Python 模拟面试技术题及答案

一、python 语法

1. 请说一下你对迭代器和生成器的区别？

答:(简单理解)

迭代器和生成器都是 Python 中特有的概念，迭代器可以看作是一个特殊的对象，每次调用该对象时会返回自身的下一个元素，从实现上来看，一个可迭代的对象必须是定义了__iter__()方法的对象，而一个迭代器必须是定义了__iter__()方法和 next()方法的对象。生成器的概念要比迭代器稍显复杂，因为生成器是能够返回一个迭代器的函数，其最大的作用是将输入对象返回为一个迭代器。Python 中使用了迭代的概念，是因为当需要循环遍历一个较大的对象时，传统的内存载入方式会消耗大量的内存，不如需要时读取一个元素的方式更为经济快

捷。

（1）迭代器是一个抽象的概念，任何对象，如果它的类有 next 方法和 iter 方法返回自己本身都可以认为他是迭代器。对于 string、list、dict、tuple 等这类容器对象，使用 for 循环遍历是很方便的。在 for 语句对容器对象调用 iter()函数，(iter()是 python 的内置函数),iter()会返回一个定义了 next()方法的迭代器对象，它在容器中逐个访问容器内元素(next()也是 python 的内置函数),在没有后续元素时，next()会抛出一个 StopIteration 异常

（2）生成器（Generator）是创建迭代器的简单而强大的工具。它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，生成器都会返回它脱离的位置（它记忆语句最后一次执行的位置和所有的数据值）．简单的说就是在函数的执行过程中，yield 语句会把你需要的值返回给调用生成器的地方，然后退出函数，下一次调用生成器函数的时候又从上次中断的地方开始执行，而生成器内的所有变量参数都会被保存下来供下一次使用。

区别：生成器能做到迭代器能做的所有事,而且因为自动创建了__iter__()和 next()方法,生成器显得特别简洁,而且生成器也是高效的，使用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法,当发生器终结时,还会自动抛出 StopIteration 异常

2. 什么是线程安全?

答:

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。

但有的时候多线程运行时,会同时竞争同一个共享资源并对他进行修改,这个时候由于线程之间竞争的问题就会造成修改错误,会产生多线程运行的结果和单线程的运行结果不一致的问题.

解决方法:

当多个线程几乎同时修改某一个共享数据的时候，需要进行同步控制 线程同步能够保证多个线程安全访问竞争资源，最简单的同步机制是引入互斥锁。互斥锁为资源引入一个状态：锁定/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。

3. socket 的 connect 方法工作原理是什么?

答:

网络编程 socket api 存在一批核心接口，而这一批核心接口就是几个看似简单的函数，尽管实际上这些函数没有一个是简单。connect 函数就是这些核心接口的一个函数，它完成主动连接的过程。connect 函数的功能是完成一个有连接协议的连接过程，对于 TCP 来说就是那个三路握手过程.它的函数原型：connect(address) 连接到 address 处的套接字。一般 address 的格式为元组（hostname,port），如果连接出错，返回 socket.error 错误。

connect 函数的功能:

connect 函数的功能可以用一句话来概括，就是完成面向连接的协议的连接过程，它是主要用于连接的。面向连接的协议如 TCP，在建立连接的时候总会有一方先发送数据，那么谁调用了 connect 谁就是先发送数据的一方。如此理解 connect 的参数就容易了，我必需指定数据发送的地址，这正好是 connect 的参数的作用.

参数元组（hostname,port）

指定数据发送的目的地，也就是服务器端的地址。这里服务器是针对 connect 说的，因为 connect 是主动连接的一方调用的，所以相应的要存在一个被连接的一方，被动连接的一方需要调用 listen 以接受 connect 的连接请求，如此被动连接的一方就是服务器了。与所有的 socket 网络接口一样，connect 总会在某个时候可能失败，此时它会返 socket.error,常见的错误有对方主机不可达或者超时错误，也可以是对方主机没有相应的进程在对应端口等待。

4. Python 中怎么简单的实现列表去重?

答:Python 中的列表去重方法有很多,下面有两种常用的简单方式

方法一:转换为集合数据类型, set(列表)

```
li = [1, 2, 3, 4, 1, 2]
s = set(li)
li = list(s)
print li
#[1, 2, 3, 4]
```

解释:列表中的元素可以重复,在集合中元素是不可以重复的,这是集合的特性.利用这个特性,我们可以把列表强制转化成集合,使得原来列表中重复的元素去重.但是注意这种方法有可能使得原有的列表中的数据顺序发生改变.

