## 第一阶段

**1.Python是如何进行内存管理的？**

答:从三个方面来说,一对象的引用计数机制,二垃圾回收机制,三内存池机制

一、对象的引用计数机制

Python内部使用引用计数，来保持追踪内存中的对象，所有对象都有引用计数。

引用计数增加的情况：

1，一个对象分配一个新名称

2，将其放入一个容器中（如列表、元组或字典）

引用计数减少的情况：

1，使用del语句对对象别名显示的销毁

2，引用超出作用域或被重新赋值

sys.getrefcount( )函数可以获得对象的当前引用计数

多数情况下，引用计数比你猜测得要大得多。对于不可变数据（如数字和字符串），解释器会在程序的不同部分共享内存，以便节约内存。

二、垃圾回收

1，当一个对象的引用计数归零时，它将被垃圾收集机制处理掉。

2，当两个对象a和b相互引用时，del语句可以减少a和b的引用计数，并销毁用于引用底层对象的名称。然而由于每个对象都包含一个对其他对象的应用，因此引用计数不会归零，对象也不会销毁。（从而导致内存泄露）。为解决这一问题，解释器会定期执行一个循环检测器，搜索不可访问对象的循环并删除它们。

三、内存池机制

Python提供了对内存的垃圾收集机制，但是它将不用的内存放到内存池而不是返回给操作系统。

1，Pymalloc机制。为了加速Python的执行效率，Python引入了一个内存池机制，用于管理对小块内存的申请和释放。

2，Python中所有小于256个字节的对象都使用pymalloc实现的分配器，而大的对象则使用系统的malloc。

3，对于Python对象，如整数，浮点数和List，都有其独立的私有内存池，对象间不共享他们的内存池。也就是说如果你分配又释放了大量的整数，用于缓存这些整数的内存就不能再分配给浮点数。

**2.什么是lambda函数？它有什么好处?**

答：lambda 表达式，通常是在需要一个函数，但是又不想费神去命名一个函数的场合下使用，也就是指匿名函数

lambda函数：首要用途是指点短小的回调函数

lambda [arguments]:expression

>>> a=lambdax,y:x+y

>>> a(3,11)

**3.Python里面如何实现tuple和list的转换？**

答：直接使用tuple和list函数就行了，type()可以判断对象的类型

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

答：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（如果用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，并且递归的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deep.deepcopy()函数}

**7.介绍一下except的用法和作用？**

答：try…except…except…[else…][finally…]

执行try下的语句，如果引发异常，则执行过程会跳到except语句。对每个except分支顺序尝试执行，如果引发的异常与except中的异常组匹配，执行相应的语句。如果所有的except都不匹配，则异常会传递到下一个调用本代码的最高层try代码中。

try下的语句正常执行，则执行else块代码。如果发生异常，就不会执行

如果存在finally语句，最后总是会执行。

**8.Python中pass语句的作用是什么？**

答：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，whileFalse:pass

**9.介绍一下Python下range()函数的用法？**

答：列出一组数据，经常用在for in range()循环中

**10.如何用Python来进行查询和替换一个文本字符串？**

答：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）

>>> import re

>>>p=re.compile(‘blue|white|red’)

>>>print(p.sub(‘colour’,'blue socks and red shoes’))

colour socks and colourshoes

>>>print(p.sub(‘colour’,'blue socks and red shoes’,count=1))

colour socks and redshoes

subn()方法执行的效果跟sub()一样，不过它会返回一个二维数组，包括替换后的新的字符串和总共替换的数量

**11.Python里面match()和search()的区别？**

答：re模块中match(pattern,string[,flags]),检查string的开头是否与pattern匹配。

re模块中research(pattern,string[,flags]),在string搜索pattern的第一个匹配值。

>>>print(re.match(‘super’, ‘superstition’).span())

(0, 5)

>>>print(re.match(‘super’, ‘insuperable’))

None

>>>print(re.search(‘super’, ‘superstition’).span())

(0, 5)

>>>print(re.search(‘super’, ‘insuperable’).span())

(2, 7)

**12.用Python匹配HTML tag的时候，<.\*>和<.\*?>有什么区别？**

答：术语叫贪婪匹配( <.\*> )和非贪婪匹配(<.\*?> )

例如:

test

<.\*> :

test

<.\*?> :

**13.Python里面如何生成随机数？**

答：random模块

随机整数：random.randint(a,b)：返回随机整数x,a<=x<=b

random.randrange(start,stop,[,step])：返回一个范围在(start,stop,step)之间的随机整数，不包括结束值。

随机实数：random.random( ):返回0到1之间的浮点数

random.uniform(a,b):返回指定范围内的浮点数。

**14.有没有一个工具可以帮助查找python的bug和进行静态的代码分析？**

答：PyChecker是一个python代码的静态分析工具，它可以帮助查找python代码的bug, 会对代码的复杂度和格式提出警告

Pylint是另外一个工具可以进行codingstandard检查

**15.如何在一个function里面设置一个全局的变量？**

答：解决方法是在function的开始插入一个global声明：

def f()

global x

**16.单引号，双引号，三引号的区别**

答：单引号和双引号是等效的，如果要换行，需要符号(\),三引号则可以直接换行，并且可以包含注释

如果要表示Let’s go 这个字符串

单引号：s4 = ‘Let\’s go’

双引号：s5 = “Let’s go”

s6 = ‘I realy like“python”!’

