CDN的全称是Content Delivery Network

- 内容分发网络
- CDN是构建在现有网络基础之上的智能虚拟网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。
- 关键技术主要有内容存储和分发技术

DNS (Domain Name System)

- 将域名和IP地址相互映射的一个分布式数据库，能够使人更方便地访问互联网

---

web前端性能优化：

- 1.图片懒加载, 使用sprite精灵图
- 2.将代码打包压缩
- 3.减少dom操作
- 4.减少首屏的加载时间
- 5.减少http请求，合理设置 HTTP缓存
- 6.使用浏览器缓存
- 7.CSS放在页面最上部，javascript放在页面最下面
- 8.减少浏览器回流(Reflow)和重绘(Repaint)

HTTP 请求方法

- get: 参数通过URL传递, 从服务器获取资源
- head:
    - HEAD方法：与GET方法的行为很类似，但服务器在响应中只返回首部，不返回实体的主体部分。这就允许客户端在未获取实际资源的情
    - 况下，对资源的首部进行检查，使用HEAD，我们可以更高效的完成以下工作：
    - 在不获取资源的情况下，了解资源的一些信息，比如资源类型；
    - 通过查看响应中的状态码，可以确定资源是否存在；
    - 通过查看首部，测试资源是否被修改；
- trace: 会在目的服务器端发起一个“回环”诊断，我们都知道，客户端在发起一个请求时，这个请求可能要穿过防火墙、代理、网关、或者其它的一些应用程序。这中间的每个节点都可能会修改原始的HTTP请求，TRACE方法允许客户端在最终将请求发送服务器时，它变成了什么样子。由于有一个“回环”诊断，在请求最终到达服务器时，服务器会弹回一条TRACE响应，并在响应主体中携带它收到的原始请求报文的最终模样。这样客户端就可以查看HTTP请求报文在发送的途中，是否被修改过了。
- options: 用于获取当前URL所支持的方法。
- post: 向服务器提交表单信息

- put: 向服务器写入文档, PUT 方法的语义就是让服务器用请求的主体部分来创建一个由所请求的URL命名的新文档, 如果那个URL已经存在的话, 就用这个主体来替换它
- patch: 是对put方法的补充, 用来对已知资源进行局部更新
- delete: 删除请求 url 指定的资源

HTTPS和HTTP的区别主要如下：

- 1. https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。
- 2. http是超文本传输协议，信息是明文传输，主要是来规范浏览器和服务器端的行为的。https则是具有安全性的ssl加密传输协议。
- 3. http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
- 4. http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

---

https: http + SSL

- SSL: 是为网络通信提供安全及数据完整性的一种安全协议。

SSH:安全的网络传输协议

- SSH（ Secure Shell） 是目前较可靠，**专为远程登录会话**和其他网络服务**提供安全性的协议**。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。SSH 建立在可靠的传输协议 TCP 之上。

HTTPS 通信过程

- 1. 客户端对服务器发起https请求 连接到服务器443端口.
- 2. 服务器采用的https有一套 ca 数字证书 包含了 公钥 和 私钥
- 3. 服务器将公钥证书传送给客户端，证书中包含了很多信息，比如证书的颁发机构，过期时间，网址，公钥等.
- 4. 客户端解析证书，验证 证书公钥是否有效，证书是否过期等，如有异常则会弹出警告信息。 证书没问题之后， 会随机生成随机值， 然后使用 证书中的公钥对这个随机值 进行非对称加密.
- 5. 将非对称加密后的随机值传送到服务器
- 6. 服务器使用私钥进行非对称解密，得到客户端产生的随机值， 然后把将要返回给客户端的内容进行对称加密， 然后发送给客户端
- 7. 客户端拿到加密后的内容后用之前生产的随机值进行对称解密， 最后获取到内容。

HTTP 完整请求流程

- 1. 客户端请求域名， 然后进行域名解析 转换成ip地址
- 2. 客户端拿到解析出来的 ip 地址后， 会进行 tcp 三次握手与服务器建立连接
- 3. 客户端向服务器发送请求
- 4. 服务器返回源代码给客户端
- 5. 客户端解析请求回来的源代码，并请求代码中的其他资源 如 静态资源图片, css, js 文件等。
- 6. 客户端进行页面渲染

