第三版：Python 80 道

什么是 Python？为什么它会如此流行？

Python 是一种解释的、高级的、通用的编程语言。

Python 的设计理念是通过使用必要的空格与空行，增强代码的可读性。

它之所以受欢迎，就是因为它具有简单易用的语法。

为什么 Python 执行速度慢，我们如何改进它？

Python 代码执行缓慢的原因，是因为它是一种解释型语言。它的代码在运行时进行解释，而不是编译为本地语言。

为了提高 Python 代码的速度，我们可以使用 CPython、Numba，或者我们也可以对代码进行一些修改。

1、 减少内存占用。

2、 使用内置函数和库。

3、 将计算移到循环外。

4、 保持小的代码库。

5、 避免不必要的循环

Python 有什么特点?

1、 易于编码

2、 免费和开源语言

3、 高级语言

4、 易于调试

5、 OOPS 支持

6、 大量的标准库和第三方模块

7、 可扩展性(我们可以用 C 或 C++编写 Python 代码)

8、 用户友好的数据结构

Python 有哪些应用?

1、 Web 开发

2、 桌面 GUI 开发

3、 人工智能和机器学习

4、 软件开发

5、 业务应用程序开发

6、 基于控制台的应用程序

7、 软件测试

8、 Web 自动化

9、 基于音频或视频的应用程序

10、 图像处理应用程序

Python 的局限性?

1、 速度

2、 移动开发

3、 内存消耗(与其他语言相比非常高)

4、 两个版本的不兼容(2，3)

5、 运行错误(需要更多测试，并且错误仅在运行时显示)

6、 简单性

Python 代码是如何执行的?

首先，解释器读取 Python 代码并检查是否有语法或格式错误。

如果发现错误，则暂停执行。如果没有发现错误，则解释器会将 Python 代码转换为等效形式或字节代码。

然后将字节码发送到 Python 虚拟机(PVM)，这里 Python 代码将被执行，如果发现任何错误，则暂停执行，否则结果将显示在输出窗口中。

![90_1.png][90_1.png]

如何在 Python 中管理内存?

Python 内存由 Python 的私有 headspace 管理。

所有的 Python 对象和数据结构都位于一个私有堆中。私用堆的分配由 Python 内存管理器负责。

Python 还内置了一个的垃圾收集器，可以回收未使用的内存并释放内存，使其可用于 headspace。

解释 Python 的内置数据结构?

Python 中主要有四种类型的数据结构。

列表：列表是从整数到字符串甚至另一个列表的异构数据项的集合。列表是可变的。列表完成了其他语言中大多数集合数据结构的工作。列表在[ ]方括号中定义。

例如：a = [1,2,3,4]

集合：集合是唯一元素的无序集合。集合运算如联合|，交集&和差异，可以应用于集合。集是不可变的。()用于表示一个集合。

例如：a = {1,2,3,4}

元组：Python 元组的工作方式与 Python 列表完全相同，只是它们是不可变的。() 用于定义元组。

例如：a =（1,2,3,4）

字典：字典是键值对的集合。它类似于其他语言中的 hash map。在字典里，键是唯一且不可变的对象。

例如：a = {'number'：[1,2,3,4]}

解释//、%、* *运算符?

//(Floor Division)-这是一个除法运算符，它返回除法的整数部分。

例如：5 // 2 = 2

%(模数)-返回除法的余数。

例如：5 % 2 = 1

**(幂)-它对运算符执行指数计算。a ** b 表示 a 的 b 次方。

例如：5 ** 2 = 25、5 ** 3 = 125

Python 中的单引号和双引号有什么区别?

在 Python 中使用单引号(' ')或双引号(" ")是没有区别的，都可以用来表示一个字符串。

这两种通用的表达方式，除了可以简化程序员的开发，避免出错之外，还有一种好处，就是可以减少转义字符的使用，使程序看起来更简洁，更清晰。

Python 中 append，insert 和 extend 的区别?

append：在列表末尾添加新元素。

insert：在列表的特定位置添加元素。

extend：通过添加新列表来扩展列表。

```
numbers = [1,2,3,4,5]
numbers.append(6)
print(numbers)
>[1,2,3,4,5,6]

## insert(position,value)
numbers.insert(2,7)
print(numbers)
>[1,2,7,4,5,6]

numbers.extend([7,8,9])
print(numbers)
>[1,2,7,4,5,6,7,8,9]

numbers.append([4,5])
>[1,2,7,4,5,6,7,8,9,[4,5]]
```

break、continue、pass 是什么?

