python常见面试题

概述

- 程序员面试时，无论是沟通的面试还是写文字的笔试，都蕴含着逻辑思维的考察，所以在整个过程中，尽量让自己的语言有一定的逻辑性。
- 逻辑表达的语言特点：
  - 这个问题，主要有几个可能，第一，...，第二，...，第三，...
  - 首先，然后，再次，最后
  - 如果是这个条件的话，那么...，否则就肯定是...
  - 而... 但是...

1. *args和**kwargs是什么，有什么用处?

> **概述**
>
> - 这2个是在函数定义时，为了接收不确定个数的参数所使用的。
> - *args接收的是不定长的位置参数，而args这个变量其实就是一个元组。
> - **kwargs接收的是不定长的命名参数，而kwargs这个变量其实就是一个字典。
>
> **应用场景**
>
> - 在实现装饰器时，为了可以对任意的函数都能进行装饰，内置的闭包函数就会用*args,**kwargs来处理。
> - 反过来，星号*把序列或者集合解包（unpack）成位置参数，两个星号**把字典解包成关键词参数
>
> **另外一种说法**
>
> 如果我们不确定往一个函数中传入多少参数的时候，我们可以使用*args（单星号）。如果我们不知道往函数中传递多少个关键词参数的时候，我们可以使用**kwargs（双星号），args、kwargs两个标识符是约定俗成的命名而已。

2. 字典有序吗，你用过字典、列表推导式吗?

> - 字典是无序的；【集合也是无序的，列表、字符串和元组是有序的】
> - 字典是一种映射关系，集合是一个没有重复元素的空间，而列表、元组和字符串维护的是一个有顺序的关系；
> - 字典的key必须是可哈希化的，所谓的可哈希化就是指该内容不可改变了，比如整数，字符串都是不变的，所以用它来作为字典的key是可以的，而字典的value是可以任意类型都可以的。
> - 使用推导式的话，一般就是用作产生列表或字典空间的，它最方便的是可以用一行代码解决条件过滤情况下的数据产生，比如在做水纹监测系统时，设备端传来的数据，经常会出现一些无效数据，后来发现是硬件问题，硬件修改难度比较大了，所以就利用软件来做过滤，因为这个硬件是第三方提供的，所以就让我们后台来把数据做了下过滤，我就是用的列表推导式过滤，代码比较好维护。

3. 什么是PEP8，你用过哪些这个规范?

> - PEP，是python规范加强的规则文档，PEP8则是python规范加强中，针对python编码格式的规范说明。
> - PEP8，主要用到tab缩进，PEP8是建议用4个空格来代表缩进。这个我的印象最深，因为linux默认的vi的tab是8个空格，所以在部署到linux有时候用vi修改代码的时候，容易出现问题。后面使用vim并且设置了下，就可以了。
> - 空行，函数和函数之间要2个空行

- 每行都有最大长度限制，具体记不清了，好像是128个字符吧。
- 空格的使用；

4. 什么是python的命名空间，谈谈python变量的读写操作?

- 在python中，变量值和变量名，是以字典结构的K-V对进行维护的。这些保存KV对的空间，就称之为命名空间。
- 当python读取一个变量的值时，会按照LEGB原则进行空间的查找，也就是说先在局部空间查找有没有这个key值，如果有，则读取，若没有，再向上一级空间查找，依次类推。若都没有，才会抛出异常。
- 不过python在写一个变量时，只会在Local空间操作，当空间没有该变量时，就会像字典那样，创建一个新的变量，如果想操作全局空间，则必须借助global关键字。

5. 什么是python的赋值，拷贝?

