Django 框架的优点

1. 功能完善稳定, 上手也很容易, 它自带有大量常用的组件,orm 和模板引擎非常好用, 适合快速开发企业级网站
2. Django 是 Python web 框架的先驱, 第三方库最丰富，如果不能直接用到 Django 中, 也一定能找到与之对应的移植
3. 用户量多, 对应的也会有广泛的实践案例和完善的在线文档, 在遇到什么问题时可以搜索在线文档寻找解决方案
4. Django 中的 app 设计是可插拔的, 不需要了可以直接删除, 对系统整体影响不大
5. 有一套完整的路由系统和强大的 admin 后台管理系统
6. orm 数据库访问组件, 使开发者无需过多学习 sql 语句就可以对数据库进行操作

Django 框架缺点

1. 过于笨重, 灵活和自由度不高, 用 Django 开发小型应用有种大材小用的感觉
2. 多度封装, 很多类和方法都进行了很完善的封装, 使用起来比较简单, 但是改动起来就比较困难

flask 框架优点

1. 自由、灵活，可扩展性强，第三方库的选择面广，开发时可以选择自己喜欢的一些组件
2. 入门简单，即便没有多少 web 开发经验，也能很快做出网站
3. 非常适用于小型网站, 适用于开发 web 服务的 API
4. Django 上能实现的功能, flask 也能找到与之类似的第三方库

flask 框架缺点

1. 代码架构需要自己设计, 依赖于开发者的能力和经验
2. 开发大型网站的自己设计路由规则, 能力不足导致代码混乱

Python 的优点

语法简单容易上手, 感觉就像读英语一样
编写 Python 程序时不需要关注内存什么的一些底层细节
可移植性好, Windows 和 Linux 上不需要什么改动就可以依旧运行
可扩展性强, 可在 Python 代码中插入用 C 或 C++编写的高并发代码
第三方库多, 可以处理更多的功能

Python 的缺点

Python 是解释性语言, 开发效率快但是运行效率慢
由于 GIL 多线程全局解释器锁的存在使得 Python 在处理多线程时无法利用多核优势

编程语言分类

编译型语言：程序被一次性全部翻译成机器语言，计算机直接以机器语言来运行此程序。
　　优点：运行效率高，可脱离语言环境独立运行。
　　缺点：开发效率低，可移植性差。
解释型语言：将程序逐条地翻译成机器语言让计算机去执行。
　　优点：修改代码简单灵活，开发效率高。
　　缺点：运行效率低。

Python 解释器的种类及特点

CPython: C 语言开发的, 使用最广的解释器
IPython: 基于 CPython 之上的一个交互式计时器 交互方式增强 功能和 CPython 一样
PyPy: 目标是执行效率 采用 JIT 技术 对 Python 代码进行动态编译，提高执行效率
JPython: 运行在 Java 上的解释器 直接把 Python 代码编译成 Java 字节码执行
IronPython: 运行在微软 .NET 平台上的解释器，把 Python 编译成. NET 的字节码

代码检测工具及其区别

pyflakes: 单独检查各个文件, 速度快, 不会检查代码风格
pylint: 查找不符合代码风格标准和有潜在问题的代码
flake8: flake8 检查规则灵活, 扩展性强

编码

ascii	---->	GBK	---->	Unicode	---->	UTF-8 (可伸缩的 unicode，节省空间)
(1 字节)		(2 字节)		(4 字节)		(英文 1 字节, 欧洲文字 2 字节；中文 3 字节)

Python2 和 Python3 的区别

1. py2 中使用 ASCII 码作为默认编码方式导致 string 有两种类型 str 和 unicode
   py3 中默认的编码方式是 unicode, 只支持 unicode 的 string, 不过多了一种 bytes 类型的字符串
2. Python2 中存在经典类和新式类的区别, 默认是继承经典类, 显式继承 object 类才是新式类
   Python3 统一采用新式类, 不写继承类的话默认继承 object 类.
3. print 语句被 Python3 废弃，统一使用 print 函数
4. exec 语句被 python3 废弃，统一使用 exec 函数
5. xrange 函数被 Python3 废弃，统一使用 range
6. long 整数类型被 Python3 废弃，统一使用 int
7. “ / ”：
   Python2: 若为两个整形数进行运算，结果为整形，但若两个数中有一个为浮点数，则结果为浮点数；
   Python3: 为真除法，运算结果不再根据参加运算的数的类型。
8. 不相等操作符"<>"被 Python3 废弃，统一使用"!="

