python面试题搜集（八）：史上最全python面试题详解 （二）（附带详细答案（关注、持续更新））

23、re的match和search区别?

re.match()从开头开始匹配string。 re.search()从anywhere 来匹配string。

```
# 多行模式>>> re.match('X', 'A\nB\nX', re.MULTILINE)  # No match
>>> re.search('^X', 'A\nB\nX', re.MULTILINE)  # Match
<_sre.SRE_Match object at ...>
```

24、什么是正则的贪婪匹配

```
1 如：String str="abcaxc";
2     Patter p="ab.*c";
3     贪婪匹配:正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。如上面使用模式p匹配字符串str，结果就是匹配到：abcaxc(ab.*c)。
4     非贪婪匹配：就是匹配到结果就好，就少的匹配字符。如上面使用模式p匹配字符串str，结果就是匹配到：abc(ab.*c)。
```

25、def func(a,b=[]) 这种写法有什么坑?

```
def func(a,b=[]):
    b.append(a)
    print(b)
func(1)
func(1)
func(1)
func(1)
```

看下结果

[1]　　[1, 1]　　[1, 1, 1]　　[1, 1, 1, 1]

函数的第二个默认参数是一个list，当第一次执行的时候实例化了一个list，第二次执行还是用第一次执行的时候实例化的地址存储，所以三次执行的结果就是 [1, 1, 1] ，想每次执行只输出[1] ，默认参数应该设置为None。

26、如何实现 “1,2,3” 变成 [‘1’,’2’,’3’] ?

```
1 # encoding: utf-8
2 a = "1,2,3"
3 b = a.split(",")
4 print b
```

27、如何实现[‘1’,’2’,’3’]变成[1,2,3]?

```
1 >>>a = ['1', '2', '3']
2 >>>b = [int(i) for i in a]
3 >>>b
```

28、a = [(1),(2),(3)] 以及 b = [(1,),(2,),(3,)]的区别?

a里面的元素都是int类型，b中的元素都是元组。

元组只有一个元素，必须要加逗号结尾

29、一行代码选出列表里面的不重复元素

```
1 b = [1,2,3,2,3,4,1,2]
2 print [i  for i in b if b.count(i)==1]
3 >>>4
```

30、logging模块的作用？以及应用场景

日志是一种可以追踪某些软件运行时所发生事件的方法。

不同的应用程序所定义的日志等级可能会有所差别，分的详细点的会包含以下几个等级：

- DEBUG
- INFO
- NOTICE
- WARNING
- ERROR
- CRITICAL
- ALERT
- EMERGENCY

```
import logging
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',
                    datefmt='%a, %d %b %Y %H:%M:%S',
                    filename='test.log',
                    filemode='a')
try:
    int(input('num >>'))
except ValueError:
    logging.error('输入的值不是一个数字')
```

所有的日志信息只能自己设置，系统不会跟你检查，然后主动输出

```
import logging
logger = logging.getLogger()
fh = logging.FileHandler('log.log',encoding='utf-8')
sh = logging.StreamHandler()     # 创建一个屏幕控制对象
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
# 文件操作符 和 格式关联
fh.setFormatter(formatter)   设置日志对象输出的格式
# logger 对象 和 文件操作符 关联
logger.addHandler(fh)    将日志文件对象添加到logger对象中
logger.addHandler(sh)    将屏幕显示对象添加到logger对象中
logging.debug('debug message')        # 低级别的 # 排错信息
logging.info('info message')            # 正常信息
logging.warning('警告错误')       # 警告信息
logging.error('error message')          # 错误信息
```

