Table of Contents

目录结构

Python2和**Python3**的区别？

- 性能：Py3.0运行 pystone benchmark的速度比Py2.5慢30%。Guido认为Py3.0有极大的优化空间，在字符串和整型操作上可以取得很好的优化效果
- 编码：Py3.X源码文件默认使用utf-8编码
- 语法：

```
1）去除了<>，全部改用!=
2）去除``，全部改用repr()
3）关键词加入as 和with，还有True,False,None
4）整型除法返回浮点数，要得到整型结果，请使用//
5）加入nonlocal语句。使用noclocal x可以直接指派外围（非全局）变量
6）去除print语句，加入print()函数实现相同的功能。同样的还有 exec语句，已经改为exec()函数
7）改变了顺序操作符的行为，例如x<y，当x和y类型不匹配时抛出TypeError而不是返回随即的 bool值
8）输入函数改变了，删除了raw_input，用input代替：
   2.X:guess = int(raw_input('Enter an integer : ')) # 读取键盘输入的方法
   3.X:guess = int(input('Enter an integer : '))
9）去除元组参数解包。不能def(a, (b, c)):pass这样定义函数了
10）新式的8进制字变量，相应地修改了oct()函数。
11）增加了 2进制字面量和bin()函数
12）扩展的可迭代解包。在Py3.X 里，a, b, *rest = seq和 *rest, a = seq都是合法的，只要求两点：rest是list
对象和seq是可迭代的。
13）新的super()，可以不再给super()传参数，
14）新的metaclass语法：
    class Foo(*bases, **kwds):
      pass
15）支持class decorator。用法与函数decorator一样：
```

- 字符串和字符串：

```
1）现在字符串只有str一种类型，但它跟2.x版本的unicode几乎一样。
2）关于字节串，请参阅“数据类型”的第2条目
```

- 数据类型：

```
1）Py3.X去除了long类型，现在只有一种整型——int，但它的行为就像2.X版本的long
2）新增了bytes类型，对应于2.X版本的八位串，定义一个bytes字面量的方法如下：
   str对象和bytes对象可以使用.encode() (str -> bytes) or .decode() (bytes -> str)方法相互转化。
 3）dict的.keys()、.items 和.values()方法返回迭代器，而之前的iterkeys()等函数都被废弃。同时去掉的还有 dict.has_key()
，用 in替代它吧
```

- 面向对象：

```
1）引入抽象基类（Abstraact Base Classes，ABCs）。
2）容器类和迭代器类被ABCs化。
3）迭代器的next()方法改名为__next__()，并增加内置函数next()，用以调用迭代器的__next__()方法
4）增加了@abstractmethod和 @abstractproperty两个 decorator，编写抽象方法（属性）更加方便。
```

- 异常：

```
1）所以异常都从 BaseException继承，并删除了StardardError
2）去除了异常类的序列行为和.message属性
3）用 raise Exception(args)代替 raise Exception, args语法
4）捕获异常的语法改变，引入了as关键字来标识异常实例
5）异常链，因为__context__在3.0a1版本中没有实现
```

- 模块变动

1）移除了cPickle模块，可以使用pickle模块代替。最终我们将会有一个透明高效的模块。
2）移除了imageop模块
3）移除了 audiodev, Bastion, bsddb185, exceptions, linuxaudiodev, md5, MimeWriter, mimify, popen2, rexec, sets, sha, stringold, strop, sunaudiodev, timing和xmllib模块
4）移除了bsddb模块(单独发布，可以从http://www.jcea.es/programacion/pybsddb.htm获取)
5）移除了new模块
6）os.tmpnam()和os.tmpfile()函数被移动到tmpfile模块下
7）tokenize模块现在使用bytes工作。主要的入口点不再是generate_tokens，而是 tokenize.tokenize()

- 其他：

1）xrange() 改名为range()，要想使用range()获得一个list，必须显式调用：
```
>>> list(range(10)) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
2）bytes对象不能hash，也不支持 b.lower()、b.strip()和b.split()方法，但对于后两者可以使用 b.strip(b’ \n\t\r \f’)和b.split(b’ ‘)来达到相同目的
3）zip()、map()和filter()都返回迭代器。而apply()、 callable()、coerce()、 execfile()、reduce()和reload ()函数都被去除了现在可以使用hasattr()来替换 callable(). hasattr()的语法如：hasattr(string, '__name__')
4）string.letters和相关的.lowercase和.uppercase被去除，请改用string.ascii_letters 等
5）如果x < y的不能比较，抛出TypeError异常。2.x版本是返回伪随机布尔值的
6）__getslice__系列成员被废弃。a[i:j]根据上下文转换为a.__getitem__(slice(I, j))或 __setitem__和 __delitem__调用
7）file类被废弃

Python高级变量类型列表详解

列表的定义

- List(列表)是Python中使用最频繁的数据类型，在其他语言中通常叫做数组
- 列表是有序的集合
- 定义列表使用 **[ ]** 定义，数据之间使用 , 分割

```
name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]
```

- 列表的索引从 **0** 开始：
  - 索引就是数据在列表中的位置编号，索引又可以被称为下标
- 【注意】: 从列表中取值时,如果超出索引范围,程序会产生异常

```
IndexError: list index out of range
```

列表的常用操作

增加

- 列表名.insert(index, 数据)：在指定位置插入数据(位置前有空元素会补位)

```
# 往列表name_list下标为0的地方插入数据
In [3]: name_list.insert(0, "Sasuke")

In [4]: name_list
Out[4]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu']

# 现有的列表下标是0-4，如果我们要在下标是6的地方插入数据，那个会自动插入到下标为5的地方，也就是
# 插入到最后
In [5]: name_list.insert(6, "Tom")

In [6]: name_list
Out[6]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu', 'Tom']
```

- 列表名.append(数据)：在列表的末尾追加数据(最常用的方法)

```
In [7]: name_list.append("Python")

In [8]: name_list
Out[8]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu', 'Tom', 'Python']
```

- 列表.extend(Iterable)：将可迭代对象中的元素追加到列表。

```
# 有两个列表 a 和 b a.extend(b) 会将b中的元素追加到列表a中
In [10]: a = [11, 22, 33]

In [11]: b = [44, 55, 66]

In [12]: a.extend(b)
```

```
In [13]: a
Out[13]: [11, 22, 33, 44, 55, 66]
# 有列表c  和 字符串 d  c.extend(d)  会将字符串d中的每个字符拆开作为元素插入到列表c
In [14]: c = ['j', 'a', 'v', 'a']

In [15]: d = "python"

In [16]: c.extend(d)

In [17]: c
Out[17]: ['j', 'a', 'v', 'a', 'p', 'y', 't', 'h', 'o', 'n']
```

取值和修改

- 取值：列表名[index] ：根据下标来取值

```
In [19]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]

In [20]: name_list[0]
Out[20]: 'zhangsan'

In [21]: name_list[3]
Out[21]: 'zhaoliu'
```

- 修改：列表名[index] = 数据：修改指定索引的数据

```
# 将列表中下标为0的值 zhangsan  修改为  Sasuke
In [22]: name_list[0] = "Sasuke"

In [23]: name_list
Out[23]: ['Sasuke', 'lisi', 'wangwu', 'zhaoliu']
```

删除

- del 列表名[index]：删除指定索引的数据

```
In [25]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]
# 删除索引是 1 的数据
In [26]: del name_list[1]

In [27]: name_list
Out[27]: ['zhangsan', 'wangwu', 'zhaoliu']
```

- 列表名.remove(数据)：删除第一个出现的指定数据

```
In [30]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu", "lisi"]
# 删除 第一次出现的 lisi 的数据
In [31]: name_list.remove("lisi")

In [32]: name_list
Out[32]: ['zhangsan', 'wangwu', 'zhaoliu', 'lisi']
```

- 列表名.pop()：删除末尾的数据,返回值: 返回被删除的元素

```
In [33]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]
# 删除最后一个元素  zhaoliu  并将元素 zhaoliu 返回
```

```
In [34]: name_list.pop()
Out[34]: 'zhaoliu'

In [35]: name_list
Out[35]: ['zhangsan', 'lisi', 'wangwu']
```

- 列表名.pop(index)：删除指定索引的数据，返回被删除的元素

```
In [36]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]
# 删除索引为 1 的数据 lisi
In [37]: name_list.pop(1)
Out[37]: 'lisi'

In [38]: name_list
Out[38]: ['zhangsan', 'wangwu', 'zhaoliu']
```

- 列表名.clear()：清空整个列表的元素

```
In [40]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]

In [41]: name_list.clear()

In [42]: name_list
Out[42]: []
```

排序

- 列表名.sort()：升序排序 从小到大

```
In [43]: a = [33, 44, 22, 66, 11]

In [44]: a.sort()

In [45]: a
Out[45]: [11, 22, 33, 44, 66]
```

- 列表名.sort(reverse=True)：降序排序 从大到小

```
In [46]: a = [33, 44, 22, 66, 11]

In [47]: a.sort(reverse=True)

In [48]: a
Out[48]: [66, 44, 33, 22, 11]
```

- 列表名.reverse()：列表逆序、反转

```
In [50]: a = [11, 22, 33, 44, 55]

In [51]: a.reverse()

In [52]: a
Out[52]: [55, 44, 33, 22, 11]
```

统计相关

- len(列表名)：得到列表的长度

```
In [53]: a = [11, 22, 33, 44, 55]

In [54]: len(a)
Out[54]: 5
```

- 列表名.count(数据)：数据在列表中出现的次数

```
In [56]: a = [11, 22, 11, 33, 11]

In [57]: a.count(11)
Out[57]: 3
```

- 列表名.index(数据)：数据在列表中首次出现时的索引，没有查到会报错。

```
In [59]: a = [11, 22, 33, 44, 22]

In [60]: a.index(22)
Out[60]: 1
```

- if 数据 in 列表： 判断列表中是否包含某元素。

```
a = [11, 22, 33, 44 ,55]
if 33 in a:
    print("找到了....")
```

循环遍历

- 使用while循环：

```
a = [11, 22, 33, 44, 55]

i = 0

while i < len(a):
    print(a[i])
    i += 1
```

- 使用for循环：

```
a = [11, 22, 33, 44, 55]

for i in a:
    print(i)
```

Python中的set集合

set集合，在Python中的书写方式的{}，集合与之前列表、元组类似，可以存储多个数据，但是这些数据是不重复的

集合对象还支持union(联合), intersection(交), difference(差)和sysmmetric_difference(对称差集)等数学运算.

快速去除列表中的重复元素

```
In [4]: a = [11,22,33,33,44,22,55]

In [5]: set(a)
Out[5]: {11, 22, 33, 44, 55}
```

交集：共有的部分

```
In [7]: a = {11,22,33,44,55}

In [8]: b = {22,44,55,66,77}

In [9]: a&b
Out[9]: {22, 44, 55}
```

并集：总共的部分

```
In [11]: a = {11,22,33,44,55}

In [12]: b = {22,44,55,66,77}

In [13]: a | b
Out[13]: {11, 22, 33, 44, 55, 66, 77}
```

差集：另一个集合中没有的部分

```
In [15]: a = {11,22,33,44,55}

In [16]: b = {22,44,55,66,77}

In [17]: b - a
Out[17]: {66, 77}
```

对称差集(在a或b中，但不会同时出现在二者中)

```
In [19]: a = {11,22,33,44,55}

In [20]: b = {22,44,55,66,77}

In [21]: a ^ b
Out[21]: {11, 33, 66, 77}
```

1、python中是如何实现list和tuple之间的转换的？

```
可以使用内置函数直接转换
list---->tuple     tuple(list)
tuple---->list     list(tuple)
```

2、解释一下python中pass语句的作用？

```
pass语句不会执行任何操作，一般作为占位符或者创建占位程序
```

3、简述你对input()函数的理解。

```
在python3中，input()获取用户输入，不论用户输入的是什么，获取到的都是字符串类型的。
在python2中有 raw_input()和input()，raw_input()和python3中的input()作用是一样的，input()输入的是什么数据类型的，获取到的就是什么数据类型的。
```

4、写出一段python代码实现删除一个list里面的重复元素

- 方法一使用map的fromkeys来自动过滤重复值

```
a = [1,2,3,4,2,1,4,5,6,3,6,7,8]
b = {}
b = b.fromkeys(a)
print(b)
a = list(b.keys())
print(a)
```

- 利用set()，set定义是定义集合的，无序，不重复。

```
a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
# set是非重复的，无序集合。可以用list来的排队对set进行排序，list()转换为列表，a.sort来排序
a=list(set(a))
print(a)
```

5、在linux中find和grep的区别？

```
Linux系统中grep命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹 配的行打印出来。grep全称是Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。
而linux下的find
功能：在目录结构中搜索文件，并执行指定的操作。此命令提供了相当多的查找条件，功能很强大。
语法：find 起始目录 寻找条件 操作
说明：find命令从指定的起始目录开始，递归地搜索其各个子目录，查找满足寻找条件的文件并对之采取相关的操作。
简单点说说，grep是查找匹配条件的行，find是搜索匹配条件的文件。
```

1、re模块中的search()和match()的区别 ？

match()函数只检测RE是不是在string的开始位置匹配，
search()会扫描整个string查找匹配；
也就是说match()只有在0位置匹配成功的话才有返回，
如果不是开始位置匹配成功的话，match()就返回none。

2、什么是lambda函数？ 有什么好处？

lambda 函数是一个可以接收任意多个参数(包括可选参数)并且返回单个表达式值的函数
1、lambda 函数比较轻便，即用即仍，很适合需要完成一项功能，但是此功能只在此一处使用，连名字都很随意的情况下；
2、匿名函数，一般用来给 filter， map 这样的函数式编程服务;
3、作为回调函数，传递给某些应用，比如消息处理

3、Python的参数传递是值传递还是引用传递？

Python的参数传递有：位置参数、默认参数、可变参数、关键字参数
函数的传值到底是值传递还是引用传递，要分情况：
不可变参数用值传递：
像整数和字符串这样的不可变对象，是通过拷贝进行传递的，因为你无论如何都不可能在原处改变不可变对象
可变参数是引用传递的：
比如像列表，字典这样的对象是通过引用传递、和C语言里面的用指针传递数组很相似，可变对象能在函数内部改变。

4、谈谈你对面向对象的理解

面向对象是相对于面向过程而言的。面向过程语言是一种基于功能分析的、以算法为中心的程序设计方法；而面向对象是一种基于结构分析的、以数据为中心的程序设计思想。在面向对象语言中有一个有很重要东西，叫做类。
面向对象有三大特性：封装、继承、多态。

5、交换两个变量的值

```
a,b = b,a
```

read、readline和readlines的区别

- read:读取整个文件
- readline：读取下一行，使用生成器方法
- readlines：读取整个文件到一个迭代器以供我们遍历

python中的is和==

- is是地址的比较
- ==是值的比较

Python中的作用域

- Python中，一个变量的作用域总是由在代码中被赋值的地方所决定
- 当Python遇到一个变量的话它会按照这的顺序进行搜索
- 本地作用域(Local)--->当前作用域被嵌入的本地作用域(Enclosing locals)--->全局/模块作用域(Global)--->内置作用域(Built-in)

单例模式的实现

- 使用 new方法

```
class Singleton(object):
    def __new__(cls, *args, **kw):
        if not hasattr(cls, '_instance'):
            orig = super(Singleton, cls)
            cls._instance = orig.__new__(cls, *args, **kw)
        return cls._instance

class MyClass(Singleton):
    a = 1
```

- 共享属性

  创建实例时把所有实例的dict指向一个字典，这样它们就具有相同的属性和方法

```
class Borg(object):
    _state = {}
    def __new__(cls, *args, **kw):
        ob = super(Borg, cls).__new__(cls, *args, **kw)
        ob.__dict__ = cls._state
        return ob

class MyClass2(Borg):
    a = 1
```

- 装饰器版本

```
def singleton(cls, *args, **kw):
    instances = {}
    def getinstance():
        if cls not in instances:
            instances[cls] = cls(*args, **kw)
        return instances[cls]
```

```
    return getinstance

@singleton
class MyClass:
  ...
```

- import方法

  作为python的模块是天然的单例模式

```
# mysingleton.py
class My_Singleton(object):
    def foo(self):
        pass

my_singleton = My_Singleton()

# to use
from mysingleton import my_singleton

my_singleton.foo()
```

字典推导式：

```
d = {key: value for (key, value) in iterable}
```

Python：赋值、浅拷贝和深拷贝

一、赋值

> 在python中，对象的赋值就是简单的对象引用，这点和C++不同，如下所示：
>
> ```
> a = [1,2,"hello",['python', 'C++']]
> b = a
> ```
>
> 在上述情况下，a和b是一样的，他们指向同一片内存，b不过是a的别名，是引用。
>
> 我们可以使用b is a 去判断，返回True，表明他们地址相同，内容相同，也可以使用id()函数来查看两个列表的地址是否相同。
>
> 赋值操作(包括对象作为参数、返回值)不会开辟新的内存空间，它只是复制了对象的引用。
>
> 也就是说除了b这个名字之外，没有其他的内存开销。
>
> 修改了a，也就影响了b;同理，修改了b，也就影响了a

二、浅拷贝(shallow copy)

> 浅拷贝会创建新对象，其内容是原对象的引用。
>
> 浅拷贝有三种形式:切片操作、工厂函数、**copy**模块中的**copy**函数
>
> 比如上述的列表a
>
> 切片操作：b = a[:] 或者 b = [x for x in a]
>
> 工厂函数：b = list(a)
>
> copy函数：b = copy.copy(a)
>
> 浅拷贝产生的列表b不再是列表a了，使用is判断可以发现他们不是同一个对象，使用id查看，他们也不指向同一片内存空间。但是当我们使用id(x) for x in a 和 id(x) for x in b来查看a和b中元素的地址时，可以看到二者包含的元素的地址是相同的。
>
> 在这种情况下，列表a和b是不同的对象，修改列表b理论上不会影响到列表a。
>
> 但是要注意的是，浅拷贝之所以称之为浅拷贝，是它仅仅只拷贝了一层，在列表a中有一个嵌套的list，如果我们修改了它，情况就不一样了。
>
> 比如：a[3].append('java')。查看列表b，会发现列表b也发生了变化，这是因为，我们修改了嵌套的list，修改外层元素，会修改它的引用，让它们指向别的位置，修改嵌套列表中的元素，列表的地址并未发生变化，指向的都是用一个位置。

三、深拷贝(deep copy)

> 深拷贝只有一种形式，copy模块中的deepcopy()函数
>
> 深拷贝和浅拷贝对应，深拷贝拷贝了对象的所有元素，包括多层嵌套的元素。因此，它的时间和空间开销要高。

> 同样的对列表a，如果使用 b = copy.deepcopy(a)，再修改列表b将不会影响到列表a，即使嵌套的列表具有更深的层次，也不会产生任何影响，因为深拷贝拷贝出来的对象根本就是一个全新的对象，不再与原来的对象有任何的关联。

四、拷贝的注意点

- 对于非容器类型，如数字、字符，以及其他的“原子”类型，没有拷贝一说，产生的都是原对象的引用。
- 如果元组变量值包含原子类型对象，即使采用了深拷贝，也只能得到浅拷贝。

hasattr() getattr() setattr() 函数使用详解

hasattr(object, name)函数

- 判断一个对象里面是否有name属性或者name方法，返回bool值，有name属性(方法)返回True，否则返回False。
- 注意：**name**要使用引号括起来

```
class function_demo(object):
    name = 'demo'
    def run(self):
        return "hello function"

functiondemo = function_demo()
res = hasattr(functiondemo, 'name')  #判断对象是否有name属性，True

res = hasattr(functiondemo, "run") #判断对象是否有run方法，True

res = hasattr(functiondemo, "age") #判断对象是否有age属性，Falsw
print(res)
```

getattr(object, name[,default]) 函数

- 获取对象object的属性或者方法，如果存在则打印出来，如果不存在，打印默认值，默认值可选。
- 注意：如果返回的是对象的方法，则打印结果是：方法的内存地址，如果需要运行这个方法，可以在后面添加括号**()**

```
class function_demo(object):
    name = 'demo'
    def run(self):
        return "hello function"


functiondemo = function_demo()
getattr(functiondemo, 'name') #获取name属性，存在就打印出来--- demo

getattr(functiondemo, "run") #获取run方法，存在打印出 方法的内存地址---<bound method function_demo.run of <__main__.func
tion_demo object at 0x10244f320>>

getattr(functiondemo, "age") #获取不存在的属性，报错如下：
Traceback (most recent call last):
  File "/Users/liuhuiling/Desktop/MT_code/OpAPIDemo/conf/OPCommUtil.py", line 39, in <module>
    res = getattr(functiondemo, "age")
AttributeError: 'function_demo' object has no attribute 'age'

getattr(functiondemo, "age", 18)  #获取不存在的属性，返回一个默认值
```

setattr(object,name,values)函数：

- 给对象的属性赋值，若属性不存在，先创建再赋值

```
class function_demo(object):
    name = 'demo'
    def run(self):
        return "hello function"
```

```
functiondemo = function_demo()
res = hasattr(functiondemo, 'age')  # 判断age属性是否存在，False
print(res)

setattr(functiondemo, 'age', 18 )  #对age属性进行赋值，无返回值

res1 = hasattr(functiondemo, 'age') #再次判断属性是否存在，True
print(res1)
```

综合使用：

```
class function_demo(object):
    name = 'demo'
    def run(self):
        return "hello function"


functiondemo = function_demo()
res = hasattr(functiondemo, 'addr') # 先判断是否存在
if res:
    addr = getattr(functiondemo, 'addr')
    print(addr)
else:
    addr = getattr(functiondemo, 'addr', setattr(functiondemo, 'addr', '北京首都'))
    #addr = getattr(functiondemo, 'addr', '美国纽约')
    print(addr)
```

Python中的os模块

官方解释：

> This module provides a portable way of using operating system dependent functionality.
>
> 提供一种方便的使用操作系统函数的方法。

os模块常用方法：

- os.remove()删除文件
- os.rename()重命名文件
- os.walk()生成目录树下的所有文件名
- os.chdir()改变目录
- os.mkdir/makedirs创建目录/多层目录
- os.rmdir/removedirs删除目录/多层目录
- os.listdir()列出指定目录的文件
- os.getcwd()取得当前工作目录
- os.chmod()改变目录权限
- os.path.basename()去掉目录路径，返回文件名
- os.path.dirname()去掉文件名，返回目录路径
- os.path.join()将分离的各部分组合成一个路径名
- os.path.split()返回（dirname(),basename())元组
- os.path.splitext()(返回filename,extension)元组
- os.path.getatime\ctime\mtime分别返回最近访问、创建、修改时间
- os.path.getsize()返回文件大小
- os.path.exists()是否存在
- os.path.isabs()是否为绝对路径
- os.path.isdir()是否为目录
- os.path.isfile()是否为文件

Python的sys模块

官方解释：

> This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
>
> 提供访问由解释器使用或维护的变量和在与解释器交互使用到的函数。

sys模块常用方法：

- sys.argv 命令行参数List，第一个元素是程序本身路径
- sys.modules.keys() 返回所有已经导入的模块列表
- sys.exc_info() 获取当前正在处理的异常类,exc_type、exc_value、exc_traceback当前处理的异常详细信息
- sys.exit(n) 退出程序，正常退出时exit(0)
- sys.hexversion 获取Python解释程序的版本值，16进制格式如：0x020403F0
- sys.version 获取Python解释程序的版本信息
- sys.maxint 最大的Int值
- sys.maxunicode 最大的Unicode值
- sys.modules 返回系统导入的模块字段，key是模块名，value是模块
- sys.path 返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值
- sys.platform 返回操作系统平台名称
- sys.stdout 标准输出
- sys.stdin 标准输入
- sys.stderr 错误输出
- sys.exc_clear() 用来清除当前线程所出现的当前的或最近的错误信息
- sys.exec_prefix 返回平台独立的python文件安装的位置
- sys.byteorder 本地字节规则的指示器，big-endian平台的值是'big',little-endian平台的值是'little'
- sys.copyright 记录python版权相关的东西
- sys.api_version 解释器的C的API版本
- sys.version_info 元组则提供一个更简单的方法来使你的程序具备Python版本要求功能

函数装饰器有什么作用？请列举出至少三个并举出一些实例？

> 装饰器本质上是一个python函数，它可以在让其他函数在不需要做任何代码的变动的前提下增加额外的功能。 装饰器的返回值也是一个函数的对象，它经常用于有切面需求的场景。 比如：插入日志、性能测试、事务处理、缓存、权限的校验等场景 有了装饰器就可以抽离出大量的与函数功能本身无关的雷同代码并发并继续使用。

迭代器和生成器函数的区别？

- 迭代器是一个更抽象的概念，任何对象，如果它的类有next方法和iter方法返回自己本身，对于string、list、dict、tuple等这类容器对象，使用for循环遍历是很方便的。在后台for语句对容器对象调用iter()函数，iter()是python的内置函数。iter()会返回一个定义了next()方法的迭代器对象，它在容器中逐个访问容器内元素，next()也是python的内置函数。在没有后续元素时，next()会抛出一个StopIteration异常。
- 生成器（Generator）是创建迭代器的简单而强大的工具。它们写起来就像是正规的函数，只是在需要返回数据的时候使用 yield 语句。每次 next()被调用时，生成器会返回它脱离的位置（它记忆语句最后一次执行的位置和所有的数据值）
- 区别：生成器能做到迭代器能做的所有事,而且因为自动创建了**iter**()和 next()方法,生成器显得特别简洁,而且生成器也是高效的，使用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法,当发生器终结时,还会自动抛出 StopIteration 异常

python中闭包的理解？

> 在函数内部再定义一个函数，并且这个函数用到了外边函数的变量，那么将这个函数以及用到的一些变量称之为闭包。返回的是内部函数的引用。

Python中**yield**的用法？

> yield简单说来就是一个生成器，这样函数它记住上次返简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 回时在函数体中的位置。对生成器第二次（或n 次）调用跳转至该函 次）调用跳转至该函 数。

copy和**deepcopy**的区别？

