****1.一家超市的顾客数据，将数据可视化并分析销售额和年龄、收入的关系并给出营销建议****

年龄 收入 销售额

34 350 123

40 450 114

37 169 135

30 189 139

44 183 117

36 80 121

32 166 133

26 120 140

32 75 133

36 40 133

答案:

import matplotlib.pyplot as plt

# 年龄

age = [34,40,37,30,44,36,32,26,32,36]

# 收入

income = [350,450,169,189,183,80,166,120,75,40]

# 销售额

sales = [123,114,135,139,117,121,133,140,133,133]

# 年龄，销售额 散点图

plt.scatter(age, sales)

plt.show()

# 收入，销售额 散点图

plt.scatter(income, sales)

plt.show()

可视化结论：

顾客年龄和销售额负相关，年龄越大销售额越低

顾客收入和销售额也呈负相关，收入越高销售额越低

建议：

营销针对人群应是一般收入100-200之间，年龄20-37之间的人群，这样营销效果更好

****2.使用原生Python代码和Numpy并行计算分别求解下面的y值****

条件：

x1 = [1,2,3,4,5]

x2 = [3,5,2,9,10]

y = x12 + x23

求y值\*\*

原生Python实现

x1 = [1,2,3,4,5]

x2 = [3,5,2,9,10]

y = []

for i in range(len(x1)):

y.append(x1[i] \*\* 22222 + x2[i] \*\* 33333)

print(y)

Numpy实现

import numpy as np

x1 = np.array([1,2,3,4,5])

x2 = np.array([3,5,2,9,10])

y = x1 \*\* 22222 + x2 \*\* 33333

print(y)

****3.返回下表中所有同学语文成绩最低的1次考试成绩****

chinese english math name test

75 69 36 张三 一

68 85 87 李四 一

54 42 59 王五 一

55 57 63 李四 二

59 35 92 王五 二

45 63 92 王五 三

61 53 76 赵六 一\*\*

import pandas as pd

df = pd.read\_table('5.class.csv', encoding='gbk’)

def top(x, n=1, column='chinese'):

return x.sort\_values(by=column)[:1]

df.groupby('name').apply(top)

****4：创建三角级数类，阅读下列代码回答问题****

import numpy as np

import matplotlib.pyplot as plt

class trigonometric\_function:

tra\_1=6

tra\_2=12

def \_init\_(self,a,k,l):

self.const1=a

self.const2=k

self.const3=l

def Additive\_term(self,x):

self.summand\_s=np.sin(self.const2\*x+self.tra\_1)

self.summand\_c=np.cos(self.const3\*x+self.tra\_2)

def trigon\_sum(self,m,n):

sum\_1=sum([sum([self.const1\*(self.summand\_s\*\*i)\*(self.summand\_c\*\*j) for i in range(m) ]) for j in range(n)])

return sum\_1

def draw\_pic(self,data):

fig,axes\_1=plt.subplots(1,1,dpi=140,figsize=(8,6))

#x=np.linspace(-5,5,500)

axes\_1.plot(x,data)

plt.show()

a.请在代码中找出类变量、实例变量、方法、对象

#类变量：tra1,tra2,const1,const2,const3,summand\_s,summand\_c

#实例变量：x,a,k,l,m,n,data(通常前面没有self.。)

#方法（属性）： Additive\_term、trigonsum、drawpic

#对象：方法、类变量和实例变量

b.请写出程序中所涉及到的三角级数

c.self 什么意思？

self 代表类的实例，self 在定义类的方法时是必须有的，虽然在调用时不必传入相应的参数。

d.请阐述下列代码运行后会有什么结果

f1=trigonometric\_function()

#类的实例化

f1.\_init\_(3,4,6)

#对类进行初始化

x=np.linspace(-5,5,500)

#生成500个-5到5之间的点

f1.Additive\_term(x)

#生成普通三角函数sin和cos

data=f1.trigon\_sum(4,5)

#生成三角级数数据

f1.draw\_pic(data)

#绘图

****5.Python数据分析通常使用的环境、工具和库都有哪些？库功能是什么？****

基础环境一般为Anaconda，

编辑器多用Ipython或JupyterNotebook，

常用库有:

Numpy：数值计算

Matplotlib：数据可视化

Pandas：数据预处理和数据分析

scikit-learn：机器学习算法建模预测

****6.数据分析的一般流程是什么****

基本流程

提出问题

准备数据

分析数据

洞察结论

详细流程

需求层

目标确定

数据层

数据获取

数据规整（清洗）

分析层

描述性分析

指标计算

数据可视化

探索性分析

建模分析

模型验证

迭代优化

输出层

数据分析报告

总结结论

提出建议并实施

****7、如何理解过拟合？****

过拟合和欠拟合一样，都是数据挖掘的基本概念。过拟合指的就是数据训练得太好，在实际的测试环境中可能会产生错误，所以适当的剪枝对数据挖掘算法来说也是很重要的。

欠拟合则是指机器学习得不充分，数据样本太少，不足以让机器形成自我认知。

****8、为什么说朴素贝叶斯是“朴素”的？****

朴素贝叶斯是一种简单但极为强大的预测建模算法。之所以称为朴素贝叶斯，是因为它假设每个输入变量是独立的。这是一个强硬的假设，实际情况并不一定，但是这项技术对于绝大部分的复杂问题仍然非常有效。

****9、SVM 最重要的思想是什么？****

SVM 计算的过程就是帮我们找到超平面的过程，它有个核心的概念叫：分类间隔。SVM 的目标就是找出所有分类间隔中最大的那个值对应的超平面。在数学上，这是一个凸优化问题。同样我们根据数据是否线性可分，把 SVM 分成硬间隔 SVM、软间隔 SVM 和非线性 SVM。

****10、K-Means 和 KNN 算法的区别是什么？****

首先，这两个算法解决的是数据挖掘中的两类问题。K-Means 是聚类算法，KNN 是分类算法。其次，这两个算法分别是两种不同的学习方式。K-Means 是非监督学习，也就是不需要事先给出分类标签，而 KNN 是有监督学习，需要我们给出训练数据的分类标识。最后，K 值的含义不同。K-Means 中的 K 值代表 K 类。KNN 中的 K 值代表 K 个最接近的邻居。

****11.python下多线程的限制以及多进程中传递参数的方式？****

python多线程有个全局解释器锁（global interpreter lock），这个锁的意思是任一时间只能有一个线程运用解释器，跟单cpu跑多个程序一个意思，我们都是轮着用的，这叫“并发”，不是“并行”。

多进程间同享数据，能够运用 multiprocessing.Value 和 multiprocessing.Array

****12.Python是怎么进行内存办理的？****

Python引用了一个内存池(memory pool)机制，即Pymalloc机制(malloc:n.分配内存)，用于办理对小块内存的请求和释放

内存池（memory pool）的概念：

当 创立很多耗费小内存的目标时，频频调用new/malloc会导致很多的内存碎片，致使功率下降。内存池的概念就是预先在内存中请求必定数量的，巨细相等 的内存块留作备用，当有新的内存需求时，就先从内存池中分配内存给这个需求，不够了之后再请求新的内存。这样做最显著的优势就是能够削减内存碎片，提高功率。

内存池的完成方式有很多，性能和适用范围也不相同。

python中的内存办理机制——Pymalloc：

python中的内存办理机制都有两套完成，一套是针对小目标，就是巨细小于256bits时,pymalloc会在内存池中请求内存空间；当大于256bits，则会直接履行new/malloc的行为来请求内存空间。 关于释放内存方面，当一个目标的引用计数变为0时，python就会调用它的析构函数。在析构时，也采用了内存池机制，从内存池来的内存会被归还到内存池中，以防止频频地释放动作。

Python初学者有什么不懂的可以私信我——我刚整理了一套2018最新的0基础入门和进阶教程，无私分享，获取方法：加q u n 227-435-450 即可获取，内附：开发工具和安装包，以及系统学习路线图。

****13.什么是lambda函数？它有什么好处?****

lambda 函数是一个能够接纳任意多个参数(包括可选参数)并且返回单个表达式值的函数。 lambda 函数不能包括指令，它们所包括的表达式不能超过一个。不要试图向lambda 函数中塞入太多的东西；假如你需要更复杂的东西，应该定义一个一般函数，然后想让它多长就多长。

****14.怎么用Python输出一个Fibonacci数列？****

1 a,b = 0, 1

2 while b

3 print (b),

4 a, b = b, a+b

****15.介绍一下Python中webbrowser的用法？****

webbrowser模块提供了一个高级接口来显现基于Web的文档，大部分情况下只需要简单的调用open()办法。

webbrowser定义了如下的反常：

exception webbrowser.Error, 当浏览器控件发作过错是会抛出这个反常

webbrowser有以下办法：

webbrowser.open(url[, new=0[, autoraise=1]])

这个办法是在默许的浏览器中显现url, 假如new = 0, 那么url会在同一个浏览器窗口下打开，假如new = 1, 会打开一个新的窗口，假如new = 2, 会打开一个新的tab, 假如autoraise ＝ true, 窗口会主动增加。

webbrowser.open\_new(url)

在默许浏览器中打开一个新的窗口来显现url, 否则，在仅有的浏览器窗口中打开url

webbrowser.open\_new\_tab(url)

在默许浏览器中当开一个新的tab来显现url, 否则跟open\_new()相同

webbrowser.get([name]) 依据name返回一个浏览器目标，假如name为空，则返回默许的浏览器

webbrowser.register(name, construtor[, instance])

注册一个名字为name的浏览器，假如这个浏览器类型被注册就能够用get()办法来获取。

****16.解释一下python的and-or语法****

与C表达式 bool ? a : b相似，但是bool and a or b，当 a 为假时，不会象C表达式 bool ? a : b 相同作业

应该将 and-or 技巧封装成一个函数：

defchoose(bool, a, b):return(booland[a]or[b])[0]

由于 [a] 是一个非空列表，它永久不会为假。甚至 a 是 0 或 '' 或其它假值，列表[a]为真，由于它有一个元素。

****17.how do I iterate over a sequence in reverse order？****

forxinreversed(sequence):

...#dosomethingwithx..

假如不是list, 最通用但是稍慢的解决方案是：

foriinrange(len(sequence)-1, -1, -1):

****x=sequence[i]****

1. ****Python是怎么进行类型变换的？****

函数 描述

Int(x [,base ]) 将x转换为一个整数

Long(x [,base ]) 将x转换为一个长整数

Float(x) 将x转换到一个浮点数

Complex(real [,imag]) 创建一个复数

Str(x) 将x转换为字符串

Repr(x) 将x转换为表达式字符串

Eval(str) 用来计算在字符串中有效python表达式，并返回一个对象

Tuple(s ) 将序列s转换为一个元组

List(s ) 将序列s转换为一个列表

Chr(x) 将一个整数转换为一个字符

Unichr(x) 将一个整数转换为unicode字符

Ord (x) 将一个字符转换为它的整数值

Hex(x) 将一个整数转换为一个十六进制字符串

Oct(x) 将一个整数转换为一个十六进制字符串

1. ****Python里面如何实现tuple和list的转换？****

**l=tuple(iplist)**

**print(l)**

**t=list(l)**

**print(t)**

1. ****请写出一段Python代码实现删除一个list里面的重复元素？****

**l=[1,1,2,3,4,5,6,7,4]**

**print(list(set(l)))**

**Or**

**d={}**

**For x in mylist:**

**d[x]=1**

**mylist=list(d.keys())**

**21.比较Java和Python**

|  |  |  |
| --- | --- | --- |
|  | | |
| 标准 | Java | 蟒蛇 |
| 便于使用 | 好 | 很好 |
| 编码速度 | 平均 | 优秀 |
| 数据类型 | 静态打字 | 动态输入 |
| 数据科学和机器学习应用 | 平均 | 很好 |

**22.什么是Python？**

Python是一种高级，解释，交互式和面向对象的脚本语言。Python的设计具有高可读性。它经常使用英语关键词，而其他语言使用标点符号，并且它的语法结构比其他语言少。

