1. 软件开发常用四种模式：瀑布模型，快速原型模型，迭代模型，敏捷模型。
2. 敏捷开发
3. 迭代开发：将开发过程拆成多个小周期。开发者先快速发布一个有效但不完美的最简版本，每一次迭代都包含**规划 设计 编码 测试 评估**五个步骤
4. 增量开发：软件的每个版本都会新增一个用户可以感知的完整功能，也就是按照新增功能来划分迭代。

优点：早期交付，降低成本；降低风险 （每个迭代大约持续2-6周）

1. 进程是系统进行资源分配和调度的**基本单位**，每个进程有自己的**独立内存空间**，进程**切换开销**大，但相对**稳定安全**。
2. 线程是进程的一个实体，是进程的一个执行单元，是CPU调度和分派的**基本单位**。

同属一个进程的线程**共享**进程全部资源。切换快但不稳定，可以并发执行，。一个线程可以**创建撤销**另一个线程。程序中同时运行多个线程完成不同工种称为多线程。

1. 协程是用户态的轻量级线程，调度由用户控制。不需锁机制，切换快。
2. 线程协程比较
3. 线程或进程可拥有多个协程
4. 线程进程都是同步机制，协程是异步机制
5. 协程能保留上一次调用时的状态。
6. 线程进程比较
7. 地址空间：线程是进程内的一个执行单元 也是进程内的可调度实体。线程共享进程的地址空间，而进程有自己独立的地址空间。
8. 资源拥有：进程是资源分配和拥有的单元，同一进程的线程共享进程的资源
9. 线程是CPU调度和分派的基本单位 进程不是。

线程是执行流的最小单位，

线程是指进程内的一个执行单元 也是进程内的可调度实体

1. **线程vs进程**
2. 进程是系统进行资源分配和调度的**最小单位**，线程是程序执行的**最小单位**。
3. 进程有自己独立的**地址空间**，每个进程都由系统分配地址空间，建立数据表来维护代码段，堆栈段，数据段。而线程是共享进程中的资源的，使用相同的地址空间。
4. 线程之间**通信**方便，同一进程下的线程共享全局变量、静态变量等数据，进程通信需要一些方式。
5. 多进程更**健壮**，多线程只要一个线程死掉，整个进程也死掉了，但多进程不是。
6. **线程安全**：指某个函数，函数库在多线程环境中被调用时，能够正确处理多线程之间的共享变量。
7. 线程安全：就是线程访问公共资源时，采用了加锁机制，当一个线程访问该类的某个数据时，进行保护，其他线程不能进行访问直到该线程处理完。不会出现数据不一致或数据污染。线程不安全就是不提供数据访问保护，有可能多个线程先后更改数据造成所得到的数据是脏数据。
8. 使用synchronized同步代码块，或者lock锁

Synchronized原理：当两个并发线程访问同一对象object中的这个synchronized同步代码块时，一个时间内只能有一个线程执行该代码块。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

1. 线程共享变量改为方法局部级变量
2. **线程池：**基本思想还是对象池的思想。开辟一块**内存空间**存放众多线程，池管理器处理线程的执行调度。**当有线程任务时，从池中取一个线程执行，完成后放回池内**，**避免**反复创建线程对象带来的性能开销，节省系统资源，提高响应速度。（创建线程对象需要调用操作系统底层API，为线程分配一系列资源，应避免频繁创建和销毁。）
3. 线程池重要参数：最大线程数；核心线程数；线程无任务时的维持时间；前者的单位；线程工厂（创建线程）；阻塞队列（存储等待执行的任务）；拒绝处理任务时的策略。
4. **线程同步（也算通信方式）方式**
5. 锁机制（互斥锁和条件变量，读写锁）：只有拥有互斥对象锁 的线程才有权访问公共资源，因为互斥对象只有一个，就可以保证公共资源不会被多个线程同时访问。（读写锁是指多线程的读操作不互斥，写操作互斥）
6. 信号量：允许多线程同时访问同一资源 但要控制同一时间访问的最大线程数量。
7. 信号：通过通知操作保持多线程同步，还可以方便实现多线程优先级的比较。
8. 进程三状态：就绪 阻塞 运行
9. 线程从创建到消亡 要经历：创建 就绪 运行 阻塞 等待 时间等待 死亡七个状态。

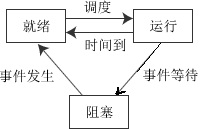
<https://blog.csdn.net/xingjing1226/article/details/81977129>