31、请用代码简答实现stack

```
1 stack的实现代码（使用python内置的list），实现起来是非常的简单，就是list的一些常用操作
2
3 class Stack(object):
4     def __init__(self):
5         self.stack = []
6
7     def push(self, value):    # 进栈
8         self.stack.append(value)
9
10    def pop(self):  #出栈
11        if self.stack:
12            self.stack.pop()
13        else:
14            raise LookupError('stack is empty!')
15
16    def is_empty(self): # 如果栈为空
17        return bool(self.stack)
18
19    def top(self):
20        #取出目前stack中最新的元素
21        return self.stack[-1]
```

```
1 定义一个头结点，左边指向队列的开头，右边指向队列的末尾，这样就可以保证我们插入一个元素和取出一个元素都是O(1)的操作，使用这种链表实现stack也是非常的方便。实现代码如下：
2
3 class Head(object):
4     def __init__(self):
5         self.left = None
6         self.right = None
7
8 class Node(object):
9     def __init__(self, value):
```

```
10          self.value = value
11          self.next = None
12
13  class Queue(object):
14      def __init__(self):
15          #初始化节点
16          self.head = Head()
17
18      def enqueue(self, value):
19          #插入一个元素
20          newnode = Node(value)
21          p = self.head
22          if p.right:
23              #如果head节点的右边不为None
24              #说明队列中已经有元素了
25              #就执行下列的操作
26              temp = p.right
27              p.right = newnode
28              temp.next = newnode
29          else:
30              #这说明队列为空，插入第一个元素
31              p.right = newnode
32              p.left = newnode
33
34      def dequeue(self):
35          #取出一个元素
36          p = self.head
37          if p.left and (p.left == p.right):
38              #说明队列中已经有元素
39              #但是这是最后一个元素
40              temp = p.left
41              p.left = p.right = None
42              return temp.value
43          elif p.left and (p.left != p.right):
44              #说明队列中有元素，而且不止一个
45              temp = p.left
46              p.left = temp.next
47              return temp.value
48
49          else:
50              #说明队列为空
51              #抛出查询错误
52              raise LookupError('queue is empty!')
53
54      def is_empty(self):
55          if self.head.left:
56              return False
57          else:
58              return True
59
60      def top(self):
61          #查询目前队列中最早入队的元素
62          if self.head.left:
63              return self.head.left.value
64          else:
65              raise LookupError('queue is empty!')
```

32、常用字符串格式化哪几种?

```
 1 一、使用%
 2 %s       字符串
 3 %c       字符
 4 %d       十进制（整数）
 5 %i       整数
 6 %u      无符号整数
 7 %o       八进制整数
 8 %x      十六进制整数
 9 %X       十六进制整数大写
10 %e       浮点数格式1
11 %E       浮点数格式2
12 %f       浮点数格式3
13 %g      浮点数格式4
14 %G      浮点数格式5
15 %%       文字%
16 >>> print("我叫%s，今年%d岁了" % ("小李", 20))
17 我叫小李，今年20岁了
```

```
二、通过{}替代%
1、正常使用
>>> print("我叫{}，今年{}岁了".format("小李", 20))
我叫小李，今年20岁了
2、还可以通过在括号里填写数字，修改格式化的顺序
>>> print("我叫{1}，今年{0}岁了".format("小李", 20))
我叫20，今年小李岁了
3、通过key取变量
>>> print("我叫{name}，今年{age}岁了".format(name="小李", age=20))
我叫小李，今年20岁了
4、传入对象
>>> class Person:
...     def __init__(self,name,age):
...
...             self.name,self.age = name,age
...     def __str__(self):
...
...             return '我叫{self.name}，今年{self.age}岁了'.format(self=self)
>>> str(Person('小李',20))
'我叫小李，今年20岁了'
5、通过下标
>>> person=['小李',20]
>>> '我叫{0[0]}，今年{0[1]}岁了'.format(person)
'我叫小李，今年20岁了'
```

33、简述 生成器、迭代器、可迭代对象 以及应用场景?

