第 1 章

初级游戏项目实战

对于学习编程的读者朋友们来说，最初下定决心学习编程的理由可能是想开发一款游戏。Python 作为一门功能强大的编程语言，可以开发出各种各样的游戏。本章将详细讲解用 Python 语言开发简易小游戏项目的知识，和读者一起体会 Python 语言的魅力。

1.1 猜数游戏

本节将详细介绍实现一个简单猜数游戏的方法。在介绍具体的实现过程之前，首先详细讲解实现本游戏需要的语法及其使用方法。

扫码看视频

1.1.1 使用条件语句

Python 语言中有三种 if 语句，分别是 if、if…else 和 if…elif…else 语句。为了节省本书篇幅，在接下来的内容中，我们只讲解前两种 if 语句的用法。

1) if 语句

最简单的 if 语句的语法格式如下所示。

```
if 判断条件：
    执行语句
```

在上述格式中，当“判断条件”为真(或非零)时表示条件成立，此时会执行 if 后面的语句，执行内容可以是多行，使用缩进来区分表示同一范围。当“判断条件”为假(或零)时会跳过 if 后面的“执行语句”，其中的“判断条件”可以是任意类型的表达式。

2) if…else 语句

在 Python 语言中，if…else 语句语法格式如下所示。

```
if 判断条件：
    statement1
else：
    statement2
```

在上述格式中，如果满足“判断条件”，就执行 statement1；如果不满足“判断条件”，则执行 statement2。

1.1.2 使用 for 循环语句

在 Python 语言中，绝大多数的循环结构都是用 for 语句来完成的。在 Java 等其他高级语言中，for 循环语句需要用循环控制变量来控制循环。而在 Python 语言的 for 循环语句中，通过循环遍历某一序列对象(例如，元组、列表、字典等)的方式构建循环，循环结束的标志是对象被遍历完成。

在 Python 语言中，for 循环语句的语法格式如下所示。

```
for iterating_var in sequence:
  statements
```

上述格式中各个参数的具体说明如下所示。

- iterating_var：表示循环变量。
- sequence：表示遍历对象，通常是元组、列表和字典等。
- statements：表示执行语句。

1.1.3 具体实现

下面的实例实现了一个猜数游戏——猜一猜年龄：系统会生成一个随机数让用户去猜，同时会给出太大或太小的提示。猜对或猜错后也会分别给出对应的提示。

实例 1-1：猜年龄游戏

实例文件 guess.py 的具体实现代码如下所示。

```
import random
guessesTaken = 0

print('你好，你是谁?')
myName = input()
number = random.randint(1, 20)
print('噢, ' + myName + ', 你很年轻啊，年龄 1 到 20 之间? ')

for guessesTaken in range(6):
    print('猜一猜')                          #打印输出文本“猜一猜”
    guess = input()
    guess = int(guess)
    if guess < number:
        print('太小!')                       #如果猜的数值小于 number，则打印输出文本“太小!”
    if guess > number:
        print('太大! ')
    if guess == number:
        break

if guess == number:
    guessesTaken = str(guessesTaken + 1)
    print('厉害 ' + myName + '你猜对了，' + guessesTaken + '很正确!')

if guess != number:
    number = str(number)
    print('别猜了，我年龄是: ' + number + '.')
```

在上述代码中，变量 number 调用 random.randint()函数产生一个随机数字，供用户猜测，

这个随机数字在 1 到 20 之间。变量 guessesTaken 的初始值为 0，将用户猜过的次数保存到这个变量中。在代码中我们设置条件 guessesTaken <6，这样可以确保循环中的代码只运行 6 次，也就是说，用户只有 6 次猜数机会。执行后会输出：

```
你好，你是谁?
aa
噢，aa，你很年轻啊，年龄 1 到 20 之间?
猜一猜
1
太小!
猜一猜
4
太小!
猜一猜
5
太小!
猜一猜
7
太小!
猜一猜
9
太小!
猜一猜
11
太小!
别猜了，我年龄是 15.
```

1.2 龙的世界

本节将详细介绍一个实现《龙的世界》游戏的方法。在介绍具体的实现过程之前，首先详细讲解实现本游戏需要的语法，并介绍这些语法的使用方法。

扫码看视频

1.2.1 使用 while 循环语句

在 Python 语言中，while 语句用于循环执行某段程序，以处理需要重复处理的相同任务。虽然绝大多数的循环结构都是用 for 循环语句来完成的，但是 while 循环语句也可以完成 for 语句的功能，只不过不如 for 循环语句那样简单明了。

在 Python 语言中，while 循环语句主要用于构建比较特别的循环。while 循环语句最大的特点是循环次数不确定，当不知道语句块或者语句需要重复多少次时，使用 while 语句是最好的选择。当 while 的条件表达式为真时，while 语句会重复执行一条语句或者语句块。

while 语句的基本格式如下所示。

```
while condition
   执行语句
```

在上述格式中，当 condition(条件表达式)为真时，会循环执行后面的“执行语句”并循环，一直到条件为假时才退出循环。如果第一次循环中条件表达式为假，那么会忽略 while 循环。如果条件表达式一直为真，会一直执行 while 循环。也就是说，会一直执行 while 循环中的“执行语句”部分，直到当条件为假时才退出循环，并执行循环体后面的语句。

1.2.2 使用函数

在 Python 程序中，使用函数之前必须先定义(声明)函数，然后才能调用它。在使用函数时，只要按照函数定义的形式向函数传递必需的参数，就可以完成相应的功能或者获得函数返回的结果。

在 Python 程序中，使用关键字 def 定义一个函数。定义函数的语法格式如下所示。

```
def<函数名>(参数列表):
   <函数语句>
   return<返回值>
```

在上述格式中，“参数列表”和“返回值”不是必需的，在 return 后面可以没有返回值，甚至也可以没有 return。如果在 return 后面没有返回值，并且没有 return 语句，这样的函数就会返回 None 值。

注意：当函数没有参数时，也必须写上包含参数的小括号，在小括号后也必须有冒号“:”。

有些函数可能既不需要传递参数，也没有返回值。例如在下面的演示代码中，使用函数输出“人生苦短，Python 是岸!”。

```
def hello() :                              #定义函数 hello()
  print("人生苦短，Python 是岸！")          #这行属于函数 hello()内的
hello()                                    #调用/使用/运行函数 hello()
```

在上述代码中，定义了一个基本的函数 hello()，其功能是输出文本“人生苦短，Python 是岸！”。执行后会输出：

```
人生苦短，Python 是岸!
```

1.2.3 实现《龙的世界》游戏

在下面的实例中实现了一个《龙的世界》游戏。在龙的世界中，龙在洞穴中装满了宝藏。有些龙很友善，愿意与你分享宝藏。而另外一些龙则很凶残，会吃掉闯入它们洞穴的任何人。玩家站在两个洞前，一个山洞住着友善的龙，另一个山洞住着饥饿的龙。玩家必须从这两个山洞之间选择一个。

实例 1-2：《龙的世界》游戏

实例文件 dragon.py 的具体实现代码如下所示。

```
import random
import time

def displayIntro():
    print('''这里是龙的世界，龙在洞穴中装满了宝藏。有些龙很友善，愿意与你分享宝藏。
          而另外一些龙则很凶残，会吃掉闯入它们洞穴的任何人。玩家站在两个洞前，一个山洞住着友善
          的龙，另一个山洞住着饥饿的龙。玩家必须从这两个山洞之间选择一个。''')
    print()

def chooseCave():
    cave = ''
    while cave != '1' and cave != '2':
        print('你选择进入哪个洞穴 ? (1 or 2)')
        cave = input()
    return cave

def checkCave(choseCave):
    print('你正在慢慢地靠近这个山洞...')
    time.sleep(2)
    print('十分黑暗、阴暗，一片混沌 ...')
    time.sleep(2)
    print('突然一条巨龙跳了出来，他张开了大大的嘴巴 ...')
    print()
    time.sleep(2)
    friendlyCave = random.randint(1, 2)
    if choseCave == str(friendlyCave):
        print('然后充满微笑地给你他的宝藏!')
    else:
        print('然后一口把你吃掉!')

playAgain = 'yes'
while playAgain == 'yes' or playAgain == 'y':
    displayIntro()
    caveNumber = chooseCave()
```

```
    checkCave(caveNumber)
    print('你还想再玩一次吗？ (yes or no)')
    playAgain = input()
```

