Python:从入门到精通

1-7 基础知识

1-2 语法基础

语句不需要以分号结尾，但是也不会错,最好不要
每个import只导入一个模块,类似于include
同一行语句间必须以分号间隔
print("")输出数值、表达式结果、字符串,逗号用于间隔(会输出空格)，+连接字符串，自动输出换行符，
    括号内逗号间隔，可定向输出,例如print("",file=filepointer)
    3.x同一行输出,print("",end=' (分隔符)')(即末尾非换行符)
同样支持ASCII，同样非0即1
is，自然语言，判断
#用于单行注释，"""XXX"""或者'''XXX'''用于多行注释
包在三引号之间且不属于任何语句的内容认为是注释
input()接受键盘输入，括号内为提示，输入的内容在3.x中作为字符串
缩进要求比较严格,同一个级别的代码块的所尽量必须相同
同样允许\接续语句，与C一样，字符串间和字符串内接续不同
区分字母大小写
中文可以作为标识符
单下划线开头的标识符表示不能直接访问的类属性，也不能通过from xxx import*导入
以双下划线开头的标识符，表示类的私有成员
以双下划线开头和结尾的是专用的标识
type()给出类型,type(1)输出<class 'int'>
通过赋值就可以改变类型，是弱类型语言
id()获取变量的内存地址
值与字面量相似，一旦出现，便分配地址
输入的数比较大时，会自动在后面加上L(2.X版本)
数值超过计算机自身的计算功能时，会自动转用高精度计算
八进制数以0o/0O开头，2.X版本可以以0开头
十六进制以0x/0X开头
str()将数值转换为字符串
数字类型有整型、浮点数、复数
"""等价于"/',在语句内，"""用于跨行
在字符串定界符引号的前面加上字符r(或R)，字符串将原样输出
假值：False,None,数值中的零,空序列,自定义对象的实例
int(x)、float(x)转换成整数、浮点数类型
complex(real,[,imag])创建复数
str(x)、repr(x)转换为字符串和表达式字符串
eval(str)计算在字符串中的有效python表达式,可以是计算式、序列、字典、集合,并返回一个对象
chr(x)将整数转换为字符、ord(x)将字符转换为整数
hex(x)、oct(x)整数转换成十六进制和八进制

3 运算符

算术运算符中，/表示精确除法，//表示取整除法(返回商的整数部分)，**表示幂
赋值运算符都是算数运算符=
比较运算符，!=表示不等于
逻辑运算符,and or not
条件表达式，if 表达式1 : else 表达式2   if a: else: 也等价于,a if a>b else b

4 流程控制

选择语句：
    if 表达式:，当该分支只有一句语句时，可以直接写在冒号后
    if 表达式: else：
    if 表达式1: elif 表达式2:
循环语句:
    while 条件表达式:
        循环体
    for 迭代变量 in 对象:
        循环体
range(start,end,step)起始(省略从0开始),终值,步长(省略为1),当只有一个参数时=(0,终值,1)，两个=(起始,终值,1)
    注意，循环并不包含终值
    2.x xrange解决range消耗内存问题，3.x直接在range中解决，并删去xrange
break,终止当前循环、continue,进入最内层循环的下一次循环
pass，什么事情都不做，用以占位
数组、字符串可以这样用:array[2],array[2:5](不包含array[5])
索引的范围(-n,n-1),-1从最后一位开始
切片,sname[start:end:step],起始(省略为0),终值(省略为序列长度),步进(省略为1，可以省略最后一个冒号)
序列相加,用+实现，必须类型相同;序列乘法，重复n次,可用于初始化:[None]*n,长度为n的None数组
关键字in用于检查元素是否为序列成员,或者字符串中是否有一段是，但不能用于集合间关系
关键字间可以组合,如 not in
max()计算列表最大元素，min()最小,len()计算长度
sum()计算元素和,sorted()对元素进行排序,str()将序列转换为字符串

5 序列

reversed()反向序列中的元素(显示为一个迭代器对象的内存地址),只能循环遍历一次
enumerate()将序列组合为一个索引序列
列表的元素可以混搭，数值、字符串、列表
list()将序列转换为列表,内容可以是range、字符串、元组或者其他可以迭代的数据
del listname,删除列表,del listname[index]/[index1:index2],删除列表中某个元素
import datetime,datetime.datetime.now(),获取系统时间
索引序列的检索，需要用()来包含
    但是for遍历，用法:for a,b in enumerate(listname)

5.1 列表

listname.append(obj),在列表末尾添加一个元素(比加号快)
listname.insert(index,obj),在列表具体位置插入一个元素，比上者效率低
listname.extend(seq),在列表末尾添加一个列表
listname.remove(obj),删除在列表中第一个等于obj的元素(如果不存在会报错),所以和count一起用
listname.count(obj),统计obj在列表中出现的次数
listname.index(obj),给出obj在列表中第一次出现的下标(不存在会报错)
listname.sort(key=None,reverse=False),key指定一个从每个列表元素中提取一个比较键,例如
    key=str.lower表示排序时不区分大小写
    reverse,True表示降序排列,否则默认升序排列
sorted(literable,key=None,reverse=False),排序后原列表的元素顺序不变,只是建立新的副本
list=[Expression for var in range/list]生成指定范围的数值列表
import random,random.randint(n1,n2),生成n1~n2的随机整数

5.2 元组

```
(element1,element2,...)/element1,element2,.../element,
tuple()转换成元组，可是range、字符串、元组或其他迭代数据
del tuplename,删除元组
元组不可修改，但是可以相加，单个元素元组一定要加,
tuple=(Expression for var in range/list)生成指定范围的元组(但是生成器对象,所以需要转换)
listname.pop(index=-1),删去列表中的某位元素,默认为-1
```

6.1 字典

```
通过键而不是通过索引来获取
字典是任意对象的无序集合
字典是可变的，并且可以任意嵌套
字典中的键必须唯一
字典中的键必须不可变
dictionary={'key1':'value1','key2':'value2',...}
dict()将元组转换为字典,或者通过给定的"键-值对"创建字典
dict(zip(list1,list2)),dict(key1=value1,key2=value2,...)
dict.fromkeys(list),创建值为空的字典
del dict,删除字典;del dict[key],删除某个元素,删除不存在的元素将报错
dictionary.clear()清楚该字典的全部元素
访问,dictionary[key];dictionary.get(key,default),寻找字典中该键，如果没有返回
default(省略返回None)
dictionary.items()获取键-值对元组列表，同样可以用于for遍历
dictionary.keys()获取键列表,dictionary.values()获取值的元组列表
dict[key]=value,用于添加新的键-值对或者修改
同样可以用推导式创建字典dict=[i:j for i,j in zip(list1,list2)]
```

6.2 集合

```
{},每个元素唯一而不重合
setname={element1,element2,...}
集合是无序的,每次输出顺序可能不同
set()将列表、元组等其他可迭代对象转换为集合
setname.add(element),向集合中添加元素
del setname,删除整个集合,setname.clear()清空集合
setname.pop()移除一个元素,setname.remove(obj),移除元素obj,都需要判断是否非空和存在该元素
集合的逻辑运算,a&b,a|b,a-b
zip()，将多个列表或换组对应位置的元素组合为元组
```

7 字符串

```
ASCII,美国标准信息交换码
GBK、GB2312是我国制定的中文编码标准
UTF-8是国积通用的编码
str表示Unicode字符(ASCII和其他)、bytes表示二进制数据(包括编码的文本)(带有b前缀的字符串)
str.encode(encoding="utf-8",errors="strict"),"gb2312","GBK",如果只有编码参数可以省略
encoding
    将字符串转化为二进制数据,也称为编码;不改变原字符串
    strict(遇到非法字符抛出异常),ignore(忽略非法字符),replace(用?替换非法字
符),xmlcharrefreplace(使用XML的字符引用)
        默认值为strict
str.split(sep,maxsplit),sep:用于指定分隔符,可以包含多个字符,默认为None,即所有空字符
```

```
    没有参数默认采用空白符进行分割,无论几个空格或者空白符都将作为一个分隔符进行分割
    maxsplit:可选参数,用于指定分割的次数,默认为-1,分割次数没有限制
    如果不指定sep,那么也不能指定maxsplit
strnew=string.join(iterable),合并字符串
str.count(sub,start,end),在[start,end)中统计次数
str.index(),count和index前面有提
str.find(sub,start,end),返回首次出现该符号或字符串的位置,不存在则返回-1
str.rfind(sub,start,end),从右边开始查找,返回首次出现该符号或字符串的位置,不存在则返回-1,但是顺序还是从左到右:'23' in '123'
str.startswith(prefix,start,end)检索字符串是否以指定字符串开头
str.endswith(prefix,start,end)检索字符串是否以指定字符串结尾
str.lower()大写转换为小写,如果没有可以转换的,就返回原字符串，否则返回新的字符串
str.upper()小写转换为大写,如果没有可以转换的,就返回原字符串，否则返回新的字符串
str.strip([chars])去掉字符串左右两侧的chars内的字符,不指定chars,默认空格和特殊字符
str.lstrip([chars])去掉字符串左侧的chars内的字符,不指定chars,默认空格和特殊字符
str.rstrip([chars])去掉字符串右侧的chars内的字符,不指定chars,默认空格和特殊字符
```

7.* 格式化字符串

```
'%[-][+][0][m][.n]格式化字符'%exp
    -左对齐,正数前方无符号,负数前面加负号
    +右对齐,正数前方加正号,负数前面加负号
    0右对齐,正数前方无符号,负数前方加负号,用0填充空白处
    m占有宽度
    .n小数点后保留的位数
    格式化字符:
        s字符串(str()表示) c单个字符    d/i十进制整数    x十六进制整数    f/F浮点数
        r字符串(repr()显示) o八进制 e指数(基底为e) E指数(基底为E) %字符%
str.format(args)
{[index][:[[fill]align][sign][#][width][.precision][type]]}
    index,索引位置
    fill,空白处填充的字符
    align,指定对齐方式(<左对齐,>右对齐,=内容右对齐、符号最左侧、只对数字有效,^居中)
    sign,有无符号数(+正数加正号,负数加负号;-正数不变,负数加负号;空格,正数加空格,负数加负号)
    #二进制、八进制、十六进制,显示0b/0o/0X开头
    width指定所占宽度
    .precision保留的小数位数
    type:
        S对字符串类型格式化 D十进制整数 C十进制整数转换为Unicode     e/E转换为科学计数法
        g/G在e和f或E和F中切换    b十进制转换为二进制 o十进制转换为八进制 x或X十进制转换为八进制
        f/F转换为浮点数(默认小数点后6位)     %显示百分比
转义字符
    \n  换行符，将光标位置移到下一行开头。
    \r  回车符，将光标位置移到本行开头。
    \t  水平制表符，也即 Tab 键，一般相当于四个空格。
    \a  蜂鸣器响铃。注意不是喇叭发声，现在的计算机很多都不带蜂鸣器了，所以响铃不一定有效。
    \b  退格（Backspace），将光标位置移到前一列。
    \\  反斜线  \'  单引号  \"  双引号
    \   在字符串行尾的续行符，即一行未完，转到下一行继续写。
```

8-14 进阶提高

8 正则表达式

常用元字符
    行定位符^str,匹配字符串str的开始位置是行头,str$是行尾，str是行头行尾
    \bstr\w*\b,从某个单词开始处,匹配字符串str,接着是任意数量的字母或数字\w*,最后是单词结尾处
    .匹配除换行符意外的任意字符 \w匹配字母或数字或下划线或汉字 \s匹配任意的空白符
    \d匹配数字 \b匹配单词的开始或结束 ^匹配字符串的开始 $匹配字符串的结束
    ()也算是以一个元字符
常用限定符
    ?匹配前面的字符零次或一次 +匹配前面的字符一次或多次 *匹配前面的字符零次或多次
    {n}匹配前面的字符n次 {n,}匹配前面的字符最少n次 {n,m}匹配前面的字符n-m次
集合关系
    包含[char]:[0-9]=\d,可以和元字符和限定符组合
    排除[^char] |选择
    分组(str1|str2)str3=str1str3|str2str3
转义字符\:\.='.'
Python模式字符串:
    '' "" '''''' 框起来
    如果有其他的特殊意义的转义符号,就需要将\转义,或者用r''进行原生字符串表示
匹配字符串
    re.match(pattern,string,[flags]),从起始位置开始匹配第一个
        pattern表示模式字符串,string表示要匹配的字符串,flags表示标志位,用于控制匹配方式
            A或ASCII对于\w \W \b \B \d \D \s \S只进行ASCII匹配(3.x)
            I或IGNORECASE执行不区分字母大小写的匹配
            M或MULTILINE将^$用于包括整个字符串的开始和结尾的每一行(默认情况下,仅适用于整个字符串的开始和结尾处)
            S或DOTALL使用.字符匹配所有字符,包括换行符
            X或VERBOSE忽略模式字符串中未转义的空格和注释
            用and连接多个标志位
        返回Match对象,可用Match.start()得到起始位置,Match.end()结束,Match.span()元组,Match.string要匹配的字符串
        Match.group()匹配数据
    re.search()与re.match()相似,但是不只是从字符串起始处开始
    re.findall()搜寻所有,以列表形式返回
替换字符串
    re.sub(pattern,repl,string,count,flags)
        pattern模式字符串,repl替换的字符串,string要被查找替换的原始字符串
        count模式匹配后替换的最大次数，默认0替换所有的匹配,flags标志位,同re.match
分割字符串
    re.split(pattern,string,[maxsplit],[flags])
        pattern表示模式字符串,string表示要匹配的字符串,maxsplit最大拆分次数,flags表示标志位,同re.match

