

# python 知识

## 1. python 中有没有 switch—case 这种语法?

python 中没有 switch--case 这种语法！！但是我们可以通过下面几种方式实现 switch--case 语句。

<https://blog.csdn.net/1460133921/article/details/74892476>

(1) 使用 if…elif…elif…else 实现 switch/case。但是随着分支的增多和修改的频繁，这种代替方式并不很好调试和维护。

(2) 使用字典 实现 switch/case

注意：dict.get(key, default=None) #返回指定键的值，如果值不在字典中返回 default 值

可以使用字典实现switch/case这种方式易维护，同时也能够减少代码量。如下是使用字典模拟的switch/case实现：

```
1 def num_to_string(num):
2     numbers = {
3         0 : "zero",
4         1 : "one",
5         2 : "two",
6         3 : "three"
7     }
8
9     return numbers.get(num, None)
10
11 if __name__ == "__main__":
12     print num_to_string(2)
13     print num_to_string(5)
```

执行结果如下：

```
1 two
2 None
```

## 2. Python 的垃圾回收机制

<https://www.bilibili.com/video/BV1W4411S7mq?from=search&seid=5343395351328988096>

<http://www.ityouknow.com/python/2020/01/06/python-gc-111.html>

垃圾不回收，会造成什么问题？---内存泄漏

python 的垃圾回收机制，以**引用计数为主，标记清除和分代回收**为辅。垃圾回收是 Python 自带的功能，并不需要程序员去手动管理内存。

**引用计数：**为每个对象维护一个 ref 的字段用来记录对象被引用的次数，每当对象被创建或者被引用时将该对象的引用次数加一，当对象的引用被销毁时该对象的引用次数减一，当对象的引用次数减到零时说明程序中已经没有任何对象持有该对象的引用。此时垃圾回收机制就会将该对象回收。

什么情况下对象引用次数 ref 会加 1？有下面四种情况：

- 1) 对象被创建 (num=2)
- 2) 对象被引用 (count=num)
- 3) 对象作为参数传递到函数内部
- 4) 对象作为一个元素添加到容器中

什么情况下对象的引用次数 ref 会减 1？对应有下面四种情况：

- 1) 对象的别名被显式销毁 (`del num`)
- 2) 对象的别名被赋予新的对象 (`num=30`)
- 3) 对象离开它的作用域 (函数局部变量)
- 4) 从容器中删除对象, 或者容器被销毁

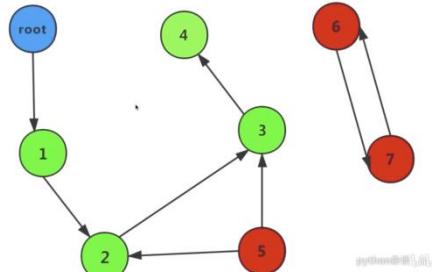
引用计数优缺点:

优点: (1) 简单; (2) 实时性高, 只要引用计数为 0, 对象就会被销毁, 内存被释放。

缺点: (1) 需要维护一个字段消耗很多资源; (2) 无法解决循环引用导致内存泄漏

(1) **标记清除**: 是一种基于对象可达性分析的回收算法, 该算法分为两个步骤: 标记和清除。

1) 标记阶段, GC (Garbage Collection 垃圾回收) 会把所有活动对象打上标记, 这些活动对象就如同一个个点, 他们之间的引用关系构成边, 这些点和边构成一个有向图, 如下图:



2) 清除阶段: 从根对象 (root) 出发, 沿着有向边, 遍历整个图, 不可达对象就是要清理的对象。这个根对象就是全局对象、调用栈、寄存器。**上图中, 5 不可达到, 6 和 7 相互引用, 这三个对象都会被回收。**

优点: 解决了相互引用对象的回收问题

缺点: 需要扫描所有对象, 有点浪费性能。(因为要标记)

(3) 由于标记清除算法需要扫描整个堆的所有对象导致其性能有所损耗, 而且当可以回收的对象越少时性能损耗越高。因此 Python 引入了**分代回收算法**。它建立在**标记回收基础上**, 是一种以**空间换时间的回收方式**。

**分代回收**: 根据内存中对象存活时间将他们分为 3 代。新生的对象放入 0 代, 如果一个对象能在第 0 代的垃圾回收过程中存活下来, GC 会将其放入到第 1 代, 如果第一代里的对象在第一代垃圾回收过程中存活下来, 则进到第二代。

分代回收的触发机制, 如下:

```
import gc  
print(gc.get_threshold())
```

上面的代码执行结果是(700, 10, 10)

- 当分配对象的个数减去释放对象的个数的差值大于700时，就会产生一次0代回收
- 10次0代回收会导致一次1代回收
- 10次1代回收会导致一次2代回收

对于第0代的对象来说，他们很可能就被使用一次，因此需要经常被回收。

经过一轮一轮的回收后，能够活着成为第2代的对象，必然是那些使用频繁的对象，而且他们已经存活很久的时间了，大概率的，还会存活很久，因此，2代回收的就不那么频繁，

优点：解决了标记清除算法需要扫描整个堆对象的性能浪费问题

缺点：用空间换时间

### 3. Python 的字典用什么实现的？

字典是通过散列表或说哈希表实现的

<https://zhuanlan.zhihu.com/p/74003719>

字典也叫哈希数组，所以本质是数组，底层是通过哈希表（散列表）实现的。

Python3.6 之前字典是无序的，python3.7 以后，字典是有序的。

Python3.6 字典底层实现：

字典底层维护一张哈希表，哈希表中每一个元素有存放了哈希值（hash）、键（key）、值（value）3个元素。

```
enteies = [  
    [None, None, None],  
    [hash, key, value],  
    [None, None, None],  
    [None, None, None],  
    [hash, key, value],]
```

增加一个元素的过程如下：

(1) 计算 key 的 hash 值【hash(key)】，再和 mask 做与操作【mask=字典最小长度 (DictMinSize) - 1】，运算后会得到一个数字【index】，这个 index 就是要插入的 enteies 哈希表中的下标位置

(2) 若 index 下标位置已经被占用，则会判断 enteies 的 key 是否与要插入的 key 相等

(3) 如果 key 相等就表示 key 已存在，则更新 value 值

(4) 如果 key 不相等，就表示 hash 冲突，则会继续向下寻找空位置，一直到找到剩余空位为止。

不同的 key 计算出的 index 值是不一样的，在 enteies 中插入的位置不一样，所以当我们遍历字典的时候，字段的顺序与我们插入的顺序是不相同的。

enteies 表是稀疏的，随着我们插入的值不同，enteies 表会越来越稀疏（enteies 也是一个会动态扩展长度的，每一次扩展长度，都会重新计算所有 key 的 hash 值），所以新的字典实现就随之出现。

Python3.7 以后：

老字典使用一张哈希表，而新字典另外还使用了一张 Indice 表来辅助，如下：

```

indices = [None, None, index, None, index, None, index]
enteies = [
    [hash0, key0, value0],
    [hash1, key1, value1],
    [hash2, key2, value2]
]

```

计算过程如下：

计算 key 的 hash 值【hash(key)】，再和 mask 做与操作【mask=字典最小长度（IndicesDictMinSize） - 1】，运算后会得到一个数字【index】，这个 index 就是要插入的 indices 的下标位置（注：具体算法与 Python 版本相关，并不一定一样）

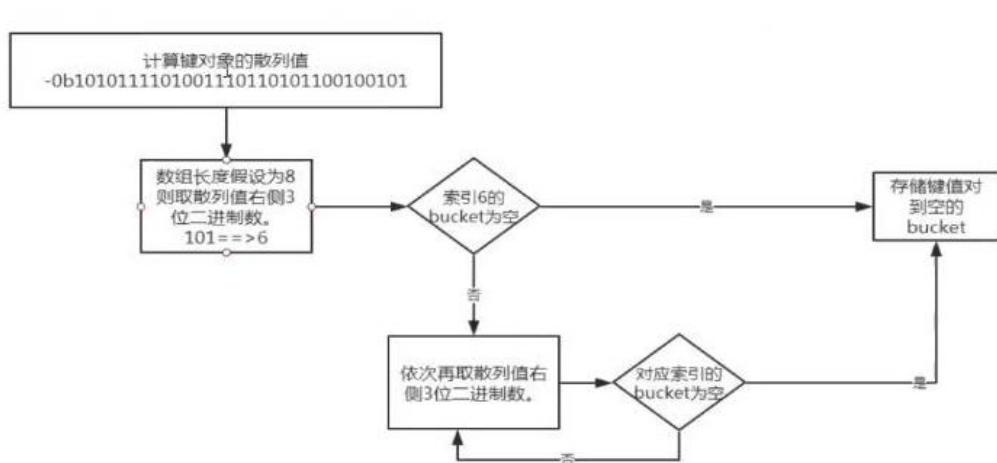
得到 index 后，会找到 indices 的位置，但是此位置不是存的 hash 值，而是存的 len(enteies)，表示该值在 enteies 中的位置

如果出现 hash 冲突，则处理方式与老字典处理方式类似。

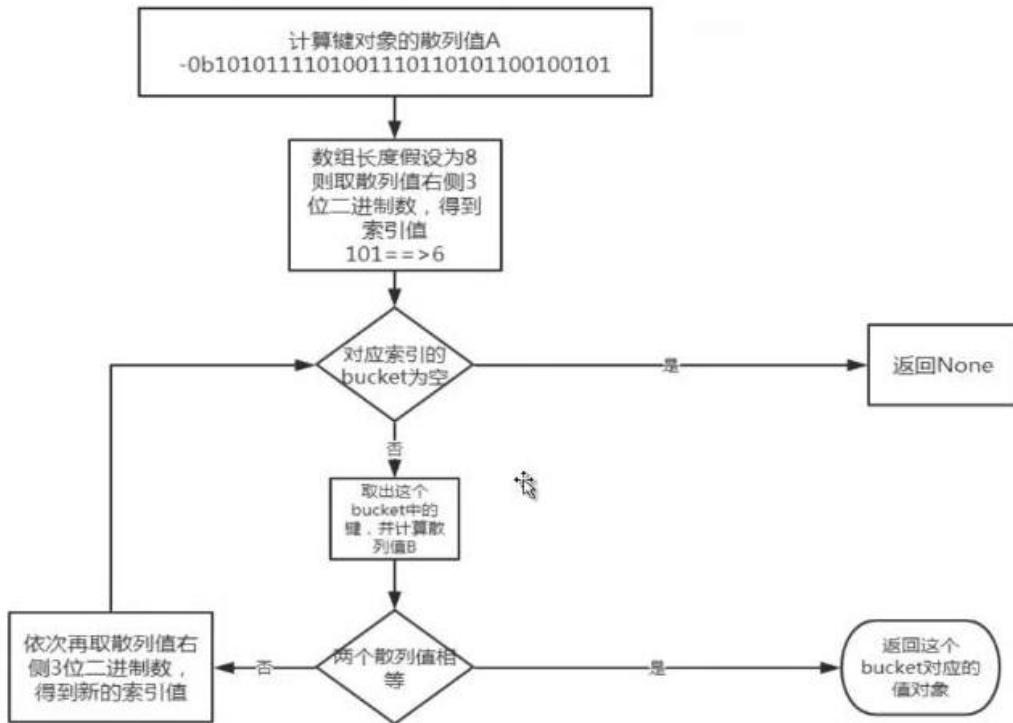
参考：<https://www.pianshen.com/article/13081237949/>

(1) 将一个键值对放在字典的过程：

流程图如下：



(2) 根据键查找‘键值对’的过程



### 用法总结一下：

1. 键必须可散列:
  - 1>数字、字符串、元组都是可以散列的
  - 2>自定义对象需要支持下面三点：
    - 1) 支持 `hash()` 函数
    - 2) 支持通过 `__eq__()` 方法检测相等性
    - 3) 若 `a==b` 为真，则 `hash(a)==hash(b)` 也为真
2. 字典在内存中开销巨大，典型的空间换时间
3. 键查询速度很快
4. 往字典里面添加新建可能导致扩容，导致散列表中键的次序发生变化，因此，不要在遍历字典的时候同时进行字典的修改。

#### 4. list 底层实现

`list` 被实现为长度可变的数组，列表 `list` 是可迭代的 (`iterable`)。python 中的 `list` 是由对其他对象引用组成的连续数组。指向这个数组的指针及其长度被保存在一个列表头结构中，这也意味着每次添加或删除一个元素时，由引用组成的数组需要该标大小（重新分配）。Python 在创建这些数组时采用了指数过 `s` 分分配，所以并不是每次操作都需要改变数组的大小。但是，也因为这个原因添加或取出元素的平均复杂度较低。

当我们建一个 `integer` (整数) 追加到 `list` 中，`l.append(1)`，将会发生什么？一个 C 内部方法 `app1()` 会被调用。

<https://zhuanlan.zhihu.com/p/143223943>

#### 5. Python 用过的模块有哪些

- (1) `requests`: 接口自动化
- (2) `json`: json 转换
- (3) `random`: 生成伪随机数

(4) selenium: web 的 UI 自动化

(5) threading: 多线程

(6) re: 正则表达式 #不说这

## 6. Python 的数据结构列举一下，它们之间的对比？（列一张表格对比）

<https://www.jianshu.com/p/5ede7fa96d83>

<https://blog.csdn.net/chenlunju/article/details/85717397>

	字符串(str)	列表(list)	元组(tuple)	字典(dict)	集合(set)
字面量	'abc' 或 "abc"	[a, b, c]	(1, 2, 3)	{'a': 1, 'b': 2}	{1, 2}
特点	不可变有序	可变并且有序	不可变并且有序	可变无序	可变无序不重复
空值	str1 = ''	list11 = []	tuple1 = ()	dict1 = {}	set1 = set()
获取	下标或切片	下标或切片	下标或切片	不支持下标	不支持下标
运算符	+*和比较运算	+*比较运算符	+*比较运算	==和!=	数学集合

### python 中一些内置函数：

1 cmp(A, B) 比较可为 list、tuple 等的 A、B, A>B 为 1, A=B 为 0, 小于为-1

2 len(obj) 计算元素个数。

3 max(obj) 返回元素最大值。

4 min(obj) 返回元素最小值。

5 reversed(obj) 对元素进行翻转

6 enumerate(obj) 遍历list的元素,返回序号及其对应的元素 for x, y in list:

7 zip() 打包成 tuple (元组), 然后返回由这些 tuples 组成的 list (列表)

## 7. Python 的元组和列表的区别（中途 ms 官补充提问：刚才你提到一个可变一个不可变，你可以再详细说说吗？我从使用场景说了一下）

见上一题。

tuple 不可变的好处：相对于 list 而言，tuple 是不可变的，这使得它可以作为 dict 的 key，或者扔进 set 里，而 list 则不行。

tuple 放弃了对元素的增删（内存结构设计上变的更精简），换取的是性能上的提升：创建 tuple 比 list 要快，存储空间比 list 占用更小。所以就出现了“能用 tuple 的地方就不用 list”的说法。

多线程并发时，tuple 是不需要加锁的，不用担心安全问题，编写也简单多了。

<https://blog.csdn.net/ruanxingzi123/article/details/83184909>

数据类型	列表(list)	元组(tuple)	集合(set)	字典(dictionary)
表示	[]	()	()	{ }
是否有序	有序	有序	无序	无序
是否读写	读写	只读	读写	读写
空定义	a_list = []	a_tuple = ()	a_set = set(); not_sure = {} 空字典	a_dict = {}
元素可修改	a_list[0] = 23	否	否	a_dict['age'] = 30
下标访问	a_list[0] = 23	a_tuple[0]	否	a_dict['age'] = 30
添加元素	+、append、extend、insert	不可添加	add、update	a_dict['new_key'] = 'value'
删除元素	del、remove、pop()、pop(1)、clear	不可删除	discard、remove、pop、clear	pop、popitem、clear
元素查找	index、count、in	in	in	a_dict['key']
布尔真值	非空	非空	非空	非空
定义	列表(list)是有序的集合，可以存放不同数据类型的数据,每个元素的都对应着一个索引来标记其位置，且索引从 0 开始	tuple 与 list 类似，不同之处在于 tuple 中的元素不能进行修改。tuple 使用小括号(), list 使用方括号[]。	dic 字典是另一种可变的容器模型，且可存储任意类型对象。字典的每个键值(key:value)对用冒号分割，每个对之间用逗号分割，整个字典在{}中	set()函数是创建一个无序不重复元素集，可添加，删除数据，计算交集、差集、并集等。python 的集合是一个无序不重复元素集，基本功能包括关系测试和消除重复元素.集合对象还支持 union(联合), intersection(交), difference(差)和 sysmmetricdifference(对称差集)等数学运算

## 8. 元组使用场景一般有：

- (1) 函数的传参和返回值，一个函数可以接收任意多个参数，一次返回多个数据；
- (2) 格式化字符串：print('name:%s, age:%d' % (name, age))
- (3) 交换两个变量的值
- (4) 让列表不可以被修改，保护数据（将 list 转换成元组，再进行操作）
- (5) 如果存储的数据和数量不变，比如你有一个函数，需要返回的是一个地点的经纬度，然后直接传给前端渲染，那么肯定选用元组更合适。
- (6) 如果存储的数据或数量是可变的，比如社交平台上的一个日志功能，是统计一个用户在一周之内看了哪些用户的帖子，那么则用列表更合适
- (7) 一般在 key 中使用元组，其他情况多数都使用列表（字典的 key 可以用元组）

题外话：

这两个有区别吗？

```
# 创建空列表  
# option A  
empty_list = list()  
  
# option B  
empty_list = []
```

Answer:

区别主要在于 list() 是一个 function call，Python 的 function call 会创建 stack，并且进行一系列参数检查的操作，比较 expensive，反观 [] 是一个内置的 C 函数，可以直接被调用，因此效率高。

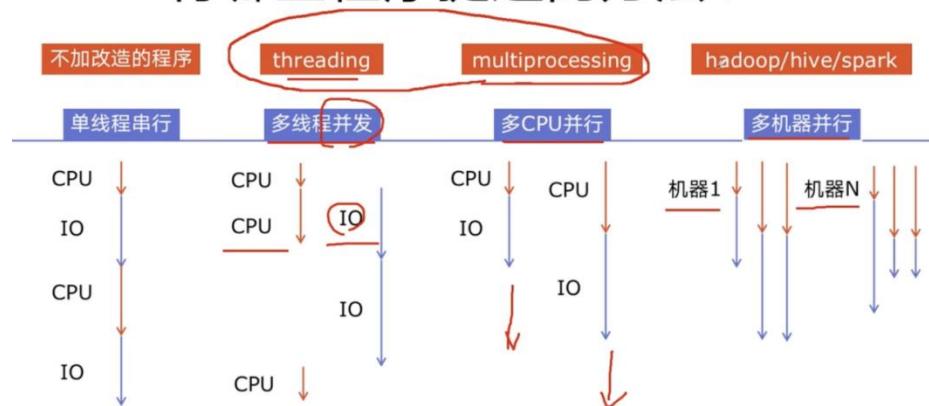
## 9. Python 的锁，多线程

- (1) 为什么要引入并发编程？

提升程序运行速度

- (2) 几种程序提速方法。IO 在计算机中指 Input/Output，也就是输入和输出

## 有哪些程序提速的方法？



- 1) 单线程串行：有一个线程，cpu 先执行，执行完成后，IO 执行；在 IO 期间，

cpu 是不做任何事情的，IO 完成后，cpu 进行运算，完成计算后，再进行下一次 IO。在 IO 期间，CPU 是等待的状态，这样在时间上有很大的浪费。

(cpu=>io=>cpu=>...)

2) 多线程并发：有一个线程，cpu 执行，执行过程中如果遇到 io，cpu 会切换到另一个 task 执行；然后当 io 执行完，会通知 cpu 进行下一步处理，(cpu 和 io 是可以同时进行的，io 的执行，如读取内存、磁盘、网络过程中，不需要 cpu 的参与。这样 cpu 就可以释放出来执行他自己的 task，实现并发加速)但是这种运行方式在原理上还是一个 cpu 来进行的

3) 多 cpu 并行：现在电脑很多都是多 cpu，所以可以实现多 cpu、多条线同时真正并行执行来进行加速

4) 大数据时代，很多程序的运行都可以用很多机器并行来进行运算。每个机器上有很多个 cpu，每个 cpu 上也可以进行并发执行。

PS：进程是资源分配的基本单位，是调度单位；

线程是 CPU 调度和分配的基本单位。

<https://www.cnblogs.com/gtscool/p/13072051.html>

(3) python 对并发编程支持

## Python对并发编程的支持

- 多线程：threading，利用CPU和IO可以同时执行的原理，让CPU不会干巴巴等待IO完成
  - 多进程：multiprocessing，利用多核CPU的能力，真正的并行执行任务
  - 异步IO：asyncio，在单线程利用CPU和IO同时执行的原理，实现函数异步执行
- 
- 使用Lock对资源加锁，防止冲突访问
  - 使用Queue实现不同线程/进程之间的数据通信，实现生产者-消费者模式
  - 使用线程池Pool/进程池Pool，简化线程/进程的任务提交、等待结束、获取结果
  - 使用subprocess启动外部程序的进程，并进行输入输出交互

(3) python 并发编程的三种方式：**多线程 thread**, **多进程 process**, **多协程 coroutine**

(4) cpu 密集型和 io 密集型

## 1、什么是CPU密集型计算、IO密集型计算？

### CPU密集型 (CPU-bound) : CPU受限制

CPU密集型也叫计算密集型，是指I/O在很短的时间就可以完成，CPU需要大量的计算和处理，特点是CPU占用率相当高

例如：压缩解压缩、加密解密、正则表达式搜索

### IO密集型 (I/O bound) : IO受限制

IO密集型指的是系统运作大部分的状况是CPU在等I/O (硬盘/内存) 的读/写操作，CPU占用率仍然较低。

例如：文件处理程序、网络爬虫程序、读写数据库程序

## (5) 多进程、多线程、多协程对比

### 2、多线程、多进程、多协程的对比

一个进程中  
可以启动N个线程

一个线程中  
可以启动N个协程

#### 多进程 Process (multiprocessing)

- 优点：可以利用多核CPU并行运算
- 缺点：占用资源最多、可启动数目比线程少
- 适用于：CPU密集型计算

#### 多线程 Thread (threading)

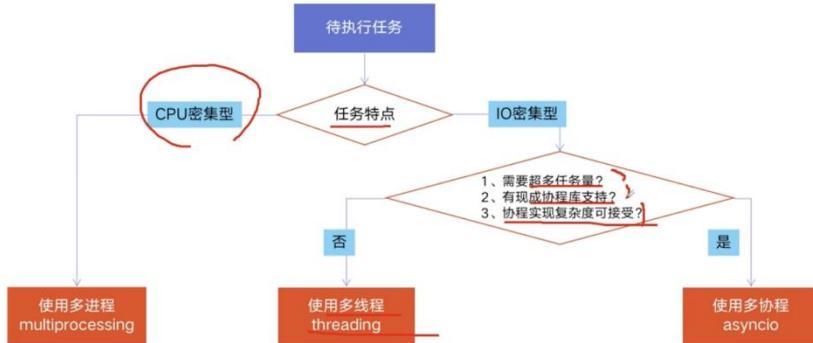
- 优点：相比进程，更轻量级、占用资源少
- 缺点：
  - 相比进程：多线程只能并发执行，不能利用多CPU (GIL)
  - 相比协程：启动数目有限制，占用内存资源，有线程切换开销
- 适用于：IO密集型计算、同时运行的任务数目要求不多

#### 多协程 Coroutine (asyncio)

- 优点：内存开销最少、启动协程数量最多
- 缺点：支持的库有限制 (aioweb vs requests)、代码实现复杂
- 适用于：IO密集型计算、需要超多任务运行、但有现成库支持的场景

## (6) 如何选择这几种技术

### 3、怎样根据任务选择对应技术？



#### (6) 全局解释器锁 GIL (Global Interpreter Lock)

##### 1) python 速度慢的原因

A. Python 是解释型语言，程序边解释边执行；而其他语言如 C、C++、JAVA 等是编译型语言，在开发完代码后，会先进行编译，将程序编译成可以直接执行的机器码，机器码放在电脑上执行速度非常快。但对于 python 来说，每次执行的都是源码，就存在一个从源码到机器码的翻译过程，这个翻译就是边解释边执行的，所以速度很慢。

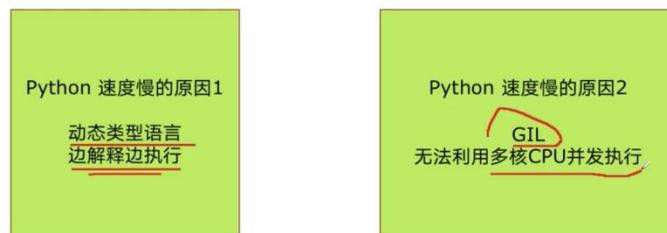
B. Python 中的变量既可以是一个数字，也可以随时切换成一个字符串，或者其他数据类型。但这需要 python 在执行过程中随时检查数据类型，导致速度变慢。  
(此处不讲)

##### C. GIL 导致 python 无法利用多核 CPU 并发执行

#### 1、Python速度慢的两大原因

相比C/C++/JAVA，Python确实慢，在一些特殊场景下，Python比C++慢100~200倍

由于速度慢的原因，很多公司的基础架构代码依然用C/C++开发  
比如各大公司阿里/腾讯/快手的推荐引擎、搜索引擎、存储引擎等底层对性能要求高的模块



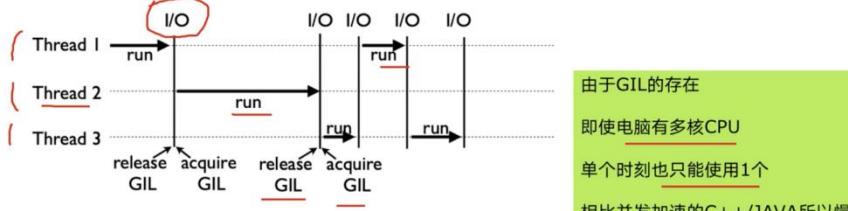
##### 2) 什么是 GIL?

