

初识Python

Python简介

Python的历史

1. 1989年圣诞节：Guido von Rossum开始写Python语言的编译器。
2. 1991年2月：第一个Python编译器（同时也是解释器）诞生，它是用C语言实现的（后面），可以调用C语言的库函数。在最早的版本中，Python已经提供了对“类”，“函数”，“异常处理”等构造块的支持，还有对列表、字典等核心数据类型，同时支持以模块为基础来构造应用程序。
3. 1994年1月：Python 1.0正式发布。
4. 2000年10月16日：Python 2.0发布，增加了完整的[垃圾回收](#)，提供了对[Unicode](#)的支持。与此同时，Python的整个开发过程更加透明，社区对开发进度的影响逐渐扩大，生态圈开始慢慢形成。
5. 2008年12月3日：Python 3.0发布，它并不完全兼容之前的Python代码，不过因为目前还有不少公司在项目和运维中使用Python 2.x版本，所以Python 3.x的很多新特性后来也被移植到Python 2.6/2.7版本中。

目前我使用的Python 3.7.x的版本是在2018年发布的，Python的版本号分为三段，形如A.B.C。其中A表示大版本号，一般当整体重写，或出现不向后兼容的改变时，增加A；B表示功能更新，出现新功能时增加B；C表示小的改动（例如：修复了某个Bug），只要有修改就增加C。如果对Python的历史感兴趣，可以阅读名为[《Python简史》](#)的网络文章。

Python的优缺点

Python的优点很多，简单的可以总结为以下几点。

1. 简单明了，学习曲线低，比很多编程语言都容易上手。
2. 开放源代码，拥有强大的社区和生态圈，尤其是在数据分析和机器学习领域。
3. 解释型语言，天生具有平台可移植性，代码可以工作于不同的操作系统。
4. 对两种主流的编程范式（面向对象编程和函数式编程）都提供了支持。
5. 代码规范程度高，可读性强，适合有代码洁癖和强迫症的人群。

Python的缺点主要集中在以下几点。

1. 执行效率稍低，对执行效率要求高的部分可以由其他语言（如：C、C++）编写。
2. 代码无法加密，但是现在很多公司都不销售卖软件而是销售服务，这个问题会被弱化。
3. 在开发时可以选择的框架太多（如Web框架就有100多个），有选择的地方就有错误。

Python的应用领域

目前Python在Web应用后端开发、云基础设施建设、DevOps、网络数据采集（爬虫）、自动化测试、数据分析、机器学习等领域都有着广泛的应用。

安装Python解释器

想要开始Python编程之旅，首先得在自己使用的计算机上安装Python解释器环境，下面将以安装官方的Python解释器为例，讲解如何在不同的操作系统上安装Python环境。官方的Python解释器是用C语言实现的，也是使用最为广泛的Python解释器，通常称之为CPython。除此之外，Python解释器还有Java语言实现的Jython、C#语言实现的IronPython以及PyPy、Brython、Pyston等版本，有兴趣的读者可以自行了解。

Windows环境

可以在[Python官方网站](#)下载到Python的Windows安装程序（exe文件），需要注意的是如果在Windows 7环境下安装Python 3.x，需要先安装Service Pack 1补丁包（可以通过一些工具软件自动安装系统补丁的功能来安装），安装过程建议勾选“Add Python 3.x to PATH”（将Python 3.x添加到PATH环境变量）并选择自定义安装，在设置“Optional Features”界面最好将“pip”、“tcl/tk”、“Python test suite”等项全部勾选上。强烈建议选择自定义的安装路径并保证路径中没有中文。安装完成会看到“Setup was successful”的提示。如果稍后运行Python程序时，出现因为缺失一些动态链接库文件而导致Python解释器无法工作的问题，可以按照下面的方法加以解决。

如果系统显示api-ms-win-crt*.dll文件缺失，可以参照[《api-ms-win-crt*.dll缺失原因分析和解决方法》](#)一文讲解的方法进行处理或者直接在[微软官网](#)下载Visual C++ Redistributable for Visual Studio 2015文件进行修复；如果是因为更新Windows的DirectX之后导致某些动态链接库文件缺失问题，可以下载一个[DirectX修复工具](#)进行修复。

Linux环境

Linux环境自带了Python 2.x版本，但是如果要更新到3.x的版本，可以在[Python的官方网站](#)下载Python的源代码并通过源代码构建安装的方式进行安装，具体的步骤如下所示（以CentOS为例）。

1. 安装依赖库（因为没有这些依赖库可能在源代码构件安装时因为缺失底层依赖库而失败）。

```
yum -y install wget gcc zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel gdbm-devel db4-devel libpcap-devel xz-devel libffi-devel
```

2. 下载Python源代码并解压缩到指定目录。

```
wget https://www.python.org/ftp/python/3.7.6/Python-3.7.6.tar.xz
xz -d Python-3.7.6.tar.xz
tar -xvf Python-3.7.6.tar
```

3. 切换至Python源代码目录并执行下面的命令进行配置和安装。

```
cd Python-3.7.6
./configure --prefix=/usr/local/python37 --enable-optimizations
make && make install
```

4. 修改用户主目录下名为.bash_profile的文件，配置PATH环境变量并使其生效。

```
cd ~
vim .bash_profile
```

```
# ... 此处省略上面的代码 ...  
  
export PATH=$PATH:/usr/local/python37/bin  
  
# ... 此处省略下面的代码 ...
```

5. 激活环境变量。

```
source .bash_profile
```

macOS环境

macOS也自带了Python 2.x版本，可以通过[Python的官方网站](#)提供的安装文件（pkg文件）安装Python 3.x的版本。默认安装完成后，可以通过在终端执行`python`命令来启动2.x版本的Python解释器，启动3.x版本的Python解释器需要执行`python3`命令。

运行Python程序

确认Python的版本

可以在Windows的命令行提示符中键入下面的命令。

```
python --version
```

在Linux或macOS系统的终端中键入下面的命令。

```
python3 --version
```

当然也可以先输入`python`或`python3`进入交互式环境，再执行以下的代码检查Python的版本。

```
import sys  
  
print(sys.version_info)  
print(sys.version)
```

编写Python源代码

可以用文本编辑工具（推荐使用[Sublime](#)、[Visual Studio Code](#)等高级文本编辑工具）编写Python源代码并用`py`作为后缀名保存该文件，代码内容如下所示。

```
print('hello, world!')
```

运行程序

切换到源代码所在的目录并执行下面的命令，看看屏幕上是否输出了"hello, world!"。

```
python hello.py
```

或

```
python3 hello.py
```

代码中的注释

注释是编程语言的一个重要组成部分，用于在源代码中解释代码的作用从而增强程序的可读性和可维护性，当然也可以将源代码中不需要参与运行的代码段通过注释来去掉，这一点在调试程序的时候经常用到。注释在随源代码进入预处理器或编译时会被移除，不会在目标代码中保留也不会影响程序的执行结果。

1. 单行注释 - 以#和空格开头的部分
2. 多行注释 - 三个引号开头，三个引号结尾

```
"""
第一个Python程序 - hello, world!
向伟大的Dennis M. Ritchie先生致敬

Version: 0.1
Author: 骆昊
"""

print('hello, world!')
# print("你好, 世界! ")
```

Python开发工具

IDLE - 自带的集成开发工具

IDLE是安装Python环境时自带的集成开发工具，如下图所示。但是由于IDLE的用户体验并不是那么好所以很少在实际开发中被采用。



```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.

>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> print('hello, world!')
hello, world!
>>> 12345 * 54321
670592745
>>> |
```

Ln: 11 Col: 4

IPython - 更好的交互式编程工具

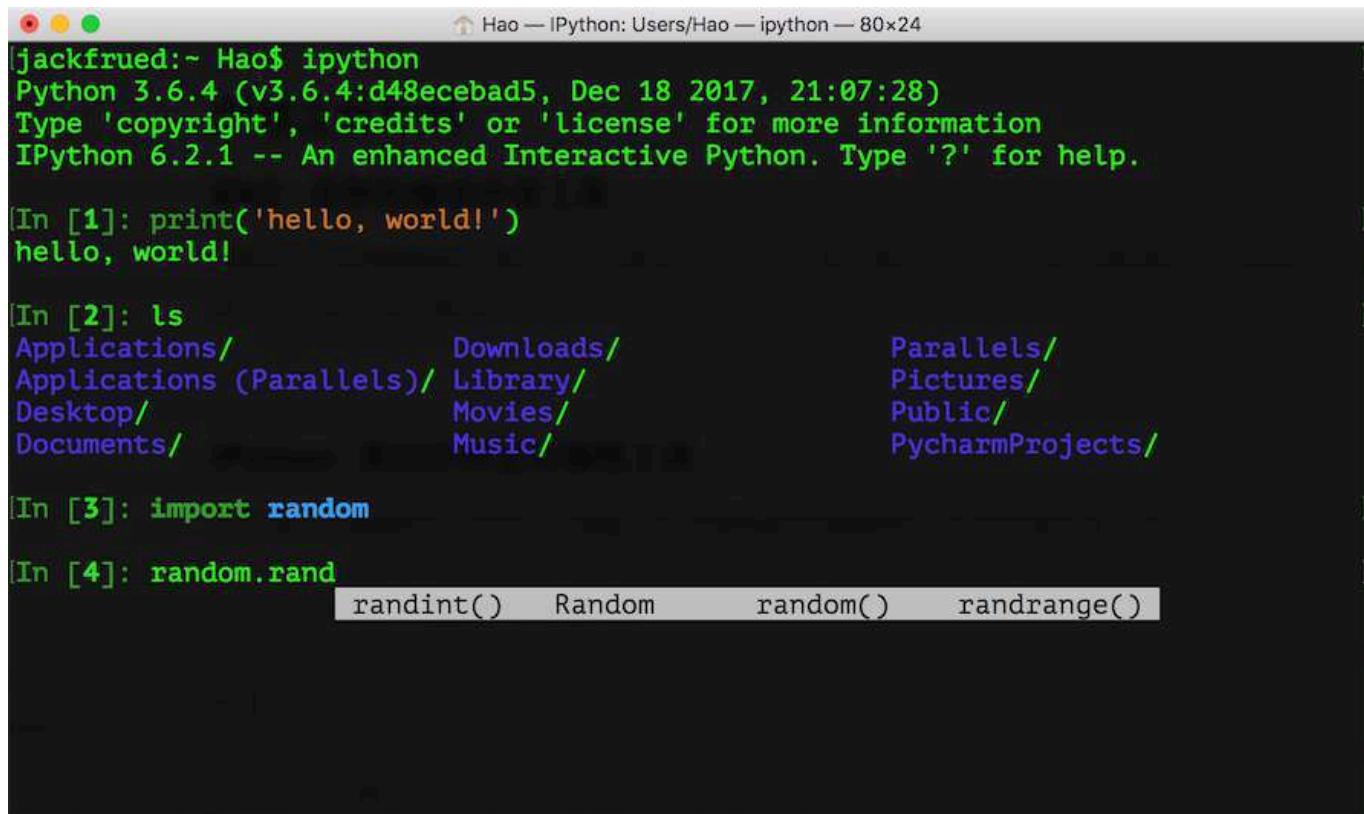
IPython是一种基于Python的交互式解释器。相较于原生的Python交互式环境，IPython提供了更为强大的编辑和交互功能。可以通过Python的包管理工具pip安装IPython，具体的操作如下所示。

```
pip install ipython
```

或

```
pip3 install ipython
```

安装成功后，可以通过下面的ipython命令启动IPython，如下图所示。



```
Hao — IPython: Users/Hao — ipython — 80x24
jackfrued:~ Hao$ ipython
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

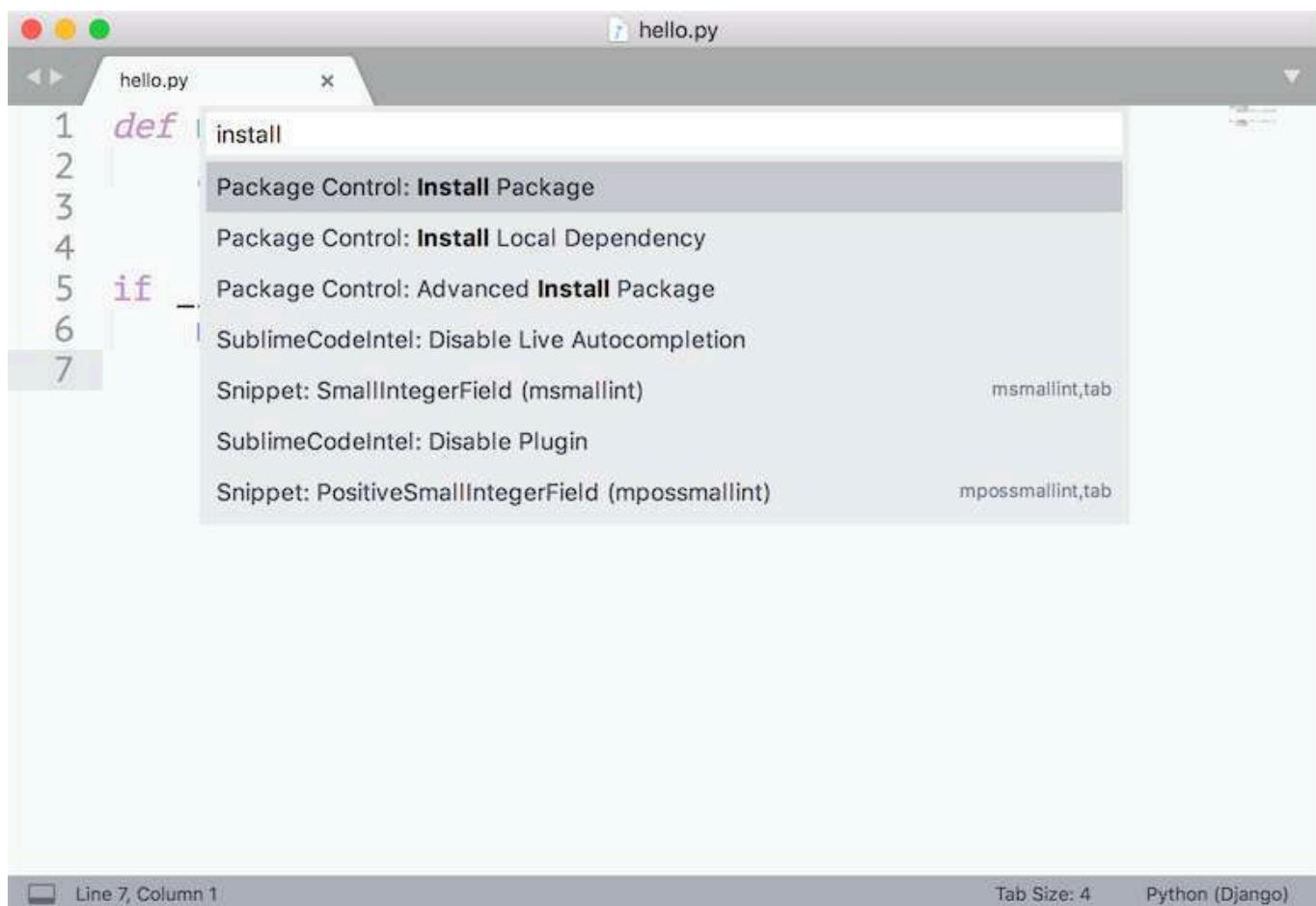
[In 1]: print('hello, world!')
hello, world!

[In 2]: ls
Applications/          Downloads/
Applications (Parallels)/ Library/
Desktop/               Movies/
Documents/              Music/
                                Parallels/
                                Pictures/
                                Public/
                                PycharmProjects/

[In 3]: import random

[In 4]: random.rand
        randint()  Random  random()  xrange()
```

Sublime Text - 高级文本编辑器



```
def install
if _:
    Package Control: Install Package
    Package Control: Install Local Dependency
    Package Control: Advanced Install Package
    SublimeCodeIntel: Disable Live Autocompletion
    Snippet: SmallIntegerField (msmallint)
    SublimeCodeIntel: Disable Plugin
    Snippet: PositiveSmallIntegerField (mpossmallint)
```

Line 7, Column 1 Tab Size: 4 Python (Django)

- 首先可以通过[官方网站](#)下载安装程序安装Sublime Text 3或Sublime Text 2。
- 安装包管理工具。

1. 通过快捷键Ctrl+`或者在View菜单中选择Show Console打开控制台，输入下面的代码。

- Sublime 3

```
import urllib.request,os;pf='Package Control.sublime-package';ipp=sublime.installed_packages_path();urllib.request.install_opener(urllib.request.build_opener(urllib.request.ProxyHandler()));open(os.path.join(ipp,pf), 'wb').write(urllib.request.urlopen('http://sublime.wbond.net/' + pf.replace(' ', '%20')).read())
```

- Sublime 2

```
import urllib2,os;pf='Package Control.sublime-package';ipp=sublime.installed_packages_path();os.makedirs(ipp)if not os.path.exists(ipp) else None;urllib2.install_opener(urllib2.build_opener(urllib2.ProxyHandler()));open(os.path.join(ipp,pf), 'wb').write(urllib2.urlopen('http://sublime.wbond.net/' + pf.replace(' ', '%20')).read());print('Please restart Sublime Text to finish installation')
```

2. 在浏览器中输入 <https://sublime.wbond.net/Package%20Control.sublime-package> 下载包管理工具的安装包，并找到安装Sublime目录下名为"Installed Packages"的目录，把刚才下载的文件放到这个文件加下，然后重启Sublime Text就搞定了。

- 安装插件。通过Preference菜单的Package Control或快捷键Ctrl+Shift+P打开命令面板，在面板中输入Install Package就可以找到安装插件的工具，然后再查找需要的插件。我们推荐大家安装以下几个插件：

- SublimeCodeIntel - 代码自动补全工具插件。
- Emmet - 前端开发代码模板插件。
- Git - 版本控制工具插件。
- Python PEP8 Autoformat - PEP8规范自动格式化插件。
- ConvertToUTF8 - 将本地编码转换为UTF-8。

说明：事实上[Visual Studio Code](#)可能是更好的选择，它不用花钱并提供了更为完整和强大的功能，有兴趣的读者可以自行研究。

PyCharm - Python开发神器

PyCharm的安装、配置和使用在[《玩转PyCharm》](#)进行了介绍，有兴趣的读者可以选择阅读。



练习

1. 在Python交互式环境中输入下面的代码并查看结果，请尝试将看到的内容翻译成中文。

```
import this
```

说明：输入上面的代码，在Python的交互式环境中可以看到Tim Peter撰写的“[Python之禅](#)”，里面讲述的道理不仅仅适用于Python，也适用于其他编程语言。

2. 学习使用turtle在屏幕上绘制图形。

说明：turtle是Python内置的一个非常有趣的模块，特别适合对计算机程序设计进行初体验的小伙伴，它最早是Logo语言的一部分，Logo语言是Wally Feurzig和Seymour Papert在1966发明的编程语言。

```
import turtle

turtle.pensize(4)
turtle.pencolor('red')

turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
```

```
turtle.mainloop()
```

提示：本章提供的代码中还有画国旗和画小猪佩奇的代码，有兴趣的读者请自行研究。

语言元素

指令和程序

计算机的硬件系统通常由五大部件构成，包括：运算器、控制器、存储器、输入设备和输出设备。其中，运算器和控制器放在一起就是我们通常所说的中央处理器，它的功能是执行各种运算和控制指令以及处理计算机软件中的数据。我们通常所说的程序实际上就是指令的集合，我们程序就是将一系列的指令按照某种方式组织到一起，然后通过这些指令去控制计算机做我们想让它做的事情。今天我们大多数时候使用的计算机，虽然它们的元器件做工越来越精密，处理能力越来越强大，但究其本质来说仍然属于“[冯·诺依曼结构](#)”的计算机。“冯·诺依曼结构”有两个关键点，一是指出要将存储设备与中央处理器分开，二是提出了将数据以二进制方式编码。二进制是一种“逢二进一”的计数法，跟我们人类使用的“逢十进一”的计数法没有实质性的区别，人类因为有十根手指所以使用了十进制（因为在数数时十根手指用完之后就只能进位了，当然凡事都有例外，玛雅人可能是因为长年光着脚的原因把脚趾头也算上了，于是他们使用了二十进制的计数法，在这种计数法的指导下玛雅人的历法就与我们平常使用的历法不一样，而按照玛雅人的历法，2012年是上一个所谓的“太阳纪”的最后一年，而2013年则是新的“太阳纪”的开始，后来这件事情被以讹传讹的方式误传为“2012年是玛雅人预言的世界末日”这种荒诞的说法，今天我们可以大胆的猜测，玛雅文明之所以发展缓慢估计也与使用了二十进制有关）。对于计算机来说，二进制在物理器件上来说是最容易实现的（高电压表示1，低电压表示0），于是在“[冯·诺依曼结构](#)”的计算机都使用了二进制。虽然我们并不需要每个程序员都能够使用二进制的思维方式来工作，但是了解二进制以及它与我们生活中的十进制之间的转换关系，以及二进制与八进制和十六进制的转换关系还是有必要的。如果你对这一点不熟悉，可以自行使用[维基百科](#)或者[百度百科](#)科普一下。

说明：近期关于[量子计算机](#)的研究已经被推到了风口浪尖，量子计算机基于量子力学进行运算，使用量子瞬移的方式来传递信息。2018年6月，Intel宣布开发出新款量子芯片并通过了在接近绝对零度环境下的测试；2019年，IBM和Google都推出了自己的量子计算机。

变量和类型

在程序设计中，变量是一种存储数据的载体。计算机中的变量是实际存在的数据或者说是存储器中存储数据的一块内存空间，变量的值可以被读取和修改，这是所有计算和控制的基础。计算机能处理的数据有很多种类型，除了数值之外还可以处理文本、图形、音频、视频等各种各样的数据，那么不同的数据就需要定义不同的存储类型。Python中的数据类型很多，而且也允许我们自定义新的数据类型（这一点在后面会讲到），我们先介绍几种常用的数据类型。

- 整型：Python中可以处理任意大小的整数（Python 2.x中有int和long两种类型的整数，但这种区分对Python来说意义不大，因此在Python 3.x中整数只有int这一种了），而且支持二进制（如0b100，换算成十进制是4）、八进制（如0o100，换算成十进制是64）、十进制（100）和十六进制（0x100，换算成十进制是256）的表示法。
- 浮点型：浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的，浮点数除了数学写法（如123.456）之外还支持科学计数法（如1.23456e2）。
- 字符串型：字符串是以单引号或双引号括起来的任意文本，比如'hello'和"hello"，字符串还有原始字符串表示法、字节字符串表示法、Unicode字符串表示法，而且可以书写成多行的形式（用三个单引号或三个双引号开头，三个单引号或三个双引号结尾）。
- 布尔型：布尔值只有True、False两种值，要么是True，要么是False，在Python中，可以直接用True、False表示布尔值（请注意大小写），也可以通过布尔运算计算出来（例如3 < 5会产生布尔值True，而2 == 1会产生布尔值False）。
- 复数型：形如3+5j，跟数学上的复数表示一样，唯一不同的是虚部的1换成了j。实际上，这个类型并不常用，大家了解一下就可以了。

变量命名

对于每个变量我们需要给它取一个名字，就如同我们每个人都有属于自己的响亮的名字一样。在Python中，变量命名需要遵循以下这些必须遵守硬性规则和强烈建议遵守的非硬性规则。

- 硬性规则：
 - 变量名由字母（广义的Unicode字符，不包括特殊字符）、数字和下划线构成，数字不能开头。
 - 大小写敏感（大写的a和小写的A是两个不同的变量）。
 - 不要跟关键字（有特殊含义的单词，后面会讲到）和系统保留字（如函数、模块等的名字）冲突。
- PEP 8要求：
 - 用小写字母拼写，多个单词用下划线连接。
 - 受保护的实例属性用单个下划线开头（后面会讲到）。
 - 私有的实例属性用两个下划线开头（后面会讲到）。

当然，作为一个专业的程序员，给变量（事实上应该是所有的标识符）命名时做到见名知意也是非常重要的。

变量的使用

下面通过几个例子来说明变量的类型和变量使用。

```
"""
使用变量保存数据并进行加减乘除运算
```

```
Version: 0.1
Author: 骆昊
"""

a = 321
b = 12
print(a + b)      # 333
print(a - b)      # 309
print(a * b)      # 3852
print(a / b)       # 26.75
```

在Python中可以使用type函数对变量的类型进行检查。程序设计中函数的概念跟数学上函数的概念是一致的，数学上的函数相信大家并不陌生，它包括了函数名、自变量和因变量。如果暂时不理解这个概念也不要紧，我们会在后续的章节中专门讲解函数的定义和使用。

```
"""
使用type()检查变量的类型
```

```
Version: 0.1
Author: 骆昊
"""

a = 100
b = 12.345
c = 1 + 5j
d = 'hello, world'
e = True
```

```
print(type(a))    # <class 'int'>
print(type(b))    # <class 'float'>
print(type(c))    # <class 'complex'>
print(type(d))    # <class 'str'>
print(type(e))    # <class 'bool'>
```

可以使用Python中内置的函数对变量类型进行转换。

- `int()`: 将一个数值或字符串转换成整数, 可以指定进制。
- `float()`: 将一个字符串转换成浮点数。
- `str()`: 将指定的对象转换成字符串形式, 可以指定编码。
- `chr()`: 将整数转换成该编码对应的字符串 (一个字符)。
- `ord()`: 将字符串 (一个字符) 转换成对应的编码 (整数)。

下面的代码通过键盘输入两个整数来实现对两个整数的算术运算。

```
"""
使用input()函数获取键盘输入(字符串)
使用int()函数将输入的字符串转换成整数
使用print()函数输出带占位符的字符串

Version: 0.1
Author: 骆昊
"""

a = int(input('a = '))
b = int(input('b = '))
print('%d + %d = %d' % (a, b, a + b))
print('%d - %d = %d' % (a, b, a - b))
print('%d * %d = %d' % (a, b, a * b))
print('%d / %d = %f' % (a, b, a / b))
print('%d // %d = %d' % (a, b, a // b))
print('%d %% %d = %d' % (a, b, a % b))
print('%d ** %d = %d' % (a, b, a ** b))
```

说明: 上面的print函数中输出的字符串使用了占位符语法, 其中`%d`是整数的占位符, `%f`是小数的占位符, `%%`表示百分号 (因为百分号代表了占位符, 所以带占位符的字符串中要表示百分号必须写成`%%`), 字符串之后的`%`后面跟的变量值会替换掉占位符然后输出到终端中, 运行上面的程序, 看看程序执行结果就明白啦。

运算符

Python支持多种运算符, 下表大致按照优先级从高到低的顺序列出了所有的运算符, 运算符的优先级指的是多个运算符同时出现时, 先做什么运算然后再做什么运算。除了我们之前已经用过的赋值运算符和算术运算符, 我们稍后会陆续讲到其他运算符的使用。

运算符	描述
<code>[] [:]</code>	下标, 切片

运算符	描述
<code>**</code>	指数
<code>~ + -</code>	按位取反, 正负号
<code>* / % //</code>	乘, 除, 模, 整除
<code>+ -</code>	加, 减
<code>>> <<</code>	右移, 左移
<code>&</code>	按位与
<code>^ \ </code>	按位异或, 按位或
<code><= < > >=</code>	小于等于, 小于, 大于, 大于等于
<code>== !=</code>	等于, 不等于
<code>is is not</code>	身份运算符
<code>in not in</code>	成员运算符
<code>not or and</code>	逻辑运算符
<code>= += -= *= /= %= /= **= &= = ^= >= <=</code>	(复合) 赋值运算符

说明: 在实际开发中, 如果搞不清楚运算符的优先级, 可以使用括号来确保运算的执行顺序。

赋值运算符

赋值运算符应该是最为常见的运算符, 它的作用是将右边的值赋给左边的变量。下面的例子演示了赋值运算符和复合赋值运算符的使用。

```
"""
赋值运算符和复合赋值运算符

Version: 0.1
Author: 骆昊
"""

a = 10
b = 3
a += b      # 相当于: a = a + b
a *= a + 2  # 相当于: a = a * (a + 2)
print(a)    # 算一下这里会输出什么
```

比较运算符和逻辑运算符

比较运算符有的地方也称为关系运算符, 包括`==`、`!=`、`<`、`>`、`<=`、`>=`, 我相信没有什么好解释的, 大家一看就能懂, 唯一需要提醒的是比较相等用的是`==`, 请注意这个地方是两个等号, 因为`=`是赋值运算符, 我们在上面刚刚讲到过, `==`才是比较相等的比较运算符。比较运算符会产生布尔值, 要么是`True`要么是`False`。

逻辑运算符有三个，分别是`and`、`or`和`not`。`and`字面意思是“而且”，所以`and`运算符会连接两个布尔值，如果两个布尔值都是`True`，那么运算的结果就是`True`；左右两边的布尔值有一个是`False`，最终的运算结果就是`False`。相信大家已经想到了，如果`and`左边的布尔值是`False`，不管右边的布尔值是什么，最终的结果都是`False`，所以在做运算的时候右边的值会被跳过（短路处理），这也就意味着在`and`运算符左边为`False`的情况下，右边的表达式根本不会执行。`or`字面意思是“或者”，所以`or`运算符也会连接两个布尔值，如果两个布尔值有任意一个是`True`，那么最终的结果就是`True`。当然，`or`运算符也是有短路功能的，在它左边的布尔值为`True`的情况下，右边的表达式根本不会执行。`not`运算符的后面会跟上一个布尔值，它的作用是得到与该布尔值相反的值，也就是说，后面的布尔值如果是`True`运算结果就是`False`，而后面的布尔值如果是`False`则运算结果就是`True`。

```
"""
```

比较运算符和逻辑运算符的使用

```
Version: 0.1
Author: 骆昊
"""

flag0 = 1 == 1
flag1 = 3 > 2
flag2 = 2 < 1
flag3 = flag1 and flag2
flag4 = flag1 or flag2
flag5 = not (1 != 2)
print('flag0 =', flag0)      # flag0 = True
print('flag1 =', flag1)      # flag1 = True
print('flag2 =', flag2)      # flag2 = False
print('flag3 =', flag3)      # flag3 = False
print('flag4 =', flag4)      # flag4 = True
print('flag5 =', flag5)      # flag5 = False
```

说明：比较运算符的优先级高于赋值运算符，所以`flag0 = 1 == 1`先做`1 == 1`产生布尔值`True`，再将这个值赋值给变量`flag0`。`print`函数可以输出多个值，多个值之间可以用`,`进行分隔，输出的内容之间默认以空格分开。

练习

练习1：华氏温度转换为摄氏温度。

提示：华氏温度到摄氏温度的转换公式为： $C = (F - 32) \div 1.8$ 。

参考答案：

```
"""
```

将华氏温度转换为摄氏温度

```
Version: 0.1
Author: 骆昊
"""

f = float(input('请输入华氏温度：'))
```

```
c = (f - 32) / 1.8
print('%.1f华氏度 = %.1f摄氏度' % (f, c))
```

说明：在使用print函数输出时，也可以对字符串内容进行格式化处理，上面print函数中的字符串`%.1f`是一个占位符，稍后会由一个float类型的变量值替换掉它。同理，如果字符串中有`%d`，后面可以用一个int类型的变量值替换掉它，而`%s`会被字符串的值替换掉。除了这种格式化字符串的方式外，还可以用下面的方式来格式化字符串，其中`{f:.1f}`和`{c:.1f}`可以先看成是`{f}`和`{c}`，表示输出时会用变量`f`和变量`c`的值替换掉这两个占位符，后面的`:.1f`表示这是一个浮点数，小数点后保留1位有效数字。

```
print(f'{f:.1f}华氏度 = {c:.1f}摄氏度')
```

练习2：输入圆的半径计算计算周长和面积。

参考答案：

```
"""
输入半径计算圆的周长和面积

Version: 0.1
Author: 骆昊
"""

radius = float(input('请输入圆的半径: '))
perimeter = 2 * 3.1416 * radius
area = 3.1416 * radius * radius
print('周长: %.2f' % perimeter)
print('面积: %.2f' % area)
```

练习3：输入年份判断是不是闰年。

参考答案：

```
"""
输入年份 如果是闰年输出True 否则输出False

Version: 0.1
Author: 骆昊
"""

year = int(input('请输入年份: '))
# 如果代码太长写成一行不便于阅读 可以使用\对代码进行折行
is_leap = year % 4 == 0 and year % 100 != 0 or \
          year % 400 == 0
print(is_leap)
```

说明：比较运算符会产生布尔值，而逻辑运算符`and`和`or`会对这些布尔值进行组合，最终也是得到一个布尔值，闰年输出`True`，平年输出`False`。

分支结构

应用场景

迄今为止，我们写的Python代码都是一条一条语句顺序执行，这种代码结构通常称之为顺序结构。然而仅有顺序结构并不能解决所有的问题，比如我们设计一个游戏，游戏第一关的通关条件是玩家获得1000分，那么在完成本局游戏后，我们要根据玩家得到分数来决定究竟是进入第二关，还是告诉玩家“Game Over”，这里就会产生两个分支，而且这两个分支只有一个会被执行。类似的场景还有很多，我们将这种结构称之为“分支结构”或“选择结构”。给大家一分钟的时间，你应该可以想到至少5个以上这样的例子，赶紧试一试。

if语句的使用

在Python中，要构造分支结构可以使用`if`、`elif`和`else`关键字。所谓**关键字**就是有特殊含义的单词，像`if`和`else`就是专门用于构造分支结构的关键字，很显然你不能够使用它作为变量名（事实上，用作其他的标识符也是不可以）。下面的例子中演示了如何构造一个分支结构。

```
"""
用户身份验证

Version: 0.1
Author: 骆昊
"""

username = input('请输入用户名: ')
password = input('请输入口令: ')
# 用户名是admin且密码是123456则身份验证成功否则身份验证失败
if username == 'admin' and password == '123456':
    print('身份验证成功!')
else:
    print('身份验证失败!')
```

需要说明的是和C/C++、Java等语言不同，Python中没有用花括号来构造代码块而是**使用了缩进的方式来表示代码的层次结构**，如果`if`条件成立的情况下需要执行多条语句，只要保持多条语句具有相同的缩进就可以了。换句话说**连续的代码如果又保持了相同的缩进那么它们属于同一个代码块**，相当于是一个执行的整体。**缩进**可以使用任意数量的空格，但通常使用4个空格，建议大家**不要使用制表键或者设置你的代码编辑工具自动将制表键变成4个空格**。

当然如果要构造出更多的分支，可以使用`if...elif...else...`结构或者嵌套的`if...else...`结构，下面的代码演示了如何利用多分支结构实现分段函数求值。

$$f(x) = \begin{cases} 3x - 5 & (x > 1) \\ x + 2 & (-1 \leq x \leq 1) \\ 5x + 3 & (x < -1) \end{cases}$$

```
"""
分段函数求值
```

```
3x - 5 (x > 1)
```

```

f(x) = x + 2  (-1 <= x <= 1)
      5x + 3  (x < -1)

Version: 0.1
Author: 骆昊
"""

x = float(input('x = '))
if x > 1:
    y = 3 * x - 5
elif x >= -1:
    y = x + 2
else:
    y = 5 * x + 3
print('f(%.2f) = %.2f' % (x, y))

```

当然根据实际开发的需要，分支结构是可以嵌套的，例如判断是否通关以后还要根据你获得的宝物或者道具的数量对你的表现给出等级（比如点亮两颗或三颗星星），那么我们就需要在`if`的内部构造出一个新的分支结构，同理`elif`和`else`中也可以再构造新的分支，我们称之为嵌套的分支结构，也就是说上面的代码也可以写成下面的样子。

```

"""
分段函数求值
3x - 5  (x > 1)
f(x) = x + 2  (-1 <= x <= 1)
      5x + 3  (x < -1)

Version: 0.1
Author: 骆昊
"""

x = float(input('x = '))
if x > 1:
    y = 3 * x - 5
else:
    if x >= -1:
        y = x + 2
    else:
        y = 5 * x + 3
print('f(%.2f) = %.2f' % (x, y))

```

说明：大家可以自己感受一下这两种写法到底是哪一种更好。在之前我们提到的Python之禅中有这么一句话“Flat is better than nested.”，之所以提倡代码“扁平化”是因为嵌套结构的嵌套层次多了之后会严重的影响代码的可读性，所以能使用扁平化的结构时就不要使用嵌套。

练习

练习1：英制单位英寸与公制单位厘米互换。

参考答案：

```
"""
英制单位英寸和公制单位厘米互换

Version: 0.1
Author: 骆昊
"""

value = float(input('请输入长度: '))
unit = input('请输入单位: ')
if unit == 'in' or unit == '英寸':
    print('%f英寸 = %f厘米' % (value, value * 2.54))
elif unit == 'cm' or unit == '厘米':
    print('%f厘米 = %f英寸' % (value, value / 2.54))
else:
    print('请输入有效的单位')
```

练习2：百分制成绩转换为等级制成绩。

要求：如果输入的成绩在90分以上（含90分）输出A；80分-90分（不含90分）输出B；70分-80分（不含80分）输出C；60分-70分（不含70分）输出D；60分以下输出E。

参考答案：

```
"""
百分制成绩转换为等级制成绩

Version: 0.1
Author: 骆昊
"""

score = float(input('请输入成绩: '))
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
elif score >= 60:
    grade = 'D'
else:
    grade = 'E'
print('对应的等级是:', grade)
```

练习3：输入三条边长，如果能构成三角形就计算周长和面积。

参考答案：

```
"""
```

判断输入的边长能否构成三角形，如果能则计算出三角形的周长和面积

Version: 0.1

Author: 骆昊

```
"""
```

```
a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))
if a + b > c and a + c > b and b + c > a:
    print('周长: %f' % (a + b + c))
    p = (a + b + c) / 2
    area = (p * (p - a) * (p - b) * (p - c)) ** 0.5
    print('面积: %f' % (area))
else:
    print('不能构成三角形')
```

说明：上面使用的通过边长计算三角形面积的公式叫做[海伦公式](#)。

循环结构

应用场景

我们在写程序的时候，一定会遇到需要重复执行某条或某些指令的场景。例如用程序控制机器人踢足球，如果机器人持球而且还没有进入射门范围，那么我们就要一直发出让机器人向球门方向移动的指令。在这个场景中，让机器人向球门方向移动就是一个需要重复的动作，当然这里还会用到上一课讲的分支结构来判断机器人是否持球以及是否进入射门范围。再举一个简单的例子，如果要实现每隔1秒中在屏幕上打印一次“hello, world”并持续打印一个小时，我们肯定不能够直接把 `print('hello, world')` 这句代码写3600遍，这里同样需要循环结构。

循环结构就是程序中控制某条或某些指令重复执行的结构。在Python中构造循环结构有两种做法，一种是 `for-in` 循环，一种是 `while` 循环。

for-in循环

如果明确的知道循环执行的次数或者要对一个容器进行迭代（后面会讲到），那么我们推荐使用 `for-in` 循环，例如下面代码中计算1~100求和的结果 ($\sum_{n=1}^{100} n$)。

```
"""
用for循环实现1~100求和
Version: 0.1
Author: 骆昊
"""

sum = 0
for x in range(101):
    sum += x
print(sum)
```

需要说明的是上面代码中的 `range(1, 101)` 可以用来构造一个从1到100的范围，当我们把这个范围放到 `for-in` 循环中，就可以通过前面的循环变量 `x` 依次取出从1到100的整数。当然，`range` 的用法非常灵活，下面给出了一个例子：

- `range(101)`：可以用来产生0到100范围的整数，需要注意的是取不到101。
- `range(1, 101)`：可以用来产生1到100范围的整数，相当于前面是闭区间后面是开区间。
- `range(1, 101, 2)`：可以用来产生1到100的奇数，其中2是步长，即每次数值递增的值。
- `range(100, 0, -2)`：可以用来产生100到1的偶数，其中-2是步长，即每次数字递减的值。

知道了这一点，我们可以用下面的代码来实现1~100之间的偶数求和。

```
"""
用for循环实现1~100之间的偶数求和
Version: 0.1
Author: 骆昊
"""


```

```
sum = 0
for x in range(2, 101, 2):
    sum += x
print(sum)
```

当然，也可以通过在循环中使用分支结构的方式来实现相同的功能，代码如下所示。

```
"""
用for循环实现1~100之间的偶数求和
```

```
Version: 0.1
Author: 骆昊
"""
```

```
sum = 0
for x in range(1, 101):
    if x % 2 == 0:
        sum += x
print(sum)
```

说明：相较于上面直接跳过奇数的做法，下面这种做法很明显并不是很好的选择。

while循环

如果要构造不知道具体循环次数的循环结构，我们推荐使用`while`循环。`while`循环通过一个能够产生或转换出`bool`值的表达式来控制循环，表达式的值为`True`则继续循环；表达式的值为`False`则结束循环。

下面我们通过一个“猜数字”的小游戏来看看如何使用`while`循环。猜数字游戏的规则是：计算机出一个1到100之间的随机数，玩家输入自己猜的数字，计算机给出对应的提示信息（大一点、小一点或猜对了），如果玩家猜中了数字，计算机提示用户一共猜了多少次，游戏结束，否则游戏继续。

```
"""
猜数字游戏
```

```
Version: 0.1
Author: 骆昊
"""

import random

answer = random.randint(1, 100)
counter = 0
while True:
    counter += 1
    number = int(input('请输入: '))
    if number < answer:
        print('大一点')
    elif number > answer:
        print('小一点')
```

```
else:  
    print('恭喜你猜对了!')  
    break  
print('你总共猜了%d次' % counter)  
if counter > 7:  
    print('你的智商余额明显不足')
```

上面的代码中使用了**break**关键字来提前终止循环，需要注意的是**break**只能终止它所在的那个循环，这一点在使用嵌套的循环结构（下面会讲到）需要引起注意。除了**break**之外，还有另一个关键字是**continue**，它可以用来放弃本次循环后续的代码直接让循环进入下一轮。

和分支结构一样，循环结构也是可以嵌套的，也就是说在循环中还可以构造循环结构。下面的例子演示了如何通过嵌套的循环来输出一个九九乘法表。

```
'''  
输出乘法口诀表(九九表)  
  
Version: 0.1  
Author: 骆昊  
'''  
  
for i in range(1, 10):  
    for j in range(1, i + 1):  
        print('%d*%d=%d' % (i, j, i * j), end='\t')  
    print()
```

练习

练习1：输入一个正整数判断是不是素数。

提示：素数指的是只能被1和自身整除的大于1的整数。

参考答案：

```
'''  
输入一个正整数判断它是不是素数  
  
Version: 0.1  
Author: 骆昊  
Date: 2018-03-01  
'''  
  
from math import sqrt  
  
num = int(input('请输入一个正整数: '))  
end = int(sqrt(num))  
is_prime = True  
for x in range(2, end + 1):  
    if num % x == 0:  
        is_prime = False
```

```
        break
if is_prime and num != 1:
    print('%d是素数' % num)
else:
    print('%d不是素数' % num)
```

练习2：输入两个正整数，计算它们的最大公约数和最小公倍数。

提示：两个数的最大公约数是两个数的公共因子中最大的那个数；两个数的最小公倍数则是能够同时被两个数整除的最小的那个数。

参考答案：

```
"""
输入两个正整数计算它们的最大公约数和最小公倍数

Version: 0.1
Author: 骆昊
Date: 2018-03-01
"""

x = int(input('x = '))
y = int(input('y = '))
# 如果x大于y就交换x和y的值
if x > y:
    # 通过下面的操作将y的值赋给x，将x的值赋给y
    x, y = y, x
# 从两个数中较小的数开始做递减的循环
for factor in range(x, 0, -1):
    if x % factor == 0 and y % factor == 0:
        print('%d和%d的最大公约数是%d' % (x, y, factor))
        print('%d和%d的最小公倍数是%d' % (x, y, x * y // factor))
        break
```

练习3：打印如下所示的三角形图案。

```
*
```

```
**
```

```
***
```

```
****
```

```
*****
```

```
*
```

```
**
```

```
***
```

```
****  
*****
```

```
*  
***  
*****  
*****  
*****
```

参考答案：

```
'''
```

打印三角形图案

Version: 0.1

Author: 骆昊

```
'''
```

```
row = int(input('请输入行数: '))
for i in range(row):
    for _ in range(i + 1):
        print('*', end=' ')
    print()
```

```
for i in range(row):
    for j in range(row):
        if j < row - i - 1:
            print(' ', end=' ')
        else:
            print('*', end=' ')
    print()
```

```
for i in range(row):
    for _ in range(row - i - 1):
        print(' ', end=' ')
    for _ in range(2 * i + 1):
        print('*', end=' ')
    print()
```

构造程序逻辑

学完前面的几个章节后，我觉得有必要在这里带大家做一些练习来巩固之前所学的知识，虽然迄今为止我们学习的内容只是Python的冰山一角，但是这些内容已经足够我们来构建程序中的逻辑。对于编程语言的初学者来说，在学习了Python的核心语言元素（变量、类型、运算符、表达式、分支结构、循环结构等）之后，必须做的一件事情就是尝试用所学知识去解决现实中的问题，换句话说就是锻炼自己把用人类自然语言描述的算法（解决问题的方法和步骤）翻译成Python代码的能力，而这件事情必须通过大量的练习才能达成。

我们在本章为大家整理了一些经典的案例和习题，希望通过这些例子，一方面帮助大家巩固之前所学的Python知识，另一方面帮助大家了解如何建立程序中的逻辑以及如何运用一些简单的算法解决现实中的问题。

经典的例子

1. 寻找水仙花数。

说明：水仙花数也被称为超完全数字不变数、自恋数、自幂数、阿姆斯特朗数，它是一个3位数，该数字每个位上数字的立方之和正好等于它本身，例如： $1^3 + 5^3 + 3^3 = 153$ 。

```
"""
找出所有水仙花数

Version: 0.1
Author: 骆昊
"""

for num in range(100, 1000):
    low = num % 10
    mid = num // 10 % 10
    high = num // 100
    if num == low ** 3 + mid ** 3 + high ** 3:
        print(num)
```

在上面的代码中，我们通过整除和求模运算分别找出了一个三位数的个位、十位和百位，这种小技巧在实际开发中还是常用的。用类似的方法，我们还可以实现将一个正整数反转，例如：将12345变成54321，代码如下所示。

```
"""
正整数的反转

Version: 0.1
Author: 骆昊
"""

num = int(input('num = '))
reversed_num = 0
while num > 0:
    reversed_num = reversed_num * 10 + num % 10
```

```
num /= 10
print(reversed_num)
```

2. 百钱百鸡问题。

说明：百钱百鸡是我国古代数学家[张丘建](#)在《算经》一书中提出的数学问题：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？翻译成现代文是：公鸡5元一只，母鸡3元一只，小鸡1元三只，用100块钱买一百只鸡，问公鸡、母鸡、小鸡各有多少只？

....

《百钱百鸡》问题

Version: 0.1

Author: 骆昊

....

```
for x in range(0, 20):
    for y in range(0, 33):
        z = 100 - x - y
        if 5 * x + 3 * y + z / 3 == 100:
            print('公鸡: %d只, 母鸡: %d只, 小鸡: %d只' % (x, y, z))
```

上面使用的方法叫做**穷举法**，也称为**暴力搜索法**，这种方法通过一项一项的列举备选解决方案中所有可能的候选项并检查每个候选项是否符合问题的描述，最终得到问题的解。这种方法看起来比较笨拙，但对于运算能力非常强大的计算机来说，通常都是一个可行的甚至是不错的选择，而且问题的解如果存在，这种方法一定能够找到它。

3. CRAPS赌博游戏。

说明：CRAPS又称花旗骰，是美国拉斯维加斯非常受欢迎的一种的桌上赌博游戏。该游戏使用两粒骰子，玩家通过摇两粒骰子获得点数进行游戏。简单的规则是：玩家第一次摇骰子如果摇出了7点或11点，玩家胜；玩家第一次如果摇出2点、3点或12点，庄家胜；其他点数玩家继续摇骰子，如果玩家摇出了7点，庄家胜；如果玩家摇出了第一次摇的点数，玩家胜；其他点数，玩家继续要骰子，直到分出胜负。

....

Craps赌博游戏

我们设定玩家开始游戏时有1000元的赌注
游戏结束的条件是玩家输光所有的赌注

Version: 0.1

Author: 骆昊

....

```
from random import randint

money = 1000
while money > 0:
```

```

print('你的总资产为:', money)
needs_go_on = False
while True:
    debt = int(input('请下注: '))
    if 0 < debt <= money:
        break
first = randint(1, 6) + randint(1, 6)
print('玩家摇出了%d点' % first)
if first == 7 or first == 11:
    print('玩家胜!')
    money += debt
elif first == 2 or first == 3 or first == 12:
    print('庄家胜!')
    money -= debt
else:
    needs_go_on = True
while needs_go_on:
    needs_go_on = False
    current = randint(1, 6) + randint(1, 6)
    print('玩家摇出了%d点' % current)
    if current == 7:
        print('庄家胜')
        money -= debt
    elif current == first:
        print('玩家胜')
        money += debt
    else:
        needs_go_on = True
print('你破产了, 游戏结束!')

```

####有用的练习

1. 生成斐波那契数列的前20个数。

说明：斐波那契数列 (Fibonacci sequence) , 又称黄金分割数列, 是意大利数学家莱昂纳多·斐波那契 (Leonardoda Fibonacci) 在《计算之书》中提出一个在理想假设条件下兔子成长率的问题而引入的数列, 所以这个数列也被戏称为"兔子数列"。斐波那契数列的特点是数列的前两个数都是1, 从第三个数开始, 每个数都是它前面两个数的和, 形如: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...。斐波那契数列在现代物理、准晶体结构、化学等领域都有直接的应用。

2. 找出10000以内的完美数。

说明：完美数又称为完全数或完备数, 它的所有的真因子 (即除了自身以外的因子) 的和 (即因子函数) 恰好等于它本身。例如: 6 ($6=1+2+3$) 和28 ($28=1+2+4+7+14$) 就是完美数。完美数有很多神奇的特性, 有兴趣的可以自行了解。

3. 输出100以内所有的素数。

说明：素数指的是只能被1和自身整除的正整数 (不包括1) 。

上面练习的参考答案在本章对应的代码目录中, 如果需要帮助请读者自行查看参考答案。

函数和模块的使用

在讲解本章节的内容之前，我们先来研究一道数学题，请说出下面的方程有多少组正整数解。

$$x_1 + x_2 + x_3 + x_4 = 8$$

事实上，上面的问题等同于将8个苹果分成四组每组至少一个苹果有多少种方案。想到这一点问题的答案就呼之欲出了。

$$C_M^N = \frac{M!}{N!(M-N)!}, (M=7, N=3)$$

可以用Python的程序来计算出这个值，代码如下所示。

```
"""
输入M和N计算C(M,N)

Version: 0.1
Author: 骆昊
"""

m = int(input('m = '))
n = int(input('n = '))
fm = 1
for num in range(1, m + 1):
    fm *= num
fn = 1
for num in range(1, n + 1):
    fn *= num
fm_n = 1
for num in range(1, m - n + 1):
    fm_n *= num
print(fm // fn // fm_n)
```

函数的作用

不知道大家是否注意到，在上面的代码中，我们做了3次求阶乘，这样的代码实际上就是重复代码。编程大师 Martin Fowler先生曾经说过：“**代码有很多种坏味道，重复是最坏的一种！**”，要写出高质量的代码首先要解决的就是重复代码的问题。对于上面的代码来说，我们可以将计算阶乘的功能封装到一个称之为“函数”的功能模块中，在需要计算阶乘的地方，我们只需要“调用”这个“函数”就可以了。

定义函数

在Python中可以使用**def**关键字来定义函数，和变量一样每个函数也有一个响亮的名字，而且命名规则跟变量的命名规则是一致的。在函数名后面的圆括号中可以放置传递给函数的参数，这一点和数学上的函数非常相似，程序中函数的参数就相当于是数学上说的函数的自变量，而函数执行完成后我们可以通过**return**关键字来返回一个值，这相当于数学上说的函数的因变量。

在了解了如何定义函数后，我们可以对上面的代码进行重构，所谓重构就是在不影响代码执行结果的前提下对代码的结构进行调整，重构之后的代码如下所示。

```
"""
输入M和N计算C(M,N)

Version: 0.1
Author: 骆昊
"""

def fac(num):
    """求阶乘"""
    result = 1
    for n in range(1, num + 1):
        result *= n
    return result

m = int(input('m = '))
n = int(input('n = '))
# 当需要计算阶乘的时候不用再写循环求阶乘而是直接调用已经定义好的函数
print(fac(m) // fac(n) // fac(m - n))
```

说明： Python的math模块中其实已经有一个名为factorial函数实现了阶乘运算，事实上求阶乘并不用自己定义函数。下面的例子中，我们讲的函数在Python标准库已经实现过了，我们这里是为了讲解函数的定义和使用才把它们又实现了一遍，**实际开发中并不建议做这种低级的重复劳动。**

函数的参数

函数是绝大多数编程语言中都支持的一个代码的"构建块"，但是Python中的函数与其他语言中的函数还是有很多不太相同的地方，其中一个显著的区别就是Python对函数参数的处理。在Python中，函数的参数可以有默认值，也支持使用可变参数，所以Python并不需要像其他语言一样支持**函数的重载**，因为我们在定义一个函数的时候可以让它有多种不同的使用方式，下面是两个小例子。

```
from random import randint

def roll_dice(n=2):
    """摇色子"""
    total = 0
    for _ in range(n):
        total += randint(1, 6)
    return total

def add(a=0, b=0, c=0):
    """三个数相加"""
    return a + b + c

# 如果没有指定参数那么使用默认值摇两颗色子
print(roll_dice())
# 摆三颗色子
print(roll_dice(3))
```

```
print(add())
print(add(1))
print(add(1, 2))
print(add(1, 2, 3))
# 传递参数时可以不按照设定的顺序进行传递
print(add(c=50, a=100, b=200))
```

我们给上面两个函数的参数都设定了默认值，这也就意味着如果在调用函数的时候如果没有传入对应参数的值时将使用该参数的默认值，所以在上面的代码中我们可以用各种不同的方式去调用add函数，这跟其他很多语言中函数重载的效果是一致的。

其实上面的add函数还有更好的实现方案，因为我们可能会对0个或多个参数进行加法运算，而具体有多少个参数是由调用者来决定，我们作为函数的设计者对这一点是一无所知的，因此在不确定参数个数的时候，我们可以使用可变参数，代码如下所示。

```
# 在参数名前面的*表示args是一个可变参数
def add(*args):
    total = 0
    for val in args:
        total += val
    return total
```

```
# 在调用add函数时可以传入0个或多个参数
print(add())
print(add(1))
print(add(1, 2))
print(add(1, 2, 3))
print(add(1, 3, 5, 7, 9))
```

用模块管理函数

对于任何一种编程语言来说，给变量、函数这样的标识符起名字都是一个让人头疼的问题，因为我们会遇到命名冲突这种尴尬的情况。最简单的场景就是在同一个.py文件中定义了两个同名函数，由于Python没有函数重载的概念，那么后面的定义会覆盖之前的定义，也就意味着两个函数同名函数实际上只有一个存在的。

```
def foo():
    print('hello, world!')

def foo():
    print('goodbye, world!')

# 下面的代码会输出什么呢?
foo()
```

当然上面的这种情况我们很容易就能避免，但是如果项目是由多人协作进行团队开发的时候，团队中可能有多个程序员都定义了名为`foo`的函数，那么怎么解决这种命名冲突呢？答案其实很简单，Python中每个文件就代表了一个模块（module），我们在不同的模块中可以有同名的函数，在使用函数的时候我们通过`import`关键字导入指定的模块就可以区分到底要使用的是哪个模块中的`foo`函数，代码如下所示。

`module1.py`

```
def foo():
    print('hello, world!')
```

`module2.py`

```
def foo():
    print('goodbye, world!')
```

`test.py`

```
from module1 import foo

# 输出hello, world!
foo()

from module2 import foo

# 输出goodbye, world!
foo()
```

也可以按照如下所示的方式来区分到底要使用哪一个`foo`函数。

`test.py`

```
import module1 as m1
import module2 as m2

m1.foo()
m2.foo()
```

但是如果将代码写成了下面的样子，那么程序中调用的是最后导入的那个`foo`，因为后导入的`foo`覆盖了之前导入的`foo`。

`test.py`

```
from module1 import foo
from module2 import foo
```

```
# 输出goodbye, world!
foo()
```

test.py

```
from module2 import foo
from module1 import foo

# 输出hello, world!
foo()
```

需要说明的是，如果我们导入的模块除了定义函数之外还有可以执行代码，那么Python解释器在导入这个模块时就会执行这些代码，事实上我们可能并不希望如此，因此如果我们在模块中编写了执行代码，最好是将这些执行代码放入如下所示的条件中，这样的话除非直接运行该模块，if条件下的这些代码是不会执行的，因为只有直接执行的模块的名字才是“`__main__`”。

module3.py

```
def foo():
    pass

def bar():
    pass

# __name__是Python中一个隐含的变量它代表了模块的名字
# 只有被Python解释器直接执行的模块的名字才是__main__
if __name__ == '__main__':
    print('call foo()')
    foo()
    print('call bar()')
    bar()
```

test.py

```
import module3

# 导入module3时 不会执行模块中if条件成立时的代码 因为模块的名字是module3而不是__main__
```

练习

练习1：实现计算求最大公约数和最小公倍数的函数。

参考答案：

```
def gcd(x, y):
    """求最大公约数"""
    (x, y) = (y, x) if x > y else (x, y)
    for factor in range(x, 0, -1):
        if x % factor == 0 and y % factor == 0:
            return factor

def lcm(x, y):
    """求最小公倍数"""
    return x * y // gcd(x, y)
```

练习2：实现判断一个数是不是回文数的函数。

参考答案：

```
def is_palindrome(num):
    """判断一个数是不是回文数"""
    temp = num
    total = 0
    while temp > 0:
        total = total * 10 + temp % 10
        temp //= 10
    return total == num
```

练习3：实现判断一个数是不是素数的函数。

参考答案：

```
def is_prime(num):
    """判断一个数是不是素数"""
    for factor in range(2, int(num ** 0.5) + 1):
        if num % factor == 0:
            return False
    return True if num != 1 else False
```

练习4：写一个程序判断输入的正整数是不是回文素数。

参考答案：

```
if __name__ == '__main__':
    num = int(input('请输入正整数: '))
```

```
if is_palindrome(num) and is_prime(num):
    print('%d是回文素数' % num)
```

注意：通过上面的程序可以看出，当我们将代码中重复出现的和相对独立的功能抽取成函数后，我们可以组合使用这些函数来解决更为复杂的问题，这也是我们为什么要定义和使用函数的一个非常重要的原因。

变量的作用域

最后，我们来讨论一下Python中有关变量作用域的问题。

```
def foo():
    b = 'hello'

    # Python中可以在函数内部再定义函数
    def bar():
        c = True
        print(a)
        print(b)
        print(c)

    bar()
    # print(c) # NameError: name 'c' is not defined

if __name__ == '__main__':
    a = 100
    # print(b) # NameError: name 'b' is not defined
    foo()
```

上面的代码能够顺利的执行并且打印出100、hello和True，但我们注意到了，在`bar`函数的内部并没有定义`a`和`b`两个变量，那么`a`和`b`是从哪里来的。我们在上面代码的`if`分支中定义了一个变量`a`，这是一个全局变量（global variable），属于全局作用域，因为它没有定义在任何一个函数中。在上面的`foo`函数中我们定义了变量`b`，这是一个定义在函数中的局部变量（local variable），属于局部作用域，在`foo`函数的外部并不能访问到它；但对于`foo`函数内部的`bar`函数来说，变量`b`属于嵌套作用域，在`bar`函数中我们是可以访问到它的。`bar`函数中的变量`c`属于局部作用域，在`bar`函数之外是无法访问的。事实上，Python查找一个变量时会按照“局部作用域”、“嵌套作用域”、“全局作用域”和“内置作用域”的顺序进行搜索，前两者我们在上面的代码中已经看到了，所谓的“内置作用域”就是Python内置的那些标识符，我们之前用过的`input`、`print`、`int`等都属于内置作用域。

再看看下面这段代码，我们希望通过函数调用修改全局变量`a`的值，但实际上下面的代码是做不到的。

```
def foo():
    a = 200
    print(a) # 200

if __name__ == '__main__':
    a = 100
```

```
foo()  
print(a) # 100
```

在调用`foo`函数后，我们发现`a`的值仍然是100，这是因为当我们在函数`foo`中写`a = 200`的时候，是重新定义了一个名字为`a`的局部变量，它跟全局作用域的`a`并不是同一个变量，因为局部作用域中有了自己的变量`a`，因此`foo`函数不再搜索全局作用域中的`a`。如果我们希望在`foo`函数中修改全局作用域中的`a`，代码如下所示。

```
def foo():  
    global a  
    a = 200  
    print(a) # 200  
  
if __name__ == '__main__':  
    a = 100  
    foo()  
    print(a) # 200
```

我们可以使用`global`关键字来指示`foo`函数中的变量`a`来自于全局作用域，如果全局作用域中没有`a`，那么下面一行的代码就会定义变量`a`并将其置于全局作用域。同理，如果我们希望函数内部的函数能够修改嵌套作用域中的变量，可以使用`nonlocal`关键字来指示变量来自于嵌套作用域，请大家自行试验。

在实际开发中，我们应该尽量减少对全局变量的使用，因为全局变量的作用域和影响过于广泛，可能会发生意外之外的修改和使用，除此之外全局变量比局部变量拥有更长的生命周期，可能导致对象占用的内存长时间无法被[垃圾回收](#)。事实上，减少对全局变量的使用，也是降低代码之间耦合度的一个重要举措，同时也是对[迪米特法则](#)的践行。减少全局变量的使用就意味着我们应该尽量让变量的作用域在函数的内部，但是如果我们将一个局部变量的生命周期延长，使其在定义它的函数调用结束后依然可以使用它的值，这时候就需要使用[闭包](#)，这个我们在后续的内容中进行讲解。

说明：很多人经常会将“闭包”和“匿名函数”混为一谈，但实际上它们并不是一回事，如果想了解这个概念，可以看看[维基百科](#)的解释或者[知乎](#)上对这个概念的讨论。

说了那么多，其实结论很简单，从现在开始我们可以将Python代码按照下面的格式进行书写，这一点点的改进其实就是在我们理解了函数和作用域的基础上跨出的巨大的一步。

```
def main():  
    # Todo: Add your code here  
    pass  
  
if __name__ == '__main__':  
    main()
```

字符串和常用数据结构

使用字符串

第二次世界大战促使了现代电子计算机的诞生，最初计算机被应用于导弹弹道的计算，而在计算机诞生后的很多年时间里，计算机处理的信息基本上都是数值型的信息。世界上的第一台电子计算机叫ENIAC（电子数值积分计算机），诞生于美国的宾夕法尼亚大学，每秒钟能够完成约5000次浮点运算。随着时间的推移，虽然数值运算仍然是计算机日常工作中最为重要的事情之一，但是今天的计算机处理得更多的数据可能都是以文本的方式存在的，如果我们希望通过Python程序操作这些文本信息，就必须要先了解字符串类型以及与它相关的知识。

所谓**字符串**，就是由零个或多个字符组成的有限序列，一般记为 $s = a_1a_2 \dots a_n (0 \leq n \leq \infty)$ 。在Python程序中，如果我们把单个或多个字符用单引号或者双引号包围起来，就可以表示一个字符串。

```
s1 = 'hello, world!'
s2 = "hello, world!"
# 以三个双引号或单引号开头的字符串可以折行
s3 = """
hello,
world!
"""
print(s1, s2, s3, end='')
```

可以在字符串中使用\（反斜杠）来表示转义，也就是说\后面的字符不再是它原来的意义，例如：\n不是代表反斜杠和字符n，而是表示换行；而\t也不是代表反斜杠和字符t，而是表示制表符。所以如果想在字符串中表示'要写成\'，同理想表示\要写成\\。可以运行下面的代码看看会输出什么。

```
s1 = '\'hello, world!\''
s2 = '\n\\hello, world!\\\'\n'
print(s1, s2, end='')
```

在\后面还可以跟一个八进制或者十六进制数来表示字符，例如\141和\x61都代表小写字母a，前者是八进制的表示法，后者是十六进制的表示法。也可以在\后面跟Unicode字符编码来表示字符，例如\u9a86\u660a代表的是中文“骆昊”。运行下面的代码，看看输出了什么。

```
s1 = '\141\142\143\x61\x62\x63'
s2 = '\u9a86\u660a'
print(s1, s2)
```

如果不希望字符串中的\表示转义，我们可以通过在字符串的最前面加上字母r来加以说明，再看看下面的代码又会输出什么。

```
s1 = r'\hello, world!\''  
s2 = r'\n\hello, world!\\\n'  
print(s1, s2, end='')
```

Python为字符串类型提供了非常丰富的运算符，我们可以使用`+`运算符来实现字符串的拼接，可以使用`*`运算符来重复一个字符串的内容，可以使用`in`和`not in`来判断一个字符串是否包含另外一个字符串（成员运算），我们也可以用`[]`和`[:]`运算符从字符串取出某个字符或某些字符（切片运算），代码如下所示。

```
s1 = 'hello' * 3  
print(s1) # hello hello hello  
s2 = 'world'  
s1 += s2  
print(s1) # hello hello hello world  
print('ll' in s1) # True  
print('good' in s1) # False  
str2 = 'abc123456'  
# 从字符串中取出指定位置的字符(下标运算)  
print(str2[2]) # c  
# 字符串切片(从指定的开始索引到指定的结束索引)  
print(str2[2:5]) # c12  
print(str2[2:]) # c123456  
print(str2[2::2]) # c246  
print(str2[::-2]) # ac246  
print(str2[::-1]) # 654321cba  
print(str2[-3:-1]) # 45
```

在Python中，我们还可以通过一系列的方法来完成对字符串的处理，代码如下所示。

```
str1 = 'hello, world!'  
# 通过内置函数len计算字符串的长度  
print(len(str1)) # 13  
# 获得字符串首字母大写的拷贝  
print(str1.capitalize()) # Hello, world!  
# 获得字符串每个单词首字母大写的拷贝  
print(str1.title()) # Hello, World!  
# 获得字符串变大写后的拷贝  
print(str1.upper()) # HELLO, WORLD!  
# 从字符串中查找子串所在位置  
print(str1.find('or')) # 8  
print(str1.find('shit')) # -1  
# 与find类似但找不到子串时会引发异常  
# print(str1.index('or'))  
# print(str1.index('shit'))  
# 检查字符串是否以指定的字符串开头  
print(str1.startswith('He')) # False  
print(str1.startswith('hel')) # True  
# 检查字符串是否以指定的字符串结尾  
print(str1.endswith('!')) # True
```

```
# 将字符串以指定的宽度居中并在两侧填充指定的字符
print(str1.center(50, '*'))
# 将字符串以指定的宽度靠右放置左侧填充指定的字符
print(str1.rjust(50, ' '))
str2 = 'abc123456'
# 检查字符串是否由数字构成
print(str2.isdigit()) # False
# 检查字符串是否以字母构成
print(str2.isalpha()) # False
# 检查字符串是否以数字和字母构成
print(str2.isalnum()) # True
str3 = ' jackfrued@126.com '
print(str3)
# 获得字符串修剪左右两侧空格之后的拷贝
print(str3.strip())
```

我们之前讲过，可以用下面的方式来格式化输出字符串。

```
a, b = 5, 10
print('%d * %d = %d' % (a, b, a * b))
```

当然，我们也可以用字符串提供的方法来完成字符串的格式，代码如下所示。

```
a, b = 5, 10
print('{0} * {1} = {2}'.format(a, b, a * b))
```

Python 3.6以后，格式化字符串还有更为简洁的书写方式，就是在字符串前加上字母f，我们可以使用下面的语法糖来简化上面的代码。

```
a, b = 5, 10
print(f'{a} * {b} = {a * b}')
```

除了字符串，Python还内置了多种类型的数据结构，如果要在程序中保存和操作数据，绝大多数时候可以利用现有的数据结构来实现，最常用的包括列表、元组、集合和字典。

使用列表

不知道大家是否注意到，刚才我们讲到的字符串类型（str）和之前我们讲到的数值类型（int和float）有一些区别。数值类型是标量类型，也就是说这种类型的对象没有可以访问的内部结构；而字符串类型是一种结构化的、非标量类型，所以才会有一系列的属性和方法。接下来我们要介绍的列表（list），也是一种结构化的、非标量类型，它是值的有序序列，每个值都可以通过索引进行标识，定义列表可以将列表的元素放在[]中，多个元素用,进行分隔，可以使用for循环对列表元素进行遍历，也可以使用[]或[:]运算符取出列表中的一个或多个元素。

下面的代码演示了如何定义列表、如何遍历列表以及列表的下标运算。

```
list1 = [1, 3, 5, 7, 100]
print(list1) # [1, 3, 5, 7, 100]
# 乘号表示列表元素的重复
list2 = ['hello'] * 3
print(list2) # ['hello', 'hello', 'hello']
# 计算列表长度(元素个数)
print(len(list1)) # 5
# 下标(索引)运算
print(list1[0]) # 1
print(list1[4]) # 100
# print(list1[5]) # IndexError: list index out of range
print(list1[-1]) # 100
print(list1[-3]) # 5
list1[2] = 300
print(list1) # [1, 3, 300, 7, 100]
# 通过循环用下标遍历列表元素
for index in range(len(list1)):
    print(list1[index])
# 通过for循环遍历列表元素
for elem in list1:
    print(elem)
# 通过enumerate函数处理列表之后再遍历可以同时获得元素索引和值
for index, elem in enumerate(list1):
    print(index, elem)
```

下面的代码演示了如何向列表中添加元素以及如何从列表中移除元素。

```
list1 = [1, 3, 5, 7, 100]
# 添加元素
list1.append(200)
list1.insert(1, 400)
# 合并两个列表
# list1.extend([1000, 2000])
list1 += [1000, 2000]
print(list1) # [1, 400, 3, 5, 7, 100, 200, 1000, 2000]
print(len(list1)) # 9
# 先通过成员运算判断元素是否在列表中, 如果存在就删除该元素
if 3 in list1:
    list1.remove(3)
if 1234 in list1:
    list1.remove(1234)
print(list1) # [1, 400, 5, 7, 100, 200, 1000, 2000]
# 从指定的位置删除元素
list1.pop(0)
list1.pop(len(list1) - 1)
print(list1) # [400, 5, 7, 100, 200, 1000]
# 清空列表元素
list1.clear()
print(list1) # []
```

和字符串一样，列表也可以做切片操作，通过切片操作我们可以实现对列表的复制或者将列表中的一部分取出 来创建出新的列表，代码如下所示。

```

fruits = ['grape', 'apple', 'strawberry', 'waxberry']
fruits += ['pitaya', 'pear', 'mango']
# 列表切片
fruits2 = fruits[1:4]
print(fruits2) # apple strawberry waxberry
# 可以通过完整切片操作来复制列表
fruits3 = fruits[:]
print(fruits3) # ['grape', 'apple', 'strawberry', 'waxberry', 'pitaya', 'pear',
'mango']
fruits4 = fruits[-3:-1]
print(fruits4) # ['pitaya', 'pear']
# 可以通过反向切片操作来获得倒转后的列表的拷贝
fruits5 = fruits[::-1]
print(fruits5) # ['mango', 'pear', 'pitaya', 'waxberry', 'strawberry', 'apple',
'grape']

```

下面的代码实现了对列表的排序操作。

```

list1 = ['orange', 'apple', 'zoo', 'internationalization', 'blueberry']
list2 = sorted(list1)
# sorted函数返回列表排序后的拷贝不会修改传入的列表
# 函数的设计就应该像sorted函数一样尽可能不产生副作用
list3 = sorted(list1, reverse=True)
# 通过key关键字参数指定根据字符串长度进行排序而不是默认的字母表顺序
list4 = sorted(list1, key=len)
print(list1)
print(list2)
print(list3)
print(list4)
# 给列表对象发出排序消息直接在列表对象上进行排序
list1.sort(reverse=True)
print(list1)

```

生成式和生成器

我们还可以使用列表的生成式语法来创建列表，代码如下所示。

```

f = [x for x in range(1, 10)]
print(f)
f = [x + y for x in 'ABCDE' for y in '1234567']
print(f)
# 用列表的生成表达式语法创建列表容器
# 用这种语法创建列表之后元素已经准备就绪所以需要耗费较多的内存空间
f = [x ** 2 for x in range(1, 1000)]
print(sys.getsizeof(f)) # 查看对象占用内存的字节数

```

```

print(f)
# 请注意下面的代码创建的不是一个列表而是一个生成器对象
# 通过生成器可以获取到数据但它不占用额外的空间存储数据
# 每次需要数据的时候就通过内部的运算得到数据(需要花费额外的时间)
f = (x ** 2 for x in range(1, 1000))
print(sys.getsizeof(f)) # 相比生成式生成器不占用存储数据的空间
print(f)
for val in f:
    print(val)

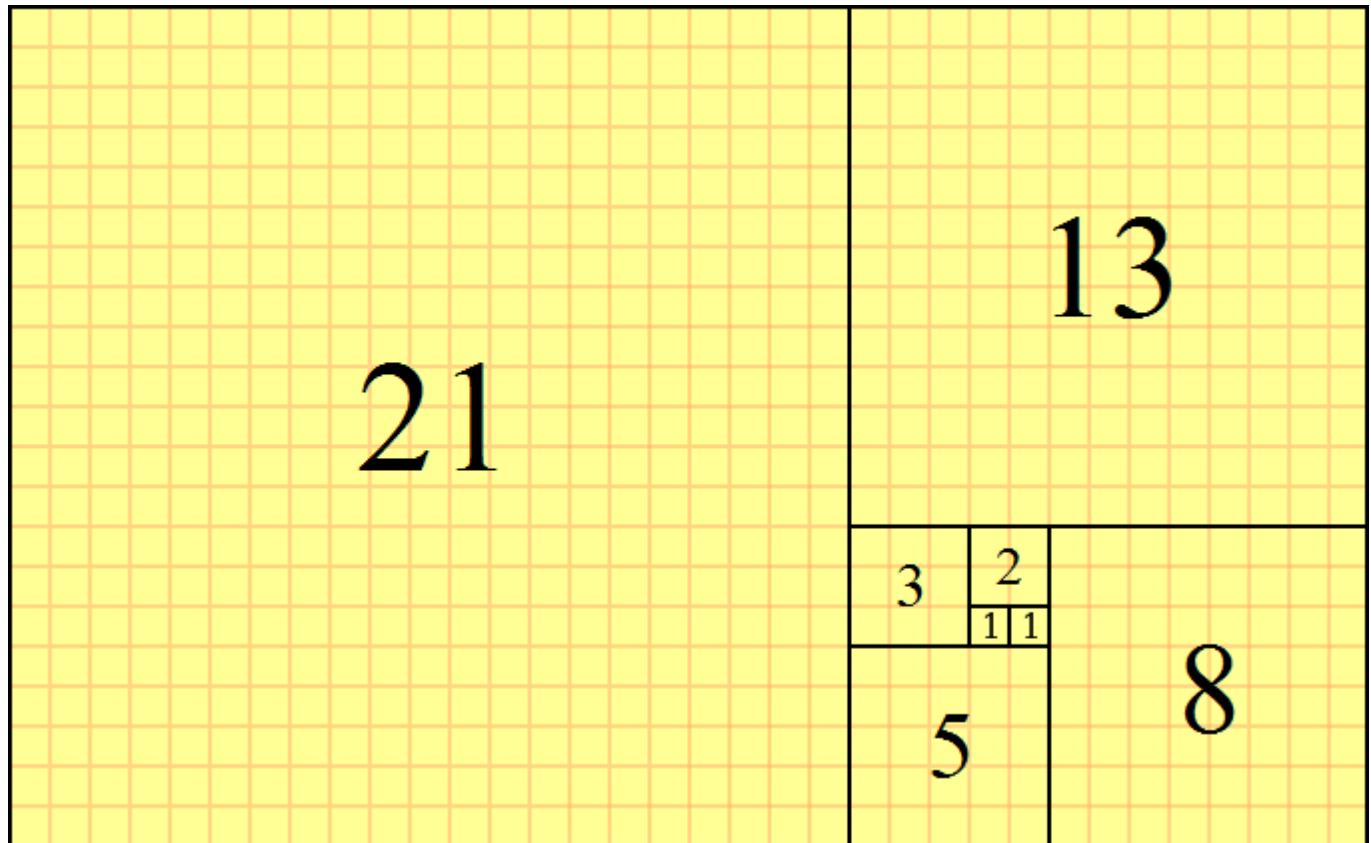
```

除了上面提到的生成器语法，Python中还有另外一种定义生成器的方式，就是通过`yield`关键字将一个普通函数改造成生成器函数。下面的代码演示了如何实现一个生成斐波拉切数列的生成器。所谓斐波拉切数列可以通过下面`递归`的方法来进行定义：

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} (n \geq 2)$$



```

def fib(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
        yield a

def main():

```

```

for val in fib(20):
    print(val)

if __name__ == '__main__':
    main()

```

使用元组

Python中的元组与列表类似也是一种容器数据类型，可以用一个变量（对象）来存储多个数据，不同之处在于元组的元素不能修改，在前面的代码中我们已经不止一次使用过元组了。顾名思义，我们把多个元素组合到一起就形成了一个元组，所以它和列表一样可以保存多条数据。下面的代码演示了如何定义和使用元组。

```

# 定义元组
t = ('骆昊', 38, True, '四川成都')
print(t)
# 获取元组中的元素
print(t[0])
print(t[3])
# 遍历元组中的值
for member in t:
    print(member)
# 重新给元组赋值
# t[0] = '王大锤' # TypeError
# 变量t重新引用了新的元组原来的元组将被垃圾回收
t = ('王大锤', 20, True, '云南昆明')
print(t)
# 将元组转换成列表
person = list(t)
print(person)
# 列表是可以修改它的元素的
person[0] = '李小龙'
person[1] = 25
print(person)
# 将列表转换成元组
fruits_list = ['apple', 'banana', 'orange']
fruits_tuple = tuple(fruits_list)
print(fruits_tuple)

```

这里有一个非常值得探讨的问题，我们已经有了列表这种数据结构，为什么还需要元组这样的类型呢？

1. 元组中的元素是无法修改的，事实上我们在项目中尤其是多线程环境（后面会讲到）中可能更喜欢使用的是那些不变对象（一方面因为对象状态不能修改，所以可以避免由此引起的不必要的程序错误，简单的说就是一个不变的对象要比可变的对象更加容易维护；另一方面因为没有任何一个线程能够修改不变对象的内部状态，一个不变对象自动就是线程安全的，这样就可以省掉处理同步化的开销。一个不变对象可以方便的被共享访问）。所以结论就是：如果不需要对元素进行添加、删除、修改的时候，可以考虑使用元组，当然如果一个方法要返回多个值，使用元组也是不错的选择。
2. 元组在创建时间和占用的空间上面都优于列表。我们可以使用sys模块的getsizeof函数来检查存储同样的元素的元组和列表各自占用了多少内存空间，这个很容易做到。我们也可以在ipython中使用魔法指

令%timeit来分析创建同样内容的元组和列表所花费的时间，下图是我的macOS系统上测试的结果。

```
>Last login: Thu Mar  8 22:17:13 on console
[jackfrued:~ Hao$ ipython
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help
.

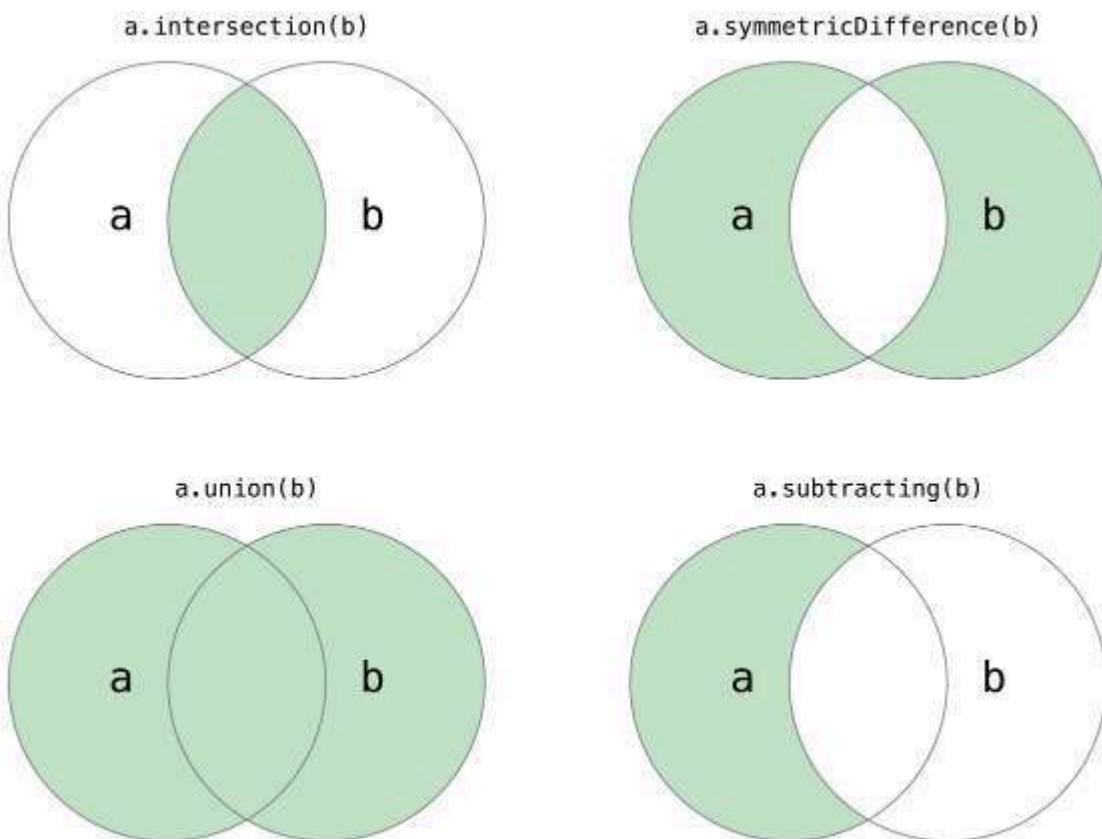
In [1]: %timeit [1, 2, 3, 4, 5]
73.3 ns ± 0.445 ns per loop (mean ± std. dev. of 7 runs, 100000000
loops each)

In [2]: %timeit (1, 2, 3, 4, 5)
16.6 ns ± 0.491 ns per loop (mean ± std. dev. of 7 runs, 100000000
loops each)

In [3]: 
```

使用集合

Python中的集合跟数学上的集合是一致的，不允许有重复元素，而且可以进行交集、并集、差集等运算。



可以按照下面代码所示的方式来创建和使用集合。

```
# 创建集合的字面量语法
set1 = {1, 2, 3, 3, 3, 2}
print(set1)
print('Length =', len(set1))
# 创建集合的构造器语法(面向对象部分会进行详细讲解)
set2 = set(range(1, 10))
set3 = set((1, 2, 3, 3, 2, 1))
print(set2, set3)
# 创建集合的推导式语法(推导式也可以用于推导集合)
set4 = {num for num in range(1, 100) if num % 3 == 0 or num % 5 == 0}
print(set4)
```

向集合添加元素和从集合删除元素。

```
set1.add(4)
set1.add(5)
set2.update([11, 12])
set2.discard(5)
if 4 in set2:
    set2.remove(4)
print(set1, set2)
print(set3.pop())
print(set3)
```

集合的成员、交集、并集、差集等运算。

```
# 集合的交集、并集、差集、对称差运算
print(set1 & set2)
# print(set1.intersection(set2))
print(set1 | set2)
# print(set1.union(set2))
print(set1 - set2)
# print(set1.difference(set2))
print(set1 ^ set2)
# print(set1.symmetric_difference(set2))
# 判断子集和超集
print(set2 <= set1)
# print(set2.issubset(set1))
print(set3 <= set1)
# print(set3.issubset(set1))
print(set1 >= set2)
# print(set1.issuperset(set2))
print(set1 >= set3)
# print(set1.issuperset(set3))
```

说明： Python中允许通过一些特殊的方法来为某种类型或数据结构自定义运算符（后面的章节中会讲到），上面的代码中我们对集合进行运算的时候可以调用集合对象的方法，也可以直接使用对应的运算符，例如&运算符跟intersection方法的作用就是一样的，但是使用运算符让代码更加直观。

使用字典

字典是另一种可变容器模型，Python中的字典跟我们生活中使用的字典是一样的，它可以存储任意类型对象，与列表、集合不同的是，字典的每个元素都是由一个键和一个值组成的“键值对”，键和值通过冒号分开。下面的代码演示了如何定义和使用字典。

```
# 创建字典的字面量语法
scores = {'骆昊': 95, '白元芳': 78, '狄仁杰': 82}
print(scores)
# 创建字典的构造器语法
items1 = dict(one=1, two=2, three=3, four=4)
# 通过zip函数将两个序列压成字典
items2 = dict(zip(['a', 'b', 'c'], '123'))
# 创建字典的推导式语法
items3 = {num: num ** 2 for num in range(1, 10)}
print(items1, items2, items3)
# 通过键可以获取字典中对应的值
print(scores['骆昊'])
print(scores['狄仁杰'])
# 对字典中所有键值对进行遍历
for key in scores:
    print(f'{key}: {scores[key]}')
# 更新字典中的元素
scores['白元芳'] = 65
scores['诸葛王朗'] = 71
scores.update(冷面=67, 方启鹤=85)
print(scores)
if '武则天' in scores:
    print(scores['武则天'])
print(scores.get('武则天'))
# get方法也是通过键获取对应的值但是可以设置默认值
print(scores.get('武则天', 60))
# 删除字典中的元素
print(scores.popitem())
print(scores.popitem())
print(scores.pop('骆昊', 100))
# 清空字典
scores.clear()
print(scores)
```

练习

练习1：在屏幕上显示跑马灯文字。

参考答案：

```
import os
import time

def main():
    content = '北京欢迎你为你开天辟地.....'
    while True:
        # 清理屏幕上的输出
        os.system('cls') # os.system('clear')
        print(content)
        # 休眠200毫秒
        time.sleep(0.2)
        content = content[1:] + content[0]

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

练习2：设计一个函数产生指定长度的验证码，验证码由大小写字母和数字构成。

参考答案：

```
import random

def generate_code(code_len=4):
    """
    生成指定长度的验证码
    :param code_len: 验证码的长度(默认4个字符)
    :return: 由大小写英文字母和数字构成的随机验证码
    """
    all_chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    last_pos = len(all_chars) - 1
    code = ''
    for _ in range(code_len):
        index = random.randint(0, last_pos)
        code += all_chars[index]
    return code
```

练习3：设计一个函数返回给定文件名的后缀名。

参考答案：

```
def get_suffix(filename, has_dot=False):
    """
```

获取文件名的后缀名

```

:param filename: 文件名
:param has_dot: 返回的后缀名是否需要带点
:returns: 文件的后缀名
"""

pos = filename.rfind('.')
if 0 < pos < len(filename) - 1:
    index = pos if has_dot else pos + 1
    return filename[index:]
else:
    return ''

```

练习4：设计一个函数返回传入的列表中最大和第二大的元素的值。

参考答案：

```

def max2(x):
    m1, m2 = (x[0], x[1]) if x[0] > x[1] else (x[1], x[0])
    for index in range(2, len(x)):
        if x[index] > m1:
            m2 = m1
            m1 = x[index]
        elif x[index] > m2:
            m2 = x[index]
    return m1, m2

```

练习5：计算指定的年月日是这一年的第几天。

参考答案：

```

def is_leap_year(year):
    """
    判断指定的年份是不是闰年

    :param year: 年份
    :return: 闰年返回True平年返回False
    """
    return year % 4 == 0 and year % 100 != 0 or year % 400 == 0

def which_day(year, month, date):
    """
    计算传入的日期是这一年的第几天

    :param year: 年
    :param month: 月
    :param date: 日
    :return: 第几天
    """

```

```

    """
days_of_month = [
    [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31],
    [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
][is_leap_year(year)]
total = 0
for index in range(month - 1):
    total += days_of_month[index]
return total + date

def main():
    print(which_day(1980, 11, 28))
    print(which_day(1981, 12, 31))
    print(which_day(2018, 1, 1))
    print(which_day(2016, 3, 1))

if __name__ == '__main__':
    main()

```

练习6：打印杨辉三角。

参考答案：

```

def main():
    num = int(input('Number of rows: '))
    yh = [[]] * num
    for row in range(len(yh)):
        yh[row] = [None] * (row + 1)
        for col in range(len(yh[row])):
            if col == 0 or col == row:
                yh[row][col] = 1
            else:
                yh[row][col] = yh[row - 1][col] + yh[row - 1][col - 1]
            print(yh[row][col], end='\t')
        print()

if __name__ == '__main__':
    main()

```

综合案例

案例1：双色球选号。

```
from random import randrange, randint, sample
```

```

def display(balls):
    """
    输出列表中的双色球号码
    """
    for index, ball in enumerate(balls):
        if index == len(balls) - 1:
            print('|', end=' ')
        print('%02d' % ball, end=' ')
    print()

def random_select():
    """
    随机选择一组号码
    """
    red_balls = [x for x in range(1, 34)]
    selected_balls = []
    selected_balls = sample(red_balls, 6)
    selected_balls.sort()
    selected_balls.append(randint(1, 16))
    return selected_balls

def main():
    n = int(input('机选几注: '))
    for _ in range(n):
        display(random_select())

if __name__ == '__main__':
    main()

```

说明：上面使用random模块的sample函数来实现从列表中选择不重复的n个元素。

综合案例2：约瑟夫环问题。

....

《幸运的基督徒》

有15个基督徒和15个非基督徒在海上遇险，为了能让一部分人活下来不得不将其中15个人扔到海里面去，有个人想了个办法就是大家围成一个圈，由某个人开始从1报数，报到9的人就扔到海里面，他后面的人接着从1开始报数，报到9的人继续扔到海里面，直到扔掉15个人。由于上帝的保佑，15个基督徒都幸免于难，问这些人最开始是怎么站的，哪些位置是基督徒哪些位置是非基督徒。

....