```
copy.copy()浅拷贝，只拷贝父对象，不会拷贝对象的内部的子对象。
copy.deepcopy()深拷贝，拷贝对象及其子对象。
In [2]: import copy

In [3]: a = [1,2,3,4,['a','b']] # 原始对象

In [4]: b = a # 赋值 传对象的引用

In [5]: c = copy.copy(a)  # 对象拷贝  浅拷贝

In [6]: d = copy.deepcopy(a)  # 对象拷贝 深拷贝

In [7]: a.append(5)  # 修改对象a

In [8]: a[4].append('c')  # 修改对象a中的['a','b']列表对象

In [9]: print('a=', a)
a= [1, 2, 3, 4, ['a', 'b', 'c'], 5]

In [10]: print('b=', b)
b= [1, 2, 3, 4, ['a', 'b', 'c'], 5]

In [11]: print('c=', c)
c= [1, 2, 3, 4, ['a', 'b', 'c']]

In [12]: print('d=', d)
d= [1, 2, 3, 4, ['a', 'b']]
```

一句话的阶乘函数

```
reduce(lambda x,y: x*y, range(1,n+1))
```

介绍一下**except**的用法和作用

```
except: #捕获所有异常
except: <异常名>: #捕获指定异常
except:<异常名 1，异常名 2):捕获异常 1 或者异常 2
except:<异常名>,<数据>:捕获指定异常及其附加的数据
except:<异常名 1,异常名 2>:<数据>:捕获异常名 1 或者异常名 2,及附加的数据
```

@(decorator)的作用是什么？

> 装饰器是一个很著名的设计模式，经常被用于有切面需求的场景，较为经典的有插入日志、性能测试、事务处理等。装饰器是解决这类问题的绝佳设计，有了装饰器，我们就可以抽离出大量函数中与函数功能本身无关的雷同代码并继续重用。概括的讲，装饰器的作用就是为已经存在的对象添加额外的功能

io密集型和**cpu**密集型区别

- io 密集型：系统运作，大部分的状况是 CPU 在等 I/O (硬盘/内存) 的读/写
- Cpu 密集型：大部份时间用来做计算、逻辑判断等 CPU 动作的程序称之 CPU密集型

什么是线程安全？

> 线程安全是在多线程的环境下，能够保证多个线程同时执行时程序依旧运行正确, 而且要保证对于共享的数据可以由多个线程存取，但是同一时刻只能有一个线程进行存取。多线程环境下解决资源竞争问题的办法是加锁来保证存取操作的唯一性。

Python中**yield**的用法？

> yield 简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 简单说来就是一个生成器，这样函数它记住上次返 回时在函数体中的位置。对生成器第 二次（或n 次）调用跳转至该函 次）调用跳转至该函数。

谈谈你对**GIL**锁对**Python**多线程的影响？

> GIL的全称是Global Interpreter Lock(全局解释器锁)，来源是python设计之初的考虑，为了数据安全所做的决定。每个CPU在同一时间只能执行一个线程（在单核CPU下的多线程其实都只是并发，不是并行，并发和并行从宏观上来讲都是同时处理多路请求的概念。但并发和并行又有区别，并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔内发生。） 在Python多线程下，每个线程的执行方式： 1、获取GIL
> 2、执行代码直到sleep或者是python虚拟机将其挂起。 3、释放GIL
> 可见，某个线程想要执行，必须先拿到GIL，我们可以把GIL看作是“通行证”，并且在一个python进程中，GIL只有一个。拿不到通行证的线程，就不允许进入CPU执行。 在Python2.x里，GIL的释放逻辑是当前线程遇见IO操作或者ticks计数达到100（ticks可以看作是Python自身的一个计数器，专门做用于GIL，每次释放后归零，这个计数可以通过 sys.setcheckinterval 来调整），进行释放。而每次释放GIL锁，线程进行锁竞争、切换线程，会消耗资源。并且由于GIL锁存在，python里一个进程永远只能同时执行一个线程(拿到GIL的线程才能执行)。 IO密集型代码(文件处理、网络爬虫等)，多线程能够有效提升效率(单线程下有IO操作会进行IO等待，造成不必要的时间浪费，而开启多线程能在线程A等待时，自动切换到线程B，可以不浪费CPU的资源，从而能提升程序执行效率)，所以多线程对IO密集型代码比较友好。

什么是阻塞？什么是非阻塞？

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

递归函数终止的条件？

> 递归的终止条件一般定义在递归函数内部，在递归调用前要做一个条件判断，根据判断的结果选择是继续调用自身，还是return;返回终止递归。 终止的条件： 1.判断递归的次数是否达到某一限定值 2.判断运算的结果是否达到某个范围等，根据设计的目的来选择

软连接和硬链接的区别？

> 软连接类似windows的快捷方式，当删除源文件，那么软链接失效。硬链接可以理解为源文件的一个别名。多个别名所代表的是同一个文件。当rm一个文件的时候，那么此文件的硬链接数减1，当硬链接数为0的时候，文件删除。

单例模式的应用场景有哪些？

> 单例模式应用的场景一般发现在以下条件下：　　（1）资源共享的情况下，避免由于资源操作时导致的性能或损耗等。如日志文件，应用配置。　　（2）控制资源的情况下，方便资源之间的互相通信。如线程池等。 1.网站的计数器 2.应用配置 3.多线程池 4.数据库配置，数据库连接池 5.应用程序的日志应用....

Python中是如何进行内存管理的？

- 垃圾回收：python不像C++，Java等语言一样，他们可以不用事先声明变量类型而直接对变量进行赋值。对Python语言来讲，对象的类型和内存都是在运行时确定的。这也是为什么我们称Python语言为动态类型的原因（这里我们把动态类型可以简单的归结为对变量内存地址的分配是在运行时自动判断变量类型并对变量进行赋值）。
- 引用计数：Python采用了类似Windows内核对象一样的方式来对内存进行管理。每一个对象，都维护这一个对指向该对对象的引用的计数。当变量被绑定在一个对象上的时候，该变量的引用计数就是1，(还有另外一些情况也会导致变量引用计数的增加),系统会自动维护这些标签，并定时扫描，当某标签的引用计数变为0的时候，该对就会被回收。
- 内存池机制Python的内存机制以金字塔行，1，2层主要有操作系统进行操作
  - 第0层是C中的malloc，free等内存分配和释放函数进行操作
  - 第1层和第2层是内存池，有Python的接口函数PyMem_Malloc函数实现，当对象小于256K时有该层直接分配内存
  - 第3层是最上层，也就是我们对Python对象的直接操作
- 在 C 中如果频繁的调用malloc 与 free 时,是会产生性能问题的.再加上频繁的分配与释放小块的内存会产生内存碎片. Python 在这里主要干的工作有:
  - 如果请求分配的内存在1~256字节之间就使用自己的内存管理系统,否则直接使用 malloc
  - 这里还是会调用 malloc 分配内存,但每次会分配一块大小为256k的大块内存.
  - 经由内存池登记的内存到最后还是会回收到内存池,并不会调用 C 的 free 释放掉.以便下次使用.对于简单的Python对象，例如数值、字符串，元组（tuple不允许被更改)采用的是复制的方式(深拷贝?)，也就是说当将另一个变量B赋值给变量A时，虽然A和B的内存空间仍然相同，但当A的值发生变化时，会重新给A分配空间，A和B的地址变得不再相同

如何用Python来进行查询和替换一个文本字符串？

- 可以使用re模块中的sub()函数或者subn()函数来进行查询和替换
- 格式：sub(replacement, string[,count=0])（replacement是被替换成的文本，string是需要被替换的文本，count是一个可选参数，指最大被替换的数量）

Python中常见的设计模式有哪些？

- 1、创建型模式
  - 前面讲过，社会化的分工越来越细，自热在软件设计方面也是如此，因此对象的创建和对象的使用分开也就成为了必然趋势，因为对象的创建会消耗掉系统的很多资源，所以单独对对象的创建进行研究，从而能够高效的创建对象就是创建型模式要探讨的问题。这里有6个具体的创建型模式可供研究，他们分别是:

```
简单工厂模式（Simple Factory）；
工厂方法模式（Factory Method）；
抽象工厂模式（Abstract Factory）；
创建者模式（Builder）；
原型模式（Prototype）；
单例模式（Singleton）。
说明：严格来说，简单工厂模式不是GoF总结出来的23种设计模式之一。
```

- 2、结构型模式

- 在解决了对象的创建问题之后，对象的组成以及对象之间的依赖关系就成了开发人员关注的焦点，因为如何设计对象的结构、继承和依赖关系会影响到后续程序的维护性、代码的健壮性、耦合性等，对象结构的设计很容易体现出设计人员水平的高低，这里有7个具体的结构型模式可供研究，他们分别是：

```
外观模式（Facade）；
适配器模式（Adapter）；
代理模式（Proxy）；
装饰模式（Decorator）；
桥模式（Bridge）；
组合模式（Composite）；
享元模式（Flyweight）
```

- 3、行为型模式
  - 在对象节后和对象的创建问题都解决了之后，就剩下对象的行为问题了，如果对象的行为设计的好，那么对象的行为就会更清晰，他们之间的协作效率就会提高，这里有11个具体的行为型模式可供研究，他们分别是：

```
模板方法模式（Template Method）；
观察者模式（Observer）；
状态模式（State）；
策略模式（Strategy）；
职责链模式（Chain of Responsibility）；
命令模式（Command）；
访问者模式（Visitor）；
调停者模式（Mediator）；
备忘录模式（Memento）；
迭代器模式（Iterator）；
解释器模式（Interpreter）
```

Python里面如何拷贝一个对象？**(**赋值、深拷贝、浅拷贝的区别**)**

- 赋值(=),就是创建了对象的一个新的引用，修改其中任意一个变量都会影响到另一个。
- 浅拷贝：创建一个新的对象，但它包含的是对原始对象中包含项的引用(如果引用的方式修改其中一个对象，另外一个也会修改改变)(1/完全切片的方法。2.工厂函数，如list()。3.copy模块的copy()函数)
- 深拷贝：创建一个新的对象，并且递归复制它所包含的对象(修改其中一个，另外一个不会改变)(copy模块的deepcopy()函数)

Python中数组有哪些类型？字典可以是有序的吗？

```
List Tuple Dictionary
from collections import OrderedDict
dic=OrderedDict()#声明有序字典
```

find和**grep**的区别？

- **find**通常用来在特定的目录下搜索符合条件的文件，也可以用来搜索特定用户属主的文件。
- **grep**命令是一种强大的文本搜索工具，grep搜索内容串可以是正则表达式，允许对文本文件进行模式查找。如果找到匹配模式，grep打印包含模式的所有行。

什么是阻塞？什么是非阻塞？

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

Linux重定向命令有哪些？有什么区别？

- 重定向>

  Linux允许将命令执行结果重定向到一个文件，本应显示在终端上的内容保存到指定文件中。如：ls > test.txt (test.txt 如果不存在，则创建，存在则覆盖其内容)

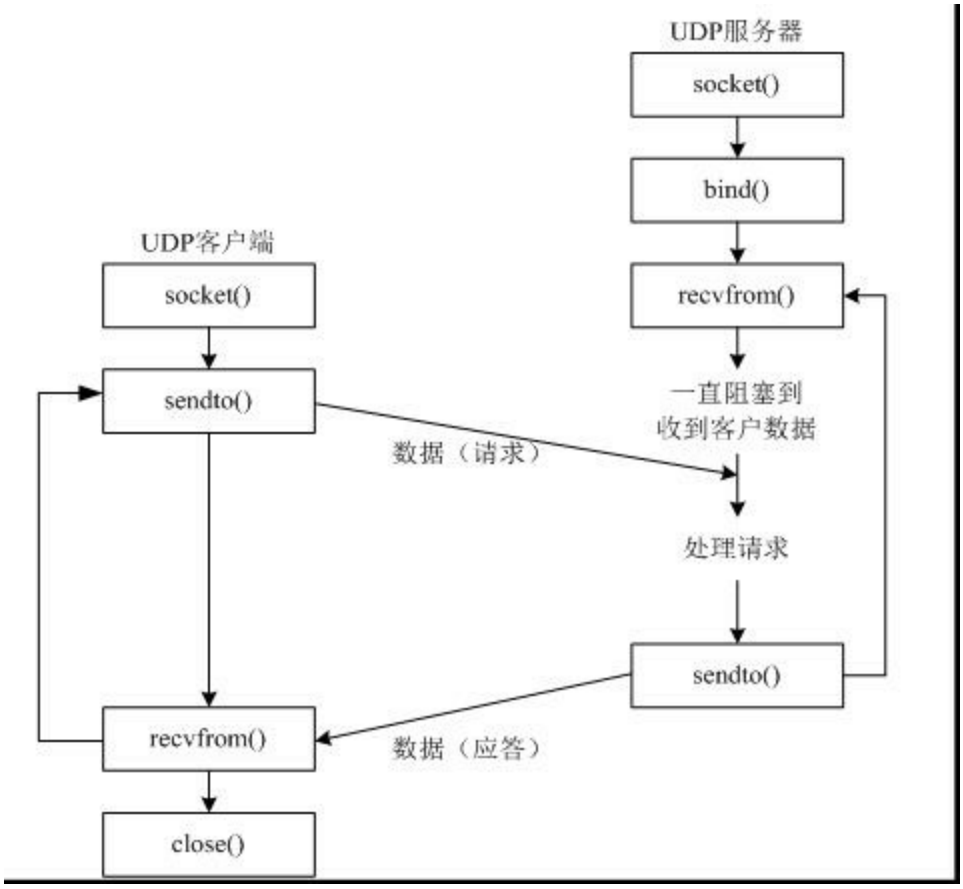
- 重定向>>

  >>这个是将输出内容追加到目标文件中。如果文件不存在，就创建文件；如果文件存在,则将新的内容追加到那个文件的末尾，该文件中的原有内容不受影响。

软连接和硬链接的区别？

> 软连接类似windows的快捷方式，当删除源文件，那么软链接失效。硬链接可以理解为源文件的一个别名。多个别名所代表的是同一个文件。当rm一个文件的时候，那么此文件的硬链接数减1，当硬链接数为0的时候，文件删除。

UDP总结



使用**udp**发送/接收数据步骤：

- 创建客户端套接字
- 发送/接收数据
- 关闭套接字

```
import socket


def main():
    # 1、创建udp套接字
    # socket.AF_INET  表示IPv4协议  AF_INET6 表示IPv6协议
    # socket.SOCK_DGRAM  数据报套接字，只要用于udp协议
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # 2、准备接收方的地址
    # 元组类型  ip是字符串类型   端口号是整型
    dest_addr = ('192.168.113.111', 8888)

    # 要发送的数据
    send_data = "我是要发送的数据"
    # 3、发送数据
    udp_socket.sendto(send_data.encode("utf-8"), dest_addr)

    # 4、等待接收方发送的数据  如果没有收到数据则会阻塞等待，直到收到数据
    # 接收到的数据是一个元组   (接收到的数据，发送方的ip和端口)
```

```
    # 1024  表示本次接收的最大字节数
    recv_data, addr = udp_socket.recvfrom(1024)

    # 5、关闭套接字
    udp_socket.close()

if __name__ == '__main__':
    main()
```

编码的转换

> str -->bytes: encode编码
>
> bytes--> str: decode()解码

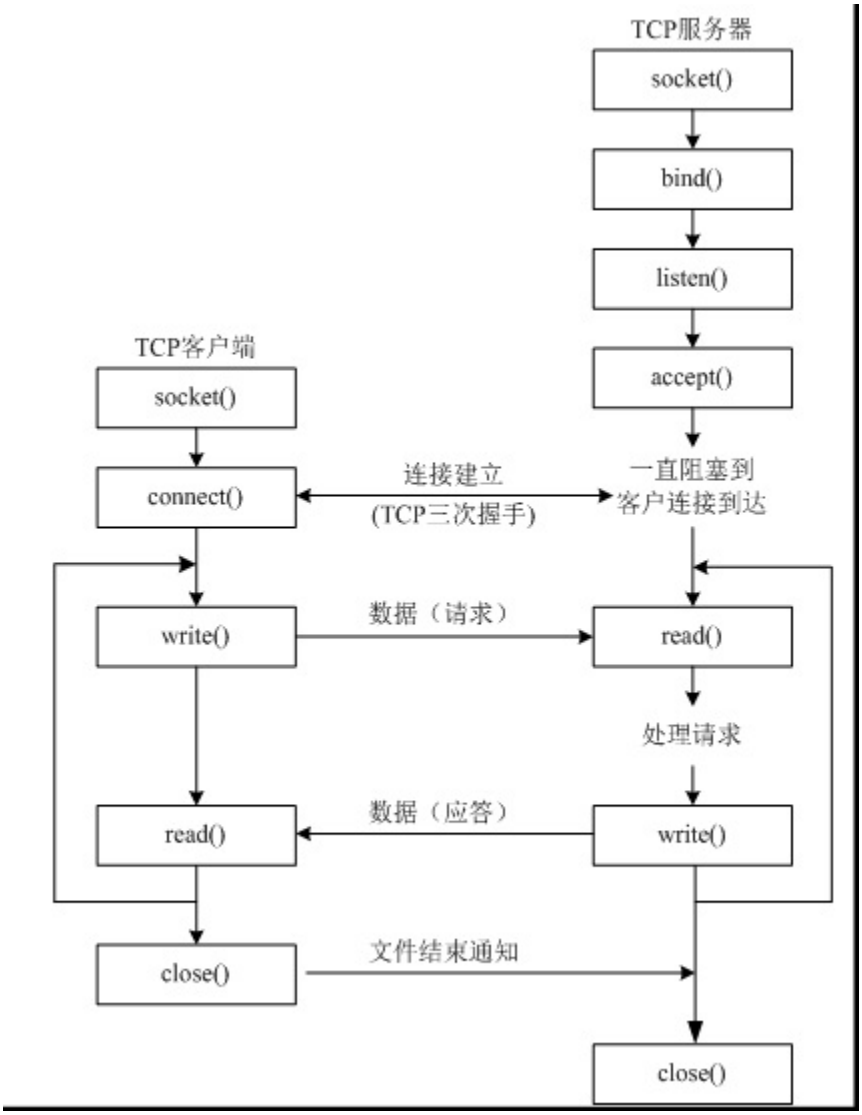
UDP绑定端口号：

- 创建socket套接字
- 绑定端口号
- 接收/发送数据
- 关闭套接字

```
import socket


def main():
    # 1、创建udp套接字
    # socket.AF_INET  表示IPv4协议  AF_INET6 表示IPv6协议
    # socket.SOCK_DGRAM  数据报套接字，只要用于udp协议
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # 2、绑定端口
    # 元组类型  ip一般不写 表示本机的任何的一个ip
    local_addr = ('', 7777)
    udp_socket.bind(local_addr)

    # 3、准备接收方的地址
    # 元组类型  ip是字符串类型   端口号是整型
    dest_addr = ('192.168.113.111', 8888)

    # 要发送的数据
    send_data = "我是要发送的数据"
    # 4、发送数据
    udp_socket.sendto(send_data.encode("utf-8"), dest_addr)

    # 5、等待接收方发送的数据  如果没有收到数据则会阻塞等待，直到收到数据
    # 接收到的数据是一个元组   (接收到的数据，发送方的ip和端口)
    # 1024  表示本次接收的最大字节数
    recv_data, addr = udp_socket.recvfrom(1024)

    # 6、关闭套接字
    udp_socket.close()

if __name__ == '__main__':
    main()
```

注意点：绑定端口要在发送数据之前进行绑定。

TCP总结



TCP客户端的创建流程：

- 创建TCP的socket套接字
- 连接服务器
- 发送数据给服务器端
- 接收服务器端发送来的消息
- 关闭套接字

```
import socket


def main():
    # 1、创建客户端的socket
    # socket.AF_INET  表示IPv4协议  AF_INET6 表示IPv6协议
    # socket.SOCK_STREAM  流式套接字，只要用于TCP协议
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    # 2、构建目标地址
    server_ip = input("请输入服务器端的IP地址：")
    server_port = int(input("请输入服务器端的端口号："))

    # 3、连接服务器
    # 参数：元组类型    ip是字符串类型    端口号是整型
    client_socket.connect((server_ip, server_port))

    # 要发送给服务器端的数据
    send_data = "我是要发送给服务器端的数据"

    # 4、发送数据
    client_socket.send(send_data.encode("gbk"))

    # 5、接收服务器端恢复的消息，  没有消息会阻塞
    # 1024表示接收的最大字节数
    recv_date= client_socket.recv(1024)
    print("接收到的数据是：", recv_date.decode('gbk'))

    # 6、关闭套接字
    client_socket.close()

if __name__ == '__main__':
    main()
```

TCP服务器端的创建流程

- 创建TCP服务端的socket
- bing绑定ip地址和端口号
- listen使套接字变为被动套接字
- accept取出一个客户端连接，用于服务
- recv/send接收和发送消息
- 关闭套接字

```
import socket


def main():
    # 1、创建tcp服务端的socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # 2、绑定
    server_socket.bind(('', 8888))

    # 3、listen使套接字变为被动套接字
    server_socket.listen(128)

    # 4、如果有新的客户端来链接服务器，那么就产生一个新的套接字专门为这个客户端服务
    # client_socket用来为这个客户端服务
    # tcp_server_socket就可以省下来专门等待其他新客户端的链接
    client_socket, client_addr = server_socket.accept()

    # 5、接收客户端发来的消息
    recv_data = client_socket.recv(1024)
    print("接收到客户端%s的数据：%s" % (str(client_addr), recv_data.decode('gbk')))

    # 6、回复数据给客户端
    client_socket.send("收到消息".encode('gbk'))

    # 7、关闭套接字
    client_socket.close()
    server_socket.close()
```

```
if __name__ == '__main__':
    main()
```

注意点：

- tcp服务器一般都需要绑定，否则客户端找不到服务器
- tcp客户端一般不绑定，因为是主动链接服务器，所以只要确定好服务器的ip、port等信息就好，本地客户端可以随机
- tcp服务器中通过listen可以将socket创建出来的主动套接字变为被动的，这是做tcp服务器时必须要做的
- 当客户端需要链接服务器时，就需要使用connect进行链接，udp是不需要链接的而是直接发送，但是tcp必须先链接，只有链接成功才能通信
- 当一个tcp客户端连接服务器时，服务器端会有1个新的套接字，这个套接字用来标记这个客户端，单独为这个客户端服务
- listen后的套接字是被动套接字，用来接收新的客户端的连接请求的，而accept返回的新套接字是标识这个新客户端的
- 关闭listen后的套接字意味着被动套接字关闭了，会导致新的客户端不能够链接服务器，但是之前已经链接成功的客户端正常通信。
- 关闭accept返回的套接字意味着这个客户端已经服务完毕
- 当客户端的套接字调用close后，服务器端会recv解阻塞，并且返回的长度为0，因此服务器可以通过返回数据的长度来区别客户端是否已经下线；同理 当服务器断开tcp连接的时候 客户端同样也会收到0字节数据。

另外**TCP**和**UDP**的优缺点已经区别在网络编程面试题**-01**这一篇中。

进程总结

什么是进程？

> 程序运行在操作系统上的一个实例，就称之为进程。进程需要相应的系统资源：内存、时间片、pid

创建进程

- 首先要导入multiprocessing中的Process
- 创建一个Process对象
  - 创建Process对象时，可以传递参数

    > p = Process(target=XXX, args=(元组,) , kwargs={key:value})
    >
    > target = XXX 指定的任务函数,不用加()
    >
    > args=(元组,) , kwargs={key:value} 给任务函数传递的参数
- 使用start()启动进程
- 结束进程

Process语法结构：

> Process([group [, target [, name [, args [, kwargs]]]]])

- target：如果传递了函数的引用，可以让这个子进程就执行函数中的代码
- args：给target指定的函数传递的参数，以元组的形式进行传递
- kwargs：给target指定的函数传递参数，以字典的形式进行传递
- name：给进程设定一个名字，可以省略
- group：指定进程组，大多数情况下用不到

> Process创建的实例对象的常用方法有：

- start()：启动子进程实例(创建子进程)
- is_alive()：判断进程子进程是否还在活着
- join(timeout)：是否等待子进程执行结束，或者等待多少秒
- terminate()：不管任务是否完成，立即终止子进程

> Process创建的实例对象的常用属性：

- name：当前进程的别名，默认为Process-N,N为从1开始递增的整数
- pid：当前进程的pid(进程号)

给子进程指定函数传递参数**Demo**：