## 2、GIL 是什么？

全局解释器锁（英语：Global Interpreter Lock，缩写GIL）

是计算机程序设计语言解释器用于同步线程的一种机制，它使得任何时刻仅有一个线程在执行。

即便在多核心处理器上，使用 GIL 的解释器也只允许同一时间执行一个线程。



- When a thread is running, it holds the GIL

- GIL released on I/O (read, write, send, recv, etc.)

由于 GIL 的存在

即使电脑有多核CPU

单个时刻也只能使用1个

相比并发加速的C++/JAVA所以慢



当一个线程在运行的时候，它会持有 GIL，当遇到 io 后（io 如读写内存，发送网络等），会释放 GIL。

上图流程：当前 t1 在执行，此时它持有 gil 锁，当 t1 遇到 io 后，会释放 gil 锁，此时，t2 才能进入运行状态（t1 运行时，t2 和 t3 不能运行；t2 运行时，t1 和 t3 不能运行，即同一时刻只有一个 thread 在运行），持有 gil 锁，。。。。整个过程中，看黑线，虽然是多线程，但是同一时刻只有一个线程在运行。

3) 为什么 python 要有 GIL 这个东西？

## 3、为什么有 GIL 这个东西？

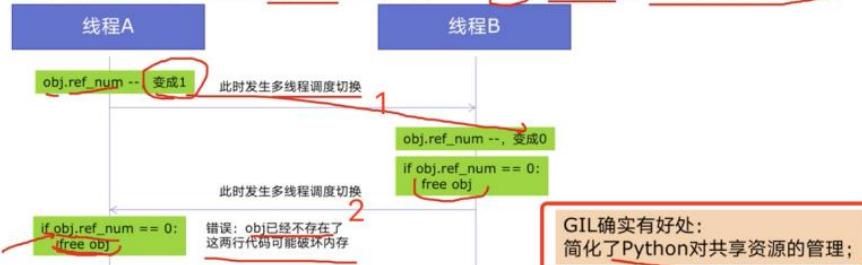
简而言之：Python设计初期，为了规避并发问题引入了GIL，现在想去除却去不掉了！

为了解决多线程之间数据完整性和状态同步问题

原因  
详解

Python中对象的管理，是使用引用计数器进行的，引用数为0则释放对象

开始：线程A和线程B都引用了对象obj, obj.ref\_num = 2, 线程A和B都想撤销对obj的引用



GIL确实有好处：  
简化了Python对共享资源的管理；

在系统执行过程中，线程会随时发生切换，即把执行权交给另一个线程

4) 怎样规避 GIL 带来的限制

## 4、怎样规避GIL带来的限制？

1、多线程 `threading` 机制依然是有用的，用于IO密集型计算

因为在 I/O (read, write, send, recv, etc.)期间，线程会释放GIL，实现CPU和IO的并行  
因此多线程用于IO密集型计算依然可以大幅提升速度

但是多线程用于CPU密集型计算时，只会更加拖慢速度

2、使用 `multiprocessing` 的多进程机制实现并行计算、利用多核CPU优势

为了应对GIL的问题，Python提供了`multiprocessing`

6. Python 装饰器，闭包（装饰器基于闭包函数）

<https://www.zhihu.com/question/431450010/answer/1591944196>

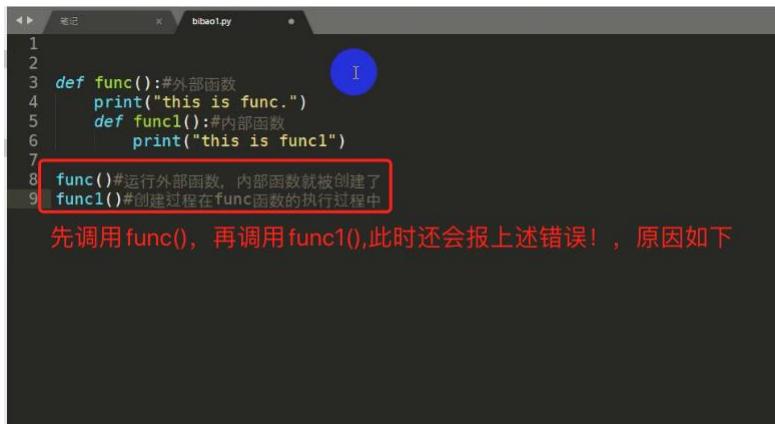
<https://www.bilibili.com/video/BV1JW411i7HR?from=search&seid=5443920735821518345>

[https://www.bilibili.com/video/BV1ZJ411y7Te/?spm\\_id\\_from=333.788.recommend\\_more\\_video.-1](https://www.bilibili.com/video/BV1ZJ411y7Te/?spm_id_from=333.788.recommend_more_video.-1)

<https://www.bilibili.com/video/BV1JW411i7HR?from=search&seid=5443920735821518345>

```
def func():#外部函数
    print("this is func.")
    def func1():#内部函数
        print("this is func1")
    func1()#创建过程在func函数的执行过程中 不可以直接调用内部函数
File "C:\Users\Administrator\Desktop\闭包装饰器\bibao1.py", line 8, in <module>
只有在调用func()后，func1()才会被创建

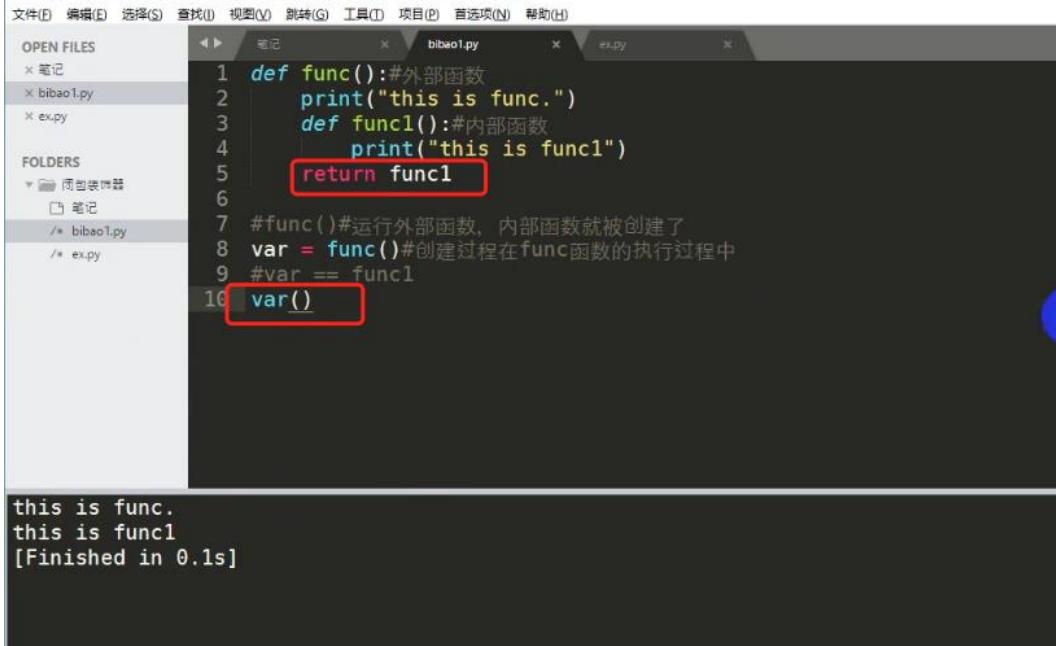
Traceback (most recent call last):
  File "C:\Users\Administrator\Desktop\闭包装饰器\bibao1.py", line 8, in <module>
    func1()#创建过程在func函数的执行过程中
    ^
NameError: name 'func1' is not defined
[Finished in 0.2s with exit code 1]
[shell_cmd: python -u "C:\Users\Administrator\Desktop\闭包装饰器\bibao1.py"]
[dir: C:\Users\Administrator\Desktop\闭包装饰器]
```



```
1
2
3 def func():#外部函数
4     print("this is func.")
5     def func1():#内部函数
6         print("this is func1")
7
8 func()#运行外部函数，内部函数就被创建了
9 func1()#创建过程在func函数的执行过程中
```

先调用 func(), 再调用 func1(), 此时还会报上述错误! , 原因如下

函数内的各种属性只有在函数运行期间存活，当函数运行完成后，函数内的各种属性就释放了。那如何在调用完外部函数后，内部函数仍然存活呢？可以在外部函数将内部函数 return 回来！方法如下：



```
1 def func():#外部函数
2     print("this is func.")
3     def func1():#内部函数
4         print("this is func1")
5     return func1
6
7 #func()#运行外部函数，内部函数就被创建了
8 var = func()#创建过程在func函数的执行过程中
9 #var == func1
10 var()
```

this is func.  
this is func1  
[Finished in 0.1s]

```
OPEN FILES
× 笔记
× bibao1.py
× ex.py

FOLDERS
+ 笔记装饰器
  □ 笔记
  /* bibao1.py
  /* ex.py

1 def func():#外部函数
2     a = 1 #外部函数作用域里的变量
3     print("this is func.")
4     def func1(num):#内部函数
5         print("this is func1")
6         print(num + a) #内部函数引用外部变量
7         return func1 #函数
8
9 #func()#运行外部函数, 内部函数就被创建了
10 var = func()#创建过程在func函数的执行过程中
11 #var == func1
12 var(3)

this is func.
this is func1
4
[Finished in 0.1s]
```

```
C:\Users\Administrator\Desktop\闭包装饰器\bibao1.py (闭包装饰器) - Sublime Text
文件(E) 编辑(E) 选择(S) 查找(I) 视图(V) 切换(G) 工具(T) 项目(P) 首选项(N) 帮助(H)

OPEN FILES
× 笔记
× bibao1.py
× ex.py

FOLDERS
+ 笔记装饰器
  □ 笔记
  /* bibao1.py
  /* ex.py

1 def func():#外部函数
2     a = 1 #外部函数作用域里的变量
3     print("this is func.")
4     def func1(num):#内部函数
5         print("this is func1")
6         print(num + a)
7         return func1 #函数
8
9 #func()#运行外部函数, 内部函数就被创建了
10 var = func()#创建过程在func函数的执行过程中
11 #var == func1
12 var(3)

this is func.
this is func1
4
[Finished in 0.1s]
```

上面这个示例就是一个闭包了

注意：

return func1: 返回集装箱，即函数对象，函数名

return func1(): 返回的是函数调用，即在这种返回方式下，函数将被调用，即返回 func1() 被执行完成后的结果

```
1 def func1(func):#外部闭包函数的参数是被装饰的函数对象
2     def func2():
3         print('aaabbb')
4         return func()#返回了外部函数接收的被装饰函数的调用
5     return func2
6 #func1() takes 0 positional arguments but 1 was given
7
8 #return func #返回了函数对象
9 #return func() #返回的是一个函数调用
10
11 #func1(myprint)() #接收被装饰的函数作为参数，而且还要继续调用一次
12 #func2() -> print('aaabbb') -> return myprint()
13
14 @func1
15 def myprint():
16     print('你好，我是print')
17
18
19
20
21 myprint() #func1(myprint)()
```

```
1 def arg_func(sex):
2     def func1(b_func):
3         def func2():
4             if sex == 'man':
5                 print('你不可以生娃')
6             if sex == 'woman':
7                 print('你可以生娃')
8             return b_func()    返回函数调用
9         return func2
10
11
12
13 #arg_func(sex='man')()() > func1
14 #func1() > func2
15 #func2() > ('你不可以生娃') or print('你可以生娃') b_func()
16
17 @arg_func(sex='man')
18 def man():
19     print('好好上班。')
20
21 @arg_func(sex='woman')
22 def woman():
23     print('好好上班。')
24
25
26 man()
27 woman()
```

## 10. 什么是闭包？

如果一个外函数中定义了一个内函数，且内函数体内引用到了体外的变量（非全局变量），这时外函数通过 return 返回**内部函数**时，会把定义时涉及到的**外部引用变量和内函数打包成一个整体(闭包)返回**。一般一个函数运行结束的时候，临时变量会被销毁。但是闭包是一个特别的情况。当外函数发现，自己的临时变量会在将来的内函数中用到，自己在结束的时候，返回内函数的同时，会把外函数的临时变量同内函数绑定在一起。这样即使外函数已经结束了，内函数仍然能够使用外函数的临时变量。这就是闭包的强大之处。

(2) 创建一个闭包必须要满足以下几点：

- 1) 必须有一个内嵌函数
- 2) 内嵌函数必须引用外部函数中的变量
- 3) 外部函数的返回值必须是内嵌函数

(3) **闭包作用：它保存了函数的外部变量，不会随着变量的改变而改变了**

(4) **闭包和装饰器区别和联系：闭包传递的是变量，而装饰器传递的是函数，**

除此之外没有任何区别，或者说装饰器是闭包的一种，它只是传递函数的闭包。

## 11. 装饰器应用场景：

- 1) 引入日志
- 2) 函数执行时间统计
- 5) 权限校验等场景
- 6) 数据类型校验
- 3) 执行函数前预备处理
- 4) 执行函数后清理功能
- 6) 缓存

(5) 装饰器作用：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

## 12. 写一个装饰器示例：

```
21     from functools import wraps
22
23     def use_logging(level):
24         def decorator(func):
25             @wraps(func)
26             def wrapper(*args, **kwargs):
27                 if level == 'warn':
28                     logging.warning('%s is warnnnnn' % func.__name__)
29                 elif level == 'info':
30                     logging.info('%s is infoooooo' % func.__name__)
31                 else:
32                     logging.error('%s is errorrrrrr' % func.__name__)
33                 return func(*args, **kwargs)
34             return wrapper
35         return decorator
36
37     @use_logging(level='warn')
38     def foo(name):
39         print('i am %s' % name)
40
41     print(foo.__name__, foo.__doc__)
```

## 13. Python 的迭代器和生成器了解有哪些？

<https://www.cnblogs.com/wj-1314/p/8490822.html>

迭代是 Python 最强大的功能之一，是访问集合元素的一种方式。

迭代器实现了两个方法，一个是`__iter__()`，一个是`__next__()`，`__iter__()`用来返回迭代器本身，`__next__`用来取出下一个元素。

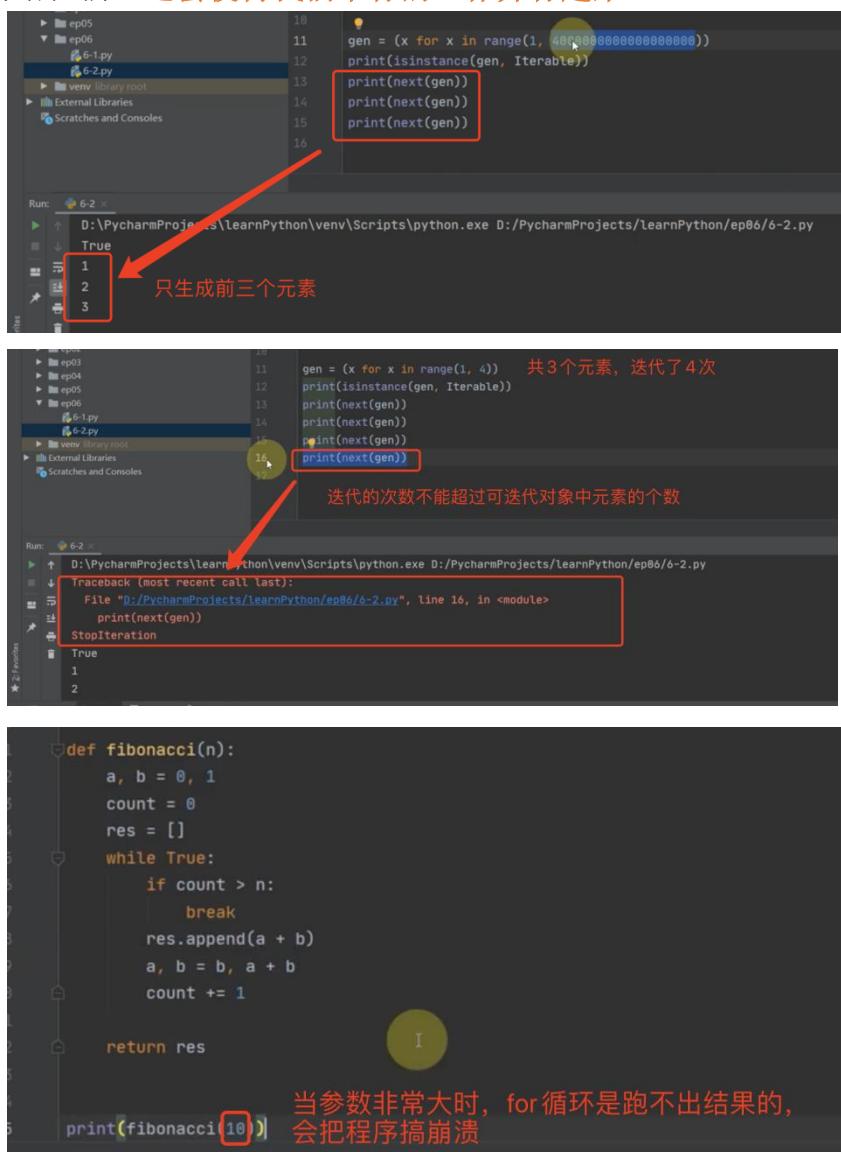
生成器也是迭代器的一种，迭代器只能记住自身的执行状态并等待下一次迭代，而生成器除了也会记住执行状态，还可以通过`yield`语句控制使多个生成器切换执行，例如手枪只能由一个枪口打出子弹，而加特林机枪可以通过旋转控制（`yield`切换）用多个枪口依次出子弹，威力也会更强，生成器是实现异步协程的重要基础。

在 Python 中，这种一边循环一边计算的机制，称为生成器：generator

生成器：边迭代边输出

生成器要解决的问题：如果我们要遍历的一个 list 中元素很多，或者元素占用的空间超级大。比如：

**生成器使用场景：**比如我们要获取并输出 1000 个文件，每个文件 500M，在网络传输中这是很耗时的。我们并不希望将所有文件都下载完成后，在进行下一步操作（循环，遍历，格式转换等等），我们希望的是下载完一个文件输出一个文件，让我的下级调用着直接对它进行转换处理等等，不用等我所有文件都下载完再给他们。**这会使得我们串行的工作并行起来。**



```

6-2.py
def fibonacci(n):
    a, b = 0, 1
    count = 0
    while True:
        if count > n:
            break
        yield a + b
        a, b = b, a + b
        count += 1
    gen = fibonacci(1000000000000)
for i in gen:
    if i > 1000:
        break
    print(i)

```

采用生成器，会边迭代边输出

<https://www.bilibili.com/video/BV1T5411Y7mP?from=search&seid=15455098079985906915>

14. `__new__`, `__init__`, `__call__` 分别是干嘛的?

<https://www.cnblogs.com/huchong/p/8260151.html>

`__new__`: 对象的创建, 是一个静态方法, 第一个参数是 `cls`。(想想也是, 不可能是 `self`, 对象还没创建, 哪来的 `self`)。其必须要有返回值, 返回实例化出来的实例, 需要注意的是, 可以 `return` 父类`__new__()`出来的实例, 也可以直接将 `object` 的`__new__()`出来的实例返回。

`__init__`: 对象的初始化, 是一个实例方法, 第一个参数是 `self`, 该 `self` 参数就是`__new__()`返回的实例, `__init__()`在`__new__()`的基础上可以完成一些其它初始化的动作, `__init__()`不需要返回值。

`__call__`: 对象可 `call`, 注意不是类, 是对象。

### `__call__`

在Python中, 函数其实是一个对象:

```

1 | >>> f = abs
2 | >>> f.__name__
3 | 'abs'
4 | >>> f(-123)

```

由于 `f` 可以被调用, 所以, `f` 被称为可调用对象。

所有的函数都是可调用对象。

一个类实例也可以变成一个可调用对象, 只需要实现一个特殊方法`__call__()`。

我们把 Person 类变成一个可调用对象：

```
1 class Person(object):
2     def __init__(self, name, gender):
3         self.name = name
4         self.gender = gender
5
6     def __call__(self, friend):
7         print 'My name is %s...' % self.name
8         print 'My friend is %s...' % friend
```

现在可以对 Person 实例直接调用：

```
1 >>> p = Person('Bob', 'male')
2 >>> p('Tim')
3 My name is Bob...
4 My friend is Tim...
```

举例：实例化一个对象 obj=Foo()时，会先执行 Foo 类的\_\_new\_\_方法（自己有就用自己的，没有就用父类的）创建一个对象，并返回；然后执行\_\_init\_\_方法（自己有就用自己的，没有就用父类的），对创建的对象进行初始化。  
obj( )会执行 Foo 类的\_\_call\_\_方法，自己有就用自己的，没有则用父类的。

## 15. 进程、线程有什么区别？什么情况下用进程？什么情况下用线程？进程通信方式？

进程是指在系统中正在运行的一个应用程序；程序一旦运行就是进程，或者更专业化来说：**进程是指程序执行时的一个实例；线程是进程的一个实体。**

**进程——资源分配的最小单位，线程——程序执行的最小单位。**

线程进程的区别体现在几个方面：

因为进程拥有独立的堆栈空间和数据段，所以每当启动一个新的进程必须分配给它独立的地址空间，建立众多的数据表来维护它的代码段、堆栈段和数据段，这对于多进程来说十分“奢侈”，系统开销比较大。而线程不一样，**线程拥有独立的堆栈空间，但是共享数据段，它们彼此之间使用相同的地址空间，共享大部分数据，比进程更节俭，开销比较小，切换速度也比进程快，效率高，但是正由于进程之间独立的特点，使得进程安全性比较高，也因为进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。一个线程死掉就等于整个进程死掉。**

(1) **开销：**进程拥有独立的堆栈空间和数据段，系统开销较大；**线程拥有独立的堆栈空间，但是共享数据段，它们彼此之间使用相同的地址空间，共享大部分数据，系统开销比较小。**

(2) **数据共享：**属于同一个进程的所有线程共享该进程的所有资源，包括文件描述符。而不同的进程相互独立

(3) **安全：**由于进程之间相互独立的特点，进程安全性较高，一个进程崩溃后，不会对其他进程产生影响；但是线程只是进程中的不同执行路径，一个线程死掉就等于整个进程死掉。

(4) **通信机制：**因为进程之间互不干扰，相互独立，进程的通信机制相对很复

杂，譬如管道，信号，消息队列，共享内存，套接字等通信机制，而线程由于共享数据段所以通信机制很方便。

(5) 两者关系：一个线程必定也只能属于一个进程，而进程可以拥有多个线程而且至少拥有一个线程。

(5) 控制块：线程又称为轻量级进程，进程有进程控制块，线程有线程控制块。进程适用于cpu密集型计算（计算量大的场景）

优点：可以利用多核cpu并行运算

缺点：占用资源多，可启动比线程少

线程适用于IO密集型计算（输入输出多的场景）

优点：相比进程，更轻量级，占用资源更少

缺点：多线程只能并发执行，不能利用多cpu(GIL)

进程与线程的选择取决于以下几点：

(1) 创建销毁：需要频繁创建销毁的优先使用线程；因为对进程来说，创建和销毁一个进程的代价是很大的。

(2) 切换频繁：线程的切换速度快，所以在需要大量计算，切换频繁时用线程，还有耗时的操作使用线程可提高应用程序的响应。

(3) 因为对CPU系统的效率使用上线程更占优，所以多机分布用进程，多核分布用线程。

(4) 并行操作：并行操作时使用线程，如C/S架构的服务器端并发线程响应用户的请求。

(5) 稳定安全：需要更稳定安全时，适合选择进程；

(6) 速度：需要速度时，选择线程更好

## 16. 简单讲，进程和线程的关系：

(1) 包含关系：一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。

(2) 资源共享：资源分配给进程，同一进程的所有线程共享该进程的所有资源。

(3) 处理机分给线程，即真正在处理机上运行的是线程。

(4) 线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

操作系统的设计，因此可以归结为三点：

(1) 以多进程形式，允许多个任务同时运行；

(2) 以多线程形式，允许单个任务分成不同的部分运行；

(3) 提供协调机制，一方面防止进程之间和线程之间产生冲突，另一方面允许进程之间和线程之间共享资源。

结论：

(1) 线程是进程的一部分

(2) CPU调度的是线程

(3) 系统为进程分配资源，不对线程分配资源

## 17. 进程间的通信方式：

[https://blog.csdn.net/violet\\_echo\\_0908/article/details/51201278](https://blog.csdn.net/violet_echo_0908/article/details/51201278)

1) 有名管道(pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

2) 高级管道(popen)：将另一个程序当做一个新的进程在当前程序进程中启动，

则它算是当前程序的子进程，这种方式我们成为高级管道方式。

3) 有名管道 (named pipe) : 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

4) 消息队列( message queue ) : 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

5) 信号量( semaphore ) : 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

6) 信号 ( signal ) : 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

7) 共享内存( shared memory ) : 共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

8) 套接字( socket ) : 套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

## 18. 什么是 ORM? 为什么要用 ORM? 不用 ORM 会带来什么影响?

<https://blog.csdn.net/lisheng19870305/article/details/106919820>

ORM(Object-relational mapping)，对象关系映射。是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术。简单的说，ORM 是通过使用描述对象和数据库之间映射的元数据，将程序中的对象自动持久化到关系数据库中。  
<https://www.bilibili.com/video/BV1uJ411N7nh?from=search&seid=2010137152461554789>

## 19. 将数据库转化为对象，通过对对象对数据库进行操作 (entry)

20. with 的使用，和 try...except... 有啥不同

<https://www.bilibili.com/video/BV137411k7gv?from=search&seid=7525362641510529605>

[https://blog.csdn.net/sinat\\_38682860/article/details/108489882](https://blog.csdn.net/sinat_38682860/article/details/108489882)

with 语句适用于对资源进行访问的场合，确保不管使用过程中是否发生异常都会执行必要的“清理”操作，释放资源。比如文件使用后自动关闭 / 线程中锁的自动获取和释放等。

```
1 | with open("1.txt") as file:  
2 |     data = file.read()
```

with 工作原理：

(0) 不是所有对象都可以放在 with 语句中。能放到 with 语句中的对象都有一个特性：具有 `__enter__()` 和 `__exit__()` 两个私有函数