9 函数

创建函数
    def functionname([parameterlist]):(函数名称,定向函数中传递的参数,以逗号分隔)
        ['''comments''']为函数指定注释
        [functionbody]函数体,如果有返回值用return(这两行注意缩进)
调用函数
    functionname([parametersvalue])
        functionname函数名称,必须已经创建好
        parametersvalue指定各个参数的值
参数传递
    位置参数,数量、位置必须与定义时一致
    关键字参数，使用形式参数的名字来确定输入的参数值

    还可以为参数设置默认值,functionname.__defaults__查看函数的默认值参数的当前值(结果为元组)
可变参数
    parameter,接收任意多个实际参数并放入元组,将一个已经存在的列表作为参数,在其名字前加
    **parameter,接收任意多个类似关键字参数一样显式赋值的实际参数，并放入字典,通过parameter.items()访问
        将一个已经存在的列表作为参数,在其名字前加**
返回值
    return [value],value可以是多个
        没有return时,默认返回None
作用域
    局部变量,只在韩素内部有效，运行结束后不存在
    全局变量
        函数外定义全局变量
        函数内使用global修饰，便可以修改函数外同名变量
匿名函数
    result = lambda [arg1 [,arg2,...,argn]]:expression
        result用于调用lambda表达式
        [arg1 [,arg2,...,argn]]:指定要传递的参数列表
        expression指定一个实现具体功能的表达式
        只能有一个表达式,只能有一个返回值,不能是for while

10 面向对象程序设计 & 类

面向对象程序设计的特点：封装、继承、多态
封装对象的属性和行为的载体，具有相同属性和行为的一类实体

10.1 定义类

class ClassName:指定类名，一般大写字母开头，一般以"驼峰式命名法"命名
    '''类的帮助信息'''   指定类的文档字符串
    statement   类体，主要由类变量(或类成员)、方法或属性等定义语句组成
类的名字可以被用来赋值
创建类的实例:instanceName=ClassName(parameterlist)
创建__init__()方法
    每次创建一个类的新实例时，都会执行，必须包含一个self参数，并且必须是第一个参数

10.2 类的成员

创建实例方法并访问
    def functionName(self,parameterlist):(self也可以是其他单词，parameterlist除self外的参数)
        block   方法体，具体的实现功能
    instanceName.functionName(parametersvalue)(instanceName为类的实例名称,functionName为要调用的方法名称)
创建数据成员并访问
    类属性,定义在类中，并且在函数体外的属性。类属性可以在类的所有实例之间共享值(公用)
        可以动态地为类和对象修改属性,ClassName.newname=...
        通过类名和实例名都可以访问
    实例属性,定义在类的方法中的属性,只作用于当前实例中,只能通过实例名访问
        实例属性通过实例名称修改,只影响该实例
访问限制
    _foo以单下划线开头的表示protected(保护)类型的成员,只允许类本身和子类进行访问,但不能使用"from module import*"语句导入

```
    __foo双下划线表示private(私有)类型的成员,只允许定义该方法的类本身进行访问,而且也不能通过类的实例进行访问
        但是可以通过"类的实例名.类名__xxx"方式访问
    __foo__首尾双下划线表示定义特殊方法,一般是系统定义名字,如__init__()
```

10.3 属性

```
创建用于计算的属性
    @property
    def methodname(self):(methodname指定方法名,一般使用小写字母开头，最后将作为创建的属性名)
        block(self表示类的实例,block方法体:实现的具体功能)
为属性添加安全保护机制
    方法转换为属性后,可以直接通过访问方法名来访问方法,不需要加(),也意味着不能加参数
        成为属性后,不能在类体外修改,但是可以通过修改类内类属性或实例进行相应变化
    @methodname.setter
    def methona(self,parametersvalue):
        约束条件:
            block    通过添加私有属性，然后经过类内私有属性修改属性
```

10.4 继承

```
class ClassName(baseclasslist):(baseclasslist指定要继承的基类,如果不指定将使用所有python对象的根类object)
    '''类的帮助信息'''
    statement
方法重写,基类的成员都会被派生类继承
派生类中调用基类的__init__()
    super().__init__()
```

11 模块

11.1 自定义模块

```
创建模块 创建文件   名称"模块名.py"   模块名区分大小写
    模块名不能是Python自带的标准模块名称,模块文件的扩展.py
使用import语句导入模块   import modulename [as alias]
    modulename要导入的模块名称   [as alias]为给模块起的别名,通过该别名也可以使用模块
    一个模块别称用一次as
    调用模块中的变量、函数或者类时,需要在变量名、函数名或者类名前添加"模块名."作为前缀
使用from ... import ...语句导入模块   from modulename import member
    member指定导入的变量、函数或者类等,可以同时导入多个,也可以用通配符*导入全部定义
    调用这种方式导入的模块的变量、函数或者类时,不需要加"模块名."前缀
模块搜索目录
    查找顺序
        (1)在当前目录(执行Python脚本文件所在目录)
        (2)PYTHONPATH(环境变量)下的每个目录中查找
        (3)Python的默认安装目录
    临时添加   执行当前文件的窗口中有效,窗口关闭后失效
        import sys
        sys.path.append("PATH")
    增加.pth文件   (只在当前版本有效)
        在Python安装目录下的Lib\site=packages子目录中,创建一个扩展名为.pth的文件
        在该文件中添加要导入模块所在的目录,重新打开要执行的导入模块的Python文件
    在PYTHONPATH环境变量中添加
```

```
        添加完成后,重新打开要执行的导入模块的Python文件
```

11.2 Python中的包

```
Python程序的包结构
    实际项目开发时,通常情况下,会创建多个包用于存放不同类的文件
创建包:就是创建文件夹
使用包:
    使用import 完整包名.模块名
    from 完整包名 import 模块名
    from 完整包名.模块名 import 定义名/*
导入模块会执行模块中的代码
    __name__在顶级模块中的值为"__main__",在非顶级模块中为该模块名称
```

11.3 引用其他模块

```
导入标准模块:import modulename
使用标准模块:modulename.member
第三方模块的下载与安装
    pip <command> [modulename]
        command 用于要执行的命令:install、uninstall、list
        modulename  用于指定要进行操作的模块
导入模块顺序:标准模块、第三方模块、自定义模块
```

12 异常处理及程序调试

12.1 异常概述

```
SyntaxError:invalid syntax(无效的语法)
ZeroDivisionError(除数为0引发的错误)
NameError(尝试访问一个没有声明的变量引发的错误)
IndexError(索引超出序列范围引发的错误)
IndentationError(缩进错误)
ValueError(传入的值错误)
KeyError(请求一个不存在的字典关键字引发的错误)
IOError(输入输出错误(如要读取的文件不存在))
ImportError(当import语句无法找到模块或from无法在模块中找到相应的名称时引发的错误)
AttributeError(尝试访问未知的对象属性引发的错误)
TypeError(类型不合适引发的错误)
MemoryError(内存不足)
AssertionError(断言为假)
FileNotFoundError(文件不存在或未找到)
```

12.2 异常处理语句

```
try...except
    try:
        block1
    except [ExceptionName [as alias]]:
        block2
    捕获ExceptionName的异常,默认全部异常,别名将记录异常的具体内容
    可以同时处理多个异常,别名也可以公用
try...except...else
    else:
```

```
        block3
    没有捕获异常时执行block1、block3
try...except...finally
    finally:
        block3
    有无异常都会执行block3
    可以用于执行清理代码
else和finally可以一起使用
使用raise语句抛出异常
    raise [ExceptionName[(reason)]]
        主动抛出异常ExceptionName,附带描述信息reason
    抛出异常对象尽量合理
```

12.3 程序调试

```
使用自带的IDLE进行程序调试
(1)打开IDLE(Python shell),主菜单上Debug-Debugger-Debug Control,显示DEBUG ON
(2)Python shell窗口中选择File-Open,打开要调试的文件,添加需要的断点
    对应行右键,Set Breakpoint/删除Clear Breakpoint
    添加断点的原则,程序执行到这个位置时,要查看某些变量的值
(3)快捷键F5,执行程序,Globals将显示全局变量,默认只显示局部变量
(4)Go跳至断点,Step进入要执行的函数,Over单步执行,Out跳出所在的函数,Quit结束调试
(5)全部断点执行完毕,按钮将不可用
(6)关闭Debug Control窗口,显示DEBUG OFF

使用assert语句调试程序
    assert expression [,reason]
        expression为真,什么都不做,假则抛出AssertionError异常
        reason对判断条件进行描述
```

13 文件及目录操作

13.1 基本文件操作

```
fp=open(filename[,mode[,buffering]])
    filename同目录或对应路径下的文件名称
    mode指定文件的打开方式,默认只读
        r只读,文件指针位于开头   rb二进制格式只读,文件指针开头(非文本文件)
        r+可以读取,或者写入新的内容覆盖原有内容(从文件开头覆盖)
        rb+以二进制格式打开
        w以只写模式打开文件  wb以二进制格式、只写模式打开文件
        w+打开文件后清除原有内容,对这个空文件有读写权限
        wb+以二进制格式打开文件,并采用读写模式
        a以追加模式打开文件,如果文件已经存在,文件指针位于文件末尾,否则创建新指针
        ab以追加模式、二进制格式打开文件
        a+以读写模式打开文件      ab+以二进制格式、读写模式打开文件
    以非可创建模式打开不存在的文件会抛出错误
    以二进制文件打开创建BufferedReader对象,以文本文件打开创建TextIOWrapper对象
    打开文件时指定编码方式,encoding='utf-8'/'gbk
关闭文件
    file.close()
        先刷新缓冲区中还没有写入的信息,然后再关闭文件
打开文件时使用with语句
    with expression as target:
        with-body
        expression用于指定的表达式,target用于指定变量并将expression的结果存入target
```

with-body指定with语句体,expression和target可以多个
内部出错后,将执行清除语句,详见with具体实现
用来打开文件,如果出现任何问题,将及时关闭文件
写入文件内容
file.write(string),file必须以w或a的方式打开,不然将抛出错误
在向文件中写入内容后,如果不想马上关闭文件,也可以调用文件对象提供的flush()方法,把缓冲区的内容写入文件,这样也能保证数据全部写入磁盘
file.writelines()把字符串列表写入文件，但是不添加换行符
两种写入方法都不会在输出末尾添加换行符
读取文件
读取指定字符
file.read([size])(size指定要读取的字符个数,省略则一次性读取所有内容)
中英文数字按照一个字符计算
file.seek(offset[,whence])(offset指定移动的字符个数,whence=0从文件头开始,1从当前位置开始计算,2从文件尾开始,默认为0)
如果打开方式没有b,只允许从文件头开始计算相对位置
offset的值按照中文两个字符、英文数字一个字符计算
读取一行
file.readline()
每次读取一行数据,同样需要以含读模式打开
读取全部行
file.readlines()
将每一行以字符串列表形式返回

13.2 目录操作

os和os.path模块
导入os后也可以使用os.path子模块
os提供通用变量
name操作系统类型:Windows:nt,Linux、UNIX、Mac OS:posix
linesep当前操作系统换行符:Windows:\r\n
sep当前操作系统路径分隔符:Windows:\\,可以用/代题
os提供操作目录函数
listdir(path)返回指定路径下的文件和目录信息(字典序列表)
removedirs(path1/path2...)删除多级目录
chdir(path)把path设置为当前工作目录
os.path提供操作目录函数
splitext(str)分离文件名和扩展名
basename(path)从一个目录中提取文件名
dirname(path)从一个路径中提取文件路径,不包括文件名
isdir(path)用于判断是否为有效路径
路径
相对路径依赖于当前工作目录(当前文件所在目录)
os.path.getcwd()返回当前的工作目录
绝对路径(在使用文件时指定文件的实际路径)
os.path.abspath(path)获取文件或目录的绝对路径
拼接路径(并不会检测该路径是否存在)
os.path.join(path,name)将目录与目录或者文件名拼接起来
如果拼接的路径存在多个绝对路径,以最后一次出现的绝对路径为准,并且从这个路径开始
判断目录是否存在
os.path.exists(path)判断目录或者文件是否存在,如果存在则返回True,否则返回False
创建目录
创建一级目录
os.mkdir(path[,mode=0o777])创建目录
path可以是绝对或者相对路径
mode指定数值模式,默认0777,在非UNIX系统上无效或被忽略
已存在将抛出FileExistsError