break：在满足条件时，它将导致程序退出循环。

continue：将返回到循环的开头，它使程序在当前循环迭代中的跳过所有剩余语句。

pass：使程序传递所有剩余语句而不执行。

区分 Python 中的 remove，del 和 pop?

remove：将删除列表中的第一个匹配值，它以值作为参数。

del：使用索引删除元素，它不返回任何值。

pop：将删除列表中顶部的元素，并返回列表的顶部元素。

```
numbers = [1,2,3,4,5]
numbers.remove(5)
> [1,2,3,4]

del numbers[0]
>[2,3,4]

numbers.pop()
>4
```

什么是 switch 语句。如何在 Python 中创建 switch 语句?

switch 语句是实现多分支选择功能，根据列表值测试变量。

switch 语句中的每个值都被称为一个 case。

在 Python 中，没有内置 switch 函数，但是我们可以创建一个自定义的 switch 语句。

```
switcher = {
    1: "January",
    2: "February",
    3: "March",
    4: "April",
    5: "May",
    6: "June",
    7: "July",
    8: "August",
    9: "September",
    10: "October",
    11: "November",
    12: "December"
}
month = int(input())
print(switcher.get(month))

> 3
march
```

举例说明 Python 中的 range 函数?

range：range 函数返回从起点到终点的一系列序列。

range(start, end, step)，第三个参数是用于定义范围内的步数。

```
# number
for i in range(5):
    print(i)
> 0,1,2,3,4

# (start, end)
for i in range(1, 5):
    print(i)
> 1,2,3,4

# (start, end, step)
for i in range(0, 5, 2):
    print(i)
>0,2,4
```

==和 is 的区别是?

==比较两个对象或值的相等性。

is 运算符用于检查两个对象是否属于同一内存对象。

```
lst1 = [1,2,3]
lst2 = [1,2,3]
```

```
lst1 == lst2
>True

lst1 is lst2
>False
```

如何更改列表的数据类型?

要将列表的数据类型进行更改，可以使用 tuple()或者 set()。

```
lst = [1,2,3,4,2]

# 更改为集合
set(lst)      ## {1,2,3,4}
# 更改为元组
tuple(lst)    ## (1,2,3,4,2)
```

Python 中注释代码的方法有哪些?

在 Python 中，我们可以通过下面两种方式进行注释。

1、 三引号'''，用于多行注释。

2、 单井号#，用于单行注释。

!=和 is not 运算符的区别?

!=如果两个变量或对象的值不相等，则返回 true。

is not 是用来检查两个对象是否属于同一内存对象。

```
lst1 = [1,2,3,4]
lst2 = [1,2,3,4]

lst1 != lst2
>False

lst1 is not lst2
>True
```

Python 是否有 main 函数?

是的，它有的。只要我们运行 Python 脚本，它就会自动执行。

什么是 lambda 函数?

Lambda 函数是不带名称的单行函数，可以具有 n 个参数，但只能有一个表达式。也称为匿名函数。

```
a = lambda x, y:  x + y
print(a(5, 6))

> 11
```

22、iterables 和 iterators 之间的区别?

iterable：可迭代是一个对象，可以对其进行迭代。在可迭代的情况下，整个数据一次存储在内存中。

iterators：迭代器是用来在对象上迭代的对象。它只在被调用时被初始化或存储在内存中。迭代器使用 next 从对象中取出元素。

```
# List is an iterable
lst = [1,2,3,4,5]
for i in lst:
    print(i)

# iterator
lst1 = iter(lst)
next(lst1)
>1
next(lst1)
>2
for i in lst1:
    print(i)
>3,4,5
```

Python 中的 Map Function 是什么?

map 函数在对可迭代对象的每一项应用特定函数后，会返回 map 对象。

解释 Python 中的 Filter?

过滤器函数，根据某些条件从可迭代对象中筛选值。

```
# iterable
lst = [1,2,3,4,5,6,7,8,9,10]

def even(num):
    if num%2==0:
        return num

# filter all even numbers
list(filter(even,lst))
-----------------------------------------
[2, 4, 6, 8, 10]
```

解释 Python 中 reduce 函数的用法?

reduce()函数接受一个函数和一个序列，并在计算后返回数值。

```
from functools import reduce

a = lambda x,y:x+y
print(reduce(a,[1,2,3,4]))

> 10
```

什么是 pickling 和 unpickling?

pickling 是将 Python 对象(甚至是 Python 代码)，转换为字符串的过程。

unpickling 是将字符串，转换为原来对象的逆过程。

解释*args 和**kwargs?