在上述代码中，函数 chooseCave()用于询问玩家想要进入哪一个洞，是 1 号洞还是 2 号洞。在具体实现时，使用一条 while 语句来请玩家选择一个洞，while 语句标志着一个 while 循环的开始。for 循环会循环一定的次数，而 while 循环只要某一个条件为 True 就会一直重复。函数 chooseCave()需要确定玩家输入的是 1 还是 2，而不是任何其他的内容。这里会有一个循环来持续询问玩家，直到他们输入了两个有效答案中的一个为止，这就是所谓的输入验证(input validation)。执行后会输出：

```
这里是龙的世界，龙在洞穴中装满了宝藏。有些龙很友善，愿意与你分享宝藏。
而另外一些龙则很凶残，会吃掉闯入它们洞穴的任何人。玩家站在两个洞前，一个山洞住着友善的龙，
另一个山洞住着饥饿的龙。玩家必须从这两个山洞之间选择一个。

你选择进入哪个洞穴 ？ (1 or 2)
1
你正在慢慢地靠近这个山洞...
十分黑暗、阴暗，一片混沌 ...
突然一条巨龙跳了出来，他张开了大大的嘴巴 ...

然后充满微笑地给你他的宝藏！
你还想再玩一次吗？ (yes or no)
```

1.3 黑白棋游戏

本节将详细介绍实现黑白棋(Reversi)游戏的方法。在介绍本项目的实现过程中，详细讲解了每一行实现代码的功能，保证读者能够看懂所有的代码。

扫码看视频

1.3.1 笛卡尔坐标系

笛卡尔坐标系是直角坐标系和斜角坐标系的统称，两条数轴相互垂直的笛卡尔坐标系称为笛卡尔直角坐标系，否则称为笛卡尔斜角坐标系。

1) 2D 笛卡尔坐标系

2D 笛卡尔坐标系是指处于同一个平面的坐标系。每个 2D 笛卡尔坐标系都有一个特殊的点，称为原点，是坐标系的中心；都有两条过原点的直线向两边无限延伸，称为“轴”。

在 2D 笛卡尔坐标系中，习惯将水平的轴称为 x 轴，向右为 x 轴的正方向。将垂直的轴称为 y 轴，向上为 y 轴正方向。开发者可以根据自己的需要来指定坐标轴的指向，也可以指定轴的正方向。

在 2D 笛卡尔坐标系中有 8 种可能的轴的指向，如图 1-1 所示。无论如何选择 x 轴和 y 轴的方向，都能通过旋转使得 x 轴向右为正，y 轴向上为正，因此，所有的 2D 坐标系都是等价的。

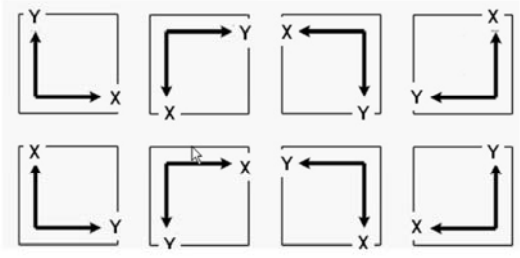


图 1-1　2D 笛卡尔坐标系中有 8 种可能的轴的指向

为了在笛卡尔坐标系中定位点，引入了笛卡尔坐标的概念。在 2D 笛卡尔坐标系中，(x, y)可以定位一个点，坐标的每个分量都表明了该点与原点之间的距离和方位：每个分量都是到相应轴的有符号距离(“有符号距离”指的是在某个方向上距离为正，在相反方向上距离为负)。

2)　3D 笛卡尔坐标系

为了表示三维坐标系，在笛卡尔坐标系中引入第三个轴——z 轴。在一般情况下，这三个轴相互垂直，即每个轴垂直于其他两个轴。在 2D 笛卡尔坐标系中，通常设置 x 轴向右为正、y 轴向上为正为标准形式，但是在 3D 笛卡尔坐标系中没有标准形式。在 3D 笛卡尔坐标系中定位一个点需要 3 个数：x、y 和 z，分别代表该点到 yz、xz 和 xy 平面的有符号距离，如图 1-2 所示。

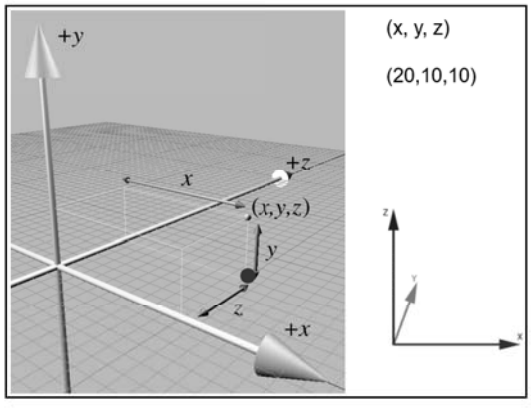


图 1-2　3D 笛卡尔坐标系中的定位点

1.3.2 实例介绍

黑白棋是一款在棋盘上玩的游戏，是用带有 x 和 y 轴坐标的 2D 笛卡尔坐标系实现。有一个 8×8 的游戏板，一方的棋子是黑色，另一方的棋子是白色(在游戏中分别使用 O 和 X 来代替这两种颜色)。开始的时候，棋盘界面如图 1-3 所示。

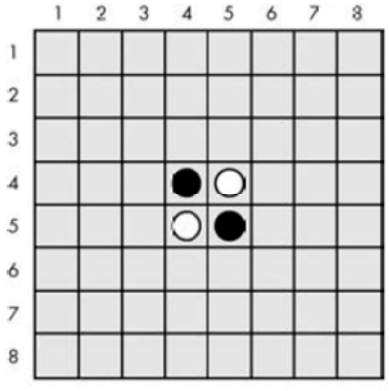


图 1-3 棋盘初始界面

1.3.3 具体实现

实例 1-3：黑白棋游戏

实例文件 Reversi.py 的主要实现代码如下所示。

```
import random
import sys
WIDTH = 8                                  #设置棋盘的宽度是 8 个单元格
HEIGHT = 8                                 #设置棋盘的高度是 8 个单元格
def drawBoard(board):
    print('  12345678')
    print(' +--------+')
    for y in range(HEIGHT):
        print('%s|' % (y+1), end='')
        for x in range(WIDTH):
            print(board[x][y], end='')
        print('|%s' % (y+1))
    print(' +--------+')
    print('  12345678')
```

```
def getNewBoard():
    board = []
    for i in range(WIDTH):
        board.append([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '])
    return board

def isValidMove(board, tile, xstart, ystart):
    #如果玩家在空间 x 上移动，则 y 无效，返回 False。 如果它是一个有效的移动，
    #则返回一个空格列表，如果玩家在这里移动的话，它们会变成玩家的列表
    if board[xstart][ystart] != ' ' or not isOnBoard(xstart, ystart):
        return False

    if tile == 'X':
        otherTile = 'O'
    else:
        otherTile = 'X'

    tilesToFlip = []
    for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1],
        [-1, -1], [-1, 0], [-1, 1]]:
        x, y = xstart, ystart
        x += xdirection # First step in the x direction
        y += ydirection # First step in the y direction
        while isOnBoard(x, y) and board[x][y] == otherTile:
            #继续在 XY 方向前进
            x += xdirection
            y += ydirection
            if isOnBoard(x, y) and board[x][y] == tile:
                #有一些东西翻转过来。沿着相反的方向走，直到我们到达原始空间，注意沿途所有的瓦片
                while True:
                    x -= xdirection
                    y -= ydirection
                    if x == xstart and y == ystart:
                        break
                    tilesToFlip.append([x, y])

    if len(tilesToFlip) == 0: #如果没有翻转瓦片，这不是有效的移动
        return False
    return tilesToFlip

def isOnBoard(x, y):
    #如果坐标位于板上，则返回 True
    return x >= 0 and x <= WIDTH - 1 and y >= 0 and y <= HEIGHT - 1
```

```
def getBoardWithValidMoves(board, tile):
    #返回一个新的棋盘，标明玩家可以做出的有效动作
    boardCopy = getBoardCopy(board)

    for x, y in getValidMoves(boardCopy, tile):
        boardCopy[x][y] = '.'
    return boardCopy

def getValidMoves(board, tile):
    #返回给定板上给定玩家的有效移动列表[x，y]
    validMoves = []
    for x in range(WIDTH):
        for y in range(HEIGHT):
            if isValidMove(board, tile, x, y) != False:
                validMoves.append([x, y])
    return validMoves

def getScoreOfBoard(board):
    #通过计算瓦片来确定分数。获取棋盘上黑白双方的棋子数
    xscore = 0
    oscore = 0
    for x in range(WIDTH):
        for y in range(HEIGHT):
            if board[x][y] == 'X':
                xscore += 1
            if board[x][y] == 'O':
                oscore += 1
    return {'X':xscore, 'O':oscore}

def enterPlayerTile():
    #让玩家输入他们想要的瓦片
    #返回一个列表，玩家的瓦片作为第一个项目，计算机的瓦片作为第二个
    tile = ''
    while not (tile == 'X' or tile == 'O'):
        print('Do you want to be X or O?')
        tile = input().upper()
    #列表中的第一个元素是玩家的棋子，第二个元素是计算机的棋子
    if tile == 'X':
        return ['X', 'O']
    else:
        return ['O', 'X']

def whoGoesFirst():
    #随机选择谁先走
    if random.randint(0, 1) == 0:
```

```
        return 'computer'
    else:
        return 'player'

def makeMove(board, tile, xstart, ystart):
    #把棋子放在(Xstart,YStart)的棋盘上，然后翻转对手的任何棋子
    #如果这是无效移动，就返回 False；如果有效，则返回 True
    tilesToFlip = isValidMove(board, tile, xstart, ystart)
    if tilesToFlip == False:
        return False
    board[xstart][ystart] = tile
    for x, y in tilesToFlip:
        board[x][y] = tile
    return True

def getBoardCopy(board):
    #复制一个棋盘给计算机落子使用
    boardCopy = getNewBoard()
    for x in range(WIDTH):
        for y in range(HEIGHT):
            boardCopy[x][y] = board[x][y]
    return boardCopy

def isOnCorner(x, y):
    #如果位置位于四个角之一，则返回 True
    return (x == 0 or x == WIDTH - 1) and (y == 0 or y == HEIGHT - 1)

def getPlayerMove(board, playerTile):
    #让玩家设置移动
    #返回移动坐标[X,Y](或返回字符串'hints'或'quit')
    DIGITS1TO8 = '1 2 3 4 5 6 7 8'.split()
    while True:
        print('Enter your move, "quit" to end the game, or "hints" to toggle hints.')
        move = input().lower()
        if move == 'quit' or move == 'hints':
            return move

        if len(move) == 2 and move[0] in DIGITS1TO8 and move[1] in DIGITS1TO8:
            x = int(move[0]) - 1
            y = int(move[1]) - 1
            if isValidMove(board, playerTile, x, y) == False:
                continue
            else:
                break
        else:
```

```
            print('That is not a valid move. Enter the column (1-8) and then the row
               (1-8).')
            print('For example, 81 will move on the top-right corner.')

    return [x, y]

def getComputerMove(board, computerTile):
    #给定一块棋盘面板和计算机的棋子，确定移动的位置，并将其作为[X,Y]列表返回
    possibleMoves = getValidMoves(board, computerTile)
    random.shuffle(possibleMoves)            #随机移动
    #如果有棋子的话，一定要去拐角处
    for x, y in possibleMoves:
        if isOnCorner(x, y):
            return [x, y]
    #找到可能得分最高的动作
    bestScore = -1
    for x, y in possibleMoves:
        boardCopy = getBoardCopy(board)
        makeMove(boardCopy, computerTile, x, y)
        score = getScoreOfBoard(boardCopy)[computerTile]
        if score > bestScore:
            bestMove = [x, y]
            bestScore = score
    return bestMove

def printScore(board, playerTile, computerTile):
    scores = getScoreOfBoard(board)
    print('You: %s points. Computer: %s points.' % (scores[playerTile],
        scores[computerTile]))
```