这就是单引号和双引号都可以表示字符串的原因了

17.到底什么是Python？你可以在回答中与其他技术进行对比（也鼓励这样做）

答：Python是一种解释型语言。这就是说，与C语言和C的衍生语言不同，Python代码在运行之前不需要编译。其他解释型语言还包括PHP和Ruby。

Python是动态类型语言，指的是你在声明变量时，不需要说明变量的类型。你可以直接编写类似x=111和x="I'm a string"这样的代码，程序不会报错。

Python非常适合面向对象的编程（OOP），因为它支持通过组合（composition）与继承（inheritance）的方式定义类（class）。Python中没有访问说明符（access specifier，类似C++中的public和private），这么设计的依据是“大家都是成年人了”。

在Python语言中，函数是第一类对象（first-class objects）。这指的是它们可以被指定给变量，函数既能返回函数类型，也可以接受函数作为输入。类（class）也是第一类对象。

Python代码编写快，但是运行速度比编译语言通常要慢。好在Python允许加入基于C语言编写的扩展，因此我们能够优化代码，消除瓶颈，这点通常是可以实现的。numpy就是一个很好地例子，它的运行速度真的非常快，因为很多算术运算其实并不是通过Python实现的。

Python用途非常广泛——网络应用，自动化，科学建模，大数据应用，等等。它也常被用作“胶水语言”，帮助其他语言和组件改善运行状况。

Python让困难的事情变得容易，因此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。

**17.阅读下面的代码，写出A0，A1至An的最终值。**

A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))

A1 = range(10)

A2 = [i for i in A1 if i in A0]

A3 = [A0[s] for s in A0]

A4 = [i for i in A1 if i in A3]

A5 = {i:i\*i for i in A1}

A6 = [[i,i\*i] for i in A1]

答：A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}

A1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

A2 = []

A3 = [1, 3, 2, 5, 4]

A4 = [1, 2, 3, 4, 5]

A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]

**18.下面代码会输出什么：**

**def f(x,l=[]):**

**for i in range(x):**

**l.append(i\*i)**

**print (l)**

**f(2)f(3,[3,2,1])f(3)**

答：[0, 1]

[3, 2, 1, 0, 1, 4]

[0, 1, 0, 1, 4]

第一个函数调用十分明显，for循环先后将0和1添加至了空列表l中。l是变量的名字，指向内存中存储的一个列表。第二个函数调用在一块新的内存中创建了新的列表。l这时指向了新生成的列表。之后再往新列表中添加0、1、2和4。很棒吧。第三个函数调用的结果就有些奇怪了。它使用了之前内存地址中存储的旧列表。这就是为什么它的前两个元素是0和1了

**21.“猴子补丁”（monkey patching）指的是什么？这种做法好吗？**

答：“猴子补丁”就是指，在函数或对象已经定义之后，再去改变它们的行为。

举个例子：

import datetime

datetime.datetime.now = lambda: datetime.datetime(2012, 12, 12)

大部分情况下，这是种很不好的做法 - 因为函数在代码库中的行为最好是都保持一致。打“猴子补丁”的原因可能是为了测试。mock包对实现这个目的很有帮助。

**22. 这两个参数是什么意思：\*args，\*\*kwargs？我们为什么要使用它们？**

答：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。args和kwargs这两个标识符是约定俗成的用法，你当然还可以用\*bob和\*\*billy，但是这样就并不太妥。

**23.阅读下面的代码，它的输出结果是什么？**

**class A(object):**

**def go(self):**

**print "go A go!"**

**def stop(self):**

**print "stop A stop!"**

**def pause(self):**

**raise Exception("Not Implemented")**

**class B(A):**

**def go(self):**

**super(B, self).go()**

**print "go B go!"**

**class C(A):**

**def go(self):**

**super(C, self).go()**

**print "go C go!"**

**def stop(self):**

**super(C, self).stop()**

**print "stop C stop!"**

**class D(B,C):**

**def go(self):**

**super(D, self).go()**

**print "go D go!"**

**def stop(self):**

**super(D, self).stop()**

**print "stop D stop!"**

**def pause(self):**

**print "wait D wait!"**

**class E(B,C): pass**

**a = A()**

**b = B()**

**c = C()**

**d = D()**

**e = E()**

**# 说明下列代码的输出结果**

**a.go()**

**b.go()**

**c.go()**

**d.go()**

**e.go()**

**a.stop()**

**b.stop()**

**c.stop()**

**d.stop()**

**e.stop()**

**a.pause()**

**b.pause()**

**c.pause()**

**d.pause()**

**e.pause()**

答：输出结果以注释的形式表示：

a.go()

# go A go!

b.go()

# go A go!

# go B go!

c.go()

# go A go!

# go C go!

d.go()

# go A go!

# go C go!

# go B go!

# go D go!

e.go()

# go A go!

# go C go!

# go B go!

a.stop()

# stop A stop!

b.stop()

# stop A stop!

c.stop()

# stop A stop!

# stop C stop!

d.stop()

# stop A stop!

# stop C stop!

# stop D stop!

e.stop()

# stop A stop!

a.pause()

# ... Exception: Not Implemented

b.pause()

# ... Exception: Not Implemented

c.pause()

# ... Exception: Not Implemented

d.pause()

# wait D wait!

e.pause()