**23.PYTHONPATH环境变量的目的是什么？**

PYTHONPATH - 它的作用类似于PATH。此变量告诉Python解释器在何处找到导入程序的模块文件。它应该包括Python源库目录和包含Python源代码的目录。PYTHONPATH有时由Python安装程序预设

**24.PYTHONSTARTUP，PYTHONCASEOK，PYTHONHOME，PYTHONSTARTUP环境变量的目的是什么？**

PYTHONSTARTUP - 它包含包含Python源代码的初始化文件的路径。每次启动解释器时都会执行它。它在Unix中命名为.pythonrc.py，它包含加载实用程序或修改PYTHONPATH的命令

PYTHONCASEOK - 在Windows中用于指示Python在import语句中查找第一个不区分大小写的匹配项。将此变量设置为任何值以激活它。

PYTHONHOME - 它是另一种模块搜索路径。它通常嵌入在PYTHONSTARTUP或PYTHONPATH目录中，以便于切换模块库。

**25.Python中支持的数据类型是什么？**

Python有五种标准数据类型 -

* 数字
* 串
* 名单
* 元组
* 字典

**26.列表和元组有什么区别？**

|  |  |
| --- | --- |
| **LIST** | **TUPLES** |
| 列表是可变的，即可以编辑它们。 | 元组是不可变的（元组是无法编辑的列表）。 |
| 列表比元组慢。 | 元组比列表更快。 |
| 语法：list\_1 = [10，'Chelsea'，20] | 语法：tup\_1 =（10，'Chelsea'，20） |

**27.如何在Python中管理内存？**

1. Python内存由Python私有堆空间管理。所有Python对象和数据结构都位于私有堆中。程序员无权访问此私有堆，解释器负责处理此私有堆。
2. Python对象的Python堆空间分配由Python内存管理器完成。核心API提供了一些程序员编写代码的工具。
3. Python还有一个内置的垃圾收集器，它可以回收所有未使用的内存并释放内存并使其可用于堆空间。

**28.用一个例子解释Python中的继承。**

继承允许一个类获得另一个类的所有成员（比如属性和方法）。继承提供代码可重用性，使创建和维护应用程序变得更容易。我们继承的类称为超类，继承的类称为派生/子类。

它们是Python支持的不同类型的继承：

1. 单一继承 - 派生类获取单个超类的成员。
2. 多级继承 - 从基类base1继承的派生类d1，d2继承自base2。
3. 分层继承 - 从一个基类可以继承任意数量的子类
4. 多重继承 - 派生类从多个基类继承。

**29.每当Python退出时，为什么不是所有的内存都被解除分配？**

1. 每当Python退出时，尤其是那些对其他对象具有循环引用的Python模块或者从全局名称空间引用的对象并不总是被解除分配或释放。
2. 无法解除分配C库保留的那些内存部分。
3. 退出时，由于拥有自己的高效清理机制，Python会尝试取消分配/销毁其他所有对象。

**30.Python中的字典是什么？**

Python中的内置数据类型称为字典。它定义了键和值之间的一对一关系。字典包含一对键及其对应的值。字典由键索引。

我们来举个例子：

以下示例包含一些键。国家，资本和下午。它们的相应值分别是印度，德里和莫迪。

字典= { '国家'： '印度'， '资本'： '德里'， 'PM'： '莫迪'}

打印字典[国家]

**31.写一个单行，用于计算文件中大写字母的数量。即使文件太大而无法放入内存，您的代码也应该可以正常工作。**

我们先写一个多行解决方案，然后将其转换为一个班轮代码。

1打开（SOME\_LARGE\_FILE）为fh：

2 count = 0  
3 text = fh.read（）  
4表示文本中的字符：  
5 if character.isupper（）：  
6 count + = 1

**32.在Python中为数值数据集编写排序算法。**

以下代码可用于在Python中对列表进行排序：

list = [“1”，“4”，“0”，“6”，“9”]

list = [int（i）for i in list]

list.sort（）

打印（列表）

**33.你将如何扭转名单？**

list.reverse（） - 反转列表对象。

**34.如何从列表中删除最后一个对象？**

list.pop（obj = list [-1]） - 从列表中删除并返回最后一个对象或obj。

**35.什么是负指数，为什么使用它们？**

Python中的序列是索引的，它由正数和负数组成。积极的数字使用'0'作为第一个索引，'1'作为第二个索引，过程继续进行。

负数的索引从'-1'开始，表示序列中的最后一个索引，' - 2'作为倒数第二个索引，序列像正数一样前进。

负索引用于从字符串中删除任何换行符，并允许该字符串除了作为S [： - 1]给出的最后一个字符。负索引还用于显示索引以正确的顺序表示字符串。

**36.解释Python中“re”模块的split（），sub（），subn（）方法。**

为了修改字符串，Python的“re”模块提供了3种方法。他们是：

* split（） - 使用正则表达式模式将给定字符串“拆分”到列表中。
* sub（） - 查找正则表达式模式匹配的所有子字符串，然后用不同的字符串替换它们
* subn（） - 它类似于sub（），并且还返回新字符串和no。替换。

**37.range和xrange有什么区别？**

在大多数情况下，xrange和range在功能方面完全相同。它们都提供了一种生成整数列表供您使用的方法，但是您可以随意使用。唯一的区别是range返回一个Python列表对象，x range返回一个xrange对象。

这意味着xrange实际上并不像运行时那样在运行时生成静态列表。它使用称为yielding的特殊技术根据需要创建值。该技术与一种称为生成器的对象一起使用。这意味着，如果你有一个非常巨大的范围，你想生成一个列表，比如10亿，xrange就是要使用的功能。

如果你有一个真正的内存敏感系统，例如你正在使用的手机，尤其如此，因为范围将使用尽可能多的内存来创建整数数组，这可能导致内存错误并导致崩溃程序。这是一个记忆饥饿的野兽。

**38.什么是酸洗和去除？**

Pickle模块接受任何Python对象并将其转换为字符串表示形式，并使用dump函数将其转储到文件中，此过程称为pickling。从存储的字符串表示中检索原始Python对象的过程称为unpickling。

**39.Python中的map函数是什么？**

map函数执行作为第一个参数给出的函数，作为第二个参数给出的iterable的所有元素。如果给定的函数包含多于1个参数，则给出了许多迭代。#Tollow链接以了解更多类似的功能

**40.如何在NumPy数组中获得N个最大值的索引？**

我们可以使用以下代码获取NumPy数组中N个最大值的索引：  
import numpy as np

arr = np.array（[1,3,2,4,5]）

打印（arr.argsort（）[ - 3：] [:: - 1]）

**41.什么是Python模块？**

模块是一个Python脚本，通常包含import语句，函数，类和变量定义，以及Python可运行代码，它“生活”文件的扩展名为“.py”。zip文件和DLL文件也可以是模块。在模块中，您可以将模块名称称为存储在全局变量名称中的字符串。

**42.在Python中命名与文件相关的模块？**

Python为库/模块提供了一些函数，使您可以在文件系统上操作文本文件和二进制文件。使用它们，您可以创建文件，更新其内容，复制和删除文件。这些库是：os，os.path和shutil。

这里，os和os.path - 模块包括用于访问文件系统的函数

shutil - module使您可以复制和删除文件。

**43.解释使用with声明？**

在python中，通常“with”语句用于打开文件，处理文件中存在的数据，还可以在不调用close（）方法的情况下关闭文件。“with”语句通过提供清理活动使异常处理更简单。

一般形式：

使用open（“filename”，“mode”）作为file-var：

处理声明

注意：无需通过在file-var.close（）上调用close（）来关闭文件

**44.解释Python支持的所有文件处理模式？**

Python允许您以三种模式之一打开文件。他们是：

通过分别指定标志“r”，“w”，“rw”，“a”，只读模式，只写模式，读写模式和附加模式。

通过指定选项“t”，可以在上述任何一种模式中打开文本文件

“r”，“w”，“rw”和“a”，以便前面的模式变为“rt”，“wt”，“rwt”和“at”。二进制文件可以在任何一个中打开上述模式通过指定选项“b”以及“r”，“w”，“rw”和“a”使得前面的模式变为“rb”，“wb”，“rwb”，“ab” 。

**45.Python支持多少种序列？这些是什么？**

Python支持7种序列类型。它们是str，list，tuple，unicode，byte array，xrange和buffer。其中xrange在python 3.5.X中已弃用。

**46.你如何在Python中执行模式匹配？说明**

正则表达式/ RE /正则表达式使我们能够指定可以匹配给定字符串的特定“部分”的表达式。例如，我们可以定义一个正则表达式来匹配单个字符或数字，电话号码或电子邮件地址等.Python的“re”模块提供正则表达式模式，并从Python 2.5的更高版本中引入。“re”模块提供搜索文本字符串的方法，或者替换文本字符串以及基于定义的模式拆分文本字符串的方法。

**47.如何以相反的顺序显示文本文件的内容？**

1. 将给定文件转换为列表。
2. 使用reversed（）反转列表
3. 例如：for reverse in reverse（list（open（“file-name”，“r”）））：
4. 打印（行）

**48.NumPy和SciPy有什么区别？**

1. 在理想的世界中，NumPy只包含数组数据类型和最基本的操作：索引，排序，整形，基本元素智能函数等。
2. 所有数字代码都将驻留在SciPy中。但是，NumPy的一个重要目标是兼容性，因此NumPy试图保留其前任任何一个支持的所有功能。
3. 因此，NumPy包含一些线性代数函数，即使它们更恰当地属于SciPy。无论如何，SciPy包含更多全功能版本的线性代数模块，以及许多其他数值算法。
4. 如果你使用python进行科学计算，你应该安装NumPy和SciPy。大多数新功能属于SciPy而非NumPy。

**49.以下哪项是无效陈述？**

#### ****a）abc = 1,000,000****

#### ****b）abc = 1000 2000 3000****

#### ****c）a，b，c = 1000,2000,3000****

#### ****d）a\_b\_c = 1,000,000****

答案：b

**50.以下产出是什么？尝试：如果'1'！= 1：加注**

#### ****a）发生了一些错误****

#### ****b）没有发生一些错误****

#### ****c）无效代码****

#### ****d）以上都不是****

答案：C

**51.假设list1是[2,33,222,14,25]，什么是list1 [-1]？**

25

**52.要打开文件c：\ scores.txt进行写入？**

fileWriter = open（“c：\\ scores.txt”，“w”）

**53.为统计，数值和科学计算命名一些Python模块？**

numPy - 此模块提供数组/矩阵类型，它对在数组上进行计算很有用。scipy - 这个模块提供了进行数值积分，求解微分方程等的方法.pylab是一个用于生成和保存图的模块

**54.什么是TkInter？**

TkInter是Python库。它是GUI开发的工具包。它为GUI应用程序中使用的各种GUI工具或小部件（如按钮，标签，文本框，单选按钮等）提供支持。它们的共同属性包括尺寸，颜色，字体，光标等。

**55.Python面向对象吗？什么是面向对象编程？**

是。Python是面向对象的编程语言。OOP是基于类和被称为对象的类的实例的编程范例。OOP的特点是：

封装，数据抽象，继承，多态。

**56.什么是多线程？举个例子。**

这意味着通过调用多个线程同时运行多个不同的程序。进程中的多个线程使用主线程引用数据空间，并且它们可以相互通信以更容易地共享信息。线程是轻量级进程并且具有较少的内存开销。线程可以仅用于快速任务，如计算结果，也可以在主程序运行时在后台运行其他进程。

**57.Python是否支持Java中的接口？讨论。**

Python不提供Java中的接口。抽象基类（ABC）及其功能由Python的“abc”模块提供。抽象基类是一种机制，用于指定其实现子类必须实现哪些方法。ABC'c的使用提供了一种关于方法及其预期行为的“理解”。该模块从Python 2.7版本开始提供。

**58.什么是Accessor，mutators，@ property？**

访问器和变换器通常被称为“Java”之类的语言中的getter和setter。例如，如果x是用户定义的类的属性，则该类将具有名为setX（）和getX（）的方法。Python有一个@property“装饰器”，允许你使用ad getters和setter来访问类的属性。