```
import os
from multiprocessing import Process

import time
```

```
def pro_func(name, age, **kwargs):
    for i in range(5):
        print("子进程正在运行中,name=%s, age=%d, pid=%d" %(name, age, os.getpid()))
        print(kwargs)
        time.sleep(0.2)


if __name__ == '__main__':
    # 创建Process对象
    p = Process(target=pro_func, args=('小明',18), kwargs={'m': 20})
    # 启动进程
    p.start()
    time.sleep(1)
    # 1秒钟之后，立刻结束子进程
    p.terminate()
    p.join()
```

注意：进程间不共享全局变量。

进程之间的通信-Queue

> 在初始化Queue()对象时，(例如q=Queue()，若在括号中没有指定最大可接受的消息数量，或数量为负值时，那么就代表可接受的消息数量没有上限-直到内存的尽头)

- Queue.qsize()：返回当前队列包含的消息数量。
- Queue.empty()：如果队列为空，返回True,反之False。
- Queue.full()：如果队列满了，返回True，反之False。
- Queue.get([block[,timeout]])：获取队列中的一条消息，然后将其从队列中移除，block默认值为True。
  - 如果block使用默认值，且没有设置timeout（单位秒），消息列队如果为空，此时程序将被阻塞（停在读取状态），直到从消息列队读到消息为止，如果设置了timeout，则会等待timeout秒，若还没读取到任何消息，则抛出"Queue.Empty"异常；
  - 如果block值为False，消息列队如果为空，则会立刻抛出"Queue.Empty"异常；
    - Queue.get_nowait()：相当Queue.get(False)；
    - Queue.put(item,[block[, timeout]])：将item消息写入队列，block默认值为True；
  - 如果block使用默认值，且没有设置timeout（单位秒），消息列队如果已经没有空间可写入，此时程序将被阻塞（停在写入状态），直到从消息列队腾出空间为止，如果设置了timeout，则会等待timeout秒，若还没空间，则抛出"Queue.Full"异常；
  - 如果block值为False，消息列队如果没有空间可写入，则会立刻抛出"Queue.Full"异常；
    - Queue.put_nowait(item)：相当Queue.put(item, False)；

进程间通信Demo：

```
from multiprocessing import Process, Queue
import os, time, random

# 写数据进程执行的代码:
def write(q):
    for value in ['A', 'B', 'C']:
        print('Put %s to queue...' % value)
        q.put(value)
        time.sleep(random.random())

# 读数据进程执行的代码:
def read(q):
    while True:
        if not q.empty():
```

```
            value = q.get(True)
            print('Get %s from queue.' % value)
            time.sleep(random.random())
        else:
            break

if __name__=='__main__':
    # 父进程创建Queue，并传给各个子进程：
    q = Queue()
    pw = Process(target=write, args=(q,))
    pr = Process(target=read, args=(q,))
    # 启动子进程pw，写入：
    pw.start()
    # 等待pw结束：
    pw.join()
    # 启动子进程pr，读取：
    pr.start()
    pr.join()
    # pr进程里是死循环，无法等待其结束，只能强行终止：
    print('')
    print('所有数据都写入并且读完')
```

进程池**Pool**

```
# -*- coding:utf-8 -*-
from multiprocessing import Pool
import os, time, random

def worker(msg):
    t_start = time.time()
    print("%s开始执行,进程号为%d" % (msg,os.getpid()))
    # random.random()随机生成0~1之间的浮点数
    time.sleep(random.random()*2)
    t_stop = time.time()
    print(msg,"执行完毕，耗时%0.2f" % (t_stop-t_start))

po = Pool(3)  # 定义一个进程池，最大进程数3
for i in range(0,10):
    # Pool().apply_async(要调用的目标,(传递给目标的参数元祖,))
    # 每次循环将会用空闲出来的子进程去调用目标
    po.apply_async(worker,(i,))

print("----start----")
po.close()  # 关闭进程池，关闭后po不再接收新的请求
po.join()  # 等待po中所有子进程执行完成，必须放在close语句之后
print("-----end-----")
```

multiprocessing.Pool常用函数解析：

- apply_async(func[, args[, kwds]]) ：使用非阻塞方式调用func（并行执行，堵塞方式必须等待上一个进程退出才能执行下一个进程），args为传递给func的参数列表，kwds为传递给func的关键字参数列表；
- close()：关闭Pool，使其不再接受新的任务；
- terminate()：不管任务是否完成，立即终止；
- join()：主进程阻塞，等待子进程的退出， 必须在close或terminate之后使用；

进程池中使用**Queue**

> 如果要使用Pool创建进程，就需要使用multiprocessing.Manager()中的Queue()，而不是multiprocessing.Queue()，否则会得到一条如下的错误信息：

> RuntimeError: Queue objects should only be shared between processes through inheritance.

```
from multiprocessing import Manager,Pool
import os,time,random

def reader(q):
    print("reader启动(%s),父进程为(%s)" % (os.getpid(), os.getppid()))
    for i in range(q.qsize()):
        print("reader从Queue获取到消息：%s" % q.get(True))

def writer(q):
    print("writer启动(%s),父进程为(%s)" % (os.getpid(), os.getppid()))
    for i in "itcast":
        q.put(i)

if __name__=="__main__":
    print("(%s) start" % os.getpid())
    q = Manager().Queue()  # 使用Manager中的Queue
    po = Pool()
    po.apply_async(writer, (q,))

    time.sleep(1)  # 先让上面的任务向Queue存入数据，然后再让下面的任务开始从中取数据

    po.apply_async(reader, (q,))
    po.close()
    po.join()
    print("(%s) End" % os.getpid())
```

简述**TCP**和**UDP**的区别以及优缺点

- UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息。
  - 优点：UDP速度快、操作简单、要求系统资源较少，由于通讯不需要连接，可以实现广播发送
  - 缺点：UDP传送数据前并不与对方建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收，也不重复发送，不可靠。
- TCP是面向连接的通讯协议，通过三次握手建立连接，通讯完成时四次挥手
  - 优点：TCP在数据传递时，有确认、窗口、重传、阻塞等控制机制，能保证数据正确性，较为可靠。
  - 缺点：TCP相对于UDP速度慢一点，要求系统资源较多。

简述浏览器通过**WSGI**请求动态资源的过程

- 发送http请求动态资源给web服务器
- web服务器收到请求后通过WSGI调用一个属性给应用程序框架
- 应用程序框架通过引用WSGI调用web服务器的方法，设置返回的状态和头信息。
- 调用后返回，此时web服务器保存了刚刚设置的信息
- 应用程序框架查询数据库，生成动态页面的body的信息
- 把生成的body信息返回给web服务器
- web服务器吧数据返回给浏览器

描述用浏览器访问**www.baidu.com**的过程

- 先要解析出baidu.com对应的ip地址
  - 要先使用arp获取默认网关的mac地址
  - 组织数据发送给默认网关(ip还是dns服务器的ip，但是mac地址是默认网关的mac地址)
  - 默认网关拥有转发数据的能力，把数据转发给路由器
  - 路由器根据自己的路由协议，来选择一个合适的较快的路径转发数据给目的网关
  - 目的网关(dns服务器所在的网关)，把数据转发给dns服务器
  - dns服务器查询解析出baidu.com对应的ip地址，并原路返回请求这个域名的client
- 得到了baidu.com对应的ip地址之后，会发送tcp的3次握手，进行连接
  - 使用http协议发送请求数据给web服务器
  - web服务器收到数据请求之后，通过查询自己的服务器得到相应的结果，原路返回给浏览器。
  - 浏览器接收到数据之后通过浏览器自己的渲染功能来显示这个网页。
  - 浏览器关闭tcp连接，即4次挥手结束，完成整个访问过程

AJAX试什么？如何使用**AJAX**？

- **ajax**(异步的javascript 和xml) 能够刷新局部网页数据而不是重新加载整个网页。
- 第一步，创建xmlhttprequest对象，var xmlhttp =new XMLHttpRequest（);XMLHttpRequest对象用来和服务器交换数据。
- 第二步，使用xmlhttprequest对象的open()和send()方法发送资源请求给服务器。
- 第三步，使用xmlhttprequest对象的responseText或responseXML属性获得服务器的响应。
- 第四步，onreadystatechange函数，当发送请求到服务器，我们想要服务器响应执行一些功能就需要使用onreadystatechange函数，每次xmlhttprequest对象的readyState发生改变都会触发onreadystatechange函数。

Post和**Get**请求的区别**?**

- **GET**请求，请求的数据会附加在URL之后，以?分割URL和传输数据，多个参数用&连接。URL的编码格式采用的是ASCII编码，而不是uniclde，即是说所有的非ASCII字符都要编码之后再传输。
- **POST**请求：POST请求会把请求的数据放置在HTTP请求包的包体中。上面的item=bandsaw就是实际的传输数据。
- 因此，**GET**请求的数据会暴露在地址栏中，而**POST**请求则不会。
- 传输数据的大小：
  - 在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对于GET，特定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会受到URL长度的限制。
  - 对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST提交数据大小进行限制，Apache、IIS都有各自的配置。
- 安全性：
  - **POST**的安全性比GET的高。这里的安全是指真正的安全，而不同于上面GET提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，通过GET请求，用户名和密码都会暴露再URL上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET请求提交的数据还可能会造成Cross-site request frogery攻击。

死锁：

- 原因:
  - 1.竞争资源
  - 2.程序推进顺序不当
- 必要条件：
  - 1.互斥条件
  - 2.请求和保持条件
  - 3.不剥夺条件
  - 环路等待条件
- 处理死锁基本方法：
  - 预防死锁(摒弃除1以外的条件)
  - 避免死锁(银行家算法)
  - 检测死锁(资源分配图)
  - 解除死锁:
    - 1.剥夺死锁
    - 2.撤销进程

cookie 和**session** 的区别？

> **1**、cookie数据存放在客户的浏览器上，session数据放在服务器上。
>
> **2**、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗考虑到安全应当使用session。
>
> **3**、session会在一定时间内保存在服务器上。当访问增多，会比较占用服务器的性能考虑到减轻服务器性能方面，应当使用COOKIE。
>
> **4**、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。
>
> **5**、建议： 将登陆信息等重要信息存放为SESSION 其他信息如果需要保留，可以放在COOKIE中

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

说一下**mysql**数据库存储的原理？

储存过程是一个可编程的函数，它在数据库中创建并保存。它可以有SQL语句和一些特殊的控制结构组成。当希望在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟。它允许控制数据的访问方式。存储过程通常有以下优点：
1、存储过程能实现较快的执行速度
2、存储过程允许标准组件是编程。
3、存储过程可以用流程控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。
4、存储过程可被作为一种安全机制来充分利用。
5、存储过程能够减少网络流量

事务的特性？

1、原子性(Atomicity)：事务中的全部操作在数据库中是不可分割的，要么全部完成，要么均不执行。
2、一致性(Consistency)：几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致。
3、隔离性(Isolation)：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的。
4、持久性(Durability)：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障

数据库索引

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用B_TREE。B_TREE索引加速了数据访问，因为存储引擎不会再去扫描整张表得到需要的数据；相反，它从根节点开始，根节点保存了子节点的指针，存储引擎会根据指针快速寻找数据。

数据库怎么优化查询效率？

1、储存引擎选择：如果数据表需要事务处理，应该考虑使用InnoDB，因为它完全符合ACID特性。如果不需要事务处理，使用默认存储引擎MyISAM是比较明智的
2、分表分库，主从。
3、对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引
4、应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描
5、应尽量避免在 where 子句中使用 != 或 <> 操作符，否则将引擎放弃使用索引而进行全表扫描
6、应尽量避免在 where 子句中使用 or 来连接条件，如果一个字段有索引，一个字段没有索引，将导致引擎放弃使用索引而进行全表扫描
7、Update 语句，如果只更改1、2个字段，不要Update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志
8、对于多张大数据量（这里几百条就算大了）的表JOIN，要先分页再JOIN，否则逻辑读会很高，性能很差。

redis 和 **mysql** 的区别

- readis 是内存数据库， 数据保存在内存中， 速度快。
- mysql 是关系型数据库， 持久化存储， 存放在磁盘里面， 功能强大。 检索的话， 会涉及到一定的 IO， 数据访问也就慢。

redis受攻击怎么办？

- 主从
- 持久化存储
- Redis不以root账户启动
- 设置复杂的密码
- 不允许key方式登录

数据库优化方案

- 优化索引、SQL 语句、分析慢查询
- 设计表的时候严格根据数据库的设计范式来设计数据库
- 使用缓存，把经常访问到的数据而且不需要经常变化的数据放在缓存中，能节约磁盘IO
- 优化硬件；采用SSD，使用磁盘队列技术(RAID0,RAID1,RDID5)等；
- 采用MySQL 内部自带的表分区技术，把数据分层不同的文件，能够提高磁盘的读取效率
- 垂直分表；把一些不经常读的数据放在一张表里，节约磁盘I/O
- 主从分离读写；采用主从复制把数据库的读操作和写入操作分离开来
- 分库分表分机器（数据量特别大），主要的的原理就是数据路由
- 选择合适的表引擎，参数上的优化
- 进行架构级别的缓存，静态化和分布式
- 不采用全文索引
- 采用更快的存储方式，例如NoSql存储经常访问的数据

Mysql集群的优缺点

- 优点：
  - 99.999%的高可用性
  - 快速的自动失效切换
  - 灵活的分布式体系结构，没有单点故障
  - 高吞吐量和低延迟
  - 可扩展性强，支持在线扩容
- 缺点：
  - 存在很多限制，比如：不支持外键
  - 部署、管理、配置很复杂
  - 占用磁盘空间大、内存大
  - 备份和恢复不方便
  - 重启的时候，数据节点将数据load到内存需要很长的时间

你用的**mysql**是哪个引擎，各引擎之间有什么区别？

主要 **MyISAM** 与 **InnoDB** 两个引擎，其主要区别如下：

- InnoDB 支持事务，MyISAM 不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一些列增删改中只要哪个出错还可以回滚还原，而 MyISAM就不可以了；
- MyISAM 适合查询以及插入为主的应用，InnoDB 适合频繁修改以及涉及到安全性较高的应用；
- InnoDB 支持外键，MyISAM 不支持；
- MyISAM 是默认引擎，InnoDB 需要指定；
- InnoDB 不支持 FULLTEXT 类型的索引；
- InnoDB 中不保存表的行数，如 select count() *from table* 时，*InnoDB*；需要扫描一遍整个表来计算有多少行，但是 *MyISAM* 只要简单的读出保存好的行数即可。注意的是，当 *count()*语句包含 where 条件时 MyISAM 也需要扫描整个表；
- 对于自增长的字段，InnoDB 中必须包含只有该字段的索引，但是在 MyISAM表中可以和其他字段一起建立联合索引；清空整个表时，InnoDB 是一行一行的删除，效率非常慢。MyISAM 则会重建表；
- InnoDB 支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like '%lee%'

MongoDB数据库

MongoDB 是一个面向文档的数据库系统。 使用 C++编写， 不支持SQL， 但有自己功能强大的查询语法。 MongoDB 使用 BSON 作为数据存储和传输的格式。 BSON 是一种类似 JSON 的二进制序列化文档， 支持嵌套对象和数组。 MongoDB 很像 MySQL， document 对应 MySQL 的 row， collection对应 MySQL 的 table 应用场景：

- 1.网站数据： mongo 非常适合实时的插入， 更新与查询， 并具备网站实时数据存储所需的复制及高度伸缩性。
- 2.缓存： 由于性能很高， mongo 也适合作为信息基础设施的缓存层。 在系统重启之后， 由 mongo 搭建的持久化缓存可以避免下层的数据源过载。
- 3.大尺寸、 低价值的数据： 使用传统的关系数据库存储一些数据时可能会比较贵， 在此之前， 很多程序员往往会选择传统的文件进行存储。
- 4.高伸缩性的场景： mongo 非常适合由数十或者数百台服务器组成的数据库。
- 5.用于对象及 JSON 数据的存储： mongo 的 BSON 数据格式非常适合文档格式化的存储及查询。
- 6.重要数据： mysql， 一般数据： mongodb， 临时数据： memcache
- 7.对于关系数据表而言， mongodb 是提供了一个更快速的视图view； 而对于 PHP 程序而言， mongodb 可以作为一个持久化的数组来使用， 并且这个持久化的数组还可以支持排序、 条件、 限制等功能。
- 8.将 mongodb 代替 mysql 的部分功能， 主要一个思考点就是：把 mongodb 当作 mysql 的一个 view（视图）， view 是将表数据整合成业务数据的关键。 比如说对原始数据进行报表， 那么就要先把原始数据统计后生成 view， 在对 view 进行查询和报表。

不适合的场景：

- a.高度事物性的系统： 例如银行或会计系统。 传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。
- b.传统的商业智能应用： 针对特定问题的 BI 数据库会对产生高度优化的查询方式。对于此类应用， 数据仓库可能是更合适的选择。
- c.需要 SQL 的问题
- d.重要数据， 关系数据

优点：

- 弱一致性（最终一致）， 更能保证用户的访问速度文档结构的存储方式， 能够更便捷的获取数内置 GridFS， 高效存储二进制大对象 (比如照片和视频)
- 支持复制集、 主备、 互为主备、 自动分片等特性
- 动态查询
- 全索引支持,扩展到内部对象和内嵌数组

缺点：

- 不支持事务
- MongoDB 占用空间过大
- 维护工具不够成熟

Mysql 和 redis 高可用性

- MySQL Replication 是 MySQL 官方提供的主从同步方案， 用于将一个 MySQL 实例的数据， 同步到另一个实例中。 Replication 为保证数据安全做了重要的保证， 也是现在运用最广的 MySQL 容灾方案。Replication 用两个或以上的实例搭建了 MySQL 主从复制集群， 提供单点写入， 多点读取的服务， 实现了读的 scale out.
- Sentinel 是 Redis 官方为集群提供的高可用解决方案。 在实际项目中可以使用 sentinel 去做 redis 自动故障转移， 减少人工介入的工作量。 另外 sentinel 也给客户端提供了监控消息的通知， 这样客户端就可根据消息类型去判断服务器的状态， 去做对应的适配操作。

下面是 **Sentinel** 主要功能列表：

- Monitoring： Sentinel 持续检查集群中的 master、 slave 状态，判断是否存活。
- Notification： 在发现某个 redis 实例死的情况下， Sentinel能通过 API 通知系统管理员或其他程序脚本。 Automatic failover： 如果一个 master 挂掉后， sentinel 立马启动故障转移， 把某个 slave 提升为 master。 其他的 slave 重新配置指向新 master。
- Configuration provider： 对于客户端来说 sentinel 通知是有效可信赖的。 客户端会连接 sentinel 去请求当前 master 的地址， 一旦发生故障 sentinel 会提供新地址给客户端。

Redis和MongoDB的优缺点

MongoDB 和 Redis 都是 NoSQL， 采用结构型数据存储。 二者在使用场景中， 存在一定的区别， 这也主要由于二者在内存映射的处理过程， 持久化的处理方法不同。 MongoDB 建议集群部署， 更多的考虑到集群方案，Redis 更偏重于进程顺序写入， 虽然支持集群， 也仅限于主-从模式. **Redis** 优点：

- 读写性能优异
- 支持数据持久化， 支持 AOF 和 RDB 两种持久化方式
- 支持主从复制， 主机会自动将数据同步到从机， 可以进行读写分离。
- 数据结构丰富： 除了支持 string 类型的 value 外还支持string、 hash、 set、 sortedset、 list 等数据结构。

Redis缺点：

- Redis 不具备自动容错和恢复功能， 主机从机的宕机都会导致前端部分读写请求失败， 需要等待机器重启或者手动切换前端的 IP才能恢复。
- 主机宕机， 宕机前有部分数据未能及时同步到从机， 切换 IP后还会引入数据不一致的问题， 降低了系统的可用性。
- Redis 的主从复制采用全量复制， 复制过程中主机会 fork 出一个子进程对内存做一份快照， 并将子进程的内存快照保存为文件发送给从机， 这一过程需要确保主机有足够多的空余内存。 若快照文件较大， 对集群的服务能力会产生较大的影响， 而且复制过程是在从机新加入集群或者从机和主机网络断开重连时都会进行， 也就是网络波动都会造成主机和从机间的一次全量的数据复制， 这对实际的系统运营造成了不小的麻烦。
- Redis 较难支持在线扩容， 在集群容量达到上限时在线扩容会变得很复杂。 为避免这一问题， 运维人员在系统上线时必须确保有足够的空间， 这对资源造成了很大的浪费。

MongoDB优点**:**

- 弱一致性（最终一致）， 更能保证用户的访问速度文档结构的存储方式， 能够更便捷的获取数内置 GridFS， 高效存储二进制大对象 (比如照片和视频)
- 支持复制集、 主备、 互为主备、 自动分片等特性
- 动态查询
- 全索引支持,扩展到内部对象和内嵌数组

MongoDB缺点：

- 不支持事务
- MongoDB 占用空间过大
- 维护工具不够成熟

数据库负载均衡

负载均衡集群是由一组相互独立的计算机系统构成，通过常规网络或专用网络进行连接， 由路由器衔接在一起，各节点相互协作、 共同负载、 均衡压力， 对客户端来说， 整个群集可以视为一台具有超高性能的独立服务器。

1、 实现原理

实现数据库的负载均衡技术，首先要有一个可以控制连接数据库的控制端。 在这里， 它截断了数据库和程序的直接连接， 由所有的程序来访问这个中间层， 然后再由中间层来访问数据库。 这样， 我们就可以具体控制访问某个数据库了，然后还可以根据数据库的当前负载采取有效的均衡策略， 来调整每次连接到哪个数据库。

2、 实现多据库数据同步

对于负载均衡，最重要的就是所有服务器的数据都是实时同步的。这是一个集群所必需的， 因为， 如果数不据实时、 不同步， 那么用户从一台服务器读出的数据， 就有别于从另一台服务器读出的数据， 这是不能允许的。 所以必须实现数据库的数据同步。 这样， 在查询的时候就可以有多个资源， 实现均衡。比较常用的方法是 Moebius for SQL Server 集群， Moebius for SQL Server 集群采用将核心程序驻留在每个机器的数据库中的办法， 这个核心程序称为 Moebius for SQL Server 中间件， 主要作用是监测数据库内数据的变化并将变化的数据同步到其他数据库中。 数据同步完成后客户端才会得到响应， 同步过程是并发完成的，所以同步到多个数据库和同步到一个数据库的时间基本相等； 另外同步的过程是在事务的环境下完成的， 保证了多份数据在任何时刻数据的一致性。 正因为 Moebius 中间件宿主在数据库中的创新， 让中间件不但能知道数据的变化， 而且知道引起数据变化的 SQL 语句，根据 SQL 语句的类型智能的采取不同的数据同步的策略以保证数据同步成本的最小化。数据条数很少， 数据内容也不大， 则直接同步数据数据条数很少， 但是里面包含大数据类型， 比如文本， 二进制数 据等， 则先对数据进行压缩然后再同步， 从而减少网络带宽的占用和传输所用的时间。数据条数很多，此时中间件会拿到造成数据变化的 SQL 语句， 然后对 SQL 语句进行解析， 分析其执行计划和执行成本， 并选择是同步数据还是同步 SQL 语句到其他的数据库中。此种情况应用在对表结构进行调整或者批量更改数据的时候非常有用。

3、 优缺点优点：

优点：

- (1) 扩展性强： 当系统要更高数据库处理速度时， 只要简单地增加数据库服务器就 可以得到扩展。
- (2) 可维护性： 当某节点发生故障时， 系统会自动检测故障并转移故障节点的应用， 保证数据库的持续工作。
- (3) 安全性： 因为数据会同步的多台服务器上， 可以实现数据集的冗余， 通过多份数据来保证安全性。 另外它成功地将数据库放到了内网之中， 更好地保护了数据库的安全性。
- (4) 易用性： 对应用来说完全透明， 集群暴露出来的就是一个IP

缺点：

- (1) 不能够按照 Web 服务器的处理能力分配负载。
- (2) 负载均衡器(控制端)故障， 会导致整个数据库系统瘫痪。

Mysql集群操作步骤

MySQL 群集需要有一组计算机， 每台计算机的角色可能是不一样的。 MySQL 群集中有三种节点： 管理节点、 数据节点和 SQL 节点。 群集中的某计算机可能是某一种节点，也可能是两种或三种节点的集合。这三种节点只是在逻辑上的划分，所以它们不一定和物理计算机是一一对应的关系。 管理节点（也可以称管理服务器） 主要负责管理数据节点和 SQL节点， 还有群集配置文件和群集日志文件。 它监控其他节点的工作状态， 能够启动、 关闭或重启某个节点。 其他节点从管理节点检索配置数据，当数据节点有新事件时就把事件信息发送给管理节点并写入群集日志。 数据节点用于存储数据。 SQL 节点跟一般的 MySQL 服务器是一样的， 我们可以通过它进行 SQL 操作。用的 MysqlServer 已经不能满足群集的要求，配置群集需要使用MySQLCluster。

MySQLCluster 的配置

首先找三台电脑， 或者是开三个虚拟机， 管理节点部署在一台机子上， 其他两台每台都部署一个数据节点和一个 SQL 节点。 这里以两台机子举例， 其中一台机器 A（IP 为 192.168.193.90） 部署管理节点、 数据节点和 SQL 节点， 另一台机器 B（IP 为 192.168.193.50）部署数据节点和 SQL 节点。 其实最好不要将管理节点跟数据节点部署到一台机子上，因为如果数据节点宕机会导致管理节点也不可用， 整个 MySQL 群集就都不可用了。 所以一个 MySQL 群集理想情况下至少有三台服务器， 将管理节点单独放到一台服务器上。 暂以两台举例， 只是为了说明三种节点的配置启动方法。将上面下载的安装包解压， 并改文件夹名为 mysql， 因为需要多次在命令行中操作， 所以名字改短后更容易输入。配置管理节点， 配置数据节点， 配置 SQL 节点启动管理节点， 启动数据节点， 启动 SQL 节点

测试 **MySQLCluster**： 我们需要测试三种情况：

- 1.在任一 SQL 节点对数据节点进行操作后， 各数据节点是否能够实现数据同步。 例如， 我们在机器 A 上新创建一个数据库 myDB， 然后再建一个表 student（新建表如下命令： createtable student (id int(2)) engine=ndbcluster） ， 插入若干数据， 接着我们到机器 B上查看是否能看到新的数据库 myDB 和新的表 student 以及插入数据。
- 2.当关闭任一数据节点后， 在所有 SQL 节点中进行操作是否不受其影响。 例如， 我们关闭机器 A 上的数据节点服务， 在两台主机上应该能够继续对数据库进行各种操作。
- 3.关闭某数据节点进行了数据库操作， 然后重新启动， 所有 SQL节点的操作是否正常。

怎样解决海量数据的存储和访问造成系统设计瓶颈的问题？

- 水平切分数据库： 可以降低单台机器的负载， 同时最大限度的降低了宕机造成的损失； 分库降低了单点机器的负载； 分表， 提高了数据操作的效率，
- 负载均衡策略： 可以降低单台机器的访问负载， 降低宕机的可能性；
- 集群方案：解决了数据库宕机带来的单点数据库不能访问的问题；
- 读写分离策略：最大限度了提高了应用中读取数据的速度和并发量；

redis 基本类型、 相关方法

Redis 支持五种数据类型： string（字符串） 、 hash（哈希）、list（列表） 、 set（集合） 及 zset(sorted set：有序集合)。

- 一、 **String**

  String 是 Redis 最为常用的一种数据类型， String 的数据结构为 key/value 类型， String 可以包含任何数据。常用命令: set,get,decr,incr,mget 等

- 二、 **Hash**

Hash 类型可以看成是一个 key/value 都是 String 的 Map 容器。 常用命令： hget,hset,hgetall 等。

- 三、 **List**

  List 用于存储一个有序的字符串列表， 常用的操作是向队列两端添加元素或者获得列表的某一片段。

  常用命令： lpush,rpush,lpop,rpop,lrange 等

- 四、 **Set**

  Set 可以理解为一组无序的字符集合， Set 中相同的元素是不会重复出现的， 相同的元素只保留一个。 常用命令： sadd,spop,smembers,sunion 等

- 五、 **Sorted Set**（有序集合）

  有序集合是在集合的基础上为每一个元素关联一个分数， Redis通过分数为集合中的成员进行排序。 常用命令： zadd,zrange,zrem,zcard 等

redis的事务

Redis 事务允许一组命令在单一步骤中执行。

事务有两个属性，说明如下：

事务是一个单独的隔离操作： 事务中的所有命令都会序列化、 按顺序地执行。 事务在执行的过程中， 不会被其他客户端发送来的命令请求所打断。

Redis 事务是原子的。 原子意味着要么所有的命令都执行， 要么都不执行；

一个事务从开始到执行会经历以下三个阶段：

- 开始事务
- 命令入队
- 执行事务

redis的使用场景有哪些？

- 1.取最新 N 个数据的操作
- 2.排行榜应用,取 TOP N 操作
- 3.需要精准设定过期时间的应用
- 4.计数器应用
- 5.uniq 操作,获取某段时间所有数据排重值
- 6.Pub/Sub 构建实时消息系统
- 7.构建队列系统
- 8.缓存

redis 默认端口， 默认过期时间， Value 最多可以容纳的数据长度？

- 默认端口： 6379
- 默认过期时间： 可以说永不过期， 一般情况下， 当配置中开启了超出最大内存限制就写磁盘的话， 那么没有设置过期时间的 key可能会被写到磁盘上。 假如没设置， 那么 REDIS 将使用 LRU 机制， 将内存中的老数据删除， 并写入新数据。
- Value 最多可以容纳的数据长度是： 512M。

sqlserver， MySQL ， Oracle http， redis， https 默认端口号？

- sqlserver： 1433
- MySQL： 3306
- Oracle ： 1521
- http： 80
- https： 443
- redis： 6379

redis 缓存命中率计算？

Redis 提供了 INFO 这个命令， 能够随时监控服务器的状态， 只用 telnet 到对应服务器的端口， 执行命令即可：telnet localhost 6379 info 在输出的信息里面有这几项和缓存的状态比较有关系：

- keyspace_hits:14414110
- keyspace_misses:3228654
- used_memory:433264648
- expired_keys:1333536
- evicted_keys:1547380

通过计算 hits 和 miss， 我们可以得到缓存的命中率： 14414110/ (14414110 + 3228654) = 81% ， 一个缓存失效机制， 和过期时间设计良好的系统， 命中率可以做到 95%以上

redis有多少个库？

Redis一个实例下有**16**个库

2017-12-20

什么是**css**初始化？有什么好处？

- CSS初始化是指重设浏览器的样式。不同的浏览器默认的样式可能不尽相同，如果没对CSS初始化往往会出现浏览器之间的页面差异。
- 好处：能够统一标签在各大主流浏览器中的默认样式，使得我们开发网页内容时更加方便简洁，同时减少CSS代码量，节约网页下载时间。

简述浮动的特征和清除浮动的方法？

- 浮动的特征：
  - 浮动元素有左浮动(float:left)和右浮动(float:right)两种
  - 浮动的元素会向左或向右浮动，碰到父元素边界、其他元素才停下来
  - 相邻浮动的块元素可以并在一行，超出父级宽度就换行
  - 浮动让行内元素或块元素转化为有浮动特性的行内块元素(此时不会有行内块元素间隙问题)
  - 父元素如果没有设置尺寸(一般是高度不设置)，父元素内整体浮动的子元素无法撑开父元素，父元素需要清除浮动
- 清除浮动的方法：
  - 父级上增加属性overflow：hidden
  - 在最后一个子元素的后面加一个空的div，给它样式属性 clear:both
  - 使用成熟的清浮动样式类，clearfix