Python可迭代对象（Iterable） Python中经常使用`for`来对某个对象进行遍历，此时被遍历的这个对象就是可迭代对象，像常见的`list`,`tuple`都是。如果给一个准确的定义的话，就是只要它定义了可以返回一个迭代器的`__iter__`方法，或者定义了可以支持下标索引的`__getitem__`方法(这些双下划线方法会在其他章节中全面解释)，那么它就是一个可迭代对象。

Python迭代器（iterator） 迭代器是通过`next()`来实现的，每调用一次他就会返回下一个元素，当没有下一个元素的时候返回一个`StopIteration`异常，所以实际上定义了这个方法的都算是迭代器。可以用通过下面例子来体验一下迭代器：

生成器（Generators） 生成器是构造迭代器的最简单有力的工具，与普通函数不同的只有在返回一个值的时候使用`yield`来替代`return`，然后`yield`会自动构建好`next()`和`iter()`

因为迭代器如此普遍，python专门为for关键字做了迭代器的语法糖。在for循环中，Python将自动调用工厂函数iter()获得迭代器，自动调用next()获取元素，还完成了检查StopIteration异常的工作。

```
1 1.3 定义迭代器
2 下面一个例子——斐波那契数列
3 # -*- coding: cp936 -*-
4 class Fabs(object):
5     def __init__(self,max):
6         self.max = max
7         self.n, self.a, self.b = 0, 0, 1  #特别指出：第0项是0，第1项是第一个1.整个数
列从1开始
8     def __iter__(self):
9         return self
10    def next(self):
11        if self.n < self.max:
12            r = self.b
13            self.a, self.b = self.b, self.a + self.b
14            self.n = self.n + 1
15            return r
16        raise StopIteration()
17
18 print Fabs(5)
19 for key in Fabs(5):
20     print key
21
22 结果
23 <__main__.Fabs object at 0x01A63090>
24 1
25 1
26 2
27 3
28 5
```

34、用Python实现一个二分查找的函数

```
1 #!usr/bin/env python
2 #encoding:utf-8
3 def binary_search(num_list, x):
```

```
    '''
    二分查找
    '''
    num_list=sorted(num_list)
    left, right = 0, len(num_list)
    while left < right:
        mid = (left + right) / 2
        if num_list[mid] > x:
            right = mid
        elif num_list[mid] < x:
            left = mid + 1
        else:
            return '待查元素{0}在列表中下标为：{1}'.format(x, mid)
    return  '待查找元素%s不存在指定列表中'%x

if __name__ == '__main__':
    num_list = [34,6,78,9,23,56,177,33,2,6,30,99,83,21,17]
    print binary_search(num_list, 34)
    print binary_search(num_list, 177)
    print binary_search(num_list, 21)
    print binary_search(num_list, 211)
    print binary_search(num_list, 985)
》》》
待查元素34在列表中下标为：9
待查元素177在列表中下标为：14
待查元素21在列表中下标为：5
待查找元素211不存在指定列表中
待查找元素985不存在指定列表中
```

35、谈谈你对闭包的理解。

```
#闭包函数的实例
# outer是外部函数 a和b都是外函数的临时变量
def outer( a ):
    b = 10
    # inner是内函数
    def inner():
        #在内函数中 用到了外函数的临时变量
        print(a+b)
    # 外函数的返回值是内函数的引用
    return inner

if __name__ == '__main__':
    # 在这里我们调用外函数传入参数5
    #此时外函数两个临时变量 a是5 b是10 ，并创建了内函数，然后把内函数的引用返回存给了demo
    # 外函数结束的时候发现内部函数将会用到自己的临时变量，这两个临时变量就不会释放，会绑定给这个内部函数
    demo = outer(5)
    # 我们调用内部函数，看一看内部函数是不是能使用外部函数的临时变量
    # demo存了外函数的返回值，也就是inner函数的引用，这里相当于执行inner函数
    demo() # 15

    demo2 = outer(7)
```

```
22      demo2()#17
```