1. **创建**（NEW） new方法调用之后，start方法调用之前
2. **就绪**（READY） 线程对象创建后其他线程调用了该对象的 start方法，该状态的线程位于可运行线程池中等待被线程调度选中，获取CPU使用权
3. **运行**（RUNNING）：获取了cpu的时间片，执行run方法；以上两个状态统称RUNNABLE
4. **阻塞**（BLOCKED）：线程阻塞于锁
5. **等待**（WAITING）：等待其他线程做出一些特定动作（通知或中断）
6. **时间等待**（TIMED WAITING）：不同于前者，可以在指定时间后自行返回
7. **消亡**（TERMINATED）：线程执行完毕
8. 为何使用多线程？

<https://blog.csdn.net/zhh1072773034/article/details/74240897>

答：为了解决负载均衡问题,充分利用CPU资源.提高CPU的使用率,采用多线程的方式去同时完成几件事情而不互相干扰.为了处理大量的IO操作时或处理的情况需要花费大量的时间等等,比如:读写文件,视频图像的采集,处理,显示,保存等

1. **进程的通信方式**
2. 管道：半双工的通信方式，单向流动，只能亲缘关系进程通信。
3. 命名管道:也是半双工，但允许无亲缘关系进程间的通信（亲缘关系一般指父子进程）
4. 消息队列：消息构成的链表，存放在内核中并由消息队列标识符标识，克服了信号传递信息少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。
5. 共享内存：映射一段被进程共享的内存，是为针对其他进程通信方式运行效率低而设计的，往往与其他通信机制如信号结合使用。
6. 信号量：可以控制多个进程对共享资源的访问。
7. 信号：用于通知接收进程某事件已发生
8. 套接字：可用于不同机器间的进程通信。
9. 进程三个状态的转换条件：<https://www.cnblogs.com/caidi/p/6692231.html>



就绪状态 -> 运行状态：处于就绪状态的**进程被调度**后，获得处理机资源（分派处理机时间片），于是进程由就绪状态转换为运行状态。  
运行状态 -> 就绪状态：处于运行状态的进程在**时间片用完**后，不得不让出处理机，从而进程由运行状态转换为就绪状态。此外，在可剥夺的操作系统中，当有**更高优先级的进程就绪**时，调度程度将正执行的进程转换为就绪状态，让更高优先级的进程执行。  
运行状态 -> 阻塞状态：当**进程请求某一资源（如外设）的使用和分配或等待某一事件的发生（如I/O操作的完成）时**，它就从运行状态转换为阻塞状态。进程以系统调用的形式请求操作系统提供服务，这是一种特殊的、由运行用户态程序调用操作系统内核过程的形式。  
阻塞状态 -> 就绪状态：当**进程等待的事件到来**时，如I/O操作结束或中断结束时，中断处理程序必须把相应进程的状态由阻塞状态转换为就绪状态。

1. **进程的调度算法：**<https://blog.csdn.net/leex_brave/article/details/51638300>
2. **先来先服务**
3. **最短作业优先**
4. **最高响应比优先**
5. **时间片轮转算法**
6. **多级反馈队列**
7. **抢占式进程调度和非抢占式进程调度**[**https://blog.csdn.net/qq\_34173549/article/details/79936219**](https://blog.csdn.net/qq_34173549/article/details/79936219)
8. **非抢占式：**

**系统一旦把处理机分配给就绪队列中优先权最高的进程后，该进程便一直执行下去，直至完成；或因发生某事件使该进程放弃处理机时，系统方可再将处理机重新分配给另一优先权最高的进程。这种调度算法主要用于批处理系统中；也可用于某些对实时性要求不严的实时系统中。**

1. **抢占式：**

**系统同样是把处理机分配给优先权最高的进程，使之执行。但在其执行期间，只要又出现了另一个其优先权更高的进程，进程调度程序就立即停止当前进程(原优先权最高的进程)的执行，重新将处理机分配给新到的优先权最高的进程。也就是每出现一个进程就与当前进程做优先级比较。**

**这种抢占式的优先权调度算法能更好地满足紧迫作业的要求，故而常用于要求比较严格的实时系统中，以及对性能要求较高的批处理和分时系统中。**

1. **大小端字节序**

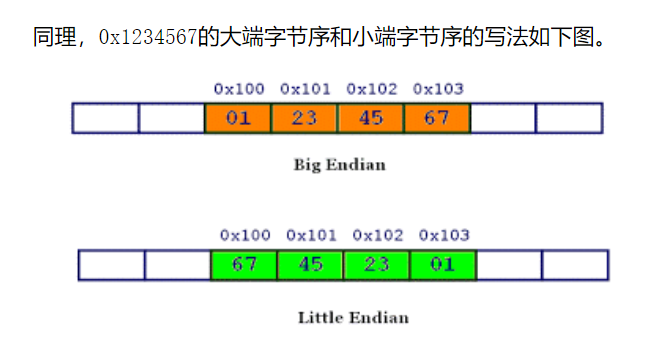
[**https://www.cnblogs.com/gremount/p/8830707.html**](https://www.cnblogs.com/gremount/p/8830707.html)