```

def main():
    persons = [True] * 30
    counter, index, number = 0, 0, 0
    while counter < 15:
        if persons[index]:
            number += 1
        index = (index + 1) % 30
        counter += 1

```

```
if number == 9:
    persons[index] = False
    counter += 1
    number = 0
index += 1
index %= 30
for person in persons:
    print('基' if person else '非', end='')

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

综合案例3：井字棋游戏。

```
import os

def print_board(board):
    print(board['TL'] + ' | ' + board['TM'] + ' | ' + board['TR'])
    print('---+---')
    print(board['ML'] + ' | ' + board['MM'] + ' | ' + board['MR'])
    print('---+---')
    print(board['BL'] + ' | ' + board['BM'] + ' | ' + board['BR'])

def main():
    init_board = {
        'TL': ' ', 'TM': ' ', 'TR': ' ',
        'ML': ' ', 'MM': ' ', 'MR': ' ',
        'BL': ' ', 'BM': ' ', 'BR': ' '
    }
    begin = True
    while begin:
        curr_board = init_board.copy()
        begin = False
        turn = 'x'
        counter = 0
        os.system('clear')
        print_board(curr_board)
        while counter < 9:
            move = input('轮到%s走棋, 请输入位置: ' % turn)
            if curr_board[move] == ' ':
                counter += 1
                curr_board[move] = turn
                if turn == 'x':
                    turn = 'o'
                else:
                    turn = 'x'
            os.system('clear')
```

```
print_board(curr_board)
choice = input('再玩一局?(yes|no)')
begin = choice == 'yes'

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

说明：最后这个案例来自《Python编程快速上手:让繁琐工作自动化》一书（这本书对有编程基础想迅速使用Python将日常工作自动化的人来说还是不错的选择），对代码做了一点点的调整。

面向对象编程基础

活在当下的程序员应该都听过"面向对象编程"一词，也经常有人问能不能用一句话解释下什么是"面向对象编程"，我们先来看看比较正式的说法。

"把一组数据结构和处理它们的方法组成对象 (object)，把相同行为的对象归纳为类 (class)，通过类的封装 (encapsulation) 隐藏内部细节，通过继承 (inheritance) 实现类的特化 (specialization) 和泛化 (generalization)，通过多态 (polymorphism) 实现基于对象类型的动态分派。"

这样一说是更不明白了。所以我们还是看看更通俗易懂的说法，下面这段内容来自于[知乎](#)。



成心文

不会拍电影的程序员不是好机长

51 人赞同了该回答

一句话说明什么是面向对象？你个土鳖，你们全家都是土鳖！

好像有人说过这样的话，当头棒喝的方式虽然情感上不易接受，但记忆效果十分显著。

好吧，如果你觉得"土鳖"实在难听也不能准确定位你的档次，你可以自行将其替换为"土豪"，whatever。

面向对象思想有三大要素：封装、继承和多态。

- 封装：不管你是土鳖还是土豪，不管你中午吃的是窝头还是鲍鱼，你的下水都在你肚皮里，别人看不到你中午吃了啥，除非你自己说给他们听（或者画给他们看，whatever）；
- 继承：刚说了，你个土鳖/豪，你们全家都是土鳖/豪。冰冻三尺非一日之寒，你有今天，必定可以从你爸爸妈妈那里追根溯源。正所谓虎父无犬子，正恩同学那么狠，他爹正日就不是什么善茬，更甭说他爷爷日成，明白了吗？
- 多态：哲学家说过，世上不会有两个一模一样的双胞胎。即使你从你父亲那里继承来的土鳖/豪气质，也不可能完全是从一个模子里刻出来的，总会有些差别。比如你爸喜欢蹲在门前吃面，你喜欢骑在村口的歪脖子树上吃，或者反过来。当然，也可能令尊爱吃龙虾鲍鱼时旁边有几个艺校小女生喝酒唱歌助兴，你可能更喜欢弄个街舞乐队来吹拉弹唱。

说明：以上的内容来自于网络，不代表作者本人的观点和看法，与作者本人立场无关，相关责任不由作者承担。

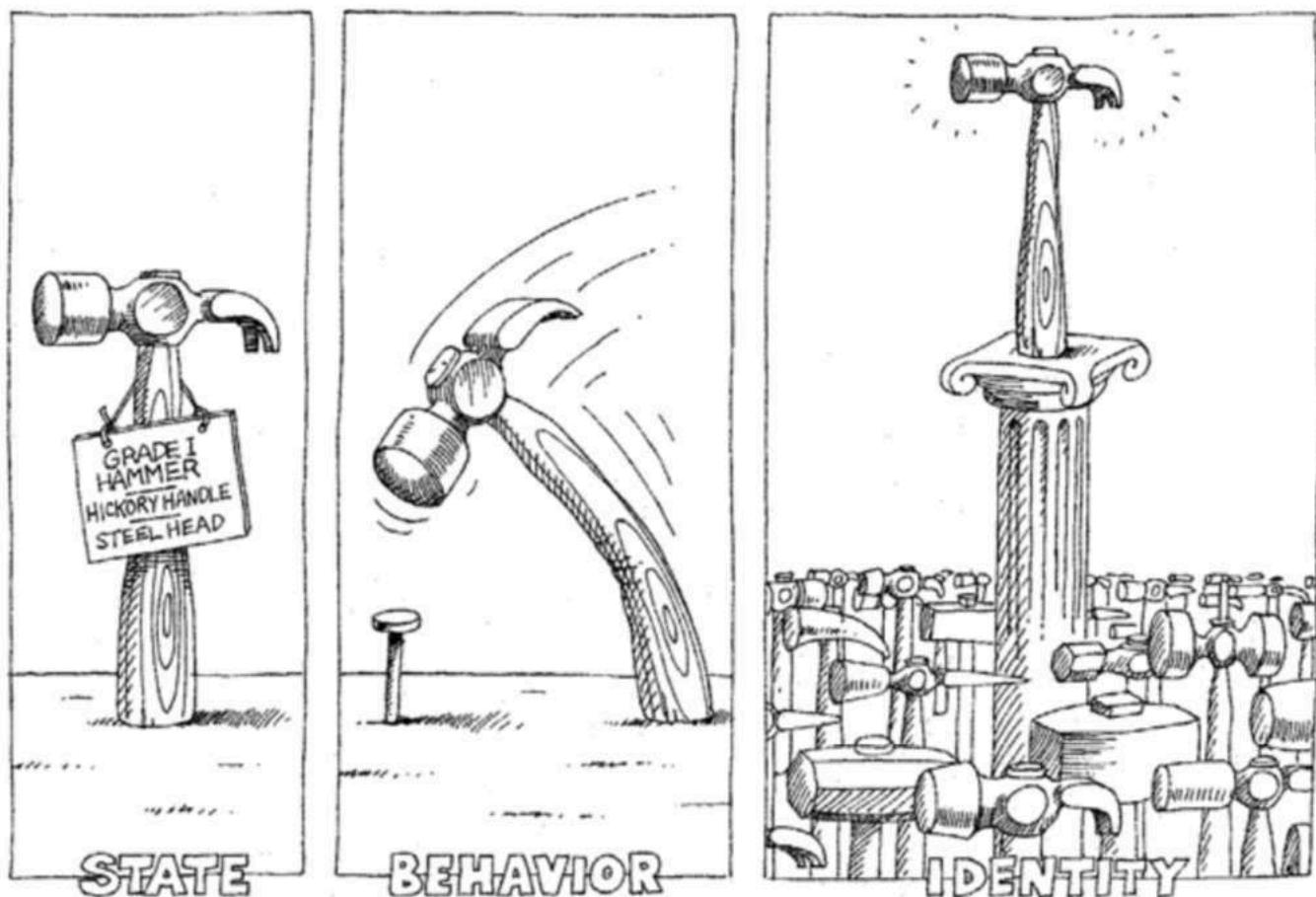
之前我们说过"程序是指令的集合"，我们在程序中书写的语句在执行时会变成一条或多条指令然后由CPU去执行。当然为了简化程序的设计，我们引入了函数的概念，把相对独立且经常重复使用的代码放置到函数中，在需要使用这些功能的时候只要调用函数即可；如果一个函数的功能过于复杂和臃肿，我们又可以进一步将函数继续切分为子函数来降低系统的复杂性。但是说了这么多，不知道大家是否发现，所谓编程就是程序员按照计算机的工作方式控制计算机完成各种任务。但是，计算机的工作方式与正常人类的思维模式是不同的，如果编程就必须得抛弃人类正常的思维方式去迎合计算机，编程的乐趣就少了很多，"每个人都应该学习编程"这样的豪言壮语就只能说说而已。当然，这些还不是最重要的，最重要的是当我们需要开发一个复杂的系统时，代码的复杂性会让开发和维护工作都变得举步维艰，所以在上世纪60年代末期，"软件危机"、"软件工程"等一系列的概念开始在行业中出现。

当然，程序员圈子内的人都知道，现实中并没有解决上面所说的这些问题的“**银弹**”，真正让软件开发者看到希望的是上世纪70年代诞生的**Smalltalk**编程语言中引入的面向对象的编程思想（面向对象编程的雏形可以追溯到更早期的**Simula**语言）。按照这种编程理念，程序中的数据和操作数据的函数是一个逻辑上的整体，我们称之为“对象”，而我们解决问题的方式就是创建出需要的对象并向对象发出各种各样的消息，多个对象的协同工作最终可以让我们构造出复杂的系统来解决现实中的问题。

说明：当然面向对象也不是解决软件开发中所有问题的最后的“银弹”，所以今天的高级程序设计语言几乎都提供了对多种编程范式的支持，Python也不例外。

类和对象

简单的说，类是对象的蓝图和模板，而对象是类的实例。这个解释虽然有点像用概念在解释概念，但是从这句话我们至少可以看出，类是抽象的概念，而对象是具体的东西。在面向对象编程的世界中，一切皆为对象，对象都有属性和行为，每个对象都是独一无二的，而且对象一定属于某个类（型）。当我们把一大堆拥有共同特征的对象的静态特征（属性）和动态特征（行为）都抽取出来后，就可以定义出一个叫做“类”的东西。



定义类

在Python中可以使用**class**关键字定义类，然后在类中通过之前学习过的函数来定义方法，这样就可以将对象的动态特征描述出来，代码如下所示。

```
class Student(object):
    # __init__是一个特殊方法用于在创建对象时进行初始化操作
    # 通过这个方法我们可以为学生对象绑定name和age两个属性
    def __init__(self, name, age):
```

```

    self.name = name
    self.age = age

def study(self, course_name):
    print('%s正在学习%s.' % (self.name, course_name))

# PEP 8要求标识符的名字用全小写多个单词用下划线连接
# 但是部分程序员和公司更倾向于使用驼峰命名法(驼峰标识)
def watch_movie(self):
    if self.age < 18:
        print('%s只能观看《熊出没》.' % self.name)
    else:
        print('%s正在观看岛国爱情大电影.' % self.name)

```

说明：写在类中的函数，我们通常称之为（对象的）方法，这些方法就是对象可以接收的消息。

创建和使用对象

当我们定义好一个类之后，可以通过下面的方式来创建对象并给对象发消息。

```

def main():
    # 创建学生对象并指定姓名和年龄
    stu1 = Student('骆昊', 38)
    # 给对象发study消息
    stu1.study('Python程序设计')
    # 给对象发watch_av消息
    stu1.watch_movie()
    stu2 = Student('王大锤', 15)
    stu2.study('思想品德')
    stu2.watch_movie()

if __name__ == '__main__':
    main()

```

访问可见性问题

对于上面的代码，有C++、Java、C#等编程经验的程序员可能会问，我们给Student对象绑定的name和age属性到底具有怎样的访问权限（也称为可见性）。因为在很多面向对象编程语言中，我们通常会将对象的属性设置为私有的（private）或受保护的（protected），简单的说就是不允许外界访问，而对象的方法通常都是公开的（public），因为公开的方法就是对象能够接受的消息。在Python中，属性和方法的访问权限只有两种，也就是公开的和私有的，如果希望属性是私有的，在给属性命名时可以用两个下划线作为开头，下面的代码可以验证这一点。

```

class Test:

    def __init__(self, foo):
        self.__foo = foo

```

```
def __bar(self):
    print(self.__foo)
    print('__bar')

def main():
    test = Test('hello')
    # AttributeError: 'Test' object has no attribute '__bar'
    test.__bar()
    # AttributeError: 'Test' object has no attribute '__foo'
    print(test.__foo)

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

但是，Python并没有从语法上严格保证私有属性或方法的私密性，它只是给私有的属性和方法换了一个名字来妨碍对它们的访问，事实上如果你知道更换名字的规则仍然可以访问到它们，下面的代码就可以验证这一点。之所以这样设定，可以用这样一句名言加以解释，就是“**We are all consenting adults here**”。因为绝大多数程序员都认为开放比封闭要好，而且程序员要自己为自己的行为负责。

```
class Test:

    def __init__(self, foo):
        self.__foo = foo

    def __bar(self):
        print(self.__foo)
        print('__bar')

    def main():
        test = Test('hello')
        test._Test__bar()
        print(test._Test__foo)

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

在实际开发中，我们并不建议将属性设置为私有的，因为这会导致子类无法访问（后面会讲到）。所以大多数Python程序员会遵循一种命名惯例就是让属性名以单下划线开头来表示属性是受保护的，本类之外的代码在访问这样的属性时应该要保持慎重。这种做法并不是语法上的规则，单下划线开头的属性和方法外界仍然是可以访问的，所以更多的时候它是一种暗示或隐喻，关于这一点可以看看我的[《Python - 那些年我们踩过的那些坑》](#)文章中的讲解。

面向对象的支柱

面向对象有三大支柱：封装、继承和多态。后面两个概念在下一个章节中进行详细的说明，这里我们先说一下什么是封装。我自己对封装的理解是"隐藏一切可以隐藏的实现细节，只向外界暴露（提供）简单的编程接口"。我们在类中定义的方法其实就是把数据和对数据的操作封装起来了，在我们创建了对象之后，只需要给对象发送一个消息（调用方法）就可以执行方法中的代码，也就是说我们只需要知道方法的名字和传入的参数（方法的外部视图），而不需要知道方法内部的实现细节（方法的内部视图）。

练习

练习1：定义一个类描述数字时钟。

参考答案：

```
from time import sleep

class Clock(object):
    """数字时钟"""

    def __init__(self, hour=0, minute=0, second=0):
        """初始化方法

        :param hour: 时
        :param minute: 分
        :param second: 秒
        """
        self._hour = hour
        self._minute = minute
        self._second = second

    def run(self):
        """走字"""
        self._second += 1
        if self._second == 60:
            self._second = 0
            self._minute += 1
            if self._minute == 60:
                self._minute = 0
                self._hour += 1
                if self._hour == 24:
                    self._hour = 0

    def show(self):
        """显示时间"""
        return '%02d:%02d:%02d' % \
            (self._hour, self._minute, self._second)

    def main():
        clock = Clock(23, 59, 58)
        while True:
            print(clock.show())
            sleep(1)
```

```
clock.run()
```

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

练习2：定义一个类描述平面上的点并提供移动点和计算到另一个点距离的方法。

参考答案：

```
from math import sqrt

class Point(object):

    def __init__(self, x=0, y=0):
        """初始化方法

        :param x: 横坐标
        :param y: 纵坐标
        """
        self.x = x
        self.y = y

    def move_to(self, x, y):
        """移动到指定位置

        :param x: 新的横坐标
        :param y: 新的纵坐标
        """
        self.x = x
        self.y = y

    def move_by(self, dx, dy):
        """移动指定的增量

        :param dx: 横坐标的增量
        :param dy: 纵坐标的增量
        """
        self.x += dx
        self.y += dy

    def distance_to(self, other):
        """计算与另一个点的距离

        :param other: 另一个点
        """
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx ** 2 + dy ** 2)
```

```
def __str__(self):  
    return '%s, %s' % (str(self.x), str(self.y))  
  
def main():  
    p1 = Point(3, 5)  
    p2 = Point()  
    print(p1)  
    print(p2)  
    p2.move_by(-1, 2)  
    print(p2)  
    print(p1.distance_to(p2))  
  
if __name__ == '__main__':  
    main()
```

说明：本章中的插图来自于Grady Booch等著作的《面向对象分析与设计》一书，该书是讲解面向对象编程的经典著作，有兴趣的读者可以购买和阅读这本书来了解更多的面向对象的相关知识。

面向对象进阶

在前面的章节我们已经了解了面向对象的入门知识，知道了如何定义类，如何创建对象以及如何给对象发消息。为了能够更好的使用面向对象编程思想进行程序开发，我们还需要对Python中的面向对象编程进行更为深入的了解。

@property装饰器

之前我们讨论过Python中属性和方法访问权限的问题，虽然我们不建议将属性设置为私有的，但是如果直接将属性暴露给外界也是有问题的，比如我们没有办法检查赋给属性的值是否有效。我们之前的建议是将属性命名以单下划线开头，通过这种方式来暗示属性是受保护的，不建议外界直接访问，那么如果想访问属性可以通过属性的getter（访问器）和setter（修改器）方法进行对应的操作。如果要做到这点，就可以考虑使用@property包装器来包装getter和setter方法，使得对属性的访问既安全又方便，代码如下所示。

```
class Person(object):

    def __init__(self, name, age):
        self._name = name
        self._age = age

    # 访问器 - getter方法
    @property
    def name(self):
        return self._name

    # 访问器 - getter方法
    @property
    def age(self):
        return self._age

    # 修改器 - setter方法
    @age.setter
    def age(self, age):
        self._age = age

    def play(self):
        if self._age <= 16:
            print('%s正在玩飞行棋.' % self._name)
        else:
            print('%s正在玩斗地主.' % self._name)

def main():
    person = Person('王大锤', 12)
    person.play()
    person.age = 22
    person.play()
    # person.name = '白元芳' # AttributeError: can't set attribute
```

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

__slots__魔法

我们讲到这里，不知道大家是否已经意识到，Python是一门[动态语言](#)。通常，动态语言允许我们在程序运行时给对象绑定新的属性或方法，当然也可以对已经绑定的属性和方法进行解绑定。但是如果我们需要限定自定义类型的对象只能绑定某些属性，可以通过在类中定义__slots__变量来进行限定。需要注意的是__slots__的限定只对当前类的对象生效，对子类并不起任何作用。

```
class Person(object):

    # 限定Person对象只能绑定_name, _age和_gender属性
    __slots__ = ('_name', '_age', '_gender')

    def __init__(self, name, age):
        self._name = name
        self._age = age

    @property
    def name(self):
        return self._name

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, age):
        self._age = age

    def play(self):
        if self._age <= 16:
            print('%s正在玩飞行棋.' % self._name)
        else:
            print('%s正在玩斗地主.' % self._name)

    def main():
        person = Person('王大锤', 22)
        person.play()
        person._gender = '男'
        # AttributeError: 'Person' object has no attribute '_is_gay'
        # person._is_gay = True
```

静态方法和类方法

之前，我们在类中定义的方法都是对象方法，也就是说这些方法都是发送给对象的消息。实际上，我们写在类中的方法并不需要都是对象方法，例如我们定义一个“三角形”类，通过传入三条边长来构造三角形，并提供计

算周长和面积的方法，但是传入的三条边长未必能构造出三角形对象，因此我们可以先写一个方法来验证三条边长是否可以构成三角形，这个方法很显然就不是对象方法，因为在调用这个方法时三角形对象尚未创建出来（因为都不知道三条边能不能构成三角形），所以这个方法是属于三角形类而并不属于三角形对象的。我们可以使用静态方法来解决这类问题，代码如下所示。

```
from math import sqrt

class Triangle(object):

    def __init__(self, a, b, c):
        self._a = a
        self._b = b
        self._c = c

    @staticmethod
    def is_valid(a, b, c):
        return a + b > c and b + c > a and a + c > b

    def perimeter(self):
        return self._a + self._b + self._c

    def area(self):
        half = self.perimeter() / 2
        return sqrt(half * (half - self._a) *
                   (half - self._b) * (half - self._c))

def main():
    a, b, c = 3, 4, 5
    # 静态方法和类方法都是通过给类发消息来调用的
    if Triangle.is_valid(a, b, c):
        t = Triangle(a, b, c)
        print(t.perimeter())
        # 也可以通过给类发消息来调用对象方法但是要传入接收消息的对象作为参数
        # print(Triangle.perimeter(t))
        print(t.area())
        # print(Triangle.area(t))
    else:
        print('无法构成三角形。')

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

和静态方法比较类似，Python还可以在类中定义类方法，类方法的第一个参数约定名为cls，它代表的是当前类相关的信息的对象（类本身也是一个对象，有的地方也称之为类的元数据对象），通过这个参数我们可以获取和类相关的信息并且可以创建出类的对象，代码如下所示。

```
from time import time, localtime, sleep

class Clock(object):
    """数字时钟"""

    def __init__(self, hour=0, minute=0, second=0):
        self._hour = hour
        self._minute = minute
        self._second = second

    @classmethod
    def now(cls):
        ctime = localtime(time())
        return cls(ctime.tm_hour, ctime.tm_min, ctime.tm_sec)

    def run(self):
        """走字"""
        self._second += 1
        if self._second == 60:
            self._second = 0
            self._minute += 1
            if self._minute == 60:
                self._minute = 0
                self._hour += 1
                if self._hour == 24:
                    self._hour = 0

    def show(self):
        """显示时间"""
        return '%02d:%02d:%02d' % \
            (self._hour, self._minute, self._second)

def main():
    # 通过类方法创建对象并获取系统时间
    clock = Clock.now()
    while True:
        print(clock.show())
        sleep(1)
        clock.run()

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

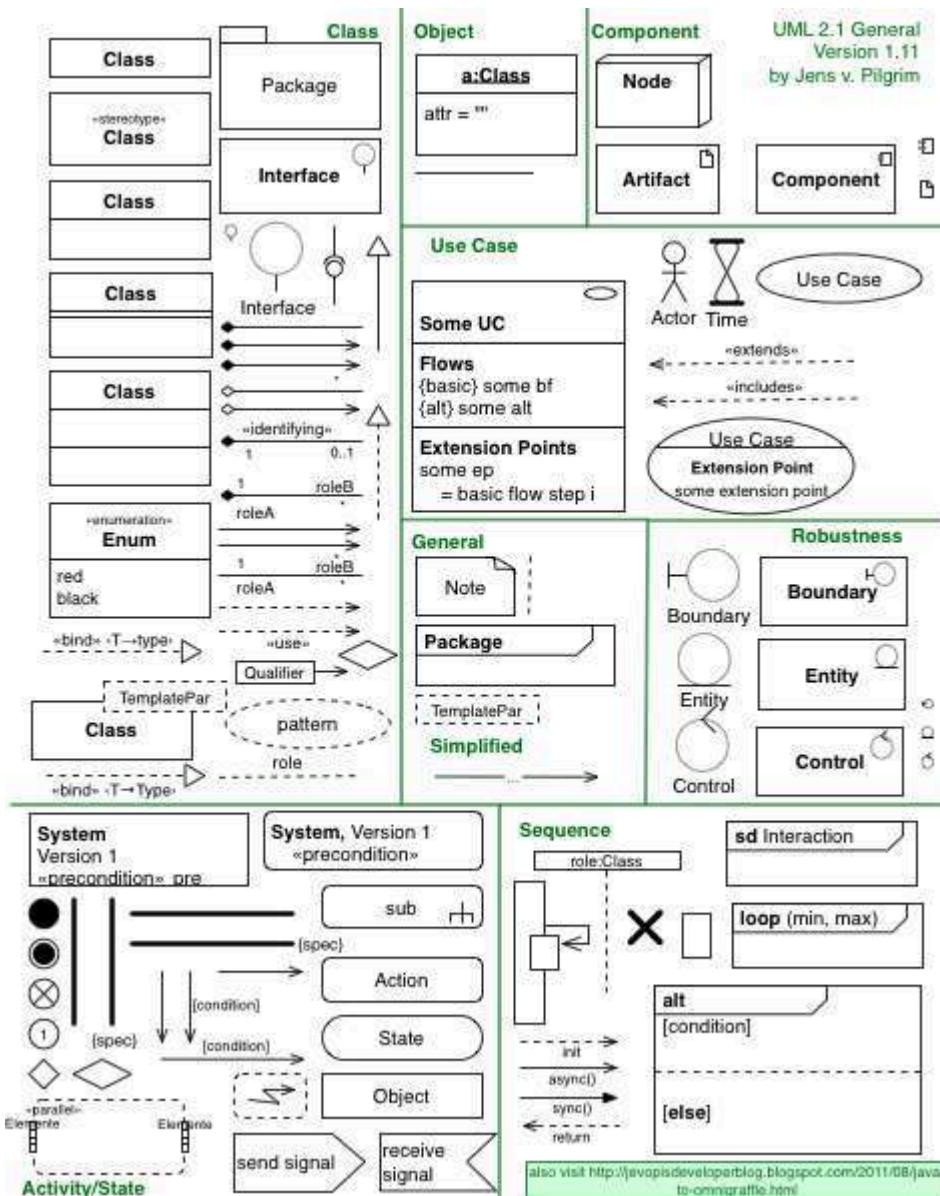
类之间的关系

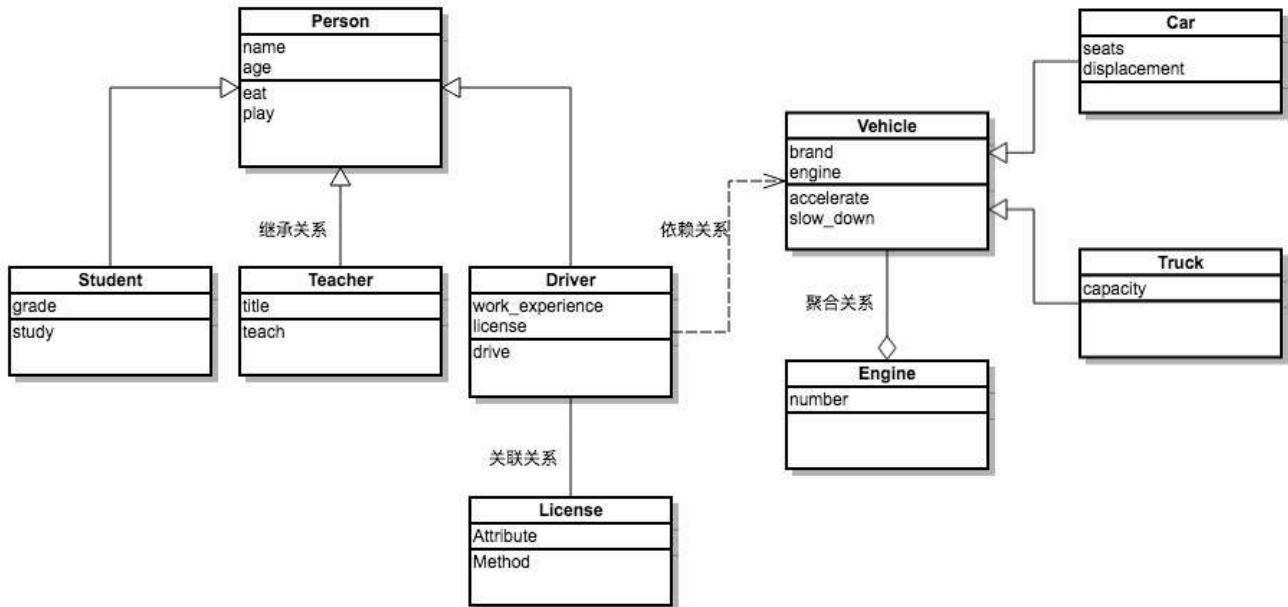
简单的说，类和类之间的关系有三种：is-a、has-a和use-a关系。

- is-a关系也叫继承或泛化，比如学生和人的关系、手机和电子产品的关系都属于继承关系。

- has-a关系通常称之为关联，比如部门和员工的关系，汽车和引擎的关系都属于关联关系；关联关系如果是整体和部分的关联，那么我们称之为聚合关系；如果整体进一步负责了部分的生命周期（整体和部分是不可分割的，同时同在也同时消亡），那么这种就是最强的关联关系，我们称之为合成关系。
- use-a关系通常称之为依赖，比如司机有一个驾驶的行为（方法），其中（的参数）使用到了汽车，那么司机和汽车的关系就是依赖关系。

我们可以使用一种叫做UML（统一建模语言）的东西来进行面向对象建模，其中一项重要的工作就是把类和类之间的关系用标准化的图形符号描述出来。关于UML我们在这里不做详细的介绍，有兴趣的读者可以自行阅读《UML面向对象设计基础》一书。





利用类之间的这些关系，我们可以在已有类的基础上来完成某些操作，也可以在已有类的基础上创建新的类，这些都是实现代码复用的重要手段。复用现有的代码不仅可以减少开发的工作量，也有利于代码的管理和维护，这是我们在日常工作中都会使用到的技术手段。

继承和多态

刚才我们提到了，可以在已有类的基础上创建新类，这其中的一种做法就是让一个类从另一个类那里将属性和方法直接继承下来，从而减少重复代码的编写。提供继承信息的我们称之为父类，也叫超类或基类；得到继承信息的我们称之为子类，也叫派生类或衍生类。子类除了继承父类提供的属性和方法，还可以定义自己特有的属性和方法，所以子类比父类拥有的更多的能力，在实际开发中，我们经常会用子类对象去替换掉一个父类对象，这是面向对象编程中一个常见的行为，对应的原则称之为[里氏替换原则](#)。下面我们先看一个继承的例子。

```

class Person(object):
    """人"""

    def __init__(self, name, age):
        self._name = name
        self._age = age

    @property
    def name(self):
        return self._name

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, age):
        self._age = age

    def play(self):
        print('%s正在愉快的玩耍.' % self._name)
  
```

```
def watch_av(self):
    if self._age >= 18:
        print('%s正在观看爱情动作片.' % self._name)
    else:
        print('%s只能观看《熊出没》.' % self._name)

class Student(Person):
    """学生"""

    def __init__(self, name, age, grade):
        super().__init__(name, age)
        self._grade = grade

    @property
    def grade(self):
        return self._grade

    @grade.setter
    def grade(self, grade):
        self._grade = grade

    def study(self, course):
        print('%s的%s正在学习%s.' % (self._grade, self._name, course))

class Teacher(Person):
    """老师"""

    def __init__(self, name, age, title):
        super().__init__(name, age)
        self._title = title

    @property
    def title(self):
        return self._title

    @title.setter
    def title(self, title):
        self._title = title

    def teach(self, course):
        print('%s%s正在讲%s.' % (self._name, self._title, course))

def main():
    stu = Student('王大锤', 15, '初三')
    stu.study('数学')
    stu.watch_av()
    t = Teacher('骆昊', 38, '砖家')
    t.teach('Python程序设计')
    t.watch_av()
```

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

子类在继承了父类的方法后，可以对父类已有的方法给出新的实现版本，这个动作称之为方法重写（override）。通过方法重写我们可以让父类的同一个行为在子类中拥有不同的实现版本，当我们调用这个经过子类重写的方法时，不同的子类对象会表现出不同的行为，这个就是多态（poly-morphism）。

```
from abc import ABCMeta, abstractmethod

class Pet(object, metaclass=ABCMeta):
    """宠物"""

    def __init__(self, nickname):
        self._nickname = nickname

    @abstractmethod
    def make_voice(self):
        """发出声音"""
        pass

class Dog(Pet):
    """狗"""

    def make_voice(self):
        print('%s: 汪汪汪...' % self._nickname)

class Cat(Pet):
    """猫"""

    def make_voice(self):
        print('%s: 喵...喵...' % self._nickname)

def main():
    pets = [Dog('旺财'), Cat('凯蒂'), Dog('大黄')]
    for pet in pets:
        pet.make_voice()

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

在上面的代码中，我们将Pet类处理成了一个抽象类，所谓抽象类就是不能够创建对象的类，这种类的存在就是专门为了让其他类去继承它。Python从语法层面并没有像Java或C#那样提供对抽象类的支持，但是我们可以通过abc模块的ABCMeta元类和abstractmethod包装器来达到抽象类的效果，如果一个类中存在抽象方法那么这个类就不能够实例化（创建对象）。上面的代码中，Dog和Cat两个子类分别对Pet类中的make_voice抽象方

法进行了重写并给出了不同的实现版本，当我们在`main`函数中调用该方法时，这个方法就表现出了多态行为（同样的方法做了不同的事情）。

综合案例

案例1：奥特曼打小怪兽。

```
from abc import ABCMeta, abstractmethod
from random import randint, randrange

class Fighter(object, metaclass=ABCMeta):
    """战斗者"""

    # 通过__slots__魔法限定对象可以绑定的成员变量
    __slots__ = ('_name', '_hp')

    def __init__(self, name, hp):
        """初始化方法

        :param name: 名字
        :param hp: 生命值
        """
        self._name = name
        self._hp = hp

    @property
    def name(self):
        return self._name

    @property
    def hp(self):
        return self._hp

    @hp.setter
    def hp(self, hp):
        self._hp = hp if hp >= 0 else 0

    @property
    def alive(self):
        return self._hp > 0

    @abstractmethod
    def attack(self, other):
        """攻击

        :param other: 被攻击的对象
        """
        pass

class Ultraman(Fighter):
```

"""奥特曼"""

```
__slots__ = ('_name', '_hp', '_mp')

def __init__(self, name, hp, mp):
    """初始化方法

    :param name: 名字
    :param hp: 生命值
    :param mp: 魔法值
    """
    super().__init__(name, hp)
    self._mp = mp

def attack(self, other):
    other.hp -= randint(15, 25)

def huge_attack(self, other):
    """究极必杀技(打掉对方至少50点或四分之三的血)

    :param other: 被攻击的对象

    :return: 使用成功返回True否则返回False
    """
    if self._mp >= 50:
        self._mp -= 50
        injury = other.hp * 3 // 4
        injury = injury if injury >= 50 else 50
        other.hp -= injury
        return True
    else:
        self.attack(other)
        return False

def magic_attack(self, others):
    """魔法攻击

    :param others: 被攻击的群体

    :return: 使用魔法成功返回True否则返回False
    """
    if self._mp >= 20:
        self._mp -= 20
        for temp in others:
            if temp.alive:
                temp.hp -= randint(10, 15)
        return True
    else:
        return False

def resume(self):
    """恢复魔法值"""

    incr_point = randint(1, 10)
    self._mp += incr_point
```

```
    return incr_point

    def __str__(self):
        return '~~~%s奥特曼~~~\n' % self._name + \
            '生命值: %d\n' % self._hp + \
            '魔法值: %d\n' % self._mp

class Monster(Fighter):
    """小怪兽"""

    __slots__ = ('_name', '_hp')

    def attack(self, other):
        other.hp -= randint(10, 20)

    def __str__(self):
        return '~~~%s小怪兽~~~\n' % self._name + \
            '生命值: %d\n' % self._hp

def is_any_alive(monsters):
    """判断有没有小怪兽是活着的"""
    for monster in monsters:
        if monster.alive > 0:
            return True
    return False

def select_alive_one(monsters):
    """选中一只活着的小怪兽"""
    monsters_len = len(monsters)
    while True:
        index = randrange(monsters_len)
        monster = monsters[index]
        if monster.alive > 0:
            return monster

def display_info(ultraman, monsters):
    """显示奥特曼和小怪兽的信息"""
    print(ultraman)
    for monster in monsters:
        print(monster, end=' ')

def main():
    u = Ultraman('路昊', 1000, 120)
    m1 = Monster('狄仁杰', 250)
    m2 = Monster('白元芳', 500)
    m3 = Monster('王大锤', 750)
    ms = [m1, m2, m3]
    fight_round = 1
    while u.alive and is_any_alive(ms):
```

```

print('=====第%02d回合===== ' % fight_round)
m = select_alive_one(ms) # 选中一只小怪兽
skill = randint(1, 10) # 通过随机数选择使用哪种技能
if skill <= 6: # 60%的概率使用普通攻击
    print('%s使用普通攻击打了%s.' % (u.name, m.name))
    u.attack(m)
    print('%s的魔法值恢复了%d点.' % (u.name, u.resume()))
elif skill <= 9: # 30%的概率使用魔法攻击(可能因魔法值不足而失败)
    if u.magic_attack(ms):
        print('%s使用了魔法攻击.' % u.name)
    else:
        print('%s使用魔法失败.' % u.name)
else: # 10%的概率使用究极必杀技(如果魔法值不足则使用普通攻击)
    if u.huge_attack(m):
        print('%s使用究极必杀技虐了%s.' % (u.name, m.name))
    else:
        print('%s使用普通攻击打了%s.' % (u.name, m.name))
        print('%s的魔法值恢复了%d点.' % (u.name, u.resume()))
if m.alive > 0: # 如果选中的小怪兽没有死就回击奥特曼
    print('%s回击了%s.' % (m.name, u.name))
    m.attack(u)
display_info(u, ms) # 每个回合结束后显示奥特曼和小怪兽的信息
fight_round += 1
print('\n=====战斗结束!=====\\n')
if u.alive > 0:
    print('%s奥特曼胜利!' % u.name)
else:
    print('小怪兽胜利!')

if __name__ == '__main__':
    main()

```

案例2：扑克游戏。

```

import random

class Card(object):
    """一张牌"""

    def __init__(self, suite, face):
        self._suite = suite
        self._face = face

    @property
    def face(self):
        return self._face

    @property
    def suite(self):

```

```
    return self._suite

def __str__(self):
    if self._face == 1:
        face_str = 'A'
    elif self._face == 11:
        face_str = 'J'
    elif self._face == 12:
        face_str = 'Q'
    elif self._face == 13:
        face_str = 'K'
    else:
        face_str = str(self._face)
    return '%s%s' % (self._suite, face_str)

def __repr__(self):
    return self.__str__()

class Poker(object):
    """一副牌"""

    def __init__(self):
        self._cards = [Card(suite, face)
                      for suite in '♠♥♦♣'
                      for face in range(1, 14)]
        self._current = 0

    @property
    def cards(self):
        return self._cards

    def shuffle(self):
        """洗牌(随机乱序)"""
        self._current = 0
        random.shuffle(self._cards)

    @property
    def next(self):
        """发牌"""
        card = self._cards[self._current]
        self._current += 1
        return card

    @property
    def has_next(self):
        """还有没有牌"""
        return self._current < len(self._cards)

class Player(object):
    """玩家"""

    def __init__(self, name):
```

```
self._name = name
self._cards_on_hand = []

@property
def name(self):
    return self._name

@property
def cards_on_hand(self):
    return self._cards_on_hand

def get(self, card):
    """摸牌"""
    self._cards_on_hand.append(card)

def arrange(self, card_key):
    """玩家整理手上的牌"""
    self._cards_on_hand.sort(key=card_key)

# 排序规则-先根据花色再根据点数排序
def get_key(card):
    return (card.suite, card.face)

def main():
    p = Poker()
    p.shuffle()
    players = [Player('东邪'), Player('西毒'), Player('南帝'), Player('北丐')]
    for _ in range(13):
        for player in players:
            player.get(p.next)
    for player in players:
        print(player.name + ':', end=' ')
        player.arrange(get_key)
        print(player.cards_on_hand)

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

说明：大家可以自己尝试在上面代码的基础上写一个简单的扑克游戏，例如21点(Black Jack)，游戏的规则可以自己在网上找一找。

案例3：工资结算系统。

```
"""

某公司有三种类型的员工 分别是部门经理、程序员和销售员
需要设计一个工资结算系统 根据提供的员工信息来计算月薪
部门经理的月薪是每月固定15000元
程序员的月薪按本月工作时间计算 每小时150元
```

销售员的月薪是1200元的底薪加上销售额5%的提成

```
"""
from abc import ABCMeta, abstractmethod

class Employee(object, metaclass=ABCMeta):
    """员工"""

    def __init__(self, name):
        """
        初始化方法

        :param name: 姓名
        """
        self._name = name

    @property
    def name(self):
        return self._name

    @abstractmethod
    def get_salary(self):
        """
        获得月薪

        :return: 月薪
        """
        pass

class Manager(Employee):
    """部门经理"""

    def get_salary(self):
        return 15000.0

class Programmer(Employee):
    """程序员"""

    def __init__(self, name, working_hour=0):
        super().__init__(name)
        self._working_hour = working_hour

    @property
    def working_hour(self):
        return self._working_hour

    @working_hour.setter
    def working_hour(self, working_hour):
        self._working_hour = working_hour if working_hour > 0 else 0

    def get_salary(self):
        return 150.0 * self._working_hour
```

```
class Salesman(Employee):
    """销售员"""

    def __init__(self, name, sales=0):
        super().__init__(name)
        self._sales = sales

    @property
    def sales(self):
        return self._sales

    @sales.setter
    def sales(self, sales):
        self._sales = sales if sales > 0 else 0

    def get_salary(self):
        return 1200.0 + self._sales * 0.05

def main():
    emps = [
        Manager('刘备'), Programmer('诸葛亮'),
        Manager('曹操'), Salesman('荀彧'),
        Salesman('吕布'), Programmer('张辽'),
        Programmer('赵云')
    ]
    for emp in emps:
        if isinstance(emp, Programmer):
            emp.working_hour = int(input('请输入%s本月工作时间: ' % emp.name))
        elif isinstance(emp, Salesman):
            emp.sales = float(input('请输入%s本月销售额: ' % emp.name))
    # 同样是接收get_salary这个消息但是不同的员工表现出了不同的行为(多态)
    print('%s本月工资为: %s元' %
          (emp.name, emp.get_salary()))

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

图形用户界面和游戏开发

基于tkinter模块的GUI

GUI是图形用户界面的缩写，图形化的用户界面对使用过计算机的人来说应该都不陌生，在此也无需进行赘述。Python默认的GUI开发模块是tkinter（在Python 3以前的版本中名为Tkinter），从这个名字就可以看出它是基于Tk的，Tk是一个工具包，最初是为Tcl设计的，后来被移植到很多其他的脚本语言中，它提供了跨平台的GUI控件。当然Tk并不是最新和最好的选择，也没有功能特别强大的GUI控件，事实上，开发GUI应用并不是Python最擅长的工作，如果真的需要使用Python开发GUI应用，wxPython、PyQt、PyGTK等模块都是不错的选择。

基本上使用tkinter来开发GUI应用需要以下5个步骤：

1. 导入tkinter模块中我们需要的东西。
2. 创建一个顶层窗口对象并用它来承载整个GUI应用。
3. 在顶层窗口对象上添加GUI组件。
4. 通过代码将这些GUI组件的功能组织起来。
5. 进入主事件循环(main loop)。

下面的代码演示了如何使用tkinter做一个简单的GUI应用。

```
import tkinter
import tkinter.messagebox

def main():
    flag = True

    # 修改标签上的文字
    def change_label_text():
        nonlocal flag
        flag = not flag
        color, msg = ('red', 'Hello, world!')\
            if flag else ('blue', 'Goodbye, world!')
        label.config(text=msg, fg=color)

    # 确认退出
    def confirm_to_quit():
        if tkinter.messagebox.askokcancel('温馨提示', '确定要退出吗?'):
            top.quit()

    # 创建顶层窗口
    top = tkinter.Tk()
    # 设置窗口大小
    top.geometry('240x160')
    # 设置窗口标题
    top.title('小游戏')
    # 创建标签对象并添加到顶层窗口
    label = tkinter.Label(top, text='Hello, world!', font='Arial -32', fg='red')
    label.pack(expand=1)
```

```

# 创建一个装按钮的容器
panel = tkinter.Frame(top)
# 创建按钮对象 指定添加到哪个容器中 通过command参数绑定事件回调函数
button1 = tkinter.Button(panel, text='修改', command=change_label_text)
button1.pack(side='left')
button2 = tkinter.Button(panel, text='退出', command=confirm_to_quit)
button2.pack(side='right')
panel.pack(side='bottom')
# 开启主事件循环
tkinter.mainloop()

if __name__ == '__main__':
    main()

```

需要说明的是，GUI应用通常是事件驱动式的，之所以要进入主事件循环就是要监听鼠标、键盘等各种事件的发生并执行对应的代码对事件进行处理，因为事件会持续的发生，所以需要这样的一个循环一直运行着等待下一个事件的发生。另一方面，Tk为控件的摆放提供了三种布局管理器，通过布局管理器可以对控件进行定位，这三种布局管理器分别是：Placer（开发者提供控件的大小和摆放位置）、Packer（自动将控件填充到合适的位置）和Grid（基于网格坐标来摆放控件），此处不进行赘述。

使用Pygame进行游戏开发

Pygame是一个开源的Python模块，专门用于多媒体应用（如电子游戏）的开发，其中包含对图像、声音、视频、事件、碰撞等的支持。Pygame建立在[SDL](#)的基础上，SDL是一套跨平台的多媒体开发库，用C语言实现，被广泛的应用于游戏、模拟器、播放器等的开发。而Pygame让游戏开发者不再被底层语言束缚，可以更多的关注游戏的功能和逻辑。

下面我们来完成一个简单的小游戏，游戏的名字叫“大球吃小球”，当然完成这个游戏并不是重点，学会使用Pygame也不是重点，最重要的我们要在这个过程中体会如何使用前面讲解的面向对象程序设计，学会用这种编程思想去解决现实中的问题。

制作游戏窗口

```

import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:

```

```

running = False

if __name__ == '__main__':
    main()

```

在窗口中绘图

可以通过pygame中draw模块的函数在窗口上绘图，可以绘制的图形包括：线条、矩形、多边形、圆、椭圆、圆弧等。需要说明的是，屏幕坐标系是将屏幕左上角设置为坐标原点(0, 0)，向右是x轴的正向，向下是y轴的正向，在表示位置或者设置尺寸的时候，我们默认的单位都是像素。所谓像素就是屏幕上的一个点，你可以用浏览图片的软件试着将一张图片放大若干倍，就可以看到这些点。pygame中表示颜色用的是色光三原色表示法，即通过一个元组或列表来指定颜色的RGB值，每个值都在0~255之间，因为是每种原色都用一个8位 (bit) 的值来表示，三种颜色相当于一共由24位构成，这也就是常说的“24位颜色表示法”。

```

import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 设置窗口的背景色(颜色是由红绿蓝三原色构成的元组)
    screen.fill((242, 242, 242))
    # 绘制一个圆(参数分别是：屏幕，颜色，圆心位置，半径，0表示填充圆)
    pygame.draw.circle(screen, (255, 0, 0), (100, 100), 30, 0)
    # 刷新当前窗口(渲染窗口将绘制的图像呈现出来)
    pygame.display.flip()
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

    if __name__ == '__main__':
        main()

```

加载图像

如果需要直接加载图像到窗口上，可以使用pygame中image模块的函数来加载图像，再通过之前获得的窗口对象的blit方法渲染图像，代码如下所示。

```
import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 设置窗口的背景色(颜色是由红绿蓝三原色构成的元组)
    screen.fill((255, 255, 255))
    # 通过指定的文件名加载图像
    ball_image = pygame.image.load('./res/ball.png')
    # 在窗口上渲染图像
    screen.blit(ball_image, (50, 50))
    # 刷新当前窗口(渲染窗口将绘制的图像呈现出来)
    pygame.display.flip()
    running = True
    # 开启一个事件循环处理发生的事件
    while running:
        # 从消息队列中获取事件并对事件进行处理
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

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

实现动画效果

说到动画这个词大家都不会陌生，事实上要实现动画效果，本身的原理也非常简单，就是将不连续的图片连续的播放，只要每秒钟达到了一定的帧数，那么就可以做出比较流畅的动画效果。如果要让上面代码中的小球动起来，可以将小球的位置用变量来表示，并在循环中修改小球的位置再刷新整个窗口即可。

```
import pygame

def main():
    # 初始化导入的pygame中的模块
    pygame.init()
    # 初始化用于显示的窗口并设置窗口尺寸
    screen = pygame.display.set_mode((800, 600))
    # 设置当前窗口的标题
    pygame.display.set_caption('大球吃小球')
    # 定义变量来表示小球在屏幕上的位置
    x, y = 50, 50
    running = True
    # 开启一个事件循环处理发生的事件
```

```

while running:
    # 从消息队列中获取事件并对事件进行处理
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    screen.fill((255, 255, 255))
    pygame.draw.circle(screen, (255, 0, 0,), (x, y), 30, 0)
    pygame.display.flip()
    # 每隔50毫秒就改变小球的位置再刷新窗口
    pygame.time.delay(50)
    x, y = x + 5, y + 5

if __name__ == '__main__':
    main()

```

碰撞检测

通常一个游戏中会有很多对象出现，而这些对象之间的“碰撞”在所难免，比如炮弹击中了飞机、箱子撞到了地面等。碰撞检测在绝大多数的游戏中都是一个必须得处理的至关重要的问题，pygame的sprite（动画精灵）模块就提供了对碰撞检测的支持，这里我们暂时不介绍sprite模块提供的功能，因为要检测两个小球有没有碰撞其实非常简单，只需要检查球心的距离有没有小于两个球的半径之和。为了制造出更多的小球，我们可以通过对鼠标事件的处理，在点击鼠标的位置创建颜色、大小和移动速度都随机的小球，当然要做到这一点，我们可以把之前学习到的面向对象的知识应用起来。

```

from enum import Enum, unique
from math import sqrt
from random import randint

import pygame

@unique
class Color(Enum):
    """颜色"""

    RED = (255, 0, 0)
    GREEN = (0, 255, 0)
    BLUE = (0, 0, 255)
    BLACK = (0, 0, 0)
    WHITE = (255, 255, 255)
    GRAY = (242, 242, 242)

    @staticmethod
    def random_color():
        """获得随机颜色"""
        r = randint(0, 255)
        g = randint(0, 255)
        b = randint(0, 255)
        return (r, g, b)

```

```

class Ball(object):
    """球"""

    def __init__(self, x, y, radius, sx, sy, color=Color.RED):
        """初始化方法"""
        self.x = x
        self.y = y
        self.radius = radius
        self.sx = sx
        self.sy = sy
        self.color = color
        self.alive = True

    def move(self, screen):
        """移动"""
        self.x += self.sx
        self.y += self.sy
        if self.x - self.radius <= 0 or \
            self.x + self.radius >= screen.get_width():
            self.sx = -self.sx
        if self.y - self.radius <= 0 or \
            self.y + self.radius >= screen.get_height():
            self.sy = -self.sy

    def eat(self, other):
        """吃其他球"""
        if self.alive and other.alive and self != other:
            dx, dy = self.x - other.x, self.y - other.y
            distance = sqrt(dx ** 2 + dy ** 2)
            if distance < self.radius + other.radius \
                and self.radius > other.radius:
                other.alive = False
                self.radius = self.radius + int(other.radius * 0.146)

    def draw(self, screen):
        """在窗口上绘制球"""
        pygame.draw.circle(screen, self.color,
                           (self.x, self.y), self.radius, 0)

```

事件处理

可以在事件循环中对鼠标事件进行处理，通过事件对象的`type`属性可以判定事件类型，再通过`pos`属性就可以获得鼠标点击的位置。如果要处理键盘事件也是在这个地方，做法与处理鼠标事件类似。

```

def main():
    # 定义用来装所有球的容器
    balls = []
    # 初始化导入的pygame中的模块
    pygame.init()

```

```

# 初始化用于显示的窗口并设置窗口尺寸
screen = pygame.display.set_mode((800, 600))
# 设置当前窗口的标题
pygame.display.set_caption('大球吃小球')
running = True
# 开启一个事件循环处理发生的事件
while running:
    # 从消息队列中获取事件并对事件进行处理
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # 处理鼠标事件的代码
    if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
        # 获得点击鼠标的位置
        x, y = event.pos
        radius = randint(10, 100)
        sx, sy = randint(-10, 10), randint(-10, 10)
        color = Color.random_color()
        # 在点击鼠标的位置创建一个球(大小、速度和颜色随机)
        ball = Ball(x, y, radius, sx, sy, color)
        # 将球添加到列表容器中
        balls.append(ball)
    screen.fill((255, 255, 255))
    # 取出容器中的球 如果没被吃掉就绘制 被吃掉了就移除
    for ball in balls:
        if ball.alive:
            ball.draw(screen)
        else:
            balls.remove(ball)
    pygame.display.flip()
    # 每隔50毫秒就改变球的位置再刷新窗口
    pygame.time.delay(50)
    for ball in balls:
        ball.move(screen)
    # 检查球有没有吃到其他的球
    for other in balls:
        ball.eat(other)

if __name__ == '__main__':
    main()

```

上面的两段代码合在一起，我们就完成了“大球吃小球”的游戏（如下图所示），准确的说它算不上一个游戏，但是做一个小游戏的基本知识我们已经通过这个例子告诉大家了，有了这些知识已经可以开始你的小游戏开发之旅了。其实上面的代码中还有很多值得改进的地方，比如刷新窗口以及让球移动起来的代码并不应该放在事件循环中，等学习了多线程的知识后，用一个后台线程来处理这些事可能是更好的选择。如果希望获得更好的用户体验，我们还可以在游戏中加入背景音乐以及在球与球发生碰撞时播放音效，利用pygame的mixer和music模块，我们可以很容易的做到这一点，大家可以自行了解这方面的知识。事实上，想了解更多的关于pygame的知识，最好的教程是[pygame的官方网站](#)，如果英语没毛病就可以赶紧去看看啦。如果想开发3D游戏，pygame就显得力不从心了，对3D游戏开发如果有兴趣的读者不妨看看[Panda3D](#)。

文件和异常

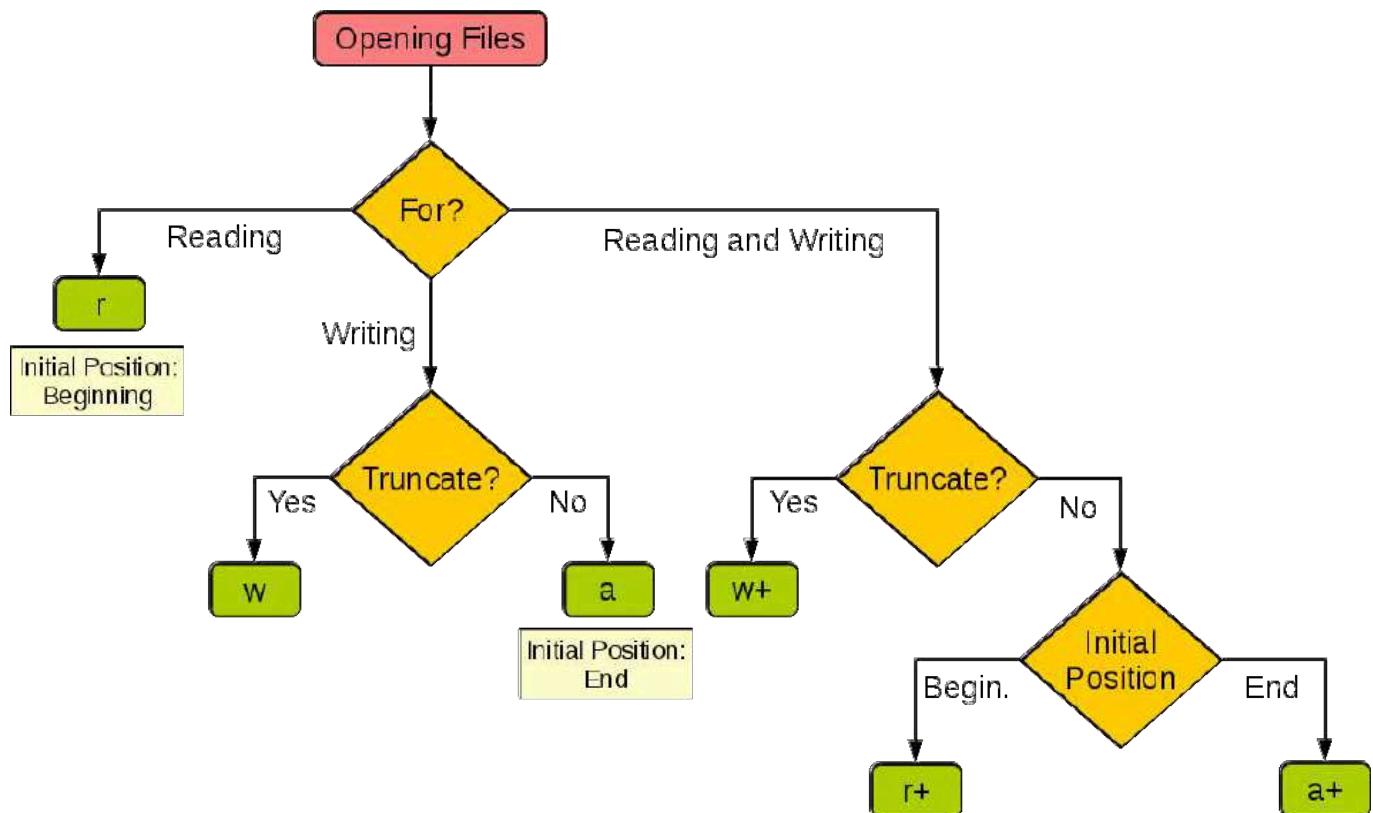
实际开发中常常会遇到对数据进行持久化操作的场景，而实现数据持久化最直接简单的方式就是将数据保存到文件中。说到“文件”这个词，可能需要先科普一下关于文件系统的知识，但是这里我们并不浪费笔墨介绍这个概念，请大家自行通过维基百科进行了解。

在Python中实现文件的读写操作其实非常简单，通过Python内置的`open`函数，我们可以指定文件名、操作模式、编码信息等来获得操作文件的对象，接下来就可以对文件进行读写操作了。这里所说的操作模式是指要打开什么样的文件（字符文件还是二进制文件）以及做什么样的操作（读、写还是追加），具体的如下表所示。

操作模式 具体含义

'r'	读取（默认）
'w'	写入（会先截断之前的内容）
'x'	写入，如果文件已经存在会产生异常
'a'	追加，将内容写入到已有文件的末尾
'b'	二进制模式
't'	文本模式（默认）
'+'	更新（既可以读又可以写）

下面这张图来自于菜鸟教程网站，它展示了如果根据应用程序的需要来设置操作模式。



读写文本文件

读取文本文件时，需要在使用`open`函数时指定好带路径的文件名（可以使用相对路径或绝对路径）并将文件模式设置为`'r'`（如果不指定，默认值也是`'r'`），然后通过`encoding`参数指定编码（如果不指定，默认值是`None`，那么在读取文件时使用的是操作系统默认的编码），如果不能保证保存文件时使用的编码方式与`encoding`参数指定的编码方式是一致的，那么就可能因无法解码字符而导致读取失败。下面的例子演示了如何读取一个纯文本文件。

```
def main():
    f = open('致橡树.txt', 'r', encoding='utf-8')
    print(f.read())
    f.close()

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

请注意上面的代码，如果`open`函数指定的文件并不存在或者无法打开，那么将引发异常状况导致程序崩溃。为了让代码有一定的健壮性和容错性，我们可以使用Python的异常机制对可能在运行时发生状况的代码进行适当的处理，如下所示。

```
def main():
    f = None
    try:
        f = open('致橡树.txt', 'r', encoding='utf-8')
        print(f.read())
    except FileNotFoundError:
        print('无法打开指定的文件!')
    except LookupError:
        print('指定了未知的编码!')
    except UnicodeDecodeError:
        print('读取文件时解码错误!')
    finally:
        if f:
            f.close()

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

在Python中，我们可以将那些在运行时可能会出现状况的代码放在`try`代码块中，在`try`代码块的后面可以跟上一个或多个`except`来捕获可能出现的异常状况。例如在上面读取文件的过程中，文件找不到会引发`FileNotFoundException`，指定了未知的编码会引发`LookupError`，而如果读取文件时无法按指定方式解码会引发`UnicodeDecodeError`，我们在`try`后面跟上了三个`except`分别处理这三种不同的异常状况。最后我们使用`finally`代码块来关闭打开的文件，释放掉程序中获取的外部资源，由于`finally`块的代码不论程序正常还是异常都会执行到（甚至是调用了`sys`模块的`exit`函数退出Python环境，`finally`块都会被执行，因为`exit`函数实质上是引发了`SystemExit`异常），因此我们通常把`finally`块称为“总是执行代码块”，它最适合用来做释放外部资源的操作。如果不愿意在`finally`代码块中关闭文件对象释放资源，也可以使用上下文语法，通过`with`关键字指定文件对象的上下文环境并在离开上下文环境时自动释放文件资源，代码如下所示。

```
def main():
    try:
        with open('致橡树.txt', 'r', encoding='utf-8') as f:
            print(f.read())
    except FileNotFoundError:
        print('无法打开指定的文件!')
    except LookupError:
        print('指定了未知的编码!')
    except UnicodeDecodeError:
        print('读取文件时解码错误!')

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

除了使用文件对象的`read`方法读取文件之外，还可以使用`for-in`循环逐行读取或者用`readlines`方法将文件按行读取到一个列表容器中，代码如下所示。

```
import time

def main():
    # 一次性读取整个文件内容
    with open('致橡树.txt', 'r', encoding='utf-8') as f:
        print(f.read())

    # 通过for-in循环逐行读取
    with open('致橡树.txt', mode='r') as f:
        for line in f:
            print(line, end=' ')
            time.sleep(0.5)
    print()

    # 读取文件按行读取到列表中
    with open('致橡树.txt') as f:
        lines = f.readlines()
    print(lines)

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

要将文本信息写入文件也非常简单，在使用`open`函数时指定好文件名并将文件模式设置为`'w'`即可。注意如果需要对文件内容进行追加式写入，应该将模式设置为`'a'`。如果要写入的文件不存在会自动创建文件而不是引发异常。下面的例子演示了如何将1-9999之间的素数分别写入三个文件中（1-99之间的素数保存在a.txt中，100-999之间的素数保存在b.txt中，1000-9999之间的素数保存在c.txt中）。

```

from math import sqrt

def is_prime(n):
    """判断素数的函数"""
    assert n > 0
    for factor in range(2, int(sqrt(n)) + 1):
        if n % factor == 0:
            return False
    return True if n != 1 else False

def main():
    filenames = ('a.txt', 'b.txt', 'c.txt')
    fs_list = []
    try:
        for filename in filenames:
            fs_list.append(open(filename, 'w', encoding='utf-8'))
        for number in range(1, 10000):
            if is_prime(number):
                if number < 100:
                    fs_list[0].write(str(number) + '\n')
                elif number < 1000:
                    fs_list[1].write(str(number) + '\n')
                else:
                    fs_list[2].write(str(number) + '\n')
    except IOError as ex:
        print(ex)
        print('写文件时发生错误!')
    finally:
        for fs in fs_list:
            fs.close()
    print('操作完成!')

if __name__ == '__main__':
    main()

```

读写二进制文件

知道了如何读写文本文件要读写二进制文件也就很简单了，下面的代码实现了复制图片文件的功能。

```

def main():
    try:
        with open('guido.jpg', 'rb') as fs1:
            data = fs1.read()
            print(type(data)) # <class 'bytes'>
        with open('吉多.jpg', 'wb') as fs2:
            fs2.write(data)
    except FileNotFoundError as e:

```

```

        print('指定的文件无法打开.')
    except IOError as e:
        print('读写文件时出现错误.')
    print('程序执行结束.')

if __name__ == '__main__':
    main()

```

读写JSON文件

通过上面的讲解，我们已经知道如何将文本数据和二进制数据保存到文件中，那么这里还有一个问题，如果希望把一个列表或者一个字典中的数据保存到文件中又该怎么做呢？答案是将数据以JSON格式进行保存。JSON是“JavaScript Object Notation”的缩写，它本来是JavaScript语言中创建对象的一种字面量语法，现在已经被广泛的应用于跨平台跨语言的数据交换，原因很简单，因为JSON也是纯文本，任何系统任何编程语言处理纯文本都是没有问题的。目前JSON基本上已经取代了XML作为异构系统间交换数据的事实标准。关于JSON的知识，更多的可以参考[JSON的官方网站](#)，从这个网站也可以了解到每种语言处理JSON数据格式可以使用的工具或三方库，下面是一个JSON的简单例子。

```
{
    "name": "骆昊",
    "age": 38,
    "qq": 957658,
    "friends": ["王大锤", "白元芳"],
    "cars": [
        {"brand": "BYD", "max_speed": 180},
        {"brand": "Audi", "max_speed": 280},
        {"brand": "Benz", "max_speed": 320}
    ]
}
```

可能大家已经注意到了，上面的JSON跟Python中的字典其实是一样的，事实上JSON的数据类型和Python的数据类型是很容易找到对应关系的，如下面两张表所示。

JSON	Python
object	dict
array	list
string	str
number (int / real)	int / float
true / false	True / False
null	None

Python	JSON
dict	object

Python	JSON
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True / False	true / false
None	null

我们使用Python中的json模块就可以将字典或列表以JSON格式保存到文件中，代码如下所示。

```
import json

def main():
    mydict = {
        'name': '骆昊',
        'age': 38,
        'qq': 957658,
        'friends': ['王大锤', '白元芳'],
        'cars': [
            {'brand': 'BYD', 'max_speed': 180},
            {'brand': 'Audi', 'max_speed': 280},
            {'brand': 'Benz', 'max_speed': 320}
        ]
    }
    try:
        with open('data.json', 'w', encoding='utf-8') as fs:
            json.dump(mydict, fs)
    except IOError as e:
        print(e)
    print('保存数据完成!')

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

json模块主要有四个比较重要的函数，分别是：

- `dump` - 将Python对象按照JSON格式序列化到文件中
- `dumps` - 将Python对象处理成JSON格式的字符串
- `load` - 将文件中的JSON数据反序列化成对象
- `loads` - 将字符串的内容反序列化成Python对象

这里出现了两个概念，一个叫序列化，一个叫反序列化。自由的百科全书[维基百科](#)上对这两个概念是这样解释的：“序列化 (serialization) 在计算机科学的数据处理中，是指将数据结构或对象状态转换为可以存储或传输的形式，这样在需要的时候能够恢复到原先的状态，而且通过序列化的数据重新获取字节时，可以利用这些字

节来产生原始对象的副本（拷贝）。与这个过程相反的动作，即从一系列字节中提取数据结构的操作，就是反序列化（deserialization）”。

目前绝大多数网络数据服务（或称之为网络API）都是基于[HTTP协议](#)提供JSON格式的数据，关于HTTP协议的相关知识，可以看看阮一峰老师的[《HTTP协议入门》](#)，如果想了解国内的网络数据服务，可以看看[聚合数据](#)和[阿凡达数据](#)等网站，国外的可以看看[{API}Search](#)网站。下面的例子演示了如何使用[requests](#)模块（封装得足够好的第三方网络访问模块）访问网络API获取国内新闻，如何通过[json](#)模块解析JSON数据并显示新闻标题，这个例子使用了[天行数据](#)提供的国内新闻数据接口，其中的APIKey需要自己到该网站申请。

```
import requests
import json

def main():
    resp = requests.get('http://api.tianapi.com/guonei/?key=APIKey&num=10')
    data_model = json.loads(resp.text)
    for news in data_model['newslist']:
        print(news['title'])

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

在Python中要实现序列化和反序列化除了使用[json](#)模块之外，还可以使用[pickle](#)和[shelve](#)模块，但是这两个模块是使用特有的序列化协议来序列化数据，因此序列化后的数据只能被Python识别。关于这两个模块的相关知识可以自己看看网络上的资料。另外，如果要了解更多的关于Python异常机制的知识，可以看看[segmentfault](#)上面的文章[《总结：Python中的异常处理》](#)，这篇文章不仅介绍了Python中异常机制的使用，还总结了一系列的最佳实践，很值得一读。

使用正则表达式

正则表达式相关知识

在编写处理字符串的程序或网页时，经常会有查找符合某些复杂规则的字符串的需要，正则表达式就是用于描述这些规则的工具，换句话说正则表达式是一种工具，它定义了字符串的匹配模式（如何检查一个字符串是否有跟某种模式匹配的部分或者从一个字符串中将与模式匹配的部分提取出来或者替换掉）。如果你在Windows操作系统中使用过文件查找并且在指定文件名时使用过通配符（*和？），那么正则表达式也是与之类似的用来进行文本匹配的工具，只不过比起通配符正则表达式更强大，它能更精确地描述你的需求（当然你付出的代价是书写一个正则表达式比打出一个通配符要复杂得多，要知道任何给你带来好处的东西都是有代价的，就如同学习一门编程语言一样），比如你可以编写一个正则表达式，用来查找所有以0开头，后面跟着2-3个数字，然后是一个连字号“-”，最后是7或8位数字的字符串（像028-12345678或0813-7654321），这不就是国内的座机号码吗。最初计算机是为了做数学运算而诞生的，处理的信息基本上都是数值，而今天我们在日常工作中处理的信息基本上都是文本数据，我们希望计算机能够识别和处理符合某些模式的文本，正则表达式就显得非常重了。今天几乎所有的编程语言都提供了对正则表达式操作的支持，Python通过标准库中的re模块来支持正则表达式操作。

我们可以考虑下面一个问题：我们从某个地方（可能是一个文本文件，也可能是网络上的一则新闻）获得了一个字符串，希望在字符串中找出手机号和座机号。当然我们可以设定手机号是11位的数字（注意并不是随机的11位数字，因为你没有见过“25012345678”这样的手机号吧）而座机号跟上一段中描述的模式相同，如果不使用正则表达式要完成这个任务就会很麻烦。

关于正则表达式的相关知识，大家可以阅读一篇非常有名的博客叫[《正则表达式30分钟入门教程》](#)，读完这篇文章后你就可以看懂下面的表格，这是我们对正则表达式中的一些基本符号进行的扼要总结。

符号	解释	示例	说明
.	匹配任意字符	b.t	可以匹配bat / but / b#t / b1t等
\w	匹配字母/数字/下划线	b\wt	可以匹配bat / b1t / b_t等 但不能匹配b#t
\s	匹配空白字符（包括\r、\n、\t等）	love\syou	可以匹配love you
\d	匹配数字	\d\d	可以匹配01 / 23 / 99等
\b	匹配单词的边界	\bThe\b	
^	匹配字符串的开始	^The	可以匹配The开头的字符串
\$	匹配字符串的结束	.exe\$	可以匹配.exe结尾的字符串
\W	匹配非字母/数字/下划线	b\Wt	可以匹配b#t / b@t等 但不能匹配but / b1t / b_t等
\S	匹配非空白字符	love\Syou	可以匹配love#you等 但不能匹配love you
\D	匹配非数字	\d\D	可以匹配9a / 3# / 0F等
\B	匹配非单词边界	\Bio\B	

符号	解释	示例	说明
[]	匹配来自字符集的任意单一字符	[aeiou]	可以匹配任一元音字母字符
[^]	匹配不在字符集中的任意单一字符	[^aeiou]	可以匹配任一非元音字母字符
*	匹配0次或多次	\w*	
+	匹配1次或多次	\w+	
?	匹配0次或1次	\w?	
{N}	匹配N次	\w{3}	
{M,}	匹配至少M次	\w{3,}	
{M,N}	匹配至少M次至多N次	\w{3,6}	
	分支	foo bar	可以匹配foo或者bar
(?#)	注释		
(exp)	匹配exp并捕获到自动命名的组中		
(? <name>exp)</name>	匹配exp并捕获到名为name的组中		
(?:exp)	匹配exp但是不捕获匹配的文本		
(?=exp)	匹配exp前面的位置	\b\w+(?=ing)	可以匹配I'm dancing中的danc
(?<=exp)	匹配exp后面的位置	(? <name>=\bdanc)\w+\b</name>	可以匹配I love dancing and reading中的第一个ing
(?!exp)	匹配后面不是exp的位置		
(?<!exp)	匹配前面不是exp的位置		
*?	重复任意次，但尽可能少重复	a.*b a.*?b	将正则表达式应用于aabab，前者会匹配整个字符串aabab，后者会匹配aab和ab两个字符串
+?	重复1次或多次，但尽可能少重复		
??	重复0次或1次，但尽可能少重复		
{M,N?}	重复M到N次，但		

符号	解释	示例	说明
	尽可能少重复		
{M,}?	重复M次以上, 但尽可能少重复		

说明: 如果需要匹配的字符是正则表达式中的特殊字符, 那么可以使用\进行转义处理, 例如想匹配小数点可以写成\\.就可以了, 因为直接写.会匹配任意字符; 同理, 想匹配圆括号必须写成\\(和\\), 否则圆括号被视为正则表达式中的分组。

Python对正则表达式的支持

Python提供了re模块来支持正则表达式相关操作, 下面是re模块中的核心函数。

函数	说明
compile(pattern, flags=0)	编译正则表达式返回正则表达式对象
match(pattern, string, flags=0)	用正则表达式匹配字符串 成功返回匹配对象 否则返回None
search(pattern, string, flags=0)	搜索字符串中第一次出现正则表达式的模式 成功返回匹配对象 否则返回None
split(pattern, string, maxsplit=0, flags=0)	用正则表达式指定的模式分隔符拆分字符串 返回列表
sub(pattern, repl, string, count=0, flags=0)	用指定的字符串替换原字符串中与正则表达式匹配的模式 可以用count指定替换的次数
fullmatch(pattern, string, flags=0)	match函数的完全匹配 (从字符串开头到结尾) 版本
findall(pattern, string, flags=0)	查找字符串所有与正则表达式匹配的模式 返回字符串的列表
finditer(pattern, string, flags=0)	查找字符串所有与正则表达式匹配的模式 返回一个迭代器
purge()	清除隐式编译的正则表达式的缓存
re.I / re.IGNORECASE	忽略大小写匹配标记
re.M / re.MULTILINE	多行匹配标记

说明: 上面提到的re模块中的这些函数, 实际开发中也可以用正则表达式对象的方法替代对这些函数的使用, 如果一个正则表达式需要重复的使用, 那么先通过compile函数编译正则表达式并创建出正则表达式对象无疑是更为明智的选择。

下面我们通过一系列的例子来告诉大家在Python中如何使用正则表达式。

例子1：验证输入用户名和QQ号是否有效并给出对应的提示信息。

....

验证输入用户名和QQ号是否有效并给出对应的提示信息

要求：用户名必须由字母、数字或下划线构成且长度在6~20个字符之间, QQ号是5~12的数字且首位不

```

能为0
"""

import re

def main():
    username = input('请输入用户名: ')
    qq = input('请输入QQ号: ')
    # match函数的第一个参数是正则表达式字符串或正则表达式对象
    # 第二个参数是要跟正则表达式做匹配的字符串对象
    m1 = re.match(r'^[0-9a-zA-Z]{6,20}$', username)
    if not m1:
        print('请输入有效的用户名.')
    m2 = re.match(r'^[1-9]\d{4,11}$', qq)
    if not m2:
        print('请输入有效的QQ号.')
    if m1 and m2:
        print('你输入的信息是有效的!')

if __name__ == '__main__':
    main()

```

提示：上面在书写正则表达式时使用了“原始字符串”的写法（在字符串前面加上了r），所谓“原始字符串”就是字符串中的每个字符都是它原始的意义，说得更直接一点就是字符串中没有所谓的转义字符啦。因为正则表达式中有很多元字符和需要进行转义的地方，如果不使用原始字符串就需要将反斜杠写作\\，例如表示数字的\d得书写成\\d，这样不仅写起来不方便，阅读的时候也会很吃力。

例子2：从一段文字中提取出国内手机号码。

下面这张图是截止到2017年底，国内三家运营商推出的手机号段。

国内三家运营商的号段分别如下：

电信号段:133/153/180/181/189/177；
 联通号段:130/131/132/155/156/185/186/145/176；
 移动号段：
 134/135/136/137/138/139/150/151/152/157/158/159/182/183/184/187/188/147/178

```

import re

def main():
    # 创建正则表达式对象 使用了前瞻和回顾来保证手机号前后不应该出现数字
    pattern = re.compile(r'(?<=\D)1[34578]\d{9}(?=\D)')
    sentence = '''
    重要的事情说8130123456789遍，我的手机号是13512346789这个靓号，
    不是15600998765，也是110或119，王大锤的手机号才是15600998765。
    '''

```

```

# 查找所有匹配并保存到一个列表中
mylist = re.findall(pattern, sentence)
print(mylist)
print('-----华丽的分隔线-----')
# 通过迭代器取出匹配对象并获得匹配的内容
for temp in pattern.finditer(sentence):
    print(temp.group())
print('-----华丽的分隔线-----')
# 通过search函数指定搜索位置找出所有匹配
m = pattern.search(sentence)
while m:
    print(m.group())
    m = pattern.search(sentence, m.end())

if __name__ == '__main__':
    main()

```

说明：上面匹配国内手机号的正则表达式并不够好，因为像14开头的号码只有145或147，而上面的正则表达式并没有考虑这种情况，要匹配国内手机号，更好的正则表达式的写法是：($?<=\text{D}$)
 $(1[38]\text{d}\{9\}|14[57]\text{d}\{8\}|15[0-35-9]\text{d}\{8\}|17[678]\text{d}\{8\})(?=\text{D})$ ，国内最近好像有19和16开头的手机号了，但是这个暂时不在我们考虑之列。

例子3：替换字符串中的不良内容

```

import re

def main():
    sentence = '你丫是傻叉吗？我操你大爷的. Fuck you.'
    purified = re.sub('[操肏艹]|fuck|shit|傻[比肏逼叉缺吊屌]|煞笔',
                      '*', sentence, flags=re.IGNORECASE)
    print(purified)  # 你丫是*吗？我*你大爷的. * you.

if __name__ == '__main__':
    main()

```

说明：re模块的正则表达式相关函数中都有一个flags参数，它代表了正则表达式的匹配标记，可以通过该标记来指定匹配时是否忽略大小写、是否进行多行匹配、是否显示调试信息等。如果需要为flags参数指定多个值，可以使用按位或运算符进行叠加，如flags=re.I | re.M。

例子4：拆分长字符串

```

import re

def main():

```

```
poem = '窗前明月光，疑是地上霜。举头望明月，低头思故乡。'
sentence_list = re.split(r'[，。，。。]', poem)
while '' in sentence_list:
    sentence_list.remove('')
print(sentence_list) # ['窗前明月光', '疑是地上霜', '举头望明月', '低头思故乡']

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

后话

如果要从事爬虫类应用的开发，那么正则表达式一定是一个非常好的助手，因为它可以帮助我们迅速的从网页代码中发现某种我们指定的模式并提取出我们需要的信息，当然对于初学者来说，要编写一个正确的适当的正则表达式可能并不是一件容易的事情（当然有些常用的正则表达式可以直接在网上找找），所以实际开发爬虫应用的时候，有很多人会选择[Beautiful Soup](#)或[Lxml](#)来进行匹配和信息的提取，前者简单方便但是性能较差，后者既好用性能也好，但是安装稍嫌麻烦，这些内容我们会在后期的爬虫专题中为大家介绍。

进程和线程

今天我们使用的计算机早已进入多CPU或多核时代，而我们使用的操作系统都是支持“多任务”的操作系统，这使得我们可以同时运行多个程序，也可以将一个程序分解为若干个相对独立的子任务，让多个子任务并发的执行，从而缩短程序的执行时间，同时也让用户获得更好的体验。因此在当下不管是用什么编程语言进行开发，实现让程序同时执行多个任务也就是常说的“并发编程”，应该是程序员必备技能之一。为此，我们需要先讨论两个概念，一个叫进程，一个叫线程。

概念

进程就是操作系统中执行的一个程序，操作系统以进程为单位分配存储空间，每个进程都有自己的地址空间、数据栈以及其他用于跟踪进程执行的辅助数据，操作系统管理所有进程的执行，为它们合理的分配资源。进程可以通过fork或spawn的方式来创建新的进程来执行其他的任务，不过新的进程也有自己独立的内存空间，因此必须通过进程间通信机制（IPC，Inter-Process Communication）来实现数据共享，具体的方式包括管道、信号、套接字、共享内存区等。

一个进程还可以拥有多个并发的执行线索，简单的说就是拥有多个可以获得CPU调度的执行单元，这就是所谓的线程。由于线程在同一个进程中，它们可以共享相同的上下文，因此相对于进程而言，线程间的信息共享和通信更加容易。当然在单核CPU系统中，真正的并发是不可能的，因为在某个时刻能够获得CPU的只有唯一的一个线程，多个线程共享了CPU的执行时间。使用多线程实现并发编程为程序带来的好处是不言而喻的，最主要的体现在提升程序的性能和改善用户体验，今天我们使用的软件几乎都用到了多线程技术，这一点可以利用系统自带的进程监控工具（如macOS中的“活动监视器”、Windows中的“任务管理器”）来证实，如下图所示。



当然多线程也并不是没有坏处，站在其他进程的角度，多线程的程序对其他程序并不友好，因为它占用了更多的CPU执行时间，导致其他程序无法获得足够的CPU执行时间；另一方面，站在开发者的角度，编写和调试多线程的程序都对开发者有较高的要求，对于初学者来说更加困难。

Python既支持多进程又支持多线程，因此使用Python实现并发编程主要有3种方式：多进程、多线程、多进程+多线程。

Python中的多进程

Unix和Linux操作系统上提供了`fork()`系统调用来创建进程，调用`fork()`函数的是父进程，创建出的是子进程，子进程是父进程的一个拷贝，但是子进程拥有自己的PID。`fork()`函数非常特殊它会返回两次，父进程中可以通过`fork()`函数的返回值得到子进程的PID，而子进程中的返回值永远都是0。Python的os模块提供了`fork()`函数。由于Windows系统没有`fork()`调用，因此要实现跨平台的多进程编程，可以使用multiprocessing模块的`Process`类来创建子进程，而且该模块还提供了更高级的封装，例如批量启动进程的进程池（`Pool`）、用于进程间通信的队列（`Queue`）和管道（`Pipe`）等。

下面用一个下载文件的例子来说明使用多进程和不使用多进程到底有什么差别，先看看下面的代码。

```
from random import randint
from time import time, sleep

def download_task(filename):
    print('开始下载%s...' % filename)
    time_to_download = randint(5, 10)
    sleep(time_to_download)
    print('%s下载完成！耗费了%d秒' % (filename, time_to_download))

def main():
    start = time()
    download_task('Python从入门到住院.pdf')
    download_task('Peking Hot.avi')
    end = time()
    print('总共耗费了%.2f秒.' % (end - start))

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

下面是运行程序得到的一次运行结果。

```
开始下载Python从入门到住院.pdf...
Python从入门到住院.pdf下载完成！耗费了6秒
开始下载Peking Hot.avi...
Peking Hot.avi下载完成！耗费了7秒
总共耗费了13.01秒.
```

从上面的例子可以看出，如果程序中的代码只能按顺序一点点的往下执行，那么即使执行两个毫不相关的下载任务，也需要先等待一个文件下载完成后才能开始下一个下载任务，很显然这并不合理也没有效率。接下来我们使用多进程的方式将两个下载任务放到不同的进程中，代码如下所示。

```
from multiprocessing import Process
from os import getpid
from random import randint
from time import time, sleep

def download_task(filename):
    print('启动下载进程，进程号[%d].' % getpid())
    print('开始下载%s...' % filename)
    time_to_download = randint(5, 10)
    sleep(time_to_download)
    print('%s下载完成！耗费了%d秒' % (filename, time_to_download))

def main():
    start = time()
    p1 = Process(target=download_task, args=( 'Python从入门到住院.pdf', ))
    p1.start()
    p2 = Process(target=download_task, args=( 'Peking Hot.avi', ))
    p2.start()
    p1.join()
    p2.join()
    end = time()
    print('总共耗费了%.2f秒.' % (end - start))

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

在上面的代码中，我们通过`Process`类创建了进程对象，通过`target`参数我们传入一个函数来表示进程启动后要执行的代码，后面的`args`是一个元组，它代表了传递给函数的参数。`Process`对象的`start`方法用来启动进程，而`join`方法表示等待进程执行结束。运行上面的代码可以明显发现两个下载任务“同时”启动了，而且程序的执行时间将大大缩短，不再是两个任务的时间总和。下面是程序的一次执行结果。

```
启动下载进程，进程号[1530].
开始下载Python从入门到住院.pdf...
启动下载进程，进程号[1531].
开始下载Peking Hot.avi...
Peking Hot.avi下载完成！耗费了7秒
Python从入门到住院.pdf下载完成！耗费了10秒
总共耗费了10.01秒.
```

我们也可以使用`subprocess`模块中的类和函数来创建和启动子进程，然后通过管道来和子进程通信，这些内容我们不在此进行讲解，有兴趣的读者可以自己了解这些知识。接下来我们将重点放在如何实现两个进程间的通

信。我们启动两个进程，一个输出Ping，一个输出Pong，两个进程输出的Ping和Pong加起来一共10个。听起来很简单吧，但是如果这样写可是错的哦。

```
from multiprocessing import Process
from time import sleep

counter = 0

def sub_task(string):
    global counter
    while counter < 10:
        print(string, end='', flush=True)
        counter += 1
        sleep(0.01)

def main():
    Process(target=sub_task, args=('Ping', )).start()
    Process(target=sub_task, args=('Pong', )).start()

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

看起来没毛病，但是最后的结果是Ping和Pong各输出了10个，Why？当我们在程序中创建进程的时候，子进程复制了父进程及其所有的数据结构，每个子进程都有自己独立的内存空间，这也就意味着两个子进程中各有一个counter变量，所以结果也就可想而知了。要解决这个问题比较简单的办法是使用multiprocessing模块中的Queue类，它是可以被多个进程共享的队列，底层是通过管道和信号量（semaphore）机制来实现的，有兴趣的读者可以自己尝试一下。

Python中的多线程

在Python早期的版本中就引入了thread模块（现在名为_thread）来实现多线程编程，然而该模块过于底层，而且很多功能都没有提供，因此目前的多线程开发我们推荐使用threading模块，该模块对多线程编程提供了更好的面向对象的封装。我们把刚才下载文件的例子用多线程的方式来实现一遍。

```
from random import randint
from threading import Thread
from time import time, sleep

def download(filename):
    print('开始下载%s...' % filename)
    time_to_download = randint(5, 10)
    sleep(time_to_download)
    print('%s下载完成！耗费了%d秒' % (filename, time_to_download))
```

```
def main():
    start = time()
    t1 = Thread(target=download, args=( 'Python从入门到住院.pdf',))
    t1.start()
    t2 = Thread(target=download, args=( 'Peking Hot.avi',))
    t2.start()
    t1.join()
    t2.join()
    end = time()
    print('总共耗费了%.3f秒' % (end - start))

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

我们可以直接使用threading模块的Thread类来创建线程，但是我们之前讲过一个非常重要的概念叫“继承”，我们可以从已有的类创建新类，因此也可以通过继承Thread类的方式来创建自定义的线程类，然后再创建线程对象并启动线程。代码如下所示。

```
from random import randint
from threading import Thread
from time import time, sleep

class DownloadTask(Thread):

    def __init__(self, filename):
        super().__init__()
        self._filename = filename

    def run(self):
        print('开始下载%s...' % self._filename)
        time_to_download = randint(5, 10)
        sleep(time_to_download)
        print('%s下载完成！耗费了%d秒' % (self._filename, time_to_download))

def main():
    start = time()
    t1 = DownloadTask('Python从入门到住院.pdf')
    t1.start()
    t2 = DownloadTask('Peking Hot.avi')
    t2.start()
    t1.join()
    t2.join()
    end = time()
    print('总共耗费了%.2f秒.' % (end - start))
```

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

因为多个线程可以共享进程的内存空间，因此要实现多个线程间的通信相对简单，大家能想到的最直接的办法就是设置一个全局变量，多个线程共享这个全局变量即可。但是当多个线程共享同一个变量（我们通常称之为“资源”）的时候，很有可能产生不可控的结果从而导致程序失效甚至崩溃。如果一个资源被多个线程竞争使用，那么我们通常称之为“临界资源”，对“临界资源”的访问需要加上保护，否则资源会处于“混乱”的状态。下面的例子演示了100个线程向同一个银行账户转账（转入1元钱）的场景，在这个例子中，银行账户就是一个临界资源，在没有保护的情况下我们很有可能会得到错误的结果。

```
from time import sleep
from threading import Thread

class Account(object):

    def __init__(self):
        self._balance = 0

    def deposit(self, money):
        # 计算存款后的余额
        new_balance = self._balance + money
        # 模拟受理存款业务需要0.01秒的时间
        sleep(0.01)
        # 修改账户余额
        self._balance = new_balance

    @property
    def balance(self):
        return self._balance

class AddMoneyThread(Thread):

    def __init__(self, account, money):
        super().__init__()
        self._account = account
        self._money = money

    def run(self):
        self._account.deposit(self._money)

def main():
    account = Account()
    threads = []
    # 创建100个存款的线程向同一个账户中存钱
    for _ in range(100):
        t = AddMoneyThread(account, 1)
        threads.append(t)
        t.start()
```

```
# 等所有存款的线程都执行完毕
for t in threads:
    t.join()
print('账户余额为: ￥%d元' % account.balance)

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

运行上面的程序，结果让人大跌眼镜，100个线程分别向账户中转入1元钱，结果居然远远小于100元。之所以出现这种情况是因为我们没有对银行账户这个“临界资源”加以保护，多个线程同时向账户中存钱时，会一起执行到`new_balance = self._balance + money`这行代码，多个线程得到的账户余额都是初始状态下的0，所以都是0上面做了+1的操作，因此得到了错误的结果。在这种情况下，“锁”就可以派上用场了。我们可以通过“锁”来保护“临界资源”，只有获得“锁”的线程才能访问“临界资源”，而其他没有得到“锁”的线程只能被阻塞起来，直到获得“锁”的线程释放了“锁”，其他线程才有机会获得“锁”，进而访问被保护的“临界资源”。下面的代码演示了如何使用“锁”来保护对银行账户的操作，从而获得正确的结果。

```
from time import sleep
from threading import Thread, Lock

class Account(object):

    def __init__(self):
        self._balance = 0
        self._lock = Lock()

    def deposit(self, money):
        # 先获取锁才能执行后续的代码
        self._lock.acquire()
        try:
            new_balance = self._balance + money
            sleep(0.01)
            self._balance = new_balance
        finally:
            # 在finally中执行释放锁的操作保证正常异常锁都能释放
            self._lock.release()

    @property
    def balance(self):
        return self._balance

class AddMoneyThread(Thread):

    def __init__(self, account, money):
        super().__init__()
        self._account = account
        self._money = money

    def run(self):
```

```
self._account.deposit(self._money)

def main():
    account = Account()
    threads = []
    for _ in range(100):
        t = AddMoneyThread(account, 1)
        threads.append(t)
        t.start()
    for t in threads:
        t.join()
    print('账户余额为: ￥%d元' % account.balance)

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

比较遗憾的一件事情是Python的多线程并不能发挥CPU的多核特性，这一点只要启动几个执行死循环的线程就可以得到证实了。之所以如此，是因为Python的解释器有一个“全局解释器锁”（GIL）的东西，任何线程执行前必须先获得GIL锁，然后每执行100条字节码，解释器就自动释放GIL锁，让别的线程有机会执行，这是一个历史遗留问题，但是即便如此，就如我们之前举的例子，使用多线程在提升执行效率和改善用户体验方面仍然是有积极意义的。

多进程还是多线程

无论是多进程还是多线程，只要数量一多，效率肯定上不去，为什么呢？我们打个比方，假设你不幸正在准备中考，每天晚上需要做语文、数学、英语、物理、化学这5科的作业，每项作业耗时1小时。如果你先花1小时做语文作业，做完了，再花1小时做数学作业，这样，依次全部做完，一共花5小时，这种方式称为单任务模型。如果你打算切换到多任务模型，可以先做1分钟语文，再切换到数学作业，做1分钟，再切换到英语，以此类推，只要切换速度足够快，这种方式就和单核CPU执行多任务是一样的了，以旁观者的角度来看，你就正在同时写5科作业。

但是，切换作业是有代价的，比如从语文切到数学，要先收拾桌子上的语文书本、钢笔（这叫保存现场），然后，打开数学课本、找出圆规直尺（这叫准备新环境），才能开始做数学作业。操作系统在切换进程或者线程时也是一样的，它需要先保存当前执行的现场环境（CPU寄存器状态、内存页等），然后，把新任务的执行环境准备好（恢复上次的寄存器状态，切换内存页等），才能开始执行。这个切换过程虽然很快，但是也需要耗费时间。如果有几千个任务同时进行，操作系统可能就主要忙着切换任务，根本没有多少时间去执行任务了，这种情况最常见的就是硬盘狂响，点窗口无反应，系统处于假死状态。所以，多任务一旦多到一个限度，反而会使得系统性能急剧下降，最终导致所有任务都做不好。

是否采用多任务的第二个考虑是任务的类型，可以把任务分为计算密集型和I/O密集型。计算密集型任务的特点是要进行大量的计算，消耗CPU资源，比如对视频进行编码解码或者格式转换等等，这种任务全靠CPU的运算能力，虽然也可以用多任务完成，但是任务越多，花在任务切换的时间就越多，CPU执行任务的效率就越低。计算密集型任务由于主要消耗CPU资源，这类任务用Python这样的脚本语言去执行效率通常很低，最能胜任这类任务的是C语言，我们之前提到过Python中有嵌入C/C++代码的机制。

除了计算密集型任务，其他的涉及到网络、存储介质I/O的任务都可以视为I/O密集型任务，这类任务的特点是CPU消耗很少，任务的大部分时间都在等待I/O操作完成（因为I/O的速度远远低于CPU和内存的速度）。对于

I/O密集型任务，如果启动多任务，就可以减少I/O等待时间从而让CPU高效率的运转。有一大类的任务都属于I/O密集型任务，这其中包括了我们很快会涉及到的网络应用和Web应用。

说明：上面的内容和例子来自于[廖雪峰官方网站的《Python教程》](#)，因为对作者文中的某些观点持有不同的看法，对原文的文字描述做了适当的调整。

单线程+异步I/O

现代操作系统对I/O操作的改进中最为重要的就是支持异步I/O。如果充分利用操作系统提供的异步I/O支持，就可以用单进程单线程模型来执行多任务，这种全新的模型称为事件驱动模型。Nginx就是支持异步I/O的Web服务器，它在单核CPU上采用单进程模型就可以高效地支持多任务。在多核CPU上，可以运行多个进程（数量与CPU核心数相同），充分利用多核CPU。用Node.js开发的服务器端程序也使用了这种工作模式，这也是当下并发编程的一种流行方案。

在Python语言中，单线程+异步I/O的编程模型称为协程，有了协程的支持，就可以基于事件驱动编写高效的多任务程序。协程最大的优势就是极高的执行效率，因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销。协程的第二个优势就是不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不用加锁，只需要判断状态就好了，所以执行效率比多线程高很多。如果想要充分利用CPU的多核特性，最简单的方法是多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。关于这方面的内容，在后续的课程中会进行讲解。

应用案例

例子1：将耗时的任务放到线程中以获得更好的用户体验。

如下所示的界面中，有“下载”和“关于”两个按钮，用休眠的方式模拟点击“下载”按钮会联网下载文件需要耗费10秒的时间，如果不使用“多线程”，我们会发现，当点击“下载”按钮后整个程序的其他部分都被这个耗时的任务阻塞而无法执行了，这显然是非常糟糕的用户体验，代码如下所示。

```
import time
import tkinter
import tkinter.messagebox

def download():
    # 模拟下载任务需要花费10秒钟时间
    time.sleep(10)
    tkinter.messagebox.showinfo('提示', '下载完成!')

def show_about():
    tkinter.messagebox.showinfo('关于', '作者: 骆昊(v1.0)')

def main():
    top = tkinter.Tk()
    top.title('单线程')
    top.geometry('200x150')
    top.wm_attributes('-topmost', True)
```

```
panel = tkinter.Frame(top)
button1 = tkinter.Button(panel, text='下载', command=download)
button1.pack(side='left')
button2 = tkinter.Button(panel, text='关于', command=show_about)
button2.pack(side='right')
panel.pack(side='bottom')

tkinter.mainloop()

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

如果使用多线程将耗时间的任务放到一个独立的线程中执行，这样就不会因为执行耗时间的任务而阻塞了主线程，修改后的代码如下所示。

```
import time
import tkinter
import tkinter.messagebox
from threading import Thread

def main():

    class DownloadTaskHandler(Thread):

        def run(self):
            time.sleep(10)
            tkinter.messagebox.showinfo('提示', '下载完成!')
            # 启用下载按钮
            button1.config(state=tkinter.NORMAL)

        def download():
            # 禁用下载按钮
            button1.config(state=tkinter.DISABLED)
            # 通过daemon参数将线程设置为守护线程(主程序退出就不再保留执行)
            # 在线程中处理耗时间的下载任务
            DownloadTaskHandler(daemon=True).start()

    def show_about():
        tkinter.messagebox.showinfo('关于', '作者: 骆昊(v1.0)')

    top = tkinter.Tk()
    top.title('单线程')
    top.geometry('200x150')
    top.wm_attributes('-topmost', 1)

    panel = tkinter.Frame(top)
    button1 = tkinter.Button(panel, text='下载', command=download)
    button1.pack(side='left')
    button2 = tkinter.Button(panel, text='关于', command=show_about)
```

```
button2.pack(side='right')
panel.pack(side='bottom')

tkinter.mainloop()

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

例子2：使用多进程对复杂任务进行“分而治之”。

我们来完成1~100000000求和的计算密集型任务，这个问题本身非常简单，有点循环的知识就能解决，代码如下所示。

```
from time import time

def main():
    total = 0
    number_list = [x for x in range(1, 100000001)]
    start = time()
    for number in number_list:
        total += number
    print(total)
    end = time()
    print('Execution time: %.3fs' % (end - start))

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

在上面的代码中，我故意先去创建了一个列表容器然后填入了100000000个数，这一步其实是比较耗时间的，所以为了公平起见，当我们将这个任务分解到8个进程中去执行的时候，我们暂时也不考虑列表切片操作花费的时间，只是把做运算和合并运算结果的时间统计出来，代码如下所示。

```
from multiprocessing import Process, Queue
from random import randint
from time import time

def task_handler(curr_list, result_queue):
    total = 0
    for number in curr_list:
        total += number
    result_queue.put(total)

def main():
    processes = []
```

```
number_list = [x for x in range(1, 100000001)]
result_queue = Queue()
index = 0
# 启动8个进程将数据切片后进行运算
for _ in range(8):
    p = Process(target=task_handler,
                args=(number_list[index:index + 12500000], result_queue))
    index += 12500000
    processes.append(p)
    p.start()
# 开始记录所有进程执行完成花费的时间
start = time()
for p in processes:
    p.join()
# 合并执行结果
total = 0
while not result_queue.empty():
    total += result_queue.get()
print(total)
end = time()
print('Execution time: ', (end - start), 's', sep='')

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

比较两段代码的执行结果（在我目前使用的MacBook上，上面的代码需要大概6秒左右的时间，而下面的代码只需要不到1秒的时间，再强调一次我们只是比较了运算的时间，不考虑列表创建及切片操作花费的时间），使用多进程后由于获得了更多的CPU执行时间以及更好的利用了CPU的多核特性，明显的减少了程序的执行时间，而且计算量越大效果越明显。当然，如果愿意还可以将多个进程部署在不同的计算机上，做成分布式进程，具体的做法就是通过`multiprocessing.managers`模块中提供的管理器将`Queue`对象通过网络共享出来（注册到网络上让其他计算机可以访问），这部分内容也留到爬虫的专题再进行讲解。