*args，是当我们不确定要传递给函数参数的数量时使用的。

```
def add（* num）：
    sum = 0
    for val in num:
        sum = val + sum
    print（sum）


add（4,5）
add（7,4,6）
add（10,34,23）
----------
9
17
57
```

**kwargs，是当我们想将字典作为参数传递给函数时使用的。

```
def intro(**data):
    print("\nData type of argument:",type(data))
    for key, value in data.items():
        print("{} is {}".format(key,value))


intro(name="alex",Age=22, Phone=1234567890)
```

```
intro(name="louis",Email="a@gmail.com",Country="Wakanda",
Age=25)
-----------------------------------------------------------
Data type of argument: <class 'dict'>
name is alex
Age is 22
Phone is 1234567890

Data type of argument: <class 'dict'>
name is louis
Email is a@gmail.com
Country is Wakanda
Age is 25
```

解释 re 模块的 split()、sub()、subn()方法?

split()：只要模式匹配，此方法就会拆分字符串。

sub()：此方法用于将字符串中的某些模式替换为其他字符串或序列。

subn()：和 sub()很相似，不同之处在于它返回一个元组，将总替换计数和新字符串作为输出。

```
import re
string = "There are two ball in the basket 101"



re.split("\W+",string)
---------------------------------------
['There', 'are', 'two', 'ball', 'in', 'the', 'basket', '101']
```

```
re.sub("[^A-Za-z]"," ",string)
---------------------------------------
'There are two ball in the basket'

re.subn("[^A-Za-z]"," ",string)
---------------------------------------
('There are two ball in the basket', 10)
```

Python 中的生成器是什么?

生成器(generator)的定义与普通函数类似，生成器使用 yield 关键字生成值。

如果一个函数包含 yield 关键字，那么该函数将自动成为一个生成器。

```
# A program to demonstrate the use of generator object with next() A
generator function
def Fun():
    yield 1
    yield 2
    yield 3

# x is a generator object
x = Fun()
print(next(x))
----------------------------
1
print(next(x))
----------------------------
2
```

如何使用索引来反转 Python 中的字符串?

```
string = 'hello'

string[::-1]
>'olleh'
```

类和对象有什么区别?

类(Class)被视为对象的蓝图。类中的第一行字符串称为 doc 字符串，包含该类的简短描述。

在 Python 中，使用 class 关键字可以创建了一个类。一个类包含变量和成员组合，称为类成员。

对象(Object)是真实存在的实体。在 Python 中为类创建一个对象，我们可以使用 obj = CLASS_NAME()

例如：obj = num()

使用类的对象，我们可以访问类的所有成员，并对其进行操作。

```
class Person:
    """ This is a Person Class"""
    # varable
    age = 10
    def greets(self):
        print('Hello')
```

```
# object
obj = Person()
print(obj.greet)
---------------------------------------
Hello
```

你对 Python 类中的 self 有什么了解?

self 表示类的实例。

通过使用 self 关键字，我们可以在 Python 中访问类的属性和方法。

注意，在类的函数当中，必须使用 self，因为类中没有用于声明变量的显式语法。

_init_在 Python 中有什么用?

“__init__”是 Python 类中的保留方法。

它被称为构造函数，每当执行代码时都会自动调用它，它主要用于初始化类的所有变量。

解释一下 Python 中的继承?

继承(inheritance)允许一个类获取另一个类的所有成员和属性。继承提供代码可重用性，可以更轻松地创建和维护应用程序。

被继承的类称为超类，而继承的类称为派生类/子类。

Python 中 OOPS 是什么?

面向对象编程，抽象(Abstraction)、封装(Encapsulation)、继承(Inheritance)、多态(Polymorphism)

什么是抽象?

抽象(Abstraction)是将一个对象的本质或必要特征向外界展示，并隐藏所有其他无关信息的过程。

什么是封装?

封装(Encapsulation)意味着将数据和成员函数包装在一起成为一个单元。

它还实现了数据隐藏的概念。

什么是多态?

多态(Polymorphism)的意思是「许多形式」。

子类可以定义自己的独特行为，并且仍然共享其父类/基类的相同功能或行为。

什么是 Python 中的猴子补丁?