**计算机硬件有两种存储数据的方式，大端字节序和小端字节序。**

**大端字节序：高位字节在前，低位字节在后，是人类读写数值的方式，所以网络传输和文件传输都是大端字节序。**

**小端字节序：低位字节在前，高位字节在后，这是计算机内部处理使用的，因为计算都是从低位开始。小端字节序方便计算机电路优先处理低位字节。**

**比如0x1234567**

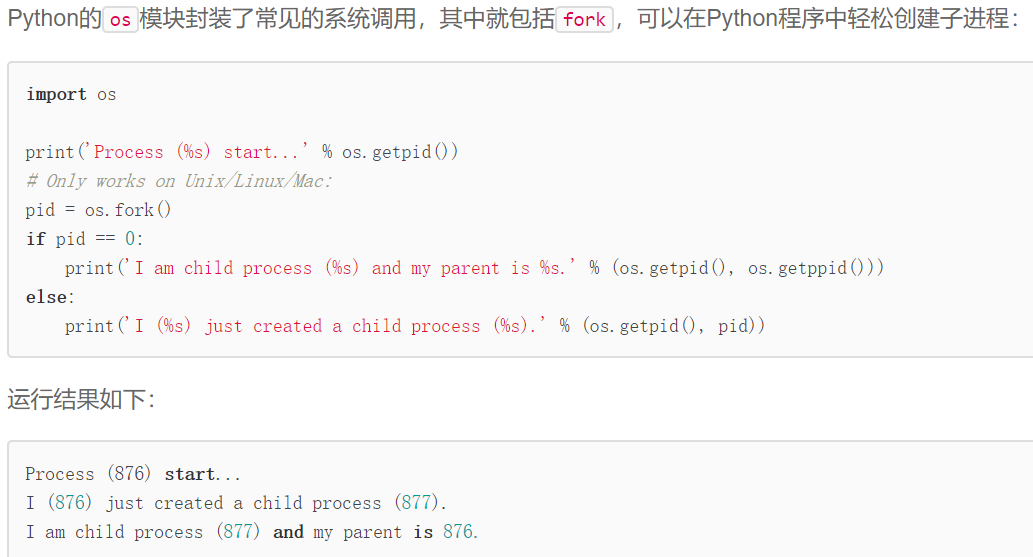


1. **黑盒测试 白盒测试常用测试方法：**<https://wenda.so.com/q/1531383235210734>
2. **黑盒测试的测试方法有：等价类划分、边界值分析法、猜错法、随机数法、因果图**
3. **白盒测试的测试方法有：代码检查法、程序变异、静态结构分析法、静态质量度量法、符号测试法、逻辑覆盖法、域测试、Z路径覆盖和基本路径测试法**

**编程：计算开平方的值？ 二分法！**

1. 从前端后端服务器阐述web达到性能优化
2. 前端：减少http请求（例如制作精灵图）；HTML和css放在页面上部，JavaScript放在页面下部（js加载较慢，优先加载html和css防止界面显示不全，影响用户体验）
3. 后端：如果有耗时操作，采用**异步**方式；代码优化，**避免循环和判断**次数太多（将最有可能的if前置）；**缓存**存储读写次数高，变化少的数据（比如网站首页信息，商品信息），先读取缓存（若没有或已失效）再访问磁盘数据库
4. 数据库：数据存放于redis，读取速度快；建立索引 外键等。
5. 1. Linux（Win不行）提供了一个fork（）系统调用，操作系统自动把当前进程（父进程）复制了一份（子进程），称为子进程；
6. 该函数被调用一次但返回两次，分别在父进程和子进程内返回，两次返回的区别是，子进程的返回值是0，而父进程的返回值是子进程的进程id。

这样做的理由是，一个父进程可以fork出许多子进程，所以父进程要记录子进程id，而子进程只需调用os.getppid()就可以拿到父进程的id。**两个进程共享代码空间但是数据空间独立，子进程数据空间的内容是父进程数据空间的完整拷贝。**



Window没有fork调用，上述代码无法在Win中执行。

1. 哈希表

哈希表的各种操作复杂度都是O(1)

1. Map va Hash Map （STL）
2. Map是红黑树存取的 查找时间复杂度是log n
3. Hash map是hash表存取的 O(1)
4. Hash map 牺牲内存换取速度
5. 多线程 join()和detach()