(1) 紧跟 with 后面的语句被赋值后，返回对象的“`__enter__()`”方法被调用，这个方法的返回值将被赋值给 as 后面的变量；

(2) 当 with 后面的代码块全部被执行完之后，将调用前面返回对象的“`__exit__()`”方法。

with 工作原理代码示例：

```
5 class Sample:  
6     def __enter__(self):  
7         print "in __enter__"  
8         return "Foo"  
9     def __exit__(self, exc_type, exc_val, exc_tb):  
10        print "in __exit__"  
11    def get_sample():  
12        return Sample()  
13 with get_sample() as sample:  
14     print "Sample: ", sample
```

代码的运行结果如下：

```
1 in __enter__  
2 Sample: Foo  
3 in __exit__
```

可以看到，整个运行过程如下：

- (1) enter()方法被执行；
- (2) enter()方法的返回值，在这个例子中是” Foo”，赋值给变量 sample；
- (3) 执行代码块，打印 sample 变量的值为” Foo”；
- (4) exit()方法被调用；

总结，在 with 后面的代码块抛出异常时，exit()方法被执行。清理资源，关闭文件等操作都可以放在 exit()方法中。总之，with-as 极大的简化了每次写 finally 的工作。

try--except--else--finally：



21. 何时 cpu 处理进程最慢？？？

22. 系统资源包括哪些？

<https://www.anhuilife.com/wan-179755915452033364.html>

包括：各种计算机硬件设备和基本系统程序。

计算机硬件设备：CPU、内存、辅存、显示控制器、显示器和键盘等输入输出设备构成。

基本系统程序：处理机管理、储存管理、I/O 管理(输入输出)设备管理、文件管理、运行程序管理等。

## 23. python（参数传递机制、可变参数与不可变参数区别、参数类型、深拷贝与浅拷贝、多线程）

### (1) 参数传递机制

Python 函数参数由实参传递给形参的过程，是由参数传递机制来控制的，根据实参类型不同，函数参数传递方式分为：值传递和引用传递（又称为地址传递）。

1) 值传递：实际上就是将实参的副本传入函数，而参数本身不会受到任何影响。值传递的实质：当系统开始执行函数时，系统对形参执行初始化，就是把实参变量的值赋给函数的形参变量，在函数中操作的并不是实际的实参变量。

如果传入的参数是不可变对象，如数字，元组，字符串，这时就是值传递。

2) 引用传递：如果实际参数的数据类型是可变对象（列表、字典），则函数参数的传递方式将采用引用传递方式。

需要注意的是，引用传递方式的底层实现，采用的依然还是值传递的方式。

结论：

不管什么类型的参数，在 Python 函数中对参数直接使用“=”符号赋值是没用的，直接使用“=”符号赋值并不能改变参数。

如果需要让函数修改某些数据，则可以通过把这些数据包装成列表、字典等可变对象，然后把列表、字典等可变对象作为参数传入函数，在函数中通过列表、字典的方法修改它们，这样才能改变这些数据。

## 24. 直接赋值、深拷贝和浅拷贝

<https://www.runoob.com/w3cnote/python-understanding-dict-copy-shallow-or-deep.html>

在 Python 中，对象赋值实际上是对象的引用。当创建一个对象，然后把它赋给另一个变量的时候，Python 并没有拷贝这个对象，而只是拷贝了这个对象的引用，我们称之为浅拷贝。

1) 直接赋值：其实就是对象的引用（别名）。

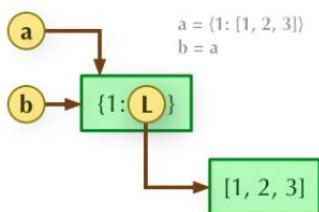
2) 浅拷贝 (copy)：拷贝父对象，不会拷贝对象的内部的子对象。

3) 深拷贝 (deepcopy)： copy 模块的 deepcopy 方法，完全拷贝了父对象及其子对象。

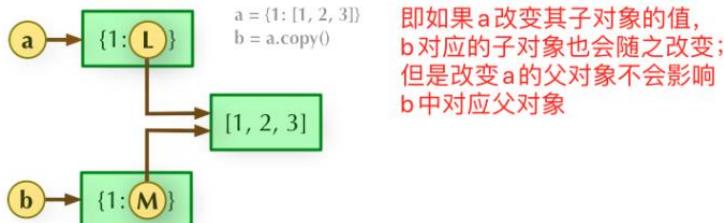
举例，append( )函数。

### 解析

1、`b = a`: 赋值引用，a 和 b 都指向同一个对象。

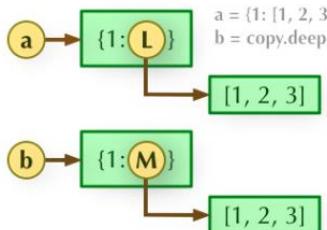


2、`b = a.copy()`: 浅拷贝，a 和 b 是一个独立的对象，但他们的子对象还是指向统一对象（是引用）。



即如果 a 改变其子对象的值，  
b 对应的子对象也会随之改变；  
但是改变 a 的父对象不会影响  
b 中对应父对象

`b = copy.deepcopy(a)`: 深度拷贝，a 和 b 完全拷贝了父对象及其子对象。两者是完全独立的。



以下实例是使用 copy 模块的 `copy.copy` ( 浅拷贝 ) 和 `(copy.deepcopy)` :

### 实例

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import copy
a = [1, 2, 3, 4, ['a', 'b']] #原始对象

b = a                      #赋值, 传对象的引用
c = copy.copy(a)            #对象拷贝, 浅拷贝
d = copy.deepcopy(a)        #对象拷贝, 深拷贝

a.append(5)                 #修改对象a
a[4].append('c')           #修改对象a中的['a', 'b']数组对象

print('a = ', a)
print('b = ', b)
print('c = ', c)
print('d = ', d)
```

以上实例执行输出结果为：

```
('a = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5])
('b = ', [1, 2, 3, 4, ['a', 'b', 'c'], 5])
('c = ', [1, 2, 3, 4, ['a', 'b', 'c']])
('d = ', [1, 2, 3, 4, ['a', 'b']])
```

直接赋值：全都变

浅拷贝：子变父不变

深拷贝：都不变

拷贝过来的是不变的，，，，，

append( )函数举例：

一个展示这种现象的示例如下（在 Python 交互式环境下）：

```
1 >>> l1 = [1,2,3]
2 >>> l2 = [3,2,1]
3 >>> l3 = [4,5,6]
4 >>> list = [l1,l2]
5 >>> list
6 [[1, 2, 3], [3, 2, 1]]
7 >>> list.append(l3)
8 >>> list
9 [[1, 2, 3], [3, 2, 1], [4, 5, 6]]
10 >>> l3.remove(4)
11 >>> l3
12 [5, 6]
13 >>> list
14 [[1, 2, 3], [3, 2, 1], [5, 6]]
```

复制

可以看到，在删除 l3 内的一个元素后，我们没有对 list 做任何处理，但 list 内通过 append(l3) 添加的元素却同步发生了变化。

这就像 C/C++ 中用指针当参数时一样，这些元素虽然用了不同的命名来表示，但在内存中指向的却是同一个位置，一旦这块内存中存储的值发生变化，则所有命名方式表示的值都会同步变化。

问题是由于 append() 添加的结果为原对象的引用导致的，那么我不直接 append 原对象，而是对原对象的每一个结果做个复制，然后再 append 这个复制体，即：将 list.append(nums) 修改为 list.append(nums.copy())

## 25. 代码：字符串转化为整数

字符串 str 转换成 int: int\_value = int(str\_value)

int 转换成字符串 str: str\_value = str(int\_value)

## 26. 面向对象语言的特性

类中包含：成员变量，成员函数

(1) 封装：封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。

(2) 继承：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。

(3) 多态：多态性 (polymorphism) 是允许你将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：**允许将子类类型的指针赋值给父类类型的指针。**

实现多态，有二种方式，覆盖，重载。

<https://www.liaoxuefeng.com/wiki/1252599548343744/1260455778791232>

## 27. 深拷贝与浅拷贝

<https://songlee24.github.io/2014/08/15/python-FAQ-02/>

## 28. 主键，外键 (done)

## 29. return 和 yeild

(1) return 是函数返回值，当执行到 return，后续的逻辑代码不再执行  
yield 是创建迭代器，可以用 for 来遍历，有点事件触发的意思

(2) return 是把数据遍历完成后一次性返回所有数据，在 for 循环外； yield

是每次输出一个数据，在 for 循环中

<https://www.jianshu.com/p/a3383b144eb6>

### 30. 慢查询

MySQL 会记录下查询超过指定时间的语句，我们将超过指定时间的 SQL 语句查询称为慢查询，都记在慢查询日志里，我们开启后可以查看究竟是哪些语句在慢查询

### 31. final 作用

被 final 关键字修饰的类不能被继承，被 final 关键字修饰的类属性和类方法不能被覆盖(重写)；

### 32. pytest 框架的结构；

pytest 使用：<https://testerhome.com/topics/23441>

结构：<https://juejin.cn/post/6873355392148865037>

<https://www.pythondf.cn/read/71712>

框架结构：

与 unittest 类似，执行用例前后会执行 setup, teardown 来增加用例的前置和后置条件。pytest 框架中使用 setup, teardown 更灵活，按照用例运行级别可以分为以下几类：

- (1) 模块级 (setup\_module/teardown\_module) 在模块始末调用
- (2) 函数级 (setup\_function/teardown\_function) 在函数始末调用 (在类外部)
- (3) 类级 (setup\_class/teardown\_class) 在类始末调用 (在类中)
- (4) 方法级 (setup\_method/teardown\_method) 在方法始末调用 (在类中)
- (5) 类里面的 (setup/teardown) 运行在调用方法的前后

调用顺序：setup\_module > setup\_class > setup\_method > setup > teardown > teardown\_method > teardown\_class > teardown\_module

### 33. 手写比较两个字典是否相等；

方法 1:直接使用==

```
1. a = {'a': 1, 'b': 2}
2. b = {'a': 1, 'b': 2}
3. c = {'a': 1, 'b': 3}
4.
5. print(a == b) # True
6. print(a == c) # False
```

方法 2:使用 operator.eq()

```
1. import operator
2.
3. a = {'a': 1, 'b': 2}
4. b = {'a': 1, 'b': 2}
5. c = {'a': 2, 'b': 2}
6. print(operator.eq(a, b)) # True
7. print(operator.eq(a, c)) # False
```

使用 comp( )

```

#!/usr/bin/python
# -*- coding: UTF-8 -*-

dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Mahnaz', 'Age': 27};
dict3 = {'Name': 'Abid', 'Age': 27};
dict4 = {'Name': 'Zara', 'Age': 7};
print "Return Value : %d" % cmp(dict1, dict2)
print "Return Value : %d" % cmp(dict2, dict3)
print "Return Value : %d" % cmp(dict1, dict4)

```

以上实例输出结果为：

```

Return Value : -1
Return Value : 1
(1) Return Value : 0

```

### 34. 遍历一个字典

- (1) 遍历 key 值： for key in dict.keys() :
- (2) 遍历 value 值： for value in dict.values() :
- (3) 遍历 key, value: for k, v in dict.items()
- (4) 遍历字典项： for kv in dict.item()

### 35. json.load、json.dump、json.loads、json.dumps 区别

[https://blog.csdn.net/Mr\\_EvanChen/article/details/77879967](https://blog.csdn.net/Mr_EvanChen/article/details/77879967)

- (1) json.dumps() 用于将 python 数据结构转化为 json
- (2) json.loads() 用于将 json 转化成 python 数据结构

如果是文件，而不是字符串，可使用 json.dump() 和 json.load() 来编码和解码 JSON 数据。

- (3) json.dump() 用于写入 json 数据
- (4) json.load() 用于从 json 文件中读取数据

### 36. 程序 core 掉的原因？（linux）

Core 的概念：当程序运行的过程中异常终止或崩溃，操作系统会将程序当时的内存状态记录下来，保存在一个文件中，这种行为就叫做 Core Dump。

原因：内存越界。空指针异常？？？

### 37. GIL (python 全局锁)

TUTORIALS

## 自动化测试-多线程&并发

### 38. Python 多线程适合 CPU 密集型程序吗？

进程适合 CPU 密集型程序，线程适合 IO 密集型程序

### 39. lambda 函数

Python 中有两种函数，一种是 def 定义的函数，另一种是 lambda 函数，即匿名函数。

- (1) lambda 函数介绍：

1) 语法：lambda arguments : expression #lambda 表示匿名函数 arguments 是参数，expression 是表达式。

2) lambda x: x \* x 相当于：

```

def f(x):
    return x * x

```

3) lambda 函数可接受任意数量的参数，但只能有一个表达式，不用写 return，返回值就是该表达式的结果。

### (2) lambda 函数优势

- 1) 代码冗余：lambda 函数减少了代码的冗余
- 2) 不用命名：用 lambda 函数，不用费神地去命名一个函数的名字，可以快速的实现某项功能
- 3) 冲突：因为函数没有名字，不必担心函数名冲突
- 4) 对象：匿名函数也是一个函数对象，也可以把匿名函数赋值给一个变量，再利用变量来调用该函数：

### (3) 什么场景用 lambda 函数？

有些时候，不需要显式地定义函数，直接传入匿名函数更方便。如下：

在Python中，对匿名函数提供了有限支持。还是以 `map()` 函数为例，计算 $f(x)=x^2$ 时，除了定义一个 `f(x)` 的函数外，还可以直接传入匿名函数：

```
>>> list(map(lambda x: x * x, [1, 2, 3, 4, 5, 6, 7, 8, 9]))  
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

## 40. python 内置函数 map( )

### (1) 语法：map(function, iterable, ...)

- 1) `map()` 函数会根据提供的函数对指定序列做映射。
- 2) 第一个参数 `function` 以参数序列中的每一个元素调用 `function` 函数，返回包含每次 `function` 函数返回值的新列表。如下，

```
>>> def square(x) :          # 计算平方数  
...     return x ** 2  
...  
>>> map(square, [1,2,3,4,5])    # 计算列表各个元素的平方  
<map object at 0x100d3d550>      # 返回迭代器  
>>> list(map(square, [1,2,3,4,5]))  # 使用 list() 转换为列表  
[1, 4, 9, 16, 25]  
>>> list(map(lambda x: x ** 2, [1, 2, 3, 4, 5]))  # 使用 lambda 匿名函数  
[1, 4, 9, 16, 25]  
>>>
```

## 41. 了解 dict 的底层结构吗

### 1. Python dict 底层结构

dict 底层使用的哈希表，支持快速查找（使用了哈希表作为底层结构（Cpython 解释器实现也使用哈希表）哈希表平均查找时间复杂度  $O(1)$ ，Cpython 解释器使用二次探查解决哈希冲突问题）。

面试经常被问到的哈希表的实现原理和底层细节，第一个问题是哈希表是怎么解决哈希冲突的，常用方法有链接法和探查法，探查法又分为线性探查和二次探查等。第二个问题是哈希表是如何扩容的，哈希表就是一个数组，不断去添加元素时候如何去触发扩容操作，当前数据存储的数量（即 `size()`）大小必须大于等于阈值；当前加入的数据是否发生了 hash 冲突，满足这两个条件就会哈希扩容。

### 2. Python list / tuple 区别

都是线性结构，支持下标访问

① `list` 是可变对象，`tuple` 是不可变对象，保存的引用不可变，如果 `tuple` 里面保存一个 `list`，`list` 本身还是可变的

```
>>> t = ([1], 2, 3)  
>>> t[2] = 3
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> t[0]
[1]
>>> t[0].append(1)
>>> t
```

([1, 1], 2, 3) # 第一个列表对象元素虽然改变，但它依然是这个列表对象

保存的引用不可变指的是，你没法替换这个对象，但是如果它本身就是一个可变对象，是可以修改这个引用指向的可变对象。

② list 没法作为字典的 key, tuple 可以（可变对象不可以 hash）。

## 42. dict 的 key 可以是 list 吗，为什么？

python 中，一个对象能不能做字典的 key，取决于该对象有没有 \_\_hash\_\_ 方法。没有 \_\_hash\_\_ 方法 (None) 的对象不可以作为字典的 key；有 \_\_hash\_\_ 方法的对象可以作为字典的 key。

python 自带类型中，除了 list、dict、set 和内部至少带有上述三种类型之一的 tuple 之外，其余的对象都能当 key。

注意，tuple 本身是有 \_\_hash\_\_ 方法的，但是如果元组中带包含不带 \_\_hash\_\_ 方法的对象，也不能做字典的 key。

## 43. 如果遇到哈希冲突怎么办？冲突点放单链表还是双链表？

<https://www.jianshu.com/p/4d3cb99d7580>

<https://zhuanlan.zhihu.com/p/29520044>

## 44. Python 怎么生成随机数？

<https://zhuanlan.zhihu.com/p/34395664>

首先需要：import random

(1) random.random()，用来随机生成一个 0 到 1 之间的浮点数，包括零。

In [1]: import random

In [2]: random.random()

Out[2]: 0.15790797219589303

(2) Random.randint(a, b)，用来生成[a, b]之间的随意整数，包括两个边界值。

In [12]: import random

In [13]: random.randint(1, 6)

Out[13]: 1

(3) random.uniform(a, b)：用来生成[a, b]之间的随意浮点数，包括两个边界值。

In [14]: import random

In [15]: random.uniform(1, 6)

Out[15]: 5.497873150216069

## 45. 数据库

## MySQL<sup>Q</sup>

### 1、MySQL 中有哪几种锁？

答：1、**表级锁**：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。

2、**行级锁**：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

3、**页面锁**：开销和加锁时间界于表锁和行锁之间；会出现死锁；**锁定粒度界于表锁和行锁之间，并发度一般。**

### 2.简述在 MySQL 数据库中 MyISAM 和 InnoDB 的区别？

答：**MyISAM**：不支持事务，但是每次查询都是原子的；支持**表级锁**，即每次操作是对整个表加锁；存储表的总行数；一个 MYISAM 表有三个文件：索引文件、表结构文件、数据文件；采用菲聚集索引，索引文件的数据域存储指向数据文件的指针。辅索引与主索引基本一致，但是辅索引不能保证唯一性。

**InnoDB**：支持 ACID 的事务，支持事务的四种隔离级别；支持行级锁及外键约束；因此可以支持写并发；不存储总行数

### 3.MySQL 中 InnoDB 支持的四种事务隔离级别名称，以及逐级之间的区别？

答：SQL 标准定义的四个隔离级别为：1、read uncommitted：读到未提交数据，最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。

2、read committed：脏读，不可重复读，允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生。

3、repeatable read：可重读，对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生

4、serializable：串行化，最高的隔离级别，完全服从 ACID 的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读

MySQL InnoDB 存储引擎的默认支持的隔离级别是 REPEATABLE-READ（可重读）

### 4.什么是脏读，缓读，不可重复读？

答：脏读：脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。

不可重复读：是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。例如，一个编辑人员两次读取同一文档，但在两次读取之间，作者重写了该文档。当编辑人员第二次读取文档时，文档已更改。原始读取不可重复。如果只有在作者全部完成编写后编辑人员才可以读取文档，则可以避免该问题。

幻读：是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。例如，一个编辑人员更改作者提交的文档，但当生产部门将其更改内容合并到该文档的主副本时，发现作者已将未编辑的新材料添加到该文档中。如果在编辑人员和生产部门完成对原始文档的处理之前，任何人都不能将新材料添加到文档中，则可以避免该问题。

不可重复读的重点是修改：同样的条件，你读取过的数据，再次读取出来发现值不一样了

幻读的重点：在于新增或者删除同样的条件，第 1 次和第 2 次读出来的记录数不一样

## 5.谈一谈MySQL的优化？

答：大体可以分为三部分：[索引的优化](#)，[sql语句的优化](#)，[表的优化](#)。在高并发网络环境下，除了优化数据库外，还会涉及到分布式缓存，CDN，数据库读写分离等高并发优化技术。

索引优化原则：

- 只要列中含有NULL值，就最好不要在此列设置索引，复合索引如果有NULL值，此列在使用时也不会使用索引
- 尽量使用短索引，如果可以，应该制定一个前缀长度
- 对于经常在where子句使用的列，最好设置索引，这样会加快查找速度
- 对于有多个列where或者order by子句的，应该建立复合索引
- 对于like语句，以%或者\_开头的不会使用索引，以%结尾会使用索引
- 尽量不要在列上进行运算（函数操作和表达式操作）
- 尽量不要使用not in和<>操作

sql语句的优化：

- 查询时，能不要\*就不用\*，尽量写全字段名
- 大部分情况连接效率远大于子查询
- 多使用explain和profile分析查询语句
- 查看慢查询日志，找出执行时间长的sql语句优化
- 多表连接时，尽量小表驱动大表，即小表 join 大表
- 在千万级分页时使用limit
- 对于经常使用的查询，可以开启缓存

数据库表优化：

- 表的字段尽可能用NOT NULL
- 字段长度固定的表查询会更快
- 把数据库的大表按时间或一些标志分成小表
- 将表拆分

数据表拆分：主要就是垂直拆分和水平拆分。

水平切分：将记录散列到不同的表中，各表的结构完全相同，每次从分表中查询，提高效率。

垂直切分：将表中大字段单独拆分到另外一张表，形成一对一的关系。

## 6.什么是内联接、左外联接、右外联接？

答：内联接（Inner Join）：匹配2张表中相关联的记录。

左外联接（Left Outer Join）：除了匹配2张表中相关联的记录外，还会匹配左表中剩余的记录，右表中未匹配到的字段用NULL表示。[查询关联和左表全部](#)

右外联接（Right Outer Join）：除了匹配2张表中相关联的记录外，还会匹配右表中剩余的记录，左表中未匹配到的字段用NULL表示。[查询关联和右表全部](#)

## 7. 如何通俗地理解三个范式？

答：第一范式：1NF 是对属性的原子性约束，要求属性具有原子性，不可再分解；第一范式的目的是确保每列的原子性：如果每列都是不可再分的最小数据单元（也称为最小的原子单元），则满足第一范式（1NF）：

The diagram illustrates the decomposition of a 1NF table into two 2NF tables. On the left, a single table 'BuyerID' has columns 'BuyerID' and 'Address'. It contains four rows: 1 (中国北京市), 2 (美国纽约市), 3 (英国利物浦), and 4 (日本东京市). An arrow points to the right, leading to two separate tables: 'BuyerID' and 'Country'. The 'BuyerID' table has columns 'BuyerID' and 'Country', with rows 1 (中国) and 2 (美国). The 'Country' table has columns 'Country' and 'City', with rows 1 (中国, 北京), 2 (中国, 纽约), 3 (日本, 东京), and 4 (日本, 中国).

BuyerID	Address
1	中国北京市
2	美国纽约市
3	英国利物浦
4	日本东京市
...	...

BuyerID	Country	City
1	中国	北京
1	中国	北京
4	日本	东京
2	美国	纽约
...	...	...

第二范式：2NF 是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；首先满足第一范式，并且表中非主键列不存在对主键的部分依赖。第二范式要求每个表只描述一件事情。

The diagram illustrates the decomposition of a 2NF table into two 3NF tables. On the left, a table 'Orders' has columns '字段' and '例子'. It contains four rows: 订单编号 001, 产品编号 A001, 订购日期 2000-2-3, and 价格 \$29.00. An arrow points to the right, leading to two separate tables: 'Orders' and 'Products'. The 'Orders' table has columns '字段' and '例子', with rows 订单编号 001 and 订购日期 2000-2-3. The 'Products' table has columns '字段' and '例子', with rows 产品编号 A001 and 价格 \$29.00.

Orders	
字段	例子
订单编号	001
产品编号	A001
订购日期	2000-2-3
价格	\$29.00
...	...

Orders	
字段	例子
订单编号	001
订购日期	2000-2-3
产品编号	A001
价格	\$29.00

第三范式：3NF 是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。第三范式定义是，满足第二范式，并且表中的列不存在对非主键列的传递依赖。除了主键订单编号外，顾客姓名依赖于非主键顾客编号。

The diagram illustrates the decomposition of a 3NF table into two 3NF tables. On the left, a table 'Orders' has columns '字段' and '例子'. It contains five rows: 订单编号 001, 订购日期 2000-2-3, 顾客编号 AB001, 顾客姓名 Tony, and ... . An arrow points to the right, leading to two separate tables: 'Orders' and 'Customer'. The 'Orders' table has columns '字段' and '例子', with rows 订单编号 001 and 订购日期 2000-2-3. The 'Customer' table has columns '字段' and '例子', with rows 顾客编号 AB001 and 顾客姓名 Tony.

Orders	
字段	例子
订单编号	001
订购日期	2000-2-3
顾客编号	AB001
顾客姓名	Tony
...	...

Orders	
字段	例子
订单编号	001
订购日期	2000-2-3
顾客编号	AB001
顾客姓名	Tony
...	...

## 8. 什么是基本表？什么是视图？试述视图的优点？

答：基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表。

优点：(1) 视图能够简化用户的操作 (2) 视图使用户能以多种角度看待同一数据；  
(3) 视图为数据库提供了一定程度的逻辑独立性；(4) 视图能够对机密数据提供安全保护

## 9. MySQL 数据库作发布系统的存储，一天五万条以上的增量，预计运维三年，怎么优化？

答：1、设计良好的数据库结构，允许部分数据冗余，尽量避免 join 查询，提高效率。

- 2、选择合适的表字段数据类型和存储引擎，适当的添加索引。
- 3、MySQL 库主从读写分离。
- 4、找规律分表，减少单表中的数据量提高查询速度。
- 5、添加缓存机制，比如 memcached, apc 等。
- 6、不经常改动的页面，生成静态页面。
- 7、书写高效率的 SQL。比如 SELECT \* FROM TABLE 改为 SELECT f1,f2,f3 FROM TABLE.