方法二:不改变原有列表顺序的方法

```
list2 = []
list1 = [1,2,3,2,2,2,4,6,5]
for i in list1:
    if i not in list2:
        list2.append(i)
list2
[1, 2, 3, 4, 6, 5]
```

解释:通过创建一个新的列表 list2 存贮去重后的数据,如以上代码显示,对 list1 中的数据依次进行 for 循环遍历.如果 list1 中的数据在 list2 中没有则添加到 list2

中,如果 list2 中有该数据则不添加,如此直到对 list1 中数据遍历完成就可以使得 list2 中存储保持原有顺序且不重复的数据.

5. python 中 yield 的用法?

答:

通常的 for...in...循环中,in 后面是一个数组,这个数组就是一个可迭代对象,类似的还有列表,字符串,文件。它可以是 mylist = [1, 2, 3],也可以是 mylist = [x*x for x in range(3)]。 它的缺陷是所有数据都在内存中,如果有海量数据的话将会非常耗内存。

而生成器就不是,他不需要把所有的数据都存放在内层中,而是当使用到的时候再去获取.这是由于生成器是可以迭代的,生成器(generator)能够迭代的关键是它有一个 next()方法,工作原理就是通过重复调用 next()方法,不断的获取下一数据,而不是一次性把所有的数据都存放到内存中,这样重复操作直到捕获一个异常才会停止。比如 mygenerator = (x*x for x in range(3)),注意这里用到了() 这里相当于一个函数,它就不是数组,而上面的例子都是 [ ] 列表。

而带有 yield 的函数就跟我们举的例子 mygenerator = (x*x for x in range(3))一样,不再是一个普通函数,而是一个生成器 generator,可用于迭代,工作原理同上。 yield 是一个类似 return 的关键字,迭代一次遇到 yield 时就返回 yield 后面的值。重点是:下一次迭代时,从上一次迭代遇到的 yield 后面的代码开始执行。

简要理解:yield 就是 return 返回一个值,并且记住这个返回的位置,下

次迭代就从这个位置后开始。

6. 什么是面向对象编程?

面向对象的编程产生的原因：由于面向过程编程在构造系统时，无法解决重用，维护，扩展的问题，而且逻辑过于复杂，代码晦涩难懂，因此，人们开始想能不能让计算机直接模拟现实的环境，以人类解决问题的方法，思路，习惯和步骤来设计相应的应用程序。于是，面向对象的编程思想就产生了。

面向对象的编程的主要思想是把构成问题的各个事物分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述一个事物在解决问题的过程中经历的步骤和行为。对象作为程序的基本单位，将程序和数据封装其中，以提高程序的重用性，灵活性和可扩展性。

类是创建对象的模板，一个类可以创建多个对象。对象是类的实例化。 类是 抽象的，不占用存储空间,而对象是具体的，占用存储空间。 面向对象有三大特性：封装，继承，多态。

7. 谈谈你对 GIL 锁对 python 多线程的影响?

答:

GIL 的全称是 Global Interpreter Lock(全局解释器锁)，来源是 python 设计之初的考虑，为了数据安全所做的决定。每个 CPU 在同一时间只能执行一个线程（在单核 CPU 下的多线程其实都只是并发，不是并行，并发和并行从宏观

上来讲都是同时处理多路请求的概念。但并发和并行又有区别，并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔内发生。）

在 Python 多线程下，每个线程的执行方式：

1、获取 GIL

2、执行代码直到 sleep 或者是 python 虚拟机将其挂起。

3、释放 GIL

可见，某个线程想要执行，必须先拿到 GIL，我们可以把 GIL 看作是“通行证”，并且在一个 python 进程中，GIL 只有一个。拿不到通行证的线程，就不允许进入 CPU 执行。

GIL 锁存在下多线程的使用场景:

1.密集型科学运算(CPU 没有闲置)

由于 GIL 锁保证同一时刻只有一个线程可以使用 CPU,这使得多线程之间会对 GIL 锁进行争夺,争夺的过程中谁获取了 GIL 锁谁就可以使用 CPU,这种争夺状态会一直持续,这就造成了资源的浪费,浪费在了抢夺资源上,在这种情况下我们会发现如果只是用一个线程反而会避免这种资源的竞争,同时提高 CPU 的使用率,所以在科学计算这种没有 CPU 闲置的情况下我们倾向于使用单线程.

2.I/O 操作(有 CPU 闲置)

像爬虫,网络请求这类操作往往受到网速等原因的限制,会造成一个线程占用 CPU 的同时等待网络数据的获取,这个时候 GIL 锁会自动判定如果线程使用的 CPU 闲置他就会自动切换到另外一个线程,由此可以节省线程占用 CPU 阻塞等

待网络请求的时间.所以在像网络操作这种有 CPU 闲置的情况下我们倾向使用多线程

8. python 是如何进行内存管理的?