Join()：线程运行完 main函数才能结束；detach()：main函数不必等线程结束才能结束

## 计算机网络

1. 网络协议是什么？

在计算机网络中要做到有条不紊地交换数据，就必须遵守一些事先约定好的规则，这些规则称为网络协议。

1. 为什么要对网络协议分层？
2. 简化问题难度和复杂度。由于各层独立，可以分割大问题为小问题。
3. 灵活性好。只要层间接口关系不变，一层的技术变化不影响其它层。
4. 易于实现维护
5. 促进标准化工作。

缺点：功能可能出现在多个层中，产生额外开销。

1. OSI七层协议模型：应用层，表示层，会话层，运输层，网络层，数据链路层，物理层

五层协议的体系结构：应用层，运输层，网络层，数据链路层，物理层

TCP/IP是一个四层体系结构，应用层，运输层，网际层和网络接口层。

1. TCP/IP协议簇：
2. 应用层：

通过应用进程间的交互来完成特定的网络应用。应用层协议指应用进程间通信和交互的规则。

应用层协议：DNS（域名系统），HTTP，SMTP（支持电子邮件）

1. 运输层：

向进程通信提供通用的数据传输服务。

主要使用两个协议：TCP（面向连接的，可靠的）UDP（无连接的，不保证数据可靠性）

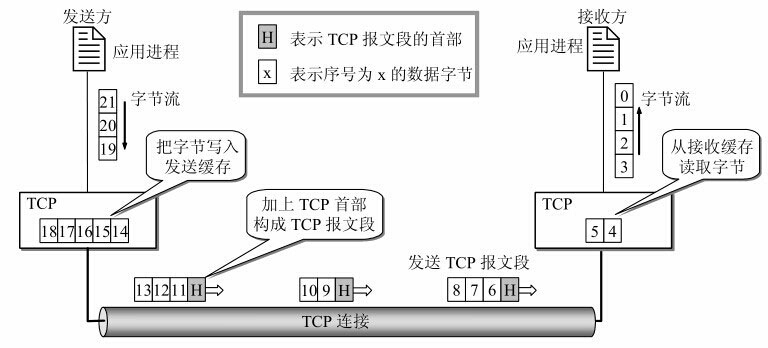
|  |  |  |
| --- | --- | --- |
|  | TCP | UDP |
| **是否连接** | 面向连接 | 无连接 |
| **是否可靠** | 可靠传输，使用拥塞避免和流量控制 | 不可靠传输，无拥塞避免和流量控制 |
| **连接对象个数** | 只能是一对一 | 支持一对多，多对一，一对一和多对多交互通信 |
| **传输方式** | 面向字节流 | 面向报文 |
| **首部开销** | 最小20字节最大60 | 仅8字节 |
| **场景** | 适用于要求可靠传输的应用（如文件传输） | 适用于实时应用（视频会议，直播等） |

**面向报文：**UDP对应用程序交付的报文，加上首部后就交给IP层，不合并不拆分，接受来自IP层的报文时，去掉首部后就交付上层的应用进程。也就是说UDP一次交付一个完整的报文**。**

**面向字节流：**虽然应用进程与TCP的交互是一次一个的数据块，但TCP将其视为一连串的字节流，它不知道字节流的含义，但保证接收方收到相同的字节流。

**TCP将应用进程交付的字节流存放在TCP缓存中，根据对方给出的窗口值和网络拥塞程度决定发送窗口的大小，报文段的长度就是TCP缓存区的容量（因此会发生拆包粘包）（TCP是面向字节流，但传输的数据单元是报文段），而UDP的报文段长度是应用进程给出的。**

**TCP是端对端通信，连接的端点是socket（IP地址：端口号），UDP的报文首部存放了源IP 目的IP和 源端口号 目的端口号。**



1. 运行在TCP协议之上的协议：

HTTP，HTTPS，FTP（文件传输协议），SMTP（简单邮件传输协议），POP3（邮局协议）等

补充：支持电子邮件的协议：SMTP，MTP，POP3，IMAP4.