## 10.简单描述 MySQL 中，索引，主键，唯一索引，联合索引的区别，对数据库的性能有什么影响（从读写两方面）？

答：索引是一种特殊的文件( InnoDB 数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

普通索引(由关键字 KEY 或 INDEX 定义的索引)的唯一任务是加快对数据的访问速度。

普通索引允许被索引的数据列包含重复的值。如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用

关键字 UNIQUE 把它定义为一个唯一索引。也就是说，唯一索引可以保证数据记录的唯一性。

主键，是一种特殊的唯一索引，在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字 PRIMARY KEY 来创建。

索引可以覆盖多个数据列，如像 INDEX(columnA, columnB)索引，这就是联合索引。

索引可以极大的提高数据的查询速度，但是会降低插入、删除、更新表的速度，因为在执行这些写操作时，还要操作索引文件

## 11.简单聊聊你所认知的分库分表？

答：分表，能够解决单表数据量过大带来的查询效率下降的问题；

分库，面对高并发的读写访问，当数据库master服务器无法承载写操作压力时，不管如何扩展slave服务器，此时都没有意义。此时，则需要通过数据分库策略，提高数据库并发访问能力。

优点，分库、分表技术优化了数据存储方式，有效减小数据库服务器的负担、缩短查询响应时间

## 12.聊聊你理解的索引，索引是不是越多越好？创建索引的原则是什么？

答：索引 (Index) 是帮助 MySQL 高效获取数据的数据结构。常见的查询算法，顺序查找，二分查找，二叉排序树查找，哈希散列法，分块查找，平衡多路搜索树 B 树 (B-tree)，索引是对数据库表中一个或多个列的值进行排序的结构，建立索引有助于快速获取信息。你也可以这样理解：索引就是加快检索表中数据的方法。数据库的索引类似于书籍的索引。在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息。在数据库中，索引也允许数据库程序迅速地找到表中的数据，而不必扫描整个数据库。

mysql 有4种不同的索引： 主键索引 (PRIMARY) 唯一索引 (UNIQUE) 普通索引 (INDEX) 全文索引 (FULLTEXT)

索引并非是越多越好，创建索引也需要耗费资源，一是增加了数据库的存储空间，二是在插入和删除时要花费较多的时间维护索引

1. 索引加快数据库的检索速度
2. 索引降低了插入、删除、修改等维护任务的速度
3. 唯一索引可以确保每一行数据的唯一性
4. 通过使用索引，可以在查询的过程中使用优化器，提高系统的性能
5. 索引需要占物理和数据空间

常见索引原则有：

1. 选择唯一性 索引，唯一性索引的值是唯一的，可以更快速的通过该索引来确定某条记录。
2. 为 经常需要排序、分组和联合操作的字段 建立索引。
3. 为 常用作为查询条件的字段 建立索引。
4. 限制索引的数目： 越多的索引，会使更新表变得很浪费时间。尽量使用数据量少的索引
5. 如果索引的值很长，那么查询的速度会受到影响。尽量使用前缀来索引
6. 如果索引字段的值很长，最好 使用值的前缀来索引。
7. 删除不再使用或者很少使用的索引
8. 最左前缀匹配原则，非常重要的原则。
9. 尽量选择区分度高的列作为索引区分度的公式是表示字段不重复的比例
10. 索引列不能参与计算，保持列“干净”： 带函数的查询不参与索引。
11. 尽量的扩展索引，不要新建索引

## 13. 数据库的事务？

答：事务(TRANSACTION)是作为单个逻辑工作单元执行的一系列操作，这些操作作为一个整体一起向系统提交，要么都执行、要么都不执行。事务是一个不可分割的工作逻辑单元，事务必须具备以下四个属性，简称 ACID 属性：

### 1、原子性 (Atomicity)

原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须完全应用到数据库，如果操作失败则不能对数据库有任何影响。

### 2、一致性 (Consistency)

一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。举例来说，假设用户A和用户B两者的钱加起来一共是1000，那么不管A和B之间如何转账、转几次账，事务结束后两个用户的钱相加起来应该还是1000，这就是事务的一致性。

### 3、隔离性 (Isolation)

隔离性是当多个用户并发访问数据库时，比如同时操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

### 4、持久性 (Durability)

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。例如我们在使用 JDBC 操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务已经正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成。否则的话就会造成我们虽然看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。这是不允许的。

## 14. LIKE 声明中的 % 和 \_ 是什么意思？

答：%对应于0个或更多字符，\_只是 LIKE 语句中的一个字符。

## 45. 1 数据库：查询成绩最高的人的姓名

```
select name from student where score = (select max(score) from student)
```

## 46. 自己建两个表并连表查询。

```
① create table person (
    p_id int not null primary key auto_increment comment '人员id',#关联字段
    p_name varchar(30) not null comment '人员名称',
    p_age int not null comment '人员年龄'
);

create table order (
    or_id int not null comment '订单id',
    or_name varchar(30) not null comment '订单名称',
    or_desc varchar(50) not null comment '订单描述',
    p_id int not null comment '订单人id',#关联字段
    foreign key (p_id) references person(p_id) #两表可以进行联合查询，前提是外键可以使量表关联
);

#两表查询：查询person表中有过订单记录的人，显示字段：p_id, p_name, or_id
select person.p_id, person.p_name, order.or_id from person right join order on person.p_id=order.p_id
```

(2) 表建立完成后，添加外键：

```
alter table order|
add foreign key p_fk (p_id) references person(p_id)
```

## 47. 数据库的索引是什么？怎么样实现的？举个例子？

<https://blog.csdn.net/weiliangliang111/article/details/51333169>

<https://crazyfzw.github.io/2018/07/18/RDBMS-INDEX/>

MySQL 数据库支持多种索引类型，如 BTree 索引，哈希索引，全文索引等等。其中 BTree 索引最常用。

索引 (Index)：是帮助数据库高效获取数据的一种数据结构。

索引的本质：索引是数据结构。

之所以要建立索引，其实就是为了构建一种数据结构，然后可以在上面应用一种高效的查询算法，最终提高数据的查询速度。

索引实现：B+树

索引优势：加快搜索速度

索引缺点：

- 1) 索引会占用空间。表越大，索引占用空间越大；
- 2) 性能损失。主要指更新操作，当在表中添加、删除或者更新行数据的时候，在索引中也会有相同的操作。建立在某列（或多列）索引需要保存该列最新的数据。

#### 48. 设计索引要注意什么？

- 1) 表记录比较少，没有必要建立索引
- 2) 索引的选择性较低。所谓索引的选择性（Selectivity），是指不重复的索引值（也叫基数，Cardinality）与表记录数（#T）的比值。选择性越高的索引价值越大，这是由 B+Tree 的性质决定的。

设计索引应注意：

- 1) **看数据量**，根据记录数，看是否有建索引的必要；
- 2) 根据计算字段的索引选择性判断给某个字段加索引是否比较有价值；
- 3) 看能否使用前缀索引取代全列索引，综合考虑索引选择性与 key 的长度，做个折中，尽可能使得前缀索引的选择性接近全列索引，同时又减小索引 key 的长度，从而减少了索引文件的大小和维护开销。

#### 49. 数据库事务的了解

略

##### 1. 主键，外键的了解

<https://www.jianshu.com/p/394f8aa724f4>

##### 2. 主键、外键和索引的区别

- |   |                                 |
|---|---------------------------------|
| 1 | 定义：                             |
| 2 | 主键：唯一标识一条记录，不能有重复，不允许为空。        |
| 3 | 外键：表的外键是另一表的主键，外键是可以有重复的，可以是空值。 |
| 4 | 索引：该字段没有重复值，但可以有一个空值。           |

- |   |                |
|---|----------------|
| 1 | 作用：            |
| 2 | 主键：用来保证数据完整性   |
| 3 | 外键：用来和其他表建立联系用 |
| 4 | 索引：用来提高查询排序的速度 |

- |   |                  |
|---|------------------|
| 1 | 个数：              |
| 2 | 主键：主键只能有一个。      |
| 3 | 外键：一个表可以有多个外键。   |
| 4 | 索引：一个表可以有多个唯一索引。 |

<b>id</b>	<b>name</b>	<b>other columns...</b>
1	一班	...
2	二班	...

但是我们如何确定 `students` 表的一条记录，例如，`id=1` 的小明，属于哪个班级呢？

由于一个班级可以有多个学生，在关系模型中，这两个表的关系可以称为“一对多”，即一个 `classes` 的记录可以对应多个 `students` 表的记录。

为了表达这对多的关系，我们需要在 `students` 表中加入一列 `class_id`，让它的值与 `classes` 表的某条记录相对应：

<b>id</b>	<b>class_id</b>	<b>name</b>	<b>other columns...</b>
1	1	小明	...
2	1	小红	...
5	2	小白	...

这样，我们就可以根据 `class_id` 这个列直接定位出一个 `students` 表的记录应该对应到 `classes` 的哪条记录。

例如：

- 小明的 `class_id` 是 `1`，因此，对应的 `classes` 表的记录是 `id=1` 的一班；
- 小红的 `class_id` 是 `1`，因此，对应的 `classes` 表的记录是 `id=1` 的一班；
- 小白的 `class_id` 是 `2`，因此，对应的 `classes` 表的记录是 `id=2` 的二班。

在 `students` 表中，通过 `class_id` 的字段，可以把数据与另一张表关联起来，这种列称为 `外键`。

外键并不是通过列名实现的，而是通过定义外键约束实现的：

```
ALTER TABLE students
ADD CONSTRAINT fk_class_id
FOREIGN KEY (class_id)
REFERENCES classes (id);
```

其中，外键约束的名称 `fk_class_id` 可以任意，`FOREIGN KEY (class_id)` 指定了 `class_id` 作为外键，`REFERENCES classes (id)` 指定了这个外键将关联到 `classes` 表的 `id` 列（即 `classes` 表的主键）。

通过定义外键约束，关系数据库可以保证无法插入无效的数据。即如果 `classes` 表不存在 `id=99` 的记录，`students` 表就无法插入 `class_id=99` 的记录。

由于外键约束会降低数据库的性能，大部分互联网应用程序为了追求速度，并不设置外键约束，而是仅靠应用程序自身来保证逻辑的正确性。这种情况下，`class_id` 仅是一个普通的列，只是它起到了外键的作用而已。

2. 一个人员表，一个部门表，人员表中存了部门 `id`，查人员表和部门表所有数据

```
select * from (select * from e full join d on e.id=d.id);
```

3. 查询一个城市列表里面重复的城市名，并且统计重复次数

```
select name as city_name, count(*) as count from cities group by name
having count>1
```

## 50. 什么是基本表？什么是视图？试述视图的优点？

<https://blog.csdn.net/helloxiaoze/article/details/80171793>

答：基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表。

优点：

- (1) 视图能够简化用户的操作
- (2) 视图使用户能以多种角度看待同一数据；
- (3) 视图为数据库提供了一定程度的逻辑独立性；
- (4) 视图能够对机密数据提供安全保护

## 51. `count(*)`, `count(字段)`, `count(1)` 的区别？

<https://blog.csdn.net/iFuMI/article/details/77920767>

(1) 执行效果上：

- (1) count(1) 会统计表中的所有的记录数，包含字段为 null 的记录。
- (2) count(字段) 会统计该字段在表中出现的次数，忽略字段为 null 的情况。即不统计字段为 null 的记录。
- 3) count(列名) 只包括列名那一列，在统计结果的时候，会忽略列值为空（这里的空不是只空字符串或者 0，而是表示 null）的计数，即某个字段值为 NULL 时，不统计。

(2) 执行效率上：

- 1) 列名为主键，count(列名) 会比 count(1) 快
- 2) 列名不为主键，count(1) 会比 count(列名) 快
- 3) 如果表多个列并且没有主键，则 count(1) 的执行效率优于 count(\*)
- 4) 如果有主键，则 select count(主键) 的执行效率是最优的
- 5) 如果表只有一个字段，则 select count(\*) 最优。

mysql 存储引擎、innodb 和 MyISAM 区别、索引的底层、底层为什么要采用 B 树或 B+ 树、B 树和 B+ 树区别、事务概念及其特性、多表查询、普通 sql 和存储过程的区别

52. 找出更新后的前十个数据；还有一个是考 group by；左连接右连接

53. sql 注入

[https://blog.csdn.net/github\\_36032947/article/details/78442189](https://blog.csdn.net/github_36032947/article/details/78442189)

(1) 如何理解 SQL 注入（攻击）？

SQL 注入是一种将 SQL 代码添加到输入参数中，传递到服务器解析并执行的一种攻击手法。

SQL 注入攻击是输入参数未经过滤，然后直接拼接到 SQL 语句当中解析，执行达到预想之外的一种行为，称之为 SQL 注入攻击。

(2) SQL 注入是怎么产生的？

- 1) WEB 开发人员无法保证所有的输入都已做过滤
- 2) 攻击者利用发送给 SQL 服务器的输入参数构造可执行的 SQL 代码（可加入到 get 请求、post 请求、http 头信息、cookie 中）
- 3) 数据库未做相应的安全配置

54. redis 攻击方法？

<https://www.cnblogs.com/sdgfsdgfsd/p/13332146.html>

55. Hbase、redis、es 了解

56. 数据库删除有几种方式，他们的区别是什么

3 种：<https://zhuanlan.zhihu.com/p/89113378>

(1) DELETE 语句执行删除的过程是每次从表中删除一行，并且同时将该行的删除操作作为事务记录在日志中保存以便进行回滚操作。

(2) TRUNCATE TABLE 则一次性地从表中删除所有的数据并不把单独的删除操作记录记入日志保存，删除行是不能恢复的。并且在删除的过程中不会激活与表有关的删除触发器。执行速度快。TRUNCATE 只能对 TABLE；DELETE 可以是 table

## 和 view

(3) DROP 则删除整个表（结构和数据）。TRUNCATE 和 DELETE 只删除数据表和索引所占空间。当表被 TRUNCATE 后，这个表和索引所占用的空间会恢复到初始大小

## 4. 慢查询: <https://tech.meituan.com/2014/06/30/mysql-index.html>

与索引相关

### 57. 股票最大利润

我们需要找出给定数组中两个数字之间的最大差值（即，最大利润）。此外，第二个数字（卖出价格）必须大于第一个数字（买入价格）。

形式上，对于每组  $i$  和  $j$ （其中  $j > i$ ）我们需要找出  $\max(prices[j] - prices[i])$ 。

# 此方法会超时

class Solution:

```
def maxProfit(self, prices: List[int]) -> int:
    ans = 0
    for i in range(len(prices)):
        for j in range(i + 1, len(prices)):
            ans = max(ans, prices[j] - prices[i])
    return ans
```

### 58. 删掉排序数组中的重复项，然后让你写测试用例自测一下

```
1 class Solution(object):
2     def removeDuplicates(self, nums):
3         """
4             :type nums: List[int]
5             :rtype: int
6         """
7         i=0
8         while i<len(nums)-1:
9             if nums[i]==nums[i+1]:
10                 #pop返回的是被移除的元素，remove返回的是移除元素后的列表
11                 nums.pop(i) #list.pop(i)，注意是按照索引pop
12             else:
13                 i+=1
14         return len(nums)
15
16     for i in range(len(nums) - 1):
17         if nums[i] == nums[i + 1]:
18             nums.pop(i)
19     return len(nums)
```

不可以用for循环，因为list长度是变化的

测试用例

代码执行结果

调试器 Beta

执行结果： 执行出错

```
> IndexError: list index out of range
if nums[i] == nums[i + 1]:
Line 17 in removeDuplicates (Solution.py)
```

<https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array/>

### 59. 寻找缺失的第一个正数，说思路，自测

<https://leetcode-cn.com/problems/first-missing-positive/>

1. 给出一个数组，如[7864, 284, 347, 7732, 8498]，现在需要将数组中的数字拼接起来，如按顺序依次拼接为：78642843477328498，数组中的数字拼接顺序可以任意，编写程序，返回「最大的可能拼出的数字」。（以上面数组为例，返回：849878647732347284）

4. (1) 有一个文件，文件中每一行为一个 HTTP 请求响应的 log，每行的 log 格式为：[HTTP URL] [请求响应码，如 200, 404] [请求响应时间，以 ms 为单位]  
例如`http://www.baidu.com 200 345`。

求一下所有响应码为 200 的请求的响应时间平均值，单位 ms。

(2) 并针对以上程序，设计测试用例

5. 输入队列，队列元素为不限位数的数字字符串。要求输出队列元素排列组成的最大数字字符串。

60. 统计一个字符串中出现次数最多的字母和次数。

```
1 | strs = input('请输入你的字符串: ')
2 | dicts = {}
3 |
4 | for i in strs:
5 |     dicts[i] = strs.count(i) # 构造字典, key=字母, value=字母次数, 次数用count统计
6 |
7 | max_zimu = max(dicts.values())
8 |
9 | for key,value in dicts.items():
10 |     if value==max_zimu:
11 |         print(key,value)
```

解题思路：输入的字符串存储到一个字典中，关键语句`dicts[i] = strs.count(i)`，然后取出字典中value最大的值，作用在后面与字典的遍历中取值比较。

```
1 | strs = input('请输入你的字符串: ')
2 | result = {}
3 | for x in strs:
4 |     result[x] = result.get(x,0)+1
5 | print(result)
```

这种方法是直接查看全部key的次数

```
请输入你的字符串: abcbcaab
a 3
-----
{'a': 3, 'b': 2, 'c': 2, 's': 1}
-----
Process finished with exit code 0
```

61. 给定字符串如何判断是否符合 ip 地址？

62. 在不排序的情况下，怎么样的得到一个先大后小的数组的最大值？

63. 模仿微信拼手气红包，总共 m 元，随机放到 n 个红包里？

64. 编程：判断一个字符串是否符合 ipv4 格式，对此代码编写测试用例

65. 一个数组中出现最多的数

数组Q中有一个数字出现的次数超过数组长度的一半，请找出这个数字。例如输入一个长度为9的数组{1,2,3,2,2,2,5,4,2}。由于数字2在数组中出现了5次，超过数组长度的一半，因此输出2。如果不存在则输出0。

```
1 class Solution {
2 public:
3     int MoreThanHalfNum_Solution(vector<int> numbers) {
4         sort(numbers.begin(),numbers.end());
5         int n=numbers.size();
6         int c=0;
7         for(int i=0;i<n;i++){
8
9             if((n%2==1)&&numbers[i]==numbers[(n-1)/2+i])
10             {c=numbers[i];}
11             else if((n%2==0)&&numbers[i]==numbers[n/2+i])
12             {c=numbers[i];}
13
14         }
15         return c;
16     }};

```

复制

## 66. 一组数组，找出出现超过一半的数。

现在有一个数组，已知一个数出现的次数超过了一半，请用O(n)的复杂度的算法找出这个数。

思路1：

创建一个hash\_map，key为数组中的数，value为此数出现的次数。遍历一遍数组，用hash\_map统计每个数出现的次数，并用两个值存储目前出现次数最多的数和对应出现的次数。

这样可以做到O(n)的时间复杂度Q和O(n)的空间复杂度，满足题目要求。

但是没有利用“一个数出现的次数超过了一半”这个特点。也许算法还有提高的空间。

思路2（好）：

使用两个变量A和B，其中A存储某个数组中的数，B用来计数。开始时将B初始化为0。

遍历数组，如果B=0，则令A等于当前数，令B等于1；如果当前数与A相同，则B=B+1；如果当前数与A不同，则令B=B-1。遍历结束时，A中的数就是要找的数。

这个算法的时间复杂度是O(n)，空间复杂度Q为O(1)。

## 67. 代码（1、两个链表的第一个公共结点 2、跳台阶）

<https://www.nowcoder.com/practice/6ab1d9a29e88450685099d45c9e31e46?tpId=117&tab=answerKey>

<https://www.nowcoder.com/practice/8c82a5b80378478f9484d87d1c5f12a4?tpId=117&tab=answerKey>

## 68. 骆峰字符串问题，给定一个骆峰样式的字符串例如“AaAbCbbcvQv.....” →“bc”，两个一样的字符夹着一个不一样的字符，处理这个字符串去掉所有的骆峰

### 69. 如何判断字符串是否为完全循环字符串

字符串[::1]输出判定是否与原字符串相等

### 70. 设计一个取模计算器

### Python中的取模运算和取余运算

取模运算和取余运算是两个概念，虽然他们有重叠部分，但又不一致。不一致的地方在于对负整数进行除法时，操作不一样。

对于整数 a 和 b，进行取模运算和取余运算可以总结分为 2 个步骤：

1. 计算整数商： $c = \text{取整}(a / b)$ ；
2. 计算模或余数： $r = a - c * b$ 。

两者的区别就在于第 1 步中的计算整数商不同，取模是向负无穷方向取整（即向下取整），取余是向 0 方向取整（即商大于 0 时向下取整，小于 0 时向上取整）。

Python3 中两个运算实现方式：

```
1 # 取模, Python中可直接用%, 计算模, r = a % b
2 def mod(a, b):
3     c = a // b
4     r = a - c * b
5     return r
6
7 # 取余
8 def rem(a, b):
9     c = int(a / b)
10    r = a - c * b
11    return r
```

## 71. 如何判断 linkedlist 是否有环

题目：Given a linked list, determine if it has a cycle in it.

Follow up:

Can you solve it without using extra space?

思路：快指针、慢指针。有环的话，一定不会遇到NULL；没环的话，fast一定先到NULL。该算法的时间复杂度为O(N)，空间复杂度为O(1)。

代码：

```
1 /**
2  * Definition for singly-linked List.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8 */
9 class Solution {
10 public:
11     bool hasCycle(ListNode *head) {
12         ListNode *slow, *fast;
13         slow = fast = head;
14         while(fast != NULL && fast->next != NULL)
15         {
16             slow = slow->next;
17             fast = fast->next->next;
18             if(slow == fast)
19                 return true;
20         }
21         return false;
22     }
23 };
```

复制

## 72. 最长公共子串

### 73. 字符串子串反转；

字符串括号匹配个数（多种算法：栈、动态规划），什么情况可以用动态规划

<https://www.nowcoder.com/study/live/489/1/1>

### 74. 21. [1, 2, 3, 4, 5, 6, 7]指定步长 n 反转，要求空间复杂度为 1

n=2 输出：[2, 1, 4, 3, 6, 5, 7]

n=3 输出：[3, 2, 1, 6, 5, 4, 7]

n=4 输出：[4, 3, 2, 1, 7, 6, 5]

### 75. 数据结构：什么是平衡二叉树；撕代码：判断一棵二叉树是不是平衡二叉树；

## 76. 怎么找到两个链表的第一个公共节点

22. (1) 数组和链表的区别 (2) 找出两个链表重合的部分 (只用说思路) (3) 写代码: 给一堆字符串, 找这些字符串的最长公共前缀 (4) 你代码的时间复杂度

## 77. 算法题: 罗马字符转整数

## 78. 用一行代码实现 1 到 100 累加

方法一: `sum([i for i in range(1, 101)])`

方法二: `sum(range(1, 101))`

`sum` 函数用法:

`sum()` 方法对序列进行求和计算。

### 语法

以下是 `sum()` 方法的语法:

```
sum(iterable[, start])
```

### 参数

- `iterable` -- 可迭代对象, 如: 列表、元组、集合。
- `start` -- 指定相加的参数, 如果没有设置这个值, 默认为0。

### 返回值

返回计算结果。

## 79. 计算机网络

1. 打开浏览器, 从输入 `http://www.baidu.com` 到看到浏览器显示页面, 这个过程中, 都有哪些步骤和环节?

<https://juejin.cn/post/6844903616101220366>

<https://segmentfault.com/a/1190000006879700>

<https://blog.fundebug.com/2019/02/28/what-happens-from-url-to-webpage/>

/

- (1) URL 输入
- (2) DNS 解析
- (3) TCP 连接
- (4) 发送 HTTP 请求
- (5) 负载均衡
- (6) 服务器处理请求并返回 HTTP 报文
- (7) 浏览器解析渲染页面
- (8) 连接结束

## 80. DNS 解析之前, 还完成了哪些步骤?

查询 ip 地址, 即网址到 ip 地址的转换。步骤如下:

查找顺序: 浏览器缓存--> 操作系统缓存--> 本地 host 文件 --> 路由器缓存--> ISP DNS 缓存 --> 顶级 DNS 服务器/根 DNS 服务器。

DNS 查询的两种方式: 递归查询和迭代查询。

第一步：客户机提出：域名解析请求，并将该请求发送给本地的域名服务器。

第二步：当本地的域名服务器收到请求后，就先查询本地的缓存，如果有该纪录项，则本地的域名服务器就直接把查询的结果返回。

第三步：如果本地的缓存中没有该纪录，则本地域名服务器就直接把请求发给根域名服务器，然后根域名服务器再返回给本地域名服务器一个所查询域（根的子域）的主域名服务器的地址。

第四步：本地服务器再向上一步返回的域名服务器发送请求，然后接受请求的服务器查询自己的缓存，如果没有该纪录，则返回相关的下级的域名服务器的地址。

第五步：重复第四步，直到找到正确的纪录。

第六步：本地域名服务器把返回的结果保存到缓存，以备下一次使用，同时还将结果返回给客户机。

## 81. 网络四层协议

应用层、传输层、网际层、网路接口层

## 82. http 和 https（证书，加密）

<https://zhuanlan.zhihu.com/p/72616216>

<https://blog.csdn.net/xionghuixionghui/article/details/68569282>

## 83. (1) HTTP 和 HTTPS 的基本概念

- 1) HTTP：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从 WWW 服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。
- 2) HTTPS：是以安全为目标的 HTTP 通道，简单讲是 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。

HTTPS 协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

## 84. HTTP 与 HTTPS 有什么区别？

http 协议和 https 协议的区别：传输信息安全性不同、连接方式不同、端口不同、证书申请方式不同

### 传输信息安全性不同：

- 1) http 协议：是超文本传输协议，信息是明文传输。如果攻击者截取了 Web 浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息。
- 2) https 协议：是具有安全性的 ssl 加密传输协议，为浏览器和服务器之间的通信加密，确保数据传输的安全。

### 连接方式不同：

- 1) http 协议：http 的连接很简单，是无状态的。
- 2) https 协议：是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议。

### 端口不同：

- 1) http 协议：使用的端口是 80。
- 2) https 协议：使用的端口是 443.