# ...Exception: Not Implemented

**24.阅读下面的代码，它的输出结果是什么？**

**class Node(object):**

**def \_\_init\_\_(self,sName):**

**self.\_lChildren = []**

**self.sName = sName**

**def \_\_repr\_\_(self):**

**return "<Node '{}'>".format(self.sName)**

**def append(self,\*args,\*\*kwargs):**

**self.\_lChildren.append(\*args,\*\*kwargs)**

**def print\_all\_1(self):**

**print self**

**for oChild in self.\_lChildren:**

**oChild.print\_all\_1()**

**def print\_all\_2(self):**

**def gen(o):**

**lAll = [o,]**

**while lAll:**

**oNext = lAll.pop(0)**

**lAll.extend(oNext.\_lChildren)**

**yield oNext**

**for oNode in gen(self):**

**print oNode**

**oRoot = Node("root")**

**oChild1 = Node("child1")**

**oChild2 = Node("child2")**

**oChild3 = Node("child3")**

**oChild4 = Node("child4")**

**oChild5 = Node("child5")**

**oChild6 = Node("child6")**

**oChild7 = Node("child7")**

**oChild8 = Node("child8")**

**oChild9 = Node("child9")**

**oChild10 = Node("child10")**

**oRoot.append(oChild1)**

**oRoot.append(oChild2)**

**oRoot.append(oChild3)**

**oChild1.append(oChild4)**

**oChild1.append(oChild5)**

**oChild2.append(oChild6)**

**oChild4.append(oChild7)**

**oChild3.append(oChild8)**

**oChild3.append(oChild9)**

**oChild6.append(oChild10)**

**# 说明下面代码的输出结果**

**oRoot.print\_all\_1()**

**oRoot.print\_all\_2()**

答：oRoot.print\_all\_1()会打印下面的结果

<Node 'root'>

<Node 'child1'>

<Node 'child4'>

<Node 'child7'>

<Node 'child5'>

<Node 'child2'>

<Node 'child6'>

<Node 'child10'>

<Node 'child3'>

<Node 'child8'>

<Node 'child9'>

oRoot.print\_all\_1()会打印下面的结果：

<Node 'root'>

<Node 'child1'>

<Node 'child2'>

<Node 'child3'>

<Node 'child4'>

<Node 'child5'>

<Node 'child6'>

<Node 'child8'>

<Node 'child9'>

<Node 'child7'>

<Node 'child10'>

print\_all\_1会以深度优先（depth-first）的方式遍历树(tree),而print\_all\_2则是宽度优先（width-first）

**25.简要描述Python的垃圾回收机制（garbage collection）。**

答：这里能说的很多。你应该提到下面几个主要的点：:

Python在内存中存储了每个对象的引用计数（reference count）。如果计数值变成0，那么相应的对象就会小时，分配给该对象的内存就会释放出来用作他用。

偶尔也会出现引用循环（reference cycle）。垃圾回收器会定时寻找这个循环，并将其回收。举个例子，假设有两个对象o1和o2，而且符合o1.x == o2和o2.x == o1这两个条件。如果o1和o2没有其他代码引用，那么它们就不应该继续存在。但它们的引用计数都是1。

Python中使用了某些启发式算法（heuristics）来加速垃圾回收。例如，越晚创建的对象更有可能被回收。对象被创建之后，垃圾回收器会分配它们所属的代（generation）。每个对象都会被分配一个代，而被分配更年轻代的对象是优先被处理的。

**26.将下面的函数按照执行效率高低排序。它们都接受由0至1之间的数字构成的列表作为输入。这个列表可以很长。一个输入列表的示例如下：[random.random() for i in range(100000)]。你如何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 lIn 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)]

答：按执行效率从高到低排列：f2、f1和f3。要证明这个答案是对的，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 cProfile

lIn = [random.random() for i in range(100000)]

cProfile.run('f1(lIn)')

cProfile.run('f2(lIn)')

cProfile.run('f3(lIn)')

**27.你是如何理解Python装饰器的？**

答：装饰器是一个工厂函数，接受一个函数作为参数，然后返回一个新函数，其闭包中包含被装饰的函数。有了装饰器，可以提取大量函数中与本身功能无关的类似代码 ( 这块在Flask中用于定义路由的@app.route，就是一个很好的例子)，达到代码重用的目的。可应用于插入日志、性能测试、事务处理等方面

**28.python写的CGI脚本，用print为什么不是打印到控制台，而是发送到客户端？**

答：print的输出是标准输出，可以指向任何可以输出的流，包括http输出

**29.什么是闭包？如何用Python写闭包？**

闭包可以实现先将一个参数传递给一个函数，而并不立即执行，以达到延迟求值的目的。满足以下三个条件：必须有一个内嵌函数；内嵌函数必须引用外部函数中变量；外部函数返回值必须是内嵌函数。

def delay\_fun(x, y):

def caculator():

return x+y

return caculator

print('返回一个求和的函数，并不求和')

msum = delay\_fun(3,4)

print('调用并求和:')

print(msum())

**30.Python的参数传递是值传递还是引用传递**

**举例说明Python函数参数传递的几种形式，并说明函数传参是值传递还是引用传递**

1).Python的参数传递有：

位置参数

默认参数，

可变参数,

关键字参数

2).函数的传值到底是值传递还是引用传递，要分情况

a.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

b.可变参数是用引用传递的

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.

**31.字符串格式化:%和.format的区别**

字符串的format函数非常灵活，很强大,可以接受的参数不限个数，并且位置可以不按顺序，而且有较为强大的格式限定符(比如:填充，对齐,精度等)

**32.写一个函数, 输入一个字符串, 返回倒序排列的结果**

输入: string\_reverse(‘abcdef’), 返回: ‘fedcba’,写出你能想到的多种方法

1).利用字符串本身的翻转

def string\_reverse1(text='abcdef'):

return text[::-1]

2).把字符串变成列表，用列表的reverse函数

3).新建一个列表，从后往前取

4).利用双向列表deque中的extendleft函数

5).递归

**33.按升序合并如下两个list, 并去除重复的元素**

list1 = [2, 3, 8, 4, 9, 5, 6]

list2 = [5, 6, 10, 17, 11, 2]

1).最简单的方法用set

list3=list1+list2

print set(list3)

2).递归

先选一个中间数，然后一边是小的数字，一边是大的数字，然后再循环递归，排完序(是不是想起了c里面的冒泡)

**34. 以下的代码的输出将是什么? 说出你的答案并解释**

class Parent(object):

x = 1

class Child1(Parent):

pass

class Child2(Parent):

pass

print Parent.x, Child1.x, Child2.x

Child1.x = 2

print Parent.x, Child1.x, Child2.x

Parent.x = 3

print Parent.x, Child1.x, Child2.x

>>

1 1 1

1 2 1

3 2 3

解答:

使你困惑或是惊奇的是关于最后一行的输出是 3 2 3 而不是 3 2 1。为什么改变了 Parent.x 的值还会改变 Child2.x 的值，但是同时 Child1.x 值却没有改变？

这个答案的关键是，在 Python中，类变量在内部是作为字典处理的。如果一个变量的名字没有在当前类的字典中发现，将搜索祖先类（比如父类）直到被引用的变量名被找到.