- 7. web 服务器断开连接(四次握手), 有些特殊请求不断开

HTTP/1.0 和 HTTP/1.1 区别

1. 连接方式 : HTTP 1.0 为短连接，HTTP 1.1 支持长连接。
2. 状态响应码 : HTTP/1.1中新加入了大量的状态码，光是错误响应状态码就新增了24种。
3. 缓存处理 : 在 HTTP1.0 中主要使用 header 里的 If-Modified-Since,Expires 来做为缓存判断的标准，HTTP1.1 则引入了更多的缓存控制策略例如 Etag，If-Unmodified-Since, If-Match, If-None-Match 等更多可供选择的缓存头来控制缓存策略。
4. 带宽优化及网络连接的使用 :HTTP1.0 中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1 则在请求头引入了 range 头域，它允许只请求资源的某个部分，即返回码是 206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。
5. Host头处理 : HTTP/1.1在请求头中加入了Host字段。

HTTP 状态码

- 1**， 信息状态码
- 2**， 成功状态码
- 3**， 重定向状态码
- 4**， 客户端错误状态码
- 5**， 服务器错误状态码

TCP 三次握手

1. 客户端–发送带有 SYN（同步序列编号） 标志的数据包–一次握手–服务端
2. 服务端–发送带有 SYN/ACK（确认字符） 标志的数据包–二次握手–客户端
3. 客户端–发送带有带有 ACK 标志的数据包–三次握手–服务端

为什么要三次握手 ？

- 三次握手的目的是建立可靠的通信信道，说到通讯，简单来说就是数据的发送与接收，而三次握手最主要的目的就是双方确认自己与对方的发送与接收是正常的。

Cookie 和 Session

> **Cookie** 实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端会把Cookie保存起来。

当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

> **Session** 是服务器端使用的一种记录客户端状态的机制，使用上比Cookie简单一些，相应的也增加了服务器的存储压力。 如果说Cookie机制是通过检查客户身上的“通行证”来确定客户身份的话，那么Session机制就是通过检查服务器上的“客户明细表”来确认客户身份。Session相当于程序在服务器上建立的一份客户档案，客户来访的时候只需要查询客户档案表就可以了。

Session和Cookie的区别？

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

URI,URL,URN

- URI: 统一资源标识符, 统一资源标识符，根据某一种规则将一个资源独一无二地标识出来。
- URL: 统一资源定位符, 统一资源定位符，它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何定位这个资源。
- URN: 统一资源名, 统一资源名称。

URL 和 URN 是 URI 的子集

- 统一资源标志符URI就是在某一规则下能把一个资源独一无二地标识出来。拿人做例子，假设这个世界上所有人的名字都不能重复，那么名字就是URI的一个实例，通过名字这个字符串就可以标识出唯一的一个人。现实当中名字当然是会重复的，所以身份证号才是URI，通过身份证号能让我们能且仅能确定一个人。那统一资源定位符URL是什么呢。也拿人做例子然后跟HTTP的URL做类比，就可以有：动物住址协议://地球/中国/浙江省/杭州市/西湖区/某大学/14号宿舍楼/525号寝/张三.人可以看到，这个字符串同样标识出了唯一的一个人，起到了URI的作用，所以URL是URI的子集。URL是以描述人的位置来唯一确定一个人的。在上文我们用身份证号也可以唯一确定一个人。对于这个在杭州的张三，我们也可以用：身份证号：123456789

什么是线程和进程?

- **进程** 是程序的一次执行过程，是系统运行程序的基本单位，因此进程是动态的。系统运行一个程序即是一个进程从创建，运行到消亡的过程。
- **线程** 与进程相似，但线程是一个比进程更小的执行单位。一个进程在其执行的过程中可以产生多个线程。与进程不同的是同类的多个线程共享进程的堆和方法区资源，但每个线程有自己的程序计数器、虚拟机栈和本地方法栈，所以系统在产生一个线程，或是在各个线程之间作切换工作时，负担要比进程小得多，也正因为如此，线程也被称为轻量级进程。
- **区别** 线程是进程划分成的更小的运行单位。线程和进程最大的不同在于基本上各进程是独立的，而各线程则不一定，因为同一进程中的线程极有可能会相互影响。线程执行开销小，但不利于资源的管理和保护；而进程正相反。