**59.区分append（）和extend（）方法。**

append（）和extend（）方法都是list的方法。这些方法用于添加列表末尾的元素。

append（element） - 在调用此方法的列表末尾添加给定元素。

extend（another-list） - 在列表的末尾添加另一个列表的元素，称为extend方法。

**60.列举一些用于在Python中实现面向功能的编程的方法？**

Python支持方法（在Python3中称为迭代器），例如filter（），map（）和reduce（），当您需要迭代列表中的项目，创建字典或提取子集时，它们非常有用。一个列表。

filter（） - 使您能够基于条件逻辑提取值的子集。

map（） - 它是一个内置函数，它将函数应用于iterable中的每个项目。

reduce（） - 重复执行序列的成对减少，直到计算出单个值。

**61.以下产出是什么？**

**x = ['ab'，'cd']   
print（len（map（list，x）））**

由于map没有len（），因此发生TypeError。

**62.以下内容的输出是什么？**

**x = ['ab'，'cd']   
print（len（list（map（list，x））））**

说明：每个字符串的长度为2。

**63.以下哪项不是创建集合的正确语法？**

1. **a）设置（[[1,2]，[3,4]]）**
2. **b）设置（[1,2,2,3,4]）**
3. **c）设置（（1,2,3,4））**
4. **d）{1,2,3,4}**

答案：a

Explanation：为set设置的参数必须是可迭代的。

**64.解释一些在Python中实现面向功能的编程的方法。**

有时，当我们想要遍历列表时，一些方法会派上用场。

1. 过滤（）

过滤器允许我们根据条件逻辑过滤一些值。

>>> list（filter（lambda x：x> 5，range（8）））

[6,7]

1. 地图（）

Map将函数应用于iterable中的每个元素。

>>> list（map（lambda x：x \*\* 2，range（8）））

[0,1,4,9,16,25,36,49]

1. 降低（）

在我们达到单个值之前，Reduce会反复减少序列顺序

>>>来自functools import reduce

>>> reduce（lambda x，y：xy，[1,2,3,4,5]）

-13

**65.解释Python Flask中的数据库连接？**

Flask支持数据库驱动的应用程序（RDBS）。这样的系统需要创建一个模式，这需要将shema.sql文件传递给sqlite3命令。因此，您需要安装sqlite3命令才能在Flask中创建或启动数据库。

Flask允许以三种方式请求数据库

* before\_request（）：在请求之前调用它们并且不传递任何参数
* after\_request（）：在请求后调用它们并传递将发送到客户端的响应
* teardown\_request（）：在引发异常时调用它们，并且不保证响应。在构建响应之后调用它们。不允许它们修改请求，并忽略它们的值。

**66.编写一个Python函数来检查传递的字符串是否是回文结构？注意：回文是一个单词，短语或序列，向前读取相同的内容，例如，女士或护士。**

def isPalindrome（string）：  
left\_pos = 0  
right\_pos = len（string） - 1

而right\_pos> = left\_pos：  
if if string [left\_pos] == string [right\_pos]：  
return False  
left\_pos + = 1  
right\_pos - = 1  
return  
print print（isPalindrome（'aza'））

**67.编写一个Python程序来计算数字列表的总和。**

def list\_sum（num\_List）：  
如果len（num\_List）== 1：  
return num\_List [0]  
else：  
return num\_List [0] + list\_sum（num\_List [1：]）

print（list\_sum（[2,4,5,6,7]））

样本输出：

24

**68.如何通过Python代码从MySQL数据库中的表中检索数据？说明。**

1. 将MySQLdb模块导入为：import MySQLdb
2. 建立与数据库的连接。
3. db = MySQLdb.connect（“host”=“local host”，“database-user”=“user-name”，“password”=“password”，“database-name”=“database”）
4. 在建立的连接上初始化游标变量：c1 = db.cursor（）
5. 通过定义所需的查询字符串来检索信息。s =“选择\*来自dept”
6. 使用fetch（）方法获取数据并打印它。data = c1.fetch（s）
7. 关闭数据库连接。close（）

**69.编写一个Python程序来读取文件中的随机行。**

import random  
def random\_line（fname）：  
lines = open（fname）.read（）。splitlines（）  
return random.choice（lines）  
print（random\_line（'test.txt'））

**70.编写一个Python程序来计算文本文件中的行数。**

def file\_lengthy（fname）：  
open（fname）为f：  
for i，l in enumerate（f）：  
pass  
return i + 1  
print（“file of lines：”，file\_lengthy（“test.txt”））

**71.Python的数据类型有哪些？**

int（整数），float（浮点数），string（字符串），dict（字典），tuple（元组），list（列表），data（日期），set（集合），bool（布尔类型），数字

**72.他们的区别有哪些？**

string，tuple：可迭代、不可改变、可索引。

tuple和list：tuple不可变；都可迭代。

tuple、string、数字：不可变。

int、float：不可改变。

list：可迭代、可索引、可改变。

set和dict：set没有对应的value值；都是可变的；内部元素都不可变；都无索引。

set和list：都是可变类型；set是无序且元素唯一；set获取元素只能for循环；list是有序且元素不唯一，可以根据索引进行切片。

list和dict：都是可变类型，都是可迭代；dict的key是不可变对象；dict需要储存空间大于list列表；dict查询效率远高于列表（在我看来，就像key是加了一个索引）。

**73.知道MySQL引擎吗？说一下。说一下你知道数据库引擎类型有哪些？**

**（介绍4种）**

- MyISAM存储引擎：不支持事务、不支持外键，优点是访问速度快，对事物完整性没有要求，以select（查询），insert（插入）为主的应用基本可以用这个引擎来创建表。支持3种不同的存储格式，分别是：静态表、动态表、压缩表。

- innoDB存储引擎：提供了具有提交、回滚和崩溃恢复能力的事务安全。但比MyISAM引擎，写的处理效率会差一些，并且会占用更多的磁盘空间以保留数据和索引。有点：支持自动增长列，支持外键约束。

- MEMORY引存储擎：使用存在于内存中的内容来创建表。每个memory表实际对应一个磁盘文件，格式是.frm。优点：memory表的访问速度非常快（因为数据在内存中，并且默认使用hash索引）缺点：一旦服务关闭，表中的数据就会丢失掉。

- MERGE存储引擎：Merge存储引擎是一组MyISAM表的组合，这些MyISAM报必须结构完全相同，merge表本身并没有数据，对merge类型的表可以进行查询，更新，删除操作，这些操作实际上是对内部的MyISAM表进行的。

**74.说一下B+树和B树的区别。**

- B+树中只有叶子节点会带有指向记录的指针，而B树则所有节点都带有，在内部节点出现的索引项不会再出现在叶子节点中。

- B+树中所有叶子节点都是通过指针连接在一起，而B树不会。

- B+树的优点：1.非叶子节点不会带上ROWID，这样，一个快中可以容纳更多的索引项，一是可以降低树的高度。二是一个内部节点可以定位更多的子叶节点。

2.子叶节点之间通过指针来连接，范围扫面将十分简单，而对于B书来说，则需要在子叶节点和内部节点不停的往返移动。

- B树的优点：对于在内部节点的数据可直接得到，不必根据子叶节点来定位。

**75.说一下MySQL优化。**

- SQL语句优化

- 避免select \* ，将需要查找的字段列出来

- 使用连接（join）来代替子查询

- 拆分大的delete或insert语句

- 使用limit对查询结果的记录进行限定

- 用exists代替in是一个好的选择

- 用Where子句替换having子句，因为having只会在检索出所有记录之后才对结果集进行过滤。

- 表结构优化

- 选择正确的存储引擎

- 使用可存下数据的最小的数据类型，整型 < date,time < char,varchar<blob

- 使用简单的数据类型，整型比字符处理开销更小，因为字符串的比较更复杂。如int类型存储时间类型，bigint类型转ip函数

- 使用合理的字段属性长度，固定长度的表会更快

等等。

**76. 说一下MySQL和mongdb的区别。**

- MongoDB（文档型数据库、非关系型数据库）：提供可扩展的高性能数据存储。

存储方式：虚拟内存+持久化。查询语句：独特的MongoDB的查询方式。适合场景：事件的记录，内容管理或博客平台等等。架构特点：可以通过副本集，以及分片来实现高可用。数据处理：数据是存储在硬盘上的，只不过需要经常读取的数据会被加载到内存中，将数据存储在物理内存中，从而达到高速度读写。优势：适量级的内存的MongoDB的性能是非常迅速的、高扩展、自身的Failover机制、json的存储格式。缺点：sql语句与数据结构不如mysql那么亲和。不支持事物操作、占用空间大

- MySQL（关系型数据库）：在不同的引擎上有不同的存储方式。查询语句是使用传统的sql语句。开源数据库的份额在不断增加。缺点：在海量数据处理的时候效率会显著变慢。

**77. 说一下Python异步函数。**

python3.4+ 加了内置异步库asyncio。它是用来编写并发代码的库，使用async/await语法。用于：

- 并发地运行Python协程，并对其执行过程实现完全控制

- 执行网络IO和IPC

- 控制子进程

- 通过队列实现分布任务

**78. 事务隔离级别有哪些？**

一般的数据库，都包含：读末提交、读提交、可重复读、串行化

**79. mysql左连接查询和右连接查询的区别。**

左连接查询，以左表为参照，显示所有数据；右连接查询以游标为参照，显示所有输出。

**80. 你的这个项目的具体需求是什么？**

这个就看你是什么项目了。

**81. 出题：a={'a':1,'b':2,'c':3},b=c\_class()**

现在不知道b的属性有哪些，写一个程序通过a的键得到b的属性，若有则输出a中键对应的值，若无，不输出，也不能为b创建属性。1分钟

**82. 出题：写一个老师，学生，教室，课程的关系。3分钟**

老师和学生多对多，中间表是课程表，教室和课程是一对一关系。

**83. python基础：**

- list乘法：a\_list\*2 == a\_list+a\_list

- 匿名函数：f = lambda arg1,arg2:arg1+arg2 ， 冒号之前的arg1，arg2代表这个匿名函数的参数。匿名函数不需要return来返回值，表达式本身就是返回值。

- map：会根据提供的函数对指定序列做映射。map(function,iterable1,iterable2,...)

第一个参数function以参数序列中每一个元素调用function函数，返回包含每次function函数返回值的迭代器。（python2.x返回列表，python3.x返回迭代器）

- 类中的\_\_dict\_\_属性：\_\_dict\_\_属性是一个字典，它包含了该对象所有的属性。如：class f：a=5 ； f.\_\_dict\_\_ 字典里面就会包含‘a’：5。

**84. 写一个冒泡排序：**

```python

def bubble(nums):

for i in range(len(nums)-1):

for k in range(len(nums)-i-1):

if nums[k] > nums[k+1]:

nums[k],nums[k+1] = nums[k+1],nums[k]

return nums

a = [1,7,3,7,8,2,23,6,32,4,23]

print(bubble(a))

# [1, 2, 3, 4, 6, 7, 7, 8, 23, 23, 32]

```

如果是已经排序好的列表怎么处理：

```python

if sorted(nums)==nums:

break

```

**85. 求最大升序列表（[2,5,3,7,6,9],得出[2,5,7,9]）**

```python

def max\_p(nums):

nums\_list = []

for i in range(len(nums)-1):

if nums\_list:

if nums\_list[-1] < nums[i]:

nums\_list.append(nums[i])

else:

nums\_list.append(nums[i])

return nums\_list

a = [1,7,3,7,8,2,23,6,32,4,23]

print(max\_p(a))

# [1, 7, 8, 23, 32]

```

**86. 说一下redis。**

redis是一个（开源）高性能的key-value数据库。它可以用作数据库、缓存和消息中间件，它支持存储的value类型相对memcached数据库更多，包括string、list、set、zset和hash（哈希类型）

22. 说一下Docker。

Docker是一个基于Go语言实现的开源应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到服务器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。最重要的是，他们不依赖任何语言、框架包括系统。