首先，在父类中设置 x = 1 会使得类变量 x 在引用该类和其任何子类中的值为 1。这就是因为第一个 print 语句的输出是 1 1 1

然后，如果任何它的子类重写了该值（例如，我们执行语句 Child1.x = 2）该值仅仅在子类中被改变。这就是为什么第二个 print 语句的输出是 1 2 1

最后，如果该值在父类中被改变（例如，我们执行语句 Parent.x = 3），这个改变会影响到任何未重写该值的子类当中的值（在这个示例中被影响的子类是 Child2）。这就是为什么第三个 print 输出是 3 2 3

**35. 下面的代码会不会报错**

list = ['a', 'b', 'c', 'd', 'e']

print list[10:]

不会报错，而且会输出一个 []，并且不会导致一个 IndexError

解答:

当试图访问一个超过列表索引值的成员将导致 IndexError（比如访问以上列表的 list[10]）。尽管如此，试图访问一个列表的以超出列表长度数作为开始索引的切片将不会导致 IndexError，并且将仅仅返回一个空列表

一个讨厌的小问题是它会导致出现 bug ，并且这个问题是难以追踪的，因为它在运行时不会引发错误，吐血啊~~

**36. 说出下面list1,list2,list3的输出值**

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

>>

list1 = [10, 'a']

list2 = [123]

list3 = [10, 'a']

许多人会错误的认为 list1 应该等于 [10] 以及 list3 应该等于 ['a']。认为 list 的参数会在 extendList 每次被调用的时候会被设置成它的默认值 []。

尽管如此，实际发生的事情是，新的默认列表仅仅只在函数被定义时创建一次。随后当 extendList 没有被指定的列表参数调用的时候，其使用的是同一个列表。这就是为什么当函数被定义的时候，表达式是用默认参数被计算，而不是它被调用的时候。

因此，list1 和 list3 是操作的相同的列表。而list2是操作的它创建的独立的列表（通过传递它自己的空列表作为list参数的值）

所以这一点一定要切记切记.下面我们把list置为None就可以避免一些麻烦了

1. **python的底层网络交互模块有哪些？**

答：socket, urllib,urllib3 , requests, grab, pycurl

**38. python网络交互时，二进制打包解包的模块有哪些**

答：打包：pack(), 解包：upk()

**39. python的测试框架有哪些？试列出常用的3个或更多**

答：unittest, nose,unittest2, doctest, pytest

1. **一行把[1,2,3,1,2,3] 中的重复元素剔除**

答：list(set([1,2,3,1,2,3]))

## 第二阶段

**41、Python和多线程（multi-threading）。这是个好主意吗？列举一些让Python代码以并行方式运行的方法**

答：Python并不支持真正意义上的多线程。Python中提供了多线程包，但是如果你想通过多线程提高代码的速度，使用多线程包并不是个好主意。Python中有一个被称为Global Interpreter Lock（GIL）的东西，它会确保任何时候你的多个线程中，只有一个被执行。线程的执行速度非常之快，会让你误以为线程是并行执行的，但是实际上都是轮流执行。经过GIL这一道关卡处理，会增加执行的开销。这意味着，如果你想提高代码的运行速度，使用threading包并不是一个很好的方法。

不过还是有很多理由促使我们使用threading包的。如果你想同时执行一些任务，而且不考虑效率问题，那么使用这个包是完全没问题的，而且也很方便。但是大部分情况下，并不是这么一回事，你会希望把多线程的部分外包给操作系统完成（通过开启多个进程），或者是某些调用你的Python代码的外部程序（例如Spark或Hadoop），又或者是你的Python代码调用的其他代码（例如，你可以在Python中调用C函数，用于处理开销较大的多线程工作）。

**42.谈谈你对Python中多线程和多进程的理解**

答：进程：是资源分配的最小单位，创建和销毁开销较大；

线程：是CPU调度的最小单位，开销小，切换速度快；

操作系统将CPU时间片分配给多个线程，每个线程在指定放到时间片内完成。操作系统不断从一个线程切换到另一个线程执行，宏观上看就好像是多个线程一起执行。

Python中由于全局锁 (GIL)的存在导致，同一时间只有一个获得GIL的线程在跑，其他线程则处于等待状态，这导致了多线程只是在做分时切换，并不能利用多核。

多线程与多进程的区别：（1）多进程中同一个变量各自有一份拷贝在每个进程中，互不影响；（2）多线程中，所有变量都由所有线程共享，任何一个变量都可被任何一个线程修改。线程之间共享数据的最大危险在于多个线程同时更改一个变量，把内容改乱。

**43.什么是数据库的索引？索引的优缺点有哪些？**

答：1. 什么是索引：

索引就像是书的目录，是与表或视图关联的磁盘上结构，可以加快从表或视图中检索行的速度。索引中包含由表或视图中的一列或多列生成的键。这些键存储在一个结构（BTree）中，使SQL可以快速有效地查找与键值关联的行。