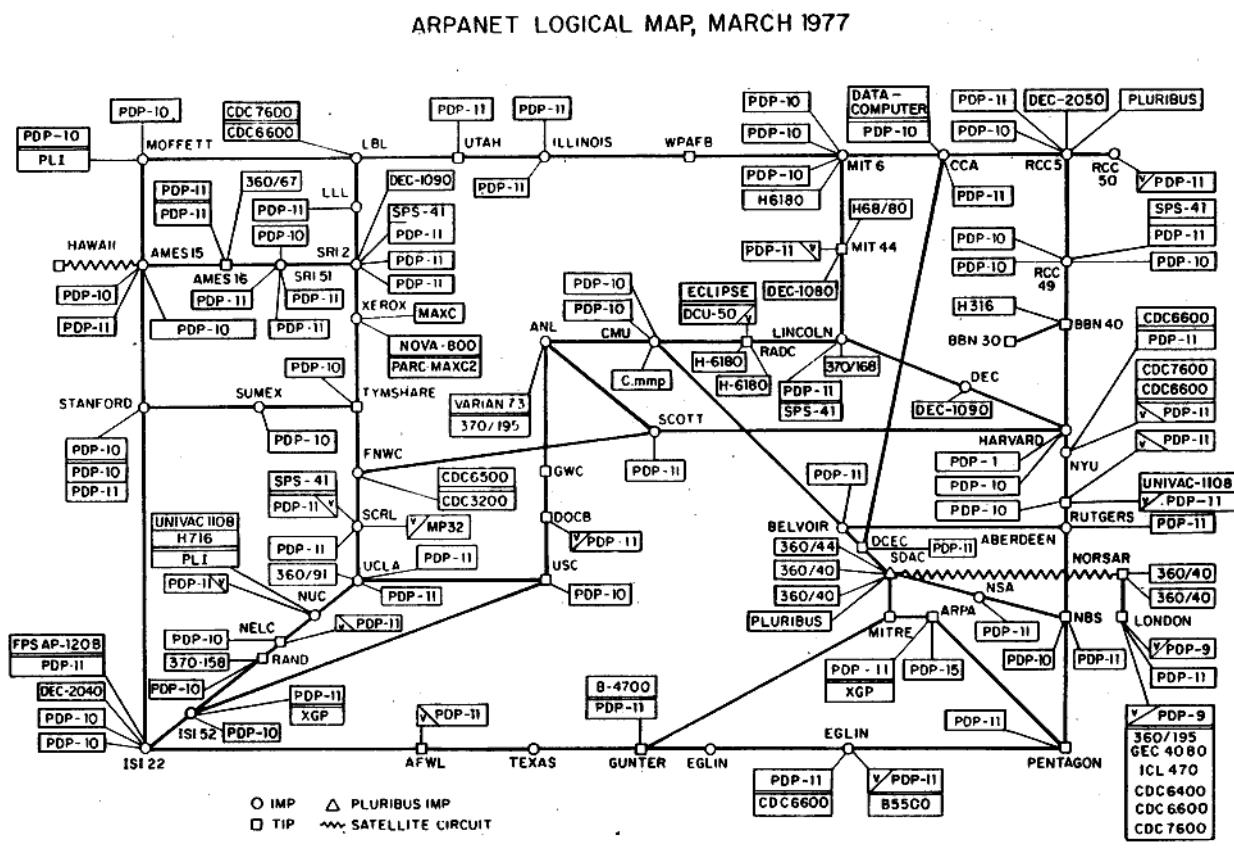
网络编程入门

计算机网络基础

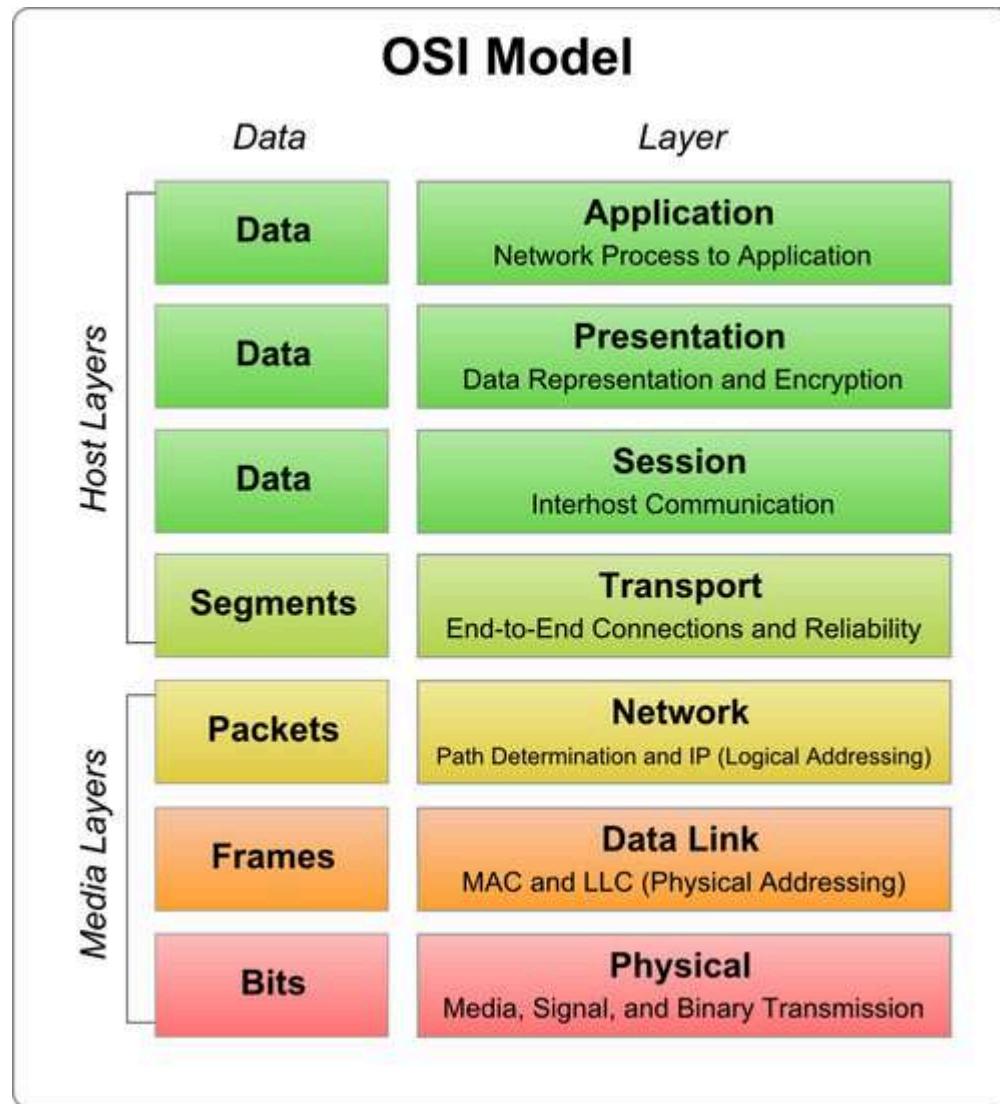
计算机网络是独立自主的计算机互联而成的系统的总称，组建计算机网络最主要的是实现多台计算机之间的通信和资源共享。今天计算机网络中的设备和计算机网络的用户已经多得不可计数，而计算机网络也可以称得上是一个“复杂巨系统”，对于这样的系统，我们不可能用一两篇文章把它讲清楚，有兴趣的读者可以自行阅读Andrew S.Tanenbaum老师的经典之作《计算机网络》或Kurose和Ross老师合著的《计算机网络:自顶向下方法》来了解计算机网络的相关知识。

计算机网络发展史

1. 1960s - 美国国防部ARPANET项目问世，奠定了分组交换网络的基础。



2. 1980s - 国际标准化组织 (ISO) 发布OSI/RM，奠定了网络技术标准化的基础。



3. 1990s - 英国人[蒂姆·伯纳斯-李](#)发明了图形化的浏览器，浏览器的简单易用性使得计算机网络迅速被普及。

在没有浏览器的年代，上网是这样的。



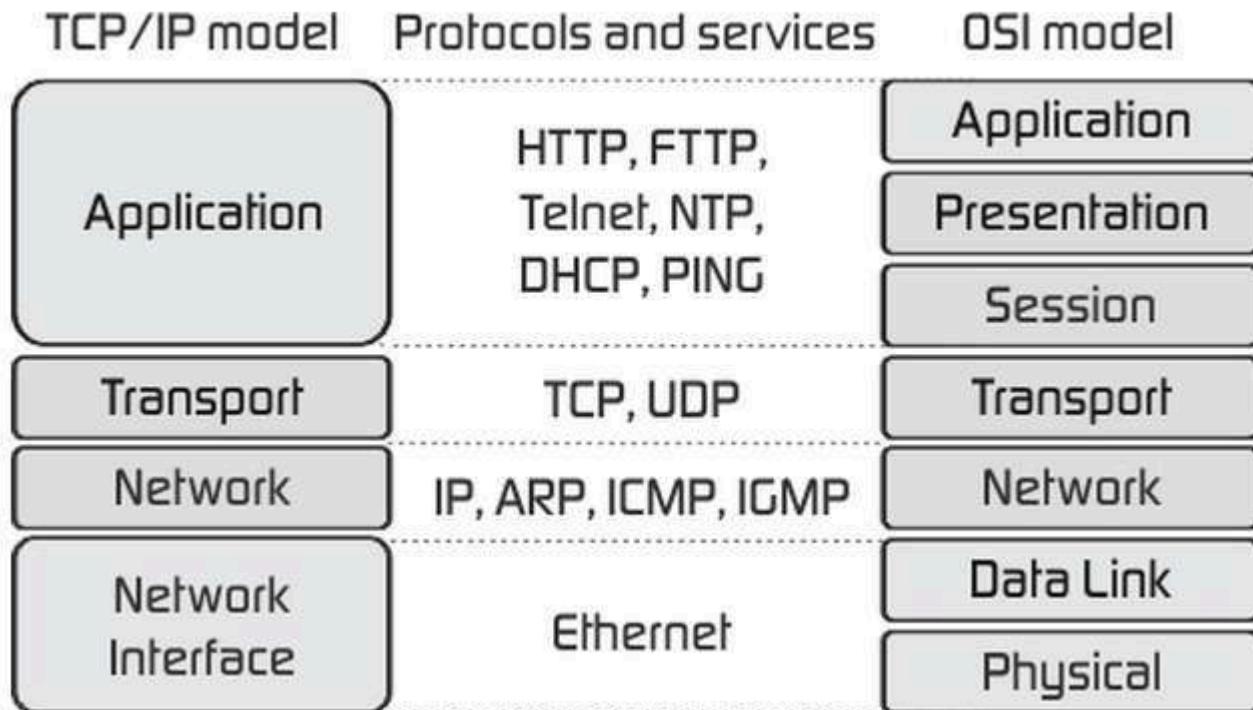
有了浏览器以后，上网是这样的。



TCP/IP模型

实现网络通信的基础是网络通信协议，这些协议通常是由**互联网工程任务组**（IETF）制定的。所谓“协议”就是通信计算机双方必须共同遵从的一组约定，例如怎样建立连接、怎样互相识别等，网络协议的三要素是：语法、语义和时序。构成我们今天使用的Internet的基础的是TCP/IP协议族，所谓协议族就是一系列的协议及其构

成的通信模型，我们通常也把这套东西称为TCP/IP模型。与国际标准化组织发布的OSI/RM这个七层模型不同，TCP/IP是一个四层模型，也就是说，该模型将我们使用的网络从逻辑上分解为四个层次，自底向上依次是：网络接口层、网络层、传输层和应用层，如下图所示。



IP通常被翻译为网际协议，它服务于网络层，主要实现了寻址和路由的功能。接入网络的每一台主机都需要有自己的IP地址，IP地址就是主机在计算机网络上的身份标识。当然由于IPv4地址的匮乏，我们平常在家里、办公室以及其他可以接入网络的公共区域上网时获得的IP地址并不是全球唯一的IP地址，而是一个[局域网（LAN）](#)中的内部IP地址，通过[网络地址转换（NAT）服务](#)我们也可以实现对网络的访问。计算机网络上有大量的被我们称为“[路由器](#)”的网络中继设备，它们会存储转发我们发送到网络上的数据分组，让从源头发出的数据最终能够找到传送到目的地通路，这项功能就是所谓的路由。

TCP全称传输控制协议，它是基于IP提供的寻址和路由服务而建立起来的负责实现端到端可靠传输的协议，之所以将TCP称为可靠的传输协议是因为TCP向调用者承诺了三件事情：

1. 数据不传丢不传错（利用握手、校验和重传机制可以实现）。
2. 流量控制（通过滑动窗口匹配数据发送者和接收者之间的传输速度）。
3. 拥塞控制（通过RTT时间以及对滑动窗口的控制缓解网络拥堵）。

网络应用模式

1. C/S模式和B/S模式。这里的C指的是Client（客户端），通常是一个需要安装到某个宿主操作系统上的应用程序；而B指的是Browser（浏览器），它几乎是所有图形化操作系统都默认安装了一个应用软件；通过C或B都可以实现对S（服务器）的访问。关于二者的比较和讨论在网络上有一大堆的文章，在此我们就不再浪费笔墨了。
2. 去中心化的网络应用模式。不管是B/S还是C/S都需要服务器的存在，服务器就是整个应用模式的中心，而去中心化的网络应用通常没有固定的服务器或者固定的客户端，所有应用的使用者既可以作为资源的提供者也可以作为资源的访问者。

基于HTTP协议的网络资源访问

HTTP (超文本传输协议)

HTTP是超文本传输协议 (Hyper-Text Transfer Protocol) 的简称, 维基百科上对HTTP的解释是: 超文本传输协议是一种用于分布式、协作式和超媒体信息系统的应用层协议, 它是[万维网](#)数据通信的基础, 设计HTTP最初的目的就是为了提供一种发布和接收[HTML](#)页面的方法, 通过HTTP或者[HTTPS](#) (超文本传输安全协议) 请求的资源由URI ([统一资源标识符](#)) 来标识。关于HTTP的更多内容, 我们推荐阅读阮一峰老师的[《HTTP 协议入门》](#), 简单的说, 通过HTTP我们可以获取网络上的 (基于字符的) 资源, 开发中经常会用到的网络API (有的地方也称之为网络数据接口) 就是基于HTTP来实现数据传输的。

JSON格式

JSON (JavaScript Object Notation) 是一种轻量级的数据交换语言, 该语言以易于让人阅读的文字 (纯文本) 为基础, 用来传输由属性值或者序列性的值组成的数据对象。尽管JSON是最初只是Javascript中一种创建对象的字面量语法, 但它在当下更是一种独立于语言的数据格式, 很多编程语言都支持JSON格式数据的生成和解析, Python内置的json模块也提供了这方面的功能。由于JSON是纯文本, 它和[XML](#)一样都适用于异构系统之间的数据交换, 而相较于XML, JSON显得更加的轻便和优雅。下面是表达同样信息的XML和JSON, 而JSON的优势是相当直观的。

XML的例子:

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <from>Alice</from>
  <to>Bob</to>
  <content>Will you marry me?</content>
</message>
```

JSON的例子:

```
{
  "from": "Alice",
  "to": "Bob",
  "content": "Will you marry me?"}
```

requests库

requests是一个基于HTTP协议来使用网络的第三库, 其[官方网站](#)有这样的一句介绍它的话: "Requests是唯一的一个[非转基因](#)的Python HTTP库, 人类可以安全享用。"简单的说, 使用requests库可以非常方便的使用HTTP, 避免安全缺陷、冗余代码以及"重复发明轮子" (行业黑话, 通常用在软件工程领域表示重新创造一个已有的或是早已被优化过的基本方法)。前面的文章中我们已经使用过这个库, 下面我们还是通过requests来实现一个访问网络数据接口并从中获取美女图片下载链接然后下载美女图片到本地的例子程序, 程序中使用了[天行数据](#)提供的网络API。

我们可以先通过pip安装requests及其依赖库。

```
pip install requests
```

如果使用PyCharm作为开发工具，可以直接在代码中书写`import requests`，然后通过代码修复功能来自动下载安装requests。

```
from time import time
from threading import Thread

import requests

# 继承Thread类创建自定义的线程类
class DownloadHanlder(Thread):

    def __init__(self, url):
        super().__init__()
        self.url = url

    def run(self):
        filename = self.url[self.url.rfind('/') + 1:]
        resp = requests.get(self.url)
        with open('/Users/Hao/' + filename, 'wb') as f:
            f.write(resp.content)

    def main():
        # 通过requests模块的get函数获取网络资源
        # 下面的代码中使用了天行数据接口提供的网络API
        # 要使用该数据接口需要在天行数据的网站上注册
        # 然后用自己的Key替换掉下面代码中的APIKey即可
        resp = requests.get(
            'http://api.tianapi.com/meinv/?key=APIKey&num=10')
        # 将服务器返回的JSON格式的数据解析为字典
        data_model = resp.json()
        for mm_dict in data_model['newslist']:
            url = mm_dict['picUrl']
            # 通过多线程的方式实现图片下载
            DownloadHanlder(url).start()

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

基于传输层协议的套接字编程

套接字这个词对很多不了解网络编程的人来说显得非常晦涩和陌生，其实说得通俗点，套接字就是一套用C语言写成的应用程序开发库，主要用于实现进程间通信和网络编程，在网络应用开发中被广泛使用。在Python中

也可以基于套接字来使用传输层提供的传输服务，并基于此开发自己的网络应用。实际开发中使用的套接字可以分为三类：流套接字（TCP套接字）、数据报套接字和原始套接字。

TCP套接字

所谓TCP套接字就是使用TCP协议提供的传输服务来实现网络通信的编程接口。在Python中可以通过创建socket对象并指定type属性为SOCK_STREAM来使用TCP套接字。由于一台主机可能拥有多个IP地址，而且很有可能会配置多个不同的服务，所以作为服务器端的程序，需要在创建套接字对象后将其绑定到指定的IP地址和端口上。这里的端口并不是物理设备而是对IP地址的扩展，用于区分不同的服务，例如我们通常将HTTP服务跟80端口绑定，而MySQL数据库服务默认绑定在3306端口，这样当服务器收到用户请求时就可以根据端口号来确定到底用户请求的是HTTP服务器还是数据库服务器提供的服务。端口的取值范围是0~65535，而1024以下的端口我们通常称之为“著名端口”（留给像FTP、HTTP、SMTP等“著名服务”使用的端口，有的地方也称之为“熟知端口”），自定义的服务通常不使用这些端口，除非自定义的是HTTP或FTP这样的著名服务。

下面的代码实现了一个提供时间日期的服务器。

```
from socket import socket, SOCK_STREAM, AF_INET
from datetime import datetime

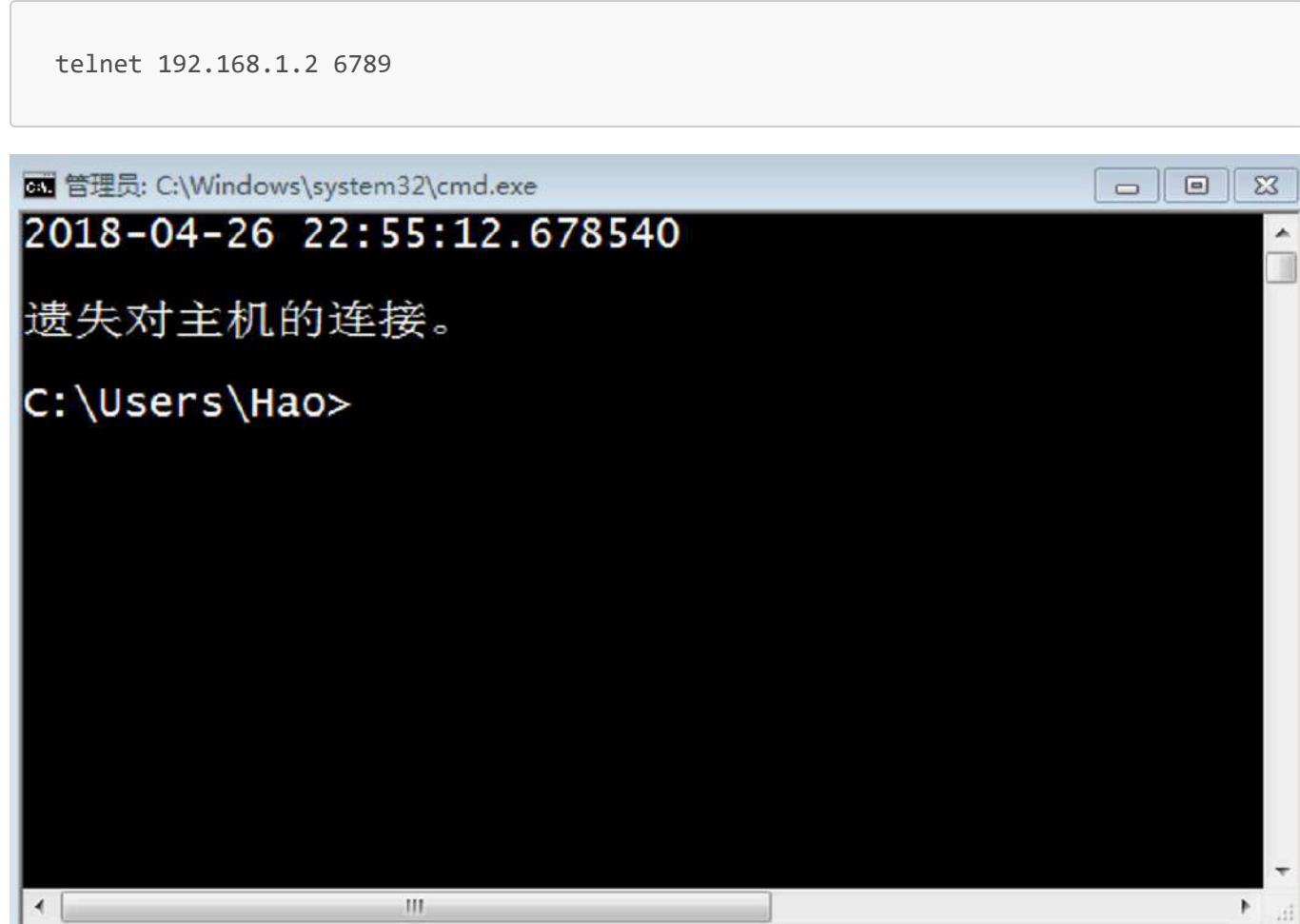
def main():
    # 1. 创建套接字对象并指定使用哪种传输服务
    # family=AF_INET - IPv4地址
    # family=AF_INET6 - IPv6地址
    # type=SOCK_STREAM - TCP套接字
    # type=SOCK_DGRAM - UDP套接字
    # type=SOCK_RAW - 原始套接字
    server = socket(family=AF_INET, type=SOCK_STREAM)
    # 2. 绑定IP地址和端口(端口用于区分不同的服务)
    # 同一时间在同一个端口上只能绑定一个服务否则报错
    server.bind(('192.168.1.2', 6789))
    # 3. 开启监听 - 监听客户端连接到服务器
    # 参数512可以理解为连接队列的大小
    server.listen(512)
    print('服务器启动开始监听...')

    while True:
        # 4. 通过循环接收客户端的连接并作出相应的处理(提供服务)
        # accept方法是一个阻塞方法如果没有客户端连接到服务器代码不会向下执行
        # accept方法返回一个元组其中的第一个元素是客户端对象
        # 第二个元素是连接到服务器的客户端的地址(由IP和端口两部分构成)
        client, addr = server.accept()
        print(str(addr) + '连接到了服务器。')

        # 5. 发送数据
        client.send(str(datetime.now()).encode('utf-8'))
        # 6. 断开连接
        client.close()

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

运行服务器程序后我们可以通过Windows系统的telnet来访问该服务器，结果如下图所示。



当然我们也可以通过Python的程序来实现TCP客户端的功能，相较于实现服务器程序，实现客户端程序就简单多了，代码如下所示。

```
from socket import socket

def main():
    # 1. 创建套接字对象默认使用IPv4和TCP协议
    client = socket()
    # 2. 连接到服务器(需要指定IP地址和端口)
    client.connect(('192.168.1.2', 6789))
    # 3. 从服务器接收数据
    print(client.recv(1024).decode('utf-8'))
    client.close()

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

需要注意的是，上面的服务器并没有使用多线程或者异步I/O的处理方式，这也就意味着当服务器与一个客户端处于通信状态时，其他的客户端只能排队等待。很显然，这样的服务器并不能满足我们的需求，我们需要的服

服务器是能够同时接纳和处理多个用户请求的。下面我们将设计一个使用多线程技术处理多个用户请求的服务器，该服务器会向连接到服务器的客户端发送一张图片。

服务器端代码：

```
from socket import socket, SOCK_STREAM, AF_INET
from base64 import b64encode
from json import dumps
from threading import Thread

def main():
    # 自定义线程类
    class FileTransferHandler(Thread):

        def __init__(self, cclient):
            super().__init__()
            self.cclient = cclient

        def run(self):
            my_dict = {}
            my_dict['filename'] = 'guido.jpg'
            # JSON是纯文本不能携带二进制数据
            # 所以图片的二进制数据要处理成base64编码
            my_dict['filedata'] = data
            # 通过dumps函数将字典处理成JSON字符串
            json_str = dumps(my_dict)
            # 发送JSON字符串
            self.cclient.send(json_str.encode('utf-8'))
            self.cclient.close()

    # 1. 创建套接字对象并指定使用哪种传输服务
    server = socket()
    # 2. 绑定IP地址和端口(区分不同的服务)
    server.bind(('192.168.1.2', 5566))
    # 3. 开启监听 - 监听客户端连接到服务器
    server.listen(512)
    print('服务器启动开始监听...')
    with open('guido.jpg', 'rb') as f:
        # 将二进制数据处理成base64再解码成字符串
        data = b64encode(f.read()).decode('utf-8')
    while True:
        client, addr = server.accept()
        # 启动一个线程来处理客户端的请求
        FileTransferHandler(client).start()

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

客户端代码：

```
from socket import socket
from json import loads
from base64 import b64decode

def main():
    client = socket()
    client.connect(('192.168.1.2', 5566))
    # 定义一个保存二进制数据的对象
    in_data = bytes()
    # 由于不知道服务器发送的数据有多大每次接收1024字节
    data = client.recv(1024)
    while data:
        # 将收到的数据拼接起来
        in_data += data
        data = client.recv(1024)
    # 将收到的二进制数据解码成JSON字符串并转换成字典
    # loads函数的作用就是将JSON字符串转成字典对象
    my_dict = loads(in_data.decode('utf-8'))
    filename = my_dict['filename']
    filedata = my_dict['filedata'].encode('utf-8')
    with open('/Users/Hao/' + filename, 'wb') as f:
        # 将base64格式的数据解码成二进制数据并写入文件
        f.write(b64decode(filedata))
    print('图片已保存。')

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

在这个案例中，我们使用了JSON作为数据传输的格式（通过JSON格式对传输的数据进行了序列化和反序列化的操作），但是JSON并不能携带二进制数据，因此对图片的二进制数据进行了Base64编码的处理。Base64是一种用64个字符表示所有二进制数据的编码方式，通过将二进制数据每6位一组的方式重新组织，刚好可以用0~9的数字、大小写字母以及“+”和“/”总共64个字符表示从000000到111111的64种状态。[维基百科](#)上有关于Base64编码的详细讲解，不熟悉Base64的读者可以自行阅读。

说明： 上面的代码主要为了讲解网络编程的相关内容因此并没有对异常状况进行处理，请读者自行添加异常处理代码来增强程序的健壮性。

UDP套接字

传输层除了有可靠的传输协议TCP之外，还有一种非常轻便的传输协议叫做用户数据报协议，简称UDP。TCP和UDP都是提供端到端传输服务的协议，二者的差别就如同打电话和发短信的区别，后者不对传输的可靠性和可达性做出任何承诺从而避免了TCP中握手和重传的开销，所以在强调性能和而不是数据完整性的场景中（例如传输网络音视频数据），UDP可能是更好的选择。可能大家会注意到一个现象，就是在观看网络视频时，有时会出现卡顿，有时会出现花屏，这无非就是部分数据传丢或传错造成的。在Python中也可以使用UDP套接字来创建网络应用，对此我们不进行赘述，有兴趣的读者可以自行研究。

网络应用开发

发送电子邮件

在即时通信软件如此发达的今天，电子邮件仍然是互联网上使用最为广泛的应用之一，公司向应聘者发出录用通知、网站向用户发送一个激活账号的链接、银行向客户推广它们的理财产品等几乎都是通过电子邮件来完成的，而这些任务应该都是由程序自动完成的。

就像我们可以用HTTP（超文本传输协议）来访问一个网站一样，发送邮件要使用SMTP（简单邮件传输协议），SMTP也是一个建立在TCP（传输控制协议）提供的可靠数据传输服务的基础上的应用级协议，它规定了邮件的发送者如何跟发送邮件的服务器进行通信的细节，而Python中的smtplib模块将这些操作简化成了几个简单的函数。

下面的代码演示了如何在Python发送邮件。

```
from smtplib import SMTP
from email.header import Header
from email.mime.text import MIMEText

def main():
    # 请自行修改下面的邮件发送者和接收者
    sender = 'abcdefg@126.com'
    receivers = ['uvwxyz@qq.com', 'uvwxyz@126.com']
    message = MIMEText('用Python发送邮件的示例代码。', 'plain', 'utf-8')
    message['From'] = Header('王大锤', 'utf-8')
    message['To'] = Header('骆昊', 'utf-8')
    message['Subject'] = Header('示例代码实验邮件', 'utf-8')
    smtpper = SMTP('smtp.126.com')
    # 请自行修改下面的登录口令
    smtpper.login(sender, 'secretpass')
    smtpper.sendmail(sender, receivers, message.as_string())
    print('邮件发送完成!')

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

如果要发送带有附件的邮件，那么可以按照下面的方式进行操作。

```
from smtplib import SMTP
from email.header import Header
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart

import urllib

def main():
```

```
# 创建一个带附件的邮件消息对象
message = MIMEMultipart()

# 创建文本内容
text_content = MIMEText('附件中有本月数据请查收', 'plain', 'utf-8')
message['Subject'] = Header('本月数据', 'utf-8')
# 将文本内容添加到邮件消息对象中
message.attach(text_content)

# 读取文件并将文件作为附件添加到邮件消息对象中
with open('/Users/Hao/Desktop/hello.txt', 'rb') as f:
    txt = MIMEText(f.read(), 'base64', 'utf-8')
    txt['Content-Type'] = 'text/plain'
    txt['Content-Disposition'] = 'attachment; filename=hello.txt'
    message.attach(txt)

# 读取文件并将文件作为附件添加到邮件消息对象中
with open('/Users/Hao/Desktop/汇总数据.xlsx', 'rb') as f:
    xls = MIMEText(f.read(), 'base64', 'utf-8')
    xls['Content-Type'] = 'application/vnd.ms-excel'
    xls['Content-Disposition'] = 'attachment; filename=month-data.xlsx'
    message.attach(xls)

# 创建SMTP对象
smtper = SMTP('smtp.126.com')
# 开启安全连接
# smtper.starttls()
sender = 'abcdefg@126.com'
receivers = ['uvwxyz@qq.com']
# 登录到SMTP服务器
# 请注意此处不是使用密码而是邮件客户端授权码进行登录
# 对此有疑问的读者可以联系自己使用的邮件服务器客服
smtper.login(sender, 'secretpass')
# 发送邮件
smtper.sendmail(sender, receivers, message.as_string())
# 与邮件服务器断开连接
smtper.quit()
print('发送完成!')

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

发送短信

发送短信也是项目中常见的功能，网站的注册码、验证码、营销信息基本上都是通过短信来发送给用户的。在下面的代码中我们使用了[互亿无线](#)短信平台（该平台为注册用户提供了50条免费短信以及常用开发语言发送短信的demo，可以登录该网站并在用户自服务页面中对短信进行配置）提供的API接口实现了发送短信的服务，当然国内的短信平台很多，读者可以根据自己的需要进行选择（通常会考虑费用预算、短信到达率、使用的难易程度等指标），如果需要在商业项目中使用短信服务建议购买短信平台提供的套餐服务。

```
import urllib.parse
import http.client
import json

def main():
    host = "106.ihuyi.com"
    sms_send_uri = "/webservice/sms.php?method=Submit"
    # 下面的参数需要填入自己注册的账号和对应的密码
    params = urllib.parse.urlencode({'account': '你自己的账号', 'password' : '你自己的密码', 'content': '您的验证码是: 147258。请不要把验证码泄露给其他人。', 'mobile': '接收者的手机号', 'format':'json' })
    print(params)
    headers = {'Content-type': 'application/x-www-form-urlencoded', 'Accept': 'text/plain'}
    conn = http.client.HTTPConnection(host, port=80, timeout=30)
    conn.request('POST', sms_send_uri, params, headers)
    response = conn.getresponse()
    response_str = response.read()
    jsonstr = response_str.decode('utf-8')
    print(json.loads(jsonstr))
    conn.close()

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

图像和办公文档处理

用程序来处理图像和办公文档经常出现在实际开发中，Python的标准库中虽然没有直接支持这些操作的模块，但我们可以借助Python生态圈中的第三方模块来完成这些操作。

操作图像

计算机图像相关知识

1. 颜色。如果你有使用颜料画画的经历，那么一定知道混合红、黄、蓝三种颜料可以得到其他的颜色，事实上这三种颜色就是被我们称为美术三原色的东西，它们是不能再分解的基本颜色。在计算机中，我们可以将红、绿、蓝三种色光以不同的比例叠加来组合成其他的颜色，因此这三种颜色就是色光三原色，所以我们通常会将一个颜色表示为一个RGB值或RGBA值（其中的A表示Alpha通道，它决定了透过这个图像的像素，也就是透明度）。

名称	RGBA值	名称	RGBA值
White	(255, 255, 255, 255)	Red	(255, 0, 0, 255)
Green	(0, 255, 0, 255)	Blue	(0, 0, 255, 255)
Gray	(128, 128, 128, 255)	Yellow	(255, 255, 0, 255)
Black	(0, 0, 0, 255)	Purple	(128, 0, 128, 255)

2. 像素。对于一个由数字序列表示的图像来说，最小的单位就是图像上单一颜色的小方格，这些小方块都有一个明确的位置和被分配的色彩数值，而这些一小方格的颜色和位置决定了该图像最终呈现出来的样子，它们是不可分割的单位，我们通常称之为像素（pixel）。每一个图像都包含了一定量的像素，这些像素决定图像在屏幕上所呈现的大小。

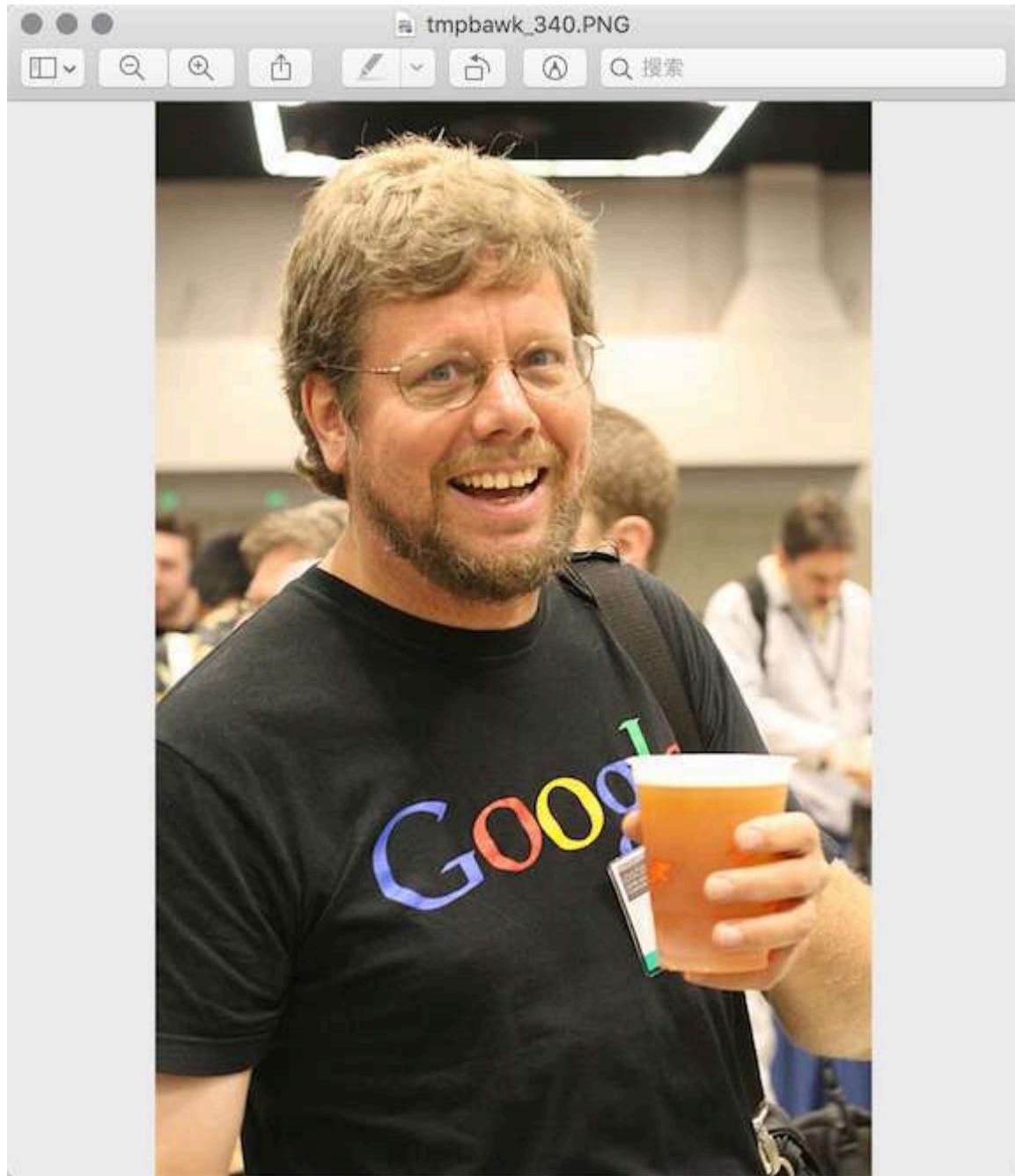
用Pillow操作图像

Pillow是由从著名的Python图像处理库PIL发展出来的一个分支，通过Pillow可以实现图像压缩和图像处理等各种操作。可以使用下面的命令来安装Pillow。

```
pip install pillow
```

Pillow中最为重要的是Image类，读取和处理图像都要通过这个类来完成。

```
>>> from PIL import Image
>>>
>>> image = Image.open('./res/guido.jpg')
>>> image.format, image.size, image.mode
('JPEG', (500, 750), 'RGB')
>>> image.show()
```



1. 剪裁图像

```
>>> image = Image.open('./res/guido.jpg')
>>> rect = 80, 20, 310, 360
>>> image.crop(rect).show()
```



2. 生成缩略图

```
>>> image = Image.open('./res/guido.jpg')
>>> size = 128, 128
>>> image.thumbnail(size)
>>> image.show()
```



3. 缩放和黏贴图像

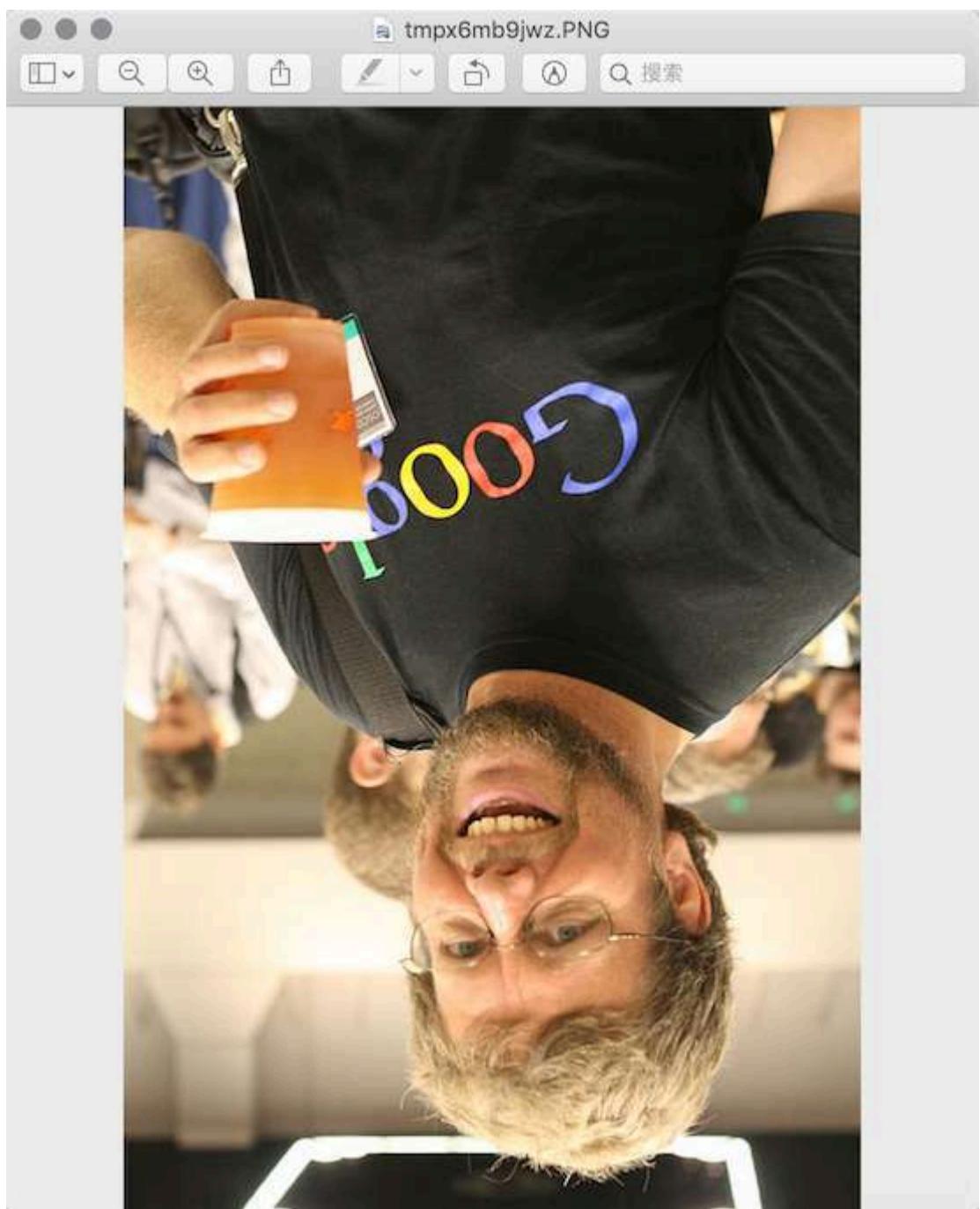
```
>>> image1 = Image.open('./res/luohao.png')
>>> image2 = Image.open('./res/guido.jpg')
>>> rect = 80, 20, 310, 360
>>> guido_head = image2.crop(rect)
>>> width, height = guido_head.size
```

```
>>> image1.paste(guido_head.resize((int(width / 1.5), int(height / 1.5))),  
(172, 40))
```



4. 旋转和翻转

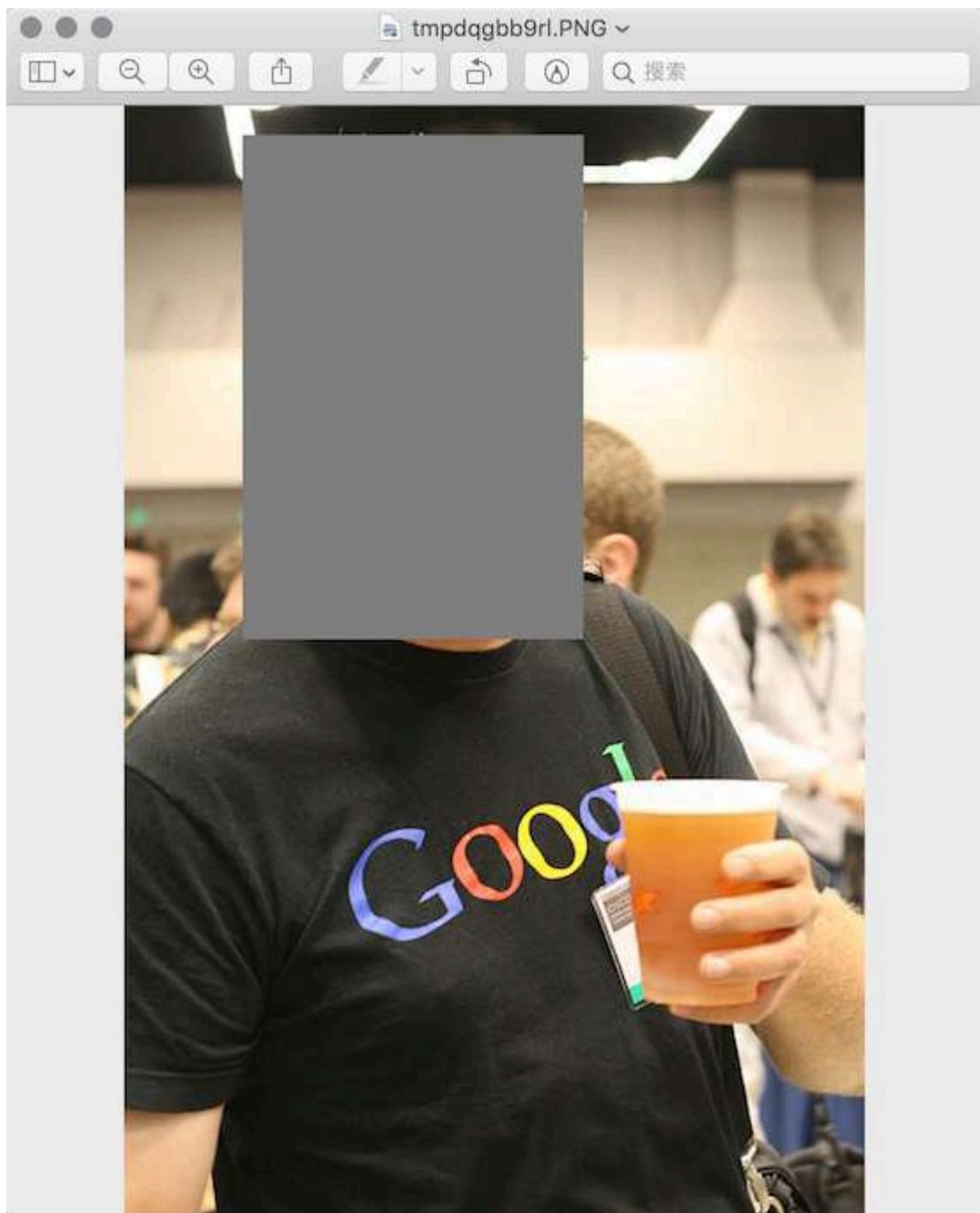
```
>>> image = Image.open('./res/guido.png')  
>>> image.rotate(180).show()  
>>> image.transpose(Image.FLIP_LEFT_RIGHT).show()
```





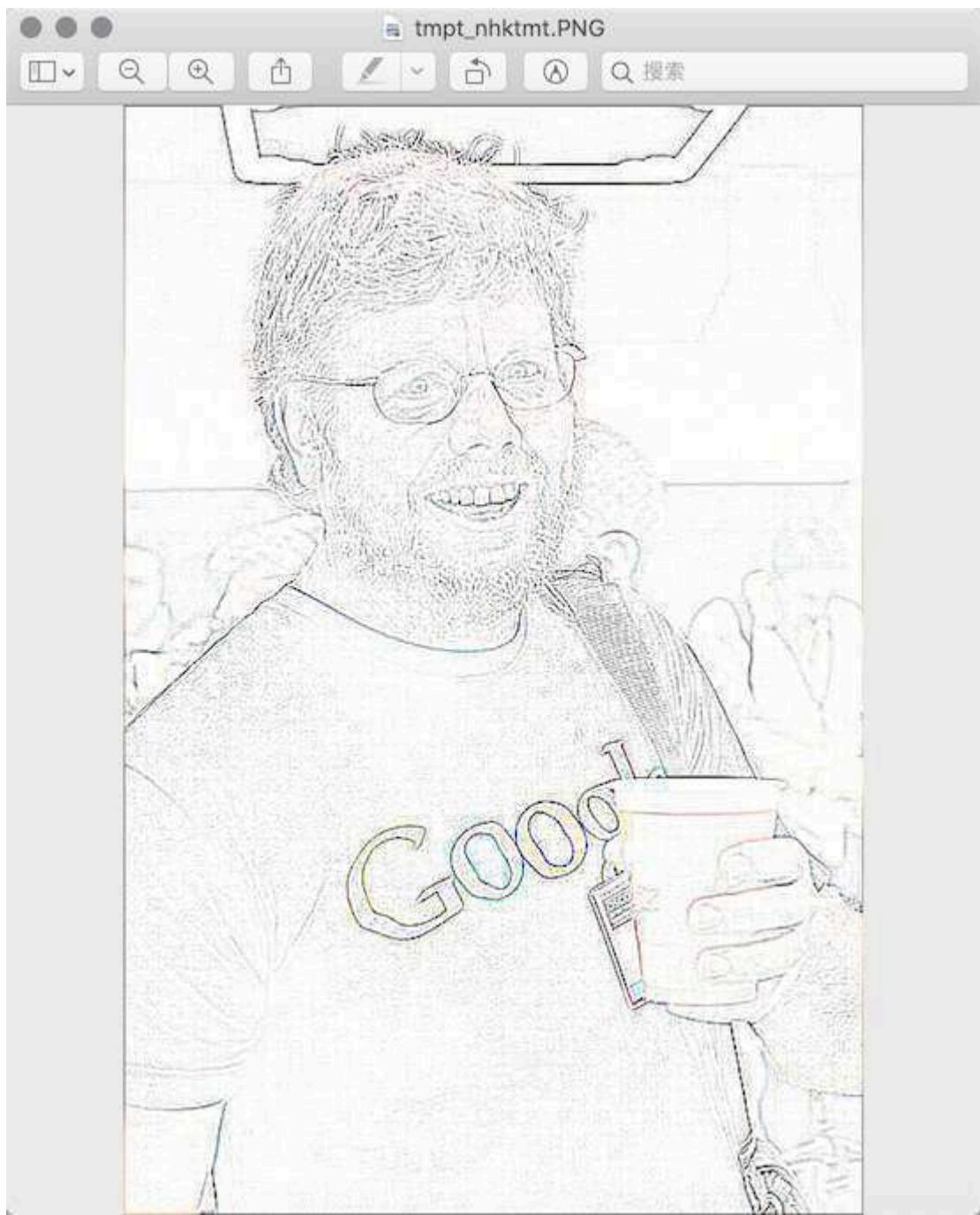
5. 操作像素

```
>>> image = Image.open('./res/guido.jpg')
>>> for x in range(80, 310):
...     for y in range(20, 360):
...         image.putpixel((x, y), (128, 128, 128))
...
>>> image.show()
```



6. 滤镜效果

```
>>> from PIL import Image, ImageFilter  
>>>  
>>> image = Image.open('./res/guido.jpg')  
>>> image.filter(ImageFilter.CONTOUR).show()
```



处理Excel电子表格

Python的openpyxl模块让我们可以在Python程序中读取和修改Excel电子表格，由于微软从Office 2007开始使用了新的文件格式，这使得Office Excel和LibreOffice Calc、OpenOffice Calc是完全兼容的，这就意味着openpyxl模块也能处理来自这些软件生成的电子表格。

```
import datetime

from openpyxl import Workbook

wb = Workbook()
ws = wb.active

ws['A1'] = 42
ws.append([1, 2, 3])
```

```
ws['A2'] = datetime.datetime.now()  
  
wb.save("sample.xlsx")
```

处理Word文档

利用python-docx模块，Python可以创建和修改Word文档，当然这里的Word文档不仅仅是指通过微软的Office软件创建的扩展名为docx的文档，LibreOffice Writer和OpenOffice Writer都是免费的字处理软件。

```
from docx import Document  
from docx.shared import Inches  
  
document = Document()  
  
document.add_heading('Document Title', 0)  
  
p = document.add_paragraph('A plain paragraph having some ')  
p.add_run('bold').bold = True  
p.add_run(' and some ')  
p.add_run('italic.').italic = True  
  
document.add_heading('Heading, level 1', level=1)  
document.add_paragraph('Intense quote', style='Intense Quote')  
  
document.add_paragraph(  
    'first item in unordered list', style='List Bullet'  
)  
document.add_paragraph(  
    'first item in ordered list', style='List Number'  
)  
  
document.add_picture('monty-truth.png', width=Inches(1.25))  
  
records = (  
    (3, '101', 'Spam'),  
    (7, '422', 'Eggs'),  
    (4, '631', 'Spam, spam, eggs, and spam')  
)  
  
table = document.add_table(rows=1, cols=3)  
hdr_cells = table.rows[0].cells  
hdr_cells[0].text = 'Qty'  
hdr_cells[1].text = 'Id'  
hdr_cells[2].text = 'Desc'  
for qty, id, desc in records:  
    row_cells = table.add_row().cells  
    row_cells[0].text = str(qty)  
    row_cells[1].text = id  
    row_cells[2].text = desc  
  
document.add_page_break()
```

```
document.save('demo.docx')
```

Python语言进阶

重要知识点

- 生成式 (推导式) 的用法

```

prices = {
    'AAPL': 191.88,
    'GOOG': 1186.96,
    'IBM': 149.24,
    'ORCL': 48.44,
    'ACN': 166.89,
    'FB': 208.09,
    'SYMC': 21.29
}
# 用股票价格大于100元的股票构造一个新的字典
prices2 = {key: value for key, value in prices.items() if value > 100}
print(prices2)

```

说明：生成式 (推导式) 可以用来生成列表、集合和字典。

- 嵌套的列表的坑

```

names = ['关羽', '张飞', '赵云', '马超', '黄忠']
courses = ['语文', '数学', '英语']
# 录入五个学生三门课程的成绩
# 错误 - 参考http://pythontutor.com/visualize.html#mode=edit
# scores = [[None] * len(courses)] * len(names)
scores = [[None] * len(courses) for _ in range(len(names))]
for row, name in enumerate(names):
    for col, course in enumerate(courses):
        scores[row][col] = float(input(f'请输入{name}的{course}成绩: '))
print(scores)

```

Python Tutor - VISUALIZE CODE AND GET LIVE HELP

- heapq模块 (堆排序)

```

"""
从列表中找出最大的或最小的N个元素
堆结构(大根堆/小根堆)
"""

import heapq

list1 = [34, 25, 12, 99, 87, 63, 58, 78, 88, 92]
list2 = [
    {'name': 'IBM', 'shares': 100, 'price': 91.1},

```

```

        {'name': 'AAPL', 'shares': 50, 'price': 543.22},
        {'name': 'FB', 'shares': 200, 'price': 21.09},
        {'name': 'HPQ', 'shares': 35, 'price': 31.75},
        {'name': 'YHOO', 'shares': 45, 'price': 16.35},
        {'name': 'ACME', 'shares': 75, 'price': 115.65}
    ]
print(heapq.nlargest(3, list1))
print(heapq.nsmallest(3, list1))
print(heapq.nlargest(2, list2, key=lambda x: x['price']))
print(heapq.nlargest(2, list2, key=lambda x: x['shares']))

```

- **itertools**模块

```

"""
迭代工具模块
"""
import itertools

# 产生ABCD的全排列
itertools.permutations('ABCD')
# 产生ABCDE的五选三组合
itertools.combinations('ABCDE', 3)
# 产生ABCD和123的笛卡尔积
itertools.product('ABCD', '123')
# 产生ABC的无限循环序列
itertools.cycle(('A', 'B', 'C'))

```

- **collections**模块

常用的工具类：

- **namedtuple**：命名元组，它是一个类工厂，接受类型的名称和属性列表来创建一个类。
- **deque**：双端队列，是列表的替代实现。Python中的列表底层是基于数组来实现的，而deque底层是双向链表，因此当你需要在头尾添加和删除元素时，deque会表现出更好的性能，渐近时间复杂度为\$O(1)\$。
- **Counter**：**dict**的子类，键是元素，值是元素的计数，它的**most_common()**方法可以帮助我们获取出现频率最高的元素。**Counter**和**dict**的继承关系我认为是值得商榷的，按照CARP原则，**Counter**跟**dict**的关系应该设计为关联关系更为合理。
- **OrderedDict**：**dict**的子类，它记录了键值对插入的顺序，看起来既有字典的行为，也有链表的行为。
- **defaultdict**：类似于字典类型，但是可以通过默认的工厂函数来获得键对应的默认值，相比字典中的**setdefault()**方法，这种做法更加高效。

```

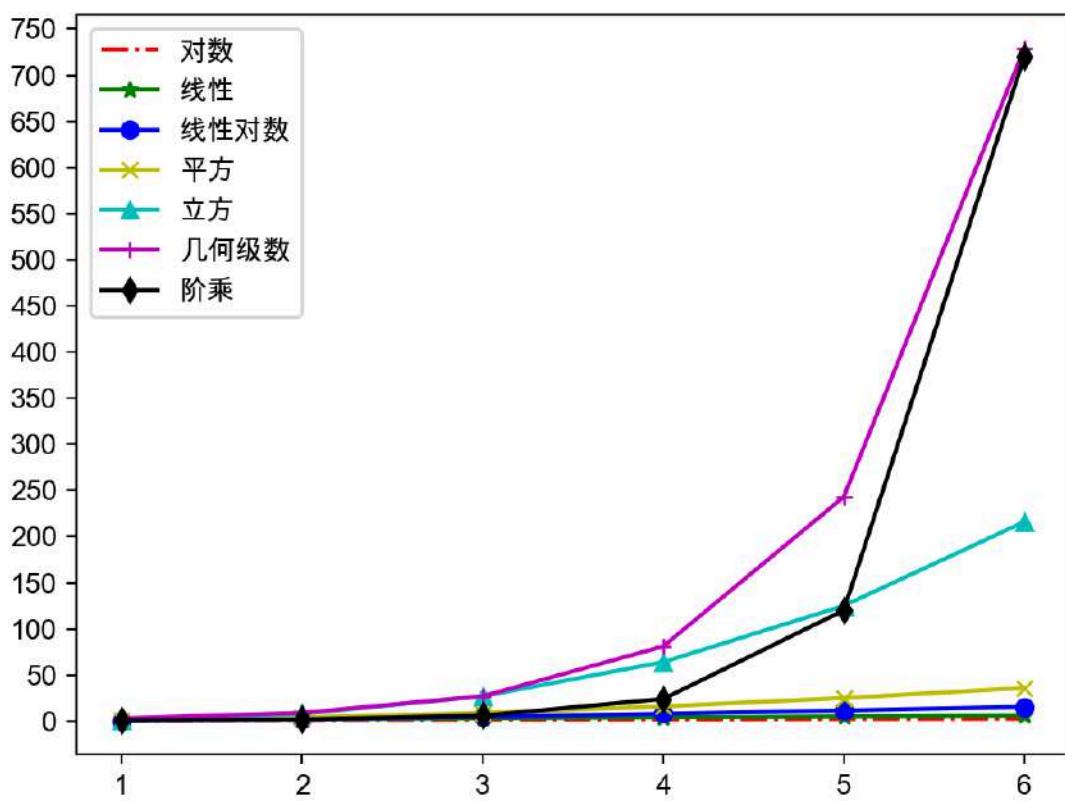
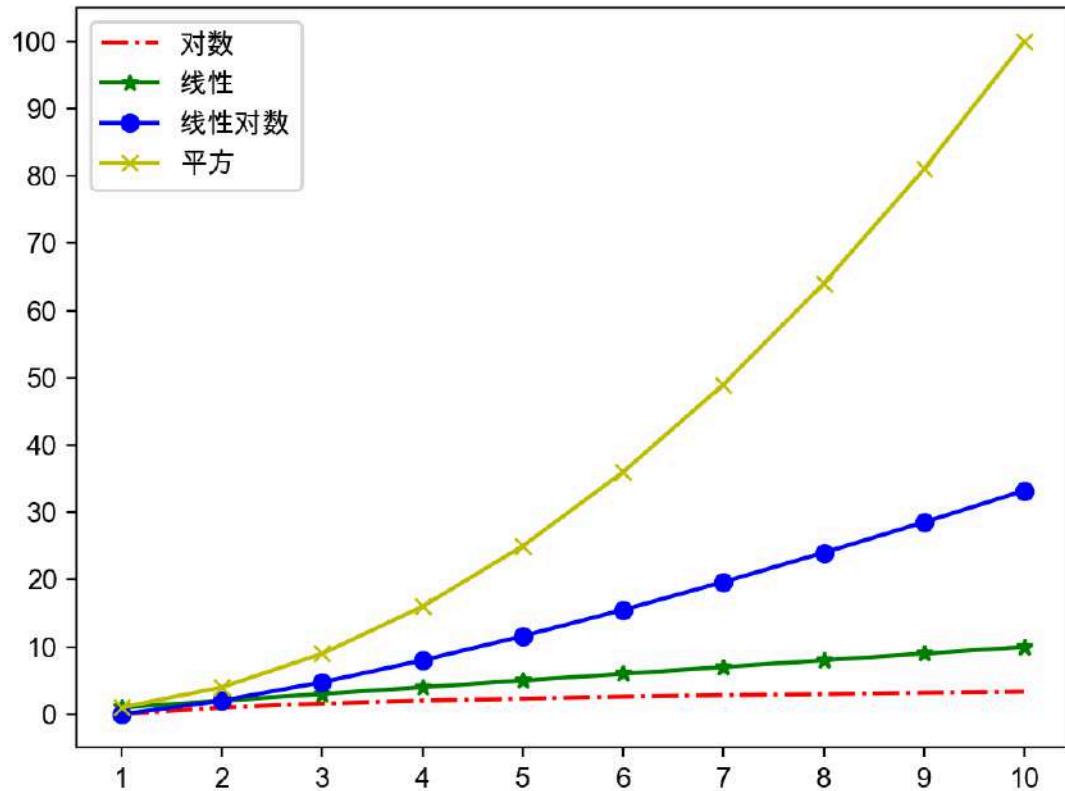
"""
找出序列中出现次数最多的元素
"""
from collections import Counter

```

```
words = [
    'look', 'into', 'my', 'eyes', 'look', 'into', 'my', 'eyes',
    'the', 'eyes', 'the', 'eyes', 'the', 'eyes', 'not', 'around',
    'the', 'eyes', "don't", 'look', 'around', 'the', 'eyes',
    'look', 'into', 'my', 'eyes', "you're", 'under'
]
counter = Counter(words)
print(counter.most_common(3))
```

数据结构和算法

- 算法：解决问题的方法和步骤
- 评价算法的好坏：渐近时间复杂度和渐近空间复杂度。
- 渐近时间复杂度的大O标记：
 - $O(c)$ - 常量时间复杂度 - 布隆过滤器 / 哈希存储
 - $O(\log_2 n)$ - 对数时间复杂度 - 折半查找 (二分查找)
 - $O(n)$ - 线性时间复杂度 - 顺序查找 / 计数排序
 - $O(n * \log_2 n)$ - 对数线性时间复杂度 - 高级排序算法 (归并排序、快速排序)
 - $O(n^2)$ - 平方时间复杂度 - 简单排序算法 (选择排序、插入排序、冒泡排序)
 - $O(n^3)$ - 立方时间复杂度 - Floyd算法 / 矩阵乘法运算
 - $O(2^n)$ - 几何级数时间复杂度 - 汉诺塔
 - $O(n!)$ - 阶乘时间复杂度 - 旅行经销商问题 - NPC



- 排序算法（选择、冒泡和归并）和查找算法（顺序和折半）

```
def select_sort(items, comp=lambda x, y: x < y):
    """简单选择排序"""
    items = items[:]
    for i in range(len(items) - 1):
        min_index = i
        for j in range(i + 1, len(items)):
            if comp(items[j], items[min_index]):
                min_index = j
        items[i], items[min_index] = items[min_index], items[i]
    return items
```

```
def bubble_sort(items, comp=lambda x, y: x > y):
    """冒泡排序"""
    items = items[:]
    for i in range(len(items) - 1):
        swapped = False
        for j in range(len(items) - 1 - i):
            if comp(items[j], items[j + 1]):
                items[j], items[j + 1] = items[j + 1], items[j]
                swapped = True
        if not swapped:
            break
    return items
```

```
def bubble_sort(items, comp=lambda x, y: x > y):
    """搅拌排序(冒泡排序升级版)"""
    items = items[:]
    for i in range(len(items) - 1):
        swapped = False
        for j in range(len(items) - 1 - i):
            if comp(items[j], items[j + 1]):
                items[j], items[j + 1] = items[j + 1], items[j]
                swapped = True
        if swapped:
            swapped = False
            for j in range(len(items) - 2 - i, i, -1):
                if comp(items[j - 1], items[j]):
                    items[j], items[j - 1] = items[j - 1], items[j]
                    swapped = True
        if not swapped:
            break
    return items
```

```
def merge(items1, items2, comp=lambda x, y: x < y):
    """合并(将两个有序的列表合并成一个有序的列表)"""
    items = []
```

```

index1, index2 = 0, 0
while index1 < len(items1) and index2 < len(items2):
    if comp(items1[index1], items2[index2]):
        items.append(items1[index1])
        index1 += 1
    else:
        items.append(items2[index2])
        index2 += 1
    items += items1[index1:]
    items += items2[index2:]
return items

def merge_sort(items, comp=lambda x, y: x < y):
    return _merge_sort(list(items), comp)

def _merge_sort(items, comp):
    """归并排序"""
    if len(items) < 2:
        return items
    mid = len(items) // 2
    left = _merge_sort(items[:mid], comp)
    right = _merge_sort(items[mid:], comp)
    return merge(left, right, comp)

```

```

def seq_search(items, key):
    """顺序查找"""
    for index, item in enumerate(items):
        if item == key:
            return index
    return -1

```

```

def bin_search(items, key):
    """折半查找"""
    start, end = 0, len(items) - 1
    while start <= end:
        mid = (start + end) // 2
        if key > items[mid]:
            start = mid + 1
        elif key < items[mid]:
            end = mid - 1
        else:
            return mid
    return -1

```

- 常用算法:

- 穷举法 - 又称为暴力破解法，对所有的可能性进行验证，直到找到正确答案。
- 贪婪法 - 在对问题求解时，总是做出在当前看来最好的选择，不追求最优解，快速找到满意解。
- 分治法 - 把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题，直到可以直接求解的程度，最后将子问题的解进行合并得到原问题的解。
- 回溯法 - 回溯法又称为试探法，按选优条件向前搜索，当搜索到某一步发现原先选择并不优或达不到目标时，就退回一步重新选择。
- 动态规划 - 基本思想也是将待求解问题分解成若干个子问题，先求解并保存这些子问题的解，避免产生大量的重复运算。

穷举法例子：百钱百鸡和五人分鱼。

```

# 公鸡5元一只 母鸡3元一只 小鸡1元三只
# 用100元买100只鸡 问公鸡/母鸡/小鸡各多少只
for x in range(20):
    for y in range(33):
        z = 100 - x - y
        if 5 * x + 3 * y + z // 3 == 100 and z % 3 == 0:
            print(x, y, z)

# A、B、C、D、E五人在某天夜里合伙捕鱼 最后疲惫不堪各自睡觉
# 第二天A第一个醒来 他将鱼分为5份 扔掉多余的1条 拿走自己的一份
# B第二个醒来 也将鱼分为5份 扔掉多余的1条 拿走自己的一份
# 然后C、D、E依次醒来也按同样的方式分鱼 问他们至少捕了多少条鱼
fish = 6
while True:
    total = fish
    enough = True
    for _ in range(5):
        if (total - 1) % 5 == 0:
            total = (total - 1) // 5 * 4
        else:
            enough = False
            break
    if enough:
        print(fish)
        break
    fish += 5

```

贪婪法例子：假设小偷有一个背包，最多能装20公斤赃物，他闯入一户人家，发现如下表所示的物品。很显然，他不能把所有物品都装进背包，所以必须确定拿走哪些物品，留下哪些物品。

名称	价格 (美元)	重量 (kg)
电脑	200	20
收音机	20	4
钟	175	10
花瓶	50	2

名称	价格 (美元)	重量 (kg)
书	10	1
油画	90	9

....
贪婪法：在对问题求解时，总是做出在当前看来是最好的选择，不追求最优解，快速找到满意解。

输入：

```
20 6
电脑 200 20
收音机 20 4
钟 175 10
花瓶 50 2
书 10 1
油画 90 9
....
```

```
class Thing(object):
    """物品"""

    def __init__(self, name, price, weight):
        self.name = name
        self.price = price
        self.weight = weight

    @property
    def value(self):
        """价格重量比"""
        return self.price / self.weight

def input_thing():
    """输入物品信息"""
    name_str, price_str, weight_str = input().split()
    return name_str, int(price_str), int(weight_str)

def main():
    """主函数"""
    max_weight, num_of_things = map(int, input().split())
    all_things = []
    for _ in range(num_of_things):
        all_things.append(Thing(*input_thing()))
    all_things.sort(key=lambda x: x.value, reverse=True)
    total_weight = 0
    total_price = 0
    for thing in all_things:
        if total_weight + thing.weight <= max_weight:
            print(f'小偷拿走了{thing.name}')
            total_weight += thing.weight
            total_price += thing.price
    print(f'总价值: {total_price}美元')
```

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

分治法例子：快速排序。

```
"""
快速排序 - 选择枢轴对元素进行划分，左边都比枢轴小右边都比枢轴大
"""
def quick_sort(items, comp=lambda x, y: x <= y):
    items = list(items)[:]
    _quick_sort(items, 0, len(items) - 1, comp)
    return items

def _quick_sort(items, start, end, comp):
    if start < end:
        pos = _partition(items, start, end, comp)
        _quick_sort(items, start, pos - 1, comp)
        _quick_sort(items, pos + 1, end, comp)

def _partition(items, start, end, comp):
    pivot = items[end]
    i = start - 1
    for j in range(start, end):
        if comp(items[j], pivot):
            i += 1
            items[i], items[j] = items[j], items[i]
    items[i + 1], items[end] = items[end], items[i + 1]
    return i + 1
```

回溯法例子：骑士巡逻。

```
"""
递归回溯法：叫称为试探法，按选优条件向前搜索，当搜索到某一步，发现原先选择并不优或达不到目标时，就退回一步重新选择，比较经典的问题包括骑士巡逻、八皇后和迷宫寻路等。
"""
import sys
import time

SIZE = 5
total = 0

def print_board(board):
    for row in board:
        for col in row:
```

```

        print(str(col).center(4), end=' ')
        print()

def patrol(board, row, col, step=1):
    if row >= 0 and row < SIZE and \
       col >= 0 and col < SIZE and \
       board[row][col] == 0:
        board[row][col] = step
    if step == SIZE * SIZE:
        global total
        total += 1
        print(f'第{total}种走法: ')
        print_board(board)
        patrol(board, row - 2, col - 1, step + 1)
        patrol(board, row - 1, col - 2, step + 1)
        patrol(board, row + 1, col - 2, step + 1)
        patrol(board, row + 2, col - 1, step + 1)
        patrol(board, row + 2, col + 1, step + 1)
        patrol(board, row + 1, col + 2, step + 1)
        patrol(board, row - 1, col + 2, step + 1)
        patrol(board, row - 2, col + 1, step + 1)
        board[row][col] = 0

def main():
    board = [[0] * SIZE for _ in range(SIZE)]
    patrol(board, SIZE - 1, SIZE - 1)

if __name__ == '__main__':
    main()

```

动态规划例子：子列表元素之和的最大值。

说明：子列表指的是列表中索引（下标）连续的元素构成的列表；列表中的元素是int类型，可能包含正整数、0、负整数；程序输入列表中的元素，输出子列表元素求和的最大值，例如：

输入：1 -2 3 5 -3 2

输出：8

输入：0 -2 3 5 -1 2

输出：9

输入：-9 -2 -3 -5 -3

输出：-2

```

def main():
    items = list(map(int, input().split()))

```

```

overall = partial = items[0]
for i in range(1, len(items)):
    partial = max(items[i], partial + items[i])
    overall = max(partial, overall)
print(overall)

if __name__ == '__main__':
    main()

```

说明：这个题目最容易想到的解法是使用二重循环，但是代码的时间性能将会变得非常的糟糕。
使用动态规划的思想，仅仅是多用了两个变量，就将原来 $O(N^2)$ 复杂度的问题变成了 $O(N)$ 。

函数的使用方式

- 将函数视为“一等公民”
 - 函数可以赋值给变量
 - 函数可以作为函数的参数
 - 函数可以作为函数的返回值
- 高阶函数的用法 (`filter`、`map`以及它们的替代品)

```

items1 = list(map(lambda x: x ** 2, filter(lambda x: x % 2, range(1, 10))))
items2 = [x ** 2 for x in range(1, 10) if x % 2]

```

- 位置参数、可变参数、关键字参数、命名关键字参数
- 参数的元信息（代码可读性问题）
- 匿名函数和内联函数的用法（`lambda`函数）
 - Python搜索变量的LEGB顺序（Local >>> Embedded >>> Global >>> Built-in）
 - `global`和`nonlocal`关键字的作用

`global`：声明或定义全局变量（要么直接使用现有的全局作用域的变量，要么定义一个变量放到全局作用域）。

`nonlocal`：声明使用嵌套作用域的变量（嵌套作用域必须存在该变量，否则报错）。
- 装饰器函数（使用装饰器和取消装饰器）

例子：输出函数执行时间的装饰器。

```

def record_time(func):
    """自定义装饰函数的装饰器"""

```

```
@wraps(func)
def wrapper(*args, **kwargs):
    start = time()
    result = func(*args, **kwargs)
    print(f'{func.__name__}: {time() - start}秒')
    return result

return wrapper
```

如果装饰器不希望跟print函数耦合，可以编写可以参数化的装饰器。

```
from functools import wraps
from time import time

def record(output):
    """可以参数化的装饰器"""

    def decorate(func):

        @wraps(func)
        def wrapper(*args, **kwargs):
            start = time()
            result = func(*args, **kwargs)
            output(func.__name__, time() - start)
            return result

        return wrapper

    return decorate
```

```
from functools import wraps
from time import time

class Record():
    """通过定义类的方式定义装饰器"""

    def __init__(self, output):
        self.output = output

    def __call__(self, func):

        @wraps(func)
        def wrapper(*args, **kwargs):
            start = time()
            result = func(*args, **kwargs)
            self.output(func.__name__, time() - start)
            return result
```

```
    return wrapper
```

说明：由于对带装饰功能的函数添加了@wraps装饰器，可以通过`func.__wrapped__`方式获得被装饰之前的函数或类来取消装饰器的作用。

例子：用装饰器来实现单例模式。

```
from functools import wraps

def singleton(cls):
    """装饰类的装饰器"""
    instances = {}

    @wraps(cls)
    def wrapper(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]

    return wrapper

@singletton
class President:
    """总统(单例类)"""
    pass
```

提示：上面的代码中用到了闭包（closure），不知道你是否已经意识到了。还没有一个小问题就是，上面的代码并没有实现线程安全的单例，如果要实现线程安全的单例应该怎么做呢？

线程安全的单例装饰器。

```
from functools import wraps
from threading import RLock

def singleton(cls):
    """线程安全的单例装饰器"""
    instances = {}
    locker = RLock()

    @wraps(cls)
    def wrapper(*args, **kwargs):
        if cls not in instances:
            with locker:
                if cls not in instances:
                    instances[cls] = cls(*args, **kwargs)
```

```
    return instances[cls]

    return wrapper
```

提示：上面的代码用到了with上下文语法来进行锁操作，因为锁对象本身就是上下文管理器对象（支持__enter__和__exit__魔术方法）。在wrapper函数中，我们先做了一次不带锁的检查，然后再做带锁的检查，这样做比直接加锁检查性能要更好，如果对象已经创建就没有必要再去加锁而是直接返回该对象就可以了。

面向对象相关知识

- 三大支柱：封装、继承、多态

例子：工资结算系统。

```
"""
月薪结算系统 - 部门经理每月15000 程序员每小时200 销售员1800底薪加销售额5%提成
"""

from abc import ABCMeta, abstractmethod

class Employee(metaclass=ABCMeta):
    """员工(抽象类)"""

    def __init__(self, name):
        self.name = name

    @abstractmethod
    def get_salary(self):
        """结算月薪(抽象方法)"""
        pass

class Manager(Employee):
    """部门经理"""

    def get_salary(self):
        return 15000.0

class Programmer(Employee):
    """程序员"""

    def __init__(self, name, working_hour=0):
        self.working_hour = working_hour
        super().__init__(name)

    def get_salary(self):
        return 200.0 * self.working_hour
```

```

class Salesman(Employee):
    """销售员"""

    def __init__(self, name, sales=0.0):
        self.sales = sales
        super().__init__(name)

    def get_salary(self):
        return 1800.0 + self.sales * 0.05


class EmployeeFactory:
    """创建员工的工厂 (工厂模式 - 通过工厂实现对象使用者和对象之间的解耦合)"""

    @staticmethod
    def create(emp_type, *args, **kwargs):
        """创建员工"""
        all_emp_types = {'M': Manager, 'P': Programmer, 'S': Salesman}
        cls = all_emp_types[emp_type.upper()]
        return cls(*args, **kwargs) if cls else None


def main():
    """主函数"""
    emps = [
        EmployeeFactory.create('M', '曹操'),
        EmployeeFactory.create('P', '荀彧', 120),
        EmployeeFactory.create('P', '郭嘉', 85),
        EmployeeFactory.create('S', '典韦', 123000),
    ]
    for emp in emps:
        print(f'{emp.name}: {emp.get_salary():.2f}元')

if __name__ == '__main__':
    main()

```

- 类与类之间的关系

- is-a关系: 继承
- has-a关系: 关联 / 聚合 / 合成
- use-a关系: 依赖

例子: 扑克游戏。

```

"""
经验: 符号常量总是优于字面常量, 枚举类型是定义符号常量的最佳选择
"""
from enum import Enum, unique

import random

```

```
@unique
class Suite(Enum):
    """花色"""

    SPADE, HEART, CLUB, DIAMOND = range(4)

    def __lt__(self, other):
        return self.value < other.value


class Card:
    """牌"""

    def __init__(self, suite, face):
        """初始化方法"""
        self.suite = suite
        self.face = face

    def show(self):
        """显示牌面"""
        suites = ['♠', '♥', '♣', '♦']
        faces = ['', 'A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J',
        'Q', 'K']
        return f'{suites[self.suite.value]}{faces[self.face]}'

    def __repr__(self):
        return self.show()


class Poker:
    """扑克"""

    def __init__(self):
        self.index = 0
        self.cards = [Card(suite, face)
                     for suite in Suite
                     for face in range(1, 14)]


    def shuffle(self):
        """洗牌 (随机乱序) """
        random.shuffle(self.cards)
        self.index = 0


    def deal(self):
        """发牌"""
        card = self.cards[self.index]
        self.index += 1
        return card


    @property
    def has_more(self):
        return self.index < len(self.cards)
```

```

class Player:
    """玩家"""

    def __init__(self, name):
        self.name = name
        self.cards = []

    def get_one(self, card):
        """摸一张牌"""
        self.cards.append(card)

    def sort(self, comp=lambda card: (card.suite, card.face)):
        """整理手上的牌"""
        self.cards.sort(key=comp)

def main():
    """主函数"""
    poker = Poker()
    poker.shuffle()
    players = [Player('东邪'), Player('西毒'), Player('南帝'), Player('北丐')]
    while poker.has_more():
        for player in players:
            player.get_one(poker.deal())
    for player in players:
        player.sort()
        print(player.name, end=': ')
        print(player.cards)

if __name__ == '__main__':
    main()

```

说明：上面的代码中使用了Emoji字符来表示扑克牌的四种花色，在某些不支持Emoji字符的系统上可能无法显示。

- 对象的复制（深复制/深拷贝/深度克隆和浅复制/浅拷贝/影子克隆）
- 垃圾回收、循环引用和弱引用

Python使用了自动化内存管理，这种管理机制以**引用计数**为基础，同时也引入了**标记-清除**和**分代收集**两种机制为辅的策略。

```

typedef struct _object {
    /* 引用计数 */
    int ob_refcnt;
    /* 对象指针 */
    struct _typeobject *ob_type;
} PyObject;

```

```

/* 增加引用计数的宏定义 */
#define Py_INCREF(op) ((op)->ob_refcnt++)
/* 减少引用计数的宏定义 */
#define Py_DECREF(op) \ //减少计数
    if (--(op)->ob_refcnt != 0) \
    ; \
else \
    __Py_Dealloc((PyObject *) (op))

```

导致引用计数+1的情况：

- 对象被创建，例如`a = 23`
- 对象被引用，例如`b = a`
- 对象被作为参数，传入到一个函数中，例如`f(a)`
- 对象作为一个元素，存储在容器中，例如`list1 = [a, a]`

导致引用计数-1的情况：

- 对象的别名被显式销毁，例如`del a`
- 对象的别名被赋予新的对象，例如`a = 24`
- 一个对象离开它的作用域，例如`f`函数执行完毕时，`f`函数中的局部变量（全局变量不会）
- 对象所在的容器被销毁，或从容器中删除对象

引用计数可能会导致循环引用问题，而循环引用会导致内存泄露，如下面的代码所示。为了解决这个问题，Python中引入了“标记-清除”和“分代收集”。在创建一个对象的时候，对象被放在第一代中，如果在第一代的垃圾检查中对象存活了下来，该对象就会被放到第二代中，同理在第二代的垃圾检查中对象存活下来，该对象就会被放到第三代中。

```

# 循环引用会导致内存泄露 - Python除了引用技术还引入了标记清理和分代回收
# 在Python 3.6以前如果重写__del__魔术方法会导致循环引用处理失效
# 如果不想造成循环引用可以使用弱引用
list1 = []
list2 = []
list1.append(list2)
list2.append(list1)

```

以下情况会导致垃圾回收：

- 调用`gc.collect()`
- `gc`模块的计数器达到阈值
- 程序退出

如果循环引用中两个对象都定义了`__del__`方法，`gc`模块不会销毁这些不可达对象，因为`gc`模块不知道应该先调用哪个对象的`__del__`方法，这个问题在Python 3.6中得到了解决。

也可以通过`weakref`模块构造弱引用的方式来解决循环引用的问题。

- 魔法属性和方法（请参考《Python魔法方法指南》）

有几个小问题请大家思考：

- 自定义的对象能不能使用运算符做运算？
- 自定义的对象能不能放到set中？能去重吗？
- 自定义的对象能不能作为dict的键？
- 自定义的对象能不能使用上下文语法？
- 混入 (Mixin)

例子：自定义字典限制只有在指定的key不存在时才能在字典中设置键值对。

```
class SetOnceMappingMixin:
    """自定义混入类"""
    __slots__ = ()

    def __setitem__(self, key, value):
        if key in self:
            raise KeyError(str(key) + ' already set')
        return super().__setitem__(key, value)

class SetOnceDict(SetOnceMappingMixin, dict):
    """自定义字典"""
    pass

my_dict= SetOnceDict()
try:
    my_dict['username'] = 'jackfrued'
    my_dict['username'] = 'hellokitty'
except KeyError:
    pass
print(my_dict)
```

- 元编程和元类

对象是通过类创建的，类是通过元类创建的，元类提供了创建类的元信息。所有的类都直接或间接的继承自object，所有的元类都直接或间接的继承自type。

例子：用元类实现单例模式。

```
import threading

class SingletonMeta(type):
    """自定义元类"""

    def __init__(cls, *args, **kwargs):
        cls.__instance = None
        cls.__lock = threading.RLock()
```

```

super().__init__(*args, **kwargs)

def __call__(cls, *args, **kwargs):
    if cls.__instance is None:
        with cls.__lock:
            if cls.__instance is None:
                cls.__instance = super().__call__(*args, **kwargs)
    return cls.__instance

class President(metaclass=SingletonMeta):
    """总统(单例类)"""

    pass

```

- 面向对象设计原则

- 单一职责原则 (SRP) - 一个类只做该做的事情 (类的设计要高内聚)
- 开闭原则 (OCP) - 软件实体应该对扩展开发对修改关闭
- 依赖倒转原则 (DIP) - 面向抽象编程 (在弱类型语言中已经被弱化)
- 里氏替换原则 (LSP) - 任何时候可以用子类对象替换掉父类对象
- 接口隔离原则 (ISP) - 接口要小而专不要大而全 (Python中没有接口的概念)
- 合成聚合复用原则 (CARP) - 优先使用强关联关系而不是继承关系复用代码
- 最少知识原则 (迪米特法则, LoD) - 不要给没有必然联系的对象发消息

说明：上面加粗的字母放在一起称为面向对象的**SOLID**原则。

- GoF设计模式

- 创建型模式：单例、工厂、建造者、原型
- 结构型模式：适配器、门面（外观）、代理
- 行为型模式：迭代器、观察者、状态、策略

例子：可插拔的哈希算法（策略模式）。

```

class StreamHasher():
    """哈希摘要生成器"""

    def __init__(self, alg='md5', size=4096):
        self.size = size
        alg = alg.lower()
        self.hasher = getattr(__import__('hashlib'), alg.lower())()

    def __call__(self, stream):
        return self.to_digest(stream)

    def to_digest(self, stream):
        """生成十六进制形式的摘要"""
        for buf in iter(lambda: stream.read(self.size), b''):
            self.hasher.update(buf)
        return self.hasher.hexdigest()

```

```

def main():
    """主函数"""
    hasher1 = StreamHasher()
    with open('Python-3.7.6.tgz', 'rb') as stream:
        print(hasher1.to_digest(stream))
    hasher2 = StreamHasher('sha1')
    with open('Python-3.7.6.tgz', 'rb') as stream:
        print(hasher2(stream))

if __name__ == '__main__':
    main()

```

迭代器和生成器

- 迭代器是实现了迭代器协议的对象。
 - Python中没有像protocol或interface这样的定义协议的关键字。
 - Python中用魔术方法表示协议。
 - __iter__和__next__魔术方法就是迭代器协议。

```

class Fib(object):
    """迭代器"""

    def __init__(self, num):
        self.num = num
        self.a, self.b = 0, 1
        self.idx = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.idx < self.num:
            self.a, self.b = self.b, self.a + self.b
            self.idx += 1
            return self.a
        raise StopIteration()

```

- 生成器是语法简化版的迭代器。

```

def fib(num):
    """生成器"""
    a, b = 0, 1
    for _ in range(num):
        a, b = b, a + b
        yield a

```

- 生成器进化为协程。

生成器对象可以使用`send()`方法发送数据，发送的数据会成为生成器函数中通过`yield`表达式获得的值。这样，生成器就可以作为协程使用，协程简单的说就是可以相互协作的子程序。

```
def calc_avg():
    """流式计算平均值"""
    total, counter = 0, 0
    avg_value = None
    while True:
        value = yield avg_value
        total, counter = total + value, counter + 1
        avg_value = total / counter

gen = calc_avg()
next(gen)
print(gen.send(10))
print(gen.send(20))
print(gen.send(30))
```

并发编程

Python中实现并发编程的三种方案：多线程、多进程和异步I/O。并发编程的好处在于可以提升程序的执行效率以及改善用户体验；坏处在于并发的程序不容易开发和调试，同时对其他程序来说它并不友好。

- 多线程：Python中提供了`Thread`类并辅以`Lock`、`Condition`、`Event`、`Semaphore`和`Barriern`。Python中有GIL来防止多个线程同时执行本地字节码，这个锁对于CPython是必须的，因为CPython的内存管理并不是线程安全的，因为GIL的存在多线程并不能发挥CPU的多核特性。

```
"""
面试题：进程和线程的区别和联系？
进程 - 操作系统分配内存的基本单位 - 一个进程可以包含一个或多个线程
线程 - 操作系统分配CPU的基本单位
并发编程 (concurrent programming)
1. 提升执行性能 - 让程序中没有因果关系的部分可以并发的执行
2. 改善用户体验 - 让耗时间的操作不会造成程序的假死
"""

import glob
import os
import threading
```

```
from PIL import Image
```

```
PREFIX = 'thumbnails'
```

```
def generate_thumbnail(infile, size, format='PNG'):
    """生成指定图片文件的缩略图"""
    file, ext = os.path.splitext(infile)
```

```

file = file[file.rfind('/') + 1:]
outfile = f'{PREFIX}/{file}_{size[0]}_{size[1]}.{ext}'
img = Image.open(infile)
img.thumbnail(size, Image.ANTIALIAS)
img.save(outfile, format)

def main():
    """主函数"""
    if not os.path.exists(PREFIX):
        os.mkdir(PREFIX)
    for infile in glob.glob('images/*.png'):
        for size in (32, 64, 128):
            # 创建并启动线程
            threading.Thread(
                target=generate_thumbnail,
                args=(infile, (size, size)))
            .start()

if __name__ == '__main__':
    main()

```

多个线程竞争资源的情况。

```

"""
多线程程序如果没有竞争资源处理起来通常也比较简单
当多个线程竞争临界资源的时候如果缺乏必要的保护措施就会导致数据错乱
说明：临界资源就是被多个线程竞争的资源
"""

import time
import threading

from concurrent.futures import ThreadPoolExecutor


class Account(object):
    """银行账户"""

    def __init__(self):
        self.balance = 0.0
        self.lock = threading.Lock()

    def deposit(self, money):
        # 通过锁保护临界资源
        with self.lock:
            new_balance = self.balance + money
            time.sleep(0.001)
            self.balance = new_balance

    def main():

```

```

"""主函数"""
account = Account()
# 创建线程池
pool = ThreadPoolExecutor(max_workers=10)
futures = []
for _ in range(100):
    future = pool.submit(account.deposit, 1)
    futures.append(future)
# 关闭线程池
pool.shutdown()
for future in futures:
    future.result()
print(account.balance)

if __name__ == '__main__':
    main()

```

修改上面的程序，启动5个线程向账户中存钱，5个线程从账户中取钱，取钱时如果余额不足就暂停线程进行等待。为了达到上述目标，需要对存钱和取钱的线程进行调度，在余额不足时取钱的线程暂停并释放锁，而存钱的线程将钱存入后要通知取钱的线程，使其从暂停状态被唤醒。可以使用`threading`模块的`Condition`来实现线程调度，该对象也是基于锁来创建的，代码如下所示：

```

"""
多个线程竞争一个资源 - 保护临界资源 - 锁 (Lock/RLock)
多个线程竞争多个资源 (线程数>资源数) - 信号量 (Semaphore)
多个线程的调度 - 暂停线程执行/唤醒等待中的线程 - Condition
"""

from concurrent.futures import ThreadPoolExecutor
from random import randint
from time import sleep

import threading

class Account:
    """
    银行账户
    """

    def __init__(self, balance=0):
        self.balance = balance
        lock = threading.RLock()
        self.condition = threading.Condition(lock)

    def withdraw(self, money):
        """
        取钱
        """
        with self.condition:
            while money > self.balance:
                self.condition.wait()
            new_balance = self.balance - money
            sleep(0.001)
            self.balance = new_balance

```

```

def deposit(self, money):
    """存钱"""
    with self.condition:
        new_balance = self.balance + money
        sleep(0.001)
        self.balance = new_balance
        self.condition.notify_all()

def add_money(account):
    while True:
        money = randint(5, 10)
        account.deposit(money)
        print(threading.current_thread().name,
              ':', money, '====>', account.balance)
        sleep(0.5)

def sub_money(account):
    while True:
        money = randint(10, 30)
        account.withdraw(money)
        print(threading.current_thread().name,
              ':', money, '<====', account.balance)
        sleep(1)

def main():
    account = Account()
    with ThreadPoolExecutor(max_workers=15) as pool:
        for _ in range(5):
            pool.submit(add_money, account)
        for _ in range(10):
            pool.submit(sub_money, account)

if __name__ == '__main__':
    main()

```

- 多进程：多进程可以有效的解决GIL的问题，实现多进程主要的类是Process，其他辅助的类跟threading模块中的类似，进程间共享数据可以使用管道、套接字等，在multiprocessing模块中有一个Queue类，它基于管道和锁机制提供了多个进程共享的队列。下面是官方文档上关于多进程和进程池的一个示例。

....

多进程和进程池的使用
 多线程因为GIL的存在不能够发挥CPU的多核特性
 对于计算密集型任务应该考虑使用多进程
 time python3 example22.py
 real 0m11.512s

```
user      0m39.319s
sys      0m0.169s
使用多进程后实际执行时间为11.512秒，而用户时间39.319秒约为实际执行时间的4倍
这就证明我们的程序通过多进程使用了CPU的多核特性，而且这台计算机配置了4核的CPU
"""

import concurrent.futures
import math

PRIMES = [
    1116281,
    1297337,
    104395303,
    472882027,
    533000389,
    817504243,
    982451653,
    112272535095293,
    112582705942171,
    112272535095293,
    115280095190773,
    115797848077099,
    1099726899285419
] * 5

def is_prime(n):
    """判断素数"""
    if n % 2 == 0:
        return False

    sqrt_n = int(math.floor(math.sqrt(n)))
    for i in range(3, sqrt_n + 1, 2):
        if n % i == 0:
            return False
    return True

def main():
    """主函数"""
    with concurrent.futures.ProcessPoolExecutor() as executor:
        for number, prime in zip(PRIMES, executor.map(is_prime, PRIMES)):
            print('%d is prime: %s' % (number, prime))

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

重点：多线程和多进程的比较。

以下情况需要使用多线程：

1. 程序需要维护许多共享的状态（尤其是可变状态），Python中的列表、字典、集合都是线程安全的，所以使用线程而不是进程维护共享状态的代价相对较小。

2. 程序会花费大量时间在I/O操作上，没有太多并行计算的需求且不需占用太多的内存。

以下情况需要使用多进程：

1. 程序执行计算密集型任务（如：字节码操作、数据处理、科学计算）。
2. 程序的输入可以并行的分成块，并且可以将运算结果合并。
3. 程序在内存使用方面没有任何限制且不强依赖于I/O操作（如：读写文件、套接字等）。

- 异步处理：从调度程序的任务队列中挑选任务，该调度程序以交叉的形式执行这些任务，我们并不能保证任务将以某种顺序去执行，因为执行顺序取决于队列中的一项任务是否愿意将CPU处理时间让位给另一项任务。异步任务通常通过多任务协作处理的方式来实现，由于执行时间和顺序的不确定，因此需要通过回调式编程或者future对象来获取任务执行的结果。Python 3通过asyncio模块和await和async关键字（在Python 3.7中正式被列为关键字）来支持异步处理。

```
"""
异步I/O - async / await
"""

import asyncio

def num_generator(m, n):
    """指定范围的数字生成器"""
    yield from range(m, n + 1)

async def prime_filter(m, n):
    """素数过滤器"""
    primes = []
    for i in num_generator(m, n):
        flag = True
        for j in range(2, int(i ** 0.5 + 1)):
            if i % j == 0:
                flag = False
                break
        if flag:
            print('Prime =>', i)
            primes.append(i)

        await asyncio.sleep(0.001)
    return tuple(primes)

async def square_mapper(m, n):
    """平方映射器"""
    squares = []
    for i in num_generator(m, n):
        print('Square =>', i * i)
        squares.append(i * i)

        await asyncio.sleep(0.001)
    return squares
```

```

def main():
    """主函数"""
    loop = asyncio.get_event_loop()
    future = asyncio.gather(prime_filter(2, 100), square_mapper(1, 100))
    future.add_done_callback(lambda x: print(x.result()))
    loop.run_until_complete(future)
    loop.close()

if __name__ == '__main__':
    main()

```

说明：上面的代码使用`get_event_loop`函数获得系统默认的事件循环，通过`gather`函数可以获得一个`future`对象，`future`对象的`add_done_callback`可以添加执行完成时的回调函数，`loop`对象的`run_until_complete`方法可以等待通过`future`对象获得协程执行结果。

Python中有一个名为[aiohttp](#)的三方库，它提供了异步的HTTP客户端和服务器，这个三方库可以跟[asyncio](#)模块一起工作，并提供了对[Future](#)对象的支持。Python 3.6中引入了[async](#)和[await](#)来定义异步执行的函数以及创建异步上下文，在Python 3.7中它们正式成为了关键字。下面的代码异步的从5个URL中获取页面并通过正则表达式的命名捕获组提取了网站的标题。

```

import asyncio
import re

import aiohttp

PATTERN = re.compile(r'<title>(?P<title>.**)</title>')

async def fetch_page(session, url):
    async with session.get(url, ssl=False) as resp:
        return await resp.text()

async def show_title(url):
    async with aiohttp.ClientSession() as session:
        html = await fetch_page(session, url)
        print(PATTERN.search(html).group('title'))

def main():
    urls = ('https://www.python.org/',
            'https://git-scm.com/',
            'https://www.jd.com/',
            'https://www.taobao.com/',
            'https://www.douban.com/')
    loop = asyncio.get_event_loop()
    cos = [show_title(url) for url in urls]
    loop.run_until_complete(asyncio.wait(cos))
    loop.close()

```

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

重点：异步I/O与多进程的比较。

当程序不需要真正的并发性或并行性，而是更多的依赖于异步处理和回调时，`asyncio`就是一种很好的选择。如果程序中有大量的等待与休眠时，也应该考虑`asyncio`，它很适合编写没有实时数据处理需求的Web应用服务器。

Python还有很多用于处理并行任务的三方库，例如：`joblib`、`PyMP`等。实际开发中，要提升系统的可扩展性和并发性通常有垂直扩展（增加单个节点的处理能力）和水平扩展（将单个节点变成多个节点）两种做法。可以通过消息队列来实现应用程序的解耦合，消息队列相当于是多线程同步队列的扩展版本，不同机器上的应用程序相当于就是线程，而共享的分布式消息队列就是原来程序中的Queue。消息队列（面向消息的中间件）的最流行和最标准化的实现是AMQP（高级消息队列协议），AMQP源于金融行业，提供了排队、路由、可靠传输、安全等功能，最著名的实现包括：Apache的ActiveMQ、RabbitMQ等。

要实现任务的异步化，可以使用名为`Celery`的三方库。`Celery`是Python编写的分布式任务队列，它使用分布式消息进行工作，可以基于RabbitMQ或Redis来作为后端的消息代理。

Web前端概述

说明：本文使用的一部分插图来自Jon Duckett先生的*HTML and CSS: Design and Build Websites*—书，这是一本非常棒的前端入门书，有兴趣的读者可以在亚马逊或者其他网站上找到该书的购买链接。

HTML 是用来描述网页的一种语言，全称是 Hyper-Text Markup Language，即超文本标记语言。我们浏览网页时看到的文字、按钮、图片、视频等元素，它们都是通过 HTML 书写并通过浏览器来呈现的。

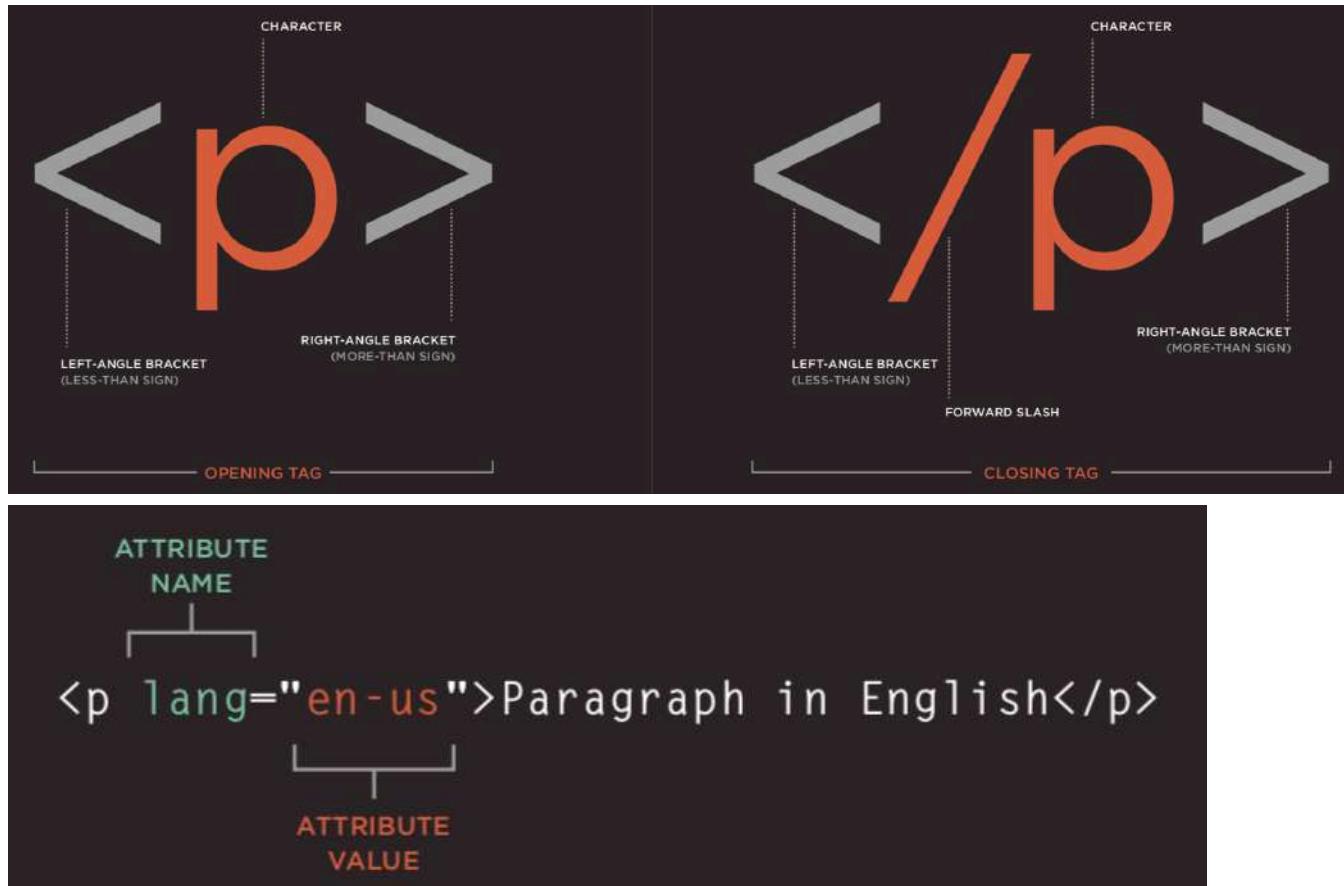
HTML简史

1. 1991年10月：一个非正式CERN（[欧洲核子研究中心](#)）文件首次公开18个HTML标签，这个文件的作者是物理学家[蒂姆·伯纳斯-李](#)，因此他是[万维网](#)的发明者，也是[万维网联盟](#)的主席。
2. 1995年11月：HTML 2.0标准发布（RFC 1866）。
3. 1997年1月：HTML 3.2作为[W3C](#)推荐标准发布。
4. 1997年12月：HTML 4.0作为W3C推荐标准发布。
5. 1999年12月：HTML4.01作为W3C推荐标准发布。
6. 2008年1月：HTML5由W3C作为工作草案发布。
7. 2011年5月：W3C将HTML5推进至“最终征求”（Last Call）阶段。
8. 2012年12月：W3C指定HTML5作为“候选推荐”阶段。
9. 2014年10月：HTML5作为稳定W3C推荐标准发布，这意味着HTML5的标准化已经完成。

HTML5新特性

1. 引入原生多媒体支持（audio和video标签）
2. 引入可编程内容（canvas标签）
3. 引入语义Web（article、aside、details、figure、footer、header、nav、section、summary等标签）
4. 引入新的表单控件（日历、邮箱、搜索、滑条等）
5. 引入对离线存储更好的支持（localStorage和sessionStorage）
6. 引入对定位、拖放、WebSocket、后台任务等的支持

使用标签承载内容



结构

- html
 - head
 - title
 - meta
 - body

文本

- 标题 (heading) 和段落 (paragraph)
 - h1 ~ h6
 - p
- 上标 (superscript) 和下标 (subscript)
 - sup
 - sub
- 空白 (白色空间折叠)
- 折行 (break) 和水平标尺 (horizontal ruler)
 - br
 - hr
- 语义化标签
 - 加粗和强调 - strong
 - 引用 - blockquote
 - 缩写词和首字母缩写词 - abbr / acronym
 - 引文 - cite
 - 所有者联系信息 - address

- 内容的修改 - ins / del

列表 (list)

- 有序列表 (ordered list) - ol / li
- 无序列表 (unordered list) - ul / li
- 定义列表 (definition list) - dl / dt / dd

链接 (anchor)

- 页面链接
- 锚链接
- 功能链接

图像 (image)

- 图像存储位置

RELATIVE LINK TYPE

EXAMPLE (from diagram on previous page)

SAME FOLDER

To link to a file in the same folder, just use the file name. (Nothing else is needed.)

