第 3 章

巩固 200 例

例 301　单例模式

1. 问题描述

单例是最常见的设计模式之一。对于任何时刻，如果某个类只存在且最多存在一个具体的实例，则称这种设计模式为单例。例如，对于 class Mouse，应将其设计为单例模式，设计一个 getInstance()方法，对于给定的类，每次调用 getInstance()时，都可得到同一个实例。

2. 问题示例

在 Python 中使用 getInstance()方法获得实例 a 和 b，即 a＝A. getInstance()，b＝A. getInstance()，a 应等于 b。

3. 代码实现

相关代码如下。

```
class Solution:
    #返回这个类的同一个实例
    instance = None
    @classmethod
    def getInstance(cls):
        if cls.instance is None:
            cls.instance = Solution()
        return cls.instance
if __name__ == '__main__':
    temp = Solution()
    a = temp.getInstance()
    b = temp.getInstance()
    print('输入：“a = temp.getInstance()” 和 “b = temp.getInstance()”')
    print('输出：a 和 b 是否得到同一个实例?' + str(a == b))
```

4. 运行结果

```
输入：“a = temp.getInstance()” 和 “b = temp.getInstance()”
输出：a 和 b 是否得到同一个实例?True
```

例 302　字符串置换

1. 问题描述

对于给定的两个字符串，设计一个方法判定其中一个字符串是否为另一个字符串的置换。置换是指通过改变顺序使得两个字符串相等。

2. 问题示例

若输入 abcd、bcad，则输出 True。若输入 aac、abc，则输出 False。

3. 代码实现

相关代码如下。

```
class Solution:
    #参数 str1 和 str2: 需要判断的两个字符串
    #返回值: 布尔类型,判断两个字符串是否可以置换
    def permutation(self, str1, str2):
        s1 = "".join((lambda x: (x.sort(), x)[1])(list(str1)))
        s2 = "".join((lambda x: (x.sort(), x)[1])(list(str2)))
        if s1 == s2:
            return True
        else:
            return False
if __name__ == '__main__':
    temp = Solution()
    str1 = 'abcdd'
    str2 = 'dbcad'
    str3 = 'acd'
    str4 = 'abc'
    print('输入: str1 = ' + str1 + 'str2 = ' + str2)
    print('输出: ' + str(temp.permutation(str1, str2)))
    print('输入: str3 = ' + str3 + 'str4 = ' + str4)
    print('输出: ' + str(temp.permutation(str3, str4)))
```

4. 运行结果

```
输入: str1 = abcdd str2 = dbcad
输出: True
输入: str3 = acd str4 = abc
输出: False
```

例 303　字符串替换

1. 问题描述

设计一种方法，通过对重复字符计数进行基本的字符串压缩。如字符串"aabccccaaa"

可压缩为"a2b1c5a3"。如果压缩后的字符数不小于原始的字符数，则返回原始的字符串。可以假设字符串仅包括 a～z 的字母。

2. 问题示例

输入 str＝"aabcccccaaa"，输出"a2b1c5a3"；输入 str＝"aabbcc"，输出"aabbcc"。

3. 代码实现

相关代码如下。

```
class Solution:
#参数 string: 字符数组
#返回值: 压缩后的字符串
    def compressString(self, string):
        l = len(string)
        if l <= 2:
            return string
        length = 1
        res = ''
        for i in range(1, l):
            if string[i] != string[i - 1]:
                res = res + string[i - 1] + str(length)
                length = 1
            else:
                length += 1
        if string[-1] != string[-2]:
            res = res + string[-1] + '1'
        else:
            res = res + string[-1] + str(length)
        if len(string) <= len(res):
            return string
        else:
            return res
if __name__ == '__main__':
    temp = Solution()
    str1 = 'aaabbccccdde'
    str2 = 'hellooo'
    print('输入: '+str1)
    print('输出: '+temp.compressString(str1))
    print('输入: ' + str2)
    print('输出: ' + temp.compressString(str2))
```

4. 运行结果

```
输入: aaabbccccdde
输出: a3b2c4d2e1
输入: hellooo
输出: hellooo
```

例 304　用 isSubstring 判断字符串的循环移动

1. 问题描述

假定有一种 isSubstring 方法，可以检验某个单词是否为另一个单词的子字符串。给定 $s1$ 和 $s2$，请设计一种方法检验 $s2$ 是否为 $s1$ 循环移动后的字符串。注意，只能调用一次 isSubstring。

2. 问题示例

输入 $s1$＝"waterbottle"，$s2$＝"erbottlewat"，输出 True，因为"waterbottle"是"erbottlewat"的一种循环移动后的字符串。输入 $s1$＝"apple"，$s2$＝"ppale"，输出 False，因为"apple"不是"ppale"的一种循环移动后的字符串。