2. 为什么要建立索引，即索引的优点：

建立索引的列可以保证行的唯一性，生成唯一的rowId

建立索引可以有效缩短数据的检索时间

建立索引可以加快表与表之间的连接

为用来排序或者是分组的字段添加索引可以加快分组和排序顺序

3. 索引的缺点：

创建索引和维护索引需要时间成本，这个成本随着数据量的增加而加大

创建索引和维护索引需要空间成本，每一条索引都要占据数据库的物理存储空间，数据量越大，占用空间也越大（数据表占据的是数据库的数据空间）

会降低表的增删改的效率，因为每次增删改索引需要进行动态维护，导致时间变长

**44.你认为什么样的表和列需要建立索引？**

答：总的来说就是数据量大的，经常进行查询操作的表要建立索引

表与表连接用于多表联合查询的约束条件的字段应当建立索引

用于排序的字段可以添加索引，用于分组的字段应当视情况看是否需要添加索引。

添加多列索引的时候，对应的多条件查询可以触发该索引的同时，索引最左侧的列的单条件查询也可以触发。

如果有些表注定只会进行查询所有，也就没必要添加索引，因为查询全部只能进行全量搜索即扫描全表。

**45.建立索引需要遵守什么原则？**

答：表中字段建立索引应该遵循几个原则：

越小的数据类型通常更好：越小的数据类型通常在磁盘、内存中都需要更少的空间，处理起来更快。

简单的数据类型更好：整型数据比起字符，处理开销更小，因为字符串的比较更复杂，处理起来也更耗时。

尽量避免NULL：应该指定列为NOT NULL。含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。你应该用0、一个特殊的值或者一个空串代替空值。

对非唯一的字段，例如“性别”这种大量重复值的字段，增加索引也没有什么意义，所以索引的建立应当更多的选取唯一性更高的字段。

**46.简单说说索引的工作原理**

答：索引的原理大致概括为以空间换时间，数据库在未添加索引的时候进行查询默认的是进行全量搜索，也就是进行全局扫描，有多少条数据就要进行多少次查询，然后找到相匹配的数据就把他放到结果集中，直到全表扫描完。而建立索引之后，会将建立索引的KEY值放在一个n叉树上（BTree）。因为B树的特点就是适合在磁盘等直接存储设备上组织动态查找表，每次以索引进行条件查询时，会去树上根据key值直接进行搜索

**47.谈谈你对MySQL中字符集和排序规则的理解**

答： 数据库需要适应各种语言和字符就需要支持不同的字符集（Character Set），每种字符集也有各自的排序规则（Collation）。

字符集，即用于定义字符在数据库中的编码的集合。 常见的字符集：utf-8 gbk 等。

数据库中的排序规则用来定义字符在进行排序和比较的时候的一种规则。 常见的如下：

（1） utf8\_general\_cs 和 utf8\_general\_ci （后缀"\_cs"或者"\_ci"意思是区分大小写和不区分大小写（Case Sensitive & Case Insensitve））

（2） utf8\_bin 规定每个字符串用二进制编码存储，区分大小写，可以直接存储二进制的

说明：所为排序规则，就是指字符比较时是否区分大小写，以及是按照字符编码进行比较还是直接用二进制数据比较。

**48.varchar与char的区别是什么？大小限制？utf8字符集下varchar最多能存多少个字符**

答： char和varchar最大的不同就是一个是固定长度,一个是可变长度.由于是可变长度,因此存储的是实际字符串再加上一个记录字符串长度的字节。如果分配给char或varchar列的值超过 列的最大长度,则对值进行裁剪.

varchar(M)和char(M),M都表示字符数.varchar的最大长度为65535个字节,不同的编码所对应的最大可存储的字符数不同.char最多可以存放255个字符,不同的编码最大可用字节数不同.

字符类型若为utf8，每个字符最多占3个字节，varchar最大长度不能超过21845。

**49.primary key和unique的区别？**

答：1、primary key = unique + not null

2、唯一约束和主键一样都是约束的范畴，而且都可以作为外键的参考，不同的是，一张表只能有一个主键

3、主键和唯一约束的创建需要依靠索引，如果在创建主键或唯一约束的时候没有已经建好的索引可以使用的话，Oracle会自动建立一个唯一的索引。

**50.是否有必要使用外键？为什么不用外键？**

正方（需要）

1.数据一致性

由数据库自身保证数据一致性，完整性，更可靠，因为程序很难100％保证数据 的完整性，而用外键即使在数据库服务器当机或者出现其他问题的时候，也能够最大限度的保证数据的一致性和完整性。

2.ER图可靠性

有主外键的数据库设计可以增加ER图的可读性，这点在数据库设计时非常重要。

3.使设计更全面

外键在一定程度上说明的业务逻辑，会使设计周到具体全面

4.级联性能未必最低

除非能证明触发器的性能和可维护性比外键更优，否则凭什么一定要否定外键呢？

5.程序能完全保证安全性吗？

使用应用程序来维护数据完整性，则是严重降低数据库安全性的一种做法。连接字符串里如果有uid和pwd或者类似的这两个东西，可以使用工具获取你数据库的帐号。因为每次建立数据库联接的时候都会在网线上以标准格式传递这些信息。从此编码内的任何保证都是空的。

6.导致冗余

不使用外键，会导致数据冗余，在级联最底层的表可能会重复好几层的数据 必然导致最底层的表数据量翻倍，IO瓶颈是数据库性能瓶颈之一。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