在上述代码中，虽然函数 drawBoard()会在屏幕上显示一个游戏板数据结构，但是还需要一种创建这些游戏板数据结构的方式。函数 getNewBoard()创建了一个新的游戏板数据结构，并返回 8 个列表中的一列，其中的每一个列表包含了 8 个空格的字符串，表示没有落子的一个空白游戏板。

当给定了一个游戏板数据结构、玩家的棋子以及玩家落子的 x、y 坐标后，如果 Reversi 游戏规则允许在该坐标上落子，isValidMove()函数应该返回 True，否则返回 False。对于一次有效的移动，它必须位于游戏板之上，并且还要至少能够反转对手的一个棋子。这个函数使用了游戏板上的几个 x 坐标和 y 坐标，因此，变量 xstart 和变量 ystart 记录了最初的 x 坐标和 y 坐标。

函数 getScoreOfBoard()使用嵌套 for 循环检查游戏板上的所有 64 个格子(8 行乘以 8 列，一共是 64 个格子)，并且看看哪些棋子在上面(如果有棋子的话)。

执行程序后会输出以下内容：

```
Welcome to Reversegam!
Do you want to be X or O?
X
The computer will go first.
  12345678
 +--------+
1|        |1
2|        |2
3|        |3
4|   XO   |4
5|   OX   |5
6|        |6
7|        |7
8|        |8
 +--------+
  12345678
You: 2 points. Computer: 2 points.
Press Enter to see the computer's move.
  12345678
 +--------+
1|        |1
2|        |2
3|        |3
4|  OOO   |4
5|   OX   |5
6|        |6
7|        |7
8|        |8
 +--------+
  12345678
You: 1 points. Computer: 4 points.
Enter your move, "quit" to end the game, or "hints" to toggle hints.
```

在本项目中，游戏板数据结构只是一个 Python 列表值，我们需要一种更好的方法在屏幕上展示它。函数 drawBoard()可根据 board 中的数据结构来打印当前游戏板。

函数 isOnBoard()用于检查 x 坐标和 y 坐标是否在游戏板之上，以及位置是否为空。通过这个函数确保了坐标 x 和坐标 y 都在 0 和游戏板的 WIDTH 或 HEIGHT-1 之间。

函数 getValidMoves()返回了包含两元素的一个列表。这个列表中保存了给定的 tile 可以在参数 board 中进行的所有有效移动的 x、y 坐标。

在函数 getScoreOfBoard()中，对于每个 X 棋子，通过代码“xscore += 1”将 xscore 值加 1。对于每个 O 棋子，通过代码“oscore += 1”将 oscore 值加 1。然后，该函数将 xscore 和 oscore 的值返回到一个字典中。

1.4 益智类游戏：俄罗斯方块

俄罗斯方块是一款风靡全球的电视游戏机和掌上游戏机游戏，这款游戏最初是由阿列克谢·帕基特诺夫(Alexey Pazhitnov)制作的，它看似简单却变化无穷，令人着迷。本节将介绍使用 Python+Pygame 开发一个简单俄罗斯方块游戏的方法，并详细介绍其具体的实现流程。

扫码看视频

1.4.1 规划需要的图形

在本游戏项目中，主要用到了以下 4 类图形。

- 边框：由 10×20 个空格组成，方块就落在这里面。
- 盒子：组成方块的小方块，是方块的基本单元。
- 方块：从边框顶掉下的东西，游戏者可以翻转和改变其位置。每个方块由 4 个盒子组成。
- 形状：不同类型的方块，这里形状的名字分别是 T、S、Z、J、L、I 和 O。在本实例中预先规划了如图 1-4 所示的 7 种形状。

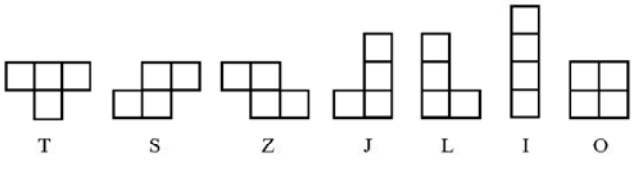


图 1-4 7 种形状的方块

除了准备上述 4 类图形外，还需要用到以下两个术语。

(1) 模板：用一个列表存放形状被翻转后的所有可能样式。所有可能的样式全部存放在模板变量里面，变量名字可以是 S_SHAPE_TEMPLATE 或 J_SHAPE_TEMPLATE 等。

(2) 着陆(碰撞)：当一个方块到达边框的底部或接触到其他盒子时，我们称这个方块着陆了，此时另一个新的方块会出现并开始下落。

1.4.2 具体实现

实例 1-4：俄罗斯方块游戏

本俄罗斯方块游戏的实现文件是 els.py，具体实现流程如下所示。

(1) 首先使用 import 语句引入 Python 的内置库和游戏库 Pygame，然后定义一些项目

用到的变量，并进行初始化，具体实现代码如下所示。

```
import random, time, pygame, sys
from pygame.locals import *
FPS = 25
WINDOWWIDTH = 640
WINDOWHEIGHT = 480
BOXSIZE = 20
BOARDWIDTH = 10
BOARDHEIGHT = 20
BLANK = '.'
MOVESIDEWAYSFREQ = 0.15
MOVEDOWNFREQ = 0.1
XMARGIN = int((WINDOWWIDTH - BOARDWIDTH * BOXSIZE) / 2)
TOPMARGIN = WINDOWHEIGHT - (BOARDHEIGHT * BOXSIZE) - 5
#RGB
WHITE    = (255, 255, 255)
GRAY     = (185, 185, 185)
BLACK    = ( 0,  0,  0)
RED      = (155,  0,  0)
LIGHTRED  = (175, 20, 20)
GREEN    = ( 0, 155,  0)
LIGHTGREEN = ( 20, 175, 20)
BLUE     = ( 0,  0, 155)
LIGHTBLUE  = ( 20, 20, 175)
YELLOW   = (155, 155,  0)
LIGHTYELLOW = (175, 175, 20)
BORDERCOLOR = BLUE
BGCOLOR = BLACK
TEXTCOLOR = WHITE
TEXTSHADOWCOLOR = GRAY
COLORS   = (  BLUE,   GREEN,   RED,   YELLOW)
LIGHTCOLORS = (LIGHTBLUE, LIGHTGREEN, LIGHTRED, LIGHTYELLOW)
assert len(COLORS) == len(LIGHTCOLORS) # each color must have light color
TEMPLATEWIDTH = 5
TEMPLATEHEIGHT = 5
```