PEP8 规范

一种编程规范, 遵循 pep8 使代码具有可读性
每一级缩进使用 4 个空格
所有行限制的最大字符数为 79, 没有结构化限制的大块文本(注释)每行的最大字符数限制在 72
顶层函数和类的定义，前后用两个空行隔开, 类里的方法定义用一个空行隔开
import 导入分开写(不推荐 import os,sys), 禁止使用通配符导入
导入总是在文件顶部, 导入顺序: 标准库导入 > 第三方库导入 > 本地导入
使用驼峰式命名
函数中对该函数功能和参数加注释
类中对该类加注释进行说明

什么是网络编程

网络编程就是计算机通过使用 IP(或域名)和端口连接到另一台计算机上对应的程序, 然后按照规定的协议进行两台计算机之间消息的发送和接收.
对于网络编程基本上都是基于请求/响应方式的，也就是一个设备发送请求数据给另外一个，然后接收另一个设备的反馈. 发起连接程序的一方被称作客户端(Client)，等待其他程序连接的程序被称作服务器(Server)。客户端程序可以在需要的时候启动，而服务器为了能够时刻响应连接，则需要一直启动, 连接一旦建立以后，就客户端和服务器端就可以进行数据传递了.

OSI7 层模型

物理层：主要是基于电器特性发送高低电压（电信号），高电压对应数字 1，低电压对应数字 0
数据链路层：定义了电信号的分组方式
网络层：引入一套新的地址用来区分不同的广播域/子网，这套地址即网络地址(IP 协议, ARP 协议)
传输层：建立端口到端口的通信(TCP, UDP 协议)
会话层：建立客户端与服务端连接
表示层：对来自应用层的命令和数据进行解释，按照一定格式传给会话层。如编码、数据格式转换、加密解密
应用层：规定应用程序的数据格式(HTTP/HTTPS 协议)
IP 协议对应于网络层
TCP 协议对应于传输层
HTTP 协议对应于应用层
TCP/IP 协议是传输层协议，主要解决数据如何在网络中传输，而 HTTP 协议是应用层协议, 主要解决如何包装数据。
WEB 使用 HTTP 协议作应用层协议，封装 HTTP 文本信息，然后使用 TCP/IP 做传输层协议将文本信息发到网络上。

IP

IP 地址由四个 0-255 之间的数字组成以"."作为分隔符, 它是分配给计算机网卡的, 每个网卡都由一个唯一的一个 IP 地址.

端口

0-65535, 用来确定电脑上具体某一个应用程序.

数据传输方式 -- socket 方式

通过 socket 套接字实现客户端和服务端之间的物理连接, 基于 tcp 协议或 udp 协议进行数据传输.
socket 处于网络协议的传输层.

什么是 socket
socket 是对 TCP/IP 协议的封装，它提供了一个针对 TCP 或者 UDP 编程的接口, 程序员可以用这个接口做网络开发.

进程
进程往小了说就是正在运行的程序的一个实例, 我们在运行一个 Python 文件或运行一个 app, 都叫做一个进程.
往大了说就是操作系统进行资源分配的基本单位, 进程既是基本的资源分配单元，也是基本的执行单元。
进程是一个能独立运行的实体。每一个进程都有它自己的地址空间，一般情况下，包括文本区域（text region）（python 的文件）、数据区域（data region）（python 文件中定义的一些变量数据）和堆栈（stack region）。文本区域存储处理器执行的代码；数据区域存储变量和进程执行期间使用的动态分配的内存；堆栈区域存储着活动过程调用的指令和本地变量。

进程特性
动态性：进程的实质是程序在系统中的一次执行过程，进程是动态产生，动态消亡的
并发性：任何进程都可以同其他进程一起并发执行
独立性：进程是一个能独立运行的基本单位，同时也是操作系统分配资源和调度的独立单位
异步性：即进程间都是相互独立的, 都按各自独立的、不可预知的速度向前推进
结构特征：进程由程序、数据和进程控制块三部分组成

进程间通信
队列(Queue 底层就是通过管道和锁实现的)
管道(pipe)
Manager + Lock 实现数据共享(m = Manager(), loc = Lock(), lst = m.list(range(100)) 注意,在声明共享数据 lst 时要用 Manager 声明一下)

并发
是伪并行, 即看起来是同时运行, 其实是 CPU 在多个程序之间不断地去切换, 使每个进程各自运行几十或几百毫秒，这样在 1 秒内，CPU 却可以运行多个进程, 这就给人产生了并行的错觉. 单个 CPU + 多道技术就可以实现并发效果.