答:(基本理解:足够面试使用)

一、垃圾回收：python 不像 C++，Java 等语言一样，他们可以不用事先声明变量类型而直接对变量进行赋值。对 Python 语言来讲，对象的类型和内存都是在运行时确定的。这也是为什么我们称 Python 语言为动态类型的原因（这里我们把动态类型可以简单的归结为对变量内存地址的分配是在运行时自动判断变量类型并对变量进行赋值）。

二、引用计数：Python 采用了类似 Windows 内核对象一样的方式来对内存进行管理。每一个对象，都维护这一个对指向该对对象的引用的计数。当变量被绑定在一个对象上的时候，该变量的引用计数就是 1，(还有另外一些情况也会导致变量引用计数的增加),系统会自动维护这些标签，并定时扫描，当某标签的引用计数变为 0 的时候，该对就会被回收。

(深入理解:加深对内存机制的理解)

三、内存池机制 Python 的内存机制如同金字塔一般，

第 0 层是 C 语言中的 malloc(c 语言中申请内存空间的方法)，free(C 语言中释放内存的方法)等内存分配和释放函数进行操作；

第 1 层和第 2 层是内存池，有 Python 的接口函数 PyMem_Malloc 函数实

现，当对象小于 256K 时有该层直接分配内存；

第 3 层是最上层，也就是我们对 Python 对象的直接操作；

在 C 中如果频繁的调用 malloc 与 free 时,是会产生性能问题的.再加上频繁的分配与释放小块的内存会产生内存碎片. Python 在这里主要干的工作有:

如果请求分配的内存在 1~256 字节之间就使用自己的内存管理系统,否则直接使用 malloc.

这里还是会调用 malloc 分配内存,但每次会分配一块大小为 256k 的大块内存.

经由内存池登记的内存到最后还是会回收到内存池,并不会调用 C 的 free 释放掉.以便下次使用.对于简单的 Python 对象，例如数值、字符串，元组（tuple 不允许被更改）采用的是复制的方式，也就是说当将另一个变量 B 赋值给变量 A 时，虽然 A 和 B 的内存空间仍然相同，但当 A 的值发生变化时，会重新给 A 分配空间，A 和 B 的地址变得不再相同

9. 请描述线程、进程、协程的优缺点

进程、线程和协程是三个在多任务处理中常听到的概念，三者各有区别又相互联系。

(1)进程

进程是一个程序在一个数据集中的一次动态执行过程，可以简单理解为“正在执行的程序”，它是 CPU 资源分配和调度的独立单位。 进程一般由程序、数

据集、进程控制块三部分组成。我们编写的程序用来描述进程要完成哪些功能以及如何完成；数据集则是程序在执行过程中所需要使用的资源；进程控制块用来记录进程的外部特征，描述进程的执行变化过程，系统可以利用它来控制和管理进程，它是系统感知进程存在的唯一标志。

进程的缺点:　创建、撤销和切换的开销比较大。

(2)线程

线程是在进程之后发展出来的概念。 线程也叫轻量级进程，它是一个基本的 CPU 执行单元，也是程序执行过程中的最小单元，由线程 ID、程序计数器、寄存器集合和堆栈共同组成。一个进程可以包含多个线程。

线程的优点:　减小了程序并发执行时的开销，提高了操作系统的并发性能.

缺点的缺点:　线程没有自己的系统资源，只拥有在运行时必不可少的资源，但同一进程的各线程可以共享进程所拥有的系统资源，如果把进程比作一个车间，那么线程就好比是车间里面的工人。不过对于某些独占性资源存在锁机制，处理不当可能会产生“死锁”。

(3)协程

协程是一种用户态的轻量级线程，又称微线程，英文名 Coroutine，协程的调度完全由用户控制。人们通常将协程和子程序（函数）比较着理解。 子程序调用总是一个入口，一次返回，一旦退出即完成了子程序的执行。 协程的起始处是第一个入口点，在协程里，返回点之后是接下来的入口点。在 python 中，

协程可以通过 yield 来调用其它协程。通过 yield 方式转移执行权的协程之间不是调用者与被调用者的关系，而是彼此对称、平等的，通过相互协作共同完成任务。其运行的大致流程如下： 第一步，协程 A 开始执行。 第二步，协程 A 执行到一半，进入暂停，通过 yield 命令将执行权转移到协程 B。 第三步，（一段时间后）协程 B 交还执行权。 第四步，协程 A 恢复执行。 1 2 3 4 协程的特点在于是一个线程执行，

与多线程相比，其优势体现在： 协程的执行效率非常高。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。 协程不需要多线程的锁机制。在协程中控制共享资源不加锁，只需要判断状态就好了。

提示:利用多核 CPU 最简单的方法是多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能.