创建多级目录
os.makedirs(path1/path2...[,mode=0o777])创建多级目录
path、mode同一级
删除目录
os.rmdir(path)删除目录
如果目录不存在抛出FileNotFoundError
只能删除空目录,删除非空目录:shutil模块的rmtree()
遍历目录
os.walk(top[,topdown][,onerror][,followlinks])编历目录树,该方法返回一个元组,
包括所有路径名、目录列表和文件列表三个元素
top要遍历内容的根目录
topdown遍历的顺序,默认True自上而下,False自下而上
onerror指定错误处理方式,默认忽略,或者指定错误处理函数
followlinks,默认情况下不会向下转换成解析到目录的符号连接,设为True则支持
只在UNIX和Windows有效

13.3 高级文件操作

os模块提供文件相关函数
access(path,accessmode)
获取对文件是否有指定的访问权限(读取/写入/执行权限)
accessmode:R_OK(读取)、W_OK(写入)、X_OK(执行)、F_OK(存在)
如果有则返回1
chmod(path,mode)修改path指定文件的访问权限
startfile(path[,operation])使用关联的应用程序打开path指定的文件
删除文件
os.remove(path)
python没有内置删除文件的函数
相对绝对都行
不存在抛出FileNotFoundError
重命名文件和目录
rename(src,dst)将文件或目录重命名为dst
重命名目录时只能修改最后一级的目录名称,即不能移动到不存在的目录下
不存在抛出FileNotFoundError
获取文件基本信息
stat(path)返回path指定的文件信息
返回属性
st_mode保护模式 st_dev设备名 st_ino索引号 st_uid用户ID
st_nlink硬链接号(被连接数目) st_gid组ID st_size文件大小,单位为字节
st_atime最后一次访问时间 st_mtime最后一次修改时间
st_ctime最后一次状态变化时间(系统不同返回结果也不同):Windows文件创建时间

14 操作数据库

MySQL和SQLite需要进一步学习,以下内容仍然比较基础浅显

14.1 数据库编程接口

连接对象
主要提供数据库游标对象和提交/回滚事物的方法，以及关闭数据库连接
获取连接对象
connect()参数
dsn数据源名称,给出该参数表示数据库依赖
user用户名 password用户密码
host主机名 database数据库名称

不同模块connect使用的参数不完全相同,要以具体的数据库模块为准
连接对象的方法
connect()返回连接对象,表示目前和数据库的会话,该对象支持的方法
close()关闭数据库连接 commit()提交事务 rollback()回滚事务
cursor()获取游标对象,操作数据库,如执行DML操作,调用存储过程等
游标对象
代表数据库中的游标,用于指示抓取数据操作的上下文
主要执行SQL语句、调用存储过程、获取查询结果等方法
通过使用连接对象的cursor()方法,可以获取到游标对象
属性:description数据库列类型和值的描述信息
rowcount回返结果的行数统计信息,如SELECT、UPDATE、CALLPROC等
游标对象方法
callproc(procname[,parameters])调用存储过程,需要数据库支持
close()关闭当前游标
execute(operation[,parameters])执行数据库操作,SQL语句或者数据库命令
executemany(operation,seq_of_params)用于批量操作,如批量更新
fetchone()获取查询结果集中的下一条记录
fetchmany(size)获取指定数量的记录
nextset()跳至下一个可用的结果集
arraysize指定使用fetchmany()的行数,默认为1
setinputsizes(sizes)设置在调用execute*()方法时分配的内存区域大小
setoutputsize(sizes)设置列缓冲区大小,对大数据列如LONGS和BLOBS尤其有用

14.2 使用SQLite

创建数据库文件
python内置SQLite3,可以直接导入sqlite3模块
数据库操作通用流程
创建connection--获取cursor--执行SQL语句,处理数据结果--关闭cursor--关闭connection
ex:cursor.execute('create table user (id int(10) primary key, name varchar(20))')
操作SQLite
新增用户数据信息
insert into 表名(字段名 1,...,字段名 n) values(字段值 1,...,字段值n)
在user表中有两个字段,字段名分别为id和name,而字段值需要根据字段的数据类型来赋值
id是一个长度为10的整型,name是长度为20的字符串型数据
ex:cursor.execute('insert into user (id,name) values("id","name")')
记得commit()
查看用户数据信息
select 字段名 1,... from 表名 where 查询条件
查询数据时使用方式
fetchone()获取查询结果集中的下一条记录,返回元组
fetchmany(size)获取指定数量的记录
fetchall()获取结构集的所有记录,返回元组构成列表
ex:cursor.execute('select * from user where id>?',(1,))引号外元组内容依次代替?
?为占位符,避免SQL注入风险
修改用户数据信息
update 表名 set 字段名=字段值 where 查询条件
ex:cursor.execute('update user set name =? where id =?',('王牌空袭',2))
删除用户信息
delete from 表名 where 查询条件
ex:cursor.execute('delete from user where id =?',(2,))

14.3 使用MySQL

```
主机名localhost或者IP地址127.0.0.1
用户名以及对应密码需要记住(分为个人用户和root)
不可轻易转移MySQL文件位置
使用Navicat for MySQL会更加方便
    连接名、主机名或IP地址、端口3306、用户名密码
    新建数据库,自己确定字符集和排序规则
```

14.4 使用PyMySQL模块

```
连接数据库
    pymysql.connect("主机名","用户名","密码","数据库名称")
    execute("SELECT VERSION()")获取MySQL当前版本
    execute("DROP TABLE IF EXISTS name")如果已经存在表name则将其删除
    ex: sql="""CREATE TABLE books(
            id int(8) NOT NULL AUTO_INCREMENT,
            name varchar(50) NOT NULL,
            category varchar(50) NOT NULL,
            price decimal(10,2) DEFAULT NULL,
            publish_time date DEFAULT NULL,
            PRIMARY KEY (id)
        )ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
        """
操作MySQL数据表
    executemany(operation,seq_of_params)
        operation操作的语句      seq_of_params参数序列
    ex:cursor.executemany("insert into books(name,category,price,publish_time)\
        values(%s,%s,%s,%s)",data)
```

** 模块专项

**.1 sys

```
sys是Python自带模块,该模块提供了一系列有关Python运行环境的变量和参数
常见用法与含义
    sys.argv         该方法用于获取当前正在执行的命令行参数的参数列表
    sys.path         该方法用于获取指定模块路径的字符串集合
    sys.exit()       该方法用于退出程序,当参数非0时,会引发一个SystemExit异常,从而可以在主程
序中捕获该异常
    sys.platform     该方法用于获取当前系统平台
    sys.modules      该方法是用于加载模块的字典,每当程序员导入新的模块,sys.modules将自动记
录该模块.当相同模块第二次导入时Python将从该字典中进行查询,从而加快程序的运行速度
    sys.getdefaultencoding()     该方法用于获取当前系统编码方式
```

**.2 time

```
该模块提供了用于处理时间的各种方法
time.time()自1970年来时间戳,9组数字元组处理时间
time.localtime()处理时间戳
time.sleep(interval)程序挂起interval秒
time.asctime()将localtime结果转换为可读格式
time.strftime()将localtime结果转换为格式化日期
    %y 两位数的年份表示（00-99）
    %Y 四位数的年份表示（000-9999）
```

```
    %m 月份（01-12）
    %d 月内中的一天（0-31）
    %H 24小时制小时数（0-23）
    %I 12小时制小时数（01-12）
    %M 分钟数（00-59）
    %S 秒（00-59）
    %a 本地简化星期名称
    %A 本地完整星期名称
    %b 本地简化的月份名称
    %B 本地完整的月份名称
    %c 本地相应的日期表示和时间表示
    %j 年内的一天（001-366）
    %p 本地A.M.或P.M.的等价符
    %U 一年中的星期数（00-53）星期天为星期的开始
    %w 星期（0-6），星期天为星期的开始
    %W 一年中的星期数（00-53）星期一为星期的开始
    %x 本地相应的日期表示
    %X 本地相应的时间表示
    %Z 当前时区的名称
    %% %号本身
```

**.3 calendar

```
calendar.month(year,month)获取某一月日历
```

**.4 turtle

15-21 高级应用

15 GUI界面编程

15.1 初识GUI

```
GUI是Graphical User Interface(图形用户界面)的缩写
GUI程序三要素:输入、处理和输出
流行GUI工具包
    wxPython、Kivy、Flexx、PyQt、Tkinter、Pywin32、PyGTK、pyui4win
```

15.2 创建应用程序

```
基础对象
    应用程序对象
        管理主事件循环,是wxPython程序的动力
    顶级窗口
        管理最重要的数据,控制并呈现给用户
创建wx.App子类
    创建使用wx.App子类的步骤
        定义这个子类
        在定义的字类中写一个OnInit()初始化方法
        在程序的主要部分创建这个类的一个实例
        调用应用程序实例的MainLoop()方法,这个方法将程序的控制权交给wxPython
直接使用wx.App
    通常,在系统中只有一个窗口的话,可以不创建wx.App子类,直接使用wx.App子类
        这个类提供了一个最基本的OnInit()初始化方法
使用 wx.Frame框架
```

在GUI中,框架也称为窗口,是一个容器.
用户可以将它在屏幕上任意移动,并对它进行缩放
通常包含诸如标题栏、菜单等
在wxPython中,wx.Frame是所有框架的父类
当用户创建wx.Frame的子类时,子类应该调用其父类的构造器
wx.Frame.__init__(),构造器语法格式如下

```
wx.Frame(parent,id=-1,title="",pos=wx.DefaultPosition,size=wx.DefaultSize,style=
wx.DEFAULT_FRAME_STYLE,name="frame")
```

parent框架的父窗口,顶级窗口的值为None
id关于新窗口的wxPython ID号,通常设为-1,让wxPython自动生成新的ID
title窗口的标题
pos一个wx.Point对象,它指定这个新窗口的左上角在屏幕的位置,在图形用户界面程序中,通常(0,0)是显示器的左上角,这个默认的(-1,-1)将让系统决定窗口的位置
size一个wx.Size对象,它指定这个窗口的初始尺寸,这个默认的(-1,-1)让系统决定窗口的初始尺寸
style指定窗口的类型的常量,可以使用或运算来组合它们
name框架的内在的名字,可以使用它来寻找这个窗口

15.3 常用控件

StaticText文本类

```
wx.StaticText(parent,id,label,pos=wx.DefaultPosition,size=wx.DefaultSize,style=0
,name="staticText")
```

parent父窗口部件
id标识符
label显示在静态空间中的文本内容
pos一个wx.Point或Python元组,是窗口部件的位置
size一个wx.size或Python元组,是窗口部件的尺寸
style样式标记 name对象的名字
wx.Panel(self)创建画板

```
wx.Font(pointSize,family,style,weight,underline=False,faceName="",encoding=wx.FO
NTENCODING_DEFAULT)
```

pointSize字体的整数尺寸,单位为磅
family用于快速指定一个字体而无须知道该字体的实际名字
style指明字体是否倾斜
weight指明字体的醒目程度
underline仅在Windows系统下有效,True加下划线
faceName指定字体名
encoding允许在几个编码中选择一个,大多数情况可以使用默认编码
TextCtrl输入文本类

```
wx.TextCtrl(parent,id,value="",pos=wx.DefaultPosition,size=wx.DefaultSize,style=
0,validator=wx.DefaultValidator,name=wx.TextCtrlNameStr)
```

style单行wx.TextCtrl的样式,取值及说明:
wx.TE_CENTER控件中的文本居中
wx.TE_LEFT控件中的文本左对齐,默认行为
wx.TE_NOHIDESEL文本始终高亮显示,只适用于Windows系统
wx.TE_PASSWORD不显示所输入的文本,以星号(*)代替显示
wx.TE_PROCESS_ENTER如果使用该参数,那么当用户在控件内按Enter键时,一个文本输入事件将被出发,否则按键事件内在的由该文本控件或对话框管理
wx.TE_PROCESS_TAB如果指定了这个样式,那么通常的字符事件在Tab键按下时创建(一般意味一个制表符将被插入文本),否则tab由对话框来管理,通常是控件间的切换
wx.TE_READONLY文本控件为只读,用户不能修改其中的文本
wx.TE_RIGHT控件中的文本右对齐

value显示在该控件中的初始文本
validator常用于过滤数据以确保只能输入要接收的数据
Button按钮类
按钮是GUI界面中应用最为广泛的控件,它常用于捕获用户生成的单击事件

```
wx.Button(parent,id,label,pos,size=wxDefaultSize,style=0,validator,name="button"
)
```