反方（不需要）

1.程序逻辑

某些程序逻辑中，程序的逻辑已经足够保证完整性，我会在存储过程或包等地方做严谨的判断；

2.性能问题

这是很多人不喜欢用的关键原因，比如一个业务流水表，频繁插入数据，如果这个表身上有3外键，那么每次插入一条，就必须对这3个外键对应的3个表做相应的查找判断有无对应数据，如果这3个表也很大，那就这3个表的判断时间就很常，虽然外键指向的关联表的字段肯定是索引，但是我觉得很多时候，这样的判断本来就在程序里控制好了，通过外键再判断一次，就是降低性能；而且其实有的地方判不判断也无所谓的，但是用了外键，就必须化时间去判断，无论oracle内部多么优化外键对于数据的检索速度，它总是一个不小的消耗；

3.维护麻烦

很多公司的软件都是定制的，这种定制的东西，随意性相对较大，项目开发实施过程中，需要经常对表修修补补；还有就是业务逻辑有bug或者其他情况，需要经常手工维护数据，有错综复杂的外键关联着，很是麻烦；

4.外键定死了先后生成关系

外键定死了两个表之间数据的先后生成关系，最常见的是单据主从表，有的时候，在生成单据的时候，是先生成明细，再生成主表；如果钉死了外键，这个就没法实现；

PS: 当然有些关键的业务，确实需要外键；

结论：

1，在大型系统中（性能要求不高，安全要求高），使用外键；在大型系统中（性能要求高，安全自己控制），不用外键；小系统随便，最好用外键。

2，用外键要适当，不能过分追求

3，不用外键而用程序控制数据一致性和完整性时，应该写一层来保证，然后个个应用通过这个层来访问数据库。

**51.MySQL中外键是否要添加索引？**

答：如果没有索引的话，MySQL每次在你插入，修改数据的时候都需要全表扫描主表中所有的记录以确定是否遵守外键约束，效率上显示不合适，所以要添加。

**52、socket通信过程？**  
服务器调用socket()、bind()、listen()完成初始化后，调用accept()阻塞等待，处于监听端口的状态，客户端调用socket()初始化后，调用connect()发出SYN段并阻塞等待服务器应答，服务器应答一个SYN-ACK段，客户端收到后从connect()返回，同时应答一个ACK段，服务器收到后从accept()返回

****

**53、索引的缺点？**  
第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。   
第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。   
第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度

**54. 用一条SQL语句 查询出每门课都大于80分的学生姓名。表scores如下：**

name   kecheng   fenshu

张三     语文       81

张三     数学       75

李四     语文       76

答：SELECT DISTINCT name FROM grade WHERE name NOT IN(SELECT DISTINCTname FROM grade WHERE score <=80);

更简单的：

SELECT name FROMgrade GROUP BY name HAVING MIN(score) > 80；

## 第三阶段

**55. 浏览器渲染原理解析**

答：

1、首先渲染引擎下载HTML，解析生成DOM Tree

2、遇到css标签或JS脚本标签就新起线程去下载他们，并继续构建DOM。（其中css是异步下载同步执行）浏览器引擎通过 DOM Tree 和 CSS Rule Tree 构建 Rendering Tree

3、 通过 CSS Rule Tree 匹配 DOM Tree 进行定位坐标和大小，这个过程称为 Flow 或 Layout 。

4、最终通过调用Native GUI 的 API 绘制网页画面的过程称为 Paint 。

当用户在浏览网页时进行交互或通过 js 脚本改变页面结构时，以上的部分操作有可能重复运行，此过程称为 Repaint 或 Reflow。 重排是指dom树发生结构变化后，需要重新构建dom结构。 重绘是指dom节点样式改变，重新绘制。 重排一定会带来重绘，重绘不一定有重排。

如何减少浏览器重排：将需要多次重排的元素，position属性设为absolute或fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。

**56. 什么是盒子模型？**

在网页中，一个元素占有空间的大小由几个部分构成，其中包括元素的内容（content），元素的内边距（padding），元素的边框（border），元素的外边距（margin）四个部分。这四个部分占有的空间中，有的部分可以显示相应的内容，而有的部分只用来分隔相邻的区域或区域。4个部分一起构成了css中元素的盒模型

清除浮动有几种方式？各自的优缺点是？

1,父级定义height（优点：代码少，简单易掌握；缺点：只适合固定高度元素，无法高度自适应）

2，结尾处使用空标签清除浮动：Clear:both（优点：理论上能清除所有标签的浮动，简单代码少浏览器支持好；缺点：增加了无意义的标签）

3，父级定义伪类after和zoom（优点：浏览器支持好；缺点：代码多，两句代码结合使用才能让主流浏览器都支持）

4，父级定义overflow:hidden（优点：代码少，简单，浏览器支持好；缺点：必须定义width或者zoom，同时不能定义height，不能和position配合使用）

5，父级定义overflow:auto（优点：代码少，简单，浏览器支持好；缺点：必须定义width或者zoom，同时不能定义height，内部元素高度超过父级会出现滚动条）

还有，父级一起浮动，父级定义display:table，结尾加br的clear：both等

**57. CSS选择器都有哪些**

派生选择器（用HTML标签申明）

id选择器（用DOM的ID申明）

类选择器（用一个样式类名申明）

属性选择器（用DOM的属性申明，属于CSS2，IE6不支持，不常用，不知道就算了）

除了前3种基本选择器，还有一些扩展选择器，包括

后代选择器（利用空格间隔，比如div .a{ }）

群组选择器（利用逗号间隔，比如p,div,#a{ }）

**58. 超链接访问过后hover样式就不出现的问题是什么？如何解决？**

答案：被点击访问过的超链接样式不在具有hover和active了,解决方法是改变CSS属性的排列顺序: L-V-H-A（link,visited,hover,active）