深拷贝，浅拷贝区别

- **浅拷贝**：浅拷贝会在堆上创建一个新的对象，不过，如果原对象内部的属性是引用类型的话，浅拷贝会直接复制内部对象的引用地址，也就是说拷贝对象和原对象共用同一个内部对象。
- **深拷贝** ：深拷贝会完全复制整个对象，包括这个对象所包含的内部对象。

什么是序列化?什么是反序列化?

- **序列化：** 将数据结构或对象转换成二进制字节流的过程。序列化的主要目的是通过网络传输对象或者说是将对象存储到文件系统、数据库、内存中。
- **反序列化**：将在序列化过程中所生成的二进制字节流转换成数据结构或者对象的过程

泛型

- **泛型：** 所谓泛型，就是允许在定义类、接口时通过一个标识表示类中某个属性的类型或者是某个方法的返回值及参数类型。这个类型参数将在使用时（例如，继承或实现这个接口，用这个类型声明变量、创建对象时确定（即传入实际的类型参数，也称为类型实参）。
- **好处**

1. 使用泛型编写的程序代码
2. 多种数据类型执行相同的代码使用泛型可以复用代码。

什么是操作系统？

1. 操作系统（Operating System，简称 OS）是管理计算机硬件与软件资源的程序，是计算机的基石。
2. 操作系统本质上是一个运行在计算机上的软件程序 ，用于管理计算机硬件和软件资源。
3. 操作系统存在屏蔽了硬件层的复杂性。 操作系统就像是硬件使用的负责人，统筹着各种相关事项。
4. 操作系统的内核（Kernel）是操作系统的核心部分，它负责系统的内存管理，硬件设备的管理，文件系统的管理以及应用程序的管理。 内核是连接应用程序和硬件的桥梁，决定着系统的性能和稳定性。

进程有哪几种状态?

- **创建状态** ：进程正在被创建，尚未到就绪状态。
- **就绪状态** ：进程已处于准备运行状态，即进程获得了除了处理器之外的一切所需资源，一旦得到处理器资源(处理器分配的时间片)即可运行。
- **运行状态**：进程正在处理器上上运行(单核 CPU 下任意时刻只有一个进程处于运行状态)。
- **阻塞状态**：又称为等待状态，进程正在等待某一事件而暂停运行如等待某资源为可用或等待 IO 操作完成。即使处理器空闲，该进程也不能运行。
- **结束状态**：进程正在从系统中消失。可能是进程正常结束或其他原因中断退出运行。

什么是死锁

- **死锁** 描述的是这样一种情况：多个进程/线程同时被阻塞，它们中的一个或者全部都在等待某个资源被释放。由于进程/线程被无限期地阻塞，因此程序不可能正常终止。

> 产生死锁的四个必要条件是什么?

1. 互斥：资源必须处于非共享模式，即一次只有一个进程可以使用。如果另一进程申请该资源，那么必须等待直到该资源被释放为止。
2. 占有并等待：一个进程至少应该占有一个资源，并等待另一资源，而该资源被其他进程所占有。
3. 非抢占：资源不能被抢占。只能在持有资源的进程完成任务后，该资源才会被释放。
4. 循环等待：有一组等待进程 {P0, P1,..., Pn}， P0 等待的资源被 P1 占有，P1 等待的资源被 P2 占有，....., Pn-1 等待的资源被 Pn 占有，Pn 等待的资源被 P0 占有。

解决死锁的方法

解决死锁的方法可以从多个角度去分析，一般的情况下，有预防，避免，检测和解除四种。

1. **预防** 是采用某种策略，限制并发进程对资源的请求，从而使得死锁的必要条件在系统执行的任何时间上都不满足。

2. **避免** 则是系统在分配资源时，根据资源的使用情况提前做出预测，从而避免死锁的发生

3. **检测** 是指系统设有专门的机构，当死锁发生时，该机构能够检测死锁的发生，并精确地确定与死锁有关的进程和资源。

4. **解除** 是与检测相配套的一种措施，用于将进程从死锁状态下解脱出来。

什么是虚拟内存(Virtual Memory)?