10. 深拷贝和浅拷贝的区别

答: 通俗的说 浅拷贝只拷贝引用(也就是地址)，深拷贝就是拷贝具体的值，重新开辟新的空间存储这些值。就像是浅拷贝就是你的影子,深拷贝是你的克隆人，你没了影子也就没了,但是克隆人还活着。

深拷贝和浅拷贝有很多不同的情况,不同情况下都有不同的特点

1 拷贝单一的可变数据类型的时候 如 a = [1,2] 这种单一的列表

深拷贝和浅拷贝都是一样的,都会开辟新的一片新空间,存储数据,保证数据的

独立性

2 拷贝复杂的有嵌套数据类型 如 a = [1,2],b = [3,4],c = [a,b]拷贝 c 的时候

d = copy.copy(c)

浅拷贝:只会给 d 开辟一片空间 存贮 c 中的数据,也就是存储 a,b 的引用, 而并不是开辟多个空间去存储 a 中 1,2 数据,b 中的 3,4 数据,这样以来 a 如果发生变化 d 也会跟着发生变化,没有办法保证数据的独立性.

d = copy.deepcopy(c)

深拷贝:会给 d 开辟多片空间,存储所有的数据,可以保证独立空间,保证数据的独立性

3 拷贝不可变数据类型 如元组这样的类型

深拷贝和浅拷贝都一样,因为不可变类型数据不会变化,所以不需要保证数据的独立性,直接引用这个数据就可以,而不会开辟新的空间.

提示:ptyhon 中大多情况都是浅拷贝

二、Linux 基础和数据结构与算法

11. 10 个常用的 Linux 命令?

b) 答案：略

2. find 和 grep 的区别?

a) grep 命令是一种强大的文本搜索工具，grep 搜索内容串可以是正则表达式，允许对文本文件进行模式查找。如果找到匹配模式， grep 打印包含模式的所有行。

b) find 通常用来在特定的目录下搜索符合条件的文件，也可以用来搜索特定用户属主的文件。

3. 什么是阻塞？什么是非阻塞？

阻塞调用是指调用结果返回之前，当前线程会被挂起。函数只有在得到结果之后才会返回。有人也许会把阻塞调用和同步调用等同起来，实际上他是不同的。对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前函数没有返回而已。例如，我们在 CSocket 中调用 Receive 函数，如果缓冲区中没有数据，这个函数就会一直等待，直到有数据才返回。而此时，当前线程还会继续处理各种各样的消息。如果主窗口和调用函数在同一个线程中，除非你在特殊的界面操作函数中调用，其实主界面还是应该可以刷新。socket 接收数据的另外一个函数 recv 则是一个阻塞调用的例子。当 socket 工作在阻塞模式的时候，如果没有数据的情况下调用该函数，则当前线程就会被挂起，直到有数据为止。

非阻塞和阻塞的概念相对应，指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。

4. 描述数组、链表、队列、堆栈的区别？

数组与链表是数据存储方式的概念，数组在连续的空间中存储数据，而链表可以在非连续的空间中存储数据；

队列和堆栈是描述数据存取方式的概念，队列是先进先出，而堆栈是后进先出；队列和堆栈可以用数组来实现，也可以用链表实现。

5. 你知道几种排序,讲一讲你最熟悉的一种？

排序方法	平均情况	最好情况	最坏情况	辅助空间	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
希尔排序	$O(n \log n)$~ $O(n^2)$	$O(n^{1.3})$	$O(n^2)$	O(1)	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	O(1)	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$~ $O(n)$	不稳定

6. 如果有一个文本 temp.txt 中有且只有一个 1.8，此时我们想将其更换为 2.0，请在不适用编辑器的前提下实现该功能

sed –i s#1.8#2.0# temp.txt

7. 如何使用两个栈结构实现一个队列

将数据压入一个栈，然后将其分别弹栈，并将弹栈数据压入另一个栈，然后另一个栈出栈

8. 如何检查最近十分钟内变动过的文件

find / -mmin -10

9. 在当前目录下查找大小小于 10m 的文件

find / -size -10M

10. 已知二叉树的先序:0 1 3 7 8 4 2 5 6,中序:7 3 8 1 9 4 0 5 2 6，求后序遍历顺序

7 8 3 9 4 1 5 6 2 0

三、Web 框架

1. Flask 请求上下文和应用上下文的区别和作用?

答：a)请求上下文中包含 request, session 两个变量，应用上下文中包含 current_app, g 两个变量

b)请求上下文：保存了客户端和服务器交互的数据。应用上下文：flask 应用程序运行过程中，保存的一些配置信息，比如程序名、数据库连接、应用信息等。

c)请求上下文的变量需要在有请求发生之后，处理请求的整个过程中才能够使用，应用上下文变量需要在当前 Flask 应用开启之后才能使用，并且可以手动开启一个 Flask 应用上下文