3. 代码实现

相关代码如下。

```
class Solution:
    #参数 s1 为第 1 个字符串
    #参数 s2 为第 2 个字符串
    #返回值: 布尔类型
    def isRotation(self, s1, s2):
        if len(s1) != len(s2) or len(s1) == 0:
            return False
        s1s1 = s1 + s1
        return self.isSubstring(s1s1, s2)
    def isSubstring(self, s, t):
        return s.find(t) != -1
if __name__ == '__main__':
    temp = Solution()
    s1 = 'abcdef'
    s2 = 'defabc'
    s3 = 'abcefg'
    s4 = 'efgacb'
    print('输入: s1 = ' + s1 + 's2 = ' + s2)
    print('输出: ' + str(temp.isRotation(s1, s2)))
    print('输入: s3 = ' + s3 + 's4 = ' + s4)
    print('输出: ' + str(temp.isRotation(s3, s4)))
```

4. 运行结果

```
输入: s1 = abcdef  s2 = defabc
输出: True
输入: s3 = abcefg  s4 = efgacb
输出: False
```

例 305　能否到达终点

1. 问题描述

给定一个大小为 m 行 n 列的矩阵表示地图，1 代表空地，0 代表障碍物，9 代表终点。判断从(0,0)开始能否到达终点，若能到达终点则返回 True，否则返回 False。

2. 问题示例

输入[[1,1,1],[1,1,1],[1,1,9]]，输出 True。

3. 代码实现

相关代码如下。

```
import queue as Queue
DIRECTIONS = [(-1, 0), (1, 0), (0, 1), (0, -1)]
SAPCE = 1
OBSTACLE = 0
ENDPOINT = 9
class Solution:
#参数 map: 一个地图
#返回值: 布尔类型,判断能否到达终点
    def reachEndpoint(self, map):
        if not map or not map[0]:
            return False
        self.n = len(map)
        self.m = len(map[0])
        queue = Queue.Queue()
        queue.put((0, 0))
        while not queue.empty():
            curr = queue.get()
            for i in range(4):
                x = curr[0] + DIRECTIONS[i][0]
                y = curr[1] + DIRECTIONS[i][1]
                if not self.isValid(x, y, map):
                    continue
                if map[x][y] == ENDPOINT:
                    return True
                queue.put((x, y))
                map[x][y] = OBSTACLE
        return False
    def isValid(self, x, y, map):
        if x < 0 or x >= self.n or y < 0 or y >= self.m:
            return False
        if map[x][y] == OBSTACLE:
            return False
        return True
```

```
#主函数
if __name__ == '__main__':
    map = [[1, 1, 1], [1, 1, 1], [1, 1, 9]]
    print("地图:", map)
    solution = Solution()
    print("能否到达终点:#", solution.reachEndpoint(map))
```

4. 运行结果

```
地图:[[1, 1, 1], [1, 1, 1], [1, 1, 9]]
能否到达终点: #True
```

例 306　成绩等级

1. 问题描述

实现 Student 类,包含如下成员函数和方法:两个公有成员名称(name)和成绩(score),分别是字符串类型和整数类型;一个构造函数,接收一个参数 name;一个公有成员函数 getLevel(),返回学生的成绩等级(一个字符)。

2. 问题示例

对学生的成绩分段如下:A,score >= 90;B,80≤score < 90;C,60≤score < 80;D,score < 60。如果学生的成绩为 95,则返回 A。

3. 代码实现

相关代码如下。

```
class Student:
    def __init__(self, name):
        self.name = name
        self.score = 0
    def getLevel(self):
        if self.score >= 90:
            return 'A'
        elif self.score >= 80:
            return 'B'
        elif self.score >= 60:
            return 'C'
        else:
            return 'D'
if __name__ == '__main__':
    student1 = Student('Jack')
    student1.score = 60
    student2 = Student('Ama')
    student2.score = 95
    print('输入: 学生 1 Jack,成绩 60')
```

```
    print('输出：学生等级' + student1.getLevel())
    print('输入：学生 2 Ama,成绩 95')
    print('输出：学生等级' + student2.getLevel())
```