在上述实例代码中，BOXSIZE、BOARDWIDTH 和 BOARDHEIGHT 的功能是建立游戏与屏幕像素点的联系。其中下面的两个语句：

```
MOVESIDEWAYSFREQ = 0.15
MOVEDOWNFREQ = 0.1
```

表示每当游戏玩家按键盘上的方向左键或右键，下降的方块相应地向左或向右移一个格子。另外，游戏玩家也可以一直按方向左键或右键让方块保持移动。MOVESIDEWAYSFREQ 表示如果一直按方向左键或右键，那么方块会每 0.15 秒移动一次。而 MOVEDOWNFREQ 与

MOVESIDEWAYSFREQ 一样，表示当游戏玩家一直按方向下键时方块下落的频率。

下面的两个变量，表示游戏界面的高度和宽度：

```
XMARGIN = int((WINDOWWIDTH - BOARDWIDTH * BOXSIZE) / 2)
TOPMARGIN = WINDOWHEIGHT - (BOARDHEIGHT * BOXSIZE) - 5
```

要想理解上述两个变量的含义，可查看图 1-5 所示。

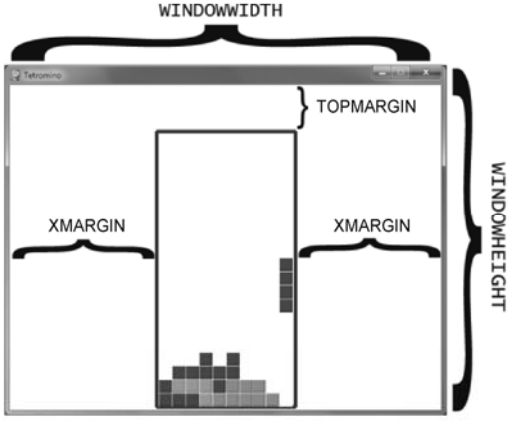


图 1-5 游戏界面

剩余的变量都是和颜色定义相关的，读者需要注意的是，COLORS 和 LIGHTCOLORS 这两个变量：COLORS 是组成方块的小方块的颜色；而 LIGHTCOLORS 是围绕在小方块周围的颜色，是为了突出轮廓而设计的。

(2) 定义方块形状，其中定义了 T、S、Z、J、L、I 和 O 共计 7 种方块形状，具体实现代码如下所示。

```
S_SHAPE_TEMPLATE = [['.....',
                     '.....',
                     '..OO.',
                     '.OO..',
                     '.....'],
                    ['.....',
                     '..O..',
                     '..OO.',
                     '...O.',
                     '.....']]

Z_SHAPE_TEMPLATE = [['.....',
                     '.....',
                     '.OO..',
                     '..OO.',
```

```
                    '.....'],
                    ['.....',
                    '..O..',
                    '.OO..',
                    '.O...',
                    '.....']]
#省略部分代码
```

在定义每个方块时，必须知道每个类型的方块有多少种形状。上述代码在列表中嵌入了含有字符串的列表来构成一个模板，一个方块类型的模板包含了这个方块可能变换的所有形状。比如“I”形状的模板代码如下所示。

```
I_SHAPE_TEMPLATE = [['..O..',
                    '..O..',
                    '..O..',
                    '..O..',
                    '.....'],
                    ['.....',
                    '.....',
                    'OOOO.',
                    '.....',
                    '.....']]
```

在定义每种方块形状的模板之前，可用以下两行代码表示组成形状的行和列：

```
TEMPLATEWIDTH = 5
TEMPLATEHEIGHT = 5
```

方块形状的行和列的具体结构如图 1-6 所示。

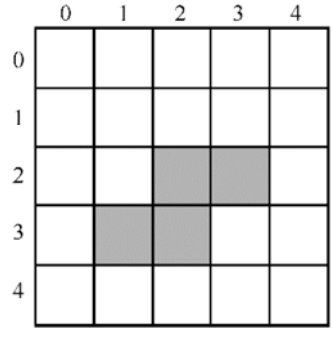


图 1-6　方块形状的行和列的具体结构

(3)　定义字典变量 PIECES 来存储所有的不同形状的模板，因为每个模板中包含一个块的所有变换形状，那就意味着 PIECES 变量包含了每个类型的方块和所有的变换形状，这就是存放游戏中用到的形状的数据结构。具体实现代码如下所示。

```
PIECES = {'S': S_SHAPE_TEMPLATE,
          'Z': Z_SHAPE_TEMPLATE,
          'J': J_SHAPE_TEMPLATE,
          'L': L_SHAPE_TEMPLATE,
          'I': I_SHAPE_TEMPLATE,
          'O': O_SHAPE_TEMPLATE,
          'T': T_SHAPE_TEMPLATE}
```

(4) 编写主函数 main()，其主要功能是创建一些全局变量和在游戏开始之前显示一个开始画面。具体实现代码如下所示。

```
def main():
  global FPSCLOCK, DISPLAYSURF, BASICFONT, BIGFONT
  pygame.init()
  FPSCLOCK = pygame.time.Clock()
  DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
  BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
  BIGFONT = pygame.font.Font('freesansbold.ttf', 100)
  pygame.display.set_caption('Tetromino')

  while True:
    runGame()
    showTextScreen('Game Over')
```

上述代码中的 runGame()函数是核心。在循环中首先简单地随机决定采用某个背景音乐，然后调用 runGame()函数运行游戏。当游戏失败，runGame()就会返回 main()函数，这时会停止背景音乐和显示游戏失败的画面。当游戏玩家按下一个键时，函数 showTextScreen()会显示游戏失败，并再次开始下一次游戏。

(5) 编写函数 runGame()启动运行游戏，具体实现流程如下所示。

① 在游戏开始前设置在运行过程中用到的几个变量，具体实现代码如下所示。

```
def runGame():
  #setup variables for the start of the game
  board = getBlankBoard()
  lastMoveDownTime = time.time()
  lastMoveSidewaysTime = time.time()
  lastFallTime = time.time()
  movingDown = False # note: there is no movingUp variable
  movingLeft = False
  movingRight = False
  score = 0
  level, fallFreq = calculateLevelAndFallFreq(score)

  fallingPiece = getNewPiece()
  nextPiece = getNewPiece()
```

② 在游戏开始和方块掉落之前需要初始化一些和游戏开始相关的变量。变量 fallingPiece 被赋值成当前掉落的变量，变量 nextPiece 被赋值成游戏玩家可以在屏幕 NEXT 区域看见的下一个方块。具体实现代码如下所示。

```
while True: # game loop
   if fallingPiece == None:
      # No fallingpiece in play, so start a new piece at the top
      fallingPiece = nextPiece
      nextPiece = getNewPiece()
      lastFallTime = time.time()  #reset lastFallTime

      if not isValidPosition(board, fallingPiece):
           return  #can't fit a new piece on the board, so game over

   checkForQuit()
```

上述代码包含了当方块往底部掉落时的所有代码。变量 fallingPiece 在方块着陆后被设置成 None。这意味着 nextPiece 变量中的下一个方块应该被赋值给 fallingPiece 变量，然后一个随机的方块又会被赋值给 nextPiece 变量。变量 lastFallTime 被赋值成当前时间，这样就可以通过变量 fallFreq 控制方块下落的频率。来自函数 getNewPiece()的方块只有一部分被放置在方框区域中，但如果这是一个非法的位置，比如此时游戏方框已经被填满(isValidPosition()函数返回 False，那么就知道方框已经满了)，这说明游戏玩家输掉了游戏。当这些情况发生时，runGame()函数就会返回。

③ 实现暂停游戏。如果游戏玩家按 P 键，游戏就会暂停。我们应该隐藏游戏界面以防止游戏者作弊(否则游戏者会看着画面思考怎么处理方块)，用 DISPLAYSURF.fill(BGCOLOR)就可以实现这个效果。具体实现代码如下所示。

```
      for event in pygame.event.get(): # event handling loop
         if event.type == KEYUP:
            if (event.key == K_p):
               #Pausing the game
               DISPLAYSURF.fill(BGCOLOR)
               #pygame.mixer.music.stop()
               showTextScreen('Paused')
               lastFallTime = time.time()
               lastMoveDownTime = time.time()
               lastMoveSidewaysTime = time.time()
```

④ 按下方向键或 A、D、S 键会把 movingLeft、movingRight 和 movingDown 变量设置为 False，这说明游戏玩家不再想要在此方向上移动方块。后面的代码会基于 moving 变量处理一些事情。

⑤ 如果上方向键或 W 键被按下，那么就会翻转方块并将存储在 fallingPiece 字典中的

rotation 值加 1。但是，当 rotation 值大于所有当前类型方块的形状的数目(此变量存储在 len(PIECES[fallingPiece['shape']])变量中)时，那么它将翻转到最初的形状。

⑥ 如果向下的方向键被按下，游戏玩家此时希望方块下降得比平常快。fallingPiece['y'] += 1 使方块下落一个格子(前提是这是一个有效的下落)，movingDown 被设置为 True，lastMoveDownTime 变量也被设置为当前时间，这个变量以后将被用来检查当向下的方向键一直按下时，保证方块以一个比平常快的速率下降。