2. 说出 CSRF 攻击的原理和防范措施

a) 攻击原理：

i. 用户 C 访问正常网站 A 时进行登录，浏览器保存 A 的 cookie

ii. 用户 C 再访问攻击网站 B，网站 B 上有某个隐藏的链接或者图片标签会自动请求网站 A 的 URL 地址,例如表单提交，传指定的参数

iii. 而攻击网站 B 在访问网站 A 的时候，浏览器会自动带上网站 A 的 cookie

iv. 所以网站 A 在接收到请求之后可判断当前用户是登录状态，所以根据用户的权限做具体的操作逻辑，造成网站攻击成功

b) 防范措施：

i. 在指定表单或者请求头的里面添加一个随机值做为参数

ii. 在响应的 cookie 里面也设置该随机值

iii. 那么用户 C 在正常提交表单的时候会默认带上表单中的随机值，浏览器会自动带上 cookie 里面的随机值，那么服务器下次接受到请求之后就可以取出两个值进行校验

iv. 而对于网站 B 来说网站 B 在提交表单的时候不知道该随机值是什么，所以就形成不了攻击

3. cookie 和 session 工作机制，以及这两的区别是什么？

答：cookie：浏览器的一种数据缓存机制，可用于保存用户的登录状态，缓存用户相关操作数据。其工作机制可能用登录来进行描述为：

a) 用户在浏览器上点击登录，向服务器发起请求

b) 服务端接收到请求之后并处理登录请求，然后将当前用户的相关标识(例如 user_id) 设置到响应的 cookie 中去

c) 浏览器接收到响应之后，会自动保存 cookie 中的相关信息

d) 下次再次请求，浏览器会在请求中自动带上 cookie 信息，服务端接收到请求之后就能通过请求中的 cookie 信息判断当前是哪个用户进行请求

session：cookie 延伸出来的数据缓存机制，为了解决 cookie 中不能存储敏感信息等问题。工作机制是：

a) 用户在浏览器上点击登录，向服务器发起请求

b) 服务端接收到请求之后并处理登录请求，存储需要记录用户登录信息的相关数据到 session 中，并且使用一个 session_id 与该数据进行对应

c) 将 session_id 放在响应的 cookie 中进行响应

d) 浏览器接收到响应之后，会自动保存 cookie 中的相关信息，这样就会在本地

存储 session_id

e) 下次请求，浏览器会再 带上 cookie 中的 session_id 去请求，服务端接收到请求之后可以取到 session_id,再通过 session_id 取到第 2 步保存服务器的数据

f) 而对于 session_id，只是一个单纯的字符串，其所对应的数据存储在服务端所以比 cookie 安全

g) 而 session 的存储又可以与 redis 进行对接，专门用一个服务器跑 redis 服务，用于存储 session 数据，那么就可以解决 web 项目做了负载均衡之后，不同请求发到不同的服务器所造成的 session 丢失问题。

区别：数据存储位置，安全性，数据存储大小

4. 请求勾子在项目中的应用场景

a) before_first_request

i. 在第一次请求之前调用

ii. 可以做一些初始化操作

b) before_request

i. 可以对用户的权限作校验

ii. 可以对接口请求权限做校验

iii. 可以对非法请求做校验

iv. 可以直接返回响应

c) after_request

i. 可以统一的响应做一些值的修改或者添加，再返回给客户端

d) teardown_reqeust

i. 在请求结束之后会调用，如果当前请求有错误抛出，可以在些统一进行错误处理

e) 类似于 Django 的中间件

5. 请解释或描述一下 Django 的架构

对于 Django 框架遵循 MVC 设计，并且有一个专有名词：MVT

M 全拼为 Model，与 MVC 中的 M 功能相同，负责数据处理，内嵌了 ORM 框架

V 全拼为 View，与 MVC 中的 C 功能相同，接收 HttpRequest，业务处理，返回 HttpResponse

T 全拼为 Template，与 MVC 中的 V 功能相同，负责封装构造要返回的 html，内嵌了模板引擎

6. 你对 Django 的认识?

答：Django 是走大而全的方向，它最出名的是其全自动化的管理后台：只需要使用起 ORM，做简单的对象定义，它就能自动生成数据库结构、以及全功能的管理后台。

Django 内置的 ORM 跟框架内的其他模块耦合程度高。

应用程序必须使用 Django 内置的 ORM，否则就不能享受到框架内提供的种种基于其 ORM 的便利；理论上可以切换掉其 ORM 模块，但这就相当于要把装修完毕的房子拆除重新装修，倒不如一开始就去毛胚房做全新的装修。

Django 的卖点是超高的开发效率，其性能扩展有限；采用 Django 的项目，在流量达到一定规模后，都需要对其进行重构，才能满足性能的要求。

Django 适用的是中小型的网站，或者是作为大型网站快速实现产品雏形的工具。

Django 模板的设计哲学是彻底的将代码、样式分离； Django 从根本上杜绝在模板中进行编码、处理数据的可能

7. 前后端分离和前后端不分离是什么，各有什么优缺点?