```
.clearfix:after,.clearfix:before{ content: "";display: table;}
.clearfix:after{ clear:both;}
.clearfix{zoom:1;}
```

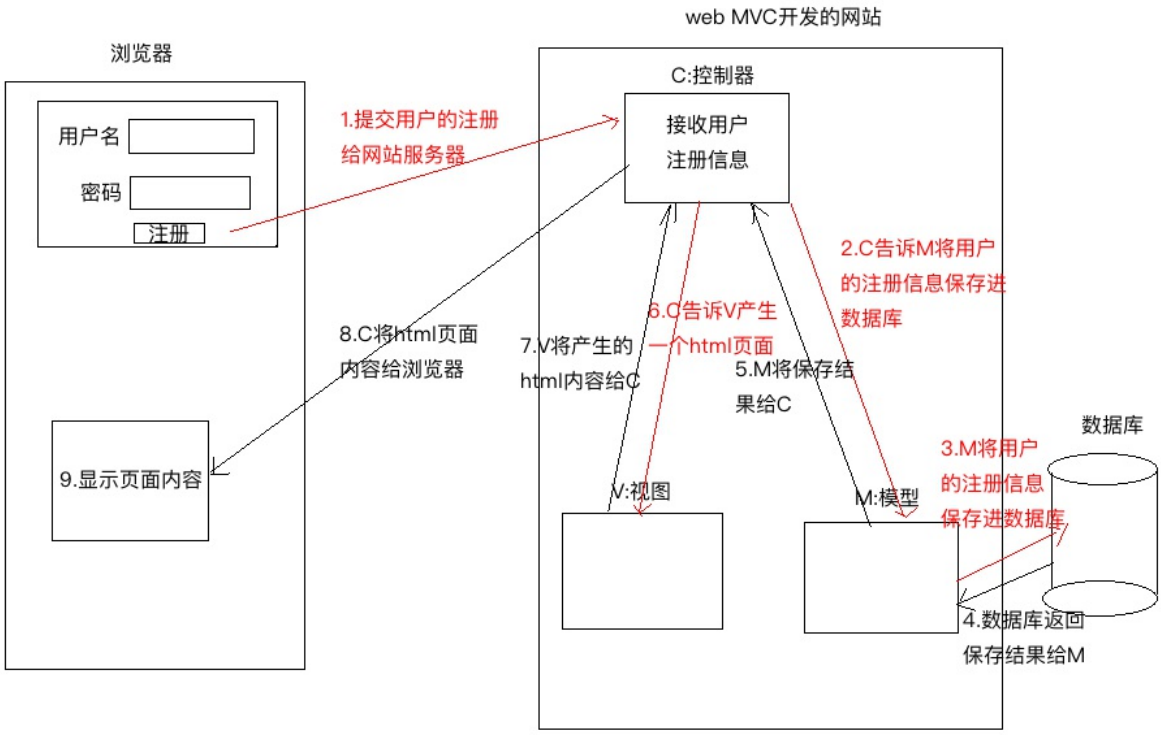
全局目录导航

MVC解读

M：Model，模型，和数据库进行交互

V：View，视图，负责产生Html页面

C：Controller，控制器，接收请求，进行处理，与M和V进行交互，返回应答。



1、 用户点击注按钮，将要注册的信息发送给网站服务器。

2、 Controller控制器接收到用户的注册信息，Controller会告诉Model层将用户的注册信息保存到数据库

3、 Model层将用户的注册信息保存到数据库

4、 数据保存之后将保存的结果返回给Model模型，

5、 Model层将保存的结果返回给Controller控制器。

6、 Controller控制器收到保存的结果之后，或告诉View视图，view视图产生一个html页面。

7、 View将产生的Html页面的内容给了Controller控制器。

8、 Controller将Html页面的内容返回给浏览器。

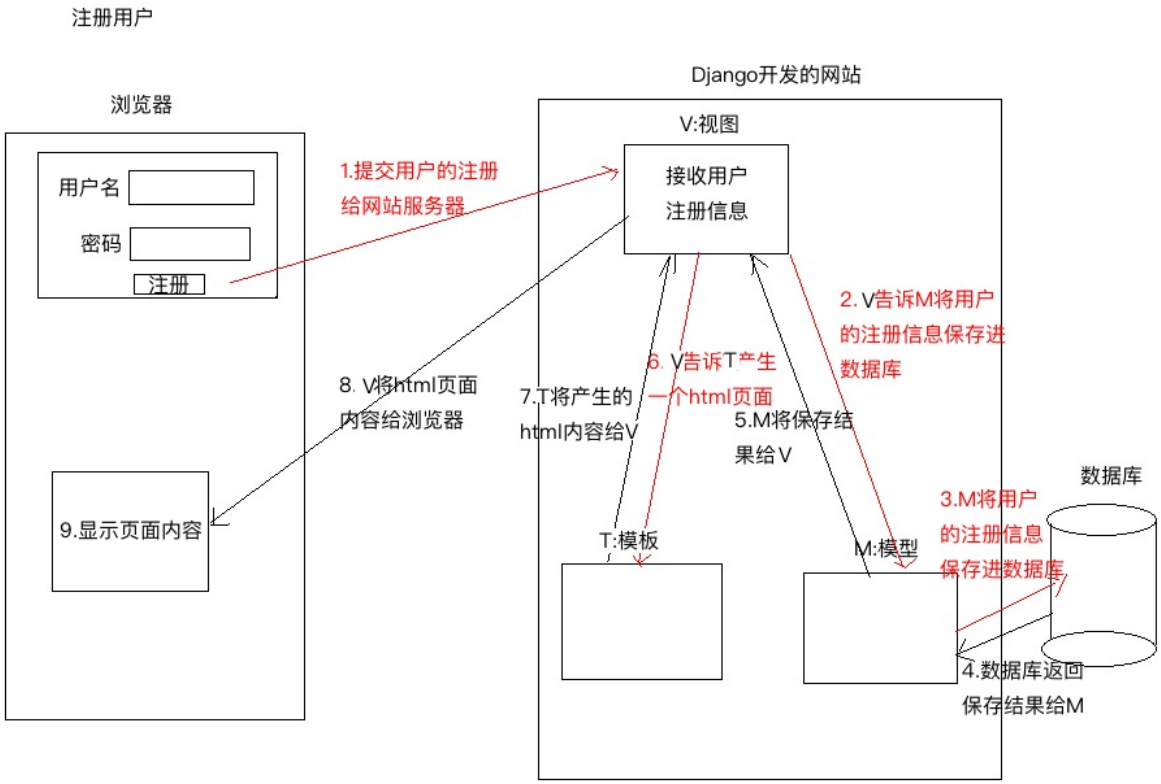
9、 浏览器接受到服务器Controller返回的Html页面进行解析展示。

MVT解读

M：Model，模型，和MVC中的M功能相同，和数据库进行交互。

V：view，视图，和MVC中的C功能相同，接收请求，进行处理，与M和T进行交互，返回应答。

T：Template，模板，和MVC中的V功能相同，产生Html页面



1、 用户点击注册按钮，将要注册的内容发送给网站的服务器。

2、 View视图，接收到用户发来的注册数据，View告诉Model将用户的注册信息保存进数据库。

3、 Model层将用户的注册信息保存到数据库中。

4、 数据库将保存的结果返回给Model

5、 Model将保存的结果给View视图。

6、 View视图告诉Template模板去产生一个Html页面。

7、 Template生成html内容返回给View视图。

8、 View将html页面内容返回给浏览器。

9、 浏览器拿到view返回的html页面内容进行解析，展示。

django 中当一个用户登录 **A** 应用服务器（进入登录状态），然后下次请求被 **nginx** 代理到 **B** 应用服务器会出现什么影响？

> 如果用户在**A**应用服务器登陆的**session**数据没有共享到**B**应用服务器，那么之前的登录状态就没有了。

跨域请求问题**django**怎么解决的（原理）

- 启用中间件
- post请求
- 验证码
- 表单中添加 **csrf_token** 标签

请解释或描述一下**Django**的架构

对于**Django**框架遵循**MVC**设计，并且有一个专有名词：**MVT**

- **M**全拼为Model，与MVC中的M功能相同，负责数据处理，内嵌了ORM框架
- **V**全拼为View，与MVC中的C功能相同，接收HttpRequest，业务处理，返回HttpResponse
- **T**全拼为Template，与MVC中的V功能相同，负责封装构造要返回的html，内嵌了模板引擎

Django对数据查询结果排序怎么做，降序怎么做，查询大于某个字段怎么做

- 排序使用order_by()
- 降序需要在排序字段名前加-
- 查询字段大于某个值：使用filter(字段名_gt=值)

说一下**Django，MIDDLEWARES**中间件的作用？

> 中间件是介于request与response处理之间的一道处理过程，相对比较轻量级，并且在全局上改变django的输入与输出。

你对**Django**的认识？

> **Django**是走大而全的方向，它最出名的是其全自动化的管理后台：只需要使用起ORM，做简单的对象定义，它就能自动生成数据库结构、以及全功能的管理后台。
>
> **Django**内置的ORM跟框架内的其他模块耦合程度高。
>
> 应用程序必须使用Django内置的ORM，否则就不能享受到框架内提供的种种基于其ORM的便利；理论上可以切换掉其ORM模块，但这就相当于要把装修完毕的房子拆除重新装修，倒不如一开始就去毛胚房做全新的装修。
>
> **Django**的卖点是超高的开发效率，其性能扩展有限；采用Django的项目，在流量达到一定规模后，都需要对其进行重构，才能满足性能的要求。
>
> **Django**适用的是中小型的网站，或者是作为大型网站快速实现产品雏形的工具。
>
> Django
>
> 模板的设计哲学是彻底的将代码、样式分离； Django从根本上杜绝在模板中进行编码、处理数据的可能。

Django重定向你是如何实现的？用的什么状态码？

- 使用**HttpResponseRedirect**
- **redirect**和**reverse**
- 状态码：**302,301**

创建项目，创建应用的命令？

> django-admin startproject 项目名
>
> python manage.py startapp 应用名

生成迁移文件和执行迁移文件的命令是什么？

> python manage.py makemigrations
>
> python manage.py migrate

关系型数据库的关系包括哪些类型？

- ForeignKey：一对多，将字段定义在多的一端中。
- ManyToManyField：多对对：将字段定义在两端中。
- OneToOneField：一对一，将字段定义在任意一端中。

查询集返回列表的过滤器有哪些？

- all() ：返回所有的数据
- filter()：返回满足条件的数据
- exclude()：返回满足条件之外的数据，相当于sql语句中where部分的not关键字
- order_by()：排序

判断查询集正是否有数据？

> exists()：判断查询集中否有数据，如果有则返回True，没有则返回False。

Django本身提供了**runserver**，为什么不能用来部署？

> **runserver** 方法是调试 Django 时经常用到的运行方式，它使用 Django 自带的
>
> **WSGI Server** 运行，主要在测试和开发中使用，并且 runserver 开启的方式也是单进程 。
>
> uWSGI是一个 Web 服务器，它实现了WSGI 协议、uwsgi、http 等协议。注意 uwsgi 是一种通信协议，而 uWSGI 是实现 uwsgi 协议和 WSGI 协议的Web 服务器。uWSGI 具有超快的性能、低内存占用和多app 管理等优点，并且搭配着 Nginx
>
> 就是一个生产环境了，能够将用户访问请求与应用 app 隔离开，实现真正的部署。相比来讲，支持的并发量更高，方便管理多进程，发挥多核的优势，提升性能。

Django里QuerySet的get和filter方法的区别

- 【输入参数】
  - get 的参数只能是model中定义的那些字段，只支持严格匹配
  - filter 的参数可以是字段，也可以是扩展的where查询关键字，如in，like等
- 【返回值】
  - get 返回值是一个定义的model对象
  - filter 返回值是一个新的QuerySet对象，然后可以对QuerySet在进行查询返回新的QuerySet对象，支持链式操作QuerySet一个集合对象，可使用迭代或者遍历，切片等，但是不等于list类型(使用一定要注意)
- 【异常】
  - get 只有一条记录返回的时候才正常,也就说明get的查询字段必须是主键或者唯一约束的字段。当返回多条记录或者是没有找到记录的时候都会抛出异常
  - filter 有没有匹配的记录都可以。

简述Django对http请求的执行流程

> 在接受一个Http请求之前的准备
>
> 启动一个支持WSGI网关协议的服务器监听端口等待外界的Http请求，比如Django自带的开发者服务器或者uWSGI服务器。
>
> 服务器根据WSGI协议指定相应的Handler来处理Http请求，并且初始化该Handler，在Django框架中由框架自身负责实现这一个Handler。
>
> 此时服务器已处于监听状态，可以接受外界的Http请求
>
> 当一个http请求到达服务器的时候
>
> 服务器根据WSGI协议从Http请求中提取出必要的参数组成一个字典（environ）并传入Handler中进行处理。
>
> 在Handler中对已经符合WSGI协议标准规定的http请求进行分析，比如加载Django提供的中间件，路由分配，调用路由匹配的视图等。
>
> 返回一个可以被浏览器解析的符合Http协议的HttpResponse。

简述Django下的（内建）缓存机制

> Django根据设置的缓存方式，浏览器第一次请求时，cache会缓存单个变量或整个网页等内容到硬盘或者内存中，同时设置response头部，当浏览器再次发起请求时，附带f-Modified-Since请求时间到Django，Django 发现f-Modified-Since会先去参数之后，会与缓存中的过期时间相比较，如果缓存时间比较新，则会重新请求数据，并缓存起来然后返回response给客户端，如果缓存没有过期，则直接从缓存中提取数据，返回给response给客户端。

Django中model的SlugField类型字段有什么用途？

> SlugField字段是将输入的内容中的空格都替换成‘-’之后保存，Slug 是一个新闻术语，通常是某些东西的短标签。一个slug只能包含字母、数字、下划线或者是连字符，通常用来作为短标签。通常它们是用来放在URL里的。
>
> SlugField字段的Field.db_index自动设置为True。
>
> 通常根据另一个值自动生成slug来填充到SlugField的值

Django中如何加载初始化数据?

> Django在创建对象时在盗用save()方法后，ORM框架会把对象的属性转换为写入到数据库中，实现对数据库的初始化；通过操作对象，查询数据库，将查询集返回给视图函数，通过模板语言展现在前端页面

apache和nginx的区别

Nginx相对Apache的优点：

- 轻量级，同样起web 服务，比apache 占用更少的内存及资源
- 抗并发，nginx 处理请求是异步非阻塞的，支持更多的并发连接，而apache 则是阻塞型的，在高并发下nginx 能保持低资源低消耗高性能
- 配置简洁
- 高度模块化的设计，编写模块相对简单
- 社区活跃

Apache相对Nginx的优点：

- rewrite ，比nginx 的rewrite 强大
- 模块超多，基本想到的都可以找到
- 少bug ，nginx 的bug 相对较多
- 超稳定。

django中那里用到了线程，那里用到了协程，那里用到了进程？

```
1.django利用多线程增加异步任务.celery消息队列.
2.django中使用多线程发送邮件.send_mail().
3.django原生为单线程程序,当第一个请求没有完成时,第二个请求阻塞,直到第一个请求完成,第二个请  求才会执行.
使用uwsgi编程多并发,使用nginx+uwsgi提供高并发,nginx的并发能力超高,单台并发能力过万  (不绝对).
4.django自带的development server为多线程模式,但是他还有一个小问题就是它不是线程安全的。    可能在请求很多时会出现数据不同步，
当然，这一般不是问题，因为我们通常只在自己机器上调试时才用Development Server。
```

商品是怎样分类，存在怎么样的关系？

商品分类先要了解两个概念 spu 和sku, spu指一个商品集合，一般来说就是一个集合链。一个商品 的集合链会包括相似款式和不同的尺寸，sku则是最小品类单元，同一个款式的商品不同的尺码也算 不同的sku。sku多见于前台的商品编号，spu多见于后台的商品管理 例如 iphone4就是一个 SPU，与商家，与颜色、款式、套餐都无关。