⑦ 游戏玩家按下空格键，方块将会迅速下落至着陆。程序首先需要找出它着陆需要下降多少个格子。其中，有关 moving 的三个变量都要被设置为 False(保证程序知道游戏玩家已经停止了按下所有的方向键)。

⑧ 如果用户按住按键超过 0.15 秒，那么表达式“(movingLeft or movingRight)and time.time() - lastMoveSidewaysTime > MOVESIDEWAYSFREQ”返回 True，这样的话就可以将方块向左或向右移动一个格子。最后更新 lastMoveSidewaysTime 变量。

⑨ 在屏幕中绘制前面所有定义的图形，具体实现代码如下所示。

```
    DISPLAYSURF.fill(BGCOLOR)
    drawBoard(board)
    drawStatus(score, level)
    drawNextPiece(nextPiece)
    if fallingPiece != None:
        drawPiece(fallingPiece)
    pygame.display.update()
    FPSCLOCK.tick(FPS)
```

至此，整个实例介绍完毕，执行效果如图 1-7 所示。

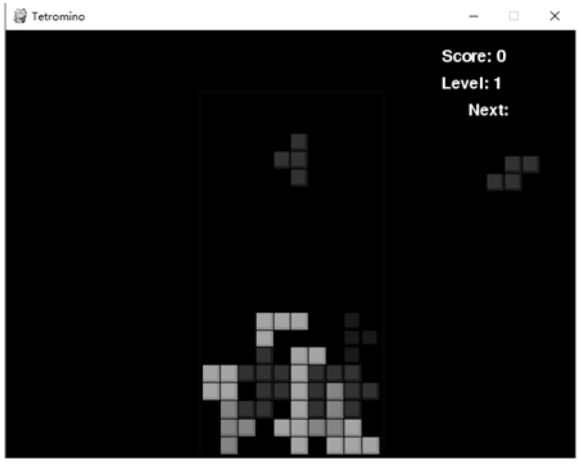


图 1-7 执行效果

第 2 章

Web 网站开发实战

Web 应用程序是一种可以通过 Web 访问的应用程序，其最大优势是用户容易访问，只需要浏览器即可，无须再安装其他软件。例如现实中我们经常浏览的新浪、搜狐、天猫等网站都是 Web 程序。本章将详细讲解使用 Python 语言开发 Web 网站的知识。

2.1 会员登录验证系统

会员登录验证系统在现实生活中比较常见，例如，输入个人用户名和密码登录天猫商城或京东商城，输入微信账号和密码登录自己的微信，这些都属于会员登录验证系统的典型应用。本节将详细讲解使用 Python 开发会员登录验证系统的方法。

扫码看视频

2.1.1 简易用户登录验证系统

在下面的实例代码中，演示了使用 Django 框架开发一个简易用户登录验证系统的过程。

实例 2-1：简易用户登录验证系统

(1) 新建一个名称为 biaodan1 的项目，然后进入 biaodan1 文件夹新建一个名为 people 的 App。

```
django-admin startproject biaodan1
cd biaodan1
python manage.py startapp people
```

(2) 在设置文件 settings.py 的 INSTALLED_APPS 定义中将上面创建的 people 添加进去：

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'people',
]
```

(3) 编写视图文件 views.py，具体实现代码如下所示。

```
from people.models import User
from functools import wraps
#说明：这个装饰器的作用，就是在每个视图函数被调用时，都验证下有没有登录，
#如果登录过，可以执行新的视图函数，
#如果没有登录，则自动跳转到登录页面
def check login(f):
    @wraps(f)
    def inner(request,*arg,**kwargs):
```

```
        if request.session.get('is_login')=='1':
            return f(request,*arg,**kwargs)
        else:
            return redirect('/login/')
    return inner

def login(request):
    #如果是 POST 请求，说明是单击登录按钮 FORM 表单才跳转到此的，那么就要验证密码，并保存 session
    if request.method=="POST":
        username=request.POST.get('username')
        password=request.POST.get('password')

        user=User.objects.filter(username=username,password=password)
        print(user)
        if user:
            #登录成功
            #1.生成特殊字符串
            #2.这个字符串当成 key，此 key 在数据库的 session 表(在数据库中一个表名是 session
            #的表)中对应一个 value
            #3.在响应中用 cookie 保存这个 key(即向浏览器写一个 cookie，此 cookie 的值即是这个
            #key 特殊字符)
            request.session['is_login']='1'  #这个 session 用于后面访问每个页面(即调用
            #每个视图函数时要用到，判断是否已经登录)
            request.session['username']=username  #这个要存储的 session 用于后面，每个
            #页面上要显示出来登录状态的用户名。
            #说明：如果需要在页面上显示出来的用户信息太多(有时还有积分、姓名、年龄等信息)，我们
            #可以只用 session 保存 user_id
            request.session['user_id']=user[0].id
            return redirect('/index/')
    #如果是 GET 请求，就说明用户刚开始登录，是使用 URL 直接进入登录页面的
    return render(request,'login.html')

@check_login
def index(request):
    user_id1=request.session.get('user_id')
    # 使用 user_id 去数据库中找到对应的 user 信息
    userobj=User.objects.filter(id=user_id1)
    print(userobj)
    if userobj:
        return render(request,'index.html',{"user":userobj[0]})
    else:
        return render(request,'index.html',{'user','匿名用户'})
```

(4) 编写模型文件 models.py，具体实现代码如下所示。

```
from django.db import models
class User(models.Model):
    username=models.CharField(max_length=16)
```

```
    password=models.CharField(max_length=32)
```

(5) 编写 URL 导航文件 urls.py，具体实现代码如下所示。

```
from django.urls import path
from django.contrib import admin
from people import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('login/', views.login),
    path('index/', views.index),
]
```

(6) 在 people 目录下创建 templates 子目录，在里面新建两个模板文件。第一个模板文件 index.html 的代码如下所示。

```
 <body>
   <h1>这是一个 index 页面</h1>
   <p>欢迎：{{user.username}}--{{user.password}}</p>
 </body>
```

第二个模板文件 login.html 的代码如下所示。

```
<body>
<h1>欢迎登录！</h1>
<form action="/login/" method="post">
    {% csrf_token %}
    <p>
        用户名：
        <input type="text" name="username">
    </p>
    <p>
        密码：
        <input type="text" name="password">
    </p>
    <p>
        <input type="submit" value="登录">
    </p>
    <hr>
</form>
</body>
```

(7) 将控制命令定位到项目根目录 biaodan1，通过如下命令根据模型文件 models.py 创建数据库表。

```
python manage.py makemigrations
python manage.py migrate
```

(8) 创建一个合法用户数据。下面的命令创建的用户名是 admin，密码是“123”。

```
python manage.py shell

>>> from people.models import User
>>> User.objects.create(username="admin", password="123")
```

开始测试程序，如果未登录时输入 http://localhost:8000/index/后，会自动跳转到 login 登录表单页面，如图 2-1 所示。输入用户名 admin 和密码 123 后，会进入登录成功页面 index.html，在页面中显示用户名和密码，如图 2-2 所示。

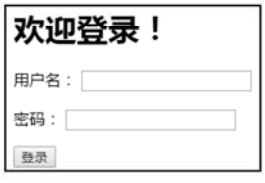


图 2-1 登录表单页面

图 2-2 登录成功页面

在浏览器的 Cookie 页面中会显示用 Session 保存的登录信息，如图 2-3 所示。

图 2-3 浏览器中存储的信息

2.1.2 使用模块 auth 实现登录验证系统

模块 auth 是 Django 框架提供的标准权限管理系统，可以实现用户身份认证、用户组和权限管理。模块 auth 可以和后台管理模块 admin 配合使用，快速建立 Web 的管理系统。在创建一个 Django 工程后，会默认使用模块 auth，在 INSTALLED_APPS 中默认显示模块 auth 对应的选项：django.contrib.auth。

在下面的实例中，演示了使用 Django 中的 auth 模块开发一个简易新闻系统的过程。本实例具有如下功能。

- 前台会员注册：用户可以注册成为系统会员。
- 登录验证：验证会员登录信息是否合法。
- 前台显示新闻信息：包括新闻列表和某条新闻的详情信息。
- 后台管理：管理员可以发布、修改或删除新闻信息。

实例 2-2：使用模块 auth 实现登录验证系统

(1) 通过如下命令新建一个名为 yanzheng 的工程，在 yanzheng 目录中，新建一个名为 blog 的 App 项目。

```
django-admin.py startproject yanzheng
cd yanzheng
python manage.py startapp blog
```

(2) 在文件 settings.py 中，将上面创建的 App 项目名 blog 添加到 INSTALLED_APPS 中。

```
INSTALLED_APPS = [
#省略部分代码
    'blog',
]
```

(3) 在路径导航文件 urls.py 中设置 URL 链接，主要实现代码如下所示。