To link to music reviews from the music homepage:
`Reviews`

CHILD FOLDER

For a child folder, use the name of the child folder, followed by a forward slash, then the file name.

To link to music listings from the homepage:
`Listings`

GRANDCHILD FOLDER

Use the name of the child folder, followed by a forward slash, then the name of the grandchild folder, followed by another forward slash, then the file name.

To link to DVD reviews from the homepage:
`Reviews`

PARENT FOLDER

Use .. / to indicate the folder above the current one, then follow it with the file name.

To link to the homepage from the music reviews:
`Home`

GRANDPARENT FOLDER

Repeat the .. / to indicate that you want to go up two folders (rather than one), then follow it with the file name.

To link to the homepage from the DVD reviews:
`Home`

- 图像及其宽高
- 选择正确的图像格式
 - JPEG
 - GIF
 - PNG

- 矢量图
- 语义化标签 - figure / figcaption

表格 (table)

- 基本的表格结构 - table / tr / td / th
- 表格的标题 - caption
- 跨行和跨列 - rowspan属性 / colspan属性
- 长表格 - thead / tbody / tfoot

表单 (form)

- 重要属性 - action / method / enctype
- 表单控件 (input) - type属性
 - 文本框 - text / 密码框 - password / 数字框 - number
 - 邮箱 - email / 电话 - tel / 日期 - date / 滑条 - range / URL - url / 搜索 - search
 - 单选按钮 - radio / 复选按钮 - checkbox
 - 文件上传 - file / 隐藏域 - hidden
 - 提交按钮 - submit / 图像按钮 - image / 重置按钮 - reset
- 下拉列表 - select / option
- 文本域 (多行文本) - textarea
- 组合表单元素 - fieldset / legend

音视频 (audio / video)

- 视频格式和播放器
- 视频托管服务
- 添加视频的准备工作
- video标签和属性 - autoplay / controls / loop / muted / preload / src
- audio标签和属性 - autoplay / controls / loop / muted / preload / src / width / height / poster

窗口 (frame)

- 框架集 (过时, 不建议使用) - frameset / frame
- 内嵌窗口 - iframe

其他

- 文档类型

```
<!doctype html>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

- **注释**

```
<!-- 这是一段注释，注释不能够嵌套 -->
```

- **属性**

- **id**: 唯一标识
- **class**: 元素所属的类，用于区分不同的元素
- **title**: 元素的额外信息（鼠标悬浮时会显示工具提示文本）
- **tabindex**: Tab键切换顺序
- **contenteditable**: 元素是否可编辑
- **draggable**: 元素是否可拖拽

- **块级元素 / 行级元素**

- **字符实体（实体替换符）**

<	Less-than sign < <	€	Cent sign ¢ ¢	'	Left single quote ‘ ‘
>	Greater-than sign > >	£	Pound sign £ £	'	Right single quote ’ ’
&	Ampersand & &	¥	Yen sign ¥ ¥	“	Left double quotes “ “
“	Quotation mark " "	€	Euro sign € €	”	Right double quotes ” ”
©	Copyright symbol © ©	®	Registered trademark ® ®	×	Multiplication sign × ×
™	Trademark ™ ™			÷	Division sign ÷ ÷

使用CSS渲染页面

简介

- CSS的作用
- CSS的工作原理
- 规则、属性和值



- 常用选择器

SELECTOR	MEANING	EXAMPLE
UNIVERSAL SELECTOR	Applies to all elements in the document	* [] Targets all elements on the page
TYPE SELECTOR	Matches element names	h1, h2, h3 [] Targets the <h1>, <h2> and <h3> elements
CLASS SELECTOR	Matches an element whose class attribute has a value that matches the one specified after the period (or full stop) symbol	.note {} Targets any element whose class attribute has a value of note p.note [] Targets only <p> elements whose class attribute has a value of note
ID SELECTOR	Matches an element whose id attribute has a value that matches the one specified after the pound or hash symbol	#introduction [] Targets the element whose id attribute has a value of introduction
CHILD SELECTOR	Matches an element that is a direct child of another	li>a [] Targets any <a> elements that are children of an element (but not other <a> elements in the page)
DESCENDANT SELECTOR	Matches an element that is a descendent of another specified element (not just a direct child of that element)	p a [] Targets any <a> elements that sit inside a <p> element, even if there are other elements nested between them
ADJACENT SIBLING SELECTOR	Matches an element that is the next sibling of another	h1+p [] Targets the first <p> element after any <h1> element (but not other <p> elements)
GENERAL SIBLING SELECTOR	Matches an element that is a sibling of another, although it does not have to be the directly preceding element	h1~p [] If you had two <p> elements that are siblings of an <h1> element, this rule would apply to both

颜色 (color)

- 如何指定颜色
- 颜色术语和颜色对比
- 背景色

文本 (text / font)

- 文本的大小和字型(font-size / font-family)

PIXELS	PERCENTAGES	EMS
TWELVE PIXEL SCALE		
h1 24px	h1 200%	h1 1.5em
h2 18px	h2 150%	h2 1.3em
h3 14px	h3 117%	h3 1.17em
body 12px	body 75%	body 100%
SIXTEEN PIXEL SCALE		
h1 32px	h1 200%	h1 2em
h2 24px	h2 150%	h2 1.5em
h3 18px	h3 133%	h3 1.125em
body 16px	body 100%	body 100%
p 1em		

SERIF

Serif fonts have extra details on the ends of the main strokes of the letters. These details are known as serifs.

SANS-SERIF

Sans-serif fonts have straight ends to letters, and therefore have a much cleaner design.

MONOSPACE

Every letter in a monospace (or fixed-width) font is the same width. (Non-monospace fonts have different widths.)

im

im

im

In print, serif fonts were traditionally used for long passages of text because they were considered easier to read.

Screens have a lower resolution than print. So, if the text is small, sans-serif fonts can be clearer to read.

Monospace fonts are commonly used for code because they align nicely, making the text easier to follow.

- 粗细、样式、拉伸和装饰(font-weight / font-style / font-stretch / text-decoration)



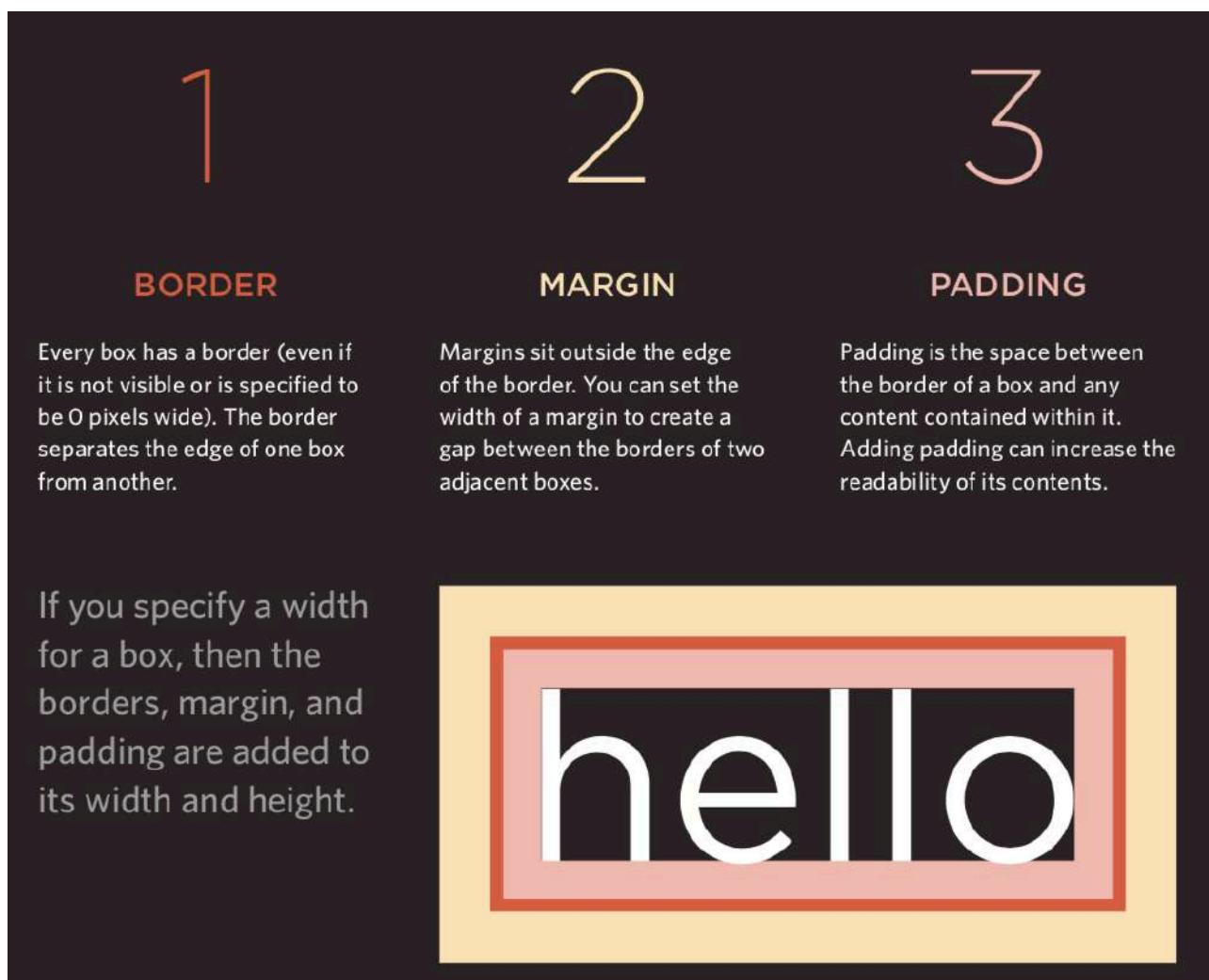
- 行间距(line-height)、字母间距(letter-spacing)和单词间距(word-spacing)
- 对齐(text-align)方式和缩进(text-indent)
- 链接样式 (:link / :visited / :active / :hover)
- CSS3新属性
 - 阴影效果 - text-shadow
 - 首字母和首行文本(:first-letter / :first-line)
 - 响应用户

盒子 (box model)

- 盒子大小的控制 (width / height)

PIXELS	PERCENTAGES	EMS
TWELVE PIXEL SCALE		
h1 24px	h1 200%	h1 1.5em
h2 18px	h2 150%	h2 1.3em
h3 14px	h3 117%	h3 1.17em
body 12px	body 75%	body 100%
SIXTEEN PIXEL SCALE		
h1 32px	h1 200%	h1 2em
h2 24px	h2 150%	h2 1.5em
h3 18px	h3 133%	h3 1.125em
body 16px	body 100%	body 100%
p 1em		

- 盒子的边框、外边距和内边距 (border / margin / padding)



- 盒子的显示和隐藏 (display / visibility)
- CSS3新属性
 - 边框图像 (border-image)

- 投影 (border-shadow)
- 圆角 (border-radius)

列表、表格和表单

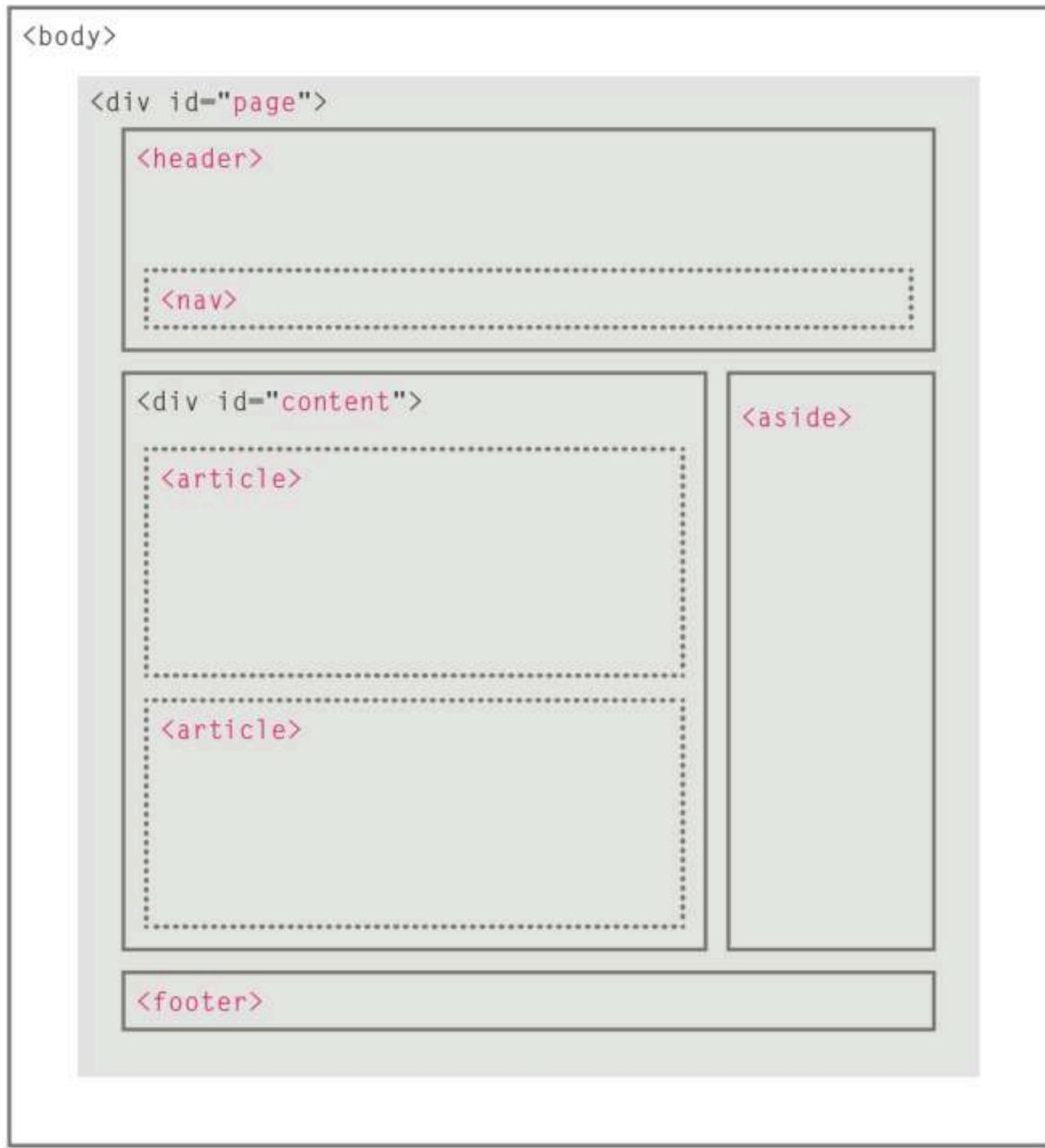
- 列表的项目符号 (list-style)
- 表格的边框和背景 (border-collapse)
- 表单控件的外观
- 表单控件的对齐
- 浏览器的开发者工具

图像

- 控制图像的大小 (display: inline-block)
- 对齐图像
- 背景图像 (background / background-image / background-repeat / background-position)

布局

- 控制元素的位置 (position / z-index)
 - 普通流
 - 相对定位
 - 绝对定位
 - 固定定位
 - 浮动元素 (float / clear)
- 网站布局
 - HTML5布局



- 适配屏幕尺寸

- 固定宽度布局
 - 流体布局
 - 布局网格

使用JavaScript控制行为

JavaScript基本语法

- 语句和注释
- 变量和数据类型
 - 声明和赋值
 - 简单数据类型和复杂数据类型
 - 变量的命名规则
- 表达式和运算符
 - 赋值运算符

- 算术运算符
- 比较运算符
- 逻辑运算符: `&&`、`||`、`!`
- 分支结构
 - `if...else...`
 - `switch...cas...default...`
- 循环结构
 - `for`循环
 - `while`循环
 - `do...while`循环
- 数组
 - 创建数组
 - 操作数组中的元素
- 函数
 - 声明函数
 - 调用函数
 - 参数和返回值
 - 匿名函数
 - 立即调用函数

面向对象

- 对象的概念
- 创建对象的字面量语法
- 访问成员运算符
- 创建对象的构造函数语法
 - `this`关键字
- 添加和删除属性
 - `delete`关键字
- 标准对象
 - `Number / String / Boolean / Symbol / Array / Function`
 - `Date / Error / Math / RegExp / Object / Map / Set`
 - `JSON / Promise / Generator / Reflect / Proxy`

BOM

- `window`对象的属性和方法
- `history`对象
 - `forward() / back() / go()`
- `location`对象
- `navigator`对象
- `screen`对象

DOM

- DOM树
- 访问元素

- `getElementById()` / `querySelector()`
- `getElementsByClassName()` / `getElementsByTagName()` / `querySelectorAll()`
- `parentNode` / `previousSibling` / `nextSibling` / `children` / `firstChild` / `lastChild`
- 操作元素
 - `nodeValue`
 - `innerHTML` / `textContent` / `createElement()` / `createTextNode()` / `appendChild()` / `insertBefore()` / `removeChild()`
 - `className` / `id` / `hasAttribute()` / `getAttribute()` / `setAttribute()` / `removeAttribute()`
- 事件处理
 - 事件类型
 - UI事件: `load` / `unload` / `error` / `resize` / `scroll`
 - 键盘事件: `keydown` / `keyup` / `keypress`
 - 鼠标事件: `click` / `dbclick` / `mousedown` / `mouseup` / `mousemove` / `mouseover` / `mouseout`
 - 焦点事件: `focus` / `blur`
 - 表单事件: `input` / `change` / `submit` / `reset` / `cut` / `copy` / `paste` / `select`
 - 事件绑定
 - HTML事件处理程序 (不推荐使用, 因为要做到标签与代码分离)
 - 传统的DOM事件处理程序 (只能附加一个回调函数)
 - 事件监听器 (旧的浏览器中不被支持)
 - 事件流: 事件捕获 / 事件冒泡
 - 事件对象 (低版本IE中的`window.event`)
 - `target` (有些浏览器使用`srcElement`)
 - `type`
 - `cancelable`
 - `preventDefault()`
 - `stopPropagation()` (低版本IE中的`cancelBubble`)
 - 鼠标事件 - 事件发生的位置
 - 屏幕位置: `screenX`和`screenY`
 - 页面位置: `pageX`和`pageY`
 - 客户端位置: `clientX`和`clientY`
 - 键盘事件 - 哪个键被按下了
 - `keyCode`属性 (有些浏览器使用`which`)
 - `String.fromCharCode(event.keyCode)`
 - HTML5事件
 - `DOMContentLoaded`
 - `hashchange`
 - `beforeunload`

JavaScript API

- 客户端存储 - `localStorage`和`sessionStorage`

```
localStorage.colorSetting = '#a4509b';
localStorage['colorSetting'] = '#a4509b';
```

```
localStorage.setItem('colorSetting', '#a4509b');
```

- 获取位置信息 - `geolocation`

```
navigator.geolocation.getCurrentPosition(function(pos) {
  console.log(pos.coords.latitude)
  console.log(pos.coords.longitude)
})
```

- 从服务器获取数据 - Fetch API
- 绘制图形 - `<canvas>`的API
- 音视频 - `<audio>`和`<video>`的API

使用jQuery

jQuery概述

1. Write Less Do More (用更少的代码来完成更多的工作)
2. 使用CSS选择器来查找元素 (更简单更方便)
3. 使用jQuery方法来操作元素 (解决浏览器兼容性问题、应用于所有元素并施加多个方法)

引入jQuery

- 下载jQuery的开发版和压缩版
- 从CDN加载jQuery

```
<script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>
<script>
  window.jQuery ||
  document.write('<script src="js/jquery-3.3.1.min.js"></script>')
</script>
```

查找元素

- 选择器
 - * / element / #id / .class / selector1, selector2
 - ancestor descendant / parent>child / previous+next / previous~siblings
- 筛选器
 - 基本筛选器: :not(selector) / :first / :last / :even / :odd / :eq(index) / :gt(index) / :lt(index) / :animated / :focus
 - 内容筛选器: :contains('...') / :empty / :parent / :has(selector)
 - 可见性筛选器: :hidden / :visible
 - 子节点筛选器: :nth-child(expr) / :first-child / :last-child / :only-child

- 属性筛选器: [attribute] / [attribute='value'] / [attribute!= 'value'] / [attribute^='value'] / [attribute\$='value'] / [attribute|= 'value'] / [attribute~='value']
- 表单: :input / :text / :password / :radio / :checkbox / :submit / :image / :reset / :button / :file / :selected / :enabled / :disabled / :checked

执行操作

- 内容操作
 - 获取/修改内容: `html()` / `text()` / `replaceWith()` / `remove()`
 - 获取/设置元素: `before()` / `after()` / `prepend()` / `append()` / `remove()` / `clone()` / `unwrap()` / `detach()` / `empty()` / `add()`
 - 获取/修改属性: `attr()` / `removeAttr()` / `addClass()` / `removeClass()` / `css()`
 - 获取/设置表单值: `val()`
- 查找操作
 - 查找方法: `find()` / `parent()` / `children()` / `siblings()` / `next()` / `nextAll()` / `prev()` / `prevAll()`
 - 筛选器: `filter()` / `not()` / `has()` / `is()` / `contains()`
 - 索引编号: `eq()`
- 尺寸和位置
 - 尺寸相关: `height()` / `width()` / `innerHeight()` / `innerWidth()` / `outerWidth()` / `outerHeight()`
 - 位置相关: `offset()` / `position()` / `scrollLeft()` / `scrollTop()`
- 特效和动画
 - 基本动画: `show()` / `hide()` / `toggle()`
 - 消失出现: `fadeIn()` / `fadeOut()` / `fadeTo()` / `fadeToggle()`
 - 滑动效果: `slideDown()` / `slideUp()` / `slideToggle()`
 - 自定义: `delay()` / `stop()` / `animate()`
- 事件
 - 文档加载: `ready()` / `load()`
 - 用户交互: `on()` / `off()`

链式操作

检测页面是否可用

```
<script>
$(document).ready(function() {
  });
</script>
```

```
<script>
$(function() {
```

```
});  
</script>
```

jQuery插件

- jQuery Validation
- jQuery Treeview
- jQuery Autocomplete
- jQuery UI

避免和其他库的冲突

先引入其他库再引入jQuery的情况。

```
<script src="other.js"></script>  
<script src="jquery.js"></script>  
<script>  
    jQuery.noConflict();  
    jQuery(function() {  
        jQuery('div').hide();  
    });  
</script>
```

先引入jQuery再引入其他库的情况。

```
<script src="jquery.js"></script>  
<script src="other.js"></script>  
<script>  
    jQuery(function() {  
        jQuery('div').hide();  
    });  
</script>
```

使用Ajax

Ajax是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术。

- 原生的Ajax
- 基于jQuery的Ajax
 - 加载内容
 - 提交表单

前端框架

渐进式框架 - [Vue.js](#)

前后端分离开发（前端渲染）必选框架。

快速上手

1. 引入Vue的JavaScript文件，我们仍然推荐从CDN服务器加载它。

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

2. 数据绑定（声明式渲染）。

```
<div id="app">
  <h1>{{ product }}库存信息</h1>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      product: 'iPhone X'
    }
  });
</script>
```

3. 条件与循环。

```
<div id="app">
  <h1>库存信息</h1>
  <hr>
  <ul>
    <li v-for="product in products">
      {{ product.name }} - {{ product.quantity }}
      <span v-if="product.quantity === 0">
        已经售罄
      </span>
    </li>
  </ul>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      products: [
        {"id": 1, "name": "iPhone X", "quantity": 20},
        {"id": 2, "name": "华为 Mate20", "quantity": 0},
        {"id": 3, "name": "小米 Mix3", "quantity": 50}
      ]
    }
  });
</script>
```

```
        ]
    }
});  
</script>
```

4. 计算属性。

```
<div id="app">
  <h1>库存信息</h1>
  <hr>
  <ul>
    <li v-for="product in products">
      {{ product.name }} - {{ product.quantity }}
      <span v-if="product.quantity === 0">
        已经售罄
      </span>
    </li>
  </ul>
  <h2>库存总量: {{ totalQuantity }}台</h2>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      products: [
        {"id": 1, "name": "iPhone X", "quantity": 20},
        {"id": 2, "name": "华为 Mate20", "quantity": 0},
        {"id": 3, "name": "小米 Mix3", "quantity": 50}
      ]
    },
    computed: {
      totalQuantity() {
        return this.products.reduce((sum, product) => {
          return sum + product.quantity
        }, 0);
      }
    }
  });
</script>
```

5. 处理事件。

```
<div id="app">
  <h1>库存信息</h1>
  <hr>
  <ul>
    <li v-for="product in products">
```

```

        {{ product.name }} - {{ product.quantity }}
        <span v-if="product.quantity === 0">
            已经售罄
        </span>
        <button @click="product.quantity += 1">
            增加库存
        </button>
    </li>
</ul>
<h2>库存总量: {{ totalQuantity }}台</h2>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            products: [
                {"id": 1, "name": "iPhone X", "quantity": 20},
                {"id": 2, "name": "华为 Mate20", "quantity": 0},
                {"id": 3, "name": "小米 Mix3", "quantity": 50}
            ],
        },
        computed: {
            totalQuantity() {
                return this.products.reduce((sum, product) => {
                    return sum + product.quantity
                }, 0);
            }
        }
    });
</script>

```

6. 用户输入。

```

<div id="app">
    <h1>库存信息</h1>
    <hr>
    <ul>
        <li v-for="product in products">
            {{ product.name }} -
            <input type="number" v-model.number="product.quantity" min="0">
            <span v-if="product.quantity === 0">
                已经售罄
            </span>
            <button @click="product.quantity += 1">
                增加库存
            </button>
        </li>
    </ul>
    <h2>库存总量: {{ totalQuantity }}台</h2>
</div>

```

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      products: [
        {"id": 1, "name": "iPhone X", "quantity": 20},
        {"id": 2, "name": "华为 Mate20", "quantity": 0},
        {"id": 3, "name": "小米 Mix3", "quantity": 50}
      ]
    },
    computed: {
      totalQuantity() {
        return this.products.reduce((sum, product) => {
          return sum + product.quantity
        }, 0);
      }
    }
  });
</script>
```

7. 通过网络加载JSON数据。

```
<div id="app">
  <h2>库存信息</h2>
  <ul>
    <li v-for="product in products">
      {{ product.name }} - {{ product.quantity }}
      <span v-if="product.quantity === 0">
        已经售罄
      </span>
    </li>
  </ul>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue"></script>
<script>
  const app = new Vue({
    el: '#app',
    data: {
      products: []
    },
    created() {
      fetch('https://jackfrued.top/api/products')
        .then(response => response.json())
        .then(json => {
          this.products = json
        });
    }
  });
</script>
```

```
});  
</script>
```

使用脚手架 - vue-cli

Vue为商业项目开发提供了非常便捷的脚手架工具vue-cli，通过工具可以省去手工配置开发环境、测试环境和运行环境的步骤，让开发者只需要关注要解决的问题。

1. 安装脚手架。
2. 创建项目。
3. 安装依赖包。
4. 运行项目。

UI框架 - Element

基于Vue 2.0的桌面端组件库，用于构造用户界面，支持响应式布局。

1. 引入Element的CSS和JavaScript文件。

```
<!-- 引入样式 -->  
<link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css">  
<!-- 引入组件库 -->  
<script src="https://unpkg.com/element-ui/lib/index.js"></script>
```

2. 一个简单的例子。

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css">  
</head>  
<body>  
<div id="app">  
<el-button @click="visible = true">点我</el-button>  
<el-dialog :visible.sync="visible" title="Hello world">  
<p>开始使用Element吧</p>  
</el-dialog>  
</div>  
</body>  
<script src="https://unpkg.com/vue/dist/vue.js"></script>  
<script src="https://unpkg.com/element-ui/lib/index.js"></script>  
<script>  
new Vue({  
  el: '#app',  
  data: {
```

```
        visible: false,
    })
})
</script>
</html>
```

3. 使用组件。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css">
    </head>
    <body>
        <div id="app">
            <el-table :data="tableData" stripe style="width: 100%">
                <el-table-column prop="date" label="日期" width="180">
                </el-table-column>
                <el-table-column prop="name" label="姓名" width="180">
                </el-table-column>
                <el-table-column prop="address" label="地址">
                </el-table-column>
            </el-table>
        </div>
    </body>
    <script src="https://unpkg.com/vue/dist/vue.js"></script>
    <script src="https://unpkg.com/element-ui/lib/index.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                tableData: [
                    {
                        date: '2016-05-02',
                        name: '王一霸',
                        address: '上海市普陀区金沙江路 1518 弄'
                    },
                    {
                        date: '2016-05-04',
                        name: '刘二狗',
                        address: '上海市普陀区金沙江路 1517 弄'
                    },
                    {
                        date: '2016-05-01',
                        name: '杨三萌',
                        address: '上海市普陀区金沙江路 1519 弄'
                    },
                    {
                        date: '2016-05-03',
                        name: '陈四吹',
                        address: '上海市普陀区金沙江路 1520 弄'
                    }
                ]
            }
        })
    </script>

```

```

        address: '上海市普陀区金沙江路 1516 弄'
    }
]
}
})
</script>
</html>

```

报表框架 - ECharts

百度出品的开源可视化库，常用于生成各种类型的报表。



基于弹性盒子的CSS框架 - Bulma

Bulma是一个基于Flexbox的现代化的CSS框架，其初衷就是移动优先 (Mobile First)，模块化设计，可以轻松用来实现各种简单或者复杂的内容布局，即使不懂CSS的开发者也能够使用它定制出漂亮的页面。

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Bulma</title>
    <link href="https://cdn.bootcss.com/bulma/0.7.4/css/bulma.min.css" rel="stylesheet">
    <style type="text/css">
        div { margin-top: 10px; }
        .column { color: #fff; background-color: #063; margin: 10px 10px; text-align: center; }
    </style>
</head>
<body>
    <div class="columns">

```

```
<div class="column">1</div>
<div class="column">2</div>
<div class="column">3</div>
<div class="column">4</div>
</div>
<div>
  <a class="button is-primary">Primary</a>
  <a class="button is-link">Link</a>
  <a class="button is-info">Info</a>
  <a class="button is-success">Success</a>
  <a class="button is-warning">Warning</a>
  <a class="button is-danger">Danger</a>
</div>
<div>
  <progress class="progress is-danger is-medium" max="100">60%</progress>
</div>
<div>
  <table class="table is-hoverable">
    <tr>
      <th>One</th>
      <th>Two</th>
    </tr>
    <tr>
      <td>Three</td>
      <td>Four</td>
    </tr>
    <tr>
      <td>Five</td>
      <td>Six</td>
    </tr>
    <tr>
      <td>Seven</td>
      <td>Eight</td>
    </tr>
    <tr>
      <td>Nine</td>
      <td>Ten</td>
    </tr>
    <tr>
      <td>Eleven</td>
      <td>Twelve</td>
    </tr>
  </table>
</div>
</body>
</html>
```

响应式布局框架 - Bootstrap

用于快速开发Web应用程序的前端框架，支持响应式布局。

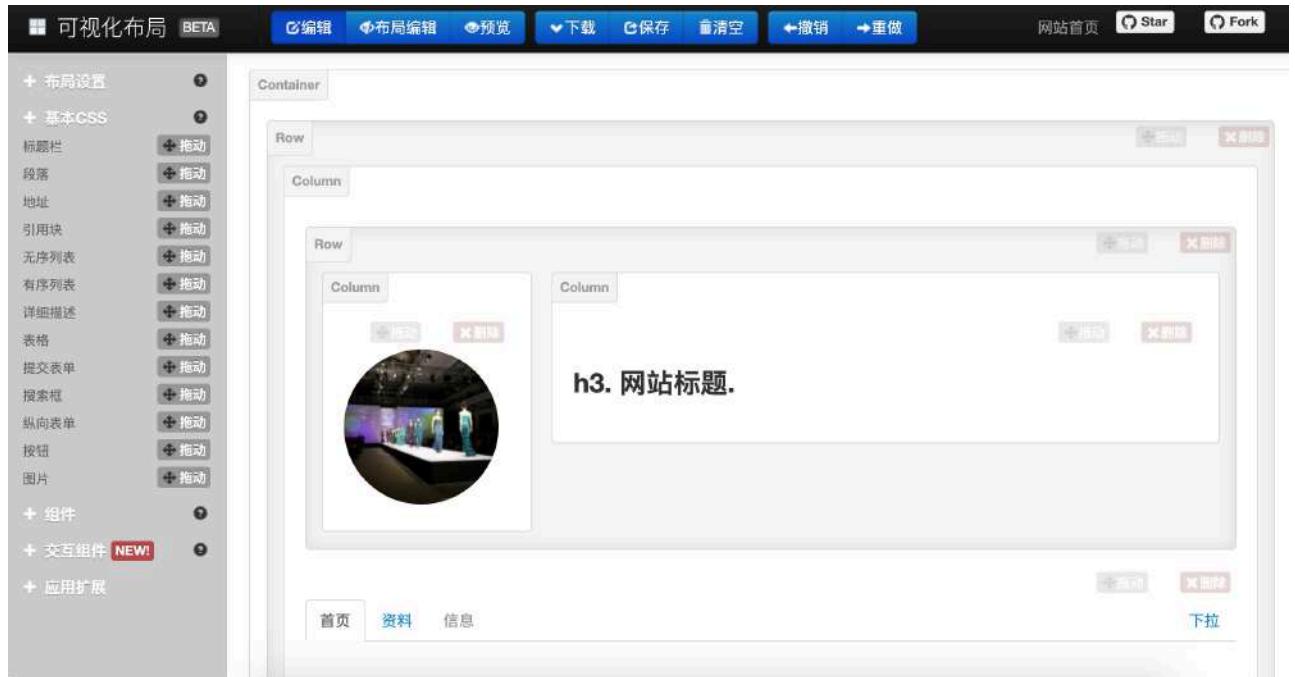
1. 特点

- 支持主流的浏览器和移动设备
- 容易上手
- 响应式设计

2. 内容

- 网格系统
- 封装的CSS
- 现成的组件
- JavaScript插件

3. 可视化



玩转Linux操作系统

说明：本文中对Linux命令的讲解都是基于名为CentOS的Linux发行版本，我自己使用的是阿里云服务器，系统版本为CentOS Linux release 7.6.1810。不同的Linux发行版本在Shell命令和工具程序上会有一些差别，但是这些差别是很小的。

操作系统发展史

只有硬件没有软件的计算机系统被称之为“裸机”，我们很难用“裸机”来完成计算机日常的工作（如存储和运算），所以必须用特定的软件来控制硬件的工作。最靠近计算机硬件的软件是系统软件，其中最为重要的就是“操作系统”。“操作系统”是控制和管理整个计算机硬件和软件资源、实现资源分配和任务调配、为系统用户以及其他软件提供接口和环境的程序的集合。

没有操作系统（手工操作）

在计算机诞生之初没有操作系统的年代，人们先把程序纸带（或卡片）装上计算机，然后启动输入机把程序送入计算机，接着通过控制台开关启动程序运行。当程序执行完毕，打印机输出计算的结果，用户卸下并取走纸带（或卡片）。第二个用户上机，重复同样的步骤。在整个过程中用户独占机器，CPU等待手工操作，资源利用率极低。

批处理系统

首先启动计算机上的一个监督程序，在监督程序的控制下，计算机能够自动的、成批的处理一个或多个用户的作业。完成一批作业后，监督程序又从输入机读取作业存入磁带机。按照上面的步骤重复处理任务。监督程序不停的处理各个作业，实现了作业的自动转接，减少了作业的建立时间和手工操作时间，提高了计算机资源的利用率。批处理系统又可以分为单道批处理系统、多道批处理系统、联机批处理系统、脱机批处理系统。

分时系统和实时系统

分时系统是把处理器的运行时间分成很短的时间片，按时间片轮流把处理器分配给各联机作业使用。若某个作业在分配给它的时间片内不能完成其计算，则该作业暂时中断，把处理器让给另一作业使用，等待下一轮调度时再继续其运行。由于计算机速度很快，作业运行轮转得很快，给每个用户的感觉是他独占了一台计算机。而每个用户可以通过自己的终端向系统发出各种操作控制命令，在充分的人机交互情况下，完成作业的运行。为了解决分时系统不能及时响应用户指令的情况，又出现了能够在严格的时间范围内完成事件处理，及时响应随机外部事件的实时系统。

通用操作系统

1. 1960s: IBM的System/360系列的机器有了统一的操作系统OS/360。
2. 1965年: AT&T的贝尔实验室加入GE和MIT的合作计划开始开发MULTICS。
3. 1969年: MULTICS项目失败，Ken Thompson赋闲在家，为了玩“Space Travel”游戏用汇编语言在当时已经被淘汰的PDP-7上开发了Unics。

注：很难想象，Unix这么伟大的操作系统，居然是一个赋闲在家的程序员（关键是老婆回娘家还带上了孩子）在一台被淘汰的设备上为了玩游戏开发出来的。

4. 1970年~1971年：Ken Thompson和Dennis Ritchie用B语言在PDP-11上重写了Unics，并在Brian Kernighan的建议下将其更名为Unix。



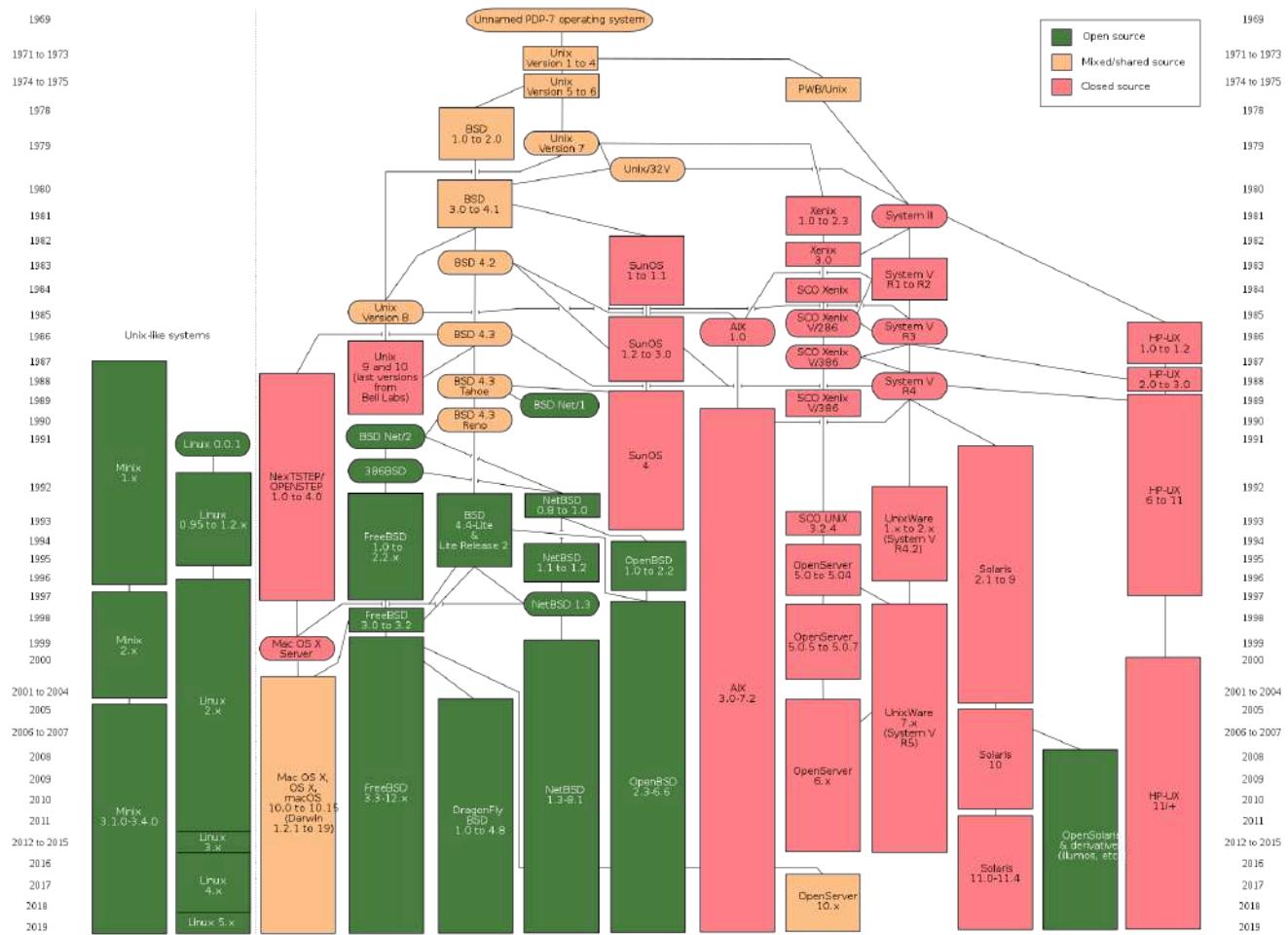
5. 1972年~1973年：Dennis Ritchie发明了C语言来取代可移植性较差的B语言，并开启了用C语言重写Unix的工作。
6. 1974年：Unix推出了里程碑意义的第5版，几乎完全用C语言来实现。
7. 1979年：从Unix第7版开始，AT&T发布新的使用条款，将Unix私有化。
8. 1987年：Andrew S. Tanenbaum教授为了能在课堂上为学生讲解操作系统运作的细节，决定在不使用任何AT&T的源代码前提下，自行开发与Unix兼容的操作系统以避免版权上的争议，该系统被命名为Minix。



9. 1991年：Linus Torvalds就读于芬兰赫尔辛基大学期间，尝试在Minix上做一些开发工作，但因为Minix只是作为教学用途的操作系统，功能并不强大，为了方便在学校的新闻组和邮件系统中读写和下载文件，Linus编写了磁盘驱动程序和文件系统，这些东西形成了Linux系统内核的雏形。



下图是Unix操作系统家族的图谱。



Linux概述

Linux是一个通用操作系统。一个操作系统要负责任务调度、内存分配、处理外围设备I/O等操作。操作系统通常由内核（运行其他程序，管理像磁盘、打印机等硬件设备的核心程序）和系统程序（设备驱动、底层库、shell、服务程序等）两部分组成。

Linux内核是芬兰人Linus Torvalds开发的，于1991年9月发布。而Linux操作系统作为Internet时代的产物，它是由全世界许多开发者共同合作开发的，是一个自由的操作系统（注意自由和免费并不是同一个概念，想了解二者的差别可以[点击这里](#)）。

Linux系统优点

1. 通用操作系统，不跟特定的硬件绑定。
2. 用C语言编写，可移植性强，有内核编程接口。
3. 支持多用户和多任务，支持安全的分层文件系统。
4. 大量的实用程序，完善的网络功能以及强大的支持文档。
5. 可靠的安全性和良好的稳定性，对开发者更友好。

Linux系统发行版本

1. [Redhat](#)
2. [Ubuntu](#)
3. [CentOS](#)
4. [Fedora](#)
5. [Debian](#)

6. openSUSE

基础命令

Linux系统的命令通常都是如下所示的格式：

命令名称 [命名参数] [命令对象]

1. 获取登录信息 - **w / who / last/ lastb**。

```
[root ~]# w
23:31:16 up 12:16, 2 users, load average: 0.00, 0.01, 0.05
USER     TTY      FROM          LOGIN@    IDLE     JCPU    PCPU WHAT
root     pts/0    182.139.66.250  23:03    4.00s  0.02s  0.00s w
jackfrue pts/1    182.139.66.250  23:26    3:56   0.00s  0.00s -bash
[root ~]# who
root     pts/0      2018-04-12 23:03 (182.139.66.250)
jackfrued pts/1      2018-04-12 23:26 (182.139.66.250)
[root ~]# who am i
root     pts/0      2018-04-12 23:03 (182.139.66.250)
[root ~]# who mom likes
root     pts/0      2018-04-12 23:03 (182.139.66.250)
[root ~]# last
root     pts/0      117.136.63.184  Sun May 26 18:57  still logged in
reboot  system boot  3.10.0-957.10.1. Mon May 27 02:52 - 19:10 (-7:-42)
root     pts/4      117.136.63.184  Sun May 26 18:51 - crash (08:01)
root     pts/4      117.136.63.184  Sun May 26 18:49 - 18:49 (00:00)
root     pts/3      117.136.63.183  Sun May 26 18:35 - crash (08:17)
root     pts/2      117.136.63.183  Sun May 26 18:34 - crash (08:17)
root     pts/0      117.136.63.183  Sun May 26 18:10 - crash (08:42)
```

2. 查看自己使用的Shell - **ps**。

Shell也被称为“壳”或“壳程序”，它是用户与操作系统内核交流的翻译官，简单的说就是人与计算机交互的界面和接口。目前很多Linux系统默认的Shell都是bash (Bourne Again SHell)，因为它可以使用tab键进行命令和路径补全、可以保存历史命令、可以方便的配置环境变量以及执行批处理操作。

```
[root ~]# ps
 PID TTY          TIME CMD
 3531 pts/0    00:00:00 bash
 3553 pts/0    00:00:00 ps
```

3. 查看命令的说明和位置 - **whatis / which / whereis**。

```
[root ~]# whatis ps
ps (1)      - report a snapshot of the current processes.
```

```
[root ~]# whatis python
python (1)      - an interpreted, interactive, object-oriented programming
language
[root ~]# whereis ps
ps: /usr/bin/ps /usr/share/man/man1/ps.1.gz
[root ~]# whereis python
python: /usr/bin/python /usr/bin/python2.7 /usr/lib/python2.7
/usr/lib64/python2.7 /etc/python /usr/include/python2.7
/usr/share/man/man1/python.1.gz
[root ~]# which ps
/usr/bin/ps
[root ~]# which python
/usr/bin/python
```

4. 清除屏幕上显示的内容 - **clear**。

5. 查看帮助文档 - **man** / **info** / **--help** / **apropos**。

```
[root@izwz97tbgo91kabnat2lo8z ~]# ps --help
Usage:
  ps [options]
Try 'ps --help <simple|list|output|threads|misc|all>'
  or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
[root@izwz97tbgo91kabnat2lo8z ~]# man ps
PS(1)                                     User Commands
PS(1)
NAME
  ps - report a snapshot of the current processes.
SYNOPSIS
  ps [options]
DESCRIPTION
```

6. 查看系统和主机名 - `uname` / `hostname`

```
[root@izwz97tbgo91kabnat2lo8z ~]# uname
Linux
[root@izwz97tbgo91kabnat2lo8z ~]# hostname
izwz97tbgo91kabnat2lo8z
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat /etc/centos-release
CentOS Linux release 7.6.1810 (Core)
```

说明：`cat`是连接文件内容并打印到标准输出的命令，后面会讲到该命令；`/etc`是Linux系统上的一个非常重要的目录，它保存了很多的配置文件；`centos-release`是该目录下的一个文件，因为我自己使用的Linux发行版本是CentOS 7.6，因此这里会有一个这样的文件。

7. 时间和日期 - **date / cal**。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# date
Wed Jun 20 12:53:19 CST 2018
[root@iZwz97tbgo91kabnat2lo8Z ~]# cal
      June 2018
Su Mo Tu We Th Fr Sa
      1  2
3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
[root@iZwz97tbgo91kabnat2lo8Z ~]# cal 5 2017
      May 2017
Su Mo Tu We Th Fr Sa
      1  2  3  4  5  6
7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

8. 重启和关机 - **reboot / shutdown**。

```
[root ~]# shutdown -h +5
Shutdown scheduled for Sun 2019-05-26 19:34:27 CST, use 'shutdown -c' to
cancel.
[root ~]#
Broadcast message from root (Sun 2019-05-26 19:29:27 CST):

The system is going down for power-off at Sun 2019-05-26 19:34:27 CST!
[root ~]# shutdown -c

Broadcast message from root (Sun 2019-05-26 19:30:22 CST):

The system shutdown has been cancelled at Sun 2019-05-26 19:31:22 CST!
[root ~]# shutdown -r 23:58
Shutdown scheduled for Sun 2019-05-26 23:58:00 CST, use 'shutdown -c' to
cancel.
[root ~]# shutdown -c

Broadcast message from root (Sun 2019-05-26 19:31:06 CST):

The system shutdown has been cancelled at Sun 2019-05-26 19:32:06 CST!
```

说明：在执行**shutdown**命令时会向登录系统的用户发出警告，可以在命令后面跟上警告消息来替换默认的警告消息，也可以在**-h**参数后通过**now**来表示立刻关机。

9. 退出登录 - **exit / logout**。

10. 查看历史命令 - **history**。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# history
...
452  ls
453  cd Python-3.6.5/
454  clear
455  history
[root@iZwz97tbgo91kabnat2lo8Z ~]# !454
```

说明：查看到历史命令之后，可以用!**历史命令编号**来重新执行该命令；通过**history -c**可以清除历史命令。

实用程序

文件和文件夹操作

1. 创建/删除空目录 - **mkdir** / **rmdir**。

```
[root ~]# mkdir abc
[root ~]# mkdir -p xyz/abc
[root ~]# rmdir abc
```

2. 创建/删除文件 - **touch** / **rm**。

```
[root ~]# touch readme.txt
[root ~]# touch error.txt
[root ~]# rm error.txt
rm: remove regular empty file 'error.txt'? y
[root ~]# rm -rf xyz
```

- **touch**命令用于创建空白文件或修改文件时间。在Linux系统中一个文件有三种时间：
 - 更改内容的时间 - **mtime**。
 - 更改权限的时间 - **ctime**。
 - 最后访问时间 - **atime**。
- **rm**的几个重要参数：
 - **-i**：交互式删除，每个删除项都会进行询问。
 - **-r**：删除目录并递归的删除目录中的文件和目录。
 - **-f**：强制删除，忽略不存在的文件，没有任何提示。

3. 切换和查看当前工作目录 - **cd** / **pwd**。

说明：**cd**命令后面可以跟相对路径（以当前路径作为参照）或绝对路径（以/开头）来切换到指定的目录，也可以用**cd ..**来返回上一级目录。请大家想一想，如果要返回到上上一级目录应该给**cd**命令加上什么样的参数呢？

4. 查看目录内容 - **ls**。

- **-l**: 以长格式查看文件和目录。
- **-a**: 显示以点开头的文件和目录（隐藏文件）。
- **-R**: 遇到目录要进行递归展开（继续列出目录下面的文件和目录）。
- **-d**: 只列出目录，不列出其他内容。
- **-S / -t**: 按大小/时间排序。

5. 查看文件内容 - **cat / tac / head / tail / more / less / rev / od**。

```
[root ~]# wget http://www.sohu.com/ -O sohu.html
--2018-06-20 18:42:34--  http://www.sohu.com/
Resolving www.sohu.com (www.sohu.com)... 14.18.240.6
Connecting to www.sohu.com (www.sohu.com)|14.18.240.6|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 212527 (208K) [text/html]
Saving to: 'sohu.html'

100%[=====] 212,527      --.-.
K/s  in 0.03s
2018-06-20 18:42:34 (7.48 MB/s) - 'sohu.html' saved [212527/212527]
[root ~]# cat sohu.html
...
[root ~]# head -10 sohu.html
<!DOCTYPE html>
<html>
<head>
<title>搜狐</title>
<meta name="Keywords" content="搜狐,门户网站,新媒体,网络媒体,新闻,财经,体育,娱乐,时尚,汽车,房产,科技,图片,论坛,微博,博客,视频,电影,电视剧"/>
<meta name="Description" content="搜狐网为用户提供24小时不间断的最新资讯,及搜索、邮件等网络服务。内容包括全球热点事件、突发新闻、时事评论、热播影视剧、体育赛事、行业动态、生活服务信息,以及论坛、博客、微博、我的搜狐等互动空间。" />
<meta name="shenma-site-verification" content="1237e4d02a3d8d73e96cbd97b699e9c3_1504254750">
<meta charset="utf-8"/>
<meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1"/>
[root ~]# tail -2 sohu.html
</body>
</html>
[root ~]# less sohu.html
...
[root ~]# cat -n sohu.html | more
...
```

说明：上面用到了一个名为**wget**的命令，它是一个网络下载器程序，可以从指定的URL下载资源。

6. 拷贝/移动文件 - **cp / mv**。

```
[root ~]# mkdir backup
[root ~]# cp sohu.html backup/
[root ~]# cd backup
[root backup]# ls
sohu.html
[root backup]# mv sohu.html sohu_index.html
[root backup]# ls
sohu_index.html
```

7. 文件重命名 - **rename**.

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# rename .htm .html *.htm
```

8. 查找文件和查找内容 - **find / grep**.

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# find / -name "*.html"
/root/sohu.html
/root/backup/sohu_index.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# find . -atime 7 -type f -print
[root@iZwz97tbgo91kabnat2lo8Z ~]# find . -type f -size +2k
[root@iZwz97tbgo91kabnat2lo8Z ~]# find . -type f -name "*.swp" -delete
[root@iZwz97tbgo91kabnat2lo8Z ~]# grep "<script>" sohu.html -n
20:<script>
[root@iZwz97tbgo91kabnat2lo8Z ~]# grep -E \<>/?script.*> sohu.html -n
20:<script>
22:</script>
24:<script src="//statics.itc.cn/web/v3/static/js/es5-shim-08e41cf3e.min.js"></script>
25:<script src="//statics.itc.cn/web/v3/static/js/es5-sham-1d5fa1124b.min.js"></script>
26:<script src="//statics.itc.cn/web/v3/static/js/html5shiv-21fc8c2ba6.js">
</script>
29:<script type="text/javascript">
52:</script>
...
...
```

说明: **grep**在搜索字符串时可以使用正则表达式, 如果需要使用正则表达式可以用**grep -E**或者直接使用**egrep**。

9. 创建链接和查看链接 - **ln / readlink**.

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sohu.html
-rw-r--r-- 1 root root 212131 Jun 20 19:15 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ln /root/sohu.html
/root/backup/sohu_backup
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sohu.html
```

```
-rw-r--r-- 2 root root 212131 Jun 20 19:15 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ln /root/sohu.html
/root/backup/sohu_backup2
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sohu.html
-rw-r--r-- 3 root root 212131 Jun 20 19:15 sohu.html
[root@iZwz97tbgo91kabnat2lo8Z ~]# ln -s /etc/centos-release sysinfo
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls -l sysinfo
lrwxrwxrwx 1 root root 19 Jun 20 19:21 sysinfo -> /etc/centos-release
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat sysinfo
CentOS Linux release 7.4.1708 (Core)
[root@iZwz97tbgo91kabnat2lo8Z ~]# cat /etc/centos-release
CentOS Linux release 7.4.1708 (Core)
```

说明：链接可以分为硬链接和软链接（符号链接）。硬链接可以认为是一个指向文件数据的指针，就像Python中对象的引用计数，每添加一个硬链接，文件的对应链接数就增加1，只有当文件的链接数为0时，文件所对应的存储空间才有可能被其他文件覆盖。我们平常删除文件时其实并没有删除硬盘上的数据，我们删除的只是一个指针，或者说是数据的一条使用记录，所以类似于“文件粉碎机”之类的软件在“粉碎”文件时除了删除文件指针，还会在文件对应的存储区域填入数据来保证文件无法再恢复。软链接类似于Windows系统下的快捷方式，当软链接链接的文件被删除时，软链接也就失效了。

10. 压缩/解压缩和归档/解归档 - **gzip / gunzip / xz**。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# wget http://download.redis.io/releases/redis-4.0.10.tar.gz
--2018-06-20 19:29:59-- http://download.redis.io/releases/redis-4.0.10.tar.gz
Resolving download.redis.io (download.redis.io)... 109.74.203.151
Connecting to download.redis.io (download.redis.io)|109.74.203.151|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 1738465 (1.7M) [application/x-gzip]
Saving to: 'redis-4.0.10.tar.gz'

100%[=====] 1,738,465 70.1KB/s
in 74s

2018-06-20 19:31:14 (22.9 KB/s) - 'redis-4.0.10.tar.gz' saved [1738465/1738465]
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls redis*
redis-4.0.10.tar.gz
[root@iZwz97tbgo91kabnat2lo8Z ~]# gunzip redis-4.0.10.tar.gz
[root@iZwz97tbgo91kabnat2lo8Z ~]# ls redis*
redis-4.0.10.tar
```

11. 归档和解归档 - **tar**。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# tar -xvf redis-4.0.10.tar
redis-4.0.10/
redis-4.0.10/.gitignore
redis-4.0.10/00-RELEASENOTES
redis-4.0.10/BUGS
redis-4.0.10/CONTRIBUTING
```

```
redis-4.0.10/COPYING
redis-4.0.10/INSTALL
redis-4.0.10/MANIFESTO
redis-4.0.10/Makefile
redis-4.0.10/README.md
redis-4.0.10/deps/
redis-4.0.10/deps/Makefile
redis-4.0.10/deps/README.md
...
```

说明：归档（也称为创建归档）和解归档都使用`tar`命令，通常创建归档需要`-cvf`三个参数，其中`c`表示创建（create），`v`表示显示创建归档详情（verbose），`f`表示指定归档的文件（file）；解归档需要加上`-xvf`参数，其中`x`表示抽取（extract），其他两个参数跟创建归档相同。

12. 将标准输入转成命令行参数 - `xargs`。

下面的命令会将查找当前路径下的html文件，然后通过`xargs`将这些文件作为参数传给`rm`命令，实现查找并删除文件的操作。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# find . -type f -name "*.html" | xargs rm -f
```

下面的命令将a.txt文件中的多行内容变成一行输出到b.txt文件中，其中`<`表示从a.txt中读取输入，`>`表示将命令的执行结果输出到b.txt中。

```
[root@iZwz97tbgo91kabnat2lo8Z ~]# xargs < a.txt > b.txt
```

说明：这个命令就像上面演示的那样常在管道（实现进程间通信的一种方式）和重定向（重新指定输入输出的位置）操作中用到，后面的内容中会讲到管道操作和输入输出重定向操作。

13. 显示文件或目录 - `basename` / `dirname`。

14. 其他相关工具。

- `sort` - 对内容排序
- `uniq` - 去掉相邻重复内容
- `tr` - 替换指定内容为新内容
- `cut` / `paste` - 剪切/黏贴内容
- `split` - 拆分文件
- `file` - 判断文件类型
- `wc` - 统计文件行数、单词数、字节数
- `iconv` - 编码转换

```
[root ~]# cat foo.txt
grape
apple
pitaya
```

```
[root ~]# cat bar.txt
100
200
300
400
[root ~]# paste foo.txt bar.txt
grape    100
apple    200
pitaya   300
        400
[root ~]# paste foo.txt bar.txt > hello.txt
[root ~]# cut -b 4-8 hello.txt
pe      10
le      20
aya    3
0
[root ~]# cat hello.txt | tr '\t' ','
grape,100
apple,200
pitaya,300
,400
[root ~]# split -l 100 sohu.html hello
[root ~]# wget https://www.baidu.com/img/bd_logo1.png
[root ~]# file bd_logo1.png
bd_logo1.png: PNG image data, 540 x 258, 8-bit colormap, non-interlaced
[root ~]# wc sohu.html
2979 6355 212527 sohu.html
[root ~]# wc -l sohu.html
2979 sohu.html
[root ~]# wget http://www.qq.com -O qq.html
[root ~]# iconv -f gb2312 -t utf-8 qq.html
```

管道和重定向

1. 管道的使用 - |。

例子：查找当前目录下文件个数。

```
[root ~]# find ./ | wc -l
6152
```

例子：列出当前路径下的文件和文件夹，给每一项加一个编号。

```
[root ~]# ls | cat -n
1  dump.rdb
2  mongodb-3.6.5
3  Python-3.6.5
```

```
4 redis-3.2.11
5 redis.conf
```

例子：查找record.log中包含AAA，但不包含BBB的记录的总数

```
[root ~]# cat record.log | grep AAA | grep -v BBB | wc -l
```

2. 输出重定向和错误重定向 - > / >> / 2>。

```
[root ~]# cat readme.txt
banana
apple
grape
apple
grape
watermelon
pear
pitaya
[root ~]# cat readme.txt | sort | uniq > result.txt
[root ~]# cat result.txt
apple
banana
grape
pear
pitaya
watermelon
```

3. 输入重定向 - <。

```
[root ~]# echo 'hello, world!' > hello.txt
[root ~]# wall < hello.txt
[root ~]#
Broadcast message from root (Wed Jun 20 19:43:05 2018):
hello, world!
[root ~]# echo 'I will show you some code.' >> hello.txt
[root ~]# wall < hello.txt
[root ~]#
Broadcast message from root (Wed Jun 20 19:43:55 2018):
hello, world!
I will show you some code.
```

4. 多重定向 - tee。

下面的命令除了在终端显示命令ls的结果之外，还会追加输出到ls.txt文件中。

```
[root ~]# ls | tee -a ls.txt
```

别名

1. alias

```
[root ~]# alias ll='ls -l'  
[root ~]# alias frm='rm -rf'  
[root ~]# ll  
...  
drwxr-xr-x  2 root      root  4096 Jun 20 12:52 abc  
...  
[root ~]# frm abc
```

2. unalias

```
[root ~]# unalias frm  
[root ~]# frm sohu.html  
-bash: frm: command not found
```

文本处理

1. 字符流编辑器 - sed。

sed是操作、过滤和转换文本内容的工具。假设有一个名为fruit.txt的文件，内容如下所示。

```
[root ~]# cat -n fruit.txt  
1  banana  
2  grape  
3  apple  
4  watermelon  
5  orange
```

接下来，我们在第2行后面添加一个pitaya。

```
[root ~]# sed '2a pitaya' fruit.txt  
banana  
grape  
pitaya  
apple  
watermelon  
orange
```

注意：刚才的命令和之前我们讲过的很多命令一样并没有改变fruit.txt文件，而是将添加了新行的内容输出到终端中，如果想保存到fruit.txt中，可以使用输出重定向操作。

在第2行前面插入一个waxberry。

```
[root ~]# sed '2i waxberry' fruit.txt
banana
waxberry
grape
apple
watermelon
orange
```

删除第3行。

```
[root ~]# sed '3d' fruit.txt
banana
grape
watermelon
orange
```

删除第2行到第4行。

```
[root ~]# sed '2,4d' fruit.txt
banana
orange
```

将文本中的字符a替换为@。

```
[root ~]# sed 's#a#@#' fruit.txt
b@nana
gr@pe
@pple
w@termelon
or@nge
```

将文本中的字符a替换为@，使用全局模式。

```
[root ~]# sed 's#a#@#g' fruit.txt
b@n@n@
gr@pe
@pple
```

```
w@termelon
or@nge
```

2. 模式匹配和处理语言 - **awk**。

awk是一种编程语言，也是Linux系统中处理文本最为强大的工具，它的作者之一和现在的维护者就是之前提到过的Brian Kernighan (ken和dmr最亲密的伙伴)。通过该命令可以从文本中提取出指定的列、用正则表达式从文本中取出我们想要的内容、显示指定的行以及进行统计和运算，总之它非常强大。

假设有一个名为fruit2.txt的文件，内容如下所示。

```
[root ~]# cat fruit2.txt
1      banana      120
2      grape       500
3      apple       1230
4      watermelon  80
5      orange       400
```

显示文件的第3行。

```
[root ~]# awk 'NR==3' fruit2.txt
3      apple       1230
```

显示文件的第2列。

```
[root ~]# awk '{print $2}' fruit2.txt
banana
grape
apple
watermelon
orange
```

显示文件的最后一列。

```
[root ~]# awk '{print $NF}' fruit2.txt
120
500
1230
80
400
```

输出末尾数字大于等于300的行。

```
[root ~]# awk '{if($3 >= 300) {print $0}}' fruit2.txt
2      grape      500
3      apple      1230
5      orange     400
```

上面展示的只是awk命令的冰山一角，更多的内容留给读者自己在实践中去探索。

用户管理

1. 创建和删除用户 - **useradd / userdel**。

```
[root home]# useradd hellokitty
[root home]# userdel hellokitty
```

- **-d** - 创建用户时为用户指定用户主目录
- **-g** - 创建用户时指定用户所属的用户组

2. 创建和删除用户组 - **groupadd / groupdel**。

说明：用户组主要是为了方便对一个组里面所有用户的管理。

3. 修改密码 - **passwd**。

```
[root ~]# passwd hellokitty
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

说明：输入密码和确认密码没有回显且必须一气呵成的输入完成（不能使用退格键），密码和确认密码需要一致。如果使用**passwd**命令时没有指定命令作用的对象，则表示要修改当前用户的密码。如果想批量修改用户密码，可以使用**chpasswd**命令。

- **-l / -u** - 锁定/解锁用户。
- **-d** - 清除用户密码。
- **-e** - 设置密码立即过期，用户登录时会强制要求修改密码。
- **-i** - 设置密码过期多少天以后禁用该用户。

4. 查看和修改密码有效期 - **chage**。

设置hellokitty用户100天后必须修改密码，过期前15天通知该用户，过期后7天禁用该用户。

```
chage -M 100 -W 15 -I 7 hellokitty
```

5. 切换用户 - **su**。

```
[root ~]# su hellokitty
[hellokitty root]$
```

6. 以管理员身份执行命令 - **sudo**。

```
[hellokitty ~]$ ls /root
ls: cannot open directory /root: Permission denied
[hellokitty ~]$ sudo ls /root
[sudo] password for hellokitty:
```

说明: 如果希望用户能够以管理员身份执行命令, 用户必须要出现在sudoers名单中, sudoers文件在 **/etc** 目录下, 如果希望直接编辑该文件也可以使用下面的命令。

7. 编辑sudoers文件 - **visudo**。

这里使用的编辑器是vi, 关于vi的知识在后面有讲解。该文件的部分内容如下所示:

```
## Allow root to run any commands anywhere
root    ALL=(ALL)    ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES,
LOCATE, DRIVERS
## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)    ALL

## Same thing without a password
# %wheel    ALL=(ALL)    NOPASSWD: ALL

## Allows members of the users group to mount and unmount the
## cdrom as root
# %users  ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom

## Allows members of the users group to shutdown this system
# %users  localhost=/sbin/shutdown -h now
```

8. 显示用户与用户组的信息 - **id**。

9. 给其他用户发消息 -**write / wall**。

发送方:

```
[root ~]# write hellokitty
Dinner is on me.
```

```
Call me at 6pm.
```

接收方：

```
[hellokitty ~]$  
Message from root on pts/0 at 17:41 ...  
Dinner is on me.  
Call me at 6pm.  
EOF
```

10. 查看/设置是否接收其他用户发送的消息 - **mesg**。

```
[hellokitty ~]$ mesg  
is y  
[hellokitty ~]$ mesg n  
[hellokitty ~]$ mesg  
is n
```

文件系统

文件和路径

- 命名规则：文件名的最大长度与文件系统类型有关，一般情况下，文件名不应该超过255个字符，虽然绝大多数的字符都可以用于文件名，但是最好使用英文大小写字母、数字、下划线、点这样的符号。文件名中虽然可以使用空格，但应该尽可能避免使用空格，否则在输入文件名时需要用将文件名放在双引号中或者通过\对空格进行转义。
- 扩展名：在Linux系统下文件的扩展名是可选的，但是使用扩展名有助于对文件内容的理解。有些应用程序要通过扩展名来识别文件，但是更多的应用程序并不依赖文件的扩展名，就像file命令在识别文件时并不是依据扩展名来判定文件的类型。
- 隐藏文件：以点开头的文件在Linux系统中是隐藏文件（不可见文件）。

目录结构

- ./bin - 基本命令的二进制文件。
- ./boot - 引导加载程序的静态文件。
- ./dev - 设备文件。
- ./etc - 配置文件。
- ./home - 普通用户主目录的父目录。
- ./lib - 共享库文件。
- ./lib64 - 共享64位库文件。
- ./lost+found - 存放未链接文件。
- ./media - 自动识别设备的挂载目录。
- ./mnt - 临时挂载文件系统的挂载点。
- ./opt - 可选插件软件包安装位置。
- ./proc - 内核和进程信息。

13. **/root** - 超级管理员用户主目录。
14. **/run** - 存放系统运行时需要的东西。
15. **/sbin** - 超级用户的二进制文件。
16. **/sys** - 设备的伪文件系统。
17. **/tmp** - 临时文件夹。
18. **/usr** - 用户应用目录。
19. **/var** - 变量数据目录。

访问权限

1. **chmod** - 改变文件模式比特。

```
[root ~]# ls -l
...
-rw-r--r-- 1 root      root 211878 Jun 19 16:06 sohu.html
...
[root ~]# chmod g+w,o+w sohu.html
[root ~]# ls -l
...
-rw-rw-rw- 1 root      root 211878 Jun 19 16:06 sohu.html
...
[root ~]# chmod 644 sohu.html
[root ~]# ls -l
...
-rw-r--r-- 1 root      root 211878 Jun 19 16:06 sohu.html
...
```

说明：通过上面的例子可以看出，用**chmod**改变文件模式比特有两种方式：一种是字符设定法，另一种是数字设定法。除了**chmod**之外，可以通过**umask**来设定哪些权限将在新文件的默认权限中被删除。

长格式查看目录或文件时显示结果及其对应权限的数值如下表所示。

文件类型	所有者权限	同组用户权限	其他用户权限
d (目录)	r w x 读 写 执行 7	r - x 读 执行 5	r - x 读 执行 5
- (文件)	r w - 读 写 6	r - - 读 4	r - - 读 4
l (链接)	r w x 读 写 执行 7	r w x 读 写 执行 7	r - x 读 执行 5

2. **chown** - 改变文件所有者。

```
[root ~]# ls -l
...
```

```
-rw-r--r-- 1 root root 54 Jun 20 10:06 readme.txt
...
[root ~]# chown hellokitty readme.txt
[root ~]# ls -l
...
-rw-r--r-- 1 hellokitty root 54 Jun 20 10:06 readme.txt
...
```

3. **chgrp** - 改变用户组。

磁盘管理

1. 列出文件系统的磁盘使用状况 - **df**。

```
[root ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        40G  5.0G  33G  14% /
devtmpfs        486M    0  486M  0% /dev
tmpfs           497M    0  497M  0% /dev/shm
tmpfs           497M  356K  496M  1% /run
tmpfs           497M    0  497M  0% /sys/fs/cgroup
tmpfs           100M    0  100M  0% /run/user/0
```

2. 磁盘分区表操作 - **fdisk**。

```
[root ~]# fdisk -l
Disk /dev/vda: 42.9 GB, 42949672960 bytes, 83886080 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000a42f4
      Device Boot      Start        End      Blocks   Id  System
  /dev/vda1   *        2048    83884031    41940992   83  Linux
Disk /dev/vdb: 21.5 GB, 21474836480 bytes, 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

3. 磁盘分区工具 - **parted**。

4. 格式化文件系统 - **mkfs**。

```
[root ~]# mkfs -t ext4 -v /dev/sdb
```

- **-t** - 指定文件系统的类型。

- `-c` - 创建文件系统时检查磁盘损坏情况。
- `-v` - 显示详细信息。

5. 文件系统检查 - **fsck**。

6. 转换或拷贝文件 - **dd**。

7. 挂载/卸载 - **mount / umount**。

8. 创建/激活/关闭交换分区 - **mkswap / swapon / swapoff**。

说明：执行上面这些命令会带有一定的风险，如果不清楚这些命令的用法，最好不用随意使用，在使用的过程中，最好对照参考资料进行操作，并在操作前确认是否要这么做。

编辑器 - vim

1. 启动vim。可以通过`vi`或`vim`命令来启动vim，启动时可以指定文件名来打开一个文件，如果没有指定文件名，也可以在保存的时候指定文件名。

```
[root ~]# vim guess.py
```

2. 命令模式、编辑模式和末行模式：启动vim进入的是命令模式（也称为Normal模式），在命令模式下输入英文字母`i`会进入编辑模式（Insert模式），屏幕下方出现`-- INSERT --`提示；在编辑模式下按下`Esc`会回到命令模式，此时如果输入英文`:`会进入末行模式，在末行模式下输入`q!`可以在不保存当前工作的情况下强行退出vim；在命令模式下输入`v`会进入可视模式（Visual模式），可以用光标选择一个区域再完成对应的操作。

3. 保存和退出vim：在命令模式下输入`:`进入末行模式，输入`wq`可以实现保存退出；如果想放弃编辑的内容输入`q!`强行退出，这一点刚才已经提到过了；在命令模式下也可以直接输入`ZZ`实现保存退出。如果只想保存文件不退出，那么可以在末行模式下输入`w`；可以在`w`后面输入空格再指定要保存的文件名。

4. 光标操作。

- 在命令模式下可以通过`h`、`j`、`k`、`l`来控制光标向左、下、上、右的方向移动，可以在字母前输入数字来表示移动的距离，例如：`10h`表示向左移动10个字符。
- 在命令模式下可以通过`Ctrl+y`和`Ctrl+e`来实现向上、向下滚动一行文本的操作，可以通过`Ctrl+f`和`Ctrl+b`来实现向前和向后翻页的操作。
- 在命令模式下可以通过输入英文字母`G`将光标移到文件的末尾，可以通过`gg`将光标移到文件的开始，也可以通过在`G`前输入数字来将光标移动到指定的行。

5. 文本操作。

- **删除：**在命令模式下可以用`dd`来删除整行；可以在`dd`前加数字来指定删除的行数；可以用`d$`来实现删除从光标处删到行尾的操作，也可以通过`d0`来实现从光标处删到行首的操作；如果想删除一个单词，可以使用`dw`；如果要删除全文，可以在输入`:%d`（其中`:`用来从命令模式进入末行模式）。
- **复制和粘贴：**在命令模式下可以用`yy`来复制整行；可以在`yy`前加数字来指定复制的行数；可以通过`p`将复制的内容粘贴到光标所在的地方。
- **撤销和恢复：**在命令模式下输入`u`可以撤销之前的操作；通过`Ctrl+r`可以恢复被撤销的操作。

- 对内容进行排序：在命令模式下输入`%!sort`。

6. 查找和替换。

- 查找操作需要输入`/`进入末行模式并提供正则表达式来匹配与之对应的内容，例如：`/doc.*\.`，输入`h`来向前搜索，也可以输入`N`来向后搜索。
- 替换操作需要输入`:`进入末行模式并指定搜索的范围、正则表达式以及替换后的内容和匹配选项，例如：`:1,$s/doc.*/hello/gice`，其中：
 - `g` - global：全局匹配。
 - `i` - ignore case：忽略大小写匹配。
 - `c` - confirm：替换时需要确认。
 - `e` - error：忽略错误。

7. 参数设定：在输入`:`进入末行模式后可以对vim进行设定。

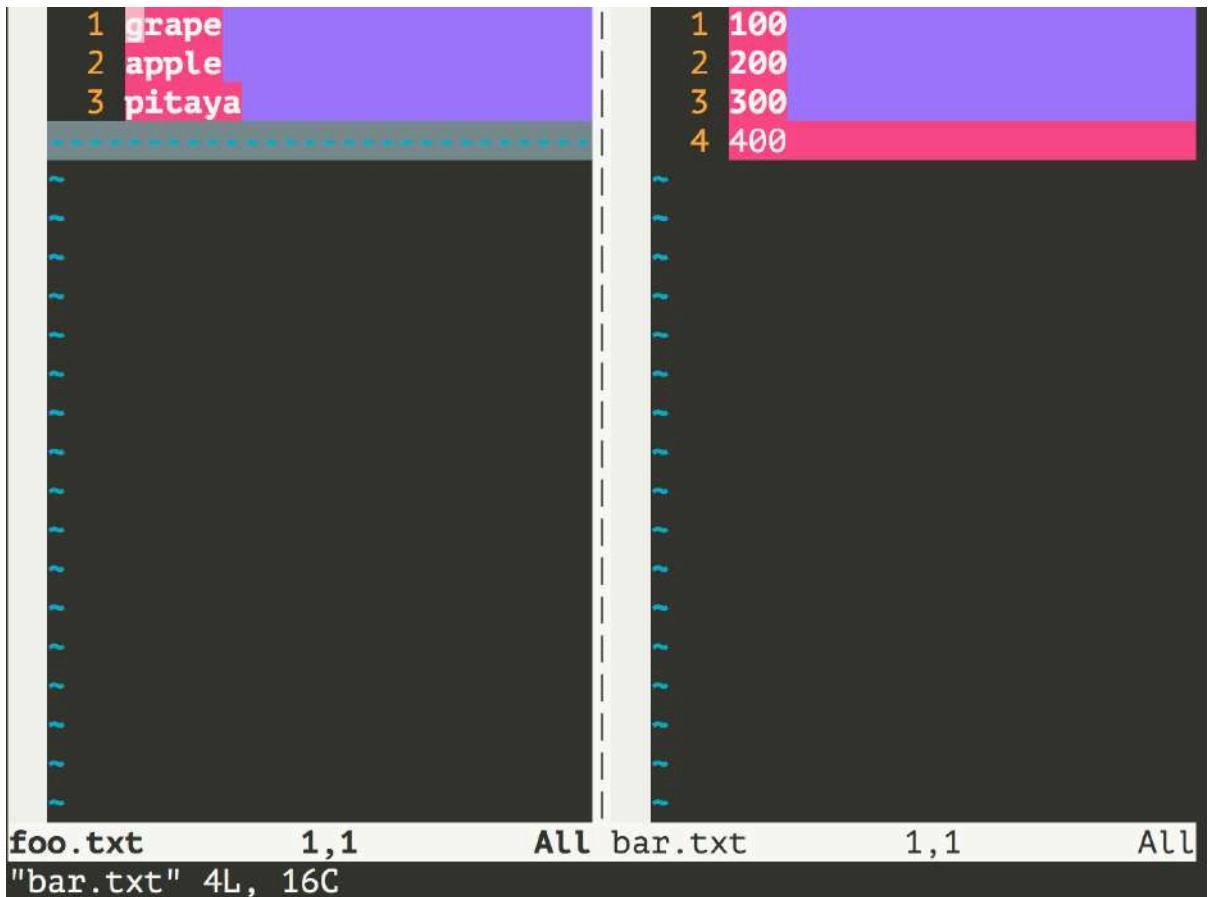
- 设置Tab键的空格数：`set ts=4`
- 设置显示/不显示行号：`set nu` / `set nonu`
- 设置启用/关闭高亮语法：`syntax on` / `syntax off`
- 设置显示标尺（光标所在的行和列）：`set ruler`
- 设置启用/关闭搜索结果高亮：`set hls` / `set nohls`

说明：如果希望上面的这些设定在每次启动vim时都能自动生效，需要将这些设定写到用户主目录下的.vimrc文件中。

8. 高级技巧

- 比较多个文件。

```
[root ~]# vim -d foo.txt bar.txt
```



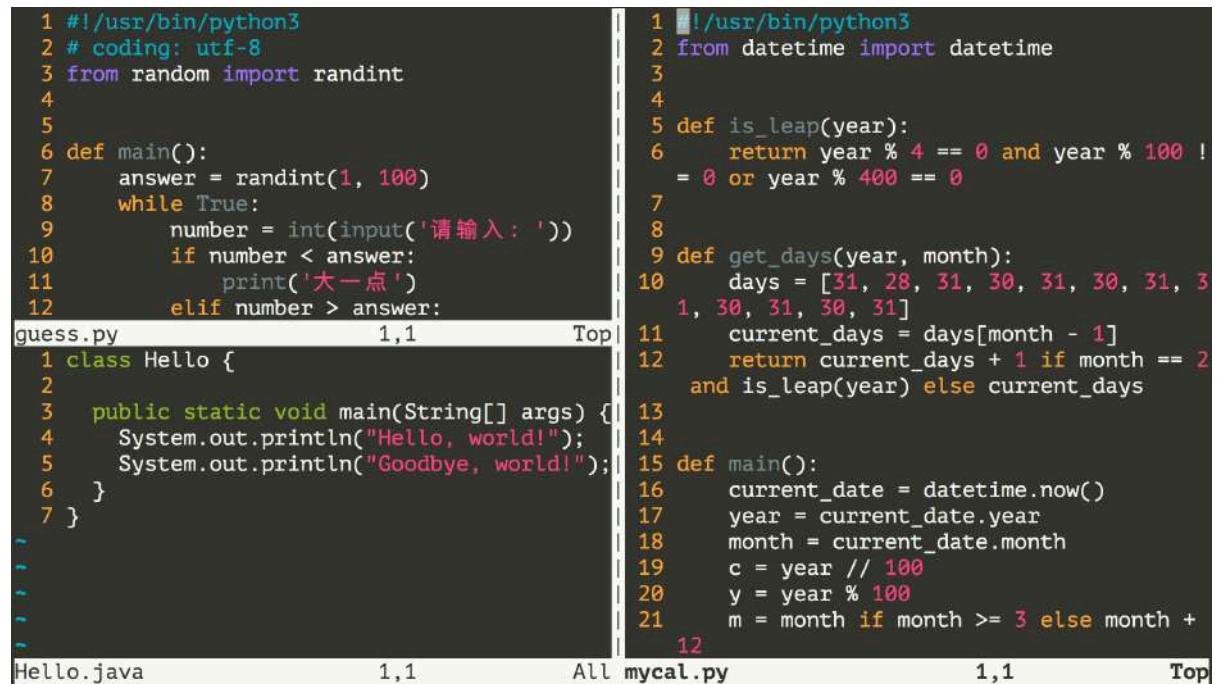
- 打开多个文件。

```
[root ~]# vim foo.txt bar.txt hello.txt
```

启动vim后只有一个窗口显示的是foo.txt，可以在末行模式中输入`ls`查看到打开的三个文件，也可以在末行模式中输入`b <num>`来显示另一个文件，例如可以用`:b 2`将bar.txt显示出来，可以用`:b 3`将hello.txt显示出来。

- 拆分和切换窗口。

可以在末行模式中输入`sp`或`vs`来实现对窗口的水平或垂直拆分，这样我们就可以同时打开多个编辑窗口，通过按两次`Ctrl+w`就可以实现编辑窗口的切换，在一个窗口中执行退出操作只会关闭对应的窗口，其他的窗口继续保留。



```

1 #!/usr/bin/python3
2 # coding: utf-8
3 from random import randint
4
5
6 def main():
7     answer = randint(1, 100)
8     while True:
9         number = int(input('请输入：'))
10        if number < answer:
11            print('大一点')
12        elif number > answer:
guess.py          1,1      Top
1 class Hello {
2
3     public static void main(String[] args) {
4         System.out.println("Hello, world!");
5         System.out.println("Goodbye, world!");
6     }
7 }

Hello.java        1,1      All mycal.py      1,1      Top

```

- 映射快捷键：在vim下可以将一些常用操作映射为快捷键来提升工作效率。

- 例子1：在命令模式下输入F4执行从第一行开始删除10000行代码的操作。

```
:map <F4> gg10000dd。
```

例子2：在编辑模式下输入_main直接补全为if __name__ == '__main__':。

```
:inoremap __main if __name__ == '__main__':
```

说明：上面例子2的inoremap中的i表示映射的键在编辑模式使用，nore表示不要递归，这一点非常重要，否则如果键对应的内容中又出现键本身，就会引发递归（相当于进入了死循环）。如果希望映射的快捷键每次启动vim时都能生效，需要将映射写到用户主目录下的.vimrc文件中。

- 录制宏。

- 在命令模式下输入qa开始录制宏（其中a是寄存器的名字，也可以是其他英文字母或0-9的数字）。
- 执行你的操作（光标操作、编辑操作等），这些操作都会被录制下来。
- 如果录制的操作已经完成了，按q结束录制。
- 通过@a（a是刚才使用的寄存器的名字）播放宏，如果要多次执行宏可以在前面加数字，例如100@a表示将宏播放100次。
- 可以试一试下面的例子来体验录制宏的操作，该例子来源于[Harttle Land网站](#)，该网站上提供了许多关于vim的使用技巧，有兴趣的可以了解一下。

```

1 | BOOL  | Boolean
2 | SINT  | Short integer
3 | INT   | Integer
4 | DINT  | Double integer
5 | LINT  | Long integer
6 | USINT | Unsigned short integer
7 | UINT  | Unsigned integer

```

- 首先按几次`<Esc>`进入normal模式，光标移到第一行，开始录制并存入m寄存器`qm`。
- 光标到行首`^`，到第二列词首`ww`，进入插入模式`i`，插入分隔符`,`，退出到normal模式`<Esc>`，到词尾`e`，进入插入模式`i`，插入分隔符`,`，退出到normal模式`<Esc>`，光标到下一行`j`。
- 结束录制`q`。
- 光标到第二行，在normal模式执行100次寄存器m中的宏`100@m`。