```
SKU=stock keeping unit(库存量单位)
SKU即库存进出计量的单位，  可以是以件、盒、托盘等为单位。
SKU是物理上不可分割的最小存货单元。在使用时要根据不同业态，不同管理模式来处理。在服装、    鞋类商品中使用最多最普遍。
例如：
纺织品中一个SKU通常表示：规格、颜色、款式。
```

支付宝支付实现？

- 1.在支付宝沙箱环境下进行，在自己电脑生成公钥（解密）和私钥（加密），将自己公钥设置在沙箱应用中，获取支付宝对应公钥，将支付宝公钥和自己电脑私钥置于项目中，用于django网站和支付宝之间的通信安全
- 2.用户点击去付款（订单id），请求django对应视图，校验之后使用python工具包调用支付宝支付接口（订单id，总金额，订单标题），支付宝返回支付页面，django引导用户到支付页面，用户登录并支付，由于本项目没有公网ip，支付宝无法返回给django支付结果，django自己调用支付查询接口获取支付结果，返回给客户支付结果

类似**qq**在线人数，用**ajax**怎么实现？

可以使用ajax的异步处理,(单使一个一个进程或线程)每隔一段时间请求数据库人数数据。

Ajax主要功能是实现异步,事件触发,Ajax 会创建一个xmlhttprequest,对象,把http方法和目标url,以及

回调函数函数设置到xmlhttprequest对象,通过xmlhttprequest对象向浏览器发送请求,请求发送后继

续响应用户,并交互,只有等到请求真正从服务器返回的时候才调用callback()函数,对响应数据进行处理.

关键的技术: xhtml css ,用来格式化输出页面,dom (动态修改文档的内容和结构,)，xml进行数据交换和

处理, javascript 进行上述技术的捆绑,使其协同工作.

总结：

```
Ajax 利用javascript创建xmlhttpscript对象,再由javascript调用xmljtmlscript对象完成随异步通讯最后通过javascript 调用dom的属性
和方法进行页面不完全刷新
```

django关闭浏览器，怎么清除**cookies**和**session**？

```
cookie是有过期时间的，如果不指定，默认关闭浏览器之后cookie就会过期。
```

```
如果 SESSION_EXPIRE_AT_BROWSER_CLOSE 设置为 False ，cookie可以在用户浏览器中保持有效达 SESSION_COOKIE_AGE 秒(存活时间)。
如果不想用户每次打开浏览器都必须重新登陆的话，可以用这个参数。
如果 SESSION_EXPIRE_AT_BROWSER_CLOSE 设置为 True ，当浏览器关闭时，Django会使cookie失效
Django中操作session：
session也是有过期时间，如果不指定，默认两周就会过期。
request.session.set_expiry(0);那么当浏览器关闭时，session失效
删除session：del request[key]
```

执行**migrate**迁移之后，怎么返回？

python manage.py makemigrations & python manage.py migrate

前者是将model层转为迁移文件migration，后者将新版本的迁移文件执行，更新数据库。 这两中命令调用默认为全局，即对所有最新更改的model或迁移文件进行操作。如果想对部分app进行操作，就要在其后追加app name：

- $ python manage.py makemigrations app_name
- $ python manage.py migrate app_name
- 如果想要精确到某个迁移文件(0004_xxx.py)：（回退）
- $ python manage.py migrate app_name 0004
- 如果想看迁移文件的执行状态，可以用showmigrations命令查看：
- $ python manage.py showmigrations

项目中使用什么调试？

- 1、在Eclipse+Pydev中调试Django

  适用于测试环境。 可进行单步调试，查看变量值，当出现except时，可以用Python标准模块traceback的print_exc()函数查看函数调用链， 是最强大的调试利器。

- 2、使用Django的error page

  适用于测试环境。 Django的error page功能很强大，能提供详细的traceback，包括局部变量的值，以及一个纯文本的异常信息。拥有同phpinfo() 一样的作用，可以展示当前应用的相关设置，包括请求中的 GET, POST and COOKIE 数据以及HTTP环境中的所有重要META fields。

- 3、django-debug-toolbar

  不确定是否用于生产环境。听说功能非常强大。

- 4、输出log到开发服务器终端中

  适用于生产环境。 借助python的logging模块

项目怎么优化？提过哪些建议？

对于开发人员来说，网站性能优化一般包括Web前端性能优化、应用服务器性能优化、存储服务器性能优化三类。

Web前端性能优化：

- 1、减少http请求 http协议是无状态的应用层协议，意味着每次http请求都需要建立通信链路、进行数据传输，而在服务器端，每个http请求都需要启动独立的线程去处理。减少http请求的数目可有效提高访问性能。减少**http**的主要手段是合并**CSS**、合并**javascript**、合并图片。
- 2、使用浏览器缓存 对一个网站而言，CSS、javascript、logo、图标，这些静态资源文件更新的频率都比较低，而这些文件又几乎是每次http请求都需要的。如果将这些文件缓存在浏览器中，可以极好的改善性能。通过设置http头中的cache-control和expires的属性，可设定浏览器缓存，缓存时间可以自定义。
- 3、启用压缩 在服务器端对文件进行压缩，在浏览器端对文件解压缩，可有效减少通信传输的数据量。如果可以的话，尽可能的将外部的脚本、样式进行合并，多个合为一个。文本文件的压缩效率可达到80%以上，因此HTML、CSS、javascript文件启用GZip压缩可达到较好的效果。但是压缩对服务器和浏览器产生一定的压力，在网络带宽良好，而服务器资源不足的情况下要综合考虑。
- 4、CSS放在页面最上部，javascript放在页面最下面 浏览器会在下载完成全部CSS之后才对整个页面进行渲染，因此最好的做法是将CSS放在页面最上面，让浏览器尽快下载CSS。 Javascript则相反，浏览器在加载javascript后立即执行，有可能会阻塞整个页面，造成页面显示缓慢，因此javascript最好放在页面最下面。

应用服务器优化

应用服务器也就是处理网站业务的服务器，网站的业务代码都部署在这里，主要优化方案有缓存、异步、集群等。

1、合理使用缓存

当网站遇到性能瓶颈时，第一个解决方案一般是缓存。在整个网站应用中，缓存几乎无处不在，无论是客户端，还是应用服务器，或是数据库服务器。在客户端和服务器的交互中，无论是数据、文件都可以缓存，合理使用缓存对网站性能优化非常重要。

缓存一般用来存放那些读写次数比较高，变化较少的数据，比如网站首页的信息、商品的信息等。应用程序读取数据时，一般是先从缓存中读取，如果读取不到或数据已失效，再访问磁盘数据库，并将数据再次写入缓存。

缓存的基本原理是将数据存储在相对有较高访问速度的存储介质中，比如内存。一方面缓存访问速度快，另一方面，如果缓存的数据是需要经过计算处理得到的，那使用缓存还可以减少服务器处理数据的计算时间。

使用缓存并不是没有缺陷：内存资源是比较宝贵的，不可能将所有数据都缓存，一般频繁修改的数据不建议使用缓存，这会导致数据不一致。

网站数据缓存一般遵循二八定律，即80%的访问都在20%的数据上。所以，一般将这20%的数据缓存，可以起到改善系统性能，提高服务器读取效率。

2、异步操作

使用消息队列将调用异步化，可以改善网站系统的性能。

在不使用消息队列的情况下，用户的请求直接写入数据库，在高并发的情况下，会对数据库造成非常大的压力，也会延迟响应时间。

在使用消息队列后，用户请求的数据会发送给消息队列服务器，消息队列服务器会开启进程，将数据异步写入数据库。消息队列服务器的处理速度远超过数据库，因此用户的响应延迟可得到改善。

消息队列可以将短时间内的高并发产生的事务消息，存储在消息队列中，从而提高网站的并发处理能力。在电商网站的促销活动中，合理使用消息队列，可以抵御短时间内用户高并发的冲击。

3、使用集群

在网站高并发访问的情况下，使用负载均衡技术，可以为一个应用构建由多台服务器组成的服务器集群，将并发访问请求，分发到多台服务器上处理，避免单一服务器因负载过大，而导致响应延迟。

4、代码优化

网站的业务逻辑代码主要部署在应用服务器上，需要处理复杂的并发事务。合理优化业务代码，也可以改善网站性能。

任何web网站都会遇到多用户的并发访问，大型网站的并发用户会达到数万。每个用户请求都会创建一个独立的系统进程去处理。由于线程比进程更轻量，占用资源更少，所以，目前主流的web应用服务器都采用多线程的方式，处理并发用户的请求，因此，网站开发多数都是多线程编程。

使用多线程的另一个原因是服务器有多个CPU，现在手机都到了8核CPU的时代，一般的服务器至少是16核CPU，要想最大限度的使用这些CPU，必须启动多线程。

那么，启动多少线程合适呢？

启动线程数和CPU内核数量成正比，和IO等待时间成正比。如果都是计算型的任务，那么线程数最多不要超过CPU内核数，因为启动再多，CPU也来不及调用。如果任务是等待读写磁盘、网络响应，那么多启动线程会提高任务并发度，提高服务器性能。

或者用个简化的公式来描述：

启动线程数 = (任务执行时间/(任务执行事件 - IO等待时间)) * CPU内核数

5、存储优化

数据的读写是网站处理并发访问的另一瓶颈。使用缓存虽然可以解决一部分数据读写压力，但很多时候，磁盘仍然是系统最严重的瓶颈。而且磁盘是网站最重要的资产，磁盘的可用性和容错性也至关重要。

机械硬盘和固态硬盘 机械硬盘是目前最常用的硬盘，通过马达带动磁头到指定磁盘的位置访问数据，每次访问数据都需要移动磁头，在读取连续数据和随机访问上，磁头移动的次数相差巨大，因此机械硬盘的性能表现差别巨大，读写效率较低。而在网站应用中，大多数数据的访问都是随机的，在这种情况下，固态硬盘具有更高的性能。但目前固态硬盘在工艺上、数据可靠性上还有待提升，因此固态硬盘的使用尚未普及，从发展趋势看，取代机械硬盘应该是迟早的事情。

总结：

网站性能优化是在用户高并发访问，网站遇到问题时的解决方案。所以网站性能优化的主要内容是改善高并发用户访问情况下的网站响应速度。

网站性能优化的最终目的是改善用户的体验。但性能优化本身也是需要综合考虑的。比如说，性能提高一倍，服务器数量也要增加一倍，这样的优化是否可以考虑？

技术是由业务驱动的，离开业务的支撑，任何性能优化都是空中楼阁。

描述一下**csrf**原理

- (1)用户浏览并登录信任网站A
- (2)产生A的cookie存放在用户浏览器中
- (3)此时用户在未登出的情况下访问危险网站B
- (4)用户不小心点击了危险网站B的链接,要求访问第三方站点A
- (5)用户浏览器带着(2)处产生的Cookie访问A
- (6)A不知道请求是用户发出的还是B发出的，所以A会执行(5)发出的操作，这样攻击者盗用了用户的身份，发出了恶意请求

xss是什么？

跨站脚本攻击(Cross Site Scripting)，为了不和层叠样式表(Cascading Style Sheets, CSS)的缩写混淆，故将跨站脚本攻击缩写为XSS。恶意攻击者往Web页面里插入恶意Script代码，当用户浏览该页之时，嵌入其中Web里面的Script代码会被执行，从而达到恶意攻击用户的目的

XSS攻击分成两类，一类是来自内部的攻击，主要指的是利用程序自身的漏洞，构造跨站语句 另一类则是来自外部的攻击，主要指的自己构造XSS跨站漏洞网页或者寻找非目标机以外的有跨站漏洞的网页。如当我们要渗透一个站点，我们自己构造一个有跨站漏洞的网页，然后构造跨站语句，通过结合其它技术，如社会工程学等，欺骗目标服务器的管理员打开

XSS分为：存储型和反射型 存储型XSS：存储型XSS，持久化，代码是存储在服务器中的，如在个人信息或发表文章等地方，加入代码，如果没有过滤或过滤不严，那么这些代码将储存到服务器中，用户访问该页面的时候触发代码执行。这种XSS比较危险，容易造成蠕虫，盗窃cookie（虽然还有种DOM型XSS，但是也还是包括在存储型XSS内）。 反射型XSS：非持久化，需要欺骗用户自己去点击链接才能触发XSS代码（服务器中没有这样的页面和内容），一般容易出现在搜索页面。

Flask和Django路由映射的区别？

在django中，路由是浏览器访问服务器时，先访问的项目中的url，再由项目中的url找到应用中url，这些url是放在一个列表里，遵从从前往后匹配的规则。在flask中，路由是通过装饰器给每个视图函数提供的，而且根据请求方式的不同可以一个url用于不同的作用。

Celery的深入了解：

- http://docs.pythontab.com/flask/flask0.10/patterns/celery.html#id5
- http://blog.csdn.net/siddontang/article/details/34447003

解释一下mvt？出了mvt你还熟悉其他的吗？mvc、mvvm

- http://blog.csdn.net/chun_long/article/details/52086565

订单模块如何避免用户抢资源?

使用乐观锁和悲观锁

Nginx主要作用？

代理和反向代理， 负载均衡

一般情况下，nginx的作用: 主要是在django部署的时候，使用nginx+ uwsgi来进行项目的部署，在最前端的nginx，可以接受所有的请求信息，nginx把静态请求自己处理(nginx的强项)，而非静态请求交给 django程序来处理。同时，从安全性上分析，用户访问的url中ip地址都是nginx的，不会访问django程序的服务器，避免受到攻击，保证我们项目程序正确执行在我们的项目中，由于使用了fdfs，即分布式静态文件存储服务器来存储商品的图片，为了提高文件的加载速度，还在 storage服务器上使用了 nginx，加快加载图片的速度

Django中的响应对象有哪些数据？

- response.status_code 返回HTTP响应码
- response.content 返回的内容
- response.content_type 返回的数据类型
- response.charset 返回的编码字符集
- response.set_cookie() 设置Cookie信息
- response.delete_cookie() 删除Cookie信息
- response.write() 向响应体中写数据

nginx如何实现负载均衡？

- 1、轮询(默认)，每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。
- weight指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。
- ip_hash每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。
- fair（第三方）按后端服务器的响应时间来分配请求，响应时间短的优先分配。
- url_hash（第三方）按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

Form表单的提交和ajax的提交

1、提交方式 form表单通常是通过在HTML中定义的action，method及submit来进行表单提交，另外也可以通过在js中调用submit函数来进行表单提交。 具体的提交方式有很多种，比如可以通过封装成XMLHttpRequest对象进行提交。Ajax是基于XMLHttpRequest进行的。 2、页面刷新 Form提交，更新数据完成后，需要转到一个空白页面再对原页面进行提交后处理。哪怕是提交给自己本身的页面，也是需要刷新的，因此局限性很大。 Ajax可以实现页面的局部刷新，整个页面不会刷新。 3、请求由谁来提交 Form提交是浏览器完成的，无论浏览器是否开启JS，都可以提交表单。 Ajax是通过js来提交请求，请求与响应均由js引擎来处理，因此不启用JS的浏览器，无法完成该操作。 4、是否可以上传文件 最初，ajax出于安全性考虑，不能对文件进行操作，所以就不能通过ajax来实现文件上传，但是通过隐藏form提交则可以实现这个功能，所以这也是用隐藏form提交的主要用途。 后来XMLHttpRequest引入了FormData类型，使得通过Ajax也可以实现文件上传。

scrapy和**scrapy-redis**有什么区别？为什么选择**redis**数据库？

- **scrapy**是一个Python爬虫框架，爬取效率极高，具有高度定制性，但是不支持分布式。而scrapy-redis一套基于redis数据库、运行在scrapy框架之上的组件，可以让scrapy支持分布式策略，Slaver端共享Master端redis数据库里的item队列、请求队列和请求指纹集合
- 为什么选择redis数据库，因为redis支持主从同步，而且数据都是缓存在内存中的，所以基于redis的分布式爬虫，对请求和数据的高频读取效率非常高

你用过的爬虫框架或者模块有哪些？谈谈他们的区别或者优缺点？

- Python自带：urllib、urllib2
- 第三方：requests
- 框架： Scrapy
- **urllib**和urllib2模块都做与请求URL相关的操作，但他们提供不同的功能
  - **urllib2.**：urllib2.urlopen可以接受一个Request对象或者url，（在接受Request对象时候，并以此可以来设置一个URL 的headers），urllib.urlopen只接收一个url
  - **urllib**有urlencode,urllib2没有，因此总是urllib，urllib2常会一起使用的原因*
- **scrapy**是封装起来的框架，他包含了下载器，解析器，日志及异常处理，基于多线程，twisted的方式处理，对于固定单个网站的爬取开发，有优势，但是对于多网站爬取 100个网站，并发及分布式处理方面，不够灵活，不便调整与括展
- **request**是一个HTTP库， 它只是用来，进行请求，对于HTTP请求，他是一个强大的库，下载，解析全部自己处理，灵活性更高，高并发与分布式部署也非常灵活，对于功能可以更好实现
- **Scrapy**优点
  - scrapy 是异步的
  - 采取可读性更强的xpath代替正则
  - 强大的统计和log系统
  - 同时在不同的url上爬行
  - 支持shell方式，方便独立调试
  - 写middleware,方便写一些统一的过滤器
  - 通过管道的方式存入数据库
- Scrapy缺点：
  - 基于python的爬虫框架，扩展性比较差
  - 基于twisted框架，运行中的exception是不会干掉reactor，并且异步框架出错后是不会停掉其他任务的，数据出错后难以察觉

常见的反爬虫和应对方法？

- 通过**Headers**反爬虫

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

- 基于用户行为反爬虫

> 还有一部分网站是通过检测用户行为，例如同一IP短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。
>
> 大多数网站都是前一种情况，对于这种情况，使用IP代理就可以解决。可以专门写一个爬虫，爬取网上公开的代理ip，检测后全部保存起来。这样的代理ip爬虫经常会用到，最好自己准备一个。有了大量代理ip后可以每请求几次更换一个ip，这在requests或者urllib2中很容易做到，这样就能很容易的绕过第一种反爬虫。
>
> 对于第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。

- 动态页面的反爬虫

> 上述的几种情况大多都是出现在静态页面，还有一部分网站，我们需要爬取的数据是通过ajax请求得到，或者通过JavaScript生成的。首先用Fiddler对网络请求进行分析。如果能够找到ajax请求，也能分析出具体的参数和响应的具体含义，我们就能采用上面的方法，直接利用requests或者urllib2模拟ajax请求，对响应的json进行分析得到需要的数据。
>
> 能够直接模拟ajax请求获取数据固然是极好的，但是有些网站把ajax请求的所有参数全部加密了。我们根本没办法构造自己所需要的数据的请求。这种情况下就用selenium+phantomJS，调用浏览器内核，并利用phantomJS执行js来模拟人为操作以及触发页面中的js脚本。从填写表单到点击按钮再到滚动页面，全部都可以模拟，不考虑具体的请求和响应过程，只是完完整整的把人浏览页面获取数据的过程模拟一遍。
>
> 用这套框架几乎能绕过大多数的反爬虫，因为它不是在伪装成浏览器来获取数据（上述的通过添加Headers一定程度上就是为了伪装成浏览器），它本身就是浏览器，phantomJS就是一个没有界面的浏览器，只是操控这个浏览器的不是人。利selenium+phantomJS能干很多事情，例如识别点触式（12306）或者滑动式的验证码，对页面表单进行暴力破解等

分布式爬虫主要解决什么问题

- ip
- 带宽
- cpu
- io

写爬虫是用多进程好？还是多线程好？ 为什么？

> IO密集型代码(文件处理、网络爬虫等)，多线程能够有效提升效率(单线程下有IO操作会进行IO等待，造成不必要的时间浪费，而开启多线程能在线程A等待时，自动切换到线程B，可以不浪费CPU的资源，从而能提升程序执行效率)。在实际的数据采集过程中，既考虑网速和响应的问题，也需要考虑自身机器的硬件情况，来设置多进程或多线程

描述下**scrapy**框架运行的机制？

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

数据库的优化？

- 优化索引、SQL 语句、分析慢查询；
- 设计表的时候严格根据数据库的设计范式来设计数据库；
- 使用缓存，把经常访问到的数据而且不需要经常变化的数据放在缓存中，能节约磁盘IO
- 优化硬件；采用SSD，使用磁盘队列技术(RAID0,RAID1,RDID5)等
- 采用MySQL 内部自带的表分区技术，把数据分层不同的文件，能够提高磁盘的读取效率；
- 垂直分表；把一些不经常读的数据放在一张表里，节约磁盘I/O；
- 主从分离读写；采用主从复制把数据库的读操作和写入操作分离开来；
- 分库分表分机器（数据量特别大），主要的的原理就是数据路由；
- 选择合适的表引擎，参数上的优化
- 进行架构级别的缓存，静态化和分布式；
- 不采用全文索引；
- 采用更快的存储方式，例如 NoSQL存储经常访问的数据**

redis中**list**底层实现有哪几种？有什么区别？

- 列表对象的编码可以是ziplist或者linkedlist
- **ziplist**是一种压缩链表，它的好处是更能节省内存空间，因为它所存储的内容都是在连续的内存区域当中的。当列表对象元素不大，每个元素也不大的时候，就采用ziplist存储。但当数据量过大时就ziplist就不是那么好用了。因为为了保证他存储内容在内存中的连续性，插入的复杂度是O(N)，即每次插入都会重新进行realloc。如下图所示，对象结构中ptr所指向的就是一个ziplist。整个ziplist只需要malloc一次，它们在内存中是一块连续的区域。

解析网页的解析器使用最多的是哪几个

> lxml，html5lib，html.parser,lxml-xml

需要登录的网页，如何解决同时限制**ip**，**cookie,session**（其中有一些是动态生成的）在不使用动态爬取的情况下？

> 解决限制IP可以使用代理IP地址池、服务器；
>
> 不适用动态爬取的情况下可以使用反编译JS文件获取相应的文件，或者换用其他平台（比如手机端）看看是否可以获取相应的json文件

验证码的解决（简单的：对图像做处理后可以得到的，困难的：验证码是点击，拖动等动态进行的？）

> 图形验证码：干扰、杂色不是特别多的图片可以使用开源库Tesseract进行识别，太过复杂的需要借助第三方打码平台
>
> 点击和拖动滑块验证码可以借助selenium、无图形界面浏览器（chromedirver或者phantomjs）和pillow包来模拟人的点击和滑动操作，pillow可以根据色差识别需要滑动的位置

使用最多的数据库（**mysql**，**mongodb**，**redis**等），对他的理解？

- MySQL数据库：开源免费的关系型数据库，需要实现创建数据库、数据表和表的字段，表与表之间可以进行关联（一对多、多对多），是持久化存储
- mongodb数据库：是非关系型数据库，数据库的三元素是，数据库、集合、文档，可以进行持久化存储，也可作为内存数据库，存储数据不需要事先设定格式，数据以键值对的形式存储
- redis数据库：非关系型数据库，使用前可以不用设置格式，以键值对的方式保存，文件格式相对自由，主要用与缓存数据库，也可以进行持久化存储

基础的数据结构有哪些？

> 集合、线性结构、树形结构、图状结构
>
> 集合结构:除了同属于一种类型外，别无其它关系
>
> 线性结构:元素之间存在一对一关系常见类型有: 数组,链表,队列,栈,它们之间在操作上有所区别.例如:链表可在任意位置插入或删除元素,而队列在队尾插入元素,队头删除元素,栈只能在栈顶进行插
>
> 入,删除操作.
>
> 树形结构:元素之间存在一对多关系常见类型有:树(有许多特例:二叉树、平衡二叉树、查找树等)
>
> 图形结构:元素之间存在多对多关系,
>
> 图形结构:中每个结点的前驱结点数和后续结点多个数可以任意

怎么去评价一个算法的好坏？

- 时间复杂度：同样的输入规模(问题规模)话费多少时间。
- 空间复杂度：同样的输入规模花费多少空间(主要是内存)。
- 以上两点越小越好。
- 稳定性：不会引文输入的不同而导致不稳定的情况发生。
- 算法的思路是否简单：越简单越容易实现的越好。

算法的特征？

数据结构算法具有五大基本特征：

- 输入
- 输出
- 有穷性
- 确定性
- 可行性

有没有了解过桶排序？

> 工作的原理是将数组分到有限数量的桶子里。每个桶子再个别排序（有可能再使用别的排序算法或是以递归方式继续使用桶排序进行排序）
>
> **1,**桶排序是稳定的
>
> **2,**桶排序是常见排序里最快的一种,比快排还要快…大多数情况下
>
> **3,**桶排序非常快,但是同时也非常耗空间,基本上是最耗空间的一种排序算法

冒泡排序的思想？

- 冒泡思想：通过无序区中相邻记录的关键字间的比较和位置的交换，使关键字最小的记录像气泡一样逐渐向上漂至水面。整个算法是从最下面的记录开始，对每两个相邻的关键字进行比较，把关键字较小的记录放到关键字较大的记录的上面，经过一趟排序后，关键字最小的记录到达最上面，接着再在剩下的记录中找关键字次小的记录，把它放在第二个位置上，依次类推，一直到所有记录有序为止
- 复杂度：时间复杂度为O（n2）,空间复杂度为O(1)

快速排序的思想？

- 快排的基本思想：通过一躺排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一不部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。
- 复杂度：快速排序是不稳定的排序算法，最坏的时间复杂度是O（n2），

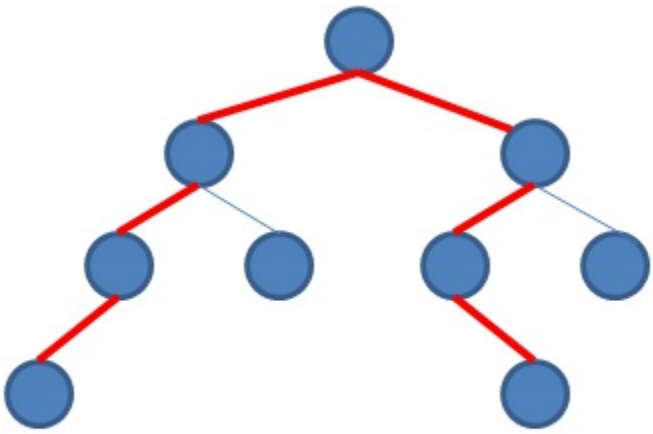
  最好的时间复杂度是**(nlogn),**空间复杂度为**O(logn)**

找出二叉树中最远节点的距离？

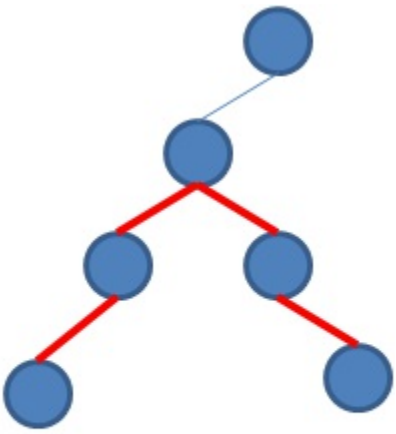
计算一个二叉树的最大距离有两个情况：

- 情况A: 路径经过左子树的最深节点，通过根节点，再到右子树的最深节点。
- 情况B: 路径不穿过根节点，而是左子树或右子树的最大距离路径，取其大者。

只需要计算这两个情况的路径距离，并取其大者，这就是二叉树的最大距离。



情况A



情况B

那种数据结构可以实现递归(栈)?

栈, 递归需要保存正在计算的上下文, 等待当前计算完成后弹出,再继续计算, 只有栈先进后出的的特性才能实现.

你知道哪些排序算法(一般是通过问题考算法)?

冒泡, 选择, 快排, 归并

使用Python写一个二叉树