多道技术
多道技术是指 CPU 一次读取多个程序放入内存，先运行第一个程序直到它出现了 IO 操作然后切换到第二个程序。因为 IO 操作慢，CPU 需要等待, 为了提高 CPU 利用率，才切换到第二个程序并运行第二个程序。

多道技术优点
提高 CPU 的利用率。在多道程序环境下，多个程序共享计算机资源当某个程序等待 I/O 操作时，CPU 可以执行其他程序，大大提高 CPU 的利用率。
提高设备的利用率。在多道程序环境下，多个程序共享系统的设备，大大提高系统设备的利用率。

并行
就是同时运行, 多个 CPU 运行多个程序.

什么是 fork
从别人的代码库中复制一份到你自己的代码库，与普通的复制不同，fork 包含了原有库中的所有提交记录，fork 后这个代码库是完全独立的，属于你自己，你可以在自己的库中做任何修改.

生产者消费者模式
生产者消费者模式是通过一个容器来解决生产者和消费者的强耦合问题。生产者和消费者彼此之间不直接通讯，而通过阻塞队列来进行通讯，所以生产者生产完数据之后不用等待消费者处理，直接扔给阻塞队列，消费者不找生产者要数据，而是直接从阻塞队列里取，阻塞队列就相当于一个缓冲区，平衡了生产者和消费者的处理能力，并且可以根据生产速度和消费速度来均衡一下多少个生产者可以为多少个消费者提供足够的服务，就可以开多进程等等，而这些进程都是到阻塞队列或者说是缓冲区中去获取或者添加数据。

为什么要用到生产者消费者模型
该模式通过平衡生产线程和消费线程的工作能力来提高程序的整体处理数据的速度。
在线程世界里，生产者就是生产数据的线程，消费者就是消费数据的线程。在多线程开发当中，如果生产者处理速度很快，而消费者处理速度很慢，那么生产者就必须等待消费者处理完，才能继续生产数据。同样的道理，如果消费者的处理能力大于生产者，那么消费者就必须等待生产者。为了解决这个问题于是引入了生产者和消费者模式。

线程
线程是供 CPU 调度的最小单位, 创建线程的开销要比创建进程的开销要小得多, 一个进程内最少有一个线程

线程特性
资源消耗少: 线程的资源消耗要比进程小得多, 所以在同一个进程中, 线程之间的切换非常迅速
共享进程资源: 同一进程中的各个线程，他们都具有相同的进程 ID, 都可以共享该进程所拥有的资源
可并发执行: 在一个进程中的多个线程之间，可以并发执行，甚至允许在一个进程中所有线程都能并发执行

多线程
一个进程中可以有多个线程, 多线程的存在是为了同步完成多项任务, 提高资源的使用效率, 但是在 Cpython 解释器中由于 GIL 全局解释器锁的存在, 使得同一进程下开启的多个线程, 在同一时刻只能有一个线程运行.

GIL 锁
全局解释器锁, 它是解释器级别的锁, 在 Python 这门语言中, 只有 Cpython 解释器才存在 GIL 锁, 由于 GIL 的存在使得同一进程下开启的多个线程, 在同一时刻只能有一个线程被 CPU 调度. 针对这一弊端可以使用多进程或者协程去解决.

有了 GIL 还需要数据级别的 Lock 吗
肯定需要. GIL 锁是解释器级的锁. 一个进程中的所有线程共享该进程内的所有资源. 如果不使用数据级别的锁, 如果一个线程用到资源中的某个数据时, 当遇到 I/O 阻塞便释放 GIL 锁并交出 CPU 的使用权, 另外一个线程抢到 GIL 锁, 拿到 CPU 的使用权后同样要对前面那个数据进行操作, 这样就导致数据混乱了, 所以有必要.