**59. 简述一下src与href的区别**

href 是指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，用于超链接。

src是指向外部资源的位置，指向的内容将会嵌入到文档中当前标签所在位置；在请求src资源时会将其指向的资源下载并应用到文档内，例如js脚本，img图片和frame等元素。当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，图片和框架等元素也如此，类似于将所指向资源嵌入当前标签内。这也是为什么将js脚本放在底部而不是头部。

**60. 什么是CSS Hack?**

一般来说是针对不同的浏览器写不同的CSS,就是 CSS Hack。 IE浏览器Hack一般又分为三种，条件Hack、属性级Hack、选择符Hack

**61. 简述同步和异步的区别**

同步是阻塞模式，异步是非阻塞模式。 同步就是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么这个进程将会一直等待下去，直到收到返回信息才继续执行下去； 异步是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。

**62. px和em的区别**

px和em都是长度单位，区别是，px的值是固定的，指定是多少就是多少，计算比较容易。em得值不是固定的，并且em会继承父级元素的字体大小。 浏览器的默认字体高都是16px。所以未经调整的浏览器都符合: 1em=16px。那么12px=0.75em, 10px=0.625em

**63. 怎样添加、移除、移动、复制、创建和查找节点？**

1）创建新节点 createDocumentFragment() //创建一个DOM片段 createElement() //创建一个具体的元素 createTextNode() //创建一个文本节点

2）添加、移除、替换、插入 appendChild() //添加 removeChild() //移除 replaceChild() //替换 insertBefore() //插入

3）查找 getElementsByTagName() //通过标签名称 getElementsByName() //通过元素的Name属性的值 getElementById() //通过元素Id，唯一性

**64. 谈谈你对箭头函数的理解**

箭头函数不属于普通的 function，所以没有独立的上下文。箭头函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象。 由于箭头函数没有自己的this，函数对象中的call、apply、bind三个方法，无法"覆盖"箭头函数中的this值。 箭头函数没有原本(传统)的函数有的隐藏arguments对象。 箭头函数不能当作generators使用，使用yield会产生错误。

在以下场景中不要使用箭头函数去定义：

定义对象方法、定义原型方法、定义构造函数、定义事件回调函数。

箭头函数里不但没有 this，也没有 arguments, super ……

**65. js继承方式及其优缺点**

原型链继承的缺点

一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

**66. 在Javascript中什么是伪数组？如何将伪数组转化为标准数组？**

伪数组（类数组）：无法直接调用数组方法或期望length属性有什么特殊的行为，但仍可以对真正数组遍历方法来遍历它们。典型的是函数的argument参数，还有像调用getElementsByTagName,document.childNodes之类的,它们都返回NodeList对象都属于伪数组。可以使用Array.prototype.slice.call(fakeArray)将数组转化为真正的Array对象。

**67. 请描述一下cookies，sessionStorage和localStorage的区别**

sessionStorage用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此sessionStorage不是一种持久化的本地存储，仅仅是会话级别的存储。而localStorage用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

**68. web storage和cookie的区别**

Web Storage的概念和cookie相似，区别是它是为了更大容量存储设计的。Cookie的大小是受限的，并且每次你请求一个新的页面的时候Cookie都会被发送过去，这样无形中浪费了带宽，另外cookie还需要指定作用域，不可以跨域调用。 除此之外，Web Storage拥有setItem,getItem,removeItem,clear等方法，不像cookie需要前端开发者自己封装setCookie，getCookie。但是Cookie也是不可以或缺的：Cookie的作用是与服务器进行交互，作为HTTP规范的一部分而存在 ，而Web Storage仅仅是为了在本地“存储”数据而生。

**69. 一次完整的HTTP事务是怎样的一个过程？**

基本流程： a. 域名解析 b. 发起TCP的3次握手 c. 建立TCP连接后发起http请求 d. 服务器端响应http请求，浏览器得到html代码 e. 浏览器解析html代码，并请求html代码中的资源 f. 浏览器对页面进行渲染呈现给用户

**70. 单页面应用的优缺点**

优点： 1.用户体验好，快，内容的改变不需要重新加载整个页面 2.基于上面一点，SPA相对服务器压力小 3.没有页面切换，就没有白屏阻塞

缺点： 1、不利于SEO 2、初次加载耗时增多 3、导航不可用 4、容易造成css命名冲突等 5、页面复杂度提高很多，复杂逻辑难度成倍

**71. Ajax的get和post方式的区别？**

1、使用Get请求时,参数在URL中显示,而使用Post方式是放在虚拟载体里面,不会显示出来。

2、 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。两种方式的参数都可以用Request来获得。

3、get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。但理论上，因服务器的不同而异.

4、get安全性非常低，post安全性较高。

5、get请求需注意缓存问题,post请求不需担心这个问题。

6、post请求必须设置Content-Type值为application/x-form-www-urlencoded

7、发送请求时,因为get请求的参数都在url里,所以send函数发送的参数为null,而post请求在使用send方法时,却需赋予其参数

**72. Ajax请求时，如何解析json数据？**

使用eval 或parse 鉴于安全性考虑 使用parse更靠谱

**73. 解释jsonp解决跨域问题的原理**

1、Ajax直接请求普通文件存在跨域无权限访问的问题，无论你是静态页面、动态网页、web服务、WCF，只要是跨域请求，一律不准；

2、但是Web页面上调用js文件时则不受是否跨域的影响（凡是拥有"src"这个属性的标签都拥有跨域的能力，比如<script>、<img>、<iframe>）；

3、于是可以判断，当前阶段如果想通过纯web端（ActiveX控件、服务端代理、属于未来的HTML5之Websocket等方式不算）跨域访问数据就只有一种可能，那就是在远程服务器上设法把数据装进js格式的文件里，供客户端调用和进一步处理；