**87. 说一下HTTP和HTTPS的区别。**

- HTTP：超文本传输协议，是一个基于请求与响应，无状态，应用层的协议，常基于TCP/IP协议传输数据，互联网上最为广泛的一种网络协议，所有的WWW文件都必须遵守这个标准。设计HTTP的初衷是为了提供一种发布和接受HTML页面的方法。特点：无状态、无连接、基于请求和响应、简单快速、灵活；无法保护数据的完整性、协议传输数据以明文形式显示。

- HTTPS：一种通过计算机网络进行安全通信的传输协议，经由HTTP进行通信，利用SSL/TLS建立全信道，加密数据包。HTTPS使用的主要目的是提供对网站服务器的身份认证，同时保护交换数据的隐私与完整性。特点：内容加密、验证身份、保护数据完整性；加密范围比较有限、SSL证书的信用链体系并不安全，特别是某些国家可以控制CA根证书的情况下，中间人攻击一样可行、成本高。

**88. 说一下三次握手和四次挥手。**

- 为什么需要三次握手呢？为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。

比如：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段，但是server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求，于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了，由于client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据，但server却以为新的运输连接已经建立，并一直等待client发来数据。所以没有采用“三次握手”，这种情况下server的很多资源就白白浪费掉了。

- 为什么需要四次挥手呢？TCP是全双工模式。通信双方都可以独立关闭自己的通信通道，也就是半关闭。

比如：client先发送FIN告知对方我已经完成数据发送了，server回复ack来确定我知道了。这样一个流程，就关闭了client的发送信息通道。但是还可以接收来自server方的数据。server此时已经知道接收不到client的数据了，但是还可以给它发送数据。如果server也没有啥数据要发送给对方了，server也会以FIN标志位发送一个信息给client，client接到后，也会传递一个ack表示知道了。这样子，双方都完成了关闭。

**89. 说一下ajax：**

ajax是异步的JavaScript和XML，优点是无刷新的与后端通信并获取数据，只更改需要更改的部分，而不是刷新整个页面，减少无意义的数据传输，提高了效率和用户体验。

**90. 高并发如何处理。**

- HTML页面静态化。

- 图片服务器分离（可以用fastdfs轻量级的分布式文件存储系统）。

- 使用缓存（用redis）。

- 数据库集群、库表散列。

- 使用负载均衡的方法（配置nginx服务器）。

- 镜像。

- CDN加速技术（内容分发网络）。

**91. 说说乐观锁及你的观点。**

乐观锁其实并不是锁。通过SQL的where子句中的条件是否满足来判断是否满足更新条件来更新数据库，通过受影响行数判断是否更新成功，如果更新失败可以再次进行尝试，如果多次尝试失败就返回更新失败的结果。

在并发比较少时建议使用乐观锁,减少加锁、释放锁的开销。在并发比较高的时候，建议使用悲观锁。mysql默认隔离级别可重复读，使用乐观锁的话需要改成读已提交。

如果乐观锁多次尝试的代价比较大，也建议使用悲观锁。

使用乐观锁方式进行解决，成本低，非阻塞，性能较高。

悲观锁，传统的关系型数据库里边就用到了很多这种锁机制，如mysql, 每次查询、修改数据操作之前就会上锁，整个数据处理过程中，将数据处于锁定状态，自己更新数据后再释放锁，可以有效防止库存冲突问题。

**92.@staticmethod和@classmethod区别。**

Python其实有3个方法,即静态方法(staticmethod),类方法(classmethod)和实例方法，对于静态方法其实和普通的方法一样,不需要对谁进行绑定,唯一的区别是调用的时候需要使用a.static\_foo(x)或者A.static\_foo(x)来调用。

**93.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的。

当 Python 遇到一个变量的话他会按照这样的顺序进行搜索：

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）。

**94. GIL线程全局锁**

线程全局锁(Global Interpreter Lock),即Python为了保证线程安全而采取的独立线程运行的限制,说白了就是一个核只能在同一时间运行一个线程.对于io密集型任务，python的多线程起到作用，但对于cpu密集型任务，python的多线程几乎占不到任何优势，还有可能因为争夺资源而变慢。。

**96.协程**

简单点说协程是进程和线程的升级版,进程和线程都面临着内核态和用户态的切换问题而耗费许多切换时间,而协程就是用户自己控制切换的时机,不再需要陷入系统的内核态.

**97.闭包**

闭包(closure)是函数式编程的重要的语法结构。闭包也是一种组织代码的结构，它同样提高了代码的可重复使用性。

当一个内嵌函数引用其外部作作用域的变量,我们就会得到一个闭包. 总结一下,创建一个闭包必须满足以下几点:

必须有一个内嵌函数

内嵌函数必须引用外部函数中的变量

外部函数的返回值必须是内嵌函数

感觉闭包还是有难度的,几句话是说不明白的,还是查查相关资料.

重点是函数运行后并不会被撤销,就像16题的instance字典一样,当函数运行完后,instance并不被销毁,而是继续留在内存空间里.这个功能类似类里的类变量,只不过迁移到了函数上.

闭包就像个空心球一样,你知道外面和里面,但你不知道中间是什么样.

**98.lambda函数**

其实就是一个匿名函数,为什么叫lambda?因为和后面的函数式编程有关.

**99.Python函数式编程**

python中函数式编程支持:

filter 函数的功能相当于过滤器。调用一个布尔函数bool\_func来迭代遍历每个seq中的元素；返回一个使bool\_seq返回值为true的元素的序列。

>>>a = [1,2,3,4,5,6,7]>>>b = filter(lambda x: x > 5, a)>>>print b>>>[6,7]

map函数是对一个序列的每个项依次执行函数，下面是对一个序列每个项都乘以2：

>>> a = map(lambda x:x\*2,[1,2,3])>>> list(a)

[2, 4, 6]

reduce函数是对一个序列的每个项迭代调用函数，下面是求3的阶乘：

>>> reduce(lambda x,y:x\*y,range(1,4))6

**100.Python里的拷贝**

引用和copy(),deepcopy()的区别

import copy

a = [1, 2, 3, 4, ['a', 'b']] #原始对象

b = a #赋值，传对象的引用

c = copy.copy(a) #对象拷贝，浅拷贝

d = copy.deepcopy(a) #对象拷贝，深拷贝

a.append(5) #修改对象a

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

print 'a = ', aprint 'b = ', bprint 'c = ', cprint 'd = ', d

输出结果：

a = [1, 2, 3, 4, ['a', 'b', 'c'], 5]

b = [1, 2, 3, 4, ['a', 'b', 'c'], 5]

c = [1, 2, 3, 4, ['a', 'b', 'c']]

d = [1, 2, 3, 4, ['a', 'b']]

**101.Python垃圾回收机制**

Python GC主要使用引用计数（reference counting）来跟踪和回收垃圾。在引用计数的基础上，通过“标记-清除”（mark and sweep）解决容器对象可能产生的循环引用问题，通过“分代回收”（generation collection）以空间换时间的方法提高垃圾回收效率。

### **1).引用计数**

PyObject是每个对象必有的内容，其中ob\_refcnt就是做为引用计数。当一个对象有新的引用时，它的ob\_refcnt就会增加，当引用它的对象被删除，它的ob\_refcnt就会减少.引用计数为0时，该对象生命就结束了。

优点:

1. 简单
2. 实时性

缺点:

1. 维护引用计数消耗资源
2. 循环引用

### **2).标记-清除机制**

基本思路是先按需分配，等到没有空闲内存的时候从寄存器和程序栈上的引用出发，遍历以对象为节点、以引用为边构成的图，把所有可以访问到的对象打上标记，然后清扫一遍内存空间，把所有没标记的对象释放。

### **3).分代技术**

分代回收的整体思想是：将系统中的所有内存块根据其存活时间划分为不同的集合，每个集合就成为一个“代”，垃圾收集频率随着“代”的存活时间的增大而减小，存活时间通常利用经过几次垃圾回收来度量。

Python默认定义了三代对象集合，索引数越大，对象存活时间越长。

举例： 当某些内存块M经过了3次垃圾收集的清洗之后还存活时，我们就将内存块M划到一个集合A中去，而新分配的内存都划分到集合B中去。当垃圾收集开始工作时，大多数情况都只对集合B进行垃圾回收，而对集合A进行垃圾回收要隔相当长一段时间后才进行，这就使得垃圾收集机制需要处理的内存少了，效率自然就提高了。在这个过程中，集合B中的某些内存块由于存活时间长而会被转移到集合A中，当然，集合A中实际上也存在一些垃圾，这些垃圾的回收会因为这种分代的机制而被延迟。

**102.read,readline和readlines**

* 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

**103.select,poll和epoll**

其实所有的I/O都是轮询的方法,只不过实现的层面不同罢了.

这个问题可能有点深入了,但相信能回答出这个问题是对I/O多路复用有很好的了解了.其中tornado使用的就是epoll的.

[selec,poll和epoll区别总结](http://www.cnblogs.com/Anker/p/3265058.html)

基本上select有3个缺点:

1. 连接数受限
2. 查找配对速度慢
3. 数据由内核拷贝到用户态

poll改善了第一个缺点

epoll改了三个缺点.

**104. 调度算法**

1. 先来先服务(FCFS, First Come First Serve)
2. 短作业优先(SJF, Shortest Job First)
3. 最高优先权调度(Priority Scheduling)
4. 时间片轮转(RR, Round Robin)
5. 多级反馈队列调度(multilevel feedback queue scheduling)

常见的调度算法总结:<http://www.jianshu.com/p/6edf8174c1eb>

实时调度算法:

1. 最早截至时间优先 EDF
2. 最低松弛度优先 LLF

**105.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
4. 环路等待条件

处理死锁基本方法:

1. 预防死锁(摒弃除1以外的条件)
2. 避免死锁(银行家算法)
3. 检测死锁(资源分配图)
4. 解除死锁
   1. 剥夺资源
   2. 撤销进程

**106.事务**

数据库事务(Database Transaction) ，是指作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全地不执行。 彻底理解数据库事务: <http://www.hollischuang.com/archives/898>

**107.数据库索引**

推荐: <http://tech.meituan.com/mysql-index.html>

[MySQL索引背后的数据结构及算法原理](http://blog.codinglabs.org/articles/theory-of-mysql-index.html)

聚集索引,非聚集索引,B-Tree,B+Tree,最左前缀原理

**108.Redis原理**

### **Redis是什么？**

1. 是一个完全开源免费的key-value内存数据库
2. 通常被认为是一个数据结构服务器，主要是因为其有着丰富的数据结构 strings、map、 list、sets、 sorted sets

### **Redis数据库**

​ 通常局限点来说，Redis也以消息队列的形式存在，作为内嵌的List存在，满足实时的高并发需求。在使用缓存的时候，redis比memcached具有更多的优势，并且支持更多的数据类型，把redis当作一个中间存储系统，用来处理高并发的数据库操作

* 速度快：使用标准C写，所有数据都在内存中完成，读写速度分别达到10万/20万
* 持久化：对数据的更新采用Copy-on-write技术，可以异步地保存到磁盘上，主要有两种策略，一是根据时间，更新次数的快照（save 300 10 ）二是基于语句追加方式(Append-only file，aof)
* 自动操作：对不同数据类型的操作都是自动的，很安全
* 快速的主--从复制，官方提供了一个数据，Slave在21秒即完成了对Amazon网站10G key set的复制。
* Sharding技术： 很容易将数据分布到多个Redis实例中，数据库的扩展是个永恒的话题，在关系型数据库中，主要是以添加硬件、以分区为主要技术形式的纵向扩展解决了很多的应用场景，但随着web2.0、移动互联网、云计算等应用的兴起，这种扩展模式已经不太适合了，所以近年来，像采用主从配置、数据库复制形式的，Sharding这种技术把负载分布到多个特理节点上去的横向扩展方式用处越来越多。

**109.Redis缺点**

* 是数据库容量受到物理内存的限制,不能用作海量数据的高性能读写,因此Redis适合的场景主要局限在较小数据量的高性能操作和运算上。
* 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。