宏会在`j`执行错误后自动结束，得到如下文件：

```

1 | `BOOL` | Boolean
2 | `SINT` | Short integer
3 | `INT`  | Integer
4 | `DINT` | Double integer
5 | `LINT` | Long integer
6 | `USINT` | Unsigned short integer
7 | `UINT` | Unsigned integer

```

软件安装和配置

使用包管理工具

1. yum - Yellowdog Updater Modified。

- `yum search`: 搜索软件包，例如`yum search nginx`。
- `yum list installed`: 列出已经安装的软件包，例如`yum list installed | grep zlib`。
- `yum install`: 安装软件包，例如`yum install nginx`。
- `yum remove`: 删除软件包，例如`yum remove nginx`。
- `yum update`: 更新软件包，例如`yum update`可以更新所有软件包，而`yum update tar`只会更新tar。
- `yum check-update`: 检查有哪些可以更新的软件包。
- `yum info`: 显示软件包的相关信息，例如`yum info nginx`。

2. rpm - Redhat Package Manager。

- 安装软件包：`rpm -ivh <packagename>.rpm`。
- 移除软件包：`rpm -e <packagename>`。
- 查询软件包：`rpm -qa`，例如可以用`rpm -qa | grep mysql`来检查是否安装了MySQL相关的软件包。

下面以Nginx为例，演示如何使用yum安装软件。

```

[root ~]# yum -y install nginx
...
Installed:
  nginx.x86_64 1:1.12.2-2.el7
Dependency Installed:
  nginx-all-modules.noarch 1:1.12.2-2.el7
  nginx-mod-http-geoip.x86_64 1:1.12.2-2.el7
  nginx-mod-http-image-filter.x86_64 1:1.12.2-2.el7
  nginx-mod-http-perl.x86_64 1:1.12.2-2.el7

```

```
nginx-mod-http-xslt-filter.x86_64 1:1.12.2-2.el7
nginx-mod-mail.x86_64 1:1.12.2-2.el7
nginx-mod-stream.x86_64 1:1.12.2-2.el7
Complete!
[root ~]# yum info nginx
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Installed Packages
Name        : nginx
Arch        : x86_64
Epoch       : 1
Version    : 1.12.2
Release    : 2.el7
Size        : 1.5 M
Repo        : installed
From repo  : epel
Summary     : A high performance web server and reverse proxy server
URL         : http://nginx.org/
License     : BSD
Description : Nginx is a web server and a reverse proxy server for HTTP, SMTP,
POP3 and
          : IMAP protocols, with a strong focus on high concurrency, performance
and low
          : memory usage.
[root ~]# nginx -v
nginx version: nginx/1.12.2
```

移除Nginx。

```
[root ~]# yum -y remove nginx
```

下面以MySQL为例，演示如何使用rpm安装软件。要安装MySQL需要先到[MySQL官方网站](#)下载对应的RPM文件，当然要选择和你使用的Linux系统对应的版本。MySQL现在是Oracle公司旗下的产品，在MySQL被收购后，MySQL的作者重新制作了一个MySQL的分支MariaDB，可以通过yum进行安装。

```
[root mysql]# ls
mysql-community-client-5.7.22-1.el7.x86_64.rpm
mysql-community-common-5.7.22-1.el7.x86_64.rpm
mysql-community-libs-5.7.22-1.el7.x86_64.rpm
mysql-community-server-5.7.22-1.el7.x86_64.rpm
[root mysql]# yum -y remove mariadb-libs
[root mysql]# yum -y install libaio
[root mysql]# rpm -ivh mysql-community-common-5.7.26-1.el7.x86_64.rpm
...
[root mysql]# rpm -ivh mysql-community-libs-5.7.26-1.el7.x86_64.rpm
...
[root mysql]# rpm -ivh mysql-community-client-5.7.26-1.el7.x86_64.rpm
...
```

```
[root mysql]#rpm -ivh mysql-community-server-5.7.26-1.el7.x86_64.rpm
```

```
...
```

说明：由于MySQL和MariaDB的底层依赖库是有冲突的，所以上面我们首先用yum移除了名为mariadb-libs的依赖库并安装了名为libaio支持异步I/O操作的依赖库。关于MySQL和MariaDB之间的关系，可以阅读[维基百科](#)上关于MariaDB的介绍。

移除安装的MySQL。

```
[root ~]# rpm -qa | grep mysql | xargs rpm -e
```

下载解压配置环境变量

下面以安装MongoDB为例，演示这类软件应该如何安装。

```
[root ~]# wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-3.6.5.tgz
--2018-06-21 18:32:53--  https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-3.6.5.tgz
Resolving fastdl.mongodb.org (fastdl.mongodb.org)... 52.85.83.16, 52.85.83.228, 52.85.83.186, ...
Connecting to fastdl.mongodb.org (fastdl.mongodb.org)|52.85.83.16|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 100564462 (96M) [application/x-gzip]
Saving to: 'mongodb-linux-x86_64-rhel70-3.6.5.tgz'

100%[=====] 100,564,462  630KB/s
in 2m 9s
2018-06-21 18:35:04 (760 KB/s) - 'mongodb-linux-x86_64-rhel70-3.6.5.tgz' saved
[100564462/100564462]
[root ~]# gunzip mongodb-linux-x86_64-rhel70-3.6.5.tgz
[root ~]# tar -xvf mongodb-linux-x86_64-rhel70-3.6.5.tar
mongodb-linux-x86_64-rhel70-3.6.5/README
mongodb-linux-x86_64-rhel70-3.6.5/THIRD-PARTY-NOTICES
mongodb-linux-x86_64-rhel70-3.6.5/MPL-2
mongodb-linux-x86_64-rhel70-3.6.5/GNU-AGPL-3.0
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongodump
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongorestore
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoexport
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoimport
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongostat
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongotop
mongodb-linux-x86_64-rhel70-3.6.5/bin/bsondump
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongofiles
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoreplay
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongoperf
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongod
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongos
mongodb-linux-x86_64-rhel70-3.6.5/bin/mongo
```

```
mongodb-linux-x86_64-rhel70-3.6.5/bin/install_compass
[root ~]# vim .bash_profile
...
PATH=$PATH:$HOME/bin:$HOME/mongodb-linux-x86_64-rhel70-3.6.5/bin
export PATH
...
[root ~]# source .bash_profile
[root ~]# mongod --version
db version v3.6.5
git version: a20ecd3e3a174162052ff99913bc2ca9a839d618
OpenSSL version: OpenSSL 1.0.1e-fips 11 Feb 2013
allocator: tcmalloc
modules: none
build environment:
  distmod: rhel70
  distarch: x86_64
  target_arch: x86_64
[root ~]# mongo --version
MongoDB shell version v3.6.5
git version: a20ecd3e3a174162052ff99913bc2ca9a839d618
OpenSSL version: OpenSSL 1.0.1e-fips 11 Feb 2013
allocator: tcmalloc
modules: none
build environment:
  distmod: rhel70
  distarch: x86_64
  target_arch: x86_64
```

说明：当然也可以通过yum来安装MongoDB，具体可以参照官方网站上给出的说明。

源代码构建安装

1. 安装Python 3.6。

```
[root ~]# yum install gcc
[root ~]# wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz
[root ~]# gunzip Python-3.6.5.tgz
[root ~]# tar -xvf Python-3.6.5.tar
[root ~]# cd Python-3.6.5
[root ~]# ./configure --prefix=/usr/local/python36 --enable-optimizations
[root ~]# yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel
sqlite-devel readline-devel tk-devel gdbm-devel db4-devel libpcap-devel xz-devel
[root ~]# make && make install
...
[root ~]# ln -s /usr/local/python36/bin/python3.6 /usr/bin/python3
[root ~]# python3 --version
Python 3.6.5
[root ~]# python3 -m pip install -U pip
[root ~]# pip3 --version
```

说明：上面在安装好Python之后还需要注册PATH环境变量，将Python安装路径下bin文件夹的绝对路径注册到PATH环境变量中。注册环境变量可以修改用户主目录下的.bash_profile或者/etc目录下的profile文件，二者的区别在于前者相当于用户环境变量，而后者相当于系统环境变量。

2. 安装Redis-3.2.12。

```
[root ~]# wget http://download.redis.io/releases/redis-3.2.12.tar.gz
[root ~]# gunzip redis-3.2.12.tar.gz
[root ~]# tar -xvf redis-3.2.12.tar
[root ~]# cd redis-3.2.12
[root ~]# make && make install
[root ~]# redis-server --version
Redis server v=3.2.12 sha=00000000:0 malloc=jemalloc-4.0.3 bits=64
build=5bc5cd3c03d6ceb6
[root ~]# redis-cli --version
redis-cli 3.2.12
```

配置服务

我们可以Linux系统下安装和配置各种服务，也就是说我们可以把Linux系统打造成数据库服务器、Web服务器、缓存服务器、文件服务器、消息队列服务器等等。Linux下的大多数服务都被设置为守护进程（驻留在系统后台运行，但不会因为服务还在运行而导致Linux无法停止运行），所以我们安装的服务通常名字后面都有一个字母d，它是英文单词daemon的缩写，例如：防火墙服务叫firewalld，我们之前安装的MySQL服务叫mysqld，Apache服务器叫httpd等。在安装好服务之后，可以使用systemctl命令或service命令来完成对服务的启动、停止等操作，具体操作如下所示。

1. 启动防火墙服务。

```
[root ~]# systemctl start firewalld
```

2. 终止防火墙服务。

```
[root ~]# systemctl stop firewalld
```

3. 重启防火墙服务。

```
[root ~]# systemctl restart firewalld
```

4. 查看防火墙服务状态。

```
[root ~]# systemctl status firewalld
```

5. 设置/禁用防火墙服务开机自启。

```
[root ~]# systemctl enable firewalld
Created symlink from /etc/systemd/system/dbus-
org.fedoraproject.FirewallD1.service to
/usr/lib/systemd/system/firewalld.service.
Created symlink from /etc/systemd/system/multi-
user.target.wants/firewalld.service to
/usr/lib/systemd/system/firewalld.service.
[root ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-
user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-
org.fedoraproject.FirewallD1.service.
```

计划任务

1. 在指定的时间执行命令。

- **at** - 将任务排队，在指定的时间执行。
- **atq** - 查看待执行的任务队列。
- **atrm** - 从队列中删除待执行的任务。

指定3天以后下午5点要执行的任务。

```
[root ~]# at 5pm+3days
at> rm -f /root/*.html
at> <EOT>
job 9 at Wed Jun  5 17:00:00 2019
```

查看待执行的任务队列。

```
[root ~]# atq
9       Wed Jun  5 17:00:00 2019 a root
```

从队列中删除指定的任务。

```
[root ~]$ atrm 9
```

2. 计划任务表 - **crontab**。

```
[root ~]# crontab -e
* * * * * echo "hello, world!" >> /root/hello.txt
```

```
59 23 * * * rm -f /root/*.log
```

说明：输入`crontab -e`命令会打开vim来编辑Cron表达式并指定触发的任务，上面我们定制了两个计划任务，一个是每分钟向/root目录下的hello.txt中追加输出`hello, world!`；另一个是每天23时59分执行删除/root目录下以log为后缀名的文件。如果不知道Cron表达式如何书写，可以参考/etc/crontab文件中的提示（下面会讲到）或者用搜索引擎找一下“Cron表达式在线生成器”来生成Cron表达式。

和crontab相关的文件在`/etc`目录下，通过修改`/etc`目录下的crontab文件也能够定制计划任务。

```
[root ~]# cd /etc
[root etc]# ls -l | grep cron
-rw-----. 1 root root      541 Aug  3  2017 anacrontab
drwxr-xr-x. 2 root root    4096 Mar 27 11:56 cron.d
drwxr-xr-x. 2 root root    4096 Mar 27 11:51 cron.daily
-rw-----. 1 root root      0 Aug  3  2017 cron.deny
drwxr-xr-x. 2 root root    4096 Mar 27 11:50 cron.hourly
drwxr-xr-x. 2 root root    4096 Jun 10  2014 cron.monthly
-rw-r--r--  1 root root    493 Jun 23 15:09 crontab
drwxr-xr-x. 2 root root    4096 Jun 10  2014 cron.weekly
[root etc]# vim crontab
1 SHELL=/bin/bash
2 PATH=/sbin:/bin:/usr/sbin:/usr/bin
3 MAILTO=root
4
5 # For details see man 4 crontabs
6
7 # Example of job definition:
8 # ----- minute (0 - 59)
9 # | ----- hour (0 - 23)
10 # | | ----- day of month (1 - 31)
11 # | | | ----- month (1 - 12) OR jan,feb,mar,apr ...
12 # | | | | ----- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat
13 # | | | |
14 # * * * * * user-name command to be executed
```

网络访问和管理

1. 安全远程连接 - `ssh`。

```
[root ~]$ ssh root@120.77.222.217
The authenticity of host '120.77.222.217 (120.77.222.217)' can't be
established.
ECDSA key fingerprint is SHA256:BhUhykv+FvnIL03I9cLRpWpaCxI91m9n7zBWrcXRa8w.
ECDSA key fingerprint is
MD5:cc:85:e9:f0:d7:07:1a:26:41:92:77:6b:7f:a0:92:65.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '120.77.222.217' (ECDSA) to the list of known
```

```
hosts.
root@120.77.222.217's password:
```

2. 通过网络获取资源 - **wget**。

- -b 后台下载模式
- -O 下载到指定的目录
- -r 递归下载

3. 发送和接收邮件 - **mail**。

4. 网络配置工具 (旧) - **ifconfig**。

```
[root ~]# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.18.61.250 netmask 255.255.240.0 broadcast 172.18.63.255
              ether 00:16:3e:02:b6:46 txqueuelen 1000 (Ethernet)
                    RX packets 1067841 bytes 1296732947 (1.2 GiB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 409912 bytes 43569163 (41.5 MiB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions
```

5. 网络配置工具 (新) - **ip**。

```
[root ~]# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
              valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
      link/ether 00:16:3e:02:b6:46 brd ff:ff:ff:ff:ff:ff
        inet 172.18.61.250/20 brd 172.18.63.255 scope global eth0
              valid_lft forever preferred_lft forever
```

6. 网络可达性检查 - **ping**。

```
[root ~]# ping www.baidu.com -c 3
PING www.a.shifen.com (220.181.111.188) 56(84) bytes of data.
64 bytes from 220.181.111.188 (220.181.111.188): icmp_seq=1 ttl=51 time=36.3
ms
64 bytes from 220.181.111.188 (220.181.111.188): icmp_seq=2 ttl=51 time=36.4
ms
64 bytes from 220.181.111.188 (220.181.111.188): icmp_seq=3 ttl=51 time=36.4
ms
--- www.a.shifen.com ping statistics ---
```

```
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 36.392/36.406/36.427/0.156 ms
```

7. 显示或管理路由表 - **route**。

8. 查看网络服务和端口 - **netstat / ss**。

```
[root ~]# netstat -nap | grep nginx
```

9. 网络监听抓包 - **tcpdump**。

10. 安全文件拷贝 - **scp**。

```
[root ~]# scp root@1.2.3.4:/root/guido.jpg
hellokitty@4.3.2.1:/home/hellokitty/pic.jpg
```

11. 文件同步工具 - **rsync**。

说明：使用**rsync**可以实现文件的自动同步，这个对于文件服务器来说相当重要。关于这个命令的用法，我们在后面讲项目部署的时候为大家详细说明。

12. 安全文件传输 - **sftp**。

```
[root ~]# sftp root@1.2.3.4
root@1.2.3.4's password:
Connected to 1.2.3.4.
sftp>
```

- **help**: 显示帮助信息。
- **ls/l1s**: 显示远端/本地目录列表。
- **cd/lcd**: 切换远端/本地路径。
- **mkdir/lmkdir**: 创建远端/本地目录。
- **pwd/lpwd**: 显示远端/本地当前工作目录。
- **get**: 下载文件。
- **put**: 上传文件。
- **rm**: 删除远端文件。
- **bye/exit/quit**: 退出sftp。

1. 查看进程 - **ps**。

```
[root ~]# ps -ef
UID      PID  PPID  C STIME TTY          TIME CMD
root      1      0  0 Jun23 ?          00:00:05 /usr/lib/systemd/systemd --
switched-root --system --deserialize 21
root      2      0  0 Jun23 ?          00:00:00 [kthreadd]
...
[root ~]# ps -ef | grep mysql
root      4943  4581  0 22:45 pts/0    00:00:00 grep --color=auto mysql
mysql    25257      1  0 Jun25 ?          00:00:39 /usr/sbin/mysql --daemonize
--pid-file=/var/run/mysql/mysql.pid
```

2. 显示进程状态树 - **pstree**。

```
[root ~]# pstree
systemd—AliYunDun—18*[{AliYunDun}]
  |-AliYunDunUpdate—3*[{AliYunDunUpdate}]
  |-2*[agetty]
  |-aliyun-service—2*[{aliyun-service}]
  |-atd
  |-auditd—{auditd}
  |-dbus-daemon
  |-dhclient
  |-irqbalance
  |-lvmtd
  |-mysqld—28*[{mysqld}]
  |-nginx—2*[nginx]
  |-ntpd
  |-polkitd—6*[{polkitd}]
  |-rsyslogd—2*[{rsyslogd}]
  |-sshd—sshd—bash—pstree
  |-systemd-journal
  |-systemd-logind
  |-systemd-udevd
  |-tuned—4*[{tuned}]
```

3. 查找与指定条件匹配的进程 - **pgrep**。

```
[root ~]$ pgrep mysqld
3584
```

4. 通过进程号终止进程 - **kill**。

```
[root ~]$ kill -1
1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
```

```
6) SIGABRT      7) SIGBUS      8) SIGFPE      9) SIGKILL      10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT    17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37)
SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42)
SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47)
SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52)
SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57)
SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62)
SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
[root ~]# kill 1234
[root ~]# kill -9 1234
```

5. 通过进程名终止进程 - **killall** / **pkill**。

结束名为mysql的进程。

```
[root ~]# pkill mysql
```

结束hellokitty用户的所有进程。

```
[root ~]# pkill -u hellokitty
```

说明：这样的操作会让hellokitty用户和服务器断开连接。

6. 将进程置于后台运行。

- **Ctrl+Z** - 快捷键，用于停止进程并置于后台。
- **&** - 将进程置于后台运行。

```
[root ~]# mongod &
[root ~]# redis-server
...
^Z
[4]+  Stopped                  redis-server
```

7. 查询后台进程 - **jobs**。

```
[root ~]# jobs
[2]  Running                 mongod &
[3]-  Stopped                 cat
[4]+  Stopped                 redis-server
```

8. 让进程在后台继续运行 - **bg**。

```
[root ~]# bg %4
[4]+  redis-server &
[root ~]# jobs
[2]  Running                 mongod &
[3]+  Stopped                 cat
[4]-  Running                 redis-server &
```

9. 将后台进程置于前台 - **fg**。

```
[root ~]# fg %4
redis-server
```

说明：置于前台的进程可以使用**Ctrl+C**来终止它。

10. 调整程序/进程运行时优先级 - **nice / renice**。

11. 用户登出后进程继续工作 - **nohup**。

```
[root ~]# nohup ping www.baidu.com > result.txt &
```

12. 跟踪进程系统调用情况 - **strace**。

```
[root ~]# pgrep mysqld
8803
[root ~]# strace -c -p 8803
strace: Process 8803 attached
^Cstrace: Process 8803 detached
% time      seconds  usecs/call      calls      errors syscall
----- -----
 99.18    0.005719      5719          1          0 restart_syscall
  0.49    0.000028        28          1          0 mprotect
  0.24    0.000014        14          1          0 clone
  0.05    0.000003        3          1          0 mmap
  0.03    0.000002        2          1          0 accept
----- -----
100.00    0.005766          5          1          0 total
```

说明：这个命令的用法和参数都比较复杂，建议大家在真正用到这个命令的时候再根据实际需要进行了解。

13. 查看当前运行级别 - **runlevel**。

```
[root ~]# runlevel
N 3
```

14. 实时监控进程占用资源状况 - **top**。

```
[root ~]# top
top - 23:04:23 up 3 days, 14:10, 1 user, load average: 0.00, 0.01, 0.05
Tasks: 65 total, 1 running, 64 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.3 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0
st
KiB Mem : 1016168 total, 191060 free, 324700 used, 500408 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 530944 avail Mem
...
```

- **-c** - 显示进程的整个路径。
- **-d** - 指定两次刷屏之间的间隔时间（秒为单位）。
- **-i** - 不显示闲置进程或僵尸进程。
- **-p** - 显示指定进程的信息。

系统诊断

1. 系统启动异常诊断 - **dmesg**。

2. 查看系统活动信息 - **sar**。

```
[root ~]# sar -u -r 5 10
Linux 3.10.0-957.10.1.el7.x86_64 (izwz97tbg091kabnat2lo8z) 06/02/2019
_x86_64_ (2 CPU)

06:48:30 PM      CPU      %user      %nice      %system      %iowait      %steal
%idle
06:48:35 PM      all      0.10      0.00      0.10      0.00      0.00
99.80

06:48:30 PM  kmemfree  kmemused  %memused  kbuffers  kbcached  kbcommit
%commit  kbactive  kbinact  kbdirty
06:48:35 PM  1772012  2108392      54.33      102816      1634528      784940
20.23      793328      1164704      0
```

- **-A** - 显示所有设备（CPU、内存、磁盘）的运行状况。
- **-u** - 显示所有CPU的负载情况。

- **-d** - 显示所有磁盘的使用情况。
- **-r** - 显示内存的使用情况。
- **-n** - 显示网络运行状态。

3. 查看内存使用情况 - **free**。

```
[root ~]# free
              total        used        free      shared  buff/cache
available
Mem:      1016168       323924      190452          356      501792
531800
Swap:          0          0          0
```

4. 虚拟内存统计 - **vmstat**。

```
[root ~]# vmstat
procs -----memory----- ---swap-- -----io---- -system-- -----cpu-
-----
r  b    swpd    free    buff    cache    si    so    bi    bo    in    cs us sy id
wa st
2  0      0 204020  79036 667532    0    0     5    18   101   58  1  0 99
0  0
```

5. CPU信息统计 - **mpstat**。

```
[root ~]# mpstat
Linux 3.10.0-957.5.1.el7.x86_64 (iZ8vba0s66jj1fmo601w4xZ)      05/30/2019
_x86_64_          (1 CPU)

01:51:54 AM  CPU  %usr  %nice  %sys %iowait  %irq  %soft  %steal
%guest  %gnice  %idle
01:51:54 AM  all   0.71    0.00    0.17    0.04    0.00    0.00    0.00
0.00    0.00   99.07
```

6. 查看进程使用内存状况 - **pmap**。

```
[root ~]# ps
  PID TTY          TIME CMD
 4581 pts/0    00:00:00 bash
 5664 pts/0    00:00:00 ps
[root ~]# pmap 4581
4581:  -bash
0000000000400000      884K r-x-- bash
00000000006dc000        4K r---- bash
00000000006dd000       36K rw--- bash
00000000006e6000       24K rw---  [ anon ]
```

```

00000000001de0000 400K rw--- [ anon ]
00007f82fe805000 48K r-x-- libnss_files-2.17.so
00007f82fe811000 2044K ----- libnss_files-2.17.so
...

```

7. 报告设备CPU和I/O统计信息 - **iostat**。

```

[root ~]# iostat
Linux 3.10.0-693.11.1.el7.x86_64 (iZwz97tbgo91kabnat2lo8Z) 06/26/2018
_x86_64_ (1 CPU)
avg-cpu: %user %nice %system %iowait %steal %idle
          0.79   0.00   0.20   0.04   0.00  98.97
Device:    tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
vda        0.85       6.78      21.32   2106565   6623024
vdb        0.00       0.01       0.00      2088       0

```

8. 显示所有PCI设备 - **lspci**。

```

[root ~]# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 Ethernet controller: Red Hat, Inc. Virtio network device
00:04.0 Communication controller: Red Hat, Inc. Virtio console
00:05.0 SCSI storage controller: Red Hat, Inc. Virtio block device
00:06.0 SCSI storage controller: Red Hat, Inc. Virtio block device
00:07.0 Unclassified device [00ff]: Red Hat, Inc. Virtio memory balloon

```

9. 显示进程间通信设施的状态 - **ipcs**。

```

[root ~]# ipcs

----- Message Queues -----
key      msqid      owner      perms      used-bytes   messages
----- Shared Memory Segments -----
key      shmid      owner      perms      bytes      nattch      status
----- Semaphore Arrays -----
key      semid      owner      perms      nsems

```

Shell编程

之前我们提到过，Shell是一个连接用户和操作系统的应用程序，它提供了人机交互的界面（接口），用户通过这个界面访问操作系统内核的服务。Shell脚本是一种为Shell编写的脚本程序，我们可以通过Shell脚本来进行系统管理，同时也可以通过它进行文件操作。总之，编写Shell脚本对于使用Linux系统的人来说，应该是一项标配技能。

互联网上有大量关于Shell脚本的相关知识，我不打算再此对Shell脚本做一个全面系统的讲解，我们通过下面的代码来感性的认识下Shell脚本就行了。

例子1：输入两个整数m和n，计算从m到n的整数求和的结果。

```
#!/usr/bin/bash
printf 'm = '
read m
printf 'n = '
read n
a=$m
sum=0
while [ $a -le $n ]
do
    sum=$((sum + a))
    a=$((a + 1))
done
echo '结果: '$sum
```

例子2：自动创建文件夹和指定数量的文件。

```
#!/usr/bin/bash
printf '输入文件夹名: '
read dir
printf '输入文件名: '
read file
printf '输入文件数量(<1000): '
read num
if [ $num -ge 1000 ]
then
    echo '文件数量不能超过1000'
else
    if [ -e $dir -a -d $dir ]
    then
        rm -rf $dir
    else
        if [ -e $dir -a -f $dir ]
        then
            rm -f $dir
        fi
    fi
    mkdir -p $dir
    index=1
```

```

while [ $index -le $num ]
do
    if [ $index -lt 10 ]
    then
        pre='00'
    elif [ $index -lt 100 ]
    then
        pre='0'
    else
        pre=''
    fi
    touch $dir'/'$file'_'$pre$index
    index=$[ index + 1 ]
done
fi

```

例子3：自动安装指定版本的Redis。

```

#!/usr/bin/bash
install_redis() {
    if ! which redis-server > /dev/null
    then
        cd /root
        wget $1$2'.tar.gz' >> install.log
        gunzip /root/$2'.tar.gz'
        tar -xf /root/$2'.tar'
        cd /root/$2
        make >> install.log
        make install >> install.log
        echo '安装完成'
    else
        echo '已经安装过Redis'
    fi
}

install_redis 'http://download.redis.io/releases/' $1

```

相关资源

1. Linux命令行常用快捷键

快捷键	功能说明
tab	自动补全命令或路径
Ctrl+a	将光标移动到命令行行首
Ctrl+e	将光标移动到命令行行尾
Ctrl+f	将光标向右移动一个字符

快捷键	功能说明
Ctrl+b	将光标向左移动一个字符
Ctrl+k	剪切从光标到行尾的字符
Ctrl+u	剪切从光标到行首的字符
Ctrl+w	剪切光标前面的一个单词
Ctrl+y	复制剪切命名剪切的内容
Ctrl+c	中断正在执行的任务
Ctrl+h	删除光标前面的一个字符
Ctrl+d	退出当前命令行
Ctrl+r	搜索历史命令
Ctrl+g	退出历史命令搜索
Ctrl+l	清除屏幕上所有内容在屏幕的最上方开启一个新行
Ctrl+s	锁定终端使之暂时无法输入内容
Ctrl+q	退出终端锁定
Ctrl+z	将正在终端执行的任务停下来放到后台
!!	执行上一条命令
!数字	执行数字对应的历史命令
!字母	执行最近的以字母打头的命令
!\$ / Esc+.	获得上一条命令最后一个参数
Esc+b	移动到当前单词的开头
Esc+f	移动到当前单词的结尾

2. man查阅命令手册的内容说明

手册中的标题	功能说明
NAME	命令的说明和介绍
SYNOPSIS	使用该命令的基本语法
DESCRIPTION	使用该命令的详细描述，各个参数的作用，有时候这些信息会出现在OPTIONS中
OPTIONS	命令相关参数选项的说明
EXAMPLES	使用该命令的参考例子
EXIT STATUS	命令结束的退出状态码，通常0表示成功执行
SEE ALSO	和命令相关的其他命令或信息

手册中的标题 功能说明

BUGS 和命令相关的缺陷的描述

AUTHOR 该命令的作者介绍

NoSQL入门

NoSQL概述

如今，大多数的计算机系统（包括服务器、PC、移动设备等）都会产生庞大的数据量。其实，早在2012年的时候，全世界每天产生的数据量就达到了2.5EB（艾字节， $\$1EB \approx 10^{18}B$ ）。这些数据有很大一部分是由关系型数据库来存储和管理的。早在1970年，E.F.Codd发表了论述关系型数据库的著名论文“*A relational model of data for large shared data banks*”，这篇文章奠定了关系型数据库的基础并在接下来的数十年时间内产生了深远的影响。实践证明，关系型数据库是实现数据持久化最为重要的方式，它也是大多数应用在选择持久化方案时的首选技术。

NoSQL是一项全新的数据库革命性运动，虽然它的历史可以追溯到1998年，但是NoSQL真正深入人心并得到广泛的应用是在进入大数据时候以后，业界普遍认为NoSQL是更适合大数据存储的技术方案，这才使得NoSQL的发展达到了前所未有的高度。2012年《纽约时报》的一篇专栏中写到，大数据时代已经降临，在商业、经济及其他领域中，决策将不再基于经验和直觉而是基于数据和分析而作出。事实上，在天文学、气象学、基因组学、生物学、社会学、互联网搜索引擎、金融、医疗、社交网络、电子商务等诸多领域，由于数据过于密集和庞大，在数据的分析和处理上也遇到了前所未有的限制和阻碍，这一切都使得对大数据处理技术的研究被提升到了新的高度，也使得各种NoSQL的技术方案进入到了公众的视野。

NoSQL数据库按照其存储类型可以大致分为以下几类：

类型	部分代表	特点
列族数据 库	HBase Cassandra Hypertable	顾名思义是按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的I/O优势，适合于批量数据处理和即时查询。
文档数据 库	MongoDB CouchDB ElasticSearch	文档数据库一般用类JSON格式存储数据，存储的内容是文档型的。这样也就有机对某些字段建立索引，实现关系数据库的某些功能，但不提供对参照完整性和分布事务的支持。
KV数 据库	DynamoDB Redis LevelDB	可以通过key快速查询到其value，有基于内存和基于磁盘两种实现方案。
图数 据库	Neo4J FlockDB JanusGraph	使用图结构进行语义查询的数据库，它使用节点、边和属性来表示和存储数据。图数据库从设计上，就可以简单快速的检索难以在关系系统中建模的复杂层次结构。
对象数 据库	db4o Versant	通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。

说明：想了解更多的NoSQL数据库，可以访问<http://nosql-database.org/>。

Redis概述

Redis是一种基于键值对的NoSQL数据库，它提供了对多种数据类型（字符串、哈希、列表、集合、有序集合、位图等）的支持，能够满足很多应用场景的需求。Redis将数据放在内存中，因此读写性能是非常惊人的。与此同时，Redis也提供了持久化机制，能够将内存中的数据保存到硬盘上，在发生意外状况时数据也不会丢掉。此

外，Redis还支持键过期、地理信息运算、发布订阅、事务、管道、Lua脚本扩展等功能，总而言之，Redis的功能和性能都非常强大，如果项目中要实现高速缓存和消息队列这样的服务，直接交给Redis就可以了。目前，国内外很多著名的企业和商业项目都使用了Redis，包括：Twitter、Github、StackOverflow、新浪微博、百度、优酷土豆、美团、小米、唯品会等。

Redis简介

2008年，一个名为Salvatore Sanfilippo的程序员为他开发的LLOOGG项目定制了专属的数据库（因为之前他无论怎样优化MySQL，系统性能已经无法再提升了），这项工作的成果就是Redis的初始版本。后来他将Redis的代码放到了全球最大的代码托管平台[Github](#)，从那以后，Redis引发了大量开发者的好评和关注，继而有数百人参与了Redis的开发和维护，这使得Redis的功能越来越强大和性能越来越好。

Redis是REmote DIctionary Server的缩写，它是一个用ANSI C编写的高性能的key-value存储系统，与其他的key-value存储系统相比，Redis有以下一些特点（也是优点）：

- Redis的读写性能极高，并且有丰富的特性（发布/订阅、事务、通知等）。
- Redis支持数据的持久化（RDB和AOF两种方式），可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
- Redis支持多种数据类型，包括：string、hash、list、set、zset、bitmap、hyperloglog等。
- Redis支持主从复制（实现读写分离）以及哨兵模式（监控master是否宕机并自动调整配置）。
- Redis支持分布式集群，可以很容易的通过水平扩展来提升系统的整体性能。
- Redis基于TCP提供的可靠传输服务进行通信，很多编程语言都提供了Redis客户端支持。

Redis的应用场景

1. 高速缓存 - 将不常变化但又经常被访问的热点数据放到Redis数据库中，可以大大降低关系型数据库的压力，从而提升系统的响应性能。
2. 排行榜 - 很多网站都有排行榜功能，利用Redis中的列表和有序集合可以非常方便的构造各种排行榜系统。
3. 商品秒杀/投票点赞 - Redis提供了对计数操作的支持，网站上常见的秒杀、点赞等功能都可以利用Redis的计数器通过+1或-1的操作来实现，从而避免了使用关系型数据的[update](#)操作。
4. 分布式锁 - 利用Redis可以跨多台服务器实现分布式锁（类似于线程锁，但是能够被多台机器上的多个线程或进程共享）的功能，用于实现一个阻塞式操作。
5. 消息队列 - 消息队列和高速缓存一样，是一个大型网站不可缺少的基础服务，可以实现业务解耦和非实时业务削峰等特性，这些我们都会在后面的项目中为大家展示。

Redis的安装和配置

可以使用Linux系统的包管理工具（如yum）来安装Redis，也可以通过在Redis的[官方网站](#)下载Redis的源代码，解压缩解归档之后通过make工具对源代码进行构建并安装，在更新这篇文档时，Redis官方提供的最新稳定版本是[Redis 5.0.10](#)。

下载：

```
wget https://download.redis.io/releases/redis-5.0.10.tar.gz
```

解压缩和解归档：

```
tar -zxf redis-5.0.10.tar.gz
```

进入Redis源代码目录：

```
cd redis-5.0.10
```

构建和安装：

```
make && make install
```

在redis源代码目录下有一个名为redis.conf的配置文件，我们可以先查看一下该文件。

```
vim redis.conf
```

下面我们对Redis的配置文件进行一个扼要的介绍。

配置Redis服务的IP地址和端口：

```
66 # IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES
67 # JUST COMMENT THE FOLLOWING LINE.
68 #
69 bind 127.0.0.1
70
71 # Protected mode is a layer of security protection, in order to avoid that
72 # Redis instances left open on the internet are accessed and exploited.
73 #
74 # When protected mode is on and if:
75 #
76 # 1) The server is not binding explicitly to a set of addresses using the
77 #     "bind" directive.
78 # 2) No password is configured.
79 #
80 # The server only accepts connections from clients connecting from the
81 # IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain
82 # sockets.
83 #
84 # By default protected mode is enabled. You should disable it only if
85 # you are sure you want clients from other hosts to connect to Redis
86 # even if no authentication is configured, nor a specific set of interfaces
87 # are explicitly listed using the "bind" directive.
88 protected-mode yes
89
90 # Accept connections on the specified port, default is 6379 (IANA #815344).
91 # If port 0 is specified Redis will not listen on a TCP socket.
92 port 6379
```

配置底层有多少个数据库：

```

183 # Set the number of databases. The default database is DB 0, you can select
184 # a different one on a per-connection basis using SELECT <dbid> where
185 # dbid is a number between 0 and 'databases'-1
186 databases 16

```

配置Redis的持久化机制 - RDB。

```

196 ##### SNAPSHOTTING #####
197 #
198 # Save the DB on disk:
199 #
200 #   save <seconds> <changes>
201 #
202 #   Will save the DB if both the given number of seconds and the given
203 #   number of write operations against the DB occurred.
204 #
205 #   In the example below the behaviour will be to save:
206 #   after 900 sec (15 min) if at least 1 key changed
207 #   after 300 sec (5 min) if at least 10 keys changed
208 #   after 60 sec if at least 10000 keys changed
209 #
210 #   Note: you can disable saving completely by commenting out all "save" lines.
211 #
212 #   It is also possible to remove all the previously configured save
213 #   points by adding a save directive with a single empty string argument
214 #   like in the following example:
215 #
216 #   save ""
217
218 save 900 1
219 save 300 10
220 save 60 10000

237 # Compress string objects using LZF when dump .rdb databases?
238 # For default that's set to 'yes' as it's almost always a win.
239 # If you want to save some CPU in the saving child set it to 'no' but
240 # the dataset will likely be bigger if you have compressible values or keys.
241 rdbcompression yes
242
243 # Since version 5 of RDB a CRC64 checksum is placed at the end of the file.
244 # This makes the format more resistant to corruption but there is a performance
245 # hit to pay (around 10%) when saving and loading RDB files, so you can disable it
246 # for maximum performances.
247 #
248 # RDB files created with checksum disabled have a checksum of zero that will
249 # tell the loading code to skip the check.
250 rdbchecksum yes
251
252 # The filename where to dump the DB
253 dbfilename dump.rdb

```

配置Redis的持久化机制 - AOF:

```
679 ##### APPEND ONLY MODE #####
680
681 # By default Redis asynchronously dumps the dataset on disk. This mode is
682 # good enough in many applications, but an issue with the Redis process or
683 # a power outage may result into a few minutes of writes lost (depending on
684 # the configured save points).
685 #
686 # The Append Only File is an alternative persistence mode that provides
687 # much better durability. For instance using the default data fsync policy
688 # (see later in the config file) Redis can lose just one second of writes in a
689 # dramatic event like a server power outage, or a single write if something
690 # wrong with the Redis process itself happens, but the operating system is
691 # still running correctly.
692 #
693 # AOF and RDB persistence can be enabled at the same time without problems.
694 # If the AOF is enabled on startup Redis will load the AOF, that is the file
695 # with the better durability guarantees.
696 #
697 # Please check http://redis.io/topics/persistence for more information.
698
699 appendonly no
700
701 # The name of the append only file (default: "appendonly.aof")
702
703 appendfilename "appendonly.aof"
```

配置访问Redis服务器的口令：

```
494 ##### SECURITY #####
495
496 # Require clients to issue AUTH <PASSWORD> before processing any other
497 # commands. This might be useful in environments in which you do not trust
498 # others with access to the host running redis-server.
499 #
500 # This should stay commented out for backward compatibility and because most
501 # people do not need auth (e.g. they run their own servers).
502 #
503 # Warning: since Redis is pretty fast an outside user can try up to
504 # 150k passwords per second against a good box. This means that you should
505 # use a very strong password otherwise it will be very easy to break.
506 #
507 # requirepass foobared
```

配置Redis的主从复制（通过主从复制可以实现读写分离）：

```

265 ##### REPLICATION #####
266
267 # Master-Replica replication. Use replicaof to make a Redis instance a copy of
268 # another Redis server. A few things to understand ASAP about Redis replication.
269 #
270 # +-----+ +-----+
271 # | Master | --> | Replica |
272 # | (receive writes) | | (exact copy) |
273 # +-----+ +-----+
274 #
275 # 1) Redis replication is asynchronous, but you can configure a master to
276 # stop accepting writes if it appears to be not connected with at least
277 # a given number of replicas.
278 # 2) Redis replicas are able to perform a partial resynchronization with the
279 # master if the replication link is lost for a relatively small amount of
280 # time. You may want to configure the replication backlog size (see the next
281 # sections of this file) with a sensible value depending on your needs.
282 # 3) Replication is automatic and does not need user intervention. After a
283 # network partition replicas automatically try to reconnect to masters
284 # and resynchronize with them.
285 #
286 # replicaof <masterip> <masterport>

```

配置慢查询：

```

977 ##### SLOW LOG #####
978
979 # The Redis Slow Log is a system to log queries that exceeded a specified
980 # execution time. The execution time does not include the I/O operations
981 # like talking with the client, sending the reply and so forth,
982 # but just the time needed to actually execute the command (this is the only
983 # stage of command execution where the thread is blocked and can not serve
984 # other requests in the meantime).
985 #
986 # You can configure the slow log with two parameters: one tells Redis
987 # what is the execution time, in microseconds, to exceed in order for the
988 # command to get logged, and the other parameter is the length of the
989 # slow log. When a new command is logged the oldest one is removed from the
990 # queue of logged commands.
991
992 # The following time is expressed in microseconds, so 1000000 is equivalent
993 # to one second. Note that a negative number disables the slow log, while
994 # a value of zero forces the logging of every command.
995 slowlog-log-slower-than 10000
996
997 # There is no limit to this length. Just be aware that it will consume memory.
998 # You can reclaim memory used by the slow log with SLOWLOG RESET.
999 slowlog-max-len 128

```

上面这些内容就是Redis的基本配置，如果你对上面的内容感到困惑也没有关系，先把Redis用起来再回头去推敲这些内容就行了。如果想找一些参考书，《Redis开发与运维》是一本不错的入门读物，而《Redis实战》是不错的进阶读物。

Redis的服务器和客户端

接下来启动Redis服务器，下面的方式将以默认的配置启动Redis服务。

```
redis-server
```

如果希望修改Redis的配置（如端口、认证口令、持久化方式等），可以通过下面两种方式。

方式一：通过参数指定认证口令和AOF持久化方式。

```
redis-server --requirepass yourpass --appendonly yes
```

方式二：通过指定的配置文件来修改Redis的配置。

```
redis-server /root/redis-5.0.10/redis.conf
```

下面我们使用第一种方式来启动Redis并将其置于后台运行，将Redis产生的输出重定向到名为redis.log的文件中。

```
redis-server --requirepass yourpass > redis.log &
```

可以通过ps或者netstat来检查Redis服务器是否启动成功。

```
ps -ef | grep redis-server
netstat -nap | grep redis-server
```

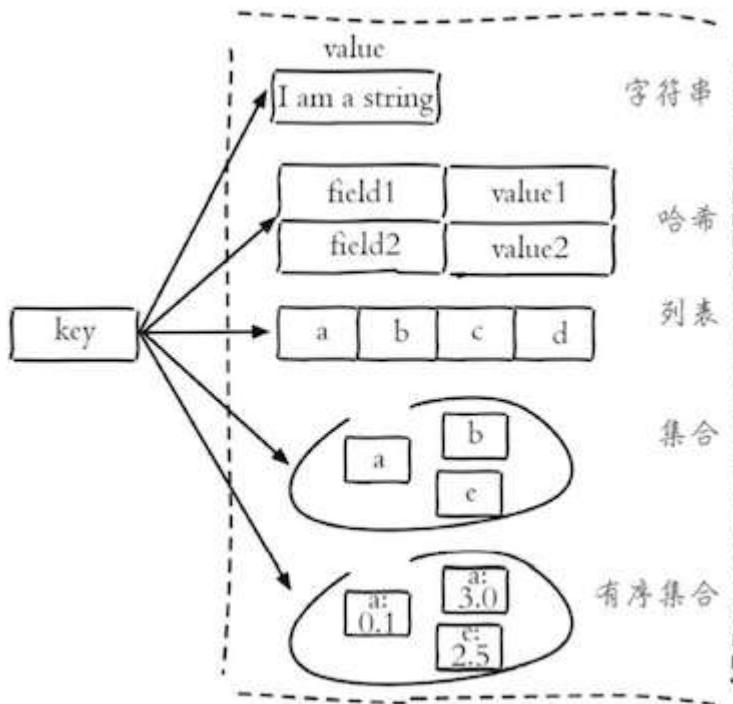
接下来，我们尝试用Redis命令行工具redis-cli去连接服务器，该工具默认连接本机的6379端口，如果需要指定Redis服务器和端口，可以使用-h和-p参数分别进行指定。

```
redis-cli
```

进入命令行工具后，就可以通过Redis的命令来操作Redis服务器，如下所示。

```
127.0.0.1:6379> auth yourpass
OK
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

Redis有着非常丰富的数据类型，也有很多的命令来操作这些数据，具体的内容可以查看[Redis命令参考](#)，在这个网站上，除了Redis的命令参考，还有Redis的详细文档，其中包括了通知、事务、主从复制、持久化、哨兵、集群等内容。



说明：上面的插图来自付磊和张益军编著的《Redis开发与运维》一书。

```

127.0.0.1:6379> set username admin
OK
127.0.0.1:6379> get username
"admin"
127.0.0.1:6379> set password "123456" ex 300
OK
127.0.0.1:6379> get password
"123456"
127.0.0.1:6379> ttl username
(integer) -1
127.0.0.1:6379> ttl password
(integer) 286
127.0.0.1:6379> hset stu1 name hao
(integer) 0
127.0.0.1:6379> hset stu1 age 38
(integer) 1
127.0.0.1:6379> hset stu1 gender male
(integer) 1
127.0.0.1:6379> hgetall stu1
1) "name"
2) "hao"
3) "age"
4) "38"
5) "gender"
6) "male"
127.0.0.1:6379> hvals stu1
1) "hao"
2) "38"
3) "male"
127.0.0.1:6379> hmset stu2 name wang age 18 gender female tel 13566778899

```

```
OK
127.0.0.1:6379> hgetall stu2
1) "name"
2) "wang"
3) "age"
4) "18"
5) "gender"
6) "female"
7) "tel"
8) "13566778899"
127.0.0.1:6379> lpush nums 1 2 3 4 5
(integer) 5
127.0.0.1:6379> lrange nums 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lpop nums
"5"
127.0.0.1:6379> lpop nums
"4"
127.0.0.1:6379> rpop nums
"1"
127.0.0.1:6379> rpop nums
"2"
127.0.0.1:6379> sadd fruits apple banana orange apple grape grape
(integer) 4
127.0.0.1:6379> scard fruits
(integer) 4
127.0.0.1:6379> smembers fruits
1) "grape"
2) "orange"
3) "banana"
4) "apple"
127.0.0.1:6379> sismember fruits apple
(integer) 1
127.0.0.1:6379> sismember fruits durian
(integer) 0
127.0.0.1:6379> sadd nums1 1 2 3 4 5
(integer) 5
127.0.0.1:6379> sadd nums2 2 4 6 8
(integer) 4
127.0.0.1:6379> sinter nums1 nums2
1) "2"
2) "4"
127.0.0.1:6379> sunion nums1 nums2
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "8"
```

```
127.0.0.1:6379> sdiff nums1 nums2
1) "1"
2) "3"
3) "5"
127.0.0.1:6379> zadd topsinger 5234 zhangxy 1978 chenyx 2235 zhoujl 3520 xuezq
(integer) 4
127.0.0.1:6379> zrange topsinger 0 -1 withscores
1) "chenyx"
2) "1978"
3) "zhoujl"
4) "2235"
5) "xuezq"
6) "3520"
7) "zhangxy"
8) "5234"
127.0.0.1:6379> zrevrange topsinger 0 -1
1) "zhangxy"
2) "xuezq"
3) "zhoujl"
4) "chenyx"
127.0.0.1:6379> zrevrank topsinger zhoujl
(integer) 2
127.0.0.1:6379> geoadd pois 116.39738549206541 39.90862689286386 tiananmen
(integer) 1
127.0.0.1:6379> geoadd pois 116.27172936413572 39.99135172904494 yiheyuan
(integer) 1
127.0.0.1:6379> geoadd pois 117.27766503308104 40.65332064313784 gubeishuizhen
(integer) 1
127.0.0.1:6379> geodist pois tiananmen gubeishuizhen km
"111.5333"
127.0.0.1:6379> geodist pois tiananmen yiheyuan km
"14.1230"
127.0.0.1:6379> georadius pois 116.86499108288572 40.40149669363615 50 km withdist
1) 1) "gubeishuizhen"
   2) "44.7408"
```

在Python程序中使用Redis

可以使用pip安装名为`redis`的三方库，该三方库的核心是一个名为`Redis`的类，`Redis`对象代表一个Redis客户端，通过该客户端可以向Redis服务器发送命令并获取执行的结果。上面我们在Redis客户端中使用的命令基本上就是`Redis`对象可以接收的消息，所以如果了解了Redis的命令就可以在Python中玩转Redis。

```
pip3 install redis
```

进入Python交互式环境，使用`redis`三方库来操作Redis。

```
>>> import redis
>>>
>>> client = redis.Redis(host='127.0.0.1', port=6379, password='yourpass')
```

```
>>>
>>> client.set('username', 'admin')
True
>>> client.hset('student', 'name', 'luohao')
1
>>> client.hset('student', 'age', 40)
1
>>> client.keys('*')
[b'username', b'student']
>>> client.get('username')
b'admin'
>>> client.hgetall('student')
{b'name': b'luohao', b'age': b'40'}
```

MongoDB概述

MongoDB简介

MongoDB是2009年问世的一个面向文档的数据库管理系统，由C++语言编写，旨在为Web应用提供可扩展的高性能数据存储解决方案。虽然在划分类别的时候后，MongoDB被认为是NoSQL的产品，但是它更像一个介于关系数据库和非关系数据库之间的产品，在非关系数据库中它功能最丰富，最像关系数据库。

MongoDB将数据存储为一个文档，一个文档由一系列的“键值对”组成，其文档类似于JSON对象，但是MongoDB对JSON进行了二进制处理（能够更快的定位key和value），因此其文档的存储格式称为BSON。关于JSON和BSON的差别大家可以看看MongoDB官方网站的文章 [《JSON and BSON》](#)。

目前，MongoDB已经提供了对Windows、macOS、Linux、Solaris等多个平台的支持，而且也提供了多种开发语言的驱动程序，Python当然是其中之一。

MongoDB的安装和启动

可以从MongoDB的[官方下载链接](#)下载MongoDB，官方提供了Windows、macOS和多种Linux版本的安装包。下面以CentOS为例，简单说一下如何安装和启动MongoDB。

下载服务器和命令行的RPM安装包。

```
wget https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.4/x86_64/RPMS/mongodb-org-server-4.4.2-1.el7.x86_64.rpm
rpm -ivh mongodb-org-server-4.4.2-1.el7.x86_64.rpm
wget https://repo.mongodb.org/yum/redhat/7/mongodb-org/4.4/x86_64/RPMS/mongodb-org-shell-4.4.2-1.el7.x86_64.rpm
rpm -ivh mongodb-org-shell-4.4.2-1.el7.x86_64.rpm
```

启动MongoDB服务器，需要先创建保存数据的文件夹。

```
mkdir -p /data/db
```

修改MongoDB的配置文件，将其中**bindIp**选项的值修改为本机IP地址而不是默认的**127.0.0.1**，本机IP地址可以通过**ifconfig**命令进行查看。

```
vim /etc/mongod.conf
```

使用**systemctl**命令启动服务。

```
systemctl start mongod
```

MongoDB基本概念

我们通过与关系型数据库的比较来说明MongoDB中的一些概念。

SQL	MongoDB
database	database
table (表)	collection (集合)
row (行)	document (文档)
column (列)	field (字段)
index	index
table joins (表连接)	(嵌套文档)
primary key	primary key

通过Shell操作MongoDB

0. 启动命令行工具，进入交互式环境。

```
mongo
```

说明：

1. 查看、创建和删除数据库。

```
> // 显示所有数据库
> show dbs
admin  0.000GB
config  0.000GB
local   0.000GB
> // 创建并切换到school数据库
> use school
switched to db school
```

```
> // 删除当前数据库
> db.dropDatabase()
{ "ok" : 1 }
```

2. 创建、删除和查看集合。

```
> // 创建并切换到school数据库
> use school
switched to db school
> // 创建colleges集合
> db.createCollection('colleges')
{ "ok" : 1 }
> // 创建students集合
> db.createCollection('students')
{ "ok" : 1 }
> // 查看所有集合
> show collections
colleges
students
> // 删除colleges集合
> db.colleges.drop()
true
```

说明：在MongoDB中插入文档时如果集合不存在会自动创建集合，所以也可以按照下面的方式通过插入文档来创建集合。

3. 文档的CRUD操作。

```
> // 向students集合插入文档
> db.students.insert({stuid: 1001, name: '骆昊', age: 40})
WriteResult({ "nInserted" : 1 })
> // 向students集合插入文档
> db.students.save({stuid: 1002, name: '王大锤', tel: '13012345678', gender: '男'})
WriteResult({ "nInserted" : 1 })
> // 查看所有文档
> db.students.find()
{ "_id" : ObjectId("5b13c72e006ad854460ee70b"), "stuid" : 1001, "name" : "骆昊", "age" : 38 }
{ "_id" : ObjectId("5b13c790006ad854460ee70c"), "stuid" : 1002, "name" : "王大锤", "tel" : "13012345678", "gender" : "男" }
> // 更新stuid为1001的文档
> db.students.update({stuid: 1001}, { '$set': {tel: '13566778899', gender: '男' } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> // 插入或更新stuid为1003的文档
> db.students.update({stuid: 1003}, { '$set': {name: '白元芳', tel: '13022223333', gender: '男' } }, { upsert=true })
WriteResult({
```

```
"nMatched" : 0,
"nUpserted" : 1,
"nModified" : 0,
"_id" : ObjectId("5b13c92dd185894d7283efab")
})
> // 查询所有文档
> db.students.find().pretty()
{
    "_id" : ObjectId("5b13c72e006ad854460ee70b"),
    "stuid" : 1001,
    "name" : "骆昊",
    "age" : 38,
    "gender" : "男",
    "tel" : "13566778899"
}
{
    "_id" : ObjectId("5b13c790006ad854460ee70c"),
    "stuid" : 1002,
    "name" : "王大锤",
    "tel" : "13012345678",
    "gender" : "男"
}
{
    "_id" : ObjectId("5b13c92dd185894d7283efab"),
    "stuid" : 1003,
    "gender" : "男",
    "name" : "白元芳",
    "tel" : "13022223333"
}
> // 查询stuid大于1001的文档
> db.students.find({stuid: {'$gt': 1001}}).pretty()
{
    "_id" : ObjectId("5b13c790006ad854460ee70c"),
    "stuid" : 1002,
    "name" : "王大锤",
    "tel" : "13012345678",
    "gender" : "男"
}
{
    "_id" : ObjectId("5b13c92dd185894d7283efab"),
    "stuid" : 1003,
    "gender" : "男",
    "name" : "白元芳",
    "tel" : "13022223333"
}
> // 查询stuid大于1001的文档只显示name和tel字段
> db.students.find({stuid: {'$gt': 1001}}, {_id: 0, name: 1, tel: 1}).pretty()
{ "name" : "王大锤", "tel" : "13012345678" }
{ "name" : "白元芳", "tel" : "13022223333" }
> // 查询name为“骆昊”或者tel为“13022223333”的文档
> db.students.find({'$or': [{name: '骆昊'}, {tel: '13022223333'}]}, {_id: 0, name: 1, tel: 1}).pretty()
{ "name" : "骆昊", "tel" : "13566778899" }
```

```

{ "name" : "白元芳", "tel" : "13022223333" }
> // 查询学生文档跳过第1条文档只查1条文档
> db.students.find().skip(1).limit(1).pretty()
{
    "_id" : ObjectId("5b13c790006ad854460ee70c"),
    "stuid" : 1002,
    "name" : "王大锤",
    "tel" : "13012345678",
    "gender" : "男"
}
> // 对查询结果进行排序(1表示升序, -1表示降序)
> db.students.find({}, {_id: 0, stuid: 1, name: 1}).sort({stuid: -1})
{ "stuid" : 1003, "name" : "白元芳" }
{ "stuid" : 1002, "name" : "王大锤" }
{ "stuid" : 1001, "name" : "骆昊" }
> // 在指定的一个或多个字段上创建索引
> db.students.ensureIndex({name: 1})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}

```

使用MongoDB可以非常方便的配置数据复制，通过冗余数据来实现数据的高可用以及灾难恢复，也可以通过数据分片来应对数据量迅速增长的需求。关于MongoDB更多的操作可以查阅[官方文档](#)，同时推荐大家阅读Kristina Chodorow写的《[MongoDB权威指南](#)》。

在Python程序中操作MongoDB

可以通过pip安装[pymongo](#)来实现对MongoDB的操作。

```
pip install pymongo
```

进入Python交互式环境，就可以执行以下的操作。

```

>>> from pymongo import MongoClient
>>>
>>> client = MongoClient('mongodb://127.0.0.1:27017')
>>> db = client.school
>>> for student in db.students.find():
...     print('学号:', student['stuid'])
...     print('姓名:', student['name'])
...     print('电话:', student['tel'])
...
学号: 1001.0
姓名: 骆昊
电话: 13566778899
学号: 1002.0

```

```
姓名: 王大锤
电话: 13012345678
学号: 1003.0
姓名: 白元芳
电话: 13022223333
>>> db.students.find().count()
3
>>> db.students.remove()
{'n': 3, 'ok': 1.0}
>>> db.students.find().count()
0
>>> from pymongo import ASCENDING
>>>
>>> coll = db.students
>>> coll.create_index([('name', ASCENDING)], unique=True)
'name_1'
>>> coll.insert_one({'stuid': int(1001), 'name': '骆昊', 'gender': True})
<pymongo.results.InsertOneResult object at 0x1050cc6c8>
>>> coll.insert_many([{'stuid': int(1002), 'name': '王大锤', 'gender': False},
{'stuid': int(1003), 'name': '白元芳', 'gender': True}])
<pymongo.results.InsertManyResult object at 0x1050cc8c8>
>>> for student in coll.find({'gender': True}):
...     print('学号:', student['stuid'])
...     print('姓名:', student['name'])
...     print('性别:', '男' if student['gender'] else '女')
...
学号: 1001
姓名: 骆昊
性别: 男
学号: 1003
姓名: 白元芳
性别: 男
```

关于pymongo更多的知识可以通过它的官方文档进行了解，也可以使用MongoEngine这样的库来简化Python程序对MongoDB的操作，除此之外，还有以异步I/O方式访问MongoDB的三方库motor都是不错的选择。

关系型数据库和MySQL概述

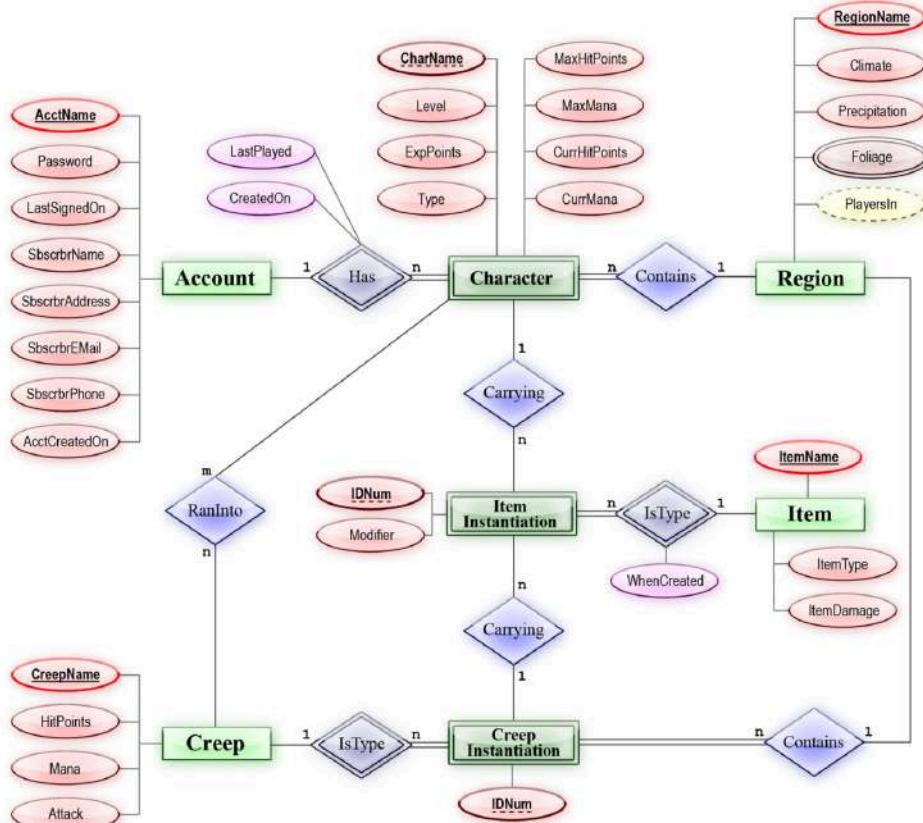
关系型数据库概述

1. 数据持久化 - 将数据保存到能够长久保存数据的存储介质中，在掉电的情况下数据也不会丢失。
2. 数据库发展史 - 网状数据库、层次数据库、关系数据库、NoSQL 数据库、NewSQL 数据库。

1970年，IBM的研究员E.F.Codd在*Communication of the ACM*上发表了名为*A Relational Model of Data for Large Shared Data Banks*的论文，提出了**关系模型**的概念，奠定了关系模型的理论基础。后来Codd又陆续发表多篇文章，论述了范式理论和衡量关系系统的12条标准，用数学理论奠定了关系数据库的基础。

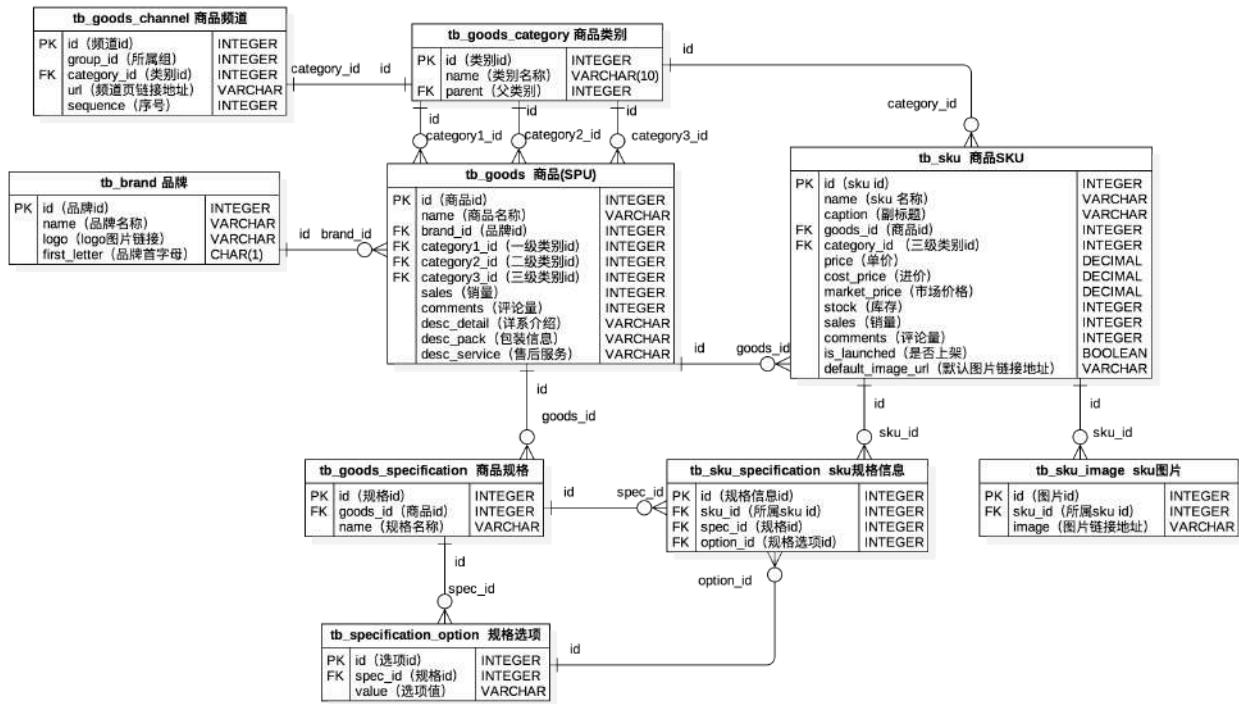
3. 关系数据库特点。
 - 理论基础：**关系代数**（关系运算、集合论、一阶谓词逻辑）。
 - 具体表象：用**二维表**（有行和列）组织数据。
 - 编程语言：**结构化查询语言**（SQL）。
4. ER模型（实体关系模型）和概念模型图。

ER模型，全称为**实体关系模型**（Entity-Relationship Model），由美籍华裔计算机科学家陈品山先生提出，是概念数据模型的高层描述方式，如下图所示。



- 实体 - 矩形框
- 属性 - 椭圆框
- 关系 - 菱形框
- 重数 - 1:1 (一对一) / 1:N (一对多) / M:N (多对多)

实际项目开发中，我们可以利用数据库建模工具（如：PowerDesigner）来绘制概念数据模型（其本质就是 ER 模型），然后再设置好目标数据库系统，将概念模型转换成物理模型，最终生成创建二维表的 SQL（很多工具都可以根据我们设计的物理模型图以及设定的目标数据库来导出 SQL 或直接生成数据表）。



5. 关系数据库产品。

- **Oracle** - 目前世界上使用最为广泛的数据库管理系统，作为一个通用的数据库系统，它具有完整的数据管理功能；作为一个关系数据库，它是一个完备关系的产品；作为分布式数据库，它实现了分布式处理的功能。在 Oracle 最新的 12c 版本中，还引入了多租户架构，使用该架构可轻松部署和管理数据库云。
- **DB2** - IBM 公司开发的、主要运行于 Unix（包括 IBM 自家的 AIX）、Linux、以及 Windows 服务器版等系统的关系数据库产品。DB2 历史悠久且被认为是最早使用 SQL 的数据库产品，它拥有较为强大的商业智能功能。
- **SQL Server** - 由 Microsoft 开发和推广的关系型数据库产品，最初适用于中小企业的数据管理，但是近年来它的应用范围有所扩展，部分大企业甚至是跨国公司也开始基于它来构建自己的数据管理系统。
- **MySQL** - MySQL 是开放源代码的，任何人都可以在 GPL (General Public License) 的许可下下载并根据个性化的需要对其进行修改。MySQL 因为其速度、可靠性和适应性而备受关注。
- **PostgreSQL** - 在 BSD 许可证下发行的开放源代码的关系数据库产品。

MySQL 简介

MySQL 最早是由瑞典的 MySQL AB 公司开发的一个开放源码的关系数据库管理系统，该公司于2008年被昇阳微系统公司（Sun Microsystems）收购。在2009年，甲骨文公司（Oracle）收购昇阳微系统公司，因此 MySQL 目前也是 Oracle 旗下产品。

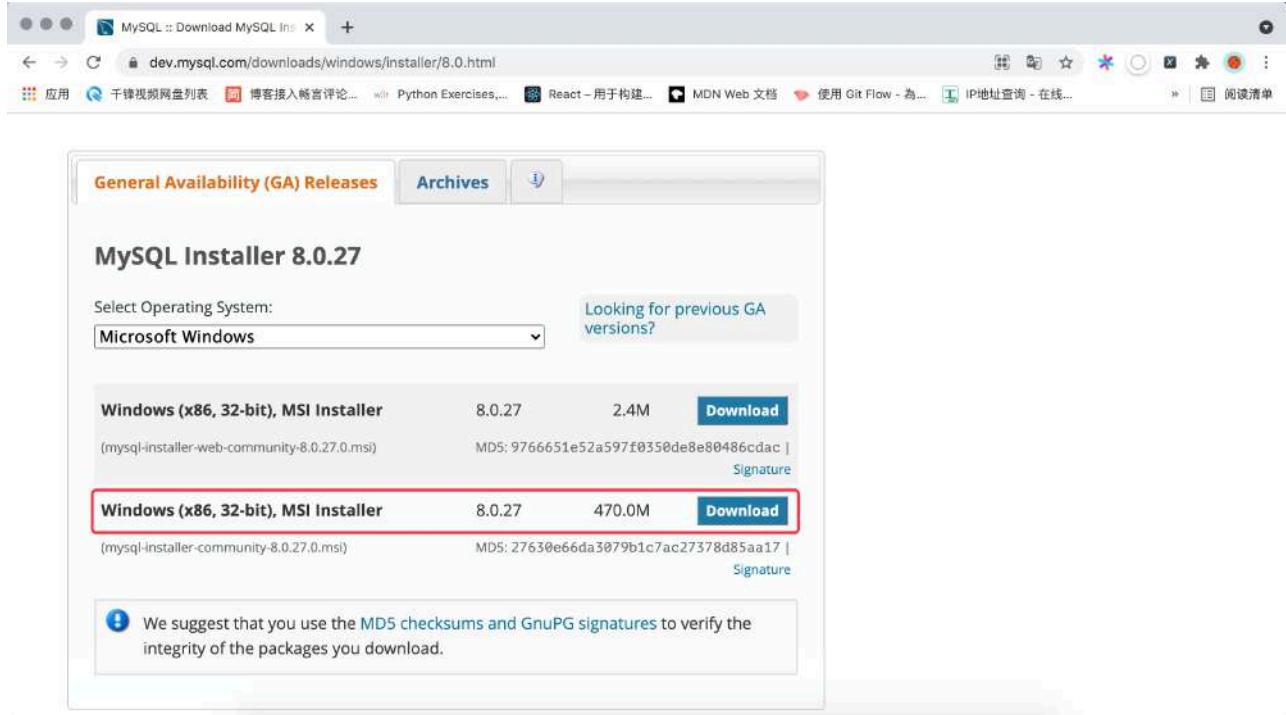
MySQL 在过去由于性能高、成本低、可靠性好，已经成为最流行的开源数据库，因此被广泛地应用于中小型网站开发。随着 MySQL 的不断成熟，它也逐渐被应用于更多大规模网站和应用，比如维基百科、谷歌（Google）、脸书（Facebook）、淘宝网等网站都使用了 MySQL 来提供数据持久化服务。

甲骨文公司收购后昇阳微系统公司，大幅调涨 MySQL 商业版的售价，且甲骨文公司不再支持另一个自由软件项目 [OpenSolaris](#) 的发展，因此导致自由软件社区对于 Oracle 是否还会持续支持 MySQL 社区版（MySQL 的各个发行版本中唯一免费的版本）有所担忧，MySQL 的创始人麦克尔·维德纽斯以 MySQL 为基础，创建了 [MariaDB](#)（以他女儿的名字命名的数据库）分支。有许多原来使用 MySQL 数据库的公司（例如：维基百科）已经陆续完成了从 MySQL 数据库到 MariaDB 数据库的迁移。

安装 MySQL

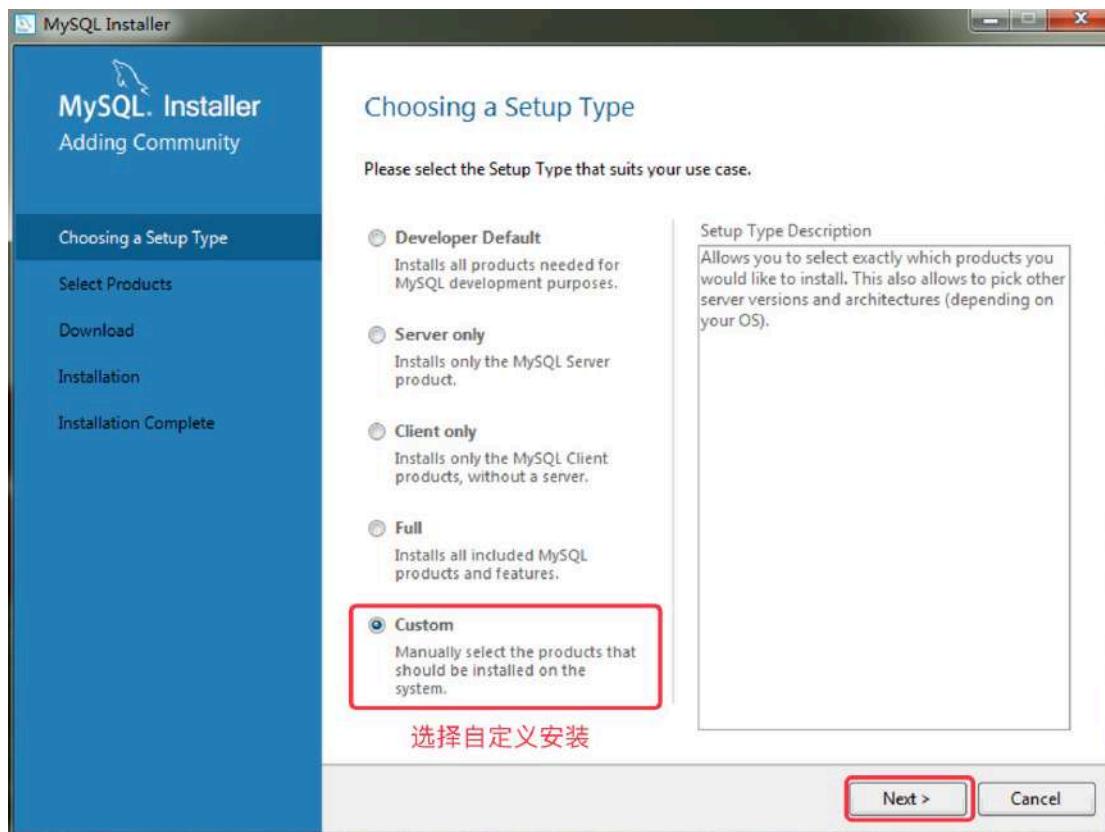
Windows 环境

1. 通过[官方网站](#)提供的下载链接下载“MySQL社区版服务器”安装程序，如下图所示，建议大家下载离线安装版的MySQL Installer。

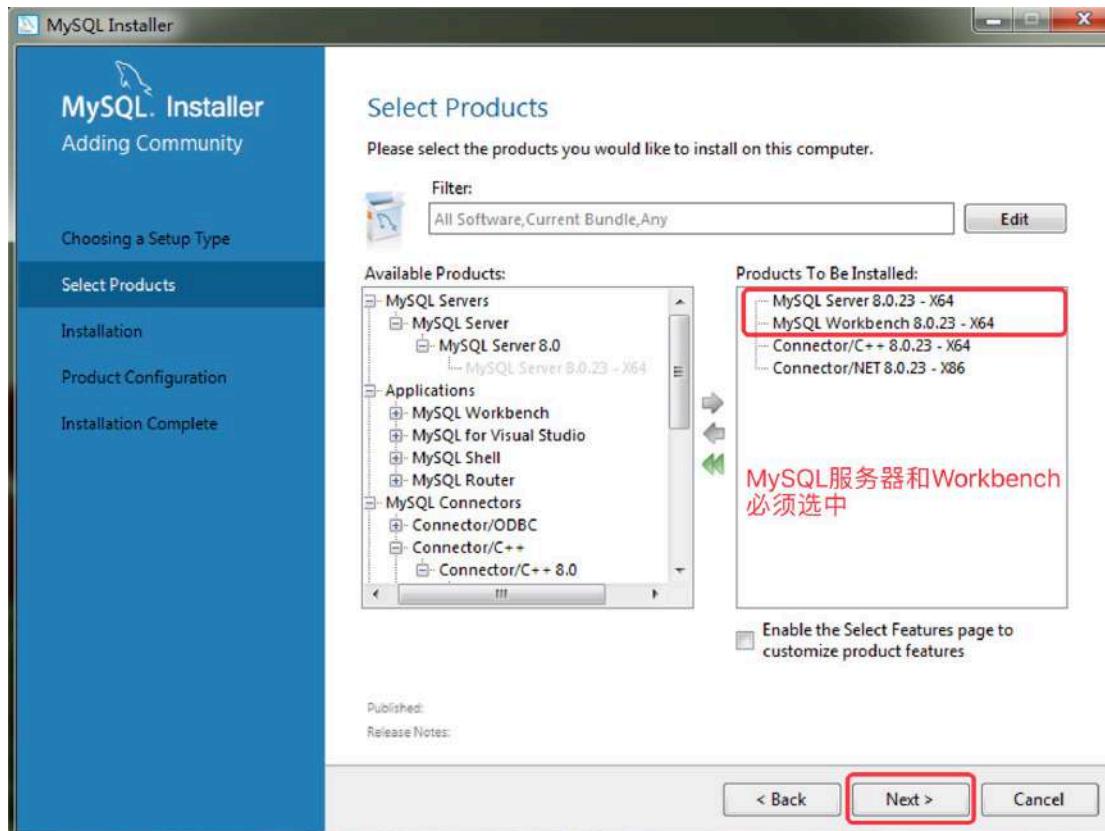


2. 运行 Installer，按照下面的步骤进行安装。

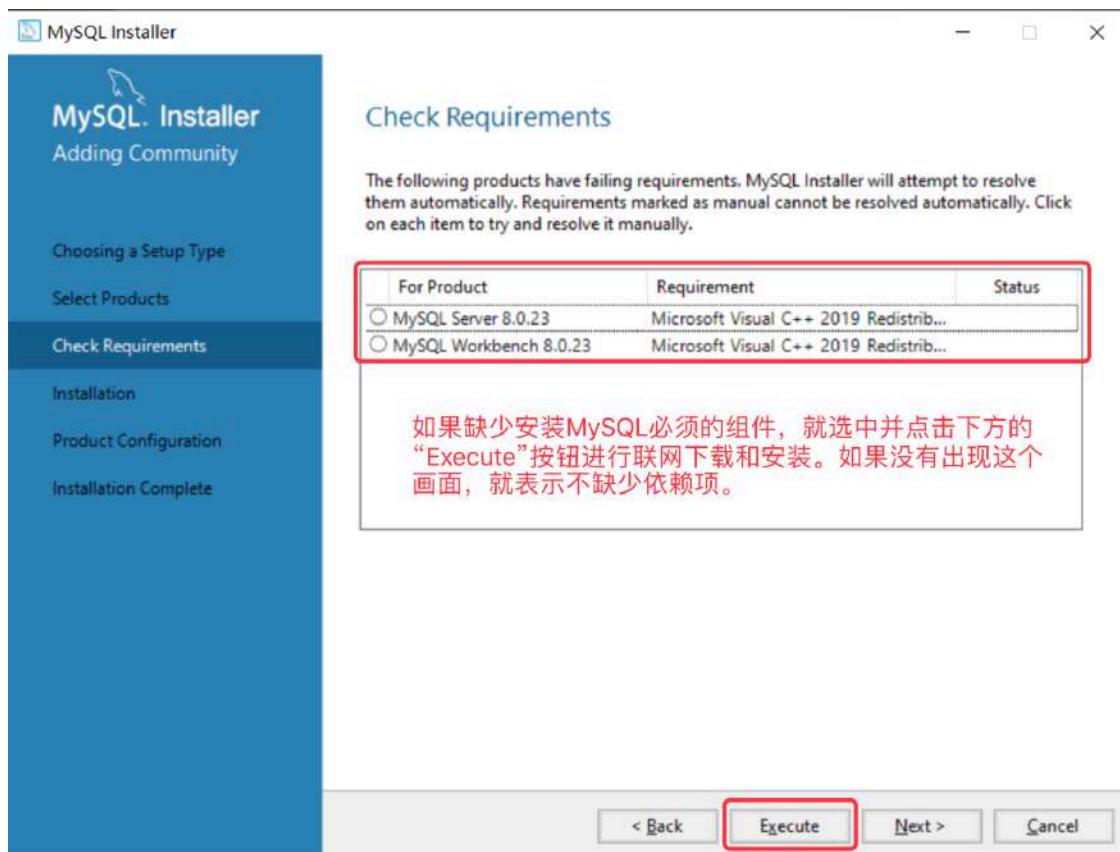
- 选择自定义安装。



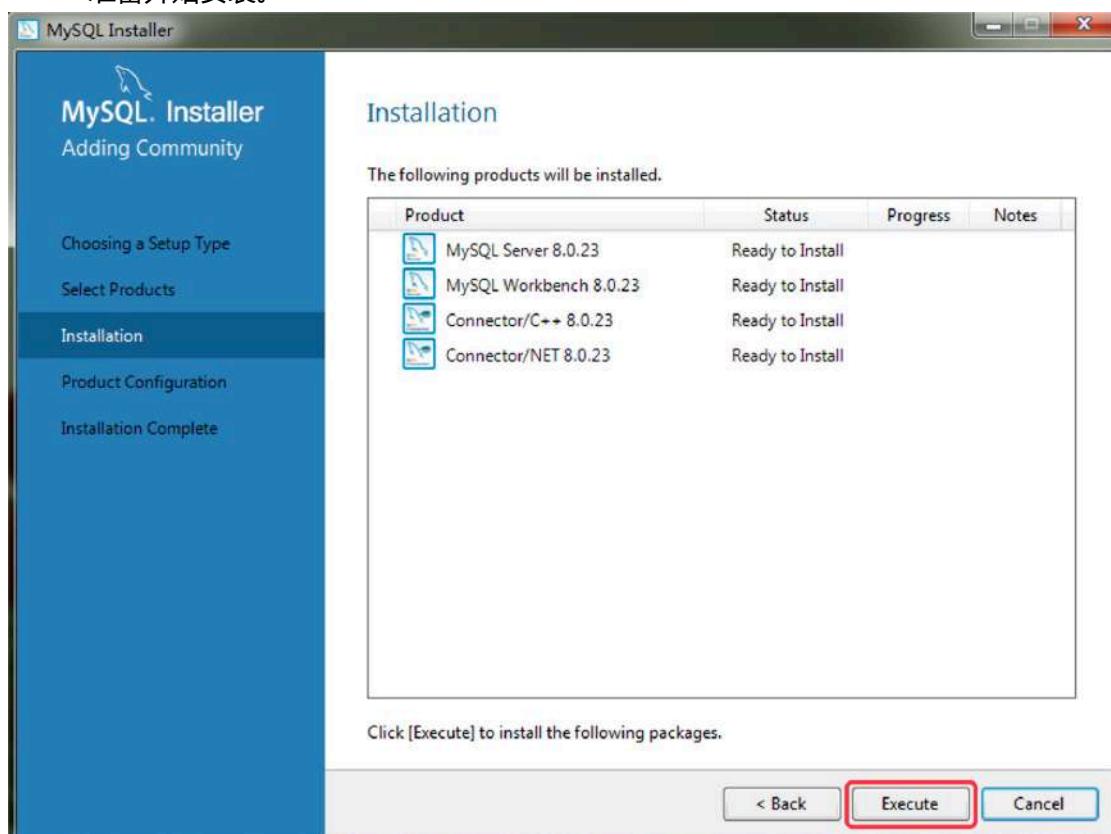
- 选择需要安装的组件。



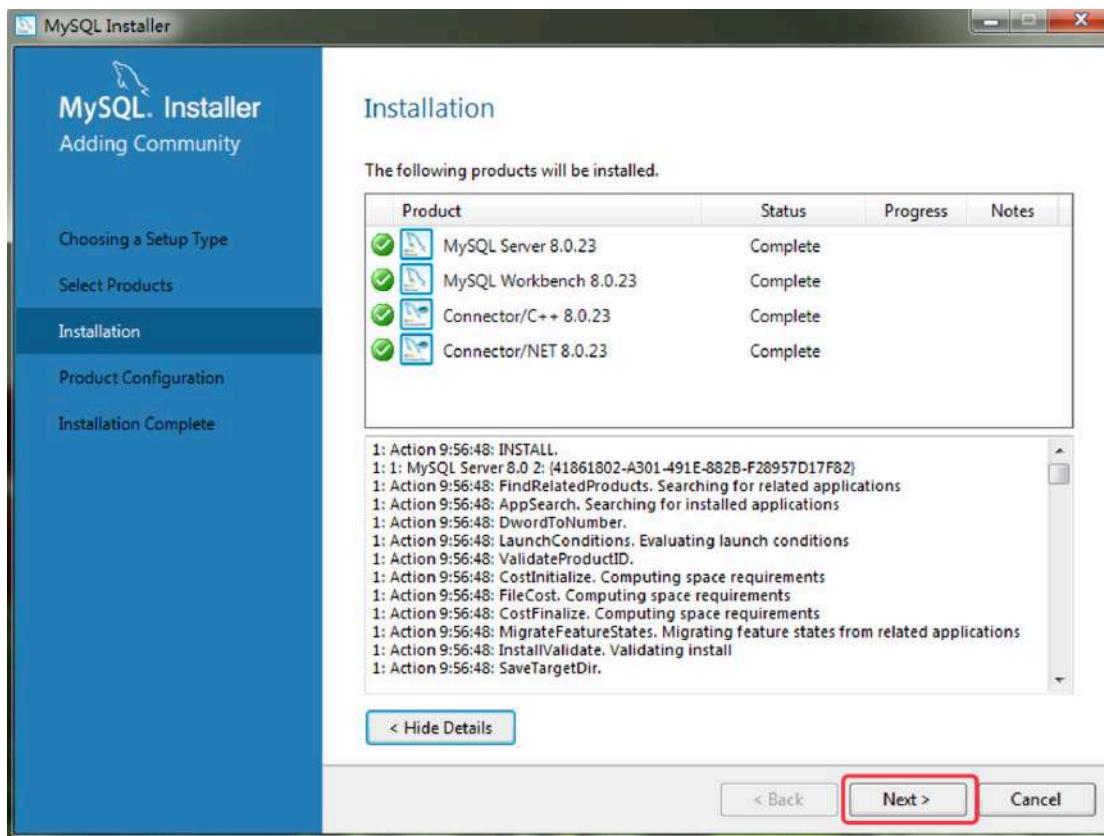
- 如果缺少依赖项，需要先安装依赖项。



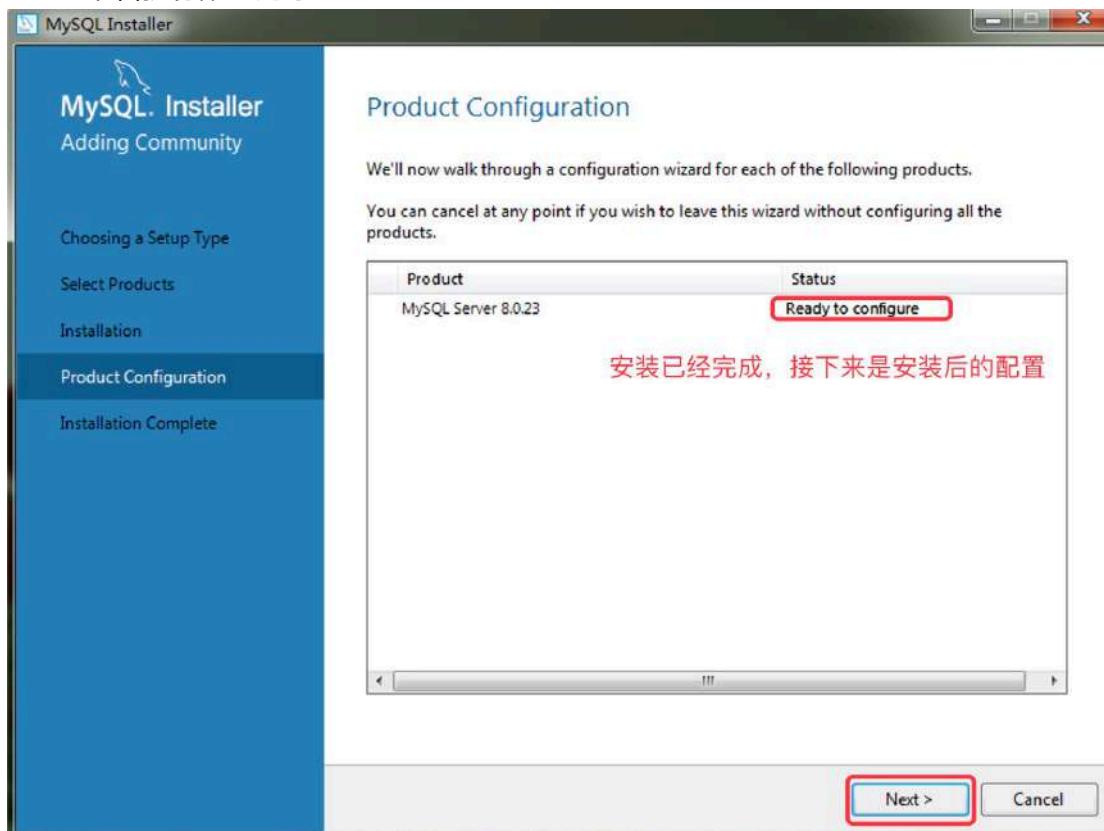
- 准备开始安装。



- 安装完成。

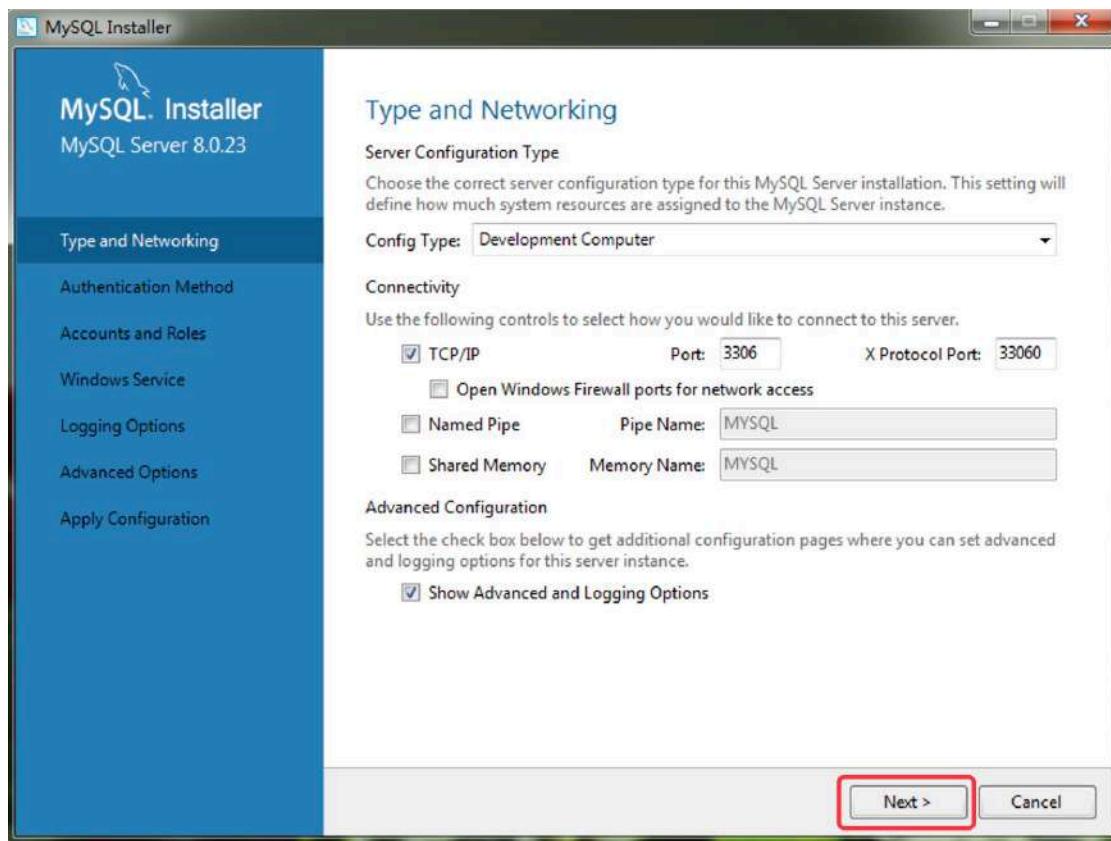


- 准备执行配置向导。

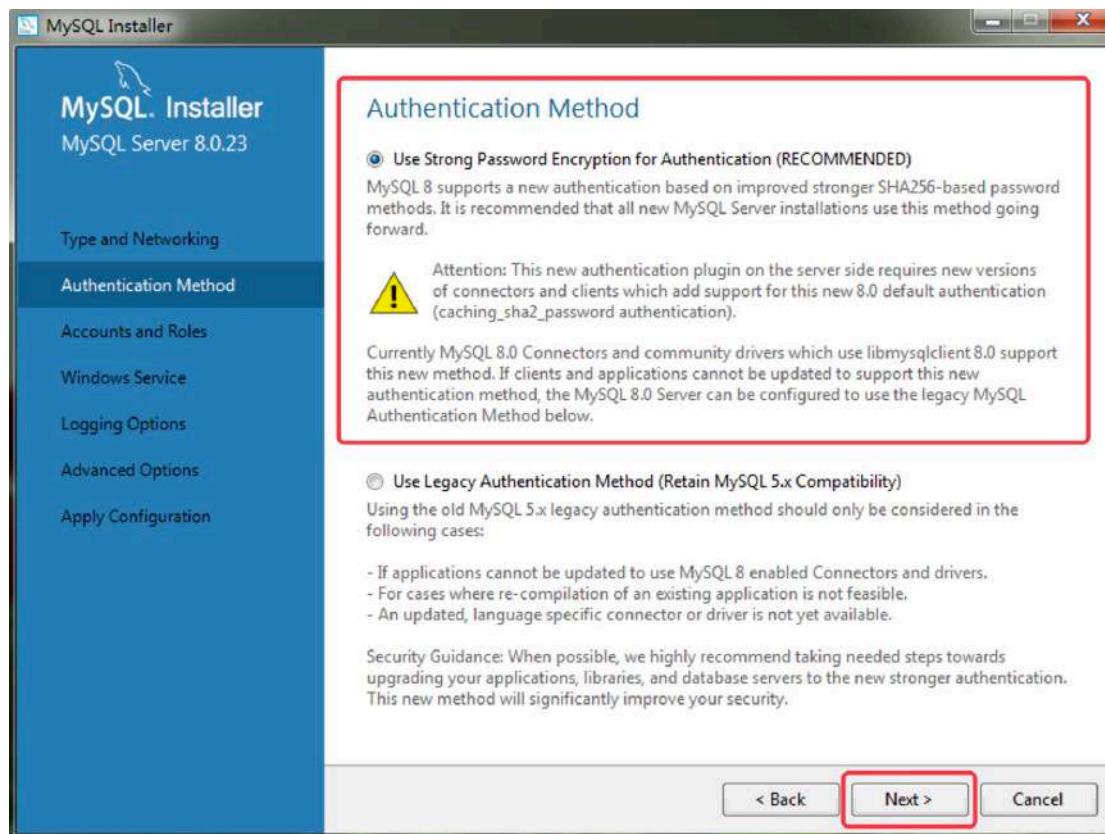


3. 执行安装后的配置向导。

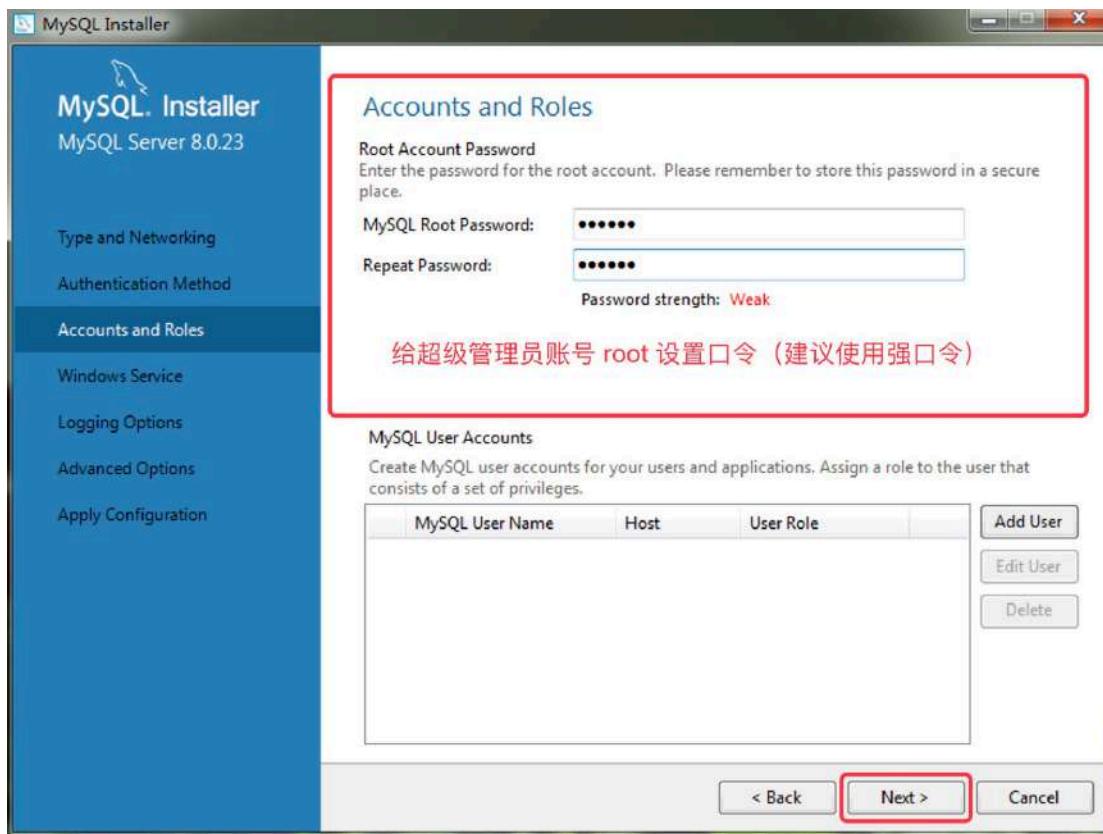
- 配置服务器类型和网络。



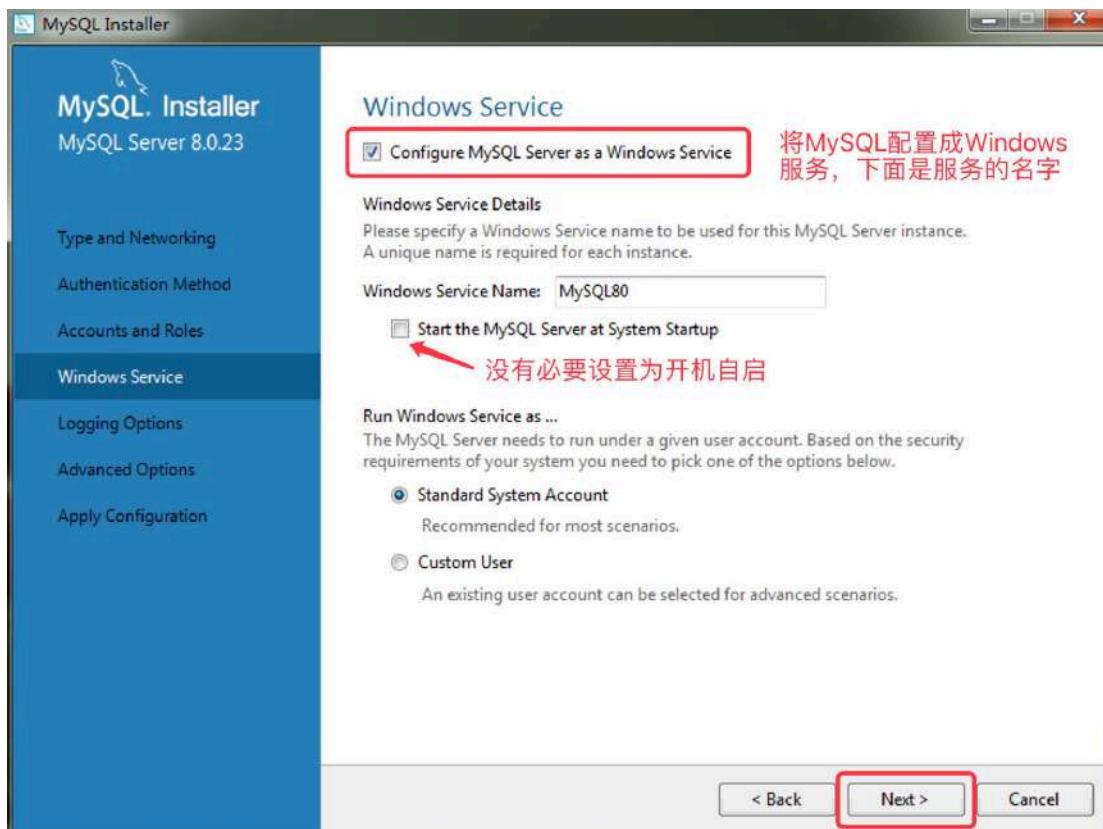
- 配置认证方法（保护密码的方式）。



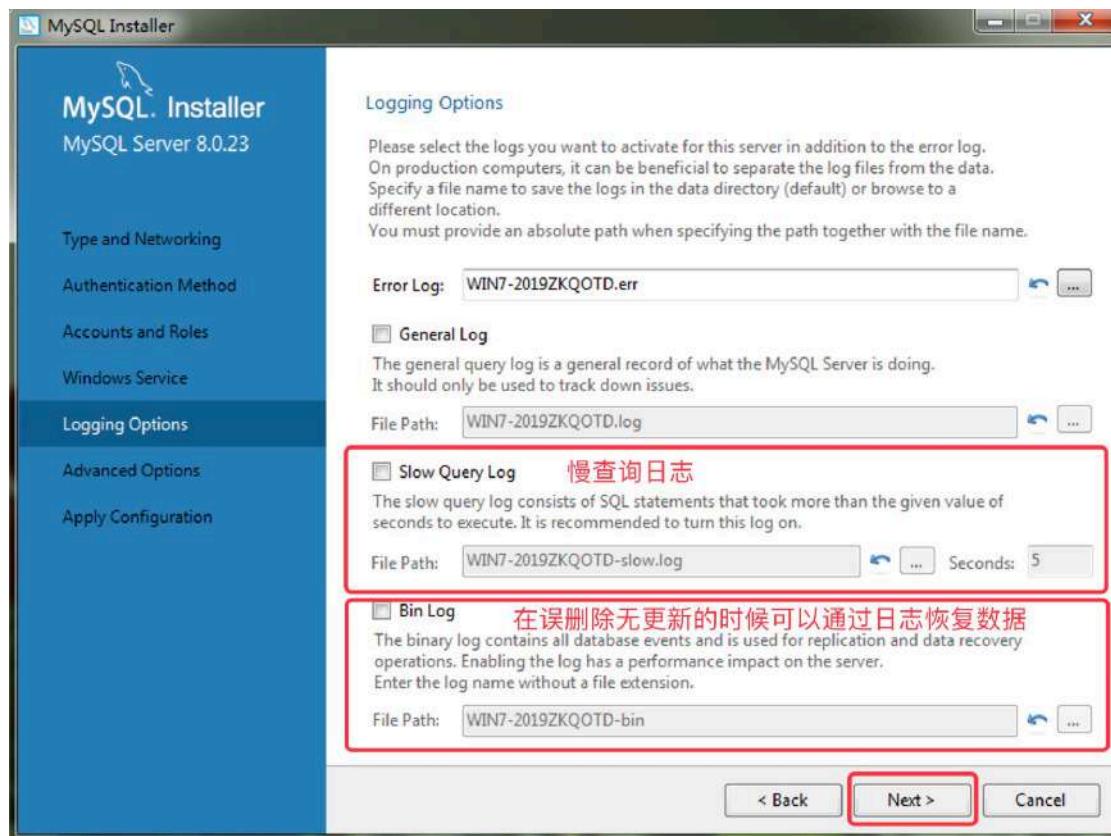
- 配置用户和角色。



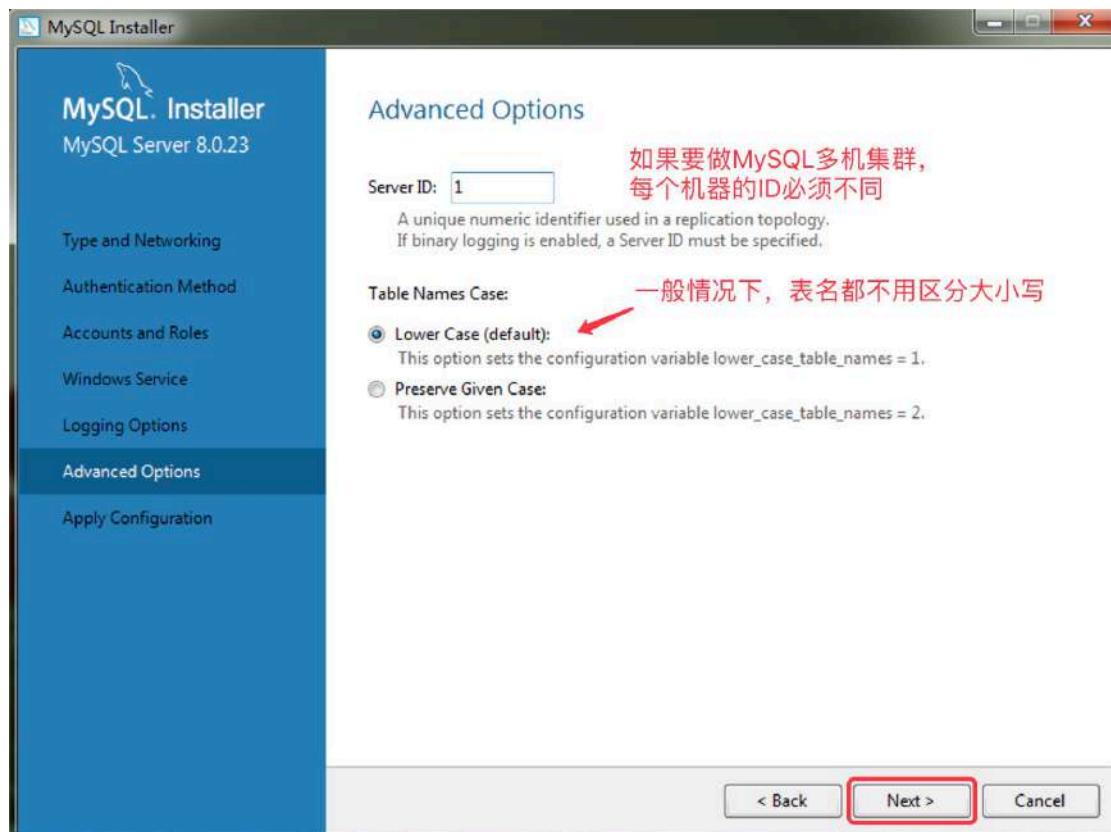
- 配置Windows服务名以及是否开机自启。



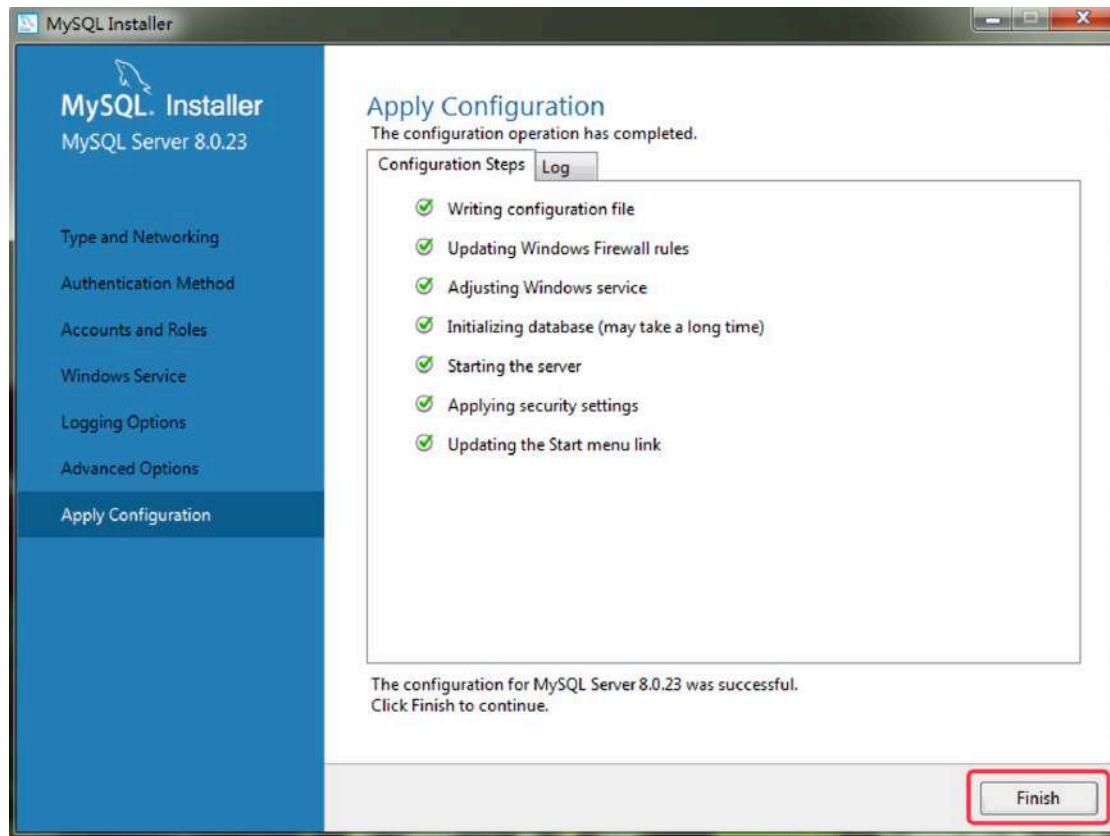
- 配置日志。



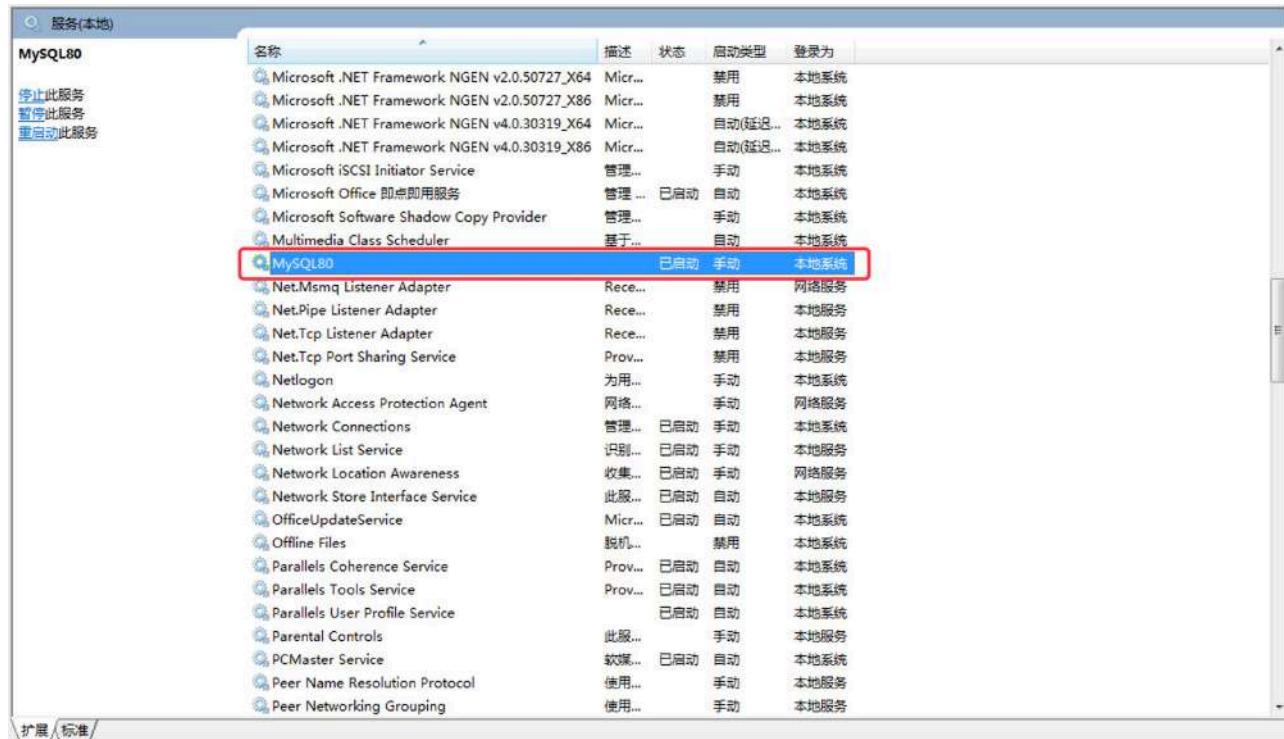
- 配置高级选项。



- 应用配置。



4. 可以在 Windows 系统的“服务”窗口中启动或停止 MySQL。

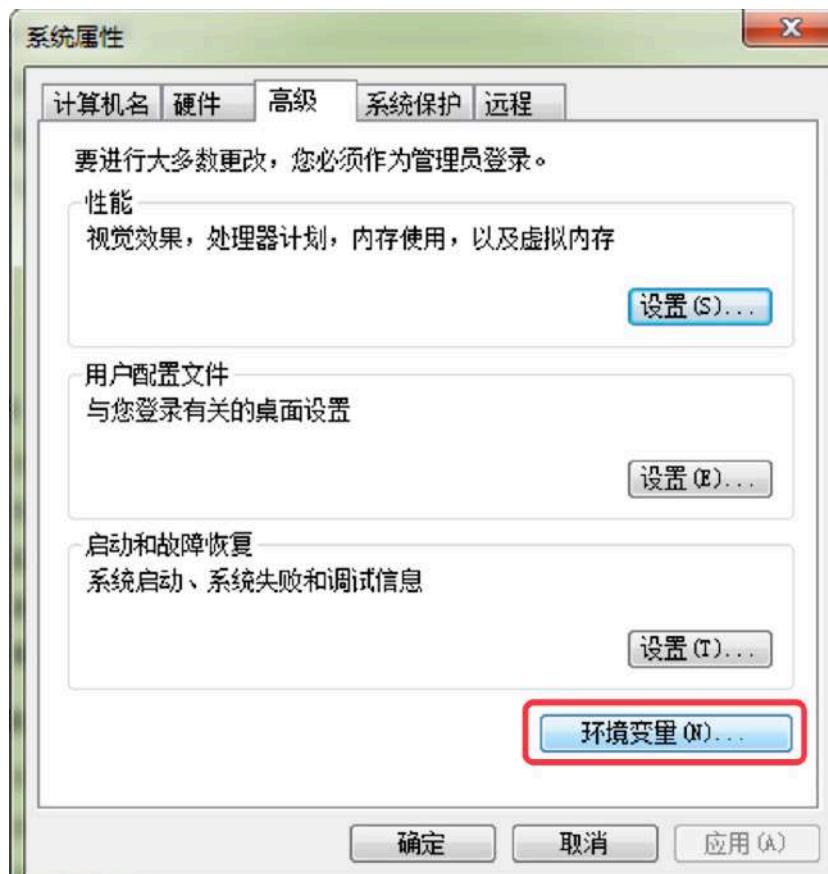


5. 配置 PATH 环境变量，以便在命令行提示符窗口使用 MySQL 客户端工具。

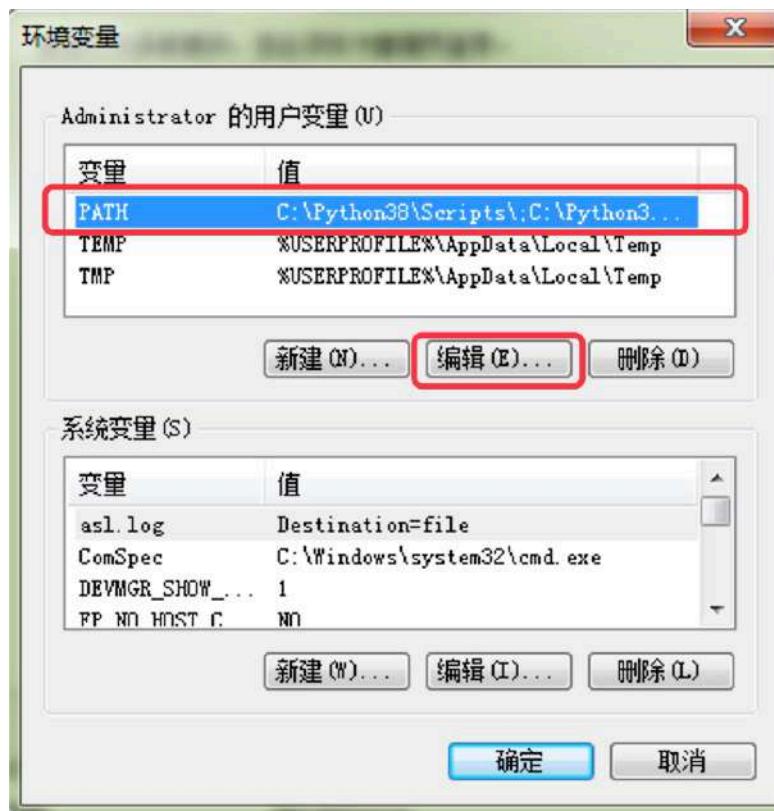
- 打开 Windows 的“系统”窗口并点击“高级系统设置”。



- 在“系统属性”的“高级”窗口，点击“环境变量”按钮。



- 修改PATH环境变量，将MySQL安装路径下的bin文件夹的路径配置到PATH环境变量中。



- 配置完成后，可以尝试在“命令提示符”下使用 MySQL 的命令行工具。