```
class TreeNode(object):
    def __init__(self, left=None, right=None, data=None):
        self.data = data
        self.left = left
        self.right = right


    def preorder(root):  # 前序遍历
        if root is None:
            return
        else:
            print(root.data)
            preorder(root.left)
            preorder(root.right)


    def inorder(root):  # 中序遍历
        if root is None:
            return
        else:
```

```
            inorder(root.left)
            print (root.data)
            inorder(root.right)


    def postorder(root):  # 后序遍历
        if root is None:
            return
        else:
            postorder(root.left)
            postorder(root.right)
            print (root.data)
```

写一个霍夫曼树

```
class TreeNode(object):
    def __init__(self, left=None, right=None, data=None):
        self.data = data
        self.left = left
        self.right = right


    # 根据数组 a 中 n 个权值建立一棵哈夫曼树，返回树根指针
    def CreateHuffman(a):
        a.sort()  # 对列表进行排序(从小到大)
        n = len(a)
        b = [TreeNode(data=a[i]) for i in range(n)]
        for i in range(n):  # 进行 n-1 次循环建立哈夫曼树
            [print(temp, end=' ') for temp in b]
            # k1表示森林中具有最小权值的树根结点的下标，k2为次最小的下标
            k1 = -1
            k2 = 0
            for j in range(n):  # 让k1初始指向森林中第一棵树，k2指向第二棵
                if b[j]:
                    if k1 == -1:
                        k1 = j
                        continue
                    k2 = j
                    break
            print('k1', k1, 'k2', k2)
            # print(b[k1].data, b[k2].data)
            for j in range(k2, n):  # 从当前森林中求出最小权值树和次最小
                if b[j]:
                    if b[j].data < b[k1].data:
                        k2 = k1
                        k1 = j
                    elif b[k2]:
                        if b[j].data < b[k2].data:
                            k2 = j

            # 由最小权值树和次最小权值树建立一棵新树，q指向树根结点
            if b[k2]:
                q = TreeNode()
                q.data = b[k1].data + b[k2].data
                q.left = b[k1]
                q.right = b[k2]

                b[k1] = q  # 将指向新树的指针赋给b指针数组中k1位置
                b[k2] = None  # k2位置为空
                return q  # 返回整个哈夫曼树的树根指针
```

深度优先遍历和广度优先遍历的区别？

- 二叉树的深度优先遍历的非递归的通用做法是采用栈，广度优先遍历的非递归的通用做法是采用队列。
- 深度优先遍历：对每一个可能的分支路径深入到不能再深入为止，而且每个结点只能访问一次。要特别注意的是，二叉树的深度优先遍历比较特殊，可以细分为先序遍历、中序遍历、后序遍历。具体说明如下：
  - 先序遍历：对任一子树，先访问根，然后遍历其左子树，最后遍历其右子树。
  - 中序遍历：对任一子树，先遍历其左子树，然后访问根，最后遍历其右子树。
  - 后序遍历：对任一子树，先遍历其左子树，然后遍历其右子树，最后访问根。
  - 广度优先遍历：又叫层次遍历，从上往下对每一层依次访问，在每一层中，从左往右（也可以从右往左）访问结点，访问完一层就进入下一层，直到没有结点可以访问为止。
- 深度优先搜素算法：不全部保留结点，占用空间少；有回溯操作(即有入栈、出栈操作)，运行速度慢。
- 广度优先搜索算法：保留全部结点，占用空间大； 无回溯操作(即无入栈、出栈操作)，运行速度快。

通常 深度优先搜索法不全部保留结点，扩展完的结点从数据库中弹出删去，这样，一般在数据库中存储的结点数就是深度值，因此它占用空间较少。

所以，当搜索树的结点较多，用其它方法易产生内存溢出时，深度优先搜索不失为一种有效的求解方法。

广度优先搜索算法，一般需存储产生的所有结点，占用的存储空间要比深度优先搜索大得多，因此，程序设计中，必须考虑溢出和节省内存空间的问题。

但广度优先搜索法一般无回溯操作，即入栈和出栈的操作，所以运行速度比深度优先搜索要快些

写一个二分查找

```
def binary_search(a, n, key):
    m = 0
    l = 0
    r = n - 1  # 闭区间[0, n - 1]
    while (l < r):
        m = l + ((r - l) >> 1)  # 向下取整
        if (a[m] < key):
            l = m + 1
        else:
            r = m

    if (a[r] == key): return r;
    return -1
```

1.什么是Python？

- Python是一种编程语言，它有对象、模块、线程、异常处理和自动内存管理，可以加入其他语言的对比
- Python是一种解释型语言，Python代码在运行之前不需要编译。
- Python是动态类型语言，在声明变量时，不需要说明变量的类型。
- Python适合面向对象的编程，因为它支持通过组合与继承的方式定义类。
- 在Python语言中，函数是第一类对象。
- Python代码编写快，但是运行速度比编译型语言通常要慢。
- Python用途广泛，常被用走"胶水语言"，可帮助其他语言和组件改善运行状况。
- 使用Python，程序员可以专注于算法和数据结构的设计，而不用处理底层的细节。

2.什么是Python自省？

python自省是python具有的一种能力，使程序员面向对象的语言所写的程序在运行时,能够获得对象的类python型。Python是一种解释型语言。为程序员提供了极大的灵活性和控制力。

3.什么是PEP8？

PEP8是一种编程规范，内容是一些关于如何让你的程序更具可读性的建议。

4.什么是pickling和unpickling？

Pickle模块读入任何Python对象，将它们转换成字符串，然后使用dump函数将其转储到一个文件中——这个过程叫做pickling。反之从存储的字符串文件中提取原始Python对象的过程，叫做unpickling。

5.什么是Python装饰器？

Python装饰器是Python中的特有变动，可以使修改函数变得更容易。

6.什么是Python的命名空间？

在Python中，所有的名字都存在于一个空间中，它们在该空间中存在和被操作——这就是命名空间。它就好像一个盒子，每一个变量名字都对应装着一个对象。当查询变量的时候，会从该盒子里面寻找相应的对象。

7.什么是字典推导式和列表推导式？

它们是可以轻松创建字典和列表的语法结构。

8.Lambda函数是什么？

这是一个常被用于代码中的单个表达式的匿名函数。

9. *args，**kwargs？参数是什么？

如果我们不确定要往函数中传入多少个参数，或者我们想往函数中以列表和元组的形式传参数时，那就使要用*args*；如果我们不知道要往函数中传入多少个关键词参数，或者想传入字典的值作为关键词参数时，那就要使用*kwargs。

10.什么是Pass语句?

Pass是一个在Python中不会被执行的语句。在复杂语句中，如果一个地方需要暂时被留白，它常常被用于占位符。

11. unittest是什么？

在Python中，unittest是Python中的单元测试框架。它拥有支持共享搭建、自动测试、在测试中暂停代码、将不同测试迭代成一组，等等的功能。

12.构造器是什么？

构造器是实现迭代器的一种机制。它功能的实现依赖于yield表达式，除此之外它跟普通的函数没有两样。

13.doc string是什么？

Python中文档字符串被称为docstring，它在Python中的作用是为函数、模块和类注释生成文档。

14.负索引是什么？

Python中的序列索引可以是正也可以是负。如果是正索引，0是序列中的第一个索引，1是第二个索引。如果是负索引，（-1）是最后一个索引而（-2）是倒数第二个索引。

15. 模块和包是什么？

在Python中，模块是搭建程序的一种方式。每一个Python代码文件都是一个模块，并可以引用其他的模块，比如对象和属性。

一个包含许多Python代码的文件夹是一个包。一个包可以包含模块和子文件夹。

16.垃圾回收是什么?

在Python中，为了解决内存泄露问题，采用了对象引用计数，并基于引用计数实现自动垃圾回收。

17. CSRF是什么？

CSRF是伪造客户端请求的一种攻击，CSRF的英文全称是Cross Site Request Forgery，字面上的意思是跨站点伪造请求

1. 如何让你的程序更具可读性？

适当地加入非前导空格，适当的空行以及一致的命名。

2. Python是如何被解释的？

Python是一种解释性语言，它的源代码可以直接运行。Python解释器会将源代码转换成中间语言，之后再翻译成机器码再执行。

3. 如何在Python中拷贝一个对象？

如果要在Python中拷贝一个对象，大多时候你可以用copy.copy()或者copy.deepcopy()。但并不是所有的对象都可以被拷贝。

4. 如何用Python删除一个文件？

使用函数os.remove("file")

5. 如何将一个数字转换成一个字符串？

你可以使用自带函数str()将一个数字转换为字符串。如果你想要八进制或者十六进制数，可以用oct()或hex()。

6. Python是如何进行内存管理的？

Python的内存管理是由私有heap空间管理的。所有的Python对象和数据结构都在一个私有heap中。程序员没有访问该heap的权限，只有解释器才能对它进行操作。为Python的heap空间分配内存是由Python的内存管理模块进行的，其核心API会提供一些访问该模块的方法供程序员使用。Python有自带的垃圾回收系统，它回收并释放没有被使用的内存，让它们能够被其他程序使用。

7. 如何实现tuple和list的转换？

以list作为参数将tuple类初始化，将返回tuple类型

以tuple作为参数将list类初始化，将返回list类型

8. Python里面如何生成随机数？

在python中用于生成随机数的模块是random，在使用前需要import. 如下例子可以酌情列举：

- random.random()：生成一个0-1之间的随机浮点数
- random.uniform(a, b)：生成[a,b]之间的浮点数
- random.randint(a, b)：生成[a,b]之间的整数
- random.randrange(a, b, step)：在指定的集合[a,b)中，以step为基数随机取一个数
- random.choice(sequence)：从特定序列中随机取一个元素，这里的序列可以是字符串，列表，元组等

9. 如何在一个function里面设置一个全局的变量

如果要给全局变量在一个函数里赋值，必须使用global语句。global VarName的表达式会告诉Python， VarName是一个全局变量，这样Python就不会在局部命名空间里寻找这个变量了

10. Python如何实现单例模式？其他23种设计模式python如何实现？

单例模式主要有四种方法：**new**、共享属性、装饰器、import。

其他23种设计模式可基本分为创建型、结构型和行为型模式。

创建模式，提供实例化的方法，为适合的状况提供相应的对象创建方法。

结构化模式，通常用来处理实体之间的关系，使得这些实体能够更好地协同工作。

行为模式，用于在不同的实体建进行通信，为实体之间的通信提供更容易，更灵活的通信方法。

各模式的实现可根据其特点编写代码（限于篇幅，此处不做示例）

11. 如何判断单向链表中是否有环

首先遍历链表，寻找是否有相同地址，借此判断链表中是否有环。如果程序进入死循环，则需要一块空间来存储指针，遍历新指针时将其和储存的旧指针比对，若有相同指针，则该链表有环，否则将这个新指针存下来后继续往下读取，直到遇见NULL，这说明这个链表无环。

12. 如何遍历一个内部未知的文件夹？

常用的有以下这几种办法：os.path.walk()，os.walk()，listdir

13. mysql数据库如何分区、分表？

分表可以通过三种方式：mysql集群、自定义规则和merge存储引擎。

分区有四类：

- RANGE 分区：基于属于一个给定连续区间的列值，把多行分配给分区。
- LIST 分区：类似于按RANGE分区，区别在于LIST分区是基于列值匹配一个离散值集合中的某个值来进行选择。
- HASH分区：基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含MySQL 中有效的、产生非负整数值的任何表达式。
- KEY 分区：类似于按HASH分区，区别在于KEY分区只支持计算一列或多列，且MySQL 服务器提供其自身的哈希函数。必须有一列或多列包含整数值。

14. 如何对查询命令进行优化？

- a. 应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索。
- b. 应尽量避免在 where 子句中对字段进行 null 值判断，避免使用!=或<>操作符，避免使用 or 连接条件，或在where子句中使用参数、对字段进行表达式或函数操作，否则会导致权标扫描
- c. 不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。
- d. 使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用。
- e. 很多时候可考虑用 exists 代替 in
- f. 尽量使用数字型字段
- g. 尽可能的使用 varchar/nvarchar 代替 char/nchar
- h. 任何地方都不要使用 select *from t* ，用具体的字段列表代替“*”，不要返回用不到的任何字段。
- i. 尽量使用表变量来代替临时表。
- j. 避免频繁创建和删除临时表，以减少系统表资源的消耗。

- k. 尽量避免使用游标，因为游标的效率较差。
- l. 在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF
- m. 尽量避免大事务操作，提高系统并发能力。
- n. 尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

15. 如何理解开源？

开源，即开放源代码。开源诞生于软件行业，它不仅仅代表软件源代码的开放，本身即意味着自由、共享和充分利用资源。开源是一种精神，是一种文化，如今已经成为软件业发展的大势所趋。

16. 如何理解MVC/MTV框架？

MVC就是把Web应用分为模型（M），控制器（C）和视图（V）三层，他们之间以一种插件式的、松耦合的方式连接在一起。MTV模式本质上和MVC是一样的，也是为了各组件间保持松耦合关系，只是定义上有些许不同。

17. MSSQL的死锁是如何产生的？

如下是死锁产生的四个必要条件：

- 互斥条件：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用毕释放。
- 请求和保持条件：指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。
- 不剥夺条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。
- 环路等待条件：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0，P1，P2，…，Pn}中的P0正在等待一个P1占用的资源；P1正在等待P2占用的资源，……，Pn正在等待已被P0占用的资源。

18. Sql注入是如何产生的，如何防止？

程序开发过程中不注意规范书写sql语句和对特殊字符进行过滤，导致客户端可以通过全局变量POST和GET提交一些sql语句正常执行。产生Sql注入。下面是防止办法：

- a. 过滤掉一些常见的数据库操作关键字，或者通过系统函数来进行过滤。
- b. 在PHP配置文件中将Register_globals=off;设置为关闭状态
- c. SQL语句书写的时候尽量不要省略小引号(tab键上面那个)和单引号
- d. 提高数据库命名技巧，对于一些重要的字段根据程序的特点命名，取不易被猜到的
- e. 对于常用的方法加以封装，避免直接暴漏SQL语句
- f. 开启PHP安全模式：Safe_mode=on;
- g. 打开magic_quotes_gpc来防止SQL注入
- h. 控制错误信息：关闭错误提示信息，将错误信息写到系统日志。
- i. 使用mysqli或pdo预处理。

19. xxs如何预防？

XSS漏洞难以检测，但是为了WEB安全仍需要尽力避免：

针对反射型和存储型XSS，需要服务端和前端共同预防，针对用户输入的数据做解析和转义，对于前端开发而言，则是善于使用escape，针对data URI内容做正则判断，禁止用户输入非显示信息。

对于DOM XSS，由于造成XSS的原因在于用户的输入，因此在前端，需要特别注意用户输入源，并对可能造成的XSS的操作需要进行字串转义。

20. 如何生成共享秘钥？ 如何防范中间人攻击？

密钥的生成是通过使用全局配置命令完成的：对于不可输出密钥是。标记（label）是可选择的；如果没有指定标记，那么密钥名称将是hostname.domain-name。

对于中间人的攻击，可以采用如下防范手段：

- a. 通过采用动态ARP检测、DHCP Snooping等控制操作来加强网络基础设施
- b. 采用传输加密
- c. 使用CASBs（云访问安全代理）
- d. 创建RASP（实时应用程序自我保护）
- e. 阻止自签名证书
- f. 强制使用SSL pinning
- g. 安装DAM（数据库活动监控）

21. 如何管理不同版本的代码？

进行版本管理。可举例告知如何使用Git（或是其他工具）进行追踪。

1. 数组和元组之间的区别？

数组在python中叫作列表。列表可以修改，而元组不可以修改，如果元组中仅有一个元素，则要在元素后加上逗号。元组和列表的查询方式一样。元组只可读不可修改，如果程序中的数据不允许修改可用元组。

2. *new*和*init*的区别？

- **init**是当实例对象创建完成后被调用的，然后设置对象属性的一些初始值。
- **new**是在实例创建之前被调用的，因为它的任务就是创建实例然后返回该实例，是个静态方法。

也就是，**new**在**init**之前被调用，**new**的返回值（实例）将传递给**init**方法的第一个参数，然后**init**给这个实例设置一些参数。

3. Python中单下划线和双下划綫的区别？

"单下划线" 开始的成员变量叫做保护变量，意思是只有类对象和子类对象自己能访问到这些变量；

"双下划线" 开始的是私有成员，意思是只有类对象自己能访问，连子类对象也不能访问到这个数据。

4. 浅拷贝与深拷贝的区别是？

在python中，对象赋值实际上是对象的引用。浅拷贝，没有拷贝子对象，所以原始数据改变，子对象会改变，而深拷贝，包含对象里面的自对象的拷贝，所以原始对象的改变不会造成深拷贝里任何子元素的改变。

5. 使用装饰器的单例和使用其他方法的单例，在后续使用中，有何区别？

Import方法改变了类本身，new方法，但是只是把所有实例对象共享属性，每次产生一个新对象。算作伪单例，共享属性方法实例化了许多个相同属性。所以，装饰器方法最为实用。

6. 多进程与多线程的区别？

- a. 简而言之,一个程序至少有一个进程，一个进程至少有一个线程。
- b. 线程的划分尺度小于进程，使得多线程程序的并发性高。
- c. 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。
- d. 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。
- e. 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

7. select和epoll的区别？

- a. select实现需要自己不断轮询所有fd集合，直到设备就绪，期间可能要睡眠和唤醒多次交替。而epoll其实也需要调用epoll_wait不断轮询就绪链表，期间也可能多次睡眠和唤醒交替，但是它是设备就绪时，调用回调函数，把就绪fd放入就绪链表中，并唤醒在epoll_wait中进入睡眠的进程。虽然都要睡眠和交替，但是select在“醒着”的时候要遍历整个fd集合，而epoll在“醒着”的时候只要判断一下就绪链表是否为空就行了，这节省了大量的CPU时间。
- b. select每次调用都要把fd集合从用户态往内核态拷贝一次，并且要把current往设备等待队列中挂一次，而epoll只要一次拷贝，而且把current往等待队列上挂也只挂一次（在epoll_wait的开始，注意这里的等待队列并不是设备等待队列，只是一个epoll内部定义的等待队列）。这也能节省不少的开销。

8. TCP和UDP的区别？边缘触发和水平触发的区别？

- a. 基本区别：
  - 基于连接与无连接
  - TCP要求系统资源较多，UDP较少；
  - UDP程序结构较简单
  - 流模式（TCP）与数据报模式(UDP);
  - TCP保证数据正确性，UDP可能丢包
  - TCP保证数据顺序，UDP不保证
- b. 编程中的区别
  - socket()的参数不同
  - UDP Server不需要调用listen和accept
  - UDP收发数据用sendto/recvfrom函数
  - TCP：地址信息在connect/accept时确定
  - UDP：在sendto/recvfrom函数中每次均 需指定地址信息
  - UDP：shutdown函数无效

9. HTTP连接：get和post的区别？

- GET请求，请求的数据会附加在URL之后，以?分割URL和传输数据，多个参数用&连接。URL的编码格式采用的是ASCII编码，而不是uniclde，即是说所有的非ASCII字符都要编码之后再传输。
- POST请求：POST请求会把请求的数据放置在HTTP请求包的包体中。上面的item=bandsaw就是实际的传输数据。

因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。

10. varchar与char的区别？

char 长度是固定的，不管你存储的数据是多少他都会都固定的长度。而varchar则处可变长度但他要在总长度上加1字符，这个用来存储位置。所以在处理速度上char要比varchar快速很多，但是对费存储空间，所以对存储不大，但在速度上有要求的可以使用char类型，反之可以用varchar类型。

11. BTree索引和hash索引的区别？

Hash 索引因其结构的特殊性，其检索效率非常高，索引的检索可以一次定位，不像B-Tree 索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问，所以 Hash 索引的查询效率要远高于 B-Tree 索引。但也有如下明显的缺点：

- a. Hash 索引仅仅能满足"="，"IN"和"<=>"查询，不能使用范围查询。
- b. Hash 索引无法被用来避免数据的排序操作。
- c. Hash 索引不能利用部分索引键查询。
- d. Hash 索引在任何时候都不能避免表扫描。
- e. Hash 索引遇到大量Hash值相等的情况后性能并不一定就会比B-Tree索引高。

12. primary key和unique的区别？

- a. 作为Primary Key的域/域组不能为null，而Unique Key可以。
- b. 在一个表中只能有一个Primary Key，而多个Unique Key可以同时存在。
- c. 逻辑设计上讲，Primary Key一般在逻辑设计中用作记录标识，这也是设置Primary Key的本来用意，而Unique Key只是为了保证域/域组的唯一性。

13. ecb和cbc模式有什么区别？

- ECB：是一种基础的加密方式，密文被分割成分组长度相等的块（不足补齐），然后单独一个个加密，一个个输出组成密文。
- CBC：是一种循环模式，前一个分组的密文和当前分组的明文异或操作后再加密，这样做的目的是增强破解难度。ECB和CBC的加密结果是不一样的，两者的模式不同，而且CBC会在第一个密码块运算时加入一个初始化向量。

14. 对称加密与非对称加密的区别？

对称加密，需要对加密和解密使用相同密钥的加密算法。由于其速度快，对称性加密通常在消息发送方需要加密大量数据时使用。所以，对称性加密也称为密钥加密。

而非对称加密算法需要两个密钥：公开密钥和私有密钥。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。

15. Xrange和range的区别？

range([start,] stop[, step])，根据start与stop指定的范围以及step设定的步长，生成一个序列。xrange 用法与 range 完全相同，所不同的是生成的不是一个list对象，而是一个生成器。要生成很大的数字序列的时候，用xrange会比range性能优很多，因为不需要一上来就开辟一块很大的内存空间。range会直接生成一个list对象，而xrange则不会直接生成一个list，而是每次调用返回其中的一个值。

16. os与sys模块的区别？

前者提供了一种方便的使用操作系统函数的方法。后者提供访问由解释器使用或维护的变量和与解释器进行交互的函数。

17. NoSQL和关系数据库的区别？

- a. SQL数据存在特定结构的表中；而NoSQL则更加灵活和可扩展，存储方式可以省是JSON文档、哈希表或者其他方式。
- b. 在SQL中，必须定义好表和字段结构后才能添加数据，例如定义表的主键(primary key)，索引(index),触发器(trigger),存储过程(stored procedure)等。表结构可以在被定义之后更新，但是如果有比较大的结构变更的话就会变得比较复杂。在NoSQL中，数据可以在任何时候任何地方添加，不需要先定义表。
- c. SQL中如果需要增加外部关联数据的话，规范化做法是在原表中增加一个外键，关联外部数据表。而在NoSQL中除了这种规范化的外部数据表做法以外，我们还能用如下的非规范化方式把外部数据直接放到原数据集中，以提高查询效率。缺点也比较明显，更新审核人数据的时候将会比较麻烦。
- d. SQL中可以使用JOIN表链接方式将多个关系数据表中的数据用一条简单的查询语句查询出来。NoSQL暂未提供类似JOIN的查询方式对多个数据集中的数据做查询。所以大部分NoSQL使用非规范化的数据存储方式存储数据。
- e. SQL中不允许删除已经被使用的外部数据，而NoSQL中则没有这种强耦合的概念，可以随时删除任何数据。
- f. SQL中如果多张表数据需要同批次被更新，即如果其中一张表更新失败的话其他表也不能更新成功。这种场景可以通过事务来控制，可以在所有命令完成后再统一提交事务。而NoSQL中没有事务这个概念，每一个数据集的操作都是原子级的。
- g. 在相同水平的系统设计的前提下，因为NoSQL中省略了JOIN查询的消耗，故理论上性能上是优于SQL的。

1.补充缺失的代码