### 证书申请方式不同

1、http 协议：免费申请。（不是不需要证书！！！）

2、https 协议：需要到 ca 申请证书，一般免费证书很少，需要交费。

资源消耗：

和 HTTP 通信相比，Https 通信会由于加减密处理消耗更多的 CPU 和内存资源；

### 85. 5. 如何理解 http 是无状态的？

HTTP 协议是无状态的，指的是协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。也就是说，打开一个服务器上的网页和你之前打开这个服务器上的网页之间没有任何联系。HTTP 是一个无状态的面向连接的协议，无状态不代表 HTTP 不能保持 TCP 连接，更不能代表 HTTP 使用的是 UDP 协议（无连接）。

### 86. TCP 三次握手

1 略

TCP 连接后就进行 http 数据传输吗？？

<https://zhuanlan.zhihu.com/p/61423830>

### 87. 长连接和短连接？

<https://www.cnblogs.com/0201zcr/p/4694945.html>

<https://www.jianshu.com/p/3fc3646fad80>

明确：在 HTTP/1.0 中，默认使用的是短连接；HTTP/1.1 起，默认使用长连接。

HTTP 协议的长连接和短连接，实质上是 TCP 协议的长连接和短连接。

通常的短连接操作步骤是：tcp 连接建立→数据传输→关闭 tcp 连接；

而长连接通常就是：tcp 连接建立→数据传输→保持连接（心跳）→数据传输→保持连接（心跳）→……→关闭 tcp 连接；

长连接使用场景：长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况，比如数据库的连接

短连接使用场景：并发量大，但每个用户无需频繁操作情况下需用短连好，比如 web 网站的 http 服务

### 88. TCP 和 UDP 的区别，主要是运用在哪些方面的？

<https://segmentfault.com/a/1190000021815671>

区别：

	TCP	UDP
可靠性	可靠	不可靠
连接性	面向连接	无连接
报文	面向字节流	面向报文
效率	传输效率低	传输效率高
双工性	全双工	一对一、一对多、多对一、多对多
流量控制	滑动窗口	无
拥塞控制	慢开始、拥塞避免、快重传、快恢复	无
传输速度	慢	快
应用场景	对效率要求低，对准确性要求高或者要求有连接的场景	对效率要求高，对准确性要求低

应用场景：

应用层协议	应用	传输层协议
<b>SMTP</b>	电子邮件	TCP
<b>TELNET</b>	远程终端接入	
<b>HTTP</b>	万维网	
<b>FTP</b>	文件传输	
<b>DNS</b>	域名转换	UDP
<b>TFTP</b>	文件传输	
<b>SNMP</b>	网络管理	
<b>NFS</b>	远程文件服务器	

### 89. 一个页面打开空白的原因, 怎么去判断是哪里出了问题?

- (1) 看下控制台是不是前端的错误导致的页面加载失败, 如果有相关的错误的话, 排查错误原因; 或者前端路由路径错误, 找前端开发确认下;
- (2) 如果前端完全没有错误信息的话, 查看接口访问是否请求正常, 再找后端确认下是不是后端的配置或者后端错误导致的页面显示失败; 或者后端服务崩溃  
注意: 网络问题不会白屏, 浏览器会告诉你网络不通;

90. http 请求方式有哪些？什么情况下使用的？(bilibili)

<https://zhuanlan.zhihu.com/p/136995030>

<https://www.runoob.com/tags/html-httpmethods.html>

<https://www.cnblogs.com/hyddd/archive/2009/03/31/1426026.html>

91. Get, post, put, delete, option

(1) GET 请求会显示请求指定的资源。一般来说 GET 方法应该只用于数据的读取，而不应当用于会产生副作用的非幂等的操作中。

GET 会请求指定的页面信息，并返回响应主体，GET 被认为是不安全的方法，因为 GET 方法会被网络蜘蛛等任意的访问。

(2) HEAD 方法与 GET 方法一样，都是向服务器发出指定资源的请求。但是，**服务器在响应 HEAD 请求时不会回传资源的内容部分**，即：响应主体。这样，我们可以不传输全部内容的情况下，就可以获取服务器的响应头信息。HEAD 方法常被用于客户端查看服务器的性能。

(3) POST 请求会向指定资源提交数据，请求服务器进行处理，如：表单数据提交、文件上传等，请求数据会被包含在请求体中。POST 方法是非幂等的方法，因为这个请求可能会创建新的资源或/和修改现有资源。

(4) PUT 请求会向指定资源位置上传其最新内容，PUT 方法是幂等的方法。通过该方法客户端可以将指定资源的最新数据传送给服务器取代指定的资源的内容。

(5) DELETE 请求用于请求服务器删除所请求 URI (统一资源标识符，Uniform Resource Identifier) 所标识的资源。DELETE 请求后指定资源会被删除，DELETE 方法也是幂等的。

(6) CONNECT 方法是 HTTP/1.1 协议预留的，能够将连接改为管道方式的代理服务器。通常用于 SSL 加密服务器的链接与非加密的 HTTP 代理服务器的通信。

(7) OPTIONS 请求与 HEAD 类似，一般也是用于客户端查看服务器的性能。这个方法会请求服务器返回该资源所支持的所有 HTTP 请求方法，该方法会用'\*'来代替资源名称，向服务器发送 OPTIONS 请求，可以测试服务器功能是否正常。JavaScript 的 XMLHttpRequest 对象进行 CORS 跨域资源共享时，就是使用 OPTIONS 方法发送嗅探请求，以判断是否有对指定资源的访问权限。

(8) TRACE 请求服务器回显其收到的请求信息，该方法主要用于 HTTP 请求的测试或诊断。

总结：get, delete, put 是幂等的，post 是非幂等的

92. get 和 post 请求的区别是什么？

## 比较 GET 与 POST

下面的表格比较了两种 HTTP 方法：GET 和 POST。

	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
书签	可收藏为书签	不可收藏为书签
缓存	能被缓存	不能缓存
编码类型	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data。为二进制数据使用多重编码。
历史	参数保留在浏览器历史中。	参数不会保存在浏览器历史中。
对数据长度的限制	是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。	无限制。
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。 在发送密码或其他敏感信息时绝不要使用 GET！	POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。

GET 和 POST 是 HTTP 协议中的两种发送请求的方法，而 HTTP 是基于 TCP/IP 的关于数据如何在万维网中如何通信的协议。GET 产生一个 TCP 数据包；POST 产生两个 TCP 数据包。对于 GET 方式的请求，浏览器会把 http header 和 data 一并发出去，服务器响应 200（返回数据）；

而对于 POST，浏览器先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）。

1. GET 与 POST 都有自己的语义，不能随便混用。
2. 据研究，在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的 TCP 在验证数据包完整性上，有非常大的优点。
3. 并不是所有浏览器都会在 POST 中发送两次包，Firefox 就只发送一次。

93. 客户端发起 http 请求连接服务器之后，是可以多次传输数据，还是每次都要发起请求？（分长连接和短连接两种情况）

HTTP 协议是一个无状态的协议，同一个客户端的这次请求和上次请求是没有对应关系。所以当客户端向服务器端发送请求，服务器端响应完毕后，两者断开连接，也不保存连接状态。下一次客户端向同样的服务器发送请求时，需要重新建立连接。

针对无状态的一些解决策略：cookie、持久连接（HTTP keep-alive）等等

94. 什么是进程和线程？进程和线程的区别？进程和线程选择？线程拥塞是怎么回事？举个例子？

<https://zhuanlan.zhihu.com/p/38240894>

<https://blog.csdn.net/linux12121/article/details/51786233/>

答：进程是指在系统中正在运行的一个应用程序；程序一旦运行就是进程，或者

更专业化来说：进程是指程序执行时的一个实例，即它是程序已经执行到课中程度的数据结构的汇集。从内核的观点看，**进程的目的就是担当分配系统资源(CPU时间、内存等)的基本单位**。线程是系统分配处理器时间资源的基本单元，或者说进程之内独立执行的一个单元执行流。**进程——资源分配的最小单位，线程——程序执行的最小单位。**

线程进程的区别体现在 4 个方面：

第一：因为进程拥有独立的堆栈空间和数据段，所以每当启动一个新的进程必须分配给它独立的地址空间，建立众多的数据表来维护它的代码段、堆栈段和数据段，这对于多进程来说十分“奢侈”，**系统开销比较大**，而线程不一样，线程拥有独立的堆栈空间，但是共享数据段，它们彼此之间使用相同的地址空间，共享大部分数据，比进程更节俭，开销比较小，切换速度也比进程快，效率高，但是正由于进程之间独立的特点，使得进程安全性比较高，也因为进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其他进程产生影响，而线程只是一个进程中的不同执行路径。一个线程死掉就等于整个进程死掉。

第二：体现在通信机制上面，正因为进程之间互不干扰，相互独立，进程的通信机制相对很复杂，譬如管道，信号，消息队列，共享内存，套接字等通信机制，而线程由于共享数据段所以通信机制很方便。。

第三：体现在 CPU 系统上面，线程使得 CPU 系统更加有效，因为操作系统会保证当线程数不大于 CPU 数目时，不同的线程运行于不同的 CPU 上。

第四：体现在程序结构上，举一个简明易懂的例子：当我们使用进程的时候，我们不自主的使用 if else 嵌套来判断 pid，使得程序结构繁琐，但是当我们使用线程的时候，基本上可以用掉它，当然程序内部执行功能单元需要使用的时候还是要使用，所以线程对程序结构的改善有很大帮助。

进程与线程的选择取决于以下几点：

- 1、需要频繁创建销毁的优先使用线程；因为对进程来说创建和销毁一个进程代价是很大的
- 2、线程的切换速度快，所以在需要大量计算，切换频繁时用线程，还有耗时的操作使用线程可提高应用程序的响应
- 3、因为对 CPU 系统的效率使用上线程更占优，所以可能要发展到多机分布的用进程，多核分布用线程；
- 4、并行操作时使用线程，如 C/S 架构的服务器端并发线程响应用户的请求；
- 5、需要更稳定安全时，适合选择进程；需要速度时，选择线程更好。

线程阻塞：

线程在运行的过程中因为某些原因而发生阻塞：该线程放弃 CPU 的使用，暂停运行，只有等到导致阻塞的原因消除之后才回复运行。或者是被其他的线程中断，该线程也会退出阻塞状态，同时抛出 InterruptedException。

举例：

- (1) 线程执行一段同步代码，但是尚且无法获得相关的同步锁，只能进入阻塞状态，等到获取了同步锁，才能回复执行。
- (2) 线程执行某些 I/O 操作，因为等待相关的资源而进入了阻塞状态。比如说监听 system.in，但是尚且没有收到键盘的输入，则进入阻塞状态。

## 95. http/ip tcp/udp

<https://www.cnblogs.com/cxuanBlog/p/12177976.html>

## 96. 网页状态码

常见状态码：

200 – 请求成功

301 – 资源（网页等）被永久转移到其它 URL

404 – 请求的资源（网页等）不存在

500 – 内部服务器错误

HTTP状态码分类		
分类	分类描述	
1**	信息，服务器收到请求，需要请求者继续执行操作	
2**	成功，操作被成功接收并处理	
3**	重定向，需要进一步的操作以完成请求	
4**	客户端错误，请求包含语法错误或无法完成请求	
5**	服务器错误，服务器在处理请求的过程中发生了错误	

400	Bad Request	客户端请求的语法错误，服务器无法理解
401	Unauthorized	请求要求用户的身份认证
402	Payment Required	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置“您所请求的资源无法找到”的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止

500	Internal Server Error	服务器内部错误，无法完成请求
501	Not Implemented	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中
504	Gateway Time-out	充当网关或代理的服务器，未及时从远端服务器获取请求
505	HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理

## 97. tcp 三次握手，tcp 的控制可靠性的策略，重传机制

为什么 A 还需要发送一次确认，进行第三次握手呢？主要是为了防止已失效的请求连接报文段突然又传送到了 B，因而产生错误。

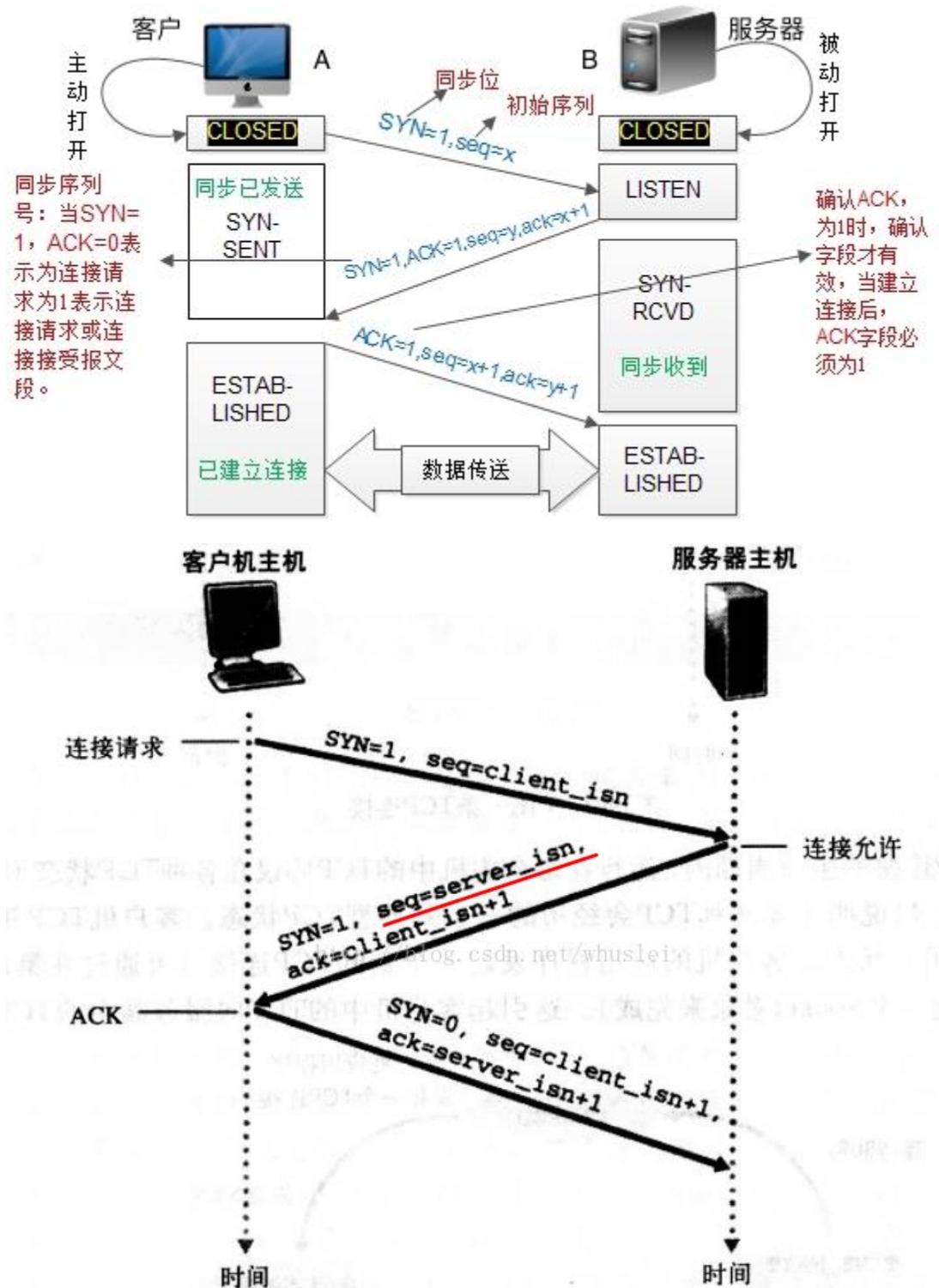


图3-39 TCP三次握手：报文段交换

### (1) 第一次握手

A 的 TCP 客户进程向 B 发出建立连接请求报文段，其中 SYN (同步位) =1，ACK (确认位) =0，seq (序号) =x。

TCP 规定，当报文段的 SYN=1 且 ACK=0 时，表明这是一个请求建立连接的；SYN 报文段（SYN=1 的报文段）不能携带数据，但是要消耗掉一个序号。

在 A 发送完毕之后，A 的 TCP 客户端进程进入 SYN-SENT（同步已发送）状态。

### （2）第二次握手

B 在收到连接请求报文段之后，如果同意建立连接，则向 A 发送确认报文段。其中 SYN=1，ACK=1，ack（确认号）=x+1，同时设置自己的初始序号 seq=y。

TCP 规定，当报文段的 SYN=1 且 ACK=1 时，表明这是一个同一建立连接响应报文段；这个报文段也不能携带数据，同样需要消耗掉一个序号。

在 B 发送完毕之后，B 的 TCP 服务端进程进入 SYN-RCVD（同步收到）状态。

### （3）第三次握手

A 在收到 B 的确认报文段之后，还需要向 B 给出确认报文段。其中 ACK=1，seq=x+1，ack=y+1。

TCP 规定，这个 ACK 报文段可以携带数据；如果不携带数据则不消耗序号，即 A 下一个数据报文段的序号仍然是 seq=x+1。

在 A 发送完毕之后，A 的 TCP 客户端进程进入 ESTABLISHED（已建立连接）状态；B 在接收到 A 的确认报文段之后，B 的服务端进程也进入 ESTABLISHED（已建立连接）状态。

以上就是所说的三次握手过程。

tcp 可靠传输策略：校验和，确认应答和序列号，超时重传，连接管理，流量控制，拥塞控制，滑动窗口。

### （1）校验和

TCP 与 UDP 区别、TCP 可靠性、http 会话保持功能如何实现、get 和 post 区别、post 主体的格式、http 协议的 header

## 98. 网页连接不上什么原因

- （1）网络断开了
- （2）后端页面无法加载
- （3）网页被劫持了，浏览器被篡改
- （4）DNS 无法解析网址
- （5）服务器负载过大
- （6）供应商网络出口出现问题

## 99. 死锁概念，必要条件，避免方式。

死锁：是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去

进程 A 占有对象 1 的锁，进程 B 占有对象 2 的锁，进程 A 需要对象 2 的锁才能继续执行，所以进程 A 会等待进程 B 释放对象 2 的锁，而进程 B 需要对象 1 的锁才能继续执行，同样会等待进程 A 释放对象 1 的锁，由于这两个进程都不释放已占有的锁，所以导致他们进入无限等待中

## 100. 产生死锁原因：

- （1）因为系统资源不足。
- （2）进程运行推进顺序不合适。
- （3）资源分配不当等。

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就

很低，否则

就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。

### 101. 产生死锁的条件：

- (1) 互斥条件：一个资源每次只能被一个进程使用。
- (2) 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。
- (3) 不剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。
- (4) 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

### 102. 死锁的解除与预防：

理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和

解除死锁。所以，在系统设计、进程调度等方面注意如何不让这四个必要条件成立，如何确

定资源的合理分配算法，避免进程永久占据系统资源。此外，也要防止进程在处于等待状态

的情况下占用资源。因此，对资源的分配要给予合理的规划。

解决死锁的方法：

(1) 避免多次锁定。尽量避免同一个线程对多个 Lock 进行锁定。例如上面的死锁程序，主线程要对 A、B 两个对象的 Lock 进行锁定，副线程也要对 A、B 两个对象的 Lock 进行锁定，这就埋下了导致死锁的隐患。

(2) 具有相同的加锁顺序。如果多个线程需要对多个 Lock 进行锁定，则应该保证它们以相同的顺序请求加锁。比如上面的死锁程序，主线程先对 A 对象的 Lock 加锁，再对 B 对象的 Lock 加锁；而副线程则先对 B 对象的 Lock 加锁，再对 A 对象的 Lock 加锁。这种加锁顺序很容易形成嵌套锁定，进而导致死锁。如果让主线程、副线程按照相同的顺序加锁，就可以避免这个问题。

(3) 使用定时锁。程序在调用 acquire() 方法加锁时可指定 timeout 参数，该参数指定超过 timeout 秒后会自动释放对 Lock 的锁定，这样就可以解开死锁了。

(4) 死锁检测。死锁检测是一种依靠算法机制来实现的死锁预防机制，它主要是针对那些不可能实现按序加锁，也不能使用定时锁的场景的。

### 103. Tcp/udp tcp 怎样保证可靠传输，超时重传机制，tcp 报头内容，校验和的作用

26. <https://segmentfault.com/a/1190000022012971>

(1) http 的全称：超文本传输协议 (HyperText Transfer Protocol)

(2) **http 请求结构：**

(3) https 是对 http 的哪一部分进行加密：通信协议

(4) http 自己手动加密和 https 使用 ssl 加密有什么区别；

(5) http 请求发送到返回响应的全过程；

### 104. Cookie 和 session 介绍及其区别

## 105. POST 是否可以把请求参数放在 URL 里？

可以，什么时候呢？

发起 post 请求的时候

## 106. 进程通信方式

### 107. 互斥和同步（操作系统）

进程（线程）之间的两种关系：互斥和同步

所谓互斥，是指散布在不同进程之间的若干程序片断，当某个进程运行其中一个程序片段时，其它进程就不能运行它们之中的任一程序片段，只能等到该进程运行完这个程序片段后才可以运行。

所谓同步，是指散布在不同进程之间的若干程序片断，它们的运行必须严格按照规定的 某种先后次序来运行，这种先后次序依赖于要完成的特定的任务。

直接制约（同步）：由于进程间相互结合而引起

间接制约（互斥）：由于进程间共享资源而引起

(1) 若干同学去图书馆借书；互斥。

(2) 流水线生产的各道工序；同步。

(3) 两队举行篮球比赛；互斥。

(4) 商品生产和社会消费。同步。

## Linux

### 108. 写一段代码，ping 一个 ip 地址，并返回成功、失败的信息。

### 109. 说 5 个以上 Linux 命令。

ls, cd, tailf, grep, cat, scp, netstat, mv, tourch, mkdir, find, vim

### 110. linux 查找当前目录下所有后缀为 .py 文件

find . / -name "\*.py"

### 111. 查看一个端口号的状态

netstat -tnalp | grep 10005

### 112. 知道的 linux 常用命令：查看指定端口进程

(1) 查看程序对应的进程号： ps -ef | grep 进程名字

(2) 查看进程号所占用的端口号： netstat -nltp | grep 进程号

### 113. 说一下 ping？

ping 命令是用来测试与目标主机的连通性。

我们经常会说“ping 一下某机器，看是不是开着”、不能打开网页时会说“你先 ping 网关地址 192.168.1.1 试试”。

ping 命令用于：确定网络和各外部主机的状态；跟踪和隔离硬件和软件问题；测试、评估和管理网络。

### 114. ping 命令格式：

ping [参数] [主机名或 IP 地址]

Ping 后面接域名可以吗？域名前面加 http/https 可以吗？

1.命令格式：

ping [参数] [主机名或IP地址]

2.命令功能：

ping命令用于：确定网络和各外部主机的状态；跟踪和隔离硬件和软件问题；测试、评估和管理网络。如果主机正在运行并连在网上，它就对回送信号进行响应。每个回送信号请求包含一个网际协议（IP）和 ICMP 头，后面紧跟一个 tim 结构，以及来填写这个信息包的足够的字节。缺省情况是连续发送回送信号请求直到接收到中断信号（Ctrl-C）。

ping 命令每秒发送一个数据报并且为每个接收到的响应打印一行输出。ping 命令计算信号往返时间和（信息）包丢失情况的统计信息，并且在完成之后显示一个简要总结。ping 命令在程序超时或当接收到 SIGINT 信号时结束。Host 参数或者是一个有效的主机名或者是因特网地址。

（1） ping 网关

ping 192.168.120.205

（2） ping 指定次数

ping -c 10 192.168.120.205

（3） 通过域名 ping 公网站点

ping -c 5 [www.58.com](http://www.58.com)

不可以。

ping 是 ICMP 协议的，HTTP 是 TCP 协议的，要 ping HTTP 的话需要 tcp ping。ping 是 ICMP 协议，发的是 ICMP 包，跟 HTTP 协议层无关，是直接发到 ip 上的，也就是说域名也可以。http://xx.com 是 http 协议定义的 url，用的是 http 协议，发送跟接收都是 http 包。

115. 自我介绍+项目

116. 介绍下做的项目，产品的业务

117. 大概讲一下你做的项目的架构，你负责的模块简单介绍下

118. 你提了这么多 bug，大概说一下怎么发现的，bug 主要来源是什么；说一个自己印象深刻的 bug，原因分析，开发怎么修改的，怎么回归的（**重点是定位 bug 和回归 bug 的过程**）

（1）业务逻辑：不符合业务逻辑

（2）前端显示：前端页面的展示有问题

（3）后端接口：后端接口测试不通过。比如在一些参数限制和并发上

（4）前后端交互：前后端交互上有问题，比如传参错误

印象深刻的 bug：

（1）之前的 bug 记不太清了，说一个最近发现的吧。一个接口，最后一级路径是由不同条件决定的。自己测试的时候发现开发只对正常路径做了判断，没有对异常路径做判断，导致异常路径也能成功请求。看了一眼代码，发现确实缺少判断。

开发加上了异常路径的判断=》自己进行了对应场景的回归。

（2）增加日志

（3）自己找调用接口，找对应端的开发

历史遗留 bug：

119. 你做过最复杂那个模块是啥

120. 工作以来写了多少用例

没有具体统计过，除了特别简单的小迭代，都会写用例

121. 工作中遇到过最棘手的问题是什么（印象最深的问题是什么），如何解决的？

平常怎么学习这些专业知识的

- (1) 平时遇到问题，会 google 搜索一下
- (2) 系统学习专业知识，会去知乎看一些学习指导，然后去对应专业网站上学习
- (3) b 站上学习资源很多，有时候也会去 b 站学习