答：是两种不同的开发模式；

前后端分离：开发服务端的人员只关心向前端提供数据，不关心数据的获取和具体如何展示，具体表现为：

a) 浏览器在访问一个页面的时候，会发起至少两类请求：1 是请求前端页面，2 是请求页面数所需要数据

b) 后端只需要使用接口响应页面上所需要的数据，接口一般要遵守 RESTful 风格，数据格式一般为 JSON 格式

c) 前端开发人员开发前端界面，并使用 JS 等代码向后端提供的接口发起请求，以获取数据，并将数据填充到界面

d) 各自己只关心自己所负责业务的逻辑

优点：开发效率高，后续扩展性强。

缺点：不利于做 SEO 优化

前后端不分离：

a) 浏览器在访问一个页面的时候，服端给出来的响应就是一个完整的 html 界面，不仅包含 html 静态的内容，还包含界面中需要显示的数据

b) 后端在返回的数据的时候，需要自行去查询数据并将数据渲染模板中，然后将渲染完成的数据响应给浏览器

优点：对于 SEO 支持性较好

缺点：数据和界面的耦合度高，扩展性较差，如果后续还有不同类型的客户端，还要单独写一套数据提供接口，

8. 说说 Django 中间键几个关键的函数，及其作用?

答:

a) process_request(self, request) Request 预处理函数

这个方法的调用时机在 Django 接收到 request 之后，但仍未解析 URL 以确定应当运行的 view 之前。Django 向它传入相应的 HttpRequest 对象，以便在方法中修改。

process_request() 应当返回 None 或 HttpResponse 对象。

如果返回 None ，Django 将继续处理这个 request，执行后续的中间件， 然后调用相应的 view。

如果返回 HttpResponse 对象，Django 将不再执行任何其它的中间件(无视其种类)以及相应的 view。 Django 将立即返回该 HttpResponse。

b) process_view(self, request, callback, callback_args, callback_kwargs) View 预

处理函数

这个方法的调用时机在 Django 执行完 request 预处理函数并确定待执行的 view 之后，但在 view 函数实际执行之前。

process_view() 应当返回 None 或 HttpResponse 对象。如果返回 None ，Django 将继续处理这个 request ，执行后续的中间件， 然后调用相应的 view。如果返回 HttpResponse 对象，Django 将不再执行 任何 其它的中间件(不论种类)以及相应的 view，Django 将立即返回。

c) process_response(self, request, response) Response 后处理函数

这个方法的调用时机在 Django 执行 view 函数并生成 response 之后。

该处理器能修改 response 的内容；一个常见的用途是内容压缩，如 gzip 所请求的 HTML 页面。

process_response() 必须返回 HttpResponse 对象. 这个 response 对象可以是传入函数的那一个原始对象（通常已被修改），也可以是全新生成的。

d) process_exception(self, request, exception) Exception 后处理函数

这个方法只有在 request 处理过程中出了问题并且 view 函数抛出了一个未捕获的异常时才会被调用。这个钩子可以用来发送错误通知，将现场相关信息输出到日志文件，或者甚至尝试从错误中自动恢复。

这个函数的参数除了一贯的 request 对象之外，还包括 view 函数抛出的实际的异常对象 exception 。

process_exception() 应当返回 None 或 HttpResponse 对象。

如果返回 None ， Django 将用框架内置的异常处理机制继续处理相应 request

如果返回 HttpResponse 对象，Django 将使用该 response 对象，而短路框架

内置的异常处理机制。

9. uWSGI 和 Nginx 的作用

答：uWSGI 是一个 Web 服务器

它实现了 WSGI 协议、uwsgi、http 等协议。注意 uwsgi 是一种通信协议，而 uWSGI 是实现 uwsgi 协议和 WSGI 协议的 Web 服务器。uWSGI 具有超快的性能、低内存占用和多 app 管理等优点

要注意 WSGI / uwsgi / uWSGI 这三个概念的区分。

WSGI 是一种通信协议。

uwsgi 是一种线路协议而不是通信协议，在此常用于在 uWSGI 服务器与其他网络服务器的数据通信。

uWSGI 是实现了 uwsgi 和 WSGI 两种协议的 Web 服务器。

nginx 是一个开源的高性能的 HTTP 服务器和反向代理：

1.作为 web 服务器，它处理静态文件和索引文件效果非常高；

2.它的设计非常注重效率，最大支持 5 万个并发连接，但只占用很少的内存空间；

3.稳定性高，配置简洁；

4.强大的反向代理和负载均衡功能，平衡集群中各个服务器的负载压力应用。

10. django 开发中数据库做过什么优化?