```
def print_directory_contents(sPath):
    """
    这个函数接收文件夹的名称作为输入参数
    返回该文件夹中文件的路径
    以及其包含文件夹中文件的路径
    """
    # 补充代码
```

```
def print_directory_contents(sPath):
    """
    这个函数接收文件夹的名称作为输入参数
    返回该文件夹中文件的路径
    以及其包含文件夹中文件的路径
    """
    # 补充代码
    import os
    for sChild in os.listdir(sPath):
        sChildPath = os.path.join(sPath, sChild)
        if os.path.isdir(sChildPath):
            print_directory_contents(sChildPath)
        else:
            print(sChildPath)
```

2.下面这段代码的输出结果是什么？请解释。

```
def extendlist(val, list=[]):
    list.append(val)
    return list

list1 = extendlist(10)
list2 = extendlist(123, [])
list3 = extendlist('a')

print("list1 = %s" %list1)
print("list2 = %s" %list2)
print("list3 = %s" %list3)
```

输出结果：

```
list1 = [10, 'a']
list2 = [123]
list3 = [10, 'a']
```

新的默认列表只在函数被定义的那一刻创建一次。当extendList被没有指定特定参数list调用时，这组list的值随后将被使用。这是因为带有默认参数的表达式在函数被定义的时候被计算，不是在调用的时候被计算。

3.下面的代码能够运行么？请解释？

```
d = DefaultDict()
d["florp"] = 127
# 能够运行。当key缺失时，执行DefaultDict类，字典的实例将自动实例化这个数列。
```

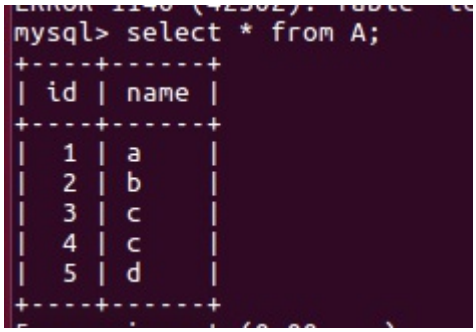
4.将函数按照执行效率高低排序，并证明自己的答案是正确的。

```
def f1(lIn):
    l1 = sorted(lIn)
    l2 = [i for i in l1 if i<0.5]
    return [i*i for i in l2]


def f2(lIn):
    l1 = [i for i in l1 if i<0.5]
    l2 = sorted(l1)
    return [i*i for i in l2]


def f3(lIn):
    l1 = [i*i for i in lIn]
    l2 = sorted(l1)
    return [i for i in l1 if i<(0.5*0.5)]
```

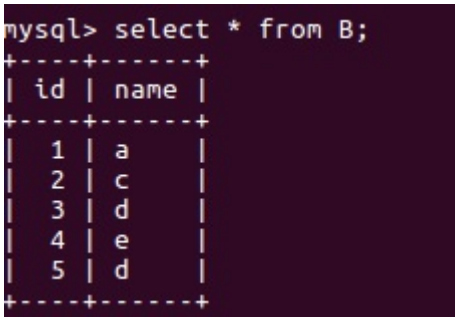
按执行效率从高到低排列：f2、f1和f3。要证明这个答案是正确的，你应该知道如何分析自己代码的性能。Python中有一个很好的程序分析包，可以满足这个需求。

```
import random
import cProfile
lIn = [random.random() for i in range(100000)]
cProfile.run('f1(lIn)')
cProfile.run('f2(lIn)')
cProfile.run('f3(lIn)')
```

1、请拿出B表中的accd，(A表中和B表中的一样的数据)



```
mysql> select * from A;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
|  4 | c    |
|  5 | d    |
+----+------+
```



```
mysql> select * from B;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | c    |
|  3 | d    |
|  4 | e    |
|  5 | d    |
+----+------+
```

> select * from B inner join on B.name = A.name

2、a = “abbbccc”，用正则匹配为abccc,不管有多少b，就出现一次

> 思路：不管有多少个b替换成一个
>
> re.sub(r'b+', 'b', a)

3、xpath使用的什么库？

> lxml

4、py2和py3的区别

- 详见 Python2和Python3的区别 一文。

5、redis里面list内容的长度？

> len key_name

6、多线程交互，访问数据，如果访问到了就不访问了，怎么避免重读？

> 创建一个已访问数据列表，用于存储已经访问过的数据，并加上互斥锁，在多线程访问数据的时候先查看数据是否已经在已访问的列表中，若已存在就直接跳过.

7、Mysql怎么限制IP访问？

> grant all privileges on . to ‘数据库中用户名’@’ip地址’ identified by ‘数据库密码’;

8、带参数的装饰器

带定定长参数的装饰器

```
def new_func(func):
    def wrappedfun(username,passwd):
        if username == 'root' and passwd == '123456789':
            print('通过认证！')
            print('开始执行附加功能')
            return func()
        else:
            print('用户名或密码错误')
            return
    return wrappedfun

@new_func
def orign():
    print('开始执行函数')
orign('root','123456789')
```

带不定长参数的装饰器

```
def new_func(func):
    def wrappedfun(*parts):
        if parts:
            counts = len(parts)
            print('本系统包含 ', end='')
            for part in parts:
                print(part, ' ', end='')
            print('等', counts, '部分')
            return func()
        else:
            print('用户名或密码错误')
            return func()

    return wrappedfun

@new_func
def orign():
    print('开始执行函数')
 orign('硬件', '软件', '用户数据')
```

同时带不定长、关键字参数的装饰器

```
def new_func(func):
    def wrappedfun(*args,**kwargs):
        if args:
            counts = len(args)
            print('本系统包含 ',end='')
            for arg in args:
                print(arg,' ',end='')
            print('等',counts,'部分')
            if kwargs:
                for k in kwargs:
                    v= kwargs[k]
                    print(k,'为：',v)
            return func()
        else:
            if kwargs:
                for kwarg in kwargs:
                    print(kwarg)
                    k,v = kwarg
```

```
                    print(k,'为: ',v)
            return func()
    return wrappedfun

@new_func
def orign():
    print('开始执行函数')

orign('硬件','软件','用户数据',总用户数=5,系统版本='CentOS 7.4')
```

Python主要的内置数据类型有哪些？

> python主要的内置数据类型有：str，int，float，tuple，list，dict，set

print(dir('a'))输出的是什么？

```
会打印出字符型的所有的内置方法
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute_
_', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__',
 '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__set
attr__', '__sizeof__', '__str__', '__subclasshook__', '_formatter_field_name_split', '_formatter_parser', 'capitalize
', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha',
'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'r
find', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
```

给定两个**list**，**A**和**B**，找出相同元素和不同元素

- 相同元素：same = A & B
- 不同元素：differ = A ^ B

请反转字符串

> new_str = old_str[::-1]

交换变量**a,b**的值

> a,b = b,a

用**select**语句输出每个城市中心距离市中心大于**20km**酒店数

> select count（hotel）i from hotel_table where distance >20 group by city

给定一个有序列表，请输出要插入值**k**所在的索引位置

```
def index(list, key)
    if key < list[0]:
        position = 0
    elif key >list[-1]:
        position = len(list)-1
    else:
    for index in range(list):
        if key>list[index] and list[index]>key:
        position = index
```

正则表达式贪婪与非贪婪模式的区别

- 在形式上非贪婪模式有一个“？”作为该部分的结束标志
- 在功能上贪婪模式是尽可能多的匹配当前正则表达式，可能会包含好几个满足正则表达式的字符串，非贪婪模式，在满足所有正则表达式的情况下尽可能少的匹配当前正则表达式

写出开头匹配字母和下划线，末尾是数字的正则表达式

> ^[A-Za-z]|_.*\d$

请说明**HTTP**状态吗的用途，请说明常见的状态码机器意义。

通过状态码告诉客户端服务器的执行状态，以判断下一步该执行什么操作

常见的状态机器码有:

- 100-199：表示服务器成功接收部分请求，要求客户端继续提交其余请求才能完成整个处理过程。
- 200-299：表示服务器成功接收请求并已完成处理过程，常用200（OK请求成功）
- 300-399：为完成请求，客户需要进一步细化请求。302（所有请求页面已经临时转移到新的url），304、307（使用缓存资源）。
- 400-499：客户端请求有错误，常用404（服务器无法找到被请求页面），403（服务器拒绝访问，权限不够）
- 500-599：服务器端出现错误，常用500（请求未完成，服务器遇到不可预知的情况）

当输入**http:mioji3.com**时，返回页面的过程中发生了什么

- 浏览器向DNS服务器发送mioji3.com域名解析请求
- DNS服务器返回解析后的ip给客户端浏览器，浏览器想该ip发送页面请求
- DNS服务器接收到请求后，查询该页面，并将页面发送给客户端浏览器
- 客户端浏览器接收到页面后，解析页面中的引用，并再次向服务器发送引用资源请求
- 服务器接收到资源请求后，查找并返回资源给客户端
- 客户端浏览器接收到资源后，渲染，输出页面展现给用户

闭包

> 在函数内部再定义一个函数，并且这个函数调用外部函数的变量，这个函数和用到的变量称之为闭包

文因互联-软件工程师

请列举**Python2**与**Python**的区别，请将下面的**Python2**的代码转换成**Python3**。

代码：

```
class Point:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return '({}, {})'.format(self.x, self.y)
points = [Point(9, 2), Point(1, 5), Point(2, 7), Point(3, 8), Point(2, 5)]

sorted_points = sorted(points,lambda (x0, y0), (x1, y1): x0 - x1 if x0 != x1 else y0 - y1,lambda point: (point.x, poi
nt.y))
# 预期结果为(1, 5), (2, 5), (2, 7), (3, 8), (9, 2)
print ', '.join(map(str, sorted_points))
```

区别：

> 1.性能：
>
> Py3.1性能比Py2.5慢15%，还有很大的提升空间
>
> 2.编码
>
> Py3.X源码文件默认使用utf-8编码
>
> 3.语法
>
> 1）去除了<>，全部改用!=
>
> 2）去除``，全部改用repr()
>
> 3）关键词加入as和with，还有True,False,None
>
> 4.字符串和字节串
>
> 1）现在字符串只有str一种类型，但它跟2.x版本的unicode几乎一样。
>
> 5.数据类型
>
> 1）Py3.X去除了long类型，现在只有一种整型——int，但它的行为就像2.X版本的long
>
> 2）新增了bytes类型，对应于2.X版本的八位串
>
> 3）dict的.keys()、.items 和.values()方法返回迭代器，而之前的iterkeys()等函数都被废弃。同时去掉的还有dict.has_key()，用 in替代
>
> 6.面向对象
>
> 1）引入抽象基类（AbstraactBase Classes，ABCs）。
>
> 2）容器类和迭代器类被ABCs化

修改后的代码：

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return '({}, {})'.format(self.x, self.y)
points = [Point(9, 2), Point(1,5), Point(2, 7), Point(3, 8), Point(2, 5)]
sorted_points = sorted(points,key=lambda p:(p.x,p.y))
print(', '.join(map(str, sorted_points)))
```

请用 **python** 写一个正则表达式实现电话号码的提取功能，下面是需要满足条件：

- a. 匹配(123)-456-7890 和 123-456-7890
- b. 不匹配(123-456-7890 和 123)-456-7890

代码：

```
import re
def get_tel(text):
    pattern = r'\(\d{3}\)-\d{3}-\d{4}|\d{3}-\d{3}-\d{4}'
    tels = re.search(pattern,text)
    if tels:
        tels = tels.group()
    print('匹配到的电话号码为:',tels)

text = input('请输入字符串:')
get_tel(text)
```

描述 **python** 开发中，协程，线程和进程的区别，请将下面代码改成可以在多核 **CPU** 上并行执行的程序。

代码：

```
import os
import sys

def dump_file(input_dir):
    if not os.path.exists(input_dir):
        print('dir {} not exists'.format(input_dir))
        return

    path_list = [os.path.join(input_dir, f).strip('\n') for f in os.listdir(input_dir)]
    file_path_list = filter(lambda file_path: os.path.isfile(file_path), path_list)
    for file_path in file_path_list:
        print('{}:{} size:{}'.format(os.getpid(), file_path, os.path.getsize(file_path)))
if __name__ == '__main__':
    if len(sys.argv) > 1:
        dump_file(sys.argv[1])
    else:
        dump_file(os.getcwd())
```

- 进程是系统分配系统资源的最小单位,进程之中可以有多个线程,一个进程中的所有资源共享,进程之间资源不会共享;
- 线程是系统进行任务调度的最小单位,一个进程中的线程共享该进程的系统资源,线程是轻量级的进程;

- 协程又称微线程,轻量级线程,执行具有原子性,执行需要程序员来调用度,可以执行效率高

修改后代码：

```
import os
import sys
import multiprocessing
def dump_file(input_dir):
    if not os.path.exists(input_dir):
        print('dir {} not exists'.format(input_dir))
        return
    path_list = [os.path.join(input_dir, f).strip('\n') for f in os.listdir(input_dir)]
    file_path_list = filter(lambda file_path: os.path.isfile(file_path),path_list)
    for file_path in file_path_list:
        print('{}:{} size:{}'.format(os.getpid(), file_path,os.path.getsize(file_path)))

if __name__ == '__main__':
    pool_size = multiprocessing.cpu_count()
    pool = multiprocessing.Pool(processes=pool_size,)
for i in range(pool_size):
    if len(sys.argv) > 1:
        pool.apply_async(dump_file, args=(sys.argv[1],))
    else:
        pool.apply_async(dump_file, args=(os.getcwd(),))
    pool.close()
    pool.join()
```

假设这里有一台服务器地址为**192.168.0.2**，开放端口为**6623**，如何连接到远程服务器操作？

> SSH -p 6623 root@192.168.0.2

找出当前目录下**2**天内新创建的多有**json**文件

> find / -name “*.json” -a -mtime +2

如何查看（监控）**CPU**使用情况，硬盘读写情况以及网络读写情况？

- ①可以调用系统资源监视器
- ②安装glances程序，使用glances名可以直接查看

如何设置一个程序在**Linux**启动的时候自动运行？

> 在/etc/rc.local 中最后一行添加要执行程序的绝对路径

简述在浏览器中输入网址到网页内容展现出来，经历了一个什么样的过程。

- ①浏览器向DNS服务器发起域名解析请求，域名解析服务器返回对应的ip地址
- ②浏览器向解析后的ip地址发起资源请求，服务器查询该请求的url资源，并向客户端浏览器返回该资源的源代码
- ③浏览器接收到该页面资源后检查页面是否用引用资源，若有则再次发起对该引用的请求，并将请求到的资源渲染输出到显示器，最终在显示器上显示出一个请求到的完整页面

请简单描述您对人工智能知识图谱的了解。

人工智能（Artificial Intelligence），英文缩写为AI。它是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学

人工智能是计算机科学的一个分支，它企图了解智能的实质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器，该领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等

实际应用在机器视觉，指纹识别，人脸识别，视网膜识别，虹膜识别，掌纹识别，专家系统，自动规划，智能搜索，定理证明，博弈，自动程序设计，智能控制，机器人学，语言和图像理解，遗传编程等

涉及的学科哲学和认知科学，数学，神经生理学，心理学，计算机科学，信息论，控制论，不定性论

请简单描述您未来**5**年的规划

在未来的1-2年内我能够在岗位沉淀下来,称为技术的中坚力量,在不断的努力后,争取晋升,在未来的3-5年内,如果公司认为我的话,可以发展称为公司的管理层或者架构师, 同时我也希望自己能够在企业的平台上得到进一步的职业能力提升

1、请介绍一下你自己

> 这是面试官100%会问的问题，一般人回答这个问题过于平常，只说姓名、年龄、爱好、所学专业等，如果你用一分钟来重复你的简历，那么，你的印象加分没有了！
>
> 不妨坦诚自信地展现自我，重点突出与应聘职位相吻合的优势。你的相关能力和素质是企业最感兴趣的信息。因为，在许多情况下，在听取你的介绍时，面试官也会抓住他感兴趣的点深入询问。所以，在进行表述时，要力求以真实为基础，顾及表达的逻辑性和条理性，避免冗长而没有重点的叙述。一定要在最短的时间内激发起面试官对你的好感。
>
> 回答范例：我叫XX，今年X岁，XXXX年毕业于XX大学。有3年的开发工作经验，我对技术有深厚的兴趣，专业知识面宽，责任心强，思路清晰，沟通力能好，精通pythonWEB开发技术体系，熟悉MVC。平常有时间看看博客，并且自己也喜欢在CSDN上写技术类的文章，与博友一起讨论。谢谢！

2、为什么来北京找工作？

> 面试官对异地求职者90%都会问的问题，主要考察你是否稳定，个人经验能力之外，排在第一位的就是稳定性，如果不够稳定，那么其余都是空谈。
>
> 回答范例：我来自河南，河南是一个农业大省，IT行业还不是很发达，我是学计算机专业的，也很喜欢这个行业，北京在国内IT行业发展是最快的，所以我想来这里谋求发展，学习更多的新技术，能够带来自我的提升。
>
> 注意：不要说以前公司有多么不好。也不要说哪个哥们混的很不错，羡慕才来北京。因为企业招人想要的都是能够长期工作的人，可能哪个哥们哪天在别的地方又混的更好了，你是不是还要跳槽？所以，只要说来学习更多新技术和管理经验就够了。

3、你为什么离开原来的公司？

- ①最重要的是：应聘者要使找招聘单位相信，应聘者在过往的单位的“离职原因”在此家招聘单位里不存在。
- ②避免把“离职原因”说得太详细、太具体。
- ③不能掺杂主观的负面感受，如“太辛苦”、“人际关系复杂”、“管理太混乱”、“公司不重视人才”、“公司排斥我们某某的员工”等。
- ④但也不能躲闪、回避，如“想换换环境”、“个人原因”等。
- ⑤不能涉及自己负面的人格特征，如不诚实、懒惰、缺乏责任感、不随和等。
- ⑥尽量使解释的理由为应聘者个人形象添彩。
- ⑦相关例子：如“我离职是因为这家公司倒闭；我在公司工作了三年多，有较深的感情；从去年始，由于市场形势突变，公司的局面急转直下；到眼下这一步我觉得很遗憾，但还要面对显示，重新寻找能发挥我能力的舞台。”同一个面试问题并非只有一个答案，而同一个答案并不是在任何面试场合都有效，关键在应聘者掌握了规律后，对面试的具体情况进行把握，有意识地揣摩面试官提出问题的心理背景，然后投其所好。

  分析：除非是薪资太低，或者是最初的工作，否则不要用薪资作为理由。“求发展”也被考官听得太多，离职理由要根据每个人的真实离职理由来设计，但是在回答时一定要表现得真诚。实在想不出来的时候，家在外地可以说是因为家中有事，须请假几个月，公司又不可能准假，所以辞职，这个答案一般面试官还能接受。

4、你最大的缺点是什么？

被面试官问的概率很大，也是HR的杀手锏和狠招，这个问题最难回答，通常面试官不希望听到求职直接回答的缺点是什么，如果求职者说自己小心眼、脾气大、工作效率低，企业肯定不会录用你。不要自作聪明地回答“我最大的缺点就是过于追求完美”，有的人以为这样回答会显得自己比较出色，但事实上，他已经岌岌可危了。面试官喜欢求职者从自己的优点说起，中间加一些小缺点，最后再把问题转到优点上，突出优点的部分，面试官喜欢聪明的求职者。

回答范例：这个问题好难回答啊！我想想……（亲和力表现，也缓解了自己的紧张情绪）

我的缺点是比较执着，比如在技术方面比较爱钻研，有的时候会为一个技术问题加班到深夜。还有就是，工作比较按部就班，总是按照项目经管的要求完成任务。另外的缺点是，总在息的工作范围内有创新意识，并没有扩展给其他同事。这些问题我想我可以进入公司后以最短的时间来解决，我的学习能力很强，我相信可以很快融入公司的企业文化，进入工作状态。我想就这些吧。

5、你未来3-5年的职业规划是怎样的？

大部分面试官司都会问你是否有职业规划，这个问题的背后是了解你的求职动机和对自己中长期职业发展的思考。在回答这个问题之前，要对自己有个清晰的认识，知道自己想往哪个方向发展以及未来有什么计划，要给面试官一种积极向上，好学上进，有追求，有规划的感觉，面试官喜欢有规划的求职者。

回答范例：我希望从现在开始，1-2年之内能够在我目前申请的这个职位上沉淀下来，通过不断的努力后，最好能有晋升，希望3-5年内可以从开发做到架构师。同时我也希望自己能够在企业的平台上得到进一步的职业能力提升。

6、你对薪资的要求？

如果你对薪酬的要求太低，那显然贬低自己的能力；如果你对薪酬的要求太高，那又会显得你分量过重，公司受用不起。一些雇主通常都事先对求职的职位定下开支预算，因而他们第一次提出的价钱往往是他们所能给予的最高价钱，他们问你只不过想证实一下这笔钱是否足以引起你对该工作的兴趣。

回答范例一：我对工资没有硬性要求，我相信贵公司在处理我的问题上会友善合理。我注重的是找对工作机会，所以只要条件公平，我则不会计较太多。

如果你必须自己说出具体数目，那就不要说一个宽泛的范围，不要说7000-8000之间，那样你将只会得到最低限底的数字，也就是7000。最好给出一个具体的数字。

7、什么时候能入职？

大多数企业会关心就职时间，最好是回答“如果被录用的话，到职日可按公司的规定上班”，如果还未辞去上一个工作，但上班时间又太近，似乎有些强人所难，因为交接至少要一个月的时间，应进一步说明原因，录取公司应该会通融的。

8、介绍一个你认为最熟悉的项目(项目经理)

这个问题在技术面试时常被问到，问这个问题的意图是想考察你的成长路径和编程习惯，因为，你最熟悉的项目往往是你成长最快的项目，那个成长最快的项目往往会给你今后的编程习惯留下很多痕迹。所以，通过你对熟悉项目的描述，有经验的他会很快锁定你技术成长中的缺陷和闪光点，从而判断是否能够“为我所用”。

你最好拿出一个自己最擅长技术的那个项目进行介绍，他听完你的介绍后，会接下来进行提问，这样他所有问的问题，你都成竹在胸了。

> 切忌拿自己参与很少的项目来介绍，一旦他深入的询问很可能你会答非所问，反而造成更严重的影响。你大强以和他谈谈在那个项目中获得的经验，这样会引起此君的共鸣，有可能的话，说出一些你自己的小技巧，他会很高兴，同时这场面试也会很轻松，拿到Offer基本没问题了。

9、如果公司录用你，你将怎样开展工作？

> 很多企业在招聘开发人员时很看重是否能够尽快上手，所以回答这个问题时要“实打实”的回答，在回答中最好强调能够“尽快”投入开发工作中，这样领导就放心了，会觉得你不是一个只会盲目工作的人，而是一个按部就班，稳打稳扎的人。
>
> 回答范例：
>
> 我对咱们公司的大体情况只有一个大概了解，在这个职位的工作性质仅仅是我自己的一个理解。作为这个职位而言，我想我首先要对本公司的主营业务要有一个了解，了解公司的业务组成部分、业务的发展方向、我们面向的客户性质等。第二我要了解所属部门在公司中的地位，以及部门的工作目标，从而确定自身的工作努力方向。第三，了解我参与项目的开发方式，架构方式，紧密配合领导工作，尽快投入具体的开发工作中。这就是我开展工作的计划。