猴子补丁(monkey patching)，是指在运行时动态修改类或模块。

```
from SomeOtherProduct.SomeModule import SomeClass

def speak(self):
    return "Hello!"

SomeClass.speak = speak
```

Python 支持多重继承吗?

Python 可以支持多重继承。多重继承意味着，一个类可以从多个父类派生。

Python 中使用的 zip 函数是什么?

zip 函数获取可迭代对象，将它们聚合到一个元组中，然后返回结果。

zip()函数的语法是 zip(*iterables)

```
numbers = [1, 2, 3]
string = ['one', 'two', 'three']
result = zip(numbers,string)

print(set(result))
-------------------------------------
{(3, 'three'), (2, 'two'), (1, 'one')}
```

解释 Python 中 map()函数?

map()函数将给定函数应用于可迭代对象(列表、元组等)，然后返回结果(map 对象)。

我们还可以在 map()函数中，同时传递多个可迭代对象。

```
numbers = (1, 2, 3, 4)
result = map(lambda x: x + x, numbers)

print(list(result))
```

Python 中的装饰器是什么?

装饰器(Decorator)是 Python 中一个有趣的功能。

它用于向现有代码添加功能。这也称为元编程，因为程序的一部分在编译时会尝试修改程序的另一部分。

```
def addition(func):
    def inner(a,b):
        print("numbers are",a,"and",b)
        return func(a,b)
    return inner

@addition
def add(a,b):
    print(a+b)

add(5,6)
---------------------------------
```

```
numbers are 5 and 6
sum: 11
```

编写程序，查找文本文件中最长的单词

```
def longest_word(filename):
    with open(filename, 'r') as infile:
            words = infile.read().split()
    max_len = len(max(words, key=len))
    return [word for word in words if len(word) == max_len]

print(longest_word('test.txt'))
-----------------------------------------------------
['comprehensions']
```

编写程序，检查序列是否为回文

```
a = input("Enter The sequence")
ispalindrome = a == a[::-1]

ispalindrome
>True
```

编写程序，打印斐波那契数列的前十项

```
fibo = [0,1]
[fibo.append(fibo[-2]+fibo[-1]) for i in range(8)]
```

```
fibo
> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

编写程序，计算文件中单词的出现频率

```
from collections import Counter

def word_count(fname):
        with open(fname) as f:
                return Counter(f.read().split())

print(word_count("test.txt"))
```

编写程序，输出给定序列中的所有质数

```
lower = int(input("Enter the lower range:"))
upper = int(input("Enter the upper range:"))
list(filter(lambda x:all(x % y != 0 for y in range(2, x)), range(lower, upper)))

------------------------------------------------
Enter the lower range:10
Enter the upper range:50
[11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

编写程序，检查数字是否为 Armstrong

将每个数字依次分离，并累加其立方(位数)。

最后，如果发现总和等于原始数，则称为阿姆斯特朗数(Armstrong)。

```
num = int(input("Enter the number:\n"))
order = len(str(num))

sum = 0
temp = num

while temp > 0:
    digit = temp % 10
    sum += digit ** order
    temp //= 10

if num == sum:
    print(num,"is an Armstrong number")
else:
    print(num,"is not an Armstrong number")
```

用一行 Python 代码，从给定列表中取出所有的偶数和奇数

```
a = [1,2,3,4,5,6,7,8,9,10]
odd, even = [el for el in a if el % 2==1], [el for el in a if el % 2==0]

print(odd,even)
> ([1, 3, 5, 7, 9], [2, 4, 6, 8, 10])
```

如何保证 Redis 中的数据都是热点数据

1、 Redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。Redis 提供 6 种数据淘汰策略：

2、 volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

3、 volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

4、 volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

5、 allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

6、 allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

7、 no-enviction（驱逐）：禁止驱逐数据

如何基于 Redis 实现发布和订阅

```
# 发布者
#coding:utf-8import time
import Redis

number_list = ['300033', '300032', '300031', '300030']
signal = ['1', '-1', '1', '-1']

rc = Redis.StrictRedis(host='***', port='6379', db=3, password='********')
for i in range(len(number_list)):
value_new = str(number_list[i]) + ' ' + str(signal[i])
```

```
rc.publish("liao", value_new)    #发布消息到 liao
```

```
# 订阅者
#coding:utf-8import time
import Redis

rc = Redis.StrictRedis(host='****', port='6379', db=3,
password='******')
ps = rc.pubsub()
ps.subscribe('liao')   #从 liao 订阅消息 for item in ps.listen():          #监听
状态：有消息发布了就拿过来
if item['type'] == 'message':
print item['channel']
print item['data']
```