**110.乐观锁和悲观锁**

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

乐观锁与悲观锁的具体区别: <http://www.cnblogs.com/Bob-FD/p/3352216.html>

**111.MVCC**

​ 全称是Multi-Version Concurrent Control，即多版本并发控制，在MVCC协议下，每个读操作会看到一个一致性的snapshot，并且可以实现非阻塞的读。MVCC允许数据具有多个版本，这个版本可以是时间戳或者是全局递增的事务ID，在同一个时间点，不同的事务看到的数据是不同的。

### **[MySQL](http://lib.csdn.net/base/mysql)的innodb引擎是如何实现MVCC的**

innodb会为每一行添加两个字段，分别表示该行****创建的版本****和****删除的版本****，填入的是事务的版本号，这个版本号随着事务的创建不断递增。在repeated read的隔离级别（[事务的隔离级别请看这篇文章](http://blog.csdn.net/chosen0ne/article/details/10036775)）下，具体各种数据库操作的实现：

* select：满足以下两个条件innodb会返回该行数据：
  + 该行的创建版本号小于等于当前版本号，用于保证在select操作之前所有的操作已经执行落地。
  + 该行的删除版本号大于当前版本或者为空。删除版本号大于当前版本意味着有一个并发事务将该行删除了。
* insert：将新插入的行的创建版本号设置为当前系统的版本号。
* delete：将要删除的行的删除版本号设置为当前系统的版本号。
* update：不执行原地update，而是转换成insert + delete。将旧行的删除版本号设置为当前版本号，并将新行insert同时设置创建版本号为当前版本号。

其中，写操作（insert、delete和update）执行时，需要将系统版本号递增。

​ 由于旧数据并不真正的删除，所以必须对这些数据进行清理，innodb会开启一个后台线程执行清理工作，具体的规则是将删除版本号小于当前系统版本的行删除，这个过程叫做purge。

通过MVCC很好的实现了事务的隔离性，可以达到repeated read级别，要实现serializable还必须加锁。

参考：[MVCC浅析](http://blog.csdn.net/chosen0ne/article/details/18093187)

**112. MyISAM和InnoDB**

MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。另外，MyISAM 对于 SELECT COUNT(\*) 这类的计算是超快无比的。

InnoDB 的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM 还慢。他是它支持“行锁” ，于是在写操作比较多的时候，会更优秀。并且，他还支持更多的高级应用，比如：事务。

mysql 数据库引擎: <http://www.cnblogs.com/0201zcr/p/5296843.html> MySQL存储引擎－**－MyISAM与InnoDB区别: <https://segmentfault.com/a/1190000008227211>**

**113.三次握手**

1. 客户端通过向服务器端发送一个SYN来创建一个主动打开，作为三次握手的一部分。客户端把这段连接的序号设定为随机数 A。
2. 服务器端应当为一个合法的SYN回送一个SYN/ACK。ACK 的确认码应为 A+1，SYN/ACK 包本身又有一个随机序号 B。
3. 最后，客户端再发送一个ACK。当服务端受到这个ACK的时候，就完成了三路握手，并进入了连接创建状态。此时包序号被设定为收到的确认号 A+1，而响应则为 B+1。

**114.注意: 中断连接端可以是客户端，也可以是服务器端. 下面仅以客户端断开连接举例, 反之亦然.**

1. 客户端发送一个数据分段, 其中的 FIN 标记设置为1. 客户端进入 FIN-WAIT 状态. 该状态下客户端只接收数据, 不再发送数据.
2. 服务器接收到带有 FIN = 1 的数据分段, 发送带有 ACK = 1 的剩余数据分段, 确认收到客户端发来的 FIN 信息.
3. 服务器等到所有数据传输结束, 向客户端发送一个带有 FIN = 1 的数据分段, 并进入 CLOSE-WAIT 状态, 等待客户端发来带有 ACK = 1 的确认报文.
4. 客户端收到服务器发来带有 FIN = 1 的报文, 返回 ACK = 1 的报文确认, 为了防止服务器端未收到需要重发, 进入 TIME-WAIT 状态. 服务器接收到报文后关闭连接. 客户端等待 2MSL 后未收到回复, 则认为服务器成功关闭, 客户端关闭连接.

图解: <http://blog.csdn.net/whuslei/article/details/6667471>

**115. ARP协议**

地址解析协议(Address Resolution Protocol)，其基本功能为透过目标设备的IP地址，查询目标的MAC地址，以保证通信的顺利进行。它是IPv4网络层必不可少的协议，不过在IPv6中已不再适用，并被邻居发现协议（NDP）所替代。

**116.urllib和urllib2的区别**

这个面试官确实问过,当时答的urllib2可以Post而urllib不可以.

1. urllib提供urlencode方法用来GET查询字符串的产生，而urllib2没有。这是为何urllib常和urllib2一起使用的原因。
2. urllib2可以接受一个Request类的实例来设置URL请求的headers，urllib仅可以接受URL。这意味着，你不可以伪装你的User Agent字符串等。

**117.Post和Get**

[GET和POST有什么区别？及为什么网上的多数答案都是错的](http://www.cnblogs.com/nankezhishi/archive/2012/06/09/getandpost.html) [知乎回答](https://www.zhihu.com/question/31640769?rf=37401322)

get: [RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1](http://tools.ietf.org/html/rfc2616" \l "section-9.3) post: [RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1](http://tools.ietf.org/html/rfc2616" \l "section-9.5)

**118.Cookie和Session**

|  | **Cookie** | **Session** |
| --- | --- | --- |
| 储存位置 | 客户端 | 服务器端 |
| 目的 | 跟踪会话，也可以保存用户偏好设置或者保存用户名密码等 | 跟踪会话 |
| 安全性 | 不安全 | 安全 |

session技术是要使用到cookie的，之所以出现session技术，主要是为了安全。

**119.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**

**120.网站用户密码保存**

1. 明文保存
2. 明文hash后保存,如md5
3. MD5+Salt方式,这个salt可以随机
4. 知乎使用了Bcrypy(好像)加密

**121.HTTP和HTTPS**

| **状态码** | **定义** |
| --- | --- |
| 1xx 报告 | 接收到请求，继续进程 |
| 2xx 成功 | 步骤成功接收，被理解，并被接受 |
| 3xx 重定向 | 为了完成请求,必须采取进一步措施 |
| 4xx 客户端出错 | 请求包括错的顺序或不能完成 |
| 5xx 服务器出错 | 服务器无法完成显然有效的请求 |

403: Forbidden 404: Not Found

HTTPS握手,对称加密,非对称加密,TLS/SSL,RSA

**122.XSRF和XSS**

* CSRF(Cross-site request forgery)跨站请求伪造
* XSS(Cross Site Scripting)跨站脚本攻击

CSRF重点在请求,XSS重点在脚本

**123.幂等 Idempotence**

HTTP方法的幂等性是指一次和多次请求某一个资源应该具有同样的****副作用****。(注意是副作用)

GET http://www.bank.com/account/123456，不会改变资源的状态，不论调用一次还是N次都没有副作用。请注意，这里强调的是一次和N次具有相同的副作用，而不是每次GET的结果相同。GET http://www.news.com/latest-news这个HTTP请求可能会每次得到不同的结果，但它本身并没有产生任何副作用，因而是满足幂等性的。

DELETE方法用于删除资源，有副作用，但它应该满足幂等性。比如：DELETE http://www.forum.com/article/4231，调用一次和N次对系统产生的副作用是相同的，即删掉id为4231的帖子；因此，调用者可以多次调用或刷新页面而不必担心引起错误。

POST所对应的URI并非创建的资源本身，而是资源的接收者。比如：POST http://www.forum.com/articles的语义是在http://www.forum.com/articles下创建一篇帖子，HTTP响应中应包含帖子的创建状态以及帖子的URI。两次相同的POST请求会在服务器端创建两份资源，它们具有不同的URI；所以，POST方法不具备幂等性。

PUT所对应的URI是要创建或更新的资源本身。比如：PUT <http://www.forum/articles/4231的语义是创建或更新ID为4231的帖子。对同一URI进行多次PUT的副作用和一次PUT是相同的；因此，PUT方法具有幂等性。>

**124. SOAP**

SOAP（原为Simple Object Access Protocol的首字母缩写，即简单对象访问协议）是交换数据的一种协议规范，使用在计算机网络Web服务（web service）中，交换带结构信息。SOAP为了简化网页服务器（Web Server）从XML数据库中提取数据时，节省去格式化页面时间，以及不同应用程序之间按照HTTP通信协议，遵从XML格式执行资料互换，使其抽象于语言实现、平台和硬件。

**125.RPC**

RPC（Remote Procedure Call Protocol）——远程过程调用协议，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC协议假定某些传输协议的存在，如TCP或UDP，为通信程序之间携带信息数据。在OSI网络通信模型中，RPC跨越了传输层和应用层。RPC使得开发包括网络分布式多程序在内的应用程序更加容易。

总结:服务提供的两大流派.传统意义以方法调用为导向通称RPC。为了企业SOA,若干厂商联合推出webservice,制定了wsdl接口定义,传输soap.当互联网时代,臃肿SOA被简化为http+xml/json.但是简化出现各种混乱。以资源为导向,任何操作无非是对资源的增删改查，于是统一的REST出现了.

进化的顺序: RPC -> SOAP -> RESTful

**126.CGI和WSGI**

CGI是通用网关接口，是连接web服务器和应用程序的接口，用户通过CGI来获取动态数据或文件等。 CGI程序是一个独立的程序，它可以用几乎所有语言来写，包括perl，c，lua，python等等。

WSGI, Web Server Gateway Interface，是Python应用程序或框架和Web服务器之间的一种接口，WSGI的其中一个目的就是让用户可以用统一的语言(Python)编写前后端。

**127.中间人攻击**

在GFW里屡见不鲜的,呵呵.

中间人攻击（Man-in-the-middle attack，通常缩写为MITM）是指攻击者与通讯的两端分别创建独立的联系，并交换其所收到的数据，使通讯的两端认为他们正在通过一个私密的连接与对方直接对话，但事实上整个会话都被攻击者完全控制。

**128.c10k问题**

所谓c10k问题，指的是服务器同时支持成千上万个客户端的问题，也就是concurrent 10 000 connection（这也是c10k这个名字的由来）。 推荐: <https://my.oschina.net/xianggao/blog/664275>

**129.HTTP1.0和HTTP1.1**

推荐: <http://blog.csdn.net/elifefly/article/details/3964766>

1. 请求头Host字段,一个服务器多个网站
2. 长链接
3. 文件断点续传
4. 身份认证,状态管理,Cache缓存

HTTP请求8种方法介绍 HTTP/1.1协议中共定义了8种HTTP请求方法，HTTP请求方法也被叫做“请求动作”，不同的方法规定了不同的操作指定的资源方式。服务端也会根据不同的请求方法做不同的响应。

GET

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

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

HEAD

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

POST

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

PUT

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

DELETE

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

CONNECT

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

OPTIONS

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

TRACE

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

HTTP/1.1之后增加的方法

在HTTP/1.1标准制定之后，又陆续扩展了一些方法。其中使用中较多的是 PATCH 方法：

PATCH

PATCH方法出现的较晚，它在2010年的RFC 5789标准中被定义。PATCH请求与PUT请求类似，同样用于资源的更新。二者有以下两点不同：

但PATCH一般用于资源的部分更新，而PUT一般用于资源的整体更新。 当资源不存在时，PATCH会创建一个新的资源，而PUT只会对已在资源进行更新。

**130.Ajax**

AJAX,Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）, 是与在不重新加载整个页面的情况下，与服务器交换数据并更新部分网页的技术。