参数与wx.TextCtrl的参数基本相同,其中参数label是显示在按钮上的文本

15.4 BoxSizer布局

相比于几何绝对位置,sizer(尺寸器)布局方式更为智能
sizer是用于自动布局一组窗口控件的算法,sizer被附加到一个通常是框架或面板的容器
在父容器中创建的子窗口控件必须被分别地添加到sizer
当sizer被附加到容器时,它随后就可以管理它所包含的子布局
wxPython提供的5个sizer
BoxSizer
在一条水平线或垂直线上的窗口部件的布局,当尺寸改变时,控制窗口部件的行为上很灵活,通常用于嵌套的样式,可用于几乎任何类型的布局
GridSizer
一个十分基础的网格布局,当用户要放置的窗口部件都是同样的尺寸且整齐地放入一个规则的网格中可以使用它
FlexGridSizer
对GridSizer稍微做了些改变,当窗口部件有不同的尺寸时,可以有更好的结果
GridBagSizer
GridSizer系列中最灵活的成员,使得网格中的窗口部件可以更随意地放置
StaticBoxSizer
一个标准的Box Sizer,带有标题和环线
BoxSizer是wxPython所提供的sizer中最简单和最灵活的
一个BoxSizer是一个垂直列或水平行,窗口部件在其中从左至右或从上到下布置在一条线上
使用BoxSizer布局
尺寸器会管理组件的尺寸
只要将部件添加到尺寸器上,再加上一些布局参数,然后让尺寸器自己去管理父组件的尺寸
wx.BoxSizer,带有决定水平或是垂直的参数(wx.HORIZONTAL,wx.VERTICAL),默认为水平
Add()方控件加入sizer,面板的SetSizer()设定它的尺寸器
Box.Add(control,proportion,flag,border)
control要添加的控件
proportion所添加控件在定义的定位方式所代表方向上占据的空间比例.如果有3个按钮的比例值分别是0、1、2,变换的比例则是自己的电阻分压
flag参数与border参数结合使用可以指定边距宽度
wx.LEFT左边距 wx.RIGHT右边距
wx.BOTTOM底边距 wx.TOP上边距
wx.ALL上下左右四个边距
通过|操作符来联合使用这些标志,border也可以使用|
flag还可以与proportion参数结合指定控件本身的对齐(排列)方式
wx.ALIGN_LEFT左边对齐 wx.ALIGN_RIGHT右边对齐
wx.ALIGN_TOP顶部对齐 wx.ALIGN_BOTTOM底边对齐
wx.ALIGN_CENTER_VERTICAL垂直对齐 wx.ALIGN_CENTER_HORIZONTAL水平对齐
wx.ALIGN_CENTER居中对齐
wx.EXPAND所添加控件将占有sizer定位方向上所有可用的空间
boder控制所添加控件的边距,在部件之间添加一些像素的空白
默认左对齐,对立设定容易产生冲突

15.5 事件处理

```
用户执行的动作就叫做事件
绑定事件
    bt_confirm.Bind(wx.EVT_BUTTON,OnclickSubmit)
        wx.EVT_BUTTON事件类型为按钮类型
            wx.EVT_MOTION产生于用户移动鼠标
            wx.ENTER_WINDOW和wx.LEAVE_WINDOW产生于鼠标进入或离开一个窗口控件
            wx.EVT_MOUSEWHEEL被绑定到鼠标滚轮的活动
        OnclickSubmit方法名,事件发生时执行该方法
GetValue()获取属性值,SetValue()设定属性值
wx.MessageBox(message)弹出消息
```

16 Pygame游戏编程

16.1 初始Pygame

```
Pygame常用模块
    Pygame做游戏开发的优势在于不需要过多地考虑底层相关的内容,而可以把工作重心放在游戏逻辑上
    Pygame集成了很多和底层相关的模块,如访问显示设备、管理事件、使用字体等
        pygame.cdrom访问光驱                pygame.cursors加载光标
        pygame.display访问显示设备         pygame.draw绘制形状、线和点
        pygame.event管理事件                pygame.font使用字体
        pygame.image加载和存储图片         pygame.key读取键盘按键
        pygame.joystick使用游戏手柄或类似的东西
        pygame.mixer声音                     pygame.mouse鼠标
        pygame.movie播放视频                pygame.music播放音频
        pygame.overlay访问高级视频叠加    pygame.rect管理矩形区域
        pygame.sndarray操作声音数据         pygame.sprite操作移动图象
        pygame.surface管理图象和屏幕       pygame.surfarray管理点阵图象数据
        pygame.time管理时间和帧信息        pygame.transform缩放和移动图象
```

16.2 Pygame基本使用

```
pygame.init()初始化pygame
pygame.display显示窗体,常用模块
    pygame.display.init()初始化display模块
    pygame.display.quit()结束display模块
    pygame.display.get_init()如果display模块已经被初始化,则返回True
    pygame.display.set_mode()初始化一个准备显示的界面
    pygame.display.set_surface()获取当前的Surface对象
    pygame.display.flip()更新整个待显示的Surface对象到屏幕上
    pygame.display.update()更新部分内容显示到屏幕上,如果没有参数则与flip功能相同
添加轮询事件检测,使得窗口不会一闪而过
    pygame.event.get()能够获取事件队列,使用for...in遍历事件,然后根据type属性判断事件类型
    pygame.QUIT表示关闭pygame窗口事件,pygame.KEYDOWN表示键盘按下事件
    pygame.MOUSEBUTTONDOWN表示鼠标按下事件
    pygame.image.load()加载图片返回值是一个Surface对象
    Surface是用来代表图片的pygame图象,可以对一个Surface对象进行涂画、变形、复制等各种操作
    屏幕也只是个Surface.pygame.display.set_mode就返回一个屏幕Surface对象
    如果需要将Surface对象画到srceen Surface对象,需要使用blit()方法,最后使用display模块的
flip方法更新整个待显示的Surface对象到屏幕上
        Surface对象的常用方法
            pygame.Surface.blit()将一个图象画到另一个图像上
            pygame.Surface.convert()转换图象的像素格式
```

pygame.Surface.convert_alpha()转换图象的像素格式,包含alpha通道的转换
pygame.Surface.fill()使用颜色填充Surface
pygame.Surface.get_rect()获取Surface的矩形区域
返回值是一个Rect对象,该对象有一个move(X,Y)方法可以用于移动矩形
添加时钟来控制程序运行的时间
pygame.time.Clock()创建Clock对象实例
clock.tick()用于设定每秒执行次数
pygame.font.SysFont("Fontname", size)设置默认字体和大小
pygame.Rect()获取矩形区域对象,该对象colliderect()方法可以判断两个矩形区域是否重叠

16.3 开发Flappy Bird游戏

Flappy Bird是越南河内独立游戏开发者阮哈东(Dong Nguyen)开发

17 网络爬虫开发

17.1 初始网络爬虫

概述
网络爬虫(又被称作网络蜘蛛、网络机器人、网页追逐者),按照指定的规则(网络爬虫的算法)自动浏览或抓取网络中的信息,通过Python可以很轻松地编写爬虫程序或者是脚本
搜索引擎就离不开网络爬虫,百度搜索引擎的爬虫名字叫做百度蜘蛛
分类
通用网络爬虫
又称作全网爬虫(Scalable Web Crawler),通用网络爬虫的爬行范围和数量巨大,对爬行速度和存储空间要求较高
主要由初始URL集合、URL队列、页面爬行模块、页面分析模块、页面数据库、连接过滤模块等构成
聚焦网络爬虫Focused Crawler
又叫做主题网络爬虫(Topical Crawler)
按照预先定义好的主题,有选择地进行相关网页爬取的一种爬虫,将爬取的目标网页定位在与主题相关的页面中,极大节省硬件和网络资源
增量式网络爬虫Incremental Web Crawler
增量式对应着增量式更新,指在更新的时候只改变更新的地方,而未改变的地方则不更新,只会在需要的时候爬行新产生或发生更新的页面,对于没有发生变化的页面则不会爬取,有效减少数据下载量,减少时间和空间上的耗费,在爬行算法上需要增加一些难度
深度网络爬虫
在互联网中,Web页面按存在方式可以分为表层网页(Surface Web)和深层网页(Deep Web),表层网页指的是不需要提交表单,使用静态的超链接就可以直接访问的静态页面;深层网页指的是那些大部分内容不能通过静态链接获取的、隐藏在搜索表单后面,需要用户提交一些关键词才能获得的Web页面.深层是表层页面信息数量的几百倍,是主要的爬取对象
深层网络爬虫主要通过6个基本功能的模块(爬行控制器、解析器、表单分析器、表单处理器、响应分析器、LVS控制器)和两个爬虫内部数据结构(URL列表、LVS表)等部分组成,其中LVS(Label Value Set)表示标签/数值集合,用来表示填充表单的数据源
网络爬虫的基本原理
(1)获取初始的URL,该URL地址是用户自己制定的初始爬取的网页
(2)爬取对应URL地址的网页时,获取新的URL地址
(3)将新的URL地址放入URL队列中
(4)从URL队列中读取新的URL,然后依据新的URL爬取网页,同时从新的网页中获取新的URL地址,重复上述的爬取过程
(5)设置停止条件,如果没有将会一直爬取到无法获取新的URL为止

17.2 网络爬虫的常用技术

Python的网络请求
    urllib
        Python自带模块,提供urlopen()方法,通过该方法指定URL发送网络请求来获取数据.子模块
            urllib.request定义打开URL(主要是HTTP)的方法和类,如身份验证、重定向、cookie等
            urllib.error主要包含异常类,基本的异常类是URLError
            urllib.parse两大类功能:URL解析和URL引用
            urllib.robotparser解析robots.txt文件
        ex:data=bytes(par.urlencode({'word':'hello'}),encoding='utf8')
        response=req.urlopen("http://httpbin.org/post",data=data)
        html=response.read()
    urllib3
        功能强大,条理清晰,用于HTTP客户端的Python库,许多的Python的原生系统已经开始使用urllib3,提供了很多Python标准库中里所没有的重要特性
            线程安全、连接池、客户端SSL/TLS验证
            使用多部分编码上传文件、Helpers用于重试请求并处理HTTP重定向
            支持gzip和deflate编码、支持HTTP和SOCKS代理
            100%的测试覆盖率
        ex:http=urllib3.PoolManager()
        response=http.request('POST','https://httpbin.org/post',fields={'word':'hello'})
        response.data访问
    requests
        是Python中实现HTTP请求的一种方式,功能特性:
            Keep-Alive&连接池、国际化域名和URL、带持久cookie的会话、
            浏览器式的SSL认证、自动内容解码、基本/摘要式的身份认证、
            优雅的key/value Cookie、自动解压、Unicode响应体、
            HTTP(S)代理支持、文件分块上传、流下载、连接超时、
            分块请求、支持.netrc
        ex: 以GET请求方式,打印多种请求信息
                response=requests.get('https://www.baidu.com')
                print(response.status_code)
                print(response.url)
                print(response.headers)
                print(response.cookies)
                print(response.text)
                print(response.content)
            以POST请求方式,发送HTTP网络请求
                data={'word':'hello'}
                response=requests.post('http://httpbin.org/post',data=data)
                print(response.content)
            还提供
                PUT:requests.put('http://httpbin.org/put',data={'key':'value'})
                DELETE:requests.delete('http://httpbin.org/delete')
                HEAD:requests.head('http://httpbin.org/get')
                OPTIONS:requests.options('http://httpbin.org/get')
        如果发现请求的URL地址中参数是跟在"?"的后面,如?key=val,requests提供了传递参数的方法,允许使用params关键字参数,以一个字符串字典来提供这些参数,允许使用data关键字参数来提供表单形式的数据
请求headers处理
    如果通过GET或POST以及其他请求方式,而出现403错误,多数为服务器为了防止恶意采集信息、使用的反爬虫设置而拒绝访问,通过模拟浏览器的头部信息来进行访问可以解决反爬设置
    (1)通过例如网络监视器获取用户代理
    (2) url='url'
        header={"user-agent": ""}
        response=requests.get(url,headers=header)

```
        print(response.content)
网络超时
    在访问一个网页时,如果该网页长时间未响应,系统就会判断该网页超时
    网络异常信息中,requests模块提供了:
        ReadTimeout 超时异常
        HTTPError   HTTP异常
        RequestException    请求异常
代理服务
    在爬取网页的过程中,经常会出现不久前可以爬取的网页现在无法爬取,是因为IP被爬取的网站的服务器屏蔽,用代理服务可以解决
    设置代理时,首先找到代理地址如122.114.31.177:808、127.0.0.1:7890
HTML解析之BeautifulSoup
    BeautifulSoup是一个用于从HTML和XML文件中提取数据的Python库,提供一些简单的函数来处理导航、搜索、修改分析树等功能
    BeautifulSoup自动将输入文档转换为Unicode编码,输出文档转换为utf-8编码,不需要考虑编码方式,除非文档没有指定一个编码方式,这是仅仅需要说明一下原始编码方式即可
BeautifulSoup的安装
    BeautifulSoup 3已经停止开发,目前推荐使用的是BeautifulSoup 4,但被移植到bs4中,导入时需要from bs4
    Beaytiful支持Python标准库中包含的HTML解析器,也支持第三方Python解析器,包括lxml、html5lib
优缺点比较
    Python标准库    BeautifulSoup(markupp,"html.parser")    Python标准库、执行速度适中  老版本文档容错能力差
    lxml的HTML解析器    BeautifulSoup(markup,"lxml")    速度快、文档容错能力强  需要安装C语言库
    lxml的XML解析器 BeautifulSoup(markup,"lxml-xml"/"xml")  速度快、唯一支持XML的解析器 需要安装C语言库
    html5lib    BeautifulSoup(markup,"html5lib")    最好的容错性、以浏览器的方式解析文档、生成HTML5式的文档 速度慢、不依赖外部扩展
BeautifulSoup的使用
    导入bs4库,HTML代码
    创建Beautiful对象,指定解析器为lxml,通过打印的方式将解析的HTML代码显示
        soup=BeautifulSoup(html_doc,features="lxml")
        prettify()将代码进行格式化处理
```