什么是 codis

Codis 是一个分布式 Redis 解决方案，对于上层的应用来说，连接到 Codis Proxy 和连接原生的 Redis Server 没有明显的区别（有一些命令不支持），上层应用可以像使用单机的 Redis 一样使用，Codis 底层会处理请求的转发，不停机的数据迁移等工作，所有后边的一切事情，对于前面的客户端来说是透明的，可以简单的认为后边连接的是一个内存无限大的 Redis 服务，当然，前段时间 Redis 官方的 3.0 出了稳定版，3.0 支持集群功能，codis 的实现原理和 3.0 的集群功能差不多。

什么是 Twemproxy

Twemproxy 是一种代理分片机制，由 Twitter 开源。Twemproxy 作为代理，可接受来自多个程序的访问，按照路由规则，转发给后台的各个 Redis 服务器，再

原路返回。该方案很好的解决了单个 Redis 实例承载能力的问题。当然，Twemproxy 本身也是单点，需要用 Keepalived 做高可用方案。通过 Twemproxy 可以使用多台服务器来水平扩张 Redis 服务，可以有效的避免单点故障问题。虽然使用 Twemproxy 需要更多的硬件资源和在 Redis 性能有一定的损失（twitter 测试约 20%），但是能够提高整个系统的 HA 也是相当划算的。不熟悉 twemproxy 的同学，如果玩过 nginx 反向代理或者 MySQL proxy，那么你肯定也懂 twemproxy 了。其实 twemproxy 不光实现了 Redis 协议

Redis 如何实现事务

参考链接

https://blog.csdn.net/hxpjava1/article/details/79553073

Redis 中 watch 的作用

1、 watch 用于在进行事务操作的最后一步也就是在执行 exec 之前对某个 key 进行监视

2、 如果这个被监视的 key 被改动，那么事务就被取消，否则事务正常执行.

3、 一般在 MULTI 命令前就用 watch 命令对某个 key 进行监控.如果想让 key 取消被监控，可以用 unwatch 命令

DNS 域名解析过程

1、 浏览器检查缓存中有没有这个域名对应的解析后的 IP 地址，如果缓存中有，解析过程结束。缓存大小、时间都有限制，时间由 TTL 属性决定；

2、 如果浏览器缓存中么有，浏览器会查找操作系统缓存中有无这个域名 DNS 解析后的结果。操作系统也有一个域名解析的过程，windows 通过 C:\Windows\System32\drivers\etc\hosts，浏览器会优先使用这个解析结果（Win7 已将 hosts 设置为只读），linux 系统中/etc/named.conf。目前为止都是在本机完成，如果未完成，才会真正请求域名服务器解析域名。

3、 “网络配置”中都会有“DNX 服务器地址”，操作系统会把域名发送给这个 LDNS，本地区的域名服务器，通常都会提供一个本地互联网接入的 DNS 解析服务。就在你所在城市的某个角落，通过 ipconfig 可以看到。

4、 如果 LDNS 仍然没有命中，则向 RootServer 域名服务器请求解析。

5、 根域名服务器向本地域名服务器返回一个所查询域的主域名服务器（gTLD Server）。国际顶级域名服务器（.com、.cn、.org 等），全球 13 台。

6、 本地域名服务器（Local DNS Server）再向上一步返回的 gTLD 发送请求。

7、 gTLD 返回域名对应 NameServer 域名服务器地址，通常由你购买域名的服务商提供。

8、 NameServer 服务器查询域名与 IP 映射关系表，返回目标 IP 记录和 TTL 值给 DNS Server 域名服务器。

9、 Local DNS Server 根据 TTL 缓存该 IP 解析。

10、 缓存结果返回给用户，用户根据 TTL 缓存到本地操作系统中，域名解析过程结束。

了解过 Hbase，DB2，SQLServer，Access 吗

1、 Hbase：HBase 是一个分布式的、面向列的开源数据库

2、 DB2：一套关系型数据库管理系统，

3、 SQLServer：SQL Server 是由 Microsoft 开发和推广的关系数据库管理系统

4、 Sccess：Access 是由微软发布的关系数据库管理系统。

JavaScript(或者 jQuery)如何选择一个 id 为 main 的容器

1、 jquery：$('#id')