```

Administrator: C:\Windows\system32\cmd.exe - mysql -u root -p

C:\Users\Administrator\Desktop>mysql
ERROR 1045 (28000): Access denied for user 'ODBC'@'localhost' (using password: NO)

C:\Users\Administrator\Desktop>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.23 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _

```

Linux 环境

下面以 CentOS 7.x 环境为例，演示如何安装 MySQL 5.7.x，如果需要在其他 Linux 系统下安装其他版本的 MySQL，请读者自行在网络上查找对应的安装教程。

- 安装 MySQL。

可以在 [MySQL 官方网站](https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-5.7.26-1.el7.x86_64.rpm-bundle.tar) 下载安装文件。首先在下载页面中选择平台和版本，然后找到对应的下载链接，直接下载包含所有安装文件的归档文件，解归档之后通过包管理工具进行安装。

```
wget https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-5.7.26-1.el7.x86_64.rpm-bundle.tar
tar -xvf mysql-5.7.26-1.el7.x86_64.rpm-bundle.tar
```

如果系统上有 MariaDB 相关的文件，需要先移除 MariaDB 相关的文件。

```
yum list installed | grep mariadb | awk '{print $1}' | xargs yum erase -y
```

更新和安装可能用到的底层依赖库。

```
yum update
yum install -y libaio libaio-devel
```

接下来可以按照如下所示的顺序用 RPM (Redhat Package Manager) 工具安装 MySQL。

```
rpm -ivh mysql-community-common-5.7.26-1.el7.x86_64.rpm
rpm -ivh mysql-community-libs-5.7.26-1.el7.x86_64.rpm
rpm -ivh mysql-community-libs-compat-5.7.26-1.el7.x86_64.rpm
rpm -ivh mysql-community-devel-5.7.26-1.el7.x86_64.rpm
rpm -ivh mysql-community-client-5.7.26-1.el7.x86_64.rpm
rpm -ivh mysql-community-server-5.7.26-1.el7.x86_64.rpm
```

可以使用下面的命令查看已经安装的 MySQL 相关的包。

```
rpm -qa | grep mysql
```

2. 配置 MySQL。

MySQL 的配置文件在 `/etc` 目录下，名为 `my.cnf`，默认的配置文件内容如下所示。

```
cat /etc/my.cnf
```

```
# For advice on how to change settings please see
# http://dev.mysql.com/doc/refman/5.7/en/server-configuration-defaults.html

[mysqld]
#
```

```
# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M
#
# Remove leading # to turn on a very important data integrity option:
# logging
# changes to the binary log between backups.
# log_bin
#
# Remove leading # to set options mainly useful for reporting servers.
# The server defaults are faster for transactions and fast SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

通过配置文件，我们可以修改 MySQL 服务使用的端口、字符集、最大连接数、套接字队列大小、最大数据包大小、日志文件的位置、日志过期时间等配置。当然，我们还可以通过修改配置文件来对 MySQL 服务器进行性能调优和安全管控。

3. 启动 MySQL 服务。

可以使用下面的命令来启动 MySQL。

```
service mysqld start
```

在 CentOS 7 中，更推荐使用下面的命令来启动 MySQL。

```
systemctl start mysqld
```

启动 MySQL 成功后，可以通过下面的命令来检查网络端口使用情况，MySQL 默认使用3306端口。

```
netstat -ntlp | grep mysql
```

也可以使用下面的命令查找是否有名为mysql的进程。

```
pgrep mysqld
```

4. 使用 MySQL 客户端工具连接服务器。

命令行工具：

```
mysql -u root -p
```

说明：启动客户端时，`-u`参数用来指定用户名，MySQL 默认的超级管理账号为`root`；`-p`表示要输入密码（用户口令）；如果连接的是其他主机而非本机，可以用`-h`来指定连接主机的主机名或IP地址。

如果是首次安装 MySQL，可以使用下面的命令来找到默认的初始密码。

```
cat /var/log/mysqld.log | grep password
```

上面的命令会查看 MySQL 的日志带有`password`的行，在显示的结果中`root@localhost:`后面的部分就是默认设置的初始密码。

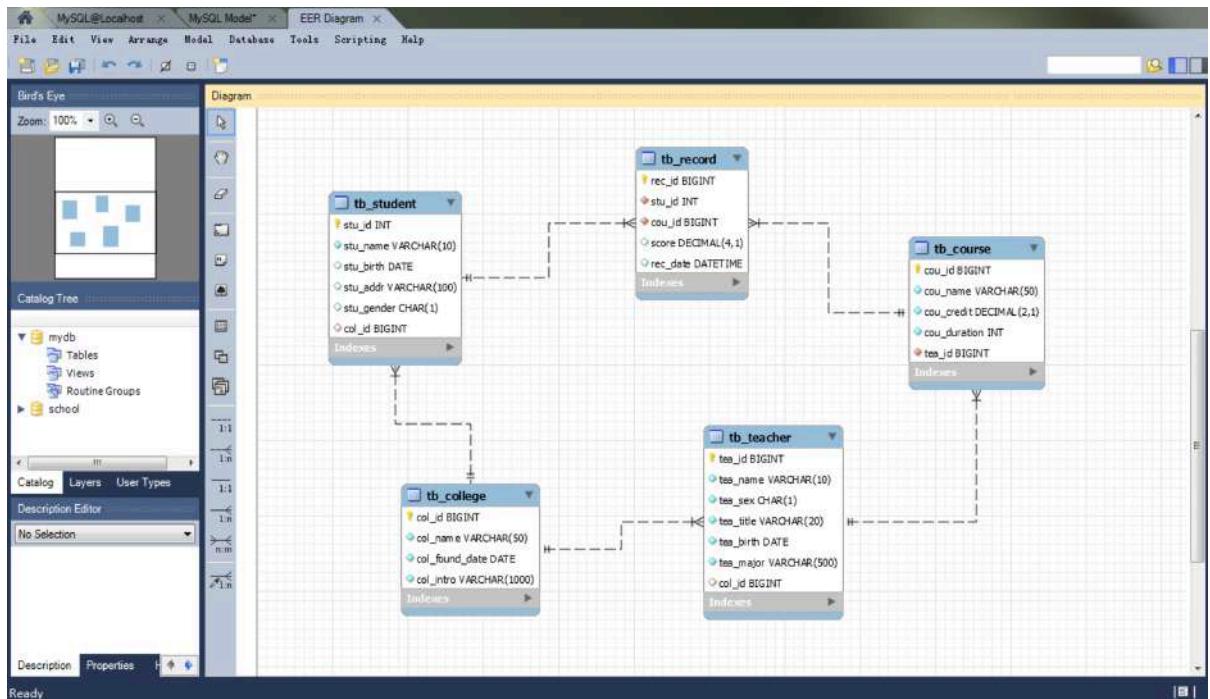
进入客户端工具后，可以通过下面的指令来修改超级管理员（root）的访问口令为`123456`。

```
set global validate_password_policy=0;
set global validate_password_length=6;
alter user 'root'@'localhost' identified by '123456';
```

说明：MySQL 较新的版本默认不允许使用弱口令作为用户口令，所以上面的代码修改了验证用户口令的策略和口令的长度。事实上我们不应该使用弱口令，因为存在用户口令被暴力破解的风险。近年来，**攻击数据库窃取数据和劫持数据库勒索比特币**的事件屡见不鲜，要避免这些潜在的风险，最为重要的一点是**不要让数据库服务器暴露在公网上**（最好的做法是将数据库置于内网，至少要做到不向公网开放数据库服务器的访问端口），另外要保管好`root`账号的口令，应用系统需要访问数据库时，通常不使用`root`账号进行访问，而是**创建其他拥有适当权限的账号来访问**。

再次使用客户端工具连接 MySQL 服务器时，就可以使用新设置的口令了。在实际开发中，为了方便用户操作，可以选择图形化的客户端工具来连接 MySQL 服务器，包括：

- MySQL Workbench（官方工具）



- Navicat for MySQL (界面简单友好)

名	行	数据长度	引擎	创建日期	修改日期	排序规则
tb_college	3	16.00 KB	InnoDB	2021-11-17 15:04:38		utf8mb4
tb_course	9	16.00 KB	InnoDB	2021-11-17 15:04:38		utf8mb4
tb_record	20	16.00 KB	InnoDB	2021-11-17 15:04:38		utf8mb4
tb_student	10	16.00 KB	InnoDB	2021-11-17 15:04:38		utf8mb4
tb_teacher	5	16.00 KB	InnoDB	2021-11-17 15:04:38		utf8mb4

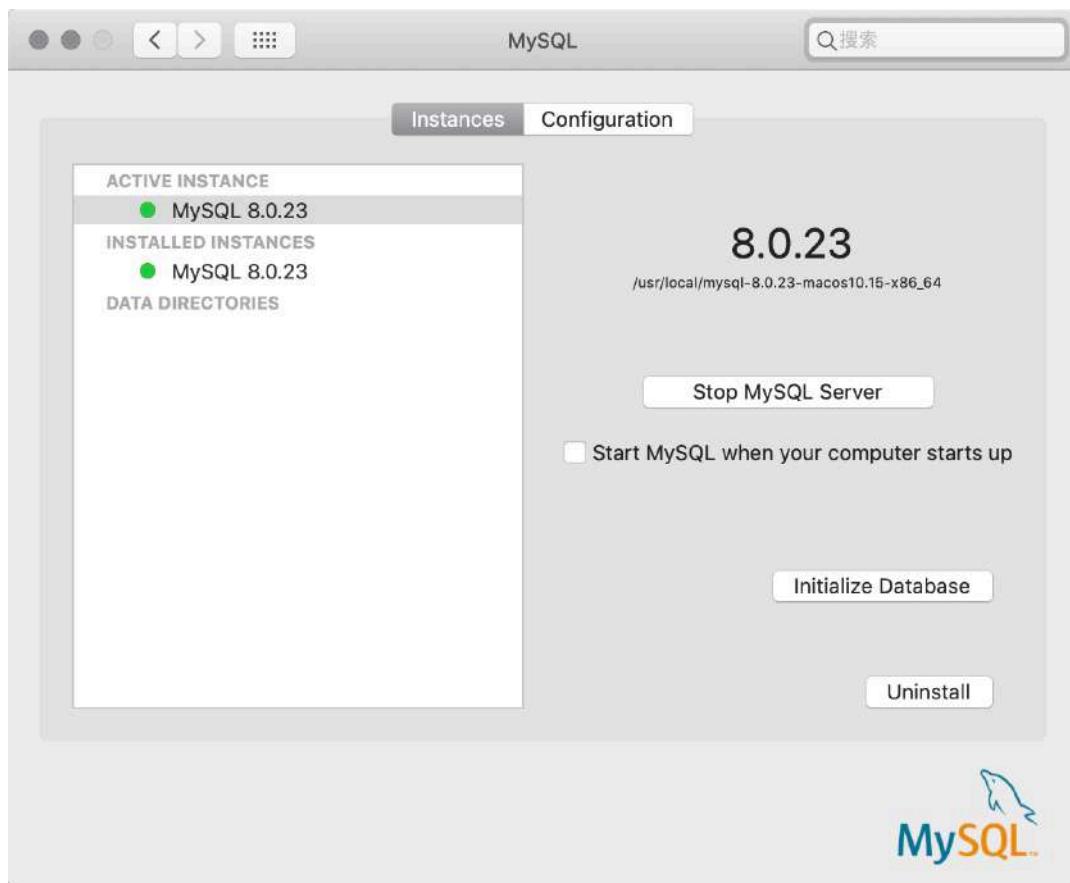
macOS环境

macOS 系统安装 MySQL 是比较简单的，只需要从刚才说到的官方网站下载 DMG 安装文件并运行就可以了，下载的时候需要根据自己使用的是 Intel 的芯片还是苹果的 M1 芯片选择下载链接，如下图所示。

macOS 11 (ARM, 64-bit), DMG Archive (mysql-8.0.27-macos11-arm64.dmg)	8.0.27	416.4M	Download
	CPU - 苹果M1芯片	MD5: 68f0fd1fbd0be128a1cb9463ad7446f7 Signature	

macOS 11 (x86, 64-bit), DMG Archive (mysql-8.0.27-macos11-x86_64.dmg)	8.0.27	422.1M	Download
	CPU - Intel芯片	MD5: a865a7b4455253be0cbd148ca5bf7421 Signature	

安装成功后，可以在“系统偏好设置”中找到“MySQL”，在如下所示的画面中，可以启动和停止 MySQL 服务器，也可以对 MySQL 核心文件的路径进行配置。



MySQL 基本命令

查看命令

1. 查看所有数据库

```
show databases;
```

2. 查看所有字符集

```
show character set;
```

3. 查看所有的排序规则

```
show collation;
```

4. 查看所有的引擎

```
show engines;
```

5. 查看所有日志文件

```
show binary logs;
```

6. 查看数据库下所有表

```
show tables;
```

获取帮助

在 MySQL 命令行工具中，可以使用`help`命令或`?`来获取帮助，如下所示。

1. 查看`show`命令的帮助。

```
? show
```

2. 查看有哪些帮助内容。

```
? contents
```

3. 获取函数的帮助。

```
? functions
```

4. 获取数据类型的帮助。

```
? data types
```

其他命令

1. 新建/重建服务器连接 - `connect` / `resetconnection`。
2. 清空当前输入 - `\c`。在输入错误时，可以及时使用`\c`清空当前输入并重新开始。
3. 修改终止符（定界符） - `delimiter`。默认的终止符是`;`，可以使用该命令修改成其他的字符，例如修改为`$`符号，可以用`delimiter $`命令。
4. 打开系统默认编辑器 - `edit`。编辑完成保存关闭之后，命令行会自动执行编辑的内容。
5. 查看服务器状态 - `status`。
6. 修改默认提示符 - `prompt`。
7. 执行系统命令 - `system`。可以将系统命令跟在`system`命令的后面执行，`system`命令也可以缩写为`\!`。
8. 执行 SQL 文件 - `source`。`source`命令后面跟 SQL 文件路径。
9. 重定向输出 - `tee` / `notee`。可以将命令的输出重定向到指定的文件中。
10. 切换数据库 - `use`。
11. 显示警告信息 - `warnings`。
12. 退出命令行 - `quit`或`exit`。

SQL 和 MySQL 详解

SQL 详解

我们通常可以将 SQL 分为四类，分别是 DDL（数据定义语言）、DML（数据操作语言）、DQL（数据查询语言）和 DCL（数据控制语言）。DDL 主要用于创建、删除、修改数据库中的对象，比如创建、删除和修改二维表，核心的关键字包括 `create`、`drop` 和 `alter`；DML 主要负责数据的插入、删除和更新，关键词包括 `insert`、`delete` 和 `update`；DQL 负责数据查询，最重要的一个关键词是 `select`；DCL 通常用于授予和召回权限，核心关键词是 `grant` 和 `revoke`。

说明：SQL 是不区分大小写的语言，为了书写和识别方便，下面的 SQL 都使用了小写字母来书写。

DDL（数据定义语言）

下面我们来实现一个选课系统的数据库，如下所示的 SQL 创建了名为 `school` 的数据库和五张表，分别是学院表（`tb_college`）、学生表（`tb_student`）、教师表（`tb_teacher`）、课程表（`tb_course`）和选课记录表（`tb_record`），其中学生和教师跟学院之间是多对一关系，课程跟老师之间也是多对一关系，学生和课程是多对多关系，选课记录表就是维持学生跟课程多对多关系的中间表。

```
-- 如果存在名为school的数据库就删除它
drop database if exists `school`;

-- 创建名为school的数据库并设置默认的字符集和排序方式
create database `school` default character set utf8mb4 collate utf8mb4_general_ci;

-- 切换到school数据库上下文环境
use `school`;

-- 创建学院表
create table `tb_college`
(
`col_id` int unsigned auto_increment comment '编号',
`col_name` varchar(50) not null comment '名称',
`col_intro` varchar(500) default '' comment '介绍',
primary key (`col_id`)
) engine=innodb auto_increment=1 comment '学院表';

-- 创建学生表
create table `tb_student`
(
`stu_id` int unsigned not null comment '学号',
`stu_name` varchar(20) not null comment '姓名',
`stu_sex` boolean default 1 not null comment '性别',
`stu_birth` date not null comment '出生日期',
`stu_addr` varchar(255) default '' comment '籍贯',
`col_id` int unsigned not null comment '所属学院',
primary key (`stu_id`),
constraint `fk_student_col_id` foreign key (`col_id`) references `tb_college`(`col_id`)
) engine=innodb comment '学生表';
```

```

-- 创建教师表
create table `tb_teacher`
(
`tea_id` int unsigned not null comment '工号',
`tea_name` varchar(20) not null comment '姓名',
`tea_title` varchar(10) default '助教' comment '职称',
`col_id` int unsigned not null comment '所属学院',
primary key (`tea_id`),
constraint `fk_teacher_col_id` foreign key (`col_id`) references `tb_college`(`col_id`)
) engine=innodb comment '老师表';

-- 创建课程表
create table `tb_course`
(
`cou_id` int unsigned not null comment '编号',
`cou_name` varchar(50) not null comment '名称',
`cou_credit` int not null comment '学分',
`tea_id` int unsigned not null comment '授课老师',
primary key (`cou_id`),
constraint `fk_course_tea_id` foreign key (`tea_id`) references `tb_teacher`(`tea_id`)
) engine=innodb comment '课程表';

-- 创建选课记录表
create table `tb_record`
(
`rec_id` bigint unsigned auto_increment comment '选课记录号',
`stu_id` int unsigned not null comment '学号',
`cou_id` int unsigned not null comment '课程编号',
`sel_date` date not null comment '选课日期',
`score` decimal(4,1) comment '考试成绩',
primary key (`rec_id`),
constraint `fk_record_stu_id` foreign key (`stu_id`) references `tb_student`(`stu_id`),
constraint `fk_record_cou_id` foreign key (`cou_id`) references `tb_course`(`cou_id`),
constraint `uk_record_stu_cou` unique (`stu_id`, `cou_id`)
) engine=innodb comment '选课记录表';

```

上面的DDL有几个地方需要强调一下：

- 创建数据库时，我们通过`default character set utf8mb4`指定了数据库默认使用的字符集为`utf8mb4`（最大4字节的utf-8编码），我们推荐使用该字符集，它也是MySQL 8.x 默认使用的字符集，因为它能够支持国际化编码，还可以存储 Emoji 字符。可以通过下面的命令查看 MySQL 支持的字符集以及默认的排序规则。

```
show character set;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
euckr	EUC-KR Korean	euckr_korean_ci	2
koi8u	KOI8-U Ukrainian	koi8u_general_ci	1
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
greek	ISO 8859-7 Greek	greek_general_ci	1
cp1250	Windows Central European	cp1250_general_ci	1
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
armSCII8	ARMSCII-8 Armenian	armSCII8_general_ci	1
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp866	DOS Russian	cp866_general_ci	1

keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci	1
macce	Mac Central European	macce_general_ci	1
macroman	Mac West European	macroman_general_ci	1
cp852	DOS Central European	cp852_general_ci	1
latin7	ISO 8859-13 Baltic	latin7_general_ci	1
utf8mb4	UTF-8 Unicode	utf8mb4_general_ci	4
cp1251	Windows Cyrillic	cp1251_general_ci	1
utf16	UTF-16 Unicode	utf16_general_ci	4
utf16le	UTF-16LE Unicode	utf16le_general_ci	4
cp1256	Windows Arabic	cp1256_general_ci	1
cp1257	Windows Baltic	cp1257_general_ci	1
utf32	UTF-32 Unicode	utf32_general_ci	4
binary	Binary pseudo charset	binary	1
geostd8	GEOSTD8 Georgian	geostd8_general_ci	1
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3
gb18030	China National Standard GB18030	gb18030_chinese_ci	4
-----+-----+-----+-----			
+			
41 rows in set (0.00 sec)			

如果要设置 MySQL 服务启动时默认使用的字符集，可以修改 MySQL 的配置并添加以下内容。

```
[mysqld]
character-set-server=utf8
```

- 在创建表的时候，可以自行选择底层的存储引擎。MySQL 支持多种存储引擎，可以通过 `show engines` 命令进行查看。MySQL 5.5 以后的版本默认使用的存储引擎是 InnoDB，它是我们推荐大家使用的存储引擎（因为更适合当下互联网应用对高并发、性能以及事务支持等方面的需求），为了 SQL 语句的向下兼容性，我们可以在建表语句结束处右圆括号的后面通过 `engine=innodb` 来指定使用 InnoDB 存储引擎。

```
show engines\G
```

```
***** 1. row *****
  Engine: InnoDB
  Support: DEFAULT
  Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
  XA: YES
  Savepoints: YES
***** 2. row *****
  Engine: MRG_MYISAM
  Support: YES
  Comment: Collection of identical MyISAM tables
Transactions: NO
  XA: NO
  Savepoints: NO
***** 3. row *****
  Engine: MEMORY
  Support: YES
  Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
  XA: NO
  Savepoints: NO
***** 4. row *****
  Engine: BLACKHOLE
  Support: YES
  Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
  XA: NO
  Savepoints: NO
***** 5. row *****
  Engine: MyISAM
  Support: YES
  Comment: MyISAM storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 6. row *****
  Engine: CSV
  Support: YES
  Comment: CSV storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
***** 7. row *****
  Engine: ARCHIVE
  Support: YES
  Comment: Archive storage engine
Transactions: NO
  XA: NO
```

```

Savepoints: NO
***** 8. row *****
  Engine: PERFORMANCE_SCHEMA
  Support: YES
  Comment: Performance Schema
Transactions: NO
  XA: NO
  Savepoints: NO
***** 9. row *****
  Engine: FEDERATED
  Support: NO
  Comment: Federated MySQL storage engine
Transactions: NULL
  XA: NULL
  Savepoints: NULL
9 rows in set (0.00 sec)

```

下面的表格对MySQL几种常用的数据引擎进行了简单的对比。

特性	InnoDB	MRG_MYISAM	MEMORY	MyISAM
存储限制	有	没有	有	有
事务	支持			
锁机制	行锁	表锁	表锁	表锁
B树索引	支持	支持	支持	支持
哈希索引			支持	
全文检索	支持 (5.6+)			支持
集群索引	支持			
数据缓存	支持		支持	
索引缓存	支持	支持	支持	支持
数据可压缩				支持
内存使用	高	低	中	低
存储空间使用	高	低		低
批量插入性能	低	高	高	高
是否支持外键	支持			

通过上面的比较我们可以了解到，InnoDB 是唯一能够支持外键、事务以及行锁的存储引擎，所以我们之前说它更适合互联网应用，而且在较新版本的 MySQL 中，它也是默认使用的存储引擎。

- 在定义表结构为每个字段选择数据类型时，如果不清楚哪个数据类型更合适，可以通过 MySQL 的帮助系统来了解每种数据类型的特性、数据的长度和精度等相关信息。

```
? data types
```

You asked for help about help category: "Data Types"
For more information, type 'help <item>', where <item> is one of the
following
topics:

- AUTO_INCREMENT
- BIGINT
- BINARY
- BIT
- BLOB
- BLOB DATA TYPE
- BOOLEAN
- CHAR
- CHAR BYTE
- DATE
- DATETIME
- DEC
- DECIMAL
- DOUBLE
- DOUBLE PRECISION
- ENUM
- FLOAT
- INT
- INTEGER
- LONGBLOB
- LONGTEXT
- MEDIUMBLOB
- MEDIUMINT
- MEDIUMTEXT
- SET DATA TYPE
- SMALLINT
- TEXT
- TIME
- TIMESTAMP
- TINYBLOB
- TINYINT
- TINYTEXT
- VARBINARY
- VARCHAR
- YEAR DATA TYPE

```
? varchar
```

Name: 'VARCHAR'

Description:

```
[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE
collation_name]
```

A variable-length string. M represents the maximum column length in characters. The range of M is 0 to 65,535. The effective maximum length of a VARCHAR is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, utf8 characters can require up to three bytes per character, so a VARCHAR column that uses the utf8 character set can be declared to be a maximum of 21,844 characters. See <http://dev.mysql.com/doc/refman/5.7/en/column-count-limit.html>.

MySQL stores VARCHAR values as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A VARCHAR column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

Note:

MySQL follows the standard SQL specification, and does not remove trailing spaces from VARCHAR values.

VARCHAR is shorthand for CHARACTER VARYING. NATIONAL VARCHAR is the standard SQL way to define that a VARCHAR column should use some predefined character set. MySQL uses utf8 as this predefined character set. <http://dev.mysql.com/doc/refman/5.7/en/charset-national.html>. NVARCHAR is shorthand for NATIONAL VARCHAR.

URL: <http://dev.mysql.com/doc/refman/5.7/en/string-type-overview.html>

在数据类型的选择上，保存字符串数据通常都使用VARCHAR和CHAR两种类型，前者通常称为变长字符串，而后者通常称为定长字符串；对于 InnoDB 存储引擎，行存储格式没有区分固定长度和可变长度列，因此VARCHAR类型和CHAR类型没有本质区别，后者不一定比前者性能更好。如果要保存的很大字符串，可以使用TEXT类型；如果要保存很大的字节串，可以使用BLOB（二进制大对象）类型。在 MySQL 中，TEXT和BLOB又分别包括TEXT、MEDIUMTEXT、LONGTEXT和BLOB、MEDIUMBLOB、LONGBLOB三种不同的类型，它们主要的区别在于存储数据的最大大小不同。保存浮点数可以用FLOAT或DOUBLE类型，FLOAT已经不推荐使用了，而且在 MySQL 后续的版本中可能会被移除掉。而保存定点数应该使用DECIMAL类型。如果要保存时间日期，DATETIME类型优于TIMESTAMP类型，因为前者能表示的时间日期范围更大。

DML (数据操作语言)

我们通过如下所示的 SQL 给上面创建的表添加数据。

```
use school;

-- 插入学院数据
insert into `tb_college`
(`col_name`, `col_intro`)
values
```

('计算机学院', '计算机学院1958年设立计算机专业, 1981年建立计算机科学系, 1998年设立计算机学院, 2005年5月, 为了进一步整合教学和科研资源, 学校决定, 计算机学院和软件学院行政班子合并统一运作、实行教学和学生管理独立运行的模式。学院下设三个系: 计算机科学与技术系、物联网工程系、计算金融系; 两个研究所: 图象图形研究所、网络空间安全研究院 (2015年成立); 三个教学实验中心: 计算机基础教学实验中心、IBM技术中心和计算机专业实验中心。'),

('外国语学院', '外国语学院设有7个教学单位, 6个文理兼收的本科专业; 拥有1个一级学科博士授予点, 3个二级学科博士授予点, 5个一级学科硕士学位授权点, 5个二级学科硕士学位授权点, 5个硕士专业授权领域, 同时还有2个硕士专业学位 (MTI) 专业; 有教职员210余人, 其中教授、副教授80余人, 教师中获得中国国内外名校博士学位和正在攻读博士学位的教师比例占专任教师的60%以上。'),

('经济管理学院', '经济学院前身是创办于1905年的经济科; 已故经济学家彭迪先、张与九、蒋学模、胡寄窗、陶大镛、胡代光, 以及当代学者刘诗白等曾先后在此任教或学习。');

-- 插入学生数据

```
insert into `tb_student`
(`stu_id`, `stu_name`, `stu_sex`, `stu_birth`, `stu_addr`, `col_id`)
values
(1001, '杨过', 1, '1990-3-4', '湖南长沙', 1),
(1002, '任我行', 1, '1992-2-2', '湖南长沙', 1),
(1033, '王语嫣', 0, '1989-12-3', '四川成都', 1),
(1572, '岳不群', 1, '1993-7-19', '陕西咸阳', 1),
(1378, '纪嫣然', 0, '1995-8-12', '四川绵阳', 1),
(1954, '林平之', 1, '1994-9-20', '福建莆田', 1),
(2035, '东方不败', 1, '1988-6-30', null, 2),
(3011, '林震南', 1, '1985-12-12', '福建莆田', 3),
(3755, '项少龙', 1, '1993-1-25', '四川成都', 3),
(3923, '杨不悔', 0, '1985-4-17', '四川成都', 3);
```

-- 插入老师数据

```
insert into `tb_teacher`
(`tea_id`, `tea_name`, `tea_title`, `col_id`)
values
(1122, '张三丰', '教授', 1),
(1133, '宋远桥', '副教授', 1),
(1144, '杨逍', '副教授', 1),
(2255, '范遥', '副教授', 2),
(3366, '韦一笑', default, 3);
```

-- 插入课程数据

```
insert into `tb_course`
(`cou_id`, `cou_name`, `cou_credit`, `tea_id`)
values
(1111, 'Python程序设计', 3, 1122),
(2222, 'Web前端开发', 2, 1122),
(3333, '操作系统', 4, 1122),
(4444, '计算机网络', 2, 1133),
(5555, '编译原理', 4, 1144),
(6666, '算法和数据结构', 3, 1144),
(7777, '经贸法语', 3, 2255),
(8888, '成本会计', 2, 3366),
(9999, '审计学', 3, 3366);
```

-- 插入选课数据

```
insert into `tb_record`
```

```
(`stu_id`, `cou_id`, `sel_date`, `score`)
values
  (1001, 1111, '2017-09-01', 95),
  (1001, 2222, '2017-09-01', 87.5),
  (1001, 3333, '2017-09-01', 100),
  (1001, 4444, '2018-09-03', null),
  (1001, 6666, '2017-09-02', 100),
  (1002, 1111, '2017-09-03', 65),
  (1002, 5555, '2017-09-01', 42),
  (1033, 1111, '2017-09-03', 92.5),
  (1033, 4444, '2017-09-01', 78),
  (1033, 5555, '2017-09-01', 82.5),
  (1572, 1111, '2017-09-02', 78),
  (1378, 1111, '2017-09-05', 82),
  (1378, 7777, '2017-09-02', 65.5),
  (2035, 7777, '2018-09-03', 88),
  (2035, 9999, '2019-09-02', null),
  (3755, 1111, '2019-09-02', null),
  (3755, 8888, '2019-09-02', null),
  (3755, 9999, '2017-09-01', 92);
```

注意：上面的`insert`语句使用了批处理的方式来插入数据，这种做法插入数据的效率比较高。

DQL (数据查询语言)

接下来，我们完成如下所示的查询。

```
-- 查询所有学生的所有信息
select * from `tb_student`;

-- 查询学生的学号、姓名和籍贯(投影)
select `stu_id`, `stu_name`, `stu_addr` from `tb_student`;

-- 查询所有课程的名称及学分(投影和别名)
select `cou_name` as 课程名称, `cou_credit` as 学分 from `tb_course`;

-- 查询所有女学生的姓名和出生日期(筛选)
select `stu_name`, `stu_birth` from `tb_student` where `stu_sex`=0;

-- 查询籍贯为“四川成都”的女学生的姓名和出生日期(筛选)
select `stu_name`, `stu_birth` from `tb_student` where `stu_sex`=0 and
`stu_addr`='四川成都';

-- 查询籍贯为“四川成都”或者性别为“女生”的学生
select `stu_name`, `stu_birth` from `tb_student` where `stu_sex`=0 or
`stu_addr`='四川成都';

-- 查询所有80后学生的姓名、性别和出生日期(筛选)
select `stu_name`, `stu_sex`, `stu_birth` from `tb_student` where
`stu_birth`>='1980-1-1' and `stu_birth`<='1989-12-31';
```

```

select `stu_name`, `stu_sex`, `stu_birth` from `tb_student`
where `stu_birth` between '1980-1-1' and '1989-12-31';

-- 补充: 将表示性别的 1 和 0 处理成“男”和“女”
select
  `stu_name` as 姓名,
  if(`stu_sex`, '男', '女') as 性别,
  `stu_birth` as 出生日期
from `tb_student`
where `stu_birth` between '1980-1-1' and '1989-12-31';

select
  `stu_name` as 姓名,
  case `stu_sex` when 1 then '男' else '女' end as 性别,
  `stu_birth` as 出生日期
from `tb_student`
where `stu_birth` between '1980-1-1' and '1989-12-31';

-- 查询学分大于2的课程的名称和学分(筛选)
select `cou_name`, `cou_credit` from `tb_course` where `cou_credit` > 2;

-- 查询学分是奇数的课程的名称和学分(筛选)
select `cou_name`, `cou_credit` from `tb_course` where `cou_credit` % 2 <> 0;

select `cou_name`, `cou_credit` from `tb_course` where `cou_credit` mod 2 <> 0;

-- 查询选择了1111的课程考试成绩在90分以上的学号(筛选)
select `stu_id` from `tb_record` where `cou_id` = 1111 and `score` > 90;

-- 查询名字叫“杨过”的学生的姓名和性别
select `stu_name`, `stu_sex` from `tb_student` where `stu_name` = '杨过';

-- 查询姓“杨”的学生姓名和性别(模糊)
-- % - 通配符 (wildcard) , 它可以匹配0个或任意多个字符
select `stu_name`, `stu_sex` from `tb_student` where `stu_name` like '杨%';

-- 查询姓“杨”名字两个字的学生姓名和性别(模糊)
-- _ - 通配符 (wildcard) , 它可以精确匹配一个字符
select `stu_name`, `stu_sex` from `tb_student` where `stu_name` like '杨_';

-- 查询姓“杨”名字三个字的学生姓名和性别(模糊)
select `stu_name`, `stu_sex` from `tb_student` where `stu_name` like '杨__';

-- 查询名字中有“不”字或“嫣”字的学号(模糊)
select `stu_name` from `tb_student` where `stu_name` like '%不%' or `stu_name` like '%嫣%';

-- 将“岳不群”改名为“岳不嫣”，比较下面两个查询的区别
update `tb_student` set `stu_name` = '岳不嫣' where `stu_id` = 1572;

select `stu_name` from `tb_student` where `stu_name` like '%不%'
union
select `stu_name` from `tb_student` where `stu_name` like '%嫣%';

```

```

select `stu_name` from `tb_student` where `stu_name` like '%不%'
union all
select `stu_name` from `tb_student` where `stu_name` like '%嫣%';

-- 查询姓“杨”或姓“林”名字三个字的学生的姓名(正则表达式模糊查询)
select `stu_name` from `tb_student` where `stu_name` regexp '[杨林].{2}';

-- 查询没有录入籍贯的学生姓名(空值处理)
select `stu_name` from `tb_student` where `stu_addr` is null;

select `stu_name` from `tb_student` where `stu_addr` <= null;

-- 查询录入了籍贯的学生姓名(空值处理)
select `stu_name` from `tb_student` where `stu_addr` is not null;

-- 下面的查询什么也查不到, 三值逻辑 --> true / false / unknown
select `stu_name` from `tb_student` where `stu_addr` = null or `stu_addr` <> null;

-- 查询学生选课的所有日期(去重)
select distinct `sel_date` from `tb_record`;

-- 查询学生的籍贯(去重)
select distinct `stu_addr` from `tb_student` where `stu_addr` is not null;

-- 查询男学生的姓名和生日按年龄从大到小排列(排序)
-- 升序: 从小到大 - asc, 降序: 从大到小 - desc
select `stu_id`, `stu_name`, `stu_birth` from `tb_student`
where `stu_sex` = 1 order by `stu_birth` asc, `stu_id` desc;

-- 补充: 将上面的生日换算成年龄(日期函数、数值函数)
select
  `stu_id` as 学号,
  `stu_name` as 姓名,
  floor(datediff(curdate(), `stu_birth`)/365) as 年龄
from `tb_student`
where `stu_sex` = 1 order by 年龄 desc, `stu_id` desc;

-- 查询年龄最大的学生的出生日期(聚合函数)
select min(`stu_birth`) from `tb_student`;

-- 查询年龄最小的学生的出生日期(聚合函数)
select max(`stu_birth`) from `tb_student`;

-- 查询编号为1111的课程考试成绩的最高分(聚合函数)
select max(`score`) from `tb_record` where `cou_id` = 1111;

-- 查询学号为1001的学生考试成绩的最低分(聚合函数)
select min(`score`) from `tb_record` where `stu_id` = 1001;

-- 查询学号为1001的学生考试成绩的平均分(聚合函数)
select avg(`score`) from `tb_record` where `stu_id` = 1001;

select sum(`score`) / count(`score`) from `tb_record` where `stu_id` = 1001;

```

```

-- 查询学号为1001的学生考试成绩的平均分,如果有null值, null值算0分(聚合函数)
select sum(`score`) / count(*) from `tb_record` where `stu_id`=1001;

select avg(ifnull(`score`, 0)) from `tb_record` where `stu_id`=1001;

-- 查询学号为1001的学生考试成绩的标准差(聚合函数)
select std(`score`), variance(`score`) from `tb_record` where `stu_id`=1001;

-- 查询男女学生的人数(分组和聚合函数)
select
    case `stu_sex` when 1 then '男' else '女' end as 性别,
    count(*) as 人数
from `tb_student` group by `stu_sex`;

-- 查询每个学院学生人数(分组和聚合函数)
select
    `col_id` as 学院,
    count(*) as 人数
from `tb_student` group by `col_id` with rollup;

-- 查询每个学院男女学生人数(分组和聚合函数)
select
    `col_id` as 学院,
    if(`stu_sex`, '男', '女') as 性别,
    count(*) as 人数
from `tb_student` group by `col_id`, `stu_sex`;

-- 查询每个学生的学号和平均成绩(分组和聚合函数)
select
    `stu_id`,
    round(avg(`score`), 1) as avg_score
from `tb_record` group by `stu_id`;

-- 查询平均成绩大于等于90分的学生的学号和平均成绩
-- 分组以前的筛选使用where子句, 分组以后的筛选使用having子句
select
    `stu_id`,
    round(avg(`score`), 1) as avg_score
from `tb_record`
group by `stu_id` having avg_score>=90;

-- 查询1111、2222、3333三门课程平均成绩大于等于90分的学生的学号和平均成绩
select
    `stu_id`,
    round(avg(`score`), 1) as avg_score
from `tb_record` where `cou_id` in (1111, 2222, 3333)
group by `stu_id` having avg_score>=90;

-- 查询年龄最大的学生的姓名(子查询/嵌套查询)
-- 嵌套查询: 把一个select的结果作为另一个select的一部分来使用
select `stu_name` from `tb_student`
where `stu_birth`=
    select min(`stu_birth`) from `tb_student`;
);

```

-- 查询选了两门以上的课程的学生姓名(子查询/分组条件/集合运算)

```
select `stu_name` from `tb_student`  
where `stu_id` in (  
    select `stu_id` from `tb_record`  
    group by `stu_id` having count(*)>2  
);
```

-- 查询学生的姓名、生日和所在学院名称

```
select `stu_name`, `stu_birth`, `col_name`  
from `tb_student`, `tb_college`  
where `tb_student`.`col_id`=`tb_college`.`col_id`;
```

```
select `stu_name`, `stu_birth`, `col_name`  
from `tb_student` inner join `tb_college`  
on `tb_student`.`col_id`=`tb_college`.`col_id`;
```

```
select `stu_name`, `stu_birth`, `col_name`  
from `tb_student` natural join `tb_college`;
```

-- 查询学生姓名、课程名称以及成绩(连接查询/联结查询)

```
select `stu_name`, `cou_name`, `score`  
from `tb_student`, `tb_course`, `tb_record`  
where `tb_student`.`stu_id`=`tb_record`.`stu_id`  
and `tb_course`.`cou_id`=`tb_record`.`cou_id`  
and `score` is not null;
```

```
select `stu_name`, `cou_name`, `score` from `tb_student`  
inner join `tb_record` on `tb_student`.`stu_id`=`tb_record`.`stu_id`  
inner join `tb_course` on `tb_course`.`cou_id`=`tb_record`.`cou_id`  
where `score` is not null;
```

```
select `stu_name`, `cou_name`, `score` from `tb_student`  
natural join `tb_record`  
natural join `tb_course`  
where `score` is not null;
```

-- 补充: 上面的查询结果取前5条数据(分页查询)

```
select `stu_name`, `cou_name`, `score`  
from `tb_student`, `tb_course`, `tb_record`  
where `tb_student`.`stu_id`=`tb_record`.`stu_id`  
and `tb_course`.`cou_id`=`tb_record`.`cou_id`  
and `score` is not null  
order by `score` desc  
limit 0,5;
```

-- 补充: 上面的查询结果取第6-10条数据(分页查询)

```
select `stu_name`, `cou_name`, `score`  
from `tb_student`, `tb_course`, `tb_record`  
where `tb_student`.`stu_id`=`tb_record`.`stu_id`  
and `tb_course`.`cou_id`=`tb_record`.`cou_id`  
and `score` is not null  
order by `score` desc  
limit 5 offset 5;
```

```

-- 补充: 上面的查询结果取第11-15条数据(分页查询)
select `stu_name`, `cou_name`, `score`
from `tb_student`, `tb_course`, `tb_record`
where `tb_student`.`stu_id`=`tb_record`.`stu_id`
and `tb_course`.`cou_id`=`tb_record`.`cou_id`
and `score` is not null
order by `score` desc
limit 5 offset 10;

-- 查询选课学生的姓名和平均成绩(子查询和连接查询)
select `stu_name`, `avg_score`
from `tb_student` inner join (
    select `stu_id` as `sid`, round(avg(`score`), 1) as avg_score
    from `tb_record` group by `stu_id`
) as `t2` on `stu_id`=`sid`;

-- 查询学生的姓名和选课的数量
select `stu_name`, `total` from `tb_student` as `t1`
inner join (
    select `stu_id`, count(*) as `total`
    from `tb_record` group by `stu_id`
) as `t2` on `t1`.`stu_id`=`t2`.`stu_id`;

-- 查询每个学生的姓名和选课数量(左外连接和子查询)
-- 左外连接: 左表 (写在join左边的表) 的每条记录都可以查出来, 不满足连表条件的地方填充 null。
select `stu_name`, coalesce(`total`, 0) as `total`
from `tb_student` as `t1`
left outer join (
    select `stu_id`, count(*) as `total`
    from `tb_record` group by `stu_id`
) as `t2` on `t1`.`stu_id`=`t2`.`stu_id`;

-- 修改选课记录表, 去掉 stu_id 列的外键约束
alter table `tb_record` drop foreign key `fk_record_stu_id`;

-- 插入两条新纪录 (注意: 没有学号为 5566 的学生)
insert into `tb_record`
values
    (default, 5566, 1111, '2019-09-02', 80),
    (default, 5566, 2222, '2019-09-02', 70);

-- 右外连接: 右表 (写在join右边的表) 的每条记录都可以查出来, 不满足连表条件的地方填充 null。
select `stu_name`, `total` from `tb_student` as `t1`
right outer join (
    select `stu_id`, count(*) as `total`
    from `tb_record` group by `stu_id`
) as `t2` on `t1`.`stu_id`=`t2`.`stu_id`;

-- 全外连接: 左表和右表的每条记录都可以查出来, 不满足连表条件的地方填充null。
-- 说明: MySQL不支持全外连接, 所以用左外连接和右外连接的并集来表示。
select `stu_name`, `total`

```

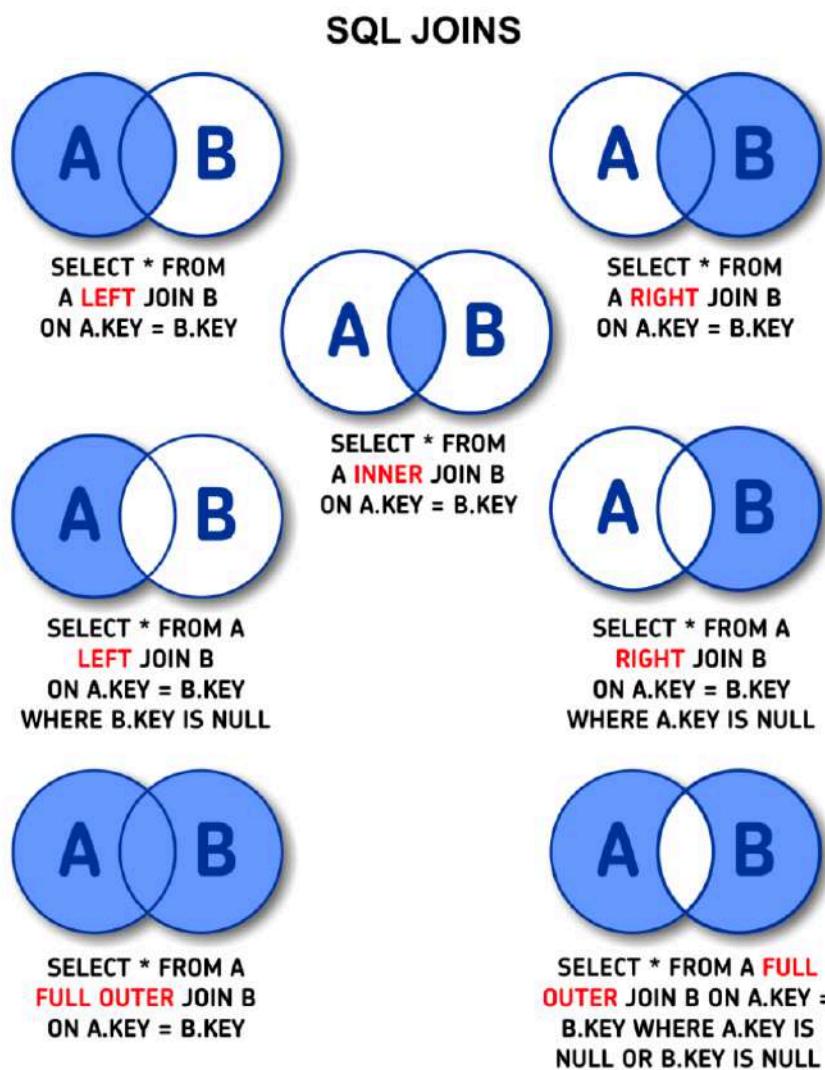
```

from `tb_student` as `t1`
left outer join (
    select `stu_id`, count(*) as `total`
    from `tb_record` group by `stu_id`
) as `t2` on `t1`.`stu_id` = `t2`.`stu_id`
union
select `stu_name`, `total` from `tb_student` as `t1`
right outer join (
    select `stu_id`, count(*) as `total`
    from `tb_record` group by `stu_id`
) as `t2` on `t1`.`stu_id` = `t2`.`stu_id`;

```

上面的DML有几个地方需要加以说明：

- MySQL目前的版本不支持全外连接，上面我们通过union操作，将左外连接和右外连接的结果求并集实现全外连接的效果。大家可以通过下面的图来加深对连表操作的认识。



- MySQL 中支持多种类型的运算符，包括：算术运算符 (+、-、*、/、%)、比较运算符 (=、<、<=、>、>=、BETWEEN...AND...)、IN、IS NULL、IS NOT NULL、LIKE、RLIKE、REGEXP）、逻辑运算符 (NOT、AND、OR、XOR) 和位运算符 (&、|、^、~、>>、<<)，我们可以在 DML 中使用这些运算符处理数据。

3. 在查询数据时，可以在SELECT语句及其子句（如WHERE子句、ORDER BY子句、HAVING子句等）中使用函数，这些函数包括字符串函数、数值函数、时间日期函数、流程函数等，如下面的表格所示。

常用字符串函数。

函数	功能
CONCAT	将多个字符串连接成一个字符串
FORMAT	将数值格式化成字符串并指定保留几位小数
FROM_BASE64 / TO_BASE64	BASE64解码/编码
BIN / OCT / HEX	将数值转换成二进制/八进制/十六进制字符串
LOCATE	在字符串中查找一个子串的位置
LEFT / RIGHT	返回一个字符串左边/右边指定长度的字符
LENGTH / CHAR_LENGTH	返回字符串的长度以字节/字符为单位
LOWER / UPPER	返回字符串的小写/大写形式
LPAD / RPAD	如果字符串的长度不足，在字符串左边/右边填充指定的字符
LTRIM / RTRIM	去掉字符串前面/后面的空格
ORD / CHAR	返回字符对应的编码/返回编码对应的字符
STRCMP	比较字符串，返回-1、0、1分别表示小于、等于、大于
SUBSTRING	返回字符串指定范围的子串

常用数值函数。

函数	功能
ABS	返回一个数的绝对值
CEILING / FLOOR	返回一个数上取整/下取整的结果
CONV	将一个数从一种进制转换成另一种进制
CRC32	计算循环冗余校验码
EXP / LOG / LOG2 / LOG10	计算指数/对数
POW	求幂
RAND	返回[0,1)范围的随机数
ROUND	返回一个数四舍五入后的结果
SQRT	返回一个数的平方根
TRUNCATE	截断一个数到指定的精度
SIN / COS / TAN / COT / ASIN / ACOS / ATAN	三角函数

常用时间日期函数。

函数	功能
<code>CURDATE / CURTIME / NOW</code>	获取当前日期/时间/日期和时间
<code>ADDDATE / SUBDATE</code>	将两个日期表达式相加/相减并返回结果
<code>DATE / TIME</code>	从字符串中获取日期/时间
<code>YEAR / MONTH / DAY</code>	从日期中获取年/月/日
<code>HOUR / MINUTE / SECOND</code>	从时间中获取时/分/秒
<code>DATEDIFF / TIMEDIFF</code>	返回两个时间日期表达式相差多少天/小时
<code>MAKEDATE / MAKETIME</code>	制造一个日期/时间

常用流程函数。

函数	功能
<code>IF</code>	根据条件是否成立返回不同的值
<code>IFNULL</code>	如果为NULL则返回指定的值否则就返回本身
<code>NULLIF</code>	两个表达式相等就返回NULL否则返回第一个表达式的值

其他常用函数。

函数	功能
<code>MD5 / SHA1 / SHA2</code>	返回字符串对应的哈希摘要
<code>CHARSET / COLLATION</code>	返回字符集/校对规则
<code>USER / CURRENT_USER</code>	返回当前用户
<code>DATABASE</code>	返回当前数据库名
<code>VERSION</code>	返回当前数据库版本
<code>FOUND_ROWS / ROW_COUNT</code>	返回查询到的行数/受影响的行数
<code>LAST_INSERT_ID</code>	返回最后一个自增主键的值
<code>UUID / UUID_SHORT</code>	返回全局唯一标识符

DCL (数据控制语言)

数据控制语言用于给指定的用户授权或者从召回指定用户的指定权限，这组操作对数据库管理员来说比较重要，将一个用户的权限最小化（刚好够用）是非常重要的，对数据库的安全至关重要。

```
-- 创建名为 wangdachui 的账号并为其指定口令，允许该账号从任意主机访问
create user 'wangdachui'@'%' identified by '123456';
```

```
-- 授权 wangdachui 可以对名为school的数据库执行 select 和 insert 操作
grant select, insert on `school`.* to 'wangdachui'@'%';

-- 召回 wangdachui 对school数据库的 insert 权限
revoke insert on `school`.* from 'wangdachui'@'%';
```

说明：创建一个可以允许任意主机登录并且具有超级管理员权限的用户在现实中并不是一个明智的决定，因为一旦该账号的口令泄露或者被破解，数据库将会面临灾难级的风险。

MySQL 详解

索引

索引是关系型数据库中用来提升查询性能最为重要的手段。关系型数据库中的索引就像一本书的目录，我们可以想象一下，如果要从一本书中找出某个知识点，但是这本书没有目录，这将是意见多么可怕的事情！我们估计得一篇一篇的翻下去，才能确定这个知识点到底在什么位置。创建索引虽然会带来存储空间上的开销，就像一本书的目录会占用一部分篇幅一样，但是在牺牲空间后换来的查询时间的减少也是非常显著的。

MySQL 数据库中所有数据类型的列都可以被索引。对于MySQL 8.0 版本的 InnoDB 存储引擎来说，它支持三种类型的索引，分别是 B+ 树索引、全文索引和 R 树索引。这里，我们只介绍使用得最为广泛的 B+ 树索引。使用 B+ 树的原因非常简单，因为它是目前在基于磁盘进行海量数据存储和排序上最有效率的数据结构。B+ 树是一棵**平衡树**，树的高度通常为3或4，但是却可以保存从百万级到十亿级的数据，而从这些数据里面查询一条数据，只需要3次或4次 I/O 操作。

B+ 树由根节点、中间节点和叶子节点构成，其中叶子节点用来保存排序后的数据。由于记录在索引上是排序过的，因此在一个叶子节点内查找数据时可以使用二分查找，这种查找方式效率非常的高。当数据很少的时候，B+ 树只有一个根节点，数据也就保存在根节点上。随着记录越来越多，B+ 树会发生分裂，根节点不再保存数据，而是提供了访问下一层节点的指针，帮助快速确定数据在哪个叶子节点上。

在创建二维表时，我们通常都会为表指定主键列，主键列上默认会创建索引，而对于 MySQL InnoDB 存储引擎来说，因为它使用的是索引组织表这种数据存储结构，所以主键上的索引就是整张表的数据，而这种索引我们也将其称之为**聚集索引** (clustered index)。很显然，一张表只能有一个聚集索引，否则表的数据岂不是要保存多次。我们自己创建的索引都是二级索引 (secondary index)，更常见的叫法是**非聚集索引** (non-clustered index)。通过我们自定义的非聚集索引只能定位记录的主键，在获取数据时可能需要再通过主键上的聚集索引进行查询，这种现象称为“回表”，因此通过非聚集索引检索数据通常比使用聚集索引检索数据要慢。

接下来我们通过一个简单的例子来说明索引的意义，比如我们要根据学生的姓名来查找学生，这个场景在实际开发中应该经常遇到，就跟通过商品名称查找商品是一个道理。我们可以使用 MySQL 的**explain**关键字来查看 SQL 的执行计划（数据库执行 SQL 语句的具体步骤）。

```
explain select * from tb_student where stuname='林震南'\G
```

```
***** 1. row *****
id: 1
select_type: SIMPLE
table: tb_student
partitions: NULL
```

```

    type: ALL
possible_keys: NULL
    key: NULL
key_len: NULL
    ref: NULL
    rows: 11
filtered: 10.00
    Extra: Using where
1 row in set, 1 warning (0.00 sec)

```

在上面的 SQL 执行计划中，有几项值得我们关注：

1. **select_type**：查询的类型。
 - **SIMPLE**：简单 SELECT，不需要使用 UNION 操作或子查询。
 - **PRIMARY**：如果查询包含子查询，最外层的 SELECT 被标记为 PRIMARY。
 - **UNION**：UNION 操作中第二个或后面的 SELECT 语句。
 - **SUBQUERY**：子查询中的第一个 SELECT。
 - **DERIVED**：派生表的 SELECT 子查询。
2. **table**：查询对应的表。
3. **type**：MySQL 在表中找到满足条件的行的方式，也称为访问类型，包括：**ALL**（全表扫描）、**index**（索引全扫描，只遍历索引树）、**range**（索引范围扫描）、**ref**（非唯一索引扫描）、**eq_ref**（唯一索引扫描）、**const / system**（常量级查询）、**NULL**（不需要访问表或索引）。在所有的访问类型中，很显然 ALL 是性能最差的，它代表的全表扫描是指要扫描表中的每一行才能找到匹配的行。
4. **possible_keys**：MySQL 可以选择的索引，但是**有可能不会使用**。
5. **key**：MySQL 真正使用的索引，如果为**NULL**就表示没有使用索引。
6. **key_len**：使用的索引的长度，在不影响查询的情况下肯定是长度越短越好。
7. **rows**：执行查询需要扫描的行数，这是一个**预估值**。
8. **extra**：关于查询额外的信息。
 - **Using filesort**：MySQL 无法利用索引完成排序操作。
 - **Using index**：只使用索引的信息而不需要进一步查表来获取更多的信息。
 - **Using temporary**：MySQL 需要使用临时表来存储结果集，常用于分组和排序。
 - **Impossible where**：**where**子句会导致没有符合条件的行。
 - **Distinct**：MySQL 发现第一个匹配行后，停止为当前的行组合搜索更多的行。
 - **Using where**：查询的列未被索引覆盖，筛选条件并不是索引的前导列。

从上面的执行计划可以看出，当我们通过学生名字查询学生时实际上是进行了全表扫描，不言而喻这个查询性能肯定是非常糟糕的，尤其是在表中的行很多的时候。如果我们需要经常通过学生姓名来查询学生，那么就应该在学生姓名对应的列上创建索引，通过索引来加速查询。

```
create index idx_student_name on tb_student(stuname);
```

再次查看刚才的 SQL 对应的执行计划。

```
explain select * from tb_student where stuname='林震南'\G
```

```
***** 1. row *****
    id: 1
  select_type: SIMPLE
      table: tb_student
    partitions: NULL
        type: ref
possible_keys: idx_student_name
      key: idx_student_name
    key_len: 62
        ref: const
      rows: 1
  filtered: 100.00
    Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

可以注意到，在对学生姓名创建索引后，刚才的查询已经不是全表扫描而是基于索引的查询，而且扫描的行只有唯一的一行，这显然大大的提升了查询的性能。MySQL 中还允许创建前缀索引，即对索引字段的前N个字符创建索引，这样的话可以减少索引占用的空间（但节省了空间很有可能会浪费时间，**时间和空间是不可调和的矛盾**），如下所示。

```
create index idx_student_name_1 on tb_student(stuname(1));
```

上面的索引相当于是根据学生姓名的第一个字来创建的索引，我们再看看 SQL 执行计划。

```
explain select * from tb_student where stuname='林震南'\G
```

```
***** 1. row *****
    id: 1
  select_type: SIMPLE
      table: tb_student
    partitions: NULL
        type: ref
possible_keys: idx_student_name
      key: idx_student_name
    key_len: 5
        ref: const
      rows: 2
  filtered: 100.00
    Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

不知道大家是否注意到，这一次扫描的行变成了2行，因为学生表中有两个姓“林”的学生，我们只用姓名的第一个字作为索引的话，在查询时通过索引就会找到这两行。

如果要删除索引，可以使用下面的SQL。

```
alter table tb_student drop index idx_student_name;
```

或者

```
drop index idx_student_name on tb_student;
```

在创建索引时，我们还可以使用复合索引、函数索引（MySQL 5.7 开始支持），用好复合索引实现索引覆盖可以减少不必要的排序和回表操作，这样就会让查询的性能成倍的提升，有兴趣的读者可以自行研究。

我们简单的为大家总结一下索引的设计原则：

1. **最适合索引的列是出现在WHERE子句和连接子句中的列。**
2. 索引列的基数越大（取值多、重复值少），索引的效果就越好。
3. 使用**前缀索引**可以减少索引占用的空间，内存中可以缓存更多的索引。
4. **索引不是越多越好**，虽然索引加速了读操作（查询），但是写操作（增、删、改）都会变得更慢，因为数据的变化会导致索引的更新，就如同书籍章节的增删需要更新目录一样。
5. 使用 InnoDB 存储引擎时，表的普通索引都会保存主键的值，所以**主键要尽可能选择较短的数据类型**，这样可以有效的减少索引占用的空间，提升索引的缓存效果。

最后，还有一点需要说明，InnoDB 使用的 B-tree 索引，数值类型的列除了等值判断时索引会生效之外，使用 **>、<、>=、<=、BETWEEN...AND...、<>** 时，索引仍然生效；对于字符串类型的列，如果使用不以通配符开头的模糊查询，索引也是起作用的，但是其他的情况会导致索引失效，这就意味着很有可能会做全表查询。

视图

视图是关系型数据库中将一组查询指令构成的结果集组合成可查询的数据表的对象。简单的说，视图就是虚拟的表，但与数据表不同的是，数据表是一种实体结构，而视图是一种虚拟结构，你也可以将视图理解为保存在数据库中被赋予名字的 SQL 语句。

使用视图可以获得以下好处：

1. 可以将实体数据表隐藏起来，让外部程序无法得知实际的数据结构，让访问者可以使用表的组成部分而不是整个表，降低数据库被攻击的风险。
2. 在大多数的情况下视图是只读的（更新视图的操作通常都有诸多的限制），外部程序无法直接透过视图修改数据。
3. 重用 SQL 语句，将高度复杂的查询包装在视图表中，直接访问该视图即可取出需要的数据；也可以将视图视为数据表进行连接查询。
4. 视图可以返回与实体数据表不同格式的数据，在创建视图的时候可以对数据进行格式化处理。

创建视图。

```
-- 创建视图
create view `vw_avg_score`
as
```

```

select `stu_id`, round(avg(`score`), 1) as `avg_score`
from `tb_record` group by `stu_id`;

-- 基于已有的视图创建视图
create view `vw_student_score`
as
  select `stu_name`, `avg_score`
  from `tb_student` natural join `vw_avg_score`;

```

提示：因为视图不包含数据，所以每次使用视图时，都必须执行查询以获得数据，如果你使用了连接查询、嵌套查询创建了较为复杂的视图，你可能会发现查询性能下降得很厉害。因此，在使用复杂的视图前，应该进行测试以确保其性能能够满足应用的需求。

使用视图。

```
select * from `vw_student_score` order by `avg_score` desc;
```

stuname	avgscore
杨过	95.6
任我行	53.5
王语嫣	84.3
纪嫣然	73.8
岳不群	78.0
东方不败	88.0
项少龙	92.0

既然视图是一张虚拟的表，那么视图中的数据可以更新吗？视图的可更新性要视具体情况而定，以下类型的视图是不能更新的：

1. 使用了聚合函数（SUM、MIN、MAX、AVG、COUNT等）、DISTINCT、GROUP BY、HAVING、UNION或者UNION ALL的视图。
2. SELECT中包含了子查询的视图。
3. FROM子句中包含了一个不能更新的视图的视图。
4. WHERE子句的子查询引用了FROM子句中的表的视图。

删除视图。

```
drop view vw_student_score;
```

说明：如果希望更新视图，可以先用上面的命令删除视图，也可以通过create or replace view来更新视图。

视图的规则和限制。

1. 视图可以嵌套，可以利用从其他视图中检索的数据来构造一个新的视图。视图也可以和表一起使用。
2. 创建视图时可以使用`order by`子句，但如果从视图中检索数据时也使用了`order by`，那么该视图中原先的`order by`会被覆盖。
3. 视图无法使用索引，也不会激发触发器（实际开发中因为性能等各方面的考虑，通常不建议使用触发器，所以我们也不对这个概念进行介绍）的执行。

函数

MySQL 中的函数跟 Python 中的函数太多的差异，因为函数都是用来封装功能上相对独立且会被重复使用的代码的。如果非要找出一些差别来，那么 MySQL 中的函数是可以执行 SQL 语句的。下面的例子，我们通过自定义函数实现了截断超长字符串的功能。

```
delimiter $$

create function truncate_string(
    content varchar(10000),
    max_length int unsigned
) returns varchar(10000) no sql
begin
    declare result varchar(10000) default content;
    if char_length(content) > max_length then
        set result = left(content, max_length);
        set result = concat(result, '.....');
    end if;
    return result;
end $$

delimiter ;
```

说明1：函数声明后面的`no sql`是声明函数体并没有使用 SQL 语句；如果函数体中需要通过 SQL 读取数据，需要声明为`reads sql data`。

说明2：定义函数前后的`delimiter`命令是为了修改定界符，因为函数体中的语句都是用`;`表示结束，如果不重新定义定界符，那么遇到的`;`的时候代码就会被截断执行，显然这不是我们想要的效果。

在查询中调用自定义函数。

```
select truncate_string('和我在成都的街头走一走，直到所有的灯都熄灭了也不停留', 10) as short_string;
```

```
+-----+
| short_string |
+-----+
```

和我在成都的街头走一.....	
-----	-----

过程

过程（又称存储过程）是事先编译好存储在数据库中的一组 SQL 的集合，调用过程可以简化应用程序开发人员的工作，减少与数据库服务器之间的通信，对于提升数据操作的性能也是有帮助的。其实迄今为止，我们使用的 SQL 语句都是针对一个或多个表的单条语句，但在实际开发中经常会遇到某个操作需要多条 SQL 语句才能完成的情况。例如，电商网站在受理用户订单时，需要做以下一系列的处理。

1. 通过查询来核对库存中是否有对应的物品以及库存是否充足。
2. 如果库存有物品，需要锁定库存以确保这些物品不再卖给别人，并且要减少可用的物品数量以反映正确的库存量。
3. 如果库存不足，可能需要进一步与供应商进行交互或者至少产生一条系统提示消息。
4. 不管受理订单是否成功，都需要产生流水记录，而且需要给对应的用户产生一条通知信息。

我们可以通过过程将复杂的操作封装起来，这样不仅有助于保证数据的一致性，而且将来如果业务发生了变动，只需要调整和修改过程即可。对于调用过程的用户来说，过程并没有暴露数据表的细节，而且执行过程比一条条的执行一组 SQL 要快得多。

下面的过程实现了查询某门课程的最高分、最低分和平均分。

```
drop procedure if exists sp_score_stat;

delimiter $$

create procedure sp_score_stat(
    courseId int,
    out maxScore decimal(4,1),
    out minScore decimal(4,1),
    out avgScore decimal(4,1)
)
begin
    select max(score) into maxScore from tb_record where cou_id=courseId;
    select min(score) into minScore from tb_record where cou_id=courseId;
    select avg(score) into avgScore from tb_record where cou_id=courseId;
end $$

delimiter ;
```

说明：在定义过程时，因为可能需要书写多条 SQL，而分隔这些 SQL 需要使用分号作为分隔符，如果这个时候，仍然用分号表示整段代码结束，那么定义过程的 SQL 就会出现错误，所以上面我们用 `delimiter $$` 将整段代码结束的标记定义为 `$$`，那么代码中的分号将不再表示整段代码的结束，整段代码只会在遇到 `end $$` 时才会执行。在定义完过程后，通过 `delimiter ;` 将结束符重新改回成分号（恢复现场）。

上面定义的过程有四个参数，其中第一个参数是输入参数，代表课程的编号，后面的参数都是输出参数，因为过程不能定义返回值，只能通过输出参数将执行结果带出，定义输出参数的关键字是 `out`，默认情况下参数都

是输入参数。

调用过程。

```
call sp_score_stat(1111, @a, @b, @c);
```

获取输出参数的值。

```
select @a as 最高分, @b as 最低分, @c as 平均分;
```

删除过程。

```
drop procedure sp_score_stat;
```

在过程中，我们可以定义变量、条件，可以使用分支和循环语句，可以通过游标操作查询结果，还可以使用事件调度器，这些内容我们暂时不在此处进行介绍。虽然我们说了很多过程的好处，但是在实际开发中，如果频繁的使用过程并将大量复杂的运算放到过程中，会给数据库服务器造成巨大的压力，而数据库往往都是性能瓶颈所在，使用过程无疑是雪上加霜的操作。所以，对于互联网产品开发，我们一般建议让数据库只做好存储，复杂的运算和处理交给应用服务器上的程序去完成，如果应用服务器变得不堪重负了，我们可以比较容易的部署多台应用服务器来分摊这些压力。

如果大家对上面讲到的视图、函数、过程包括我们没有讲到的触发器这些知识有兴趣，建议大家阅读 MySQL 的入门读物 [《MySQL必知必会》](#) 进行一般性了解即可，因为这些知识点在大家将来的工作中未必用得上，学了也可能仅仅是为了应付面试而已。

MySQL 新特性

JSON类型

很多开发者在使用关系型数据库做数据持久化的时候，常常感到结构化的存储缺乏灵活性，因为必须事先设计好所有的列以及对应的数据类型。在业务发展和变化的过程中，如果需要修改表结构，这绝对是比较麻烦和难受的事情。从 MySQL 5.7 版本开始，MySQL 引入了对 JSON 数据类型的支持（MySQL 8.0 解决了 JSON 的日志性能瓶颈问题），用好 JSON 类型，其实就是打破了关系型数据库和非关系型数据库之间的界限，为数据持久化操作带来了更多的便捷。

JSON 类型主要分为 JSON 对象和 JSON 数组两种，如下所示。

1. JSON 对象

```
{"name": "骆昊", "tel": "13122335566", "QQ": "957658"}
```

2. JSON 数组

[1, 2, 3]

[{"name": "骆昊", "tel": "13122335566"}, {"name": "王大锤", "QQ": "123456"}]

哪些地方需要用到JSON类型呢？举一个简单的例子，现在很多产品的用户登录都支持多种方式，例如手机号、微信、QQ、新浪微博等，但是一般情况下我们又不会要求用户提供所有的这些信息，那么用传统的设计方式，就需要设计多个列来对应多种登录方式，可能还需要允许这些列存在空值，这显然不是很好的选择；另一方面，如果产品又增加了一种登录方式，那么就必然要修改之前的表结构，这就更让人痛苦了。但是，有了JSON类型，刚才的问题就迎刃而解了，我们可以做出如下所示的设计。

```
create table `tb_test`
(
`user_id` bigint unsigned,
`login_info` json,
primary key (`user_id`)
) engine=innodb;

insert into `tb_test` values
(1, '{"tel": "13122335566", "QQ": "654321", "wechat": "jackfrued"}'),
(2, '{"tel": "13599876543", "weibo": "wangdachui123"}');
```

如果要查询用户的手机和微信号，可以用如下所示的 SQL 语句。

```
select
`user_id`,
json_unquote(json_extract(`login_info`, '$.tel')) as 手机号,
json_unquote(json_extract(`login_info`, '$.wechat')) as 微信
from `tb_test`;
```

user_id	手机号	微信
1	13122335566	jackfrued
2	13599876543	NULL

因为支持 JSON 类型，MySQL 也提供了配套的处理 JSON 数据的函数，就像上面用到的 `json_extract` 和 `json_unquote`。当然，上面的 SQL 还有更为便捷的写法，如下所示。

```
select
`user_id`,
```

```

`login_info` -> '$.tel' as 手机号,
`login_info` -> '$.wechat' as 微信
from `tb_test`;

```

再举个例子，如果我们的产品要实现用户画像功能（给用户打标签），然后基于用户画像给用户推荐平台的服务或消费品之类的东西，我们也可以使用 JSON 类型来保存用户画像数据，示意代码如下所示。

创建画像标签表。

```

create table `tb_tags`
(
`tag_id` int unsigned not null comment '标签ID',
`tag_name` varchar(20) not null comment '标签名',
primary key (`tag_id`)
) engine=innodb;

insert into `tb_tags` (`tag_id`, `tag_name`)
values
(1, '70后'),
(2, '80后'),
(3, '90后'),
(4, '00后'),
(5, '爱运动'),
(6, '高学历'),
(7, '小资'),
(8, '有房'),
(9, '有车'),
(10, '爱看电影'),
(11, '爱网购'),
(12, '常点外卖');

```

为用户打标签。

```

create table `tb_users_tags`
(
`user_id` bigint unsigned not null comment '用户ID',
`user_tags` json not null comment '用户标签'
) engine=innodb;

insert into `tb_users_tags` values
(1, '[2, 6, 8, 10]'),
(2, '[3, 10, 12]'),
(3, '[3, 8, 9, 11]');

```

接下来，我们通过一组查询来了解 JSON 类型的巧妙之处。

1. 查询爱看电影（有10这个标签）的用户ID。

```
select * from `tb_users` where 10 member of (user_tags->'$');
```

2. 查询爱看电影 (有10这个标签) 的80后 (有2这个标签) 用户ID。

```
select * from `tb_users` where json_contains(user_tags->'$', '[2, 10]');
```

3. 查询爱看电影或80后或90后的用户ID。

```
select `user_id` from `tb_users_tags` where json_overlaps(user_tags->'$', '[2, 3, 10]');
```

说明：上面的查询用到了`member of`谓词和两个 JSON 函数，`json_contains`可以检查 JSON 数组是否包含了指定的元素，而`json_overlaps`可以检查 JSON 数组是否与指定的数组有重叠部分。

窗口函数

MySQL 从8.0开始支持窗口函数，大多数商业数据库和一些开源数据库早已提供了对窗口函数的支持，有的也将其称之为 OLAP (联机分析和处理) 函数，听名字就知道跟统计和分析相关。为了帮助大家理解窗口函数，我们先说说窗口的概念。

窗口可以理解为记录的集合，窗口函数也就是在满足某种条件的记录集合上执行的特殊函数，对于每条记录都要在此窗口内执行函数。窗口函数和我们上面讲到的聚合函数比较容易混淆，二者的区别主要在于聚合函数是将多条记录聚合为一条记录，窗口函数是每条记录都会执行，执行后记录条数不会变。窗口函数不仅仅是几个函数，它是一套完整的语法，函数只是该语法的一部分，基本语法如下所示：

```
<窗口函数> over (partition by <用于分组的列名> order by <用户排序的列名>)
```

上面语法中，窗口函数的位置可以放以下两种函数：

1. 专用窗口函数，包括：`lead`、`lag`、`first_value`、`last_value`、`rank`、`dense_rank`和`row_number`等。
2. 聚合函数，包括：`sum`、`avg`、`max`、`min`和`count`等。

下面为大家举几个使用窗口函数的简单例子，我们先用如下所示的 SQL 建库建表。