1.设计表时，尽量少使用外键，因为外键约束会影响插入和删除性能；

2.使用缓存，减少对数据库的访问；

3.在 orm 框架下设置表时，能用 varchar 确定字段长度时，就别用 text；

4.可以给搜索频率高的字段属性，在定义时创建索引；

5.Django orm 框架下的 Querysets 本来就有缓存的；

6.如果一个页面需要多次连接数据库，最好一次性取出所有需要的数据，减少对数据库的查询次数；

7.若页面只需要数据库里某一个两个字段时，可以用 QuerySet.values()；

8.在模板标签里使用 with 标签可以缓存 Qset 的查询结果。

四、爬虫和数据库

1. scrapy 模块有实现去重么？如何实现的，如果不需要其去重该怎么办？

1) scrapy 在当前爬虫运行的过程中会去重，即请求过的 url 地址，在当前运行的过程中是不会重复请求的

2) 去重的实现：使用 sha1 加密了请求方法，请求体和请求的 url 地址，使用机密结果的 16 进制字符串作为指纹，存储在内存的集合中;后续新来了请求，同样的方式生成指纹，判断该指纹是否存在集合中，如果存在则表示请求重复

3) 不去重：可以在构造请求的时候参加参数 don_filter=True，可以让该请求能够反复被请求

2. 如果页面的数据是动态加载的该如何获取，有哪些方法

动态加载的数据的获取方法和普通的页面数据获取没有太大的区别

1. 通过抓包，获取加载数据的地址，分析其 url 规律和参数规律后发送使用发送请求

2. 使用 selenium 驱动浏览器获取数据，但是该方法效率低

3. 你常用的 mysql 引擎有哪些？各引擎间有什么区别？

主要 MyISAM 与 InnoDB 两个引擎，其主要区别如下：

一、InnoDB 支持事务，MyISAM 不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一些列增删改中只要哪个出错还可以回滚还原，而 MyISAM 就不可以了；

二、MyISAM 适合查询以及插入为主的应用，InnoDB 适合频繁修改以及涉及到安全性较高的应用；

三、InnoDB 支持外键，MyISAM 不支持；

四、MyISAM 是默认引擎，InnoDB 需要指定；

五、InnoDB 不支持 FULLTEXT 类型的索引；

六、InnoDB 中不保存表的行数，如 select count(*) from table 时，InnoDB；需要扫描一遍整个表来计算有多少行，但是 MyISAM 只要简单的读出保存好的行数即可。注意的是，当 count(*)语句包含 where 条件时 MyISAM 也需要扫描整个表；

七、对于自增长的字段，InnoDB 中必须包含只有该字段的索引，但是在 MyISAM 表中可以和其他字段一起建立联合索引；

八、清空整个表时，InnoDB 是一行一行的删除，效率非常慢。MyISAM 则会重建表；

九、InnoDB 支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like '%lee%'

4. 请描述数据在 scrapy 中的传递过程?

答：从 start_urls 里获取第一批 url 并发送请求，请求由引擎交给调度器入请求队列，获取完毕后，调度器将请求队列里的请求交给下载器去获取请求对应的响应资源，并将响应交给自己编写的解析方法做提取处理：1. 如果提取出需要的数据，则交给管道文件处理；2. 如果提取出 url，则继续执行之前的步骤（发送 url 请求，并由引擎将请求交给调度器入队列...），直到请求队列里没有请求，程序结束。

5. 在你简历中的项目，哪个项目最难，具体哪里比较难？你是如何解决的，用了多久解决的?

答:答案同问题 8

6. 在你之前的爬虫工作中，最大的收获是什么?

答：此题目为开放性问题，可以从以下角度回答

1 通过之前的工作，更全面细致的掌握了爬虫相关的技术，能够更加快速的进行爬虫相关的开发

2 爬虫的很多技能和经验需要积累的，通过之前的工作， 掌握了很多爬虫和反反爬相关的经验

3. 可以谈具体在某一个技术方面的收获，比如对 http 有了更加全面的掌握，或者是对反扒技术有了更多的思考

7. 数据库的优化?