- 这个在我们平时使用电脑特别是 Windows 系统的时候太常见了。很多时候我们使用了很多占内存的软件，这些软件占用的内存可能已经远远超出了我们电脑本身具有的物理内存。为什么可以这样呢？ 正是因为 虚拟内存 的存在，通过 虚拟内存 可以让程序可以拥有超过系统物理内存大小的可用内存空间。另外，虚拟内存为每个进程提供了一个一致的、私有的地址空间，它让每个进程产生了一种自己在独享主存的错觉。这样会更加有效地管理内存并减少出错。

虚拟内存是计算机系统内存管理的一种技术，我们可以手动设置自己电脑的虚拟内存。不要单纯认为虚拟内存只是“使用硬盘空间来扩展内存“的技术。虚拟内存的重要意义是它定义了一个连续的虚拟地址空间，并且 把内存扩展到硬盘空间。

TCP/IP 四层模型

1. **应用层**：应用层位于传输层之上，主要提供两个终端设备上的应用程序之间信息交换的服务，它定义了信息交换的格式，消息会交给下一层传输层来传输。 我们把应用层交互的数据单元称为报文。
2. **传输层**：传输层的主要任务就是负责向两台终端设备进程之间的通信提供通用的数据传输服务。 应用进程利用该服务传送应用层报文。“通用的”是指并不针对某一个特定的网络应用，而是多种应用可以使用同一个运输层服务。
3. **网络层**：网络层负责为分组交换网上的不同主机提供通信服务。 在发送数据时，网络层把运输层产生的报文段或用户数据报封装成分组和包进行传送。在 TCP/IP 体系结构中，由于网络层使用 IP 协议，因此分组也叫 IP 数据报，简称数据报。
4. **网络接口层**：我们可以把网络接口层看作是数据链路层和物理层的合体。

数据库中的事务是什么？

- 事务就是一系列的操作,这些操作完成一项任务。只要这些操作里有一个操作没有成功,事务就操作失败,发生回滚事件。即撤消前面的操作,这样可以保证数据的一致性。而且可以把操作暂时放在缓存里,等所有操作都成功有提交数据库,这样保证费时的操作都是有效操作。

什么是元组, 码, 候选码, 主码, 外码, 主属性, 非主属性？

- **元组** ： 元组（tuple）是关系数据库中的基本概念，关系是一张表，表中的每行（即数据库中的每条记录）就是一个元组，每列就是一个属性。 在二维表里，元组也称为行。
- **码** ：码就是能唯一标识实体的属性，对应表中的列。
- **候选码** ： 若关系中的某一属性或属性组的值能唯一的标识一个元组，而其任何、子集都不能再标识，则称该属性组为候选码。例如：在学生实体中，“学号”是能唯一的区分学生实体的，同时又假设“姓名”、“班级”的属性组合足以区分学生实体，那么{学号}和{姓名，班级}都是候选码。
- **主码** : 主码也叫主键。主码是从候选码中选出来的。 一个实体集中只能有一个主码，但可以有多个候选码。
- **外码** : 外码也叫外键。如果一个关系中的一个属性是另外一个关系中的主码则这个属性为外码。
- **主属性** ： 候选码中出现过的属性称为主属性。比如关系 工人（工号，身份证号，姓名，性别，部门）. 显然工号和身份证号都能够唯一标示这个关系，所以都是候选码。工号、身份证号这两个属性就是主属性。如果主码是一个属性组，那么属性组中的属性都是主属性。
- **非主属性**： 不包含在任何一个候选码中的属性称为非主属性。比如在关系——学生（学号，姓名，年龄，性别，班级）中，主码是“学号”，那么其他的“姓名”、“年龄”、“性别”、“班级”就都可以称为非主属性。

数据库设计通常分为哪几步?