- python的变量都是字典结构维护的。赋值，实际是对value值增加一个key值，没有内存的拷贝。也可以称之为引用。
- 而拷贝是新建一个value值。虽然值是一样的，但是他们有不同的id值。
- 拷贝分为浅拷贝和深拷贝，默认python的操作都是浅拷贝，比如列表切片，实例化新列表，字典；而深拷贝必须借助python的copy模块下的deepcopy方法。
- copy.copy：浅拷贝，只拷贝父对象，不拷贝父对象的子对象。
- copy.deepcop y： 深拷贝，拷贝父对象和子对象。

6. Python中如何生成随机数? 随机整数?

使用random模块

random.random() 生成（0,1）之间的随机实数

random.randint(start,end) 生成随机整数，必须指定范围

random.randrange(start, end, step) 在范围内按照step步进随机取值

random.uniform(1, 10) 生成范围内的随机实数

- 随机数原理

  random是用一个熵池的概念来获取随机数的，如果熵池固定，那么每次运行代码，获取的数字都是一样的，所以需要每次运行时，都取一个新的熵池，最好的方案就是初始化熵池时候，传递当前时间，因为当前时间每次运行都不会一样。

7. 什么是pass，什么是lambda函数

- pass语句不会执行任何操作，一般作为占位符或者创建占位程序 。
- lambda 函数也叫匿名函数。适用于需要一个函数，但是又不想费神去命名一个函数的场合。
- lambda 只是一个表达式，相比较来说函数体比def简单很多。
- 比如在排序，map，reduce这些高阶函数里，可以一行写完，不过这个也要看情况，过多使用匿名函数，在团队代码阅读上，可能会有麻烦。

8. 大数据的文件的读取，文本文件和二进制文件

- 可以通过生成器，分多次读取，每次读取数量相对少的数据（比如 100MB）进行处理，处理结束后 在读取后面的 100MB 的数据。

```
def get_larger_file(filename, block=100*1024*1024):
    with open(filename, 'rb') as fp:
        data = fp.read(block)
        while data:
            yield data
            data = fp.read(block)
larger_file = get_larger_file('a.log')
for da in larger_file:
    do_things(da)
```

- 如果是文本文件，可以一行行的读，可迭代对象 file，进行迭代遍历会自动地使用缓冲 IO （ buffered IO）以及内存管理， 而不必担心任何 大文件的问题。

```
with open('filename', 'r', encoding='utf-8') as fp:
    for line in fp:
        do_things(line)
```

- 所有文件里保存的都是二进制的字节，而二进制如果满足一定的编码表来表示字符的话，这种文件就称之为文本文件，常见的编码方式有gbk,utf-8。

9. 哈希算法

哈希运算，又叫散列运算，这个算法的特点是：

- 任意长度的数据，经过哈希算法后，将会得到一个固定长度的输出，常用的MD5得到的是128bit的结果，一般使用32个字符的16进制字符表示，而sha256输出256bit的结果，使用64个16进制字符。
- 当输入的数据哪怕有一个bit不一样，最终的结果将会完全不一样。
- 通过输出结果，不能倒推出原数据。

哈希的应用场景：

- 用户密码保存，用户登陆时的密码校验
- 哈希时，要考虑加盐的问题

10. 时间模块

- python的时间底层原理是保存的从1970年1月1日开始的秒数，使用time模块来维护这个秒数的情况。
- 为了更方便管理时间日期，python提供了一个更高级的模块datetime模块
- 一般使用strftime，把对象转换为字符串，使用strptime，把字符串转换为对象。

11. _ _ init _ _ 和 _ _ new _ _是什么?

- 在python的面向对象中，利用new方法创建对象，然后把这个创建的对象传递到init函数中，以第一个参数进行传递。
- 一般情况下，对象的名称以self命名，类的名称以cls命名。
- 一般使用init方法就可以进行对象属性的处理了，而new方法，一般在单例模式设计中，我用过。

单例模式：

所谓单例就是一个类从始至终只能产生一个实例。

12. 静态方法，类方法，对象方法是什么，什么时候使用?

- 在python的类中定义的函数，和普通函数在调用过程中，有很大的区别，普通函数直接用名字就可以调用，而类中的函数，必须通过类名或对象名.函数名的方式才能被调用。
- 这种调用方式，默认把调用者作为第一个参数传递到函数内。类作为调用者命名为cls，对象作为调用者命名为self。
- 但是有一种方法，即不归对象管，也不归类管，也就是说他的代码不依赖类或对象的任何属性，仅仅是以封装的原因写在类里了，那这种就叫做静态方法，但是从语法形式上，还是要写成类或对象.的形式，所以为了不让他传递调用者信息，python中使用了@staticmethod装饰器，这样函数申明时，第一个参数就可以不写self或cls了。
- 对象也可以调用类方法，但是第一个参数还是要传递该对象的类地址，这样的话，为了让传递的信息不是对象地址而是类地址的话，就要使用@classmethod装饰器来告诉调用时，使用类对象地址作为第一个参数的赋值。
- 默认情况都是对象方法。