36、os和sys模块的作用

os模块负责程序与操作系统的交互，提供了访问操作系统底层的接口;

sys模块负责程序与python解释器的交互，提供了一系列的函数和变量，用于操控python的运行时环境。

```
os 常用方法


os.remove(‘path/filename’) 删除文件

os.rename(oldname, newname) 重命名文件

os.walk() 生成目录树下的所有文件名

os.chdir('dirname') 改变目录

os.mkdir/makedirs('dirname')创建目录/多层目录

os.rmdir/removedirs('dirname') 删除目录/多层目录

os.listdir('dirname') 列出指定目录的文件

os.getcwd() 取得当前工作目录

os.chmod() 改变目录权限

os.path.basename(‘path/filename’) 去掉目录路径，返回文件名

os.path.dirname(‘path/filename’) 去掉文件名，返回目录路径

os.path.join(path1[,path2[,...]]) 将分离的各部分组合成一个路径名

os.path.split('path') 返回( dirname(), basename())元组

os.path.splitext() 返回 (filename, extension) 元组

os.path.getatime\ctime\mtime 分别返回最近访问、创建、修改时间

os.path.getsize() 返回文件大小

os.path.exists() 是否存在

os.path.isabs() 是否为绝对路径

os.path.isdir() 是否为目录

os.path.isfile() 是否为文件


```

```
45 sys 常用方法
46
47
48 sys.argv 命令行参数List，第一个元素是程序本身路径
49
50 sys.modules.keys() 返回所有已经导入的模块列表
51
52 sys.exc_info() 获取当前正在处理的异常类,exc_type、exc_value、exc_traceback当前处理的
异常详细信息
53
54 sys.exit(n) 退出程序，正常退出时exit(0)
55
56 sys.hexversion 获取Python解释程序的版本值，16进制格式如：0x020403F0
57
58 sys.version 获取Python解释程序的版本信息
59
60 sys.maxint 最大的Int值
61
62 sys.maxunicode 最大的Unicode值
63
64 sys.modules 返回系统导入的模块字段，key是模块名，value是模块
65
66 sys.path 返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值
67
68 sys.platform 返回操作系统平台名称
69
70 sys.stdout 标准输出
71
72 sys.stdin 标准输入
73
74 sys.stderr 错误输出
75
76 sys.exc_clear() 用来清除当前线程所出现的当前的或最近的错误信息
77
78 sys.exec_prefix 返回平台独立的python文件安装的位置
79
80 sys.byteorder 本地字节规则的指示器，big-endian平台的值是'big',little-endian平台的值
是'little'
81
82 sys.copyright 记录python版权相关的东西
83
84 sys.api_version 解释器的C的API版本
```

37、谈谈你对面向对象的理解?

在我理解,面向对象是向现实世界模型的自然延伸，这是一种“万物皆对象”的编程思想。在现实生活中的任何物体都可以归为一类事物，而每一个个体都是一类事物的实例。面向对象的编程是以对象为中心，以消息为驱动，所以程序=对象+消息。

面向对象有三大特性，封装、继承和多态。

封装就是将一类事物的属性和行为抽象成一个类，使其属性私有化，行为公开化，提高了数据的隐秘性的同时，使代码模块化。这样做使得代码的复用性更高。

继承则是进一步将一类事物共有的属性和行为抽象成一个父类，而每一个子类是一个特殊的父类--有父类的行为和属性，也有自己特有的行为和属性。这样做扩展了已存在的代码块，进一步提高了代码的复用性。

如果说封装和继承是为了使代码重用，那么多态则是为了实现接口重用。多态的一大作用就是为了解耦--为了解除父子类继承的耦合度。如果说继承中父子类的关系式IS-A的关系，那么接口和实现类之之间的关系式HAS-A。简单来说，多态就是允许父类引用(或接口)指向子类(或实现类)对象。很多的设计模式都是基于面向对象的多态性设计的。

总结一下，如果说封装和继承是面向对象的基础，那么多态则是面向对象最精髓的理论。掌握多态必先了解接口，只有充分理解接口才能更好的应用多态。

38、面向对象深度优先和广度优先是什么?

请关注，未完待续!