1. 需求分析 : 分析用户的需求，包括数据、功能和性能需求。
2. 概念结构设计 : 主要采用 E-R 模型进行设计，包括画 E-R 图。
3. 逻辑结构设计 : 通过将 E-R 图转换成表，实现从 E-R 模型到关系模型的转换。
4. 物理结构设计 : 主要是为所设计的数据库选择合适的存储结构和存取路径。
5. 数据库实施 : 包括编程、测试和试运行
6. 数据库的运行和维护 : 系统的运行与数据库的日常维护。

数据库四大特性

1. **原子性** ： 事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用；
2. **一致性**： 执行事务前后，数据保持一致，例如转账业务中，无论事务是否成功，转账者和收款人的总额应该是不变的；
3. **隔离性**： 并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据库是独立的；
4. **持久性**： 一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库发生故障也不应该对其有任何影响。

并发事务带来了哪些问题?

1. **脏读**: 当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事

务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。

2. **丢失修改**: 指在一个事务读取一个数据时，另外一个事务也访问了该数据，那么在第一个事务中修改了这个数据后，第二个事务也修改了这个数据。这样第一个事务内的修改结果就被丢失，因此称为丢失修改。 例如：事务 1 读取某表中的数据 A=20，事务 2 也读取 A=20，事务 1 修改 A=A-1，事务 2 也修改 A=A-1，最终结果 A=19，事务 1 的修改被丢失。

3. **不可重复读**: 指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。

4. **幻读**: 幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，第一个事务（T1）就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。

MySQL 日志

> 错误日志、归档日志(binlog)、事务日志(redo log)、回滚日志、查询日志、慢查询日志

- **事务日志** redo log（重做日志）它是物理日志，记录内容是“在某个数据页上做了什么修改”，属于 InnoDB 存储引擎。它让MySQL拥有了崩溃恢复能力。比如 MySQL 实例挂了或宕机了，重启时，InnoDB 存储引擎会使用redo log恢复数据，保证数据的持久性与完整性。

- **归档日志** binlog 是逻辑日志，记录内容是语句的原始逻辑，类似于“给 ID=2 这一行的 c 字段加 1”，属于 MySQL Server 层。不管用什么存储引擎，只要发生了表数据更新，都会产生 binlog 日志。

- **回滚日志** 我们知道如果想要保证事务的原子性，就需要在异常发生时，对已经执行的操作进行回滚，在 MySQL 中，恢复机制是通过 回滚日志（undo log） 实现的，所有事务进行的修改都会先记录到这个回滚日志中，然后再执行相关的操作。如果执行过程中遇到异常的话，我们直接利用 回滚日志 中的信息将数据回滚到修改之前的样子即可！并且，回滚日志会先于数据持久化到磁盘上。这样就保证了即使遇到数据库突然宕机等情况，当用户再次启动数据库的时候，数据库还能够通过查询回滚日志来回滚将之前未完成的事务。

