotting

dev.off()

R中绘图是逐层进行的。在绘制了图形之后（通过plot()函数），你可以继续添加新的图层，在其中包含新的信息

install.packages('ggplot2')

程序包ggplot2基本思想：复合多个图层，各层之是相互堆叠起来。

统计绘图实际上是从数据到几何对象的审美属性的一个映射过程，而这些几何对象应当可以进行统计变换，并且可以在坐标系统内画出。审美属性包括颜色、形状、大小，而几何对象类型可以是点、线条或柱形等性质

具体见p52

# Spark

## 《Spark快速大数据分析》

**Spark：批处理，迭代算法，交互式查询，流处理**



Spark Core实现了Spark的基本功能，包含任务调度，内存管理，错误恢复，与存储系统交互，RDD(Resilient distributed dataset:弹性分布式数据集)等

Spark SQL是Spark用来操作结构化数据的程序包，Spark SQL还支持开发者将SQL和传统的RDD编程的数据操作方式相结合

Spark Streaming是Spark提供的对实时数据进行流式计算的组件

MLib: 机器学习库

GraphX: 用来操作图的程序库，可以进行并行的图计算

数据科学家就是主要负责分析数据并建模的人。数据科学家有可能具备SQL、统计、预测建模（机器学习）等方面的经验，以及一定的使用python, matlab or R语言进行编程的能力

数据科学家在使用R或Pandas等**传统数据分析工具时所能处理的数据集受限于单机**，而有了**Spark**,就能**处理更大数据**规模的问题。

对工程师来说，Spark为开发用于集群并行执行的程序提供了一条捷径。通过封装，Spark不需要开发者关注如何在分布式系统上编程这样的复杂问题，也无需过多关注网络通信和程序容错性。Spark已经为工程师提供了足够的接口来实现常见的任务，以及对应用进行监视、审查和性能调优

**Hadoop MapReduce在迭代计算和交互计算的任务上表现得效率低下**。因此，**Spark从一开始就是为交互式查询和迭代算法设计**的，同时还支持内存和高效的容错机制

Shell工具类似R,Python, Scala, matlab, Bash等，只能使用单机的硬盘和内存来操作数据，而Spark shell可用来与分布式存储在许多机器的内存或者硬盘上的数据进行交互，并且处理过程的分发由Spark自动控制完成

在Spark中，我们通过对**分布式数据集**的操作来表达我们的计算意图，这些计算会自动地在集群上并行进行。这样的数据集被称为RDD, RDD是Spark对分布式数据和计算的基本抽象

驱动器程序Spark shell启动时自动创建SparkContext对象，即sc变量，从而可以创建RDD,在RDD上执行操作

驱动器程序一般要管理多个执行器节点，比如我们在集群上运行count()，那么不同的节点会统计文件的不同部分的行数。Spark会自动将函数如filter里的fun函数发到各个执行器节点上。这样你就可以在单一的驱动器程序中编程，并且让代码自动运行在多个节点上

### install spark

$wget <http://d3kbcqa49mib13.cloudfront.net/spark-2.0.0-bin-hadoop2.7.tgz>

$tar -xf spark…tgz

$cd spark…

### 命令行（交互式分析数据）

启动Spark中Python的shell

**$bin/pyspark**

// actions, return values

lines = sc.textFile("README.md")

lines.count()

lines.first()

//transformation, return a new RDD

pythonlines = lines.filter(lambda line: "Python" in line)

textFile.map(lambda line: len(line)).reduce(lambda a,b: a if (a>b) else b)

wordCounts = textFile.flatMap(lambda line: line.split()).

map(lambda word: (word, 1)).

reduceByKey(lambda a, b: a+b)

wordCounts.collect()