1. 优化索引、SQL 语句、分析慢查询；

2. 设计表的时候严格根据数据库的设计范式来设计数据库；

3. 使用缓存，把经常访问到的数据而且不需要经常变化的数据放在缓存中，能节约磁盘 IO；

4. 优化硬件；采用 SSD，使用磁盘队列技术(RAID0,RAID1,RDID5)等；

5. 采用 MySQL 内部自带的表分区技术，把数据分层不同的文件，能够提高磁盘的读取效率；

6. 垂直分表；把一些不经常读的数据放在一张表里，节约磁盘 I/O；

7. 主从分离读写；采用主从复制把数据库的读操作和写入操作分离开来；

8. 分库分表分机器（数据量特别大），主要的的原理就是数据路由；

9. 选择合适的表引擎，参数上的优化；

10. 进行架构级别的缓存，静态化和分布式；

11. 不采用全文索引；

12. 采用更快的存储方式，例如 NoSQL 存储经常访问的数据

8. 你遇到的最难的反扒措施是什么，是哪个网站？你是如何解决的，用了多久解决的？

该题目为开放性问题，可以从西面几个角度回答

1）.通过 Headers 反爬虫

从用户请求的 Headers 反爬虫是最常见的反爬虫策略。很多网站都会对 Headers 的 User-Agent 进行检测，还有一部分网站会对 Referer 进行检测（一些资源网站的防盗链就是检测 Referer）。如果遇到了这类反爬虫机制，可以直接在爬虫中添加 Headers，将浏览器的 User-Agent 复制到爬虫的 Headers 中；或者将 Referer 值修改为目标网站域名。对于检测 Headers 的反爬虫，在爬虫中修改或者添加 Headers 就能很好的绕过。

2）.基于用户行为反爬虫

还有一部分网站是通过检测用户行为，例如同一 IP 短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。

大多数网站都是前一种情况，对于这种情况，使用 IP 代理就可以解决。可以专门写一个爬虫，爬取网上公开的代理 ip，检测后全部保存起来。这样的代理 ip 爬虫经常会用到，最好自己准备一个。有了大量代理 ip 后可以每请求几次更换一个 ip，这在 requests 或者 urllib2 中很容易做到，这样就能很容易的绕过第一种反爬虫。

对于第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。

3）.使用 js 加密了数据或者是参数

1. 使用 selenium 模拟浏览器获取数据，但是该方法效率低

2.分析请求中的 js，观察其执行过程，了解 js 是如何加密数据，然后使用类似 js2py 模块执行 js，获取执行后的结果

9. 之前的某个项目中，一个采集了多少个网站的数据，具体是那些网站？每天能够抓取多少数据，是每天定时执行么？如何定时执行的？

答：该问题为开放性问题，可以通过下面的方式回答

这个项目抓取了网站 1，网站 2，网站 3 等 40 个网站的 XXX 数据，这些网站每个平均每天的更新数据量在 400 条左右，每天大概能够抓取 2 万条数据

程序是通过 crontab 定时执行，该项目每次运行需要完需要 1 个小时，所以在 corntab

中设置每 3 个小时执行一次

10. 有没有自己写过框架，为什么要自己实现框架？解决了现有框架的什么问题，具体框架的流程是怎么样的？

答：该问题为开放性问题，可以从下面的角度回答

a) 在工作中没有写过，但是自己之前工作之余模仿 scrapy 写过功能和 scrapy 类似的框，主要是用来练习和学习

b) 实现框架的主要目的是为了解决某些特定网站的数据采集的需求，同时能够自己实现对更多细节的控制。比如公司主要的业务就是采集全国所有高校的新闻动态，那么此时就可以完成一个高校新闻采集的爬虫框架，在这个过程中能够对每个过程进行更加细致的过程，比如提取的字段，代理的使用，数据的保存。

框架中有如下对象：

a) 三个内置对象：

i. 请求对象(Request)
ii. 响应对象(Response)
iii. 数据对象(Item)

b) 五个核心组件：

i. 爬虫组件
   构建请求信息(初始的)，也就是生成请求对象(Request)
   解析响应对象，返回数据对象(Item)或者新的请求对象(Request)
ii. 调度器组件
   缓存请求对象(Request)，并为下载器提供请求对象，实现请求的调度
   对请求对象进行去重判断
iii. 下载器组件
   根据请求对象(Request)，发起 HTTP、HTTPS 网络请求，拿到 HTTP、HTTPS 响应，构建响应对象(Response)并返回
iv. 管道组件

负责处理数据对象(Item)

v. 引擎组件

负责驱动各大组件，通过调用各自对外提供的 API 接口，实现它们之间的交互和协作

提供整个框架的启动入口

c) 两个中间件：

i. 爬虫中间件

对请求对象和数据对象进行预处理

ii. 下载器中间件

对请求对象和响应对象进行预处理

框架流程：

i. 构造 spider 中 start_urls 中的请求

ii. 传递给调取器进行保存，之后从中取出

iii. 取出的 request 对象交给下载的进行下载 ，返回 response

iv. response 交给爬虫模块进行解析， 提取结果

v. 如果结果是 request 对象，重新交给调度器，如果结果是 item 对象，交给管道处理