13. 常用魔术方法

- 在Python中，所有以双下划线_ _包起来的方法，都统称为魔术方法，例如类的初始化方法 _ _ init _ _。
- 基本定制类：

```
C.__init__(self[, arg1, ...])       对象初始化构造器
C.__new__(self[, arg1, ...])        对象创建构造器
C.__del__(self)                     对象销毁时被调用
C.__str__(self)                     可打印的字符输出，被str化时调用，如print(obj)
C.__repr__(self)                    交互模式下的输出内容
C.__call__(self, *args)             对象调用时，被执行，如p1()
```

- 属性操作类：

```
C.__getattr__(self, attr)           获取属性；仅当属性没有找到时调用
C.__setattr__(self, attr, val)      设置属性
C.__delattr__(self, attr)           删除属性
```

  - 注意事项：这几个属性操作很容易出现递归调用

```
#  错误用法
def __setattr__(self, name, value):
    self.name = value
#  正确用法
def __setattr__(self, name, value):
    self.__dict__[name] = value  # 给类中的属性名分配值
```

- 容器操作类：

```
C.__len__(self)                     容器的数量，当调用len()时被执行
C.__getitem__(self, ind)            得到元素ind索引的值，p[ind]的操作
C.__setitem__(self, ind,val)        设置元素ind索引的值
C.__delitem__(self, ind)            删除对应ind索引的元素
```

- 功能类：

```
__iter__(self)                  产生迭代器
__enter__(self)                 with的入口代码
__exit__(self, *args)           with语句块的退出时执行代码
```

- 参考文档

  python魔术方法1

  python魔术方法2

14. 怎么理解装饰器，用过或者写过什么装饰器吗?

- 装饰器本质上是一个 Python 函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外 功能，装饰器的返回值也是一个函数对象。
- 一般装饰器的使用场景：日志记录，统计代码执行时间，权限验证。
- 通用装饰器写法：利用functools模块里提供的wrap装饰器，他可以保留原函数的元信息（文档，函数名称等）

```
from functools import wraps

def user_token(fn):
    @wraps(fn)
    def decorator(*args, **kwargs):
        ...
        ret = fn(*args, **kwargs)
        return ret
    return decorator
```

15. 迭代器和生成器

- 迭代器实际是一个特殊类（鸭子），含有_ _ iter _ _方法和next方法。
- python通过iter()方法得到迭代器，然后通过next()方法不断从迭代器中获取数据，当收到stopIteration错误时，说明迭代器里没有数据了。
- python中可以通过for... in语句，来完成上述操作，in后面的返回对象，会调用这个类对象的iter()，然后通过next()不断从in中的对象内容。
- 生成器（Generator）是创建迭代器的简单而强大的工具。它们写起来就像是正规的函数，只是在需要返回数 据的时候使用 yield 语句。每次 next()被调用时，生成器会返回它脱离的位置。

16. 编码问题

- ascii是计算机最基本的编码表，不过它只编码了英语系国家的字母和标点符号和控制字符这些。
- gbk又叫国标码表，是针对中国简体字体设计的编码方式，一般一个汉字使用2个字节来描述。
- unicode又叫国际统一编码表，目的是把整个世界的符号都编码进去，使用2个字节对一个字符进行编码，共16张这个编码表，目前使用的是第0张表，然后emoj表情目前使用的是后面的表。
- 而UTF-8属于编码方式，它的目的是把unicode中出现频繁的ascii标准字符还是用1个字节来表示，而频率较低的字符使用3个字节代表，解决编码冗余的问题。

17. cookie、session和token

- 由于HTTP协议是无状态的，所以为了解决服务端如何识别每次客户请求中，该请求究竟是哪个客户，或者说就是客户身份的问题，提出了一个cookie的技术。cookie实际是一个保存在浏览器上的字符串，具有K-V