**131.unix进程间通信方式(IPC)**

1. 管道（Pipe）：管道可用于具有亲缘关系进程间的通信，允许一个进程和另一个与它有共同祖先的进程之间进行通信。
2. 命名管道（named pipe）：命名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。命名管道在文件系统中有对应的文件名。命名管道通过命令mkfifo或系统调用mkfifo来创建。
3. 信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；linux除了支持Unix早期信号语义函数sigal外，还支持语义符合Posix.1标准的信号函数sigaction（实际上，该函数是基于BSD的，BSD为了实现可靠信号机制，又能够统一对外接口，用sigaction函数重新实现了signal函数）。
4. 消息（Message）队列：消息队列是消息的链接表，包括Posix消息队列system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺
5. 共享内存：使得多个进程可以访问同一块内存空间，是最快的可用IPC形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。
6. 内存映射（mapped memory）：内存映射允许任何多个进程间通信，每一个使用该机制的进程通过把一个共享的文件映射到自己的进程地址空间来实现它。
7. 信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。
8. 套接口（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix系统的BSD分支开发出来的，但现在一般可以移植到其它类Unix系统上：Linux和System V的变种都支持套接字。

**132.红黑树**

红黑树与AVL的比较：

AVL是严格平衡树，因此在增加或者删除节点的时候，根据不同情况，旋转的次数比红黑树要多；

红黑是用非严格的平衡来换取增删节点时候旋转次数的降低；

所以简单说，如果你的应用中，搜索的次数远远大于插入和删除，那么选择AVL，如果搜索，插入删除次数几乎差不多，应该选择RB。

红黑树详解: <https://xieguanglei.github.io/blog/post/red-black-tree.html>

教你透彻了解红黑树: <https://github.com/julycoding/The-Art-Of-Programming-By-July/blob/master/ebook/zh/03.01.md>

**133.台阶问题/斐波那契**

一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

fib = lambda n: n if n <= 2 else fib(n - 1) + fib(n - 2)

第二种记忆方法

def memo(func):

cache = {}

def wrap(\*args):

if args not in cache:

cache[args] = func(\*args)

return cache[args]

return wrap

@memodef fib(i):

if i < 2:

return 1

return fib(i-1) + fib(i-2)

第三种方法

def fib(n):

a, b = 0, 1

for \_ in xrange(n):

a, b = b, a + b

return b

**134.变态台阶问题**

一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

fib = lambda n: n if n < 2 else 2 \* fib(n - 1)

**135. 矩形覆盖**

我们可以用2\*1的小矩形横着或者竖着去覆盖更大的矩形。请问用n个2\*1的小矩形无重叠地覆盖一个2\*n的大矩形，总共有多少种方法？

第2\*n个矩形的覆盖方法等于第2\*(n-1)加上第2\*(n-2)的方法。

f = lambda n: 1 if n < 2 else f(n - 1) + f(n - 2)

**136.杨氏矩阵查找**

在一个m行n列二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

使用Step-wise线性搜索。

def get\_value(l, r, c):

return l[r][c]

def find(l, x):

m = len(l) - 1

n = len(l[0]) - 1

r = 0

c = n

while c >= 0 and r <= m:

value = get\_value(l, r, c)

if value == x:

return True

elif value > x:

c = c - 1

elif value < x:

r = r + 1

return False

**137. 去除列表中的重复元素**

用集合

list(set(l))

用字典

l1 = ['b','c','d','b','c','a','a']

l2 = {}.fromkeys(l1).keys()print l2

用字典并保持顺序

l1 = ['b','c','d','b','c','a','a']

l2 = list(set(l1))

l2.sort(key=l1.index)print l2

列表推导式

l1 = ['b','c','d','b','c','a','a']

l2 = []

[l2.append(i) for i in l1 if not i in l2]

sorted排序并且用列表推导式.

l = ['b','c','d','b','c','a','a'] [single.append(i) for i in sorted(l) if i not in single] print single

**138.链表成对调换**

1->2->3->4转换成2->1->4->3.

class ListNode:

def \_\_init\_\_(self, x):

self.val = x

self.next = None

class Solution:

# @param a ListNode

# @return a ListNode

def swapPairs(self, head):

if head != None and head.next != None:

next = head.next

head.next = self.swapPairs(next.next)

next.next = head

return next

return head

**139.创建字典的方法**

### **1 直接创建**

dict = {'name':'earth', 'port':'80'}

### **2 工厂方法**

items=[('name','earth'),('port','80')]

dict2=dict(items)

dict1=dict((['name','earth'],['port','80']))

### **3 fromkeys()方法**

dict1={}.fromkeys(('x','y'),-1)dict={'x':-1,'y':-1}

dict2={}.fromkeys(('x','y'))

dict2={'x':None, 'y':None}

**140.合并两个有序列表**

知乎远程面试要求编程

尾递归

def \_recursion\_merge\_sort2(l1, l2, tmp):

if len(l1) == 0 or len(l2) == 0:

tmp.extend(l1)

tmp.extend(l2)

return tmp

else:

if l1[0] < l2[0]:

tmp.append(l1[0])

del l1[0]

else:

tmp.append(l2[0])

del l2[0]

return \_recursion\_merge\_sort2(l1, l2, tmp)

def recursion\_merge\_sort2(l1, l2):

return \_recursion\_merge\_sort2(l1, l2, [])

循环算法

思路：

定义一个新的空列表

比较两个列表的首个元素

小的就插入到新列表里

把已经插入新列表的元素从旧列表删除

直到两个旧列表有一个为空

再把旧列表加到新列表后面

def loop\_merge\_sort(l1, l2):

tmp = []

while len(l1) > 0 and len(l2) > 0:

if l1[0] < l2[0]:

tmp.append(l1[0])

del l1[0]

else:

tmp.append(l2[0])

del l2[0]

tmp.extend(l1)

tmp.extend(l2)

return tmp

pop弹出

a = [1,2,3,7]

b = [3,4,5]

def merge\_sortedlist(a,b):

c = []

while a and b:

if a[0] >= b[0]:

c.append(b.pop(0))

else:

c.append(a.pop(0))

while a:

c.append(a.pop(0))

while b:

c.append(b.pop(0))

return cprint merge\_sortedlist(a,b)

**141.二分查找**

#coding:utf-8def binary\_search(list,item):

low = 0

high = len(list)-1

while low<=high:

mid = (low+high)/2

guess = list[mid]

if guess>item:

high = mid-1

elif guess<item:

low = mid+1

else:

return mid

return None

mylist = [1,3,5,7,9]print binary\_search(mylist,3)

**142.快排**

#coding:utf-8def quicksort(list):

if len(list)<2:

return list

else:

midpivot = list[0]

lessbeforemidpivot = [i for i in list[1:] if i<=midpivot]

biggerafterpivot = [i for i in list[1:] if i > midpivot]

finallylist = quicksort(lessbeforemidpivot)+[midpivot]+quicksort(biggerafterpivot)

return finallylist

print quicksort([2,4,6,7,1,2,5])

**143.找零问题**

#coding:utf-8#values是硬币的面值values = [ 25, 21, 10, 5, 1]#valuesCounts 钱币对应的种类数#money 找出来的总钱数#coinsUsed 对应于目前钱币总数i所使用的硬币数目

def coinChange(values,valuesCounts,money,coinsUsed):

#遍历出从1到money所有的钱数可能

for cents in range(1,money+1):

minCoins = cents

#把所有的硬币面值遍历出来和钱数做对比

for kind in range(0,valuesCounts):

if (values[kind] <= cents):

temp = coinsUsed[cents - values[kind]] +1

if (temp < minCoins):

minCoins = temp

coinsUsed[cents] = minCoins

print ('面值:{0}的最少硬币使用数为:{1}'.format(cents, coinsUsed[cents]))

**144.二叉树节点**

class Node(object):

def \_\_init\_\_(self, data, left=None, right=None):

self.data = data

self.left = left

self.right = right

tree = Node(1, Node(3, Node(7, Node(0)), Node(6)), Node(2, Node(5), Node(4)))

**145.层次遍历**

def lookup(root):

row = [root]

while row:

       print(row)

       row = [kid for item in row for kid in (item.left, item.right) if kid]

**146.深度遍历**

def deep(root):

if not root:

return

print root.data

deep(root.left)

deep(root.right)

if \_\_name\_\_ == '\_\_main\_\_':

lookup(tree)

deep(tree)

**147.前中后序遍历**

深度遍历改变顺序就OK了

#coding:utf-8#二叉树的遍历#简单的二叉树节点类class Node(object):

def \_\_init\_\_(self,value,left,right):

self.value = value

self.left = left

self.right = right

#中序遍历:遍历左子树,访问当前节点,遍历右子树

def mid\_travelsal(root):

if root.left is not None:

mid\_travelsal(root.left)

#访问当前节点

print(root.value)

if root.right is not None:

mid\_travelsal(root.right)

#前序遍历:访问当前节点,遍历左子树,遍历右子树

def pre\_travelsal(root):

print (root.value)

if root.left is not None:

pre\_travelsal(root.left)

if root.right is not None:

pre\_travelsal(root.right)

#后续遍历:遍历左子树,遍历右子树,访问当前节点

def post\_trvelsal(root):

if root.left is not None:

post\_trvelsal(root.left)

if root.right is not None:

post\_trvelsal(root.right)

print (root.value)

**148.求最大树深**

def maxDepth(root):

if not root:

return 0

return max(maxDepth(root.left), maxDepth(root.right)) + 1

1. pip install openxl

150.创建一个工作薄：wb = openpyxl.Workbook()

151.新增一个sheet表单：wb.create\_sheet('test\_case')

152.保存case.xlsx文件：wb.save('cases.xlsx')

153.打开工作簿：wb = openpyxl.load\_workbook('cases.xlsx')

154.选取表单：sh = wb['Sheet1'

155.读取第一行、第一列的数据：ce = sh.cell(row = 1,column = 1)

156.按行读取数据：row\_data = list(sh.rows)

157.关闭工作薄：wb.close()

158.按列读取数据：columns\_data = list(sh.columns)

**159.写入数据之前，该文件一定要处于关闭状态**

写入第一行、第四列的数据 value = 'result'：sh.cell(row = 1,column = 4,value = 'result')

160.获取最大行总数、最大列总数：sh.max\_row、sh.max\_column

161.del 删除表单的用法：del wb['sheet\_name']

162.remove 删除表单的用法：sh = wb['sheet\_name'] wb.remove(sh)

163.举例子

import openpyxl

# 创建一个工作簿

wb = openpyxl.Workbook()

# 创建一个test\_case的sheet表单

wb.create\_sheet('test\_case')

# 保存为一个xlsx格式的文件

wb.save('cases.xlsx')

# 读取excel中的数据

# 第一步：打开工作簿

wb = openpyxl.load\_workbook('cases.xlsx')

# 第二步：选取表单

sh = wb['Sheet1']

# 第三步：读取数据

# 参数 row:行 column：列

ce = sh.cell(row = 1,column = 1) # 读取第一行，第一列的数据

print(ce.value)

# 按行读取数据 list(sh.rows)

print(list(sh.rows)[1:]) # 按行读取数据，去掉第一行的表头信息数据

for cases in list(sh.rows)[1:]:

case\_id = cases[0].value

case\_excepted = cases[1].value

case\_data = cases[2].value

print(case\_excepted,case\_data)

# 关闭工作薄

wb.close()

**164.封装一个读取用例的excel类：用来实现读取数据和写入数据的功能**

import openpyxl

class Case: #这个类用来存储用例的

\_\_slots\_\_ = [] #特殊的类属性，可以用来限制这个类创建的实例属性添加 可写可不写

pass

class ReadExcel(object): #读取excel数据的类

def \_\_init\_\_(self,file\_name,sheet\_name):

"""

这个是用来初始化读取对象的

:param file\_name: 文件名 ---> str类型

:param sheet\_name: 表单名 ———> str类型

"""

# 打开文件

self.wb = openpyxl.load\_workbook(file\_name)

# 选择表单

self.sh = self.wb[sheet\_name]

def read\_data\_line(self):

#按行读取数据转化为列表

rows\_data = list(self.sh.rows)

# print(rows\_data)

# 获取表单的表头信息

titles = []

for title in rows\_data[0]:

titles.append(title.value)

# print(titles)

#定义一个空列表用来存储测试用例

cases = []

for case in rows\_data[1:]:

# print(case)

data = []

for cell in case: #获取一条测试用例数据

# print(cell.value)

data.append(cell.value)