4. 运行结果

```
输入：学生 1 Jack,成绩 60
输出：学生等级 C
输入：学生 2 Ama,成绩 95
输出：学生等级 A
```

例 307　在排序链表中插入一个节点

1. 问题描述

在排序链表中插入一个节点。

2. 问题示例

输入 head＝1→4→6→8→null，val＝5，输出 1→4→5→6→8→null。

输入 head＝1→null，val＝2，输出 1→2→null。

3. 代码实现

相关代码如下。

```
class ListNode(object):
    def __init__(self, val, next = None):
        self.val = val
        self.next = next
class Solution:
    #参数 head：链表节点
    #参数 val：要插入的整数
    #返回值：新链表的节点
    def insertNode(self, head, val):
        dummy = ListNode(0, head)
        p = dummy
        while p.next and p.next.val < val:
            p = p.next
        node = ListNode(val, p.next)
        p.next = node
        return dummy.next
def getLinkedList(head):
    list = []
    while head is not None:
        list += [str(head.val)]
        head = head.next
    list.append('null')
    s = '->'.join(list)
```

```
        return s
if __name__ == '__main__':
    temp = Solution()
    LinkedNode1 = ListNode(1, ListNode(2, ListNode(3, ListNode(3, ListNode(5, ListNode(7))))))
    val1 = 4
    LinkedNode2 = ListNode(1, ListNode(3, ListNode(9, ListNode(11, ListNode(12, ListNode(15))))))
    val2 = 13
    print('输入: ' + getLinkedList(LinkedNode1) + 'val1 = ' + str(val1))
    print('输出: ' + getLinkedList(temp.insertNode(LinkedNode1, val1)))
    print('输入: ' + getLinkedList(LinkedNode2) + 'val2 = ' + str(val2))
    print('输出: ' + getLinkedList(temp.insertNode(LinkedNode2, val2)))
```

4. 运行结果

```
输入: 1→2→3→3→5→7→null val1 = 4
输出: 1→2→3→3→4→5→7→null
输入: 1→3→9→11→12→15→null val2 = 13
输出: 1→3→9→11→12→13→15→null
```

例 308　getter 与 setter

1. 问题描述

本例将实现一个 School 类，属性和方法分别为：一个字符串(string)类型的私有成员名称(name)；一个 setter 方法 setName，包含一个参数名称(name)；一个 getter 方法 getName，返回该对象的名称(name)。

2. 问题示例

school＝School()；
school.setName("MIT")
school.getName()
返回"MIT"作为结果。

3. 代码实现

相关代码如下。

```
class School:
    def __init__(self):
        self.__name = ''
    def setName(self, name):
        self.__name = name
    def getName(self):
        return self.__name
if __name__ == '__main__':
    school1 = School()
    school1.setName('MIT')
```

```
school2 = School()
school2.setName('UIUC')
print('输入: school1 name = ' + school1.getName())
print('输出: school2 name = ' + school2.getName())
```

4. 运行结果

```
输入: school1 name = MIT
输出: school2 name = UIUC
```

例 309　用一个数组实现 3 个栈

1. 问题描述

用一个数组实现 3 个栈,假设这 3 个栈一样大并且足够大。

2. 问题示例

ThreeStacks(5):创建 3 个栈,每个栈大小为 5。
push(0,10):把 10 放入第 1 个栈。
push(0,11):把 11 放入第 1 个栈。
push(1,20):把 20 放入第 2 个栈。
push(1,21):把 21 放入第 2 个栈。
pop(0):返回 11。
pop(1):返回 21。
peek(1):返回 20。
push(2,30):把 30 放入第 3 个栈。
pop(2):返回 30。
isEmpty(2):返回 True。
isEmpty(0):返回 False。

3. 代码实现

相关代码如下。

```
class ThreeStacks:
    #参数 size: 整型, 代表每个栈的大小
    def __init__(self, size):
        self.size = size
        self.stacks = [[], [], []]
    #stack_num: 整数,代表第几个栈
    #value: 整型,代表放入栈的数值
    def push(self, stack_num, value):
        self.stacks[stack_num].append(value)
    #stack_num: 整型,代表第几个栈
    #返回值: 整数,栈中存在的值
```

```
    def pop(self, stack_num):
        return self.stacks[stack_num].pop()
    #参数 stack_num: 整数,代表第几个栈
    #返回值: 整数,即栈中存在的值
    def peek(self, stack_num):
        return self.stacks[stack_num][-1]
    #参数 stack_num: 整数,代表第几个栈
    #返回值: 布尔类型,判断栈是否为空
    def isEmpty(self, stack_num):
        return len(self.stacks[stack_num]) == 0
if __name__ == '__main__':
    temp = ThreeStacks(5)
    temp.push(0, 5)
    temp.push(0, 10)
    temp.push(1, 5)
    a = temp.pop(0)
    b = temp.pop(1)
    temp.push(2, 10)
    temp.push(2, 3)
    c = temp.pop(2)
    d = temp.peek(2)
    e = temp.isEmpty(1)
    f = temp.isEmpty(0)
    print('''输入:temp = ThreeStacks(5)
     temp.push(0,5)
     temp.push(0,10)
     temp.push(1,5)
     temp.pop(0)
     temp.pop(1)
     temp.push(2,10)
     temp.push(2,3)
     temp.pop(2)
     temp.peek(2)
     temp.isEmpty(1)
     temp.isEmpty(0)''')
    print('输出:temp.pop(0) = ' + str(a) + 'temp.pop(1) = ' + str(b))
    print('     temp.pop(2) = ' + str(c) + 'temp.peek(2) = ' + str(d))
    print('     temp.isEmpty(1) = ' + str(e) + 'temp.isEmpty(0) = ' + str(f))
```

4. 运行结果

```
输入: temp = ThreeStacks(5)
      temp.push(0,5)
      temp.push(0,10)
      temp.push(1,5)
      temp.pop(0)
      temp.pop(1)
```