122. 你对测试的定位是啥，你觉得测试起什么作用

- (1) 首先，最基本的，发现问题，推动解决问题，保证项目质量，是一个质量把控的角色
- (2) 另外，在测试过程或者工作流程中，去发现一些痛点问题，去解决这些问题。比如，去制定一些质量相关的规范，去规范一些测试流程，去做一些自动化工具或平台来提高测试效率。
- (3) 作为一个 模块的负责人，要对这个模块的整体有把控。怎样去保证这个模块的质量，怎样去建设质量保障体系，比如在流程上，用例上，测试上以及和开发的对接上。怎样提升你的测试效率，质量的标准和限制上，要把整个框架放在脑子里。

123. 假如让你来保证整个 APP 的质量，如何实施，首要保证什么功能，如何保证？

- (1) 要对整个 app 核心功能整体有一定的了解
- (2) 首先要保证核心功能的稳定性

124. 说说接口测试的流程，介绍一下 request 有哪些内容。

分为功能和性能两部分：

- (1) 功能测试
  - 1) 业务逻辑功能：正常场景，异常场景
  - 2) 输入输出参数：参数边界、参数类型、是否必填、特殊字符等等
- request 请求包括：大致域名、接口、参数、方法、以及返回
- (2) 性能测试
  - 1) 响应时间、吞吐量、并发数、服务器资源使用率（CPU，内存，网络）

request 包含内容：

HTTP 请求报文由请求行（request line）、请求头部（request header）、请求数据和空行 4 个部分组成，最少包含三个部分，也就是说第四部分可以为空

125. 性能测试用什么工具？

Postman、 jmeter

## 126. 响应时间怎么判断是正常的？

### 127. QPS 是什么？有规定值是多少吗？

([https://blog.csdn.net/lv\\_shi\\_jun/article/details/97915469](https://blog.csdn.net/lv_shi_jun/article/details/97915469))

- (1) 并发量：同时访问同一服务的连接数
- (2) QPS：查询每秒，即每秒查询率，可以理解为每秒处理的 request 的数量
- (3) 并发量和 QPS 关系：

OPS=并发量/平均响应时间

- (4) TPS：事务每秒，即每秒产生的事务数

(5) 吞吐量：针对一个系统而言，表示系统承受压力的能力。与 request 对 cpu 的消耗、外部接口、I/O 等密切关联。单个 request 对 CPU 消耗越高，外部系统接口、I/O 影响速度越慢，系统的吞吐能力越低，反之越高。

## 128. 自动化测试发现了哪些问题？

自动化是一种手段，可以从一定程度上提高测试效率，但并不能替代人工，真正要要先产品的缺陷，手工测试是必不可少的。

### 129. 怎么保证测试的一个覆盖度？

Cr=>自动化 case 前置=>测试用例，

### 130. 自动化脚本用了哪些库？封装的类？哪些第三方库？Xpath

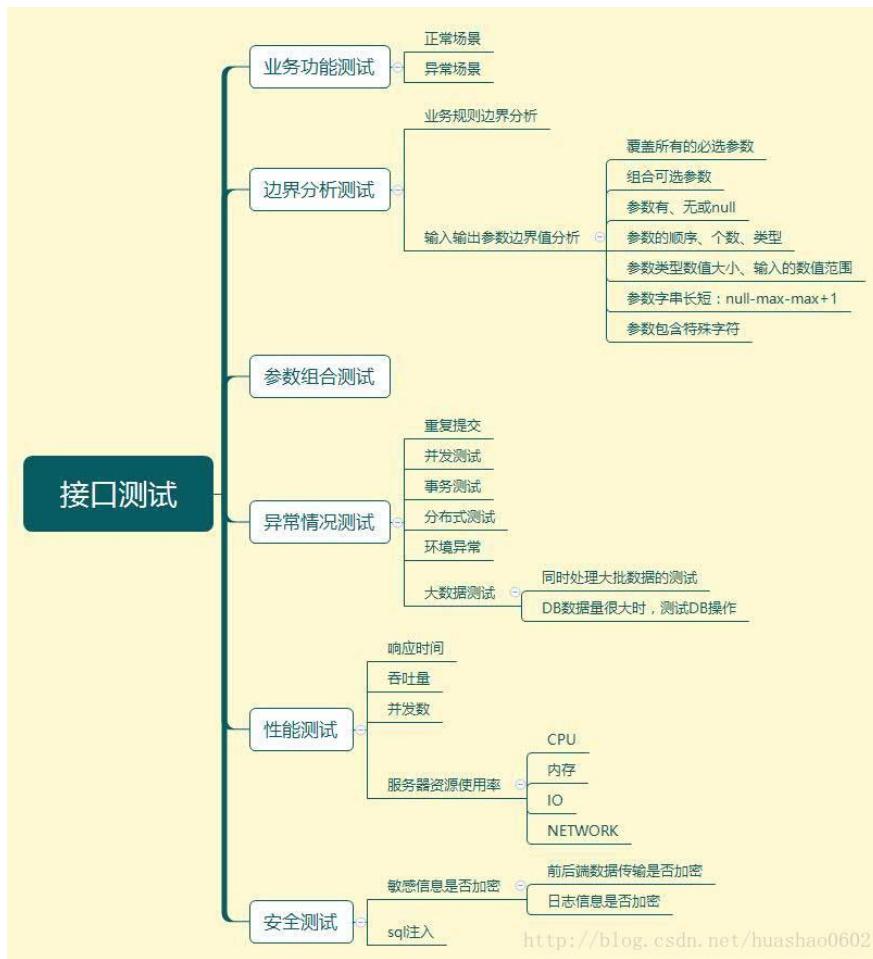
Request, json, random, threading, selenium, time, Elasticsearch 等(pymysql, Blueprint, logging, wraps)

### 131. 自动化测试中遇到过哪些问题？怎么解决的？

- (1) case 不稳定=> 场景 case，做数据前置

- (2)

### 132. 接口要测试哪些范围？



### 133. 怎么做需求分析？

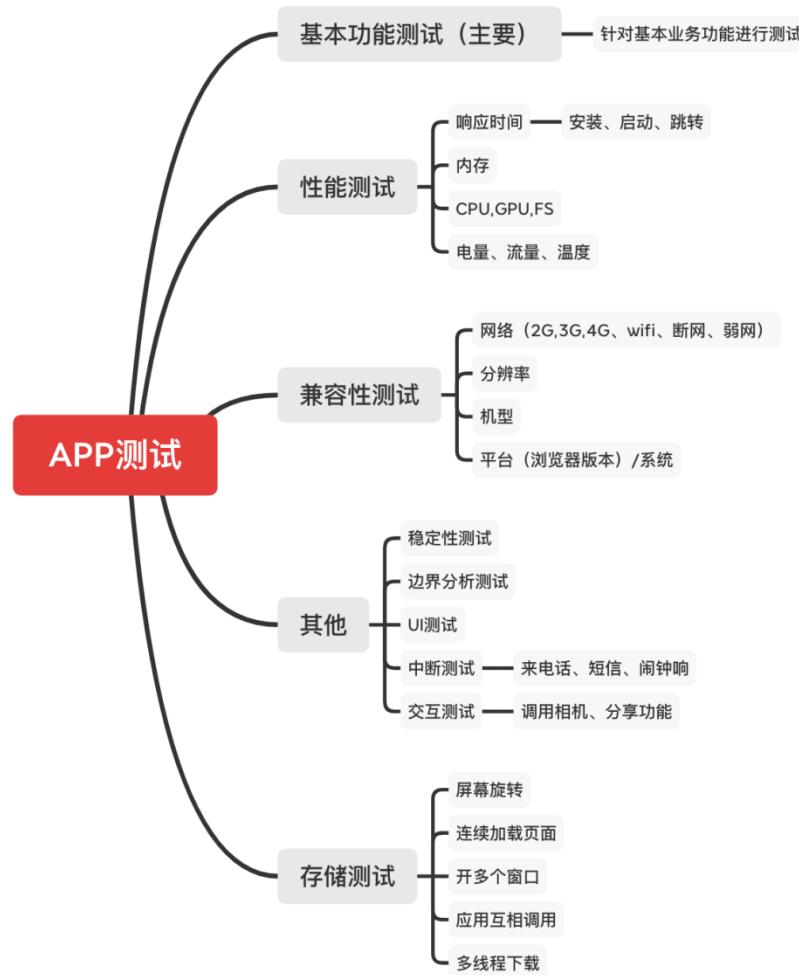
“需求分析”的过程实际上就是经历“1 用户需求 -> 2 产品需求->3 产品功能”。 1 ->2, 通过问“What”，逐步归纳；2 ->3，通过问“How”，逐步演绎。

### 134. 需求分析的时候有给过哪些建议？

### 135. 自己有排查过哪些问题？

(1) 接口报错，自己查日志，翻代码找调用的其他端的接口，找对应负责人排查原因

1. app 测试有哪些不同？



## 136. 对性能测试了解多少?

### 吞吐量

如果是从统计上讲就是在一次性能测试过程中，网络上传输的数据总量。举个例子就像手机一直用流量上网，那本月流量统计就是本月的吞吐量。

一次测试过程中传输了多少数据对于一般的性能测试来说，往往意义不太大。除非有流量限制的需求，对过程的流量总数限定在一定的范围内时候，会关注这个指标。

人们口头上甚至有很多软件实现中所说的吞吐量其实是吞吐率。通俗的讲就是网络传输速率。和利用下载工具下载东西的过程中看到那些个 **1.2MB/s** 是一样的概念。

在下载过程中看到的下载速率一般只是在想我还有多久才能把这个东西下载完成，而在性能测试过程中，这个指标则显示了在那个时点服务器对发起请求的处理能力。如果随着虚拟用户数的逐渐上升，吞吐率也在随之上升，说明服务器的处理能力还没达到饱和，还可以承载更多的用户。如果随着虚拟用户数上升，吞吐率并没有明显变化，甚至出现下降的趋势。则说明服务器的性能已经达到瓶颈，已经不能应对当前用户的请求量了。一般遇到这种情况，排查手段就是从网络带宽，数据库，应用服务器，io 磁盘，代码本身 5 个环节逐一排查系统的瓶颈所在。

## 点击率

点击率的含义不像字面上反映的是每秒有多少的点击量，而是每秒向服务器发送的请求数。有什么不一样呢？现在在页面往往包含一些图片，flash 广告，javascript 脚本，css 脚本等一堆资源，没有缓存的话，每一个都会对服务发送一次请求。所以当鼠标点击一个超链接到新的页面时，点击率统计的到的数字并不是 1，而是网页上所有对服务器资源请求的总和。需要强调一点的是点击率只是客户端发送给服务器的请求总和，并不总等于服务器处理的请求数，这个有另一个指标 TPS(Transaction Per Second, 每秒事务数)进行统计。虽然两者相似，却是表达了不同的信息，前者是客户端对服务器的压力大小，后者则是服务器处理能力的实际体现。使用过程中需要视测试场景进行选取。

## 事务通过率

性能测试过程中，所谓事务的通过率就是得到了服务器正确响应事务数在相应的总事务数中的占比，是判断一次性能测试执行过程是否有效的一个重要指标。

若事务通过率低于一个限定值，则说明服务器并不能有效处理测试过程中的全部请求。对于不同的系统，该限定值也是可以不同的。若是需要很高稳定性，限定值就会 99.9%（即一千笔业务只允许一笔业务失败）甚至更高。而如果是一般网页访问一类，则可以适当放宽到略低的水平，但一般不应低于 95%，否则就认为系统的事务处理性能是不可接受的。

**137. 你是怎么样去做项目沟通和项目管理的？**

**138. 详细介绍自己参与度最高的项目，扮演的角色以及项目的具体内容**

**139. 性能测试如何做的？性能测试关注的内容？**

**140. 介绍项目中某个功能，并且介绍如何测试的**

2. 工作经历这块因为我在简历上写的比较务实，所以回答起来没有什么问题。待改进：对于测试来说，缺少了对整个系统测试过程全局的了解和把控，后续需要着重学习准备一下

**141. 自动化测试在什么场景下使用；**

产品需求变更少，项目开发周期长，测试用例执行频繁、手工测试无法胜任（繁杂），这些条件下都可以考虑自动化测试。

适合自动化测试的类型：单元测试、回归测试、性能测试、接口测试

3. [https://blog.csdn.net/qq\\_42434318/article/details/114263419](https://blog.csdn.net/qq_42434318/article/details/114263419)

## 一、公司使用工具

**142. 介绍一下你在这个项目中是如何使用 Jenkins 的。**

**143. 说说你对敏捷模式的认识。**

是什么？

我个人的理解，就是把一个大目标细分为短期内可以完成的可运行可交付的小目标；  
好处？

1. 由于拆分成小目标，这样在短时间内就可以看到成果！在看到成果的同时也可以及时修改，因为好多产品经理也不能明确自己要做什么；这样可以弥补经过长期做出来的产品还不是想要的产品的尴尬！及早发现，及早治疗~

2. 开发人员有新鲜感！工作效率高！为什么这么说呢？因为短时间就可以看到自己的劳动成果，使人有满足感，避免在长时间的工作中消耗自己的激情！还有一点就是人性，一个人长

时间做一个事情看不到成果难免会懈怠！如果一个功能开发一个月或者更久，久而久之，自己的开发效率啊！主动性啊都没有刚开始的时候那么积极！

还有一点就是做个原型上线，看看市场用户的反馈，如果用户对你做的这个感兴趣，你可以继续丰富功能，继续挖掘；如果不 case，那就果断放弃，继续下一个 idea，也没多大的损失!!! 因为好多的产品都是试出来的，没有哪几个产品经理敢说我设计的产品一定是受用户喜爱的。其实我认为最最最重要的就是团队之间的配合度！！！默契度！！！有效沟通！！！团队之间要真正的从心里互相尊重，理解，与欣赏！！！做到一个团结，紧张，活泼的团队！！！

## 144. 了解过 Docker 不？

docker 是什么

为什么会出现 docker，常使用项目部署开发的人自然能理解，因为 docker 让服务部署更快更高效。

下面说一下 docker 是什么，有什么用，等同于说明为什么会出现 docker 这种技术。

docker 主要是让服务部署更高效。官网称之为容器 container，在项目开发中开发环境/测试环境/生产环境通过 docker 容器技术统一，减少环境误差导致产生的错误，提高开发部署速度，减少部署维护成本。

镜像技术，隔离环境

我认为 docker 是一种镜像服务技术，他将原来 linux 上运行的服务全部放在 docker 内运行，linux 和 docker 之间有一定的物理隔离和软件版本隔离效果，并且可以复制多个程序而避免重新安装，很好的提高服务器项目部署相关的工作效率，且有增加服务器安全系数。

（甚至介绍通过 docker，可以让 linux 上的程序也能在 windows 照常运行，目前还没试过）

高效部署服务

当你有一个服务开发部署好了，服务器是 linux，里面安装了 mysql，nginx，java-application，redis，jdk8；现在服务器要扩展，同样的服务部署到多台服务器，要怎么做才高效？重新安装 mysql，nginx，jdk8，然后各种配置端口密码各种参数，再重新启动每个程序？当你作为一个部署负责人的时候会觉得重复繁琐，并且配置细节很多，不小心会出差错，这时候镜像技术 docker 就能使服务部署快速，并避免少犯错，这时候 docker 可真是个好东西。

将原来的运行服务放在 docker 内运行，需要部署到其他服务器时，打包整个 docker 生成 docker 镜像，然后复制 docker 到新的 linux 服务器启动就好了，mysql，nginx，redis，等等很多东西不需要重新配置部署，只需要启动镜像 docker 和一些个性化的脚本

## 145. 智力题+开放题

1. 抖音的上传视频功能，测试用例设计（中间说兼容性测试的时候，ms 官补充提问：设备种类、版本这么多，覆盖策略是什么？）

<https://my.oschina.net/u/4575195/blog/4442734>

## 146. 问题排查：现在有用户说发布不了视频，你怎么排查

**个人原因：**

- (1) 排查是不是网络问题导致，切换网络重试
- (2) 排查是不是抖音卡住了，这时候可以先将视频保存到本地，退出重新打开尝试
- (3) 抖音版本是否需要更新。低版本的抖音可能因为抖音服务器的升级无法连接到服务器，就不能上传视频，此时可以更新版本试试
- (4) 视频内容是不是不合规范
- (5) 视频格式或者大小是否符合要求

**官方原因：**

- (6) 如果是批量用户出现这种问题，而网络情况良好，极有可能是抖音官方服

## 务出现问题

147. 自己最有成就感的事情。

148. 假如有一天，报告某地区的用户都打不开 app 中的一篇资讯（一个视频），可能的原因有哪些？

149. 针对微信朋友圈发送图片功能，设计用例，不考虑视频

150. APP 和竞品 APP 之间的关系，APP 和竞品 APP 产品设计上有什么不同点，为什么要这么设计？

151. 现在有两个杯子，一个能装水 6L，一个能装水 5L，不用其他杯子的情况下，怎么倒出 3L 的水？（WTF？）

首先把 5L 的杯子倒满再倒入 6L 的杯子中，6L 的杯子里还差 1L 水

再把 5L 的杯子装满再倒入 6L 的杯子里，此时 6L 的杯子里 6L 水，5L 杯子里 4L 水

把 6L 杯子水都倒掉，把 5L 杯子剩的 4L 水导入 6L 杯

5L 杯子装满再导入 6L 杯后还需 2L 才满，此时 5L 杯子就剩下 3L 水了。

152. 微信中发送图片给他人，对这个功能设计用例

### 功能测试

1、单个图片，不以原图发出，图片是否清楚

2、单个图片，以原图发出，接受图片是否原图

3、一次最多可以发多少张图片

4、一次发出图片超出是否限制，是否发出正常提示

5、发送视频后，接收视频是否影响画面是否清楚声音是否正常

6、图片和视频一起是否可以发送

7、未勾选图片或者视频是否会发出

8、图片尺寸超大是否可以正常发送

9、图片尺寸超级小是否可以正常发送

10、图片大小很大超出1G,是否可以发出

11、不同格式的图片如：jpg、png、jpeg等格式是否可以发出

13、视频大小超大是否可以正常发送

14、不同格式的格式视频如：MP4、WMV、AVI、MOV、3GP等格式是否可以发出

## 界面测试

1、发送图片以及选择图片发送是否符合设计文档和需求文档

2、是否存在错误别字

## 兼容测试

1、不同操作平台是否可以接收图片或者视频

2、不同操作系统是否可以接收图片或者视频

3、不同微信版本是否可以接收图片或者视频

## 性能测试

1、多人同时发送图片或者视频，是否正常

2、最大的图片或者视频，发送出去是否有影响

3、发送出去之后，流量是否正常

4、发送出去耗时多久

1、弱网下，发送图片是否正常

2、在发送图片切换移动网络，WiFi是否失败

## 安全测试

1、图片设计少儿不宜的图片或者视频是否提示

2、发送图片是否存在丢失资源

## 153. 抖音评论设计测试用例。

## 154. 微信朋友圈点赞功能测试用例



## 155. APP 测试与 web 端测试的区别

## **首先从系统架构来看的话：**

web 项目，一般都是 b/s 架构，基于浏览器的，而 app 则是 c/s 的，必须要有客户端。那么在系统测试测试的时候就会产生区别了。

web 测试只要更新了服务器端，客户端就会同步会更新。而且客户端是可以保证每一个用户的客户端完全一致的。但是 app 端是不能够保证完全一致的，除非用户更新客户端。如果是 app 下修改了服务端，意味着客户端用户所使用的核心版本都需要进行回归测试一遍。

## **接着是性能方面**

web 页面可能只会关注响应时间，而 app 则还需要关心流量、电量、CPU、GPU、Memory 这些了。至于服务端的性能是没区别，这里就不谈。

## **然后是兼容方面**

web 是基于浏览器的，所以更倾向于浏览器和电脑硬件，电脑系统的方向的兼容，不过一般还是以浏览器的为主。而浏览器的兼容则是一般是选择不同的浏览器内核进行测试（IE、chrome、Firefox）。app 的测试则必须依赖 phone 或者是 pad，不仅要看分辨率，屏幕尺寸，还要看设备系统。系统总的来说也就分为 Android 和 iOS，不过国内的 Android 的定制系统太多，也是比较容易出现问题的。一般 app 的兼容测试三种方法，云测试，请团队测试，真机测试。这里说说真机的选择，首先要选择主流的机型，其次要选择不同的分辨率，尺寸，然后就是不同的操作系统。

相比较 web 测试，app 更是多了一些专项测试。

## **安装、卸载、更新：**

web 测试是基于浏览器的所以不必考虑这些。而 app 是客户端的，则必须测试安装、更新、卸载。除了常规的安装、更新、卸载还要考虑到异常场景。包括安装时的中断、弱网、安装后删除安装文件，更新的强制更新与非强制更新、增量包更新、断点续传、弱网，卸载后删除 app 相关的文件等等。

## **界面操作：**

web 测试一般都是在 pc 端的浏览器操作，主要侧重页面的加载，展示，数据的准确，页面的跳转，异常提示，鼠标的操作。

现在 app 产品的用户都是使用的触摸屏手机，所以测试的时候除了注重加载，展示，数据的准确，页面的跳转，异常提示，还要注意手势，横竖屏切换，多点触控，事件触发区域等测试。

## **出现的问题：**

web 端容易出现的 bug：页面显示慢，查询数据加载慢，跳转空白页，报错内容为代码，图片文字排版错误，提示弹窗问题，不同浏览器显示问题，cookie/session 过期.....

app 容易出现的 bug：闪退，卡死/崩溃/crash，白屏，页面卡顿，兼容性问题，网络流量问题，耗电量大，内存不足，app 使用权限问题.....

## **测试工具的不同**

就自动化来讲，web 大多用的 selenium、webdriver，而 app 则是 appium。

性能使用的工具 web 则是 LR，app 使用 Jmeter 要多一点。

接口测试的话，因为测试的对象都是后台服务的接口，所以 app 和 web 基本上一样，都可以使用 jmeter，postman，fiddle。

10. 浏览器兼容性测试的不同是什么？要测试哪些浏览器？

一般选：IE，Firefox，Google Chrome，Safari

为啥这么选：以浏览器内核分类进行测试

常见浏览器及四大内核：

(1) IE、360（兼容模式）、搜狗（兼容模式）（Trident 内核）

(2) Firefox（Gecko 内核）

(3) Chrome、360（极速模式）、搜狗（极速模式）（Blink 内核）

(4) Apple Safari（WebKit 内核）

<https://www.cnblogs.com/kuaileya/p/12021187.html>

156. 浏览器兼容性有哪些问题？

[https://blog.csdn.net/qq\\_39894133/article/details/79178328](https://blog.csdn.net/qq_39894133/article/details/79178328)

157. h5 页面要测试哪些东西？h5 页面的性能要关注哪些？

<https://cloud.tencent.com/developer/article/1527746>

158. 一张 A4 纸怎么去测试

[https://blog.csdn.net/weixin\\_44318102/article/details/105887164](https://blog.csdn.net/weixin_44318102/article/details/105887164)

功能、性能、安全性、兼容性、ui、易用性

159. 抗压能力强，平时加班多吗？

160. 平时学习些什么东西？会总结些一些东西吗？

161. 你觉得自己的优势在哪？

162. 抖音海外版 tik tok 有哪些安全隐患？

163. 20.1) 抖音搜索功能测试用例 2) 搜索方面有哪些安全性问题 3) 搜索方面如何提高

164. 设计百度首页测试用例

165. 编写一个登录界面的测试用例

166. 登录的按钮不能点击如何排查问题

167. 智力题：10 堆苹果，每堆 10 个，9 堆每个 50g，1 堆每个 40g，有一个称，求只称一次，找出这个轻的一堆

先把每个箱子编号，1~8，然后每一个编号的箱子拿出编号个数的苹果，(如 3 号箱子拿出 3 个)共 36 个，称下！少几克，那几号箱子就是轻的那箱。

168. 编写测试用例：一个接口报文

169. 微信群 3-5 人发消息测试用例

170. 个人规划、如何理解测试开发岗位

171. 与开发人员冲突的处理方法及遇到问题的解决方法

172. 接下来问了对于测试的工作和发展方向的理解及个人的缺点

173. A 给 B 发送消息后，A 看见消息发送出去了，但是 B 没有收到，怎么排查问题？

- (1) 消息状态显示错误，实际消息没发出去，
- (2) 发消息接口传的对象 id 有误，实际发给 C 了，
- (3) 消息发送到服务端了，服务端没转发给 B，
- (4) 服务端发给 B 了，B 没显示或 B 无法解析，
- (5) B 端和服务器没通信(没网)

#### 174. 电梯的测试用例

<https://blog.csdn.net/slforeverlove/article/details/47080279>

#### 175. 网页提交慢的原因

- (1) 网络原因
- (2) 前端涉及不合理，比如 js 逻辑复杂
- (3) 接口获取数据缓慢
- (4) 其他

#### 176. 飞猪购票测试用例

个人思路：购票分为购买火车票、机票、汽车票、门票

同时对于机票又有单程、往返和跨境多程的选择，有目的地的选择，有日期的选择。测试的话需要测试是否可以注册和登录系统，同时对于登录之后选择的时间是否有票、没票是否无法下单，有票的话是否可以正常显示票数及是否可以正常下单。

还需要测试就剩一整票时多人购买是否有冲突；

购票日期是否可以选择过去的时间、测试最晚购票日期功能是否正常

目的地的输入是否正常，是否有遗漏的城市，所给城市是否都可以选择

如果有疫情限制地区是否真的无法购买车票等情景

需要结合具体的接口设计进行案例编写，目前我想起来的就这些方面，如果有开发好的接口和软件在手，测试起来写的更有逻辑和针对性。

测试就是保证软件可以实现正常的功能，对于可能出现 bug 的点提前预设，保证程序无重大功能缺陷，小 bug 会在迭代的过程中不断的优化，同时上线一些客户需要的功能。

#### 177. 微信发图片的测试用例

#### 178. 项目组人员构成

#### 179. 有一个智能语音识别系统实现语音转文字，你会如何测试

从系统的易用性、稳定性、可靠性和兼容性方面进行测试

如系统使用是否方便

系统对于语音转换的成功率的测试

系统对于多语言及多方言的支持度测试

系统对语音的音量的识别度

系统对于非人的语音识别，如狗叫等的识别情况

系统在各个平台上的兼容性

（个人瞎写的）

#### 180. 测试微信的视频通话功能

1. 界面展示是否正常
2. 是否可以正常点击进入视频通话
3. 进入视频界面是否可以看到视频的对方
4. 最多可以有多少人加入视频会话

5. 视频是否可以切换显示窗的大小
6. 正在视频中是否可以接听其他人发来的视频邀请
7. 视频中是否影响手机的接听电话功能
8. 没有网络或者网络差的情况下是否可以进行视频通话
9. 视频通话的画面情绪度和声音检测
10. 点击视频通话界面是否可以显示和隐藏功能键
11. 正在视频中是否可以微信聊天
12. 正在视频中是否可以发送图片、视频
13. 视频通话的挂断功能是否正常
14. 不同网络条件下的测试
15. 不同手机类型下的测试
16. 不同的系统版本进行的测试

(以上为个人随想，缺乏思路性，容易漏掉测试点)

(以下是搜索的答案，测试的方面清晰，相对较为全面)

功能测试：

1. 视频能否连接成功
2. 声音和画面是否正常，能否同步
3. 挂断功能是否正常
4. 单人视频和多人视频是否正常；

性能测试：

5. 压力测试——长时间视频（如 12 小时）是否能保持正常，cpu,内存消耗等；
6. 稳定性测试——频繁进行视频；
7. 前后台切换——与其他应用切换，视频过程中来电话、短信等；
8. 不同网络测试：wifi 和流量

兼容性测试：

9. 在安卓和 IOS 手机上分别测试
10. 选择不同机型
11. 不同系统版本测试

界面测试：

12. 软件界面文字，图片和 Logo 显示正常
13. 操作过程中出现的各种提示显示正常
- 14.