```
urlpatterns = [
    path(r'admin/', admin.site.urls),
    path(r'', views.index),
    path(r'regist/', views.regist),
    path(r'login/', views.login),
    path(r'logout/', views.logout),
    path(r'article/', views.article),
    path(r'(?P<id>\d+)/', views.detail, name='detail'),
]
```

(4) 在视图文件 views.py 中分别实现各个页面的视图。

- index：进入系统主页。
- regist：实现会员注册视图，将表单中的注册数据添加到系统数据库。
- login：实现登录验证视图，对登录表单中的数据进行验证。
- logout：实现用户退出视图。
- article：获取系统数据库中的表 article 的信息，将 article 标题以列表形式显示出来。
- detail：获取并显示某条 article 的详细信息。

文件 views.py 的主要实现代码如下所示。

```
class UserForm(forms.Form):
    username = forms.CharField(label='用户名', max_length=100)
    password = forms.CharField(label='密 码', widget=forms.PasswordInput())

def index(request):
    return render(request, 'index.html')

def regist(request):
    if request.method == 'POST':
        uf = UserForm(request.POST)  #包含用户名和密码
        if uf.is_valid():
            #获取表单数据
            #cleaned_data 类型是字典，里面是提交成功后的信息
            username = uf.cleaned_data['username']
            password = uf.cleaned_data['password']
            #添加到数据库
            #registAdd = User.objects.get_or_create(username=username,password=password)
            registAdd = User.objects.create_user(username=username, password=password)
            #print registAdd
            if registAdd == False:
                return render(request, 'share1.html', {'registAdd': registAdd,
                    'username': username})

            else:
                #return HttpResponse('OK')
                return render(request, 'share1.html', {'registAdd': registAdd})
                #return render_to_response('share.html',{'registAdd':registAdd},
                #context_instance = RequestContext(request))
    else:
        #如果不是 post 提交数据，就不传参数创建对象，并将对象返回给前台，直接生成 input 标记，
        #内容为空
        uf = UserForm()
    #return render_to_response('regist.html',{'uf':uf},context_instance =
    #RequestContext(request))
    return render(request, 'regist1.html', {'uf': uf})

def login(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        print(username, password)
        re = auth.authenticate(username=username, password=password)  #用户认证
        if re is not None:  #如果数据库里有记录(即与数据库里的数据相匹配或者对应或者符合)
            auth.login(request, re)  #登录成功
            #跳转--redirect 指从一个旧的 url 转到一个新的 url
            return redirect('/', {'user': re})
```

```
        else:  #数据库里不存在与之对应的数据
            return render(request, 'login.html', {'login_error': '用户名或密码错误'})
            #注册失败
    return render(request, 'login.html')

def logout(request):
    auth.logout(request)
    return render(request, 'index.html')

def article(request):
    article_list = Article.objects.all()
    #每个 Model 都有一个默认的 manager 实例，名为 objects。QuerySet 有两种来源：通过 manager
    #和 QuerySet 的方法得到。mananger 的方法和 QuerySet 的方法大部分同名同含义，如 filter()、
    #update()等，但也有些不同，如 manager 有 create()、get_or_create()，而 QuerySet
    #有 delete()等
    return render(request, 'article.html', {'article_list': article_list})

def detail(request, id):
    #print id
    try:
        article = Article.objects.get(id=id)
        #print type(article)
    except Article.DoesNotExist:
        raise Http404
    return render(request, 'detail.html', locals())
```

(5) 在模型文件 models.py 中创建数据库表 Article，因为本实例并没有特意创建会员信息表，而是直接使用了 Django 自带的 user 表，所以在文件 models.py 中没有创建表 user。文件 models.py 的主要实现代码如下所示。

```
class Article(models.Model):
    title = models.CharField(u'标题', max_length=256)
    content = models.TextField(u'内容')
    pub_date = models.DateTimeField(u'发表时间', auto_now_add=True, editable=True)
    update_time = models.DateTimeField(u'更新时间', auto_now=True, null=True)

    def __unicode__(self):  #在 Python3 中用 __str__ 代替 __unicode__
        return self.title
```

根据上面的模型，通过如下命令创建数据库。

```
python manage.py makemigrations
python manage.py migrate
```

(6) 在文件 admin.py 中设置后台显示模块 ArticleAdmin，只有这样才能在后台显示新闻管理模块。文件 admin.py 的主要实现代码如下所示。

```
from blog.models import Article
class ArticleAdmin(admin.ModelAdmin):
   list_display = ('title', 'title','pub_date', 'update_time',)
admin.site.register(Article, ArticleAdmin)
```

(7) 在模板文件 login.html 中实现用户登录表单效果。一定要在<form>标记后面添加{% csrf_token %}，否则不会通过 Session 验证。文件 login.html 的主要实现代码如下所示。

```
      <form action="/login/" method="POST">{% csrf_token %}
         <h2>请登录</h2>
         <input type="text" name="username" />
         <input type="password" name="password" />
         <button type="submit">登录</button>
         <p style="color: red">{{ login_error }}</p>
      </form>
```

在模板文件 regist1.html 中实现用户注册界面效果，一定要在<form>标记后面添加{% csrf_token %}，否则不能通过 Session 验证。文件 regist1.html 的主要实现代码如下所示。

```
<form method="POST" enctype="multipart/form-data">
{% csrf_token %}
      {{uf.as_p}}
      <input type="submit" value="OK" />
   </form>
{#    <a href="http://127.0.0.1:8000/login">注册</a>#}
{% endblock %}
```

前台新闻列表 http://127.0.0.1:8000/article/的执行效果如图 2-4 所示。

新用户注册界面的执行效果如图 2-5 所示。

图 2-4 前台新闻列表界面

图 2-5 新用户注册界面

后台新闻管理界面效果如图 2-6 所示。

图 2-6　后台新闻管理界面

2.1.3　使用百度账户实现用户登录系统

在实际应用中，很多 Web 都提供了第三方账户登录功能，例如可以使用 QQ 账号、百度账号等登录论坛。在下面的实例代码中，演示了在 django-allauth 中使用百度账户的过程。

实例 2-3：在 django-allauth 中使用百度账户实现用户登录系统

(1) 通过如下命令创建一个名为 myproject 的工程，在里面新建一个名为 yanzheng 的 App。

```
django-admin.py startproject myproject
cd myproject
python manage.py startapp yanzheng
```

(2) 在配置文件 settings.py 中首先将 yanzheng 添加到 INSTALLED_APPS，然后再将 allauth 框架用到的模块添加到 INSTALLED_APPS，最后将第三方账户百度功能添加到 INSTALLED_APPS。

```
INSTALLED_APPS = [
#省略部分代码
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.github',
    'yanzheng',
    'allauth.socialaccount.providers.baidu',
]
```

(3) 在配置文件 settings.py 中设置服务器邮箱，可以用 QQ 邮箱、126、网易等第三方邮箱账号。

```
STATIC_URL = '/static/'
#邮箱设定
EMAIL_HOST = 'smtp.qq.com'
```

```
EMAIL_PORT = 25
EMAIL_HOST_USER = '729017304@qq.com'          #QQ 账号和授权码
EMAIL_HOST_PASSWORD = ''                      #密码
EMAIL_USE_TLS = True                          #这里必须是 True，否则发送不成功
EMAIL_FROM = '729017304@qq.com'               #QQ 账号
DEFAULT_FROM_EMAIL = '729017304@qq.com'
```

(4) 编写 URL 导航文件 myproject/urls.py，主要实现代码如下所示。

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('allauth.urls')),
    path('accounts/', include('myaccount.urls')),
]
```

(5) 编写模型文件 models.py，在 Django 自带的 User 模型的基础上对其进行扩展，创建 UserProfile 模型，在里面添加了 org 和 telephone 两个字段。文件 models.py 的主要实现代码如下所示。

```
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, related_name='profile')
    org = models.CharField(
        'Organization', max_length=128, blank=True)
    telephone = models.CharField(
        'Telephone', max_length=50, blank=True)
    mod_date = models.DateTimeField('Last modified', auto_now=True)
    class Meta:
        verbose_name = 'User Profile'
    def __str__(self):
        return "{}'s profile".format(self.user.username)

    def account_verified(self):
        if self.user.is_authenticated:
            result = EmailAddress.objects.filter(email=self.user.email)
            if len(result):
                return result[0].verified
        return False
```

(6) 编写 URL 路径导航文件 yanzheng/urls.py，设置通过 URL 进入用户登录成功显示的视图 profile，通过 URL 进入用户修改资料的视图 profile_update。