对格式的。他属于浏览器行为的技术，每次浏览器在发送请求到服务器时，会根据域名，路径信息来决定是否将保存在浏览器的cookie发送给服务器。

- 服务器通过set-cookie的HTTP响应，向客户端颁发凭证，每次客户端向服务器发送请求时，都会携带该凭证，那么服务器也就知道该请求究竟是哪个客户了。
- 由于cookie里保存的数据在客户端是可以直接查看的，所以这样一些隐私数据就直接暴露出来，不那么安全，为了解决这种安全问题，服务器提出一个session技术，就是将所有该用户的隐私数据全部保存在服务器端，然后服务器只需要把这些数据集的ID号通过一个加密算法后的结果发给客户端的cookie，这样，客户端里只保存了这个加密的ID号，而没有隐私数据。不过session技术还是必须依靠cookie才能实现HTTP状态识别。
- token技术主要是解决app和服务器状态的识别问题，由于app开发中，没有浏览器这种自动发送cookie的机制，那么服务器为了解决客户端身份识别问题，就可以为客户端也颁发一个凭证，让客户端访问服务器特定接口时，也携带这个凭证，那么这个凭证就称之为token。
- token可以通过HTTP请求头的字段，GET请求体，POST请求体中携带，相关的规范根据客户和服务器的约定来执行。

18. TCP/IP模型

TCP/IP模型：

从下到上： 物理接口层，网络层，传输层，应用层

物理接口层解决物理信号和信息差错问题；

网络层解决计算机身份识别的问题；

传输层解决数据包无丢失的传输；

应用层解决应用协议数据格式的解析；

19. HTTP响应状态码

见flask-rest接口开发.docx课件的3.2.3

20. Linux常见命令和权限

无标准答案，只要命令功能说明没问题即可

21. Web服务器和Web框架

- Web服务器是指实现了HTTP协议的网络服务器的统称，常见的如apache或Nginx都属于web服务器。
- Web框架是指在收到特定的HTTP请求后，对于该请求，如何通过程序逻辑进行处理，然后在通过web服务器把处理结果以HTTP响应发送给客户端的程序结构。
- web服务器关注的是数据的接收和发送功能。
- web框架关注的是收到HTTP请求后，如何管理和处理数据。

22. RESTful接口的理解

- 首先restful是一种软件架构风格或者说是一种设计风格，并不是标准，它只是提供了一组设计原则 和约束条件，主要用于客户端和服务器交互类的软件。
- 就像设计模式一样，并不是一定要遵循这些原则，而是基于这个风格设计的软件可以更简洁，更有层 次，我们可以根据开发的实际情况， 做相应的改变。
- 它里面提到了一些规范，例如：

- 1、在url接口中推荐使用Https协议，让网络接口更加安全（Https是Http的安全版，即HTTP下加入 SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL（安全套接层协议））
- 2、url中可以体现这是个API接口
- 3、url中还可以体现版本号，不同的版本可以有不同的接口，使其更加简洁，清晰
- 4、restful 提倡面向资源编程，所以在url接口中尽量要使用名词，不要使用动词
- 5、此外url中还可以添加条件去筛选匹配 6、可以根据Http不同的method，进行不同的资源操作（5种方法：GET / POST / PUT / DELETE / PATCH）

23. Mysql存储引擎

- 1.Innodb引擎 支持事务 支持锁 行锁和表锁：但是当SQL语句没有指定要锁定的具体行范围的话，Innodb 也会锁全表。 支持外键约束
- 2.MyIASM引擎 MyIASM是MySQL默认的引擎，不支持事务，也不支持行级锁和外键，只支持表级锁。
- 两种引擎的比较： 大尺寸的数据集趋向于选择InnoDB引擎，因为它支持事务处理和故障恢复。
- 数据库的大小决 定了故障恢复的时间长短，InnoDB可以利用事务日志进行数据恢复，这会比较快。主键查询 在InnoDB引擎下也会相当快，不过需要注意的是如果主键太长也会导致性能问题。大批的 INSERT语句(在每个INSERT语句中写入多行，批量插入)在MyISAM下会快一些，但是 UPDATE语句在InnoDB下则会更快一些，尤其是在并发量大的时候。