### 181. 微信的小程序打不开的原因可能是什么

- 缓存太多微信的长时间使用，会在微信内部存在很多缓存，如果长时间不清理，就会造成微信打开缓慢以及小程序打不开的状况；
- 小程序正在维修或已经停运；
- 手机运行问题，可能由于手机的问题；
- 网络问题，小程序不能打开，建议先观察一下你的手机是否连接了网络，如果未连接网络，也就不能打开了。
- 微信没有更新到最新的版本，微信的版本有时也会影响小程序道的打开，

### 182. 测试微信朋友圈点赞、评论

#### 一、点赞功能

- 1、网速对点赞的影响
- 2、点赞的人个数显示是否正确

- 3、共同好友能否看到点赞状态
- 4、能否显示点赞得人的头像和昵称，若能显示是否正确
- 5、一行显示几个点赞的头像
- 6、能否正常的点赞和取消点赞
- 7、点赞显示能否按照时间的先后
- 8、消息列表中是否显示点赞人的昵称，头像和点赞时间。

9、不同手机操作系统显示界面

10、能否及时刷新

11、点赞后还能不能评论

12、点赞是否有上限

## 二、评论功能

1、网速对评论的影响

2、共同好友能否看得到评论，非共同好友能否看到评论状态

3、评论能否按时间先后顺序显示

4、评论能否显示评论人的昵称，若能显示是否正确

5、能否回复评论

6、是否可以既评论又点赞

7、评论和点赞后是怎样现实的，分两次显示，还是一次显示

8、评论是否有上限

9、能否及时刷新

10、未登录情况下能否看得到

11、不同手机如何显示

12、是否能将评论全部显示在朋友圈下面

13、好友能否看到发圈人的评论及回复

## 三、界面显示

1、是否是显示发朋友圈的人的昵称、头像、以及具体内容

2、是否按照发朋友圈的时间距离现在远近来排序

3、图片显示是否正确

4、是否显示自己的个人朋友圈背景

5、下拉是否有更新

## 183. 有没有使用过字节的产品，说一下优缺点

抖音。

优点：推荐一些热门的视频，视频种类多，主题比较丰富。现在抖音水印没有了。

缺点：看过以下热门的视频后之后会不断推送相关的视频，感觉被绑架了。有些时候感觉到没怎么有新意的视频还上热搜了，很不解上热搜的点在哪里。还有就是会有闪退。

## 184. 场景 12 点用户登录后无操作，12.05 才有数据访问。若连接能保持半小时，什么时候连接会断开？(12: 35)

## 185. 粗细不均匀的绳子，只知道烧完需要一个小时，想得到 1.25 小时至少需要几根绳子？3 根

一根绳子从两头点燃后烧完是 0.5 小时，一半绳子烧完是 0.25 小时。

a/b/c 三根绳子

a 的两端和 b 的一端同时点燃，待 a 烧完点燃 b 的另一端和 c 的两端，就是 1.25 小时。

## 186. 说出一根牙签的用法，至少 10 种

可以当牙签、插水果吃、扎孔、清理缝隙垃圾、作为胶带的隔绝头、取手机卡、做牙签

肉，编小物件，玩游戏，切断胶带，垫桌子腿、扎孔等

### 187. 64 匹马，8 个赛道，最少通过几次比赛能找到最快的 4 匹马(12 次)

首先分成八轮选出前 8 匹跑得最快的马

再跑一次选出第 1 快的马

再跑 1 次选出第 2 快的马

再跑 1 次选出第 3 快的马

再跑 1 次选出第 4 快的马

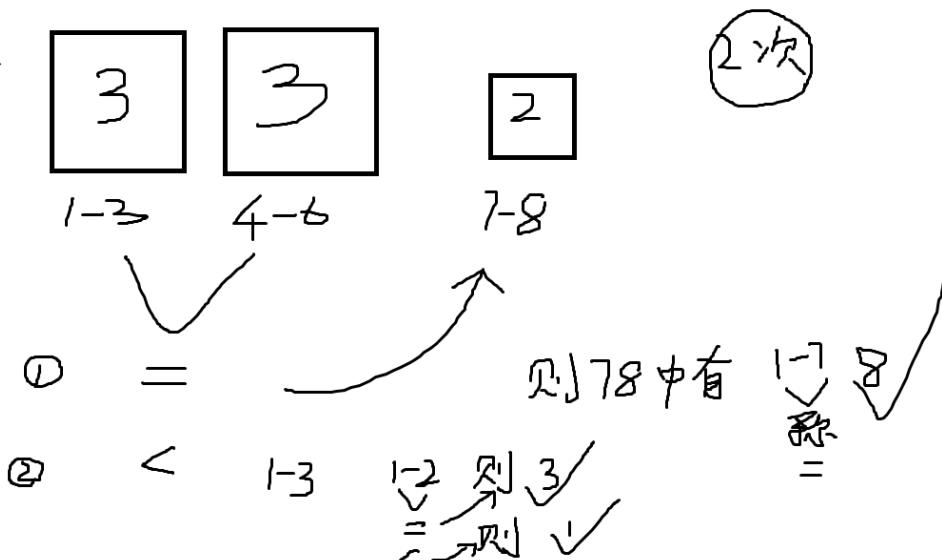
### 188. 微信发红包的测试用例（功能、性能、安全、兼容性）

### 189. 微信发红包和支付宝红包的对比

### 190. 网页缺失部分内容的原因

### 191. 微博在某些地区刷不出来的原因

### 192. 8 个球，找出其中的一个轻球最少需要多少次？(2 次)



### 193. 为什么想做测试开发工作？

(1) 第一方面是兴趣吧。因为自己在平时生活中用一些 app 的时候，也会喜欢留意他们存在的一些问题，有时候也会去客服进行一个反馈，比如刚开始用抖音的时候（举例自己发现过的问题西瓜视频、抖音回退）。

(2) 性格上比较合适。自己是一个比较细心、有耐心的人。自己负责过的项目没有出现过严重 bug。

(3) 另外，自己也比较喜欢挑战

### 194. 测试开发和测试的区别？

测试开发的核心职能依然是测试。只是说测试开发工程师在具备测试经验、熟练使用测试工具并有一定开发能力的前提下，可以自主开发平台，或对现有开源工具进行二次开发，最终目的是用开发的能力去提升测试效率。开发还是测试的一个辅助，并不能占主导。

### 195. 怎么理解测试开发这个岗位在软件开发中的作用？

一个产品的质量对于用户和公司都是至关重要的。而谁去保证产品的质量，就是测试开发人员来完成的。所以我觉得测试开发就是保证产品质量的重要岗位之一。测试开发岗位也需要和产品、研发、业务团队密切协作，提高产品的质量。

### 196. 你的职业规划？（深入业务、提升测试开发能力、带领一个测试团队，把控整个测试模块的综合能力很强的工程师）

培养自己的思维模式，深入理解业务和客户行业知识，持续学习

(1) 巩固自己的测试基础知识，在基本知识扎实的情况下提高理解需求文档的能力。

(2) 学习自动化测试工具，并将它运用到测试中。

(3) 在测试技术达到一定程度后，要学会如何带领一个测试团队。

(4) 最终希望可以成长为可以让产品的研发线更加高效，能成为全面的测试开发工程师。能发现开发人员没有发现的问题，能想到产品人员没有想到场景，成为一个综合能力很强的测试开发工程师。

### 197. 对于这个岗位，你有什么优势？

兴趣；细心、耐心；做事有条理；喜欢挑战；沟通能力

### 198. 你在测试中发现了一个bug，但是开发认为这不是一个bug，你该怎样解决？

[https://blog.csdn.net/Hningning/article/details/103251879?utm\\_term=%E9%9D%A2%E8%AF%95%E4%BD%A0%E6%80%8E%E4%B9%88%E7%90%86%E8%A7%A3%E6%B5%8B%E8%AF%95%E5%BC%80%E5%8F%91&utm\\_medium=distribute.pc\\_aggrpage\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-1-103251879&spm=3001.4430](https://blog.csdn.net/Hningning/article/details/103251879?utm_term=%E9%9D%A2%E8%AF%95%E4%BD%A0%E6%80%8E%E4%B9%88%E7%90%86%E8%A7%A3%E6%B5%8B%E8%AF%95%E5%BC%80%E5%8F%91&utm_medium=distribute.pc_aggrpage_search_result.none-task-blog-2~all~sobaiduweb~default-1-103251879&spm=3001.4430)

●首先，将问题提交到缺陷管理库进行备案。

●然后，要获取判断的依据和标准：

根据需求说明书，产品说明、设计文档等，确认实际结果是否与计划有不一致的地方，提供缺陷是都确认的直接依据；

如果没有文档依据，根据类似软件的一般特性来说明是否存在不一致的地方，来确认是否是缺陷；

根据用户的一般使用习惯，来确认是否是缺陷；

●与设计人员，开发人员和客户代表等相关人员探讨，确认是否是缺陷；

合理论述，客观严谨的向测试经理说明自己的判断理由；

●等待测试经理做出最终决定，如果仍然存在争议，可以通过公司政策所提供的渠道，向上级反应，并由上级做出决定

测试用例设计经典面试题之电梯、杯子、笔、桌子、洗衣机、椅子、ATM 等

### 199. 测试项目：电梯

●需求测试：查看电梯使用说明书、安全说明书等

●界面测试：查看电梯外观

●功能测试：测试电梯能否实现正常的上升和下降功能。电梯的按钮是否都可以用；

●电梯门的打开，关闭是否正常；报警装置是否可用，报警电话是否可用；

●通风状况如何，突然停电时的情况；是否有手机信号；

●比如说上升途中的响应。电梯本来在 1 楼，如果有人按 18 楼，那么电梯在上升到 5 楼的时候，有人按了 10 楼，这时候是否会在 10 楼先停下来；

●电梯下降到 10 层时显示满员，此时若 8 层有人等待电梯，是否在 8 层停；

●可靠性：门关上的一刹那出现障碍物，同时按关门和开门按钮，点击当前楼层号码，多次点击同一楼层的号码等等；同时按上键和下键会怎样；

●易用性：电梯的按钮的设计符合一般人使用的习惯吗。

●用户文档：使用手册是否对电梯的用法、限制、使用条件等有详细描述

●压力测试：看电梯的最大限度的承受重量。在负载过重时报警装置是否有提醒。在一定时间内不断的让电梯上升，下降。最大负载下平稳运行的最长时间。

## 200. 测试项目：杯子

需求测试: 查看杯子使用说明书

界面测试: 查看杯子外观

功能度: 用水杯装水看漏不漏；水能不能被喝到

安全性: 杯子有没有毒或细菌

可靠性: 杯子从不同高度落下的损坏程度

可移植性: 杯子在不同的地方、温度等环境下是否都可以正常使用

兼容性: 杯子是否能够容纳果汁、白水、酒精、汽油等

易用性: 杯子是否烫手、是否有防滑措施、是否方便饮用

用户文档: 使用手册是否对杯子的用法、限制、使用条件等有详细描述

疲劳测试: 将杯子盛上水(案例一)放24小时检查泄漏时间和情况；盛上汽油(案例二)放24小时检查泄漏时间和情况等

压力测试: 用根针并在针上面不断加重量，看压强多大时会穿透

跌落测试: 杯子加包装(有填充物)，在多高的情况下摔下不破损

震动测试: 杯子加包装(有填充物)，六面震动，检查产品是否能应对恶劣的铁路\公路\航空运输

测试数据: 测试数据具体编写此处略(最讨厌写测试数据了)。其中应用到：场景法、等价类划分法、因果图法、错误推测法、边界值法等方法

期望输出: 该期望输出需查阅国标、行标以及使用用户的需求

## 201. 如何测试一个杯子

在软件测试的面试中，经常会碰到类似的问题。比如：如何测试一个杯子，或者如何测试一只笔。要求你设计20个以上的test case。

这类的面试题目，是考察面试者是否熟悉各种软件测试方法，设计test case的能力，以及test sense。

首先应该反问下面试官，需求是什么样的，比如大概是个什么样的杯子。  
我回答这类问题的思路，从软件测试的各种不同方法来联想，具体如下。

### 功能测试(Function test)

1. 能否装水，
2. 除了装水，能否装其他液体。比如可乐，酒精
3. 能装多少ML的水
4. 杯子是否有刻度表
5. 杯子能否泡茶，泡咖啡
6. 杯子是否能放冰箱，做冰块
7. 杯子的材质是什么(玻璃，塑料，黄金做的)

### 界面测试(UI Test)

1. 外观好不好看。
2. 什么颜色
3. 杯子的形状是怎么样的。
4. 杯子的重量是多少
5. 杯子是否有异味
6. 杯子的图案是否合理

### 性能测试(performance test)

1. 能否装100度的开水(泡茶)
2. 能否装0度冰水

3. 装满水，放几天后，是否会漏水
4. 杯子内壁上的涂料是否容易脱落。
5. 杯子上的颜色是否容易褪色或者脱落
6. 被我坦克压下，是否会碎（这条是开玩笑的哈）

#### 安全性测试(Security test)

1. 制作杯子的材料，是否有毒
2. 放微波炉里转的时候，是否会爆炸，或者杯子是否会熔化。
3. 从桌子上掉到水泥地上是否会摔碎。
4. 杯子是否容易长细菌
5. 杯子是否有缺口，会划坏嘴巴
6. 杯子内壁上的材料，是否会溶解到水中
7. 杯子破碎后，是否会对使用者造成伤害

#### 可用性测试(Usability Test)

1. 杯子是否容易烫手
2. 杯子是否好端，好拿
3. 杯子的水是否容易喝到
4. 杯子是否有防滑措施

#### 总结

一不小心，就写了 30 多个 test case。

下次碰到“如何测试一台自动售货机”这样的问题，按照相同的思路，也能容易的回答出来

## 202. 测试题目：桌子

需求测试：查看国家相关标准。

功能：桌子是办公，或者放置用的，首先考虑桌子的面积大小是否适度。

界面：桌子的版面是否平滑，桌子有没有凹凸不平的地方

安全：桌子肯定有它的支撑点，若支撑点不稳，容易摔坏物品，使用起来也不方便。

易用：桌子的移动性好不好，它的重量是否合适

可靠性：将桌子推倒后，再检查桌子是否很容易被损坏。

性能：将很重的物品放在桌子上，看它最大承受的重量是多少…

## 203. 测试题目：洗衣机

功能测试：该洗衣机是否能正常的洗衣服

需求测试：查看洗衣机的使用说明书和安全说明书等

性能测试：使用时用电量如何，是否满足用户需求

界面测试：洗衣机的外观是否满足客户的需求

易用测试：该洗衣机是否容易操作

兼用性测试：该洗衣机除了能洗衣服以外还能洗别的吗

安全性测试：该洗衣机通电以后人接触以后是否有电

负载测试：通过逐步增加系统负载，最终确定在什么负载条件下系统性能将处于崩溃状态，以此获得系统能提供的最大服务

压力测试：通过逐步增加系统负载，最终确定在什么负载条件下系统性能将处于崩溃状态，以此获得系统能提供的最大服务级别的测试。

稳定性测试：加到一定的衣服然后过一段时间看洗衣机是否正常洗

## 204. 对一把椅子进行功能测试

### 功能测试:

1. 能不能供人坐，即能不能供人使用。
2. 坐上去是否摇晃。
3. 坐人后是否会发出响声。
4. 椅子上会不会掉颜色，即坐上去，来回摩擦椅子上的颜色会不会粘到衣服上
5. 有水撒到椅子上的时候，用布子或纸擦的时候会不会掉颜色。能不能擦干净水。
6. 坐上去会不会有塌陷的感觉。
7. 从椅子上离开的时候会不会发出响声。
8. 椅子会不会轻易挂到衣服。
9. 靠在椅背上的时候会不会，发出响声，椅子会不会摇晃。
10. 椅子脏了是能易清理干净。
11. 是否只能供一个人坐

### 性能测试:

1. 椅子能承受多大的重量，不会发出响声；能承受多大的重量不被压坏。
2. 椅子是否怕水
3. 椅子是否怕火
4. 椅子是否能在压了重物的情况下，然后摇晃，能坚持不长时间不响\不坏.
5. 椅背，用力向后靠椅背，检测椅背的向后的承受能力.

### 安全性测试:

1. 椅子的材质是否与用户说明书或质量保证书上的一样。
2. 椅子的材料是否对人体有危害。
3. 在撒到椅子上水/饮料等液体的时候，椅子会不会产生什么有害的物质。
4. 在椅子被磨损的时候，会不会有划伤或擦伤用户的可能。
5. 坐在椅子上的时候，是否安全，例如在只坐到椅子最前端的一部分时，椅子会不会失去平衡等等。
6. 在与椅子摩擦的时候，会产生一定的容量，在摩擦的比较厉害的时候，会不会，产生有害的气体或物质。例如，产生难闻的气味等等。
7. 在人坐或踩在椅子上时椅子是否稳固，即不摇晃等。

### 外观/适用性测试（界面/适用性测试）:

1. 椅子的外观是否美观实用。
2. 是否与用户说明书或质量保证书上的一样出现的实物图相同。
3. 椅子的气味/扶手/坐垫及靠垫的软硬度是否合适。
4. 椅子是否容易挪动。
5. 椅子的高度/重量/材质是否合适。
6. 椅子的适用场合是否合适

205. 反问

206. 自我介绍

207. 工作经历

208. 项目中收获了什么

209. 平时怎么学习的

210. 薪资结构和考勤介绍

211. 期望薪资

212. 为什么是这个期望

213. 当前的月薪

214. 哪里获得的招聘信息

215. 反问问题

216. 引用与指针有什么区别？

引用必须被初始化，指针不必。

引用初始化以后不能被改变，指针可以改变所指的对象。

不存在指向空值的引用，但是存在指向空值的指针。

Internet. 采用哪种网络协议？该协议的主要层次结构？Internet. 物理地址和IP. 地址转换采用什么协议？

TCP/IP 协议主要层次结构为： 应用层/传输层/网络层/数链路层。

ARP (Address Resolution Protocol) (地据址解析协议)

217. 说说你对集成测试中自顶向下集成和自底向上集成两个策略的理解，要谈出它们各自的优缺点和主要适应于哪种类型测试？

1、自顶向下集成

优点：较早地验证了主要控制和判断点；按深度优先可以首先实现和验证一个完整的软件功能；功能较早证实，带来信心；只需一个驱动，减少驱动器开发的费用；支持故障隔离。

缺点：柱的开发量大；底层验证被推迟；底层组件测试不充分。

适应于产品控制结构比较清晰和稳定；高层接口变化较小；底层接口未定义或经常可能被修改；产口控制组件具有较大的技术风险，需要尽早被验证；希望尽早能看到产品的系统功能行为。

2、自底向上集成

优点：对底层组件行为较早验证；工作最初可以并行集成，比自顶向下效率高；减少了桩的工作量；支持故障隔离。

缺点：驱动的开发工作量大；对高层的验证被推迟，设计上的错误不能被及时发现。

适应于底层接口比较稳定；高层接口变化比较频繁；底层组件较早被完成。

系统测试的策略有很多的，有性能测试、负载测试、强度测试、易用性测试、安全测试、配置测试、安装测试、文档测试、故障恢复测试、用户界面测试、恢复测试、分布测试、可用性测试。

设计系统测试计划需要参考的项目文档有软件测试计划、软件需求工件、和迭代计划。

通过画因果图来写测试用例的步骤为…及把因果图转换为状态图共五个步骤。

利用因果图生成测试用例的基本步骤是：

分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符。

分析软件规格说明描述中的语义，找出原因与结果之间，原因与原因之间对应的是什么关系？根据这些关系，画出因果图。

由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

把因果图转换成判定表。

3、请说出这些测试最好由那些人员完成，测试的是什么？

代码、函数级测试一般由白盒测试人员完成，他们针对每段代码或函数进行正确

性检验，检查其是否正确的实现了规定的功能。

模块、组件级测试主要依据是程序结构设计测试模块间的集成和调用关系，一般由测试人员完成。

系统测试在于模块测试与单元测试的基础上进行测试。了解系统功能与性能，根据测试用例进行全面的测试。

设计测试用例时应该考虑哪些方面，即不同的测试用例针对那些方面进行测试？

设计测试用例时需要注意的是，除了对整体流程及功能注意外，还要注意强度测试、性能测试、压力测试、边界值测试、稳定性测试、安全性测试等多方面。

（测试用例需要考虑的四个基本要素是输入、输出、操作和测试环境；另外，测试用例需要考虑的是测试类型（功能、性能、安全……），这部分可以参照 TP 做答。此外，还需要考虑用例的重要性和优先级）

在 windows. 下保存一个文本文件时会弹出保存对话框，如果为文件名建立测试用例，等价类应该怎样划分？

单字节，如 A; 双字节， AA、我我；特殊字符 / ‘。 ‘；、-=等；保留字，如 com; 文件格式为 8.3 格式的；文件名格式为非 8.3 格式的；/,\*等九个特殊字符。假设有一个文本框要求输入 10. 个字符的邮政编码，对于该文本框应该怎样划分等价类？

特殊字符，如 10 个\*或¥；英文字母，如 ABCDefghik；小于十个字符，如 123；大于十个字符，如 1111111111；数字和其他混合，如 123AAAAAAA；空字符；保留字符

#### 4、软件测试项目从什么时候开始？为什么？

软件测试应该在需求分析阶段就介入，因为测试的对象不仅仅是程序编码，应该对软件开发过程中产生的所有产品都测试，并且软件缺陷存在放大趋势. 缺陷发现的越晚，修复它所花费的成本就越大.