4、有一种叫做JSON的纯字符数据格式可以简洁的描述复杂数据，更妙的是JSON还被js原生支持，所以在客户端几乎可以随心所欲的处理这种格式的数据

5、web客户端通过与调用脚本一模一样的方式，来调用跨域服务器上动态生成的js格式文件（一般以JSON为后缀），显而易见，服务器之所以要动态生成JSON文件，目的就在于把客户端需要的数据装入进去。

6、客户端在对JSON文件调用成功之后，也就获得了自己所需的数据，剩下的就是按照自己需求进行处理和展现了，这种获取远程数据的方式看起来非常像AJAX，但其实并不一样。

7、为了便于客户端使用数据，逐渐形成了一种非正式传输协议，人们把它称作JSONP，该协议的一个要点就是允许用户传递一个callback参数给服务端，然后服务端返回数据时会将这个callback参数作为函数名来包裹住JSON数据，这样客户端就可以随意定制自己的函数来自动处理返回数据了。

**74. Jsonp和ajax的区别**

1、ajax和jsonp这两种技术在调用方式上“看起来”很像，目的也一样，都是请求一个url，然后把服务器返回的数据进行处理，因此jquery和ext等框架都把jsonp作为ajax的一种形式进行了封装；

2、但ajax和jsonp其实本质上是不同的东西。ajax的核心是通过XmlHttpRequest获取非本页内容，而jsonp的核心则是动态添加<script>标签来调用服务器提供的js脚本。

3、所以说，其实ajax与jsonp的区别不在于是否跨域，ajax通过服务端代理一样可以实现跨域，jsonp本身也不排斥同域的数据的获取。

4、还有就是，jsonp是一种方式或者说非强制性协议，如同ajax一样，它也不一定非要用json格式来传递数据，如果你愿意，字符串都行，只不过这样不利于用jsonp提供公开服务。

总而言之，jsonp不是ajax的一个特例，哪怕jquery等巨头把jsonp封装进了ajax，也不能改变着一点！

**75. 说一下对 Django 和 Tornado 的了解**

Django源自一个在线新闻 Web站点，于 2005 年以开源的形式被释放出来。

Django 框架的核心组件有：

用于创建模型的对象关系映射为最终用户设计的完美管理界面一流的 URL 设计设计者友好的模板语言缓存系统等等.它鼓励快速开发,并遵循MVC设计。Django遵守 BSD版权，最新发行版本是Django1.4，于2012年03月23日发布.Django的主要目的是简便、快速的开发数据库驱动的网站。它强调代码复用,多个组件可以很方便的以“插件”形式服务于整个框架，Django有许多功能强大的第三方插件，你甚至可以很方便的开发出自己的工具包。这使得Django具有很强的可扩展性。它还强调快速开发和DRY(Do Not RepeatYourself)原则。

Tornado是 FriendFeed使用的可扩展的非阻塞式 web 服务器及其相关工具的开源版本。这个 Web 框架看起来有些像 web.py 或者 Google 的 webapp，不过为了能有效利用非阻塞式服务器环境，这个 Web 框架还包含了一些相关的有用工具和优化。

Tornado 和现在的主流 Web 服务器框架（包括大多数Python 的框架）有着明显的区别：它是非阻塞式服务器，而且速度相当快。得利于其 非阻塞的方式和对epoll的运用，Tornado 每秒可以处理数以千计的连接，这意味着对于实时 Web服务来说，Tornado 是一个理想的 Web 框架。我们开发这个 Web 服务器的主要目的就是为了处理 FriendFeed 的实时功能 ——在 FriendFeed 的应用里每一个活动用户都会保持着一个服务器连接。（关于如何扩容 服务器，以处理数以千计的客户端的连接的问题。

**76. 解释下django-debug-toolbar的使用**

使用django开发站点时，可以使用django-debug-toolbar来进行调试。在settings.py中添加’debug\_toolbar.middleware.DebugToolbarMiddleware’到项目的MIDDLEWARE\_CLASSES 内。

**77. 解释下Http协议/Http协议主要特点**

HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

HTTP协议的主要特点可概括如下：

1.支持客户/服务器模式。

2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

**78. 解释下Http请求头和常见响应状态码**

Accept:指浏览器或其他客户可以接爱的MIME文件格式。可以根据它判断并返回适当的文件格式。

Accept-Charset：指出浏览器可以接受的字符编码。英文浏览器的默认值是ISO-8859-1.

Accept-Language：指出浏览器可以接受的语言种类，如en或en-us，指英语。

Accept-Encoding：指出浏览器可以接受的编码方式。编码方式不同于文件格式，它是为了压缩文件并加速文件传递速度。浏览器在接收到Web响应之后先解码，然后再检查文件格式。

Cache-Control：设置关于请求被代理服务器存储的相关选项。一般用不到。

Connection：用来告诉服务器是否可以维持固定的HTTP连接。HTTP/1.1使用Keep-Alive为默认值，这样，当浏览器需要多个文件时(比如一个HTML文件和相关的图形文件)，不需要每次都建立连接。

Content-Type：用来表名request的内容类型。可以用HttpServletRequest的getContentType()方法取得。

Cookie：浏览器用这个属性向服务器发送Cookie。Cookie是在浏览器中寄存的小型数据体，它可以记载和服务器相关的用户信息，也可以用来实现会话功能。

状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：

1xx：指示信息–表示请求已接收，继续处理

2xx：成功–表示请求已被成功接收、理解、接受

3xx：重定向–要完成请求必须进行更进一步的操作

4xx：客户端错误–请求有语法错误或请求无法实现

5xx：服务器端错误–服务器未能实现合法的请求

常见状态代码、状态描述、说明：

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常