加互斥锁和 join()方法
在 start 之后立刻使用 join(), 肯定会将 100 个任务的执行变成串行, 毫无疑问, 最终 n 的结果也肯定是 0, 是安全的, 但问题是 start 后立即 join, 任务内的所有代码都是串行执行的, 而加锁只是加锁的部分即修改共享数据的部分是串行的. 单从保证数据安全方面, 二者都可以实现, 但很明显是加锁的效率更高.

死锁
死锁是指两个或两个以上的进程或线程在执行过程中，因争夺资源而造成的一种互相等待的现象，使程序无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程

递归锁
这个 RLock 内部维护着一个 Lock 和一个 counter 变量，counter 记录了 acquire 的次数，从而使得资源可以被多次 require。直到一个线程所有的 acquire 都被 release，其他的线程才能获得资源。

多进程
一个程序可以有多个进程, 而每个进程可以有多个线程, 这样就可以保证多个线程被多个 CPU 同时执行.

多进程优点
编程更加方便: 不需要考虑锁和同步资源的问题
容错性好: 比起多线程的一个好处是一个进程崩了不影响其他进程

多线程优点
创建速度快, 方便高效地进行数据共享, 多线程的上下文切换要比多进程的切换开销要小得多

多进程和多线程的使用场景
在计算密集型时使用多进程, 在 I/O 密集型时使用多线程
在需要频繁创建或销毁的时候(比如说 Web 服务器了, 来一个连接建立一个线程，断了就销毁线程，要是使用进程，创建和销毁的代价是很难承受的)

信号量
线程和进程一样.
Semaphore 管理一个内置的计数器，每当线程调用 acquire()时内置计数器-1；调用 release() 时内置计数器+1；计数器不能小于 0；当计数器为 0 时，acquire()将阻塞其他线程直到这几个线程调用 release()。

事件
线程和进程一样.
使用 threading 库中的 Event 对象, 对象包含一个可由线程设置的信号标志, 它允许线程等待某些事件的发生. 在 初始情况下, Event 对象中的信号标志被设置为假. 如果有线程等待一个 Event 对象, 而这个 Event 对象的标志为假, 那么这个线程将会被一直阻塞直至该标志为真. 一个线程如果将一个 Event 对象的信号标志设置为真, 它将唤醒所有等待这个 Event 对象的线程. 如果一个线程等待一个已经被设置为真的 Event 对象,那么它将忽略这个事件, 继续执行.

协程
协程就是微线程, 是单线程下的并发, 它是一种用户态的轻量级线程，即协程是由用户程序自己控制调度的。如果一个任务遇到 I/O 阻塞就会从应用程序级别切换到另外一个任务去执行，以此来提升效率.协程一般应用于 IO 密集型的任务. 在处理计算密集型任务时反倒是很慢.

协程优点缺点
优点

协程的切换开销更小，属于程序级别的切换，操作系统完全感知不到，因而更加轻量级

单线程内就可以实现并发的效果，最大限度地利用 CPU

修改共享数据不需要加锁, 因为他本来就是单线程运行

缺点

协程指的是单个线程，因而一旦协程出现阻塞，将会阻塞整个线程

如何实现协程
greenlet 模块实现协程: 当切到一个任务执行时如果遇到 I/O, 那就原地阻塞，仍然是没有解决遇到 I/O 自动切换来提升效率的问题.
使用 gevent 第三方库(本身不用作协程, 而是配合 greenlet 模块在遇到 IO 阻塞时自动切换), 遇到 I/O 能自动切换任务.

如何实现高并发
用进程 + 线程 + 协程的方式来实现并发，可以达到最好的并发效果. 如果是 4 核的 CPU，一般起 5 个进程，每个进程中 20 个线程（5 倍 CPU 数量），每个线程可以起 500 个协程，大规模爬取页面的时候，等待网络延迟的时间的时候，我们就可以用协程去实现并发。 并发数量 = 5 * 20 * 500 = 50000 个并发，这是一般一个 4CPU 的机器最大的并发数.