### 脚本（独立应用）

pi.py

from random import random

from pyspark import SparkContext

def sample(p):

x, y = random(), random()

return 1 if x\*x + y\*y < 1 else 0

NUM\_SAMPLES = 100000

sc = SparkContext("local", "Pi App") 通过集群URL和应用名初始化sc,然后同shell一样操作

count = sc.parallelize(xrange(0, NUM\_SAMPLES)).map(sample).reduce(lambda a,b: a+b)

print "Pi is roughly %f" % (4.0 \* count / NUM\_SAMPLES)

**$bin/spark-submit** pi.py

spark-submit脚本会帮我们引入python程序的spark依赖。这个脚本为Spark的PythonAPI配置好了运行环境

查看spark自带的范例

$tree examples/src/main/python

$cat examples/src/main/python/pi.py

$bin/spark-submit examples/src/main/python/pi.py

### RDD编程

RDD其实就是分布式的元素集合。在Spark中，对数据的所有操作不外乎**创建RDD、转化已有RDD以及调用RDD操作进行求值**。而在这一切背后，Spark会自动将RDD中的数据分发到集群上，并将操作并行化执行

Spark中的RDD就是一个不可变的分布式对象集合。每个RDD都被分为多个分区，这些分区运行在集群中的不同节点上

Spark只会惰性计算这些RDD。它们只有第一次在一个行动操作中用到时，才会真正计算

默认情况下，Spark的RDD会在你每次对它们进行行动操作时重新计算。可以使用RDD.persist()让Spark把这个RDD缓存下来，Spark会把RDD的内容保存到内存中（以分区方式存储到集群中的各机器上）。在实际操作中，你会经常用persist()来把数据的一部分读取到内存中，并反复查询这部分数据

每个Spark程序或shell会话都按如下方式工作：

1. 从外部数据创建出输入RDD
2. 使用转化操作对RDD进行转化，以定义新的RDD
3. **对需要被重用的中间结果RDD执行persist()操作**
4. 使用行动操作触发一次并行计算

创建ADD

lines = sc.textFile("README.md") 读取一个外部数据集

list = ["pands", "i like pandas"]

lines = sc.parallelize(list) 将已有集合转RDD

RDD转换操作

返回一个新的RDD，

惰性求值，

不会改变已有的数据，会返回一个全新的RDD

rdd.map(func)

rdd.flapMap(func)

rdd.filter(func)

rdd.distinct(func) 去重

rdd.sample(withReplacement, fraction, [seed]) 采样

rdd.union(other)

rdd.intersection(other)

rdd.subtract(other)

rdd.cartesian(other) 笛卡尔积

RDD行动操作 (触发并行计算，返回值)

rdd.collect() 返回RDD中的所有元素

rdd.count() RDD中的元素个数

rdd.countByValue() 各元素在RDD中出现的次数

rdd.take(num) 从RDD中返回num个元素

rdd.top(num) 从RDD中返回前num个元素

rdd.takeOrdered(num)(myOrdering)

rdd.takeSample(withReplacement, num, [seed])

rdd.reduce(func) 整合RDD中所有数据

rdd.fold(initValue)(func) 同reduce,只是需要初始值

rdd.aggregate(initValue)(seqOp, combOp)

rdd.foreach(func)

持久化（缓存）

如果简单地对RDD调用行动操作，Spark每次都会重算RDD以及它的所有依赖。这在迭代算法中消耗大，因为迭代算法常常会多次使用同一组数据

为了避免多次计算同一个RDD，可以让Spark对数据进行持久化。当我们让Spark持久化存储一个RDD时，计算出RDD的节点会分别保存它们所求出的分区数据

RDD不同的持久化级别：在Scala and Java中，默认情况下persist()会把数据以序列化的形式缓存在JVM的堆空间中。在Python中，我们会始终序列化要持久化存储的数据，所以持久化级别默认值就是以序列化后的对象存储在JVM堆空间中。

如果要缓存的数据太多，内存中放不下，Spark会自动利用最近最小使用LRU的缓存策略把最老的分区从内存中移除