(7) 编写视图文件 views.py，分别创建用户登录成功视图和用户资料修改视图，主要实现代码如下所示。

```
@login_required
def profile(request):
    user = request.user
    return render(request, 'profile.html', {'user': user})
```

```
@login_required
def profile_update(request):
    user = request.user
    user_profile = get_object_or_404(UserProfile, user=user)
    if request.method == "POST":
        form = ProfileForm(request.POST)
        if form.is_valid():
            user.first_name = form.cleaned_data['first_name']
            user.last_name = form.cleaned_data['last_name']
            user.save()
            user_profile.org = form.cleaned_data['org']
            user_profile.telephone = form.cleaned_data['telephone']
            user_profile.save()

            return HttpResponseRedirect(reverse('yanzheng:profile'))
    else:
        default_data = {'first_name': user.first_name, 'last_name': user.last_name,
                        'org': user_profile.org, 'telephone': user_profile.telephone, }
        form = ProfileForm(default_data)
    return render(request, 'profile_update.html', {'form': form, 'user': user})
```

(8) 用户登录后可以通过表单修改自己的资料。在表单文件 forms.py 中创建两个表单，一个用于更新用户资料，另一个用于重写用户登录表单。文件 forms.py 的主要实现代码如下所示。

```
class ProfileForm(forms.Form):
    first_name = forms.CharField(label='First Name', max_length=50, required=False)
    last_name = forms.CharField(label='Last Name', max_length=50, required=False)
    org = forms.CharField(label='Organization', max_length=50, required=False)
    telephone = forms.CharField(label='Telephone', max_length=50, required=False)

class SignupForm(forms.Form):
    def signup(self, request, user):
        user_profile = UserProfile()
        user_profile.user = user
        user.save()
        user_profile.save()
```

(9) 在配置文件 settings.py 中添加如下代码，告诉 django-allauth 使用自定义的登录表单。

```
ACCOUNT_SIGNUP_FORM_CLASS = 'yanzheng.forms.SignupForm'
```

(10) 编写模板文件 profile.html，用户登录成功后通过此页面显示用户的账户信息。

(11) 编写模板文件 profile_update.html，用户登录成功后通过此页面修改自己的账户信息。

(12) 登录百度开发者中心 http://developer.baidu.com/，创建一个项目(名字自取，例如 django)，百度会自动分配 API Key 和 Secret Key，如图 2-7 所示。

图 2-7 项目 django 的 API Key 和 Secret Key

(13) 单击左侧导航栏中的“安全设置”链接，在弹出的页面中设置回调 URL。这样当百度授权登录完成后，可以跳转回自己的网站。本地回调 URL 是 http://127.0.0.1:8000/accounts/baidu/login/callback/，如图 2-8 所示。

图 2-8 “安全设置”页面

(14) 通过如下命令创建一个管理员账户：

```
python manage.py createsuperuser
```

在浏览器中输入 http://127.0.0.1:8000/admin 登录后台管理界面，单击 Social applications 链接，在新界面中单击 ADD SOCIAL APPLICATION 按钮，在弹出的表单界面中输入前面申请到的 API Key 和 Secret Key。其中，Provider 选项的值是 Baidu；Name 选项的值是 django，和百度开发者中心的名字相同；在 Client id 文本框中输入 API Key，在 Secret key

文本框中输入 Secret Key；在 Sites 选项中设置允许的站点，因为我们是本地调试，建议将 http://127.0.0.1:8000 和 http://127.0.0.1 都添加进来。最终界面效果如图 2-9 所示。

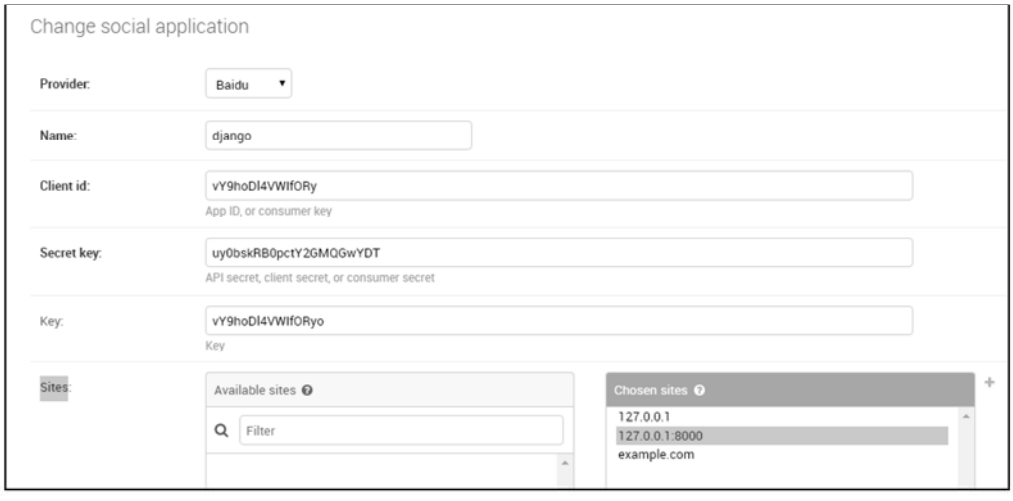


图 2-9　后台设置界面

(15) 输入 http://127.0.0.1:8000/accounts/login/进入登录页面，此时会显示 Baidu 链接，如图 2-10 所示。

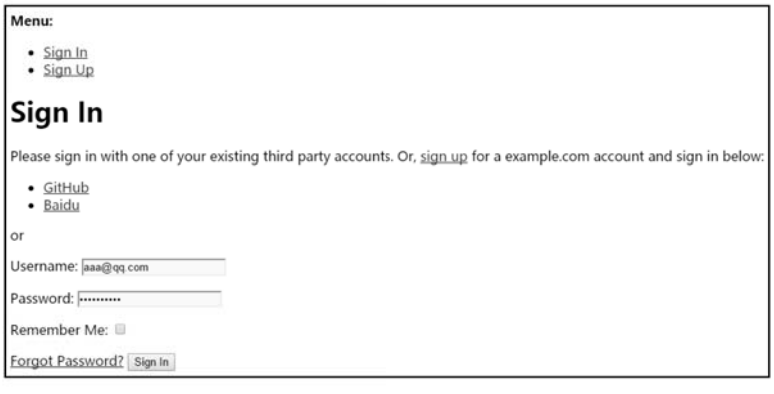


图 2-10　显示 Baidu 链接

单击 Baidu 链接后会弹出百度登录表单页面，可以通过百度账号登录系统，如图 2-11 所示。

另外，在用户注册会员时还提供了邮箱验证功能，系统会向用户的邮箱中发送一封验证邮件，如图 2-12 所示。单击邮件中的链接即可实现用户注册功能。

图 2-11　输入百度账号

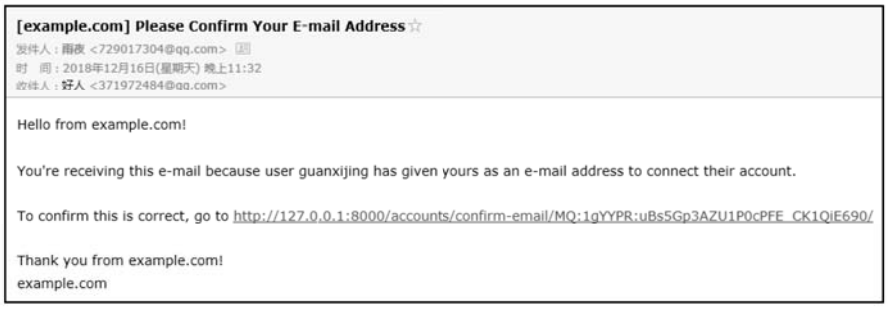


图 2-12　系统发送的验证邮件

2.2　博客发布系统

本节将创建一个完整的博客发布系统，用户通过它可以发布博客信息。本项目仿照 CSDN 的登录验证系统，使用密码签名的方式将用户密码和 Cookie 进行加密，提高了系统的安全性。

扫码看视频

2.2.1　系统设置

在配置文件 settings.py 中首先设置 SECRET_KEY，然后在 MIDDLEWARE 中添加与安全相关的中间件，例如 CsrfViewMiddleware 和 XFrameOptionsMiddleware。代码如下。

```
SECRET_KEY = '3&2xq%nb^+4k%m2rgg-ry4ybh(-o6'
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
```

```
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

2.2.2 会员注册和登录验证模块

在本项目的 user 目录中保存了会员注册和登录验证模块的实现代码，具体实现流程如下。

(1) 在模型文件 modcls.py 中设置与会员用户有关的数据库表，代码如下。

```
class UserManager(models.Manager):
    def all(self):
        return super().all().filter(is_delete=False)

    def create(self, username, password):
        user = self.model()
        user.username = username
        user.password = make_password(password)
        user.save()
        return user

    #在这里添加模型管理方法


class User(models.Model):
    username = models.CharField(max_length=16, unique=True)
    password = models.CharField(max_length=256)
    post_count = models.IntegerField(default=0)
    comm_count = models.IntegerField(default=0)
    is_delete = models.BooleanField(default=False)
    objects = UserManager()
    class Meta:
        db_table = 'users'

    #通过加密算法验证密码
    def valid_password(self, password):
        return check_password(password, self.password)
```

(2) 在文件 urls.py 中设置相关页面的路径导航，代码如下。

```
from django.urls import path
from user import views

urlpatterns = [
    path('register/', views.register, name='register'),
    path('register_handler/', views.register_handler, name='register_handler'),
```