17.3 网络爬虫开发常用框架

```
爬虫框架就是一些爬虫项目的半成品，可以将一些爬虫常用的功能写好
Scrapy爬虫框架
    高效率地爬取Web页面并从页面中提取结构化的数据
Crawley爬虫框架
    改变人们从互联网中提取数据的方式
    具体特性
        基于Eventlet构建的高速网络爬虫框架
        可以将数据存储在关系数据库中,如Postgres、MySQL、Oracle、Sqlite
        可以将爬取的数据导入为Json、XML格式
        支持非关系数据库,如Mongoodb和Couchdb
        支持命令行工具
        可以使用您喜欢的工具进行数据的提取,如XPath或Pyquery工具
        支持使用Cookie登录或访问那些只有登录才可以访问的网页
        简单易学
PySpider爬虫框架
    分布式架构,支持多种数据库后端,强大的WebUI支持脚本编辑器、任务监视器、项目管理器以及结果查看器
    具体特性
        Python脚本控制,可以用任何您喜欢的html解析包(内置pyquery)
```

使用Web界面编写调试脚本、起停脚本,监控执行状态,查看活动历史,获取结果产出
支持MySQL、MongoDB、Redis、SQLite、Elasticsearch、PostgreSQL与SQLAlchemy
支持RabbitMQ、Beanstalk、Redis和Kombu作为消息队列
支持抓取JavaScripy的页面
强大的调度控制,支持超时重爬及优先级设置
组件可替换,支持单机/分布式部署,支持Docker部署

17.4 实战项目:快手爬票

搭建QT环境
QT是Python开发窗体的工具之一，它不仅与Python有者良好的兼容性，还可有通过可视化拖曳的方式进行窗体的构建，提高开发人员的开发效率，因此收到开发人员的喜爱。
由于QT在创建窗体项目时会自动生成扩展名为ui的文件，该文件需要转换为py文件后才可以被Python识别，所以需要为QT和Pycharm开发工具进行配置
在Pycharm Project Interpreter中添加Python,并为其添加PyQt5标准模块
添加Qt Designer启动快捷工具,Working directory设置为$ProjectFileDir$
添加Qt生成的ui文件转换为py文件快捷工具PyUIC
Arguments: -m PyQt5.uic.pyuic $FileName$ -o $FileNameWithoutExtension$.py
working directory设置为$FileDir$
配置好后只需要启动Pycharm开发工具
主窗体设计
Qt拖曳控件
Pycharm开发工具创建新的Python项目
依次选择Tools-External Tools-Qt Designer
利用Qt Designer创建Main Windows
根据设计思路将指定的控件拖曳至主窗体中
除了主窗体默认创建的QWidget控件以外，其他每个QWidget就是一个显示区域的容器
窗体设计完成后,保存为.iu文件,然后在Pycharm中用Tools-External Tools-PyUIC转换
代码调试细节
自动生成的代码中已经导入了PyQt5以及其内部的常用模块.PyQt5时一套Python绑定Digia QT5应用的框架,它可用于Python 2.x和Python 3.x的版本当中
PyQt5常见类别模块
QtCore此模块用于处理时间、文件和目录、各种数据类型、流、URL、MIME类型、线程或进程
QtGui此模块包含类窗口系统集成、事件处理、二维图形、基本成像、字体和文本.它还包含了一套完整的OpenGL和OpenGL ES的绑定
QtWidgets此模块中包含的类,提供了一组用于创建经典桌面风格用户界面的UI元素
QtMultimedia此模块中包含的类,用于处理多媒体内容和API来访问的相机、收音机功能
QtNetwork此模块中包含网络编程的类.通过这些类使网络编程更简单、更便携,便于TCP/IP和UDP客户端和服务器的编码
QtPositioning此模块中包含的类,利用各种可能的来源确定位置,包括卫星、Wi-Fi等
QtWebSockets此模块中包含实现WebSocket协议的类
QtXml此模块中包含用于处理XML文件中的类.该模块为SAX和DOM API提供了解决办法
QtSvg此模块中提供了用于显示SVG文件内容的类.SVG是可缩放矢量图形,是用于描述XML中的二维图形的一种格式
QtSql此模块提供了用于处理数据库的类
QtTest此模块包含的功能,使QtPy5应用程序的单元测试
通过代码来调试主窗体中各种控件的细节处理,以及相应的属性
setupUi()方法设置窗体及其控件
window.setObjectName("")设置窗体对象名称
window.resize(x,y)设置窗体大小
window.setMini/MaximumSize(QtCore.QSize(x,y))窗体最小值、最大值
widget=QtWidgets.QWidget(window)窗体的widget控件
name.setObjectName("")设置对象名称
from PyQt5.QtGui import QPalette, QPixmap, QColor用于对控件设置背景图片

通过label控件显示图片
name=QtWidgets.QLabel(widget)设置label所属窗体
name.setGeometry(Qtcore.QRect(x,y,w,l))设置矩形控件在窗体中位置
img=QPixmap('')导入图片
name.setPixmap(img)设置调色板,将导入的图片加载入label
为widget设置背景图片需要为控件开启自动填充背景功能,然后创建调色板对象,指定调色板背景图片,最后为控件设置对应的调色板
widge.setAutoFillBackground(True)开启自动填充背景
palette=QPalette()调色板类

palette.setBrush(QPalette.Background,QtGui,QBrush(QtGui.QPixmap('')))
设置字体相关
font = QtGui.QFont()
font.setPointSize(10)
主窗体设置主Widget
MainWindow.setCentralWidget(self.centralwidget)
通过代码修改窗体或空间文字时,需要在retranslateUi()方法中进行设置
MainWindow.setWindowTitle(_translate("MainWindow",""))
name.settext(_translate("MainWindow",""))
在代码块最外层创建show_MainWindow()方法,用于显示窗体
app=QtWidgets.QApplication(sys.argv)实例化QApplication类,作为GUI主程序入口
MainWindow=QtWidgets.QMainWindow()创建MainWindow
ui=Ui_MainWindow()实例UI类
ui.setupUi(MainWindow)设置窗体UI
MainWindow.show()显示窗体
sys.exit(app.exec_())当窗体创建完成,结束主循环过程
分析网页请求参数
在发送请求时,地址中需要填写必要的参数,否则后台将无法返回前台所需要的正确信息.
通过浏览器访问网站获取请求中的完整查票请求地址,以及请求地址中的必要参数
随着网站的更新,请求地址会发生改变,要以当时获取的地址为准
下载文本转换文件
网络请求中js可能附带文本转换文件
在分析信息位置时,可以想到点击按钮仅实现了发送查询的网络请求,而并没有发现讲中文转换为英文的相关处理,此事可以判断在进入查询页面时就已经得到了将中文转换为英文的相关信息,可以试图刷新页面查看器网络监视器中的网络请求
信息的请求与显示
发送与分析信息的查询请求
访问查询请求地址,浏览器以json的方式返回查询信息
在看到加密信息后先分析数据中是否含有可用的信息,对照查询信息.
创建query()方法,创建查询请求的完整地址并通过format()方法为地址进行格式化.再将返回的json数据转换为字典类型,最后通过字典类型键值的方法取出对应的数据并进行整理与分类
由于返回的json信息顺序比较乱,所以在获取指定的数据时只能通过tmp_list分割后的列表中将数据与浏览器查询页面中的数据逐个对比才能找出数据所对应的位置
主窗体中显示查票信息
导入方法

```
from query_request import *
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
self.tableView = QtWidgets.QTableView(self.centralwidget)
self.tableView.setGeometry(QtCore.QRect(0, 320, 960, 440))
self.tableView.setObjectName("tableView")
self.model = QStandardItemModel()创建存储数据的模式

self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)根据空间自动改变列宽度并且不可修改列宽度
self.tableView.horizontalHeader().setVisible(False)设置表头不可见
```

```
            self.tableView.verticalHeader().setVisible(False)纵向表头不可见
            self.tableView.setEditTriggers(QAbstractItemView.NoEditTriggers)设置表格内容不可编辑
            self.tableView.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOn)垂直滚动条始终开启
        复选框时间处理
            self.displayTable(len(type_data),16,type_data)通过表格显示数据
        显示主窗体非法操作的消息提示框
            msg_box=QMessageBox(QMessageBox.Warning,title,message)
            msg_box.exec_()
        显示信息的表格与内容
            item=QStandardItem(data[row][column])添加表格内容
            self.model.setItem(row,column,item)向表格存储模式中添加表格信息
            self.tableView.setModel(self.model)设置表格存储数据的模式
        查询按钮单击事件
            self.textEdit.toPlainText()获取编辑框的输入内容
            messageDialog()弹出消息
        设置复选框选中与取消事件
            self.pushButton.clicked.connect(self.on_click)  # 查询按钮指定单击事件的方法
            self.name.stateChanged.connect(self.name)  # 选中与取消事件
```

18 使用进程和线程

18.1 什么是进程

```
多任务,指操作系统能够执行多个任务,每个任务就是一个进程
进程(Process)是计算机中已运行程序的实体,进程与程序不同,程序本身只是指令、数据及其组织形式的描述,进程才是程序的真正运行和实例
```

18.2 创建进程的常用形式

```
Python中有多个模块可以创建进程,常用的有os.fork()函数、multiprocessing模块和Pool进程池
    os.fork()只适合在UNIX/Linux/Mac系统上运行,在Windows操作系统不可用
使用multiprocessing模块创建进程
    该模块提供Process类来代表一个进程对象
        Process([group[,target[,name[,args[,kwargs]]]]])
            group参数未使用,值始终为None
            target表示当前进程启动时执行的可调用对象
            name为当前进程实例的别名
            args表示传递给target函数的参数元组
            kwargs表示传递给target函数的参数字典
        Process实例方法
            start()启动子进程
            is_alive()判断进程实例是否还在执行
            join([timeout])是否等待进程实例执行结束,或等待多少秒
            start()启动进程实例(创建子进程)
            run()如果没有给定target参数,对这个对象调用start()方法时,就将执行对象中的run()方法
            terminate()不管任务是否完成,立即终止
        Process类还有如下常用属性
            name当前进程实例别名
            pid当前进程实例的PID值
        os.getpid()获取当前进程PID,os.getppid()获取当前进程父进程PID
        第一次实例化Process类时,会为name属性默认
使用Process子类创建进程
```

Process(target=test)方式实现多进程可以处理简单任务
处理复杂任务的进程,通常定义一个继承Process类,每次实例化类时便实例化一个进程对象
该子类初始化时需要调用Process__init__()
使用进程池Pool创建进程
使用Process类创建进程时,如果创建多个进程时,实例化Process类较多
multiprocessing模块提供Pool类,即Pool进程池
Pool类常用方法
Pool(n),定义一个最大进程数为n的进程池
apply_async(func[,args[,kwds]])使用非阻塞方式调用func函数(并行执行,堵塞方式必须等待上一个进程退出才能执行下一个进程),args为传递给func的参数列表,kwds为传递给func的关键字列表
apply(func[,args[,kwds]])使用非阻塞方式调用func函数
close()关闭Pool,使其不再接收新的任务
terminate()不管任务是否完成,立即终止
join()主进程阻塞,等待子进程的退出,必须在close或terminate之后使用
使用阻塞方式执行多个任务,必须等待上个进程结束才能执行下一个进程
非阻塞则可以并行

18.3 进程间通信

进程间没有数据通信,multiprocessing模块提供Queue(队列)、Pipes(管道)等多种方式来交换数据
队列简介
队列(Queue)就是模仿现实中的排队,先进先出
多进程队列的使用
进程之间有时需要通信,操作系统提供了很多机制来实现进程间的通信,可以使用multiprocessing模块的Queue实现多进程之间的数据传递,Queue本身是一个消息队列程序
初始化Queue()对象,若括号中没有指定最大可接收的消息数量,或者数量为负值
Queue()常用方法
Queue.qsize()返回当前队列包含的消息数量
Queue.empty()如果队列为空,返回True,反之返回False
Queue.full()如果队列满了,返回True,反之返回False
Queue.get([block[,timeout]])获取队列中的一条信息,然后将其从队列中移除,block默认值为True
如果block使用默认值,且没有设置timeout(单位:秒),消息队列为空,此时程序将被阻塞(停在阻塞状态),直到从消息队列读到消息为止,如果设置了timeout,则会等待timeout秒,若还没读到任何消息,则会抛出Queue.Empty异常
如果block值为False,消息队列为空,则会抛出Queue.Empty异常
Queue.get_nowait()相当于Queue.get(False)
Queue.put(item,[block[timeout]])将item消息写入队列, block默认值为True
如果block使用默认值,且没有设置timeout(单位:秒),消息队列如果已经没有空间可写入,此时程序将被阻塞(停在写入状态),直到从消息队列腾出空间为止,如果设置了timeout,则会等待timeout秒,若还没有空间,则抛出Queue.Full异常
如果block值为False,消息队列没有空间可写入,则会立刻抛出Queue.Full异常
Queue.put_nowait(item)相当于Queue.put(item,False)
使用队列在进程间通信
结合multiprocessing.Process和multiprocessing.Queue可以实现进程间的通信