```
-- 创建名为hrs的数据库并指定默认的字符集
create database `hrs` default charset utf8mb4;

-- 切换到hrs数据库
use `hrs`;

-- 创建部门表
```

```

create table `tb_dept`
(
  `dno` int not null comment '编号',
  `dname` varchar(10) not null comment '名称',
  `dloc` varchar(20) not null comment '所在地',
  primary key (`dno`)
);

-- 插入4个部门
insert into `tb_dept` values
  (10, '会计部', '北京'),
  (20, '研发部', '成都'),
  (30, '销售部', '重庆'),
  (40, '运维部', '深圳');

-- 创建员工表
create table `tb_emp`
(
  `eno` int not null comment '员工编号',
  `ename` varchar(20) not null comment '员工姓名',
  `job` varchar(20) not null comment '员工职位',
  `mgr` int comment '主管编号',
  `sal` int not null comment '员工月薪',
  `comm` int comment '每月补贴',
  `dno` int not null comment '所在部门编号',
  primary key (`eno`),
  constraint `fk_emp_mgr` foreign key (`mgr`) references tb_emp (`eno`),
  constraint `fk_emp_dno` foreign key (`dno`) references tb_dept (`dno`)
);

-- 插入14个员工
insert into `tb_emp` values
  (7800, '张三丰', '总裁', null, 9000, 1200, 20),
  (2056, '乔峰', '分析师', 7800, 5000, 1500, 20),
  (3088, '李莫愁', '设计师', 2056, 3500, 800, 20),
  (3211, '张无忌', '程序员', 2056, 3200, null, 20),
  (3233, '丘处机', '程序员', 2056, 3400, null, 20),
  (3251, '张翠山', '程序员', 2056, 4000, null, 20),
  (5566, '宋远桥', '会计师', 7800, 4000, 1000, 10),
  (5234, '郭靖', '出纳', 5566, 2000, null, 10),
  (3344, '黄蓉', '销售主管', 7800, 3000, 800, 30),
  (1359, '胡一刀', '销售员', 3344, 1800, 200, 30),
  (4466, '苗人凤', '销售员', 3344, 2500, null, 30),
  (3244, '欧阳锋', '程序员', 3088, 3200, null, 20),
  (3577, '杨过', '会计', 5566, 2200, null, 10),
  (3588, '朱九真', '会计', 5566, 2500, null, 10);

```

例子1：查询按月薪从高到低排在第4到第6名的员工的姓名和月薪。

```

select * from (
  select
    `ename`, `sal`,

```

```

row_number() over (order by `sal` desc) as `rank`
from `tb_emp`
) `temp` where `rank` between 4 and 6;

```

说明：上面使用的函数`row_number()`可以为每条记录生成一个行号，在实际工作中可以根据需要将其替换为`rank()`或`dense_rank()`函数，三者的区别可以参考官方文档或阅读[《通俗易懂的学会：SQL窗口函数》](#)进行了解。在MySQL 8以前的版本，我们可以通过下面的方式来完成类似的操作。

```

select `rank`, `ename`, `sal` from (
  select @a:="@a+1" as `rank`, `ename`, `sal`
  from `tb_emp`, (select @a:=0) as t1 order by `sal` desc
) t2 where `rank` between 4 and 6;

```

例子2：查询每个部门月薪最高的两名的员工的姓名和部门名称。

```

select `ename`, `sal`, `dname`
from (
  select
    `ename`, `sal`, `dno`,
    rank() over (partition by `dno` order by `sal` desc) as `rank`
  from `tb_emp`
) as `temp` natural join `tb_dept` where `rank` <=2;

```

说明：在MySQL 8以前的版本，我们可以通过下面的方式来完成类似的操作。

```
select `ename`, `sal`, `dname` from `tb_emp` as `t1`
```

```
natural join `tb_dept` where ( select count(*) from `tb_emp` as `t2` where `t1.dno` = `t2.dno` and `t2.sal` > `t1.sal` ) < 2
order by `dno` asc, `sal` desc;
```

其他内容

范式理论

范式理论是设计关系型数据库中二维表的指导思想。

1. 第一范式：数据表的每个列的值域都是由原子值组成的，不能够再分割。
2. 第二范式：数据表里的所有数据都要和该数据表的键（主键与候选键）有完全依赖关系。
3. 第三范式：所有非键属性都只和候选键有相关性，也就是说非键属性之间应该是独立无关的。

说明：实际工作中，出于效率的考虑，我们在设计表时很有可能做出反范式设计，即故意降低方式级别，增加冗余数据来获得更好的操作性能。

数据完整性

1. 实体完整性 - 每个实体都是独一无二的
 - 主键 (primary key) / 唯一约束 (unique)
2. 引用完整性 (参照完整性) - 关系中不允许引用不存在的实体
 - 外键 (foreign key)
3. 域 (domain) 完整性 - 数据是有效的
 - 数据类型及长度
 - 非空约束 (not null)
 - 默认值约束 (default)
 - 检查约束 (check)

说明: 在 MySQL 8.x 以前, 检查约束并不起作用。

数据一致性

1. 事务: 一系列对数据库进行读/写的操作, 这些操作要么全都成功, 要么全都失败。
2. 事务的 ACID 特性
 - 原子性: 事务作为一个整体被执行, 包含在其中的对数据库的操作要么全部被执行, 要么都不执行
 - 一致性: 事务应确保数据库的状态从一个一致状态转变为另一个一致状态
 - 隔离性: 多个事务并发执行时, 一个事务的执行不应影响其他事务的执行
 - 持久性: 已被提交的事务对数据库的修改应该永久保存在数据库中
3. MySQL 中的事务操作
 - 开启事务环境

```
start transaction
```

- 提交事务

```
commit
```

- 回滚事务

```
rollback
```

4. 查看事务隔离级别

```
show variables like 'transaction_isolation';
```

Variable_name	Value
transaction_isolation	REPEATABLE-READ

可以看出，MySQL 默认的事务隔离级别是REPEATABLE-READ。

5. 修改（当前会话）事务隔离级别

```
set session transaction isolation level read committed;
```

重新查看事务隔离级别，结果如下所示。

Variable_name	Value
transaction_isolation	READ-COMMITTED

关系型数据库的事务是一个很大的话题，因为当存在多个并发事务访问数据时，就有可能出现三类读数据的问题（脏读、不可重复读、幻读）和两类更新数据的问题（第一类丢失更新、第二类丢失更新）。想了解这五类问题的，可以阅读我发布在 CSDN 网站上的[《Java面试题全集（上）》](#)一文的第80题。为了避免这些问题，关系型数据库底层是有对应的锁机制的，按锁定对象不同可以分为表级锁和行级锁，按并发事务锁定关系可以分为共享锁和独占锁。然而直接使用锁是非常麻烦的，为此数据库为用户提供了自动锁机制，只要用户指定适当的事务隔离级别，数据库就会通过分析 SQL 语句，然后为事务访问的资源加上合适的锁。此外，数据库还会维护这些锁通过各种手段提高系统的性能，这些对用户来说都是透明的。想了解 MySQL 事务和锁的细节知识，推荐大家阅读进阶读物[《高性能MySQL》](#)，这也是数据库方面的经典书籍。

ANSI/ISO SQL 92标准定义了4个等级的事务隔离级别，如下表所示。需要说明的是，事务隔离级别和数据访问的并发性是对立的，事务隔离级别越高并发性就越差。所以要根据具体的应用来确定到底使用哪种事务隔离级别，这个地方没有万能的原则。

隔离级别	脏读	不可重复读	幻读	第一类丢失更新	第二类丢失更新
READ UNCOMMITTED	允许	允许	允许	不允许	允许
READ COMMITTED	不允许	允许	允许	不允许	允许
REPEATABLE READ	不允许	不允许	允许	不允许	不允许
SERIALIZABLE	不允许	不允许	不允许	不允许	不允许

总结

关于 SQL 和 MySQL 的知识肯定远远不止上面列出的这些，比如 SQL 本身的优化、MySQL 性能调优、MySQL 运维相关工具、MySQL 数据的备份和恢复、监控 MySQL 服务、部署高可用架构等，这一系列的问题在这里都无法逐一展开来讨论，那就留到有需要的时候再进行讲解吧，各位读者也可以自行探索。

深入MySQL

索引

索引是关系型数据库中用来提升查询性能最为重要的手段。关系型数据库中的索引就像一本书的目录，我们可以想象一下，如果要从一本书中找出某个知识点，但是这本书没有目录，这将是意见多么可怕的事情！我们估计得一篇一篇的翻下去，才能确定这个知识点到底在什么位置。创建索引虽然会带来存储空间上的开销，就像一本书的目录会占用一部分篇幅一样，但是在牺牲空间后换来的查询时间的减少也是非常显著的。

MySQL 数据库中所有数据类型的列都可以被索引。对于MySQL 8.0 版本的 InnoDB 存储引擎来说，它支持三种类型的索引，分别是 B+ 树索引、全文索引和 R 树索引。这里，我们只介绍使用得最为广泛的 B+ 树索引。使用 B+ 树的原因非常简单，因为它是目前在基于磁盘进行海量数据存储和排序上最有效率的数据结构。B+ 树是一棵**平衡树**，树的高度通常为3或4，但是却可以保存从百万级到十亿级的数据，而从这些数据里面查询一条数据，只需要3次或4次 I/O 操作。

B+ 树由根节点、中间节点和叶子节点构成，其中叶子节点用来保存排序后的数据。由于记录在索引上是排序过的，因此在一个叶子节点内查找数据时可以使用二分查找，这种查找方式效率非常的高。当数据很少的时候，B+ 树只有一个根节点，数据也就保存在根节点上。随着记录越来越多，B+ 树会发生分裂，根节点不再保存数据，而是提供了访问下一层节点的指针，帮助快速确定数据在哪个叶子节点上。

在创建二维表时，我们通常都会为表指定主键列，主键列上默认会创建索引，而对于 MySQL InnoDB 存储引擎来说，因为它使用的是索引组织表这种数据存储结构，所以主键上的索引就是整张表的数据，而这种索引我们也将其称之为**聚集索引** (clustered index)。很显然，一张表只能有一个聚集索引，否则表的数据岂不是要保存多次。我们自己创建的索引都是二级索引 (secondary index)，更常见的叫法是**非聚集索引** (non-clustered index)。通过我们自定义的非聚集索引只能定位记录的主键，在获取数据时可能需要再通过主键上的聚集索引进行查询，这种现象称为“回表”，因此通过非聚集索引检索数据通常比使用聚集索引检索数据要慢。

接下来我们通过一个简单的例子来说明索引的意义，比如我们要根据学生的姓名来查找学生，这个场景在实际开发中应该经常遇到，就跟通过商品名称查找商品是一个道理。我们可以使用 MySQL 的**explain**关键字来查看 SQL 的执行计划（数据库执行 SQL 语句的具体步骤）。

```
explain select * from tb_student where stuname='林震南'\G
```

```
***** 1. row *****
  id: 1
  select_type: SIMPLE
    table: tb_student
  partitions: NULL
    type: ALL
possible_keys: NULL
    key: NULL
  key_len: NULL
    ref: NULL
  rows: 11
filtered: 10.00
```

```
Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

在上面的 SQL 执行计划中，有几项值得我们关注：

1. `select_type`: 查询的类型。
 - `SIMPLE`: 简单 SELECT，不需要使用 UNION 操作或子查询。
 - `PRIMARY`: 如果查询包含子查询，最外层的 SELECT 被标记为 PRIMARY。
 - `UNION`: UNION 操作中第二个或后面的 SELECT 语句。
 - `SUBQUERY`: 子查询中的第一个 SELECT。
 - `DERIVED`: 派生表的 SELECT 子查询。
2. `table`: 查询对应的表。
3. `type`: MySQL 在表中找到满足条件的行的方式，也称为访问类型，包括：`ALL`（全表扫描）、`index`（索引全扫描，只遍历索引树）、`range`（索引范围扫描）、`ref`（非唯一索引扫描）、`eq_ref`（唯一索引扫描）、`const / system`（常量级查询）、`NULL`（不需要访问表或索引）。在所有的访问类型中，很显然 ALL 是性能最差的，它代表的全表扫描是指要扫描表中的每一行才能找到匹配的行。
4. `possible_keys`: MySQL 可以选择的索引，但是有可能不会使用。
5. `key`: MySQL 真正使用的索引，如果为`NULL`就表示没有使用索引。
6. `key_len`: 使用的索引的长度，在不影响查询的情况下肯定是长度越短越好。
7. `rows`: 执行查询需要扫描的行数，这是一个预估值。
8. `extra`: 关于查询额外的信息。
 - `Using filesort`: MySQL 无法利用索引完成排序操作。
 - `Using index`: 只使用索引的信息而不需要进一步查表来获取更多的信息。
 - `Using temporary`: MySQL 需要使用临时表来存储结果集，常用于分组和排序。
 - `Impossible where`: `where`子句会导致没有符合条件的行。
 - `Distinct`: MySQL 发现第一个匹配行后，停止为当前的行组合搜索更多的行。
 - `Using where`: 查询的列未被索引覆盖，筛选条件并不是索引的前导列。

从上面的执行计划可以看出，当我们通过学生名字查询学生时实际上是进行了全表扫描，不言而喻这个查询性能肯定是非常糟糕的，尤其是在表中的行很多的时候。如果我们需要经常通过学生姓名来查询学生，那么就应该在学生姓名对应的列上创建索引，通过索引来加速查询。

```
create index idx_student_name on tb_student(stuname);
```

再次查看刚才的 SQL 对应的执行计划。

```
explain select * from tb_student where stuname='林震南'\G
```

```
***** 1. row *****
id: 1
select_type: SIMPLE
table: tb_student
partitions: NULL
type: ref
```

```

possible_keys: idx_student_name
      key: idx_student_name
    key_len: 62
      ref: const
     rows: 1
  filtered: 100.00
    Extra: NULL
1 row in set, 1 warning (0.00 sec)

```

可以注意到，在对学生姓名创建索引后，刚才的查询已经不是全表扫描而是基于索引的查询，而且扫描的行只有唯一的一行，这显然大大的提升了查询的性能。MySQL 中还允许创建前缀索引，即对索引字段的前N个字符创建索引，这样的话可以减少索引占用的空间（但节省了空间很有可能会浪费时间，**时间和空间是不可调和的矛盾**），如下所示。

```
create index idx_student_name_1 on tb_student(stuname(1));
```

上面的索引相当于是根据学生姓名的第一个字来创建的索引，我们再看看 SQL 执行计划。

```
explain select * from tb_student where stuname='林震南'\G
```

```

*****
1. row *****
      id: 1
select_type: SIMPLE
      table: tb_student
     partitions: NULL
      type: ref
possible_keys: idx_student_name
      key: idx_student_name
    key_len: 5
      ref: const
     rows: 2
  filtered: 100.00
    Extra: Using where
1 row in set, 1 warning (0.00 sec)

```

不知道大家是否注意到，这一次扫描的行变成了2行，因为学生表中有两个姓“林”的学生，我们只用姓名的第一个字作为索引的话，在查询时通过索引就会找到这两行。

如果要删除索引，可以使用下面的SQL。

```
alter table tb_student drop index idx_student_name;
```

或者

```
drop index idx_student_name on tb_student;
```

在创建索引时，我们还可以使用复合索引、函数索引（MySQL 5.7 开始支持），用好复合索引实现索引覆盖可以减少不必要的排序和回表操作，这样就会让查询的性能成倍的提升，有兴趣的读者可以自行研究。

我们简单的为大家总结一下索引的设计原则：

1. **最适合索引的列是出现在 WHERE 子句和连接子句中的列。**
2. 索引列的基数越大（取值多、重复值少），索引的效果就越好。
3. 使用**前缀索引**可以减少索引占用的空间，内存中可以缓存更多的索引。
4. **索引不是越多越好**，虽然索引加速了读操作（查询），但是写操作（增、删、改）都会变得更慢，因为数据的变化会导致索引的更新，就如同书籍章节的增删需要更新目录一样。
5. 使用 InnoDB 存储引擎时，表的普通索引都会保存主键的值，所以**主键要尽可能选择较短的数据类型**，这样可以有效的减少索引占用的空间，提升索引的缓存效果。

最后，还有一点需要说明，InnoDB 使用的 B-tree 索引，数值类型的列除了等值判断时索引会生效之外，使用 `>`、`<`、`>=`、`<=`、`BETWEEN...AND...`、`<>` 时，索引仍然生效；对于字符串类型的列，如果使用不以通配符开头的模糊查询，索引也是起作用的，但是其他的情况会导致索引失效，这就意味着很有可能会做全表查询。

视图

视图是关系型数据库中将一组查询指令构成的结果集组合成可查询的数据表的对象。简单的说，视图就是虚拟的表，但与数据表不同的是，数据表是一种实体结构，而视图是一种虚拟结构，你也可以将视图理解为保存在数据库中被赋予名字的 SQL 语句。

使用视图可以获得以下好处：

1. 可以将实体数据表隐藏起来，让外部程序无法得知实际的数据结构，让访问者可以使用表的组成部分而不是整个表，降低数据库被攻击的风险。
2. 在大多数的情况下视图是只读的（更新视图的操作通常都有诸多的限制），外部程序无法直接透过视图修改数据。
3. 重用 SQL 语句，将高度复杂的查询包装在视图表中，直接访问该视图即可取出需要的数据；也可以将视图视为数据表进行连接查询。
4. 视图可以返回与实体数据表不同格式的数据，在创建视图的时候可以对数据进行格式化处理。

创建视图。

```
-- 创建视图
create view `vw_avg_score`
as
  select `stu_id`, round(avg(`score`), 1) as `avg_score`
  from `tb_record` group by `stu_id`;

-- 基于已有的视图创建视图
create view `vw_student_score`
as
```

```
select `stu_name`, `avg_score`
from `tb_student` natural join `vw_avg_score`;
```

提示：因为视图不包含数据，所以每次使用视图时，都必须执行查询以获得数据，如果你使用了连接查询、嵌套查询创建了较为复杂的视图，你可能会发现查询性能下降得很厉害。因此，在使用复杂的视图前，应该进行测试以确保其性能能够满足应用的需求。

使用视图。

```
select * from `vw_student_score` order by `avg_score` desc;
```

stuname	avgscore
杨过	95.6
任我行	53.5
王语嫣	84.3
纪嫣然	73.8
岳不群	78.0
东方不败	88.0
项少龙	92.0

既然视图是一张虚拟的表，那么视图中的数据可以更新吗？视图的可更新性要视具体情况而定，以下类型的视图是不能更新的：

1. 使用了聚合函数（`SUM`、`MIN`、`MAX`、`AVG`、`COUNT`等）、`DISTINCT`、`GROUP BY`、`HAVING`、`UNION`或者`UNION ALL`的视图。
2. `SELECT`中包含了子查询的视图。
3. `FROM`子句中包含了一个不能更新的视图的视图。
4. `WHERE`子句的子查询引用了`FROM`子句中的表的视图。

删除视图。

```
drop view vw_student_score;
```

说明：如果希望更新视图，可以先用上面的命令删除视图，也可以通过`create or replace view`来更新视图。

视图的规则和限制。

1. 视图可以嵌套，可以利用从其他视图中检索的数据来构造一个新的视图。视图也可以和表一起使用。
2. 创建视图时可以使用`order by`子句，但如果从视图中检索数据时也使用了`order by`，那么该视图中原先的`order by`会被覆盖。

3. 视图无法使用索引，也不会激发触发器（实际开发中因为性能等各方面的考虑，通常不建议使用触发器，所以我们也不对这个概念进行介绍）的执行。

函数

MySQL 中的函数跟 Python 中的函数太多的差异，因为函数都是用来封装功能上相对独立且会被重复使用的代码的。如果非要找出一些差别来，那么 MySQL 中的函数是可以执行 SQL 语句的。下面的例子，我们通过自定义函数实现了截断超长字符串的功能。

```
delimiter $$

create function truncate_string(
    content varchar(10000),
    max_length int unsigned
) returns varchar(10000) no sql
begin
    declare result varchar(10000) default content;
    if char_length(content) > max_length then
        set result = left(content, max_length);
        set result = concat(result, '.....');
    end if;
    return result;
end $$

delimiter ;
```

说明1：函数声明后面的`no sql`是声明函数体并没有使用 SQL 语句；如果函数体中需要通过 SQL 读取数据，需要声明为`reads sql data`。

说明2：定义函数前后的`delimiter`命令是为了修改定界符，因为函数体中的语句都是用`;`表示结束，如果不重新定义定界符，那么遇到的`;`的时候代码就会被截断执行，显然这不是我们想要的效果。

在查询中调用自定义函数。

```
select truncate_string('和我在成都的街头走一走，直到所有的灯都熄灭了也不停留', 10) as
short_string;
```

+	-----	-----	+
	short_string		
+	-----	-----	+
	和我在成都的街头走一.....		
+	-----	-----	+

过程

过程（又称存储过程）是事先编译好存储在数据库中的一组 SQL 的集合，调用过程可以简化应用程序开发人员的工作，减少与数据库服务器之间的通信，对于提升数据操作的性能也是有帮助的。其实迄今为止，我们使用的 SQL 语句都是针对一个或多个表的单条语句，但在实际开发中经常会遇到某个操作需要多条 SQL 语句才能完成的情况。例如，电商网站在受理用户订单时，需要做以下一系列的处理。

1. 通过查询来核对库存中是否有对应的物品以及库存是否充足。
2. 如果库存有物品，需要锁定库存以确保这些物品不再卖给别人，并且要减少可用的物品数量以反映正确的库存量。
3. 如果库存不足，可能需要进一步与供应商进行交互或者至少产生一条系统提示消息。
4. 不管受理订单是否成功，都需要产生流水记录，而且需要给对应的用户产生一条通知信息。

我们可以通过过程将复杂的操作封装起来，这样不仅有助于保证数据的一致性，而且将来如果业务发生了变动，只需要调整和修改过程即可。对于调用过程的用户来说，过程并没有暴露数据表的细节，而且执行过程比一条条的执行一组 SQL 要快得多。

下面的过程实现了查询某门课程的最高分、最低分和平均分。

```
drop procedure if exists sp_score_stat;

delimiter $$

create procedure sp_score_stat(
    courseId int,
    out maxScore decimal(4,1),
    out minScore decimal(4,1),
    out avgScore decimal(4,1)
)
begin
    select max(score) into maxScore from tb_record where cou_id=courseId;
    select min(score) into minScore from tb_record where cou_id=courseId;
    select avg(score) into avgScore from tb_record where cou_id=courseId;
end $$

delimiter ;
```

说明：在定义过程时，因为可能需要书写多条 SQL，而分隔这些 SQL 需要使用分号作为分隔符，如果这个时候，仍然用分号表示整段代码结束，那么定义过程的 SQL 就会出现错误，所以上面我们用 `delimiter $$` 将整段代码结束的标记定义为 `$$`，那么代码中的分号将不再表示整段代码的结束，整段代码只会在遇到 `end $$` 时才会执行。在定义完过程后，通过 `delimiter ;` 将结束符重新改回成分号（恢复现场）。

上面定义的过程有四个参数，其中第一个参数是输入参数，代表课程的编号，后面的参数都是输出参数，因为过程不能定义返回值，只能通过输出参数将执行结果带出，定义输出参数的关键字是 `out`，默认情况下参数都是输入参数。

调用过程。

```
call sp_score_stat(1111, @a, @b, @c);
```

获取输出参数的值。

```
select @a as 最高分, @b as 最低分, @c as 平均分;
```

删除过程。

```
drop procedure sp_score_stat;
```

在过程中，我们可以定义变量、条件，可以使用分支和循环语句，可以通过游标操作查询结果，还可以使用事件调度器，这些内容我们暂时不在此处进行介绍。虽然我们说了很多过程的好处，但是在实际开发中，如果频繁的使用过程并将大量复杂的运算放到过程中，会给数据库服务器造成巨大的压力，而数据库往往都是性能瓶颈所在，使用过程无疑是雪上加霜的操作。所以，对于互联网产品开发，我们一般建议让数据库只做好存储，复杂的运算和处理交给应用服务器上的程序去完成，如果应用服务器变得不堪重负了，我们可以比较容易的部署多台应用服务器来分摊这些压力。

如果大家对上面讲到的视图、函数、过程包括我们没有讲到的触发器这些知识有兴趣，建议大家阅读 MySQL 的入门读物 [《MySQL必知必会》](#) 进行一般性了解即可，因为这些知识点在大家将来的工作中未必用得上，学了也可能仅仅是为了应付面试而已。

MySQL 新特性

JSON类型

很多开发者在使用关系型数据库做数据持久化的时候，常常感到结构化的存储缺乏灵活性，因为必须事先设计好所有的列以及对应的数据类型。在业务发展和变化的过程中，如果需要修改表结构，这绝对是比较麻烦和难受的事情。从 MySQL 5.7 版本开始，MySQL 引入了对 JSON 数据类型的支持（MySQL 8.0 解决了 JSON 的日志性能瓶颈问题），用好 JSON 类型，其实就是打破了关系型数据库和非关系型数据库之间的界限，为数据持久化操作带来了更多的便捷。

JSON 类型主要分为 JSON 对象和 JSON 数组两种，如下所示。

1. JSON 对象

```
{"name": "骆昊", "tel": "13122335566", "QQ": "957658"}
```

2. JSON 数组

```
[1, 2, 3]
```

```
[{"name": "骆昊", "tel": "13122335566"}, {"name": "王大锤", "QQ": "123456"}]
```

哪些地方需要用到JSON类型呢？举一个简单的例子，现在很多产品的用户登录都支持多种方式，例如手机号、微信、QQ、新浪微博等，但是一般情况下我们又不会要求用户提供所有的这些信息，那么用传统的设计方式，就需要设计多个列来对应多种登录方式，可能还需要允许这些列存在空值，这显然不是很好的选择；另一方面，如果产品又增加了一种登录方式，那么就必然要修改之前的表结构，这就更让人痛苦了。但是，有了JSON类型，刚才的问题就迎刃而解了，我们可以做出如下所示的设计。

```
create table `tb_test`
(
  `user_id` bigint unsigned,
  `login_info` json,
  primary key (`user_id`)
) engine=innodb;

insert into `tb_test` values
(1, '{"tel": "13122335566", "QQ": "654321", "wechat": "jackfrued"}'),
(2, '{"tel": "13599876543", "weibo": "wangdachui123"}');
```

如果要查询用户的手机和微信号，可以用如下所示的 SQL 语句。

```
select
  `user_id`,
  json_unquote(json_extract(`login_info`, '$.tel')) as 手机号,
  json_unquote(json_extract(`login_info`, '$.wechat')) as 微信
from `tb_test`;
```

user_id	手机号	微信
1	13122335566	jackfrued
2	13599876543	NULL

因为支持 JSON 类型，MySQL 也提供了配套的处理 JSON 数据的函数，就像上面用到的 `json_extract` 和 `json_unquote`。当然，上面的 SQL 还有更为便捷的写法，如下所示。

```
select
  `user_id`,
  `login_info` -> '$.tel' as 手机号,
  `login_info` -> '$.wechat' as 微信
from `tb_test`;
```

再举个例子，如果我们的产品要实现用户画像功能（给用户打标签），然后基于用户画像给用户推荐平台的服务或消费品之类的东西，我们也可以使用 JSON 类型来保存用户画像数据，示意代码如下所示。

创建画像标签表。

```
create table `tb_tags`
(
`tag_id` int unsigned not null comment '标签ID',
`tag_name` varchar(20) not null comment '标签名',
primary key (`tag_id`)
) engine=innodb;

insert into `tb_tags` (`tag_id`, `tag_name`)
values
(1, '70后'),
(2, '80后'),
(3, '90后'),
(4, '00后'),
(5, '爱运动'),
(6, '高学历'),
(7, '小资'),
(8, '有房'),
(9, '有车'),
(10, '爱看电影'),
(11, '爱网购'),
(12, '常点外卖');
```

为用户打标签。

```
create table `tb_users_tags`
(
`user_id` bigint unsigned not null comment '用户ID',
`user_tags` json not null comment '用户标签'
) engine=innodb;

insert into `tb_users_tags` values
(1, '[2, 6, 8, 10]'),
(2, '[3, 10, 12]'),
(3, '[3, 8, 9, 11]');
```

接下来，我们通过一组查询来了解 JSON 类型的巧妙之处。

1. 查询爱看电影 (有10这个标签) 的用户ID。

```
select * from `tb_users` where 10 member of (user_tags->'$');
```

2. 查询爱看电影 (有10这个标签) 的80后 (有2这个标签) 用户ID。

```
select * from `tb_users` where json_contains(user_tags->'$', '[2, 10]');
```

3. 查询爱看电影或80后或90后的用户ID。

```
select `user_id` from `tb_users_tags` where json_overlaps(user_tags->'$', '[2, 3, 10]');
```

说明：上面的查询用到了`member of`谓词和两个 JSON 函数，`json_contains`可以检查 JSON 数组是否包含了指定的元素，而`json_overlaps`可以检查 JSON 数组是否与指定的数组有重叠部分。

窗口函数

MySQL 从 8.0 开始支持窗口函数，大多数商业数据库和一些开源数据库早已提供了对窗口函数的支持，有的也将其称之为 OLAP（联机分析和处理）函数，听名字就知道跟统计和分析相关。为了帮助大家理解窗口函数，我们先说说窗口的概念。

窗口可以理解为记录的集合，窗口函数也就是在满足某种条件的记录集合上执行的特殊函数，对于每条记录都要在此窗口内执行函数。窗口函数和我们上面讲到的聚合函数比较容易混淆，二者的区别主要在于聚合函数是将多条记录聚合为一条记录，窗口函数是每条记录都会执行，执行后记录条数不会变。窗口函数不仅仅是几个函数，它是一套完整的语法，函数只是该语法的一部分，基本语法如下所示：

```
<窗口函数> over (partition by <用于分组的列名> order by <用户排序的列名>)
```

上面语法中，窗口函数的位置可以放以下两种函数：

1. 专用窗口函数，包括：`lead`、`lag`、`first_value`、`last_value`、`rank`、`dense_rank`和`row_number`等。
2. 聚合函数，包括：`sum`、`avg`、`max`、`min`和`count`等。

下面为大家举几个使用窗口函数的简单例子，我们先用如下所示的 SQL 建库建表。

```
-- 创建名为hrs的数据库并指定默认的字符集
create database `hrs` default charset utf8mb4;

-- 切换到hrs数据库
use `hrs`;

-- 创建部门表
create table `tb_dept`
(
`dno` int not null comment '编号',
`dname` varchar(10) not null comment '名称',
`dloc` varchar(20) not null comment '所在地',
primary key (`dno`)
```

```

);

-- 插入4个部门
insert into `tb_dept` values
(10, '会计部', '北京'),
(20, '研发部', '成都'),
(30, '销售部', '重庆'),
(40, '运维部', '深圳');

-- 创建员工表
create table `tb_emp`
(
`eno` int not null comment '员工编号',
`ename` varchar(20) not null comment '员工姓名',
`job` varchar(20) not null comment '员工职位',
`mgr` int comment '主管编号',
`sal` int not null comment '员工月薪',
`comm` int comment '每月补贴',
`dno` int not null comment '所在部门编号',
primary key (`eno`),
constraint `fk_emp_mgr` foreign key (`mgr`) references tb_dept (`dno`),
constraint `fk_emp_dno` foreign key (`dno`) references tb_dept (`dno`)
);

-- 插入14个员工
insert into `tb_emp` values
(7800, '张三丰', '总裁', null, 9000, 1200, 20),
(2056, '乔峰', '分析师', 7800, 5000, 1500, 20),
(3088, '李莫愁', '设计师', 2056, 3500, 800, 20),
(3211, '张无忌', '程序员', 2056, 3200, null, 20),
(3233, '丘处机', '程序员', 2056, 3400, null, 20),
(3251, '张翠山', '程序员', 2056, 4000, null, 20),
(5566, '宋远桥', '会计师', 7800, 4000, 1000, 10),
(5234, '郭靖', '出纳', 5566, 2000, null, 10),
(3344, '黄蓉', '销售主管', 7800, 3000, 800, 30),
(1359, '胡一刀', '销售员', 3344, 1800, 200, 30),
(4466, '苗人凤', '销售员', 3344, 2500, null, 30),
(3244, '欧阳锋', '程序员', 3088, 3200, null, 20),
(3577, '杨过', '会计', 5566, 2200, null, 10),
(3588, '朱九真', '会计', 5566, 2500, null, 10);

```

例子1：查询按月薪从高到低排在第4到第6名的员工的姓名和月薪。

```

select * from (
  select
    `ename`, `sal`,
    row_number() over (order by `sal` desc) as `rank`
  from `tb_emp`
) `temp` where `rank` between 4 and 6;

```

说明：上面使用的函数`row_number()`可以为每条记录生成一个行号，在实际工作中可以根据需要将其替换为`rank()`或`dense_rank()`函数，三者的区别可以参考官方文档或阅读《通俗易懂的学会：SQL窗口函数》进行了解。在MySQL 8以前的版本，我们可以通过下面的方式来完成类似的操作。

```
select `rank`, `ename`, `sal` from (
    select @a:="@a+1" as `rank`, `ename`, `sal`
    from `tb_emp`, (select @a:=0) as t1 order by `sal` desc
) t2 where `rank` between 4 and 6;
```

例子2：查询每个部门月薪最高的两名的员工的姓名和部门名称。

```
select `ename`, `sal`, `dname`
from (
    select
        `ename`, `sal`, `dno`,
        rank() over (partition by `dno` order by `sal` desc) as `rank`
    from `tb_emp`
) as `temp` natural join `tb_dept` where `rank` <=2;
```

说明：在MySQL 8以前的版本，我们可以通过下面的方式来完成类似的操作。

```
select `ename`, `sal`, `dname` from `tb_emp` as `t1`
```

natural join `tb_dept` where (select count(*) from `tb_emp` as `t2` where `t1.dno=t2.dno` and `t2.sal>t1.sal`) < 2
order by `dno asc, sal desc`;

其他内容

范式理论

范式理论是设计关系型数据库中二维表的指导思想。

1. 第一范式：数据表的每个列的值域都是由原子值组成的，不能够再分割。
2. 第二范式：数据表里的所有数据都要和该数据表的键（主键与候选键）有完全依赖关系。
3. 第三范式：所有非键属性都只和候选键有相关性，也就是说非键属性之间应该是独立无关的。

说明：实际工作中，出于效率的考虑，我们在设计表时很有可能做出反范式设计，即故意降低方式级别，增加冗余数据来获得更好的操作性能。

数据完整性

1. 实体完整性 - 每个实体都是独一无二的

- 主键 (primary key) / 唯一约束 (unique)

2. 引用完整性 (参照完整性) - 关系中不允许引用不存在的实体

- 外键 (foreign key)

3. 域 (domain) 完整性 - 数据是有效的

- 数据类型及长度
- 非空约束 (not null)
- 默认值约束 (default)
- 检查约束 (check)

说明: 在 MySQL 8.x 以前, 检查约束并不起作用。

数据一致性

1. 事务: 一系列对数据库进行读/写的操作, 这些操作要么全都成功, 要么全都失败。

2. 事务的 ACID 特性

- 原子性: 事务作为一个整体被执行, 包含在其中的对数据库的操作要么全部被执行, 要么都不执行
- 一致性: 事务应确保数据库的状态从一个一致状态转变为另一个一致状态
- 隔离性: 多个事务并发执行时, 一个事务的执行不应影响其他事务的执行
- 持久性: 已被提交的事务对数据库的修改应该永久保存在数据库中

3. MySQL 中的事务操作

- 开启事务环境

```
start transaction
```

- 提交事务

```
commit
```

- 回滚事务

```
rollback
```

4. 查看事务隔离级别

```
show variables like 'transaction_isolation';
```

Variable_name	Value
transaction_isolation	REPEATABLE-READ

可以看出，MySQL 默认的事务隔离级别是REPEATABLE-READ。

5. 修改（当前会话）事务隔离级别

```
set session transaction isolation level read committed;
```

重新查看事务隔离级别，结果如下所示。

Variable_name	Value
transaction_isolation	READ-COMMITTED

关系型数据库的事务是一个很大的话题，因为当存在多个并发事务访问数据时，就有可能出现三类读数据的问题（脏读、不可重复读、幻读）和两类更新数据的问题（第一类丢失更新、第二类丢失更新）。想了解这五类问题的，可以阅读我发布在 CSDN 网站上的[《Java面试题全集（上）》](#)一文的第80题。为了避免这些问题，关系型数据库底层是有对应的锁机制的，按锁定对象不同可以分为表级锁和行级锁，按并发事务锁定关系可以分为共享锁和独占锁。然而直接使用锁是非常麻烦的，为此数据库为用户提供了自动锁机制，只要用户指定适当的事务隔离级别，数据库就会通过分析 SQL 语句，然后为事务访问的资源加上合适的锁。此外，数据库还会维护这些锁通过各种手段提高系统的性能，这些对用户来说都是透明的。想了解 MySQL 事务和锁的细节知识，推荐大家阅读进阶读物[《高性能MySQL》](#)，这也是数据库方面的经典书籍。

ANSI/ISO SQL 92标准定义了4个等级的事务隔离级别，如下表所示。需要说明的是，事务隔离级别和数据访问的并发性是对立的，事务隔离级别越高并发性就越差。所以要根据具体的应用来确定到底使用哪种事务隔离级别，这个地方没有万能的原则。

隔离级别	脏读	不可重复读	幻读	第一类丢失更新	第二类丢失更新
READ UNCOMMITTED	允许	允许	允许	不允许	允许
READ COMMITTED	不允许	允许	允许	不允许	允许
REPEATABLE READ	不允许	不允许	允许	不允许	不允许
SERIALIZABLE	不允许	不允许	不允许	不允许	不允许

总结

关于 SQL 和 MySQL 的知识肯定远远不止上面列出的这些，比如 SQL 本身的优化、MySQL 性能调优、MySQL 运维相关工具、MySQL 数据的备份和恢复、监控 MySQL 服务、部署高可用架构等，这一系列的问题在这里都无法逐一展开来讨论，那就留到有需要的时候再进行讲解吧，各位读者也可以自行探索。

Python程序接入MySQL数据库

在 Python3 中，我们可以使用`mysqlclient`或者`pymysql`三方库来接入 MySQL 数据库并实现数据持久化操作。二者的用法完全相同，只是导入的模块名不一样。我们推荐大家使用纯 Python 的三方库`pymysql`，因为它更容易安装成功。下面我们仍然以之前创建的名为`hrs`的数据库为例，为大家演示如何通过 Python 程序操作 MySQL 数据库实现数据持久化操作。

建库建表

```
-- 创建名为hrs的数据库并指定默认的字符集
create database `hrs` default character set utf8mb4;

-- 切换到hrs数据库
use `hrs`;

-- 创建部门表
create table `tb_dept`
(
`dno` int not null comment '编号',
`dname` varchar(10) not null comment '名称',
`dloc` varchar(20) not null comment '所在地',
primary key (`dno`)
);

-- 插入4个部门
insert into `tb_dept` values
(10, '会计部', '北京'),
(20, '研发部', '成都'),
(30, '销售部', '重庆'),
(40, '运维部', '深圳');

-- 创建员工表
create table `tb_emp`
(
`eno` int not null comment '员工编号',
`ename` varchar(20) not null comment '员工姓名',
`job` varchar(20) not null comment '员工职位',
`mgr` int comment '主管编号',
`sal` int not null comment '员工月薪',
`comm` int comment '每月补贴',
`dno` int not null comment '所在部门编号',
primary key (`eno`),
constraint `fk_emp_mgr` foreign key (`mgr`) references tb_emp (`eno`),
constraint `fk_emp_dno` foreign key (`dno`) references tb_dept (`dno`)
);

-- 插入14个员工
insert into `tb_emp` values
(7800, '张三丰', '总裁', null, 9000, 1200, 20),
(2056, '乔峰', '分析师', 7800, 5000, 1500, 20),
(3088, '李莫愁', '设计师', 2056, 3500, 800, 20),
```

```
(3211, '张无忌', '程序员', 2056, 3200, null, 20),
(3233, '丘处机', '程序员', 2056, 3400, null, 20),
(3251, '张翠山', '程序员', 2056, 4000, null, 20),
(5566, '宋远桥', '会计师', 7800, 4000, 1000, 10),
(5234, '郭靖', '出纳', 5566, 2000, null, 10),
(3344, '黄蓉', '销售主管', 7800, 3000, 800, 30),
(1359, '胡一刀', '销售员', 3344, 1800, 200, 30),
(4466, '苗人凤', '销售员', 3344, 2500, null, 30),
(3244, '欧阳锋', '程序员', 3088, 3200, null, 20),
(3577, '杨过', '会计', 5566, 2200, null, 10),
(3588, '朱九真', '会计', 5566, 2500, null, 10);
```

接入MySQL

首先，我们可以在命令行或者 PyCharm 的终端中通过下面的命令安装 `pymysql`，如果需要接入 MySQL 8，还需要安装一个名为 `cryptography` 的三方库来支持 MySQL 8 的密码认证方式。

```
pip install pymysql cryptography
```

使用 `pymysql` 操作 MySQL 的步骤如下所示：

1. 创建连接。MySQL 服务器启动后，提供了基于 TCP（传输控制协议）的网络服务。我们可以通过 `pymysql` 模块的 `connect` 函数连接 MySQL 服务器。在调用 `connect` 函数时，需要指定主机 (`host`)、端口 (`port`)、用户名 (`user`)、口令 (`password`)、数据库 (`database`)、字符集 (`charset`) 等参数，该函数会返回一个 `Connection` 对象。
2. 获取游标。连接 MySQL 服务器成功后，接下来要做的就是向数据库服务器发送 SQL 语句，MySQL 会执行接收到的 SQL 并将执行结果通过网络返回。要实现这项操作，需要先通过连接对象的 `cursor` 方法获取游标 (`Cursor`) 对象。
3. 发出 SQL。通过游标对象的 `execute` 方法，我们可以向数据库发出 SQL 语句。
4. 如果执行 `insert`、`delete` 或 `update` 操作，需要根据实际情况提交或回滚事务。因为创建连接时，默认开启了事务环境，在操作完成后，需要使用连接对象的 `commit` 或 `rollback` 方法，实现事务的提交或回滚，`rollback` 方法通常会放在异常捕获代码块 `except` 中。如果执行 `select` 操作，需要通过游标对象抓取查询的结果，对应的方法有三个，分别是：`fetchone`、`fetchmany` 和 `fetchall`。其中 `fetchone` 方法会抓取到一条记录，并以元组或字典的方式返回；`fetchmany` 和 `fetchall` 方法会抓取到多条记录，以嵌套元组或列表装字典的方式返回。
5. 关闭连接。在完成持久化操作后，请不要忘记关闭连接，释放外部资源。我们通常会在 `finally` 代码块中使用连接对象的 `close` 方法来关闭连接。

代码实操

下面，我们通过代码实操的方式为大家演示上面说的五个步骤。

插入数据

```
import pymysql
```

```

no = int(input('部门编号: '))
name = input('部门名称: ')
location = input('部门所在地: ')

# 1. 创建连接 (Connection)
conn = pymysql.connect(host='127.0.0.1', port=3306,
                       user='guest', password='Guest.618',
                       database='hrs', charset='utf8mb4')

try:
    # 2. 获取游标对象 (Cursor)
    with conn.cursor() as cursor:
        # 3. 通过游标对象向数据库服务器发出SQL语句
        affected_rows = cursor.execute(
            'insert into `tb_dept` values (%s, %s, %s)',
            (no, name, location)
        )
        if affected_rows == 1:
            print('新增部门成功!!!!')
    # 4. 提交事务 (transaction)
    conn.commit()
except pymysql.MySQLError as err:
    # 4. 回滚事务
    conn.rollback()
    print(type(err), err)
finally:
    # 5. 关闭连接释放资源
    conn.close()

```

说明: 上面的127.0.0.1称为回环地址, 它代表的是本机。下面的guest是我提前创建好的用户, 该用户拥有对hrs数据库的insert、delete、update和select权限。我们不建议大家在项目中直接使用root超级管理员账号访问数据库, 这样做实在是太危险了。我们可以使用下面的命令创建名为guest的用户并为其授权。

```

create user 'guest'@'%' identified by 'Guest.618';
grant insert, delete, update, select on `hrs`.* to 'guest'@'%';

```

如果要插入大量数据, 建议使用游标对象的executemany方法做批处理 (一个insert操作后面跟上多组数据), 大家可以尝试向一张表插入10000条记录, 然后看看不使用批处理一条条的插入和使用批处理有什么差别。游标对象的executemany方法第一个参数仍然是SQL语句, 第二个参数可以是包含多组数据的列表或元组。

删除数据

```

import pymysql

no = int(input('部门编号: '))

# 1. 创建连接 (Connection)

```

```

conn = pymysql.connect(host='127.0.0.1', port=3306,
                      user='guest', password='Guest.618',
                      database='hrs', charset='utf8mb4',
                      autocommit=True)

try:
    # 2. 获取游标对象 (Cursor)
    with conn.cursor() as cursor:
        # 3. 通过游标对象向数据库服务器发出SQL语句
        affected_rows = cursor.execute(
            'delete from `tb_dept` where `dno`=%s',
            (no, )
        )
        if affected_rows == 1:
            print('删除部门成功!!!!')
finally:
    # 5. 关闭连接释放资源
    conn.close()

```

说明:如果不希望每次 SQL 操作之后手动提交或回滚事务, 可以`connect`函数中加一个名为`autocommit`的参数并将它的值设置为`True`, 表示每次执行 SQL 成功后自动提交。但是我们建议大家手动提交或回滚, 这样可以根据实际业务需要来构造事务环境。如果不愿意捕获异常并进行处理, 可以在`try`代码块后直接跟`finally`块, 省略`except`意味着发生异常时, 代码会直接崩溃并将异常栈显示在终端中。

更新数据

```

import pymysql

no = int(input('部门编号: '))
name = input('部门名称: ')
location = input('部门所在地: ')

# 1. 创建连接 (Connection)
conn = pymysql.connect(host='127.0.0.1', port=3306,
                      user='guest', password='Guest.618',
                      database='hrs', charset='utf8mb4')

try:
    # 2. 获取游标对象 (Cursor)
    with conn.cursor() as cursor:
        # 3. 通过游标对象向数据库服务器发出SQL语句
        affected_rows = cursor.execute(
            'update `tb_dept` set `dname`=%s, `dloc`=%s where `dno`=%s',
            (name, location, no)
        )
        if affected_rows == 1:
            print('更新部门信息成功!!!!')
    # 4. 提交事务
    conn.commit()
except pymysql.MySQLError as err:
    # 4. 回滚事务
    conn.rollback()

```

```

    print(type(err), err)
finally:
    # 5. 关闭连接释放资源
    conn.close()

```

查询数据

1. 查询部门表的数据。

```

import pymysql

# 1. 创建连接 (Connection)
conn = pymysql.connect(host='127.0.0.1', port=3306,
                       user='guest', password='Guest.618',
                       database='hrs', charset='utf8mb4')

try:
    # 2. 获取游标对象 (Cursor)
    with conn.cursor() as cursor:
        # 3. 通过游标对象向数据库服务器发出SQL语句
        cursor.execute('select `dno`, `dname`, `dloc` from `tb_dept`')
        # 4. 通过游标对象抓取数据
        row = cursor.fetchone()
        while row:
            print(row)
            row = cursor.fetchone()
except pymysql.MySQLError as err:
    print(type(err), err)
finally:
    # 5. 关闭连接释放资源
    conn.close()

```

说明: 上面的代码中, 我们通过构造一个`while`循环实现了逐行抓取查询结果的操作。这种方式特别适合查询结果有非常多行的场景。因为如果使用`fetchall`一次性将所有记录抓取到一个嵌套元组中, 会造成非常大的内存开销, 这在很多场景下并不是一个好主意。如果不愿意使用`while`循环, 还可以考虑使用`iter`函数构造一个迭代器来逐行抓取数据, 有兴趣的读者可以自行研究。

2. 分页查询员工表的数据。

```

import pymysql

page = int(input('页码: '))
size = int(input('大小: '))

# 1. 创建连接 (Connection)
con = pymysql.connect(host='127.0.0.1', port=3306,
                      user='guest', password='Guest.618',
                      database='hrs', charset='utf8')

try:
    # 2. 获取游标对象 (Cursor)

```

```

with con.cursor(pymysql.cursors.DictCursor) as cursor:
    # 3. 通过游标对象向数据库服务器发出SQL语句
    cursor.execute(
        'select `eno`, `ename`, `job`, `sal` from `tb_emp` order by `sal` desc
    limit %s,%s',
        ((page - 1) * size, size)
    )
    # 4. 通过游标对象抓取数据
    for emp_dict in cursor.fetchall():
        print(emp_dict)
finally:
    # 5. 关闭连接释放资源
    con.close()

```

案例讲解

下面我们为大家讲解一个将数据库表数据导出到 Excel 文件的例子，我们需要先安装openpyxl三方库，命令如下所示。

```
pip install openpyxl
```

接下来，我们通过下面的代码实现了将数据库hrs中所有员工的编号、姓名、职位、月薪、补贴和部门名称导出到一个 Excel 文件中。

```

import openpyxl
import pymysql

# 创建工作簿对象
workbook = openpyxl.Workbook()
# 获得默认的工作表
sheet = workbook.active
# 修改工作表的标题
sheet.title = '员工基本信息'
# 给工作表添加表头
sheet.append(['工号', '姓名', '职位', '月薪', '补贴', '部门'])
# 创建连接 (Connection)
conn = pymysql.connect(host='127.0.0.1', port=3306,
                       user='guest', password='Guest.618',
                       database='hrs', charset='utf8mb4')
try:
    # 获取游标对象 (Cursor)
    with conn.cursor() as cursor:
        # 通过游标对象执行SQL语句
        cursor.execute(
            'select `eno`, `ename`, `job`, `sal`, coalesce(`comm`, 0), `dname` '
            'from `tb_emp` natural join `tb_dept`'
        )
        # 通过游标抓取数据
        row = cursor.fetchone()

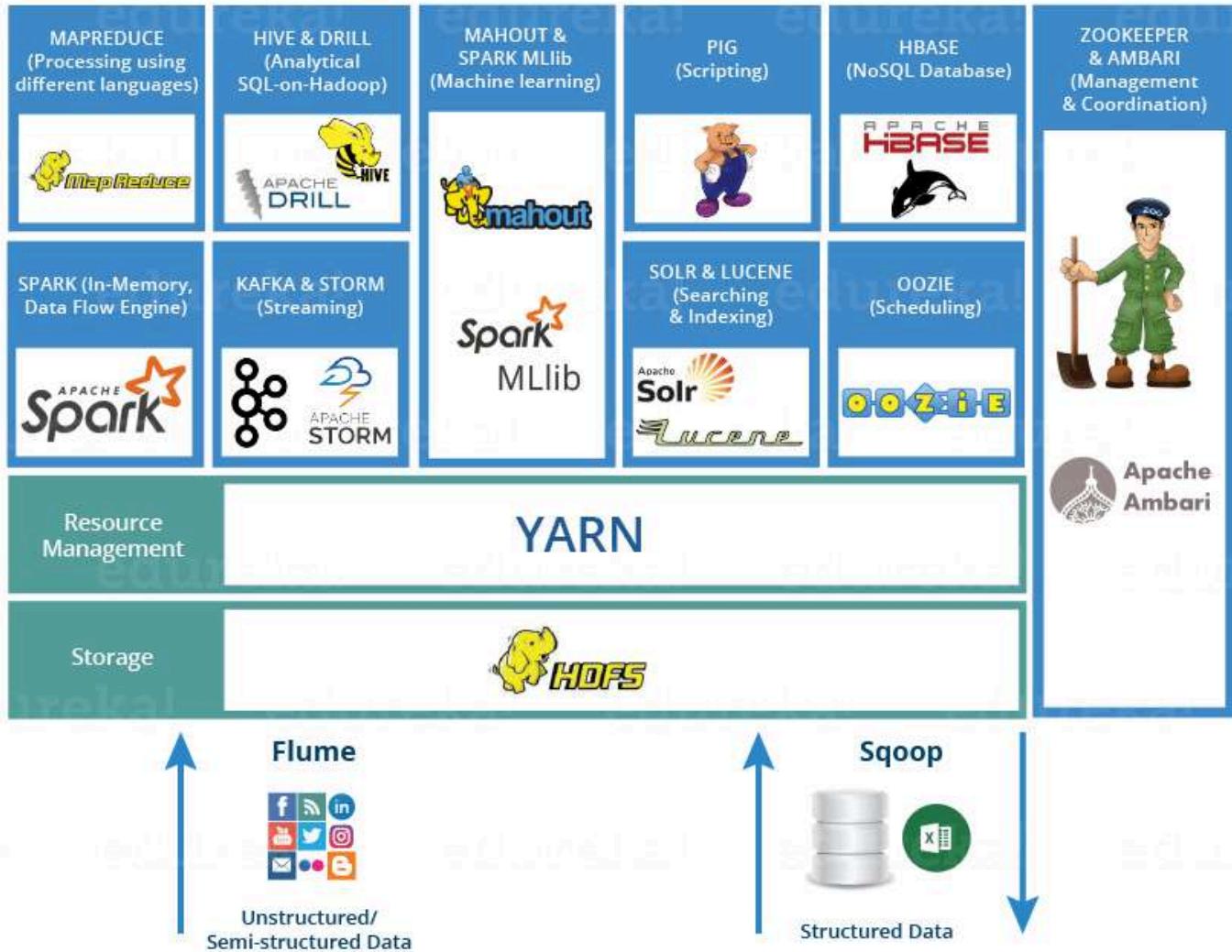
```

```
while row:
    # 将数据逐行写入工作表中
    sheet.append(row)
    row = cursor.fetchone()
# 保存工作簿
workbook.save('hrs.xlsx')
except pymysql.MySQLError as err:
    print(err)
finally:
    # 关闭连接释放资源
    conn.close()
```

大家可以参考上面的例子，试一试把 Excel 文件的数据导入到指定数据库的指定表中，看看是否可以成功。

Hive简介

Hive是Facebook开源的一款基于Hadoop的数据仓库工具，是目前应用最广泛的大数据处理解决方案，它能将SQL查询转变为MapReduce（Google提出的一个软件架构，用于大规模数据集的并行运算）任务，对SQL提供了完美的支持，能够非常方便的实现大数据统计。



说明：可以通过<https://www.edureka.co/blog/hadoop-ecosystem>来了解Hadoop生态圈。

如果要简单的介绍Hive，那么以下两点是其核心：

1. 把HDFS中结构化的数据映射成表。
2. 通过把Hive-SQL进行解析和转换，最终生成一系列基于Hadoop的MapReduce任务/Spark任务，通过执行这些任务完成对数据的处理。也就是说，即便不学习Java、Scala这样的编程语言，一样可以实现对数据的处理。

Hive和传统关系型数据库的对比如下表所示。

Hive	RDBMS
查询语言	HQL
存储数据	HDFS
执行方式	MapReduce / Spark
	Executor

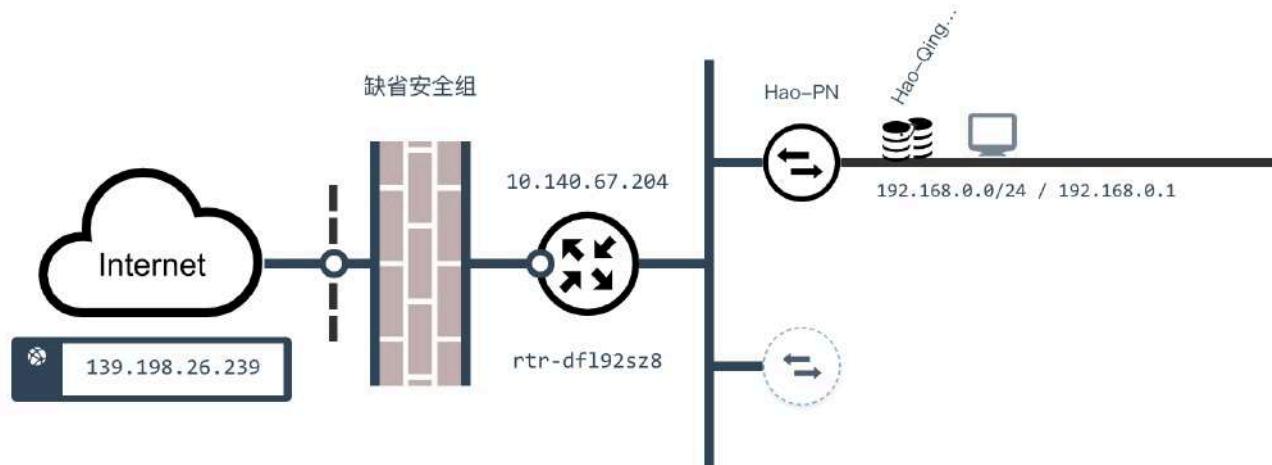
Hive	RDBMS
执行延迟 高	低
数据规模 大	小

准备工作

1. 搭建如下图所示的大数据平台。



2. 通过Client节点访问大数据平台。



3. 创建文件Hadoop的文件系统。

```
hadoop fs -mkdir /data
hadoop fs -chmod g+w /data
```

4. 将准备好的数据文件拷贝到Hadoop文件系统中。

```
hadoop fs -put /home/ubuntu/data/* /data
```

创建/删除数据库

创建。

```
create database if not exists demo;
```

或

```
hive -e "create database demo;"
```

删除。

```
drop database if exists demo;
```

切换。

```
use demo;
```

数据类型

Hive的数据类型如下所示。

基本数据类型。

数据类型	占用空间	支持版本
tinyint	1-Byte	
smallint	2-Byte	
int	4-Byte	
bigint	8-Byte	
boolean		
float	4-Byte	
double	8-Byte	
string		
binary		0.8版本
timestamp		0.8版本

数据类型	占用空间	支持版本
decimal	0.11版本	
char	0.13版本	
varchar	0.12版本	
date	0.12版本	

复杂数据类型。

数据类型	描述	例子
struct	和C语言中的结构体类似	struct<first_name:string, last_name:string>
map	由键值对构成的元素的集合	map<string,int>
array	具有相同类型的变量的容器	array<string>

创建和使用表

1. 创建内部表。

```
create table if not exists user_info
(
  user_id string,
  user_name string,
  sex string,
  age int,
  city string,
  firstactivetime string,
  level int,
  extra1 string,
  extra2 map<string,string>
)
row format delimited fields terminated by '\t'
collection items terminated by ','
map keys terminated by ':'
lines terminated by '\n'
stored as textfile;
```

2. 加载数据。

```
load data local inpath '/home/ubuntu/data/user_info/user_info.txt' overwrite
into table user_info;
```

或

```
load data inpath '/data/user_info/user_info.txt' overwrite into table user_info;
```

3. 创建分区表。

```
create table if not exists user_trade
(
  user_name string,
  piece int,
  price double,
  pay_amount double,
  goods_category string,
  pay_time bigint
)
partitioned by (dt string)
row format delimited fields terminated by '\t';
```

4. 设置动态分区。

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.max.dynamic.partitions=10000;
set hive.exec.max.dynamic.partitions.pernode=10000;
```

5. 拷贝数据 (Shell命令) 。

```
hdfs dfs -put /home/ubuntu/data/user_trade/*
/user/hive/warehouse/demo.db/user_trade
```

6. 修复分区表。

```
msck repair table user_trade;
```

查询

基本语法

```
select user_name from user_info where city='beijing' and sex='female' limit 10;
select user_name, piece, pay_amount from user_trade where dt='2019-03-24' and goods_category='food';
```

group by

```
-- 查询2019年1月到4月，每个品类有多少人购买，累计金额是多少
select goods_category, count(distinct user_name) as user_num, sum(pay_amount) as
total from user_trade where dt between '2019-01-01' and '2019-04-30' group by
goods_category;
```

```
-- 查询2019年4月支付金额超过5万元的用户
select user_name, sum(pay_amount) as total from user_trade where dt between '2019-
04-01' and '2019-04-30' group by user_name having sum(pay_amount) > 50000;
```

order by

```
-- 查询2019年4月支付金额最多的用户前5名
select user_name, sum(pay_amount) as total from user_trade where dt between '2019-
04-01' and '2019-04-30' group by user_name order by total desc limit 5;
```

常用函数

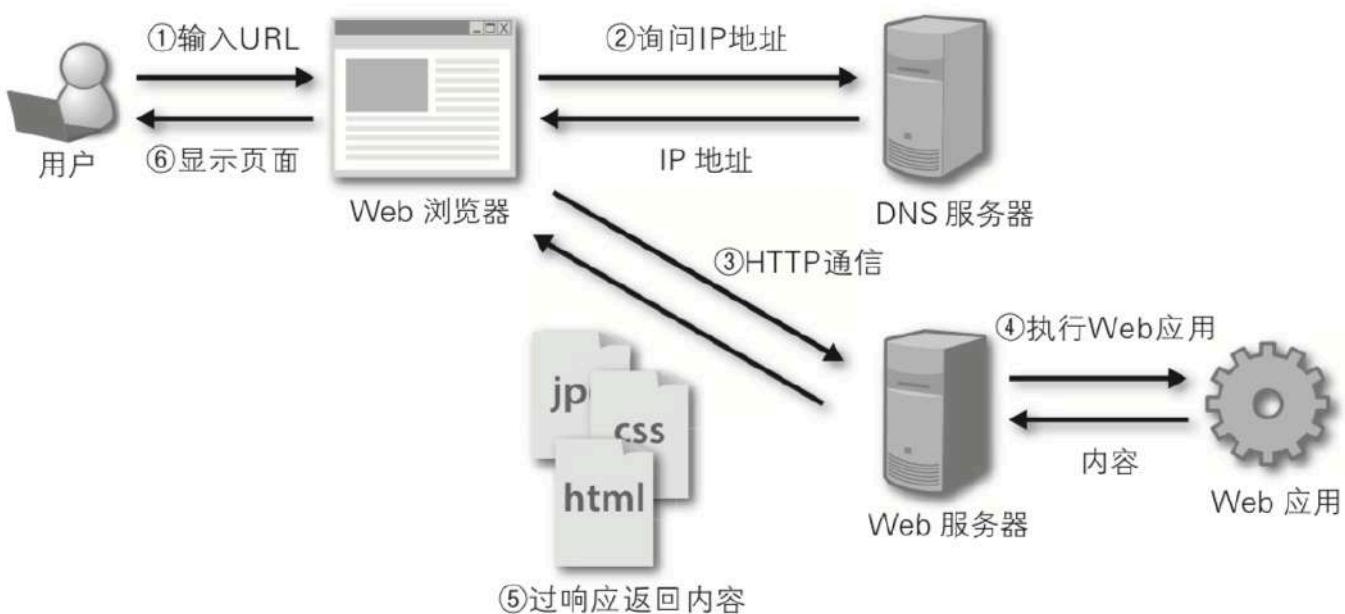
1. `from_unixtime`: 将时间戳转换成日期
2. `unix_timestamp`: 将日期转换成时间戳
3. `datediff`: 计算两个日期的时间差
4. `if`: 根据条件返回不同的值
5. `substr`: 字符串取子串
6. `get_json_object`: 从JSON字符串中取出指定的`key`对应的`value`, 如: `get_json_object(info, '$.first_name')`。

Django快速上手

Web开发的早期阶段，开发者需要手动编写每个页面，例如一个新闻门户网站，每天都要修改它的HTML页面，随着网站规模和体量的增大，这种做法一定是非常糟糕的。为了解决这个问题，开发人员想到了用程序来为Web服务器生成动态内容，也就是说网页中的动态内容不再通过手动编写而是通过程序自动生成。最早的时候，这项技术被称为CGI（公共网关接口），当然随着时间的推移，CGI暴露出的问题也越来越多，例如大量重复的样板代码，总体性能较为低下等。在时代呼唤新英雄的背景下，PHP、ASP、JSP这类Web应用开发技术在上世纪90年代中后期如雨后春笋般涌现。通常我们说的Web应用是指通过浏览器来访问网络资源的应用程序，因为浏览器的普及性以及易用性，Web应用使用起来方便简单，免除了安装和更新应用程序带来的麻烦；站在开发者的角度，也不用关心用户使用什么样的操作系统，甚至不用区分是PC端还是移动端。

Web应用机制和术语

下图向我们展示了Web应用的工作流程，其中涉及到的术语如下表所示。



说明：相信有经验的读者会发现，这张图中其实还少了很多东西，例如反向代理服务器、数据库服务器、防火墙等，而且图中的每个节点在实际项目部署时可能是一组节点组成的集群。当然，如果你对这些没有什么概念也不要紧，继续下去就行了，后面会给大家一一讲解的。

术语	解释
URL/URI	统一资源定位符/统一资源标识符，网络资源的唯一标识
域名	与Web服务器地址对应的一个易于记忆的字符串名字
DNS	域名解析服务，可以将域名转换成对应的IP地址
IP地址	网络上的主机的身份标识，通过IP地址可以区分不同的主机
HTTP	超文本传输协议，构建在TCP之上的应用级协议，万维网数据通信的基础
反向代理	代理客户端向服务器发出请求，然后将服务器返回的资源返回给客户端
Web服务器	接受HTTP请求，然后返回HTML文件、纯文本文件、图像等资源给请求者

术语	解释
Nginx	高性能的Web服务器，也可以用作 反向代理 ， 负载均衡 和 HTTP缓存

HTTP协议

这里我们先费一些笔墨来说说HTTP这个协议。HTTP（超文本传输协议）是构建于TCP（传输控制协议）之上应用级协议，它利用了TCP提供的可靠的传输服务实现了Web应用中的数据交换。按照维基百科上的介绍，设计HTTP最初的目的是提供一种发布和接收HTML页面的方法，也就是说这个协议是浏览器和Web服务器之间传输的数据的载体。关于这个协议的详细信息以及目前的发展状况，大家可以阅读[《HTTP 协议入门》](#)、[《互联网协议入门》](#)系列以及[《图解HTTPS协议》](#)这几篇文章进行了解。下图是我在四川省网络通信技术重点实验室学习和工作期间使用开源协议分析工具Ethereal（抓包工具WireShark的前身）截取的访问百度首页时的HTTP请求和响应的报文（协议数据），由于Ethereal截取的是经过网络适配器的数据，因此可以清晰的看到从物理链路层到应用层的协议数据。

HTTP请求（请求行+请求头+空行+[消息体]）：

```
Frame 4 (563 bytes on wire, 563 bytes captured)
Ethernet II, Src: Elitegro_f9:5d:82 (00:11:5b:f9:5d:82), Dst: Cisco_50:14:71 (00:1b:2a:50:14:71)
Internet Protocol, Src: 192.168.58.136 (192.168.58.136), Dst: 119.75.213.51 (119.75.213.51)
Transmission Control Protocol, Src Port: voispeed-port (3541), Dst Port: http (80), Seq: 1, Ack: 1, Len: 509
Hypertext Transfer Protocol
    GET / HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Language: zh-cn\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727)\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: www.baidu.com\r\n
    Connection: Keep-Alive\r\n
    [truncated] Cookie: BAIDUID=72675E110453F51BEAC13B6277CE022F:FG=1; BDLLFONT=0; BDUSS=VFJenJqbGZus21EM1dja3vyv\r\n
```

HTTP响应（响应行+响应头+空行+消息体）：

```
Frame 7 (668 bytes on wire, 668 bytes captured)
Ethernet II, Src: Cisco_50:14:71 (00:1b:2a:50:14:71), Dst: Elitegro_f9:5d:82 (00:11:5b:f9:5d:82)
Internet Protocol, Src: 119.75.213.51 (119.75.213.51), Dst: 192.168.58.136 (192.168.58.136)
Transmission Control Protocol, Src Port: http (80), Dst Port: voispeed-port (3541), Seq: 1421, Ack: 510, Len: 2034
[Reassembled TCP Segments (2034 bytes): #6(1420), #7(614)]
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Thu, 10 Sep 2009 04:02:47 GMT\r\n
    Server: BWS/1.0\r\n
    Content-Length: 1826\r\n
    Content-Type: text/html\r\n
    Cache-Control: private\r\n
    Expires: Thu, 10 Sep 2009 04:02:47 GMT\r\n
    Content-Encoding: gzip\r\n
    \r\n
    Content-encoded entity body (gzip): 1826 bytes -> 3719 bytes
Line-based text data: text/html
```

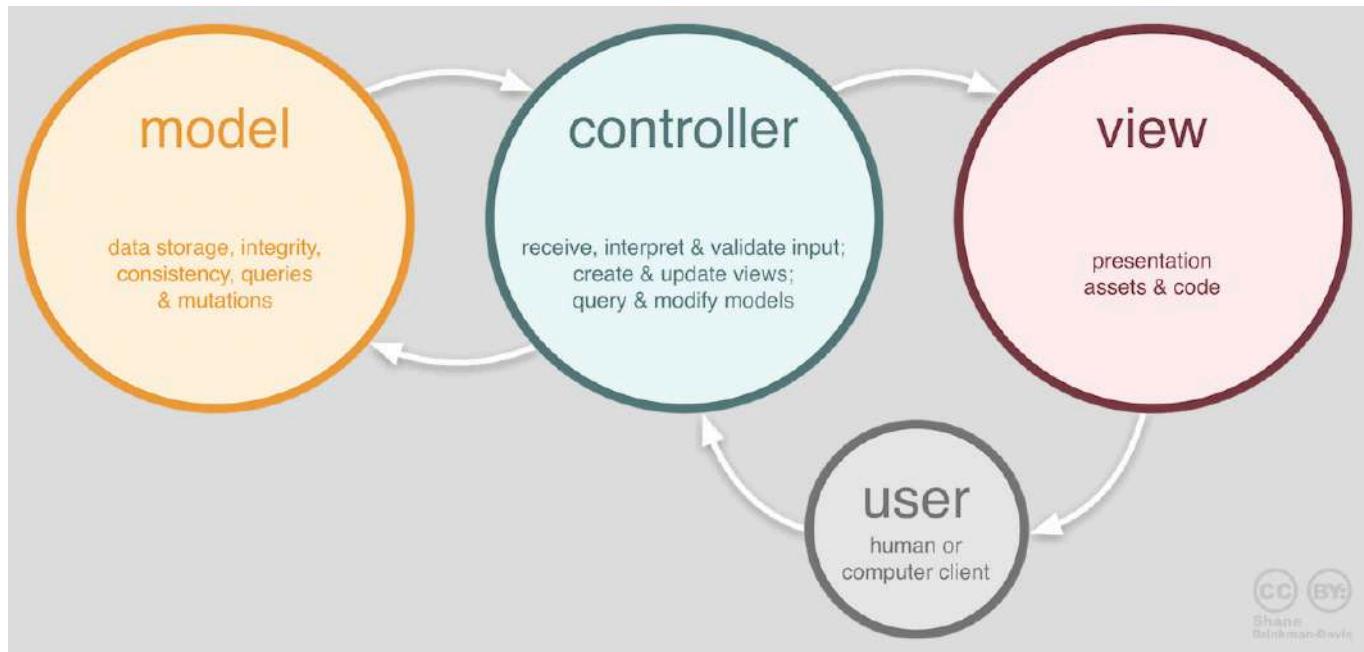
说明：这两张图是在2009年9月10日凌晨获得的，但愿这两张如同泛黄的照片般的截图能帮助你了解HTTP到底是什么样子的。当然，如果没有专业的抓包工具，也可以通过浏览器提供的“开发者工具”来查看HTTP请求和响应的数据格式。

Django概述

Python的Web框架有上百个，比它的关键字还要多。所谓Web框架，就是用于开发Web服务器端应用的基础设施，说得通俗一点就是一系列封装好的模块和工具。事实上，即便没有Web框架，我们仍然可以通过socket或

CGI来开发Web服务器端应用，但是这样做的成本和代价在商业项目中通常是不能接受的。通过Web框架，我们可以化繁为简，降低创建、更新、扩展应用程序的工作量。刚才我们说到Python有上百个Web框架，这些框架包括Django、Flask、Tornado、Sanic、Pyramid、Bottle、Web2py、web.py等。

在上述Python的Web框架中，Django无疑是最有代表性的重量级选手，开发者可以基于Django快速的开发可靠的Web应用程序，因为它减少了Web开发中不必要的开销，对常用的设计和开发模式进行了封装，并对MVC架构提供了支持（Django中称之为MTV架构）。MVC是软件系统开发领域中一种放之四海而皆准的架构，它将系统中的组件分为模型（Model）、视图（View）和控制器（Controller）三个部分并借此实现模型（数据）和视图（显示）的解耦合。由于模型和视图进行了分离，所以需要一个中间人将解耦合的模型和视图联系起来，扮演这个角色的就是控制器。稍具规模的软件系统都会使用MVC架构（或者是从MVC演进出的其他架构），Django项目中我们称之为MTV，MTV中的M跟MVC中的M没有区别，就是代表数据的模型，V代表了网页模板（显示数据的视图），而C代表了视图函数，在Django框架中，视图函数和Django框架本身一起扮演了MVC中C的角色。



Django框架诞生于2003年，它是一个在真正的应用中成长起来的项目，由劳伦斯出版集团旗下在线新闻网站的内容管理系统（CMS）研发团队（主要是Adrian Holovaty和Simon Willison）开发，以比利时的吉普赛爵士吉他手Django Reinhardt来命名。Django框架在2005年夏天作为开源框架发布，使用Django框架能用很短的时间构建出功能完备的网站，因为它代替程序员完成了那些重复乏味的劳动，剩下真正有意义的核心业务给程序员来开发，这一点就是对DRY（Don't Repeat Yourself）理念的最好践行。许多成功的网站和应用都是基于Python语言进行开发的，国内比较有代表性的网站包括：知乎、豆瓣网、果壳网、搜狐闪电邮箱、101围棋网、海报时尚网、背书吧、堆糖、手机搜狐网、咕咚、爱福窝、果库等，其中不乏使用了Django框架的产品。

快速上手

第一个Django项目

1. 检查Python环境：Django 1.11需要Python 2.7或Python 3.4以上的版本；Django 2.0需要Python 3.4以上的版本；Django 2.1和2.2需要Python 3.5以上的版本；Django 3.0需要Python 3.6以上版本。

说明：Django框架不同版本所需的Python解释器环境，可以在Django官方文档的[FAQ](#)中找到。

可以在macOS的终端中输入下面的命令检查Python解释器版本，Windows系统可以在命令行提示符中输入`python --version`。

```
python3 --version
```

也可以在Python的交互式环境中执行下面的代码来查看Python解释器的版本。

```
```Shell
import sys
sys.version
sys.version_info
```

## 2. 更新包管理工具并安装Django环境（用于创建Django项目）。

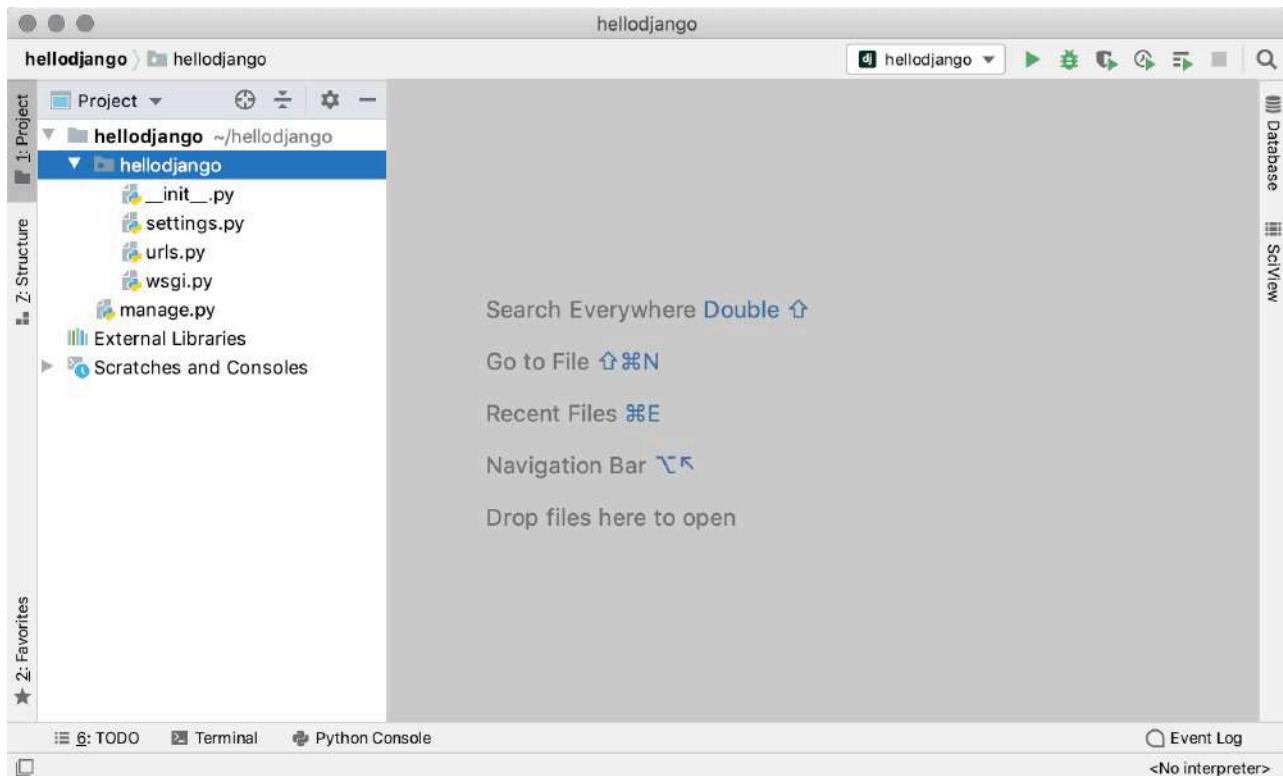
**说明：**在更新这个文档时，Django最新的正式版本是3.0.7，Django 3.0提供了对ASGI的支持，可以实现全双工的异步通信，但是目前的使用体验一般，所以暂时不推荐大家使用Django 3.0，下面我们安装的是Django 2.2.13版本。使用pip安装三方库和工具时，可以通过==来指定安装的版本。

```
pip3 install -U pip
pip3 install django==2.2.13
```

## 3. 检查Django环境并使用django-admin命令创建Django项目（项目名称为hellodjango）。

```
django-admin --version
django-admin startproject hellodjango
```

## 4. 用PyCharm打开创建好的Django项目，并为其添加虚拟环境。

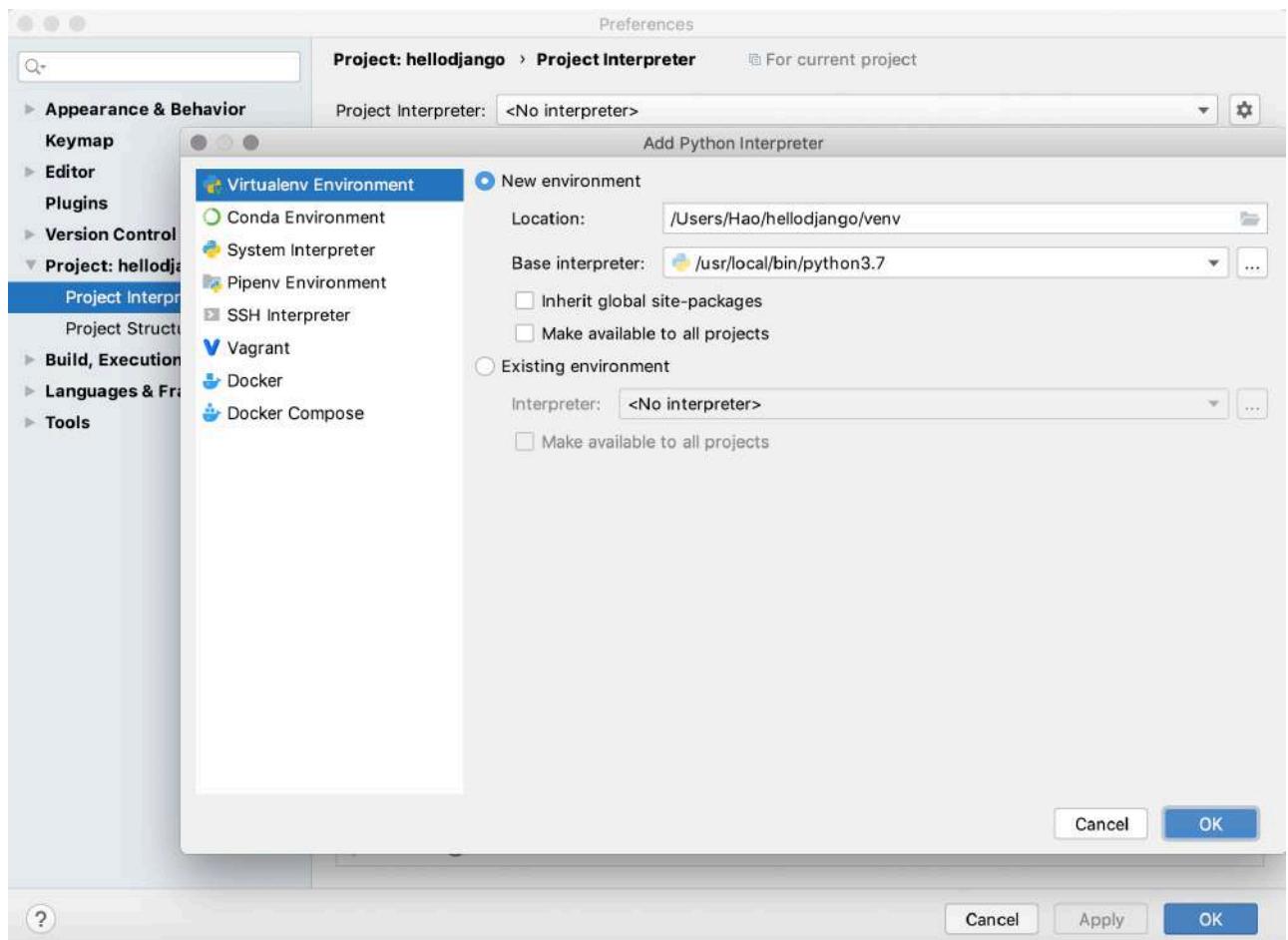


如上图所示，PyCharm的项目浏览器中，最顶层的文件夹`hellodjango`是Python项目文件夹，这个文件夹的名字并不重要，Django项目也不关心这个文件夹叫什么名字。该文件夹下有一个同名的文件夹，它是Django项目文件夹，其中包含了`__init__.py`、`settings.py`、`urls.py`、`wsgi.py`四个文件，与名为`hellodjango`的Django项目文件夹同级的还有一个名为`manage.py`的文件，这些文件的作用如下所示：

- `hellodjango/__init__.py`: 空文件，告诉Python解释器这个目录应该被视为一个Python的包。
- `hellodjango/settings.py`: Django项目的配置文件。
- `hellodjango/urls.py`: Django项目的URL映射声明，就像是网站的“目录”。
- `hellodjango/wsgi.py`: 项目运行在WSGI兼容Web服务器上的入口文件。
- `manage.py`: 管理Django项目的脚本程序。

说明：WSGI全称是Web服务器网关接口，维基百科上给出的解释是“为Python语言定义的Web服务器和Web应用程序或框架之间的一种简单而通用的接口”。

创建虚拟环境的界面如下图所示。



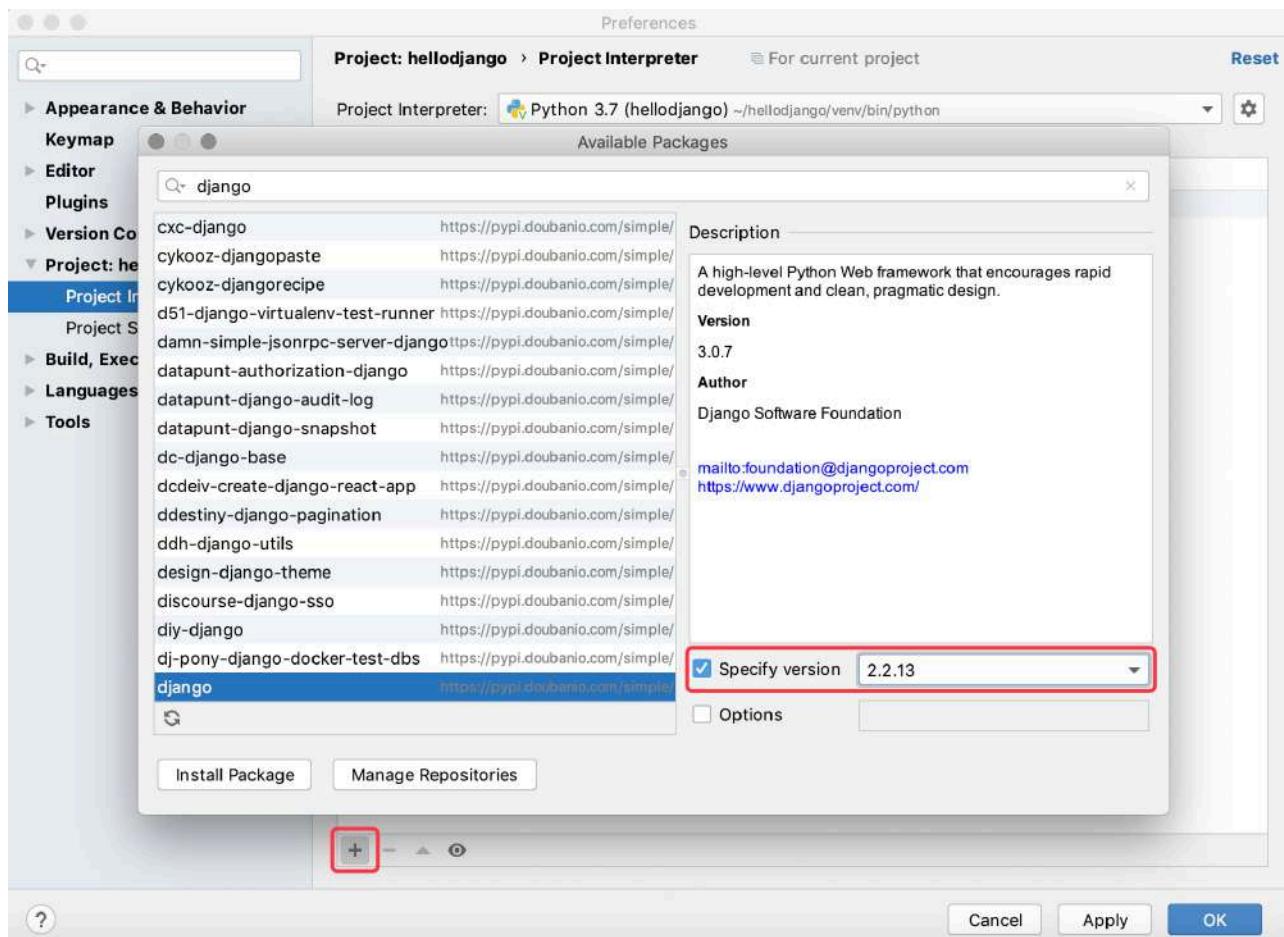
## 5. 安装项目依赖项。

方法一：打开PyCharm的终端，在终端中通过`pip`命令安装Django项目的依赖项。

**说明：**由于已经基于Python 3解释器环境为项目创建了虚拟环境，所以虚拟环境中的`python`命令对应的是Python 3的解释器，而`pip`命令对应的是Python 3的包管理工具。

```
pip install django==2.2.13
```

方法二：在PyCharm的偏好设置中，可以找到项目的解释器环境和已经安装的三方库，可以通过点击添加按钮来安装新的依赖项，需要提醒大家的是在安装Django依赖项时，需要指定版本号，否则将默认安装更新本文时最新的3.0.7版本。



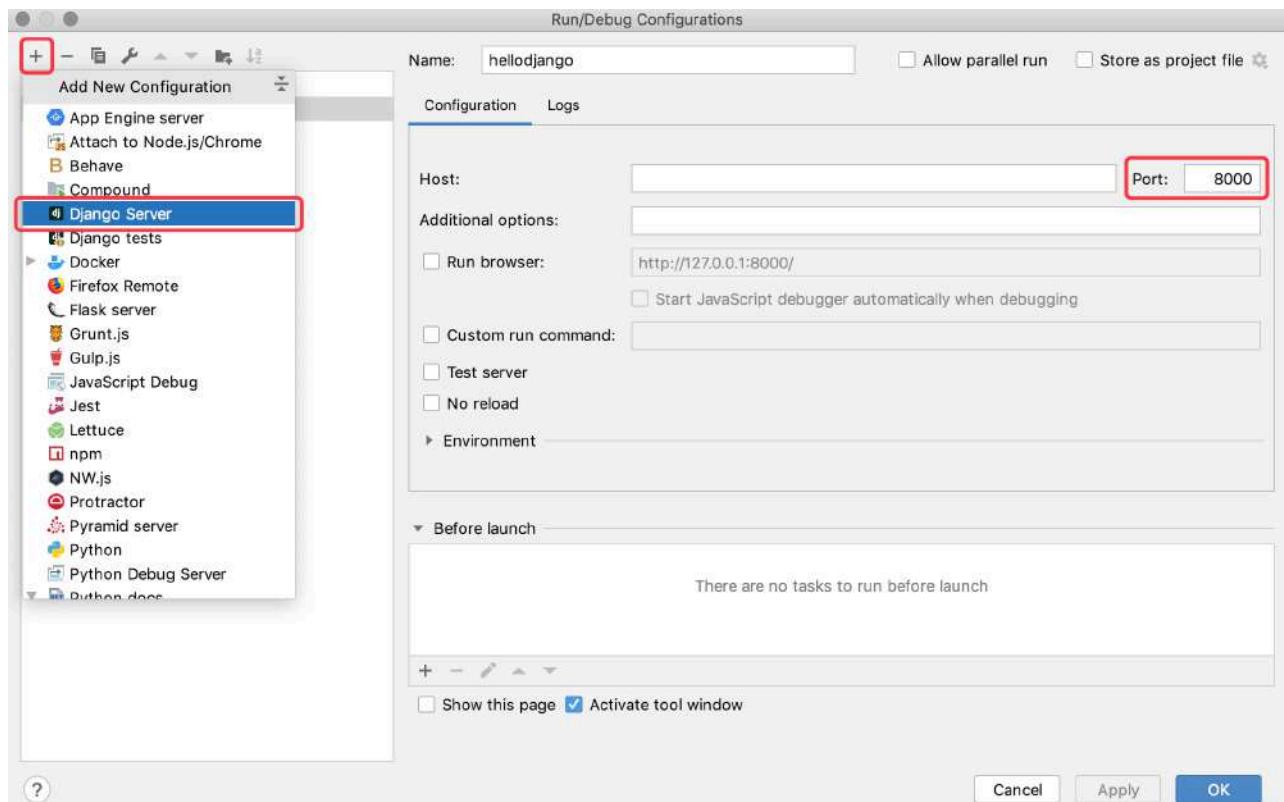
下图展示了Django版本和Python版本的对应关系，请大家自行对号入座。

### Django版本    Python版本

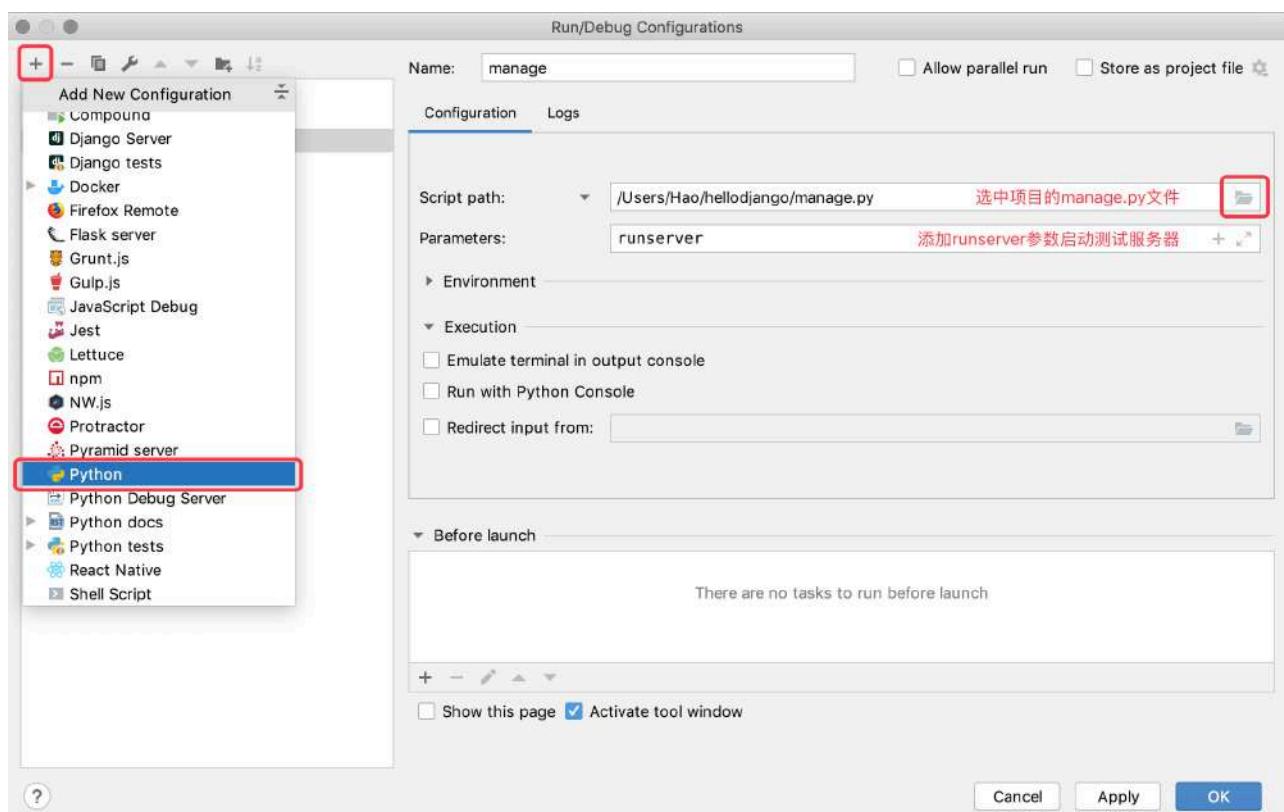
1.8	2.7、3.2、3.3、3.4、3.5
1.9、1.10	2.7、3.4、3.5
1.11	2.7、3.4、3.5、3.6、3.7 (Django 1.11.17)
2.0	3.4、3.5、3.6、3.7
2.1	3.5、3.6、3.7
2.2	3.5、3.6、3.7、3.8 (Django 2.2.8)
3.0	3.6、3.7、3.8