创建数据表

```
DROP TABLE IF EXISTS test1;
CREATE TABLE `test1` (
    `id` int(11) NOT NULL,
    `num1` int(11) NOT NULL DEFAULT '0',
    `num2` varchar(11) NOT NULL DEFAULT '',
    `type1` int(4) NOT NULL DEFAULT '0',
    `type2` int(4) NOT NULL DEFAULT '0',
    `str1` varchar(100) NOT NULL DEFAULT '',
    `str2` varchar(100) DEFAULT NULL,
    PRIMARY KEY (`id`),
    KEY `num1` (`num1`),
    KEY `num2` (`num2`),
    KEY `type1` (`type1`),
```

```
    KEY `str1` (`str1`),
    KEY `str2` (`str2`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Redis

- 简单来说 Redis 就是一个使用 C 语言开发的数据库，不过与传统数据库不同的是 Redis 的数据是存在内存中的 ，也就是它是内存数据库，所以读写速度非常快，因此 Redis 被广泛应用于缓存方向。 另外，Redis 除了做缓存之外，也经常用来做分布式锁，甚至是消息队列。 Redis 提供了多种数据类型来支持不同的业务场景。Redis 还支持事务 、持久化、Lua 脚本、多种集群方案。

Redis 特点

- 高性能：如果说，用户访问的数据属于高频数据并且不会经常改变的话，那么我们就可以很放心地将该用户访问的数据存在缓存中。从缓存读取数据，操作缓存就是操作内存，所以速度就非常快了。
- 高并发：由此可见，直接操作缓存能够承受的数据库请求数量是远远大于直接访问数据库的，所以我们可以考虑把数据库中的部分数据转移到缓存中去，这样用户的一部分请求会直接到缓存这里而不用经过数据库。进而，我们也就提高了系统整体的并发。

1. Redis 支持更丰富的数据类型（支持更复杂的应用场景）。Redis 不仅仅支持简单的 k/v 类型的数据，同时还提供 list，set，zset，hash 等数据结构的存储。
2. Redis 支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用
3. Redis 有灾难恢复机制。 因为可以把缓存中的数据持久化到磁盘上。
4. Redis 在服务器内存使用完之后，可以将不用的数据放到磁盘上。
5. Redis 使用单线程的多路 IO 复用模型。 （Redis 6.0 引入了多线程 IO）
6. Redis 支持发布订阅模型、Lua 脚本、事务等功能，并且，Redis 支持更多的编程语言。
7. Redis 同时使用了惰性删除与定期删除。

Redis 除了做缓存，还能做什么？

1. 分布式锁
2. 限流
3. 消息队列
4. 其他复杂业务场景

Nginx

> Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器

- **正向代理**：是指是一个位于客户端和原始服务器之间的服务器，为了从原始服务器取得内容， 客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。 就比如 翻墙软件巴拉巴拉。----作用：1.隐藏客户端真实IP 2.提高访问速度

- **反向代理**：是指以代理服务器来接受 Internet 上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给 Internet 上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。---作用：1.负载均衡 2.隐藏服务器真实IP 3.提供安全保障 4.提高访问速度

apche 和 nginx 的优缺

- nginx轻量级，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，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。apache 相对于nginx 的优点：rewrite比nginx 的rewrite 强大，少bug，稳定。（需要性能用nginx，求稳定就apache）。

Docker

> 一句话概括容器：容器就是将软件打包成标准化单元，以用于开发、交付和部署。
>
> - 镜像(Image): 一个特殊的文件系统
> - 容器(Container): 镜像运行时的实体, 容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的 命名空间。
> - 仓库(Repository): 集中存放镜像文件的地方

什么是 Docker ?

1. Docker 是世界领先的软件容器平台，基于 Go 语言 进行开发实现。
2. Docker 能够自动执行重复性任务，例如搭建和配置开发环境，从而解放开发人员。
3. 用户可以方便地创建和使用容器，把自己的应用放入容器。容器还可以进行版本管理、复制、分享、修改，就像管理普通的代码一样。
4. Docker 可以对进程进行封装隔离，属于操作系统层面的虚拟化技术。 由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。

- 容器镜像是轻量的、可执行的独立软件包 ，包含软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置。
- 容器化软件适用于基于 Linux 和 Windows 的应用，在任何环境中都能够始终如一地运行。
- 容器赋予了软件独立性，使其免受外在环境差异（例如，开发和预演环境的差异）的影响，让团队专注于开发，而不用在意部署的问题

Docker 容器的特点

- **轻量** : 在一台机器上运行的多个 Docker 容器可以共享这台机器的操作系统内核；它们能够迅速启动，只需占用很少的计算和内存资源。镜像是通过文件系统层进行构造的，并共享一些公共文件。这样就能尽量降低磁盘用量，并能更快地下载镜像。
- **标准** : Docker 容器基于开放式标准，能够在所有主流 Linux 版本、Microsoft Windows 以及包括 VM、裸机服务器和云在内的任何基础设施上运行。
- **安全** : Docker 赋予应用的隔离性不仅限于彼此隔离，还独立于底层的基础设施。Docker 默认提供最强的隔离，因此应用出现问题，也只是单个容器的问题，而不会波及到整台机器。

为什么要用 Docker ?

1. **一致的运行环境** ：Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现 “这段代码在我机器上没问题啊” 这类问题

2. **更快速的启动时间**：可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间
3. **隔离性**：避免公用的服务器，资源会容易受到其他用户的影响
4. **弹性伸缩，快速扩展**：善于处理集中爆发的服务器使用压力
5. **迁移方便：** 可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。
6. **持续交付和部署**：使用 Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署