#### 5、什么是回归测试？

回归测试 (regression testing)：回归测试有两类：用例回归和错误回归；用例回归是过一段时间以后再回头对以前使用过的用例在重新进行测试，看看会重新发现问题。

错误回归，就是在新版本中，对以前版本中出现并修复的缺陷进行再次验证，并以缺陷为核心，对相关修改的部分进行测试的方法。

#### 6、单元测试、集成测试、系统测试的侧重点是什么？

单元测试针对的是软件设计的最小单元 - 程序模块（面向过程中是函数、过程；面向对象中是类。），进行正确性检验的测试工作，在于发现每个程序模块内部可能存在的差错. 一般有两个步骤：人工静态检查\动态执行跟踪

集成测试针对的是通过了单元测试的各个模块所集成起来的组件进行检验，其主要内容是各个单元模块之间的接口，以及各个模块集成后所实现的功能。

系统测试针对的是集成好的软件系统，作为整个计算机系统的一个元素，与计算机硬件\外设\某些支持软件\数据和人员等其他系统元素结合在一起，要在实际的运行环境中，对计算机系统进行一系列的集成测试和确认测试。

#### 7、一个测试工程师应具备那些素质？

- 1、责任心
- 2、沟通能力
- 3、团队合作精神
- 4、耐心、细心、信心

5、时时保持怀疑态度，并且有缺陷预防的意识

6、具备一定的编程经验

## 8、你所了解的的软件测试类型都有哪些，简单介绍一下

按测试策略分类：

1、静态与动态测试

2、黑盒与白盒测试

3、手工和自动测试

4、冒烟测试

5、回归测试

按测试阶段分类：单元测试、集成测试、系统测试；

## 其他常见测试方法：

1、功能测试

2、性能测试

3、压力测试

4、负载测试

5、易用性测试

6、安装测试

7、界面测试

8、配置测试

9、文档测试

10、兼容性测试

11、安全性测试

12、恢复测试

## 9、你认为做好测试计划工作的关键是什么？

明确测试的目标，增强测试计划的实用性

编写软件测试计划得重要目的就是使测试过程能够发现更多的软件缺陷，因此软件测试计划的价值取决于它对帮助管理测试项目，并且找出软件潜在的缺陷。

因此，软件测试计划中的测试范围必须高度覆盖功能需求，测试方法必须切实可行，测试工具并且具有较高的实用性，便于使用，生成的测试结果直观、准确  
坚持“5W”规则，明确内容与过程

“5W”规则指的是“What（做什么）”、“Why（为什么做）”、“When（何时做）”、“Where（在哪里）”、“How（如何做）”。利用“5W”规则创建软件测试计划，可以帮助测试团队理解测试的目的（Why），明确测试的范围和内容（What），确定测试的开始和结束日期（When），指出测试的方法和工具（How），给出测试文档和软件的存放位置（Where）。

采用评审和更新机制，保证测试计划满足实际需求

测试计划写作完成后，如果没有经过评审，直接发送给测试团队，测试计划内容的可能不准确或遗漏测试内容，或者软件需求变更引起测试范围的增减，而测试计划的内容没有及时更新，误导测试执行人员。

分别创建测试计划与测试详细规格、测试用例。

应把详细的测试技术指标包含到独立创建的测试详细规格文档，把用于指导测试小组执行测试过程的测试用例放到独立创建的测试用例文档或测试用例管理数

据库中。

测试计划和测试详细规格、测试用例之间是战略和战术的关系，测试计划主要从宏观上规划测试活动的范围、方法和资源配置，而测试详细规格、测试用例是完成测试任务的具体战术。

#### 10、您认为做好测试用例设计工作的关键是什么？

白盒测试用例设计的关键是以较少的用例覆盖尽可能多的内部程序逻辑结果。

黑盒法用例设计的关键同样也是以较少的用例覆盖模块输出和输入接口。不可能做到完全测试，以最少的用例在合理的时间内发现最多的问题。

#### 11、你的测试职业发展目标是什么？

测试经验越多，测试能力越高。所以我的职业发展是需要时间累积的，一步步向着高级测试工程师奔去。而且我也有初步的职业规划，前 3 年累积测试经验，不断的更新自己改正自己，做好测试任务。

#### 12、测试结束的标准是什么？

从微观上来说，在测试计划中定义，比如系统在一定性能下平稳运行 72 小时，目前 Bug Tracking System 中，本版本中没有一般严重的 BUG，普通 BUG 的数量在 3 以下，BUG 修复率 90%以上等等参数，然后由开发经理，测试经理，项目经理共同签字认同版本 Release。

如果说宏观的，则是当这个软件彻底的消失以后，测试就结束了。

#### 13、一套完整的测试应该由哪些阶段组成？

可行性分析、需求分析、概要设计、详细设计、编码、单元测试、集成测试、系统测试、验收测试。

#### 14、在您以往的工作中，一条软件缺陷（或者叫 Bug）记录都包含了哪些内容？

如何提交高质量的软件缺陷（Bug）记录？

一条 Bug 记录最基本应包含：

bug 编号；

bug 严重级别，优先级；

bug 产生的模块；

首先要有 bug 摘要，阐述 bug 大体的内容；

bug 对应的版本；

bug 详细现象描述，包括一些截图、录像…等等；

bug 出现时的测试环境，产生的条件即对应操作步骤；

高质量的 Bug 记录：

通用 UI 要统一、准确

缺陷报告的 UI 要与测试的软件 UI 保持一致，便于查找定位。

（尽量使用业界惯用的表达术语和表达方法）

使用业界惯用的表达术语和表达方法，保证表达准确，体现专业化。

每条缺陷报告只包括一个缺陷

每条缺陷报告只包括一个缺陷，可以使缺陷修正者迅速定位一个缺陷，集中精力每次只修正一个缺陷。校验者每次只校验一个缺陷是否已经正确修正。

不可重现的缺陷也要报告

首先缺陷报告必须展示重现缺陷的能力。不可重现的缺陷要尽力重现，若尽力之后仍不能重现，仍然要报告此缺陷，但在报告中要注明无法再现，缺陷出现的频率。

### 明确指明缺陷类型

根据缺陷的现象，总结判断缺陷的类型。例如，即功能缺陷、界面缺陷、数据缺陷，合理化建议这是最常见的缺陷或缺陷类型，其他形式的缺陷或缺陷也从属于其中某种形式。

### 明确指明缺陷严重等级和优先等级

时刻明确严重等级和优先等级之间的差别。高严重问题可能不值得解决，小装饰性问题可能被当作高优先级。

描述 (Description)，简洁、准确，完整，揭示缺陷实质，记录缺陷或缺陷出现的位置描述要准确反映缺陷的本质内容，简短明了。为了便于在软件缺陷管理数据库中寻找制定的测试缺陷，包含缺陷发生时的用户界面 (UI) 是个良好的习惯。例如记录对话框的标题、菜单、按钮等控件的名称。

短行之间使用自动数字序号，使用相同的字体、字号、行间距，可以保证各条记录格式一致，做到规范专业。

### 每一个步骤尽量只记录一个操作

保证简洁、条理井然，容易重复操作步骤。

确认步骤完整，准确，简短。

保证快速准确的重复缺陷，“完整”即没有缺漏，“准确”即步骤正确，“简短”即没有多余的步骤。

根据缺陷，可选择是否进行图象捕捉。

为了直观的观察缺陷或缺陷现象，通常需要附加缺陷或缺陷出现的界面，以图片的形式作为附件附着在记录的“附件”部分。

为了节省空间，又能真实反映缺陷或缺陷本质，可以捕捉缺陷或缺陷产生时的全屏幕，活动窗口和局部区域。为了迅速定位、修正缺陷或缺陷位置，通常要求附加中文对照图。

### 附加必要的特殊文档和个人建议和注解 1

如果打开某个特殊的文档而产生的缺陷或缺陷，则必须附加该文档，从而可以迅速再现缺陷或缺陷。有时，为了使缺陷或缺陷修正者进一步明确缺陷或缺陷的表现，可以附加个人的修改建议或注解。

### 检查拼写和语法缺陷

在提交每条缺陷或缺陷之前，检查拼写和语法，确保内容正确，正确的描述缺陷。尽量使用短语和短句，避免复杂句型句式。

软件缺陷管理数据库的目的是便于定位缺陷，因此，要求客观的描述操作步骤，不需要修饰性的词汇和复杂的句型，增强可读性。

以上概括了报告测试缺陷的规范要求，随着软件的测试要求不同，测试者经过长期测试，积累了相应的测试经验，将会逐渐养成良好的专业习惯，不断补充新的规范书写要求。

此外，经常阅读、学习其他测试工程师的测试缺陷报告，结合自己以前的测试缺陷报告进行对比和思考，可以不断提高技巧。

### 缺陷描述内容

缺陷描述的内容可以包含缺陷操作步骤，实际结果和期望结果。操作步骤可以方便开发人员再现缺陷进行修正，有些开发的再现缺陷能力很差，虽然他明白你所指的缺陷，

但就是无法再现特别是对系统不熟悉的加入开发人员，介绍步骤可以方便他们

再现。实际结果可以让开发明白错误是什么，期望结果可以让开发了解正确的结果应该是如何。

### 15、黑盒测试和白盒测试是软件测试的两种基本方法，请分别说明各自的优点和缺点！

黑盒测试的优点：比较简单，不需要了解程序内部的代码及实现；与软件的内部实现无关；从用户角度出发，能很容易的知道用户会用到哪些功能，会遇到哪些问题；

基于软件开发文档，所以也能知道软件实现了文档中的哪些功能；在做软件自动化测试时较为方便。

黑盒测试的缺点：不可能覆盖所有的代码，覆盖率较低，大概只能达到总代码量的 30%；自动化测试的复用性较低。

白盒测试的优点：帮助软件测试人员增大代码的覆盖率，提高代码的质量，发现代码中隐藏的问题。

白盒测试的缺点：程序运行会有很多不同的路径，不可能测试所有的运行路径；测试基于代码，只能测试开发人员做的对不对，而不能知道设计的正确与否，可能会漏掉一些功能需求；系统庞大时，测试开销会非常大。

### 17、测试计划工作的目的是什么？测试计划文档的内容应该包括什么？其中哪些是最重要的？

软件测试计划是指导测试过程的纲领性文件：

领导能够根据测试计划进行宏观调控，进行相应资源配置等；

测试人员能够了解整个项目测试情况以及项目测试不同阶段的所要进行的工作等；

便于其他人员了解测试人员的工作内容，进行有关配合工作；

包含了产品概述、测试策略、测试方法、测试区域、测试配置、测试周期、测试资源、测试交流、风险分析等内容。借助软件测试计划，参与测试的项目成员，

尤其是测试管理人员，可以明确测试任务和测试方法，保持测试实施过程的顺畅沟通，跟踪和控制测试进度，应对测试过程中的各种变更。

### 17、测试计划编写 6 要素

(5W1H)

why——为什么要进行这些测试；

what——测试哪些方面，不同阶段的工作内容；

when——测试不同阶段的起止时间；

where——相应文档，缺陷的存放位置，测试环境等；

who——项目有关人员组成，安排哪些测试人员进行测试；

how——如何去做，使用哪些测试工具以及测试方法进行测试

测试计划和测试详细规格、测试用例之间是战略和战术的关系，测试计划主要从宏观上规划测试活动的范围、方法和资源配置，而测试详细规格、测试用例是完成测试任务的具体战术。

所以其中最重要的是测试测试策略和测试方法（最好是能先评审）。

### 18、黑盒测试的测试用例常见设计方法都有哪些？请分别以具体的例子来说明这些方法在测试用例设计工作中的应用

1) 等价类划分：等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。并合理地假定：测试某等价类的代表值就

等于对这一类其它值的测试。

因此，可以把全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据。取得较好的测试结果。等价类划分可有两种不同的情况：有效等价类和无效等价类。

2) 边界值分析法：是对等价类划分方法的补充。测试工作经验告诉我，大量的错误是发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。

使用边界值分析方法设计测试用例，首先应确定边界情况。通常输入和输出等价类的边界，就是应着重测试的边界情况。应当选取正好等于，刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

3) 错误猜测法：基于经验和直觉推测程序中所有可能存在的各种错误，从而有针对性的设计测试用例的方法。

错误推测方法的基本思想：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据他们选择测试用例。例如，在单元测试时曾列出的许多在模块中常见的错误。以前产品测试中曾经发现的错误等，这些就是经验的总结。

还有，输入数据和输出数据为 0 的情况。输入表格为空格或输入表格只有一行。这些都是容易发生错误的情况。可选择这些情况下的例子作为测试用例。

4) 因果图方法：前面介绍的等价类划分方法和边界值分析方法，都是着重考虑输入条件，但未考虑输入条件之间的联系，相互组合等。考虑输入条件之间的相互组合，可能会产生一些新的情况。

但要检查输入条件的组合不是一件容易的事情，即使把所有输入条件划分成等价类，他们之间的组合情况也相当多。因此必须考虑采用一种适合于描述对于多种条件的组合。

相应产生多个动作的形式来考虑设计测试用例。这就需要利用因果图（逻辑模型）。因果图方法最终生成的就是判定表。它适合于检查程序输入条件的各种组合情况。

5) 正交表分析法：可能因为大量的参数的组合而引起测试用例数量上的激增，同时，这些测试用例并没有明显的优先级上的差距，而测试人员又无法完成这么多数量的测试，就可以通过正交表来进行缩减一些用例，从而达到尽量少的用例覆盖尽量大的范围的可能性。

6) 场景分析方法：指根据用户场景来模拟用户的操作步骤，这个比较类似因果图，但是可能执行的深度和可行性更好。

7) 状态图法：通过输入条件和系统需求说明得到被测系统的所有状态，通过输入条件和状态得出输出条件；通过输入条件、输出条件和状态得出被测系统的测试用例。

8) 大纲法：大纲法是一种着眼于需求的方法，为了列出各种测试条件，就将需求转换为大纲的形式。大纲表示为树状结构，在根和每个叶子结点之间存在唯一的路径。

大纲中的每条路径定义了一个特定的输入条件集合，用于定义测试用例。树中叶子的数目或大纲中的路径给出了测试所有功能所需测试用例的大致数量。

## 19、详细的描述一个测试活动完整的过程

（供参考，本答案主要是瀑布模型的做法）

项目经理通过和客户的交流，完成需求文档，由开发人员和测试人员共同完成需求文档的评审，评审的内容包括：需求描述不清楚的地方和可能有明显冲突或者

无法实现的功能的地方。

项目经理通过综合开发人员，测试人员以及客户的意见，完成项目计划。然后 SQA 进入项目，开始进行统计和跟踪。

开发人员根据需求文档完成需求分析文档，测试人员进行评审，评审的主要内容包括是否有遗漏或双方理解不同的地方。测试人员完成测试计划文档，测试计划包括的内容上面有描述。

测试人员根据修改好的需求分析文档开始写测试用例，同时开发人员完成概要设计文档，详细设计文档。此两份文档成为测试人员撰写测试用例的补充材料。

测试用例完成后，测试和开发需要进行评审。

## 20、测试人员搭建环境

开发人员提交第一个版本，可能存在未完成功能，需要说明。测试人员进行测试，发现 BUG 后提交给 BugZilla。

开发提交第二个版本，包括 Bug Fix 以及增加了部分功能，测试人员进行测试。重复上面的工作，一般是 3-4 个版本后 BUG 数量减少，达到出货的要求。

如果有客户反馈的问题，需要测试人员协助重现并重新测试。

BUG. 管理工具的跟踪过程（用 BugZilla 为例子）

测试人员发现了 BUG，提交到 Bugzilla 中，状态为 new，BUG 的接受者为开发接口人员。发接口将 BUG 分配给相关的模块的开发人员，状态修改为已分配，开发人员和测试确认 BUG，如果是本人的 BUG，则设置为接收；如果是别的开发人员的问题，则转发出去，由下一个开发人员来进行此行为；如果认为不是问题，则需要大家讨论并确认后，拒绝这个 BUG，然后测试人员关闭此问题。

如果开发人员接受了 BUG，并修改好以后，将 BUG 状态修改为已修复，并告知测试在哪个版本中可以测试。

测试人员在新版本中测试，如果发现问题依然存在，则拒绝验证；如果已经修复，则关闭 BUG。

## 21、您认为在测试人员同开发人员的沟通过程中，如何提高沟通的效率和改善沟通的效果？维持测试人员同开发团队中其他成员良好的人际关系的关键是什么？

尽量面对面的沟通，其次是能直接通过电话沟通，如果只能通过 Email 等非及时沟通工具的话，强调必须对特性的理解深刻以及能表达清楚。

运用一些测试管理工具如 TestDirector 进行管理也是较有效的方法，同时要注意在 TestDirector 中对 BUG 有准确的描述。

在团队中建立测试人员与开发人员良好沟通中注意以下几点：

一真诚、二是团队精神、三是在专业上有共同语言、四是要对事不对人，工作至上。

当然也可以通过直接指出一些小问题，而不是进入 BUG Tracking System 来增加对方的好感。

## 22、你对测试最大的兴趣在哪里？为什么？

回答这个面试题，没有固定统一的答案，但可能是许多企业都会问到的。提供以下答案供考：

最大的兴趣，感觉这是一个有挑战性的工作；

测试是一个经验行业，工作越久越能感觉到做好测试的难度和乐趣，过自己的工作，能使软件产品越来越完善，从中体会到乐趣，答此类问题注意以下几个方面：尽可能的切合招聘企业的技术路线来表达你的兴趣，例如该企业是数据库应用的

企业，那么表示你的兴趣在数据库的测试，并且希望通过测试提升自己的数据库掌握能力。

表明你做测试的目的是为了提升能力，也是为了更好的做好测试；提升能力不是为了以后转开发或其他的，除非用人企业有这样的安排。

不要过多的表达你的兴趣在招聘企业的范畴这外。比如招聘企业是做财务软件的，可是你表现出来的是对游戏软件的兴趣；或招聘是做 JAVA 开发的，而你的兴趣是在 C 类语言程序的开发。

### 23、你自认为测试的优势在哪里？

该面试也没有固定不变的答案，但可参考以下几点，并结合自身特点：

有韧性、有耐心、做事有条理性、喜欢面对挑战、有信心做好每一件事情、较强的沟通能力、从以前的经理处都得到了很好的评价表明我做的很好。

### 24、简述你在以前的工作中做过哪些事情，比较熟悉什么？

我过去的主要工作是系统测试和自动化测试。在系统测试中，主要是对 BOSS 系统的业务逻辑功能，以及软交换系统的 Class 5 特性进行测试。性能测试中，主要是进行的压力测试，

在各个不同数量请求的情况下，获取系统响应时间以及系统资源消耗情况。自动化测试主要是通过自己写脚本以及一些第三方工具的结合来测试软交换的特性测试。

在测试中，我感觉对用户需求的完全准确的理解非常重要。另外，就是对 BUG 的管理，要以需求为依据，并不是所有 BUG 均需要修改。

测试工作需要耐心和细致，因为在新版本中，虽然多数原来发现的 BUG 得到了修复，但原来正确的功能也可能变得不正确。因此要注重迭代测试和回归测试。

### 25、在 C/C++. 中 static. 有什么用途？请至少说明两种

在函数体，一个被声明为静态的变量在这一函数被调用过程中维持其值不变。

在模块内（但在函数体外），一个被声明为静态的变量可以被模块内所用函数访问，但不能被模块外其它函数访问。它是一个本地的全局变量。

在模块内，一个被声明为静态的函数只可被这一模块内的其它函数调用。那就是，这个函数被限制在声明它的模块的本地范围内使用。

## 面试必问的 25 道数据库测试题

### 1) 什么是数据库测试？

数据库测试也称为后端测试。数据库测试分为四个不同的类别。

- 数据完整性测试
- 数据有效性测试
- 数据库相关的性能
- 测试功能，程序和触发器

### 2) 在数据库测试中，我们需要正常检查什么？

通常，我们在 DB Testing 中检查的内容是：

- 约束检查
- 验证字段大小
- 存储过程
- 将应用程序字段大小与数据库匹配
- 基于绩效的问题的索引

### 3) 解释什么是数据驱动测试？

在数据表中，为了测试多个数据，使用数据驱动的测试。通过使用它，它可以很

容易地从不同位置同时替换参数。

**4) 什么是连接并提及不同类型的连接?**

Join 用于显示两个或两个以上的表，连接类型为：

- 自然加入
- 内部联接
- 外加入
- 交叉加入

外部联接又分为两部分：

- 左外连接
- 右外连接

**5) 什么是索引并提及不同类型的索引?**

索引是数据库对象，它们是在列上创建的。为了快速获取数据，经常访问它们。

不同类型的索引是：

- B 树索引
- 位图索引
- 聚集索引
- 覆盖指数
- 非唯一索引
- 独特的指数

**6) 在测试存储过程时，测试人员采取了哪些步骤?**

测试人员将检查存储过程的标准格式，并检查字段是否正确，如存储过程中提到的更新，连接，索引，删除。

**7) 您如何知道数据库测试，是否触发了触发器?**

在查询公共审计日志时，您会知道是否触发了触发器。它位于审计日志中，您可以在其中查看触发的触发器。

**8) 在数据库测试中，测试数据加载的步骤是什么?**

以下步骤需要遵循测试数据加载

- 应该知道源数据
- 目标数据应该是已知的
- 应检查源和目标的兼容性
- 在 SQLEnterprise 管理器中，打开相应的 DTS 包后运行 DTS 包
- 您必须比较目标和数据源的列
- 应检查目标和源的行数
- 更新源中的数据后，检查更改是否显示在目标中。
- 检查 NULL 和垃圾字符

**9) 如何不使用数据库检查点，如何在 QTP 中测试 SQL 查询?**

通过在 VBScript 中编写脚本程序，我们可以连接到数据库并可以测试查询和数据库。

**10) 解释如何在 QTP 中使用 SQL 查询?**

在使用输出数据库检查点和数据库检查的 QTP 中，您必须选择 SQL 手动查询选项。选择手动查询选项后，输入“选择”查询以获取数据库中的数据，然后比较预期和实际。

**11) 为数据库测试编写测试用例的方法是什么?**

编写测试用例就像功能测试一样。首先，您必须了解应用程序的功能要求。然后

你必须决定编写测试用例的参数

- 目标：写出您想要测试的目标
- 输入法：编写要执行的操作方法或输入
- 预期：它应该如何出现在数据库中

**12) 要管理和操作测试表，您在数据库测试中使用了哪些 SQL 语句？**

SELECT, INSERT, UPDATE, DELETE 等语句用于操作表，而 ALTER TABLE, CREATE TABLE 和 DELETE TABLE 用于管理表。

**13) 如何测试数据库程序和触发器？**

要测试数据库过程和触发器，必须知道输入和输出参数。EXEC 语句可用于运行该过程并检查表的行为。

- 在 solution explorer 中打开数据库项目
- 现在，在“视图”菜单中，单击数据库架构
- 从架构视图菜单中打开项目文件夹
- 右键单击要测试的对象，然后单击“创建单元测试”对话框
- 之后创建一个新的语言测试项目
- 选择 a) 插入单元测试或 b) 创建新测试，然后单击“确定”
- 必须配置的项目将通过单击“项目配置”对话框完成。
- 配置完成后单击“确定”

**14) 如何根据需求编写测试用例，这些要求是否代表 AUT（被测试应用程序）的确切功能？**

要根据需求编写测试用例，您需要在功能方面彻底分析需求。此后，您可以考虑使用相应的测试用例设计技术，如等效分区，黑盒设计，原因效果绘图等来编写测试用例。是的，这些要求代表了 AUT 的确切功能。

**15) 什么是 DBMS？**

DBMS 代表数据库管理系统，有不同类型的 DBMS

- 网络模型
- 分层模型
- 关系模型

**16) 什么是 DML？**

DML 代表数据操作语言，它用于使用模式对象管理数据。它是 SQL 的一个子集。

**17) 什么是 DCL 命令？DCL 使用的两种命令有哪些？**

DCL 代表数据控制语言，它用于控制数据。

两种类型的 DCL 命令是：

授权：通过使用此命令，用户可以访问数据库的权限

撤消：使用此命令，用户无法访问数据库

**18) 什么是白盒测试和黑盒测试？**

黑盒测试意味着在给出特定输入时测试软件的输出。通常执行此测试以查看软件是否满足用户的要求。运行此测试不需要特定的功能输出。

进行白盒测试以检查程序的代码和逻辑的准确性。该测试由了解系统逻辑流程的程序员完成。

**19) QTP 如何评估测试结果？**

测试完成后，QTP 将生成一份报告。此报告将显示测试时检测到的检查点，系统消息和错误。测试结果窗口将显示在检查点遇到的任何不匹配。

**20) 解释 QTP 测试过程？**

- QTP 测试过程基于以下步骤：
- 创建 GUI (图形用户界面) 映射文件：标识必须测试的 GUI 对象
- 创建测试脚本：记录测试脚本
- 调试测试：应该调试测试
- 运行测试：应该运行测试用例。
- 查看结果：结果反映了测试的成功或失败
- 报告检测：如果测试失败，原因将记录在报告检测文件中

## 21) 什么是负载测试并给出一些示例？

要测量系统响应，请进行负载测试。如果负载超过用户模式，则称为压力测试。负载测试的示例是下载一组大文件，在一台计算机上执行多个应用程序，使服务器接收大量电子邮件并将许多任务分配给打印机。

## 22) 如何手动测试数据库？

手动测试数据库涉及检查后端的数据并查看前端数据的添加是否影响后端，删除，更新，插入等是否相同。

## 23) RDBMS 代表什么，SQL 使用什么是重要的 RDBMS？

RDBMS 代表使用 SQL 的关系数据库管理系统，SQL 使用的重要 RDBMS 是 Sybase，Oracle，Access，Ingres，Microsoft SQL 服务器等。

## 24) 什么是性能测试以及性能测试的瓶颈是什么？

性能测试决定了计算机系统性能的速度。它包括定量测试，如响应时间测量。性能测试中的问题是，您总是需要训练有素且经验丰富的人力，而且您使用的工具也很昂贵。

## 25) 什么是 DDL 以及它们的命令是什么？

要定义数据库结构，Developer 使用 DDL。DDL 代表数据定义语言。各种 DDL 命令包括 Create，Truncate，Drop，Alter，Comment 和 Rename。

## 十二. allure 装饰器介绍

使用方法	参数值	参数说明
@allure.suite()	测试套件	测试(集)套件，不用报告默认显示py文件名
@allure.epic()	epic描述	敏捷里面的概念，定义史诗，往下是feature
@allure.feature()	模块名称	功能点的描述，往下是story
@allure.story()	用户故事	用户故事，往下是title
@allure.tag()	测试用例标记	用于给用例打个标记
@allure.title(用例的标题)	用例的标题	重命名html报告名称
@allure.testcase()	测试用例的链接地址	对应功能测试用例系统里面的case
@allure.issue()	缺陷	对应缺陷管理系统里面的链接
@allure.description()	用例描述	测试用例的描述
@allure.step()	操作步骤	测试用例的步骤
@allure.severity()	用例等级	blocker, critical, normal, minor, trivial
@allure.link()	链接	定义一个链接，在测试报告展现
@allure.attachment()	附件	报告添加附件

### @allure 装饰器中的一些功能点：

1) 想对你的用例集分层，增加可读性，可以使用以下三个装饰器，写在类或方法前面：

- @allure.epic : 敏捷里的概念, 定义史诗
  - @allure.feature : 用于定义被测试的功能, 被测产品的需求点
  - @allure.story : 用于定义被测功能的用户场景, 即子功能点

2) 想对单个用例添加标题和用例描述, 增加可读性, 使用以下两个装饰器:

  - @allure.description : 添加测试用例描述, 参数类型为 str, 与使用'','','',''效果类似。如果想传 html, 请使用 @allure.description\_html
  - @allure.title : 修改单个测试用例标题, 支持传递关键字参数

3) 动态生成功能, 以下方法都支持动态生成, 也可以覆盖装饰器 @allure 的内容

  - allure.dynamic.feature
  - allure.dynamic.link
  - allure.dynamic.issue
  - allure.dynamic.testcase
  - allure.dynamic.story
  - allure.dynamic.title
  - allure.dynamic.description

4) 如果想对每个测试用例进行非常详细的步骤信息说明, 提高可读性, 可以使用以下两个方法:

  - allure.step : 对每个测试用例进行步骤说明
  - allure.attach : 输出内容

先看下

```
allure.attach(body, name=None, attachment_type=None, extension=None)
```

e) 定义, 参数如下

  - body: 要显示的内容 (附件)
  - name: 附件名字
  - attachment\_type: 附件类型, 是 allure.attachment\_type 里面的其中一种
  - extension: 附件的扩展名

也可以使用文件路径:

```
allure.attach.file  
(source, body, name=None, attachment_type=None, extension=None),  
source 为文件路径, 其他参数和上述一致
```