TCP 和 UDP 区别
TCP: 提供的是面向连接、可靠的服务. 当客户和服务器彼此交换数据前, 必须先在双方之间建立一个 TCP 连接, 之后才能传输数据. 通过 TCP 连接传送的数据, 无差错, 不丢失, 不重复, 且按序到达, 保证数据能从一端传到另一端.
UDP: UDP 是无连接的，即发送数据之前不需要建立连接, 所以 udp 的传输速度要比 TCP 快. UDP 不提供可靠性, 并不能保证它们能到达目的地. 而且 UDP 具有较好的实时性，工作效率比 TCP 高，适用于对高速传输和实时性有较高的通信或广播通信。
依靠 TCP 协议传输需要建立 3 次握手 4 次挥手, 而 UDP 不需要这一步.

TCP 的三次握手
a, 客户端向服务端发起连接请求(问是否可以连接)
b, 服务端对接受到连接请求进行确认
c, 客户端收到回复并确认, 连接正式建立

四次挥手
a, 客户端发送报文,告知服务端我要断开连接
b, 服务端返回报文, 告知客户端收到请求, 准备断开
c, 服务端再次发送报文给客户端, 告知准备完毕可以断开
d, 客户端发送报文进行断开
注意: 服务端是要比客户端提前断开.

如何用 Git 进行多人协同开发项目
开发人员接到新的开发任务后，如果没有拉取过远程仓库代码，就拉取一次。否则请先更新到代码最新状态。然后，由第一个人创建一个新的开发分支 dev 并推送到远程。再然后，所有开发同一功能的开发人员跟踪对应的远程开发分支，进行开发。最后功能开发完成后, 合并分支并消除差异, 由最后一人提交代码审核**(注意这次合并是在本地 master 分支下，合并本地 remote_T34 分支的内容。然后你告诉我本地 master 和本地 remote_T34 的代码都是最新的，那我合并这两个然后推送到远程是什么效果？诶~，就是合并远程 master 和远程 remote_T34！但是！此时千万不能推送到远程，因为还没有高层的点头(代码 Review)！其实这个小节主要作用就是处理代码合并时的冲突，如果有冲突，全程大概也只有这个地方高发。合并完成后，就该提交代码 Review 了)**，代码审核通过后再将代码提交到 master。功能开发完成。

如何做 code review
项目组长做代码审核. 采用 PR 模式(对主干分支的操作权限做了限制，只有特定的人才能操作，develop 分支是项目开发 Leader 和架构师，master 分支是 QA)

简述 HTTP 协议
HTTP 协议即超文本传输协议, 由服务器传输超文本到本地浏览器的一种传送协议。HTTP 是一个基于 TCP/IP 通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）

HTTP 和 HTTPS 的区别

HTTP 的 URL 以 http:// 开头，而 HTTPS 的 URL 以 https:// 开头
HTTP 是不安全的(它无法对传输的数据进行加密)，而 HTTPS 是安全的(它是通过 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议, 对传输的数据进行加密)
HTTP 标准端口是 80 ，而 HTTPS 的标准端口是 443
在 OSI 网络模型中，HTTP 工作于应用层，而 HTTPS 的安全传输机制工作在传输层
HTTP 无需证书，而 HTTPS 需要 CA 机构 wosign 的颁发的 SSL 证书

HTTP 的特性

HTTP 是无连接：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。
HTTP 是媒体独立的：这意味着，只要客户端和服务器知道如何处理的数据内容，任何类型的数据都可以通过 HTTP 发送。客户端以及服务器指定使用适合的 MIME-type 内容类型。(mimt-type 说穿了其实指的就是文件后缀名。你向 web 服务器请求一个文件，服务器会根据你的后缀名去匹配对应的值然后在响应头 response 中设置为 content-type 类型。而 content-type 是正文媒体类型，浏览器根据 content-type 的不同来分别处理你返回的东西。)
HTTP 是无状态：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。

HTTP 工作原理

HTTP 协议工作于客户端-服务端架构上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即 WEB 服务器发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。

无状态原因

浏览器与服务器是使用 socket 套接字进行通信的，服务器将请求结果返回给浏览器之后，会关闭当前的 socket 连接，而且服务器也会在处理页面完毕之后销毁页面对象。

如何解决无状态协议

1. 在客户端存储信息使用 Cookie,本地存储, Cookie 就相当于一个通行证，第一次访问的时候给客户端发送一个 Cookie，当客户端再次来的时候，拿着 Cookie(通行证)，那么服务器就知道这个是老用户。
2. 在服务器端存储信息使用 Session 或 redis.