10、你还有什么问题问我吗？

> 这个问题看上去可有可无，其实很关键，面试官不喜欢说“没有问题”的人，没有问题就是自寻死路，没有问题传达出你对公司缺乏兴趣，而只是来寻找一笔薪水。其实在面试过程中谦虚礼貌的问面试官怎么称呼，该部门工作中的信息，如项目情况，开发技术再或者说贵公司的晋升机制是什么样的等。表现出一种很积极主动的状态是非常讨巧的。也可以更多的了解到自己来的工作环境。企业很欢迎这样的求职者，因为体现出你对学习的热情和对公司的忠诚度以及你的上进心。

11、你朋友对你的评价？

> 回答提示： 想从侧面了解一下你的性格及与人相处的问题。
>
> 回答样本一：我的朋友都说我是一个可以信赖的人。因为，我一旦答应别人的事情，就一定会做到。如果我做不到，我就不会轻易许诺。
>
> 回答样本二：我觉的我是一个比较随和的人，与不同的人都可以友好相处。在我与人相处时，我总是能站在别人的角度考虑问题。

13、如果通过这次面试我们单位录用了你，但工作一段时间却发现你根本不适合这个职位，你怎么办？

> 回答提示：一段时间发现工作不适合我，有两种情况：
>
> ①如果你确实热爱这个职业，那你就要不断学习，虚心向领导和同事学习业务知识和处事经验，了解这个职业的精神内涵和职业要求，力争减少差距；
>
> ②你觉得这个职业可有可无，那还是趁早换个职业，去发现适合你的，你热爱的职业，那样你的发展前途也会大点，对单位和个人都有好处。

14、在完成某项工作时，你认为领导要求的方式不是最好的，自己还有更好的方法，你应该怎么做？

> ①.原则上我会尊重和服从领导的工作安排，同时私底下找机会以请教的口吻，婉转地表达自己的想法，看看领导是否能改变想法。

②如果领导没有采纳我的建议，我也同样会按领导的要求认真地去完成这项工作。

③.还有一种情况，假如领导要求的方式违背原则，我会坚决提出反对意见，如领导仍固执己见，我会毫不犹豫地再向上级领导反映。

15、如果你的工作出现失误，给本公司造成经济损失，你认为该怎么办？

①我本意是为公司努力工作，如果造成经济损失，我认为首要的问题是想方设法去弥补或挽回经济损失。如果我无能力负责，希望单位帮助解决。

②分清责任，各负其责，如果是我的责任，我甘愿受罚；如果是一个我负责的团队中别人的失误，也不能幸灾乐祸，作为一个团队，需要互相提携共同完成工作，安慰同事并且帮助同事查找原因总结经验。

③总结经验教训，一个人的一生不可能不犯错误，重要的是能从自己的或者是别人的错误中吸取经验教训，并在今后的工作中避免发生同类的错误。检讨自己的工作方法、分析问题的深度和力度是否不够，以致出现了本可以避免的错误。

16、如果你做的一项工作受到上级领导的表扬，但你主管领导却说是他做的，你该怎样？

我首先不会找那位上级领导说明这件事，我会主动找我的主管领导来沟通，因为沟通是解决人际关系的最好办法，但结果会有两种：①我的主管领导认识到自己的错误，我想我会视具体情况决定是否原谅他。②他更加变本加厉的来威胁我，那我会毫不犹豫地找我的上级领导反映此事，因为他这样做会造成负面影响，对今后的工作不利。

17、谈谈你对跳槽的看法？

①正常的“跳槽”能促进人才合理流动，应该支持。

②频繁的跳槽对单位和个人双方都不利，应该反对。

18、工作中你难以和同事、上司相处，你该怎么办？

①我会服从领导的指挥，配合同事的工作。

②我会从自身找原因，仔细分析是不是自己工作做得不好让领导不满意，同事看不惯。还要看看是不是为人处世方面做得不好，如果是这样的话 我会努力改正。

③如果我找不到原因，我会找机会跟他们沟通，请他们指出我的不足，有问题就及时改正。

④作为优秀的员工，应该时刻以大局为重，即使在一段时间内，领导和同事对我不理解，我也会做好本职工作，虚心向他们学习，我相信，他们会看见我在努力，总有一天会对我微笑的。

19、假设你在某单位工作，成绩比较突出，得到领导的肯定。但同时你发现同事们越来越孤立你，你怎么看这个问题？你准备怎么办？

①成绩比较突出，得到领导的肯定是件好事情，以后更加努力。

②检讨一下自己是不是对工作的热心度超过同事间交往的热心了，加强同事间的交往及共同的兴趣爱好。

③工作中，切勿伤害别人的自尊心。

④不再领导前拨弄是非。

20、你最近是否参加了培训课程？谈谈培训课程的内容。是公司资助还是自费参加？

> 公司组织参加，就是传智播客的培训课程（可以多谈谈自己学的技术）。

21、你对于我们公司了解多少?

> 在去公司面试前上网查一下该公司主营业务。如回答：贵公司有意改变策略，加强与国外大厂的OEM合作，自有品牌的部分则透过5海外经销商。

22、请说出你选择这份工作的动机？

> 这是想知道面试者对这份工作的热忱及理解度，并筛选因一时兴起而来应试的人，如果是无经验者，可以强调“就算职种不同，也希望有机会发挥之前的经验”。

23、你最擅长的技术方向是什么？

> 说和你要应聘的职位相关的课程，表现一下自己的热诚没有什么坏处。

24、你能为我们公司带来什么呢？

> 企业很想知道未来的员工能为企业做什么，求职者应再次重复自己的优势，然后说：“就我的能力，我可以做一个优秀的员工在组织中发挥能力，给组织带来高效率和更多的收益”。企业喜欢求职者就申请的职位表明自己的能力，比如申请营销之类的职位，可以说：“我可以开发大量的新客户，同时，对老客户做更全面周到的服务，开发老客户的新需求和消费。”等等。

25、最能概括你自己的三个词是什么？

> 我经常用的三个词是：适应能力强，有责任心和做事有始终，结合具体例子向主考官解释。

26、你的业余爱好是什么？

> 找一些富于团体合作精神的，这里有一个真实的故事：有人被否决掉，因为他的爱好是深海潜水。主考官说：因为这是一项单人活动，我不敢肯定他能否适应团体工作。

27、作为被面试者给我打一下分？

> 试着列出四个优点和一个非常非常非常小的缺点（可以抱怨一下设施，没有明确责任人的缺点是不会有人介意的）.

28、你怎么理解你应聘的职位？

> 把岗位职责和任务及工作态度阐述一下。

29、喜欢这份工作的哪一点？

> 相信其实大家心中一定都有答案了吧！每个人的价值观不同，自然评断的标准也会不同，但是，在回答面试官这个问题时可不能太直接就把自己心理的话说出来，尤其是薪资方面的问题，不过一些无伤大雅的回答是不错的考虑，如交通方便，工作性质及内容颇能符合自己的兴趣等等都是不错的答案，不过如果这时自己能仔细思考出这份工作的与众不同之处，相信在面试上会大大加分。

30、说说你对行业、技术发展趋势的看法？

> 企业对这个问题很感兴趣，只有有备而来的求职者能够过关。求职者可以直接在网上查找对你所申请的行业部门的信息，只有深入了解才能产生独特的见解。企业认为最聪明的求职者是对所面试的公司预先了解很多，包括公司各个部门，发展情况，在面试回答问题的时候可以提到所了解的情况，企业欢迎进入企业的人是“知己”，而不是“盲人”。

31、对工作的期望与目标何在？

> 这是面试者用来评断求职者是否对自己有一定程度的期望、对这份工作是否了解的问题。对于工作有确实学习目标的人通常学习较快，对于新工作自然较容易进入状况，这时建议你，最好针对工作的性质找出一个确实的答案，如业务员的工作可以这样回答：“我的目标是能成为一个超级业务员，将公司的产品广泛的推销出去，达到最好的业绩成效；为了达到这个目标，我一定会努力学习，而我相信以我认真负责的态度，一定可以达到这个目标。”其他类的工作也可以比照这个方式来回答，只要在目标方面稍微修改一下就可以了。

32、说你的家庭？

> 企业面试时询问家庭问题不是非要知道求职者家庭的情况，探究隐私，企业不喜欢探究个人隐私，而是要了解家庭背景对求职者的塑造和影响。企业希望听到的重点也在于家庭对求职者的积极影响。企业最喜欢听到的是：我很爱我的家庭，我的家庭一向很和睦，虽然我的父亲和母亲都是普通人，但是从小，我就看到我父亲起早贪黑，每天工作特别勤劳，他的行动无形中培养了我认真负责的态度和勤劳的精神。我母亲为人善良，对人热情，特别乐于助人，所以在单位人缘很好，她的一言一行也一直在教导我做人的道理。企业相信，和睦的家庭关系对一个人的成长有潜移默化的影响。

33、就你申请的这个职位，你认为你还欠缺什么？

> 企业喜欢问求职者弱点，但精明的求职者一般不直接回答。他们希望看到这样的求职者：继续重复自己的优势，然后说：“对于这个职位和我的能力来说，我相信自己是可以胜任的，只是缺乏经验，这个问题我想我可以进入公司以后以最短的时间来解决，我的学习能力很强，我相信可以很快融入公司的企业文化，进入工作状态。”企业喜欢能够巧妙地躲过难题的求职者。

34、你欣赏哪种性格的人？

> 诚实、不死板而且容易相处的人、有“实际行动”的人。

35、你通常如何处理别人的批评？

> ①沈默是金，不必说什么，否则情况更糟，不过我会接受建设性的批评。
>
> ②我会等大家冷静下来再讨论。

36、你为什么愿意到我们公司来工作？

> 对于这个问题，你要格外小心，如果你已经对该单位作了研究，你可以回答一些详细的原因，像“公司本身的高技术开发环境很吸引我。”、“我同公司出生在同样的时代，我希望能够进入一家与我共同成长的公司。”、“你们公司一直都稳定发展，在近几年来在市场上很有竞争力。”、“我认为贵公司能够给我提供一个与众不同的发展道路。”这都显示出你已经做了一些调查，也说明你对自己的未来有了较为具体的远景规划。

37、你和别人发生过争执吗？你是怎样解决的？

> 这是面试中最险恶的问题，其实是考官布下的一个陷阱，千万不要说任何人的过错，应知成功解决矛盾是一个协作团体中成员所必备的能力。假如你工作在一个服务行业，这个问题简直成了最重要的一个环节。你是否能获得这份工作，将取决于这个问题的回答。考官希望看到你是成熟且乐于奉献的。他们通过这个问题了解你的成熟度和处世能力。在没有外界干涉的情况下，通过妥协的方式来解决才是正确答案。

38、问题：你做过的哪件事最令自己感到骄傲？

这是考官给你的一个机会，让你展示自己把握命运的能力。这会体现你潜在的领导能力以及你被提升的可能性。假如你应聘于一个服务性质的单位，你很可能会被邀请去午餐。记住：你的前途取决于你的知识、你的社交能力和综合表现。

39、新到一个部门，一天一个客户来找你解决问题，你努力想让他满意，可是始终达不到群众得满意，他投诉你们部门工作效率低，你这个时候怎么作？

(1)首先，我会保持冷静。作为一名工作人员，在工作中遇到各种各样的问题是正常的，关键是如何认识它，积极应对，妥善处理。

(2)其次，我会反思一下客户不满意的原因。一是看是否是自己在解决问题上的确有考虑的不周到的地方，二是看是否是客户不太了解相关的服务规定而提出超出规定的要求，三是看是否是客户了解相关的规定，但是提出的要求不合理。

(3)再次，根据原因采取相对的对策。如果是自己确有不周到的地方，按照服务规定作出合理的安排，并向客户作出解释；如果是客户不太了解政策规定而造成的误解，我会向他作出进一步的解释，消除他的误会；如果是客户提出的要求不符合政策规定，我会明确地向他指出。

(4)再次，我会把整个事情的处理情况向领导作出说明，希望得到他的理解和支持。

(5)我不会因为客户投诉了我而丧失工作的热情和积极性，而会一如既往地牢记为客户服务的宗旨，争取早日做一名领导信任、公司放心、客户满意的职员。

40、对这项工作，你有哪些可预见的困难？

①不宜直接说出具体的困难，否则可能令对方怀疑应聘者不行。

②可以尝试迂回战术，说出应聘者对困难所持有的态度——工作中出现一些困难是正常的，也是难免的，但是只要有坚忍不拔的毅力、良好的合作精神以及事前周密而充分的准备，任何困难都是可以克服。

一般问这个问题，面试者的希望就比较大了，因为已经在谈工作细节，但常规思路中的回答，又被面官“骗”了。当面试官询问这个问题的时候，有两个目的。第一，看看应聘者是不是在行，说出的困难是不是在这个职位中一般都不可避免的问题。第二，是想看一下应聘者解决困难的手法对不对，及公司能否提供这样的资源。而不是想了解应聘者对困难的态度。

41、怎样对待自己的失败？

我们大家生来都不是十全十美的，我相信我有第二个机会改正我的错误。

42、什么会让你有成就感？

为贵公司竭力效劳，尽我所能，完成一个项目。

43、眼下你生活中最重要的是什么？

对我来说，能在这个领域找到工作是最重要的，能在贵公司任职对我说最重要。

44、与上级意见不一是，你将怎么办？

①一般可以这样回答“我会给上级以必要的解释和提醒，在这种情况下，我会服从上级的意见。”

②如果面试你的是总经理，而你所应聘的职位另有一位经理，且这位经理当时不在场，可以这样回答：“对于非原则性问题，我会服从上级的意见，对于涉及公司利益的重大问题，我希望能向更高层领导反映。”

> 分析：这个问题的标准答案是思路①，如果用②的回答，必死无疑。你没有摸清楚改公司的内部情况，先想打小报告，这样的人没有人敢要。

45、你工作经验欠缺，如何能胜任这项工作？

- ①如果招聘单位对应届毕业生的应聘者提出这个问题，说明招聘公司并不真正在乎“经验”，关键看应聘者怎样回答。
- ②对这个问题的回答最好要体现出应聘者的诚恳、机智、果敢及敬业。
- ③如“作为应届毕业生，在工作经验方面的确会有所欠缺，因此在读书期间我一直利用各种机会在这个行业里做兼职。我也发现，实际工作远比书本知识丰富、复杂。但我有较强的责任心、适应能力和学习能力，而且比较勤奋，所以在兼职中均能圆满完成各项工作，从中获取的经验也令我受益非浅。请贵公司放心，学校所学及兼职的工作经验使我一定能胜任这个职位。”
- 这个问题思路中的答案尚可，突出自己的吃苦能力和适应性以及学习能力（不是学习成绩）为好。

46、你希望与什么样的上级共事？

- ①通过应聘者对上级的“希望”可以判断出应聘者对自我要求的意识，这既上一个陷阱，又是一次机会。
- ②最好回避对上级具体的希望，多谈对自己的要求。
- ③如“做为刚步入社会的新人，我应该多要求自己尽快熟悉环境、适应环境，而不应该对环境提出什么要求，只要能发挥我的专长就可以了。
- 分析：这个问题比较好的回答是，希望我的上级能够在工作中对我多指导，对我工作中的错误能够立即指出。总之，从上级指导这个方面谈，不会有大的纰漏。

47、谈谈如何适应办公室工作的新环境？

- ①办公室里每个人有各自的岗位与职责，不得擅离岗位。
- ②根据领导指示和工作安排，制定工作计划，提前预备，并按计划完成。
- ③多请示并及时汇报，遇到不明白的要虚心请教。
- ④抓间隙时间，多学习，努力提高自己的政治素质和业务水平。

48、为了做好你工作份外之事，你该怎样获得他人的支持和帮助？

> 每个公司都在不断变化发展的过程中，你当然希望你的员工也是这样。你希望得到那些希望并欢迎变化的人，因为这些人明白，为了公司的发展，变化是公司日常生活中重要组成部分。这样的员工往往很容易适应公司的变化，并会对变化做出积极的响应。

49、如果你在这次面试中没有被录用，你怎么打算？

> 现在的社会是一个竞争的社会，从这次面试中也可看出这一点，有竞争就必然有优劣，有成功必定就会有失败。往往成功的背后有许多的困难和挫折，如果这次失败了也仅仅是一次而已，只有经过经验经历的积累才能塑造出一个完全的成功者。我会从以下几个方面来正确看待这次失败：①要敢于面对，面对这次失败不气馁，接受已经失去了这次机会就不会回头这个现实，从心理意志和精神上体现出对这次失败的抵抗力。要有自信，相信自己经历了这次之后经过努力一定能行，能够超越自我。②善于反思，对于这次面试经验要认真总结，思考剖析，能够从自身的角度找差距。正确对待自己，实事求是地评价自己，辩证的看待自己的长短得失，做一个明白人。③走出阴影，要克服这一次失败带给自己的心理压力，时刻牢记自己弱点，防患于未

> 然，加强学习，提高自身素质。④认真工作，回到原单位岗位上后，要实实在在、踏踏实实地工作，三十六行、行行出状元，争取在本岗位上做出一定的成绩。⑤再接再厉，成为国家公务员一直是我的梦想，以后如果有机会我仍然后再次参加竞争。

50、假如你晚上要去送一个出国的同学去机场，可单位临时有事非你办不可，你怎么办？

我觉得工作是第一位的，但朋友间的情谊也是不能偏废的，这个问题我觉得要按照当时具体的情况来决定。

- ①如果我的朋友晚上9点中的飞机，而我的加班八点就能够完成的话，那就最理想了，干完工作去机场，皆大欢喜。
- ②如果说工作不是很紧急，加班仅仅是为了明天上班的时候能把报告交到办公室，那完全可以跟领导打声招呼，先去机场然后回来加班，晚点睡就是了。
- ③如果工作很紧急，两者不可能兼顾的情况下，我觉得可以由两种选择。
  - （1）如果不是全单位都加班的话，是不是可以要其他同事来代替以下工作，自己去机场，哪怕就是代替你离开的那一会儿。
  - （2）如果连这一点都做不到的话，那只好忠义不能两全了，打电话给朋友解释一下，相信他会理解，毕竟工作做完了就完了，朋友还是可以再见面的。

51、谈谈你过去做过的成功案例？

> 举一个你最有把握的例子，把来龙去脉说清楚，而不要说了很多却没有重点。切忌夸大其词，把别人的功劳到说成自己的，很多主管为了确保要用的人是最适合的，会打电话向你的前一个主管征询对你的看法及意见，所以如果说谎，是很容易穿梆的。

52、谈谈你过去的工作经验中，最令你挫折的事情？

> 曾经接触过一个客户，原本就有耳闻他们以挑剔出名，所以事前的准备功夫做得十分充分，也投入了相当多的时间与精力，最后客户虽然并没有照单全收，但是接受的程度已经出乎我们意料之外了。原以为从此可以合作愉快，却得知客户最后因为预算关系选择了另一家代理商，之前的努力因而付诸流水。尽管如此，我还是从这次的经验学到很多，如对该产业的了解，整个team的默契也更好了。
>
> 分析：借此了解你对挫折的容忍度及调解方式。

53、如何安排自己的时间？会不会排斥加班？

> 基本上，如果上班工作有效率，工作量合理的话，应该不太需要加班。可是我也知道有时候很难避免加班，加上现在工作都采用责任制，所以我会调配自己的时间，全力配合。
>
> 分析：虽然不会有人心甘情愿的加班，但依旧要表现出高配合度的诚意。

54、为什么我们要在众多的面试者中选择你？

> 回答提示：根据我对贵公司的了解，以及我在这份工作上所累积的专业、经验及人脉，相信正是贵公司所找寻的人才。而我在工作态度、ＥＱ上，也有圆融、成熟的一面，和主管、同事都能合作愉快。
>
> 分析：别过度吹嘘自己的能力，或信口开河地乱开支票，例如一定会为该公司带来多少钱的业务等，这样很容易给人一种爱说大话、不切实际的感觉。

55、你并非毕业于名牌院校？

> 回答提示：是否毕业于名牌院校不重要，重要的是有能力完成您交给我的工作，我想我更适合贵公司这个职位。

56、怎样看待学历和能力？

> 回答提示：学历我想只要是大学专科的学历，就表明觉得我具备了根本的学习能力。剩下的，你是学士也好，还是博士也好，对于这一点的讨论，不是看你学了多少知识，而是看你在这个领域上发挥了什么，也就是所说的能力问题。一个人工作能力的高低直接决定其职场命运，而学历的高低只是进入一个企业的敲门砖，如果贵公司把学历卡在博士上，我就无法进入贵公司，当然这不一定只是我个人的损失，如果一个专科生都能完成的工作，您又何必非要招聘一位博士生呢？

57、工作中学习到了些什么？

> 回答提示：这是针对转职者提出的问题，建议此时可以配合面试工作的特点作为主要依据来回答，如业务工作需要与人沟通，便可举出之前工作与人沟通的例子，经历了哪些困难，学习到哪些经验，把握这些要点做陈述，就可以轻易过关了。

58、想过创业吗？

> 回答提示：这个问题可以显示你的冲劲，但如果你的回答是“有”的话，千万小心，下一个问题可能就是：那么为什么你不这样做呢？

59、除了本公司外，还应聘了哪些公司？

> 回答提示：很奇怪，这是相当多公司会问的问题，其用意是要概略知道应徵者的求职志向，所以这并非绝对是负面答案，就算不便说出公司名称，也应回答“销售同种产品的公司”，如果应聘的其他公司是不同业界，容易让人产生无法信任的感觉。

60、面试注意事项：

- 1，在面试官面前千万不要抖脚，手脚不要动来动去，不能有小动作。
- 2，在面试过程中，千万不要跟面试官去争论，说话太冲，太能说、抢话说、乱说都不好，遇到难题，先思考一下，切记心浮气燥，表达时口气温和，谦虚。
- 3，如果面试过程中都不错，谈的也很好，之后却没有给Offer，完全是自己意料之外的情况，这个很有可能，或许是因为公司有了其他的人选，不用介意，更不要沮丧。
- 4，在面试过程中，切忌问关于公司计划、行业机密等相关的东西，不要打探公司的内幕，机密敏感性的问题不要问东问西。
- 5，千万要注意仪容仪表，要有礼貌，最好不要有口吃，口头表达，逻辑思维很重要，不要让面试官觉得你很幼稚，太过小孩子气，显的不够稳重踏实。
- 6，在去面试之前，要熟悉自己的简历，特别是工作经历，准备好关于一些离职原因、职业规划方面的问题的回答方式。
- 7，在面试过程中，80%的面试官会让做自我介绍，所以提前要准备一下，说出的内容既要和简历相符，又要有重点有突出的地方，不能像背简历一样。
- 8，面试完后，如果等待的时间较长，没有回应，就可能没有什么希望了，自己可以打电话去了解情况。
- 9，在面试过程中，谈到薪资的时候，如果没有说明是税后工资就是税前，假如是税前6000，这里面就包括了公司给交的公积金，还有其他五险要交的费用，拿到手差不多4000左右。

61、投递简历注意事项

- 1，投简历的时间最好在早上8点多钟，因为人事9点多开始收简历，收到的简历又都是按时间来排序的，所以一般早上8点到9点投的都会排在前面，人事当天就能看到。
- 写简历：
- 1，现居住地最好能在企业附近，如果不在，只写北京，人事比较看重现居住地，如果你填写的现居地离公司远，基本上人事就不太愿意打电话了，因为她（他）怕你不会来。
- 2，期望薪资最好不要写在简历上，也不要填写在网上的简历上，具体薪资见面的时候再谈。
- 3，大部分的岗位招聘都有相应的硬性要求，比如：年龄、居住地点、工作年限等。如果自己不符合也没有关系，机会还有很多。
- 4，简历上的工作经历不能太多，否则人事会觉得你不够稳定。
- 5，开发工作，简历上的项目要抓住重点，放重要的技术点，不能千篇一律，简短、抓住中心。

62、入职后试用期：

- 1，到了公司之后，工作中不懂的地方要多问，跟同事搞好关系，多看看旁人在干什么～
- 2，有些同学被录用之后都还不知道自己的薪资待遇，也不知道公司有哪些福利，这是因为在面试过程中不敢多问。因此，如果在面试过程中，面试官已经比较明确的表达了想让你去上班后，你就可以在适当的时候问一些关于工作的情况，工资、福利待遇，上班时间，加班情况等等。