18.4 什么是线程

如果需要同时处理多个任务,可以在一个应用程序内使用多个进程,每个进程负责完成一部分工作;另一种将工作细分为多个任务的方法是使用一个进程内的多个线程
线程是操作系统能够进行运算调度的最小单位.它被包含在进程之中,是进程中的实际运作单位.一个线程指的是进程中一个单一顺序的控制流,一个进程中可以并发多个线程,每个线程并行不同的任务

18.5 创建线程

由于线程是操作系统直接支持的执行单元,高级语言(如Python、Java等)通常都内置多线程的支持.Python的标准库提供了两个模块:_thread和threading,_thread是低级模块,threading是高级模块,对_thread进行了封装.绝大多数情况下,只需要使用threading这个模块
使用threading模块创建线程
    threading提供了一个Thread类来代表一个线程对象,语法:
        Thread([group [,target [,name [,args [kwargs]]]]])
            group值为None,为以后版本而保留
            target表示一个可调用对象,线程启动时,run()方法将调用此对象,默认值为None,表示不调用任何内容
            name表示当前线程名称,默认创建一个Thread-N格式的唯一名称
            args表示传递给target函数的参数元组
            kwargs表示传递给target函数的参数字典
        对比发现,Thread类和Process类的方法基本相同
    threading.current_thread().name获取当前线程名称
使用Thread子类创建线程
    创建一个子类SubThread,继承threading.Thread线程类,并定义一个run()方法;调用start()方法开启线程,会自动调用run()方法

18.6 线程间通信

一个进程内的所有线程共享全局变量,能够在不适用其他方式的前提下完成多线程之间数据共享
什么是互斥锁
    由于线程可以对全局变量随意修改,就可能造成多线程之间对全局变量的混乱
    互斥锁Mutual exclusion,缩写Mutex,防止多个线程同时读写某一内存区域.互斥锁为资源引入一个状态:锁定和非锁定.某个线程要更改共享数据时,先将其锁定,此时资源的状态为"锁定",其他线程不能更改;直到该线程释放资源,将资源的状态变成"非锁定",其他的线程才能再次锁定该资源.互斥锁保证了每次只有一个线程进行写入操作,从而保证了多线程情况下数据的正确性
使用互斥锁
    在threading模块中使用Lock类可以方便处理锁定,Lock类的两个方法:acquire()锁定和release()释放锁
        mutex=threading.Lock()创建锁
        mutex=acquire([blocking])锁定
        mutex=release()释放锁
            acquire([blocking])获取锁定,如果有必要,需要阻塞到锁定释放为止.如果提供blocking参数并将它设置为False,当无法获取锁定时将立即返回False,如果成功获取锁定则返回True
            release()释放一个锁定,当锁定处于未锁定状态时,或者从与原本调用acquire()方法的不同线程调用此方法,将出现错误
    使用互斥锁时,要避免死锁.在多任务系统下,当一个或多个线程等待系统资源,而系统资源又被线程本身或其他线程占用时,就形成了死锁
使用队列在线程间通信
    multiprocessing模块的Queue队列可以实现进程间通信,线程间也可以使用queue模块的Queue队列
    使用Queue在线程间通信通常应用于生产者消费者模式.产生数据的模块称为生产者,而处理数据的模块称为消费者.在生产者与消费者之间的缓冲区称为仓库.生产者负责往仓库运输商品,而消费者负责从仓库立取出商品,这就构成了生产者消费者模式

19 网络编程

19.1 网络基础

为什么要使用通信协议
    计算机为了联网,就必须规定通信协议.早期的计算机网络都是由各厂商自己规定的一套协议,IBM、Apple和Microsoft都有各自的网络协议,互不兼容

为了把全世界的所有不同类型的计算机都连接起来,就必须规定一套全球通用的协议,为了实现互联网这个目标,互联网协议簇(Internet Protocol Suite)就是通用协议标准出现了.只要支持协议,便可联入互联网

TCP/IP简介

互联网协议包含了上百种协议标准,但是最重要的两个协议是TCP和IP协议,所以把互联网的协议简称为TCP/IP协议

IP协议

在通信时,通信双方必须知道对方的标识.互联网上每个计算机的唯一标识就是IP地址,实际上是一个32位整数(IPv4),以字符串表示的IP地址如172.16.254.1实际上是把32位整数按8位分组后的数字表示,目的是便于阅读

IP协议负责把数据从一台计算机通过网络发送到另一台计算机.数据被分割成一小块一小块,类似于将一个大包裹拆分成几个小包裹,然后通过IP包发送出去.由于互联网链路复杂,两台计算机之间经常有多条线路,路由器就负责决定如何把一个IP包转发出去.IP包的特点是按块发送,途径多PP个路由,但不保证都能到达,也不保证顺序到达

TCP协议

TCP协议是建立在IP协议之上的,TCP协议负责在两台计算机之间建立可靠连接,保证数据包按顺序到达.TCP协议会通过3次握手建立可靠连接

然后对每个IP包编号,确保对方按顺序收到,如果包丢掉了,就自动重发

许多常用的更高级的协议都是建立在TCP协议基础上的,比如用于浏览器的HTTP协议、发送邮件的SMTP协议等,一个TCP报文除了包含要传输的数据外,还包含源IP地址和目标IP地址,源端口和目标窗口

端口:在两台计算机通信时,只发IP地址是不够,因为同一台计算机上运行着多个网络程序.一个TCP报文来了之后,交给对象需要端口号来区分.每个网络程序都向操作系统申请唯一的端口号,两个进程在两台计算机之间建立网络连接就需要有各自的IP地址和各自的端口号

一个进程也可能同时与多个计算机建立连接,因此它会申请很多端口.端口号不是随意使用的,而是按照一定的规定进行分配.ex:80 for HTTP,21 for FTP

UDP简介

相对TCP协议,UDP协议则是面向无连接的协议.使用UDP协议时,不需要建立链接,只需要知道对方的IP地址和端口号,就可用直接发送数据包.但是,数据无法保证一定到达.虽然使用UDP传输数据不可靠,但是它的优点是比TCP协议快.对于不要求可靠到达的数据,就可以使用UDP协议

Socket简介

为了让两个程序通过网络进行通信,二者均必须使用Socket套接字.Socket英文原义是孔或插座,通常也称作套接字,用于描述IP地址和端口,是一个通信链的句柄,可以用来实现不同虚拟机或不同计算机之间的通信.在Internet上的主机上一般运行了多个服务软件,同时提供几种服务.每种服务都打开一个Socket,并绑定到一个端口上,不同的端口对应不同的服务

在Python中使用socket模块的函数socket就可以完成

s=socket.socket(AddressFamily,Type)

函数socket.socket创建一个socket,生成一个Socket对象,返回该socket的描述符

AddressFamily,可以选择AF_INET(用于Internet进程间通信)或者AF_UNIX(用于同一台机器进程间通信),实际工作中常用AF_INET

Type,套接字类型,可以是SOCK_STREAM(流式套接字,主要用于TCP协议)或者SOCK_DGRAM(数据报套接字,主要用于UDP协议)

Socket对象的内置方法

s.bind()绑定地址(host,port)到套接字,在AF_INET下以元组(host,port)的形式表示地址

s.listen()开始TCP监听.backlog指定在拒绝连接之前,操作系统可以挂起的最大连接数.该值至少为1,大部分应用程序设为5即可

s.accept()被动接收TCP客户端连接,并且以阻塞方式等待连接的到来,返回连接socket和address

s.connect()主动初始化TCP服务器连接,一般address的格式为元组(hostname,port),如果连接出错,则返回socket.error错误

s.recv()接收TCP数据,数据以字符串形式返回,bufsize指定要接收的最大数据量.flag提供有关消息的其他信息,通常可以忽略

s.send()发送TCP数据,将string中的数据发送到连接的套接字.返回值是要发送的字节数量,该数量可能小于string的字节大小

s.sendall()完整发送TCP数据.将string中的数据发送到连接的套接字,但在返回之前会尝试发送所有数据.成功则返回None,失败则抛出异常

s.recvfrom()接收UDP数据,与recv()类似,但返回值是(data,address).其中data是包含接收数据的字符串,address是发送数据的套接字地址

s.sendto()发送UDP数据,将数据发送到套接字,address是形式为(ipaddr,port)的元组,指定远程地址.返回值是发送的字节数
s.close()关闭套接字

19.2 TCP编程

由于TCP连接具有安全可靠的特性,所以TCP应用更为广泛.创建TCP连接时,主动发起连接的叫客户端,被动响应连接的叫服务器.访问网站时,个人计算机是客户端,浏览器会主动向网站发起连接,TCP连接建立起来后,后面的通信就是发送网页内容
创建TCP服务器
在程序中,如果想要完成一个服务器的功能,需要的流程如下
使用socket创建一个套接字
使用bind绑定ip和port
使用listen使套接字变为可以被动连接
使用accept等待客户端的连接
使用recv/send接收发送数据
可以尝试给本地服务器127.0.0.1:8080发送请求
创建TCP客户端
TCP的客户端比服务器简单很多
使用socket创建一个套接字
使用connect连接ip和port
使用recv/send接收发送数据
*本地间的数据发送接收,类似于命令行(Linux自带,Windows需要要外部添加)中

```
nc -l -p 8080
telnet 127.0.0.1 8080
```

*Tip,查看ip:ipconfig;python,socket.gethostbyname(socket.gethostname())
执行TCP服务器和客户端
socket.gethostname()获取主机地址

19.3 UDP编程

UDP是面向消息的协议,通信时不需要建立连接,数据的传输自然是不可靠的,UDP一般用于多点通信和实时的数据业务:
语音广播、视频、聊天软件、TFTP(简单文件传送)、SNMP(简单网络管理协议)、RIP(路由信息协议)、DNS(域名解释)
和TCP类似,使用UDP的通信双方也分为客户端和服务器
创建UDP服务器
UDP服务器不需要TCP服务器那么多的设置,因为它们不是面向连接的.除了等待传入的连接之外,几乎不需要做其他工作
创建服务器套接字
使用bind绑定ip和port
接收/发送数据
创建UDP客户端
创建客户端套接字
接收/发送数据
接收和发送的数据都是byte,所以在发送时使用encode()将字符串转换为byte,在输出时使用decode()将byte转化为字符串
执行UDP服务器和客户端
在UDP通信模型中,在通信开始之前,不需要建立相关的连接,只需要发送数据即可

20 Web编程

20.1 Web基础

HTTP协议

用户通过浏览器输入网址访问网站,浏览器被称为客户端,网站被称为服务器.实质上就是客户端向服务器发起请求,服务器接收请求后,将处理后的信息(也成为响应)传给客户端.这个过程是通过HTTP协议实现的.

HTTP(HyperText Transfer Protocol),超文本传输协议,是互联网上应用最为广泛的一种网络协议.HTTP是利用TCP在两台计算机(通常是Web服务器和客户端)之间传输信息的协议.客户端使用Web浏览器发起HTTP请求给Web服务器,Web服务器发送被请求的信息给客户端

Web服务器

当在浏览器输入URL后,浏览器会先请求DNS服务器,获得请求站点的IP地址(即根据URL地址获取其对应的IP地址),然后发送一个HTTP Request(请求,方法)给拥有该IP的主机(服务器),接着就会接受到服务器返回的HTTP Response(响应,状态码、消息体),浏览器经过渲染之后,以一种较好的效果呈现给用户.

Web服务器的工作原理可以概括为如下4个步骤

(1)建立连接:客户端通过TCP/IP协议建立到服务器的TCP连接

(2)请求过程:客户端向服务器发送HTTP协议请求包,请求服务器里的资源文档

(3)应答过程:服务器向客户端发送HTTP协议应答包,如果请求的资源包含有动态语言的内容,那么服务器会调用动态语言的解释引擎负责处理"动态内容",并将处理后得到的数据返回给客户端.由客户端解释HTML文档,在客户端屏幕上渲染图形结果

(4)关闭连接:客户端与服务器断开

HTTP协议的常用请求方法

GET	请求指定的页面信息,并返回实体主体
POST	向指定资源提交数据进行处理请求(例如提交表单或者上传文件).数据被包含在请求体中.POST请求可能会导致新的资源的建立和/或已有资源的修改
HEAD	类似于GET请求,只不过返回的响应中没有具体的内容,用于获取报头
PUT	从客户端向服务器传送的数据取代指定的文档的内容
DELETE	请求服务器删除指定的页面
OPTIONS	允许客户端查看服务器的性能

HTTP状态码含义

1**	信息,请求收到,继续处理
2**	成功,行为被成功地接收、理解和采纳
3**	重定向,为了完成请求,必须进一步指定的动作
4**	客户端错误,请求包含语法错误或者请求无法实现
5**	服务器错误,服务器不能实现一种明显无效的请求

ex:20表示请求成功已完成,404表示服务器找不到给定的资源

通过浏览器调试工具的Network,查看请求与响应的信息

General中

Request URL	请求的URL地址,也就是服务器的URL地址
Request Method	请求方式
Status Code	状态码
Remote Address	服务器IP地址,端口号

前端基础

对于Web开发,通常分为前端(Front-End)和后端(Back-End)."前端"是与用户直接交互的部分,包括Web页面的结构、Web的外观视觉表现以及Web层面的交互实现."后端"更多的是与数据库进行交互以处理响应的业务逻辑.需要考虑的是如何实现功能、数据的存取、平台的稳定性与性能等.后端的编程语言包括Python、Java、PHP、ASP.NET等,而前端编程语言主要包括HTML、CSS和JavaScript

对于浏览网站的普通用户而言,更多的是关注网站前端的美观程度和交互效果,很少去考虑后端的实现.