一次完整的 HTTP 请求所经历的 7 个步骤

HTTP 通信机制是在一次完整的 HTTP 通信过程中，Web 浏览器与 Web 服务器之间将完成下列 7 个步骤: 建立 TCP 连接-->发送请求行->发送请求头-->（到达服务器）发送状态行-->发送响应头-->发送响应数据-->断 TCP 连接.
1 基于 TCP 进行 Web 浏览器和 Web 服务器建立连接;
2 Web 浏览器向 Web 服务器发送请求行;
3 Web 浏览器发送请求头, 并发送了一空白行来通知服务器，它已经结束了该头信息的发送。
4 Web 服务器应答, HTTP/1.1 200 OK;
5 Web 服务器发送响应头和一个空白行来表示头信息的发送到此为结束;
6 Web 服务器向浏览器发送响应体数据, 以 Content-Type 应答头信息所描述的格式发送用户所请求的实际数据;
7 Web 服务器关闭 TCP 连接
一般情况下，一旦 Web 服务器向浏览器发送了请求数据，它就要关闭 TCP 连接，然后如果浏览器或者服务器在其头信息加入了这行代码：Connection:keep-alive, TCP 连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

路由器和交换机区别

交换机: 交换机是发生在网络的第二层数据链路层, 在局域网内构成网络, 进行互相通信的
路由器: 路由器发生在网络的第三层网络层, 它可以给网络分配 IP 地址, 可以把 1 个 IP 分给多个主机使用.

ARP 协议

在网络层, 以广播的形式发送数据包, 通过 IP 地址获取 Mac 地址发送数据包.

DNS 解析

域名解析. 比如 www.baidu.com 是一个域名, 而 ip 是 4 个 0-255 之间的数字, ip 很难被人记住, 所以就有了一个域名和 ip 的对应关系.

DNS 解析顺序

1.优先查找本地 dns 缓存
2.查找本地/etc/hosts 文件，是否有强制解析
3.如果没有去/etc/resolv.conf 指定的 dns 服务器中查找记录(需联网)
4.在 dns 服务器中找到解析记录后，在本地 dns 中添加缓存
5.完成一次 dns 解析

猴子补丁

在运行时替换某个方法或属性等. 即在不修改第三方代码的情况下增加原来不支持的功能.
只是在运行时为内存中的对象修改属性而对磁盘中的源文件没有影响.

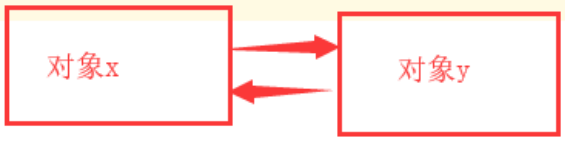
Python 中的垃圾回收机制

```
class Person:
    pass

a = Person()

a = 20
```

第一层：引用计数(针对被引用的对象)
a刚开始指向Person(),Person()的引用为1
a又指向20，此时Person()的引用为0,Person()被当做垃圾进行回收



当两个对象互相引用时候,而没有被外界引用 此时引用计数机制就没用了

```
class Person:
    pass

a = Person()

a = 20
```

第二层 标记清除(针对互相引用问题)
该机制是从引用者(a)出发,对引用者引用的对象进行标记.
Python从上往下去遍历所有的引用者(变量)，当遍历到a时候,a指向
Person(),Person()就会被标记，然后继续往下遍历，此时下面的a指向20，20被标记，Person()的标记就被清除,被当做一个垃圾进行回收.

当两个对象间互相引用而没有被外界调用时候，这俩对象不被标记，被当做垃圾回收

```
class Person:
    pass

a = Person()

a = 20
```

第三层 分代回收
Python每隔一段时间就要从上到下将所有变量引用的对象进行遍历，非常消耗资源,此时引入分代回收机制：循环10次，将10次都没有被回收的对象升级为第二代，然后将第二代遍历10次，将没有被回收的对象升级为第三代，然后去遍历第三代，这样，第一代遍历100 次，第二代遍历10 次，第三代遍历1 次，极大地提高效率