2、 JavaScript：document.getElementById("id")

css 如何隐藏一个元素

display：none

前后端分离的基本原理

前后端分离并非仅仅只是一种开发模式，而是一种架构模式（前后端分离架构）。前端项目与后端项目是两个项目，放在两个不同的服务器，需要独立部署，两个不同的工程，两个不同的代码库，不同的开发人员。前后端工程师需要约定交互接口，实现并行开发，开发结束后需要进行独立部署，前端通过 Ajax 来调用 HTTP 请求调用后端的 restful api。前端只需要关注页面的样式与动态数据的解析&渲染，而后端专注于具体业务逻辑。

如何保证 api 调用时数据的安全性

1、 通信使用 https

2、 请求签名，防止参数被篡改

3、 身份确认机制，每次请求都要验证是否合法

4、 app 中使用 ssl pinning 防止抓包操作

5、 对所有的请求和响应都进行加解密操作

如何实现响应式布局

1、 流体布局

其实就是度量单位的改变。在响应式设计的布局中，不在把像素(px)作为唯一的单位，而是采用%或者是混合%、px 为单位，设计出自己想要的布局方式。

2、 媒体查询

媒体查询可以在你根据特定的环境下查询到各种属性---------比如设备类型，分辨率、屏幕物理尺寸以及色彩等。通过使用媒体查询，可以获得设备的一些特性，以及响应式的布局方案。

3、 弹性图片

其实在做响应式布局时，大多用到的是弹性盒子进行布局，那么在设置图片的地方也应该具有一些变化以适应布局的变化。出了图片外，像图标啦！视频啦也应做一些调整用以适应布局的变化。

曾经使用过哪些前端框架

react，vue，bootstrap，elementUI，Echarts

什么是 ajax 请求？手写一个 ajax 请求

ajax（异步 JavaScript 和 XML）是指一种创建交付式网页应用的网页开发技术。可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

```
// 不使用第三方
var xhr = new XMLHttpRequest();
xhr.open("GET", url, false);
xhr.onreadtstatechange = function() {
    if (xhr.readystate == 4) {
        //响应内容解析完成，可以在客户端调用了
        if (xhr.status == 200) {
            //客户端的请求成功了
            alert(xhr.responseText);
        }
    }
}
xhr.send(null);
// 使用 ajax
$.ajax({
    type: "GET",
    url: "",
    dataType: "json",
    success: function(data) {},
    error: function(jqXHR) {}
```

```
});
```

什么是轮询和长轮询

轮询是在特定的的时间间隔（如每 1 秒），由浏览器对服务器发出 HTTP request，然后由服务器返回最新的数据给客户端的浏览器。这种传统的 HTTP request 的模式带来很明显的缺点 – 浏览器需要不断的向服务器发出请求，然而 HTTP request 的 header 是非常长的，里面包含的有用数据可能只是一个很小的值，这样会占用很多的带宽。

```
var xhr = new XMLHttpRequest();
setInterval(function() {
    xhr.open('GET', '/user');
    xhr.onreadystatechange = function() {};
    xhr.send();
}, 1000)
```

长轮询是 ajax 实现:在发送 ajax 后,服务器端会阻塞请求直到有数据传递或超时才返回。 客户端 JavaScript 响应处理函数会在处理完服务器返回的信息后，再次发出请求，重新建立连接。

```
function ajax() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/user');
    xhr.onreadystatechange = function() {
        ajax();
    };
    xhr.send();
}
```

vuex 的作用

1、 组件之间的数据通信

2、 使用单向数据流的方式进行数据的去中心化管理

vue 中的路由拦截器的作用

当某些页面需要访问权限时，可以使用路由拦截器对用户权限进行判断

axios 的作用

axios 是基于 promise 的用于浏览器和 nodejs 的 HTTP 客户端，本身有以下特征：

1、 从浏览器中创建 XMLHttpRequest;

2、 从 nodejs 发出 http 请求

3、 支持 promiseAPI

4、 拦截 请求和响应

5、 转换请求和响应数据

6、 取消请求

7、 自动转换 JSON 数据

8、 客户端支持防止 CSRF/XSRF 攻击

简述 jsonp 及其原理

JSONP 是 JSON with Padding 的略称。它是一个非官方的协议，它允许在服务器端集成 Script tags 返回至客户端，通过 javascript callback 的形式实现跨域访问（这仅仅是 JSONP 简单的实现形式）

原理：