1.HTML简介

HTML是用来描述网页的一种语言.HTML指的是超文本标记语言(Hyper Text Markup Language),它不是一种编程语言,而是一种标记语言.标记语言是一套标记标签,这种标记标签通常被称为HTML标签,它们是由尖括号包围的关键词,比如<html>.HTML标签通常是成对出现的,比如<h1>和</h1>.标签对中的第一个标签是开始标签,第二个标签是结束标签.Web浏览器的作用是读取HTML文档,并以网页的形式显示它们.浏览器不会显示HTML标签,而是使用标签来解释页面的内容

HTML代码中,第一行的<!DOCTYPEhtml>表示使用的是HTML5(最新HTML版本),其余的标签都是成对出现,并且在右侧的页面中,只显示标签里的内容,不显示标签

2.CSS简介

CSS是Cascading Style Sheet(层叠样式表)的缩写.CSS是一种标记语言,用于为HTML文档中的定义布局.例如,CSS涉及字体、颜色、边距、高度、宽度、背景图像、高级定位等方面.运用CSS样式可以让页面变得美观.

2.JavaScript简介

通常,我们所说的前端就是指HTML、CSS和JavaScript三项技术

HTML:定义网页的内容

CSS:描述网页的样式

JavaScript:描述网页的行为

JavaScript是一种可以嵌入在HTML代码中,由客户端浏览器运行的脚本语言.在网页中使用JavaScript代码,不仅可以实现网页特效,还可以响应用户请求,实现动态交互的功能.

静态服务器

在Web中,纯粹HTML格式的页面被称为"静态页面",早期的网站通常都是由静态页面组成的

定义一个HTTPServer()类,用__init__()初始化方法创建Socket实例,start()方法用于建立客户端连接,开启线程.handle_client()方法用于处理客户端请求,主要功能是通过正则表达式提取用户请求的文件名.最后拼接数据返回客户端

20.2 WSGI接口

CGI简介

当今Web开发已经很少使用纯静态页面,更多的是使用动态页面,如网站的登录和注册功能等.Web服务器不能处理表单中传递过来的与用户相关的数据,这不是Web服务器的职责.CGI应运而生

CGI(Commmon Gateway Interface),通过网关接口,它是一段程序,运行在服务器上.Web服务器将请求发送给CGI应用程序,再将CGI应用程序动态生成的HTML页面发送回客户端.CGI在Web服务器和应用之间充当了交互作用,这样才能够处理用户数据,生成并返回最终的动态HTML页面

CGI有明显的局限性,例如CGI进程针对每个请求进行创建,用完就抛弃.如果应用程序接收数千个请求,就会创建大量的语言解释器进程,这将导致服务器停机.于是CGI的加强版FastCGI(Fast Common Gateway Interface)应运而生

FastCGI使用进程/线程池来处理一连串的请求.这些进程/线程由FastCGI服务器管理,而不是Web服务器.FastCGI致力于减少网页服务器与CGI程序之间交互的开销,从而使服务器可以同时处理更多的网页请求

WSGI简介

FastCGI的工作模式实际上没有什么太大缺陷,但是在FastCGI标准下写异步的Web服务器还是不方便,所以WSGI就被创造出来了.

WSGI(Web Server Gateway Interface),服务器网关接口,是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.从层级上来讲要比CGI/FastCGI高级.WSGI中存在两种角色:接收请求的Server(服务器)和处理请求的Application(应用),它们底层是通过FastCGI沟通的.当Server收到一个请求后,可以通过Socket把环境变量和一个Callback回调函数传给后端Application,Application在完成页面组装后通过Callback把内容返回给Server,最后Server再将响应返回给Client.

定义WSGI接口

WSGI接口定义非常简单,它只求Web开发者实现一个函数,就可以响应HTTP请求

```
def application(environ,start_response):
    start_response('200 OK',[('Content-Type','text/html')])
    return [b'<h1>Hello,World!</h1>']
```

上面的application()函数就是符合WSGI标准的一个HTTP处理函数,它接收两个参数:

environ:一个包含所有HTTP请求信息的字典对象

start_response:一个发送HTTP响应的函数

整个application()函数本身没有涉及任何解析HTTP的部分,也就是说,把底层Web服务器解析部分和应用程序逻辑部分进行了分离,这样开发者就可以专心做一个领域了

调用application()函数,environ和start_response这两个参数需要从服务器获取,所以application()函数必须由WSGI服务器来调用.现在,很多服务器都符合WSGI规范,如Apache和Nginx等.此外Python内置了一个WSGI服务器,这就是wsgiref模块,它是用Python编写的WSGI服务器的参考实现."参考实现",是指该实现完全符合WSGI标准,但是不考虑任何运行效率,仅供开发和测试使用

运行WSGI服务

使用Python的wsgiref模块可以不用考虑服务器和客户端的连接、数据的发送和接收等问题,而专注于业务逻辑的实现.

20.3 Web框架

从零开始建立一些网站,可能会重复解决一些相同的问题,违反了良好编程的核心原则之一——DRY(不要重复自己)
大多数情况下,开发人员通常需要处理四项任务——数据的创建、读取、更新和删除,也成为CRUD.通过使用Web框架解决这问题
什么是Web框架
Web框架是用来简化Web开发的软件框架.框架的存在是为了避免用户重新发明轮子,并且在创建一个新的网站时帮助减少一些开销.典型的框架就提供了如下常用功能:
管理路由、访问数据库、管理会话和Cookies、创建模板来显示HTML、促进代码的重用
事实上,框架根本就不是什么新的东西,它只是一些能够实现功能的Python文件.我们可以把框架看作是工具的集合,而不是什么特定的东西.框架的存在使得建立网站更快、更容易.框架还促进了代码的重用
Python中常用的Web框架
WSGI(服务器网关接口),是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.也就是说,只要遵循WSGI接口规则,就可以自主开发Web框架.所以, 各种开发Web框架至少有上百个,关于Python框架优劣的讨论也仍在继续.选择主流的框架来学习使用,优点在于主流框架文档齐全,技术积累较多,社区繁盛,并能得到更好的支持
Django,可能是最广为人知和使用最广泛的Python Web框架.Django有世界上最大的社区和最多的包.它的文档非常完善,并且提供了一站式的解决方案.但是,Django系统耦合度较高,替换掉内置的功能比较麻烦
Flask,是一个轻量级Web应用框架.它的名字暗示了它的含义,基本上就是一个微型的胶水框架.Flask把Werkzeug和Jinja黏合在一起,所以它很容易被扩展.Flask也有许多的扩展可以供用户使用,Flask也有一群忠诚的粉丝和不断增加的用户群.它有一份很完善的文档,甚至还有一份唾手可得的常见范例
Bottle,这个框架相对来说比较新.它才是名副其实的微框架——大约4500行代码.除了Python标准库外,没有任何其他的依赖,甚至还有自己独特的一点模板语言.Bottle的文档很详细并且抓住了事物的本质.像Flask,也使用了装饰器来定义路径
Tornado,不单单是个框架,还是个Web服务器.一开始是为FriendFeed开发的,后来在2009年的时候也给Facebook使用.它是为了解决实时服务而诞生的.为了做到这一点,Tornado使用了异步非阻塞IO,所以运行速度非常快

21 Flask框架

21.1 Flask简介

Flask依赖两个外部库:Werkzeug和Jinja2.Werkzeug是一个WSGI(在Web应用和多种服务器之间的标准接口)工具集.Jinja2负责渲染模板.所以,在安装Flask之前,需要安装这两个外部库,而最简单的方式就是使用Virtualenv创建虚拟环境
安装虚拟环境
Virtualenv为每个不同项目提供一份Python安装.并没有真正安装多个Python副本,但是它确实提供了一种巧妙的方式来让各项目环境保持独立
安装virtualenv
pip install virtualenv安装virtualenv,virtualenv --version查看版本
创建虚拟环境
使用Virtualenv命令在当前文件夹创建Python虚拟环境.这个命令只有一个必须的参数,及虚拟环境的名字.创建虚拟环境后,当前文件夹中会出现一个子文件夹,名字就是上述命令中指定的参数,与虚拟环境相关的文件都保存在这个子文件夹中.按照惯例,一般虚拟环境会被命名为venv.
virtualenv venv在运行的目录下创建env文件夹,保存一个全新的虚拟环境,其中有一个私有的Python解释器
激活虚拟环境
在使用虚拟环境之前,通过venv\Scripts\activate激活虚拟环境
在虚拟环境下安装Flask
第一个Flask程序

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
```

```
    def hello_world():
        return 'Hello World!'
    if __name__ == '__main__':
        app.run(debug=True)
        #导入Flask类,这个类的实例将会是WSGI应用程序
        #创建一个该类的实例,第一个参数是应用模块或者包的名称.如果使用单一的模块,应该使用__name__,因为模块的名称将会因其作为单独应用启动还是作为模块导入而有所不同(也是'__main__'或实际的导入名).这是必需的,这样Flask才知道到哪去找模板、静态文件等.详情见Flask的文档
        使用route()装饰器告诉Flask什么样的URL能触发函数
        这个函数的名字也在生成URL时被特定的函数采用,这个函数返回我们想要显示在用户的浏览器中的信息
        用run()函数让应用运行在本地服务器上
    关闭服务器,按Ctrl+C快捷键
```

21.2 Flask基础

```
开启调试模式
    虽然run()方法适用于启动本地的开发服务器,但是用户每次修改代码后都要手动重启它.这样并不够优雅,而且Flask可以做到更好.如果启用了调试支持,服务器会在代码修改后自动重新载入,并在发生错误时提供一个相当有用的调试器
    有两种途径来启用调试模式,两种方法的效果完全相同
        直接在应用对象上设置
            app.debug=True
            app.run()
        作为run()方法的一个参数传入
            app.run(debug=True)
路由
    客户端(如Web浏览器)把请求发给Web服务器,Web服务器再把请求发送给Flask程序实例.程序实例需要知道每个URL请求运行哪些代码,所以保存了一个URL到Python函数的映射关系.处理URL和函数之间关系的程序称为路由
    在Flask程序中定义路由的最简便方式,是使用程序程序实例提供的app.route修饰器把修饰的函数注册为路由.
        修饰方式
            @app.route('/')
            def index():
                return '<h1>Hello World!<h1/>'
        修饰器是Python语言的标准特性,可以使用不同的方式修改函数的行为.惯常用法是使用修饰器把函数注册为事件的处理程序
    不仅如此,可以构造含有动态部分的URL,也可以在一个函数上附着多个规则
        1.变量规则
            要给URL添加变量部分,可以把这些特殊的字段标记为<variable_name>,这个部分将会作为命名参数传递到需要的函数.规则可以用<converter:variable_name>指定一个可选的转换器
            @app.route('/post/<int:post_id>')
            def show_post(post_id):
                return 'Post %d' % post_id
            @app.route('/user/<username>')
            def show_user_profile(username):
                return 'User %s' % username
            三种转换器
                int接受整数
                float同int,接收浮点数
                path和默认的相似,但也接收斜线
        2.构造URL
            如果Flask能匹配URL,是那么Flask也能生成它们.可以用url_for()来给指定的函数构造URL.它接收函数名作为第一个参数,也接收对应URL规则的变量部分的命名参数.未知变量部分会添加到URL末尾作为查询函数
            @app.route('/url/')
```

```
            def get_url():
                return url_for('show_post',post_id=2)
            #result:/post/5
```

3.HTTP方法

HTTP(与Web应用会话的协议)有许多不同的访问URL方法.默认情况下,路由只回应GET请求,但是通过route()装饰器传递methods函数可以改变这个行为.ex:

```
            @app.route('/login',methods=['GET','POST'])
            def login():
                if request.method=='POST':
                    do_the_login()
                else:
                    show_the_login_form()
```

HTTP方法(也经常被叫做"谓词")告知服务器,客户端想对请求的页面做些什么.

常见的HTTP方法

GET 浏览器告知服务器:只获取页面上的信息并发给我.这是最常用的方法

HEAD 浏览器告知服务器:欲获取信息,但是只关心消息头.应用应像处理GET请求一样来处理它,但是不分发实际内容.在Flask中你完全无须人工干预,底层的Werkzeug库已经替你处理好了

POST 浏览器告诉服务器:想在URL上发布新消息.并且,服务器必须确保数据已储存且仅存储一次.这是HTML表单通常发送数据到服务器上的方法

PUT 类似POST但是服务器可能触发了存储过程多次,多次覆盖掉旧值.考虑到传输中数据可能会丢失,在这种情况下浏览器和服务器之间的系统可能安全地第二次接受请求,而不破坏其他东西.因为POST只触法一次,所以用POST是不可能的.

DELETE 删除给定位置的信息

OPTIONS 给客户端提供一个敏捷的途径来弄清这个URL支持哪些HTTP方法.从Flask 0.6开始,实现了自动处理

静态文件

动态Web应用也会需要静态文件,通常是CSS和JavaScript文件.理想情况下,你已经配置好Web服务器来提供静态文件,但是在开发过程中,Flask也可以做到.只要在你的包中或是模块的所在目录中创建一个名为static的文件夹,在应用中使用/static即可访问

给静态文件生成URL,使用特殊的"static"端点名

```
url_for('static',filename='style.css')
```

这个文件应该在储存在文件系统上的static/style.css

蓝图

Flask使用蓝图(blueprints)的概念来在一个应用中或跨应用制作应用组件和支持通用的模式.蓝图很好地简化了大型应用工作的方式,并提供给Flask扩展在应用上注册操作的核心方法.一个Blueprint对象与Flask应用对象的工作方式很像,但它确实不是一个应用,而是一个描述如何构建或扩展应用的蓝图.

Flask中的蓝图为如下这些情况而设计

把一个应用分解为一个蓝图的集合.这对大型应用是理想的.一个项目可以实例化一个应用对象,初始化几个扩展,并注册一个集合的蓝图

以URL前缀和/或子域名在应用上注册一个蓝图.URL前缀/子域名中的参数即成为这个蓝图下的所有视图函数的共同的视图参数(默认情况下)

在一个应用中用不同的URL规则多次注册一个蓝图

通过蓝图提供模板过滤器、静态文件、模板和其他功能.一个蓝图不一定要实现应用或者视图函数.

初始化一个FLask扩展时,在这些情况中注册一个蓝图

Flask中的蓝图不是即插应用,因为它实际并不是一个应用——它是可以注册,甚至可以多次注册到应用上的操作集合.为什么不使用多个应用对象呢?你可以做到那样,但是你的应用的配置是分开的,并需要在WSGI层管理

22.3 模板

模板是一个包含响应文本的文件,其中包含用占位变量表示的动态部分,其具体值值在请求的上下文中才能知道.使用真实值替换变量,再返回最终得到的响应字符串,这一过程称为渲染.为了渲染模板,Flask使用了一个名为Jinja2的强大模板引擎

渲染模板

默认情况下,Flask在程序文件夹中的templates子文件中寻找模板.

```
        @app.route('/')
        def hello_world():
```

```
        return render_template('index.html')
    @app.route('/user/<username>')
    def show_user_profile(username):
        return render_template('user.html', name=username)
```

Flask提供的render_template()函数把Jinja2模板引擎集成到了程序中.render_template()函数的第一个参数是模板的文件名.随后的参数都是键值对,表示模板中变量对应的真实值.在这段代码中,第二个模板收到名为name的变量.前例中的name=username是关键字参数,这类关键字参数很常见.左边的name表示参数名,就是模板中使用的占位符;右边的name是当前作用域中的变量,表示同名参数的值

变量

前面在模板中使用的{{name}}结构表示一个变量,它是一种特殊的占位符,告诉模板引擎这个位置的值从渲染模板时使用的数据中获取.Jinja2能识别所有类型的变量,甚至是一些复杂的类型,如列表、字典和对象.在模板中使用变量的一些示例如下

```
    <p>从字典中取一个值:{{mydict['key']}}.</p>
    <p>从列表中取一个值:{{mylist[3]}}.</p>
    <p>从列表中取一个带索引的值:{{mylist[myintvar]}}.</p>
    <p>从对象的方法中取一个值:{{myobj.somemethod()}}.</p>
```

可以使用过滤器修改变量,过滤器名添加在变量名之后,中间使用竖线分隔.下述模板以首字母大写形式显示变量name的值

```
    Hello,{{name|capitalize}}
```

Jinja2提供的部分常用过滤器

```
    safe     渲染时不转义
    capitalize  把值的首字母转换成大写,其他字母转换成小写
    lower    把值转换成小写形式
    upper    把值转换成大写形式
    title    把值中每个单词的首字母都转换成大写
    trim     把值的首尾空格去掉
    striptagsd  渲染之前把值中所有的HTML标签都删掉
```

safe过滤器值得特别说明一下.默认情况下,处于安全考虑,Jinja2会转义所有变量.例如,如果一个变量的值为'<h1>Hello</h1>',Jinja2会将其渲染成'&1t;h1>Hello&1t;/h1>',浏览器能显示这个h1元素,但是不会进行解释.很多情况下需要显示变量中储存的HTML代码,这时就可使用safe过滤器

控制结构

Jinja2提供了多种控制结构,可用来改变模板的渲染流程.以下介绍最有用的控制结构

在模板中使用条件控制语句

```
    {% if user %}
    Hello,{{ user }}!
    {% else %}
    Hello,Stranger!
    {% endif %}
```

另一种常见需求是在模板中渲染一组元素.使用for循环实现这一需求

```
    <ul>
    {% for comment in comments %}
    <li>{{ comment }}</li>
    {% end for %}
    </ul>
```

Jinja2还支持宏.宏类似于Python代码中的函数

```
    {% marco render_comment(comment) %}
    <li>{{ comment }}</li>
    {% endmarco %}
    <ul>
    {% for comment in comments %}
    {{ render_comment(comment) }}
    {% endfor %}
    </ul>
```

为了重复使用宏,可也将其保存在单独的文件中,然后在需要使用的模板中导入

```
    {% import 'marcos.html' as marcos %}
    <ul>
    {% for comment in comments %}
    {{ marcos.render_comment(comment) }}
```

```
        {% endfor %}
        </ul>
```

需要在多处重复使用代码的模板代码片段可以写入单独的文件,再包含在所有模板中,以避免重复

```
        {% include 'common.html' %}
```

另一种重复使用代码的强大方式是模板继承,它类似于Python代码中的类继承

首先创建一个名为base.html的基模板

```
            <html>
            <head>
            {% block head %}
            <title>{% block title %}{% endblock %} - My Application</title>
            {% endblock %}
            </head>
            <body>
            {% block body %}
            {% endblock %}
            </body>
            <html>
```

bloc标签定义的元素可在衍生模板中修改.上例中定义了名为head、title和body的块.注意,title包含在head中.下面这个示例是基模板的衍生模板

```
            {% extends "base.html" %}
            {% block title %}Index{% endblock %}
            {% block head %}
            {{ super() }}
            <style>
            </style>
            <% endblock %>
            {% block body %}
            <h1>Hello, World!</h1>
            {% endblock %}
```

extends指令声明这个模板衍生自base.html.在extends指令之后,基模板中的3个块被重新定义,模板引擎会将其插入适当的位置,注意新定义的head块,在基模板中其内容不是空的,所以使用super()获取原来的内容

22.4 Web表单

表单是允许用户跟你的Web应用交互的基本元素.Flask自己不会帮你处理表单,但Flask-WTF插件允许用户在Flask应用中使用著名的WTForms包.这个包使得定义表单和处理表单功能变得轻松.

CSRF保护和验证

CSRF全称是cross site request forgery,即跨站请求伪造.CSRF通过第三方伪造表单数据,以POST到应用服务器上.

判断一个POST请求是否来自网站自己的表单:WTForms在渲染每个表单时生成一个独一无二的token,使这一切变得可能.那个token将在POST请求中随表单数据一起进行传递,并且会在表单被接受之前进行验证.关键在于token的值取决于储存在用户的会话(cookies)中的一个值,而且会在一定时间(默认30min)之后过时.这样只有登录了页面的用户才能提交一个有效的表单,而且仅仅是在登录页面30min之内才能这么做

默认情况下,Flask-WTF能保护所有表单免受跨站请求伪造的攻击.恶意网站把请求发送到被攻击者已登录的其他网站时就会引发CSRF攻击.为了实现CSRF保护,Flask-WTF需要程序设置一个密匙.Flask-WTF使用这个密匙生成加密令牌,再用令牌验证请求中表单数据的真伪.设置密匙的方法

```
        app=Flask(__name__)
        app.config['SECRET_KEY']='mrsoft'
```

app.config字典可用来储存框架、扩展和程序本身的配置变量.使用标准的字典句法就能把配置值添加到app.config对象中.这个对象还提供了一些方法,可以从文件或环境中导入配置值

SECRET_KEY配置变量是通用密匙,可在Flask和多个第三方扩展中使用.如其名所示,加密的强度取决于变量值的机密程度.不同的程序要使用不同的密匙,而且要保证其他人不知道你所用的字符串

表单类

使用Flask-WTF时,每个Web表单都由一个继承自Form的类表示.这个类定义表单中的一组字段,每个字段都用对象表示.字段对象可附属一个或多个验证函数.验证函数用来验证用户提交的输入值是否符合要求.使用Flask-WTF创建包含一个文本字段、密码字段和一个提交按钮的简单的Web表单

```
from flask_wtf import FlaskForm
from wtforms import StringField,PasswordField,SubmitField
from wrforms.validators import Required
classNameForm(FlaskForm)
    name=StringField('请输入姓名'.validators=[Required()])
    password=PasswordField('请输入密码',validators=[Required()])
    submit=SubmitField('Submit')
```

这个表单中的字段都定义为类变量,类变量的值是相应字段类型的对象.在上述示例中,NameForm表单中有一个名为name的文本字段、一个名为password的密码字段和一个名为submit的提交按钮.StringField类表示属性为type="text"的<input>元素.SubmitField类表示属性为type="submit"的<input>元素.字段构造函数的第一个参数是把表单渲染成HTML时使用的符号.StringField构造函数中的可选参数validators指定一个由验证函数组成的列表,在接收用户提交的数据之前验证数据.验证函数Required()确保提交的字段不为空

Form基类由Flask-WTF扩展定义,所以从flask.ext.wtf中导入.字段和验证函数却可以直接从WTForms包中导入

WTForms支持的HTML标准字段

StringField	文本字段
TextAreaField	多行文本字段
PasswordField	密码文本字段
HiddenField	隐藏文本字段
DateField	文本字段,值为datatime.date格式
DateTimeField	文本字段,值为datetime.datetime格式
IntegerField	文本字段,值为整数
DecimalField	文本字段,值为decimal.Decimal
FloatField	文本字段,值为浮点数
BooleanField	复选框,值为True和False
RadioField	一组单选框
SelectField	下拉列表
SelectMultipleField	下拉列表,可选择多个值
FileField	文件上传字段
SubmitField	表单提交按钮
FormField	把表单作为字段嵌入另一个表单
FieldList	一组指定类型的字段

WTForms内置的验证函数

Email	验证电子邮件地址
EqualTo	比较两个字段的值,常用于要求输入两次密码进行确认的情况
IPAddress	验证IPv4网络地址
Length	验证输入字符串的长度
NumberRange	验证输入的值在数字范围内
Optional	无输入值时跳过其他验证函数
Required	确保字段中有数据
Regexp	使用正则表达式验证输入值
URL	验证URL
AngOf	确保输入值在可选列表中

把表单渲染成HTML

表单字段是可调用的,在模板中调用后会渲染成HTML.假设视图函数把一个NameForm实例通过参数form传入模板,在模板中可以生成一个简单的表单

设置表单和密匙承前

```
@app.route('/', methods=['GET', 'POST'])
def index():
    form = LoginForm()
    data = {}
    if form.validate_on_submit():
        data['name'] = form.name.data
        data['password'] = form.password.data
    return render_template('index.html', form=form,data=data)
```

上述代码中,app.route修饰器中添加的methods参数告诉Flask在URL映射中把这个视图函数注册为GET和POST请求的处理程序.如果没指定methods参数,就只把视图函数注册为GET请求的处理程序.把POST加入方法列表很有必要,因为将提交表单作为POST请求进行处理更加便利.表单中也可作为GET请求提交,不过GET请求没有主体,提交的数据以查询字符串的形式附加到URL中,可在浏览器的地址栏看到.基于这个以及其他多个原因,提交表单大都作为POST请求进行处理

局部变量name和password用来存放表单中输入的有效用户名和密码,如果没有输入,其值为None.如上述代码所示,在视图函数中创建一个LoginForm类实例用于表示表单.提交表单后,如果数据能被所有验证函数接收,那么validate_on_submit()方法返回的值为True,否则返回False.这个函数的返回值决定是重新渲染表单还是处理表单提交的数据.

22 项目实战:e起去旅行网站

22.1 系统功能设计

22.1.1 系统功能结构

```
pass
```

彩蛋

```
import __hello__
输出Hello world!

import this
The Zen of Python, by Tim Peters

import antigravity
系列漫画

from __future__ import braces
SyntaxError: not a chance

from __future__ import barry_as_FLUFL
<>取代!=
```