## 6. 启动Django自带的服务器运行项目。

方法一：在“Run”菜单选择“Edit Configuration”，配置“Django server”运行项目（适用于专业版PyCharm）。



方法二：在“Run”菜单选择“Edit Configuration”，配置运行“Python”程序运行项目（适用于专业版和社区版PyCharm）。



方法三：在PyCharm的终端（Terminal）中通过命令运行项目（适用于专业版和社区版PyCharm）。

```
python manage.py runserver
```

## 7. 查看运行效果。

在浏览器中输入<http://127.0.0.1:8000>访问我们的服务器，效果如下图所示。

The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

[Django Documentation](#) [Tutorial: A Polling App](#) [Django Community](#)

### 说明：

1. 刚刚启动的Django自带的服务器只能用于开发和测试环境，因为这个服务器是纯Python编写的轻量级Web服务器，不适合在生产环境中使用。
2. 如果修改了代码，不需要为了让修改的代码生效而重新启动Django自带的服务器。但是，在添加新的项目文件时，该服务器不会自动重新加载，这个时候就得手动重启服务器。
3. 可以在终端中通过`python manage.py help`命令查看Django管理脚本程序可用的命令参数。
4. 使用`python manage.py runserver`启动服务器时，可以在后面添加参数来指定IP地址和端口号，默认情况下启动的服务器将运行在本机的`8000`端口。
5. 在终端中运行的服务器，可以通过`Ctrl+C`来停止它。通过PyCharm的“运行配置”运行的服务器直接点击窗口上的关闭按钮就可以终止服务器的运行。
6. 不能在同一个端口上启动多个服务器，因为会导致地址的冲突（端口是对IP地址的扩展，也是计算机网络地址的一部分）。

### 8. 修改项目的配置文件`settings.py`。

Django是一个支持国际化和本地化的框架，因此刚才我们看到的Django项目的默认首页也是支持国际化的，我们可以通过修改配置文件将默认语言修改为中文，时区设置为东八区。

找到修改前的配置（在`settings.py`文件第100行以后）。

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
```

修改为以下内容。

```
LANGUAGE_CODE = 'zh-hans'
TIME_ZONE = 'Asia/Chongqing'
```

刷新刚才的页面，可以看到修改语言代码和时区之后的结果。

[django](#)

[查看 Django 2.0 的 release notes](#)



您现在看见这个页面，因为您设置了 `DEBUG=True` 并且  
您还没有配置任何 URLs。



[Django 文档](#)  
主题, 参考和指南



[教程: 投票应用](#)  
开始使用 Django



[Django 社区](#)  
联系, 获取帮助, 贡献代码

## 创建自己的应用

如果要开发自己的Web应用，需要先在Django项目中创建“应用”，一个Django项目可以包含一个或多个应用。

1. 在PyCharm的终端中执行下面的命令，创建名为 `first` 的应用。

```
python manage.py startapp first
```

执行上面的命令会在当前路径下创建 `first` 目录，其目录结构如下所示：

- `__init__.py`: 一个空文件，告诉Python解释器这个目录应该被视为一个Python的包。

- `admin.py`: 可以用来注册模型，用于在Django框架自带的管理后台中管理模型。
- `apps.py`: 当前应用的配置文件。
- `migrations`: 存放与模型有关的数据库迁移信息。
  - `__init__.py`: 一个空文件，告诉Python解释器这个目录应该被视为一个Python的包。
- `models.py`: 存放应用的数据模型 (MTV中的M) 。
- `tests.py`: 包含测试应用各项功能的测试类和测试函数。
- `views.py`: 处理用户HTTP请求并返回HTTP响应的函数或类 (MTV中的V) 。

2. 修改应用目录下的视图文件`views.py`。

```
from django.http import HttpResponse

def show_index(request):
 return HttpResponse('<h1>Hello, Django!</h1>')
```

3. 修改Django项目目录下的`urls.py`文件，将视图函数和用户在浏览器中请求的路径对应。

```
from django.contrib import admin
from django.urls import path, include

from first.views import show_index

urlpatterns = [
 path('admin/', admin.site.urls),
 path('hello/', show_index),
]
```

4. 重新运行项目，并打开浏览器中访问`http://127.0.0.1:8000/hello/`。

5. 上面我们通过代码为浏览器生成了内容，但仍然是静态内容，如果要生成动态内容，可以修改`views.py`文件并添加如下所示的代码。

```
from random import sample

from django.http import HttpResponse

def show_index(request):
 fruits = [
 'Apple', 'Orange', 'Pitaya', 'Durian', 'Waxberry', 'Blueberry',
 'Grape', 'Peach', 'Pear', 'Banana', 'Watermelon', 'Mango'
]
 selected_fruits = sample(fruits, 3)
 content = '<h3>今天推荐的水果是: </h3>'
 content += '<hr>'
 content += ''
```

```

for fruit in selected_fruits:
 content += f'{fruit}'
content += ''
return HttpResponse(content)

```

6. 刷新页面查看程序的运行结果，看看每次刷新的网页的时候，是不是可以看到不一样的内容。

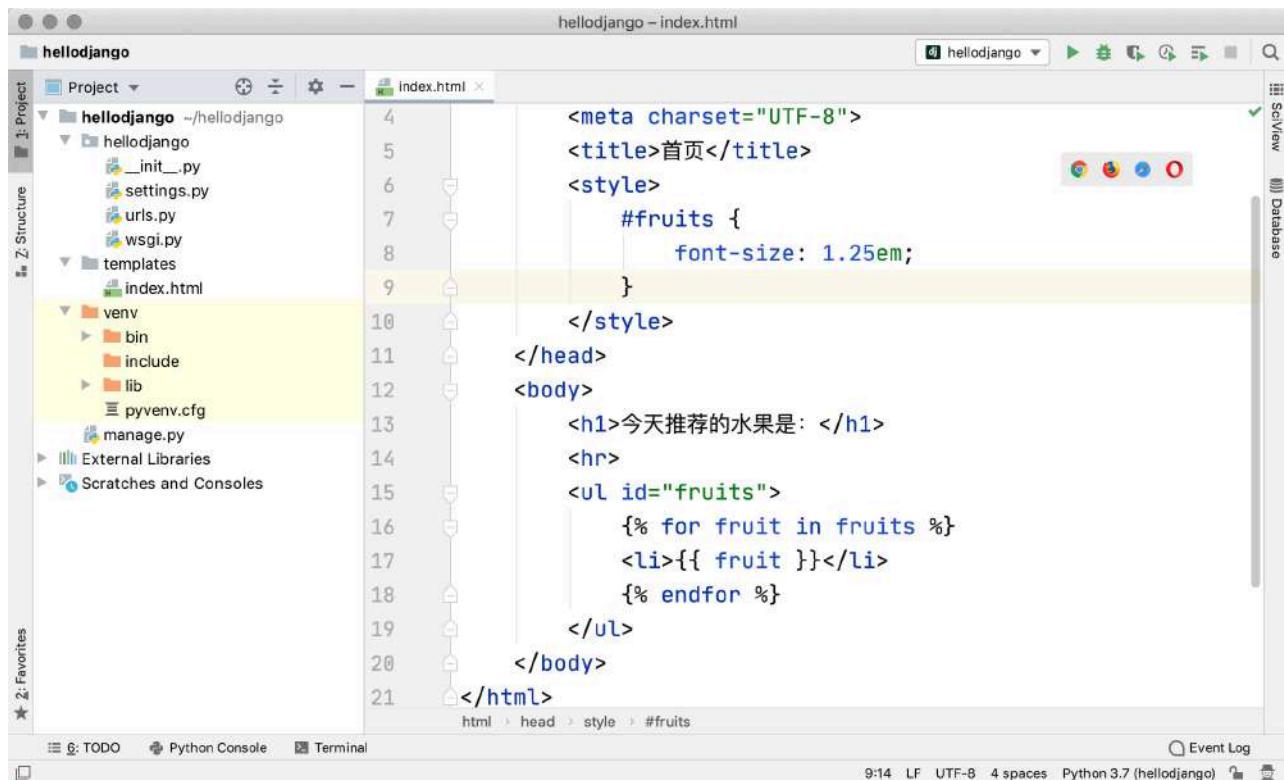
## 使用模板

上面通过拼接HTML代码的方式为浏览器生成动态内容的做法在实际开发中是无能接受的，因为实际项目中的前端页面可能非常复杂，无法用这种拼接动态内容的方式来完成，这一点大家一定能够想到。为了解决这个问题，我们可以提前准备一个模板页（MTV中的T），所谓模板页就是一个带占位符和模板指令的HTML页面。

Django框架中有一个名为`render`的便捷函数可以来完成渲染模板的操作。所谓的渲染就是用数据替换掉模板页中的模板指令和占位符，当然这里的渲染称为后端渲染，即在服务器端完成页面的渲染再输出到浏览器中。后端渲染的做法在Web应用的访问量较大时，会让服务器承受较大的负担，所以越来越多的Web应用会选择前端渲染的方式，即服务器只提供页面所需的数据（通常是JSON格式），在浏览器中通过JavaScript代码获取这些数据并渲染页面上。关于前端渲染的内容，我们会在后续的课程中为大家讲解，目前我们使用的是通过模板页进行后端渲染的做法，具体步骤如下所示。

使用模板页的步骤如下所示。

1. 在项目目录下创建名为`templates`文件夹。



2. 添加模板页`index.html`。

**说明：**实际项目开发中，静态页由前端开发者提供，后端开发者需要将静态页修改为模板页，以便通过Python程序对其进行渲染，这种做法就是上面提到的后端渲染。

```

<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="UTF-8">
 <title>首页</title>
 <style>
 #fruits {
 font-size: 1.25em;
 }
 </style>
 </head>
 <body>
 <h1>今天推荐的水果是: </h1>
 <hr>
 <ul id="fruits">
 {% for fruit in fruits %}
 {{ fruit }}
 {% endfor %}

 </body>
</html>

```

在上面的模板页中我们使用了`{{ fruit }}`这样的模板占位符语法，也使用了`{% for %}`这样的模板指令，这些都是Django模板语言 (DTL) 的一部分。关于模板语法和指令，大家可以看看官方文档，相信这些内容还是很容易理解的，并不需要过多的赘述，大家也可以参考[官方文档](#)了解模板指令和语法。

### 3. 修改`views.py`文件，调用`render`函数渲染模板页。

```

from random import sample

from django.shortcuts import render

def show_index(request):
 fruits = [
 'Apple', 'Orange', 'Pitaya', 'Durian', 'Waxberry', 'Blueberry',
 'Grape', 'Peach', 'Pear', 'Banana', 'Watermelon', 'Mango'
]
 selected_fruits = sample(fruits, 3)
 return render(request, 'index.html', {'fruits': selected_fruits})

```

`render`函数的第一个参数是请求对象`request`，第二个参数是我们要渲染的模板页的名字，第三个参数是要渲染到页面上的数据，我们通过一个字典将数据交给模板页，字典中的键就是模板页中使用的模板指令或占位符中的变量名。

### 4. 到此为止，视图函数中的`render`还无法找到模板文件`index.html`，需要修改`settings.py`文件，配置模板文件所在的路径。修改`settings.py`文件，找到`TEMPLATES`配置，修改其中的`DIRS`配置。

```
TEMPLATES = [
{
 'BACKEND': 'django.template.backends.django.DjangoTemplates',
 'DIRS': [os.path.join(BASE_DIR, 'templates'),],
 'APP_DIRS': True,
 'OPTIONS': {
 'context_processors': [
 'django.template.context_processors.debug',
 'django.template.context_processors.request',
 'django.contrib.auth.context_processors.auth',
 'django.contrib.messages.context_processors.messages',
],
 },
},
]
```

5. 重新运行项目或直接刷新页面查看结果。

## 总结

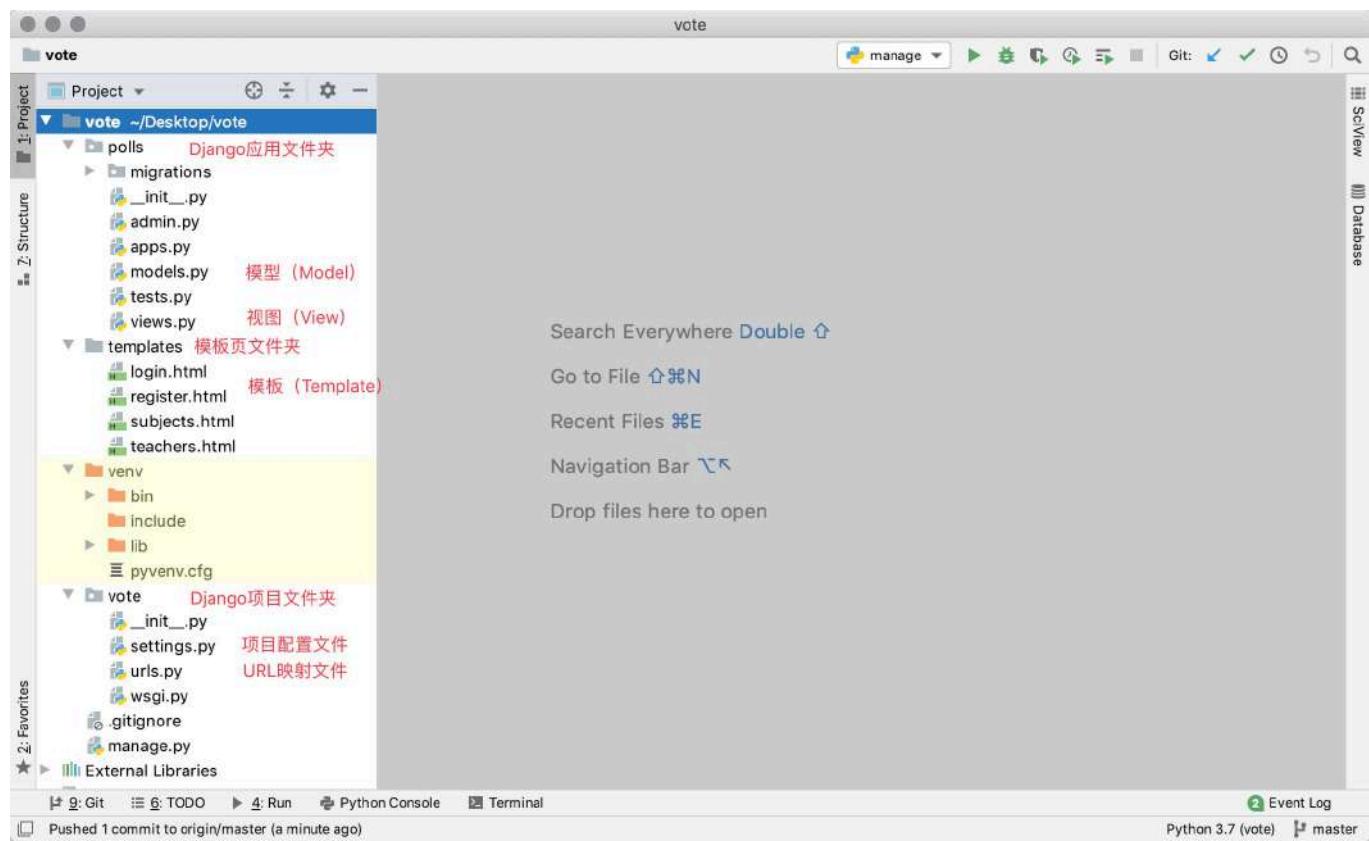
至此，我们已经利用Django框架完成了一个非常小的Web应用，虽然它并没有任何的实际价值，但是可以通过这个项目对Django框架有一个感性的认识。学习Django最好的资料肯定是它的[官方文档](#)，官方文档提供了对多国语言的支持，而且有新手教程引导初学者学习使用Django框架，建议大家通过阅读Django的官方文档来学习和使用这个框架。当然图灵社区出版的《[Django基础教程](#)》也是非常适合初学者的入门级读物，有兴趣的读者可以点击链接进行购买。

# 深入模型

在上一个章节中，我们提到了Django是基于MVC架构的Web框架，MVC架构追求的是“模型”和“视图”的解耦合。所谓“模型”说得更直白一些就是数据（的表示），所以通常也被称作“数据模型”。在实际的项目中，数据模型通常通过数据库实现持久化操作，而关系型数据库在过去和当下都是持久化的首选方案，下面我们通过完成一个投票项目来讲解和模型相关的知识点。投票项目的首页会展示某在线教育平台所有的学科；点击学科可以查看到该学科的老师及其信息；用户登录后在查看老师的页面为老师投票，可以投赞成票和反对票；未登录的用户可以通过登录页进行登录；尚未注册的用户可以通过注册页输入个人信息进行注册。在这个项目中，我们使用MySQL数据库来实现数据持久化操作。

## 创建项目和应用

我们首先创建Django项目vote并为其添加虚拟环境和依赖项。接下来，在项目下创建名为polls的应用和保存模板页的文件夹tempaltes，项目文件夹的结构如下所示。



根据上面描述的项目需求，我们准备了四个静态页面，分别是展示学科的页面subjects.html，显示学科老师的页面teachers.html，登录页面login.html，注册页面register.html，稍后我们会将静态页修改为Django项目所需的模板页。

## 配置关系型数据库MySQL

1. 在MySQL中创建数据库，创建用户，授权用户访问该数据库。

```
create database vote default charset utf8;
create user 'hellokitty'@'%' identified by 'Hellokitty.618';
```

```
grant all privileges on vote.* to 'hellokitty'@'%';
flush privileges;
```

2. 在MySQL中创建保存学科和老师信息的二维表（保存用户信息的表稍后处理）。

```
use vote;

-- 创建学科表
create table `tb_subject`
(
 `no` integer auto_increment comment '学科编号',
 `name` varchar(50) not null comment '学科名称',
 `intro` varchar(1000) not null default '' comment '学科介绍',
 `is_hot` boolean not null default 0 comment '是不是热门学科',
 primary key (`no`)
);
-- 创建老师表
create table `tb_teacher`
(
 `no` integer auto_increment comment '老师编号',
 `name` varchar(20) not null comment '老师姓名',
 `sex` boolean not null default 1 comment '老师性别',
 `birth` date not null comment '出生日期',
 `intro` varchar(1000) not null default '' comment '老师介绍',
 `photo` varchar(255) not null default '' comment '老师照片',
 `gcount` integer not null default 0 comment '好评数',
 `bcount` integer not null default 0 comment '差评数',
 `sno` integer not null comment '所属学科',
 primary key (`no`),
 foreign key (`sno`) references `tb_subject` (`no`)
);
```

3. 在虚拟环境中安装连接MySQL数据库所需的依赖项。

```
pip install mysqlclient
```

**说明：**如果因为某些原因无法安装mysqlclient三方库，可以使用它的替代品pymysql，pymysql是用纯Python开发的连接MySQL的Python库，安装更容易成功，但是需要在Django项目文件夹的\_\_init\_\_.py中添加如下所示的代码。

```
import pymysql
pymysql.install_as_MySQLdb()
```

如果使用Django 2.2及以上版本，还会遇到PyMySQL跟Django框架的兼容性问题，兼容性问题会导致项目无法运行，需要按照GitHub上PyMySQL仓库Issues中提供的方法进行处理。总体来说，

使用`PyMySQL`会比较麻烦，强烈建议大家首选安装`mysqlclient`。

4. 修改项目的`settings.py`文件，首先将我们创建的应用`polls`添加已安装的项目（`INSTALLED_APPS`）中，然后配置MySQL作为持久化方案。

```
INSTALLED_APPS = [
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.messages',
 'django.contrib.staticfiles',
 'polls',
]

DATABASES = {
 'default': {
 # 数据库引擎配置
 'ENGINE': 'django.db.backends.mysql',
 # 数据库的名字
 'NAME': 'vote',
 # 数据库服务器的IP地址（本机可以写localhost或127.0.0.1）
 'HOST': 'localhost',
 # 启动MySQL服务的端口号
 'PORT': 3306,
 # 数据库用户名和口令
 'USER': 'hellokitty',
 'PASSWORD': 'Hellokitty.618',
 # 数据库使用的字符集
 'CHARSET': 'utf8',
 # 数据库时间日期的时区设定
 'TIME_ZONE': 'Asia/Chongqing',
 }
}
```

在配置`ENGINE`属性时，常用的可选值包括：

- `'django.db.backends.sqlite3'`：SQLite嵌入式数据库。
- `'django.db.backends.postgresql'`：BSD许可证下发行的开源关系型数据库产品。
- `'django.db.backends.mysql'`：甲骨文公司经济高效的数据库产品。
- `'django.db.backends.oracle'`：甲骨文公司关系型数据库旗舰产品。

其他的配置可以参考官方文档中[数据库配置](#)的部分。

5. Django框架提供了ORM来解决数据持久化问题，ORM翻译成中文叫“对象关系映射”。因为Python是面向对象的编程语言，我们在Python程序中使用对象模型来保存数据，而关系型数据库使用关系模型，用二维表来保存数据，这两种模型并不匹配。使用ORM是为了实现对象模型到关系模型的**双向转换**，这样就不需要在Python代码中书写SQL语句和游标操作，因为这些都会由ORM自动完成。利用Django的ORM，我们可以直接将刚才创建的学科表和老师表变成Django中的模型类。

```
python manage.py inspectdb > polls/models.py
```

我们可以对自动生成的模型类稍作调整，代码如下所示。

```
from django.db import models

class Subject(models.Model):
 no = models.AutoField(primary_key=True, verbose_name='编号')
 name = models.CharField(max_length=50, verbose_name='名称')
 intro = models.CharField(max_length=1000, verbose_name='介绍')
 is_hot = models.BooleanField(verbose_name='是否热门')

 class Meta:
 managed = False
 db_table = 'tb_subject'

class Teacher(models.Model):
 no = models.AutoField(primary_key=True, verbose_name='编号')
 name = models.CharField(max_length=20, verbose_name='姓名')
 sex = models.BooleanField(default=True, verbose_name='性别')
 birth = models.DateField(verbose_name='出生日期')
 intro = models.CharField(max_length=1000, verbose_name='个人介绍')
 photo = models.ImageField(max_length=255, verbose_name='照片')
 good_count = models.IntegerField(default=0, db_column='gcount',
 verbose_name='好评数')
 bad_count = models.IntegerField(default=0, db_column='bcount',
 verbose_name='差评数')
 subject = models.ForeignKey(Subject, models.DO_NOTHING, db_column='sno')

 class Meta:
 managed = False
 db_table = 'tb_teacher'
```

**说明：**模型类都直接或间接继承自**Model**类，模型类跟关系型数据库的二维表对应，模型对象跟表中的记录对应，模型对象的属性跟表中的字段对应。如果对上面模型类的属性定义不是特别理解，可以看看本文后面提供的“模型定义参考”部分的内容。

## 使用ORM完成模型的CRUD操作

有了Django框架的ORM，我们可以直接使用面向对象的方式来实现对数据的CRUD（增删改查）操作。我们可以在PyCharm的终端中输入下面的命令进入到Django项目的交互式环境，然后尝试对模型的操作。

```
python manage.py shell
```

## 新增

```
from polls.models import Subject

subject1 = Subject(name='Python全栈开发', intro='当下最热门的学科', is_hot=True)
subject1.save()
subject2 = Subject(name='全栈软件测试', intro='学习自动化测试的学科', is_hot=False)
subject2.save()
subject3 = Subject(name='JavaEE分布式开发', intro='基于Java语言的服务器应用开发',
is_hot=True)
```

## 删除

```
subject = Subject.objects.get(no=2)
subject.delete()
```

## 更新

```
subject = Subject.objects.get(no=1)
subject.name = 'Python全栈+人工智能'
subject.save()
```

## 查询

1. 查询所有对象。

```
Subjects.objects.all()
```

2. 过滤数据。

```
查询名称为“Python全栈+人工智能”的学科
Subject.objects.filter(name='Python全栈+人工智能')

查询名称包含“全栈”的学科（模糊查询）
Subject.objects.filter(name__contains='全栈')
Subject.objects.filter(name__startswith='全栈')
Subject.objects.filter(name__endswith='全栈')

查询所有热门学科
Subject.objects.filter(is_hot=True)

查询编号大于3小于10的学科
```

```
Subject.objects.filter(no__gt=3).filter(no__lt=10)
Subject.objects.filter(no__gt=3, no__lt=10)

查询编号在3到7之间的学科
Subject.objects.filter(no__ge=3, no__le=7)
Subject.objects.filter(no__range=(3, 7))
```

### 3. 查询单个对象。

```
查询主键为1的学科
Subject.objects.get(pk=1)
Subject.objects.get(no=1)
Subject.objects.filter(no=1).first()
Subject.objects.filter(no=1).last()
```

### 4. 排序。

```
查询所有学科按编号升序排列
Subject.objects.order_by('no')
查询所有部门按部门编号降序排列
Subject.objects.order_by('-no')
```

### 5. 切片（分页查询）。

```
按编号从小到大查询前3个学科
Subject.objects.order_by('no')[:3]
```

### 6. 计数。

```
查询一共有多少个学科
Subject.objects.count()
```

### 7. 高级查询。

```
查询编号为1的学科的老师
Teacher.objects.filter(subject__no=1)
Subject.objects.get(pk=1).teacher_set.all()

查询学科名称有“全栈”二字的学科的老师
Teacher.objects.filter(subject__name__contains='全栈')
```

**说明1：**由于老师与学科之间存在多对一外键关联，所以能通过学科反向查询到该学科的老师（从一对多关系中“一”的一方查询“多”的一方），反向查询属性默认的名字是类名小写\_set（如上面例子中的teacher\_set），当然也可以在创建模型时通过ForeignKey的related\_name属性指定反向查询属性的名字。如果不希望执行反向查询可以将related\_name属性设置为'+'或者以'+'开头的字符串。

**说明2：**ORM查询多个对象时会返回QuerySet对象，QuerySet使用了惰性查询，即在创建QuerySet对象的过程中不涉及任何数据库活动，等真正用到对象时（对QuerySet求值）才向数据库发送SQL语句并获取对应的结果，这一点在实际开发中需要引起注意！

**说明3：**如果希望更新多条数据，不用先逐一获取模型对象再修改对象属性，可以直接使用QuerySet对象的update()方法一次性更新多条数据。

## 利用Django后台管理模型

在创建好模型类之后，可以通过Django框架自带的后台管理应用（admin应用）实现对模型的管理。虽然实际应用中，这个后台可能并不能满足我们的需求，但是在学习Django框架时，我们可以利用admin应用来管理我们的模型，同时也通过它来了解一个项目的后台管理系统需要哪些功能。使用Django自带的admin应用步骤如下所示。

1. 将admin应用所需的表迁移到数据库中。admin应用本身也需要数据库的支持，而且在admin应用中已经定义好了相关的数据模型类，我们只需要通过模型迁移操作就能自动在数据库中生成所需的二维表。

```
python manage.py migrate
```

2. 创建访问admin应用的超级用户账号，这里需要输入用户名、邮箱和口令。

```
python manage.py createsuperuser
```

**说明：**输入口令时没有回显也不能退格，需要一气呵成完成输入。

3. 运行项目，在浏览器中访问http://127.0.0.1:8000/admin，输入刚才创建的超级用户账号和密码进行登录。

登录后进入管理员操作平台。



注意，我们暂时还没能在`admin`应用中看到之前创建的模型类，为此需要在`polls`应用的`admin.py`文件中对需要管理的模型进行注册。

#### 4. 注册模型类。

```
from django.contrib import admin
from polls.models import Subject, Teacher
admin.site.register(Subject)
admin.site.register(Teacher)
```

注册模型类后，就可以在后台管理系统中看到它们。

## 5. 对模型进行CRUD操作。

可以在管理员平台对模型进行C（新增）、R（查看）、U（更新）、D（删除）操作，如下图所示。

- 添加学科。

- 查看所有学科。

选择 学科 来修改

动作  执行 4 个中 0 个被选

- 学科
- Subject object (6)
- Subject object (5)
- Subject object (2)
- Subject object (1)

4 学科

- 删除和更新学科。

修改 学科

名称: Python全栈开发

介绍: 最近十年最热门的学科

是否热门

删除 保存并增加另一个 保存并继续编辑 保存

## 6. 注册模型管理类。

可能大家已经注意到了，刚才在后台查看部门信息的时候，显示的部门信息并不直观，为此我们再修改 `admin.py` 文件，通过注册模型管理类，可以在后台管理系统中更好的管理模型。

```
from django.contrib import admin

from polls.models import Subject, Teacher

class SubjectModelAdmin(admin.ModelAdmin):
 list_display = ('no', 'name', 'intro', 'is_hot')
```

```
search_fields = ('name',)
ordering = ('no',)

class TeacherModelAdmin(admin.ModelAdmin):
 list_display = ('no', 'name', 'sex', 'birth', 'good_count', 'bad_count',
'subject')
 search_fields = ('name',)
 ordering = ('no',)

admin.site.register(Subject, SubjectModelAdmin)
admin.site.register(Teacher, TeacherModelAdmin)
```



动作	编号	名称	介绍	是否热门
<input type="checkbox"/>	1	Python全栈开发	最近十年最热门的学科	<input checked="" type="checkbox"/>
<input type="checkbox"/>	2	全栈自动化软件测试	学习自动化测试的学科	<input type="checkbox"/>
<input type="checkbox"/>	5	H5大前端开发	学习开发PC端和移动端用户界面的学科	<input checked="" type="checkbox"/>
<input type="checkbox"/>	6	JavaEE分布式开发	基于Java平台做开发的学科	<input checked="" type="checkbox"/>

为了更好的查看模型，我们为Subject类添加`__str__`魔法方法，并在该方法中返回学科名字。这样在如上图所示的查看老师的页面上显示老师所属学科时，就不再是Subject object(1)这样晦涩的信息，而是学科的名称。

## 实现学科页和老师页效果

1. 修改`polls/views.py`文件，编写视图函数实现对学科页和老师页的渲染。

```
from django.shortcuts import render, redirect

from polls.models import Subject, Teacher

def show_subjects(request):
 subjects = Subject.objects.all().order_by('no')
 return render(request, 'subjects.html', {'subjects': subjects})

def show_teachers(request):
 try:
 sno = int(request.GET.get('sno'))
 teachers = []
 if sno:
 subject = Subject.objects.only('name').get(no=sno)
 teachers =
 Teacher.objects.filter(subject=subject).order_by('no')
 return render(request, 'teachers.html', {
 'subject': subject,
 'teachers': teachers
 })

```

```
except (ValueError, Subject.DoesNotExist):
 return redirect('/')
```

2. 修改`templates/subjects.html`和`templates/teachers.html`模板页。

`subjects.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>学科信息</title>
 <style>
 #container {
 width: 80%;
 margin: 10px auto;
 }
 .user {
 float: right;
 margin-right: 10px;
 }
 .user>a {
 margin-right: 10px;
 }
 #main>dl>dt {
 font-size: 1.5em;
 font-weight: bold;
 }
 #main>dl>dd {
 font-size: 1.2em;
 }
 a {
 text-decoration: none;
 color: darkcyan;
 }
 </style>
</head>
<body>
 <div id="container">
 <div class="user">
 用户登录
 快速注册
 </div>
 <h1>扣丁学堂所有学科</h1>
 <hr>
 <div id="main">
 {% for subject in subjects %}
 <dl>
 <dt>
 {{ subject.name }}
 <dt>
 {% if subject.is_hot %}</dt>
```

```

 {% endif %}
 </dt>
 <dd>{{ subject.intro }}</dd>
</dl>
{% endfor %}
</div>
</div>
</body>
</html>
```

## teachers.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>老师信息</title>
 <style>
 #container {
 width: 80%;
 margin: 10px auto;
 }
 .teacher {
 width: 100%;
 margin: 0 auto;
 padding: 10px 0;
 border-bottom: 1px dashed gray;
 overflow: auto;
 }
 .teacher>div {
 float: left;
 }
 .photo {
 height: 140px;
 border-radius: 75px;
 overflow: hidden;
 margin-left: 20px;
 }
 .info {
 width: 75%;
 margin-left: 30px;
 }
 .info div {
 clear: both;
 margin: 5px 10px;
 }
 .info span {
 margin-right: 25px;
 }
 .info a {
 text-decoration: none;
 }
 </style>
</head>
<body>
 <div id="container">
 <div class="teacher">
 <div>

 <div class="photo"></div>
 </div>
 <div class="info">
 <div>
 姓名: 张三
 性别: 男
 </div>
 <div>
 年龄: 30
 职称: 教师
 </div>
 <div>
 电话: 1234567890
 邮箱: zhangsan@example.com
 </div>
 </div>
 <div>
 更多>
 </div>
 </div>
 </div>
</body>
</html>
```

```

 color: darkcyan;
 }

```

```

</style>
</head>
<body>
 <div id="container">
 <h1>{{ subject.name }}学科的老师信息</h1>
 <hr>
 {% if not teachers %}
 <h2>暂无该学科老师信息</h2>
 {% endif %}
 {% for teacher in teachers %}
 <div class="teacher">
 <div class="photo">

 </div>
 <div class="info">
 <div>
 姓名: {{ teacher.name }}
 性别: {{ teacher.sex | yesno:'男,女' }}
 出生日期: {{ teacher.birth | date:'Y年n月j日' }}
 </div>
 <div class="intro">{{ teacher.intro }}</div>
 <div class="comment">
 好评&nbsp({{ teacher.good_count }})
 &nbsp&nbsp&nbsp&nbsp&nbsp
 差评&nbsp{{ teacher.bad_count }}
 </div>
 </div>
 </div>
 {% endfor %}
 返回首页
 </div>
</body>
</html>

```

### 3. 修改vote/urls.py文件，实现映射URL。

```

from django.contrib import admin
from django.urls import path

from polls.views import show_subjects, show_teachers

urlpatterns = [
 path('admin/', admin.site.urls),
 path('', show_subjects),

```

```

 path('teachers/', show_teachers),
]

```

到此为止，页面上需要的图片（静态资源）还没有能够正常展示，我们在下一章节中为大家介绍如何处理模板页上的需要的静态资源。

## 补充内容

### Django模型最佳实践

1. 正确的为模型和关系字段命名。
2. 设置适当的`related_name`属性。
3. 用`OneToOneField`代替`ForeignKeyField(unique=True)`。
4. 通过“迁移操作”（migrate）来添加模型。
5. 用NoSQL来应对需要降低范式级别的场景。
6. 如果布尔类型可以为空要使用`NullBooleanField`。
7. 在模型中放置业务逻辑。
8. 用`<ModelName>.DoesNotExist`取代`ObjectDoesNotExist`。
9. 在数据库中不要出现无效数据。
10. 不要对`QuerySet`调用`len()`函数。
11. 将`QuerySet`的`exists()`方法的返回值用于`if`条件。
12. 用`DecimalField`来存储货币相关数据而不是`FloatField`。
13. 定义`__str__`方法。
14. 不要将数据文件放在同一个目录中。

**说明：**以上内容来自于STEELKIWI网站的[Best Practice working with Django models in Python](#)，有兴趣的小伙伴可以阅读原文。

## 模型定义参考

### 字段

对字段名称的限制

- 字段名不能是Python的保留字，否则会导致语法错误
- 字段名不能有多个连续下划线，否则影响ORM查询操作

### Django模型字段类

字段类	说明
<code>AutoField</code>	自增ID字段
<code>BigIntegerField</code>	64位有符号整数
<code>BinaryField</code>	存储二进制数据的字段，对应Python的 <code>bytes</code> 类型
<code>BooleanField</code>	存储 <code>True</code> 或 <code>False</code>
<code>CharField</code>	长度较小的字符串

字段类	说明
TextField	存储日期, 有 <code>auto_now</code> 和 <code>auto_now_add</code> 属性
DateTimeField	存储日期和日期, 两个附加属性同上
DecimalField	存储固定精度小数, 有 <code>max_digits</code> (有效位数) 和 <code>decimal_places</code> (小数点后面) 两个必要的参数
DurationField	存储时间跨度
EmailField	与 <code>CharField</code> 相同, 可以用 <code>EmailValidator</code> 验证
FileField	文件上传字段
FloatField	存储浮点数
ImageField	其他同 <code>FileField</code> , 要验证上传的是不是有效图像
IntegerField	存储32位有符号整数。
GenericIPAddressField	存储IPv4或IPv6地址
NullBooleanField	存储 <code>True</code> 、 <code>False</code> 或 <code>null</code> 值
PositiveIntegerField	存储无符号整数 (只能存储正数)
SlugField	存储slug (简短标注)
SmallIntegerField	存储16位有符号整数
TextField	存储数据量较大的文本
TimeField	存储时间
URLField	存储URL的 <code>CharField</code>
UUIDField	存储全局唯一标识符

## 字段属性

### 通用字段属性

选项	说明
<code>null</code>	数据库中对应的字段是否允许为 <code>NULL</code> , 默认为 <code>False</code>
<code>blank</code>	后台模型管理验证数据时, 是否允许为 <code>NULL</code> , 默认为 <code>False</code>
<code>choices</code>	设定字段的选项, 各元组中的第一个值是设置在模型上的值, 第二值是人类可读的值
<code>db_column</code>	字段对应到数据库表中的列名, 未指定时直接使用字段的名称
<code>db_index</code>	设置为 <code>True</code> 时将在该字段创建索引
<code>db_tablespace</code>	为有索引的字段设置使用的表空间, 默认为 <code>DEFAULT_INDEX_TABLESPACE</code>
<code>default</code>	字段的默认值

选项	说明
editable	字段在后台模型管理或ModelForm中是否显示，默认为True
error_messages	设定字段抛出异常时的默认消息的字典，其中的键包括null、blank、invalid、invalid_choice、unique和unique_for_date
help_text	表单组件旁边显示的额外的帮助文本。
primary_key	将字段指定为模型的主键，未指定时会自动添加AutoField用于主键，只读。
unique	设置为True时，表中字段的值必须是唯一的
verbose_name	字段在后台模型管理显示的名称，未指定时使用字段的名称

### ForeignKey属性

1. `limit_choices_to`: 值是一个Q对象或返回一个Q对象，用于限制后台显示哪些对象。
2. `related_name`: 用于获取关联对象的关联管理器对象（反向查询），如果不允许反向，该属性应该被设置为'+'，或者以'+'结尾。
3. `to_field`: 指定关联的字段，默认关联对象的主键字段。
4. `db_constraint`: 是否为外键创建约束，默认值为True。
5. `on_delete`: 外键关联的对象被删除时对应的动作，可取的值包括django.db.models中定义的：
  - `CASCADE`: 级联删除。
  - `PROTECT`: 抛出ProtectedError异常，阻止删除引用的对象。
  - `SET_NULL`: 把外键设置为null，当null属性被设置为True时才能这么做。
  - `SET_DEFAULT`: 把外键设置为默认值，提供了默认值才能这么做。

### ManyToManyField属性

1. `symmetrical`: 是否建立对称的多对多关系。
2. `through`: 指定维持多对多关系的中间表的Django模型。
3. `throughfields`: 定义了中间模型时可以指定建立多对多关系的字段。
4. `db_table`: 指定维持多对多关系的中间表的表名。

### 模型元数据选项

选项	说明
<code>abstract</code>	设置为True时模型是抽象父类
<code>app_label</code>	如果定义模型的应用不在INSTALLED_APPS中可以用该属性指定
<code>db_table</code>	模型使用的数据表名称
<code>db_tablespace</code>	模型使用的数据表空间
<code>default_related_name</code>	关联对象回指这个模型时默认使用的名称，默认为<model_name>_set
<code>get_latest_by</code>	模型中可排序字段的名称。
<code>managed</code>	设置为True时，Django在迁移中创建数据表并在执行flush管理命令时把表移除

选项	说明
<code>order_with_respect_to</code>	标记对象为可排序的
<code>ordering</code>	对象的默认排序
<code>permissions</code>	创建对象时写入权限表的额外权限
<code>default_permissions</code>	默认为( <code>'add', 'change', 'delete'</code> )
<code>unique_together</code>	设定组合在一起时必须独一无二的字段名
<code>index_together</code>	设定一起建立索引的多个字段名
<code>verbose_name</code>	为对象设定人类可读的名称
<code>verbose_name_plural</code>	设定对象的复数名称

## 查询参考

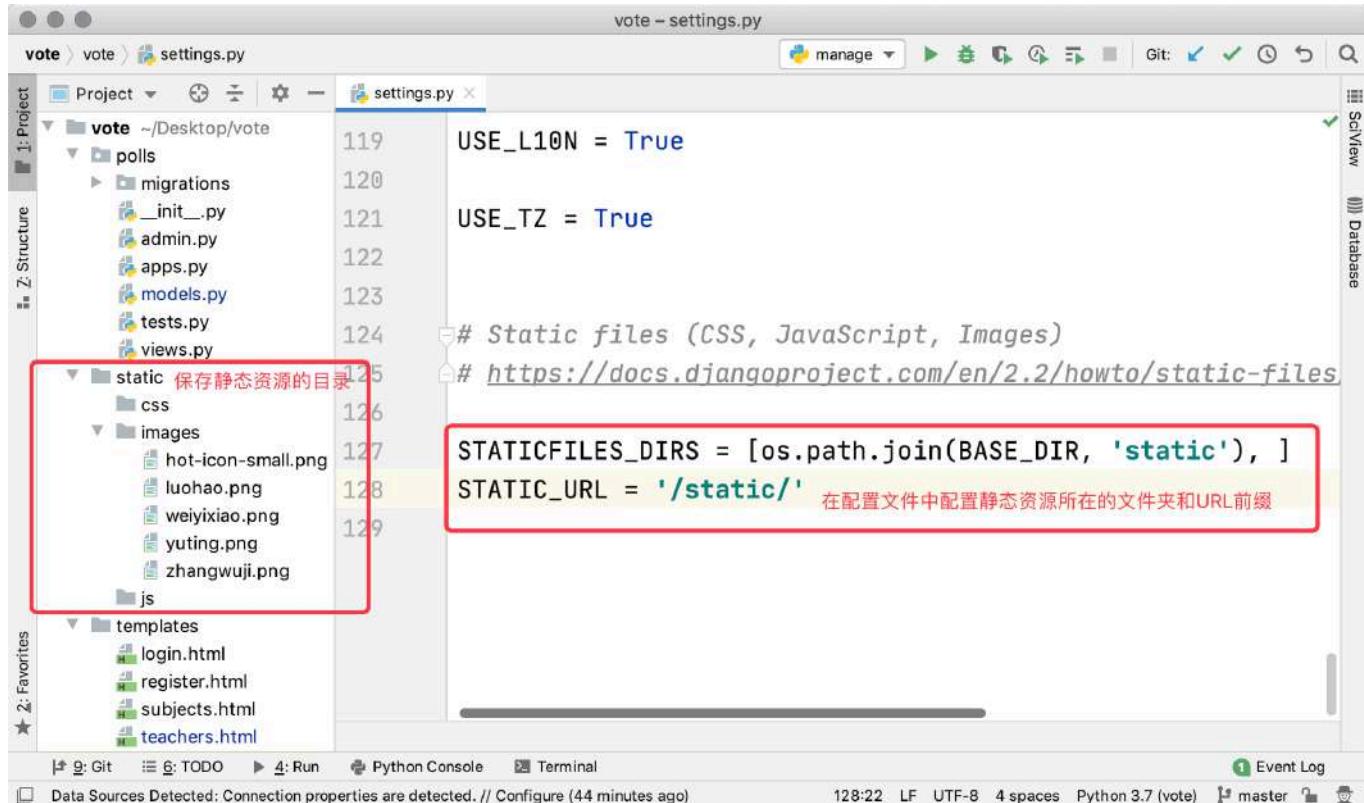
### 按字段查找可以用的条件

1. `exact` / `iexact`: 精确匹配/忽略大小写的精确匹配查询
2. `contains` / `icontains` / `startswith` / `istartswith` / `endswith` / `iendswith`: 基于`like`的模糊查询
3. `in`: 集合运算
4. `gt` / `gte` / `lt` / `lte`: 大于/大于等于/小于/小于等于关系运算
5. `range`: 指定范围查询 (SQL中的`between...and...`)
6. `year` / `month` / `day` / `week_day` / `hour` / `minute` / `second`: 查询时间日期
7. `isnull`: 查询空值 (True) 或非空值 (False)
8. `search`: 基于全文索引的全文检索 (一般很少使用)
9. `regex` / `iregex`: 基于正则表达式的模糊匹配查询

# 静态资源和Ajax请求

## 加载静态资源

如果要在Django项目中使用静态资源，可以先创建一个用于保存静态资源的目录。在vote项目中，我们将静态资源置于名为static的文件夹中，在该文件夹包含了三个子文件夹：css、js和images，分别用来保存外部CSS文件、外部JavaScript文件和图片资源，如下图所示。



为了能够找到保存静态资源的文件夹，我们还需要修改Django项目的配置文件settings.py，如下所示：

```

STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static'),]
STATIC_URL = '/static/'

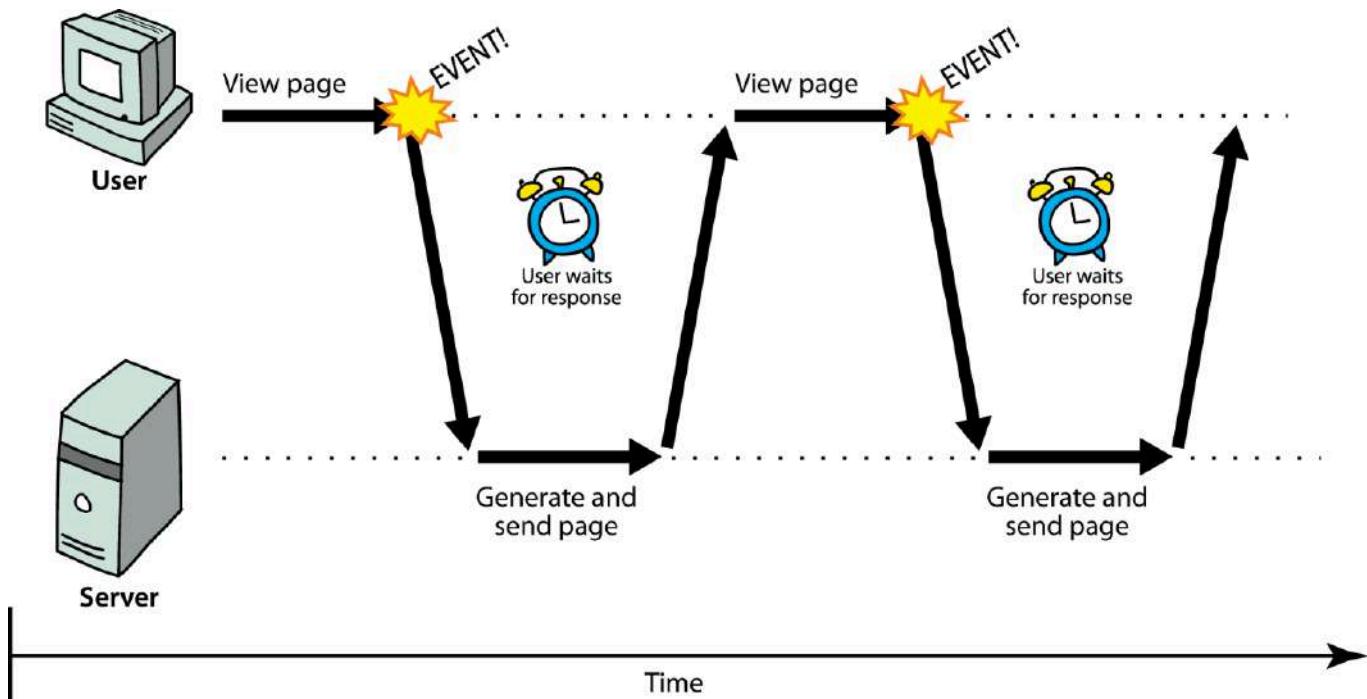
```

配置好静态资源之后，大家可以运行项目，然后看看之前我们写的页面上的图片是否能够正常加载出来。需要说明的是，在项目正式部署到线上环境后，我们通常会把静态资源交给专门的静态资源服务器（如Nginx、Apache）来处理，而不是有运行Python代码的服务器来管理静态资源，所以上面的配置并不适用于生产环境，仅供项目开发阶段测试使用。使用静态资源的正确姿势我们会在后续的章节为大家讲解。

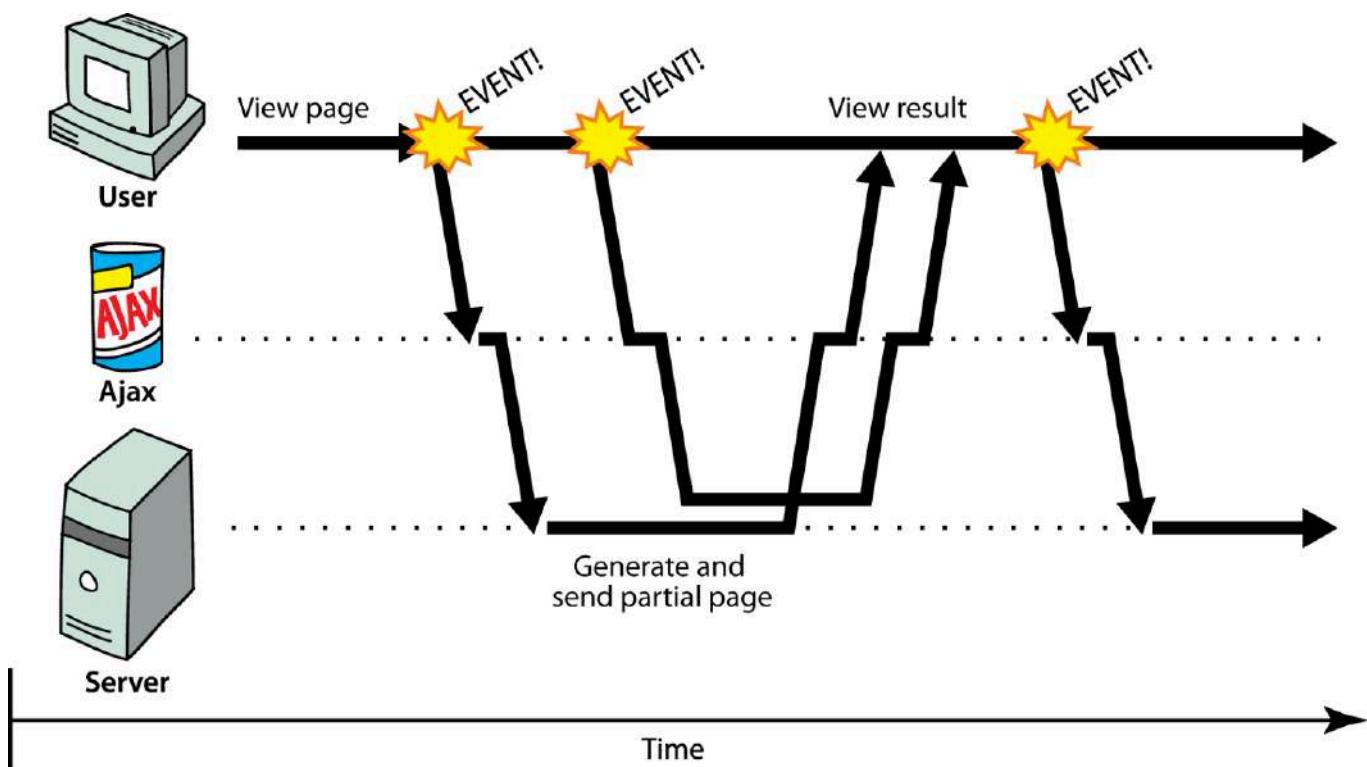
## Ajax概述

接下来就可以实现“好评”和“差评”的功能了，很明显如果能够在不刷新页面的情况下实现这两个功能会带来更好的用户体验，因此我们考虑使用Ajax技术来实现“好评”和“差评”。Ajax是Asynchronous Javascript And XML的缩写，简单的说，使用Ajax技术可以在不重新加载整个页面的情况下对页面进行局部刷新。

对于传统的Web应用，每次页面上需要加载新的内容都需要重新请求服务器并刷新整个页面，如果服务器短时间内无法给予响应或者网络状况并不理想，那么可能会造成浏览器长时间的空白并使得用户处于等待状态，在这个期间用户什么都做不了，如下图所示。很显然，这样的Web应用并不能带来很好的用户体验。



对于使用Ajax技术的Web应用，浏览器可以向服务器发起异步请求来获取数据。异步请求不会中断用户体验，当服务器返回了新的数据，我们可以通过JavaScript代码进行DOM操作来实现对页面的局部刷新，这样就相当于在不刷新整个页面的情况下更新了页面的内容，如下图所示。



在使用Ajax技术时，浏览器跟服务器通常会交换XML或JSON格式的数据，XML是以前使用得非常多的一种数据格式，近年来几乎已经完全被JSON取代，下面是两种数据格式的对比。

XML格式：

```
<?xml version="1.0" encoding="utf-8"?>
<message>
 <from>Alice</from>
```

```

<to>Bob</to>
<content>Dinner is on me!</content>
</message>

```

JSON格式：

```

{
 "from": "Alice",
 "to": "Bob",
 "content": "Dinner is on me!"
}

```

通过上面的对比，明显JSON格式的数据要紧凑得多，所以传输效率更高，而且JSON本身也是JavaScript中的一种对象表达式语法，在JavaScript代码中处理JSON格式的数据更加方便。

## 用Ajax实现投票功能

下面，我们使用Ajax技术来实现投票的功能，首先修改项目的urls.py文件，为“好评”和“差评”功能映射对应的URL。

```

from django.contrib import admin
from django.urls import path

from vote import views

urlpatterns = [
 path('', views.show_subjects),
 path('teachers/', views.show_teachers),
 path('praise/', views.praise_or_criticize),
 path('criticize/', views.praise_or_criticize),
 path('admin/', admin.site.urls),
]

```

设计视图函数praise\_or\_criticize来支持“好评”和“差评”功能，该视图函数通过Django封装的JsonResponse类将字典序列化成JSON字符串作为返回给浏览器的响应内容。

```

def praise_or_criticize(request):
 """好评"""
 try:
 tno = int(request.GET.get('tno'))
 teacher = Teacher.objects.get(no=tno)
 if request.path.startswith('/praise'):
 teacher.good_count += 1
 count = teacher.good_count
 else:
 teacher.bad_count += 1
 count = teacher.bad_count
 except:
 pass
 return JsonResponse({'count': count})

```

```
teacher.save()
data = {'code': 2000, 'mesg': '操作成功', 'count': count}
except (ValueError, Teacher.DoseNotExist):
 data = {'code': 2001, 'mesg': '操作失败'}
return JsonResponse(data)
```

修改显示老师信息的模板页，引入jQuery库来实现事件处理、Ajax请求和DOM操作。

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>老师信息</title>
 <style>
 #container {
 width: 80%;
 margin: 10px auto;
 }
 .teacher {
 width: 100%;
 margin: 0 auto;
 padding: 10px 0;
 border-bottom: 1px dashed gray;
 overflow: auto;
 }
 .teacher>div {
 float: left;
 }
 .photo {
 height: 140px;
 border-radius: 75px;
 overflow: hidden;
 margin-left: 20px;
 }
 .info {
 width: 75%;
 margin-left: 30px;
 }
 .info div {
 clear: both;
 margin: 5px 10px;
 }
 .info span {
 margin-right: 25px;
 }
 .info a {
 text-decoration: none;
 color: darkcyan;
 }
 </style>
</head>
<body>
```

```
<div id="container">
 <h1>{{ subject.name }}学科的老师信息</h1>
 <hr>
 {% if not teachers %}
 <h2>暂无该学科老师信息</h2>
 {% endif %}
 {% for teacher in teachers %}
 <div class="teacher">
 <div class="photo">

 </div>
 <div class="info">
 <div>
 姓名: {{ teacher.name }}
 性别: {{ teacher.sex | yesno:'男,女' }}
 出生日期: {{ teacher.birth }}
 </div>
 <div class="intro">{{ teacher.intro }}</div>
 <div class="comment">
 好评&nbsp&nbsp
 ({{ teacher.good_count }}
 &nbsp&nbsp&nbsp&nbsp
 差评
 &nbsp&nbsp
 ({{ teacher.bad_count }}
 </div>
 </div>
 {% endfor %}
 返回首页
 </div>
 <script src="https://cdn.bootcss.com/jquery/3.4.1/jquery.min.js"></script>
 <script>
 $(() => {
 $('.comment>a').on('click', (evt) => {
 evt.preventDefault()
 let url = $(evt.target).attr('href')
 $.getJSON(url, (json) => {
 if (json.code == 20000) {
 $(evt.target).next().text(json.count)
 } else {
 alert(json.msg)
 }
 })
 })
 })
 </script>
 </body>
</html>
```

上面的前端代码中，使用了jQuery库封装的`getJSON`方法向服务器发送异步请求，如果不熟悉前端的jQuery库，可以参考[《jQuery API手册》](#)。

## 小结

到此为止，这个投票项目的核心功能已然完成，在下面的章节中我们会要求用户必须登录才能投票，没有账号的用户可以通过注册功能注册一个账号。

# Cookie和Session

我们继续来完成上一章节中的项目，实现“用户登录”的功能，并限制只有登录的用户才能投票。

## 用户登录的准备工作

我们先为实现用户登录做一些准备工作。

1. 创建用户模型。之前我们讲解过如果通过Django的ORM实现从二维表到模型的转换（反向工程），这次我们尝试把模型变成二维表（正向工程）。

```
class User(models.Model):
 """用户"""
 no = models.AutoField(primary_key=True, verbose_name='编号')
 username = models.CharField(max_length=20, unique=True, verbose_name='用户名')
 password = models.CharField(max_length=32, verbose_name='密码')
 tel = models.CharField(max_length=20, verbose_name='手机号')
 reg_date = models.DateTimeField(auto_now_add=True, verbose_name='注册时间')
 last_visit = models.DateTimeField(null=True, verbose_name='最后登录时间')

 class Meta:
 db_table = 'tb_user'
 verbose_name = '用户'
 verbose_name_plural = '用户'
```

2. 使用下面的命令生成迁移文件并执行迁移，将User模型直接变成关系型数据库中的二维表tb\_user。

```
python manage.py makemigrations polls
python manage.py migrate polls
```

3. 用下面的SQL语句直接插入两条测试数据，通常不能将用户的密码直接保存在数据库中，因此我们将用户密码处理成对应的MD5摘要。MD5消息摘要算法是一种被广泛使用的密码哈希函数（散列函数），可以产生出一个128位（比特）的哈希值（散列值），用于确保信息传输完整一致。在使用哈希值时，通常会将哈希值表示为16进制字符串，因此128位的MD5摘要通常表示为32个十六进制符号。

```
insert into `tb_user`
(`username`, `password`, `tel`, `reg_date`)
values
('wangdachui', '1c63129ae9db9c60c3e8aa94d3e00495', '13122334455',
now()),
('hellokitty', 'c6f8cf68e5f68b0aa4680e089ee4742c', '13890006789',
now());
```

**说明：**上面创建的两个用户wangdachui和hellokitty密码分别是1qaz2wsx和Abc123!!。

4. 我们在应用下增加一个名为utils.py的模块用来保存需要使用的工具函数。Python标准库中的hashlib模块封装了常用的哈希算法，包括：MD5、SHA1、SHA256等。下面是使用hashlib中的md5类将字符串处理成MD5摘要的函数如下所示。

```
import hashlib

def gen_md5_digest(content):
 return hashlib.md5(content.encode()).hexdigest()
```

5. 编写用户登录的视图函数和模板页。

添加渲染登录页面的视图函数：

```
def login(request: HttpRequest) -> HttpResponse:
 hint = ''
 return render(request, 'login.html', {'hint': hint})
```

增加login.html模板页：

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>用户登录</title>
 <style>
 #container {
 width: 520px;
 margin: 10px auto;
 }
 .input {
 margin: 20px 0;
 width: 460px;
 height: 40px;
 }
 .input>label {
 display: inline-block;
 width: 140px;
 text-align: right;
 }
 .input>img {
 width: 150px;
 vertical-align: middle;
 }
 input[name=captcha] {
 vertical-align: middle;
 }
 form+div {
 margin-top: 10px;
 }
 </style>
</head>
<body>
 <div id="container">
 <form>
 <div>
 <label>用户名</label>
 <input type="text" name="username" />
 </div>
 <div>
 <label>密码</label>
 <input type="password" name="password" />
 </div>
 <div>

 <input type="text" name="captcha" />
 </div>
 <div>
 <input type="checkbox" name="remember" /> 记住我
 </div>
 <div>
 <input type="submit" value="登录" />
 </div>
 </form>
 <div>
 <small>忘记密码?</small>
 </div>
 </div>
</body>

```

```
 margin-top: 20px;
 }
 form+div>a {
 text-decoration: none;
 color: darkcyan;
 font-size: 1.2em;
 }
 .button {
 width: 500px;
 text-align: center;
 margin-top: 20px;
 }
 .hint {
 color: red;
 font-size: 12px;
 }

```

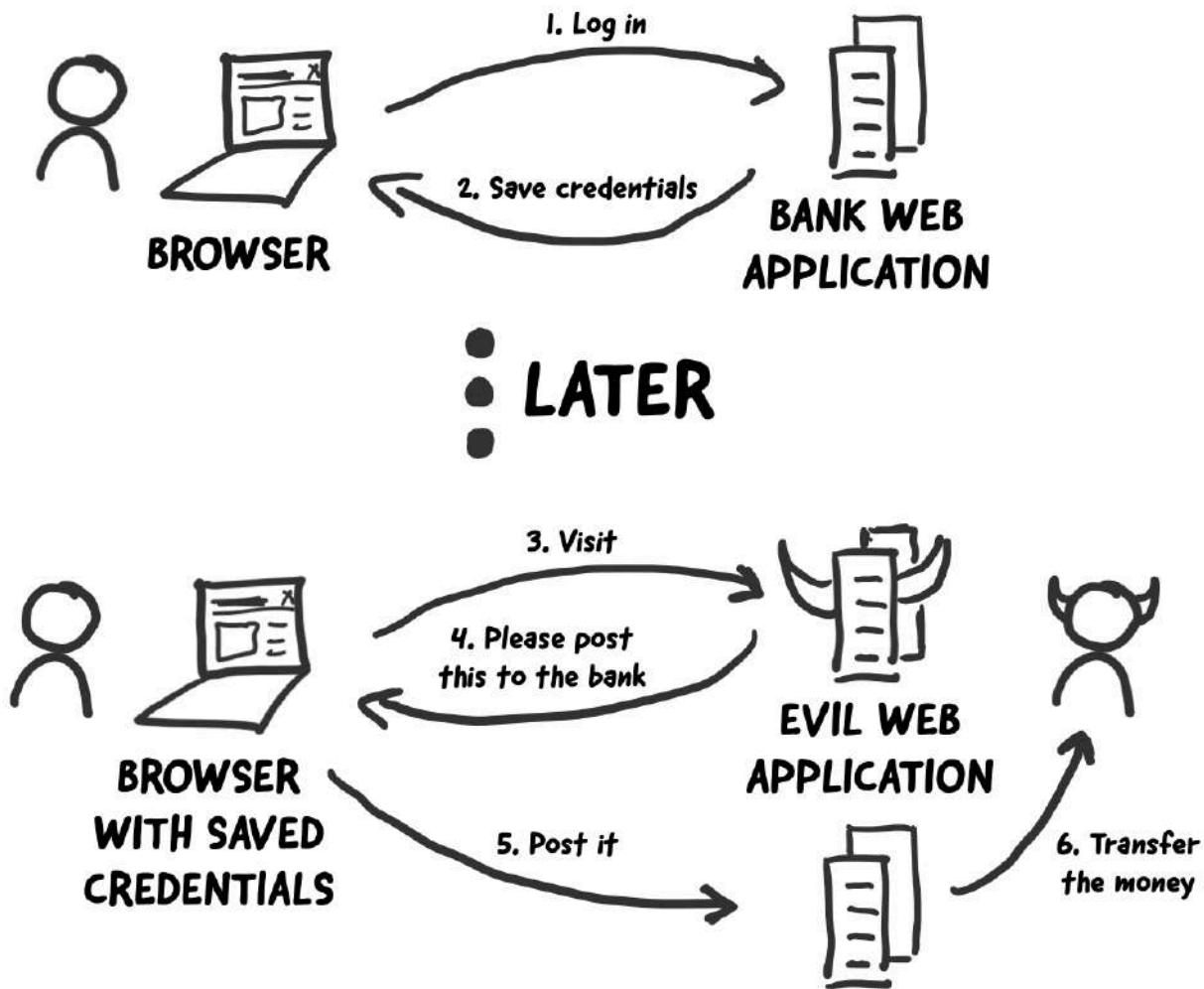
</style>

```
</head>
<body>
 <div id="container">
 <h1>用户登录</h1>
 <hr>
 <p class="hint">{{ hint }}</p>
 <form action="/login/" method="post">
 {% csrf_token %}
 <fieldset>
 <legend>用户信息</legend>
 <div class="input">
 <label>用户名: </label>
 <input type="text" name="username">
 </div>
 <div class="input">
 <label>密码: </label>
 <input type="password" name="password">
 </div>
 <div class="input">
 <label>验证码: </label>
 <input type="text" name="captcha">

 </div>
 </fieldset>
 <div class="button">
 <input type="submit" value="登录">
 <input type="reset" value="重置">
 </div>
 </form>
 <div>
 返回首页
 注册新用户
 </div>
 </div>
```

```
</body>
</html>
```

注意，在上面的表单中，我们使用了模板指令`{% csrf_token %}`为表单添加一个隐藏域（大家可以在浏览器中显示网页源代码就可以看到这个指令生成的`type`属性为`hidden`的`input`标签），它的作用是在表单中生成一个随机令牌（token）来防范跨站请求伪造（简称为CSRF），这也是Django在提交表单时的硬性要求。如果我们的表单中没有这样的令牌，那么提交表单时，Django框架会产生一个响应状态码为403的响应（禁止访问），除非我们设置了免除CSRF令牌。下图是一个关于CSRF简单生动的例子。



接下来，我们可以编写提供验证码和实现用户登录的视图函数，在此之前，我们先说说一个Web应用实现用户跟踪的方式以及Django框架对实现用户跟踪所提供的支持。对一个Web应用来说，用户登录成功后必然要让服务器能够记住该用户已经登录，这样服务器才能为这个用户提供更好的服务，而且上面说到的CSRF也是通过钓鱼网站来套取用户登录信息进行恶意操作的攻击手段，这些都是以用户跟踪技术为基础的。在理解了这些背景知识后，我们就清楚用户登录时到底需要执行哪些操作。

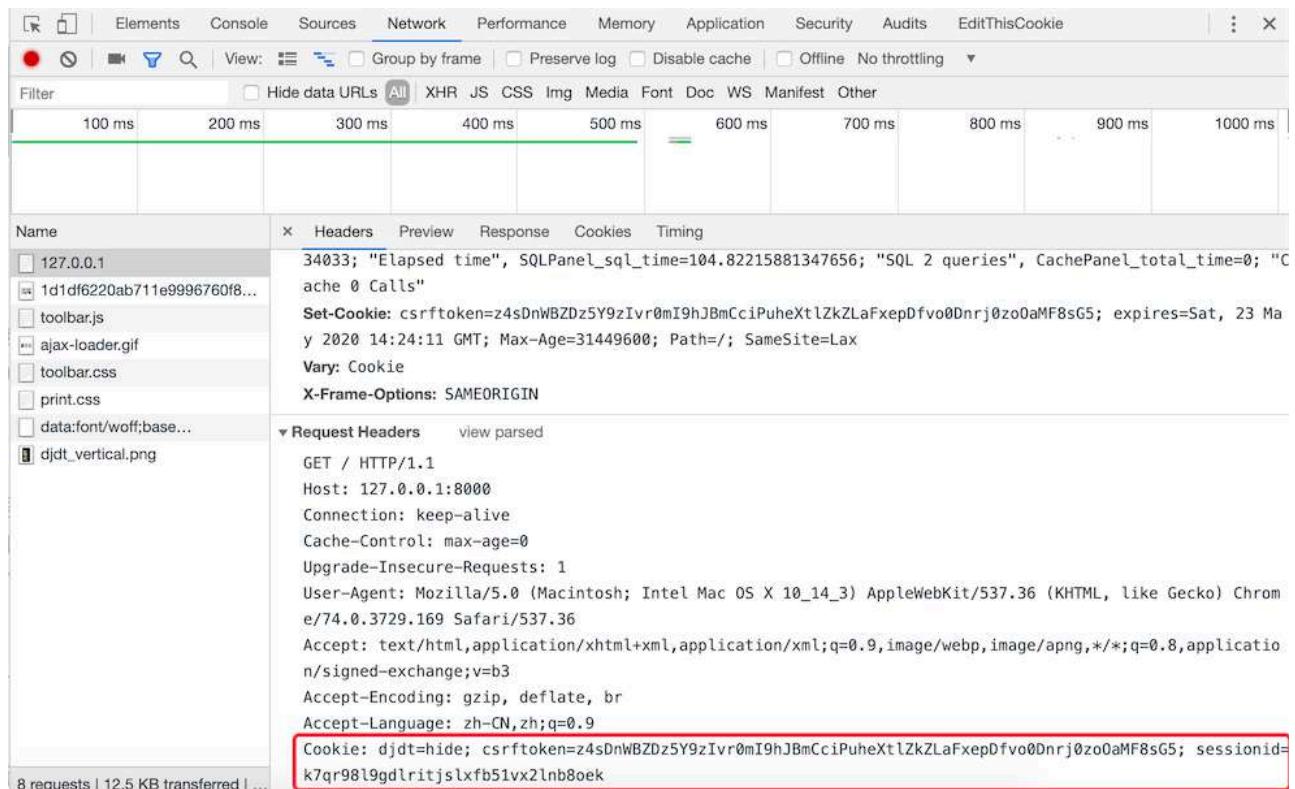
## 实现用户跟踪

如今，一个网站如果不通过某种方式记住你是谁以及你之前在网站的活动情况，失去的就是网站的可用性和便利性，继而很有可能导致网站用户的流失，所以记住一个用户（更专业的说法叫**用户跟踪**）对绝大多数Web应用来说都是必需的功能。

在服务器端，我们想记住一个用户最简单的办法就是创建一个对象，通过这个对象就可以把用户相关的信息都保存起来，这个对象就是我们常说的session（用户会话对象）。那么问题来了，HTTP本身是一个**无连接**（每次请求和响应的过程中，服务器一旦完成对客户端请求的响应之后就断开连接）、**无状态**（客户端再次发起对服务器的请求时，服务器无法得知这个客户端之前的任何信息）的协议，即便服务器通过session对象保留了用户数据，还得通过某种方式来确定当前的请求与之前保存过的哪一个session是有关联的。相信很多人都能想到，我们可以给每个session对象分配一个全局唯一的标识符来识别session对象，我们姑且称之为sessionid，每次客户端发起请求时，只要携带上这个sessionid，就有办法找到与之对应的session对象，从而实现在两次请求之间记住该用户的信息，也就是我们之前说的用户跟踪。

要让客户端记住并在每次请求时带上sessionid又有以下几种做法：

1. **URL重写**。所谓URL重写就是在URL中携带sessionid，例如：`http://www.example.com/index.html?sessionid=123456`，服务器通过获取sessionid参数的值来取到与之对应的session对象。
2. **隐藏域（隐式表单域）**。在提交表单的时候，可以通过在表单中设置隐藏域向服务器发送额外的数据。例如：`<input type="hidden" name="sessionid" value="123456">`。
3. **本地存储**。现在的浏览器都支持多种本地存储方案，包括：cookie、localStorage、sessionStorage、IndexedDB等。在这些方案中，cookie是历史最为悠久也是被诟病得最多的一种方案，也是我们接下来首先为大家讲解的一种方案。简单的说，cookie是一种以键值对方式保存在浏览器临时文件中的数据，每次请求时，请求头中会携带本站点的cookie到服务器，那么只要将sessionid写入cookie，下次请求时服务器只要读取请求头中的cookie就能够获得这个sessionid，如下图所示。



在HTML5时代要，除了cookie，还可以使用新的本地存储API来保存数据，就是刚才提到的localStorage、sessionStorage、IndexedDB等技术，如下图所示。

Key	Value
areald	"22"
hf_time	1558719091729
shshshfpb	v2wbVUrS20FY25A0sngC9MQ==

**总结一下**，要实现用户跟踪，服务器端可以为每个用户会话创建一个session对象并将session对象的ID写入到浏览器的cookie中；用户下次请求服务器时，浏览器会在HTTP请求头中携带该网站保存的cookie信息，这样服务器就可以从cookie中找到session对象的ID并根据此ID获取到之前创建的session对象；由于session对象可以用键值对的方式保存用户数据，这样之前保存在session对象中的信息可以悉数取出，服务器也可以根据这些信息判定用户身份和了解用户偏好，为用户提供更好的个性化服务。

## Django框架对session的支持

在创建Django项目时，默认的配置文件 `settings.py` 文件中已经激活了一个名为 `SessionMiddleware` 的中间件（关于中间件的知识我们在后面的章节做详细讲解，这里只需要知道它的存在即可），因为这个中间件的存在，我们可以直接通过请求对象的 `session` 属性来操作会话对象。前面我们说过，`session` 属性是一个像字典一样可以读写数据的容器对象，因此我们可以使用“键值对”的方式来保留用户数据。与此同时，`SessionMiddleware` 中间件还封装了对 cookie 的操作，在 cookie 中保存了 `sessionid`，这一点我们在上面已经提到过了。

在默认情况下，Django 将 session 的数据序列化后保存在关系型数据库中，在 Django 1.6 以后的版本中，默认的序列化数据的方式是 JSON 序列化，而在此之前一直使用 Pickle 序列化。JSON 序列化和 Pickle 序列化的差别在于前者将对象序列化为字符串（字符串形式），而后者将对象序列化为字节串（二进制形式），因为安全方面的原因，JSON 序列化成为了目前 Django 框架默认序列化数据的方式，这就要求在我们保存在 session 中的数据必须是能够 JSON 序列化的，否则就会引发异常。还有一点需要说明的是，使用关系型数据库保存 session 中的数据在大多数时候并不是最好的选择，因为数据库可能会承受巨大的压力而成为系统性能的瓶颈，在后面的章节中我们会告诉大家如何将 session 保存到缓存服务中以提升系统的性能。

## 实现用户登录验证

首先，我们在刚才的 `polls/utils.py` 文件中编写生成随机验证码的函数 `gen_random_code`，内容如下所示。

```
import random
```

```
ALL_CHARS = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
def gen_random_code(length=4):
 return ''.join(random.choices(ALL_CHARS, k=length))
```

编写生成验证码图片的类Captcha。

```
"""
图片验证码
"""

import os
import random
from io import BytesIO

from PIL import Image
from PIL import ImageFilter
from PIL.ImageDraw import Draw
from PIL.ImageFont import truetype

class Bezier:
 """贝塞尔曲线"""

 def __init__(self):
 self.tsequence = tuple([t / 20.0 for t in range(21)])
 self.beziers = {}

 def make_bezier(self, n):
 """绘制贝塞尔曲线"""
 try:
 return self.beziers[n]
 except KeyError:
 combinations = pascal_row(n - 1)
 result = []
 for t in self.tsequence:
 tpowers = (t ** i for i in range(n))
 upowers = ((1 - t) ** i for i in range(n - 1, -1, -1))
 coefs = [c * a * b for c, a, b in zip(combinations,
 tpowers, upowers)]
 result.append(coefs)
 self.beziers[n] = result
 return result

class Captcha:
 """验证码"""

 def __init__(self, width, height, fonts=None, color=None):
 self._image = None
 self._fonts = fonts if fonts else \
 [os.path.join(os.path.dirname(__file__), 'fonts', font)
```

```
 for font in ['Arial.ttf', 'Georgia.ttf', 'Action.ttf']]
 self._color = color if color else random_color(0, 200, random.randint(220,
255))
 self._width, self._height = width, height

 @classmethod
 def instance(cls, width=200, height=75):
 """用于获取Captcha对象的类方法"""
 prop_name = f'_instance_{width}_{height}'
 if not hasattr(cls, prop_name):
 setattr(cls, prop_name, cls(width, height))
 return getattr(cls, prop_name)

 def _background(self):
 """绘制背景"""
 Draw(self._image).rectangle([(0, 0), self._image.size],
 fill=random_color(230, 255))

 def _smooth(self):
 """平滑图像"""
 return self._image.filter(ImageFilter.SMOOTH)

 def _curve(self, width=4, number=6, color=None):
 """绘制曲线"""
 dx, height = self._image.size
 dx /= number
 path = [(dx * i, random.randint(0, height))
 for i in range(1, number)]
 bcoefs = Bezier().make_bezier(number - 1)
 points = []
 for coefs in bcoefs:
 points.append(tuple(sum([coef * p for coef, p in zip(coefs, ps)])
 for ps in zip(*path)))
 Draw(self._image).line(points, fill=color if color else self._color,
width=width)

 def _noise(self, number=50, level=2, color=None):
 """绘制扰码"""
 width, height = self._image.size
 dx, dy = width / 10, height / 10
 width, height = width - dx, height - dy
 draw = Draw(self._image)
 for i in range(number):
 x = int(random.uniform(dx, width))
 y = int(random.uniform(dy, height))
 draw.line(((x, y), (x + level, y)),
 fill=color if color else self._color, width=level)

 def _text(self, captcha_text, fonts, font_sizes=None, drawings=None,
squeeze_factor=0.75, color=None):
 """绘制文本"""
 color = color if color else self._color
 fonts = tuple([truetype(name, size)
 for name in fonts
```

```
 for size in font_sizes or (65, 70, 75)])
draw = Draw(self._image)
char_images = []
for c in captcha_text:
 font = random.choice(fonts)
 c_width, c_height = draw.textsize(c, font=font)
 char_image = Image.new('RGB', (c_width, c_height), (0, 0, 0))
 char_draw = Draw(char_image)
 char_draw.text((0, 0), c, font=font, fill=color)
 char_image = char_image.crop(char_image.getbbox())
 for drawing in drawings:
 d = getattr(self, drawing)
 char_image = d(char_image)
 char_images.append(char_image)
width, height = self._image.size
offset = int((width - sum(int(i.size[0]) * squeeze_factor)
 for i in char_images[:-1]) -
 char_images[-1].size[0]) / 2
for char_image in char_images:
 c_width, c_height = char_image.size
 mask = char_image.convert('L').point(lambda i: i * 1.97)
 self._image.paste(char_image,
 (offset, int((height - c_height) / 2)),
 mask)
 offset += int(c_width * squeeze_factor)

@staticmethod
def _warp(image, dx_factor=0.3, dy_factor=0.3):
 """图像扭曲"""
 width, height = image.size
 dx = width * dx_factor
 dy = height * dy_factor
 x1 = int(random.uniform(-dx, dx))
 y1 = int(random.uniform(-dy, dy))
 x2 = int(random.uniform(-dx, dx))
 y2 = int(random.uniform(-dy, dy))
 warp_image = Image.new(
 'RGB',
 (width + abs(x1) + abs(x2), height + abs(y1) + abs(y2)))
 warp_image.paste(image, (abs(x1), abs(y1)))
 width2, height2 = warp_image.size
 return warp_image.transform(
 (width, height),
 Image.QUAD,
 (x1, y1, -x1, height2 - y2, width2 + x2, height2 + y2, width2 - x2, -
 y1))

@staticmethod
def _offset(image, dx_factor=0.1, dy_factor=0.2):
 """图像偏移"""
 width, height = image.size
 dx = int(random.random() * width * dx_factor)
 dy = int(random.random() * height * dy_factor)
 offset_image = Image.new('RGB', (width + dx, height + dy))
```

```
offset_image.paste(image, (dx, dy))
return offset_image

@staticmethod
def _rotate(image, angle=25):
 """图像旋转"""
 return image.rotate(random.uniform(-angle, angle),
 Image.BILINEAR, expand=1)

def generate(self, captcha_text='', fmt='PNG'):
 """生成验证码(文字和图片)
 :param captcha_text: 验证码文字
 :param fmt: 生成的验证码图片格式
 :return: 验证码图片的二进制数据
 """
 self._image = Image.new('RGB', (self._width, self._height), (255, 255,
255))
 self._background()
 self._text(captcha_text, self._fonts,
 drawings=['_warp', '_rotate', '_offset'])
 self._curve()
 self._noise()
 self._smooth()
 image_bytes = BytesIO()
 self._image.save(image_bytes, format=fmt)
 return image_bytes.getvalue()

def pascal_row(n=0):
 """生成毕达哥拉斯三角形 (杨辉三角) """
 result = [1]
 x, numerator = 1, n
 for denominator in range(1, n // 2 + 1):
 x *= numerator
 x /= denominator
 result.append(x)
 numerator -= 1
 if n & 1 == 0:
 result.extend(reversed(result[:-1]))
 else:
 result.extend(reversed(result))
 return result

def random_color(start=0, end=255, opacity=255):
 """获得随机颜色"""
 red = random.randint(start, end)
 green = random.randint(start, end)
 blue = random.randint(start, end)
 if opacity is None:
 return red, green, blue
 return red, green, blue, opacity
```

**说明：**上面的代码中用到了三个字体文件，字体文件位于`polls/fonts`目录下，大家可以自行添加字体文件，但是需要注意字体文件的文件名跟上面代码的第45行保持一致。

接下来，我们先完成提供验证码的视图函数。

```
def get_captcha(request: HttpRequest) -> HttpResponse:
 """验证码"""
 captcha_text = gen_random_code()
 request.session['captcha'] = captcha_text
 image_data = Captcha.instance().generate(captcha_text)
 return HttpResponse(image_data, content_type='image/png')
```

注意上面代码中的第4行，我们将随机生成的验证码字符串保存到session中，稍后用户登录时，我们要将保存在session中的验证码字符串和用户输入的验证码字符串进行比对，如果用户输入了正确的验证码才能够执行后续的登录流程，代码如下所示。

```
def login(request: HttpRequest) -> HttpResponse:
 hint = ''
 if request.method == 'POST':
 username = request.POST.get('username')
 password = request.POST.get('password')
 if username and password:
 password = gen_md5_digest(password)
 user = User.objects.filter(username=username,
password=password).first()
 if user:
 request.session['userid'] = user.no
 request.session['username'] = user.username
 return redirect('/')
 else:
 hint = '用户名或密码错误'
 else:
 hint = '请输入有效的用户名和密码'
 return render(request, 'login.html', {'hint': hint})
```

**说明：**上面的代码没有对用户名和密码没有进行验证，实际项目中建议使用正则表达式验证用户输入信息，否则有可能将无效的数据交给数据库进行处理或者造成其他安全方面的隐患。

上面的代码中，我们设定了登录成功后会在session中保存用户的编号（`userid`）和用户名（`username`），页面会重定向到首页。接下来我们可以稍微对首页的代码进行调整，在页面的右上角显示出登录用户的用户名。我们将这段代码单独写成了一个名为`header.html`的HTML文件，首页中可以通过在`<body>`标签中添加`{% include 'header.html' %}`来包含这个页面，代码如下所示。

```
<div class="user">
 {% if request.session.userid %}
 {{ request.session.username }}
 注销
```

```

{%
 else %}
 登录
{%
 endif %}
 注册
</div>

```

如果用户没有登录，页面会显示登录和注册的超链接；而用户登录成功后，页面上会显示用户名和注销的链接，注销链接对应的视图函数如下所示，URL的映射与之前讲过的类似，不再赘述。

```

def logout(request):
 """注销"""
 request.session.flush()
 return redirect('/')

```

上面的代码通过session对象`flush`方法来销毁session，一方面清除了服务器上session对象保存的用户数据，一方面将保存在浏览器cookie中的sessionid删除掉，稍后我们会对如何读写cookie的操作加以说明。

我们可以通过项目使用的数据库中名为`django_session`的表来找到所有的session，该表的结构如下所示：

session_key	session_data	expire_date
c9g2gt5cxo0k2evykgpejhic5ae7bfpl	Mml4YzViYjJhOGMyMDJkY2M5Yzg3...	2019-05-25 23:16:13.898522

其中，第1列就是浏览器cookie中保存的sessionid；第2列是经过BASE64编码后的session中的数据，如果使用Python的`base64`对其进行解码，解码的过程和结果如下所示。

```

import base64

base64.b64decode('Mml4YzViYjJhOGMyMDJkY2M5Yzg3ZWIyZGViZmUzYmYxNzd1NDdmZjp7ImNhCHRj
aGEi0iJzS3d0Iiwibm8i0jEsInVzZXJuYW1lIjoiamFja2ZydWVkIn0=')

```

第3列是session的过期时间，session过期后浏览器保存的cookie中的sessionid就会失效，但是数据库中的这条对应的记录仍然会存在，如果想清除过期的数据，可以使用下面的命令。

```
python manage.py clearsessions
```

Django框架默认的session过期时间为两周（1209600秒），如果想修改这个时间，可以在项目的配置文件中添加如下所示的代码。

```

配置会话的超时时间为1天 (86400秒)
SESSION_COOKIE_AGE = 86400

```

有很多对安全性要求较高的应用都必须在关闭浏览器窗口时让会话过期，不再保留用户的任何信息，如果希望在关闭浏览器窗口时就让会话过期（cookie中的sessionid失效），可以加入如下所示的配置。

```
设置为True在关闭浏览器窗口时session就过期
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```

如果不希望将session的数据保存在数据库中，可以将其放入缓存中，对应的配置如下所示，缓存的配置和使用我们在后面讲解。

```
配置将会话对象放到缓存中存储
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
配置使用哪一组缓存来保存会话
SESSION_CACHE_ALIAS = 'default'
```

如果要修改session数据默认的序列化方式，可以将默认的`JSONSerializer`修改为`PickleSerializer`。

```
SESSION_SERIALIZER = 'django.contrib.sessions.serializers.PickleSerializer'
```

接下来，我们就可以限制只有登录用户才能为老师投票，修改后的`praise_or_criticize`函数如下所示，我们通过从`request.session`中获取`userid`来判定用户是否登录。

```
def praise_or_criticize(request: HttpRequest) -> HttpResponse:
 if request.session.get('userid'):
 try:
 tno = int(request.GET.get('tno'))
 teacher = Teacher.objects.get(no=tno)
 if request.path.startswith('/praise/'):
 teacher.good_count += 1
 count = teacher.good_count
 else:
 teacher.bad_count += 1
 count = teacher.bad_count
 teacher.save()
 data = {'code': 20000, 'mesg': '投票成功', 'count': count}
 except (ValueError, Teacher.DoesNotExist):
 data = {'code': 20001, 'mesg': '投票失败'}
 else:
 data = {'code': 20002, 'mesg': '请先登录'}
 return JsonResponse(data)
```

当然，在修改了视图函数后，`teachers.html`也需要进行调整，用户如果没有登录，就将用户引导至登录页，登录成功再返回到投票页，此处不再赘述。

在视图函数中读写cookie

下面我们将对如何使用cookie做一个更为细致的说明以便帮助大家在Web项目中更好的使用这项技术。Django封装的`HttpRequest`和`HttpResponse`对象分别提供了读写cookie的操作。

`HttpRequest`封装的属性和方法：

1. `COOKIES`属性 - 该属性包含了HTTP请求携带的所有cookie。
2. `get_signed_cookie`方法 - 获取带签名的cookie，如果签名验证失败，会产生`BadSignature`异常。

`HttpResponse`封装的方法：

1. `set_cookie`方法 - 该方法可以设置一组键值对并将其最终将写入浏览器。
2. `set_signed_cookie`方法 - 跟上面的方法作用相似，但是会对cookie进行签名来达到防篡改的作用。因为如果篡改了cookie中的数据，在不知道密钥和盐的情况下是无法生成有效的签名，这样服务器在读取cookie时会发现数据与签名不一致从而产生`BadSignature`异常。需要说明的是，这里所说的密钥就是我们在Django项目配置文件中指定的`SECRET_KEY`，而盐是程序中设定的一个字符串，你愿意设定为什么都可以，只要是一个有效的字符串。

上面提到的方法，如果不清楚它们的具体用法，可以自己查阅一下Django的[官方文档](#)，没有什么资料比官方文档能够更清楚的告诉你这些方法到底如何使用。

刚才我们说过了，激活`SessionMiddleware`之后，每个`HttpRequest`对象都会绑定一个`session`属性，它是一个类似字典的对象，除了保存用户数据之外还提供了检测浏览器是否支持cookie的方法，包括：

1. `set_test_cookie`方法 - 设置用于测试的cookie。
2. `test_cookie_worked`方法 - 检测测试cookie是否工作。
3. `delete_test_cookie`方法 - 删除用于测试的cookie。
4. `set_expiry`方法 - 设置会话的过期时间。
5. `get_expire_age/get_expire_date`方法 - 获取会话的过期时间。
6. `clear_expired`方法 - 清理过期的会话。

下面是在执行登录之前检查浏览器是否支持cookie的代码。通常情况下，浏览器默认开启了对cookie的支持，但是可能因为某种原因，用户禁用了浏览器的cookie功能，遇到这种情况我们可以在视图函数中提供一个检查功能，如果检查到用户浏览器不支持cookie，可以给出相应的提示。

```
def login(request):
 if request.method == 'POST':
 if request.session.test_cookie_worked():
 request.session.delete_test_cookie()
 # Add your code to perform login process here
 else:
 return HttpResponse("Please enable cookies and try again.")
 request.session.set_test_cookie()
 return render_to_response('login.html')
```

## Cookie的替代品

之前我们说过了，cookie的名声一直都不怎么好，当然我们在实际开发中是不会在cookie中保存用户的敏感信息（如用户的密码、信用卡的账号等）的，而且保存在cookie中的数据一般也会做好编码和签名的工作。对于支持HTML5的浏览器来说，可以使用`localStorage`和`sessionStorage`做为cookie的替代方案，相信从名字上你就

能听出二者的差别，存储在`localStorage`的数据可以长期保留；而存储在`sessionStorage`的数据会在浏览器关闭时会被清除。关于这些cookie替代品的用法，建议大家查阅[MDN](#)来进行了解。

# 制作报表

## 导出Excel报表

报表就是用表格、图表等格式来动态显示数据，所以有人用这样的公式来描述报表：

报表 = 多样的格式 + 动态的数据

有很多的三方库支持在Python程序中写Excel文件，包括`xlwt`、`xlwings`、`openpyxl`、`xlswriter`等，其中的`xlwt`虽然只支持写`xls`格式的Excel文件，但在性能方面的表现还是不错的。下面我们就以`xlwt`为例，来演示如何在Django项目中导出Excel报表。

安装`xlwt`。

```
pip install xlwt
```

导出包含所有老师信息的Excel表格的视图函数。

```
def export_teachers_excel(request):
 # 创建工作簿
 wb = xlwt.Workbook()
 # 添加工作表
 sheet = wb.add_sheet('老师信息表')
 # 查询所有老师的信息
 queryset = Teacher.objects.all()
 # 向Excel表单中写入表头
 colnames = ('姓名', '介绍', '好评数', '差评数', '学科')
 for index, name in enumerate(colnames):
 sheet.write(0, index, name)
 # 向单元格中写入老师的数据
 props = ('name', 'detail', 'good_count', 'bad_count', 'subject')
 for row, teacher in enumerate(queryset):
 for col, prop in enumerate(props):
 value = getattr(teacher, prop, '')
 if isinstance(value, Subject):
 value = value.name
 sheet.write(row + 1, col, value)
 # 保存Excel
 buffer = BytesIO()
 wb.save(buffer)
 # 将二进制数据写入响应的消息体中并设置MIME类型
 resp = HttpResponse(buffer.getvalue(), content_type='application/vnd.ms-excel')
 # 中文文件名需要处理成百分号编码
 filename = quote('老师.xls')
 # 通过响应头告知浏览器下载该文件以及对应的文件名
```

```
resp['content-disposition'] = f'attachment; filename*=utf-8\'\'{filename}'
return resp
```

映射URL。

```
urlpatterns = [
 path('excel/', views.export_teachers_excel),
]
```

## 导出PDF报表

在Django项目中，如果需要导出PDF报表，可以借助三方库reportlab来生成PDF文件的内容，再将文件的二进制数据输出给浏览器并指定MIME类型为application/pdf，具体的代码如下所示。

```
def export_pdf(request: HttpRequest) -> HttpResponse:
 buffer = io.BytesIO()
 pdf = canvas.Canvas(buffer)
 pdf.setFont("Helvetica", 80)
 pdf.setFillColorRGB(0.2, 0.5, 0.3)
 pdf.drawString(100, 550, 'hello, world!')
 pdf.showPage()
 pdf.save()
 resp = HttpResponse(buffer.getvalue(), content_type='application/pdf')
 resp['content-disposition'] = 'inline; filename="demo.pdf"'
 return resp
```

关于如何用reportlab定制PDF报表的内容，可以参考reportlab的[官方文档](#)。

## 生成前端统计图表

如果项目中需要生成前端统计图表，可以使用百度的ECharts。具体的做法是后端通过提供数据接口返回统计图表所需的数据，前端使用ECharts来渲染出柱状图、折线图、饼图、散点图等图表。例如我们要生成一个统计所有老师好评数和差评数的报表，可以按照下面的方式来做。

```
def get_teachers_data(request):
 queryset = Teacher.objects.all()
 names = [teacher.name for teacher in queryset]
 good_counts = [teacher.good_count for teacher in queryset]
 bad_counts = [teacher.bad_count for teacher in queryset]
 return JsonResponse({'names': names, 'good': good_counts, 'bad': bad_counts})
```

映射URL。

```
urlpatterns = [
 path('teachers_data/', views.get_teachers_data),
]
```

使用ECharts生成柱状图。

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>老师评价统计</title>
</head>
<body>
 <div id="main" style="width: 600px; height: 400px"></div>
 <p>
 返回首页
 </p>
 <script src="https://cdn.bootcss.com/echarts/4.2.1-rc1/echarts.min.js">
</script>
 <script>
 var myChart = echarts.init(document.querySelector('#main'))
 fetch('/teachers_data/')
 .then(resp => resp.json())
 .then(json => {
 var option = {
 color: ['#f00', '#00f'],
 title: {
 text: '老师评价统计图'
 },
 tooltip: {},
 legend: {
 data: ['好评', '差评']
 },
 xAxis: {
 data: json.names
 },
 yAxis: {},
 series: [
 {
 name: '好评',
 type: 'bar',
 data: json.good
 },
 {
 name: '差评',
 type: 'bar',
 data: json.bad
 }
]
 }
 })
 </script>
</body>
</html>
```

```
 myChart.setOption(option)
 })
</script>
</body>
</html>
```

运行效果如下图所示。

老师评价统计图

 好评  差评