# print(data)

#判断该单元格是否为字符串，如果是字符串类型则需要使用eval();如果不是字符串类型则不需要使用eval()

if isinstance(cell.value,str):

data.append(eval(cell.value))

else:

data.append(cell.value)

#将该条数据存放至cases中

# print(dict(list(zip(titles,data))))

case\_data = dict(list(zip(titles,data)))

cases.append(case\_data)

return cases

if \_\_name\_\_ == '\_\_main\_\_':

r = ReadExcel('cases.xlsx','Sheet1')

data1 = r.read\_data\_line()

print(data1)

**165.按行读取数据，存储在对象中**

import openpyxl

class Case:

pass

class ReadExcel(object):

def \_\_init\_\_(self,filename,sheetname):

self.wb = openpyxl.load\_workbook(filename)

self.sh = self.wb[sheetname]

def read\_data\_obj(self):

"""

按行读取数据 每条用例存储在一个对象中

:return:

"""

rows\_data = list(self.sh.rows)

# print(rows\_data)

# 获取表单的表头信息

titles = []

for title in rows\_data[0]:

titles.append(title.value)

# print(titles)

# 定义一个空列表用来存储测试用例

cases = []

for case in rows\_data[1:]:

# print(case)

#创建一个Case类的对象，用来保存用例数据

case\_obj = Case()

data = []

for cell in case: # 获取一条测试用例数据

# print(cell.value)

# data.append(cell.value)

# print(data)

if isinstance(cell.value,str): # 判断该单元格是否为字符串，如果是字符串类型则需要使用eval();如果不是字符串类型则不需要使用eval()

data.append(eval(cell.value))

else:

data.append(cell.value)

# 将该条数据存放至cases中

# print(dict(list(zip(titles,data))))

case\_data = list(zip(titles, data))

# print(case\_data)

for i in case\_data:

setattr(case\_obj,i[0],i[1])

# print(case\_obj)

# print(case\_obj.case\_id,case\_obj.data,case\_obj.excepted)

cases.append(case\_obj)

return cases

if \_\_name\_\_ == '\_\_main\_\_':

r = ReadExcel('cases.xlsx','Sheet1')

res = r.read\_data\_obj()

for i in res:

print(i.caseid, i.excepted, i.data)

### 166.**将测试用例封装到列表中，读取指定列的数据**

import openpyxl

class Case:

pass

class ReadExcelZy(object):

def \_\_init\_\_(self,filename,sheetname):

self.wb = openpyxl.load\_workbook(filename)

self.sheet = self.wb[sheetname]

# list1 参数为一个列表，传入的是指定读取数据的列,比如[1,2,3]

# 每一行[1,3,5]列的数据，读取出来就作为一条测试用例，放在字典中

# 所有的用例放在列表中并且进行返回

def read\_data(self,list1):

"""

:param list1: list--->要读取列 list类型

:return: 返回一个列表，每一个元素为一个用例（用例为dict类型）

"""

# 获取最大的行数

max\_r = self.sheet.max\_row

cases = [] #定义一个空列表，用来存放所有的用例数据

titles = [] #定义一个空列表，用来存放表头

# 遍历所有的行数据

for row in range(1,max\_r+1):

if row != 1: #判断是否是第一行

case\_data = [] #定义一个空列表，用来存放该行的用例数据

for column in list1:

info = self.sheet.cell(row,column).value

# print(info)

case\_data.append(info)

# print(list(zip(titles,case\_data)))

case = dict(zip(titles,case\_data)) #将该条数据和表头进行打包组合,作用相当于dict(list(zip(titles,case\_data)))

# print(case)

cases.append(case)

# print(cases)

else: #获取表头数据

for column in list1:

title = self.sheet.cell(row,column).value

titles.append(title)

# print(titles)

return cases

if \_\_name\_\_ == '\_\_main\_\_':

r = ReadExcelZy("cases.xlsx","Sheet1")

res = r.read\_data([1,2,3])

for o in res:

print(o['caseid'],o['data'],o['excepted'])

### 167.**将测试用例封装到对象中，读取指定列的数据**

import openpyxl

class Case:

pass

class ReadExcelZy(object):

def \_\_init\_\_(self,filename,sheetname):

self.wb = openpyxl.load\_workbook(filename)

self.sheet = self.wb[sheetname]

# list1 参数为一个列表，传入的是指定读取数据的列,比如[1,2,3]

# 每一行[1,3,5]列的数据，读取出来就作为一条测试用例，放在字典中

# 所有的用例放在对象中并且进行返回

def read\_data\_obj(self,list2):

max\_r1 = self.sheet.max\_row #获取最大行数

cases = []

titles = [] #用来存放表头数据

for row in range(1,max\_r1+1):

if row != 1:

case\_data = []

for column in list2:

info = self.sheet.cell(row,column).value

# print(info)

case\_data.append(info)

cases\_data = list(zip(titles,case\_data))

#将一条用例存到一个对象中（每一列对应对象的一个属性）

case\_obj = Case()

for i in cases\_data:

# print(i)

setattr(case\_obj,i[0],i[1])

# print(case\_obj.caseid,case\_obj.excepted,case\_obj.data)

cases.append(case\_obj)

else:

for column in list2:

title = self.sheet.cell(row,column).value

titles.append(title)

return cases

if \_\_name\_\_ == '\_\_main\_\_':

r = ReadExcelZy("cases.xlsx","Sheet1")

res = r.read\_data\_obj([1,2,3])

for i in res:

print(i.caseid,i.data,i.excepted)

### 168.**优化第4部分代码，将设置对象属性写在初始化方法中(封装Excel类读取数据最常用的方法)**

import openpyxl

class Case: # 这个类用来存储用例的

def \_\_init\_\_(self, attrs):

"""

初始化用例

:param attrs:zip类型——>[{key,value},(key1,value1)......]

"""

for i in attrs:

setattr(self, i[0], i[1])

class ReadExcel(object):

def \_\_init\_\_(self, filename, sheetname):

"""

定义需要打开的文件及表名

:param filename: 文件名

:param sheetname: 表名

"""

self.wb = openpyxl.load\_workbook(filename)

self.sheet = self.wb[sheetname]

def read\_data\_obj\_new(self, list2):

# 获取最大行数

max\_r1 = self.sheet.max\_row

cases = []

# 用来存放表头数据

titles = []

for row in range(1, max\_r1 + 1):

if row != 1:

case\_data = []

for column in list2:

info = self.sheet.cell(row, column).value

# print(info)

case\_data.append(info)

case = list(zip(titles, case\_data))

# 新建对象时，将对象传给Case类

case\_obj = Case(case)

# print(case\_obj.caseid,case\_obj.excepted,case\_obj.data)

cases.append(case\_obj)

else:

# 获取表头

for column in list2:

title = self.sheet.cell(row, column).value

titles.append(title)

if None in titles:

raise ValueError("传入的表头的数据有显示为空")

return cases

if \_\_name\_\_ == '\_\_main\_\_':

r = ReadExcel('cases.xlsx', 'Sheet1')

res1 = r.read\_data\_obj\_new([1, 2, 3])

for i in res1:

print(i.caseid, i.data, i.excepted)

## 169.**将测试数据参数化**

import unittest

from python.register\_new.register import register

from python.register\_new.register\_testcase\_new import RegisterTestCase

from HTMLTestRunnerNew import HTMLTestRunner

class RegisterTestCase(unittest.TestCase):

# 初始化测试用例

def \_\_init\_\_(self,modethod\_name,excepted,data):

# modethod\_name 测试用例方法名

super().\_\_init\_\_(modethod\_name)

# excepted 测试用例的预期结果

self.excepted = excepted

# data 测试用例参数值

self.data = data

def setUp(self):

print("准备测试环境，执行测试用例之前会执行此操作")

def tearDown(self):

print("还原测试环境，执行完测试用例之后会执行此操作")

def test\_register(self):

res = register(\*self.data)

try:

self.assertEquals(self.excepted,res)

except AssertionError as e:

print("该条测试用例执行未通通过")

raise e

else:

print("该条测试用例执行通过")

# 创建测试套件

suite = unittest.TestSuite()

# 将测试用例添加至测试套件中

case = [{'excepted':'{"code": 1, "msg": "注册成功"}','data':'('python1', '123456','123456')'},

{'excepted':'{"code": 0, "msg": "两次密码不一致"}','data':'('python1', '1234567','123456')'}]

for case in cases:

suite.addTest(RegisterTestCase('test\_register',case['excepted'],case['data']))

# 执行测试套件，生成测试报告

with open("report.html",'wb') as f:

runner = HTMLTestRunner(

stream = f,

verbosity = 2,

title = 'python\_test\_report',

description = '这是一份测试报告',

tester = 'WL'

)

runner.run(suite)

### 170.**将调用封装好的Excel类的完整代码流程**

import unittest

from python.register\_new.register import register

from python.register\_new.register\_testcase\_new import RegisterTestCase

from HTMLTestRunnerNew import HTMLTestRunner

from python.readexcel import ReadExcel

class RegisterTestCase(unittest.TestCase):

# 初始化测试用例

def \_\_init\_\_(self, modethod\_name, excepted, data):

# modethod\_name 测试用例方法名

super().\_\_init\_\_(modethod\_name)

# excepted 测试用例的预期结果

self.excepted = excepted

# data 测试用例参数值

self.data = data

def setUp(self):

print("准备测试环境，执行测试用例之前会执行此操作")

def tearDown(self):

print("还原测试环境，执行完测试用例之后会执行此操作")

def test\_register(self):

res = register(\*self.data)

try:

self.assertEquals(self.excepted, res)

except AssertionError as e:

print("该条测试用例执行未通通过")

raise e

else:

print("该条测试用例执行通过")

# 创建测试套件

suite = unittest.TestSuite()

# 调用封装好的读取数据的Excel类，获取测试数据

r1 = ReadExcel('cases.xlsx', 'Sheet1')

cases = r1.read\_data\_obj\_new([2, 3])

# 将测试用例添加至测试套件中

for case in cases:

# 需要使用eva()函数对except和data进行自动识别

suite.addTest(RegisterTestCase('test\_register', eval(case.excepted), eval(case.data)))

# 执行测试套件，生成测试报告

with open("report.html", 'wb') as f:

runner = HTMLTestRunner(

stream=f,

verbosity=2,

title='python\_test\_report',

description='这是一份测试报告',

tester='WL')

runner.run(suite)

**171.为什么要用ajax？Ajax应用程序的优势在于？**

1.通过异步模式，提升了用户体验  
2. 优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占用  
3. Ajax引擎在客户端运行，承担了一部分本来由服务器承担的工作，从而减少了大用户量下的服务器负载。

**172.AJAX最大的特点是什么**。  
Ajax可以实现动态不刷新（局部刷新）  
就是能在不更新整个页面的前提下维护数据。这使得Web应用程序更为迅捷地回应用户动作，并避免了在网络上发送那些没有改变过的信息。

**173.请介绍一下XMLhttprequest对象。**  
Ajax的核心是JavaScript对象XmlHttpRequest。该对象在Internet Explorer 5中首次引入，它是一种支持异步请求的技术。简而言之，XmlHttpRequest使您可以使用JavaScript向服务器提出请求并处理响应，而不阻塞用户。通过XMLHttpRequest对象，Web开发人员可以在页面加载以后进行页面的局部更新。

**174.AJAX技术体系的组成部分有哪些。**HTML，css，dom，xml，xmlHttpRequest，javascript

**175.Ajax和javascript的区别。**javascript是一种在浏览器端执行的脚本语言，Ajax是一种创建交互式网页应用的开发技术 ，它是利用了一系列相关的技术其中就包括javascript。  
Javascript是由网景公司开发的一种脚本语言，它和sun公司的java语言是没有任何关系的，它们相似的名称只是一种行销策略。  
在一般的web开发中，javascript是在浏览器端执行的，我们可以用javascript控制浏览器的行为和内容。  
在 Ajax应用中信息是如何在浏览器和服务器之间传递的  
通过XML数据或者字符串

**176.AJAX都有哪些优点和缺点？**

1、最大的一点是页面无刷新，用户的体验非常好。

2、使用异步方式与服务器通信，具有更加迅速的响应能力。

3、可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。并且减轻服务器的负担，ajax的原则是“按需取数据”，可以最大程度的减少冗余请求，和响应对服务器造成的负担。

4、基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

ajax的缺点

1、ajax不支持浏览器back按钮。

2、安全问题 AJAX暴露了与服务器交互的细节。

3、对搜索引擎的支持比较弱。

4、破坏了程序的异常机制。

5、不容易调试。

177.**ajax的实现流程是怎样的？**

(1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.