```
    path('login/', views.login, name='login'),
    path('login_handler/', views.login_handler, name='login_handler'),
    path('logout/', views.logout, name='logout'),
]
```

(3) 在表单文件 forms.py 中定义了两个类，分别实现新用户注册表单功能和会员登录表单功能。代码如下。

```
class LoginForm(forms.Form):
    username = forms.CharField(max_length=16,
               widget=forms.TextInput(attrs={'class': 'form-control',
               'placeholder': '请输入用户名'}))
    password = forms.CharField(min_length=6, max_length=32,
               widget=forms.PasswordInput(attrs={'class': 'form-control',
               'placeholder': '请输入密码'}))
    remember_me = forms.BooleanField(required=False)

    def clean(self):
        cleaned_data = self.cleaned_data
        username = self.cleaned_data['username']
        pwd = self.cleaned_data['password']
        self.valid_username(username)
        self.valid_password(pwd)
        return cleaned_data

    def valid_username(self, username):
        try:
            self.user = User.objects.filter(username=username)[0]
        except IndexError:
            raise forms.ValidationError(_('该用户不存在'))

    def valid_password(self, pwd):
        if not self.user.valid_password(pwd):
            raise forms.ValidationError(_('密码错误'))


class RegisterForm(forms.Form):
    username = forms.CharField(max_length=16,
                          widget=forms.TextInput(attrs={'class': 'form-control'}))
    password = forms.CharField(min_length=6, max_length=32,
                          widget=forms.PasswordInput(attrs={'class': 'form-control'}))
    confirm_password = forms.CharField(min_length=6, max_length=32,
                  widget=forms.PasswordInput(attrs={'class': 'form-control'}))

    def clean(self):
        cleaned_data = self.cleaned_data
        username = self.cleaned_data['username']
        pwd1 = self.cleaned_data['password']
```

```
        pwd2 = self.cleaned_data['confirm_password']
        self.valid_username(username)
        self.valid_password(pwd1, pwd2)
        return cleaned_data

    def valid_username(self, username):
        if re.findall('[^0-9a-zA-Z_]', username):
            raise forms.ValidationError(_('用户名只允许使用数字字母或下划线'))
        if User.objects.filter(username=username):
            raise forms.ValidationError(_('用户名%(username)s 已经被注册了'),
            params={'username': username})

    def valid_password(self, pwd1, pwd2):
        if re.findall('[^0-9a-zA-Z_]', pwd1):
            raise forms.ValidationError(_('密码只允许使用数字字母或下划线'))
        if pwd1 != pwd2:
            raise forms.ValidationError(_('两次密码输入不一致'))
```

(4) 在视图文件 views.py 中编写视图处理函数，根据获取的注册表单数据实现新用户注册逻辑功能，根据从登录表单获取的表单数据实现登录验证功能。此外，为了提高系统的安全性，特意定义了函数 cookie_handler(username)，用于将登录用户的 Cookie 数据进行加密。

(5) 编写系统后台文件 admin.py，在后台中添加博客信息管理功能，代码如下。

```
from django.contrib import admin

from .models import User
admin.site.register(User)
```

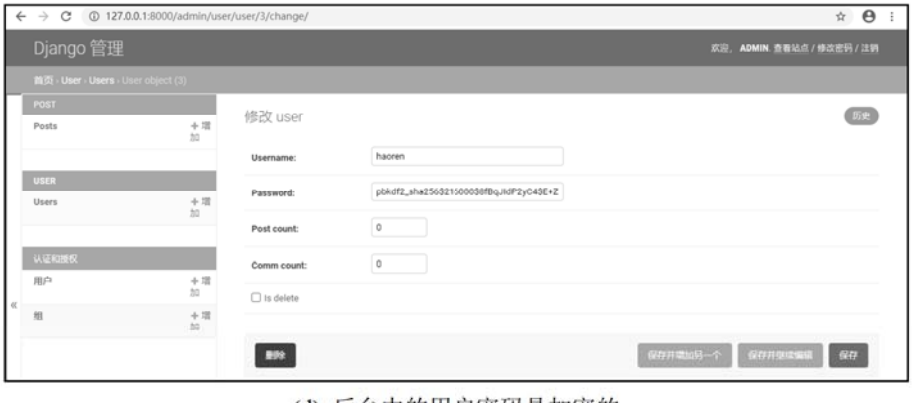
执行效果如图 2-13 所示。

(a) 会员注册页面

图 2-13 执行效果

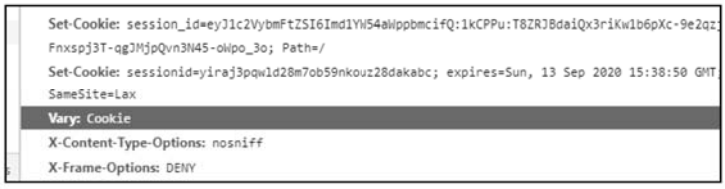
(b) 登录验证页面

(c) 会员登录成功时的首页效果



(d) 后台中的用户密码是加密的

图 2-13 执行效果(续)



(e) 在浏览器中保存的 Cookie(名字是 sessionid)也是加密的

图 2-13　执行效果(续)

2.2.3　博客发布模块

在本项目的 post 目录中保存了博客发布模块的实现代码，具体实现流程如下。

(1) 在模型文件 models.py 中设置与会员用户有关的数据库表，代码如下。

```
#帖子管理
class PostManager(models.Manager):
    def all(self):
        return super().all().filter(is_delete=False)

    def create(self, user, title, author, cont_html, cont_str):
        post = self.model()
        post.user = user
        post.author = author
        post.title = title
        post.cont_html = cont_html
        post.cont_str = cont_str
        post.save()
        return post
#在这里添加模型管理方法
#帖子
class Post(models.Model):
    title = models.CharField(max_length=50)
    author = models.CharField(max_length=16)
    cont_html = models.TextField()
    cont_str = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)
    view_count = models.IntegerField(default=0)
    like_count = models.IntegerField(default=0)
    coll_count = models.IntegerField(default=0)
    comm_count = models.IntegerField(default=0)
    is_delete = models.BooleanField(default=False)

    user = models.ForeignKey(User,on_delete=models.CASCADE)
```

```
    likes = models.ManyToManyField(User, through='Like', related_name='likes')
    colls = models.ManyToManyField(User, through='Collection', related_name='colls')
    comms = models.ManyToManyField(User, through='Comment', related_name='comms')
    objects = PostManager()
    class Meta:
        db_table = 'posts'

#点赞
class Like(models.Model):
    is_list = models.BooleanField(default=True)
    timestamp = models.DateTimeField(auto_now_add=True)
    uid = models.ForeignKey(User, related_name='like',on_delete=models.CASCADE)
    pid = models.ForeignKey(Post, related_name='like',on_delete=models.CASCADE)

#收藏
class Collection(models.Model):
    is_coll = models.BooleanField(default=True)
    timestamp = models.DateTimeField(auto_now_add=True)
    uid = models.ForeignKey(User, related_name='coll',on_delete=models.CASCADE)
    pid = models.ForeignKey(Post, related_name='coll',on_delete=models.CASCADE)

#评论
class Comment(models.Model):
    cont_str = models.CharField(max_length=256)
    is_delete = models.BooleanField(default=False)
    timestamp = models.DateTimeField(auto_now_add=True)
    uid = models.ForeignKey(User, related_name='comm',on_delete=models.CASCADE)
    pid = models.ForeignKey(Post, related_name='comm',on_delete=models.CASCADE)
    replys = models.ManyToManyField(User, through='Reply', related_name='replys')
```

(2) 在文件 urls.py 中设置相关页面的路径导航。

(3) 在表单文件 forms.py 中创建类 PostEditForm，用于获取博客表单中的信息，包括博客标题和正文。代码如下。

```
class PostEditForm(forms.Form):
    post_title = forms.CharField(max_length=50,
                 widget=forms.TextInput(attrs={'class': 'form-control',
                 'placeholder': '标题'}))
    post_content = forms.CharField(widget=forms.Textarea(attrs={'class':
                   'form-control',  'placeholder': '正文'}))

    def clean(self):
        cleaned_data = self.cleaned_data
        title = self.cleaned_data['post_title']
        cont_str = self.cleaned_data['post_content']
        cont_html = cont_str
        return cleaned_data
```

(4) 在视图文件 views.py 中编写视图处理函数，首先判断用户是否已经登录系统，如果没有登录系统，就不能发布博客；如果已经登录系统，则根据获取的表单数据实现博客发布功能。此外，为了提高系统的安全性，特意使用底层 API 中的 signing 将用户的登录信息进行加密。

(5) 编写系统后台文件 admin.py，在后台中注册添加博客信息管理功能。代码如下。

```
from django.contrib import admin
from .models import Post
class PostAdmin(admin.ModelAdmin):
    fields = ['title', 'author']
admin.site.register(Post, PostAdmin)
```

执行效果如图 2-14 所示。

图 2-14 博客发布界面