(2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.

(3)设置响应HTTP请求状态变化的函数.

(4)发送HTTP请求.

(5)获取异步调用返回的数据.

(6)使用JavaScript和DOM实现局部刷新.

**178.简述异步加载js**

异步加载的方案： 动态插入 script 标签

通过 ajax 去获取 js 代码，然后通过 eval 执行

script 标签上添加 defer 或者 async 属性

创建并插入 iframe，让它异步执行 js

**179.AJAX请求总共有多少种CALLBACK。**  
 Ajax请求总共有八种Callback  
 onSuccess  
 onFailure  
 onUninitialized  
 onLoaded  
 onInteractive  
 onComplete  
 onException

### **180.AJAX使用了哪些技术？**

HTML / XHTML和CSS - 这些技术用于显示内容和样式。

DOM - 用于动态显示和与数据交互。

XML - 用于与服务器之间传送数据

XMLHttpRequest - 用于客户端和服务器之间的异步通信。

JavaScript - 主要用于客户端验证

#### ****181.触发器的作用？****

触发器是一种特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

#### ****182.存储过程的优缺点？****

**优点：**

1）存储过程是预编译过的，执行效率高。

2）存储过程的代码直接存放于数据库中，通过存储过程名直接调用，减少网络通讯。

3）安全性高，执行存储过程需要有一定权限的用户。

4）存储过程可以重复使用，可减少数据库开发人员的工作量。

**缺点：**

移植性差

#### ****183.什么叫视图？游标是什么？****

**视图：**

是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改会影响基本表。它使得我们获取数据更容易，相比多表查询。

**游标：**

是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

#### ****184.非关系型数据库和关系型数据库区别，优势比较?****

**非关系型数据库的优势：**

* **性能：**NOSQL是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过SQL层的解析，所以性能非常高。
* **可扩展性：**同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展。

**关系型数据库的优势：**

* **复杂查询：**可以用SQL语句方便的在一个表以及多个表之间做非常复杂的数据查询。
* **事务支持：**使得对于安全性能很高的数据访问要求得以实现。

**其他：**

**1.**对于这两类数据库，对方的优势就是自己的弱势，反之亦然。

**2.**NOSQL数据库慢慢开始具备SQL数据库的一些复杂查询功能，比如MongoDB。

**3.**对于事务的支持也可以用一些系统级的原子操作来实现例如乐观锁之类的方法来曲线救国，比如Redis set nx。

#### ****185.varchar和char的使用场景?****

**1.**char的长度是不可变的，而varchar的长度是可变的。

定义一个char[10]和varchar[10]。  
如果存进去的是‘csdn’,那么char所占的长度依然为10，除了字符‘csdn’外，后面跟六个空格，varchar就立马把长度变为4了，取数据的时候，char类型的要用trim()去掉多余的空格，而varchar是不需要的。

**2.**char的存取数度还是要比varchar要快得多，因为其长度固定，方便程序的存储与查找。  
char也为此付出的是空间的代价，因为其长度固定，所以难免会有多余的空格占位符占据空间，可谓是以空间换取时间效率。  
varchar是以空间效率为首位。

**3.**char的存储方式是：对英文字符（ASCII）占用1个字节，对一个汉字占用两个字节。  
varchar的存储方式是：对每个英文字符占用2个字节，汉字也占用2个字节。

**4.**两者的存储数据都非unicode的字符数据。

#### ****186.什么是索引？****

数据库索引，是数据库管理系统中一个排序的数据结构，索引的实现通常使用B树及其变种B+树。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

#### ****187.索引的作用？****

**索引作用：**

协助快速查询、更新数据库表中数据。

为表设置索引要付出代价的：

一是增加了数据库的存储空间

二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。

**188.事物的四大特性**

原子性：要么执行，要么不执行

隔离性：所有操作全部执行完以前其他回话不能看到过程

一致性：事务前后，数据总额一致

持久性：一旦失误提交，对事物的提交就是永久的

**189.数据库隔离级别**

脏读：事务B读取事务A换没有提交的数

不可重复读：俩次事务读的数据不一致

幻读：事务A修改了数据，事务B也修改了数据，这时事务A看来，明明修改了数据，咋不一样

**190.主键 超键 候选键 外键**

主 键：数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。

超 键：在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。

候选键：是最小超键，即没有冗余元素的超键。

外 键：在一个表中存在的另一个表的主键称此表的外键。

### **191.tcp和udp的区别**

TCP：是面向连接的流传输控制协议，具有高可靠性，确保传输数据的正确性，有验证重发机制，因此不会出现丢失或乱序。

UDP：是无连接的数据报服务，不对数据报进行检查与修改，无须等待对方的应答，会出现分组丢失、重复、乱序，但具有较好的实时性，UDP段结构比TCP的段结构简单，因此网络开销也小。

### **192.流量控制和拥塞控制**

拥塞控制  
网络拥塞现象是指到达通信子网中某一部分的分组数量过多,使得该部分网络来不及处理,以致引起这部分乃至整个网络性能下降的现象,严重时甚至会导致网络通信业务陷入停顿,即出现死锁现象。拥塞控制是处理网络拥塞现象的一种机制。  
流量控制  
数据的传送与接收过程当中很可能出现收方来不及接收的情况,这时就需要对发方进行控制,以免数据丢失。

### **193.多线程如何同步**

windows

线程同步有四种方式：[临界区、内核对象、互斥量、信号量](http://blog.csdn.net/windows_nt/article/details/8953088" \t "https://blog.csdn.net/weiyuefei/article/details/_blank)。

Linux

线程同步有最常用的是：[互斥锁、条件变量和信号量。](http://blog.csdn.net/windows_nt/article/details/28676157" \t "https://blog.csdn.net/weiyuefei/article/details/_blank)

### **194.进程间通讯的方式有哪些，各有什么优缺点**

### **[进程间通信](http://blog.csdn.net/windows_nt/article/details/28682333" \t "https://blog.csdn.net/weiyuefei/article/details/_blank)**

Linux 进程间通信（IPC）以下以几部分发展而来：  
早期UNIX进程间通信、基于System V进程间通信、基于Socket进程间通信和POSIX进程间通信。  
UNIX进程间通信方式包括：管道、FIFO、信号。  
System V进程间通信方式包括：System V消息队列、System V信号灯、System V共享内存、  
POSIX进程间通信包括：posix消息队列、posix信号灯、posix共享内存。  
现在linux使用的进程间通信方式：  
（1）管道（pipe）和有名管道（FIFO）  
（2）信号（signal）  
（3）消息队列  
（4）共享内存  
（5）信号量  
（6）套接字（socket)

### **195.tcp连接建立的时候3次握手，断开连接的4次握手的具体过程**

建立连接采用的3次握手协议，具体是指：  
第一次握手是客户端connect连接到server，server accept client的请求之后，向client端发送一个消息，相当于说我都准备好了，你连接上我了，这是第二次握手，第3次握手就是client向server发送的，就是对第二次握手消息的确认。之后client和server就开始通讯了。  
断开连接的4次握手,具体如下：  
断开连接的一端发送close请求是第一次握手，另外一端接收到断开连接的请求之后需要对close进行确认，发送一个消息，这是第二次握手，发送了确认消息之后还要向对端发送close消息，要关闭对对端的连接，这是第3次握手，而在最初发送断开连接的一端接收到消息之后，进入到一个很重要的状态time\_wait状态，这个状态也是面试官经常问道的问题，最后一次握手是最初发送断开连接的一端接收到消息之后。对消息的确认。

### **196.epoll与select的区别**

select在一个进程中打开的最大fd是有限制的，由FD\_SETSIZE设置，默认值是2048。不过 epoll则没有这个限制，它所支持的fd上限是最大可以打开文件的数目，这个数字一般远大于2048，一般来说内存越大，fd上限越大，1G内存都能达到大约10w左右。  
  
select的轮询机制是系统会去查找每个fd是否数据已准备好，当fd很多的时候，效率当然就直线下降了，epoll采用基于事件的通知方式，一旦某个fd数据就绪时，内核会采用类似callback的回调机制，迅速激活这个文件描述符，而不需要不断的去轮询查找就绪的描述符，这就是epoll高效最本质的原因。  
  
无论是select还是epoll都需要内核把FD消息通知给用户空间，如何避免不必要的内存拷贝就很重要，在这点上，epoll是通过内核于用户空间mmap同一块内存实现的，而select则做了不必要的拷贝

**197.网络编程时的同步、异步、阻塞、非阻塞？**

同步：函数调用在没得到结果之前，没有调用结果，不返回任何结果。

异步：函数调用在没得到结果之前，没有调用结果，返回状态信息。

阻塞：函数调用在没得到结果之前，当前线程挂起。得到结果后才返回。

非阻塞：函数调用在没得到结果之前，当前线程不会挂起，立即返回结果。

**198.四层网络模型和七层网络模型**；

四层网络模型：

a.应用层：应用程序间沟通的层，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）等。

b.传输层：在此层中，它提供了节点间的数据传送服务，如传输控制协议（TCP）、用户数据报协议（UDP）等，TCP和UDP给数据包加入传输数据并把它传输到下一层中，这一层负责传送数据，并且确定数据已被送达并接收。

c.互连网络层：负责提供基本的数据封包传送功能，让每一块数据包都能够到达目的主机（但不检查是否被正确接收），如网际协议（IP）。

d.网络接口层：对实际的网络媒体的管理，定义如何使用实际网络（如Ethernet、Serial Line等）来传送数据。

七层网络模型:

物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

1. **IP，编口号，网关，DNS，路由的作用；**

IP：IP是32位二进制数据，通常以十进制表示，并以“.”分隔。IP地址是一种逻辑地地址，用来标识网络中一个个主机，IP有唯一性，即每台机器的IP在全世界是唯一的。

编口号：

网关：网关实质上是一个网络通向其他网络的IP地址。只有设置好网关的IP地址，TCP/IP协议才能实现不同网络之间的相互通信。网关的IP地址是具有路由功能的设备的IP地址，具有路由功能的设备有路由器、启用了路由协议的服务器（实质上相当于一台路由器）、代理服务器（也相当于一台路由器）。

DNS：域名服务器(Domain Name Server)。在Internet上域名与IP地址之间是一一对应的，域名虽然便于人们记忆，但机器之间只能互相认识IP地址，它们之间的转换工作称为域名解析，域名解析需要由专门的域名解析服务器来完成，DNS就是进行域名解析的服务。

路由：路由器是一种连接多个网络或网段的网络设备，它能将不同网络或网段之间的数据信息进行“翻译”，以使它们能够相互“读”懂对方的数据，从而构成一个更大的网络。

**200、什么时候使用TCP协议，什么时候使用UDP协议；**

a.对数据可靠性的要求。对数据要求高可靠性的应用需选择TCP协议，如验证，密码字段的传送都是不允许出错的，而对数据的可靠性要求不那么高的应用可选择UDP传送；

b.应用的实时性。TCP协议在传送过程中要使用三次握手、重传确认等手段来保证数据传输的可靠性。使用TCP协议会有较大的延迟，因此不适合对实时性要求较高的应用，如VOIP、视频监控等。相反，UDP协议则在这些应用中能发挥很好的作用。

c.网络的可靠性。由于TCP协议的提出主要是解决网络的可靠性问题，它通过各种机制来减少错误发生的概率，因此，在网络状况不是很好的情况下需选用TCP协议（如在广域网等情况），但是若在网络状况很好的情况下（如局域网等）就不需要采用TCP协议，而建议选择UDP协议来减少网络负荷。