1. 运行在UDP上的协议：

BOOTP（启动协议，用于无盘设备），NTP（网络时间协议，用于网络同步），DHCP（动态主机配置协议，动态配置IP地址）

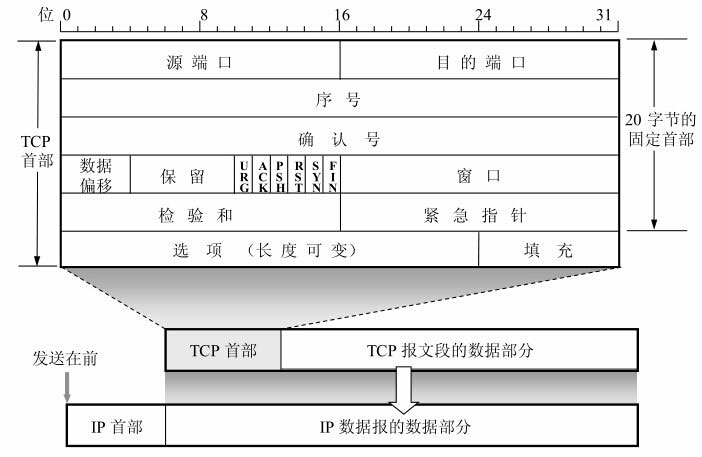
1. 运行在TCP和UDP上：

DNS（域名服务，用于完成地址查找，邮件转发等工作）

1. 网络层：

选择合适的网间路由和交换节点，确保数据及时传送。网络层将运输层的报文段或用户数据报封装成分组和包发送。由于网络层使用IP协议，分组也叫IP数据报。

1. 数据链路层和物理层 略
2. TCP/IP协议族：
3. 应用协议：HTTP， SMTP， FTP，SNMP，TELNET
4. 传输协议：TCP，UDP
5. 网际协议：IP，ICMP，ARP
6. 路由控制协议：RIP，BGP，OSPF。
7. 面向连接是指，客户端和服务端保存一份关于对方的信息用来验证身份，如ip地址，端口号等。
8. TCP传输：三次握手>数据传输>四次挥手
9. TCP报文头部结构：



1. 源端口号，目的端口号。
2. 序号：seq序号，标志报文所传输数据的第一个字节的序号。（报文中有多个字节）
3. 确认序号：ack序号，只有ACK标志位为1时才有效，标志期望收到对方下一个报文的第一个数据字节的序号。（TCP规定建立连接后，所有报文的ACK都置1）
4. 6个标志位：ACK（确认信号有效）；FIN（释放连接）SYN（建立新连接）等
5. 窗口：表示发送本报文的一方的接收窗口大小，（代表接收方允许对方发送的数据量，因为接收方数据缓存空间有限），发送方会根据该值设置自己的发送窗口大小。
6. **三次握手**：最开始客户端和服务器端都处于CLOSED状态，主动打开连接是客户端，被动打开连接的是服务器端

客户向服务器发出连接请求报文（SYN=1），服务器收到报文后，如果同意连接就返回确认报文（SYN=1，ACK=1），客户收到确认后，还要向服务给出确认(ACK=1)。。此时TCP连接建立，客户端进入ESTABLISHED状态，服务器收到确认后也进入E状态。

**为什么三次握手而不是两次握手？**

1. 防止已经失效的连接请求报文传送到服务器，从而产生错误。

如果客户发送了第一个连接请求但在网络节点中滞留时间过长，客户端一直没收到确认报文，认为服务器没有收到请求，于是再次发出请求，完成传输。此时第一次连接请求到达了服务器，这个连接本该失效，但是两次握手会使客户和服务器再次建立连接，导致不必要的错误和资源浪费。

如果使用三次握手，服务器对失效连接发出确认后，客户端不会再次确认，就避免了再次建立连接。

1. 防止服务端发给客户端的确认报文丢失。若2次握手，一旦确认报文丢失，客户端会认为没有建立连接，不会发送数据也不会接收服务器发来的数据。

**四次挥手：**

客户和服务器处于ESTABLISHED状态，客户端主动关闭，服务器端被动关闭。

**客户发出连接释放报文(FIN=1)且停止发送数据，服务器收到连接释放报文，发出确认报文(ACK)，**此时服务器进入CLOSE-WAIT状态；**客户端收到确认报文后，**进入FIN-WAIT-2状态，**等待服务器发送连接释放报文；服务器发送完最后数据后，向客户端发送连接释放报文，**进入LAST-ACK状态，等待客户端确认。**客户端收到连接释放请求后，发出确认报文，**进入TIME-WAIT状态，**经过2倍的MSL最长报文段寿命时间后，进入CLOSED状态；服务器只要收到客户端发来的确认立即进入CLOSED状态**

**为什么要等待2倍的最长报文寿命？**

1. **保证客户端发送的最后一个ACK确认报文能够到达服务器**，因为这个报文可能会丢失，服务器会重新发送连接释放报文，而客户端会在2MSL时间内收到这个重传报文，给出确认报文，并重启2MSL计时
2. 防止像三次握手中提到的**已失效连接请求出现在下次连接**中，客户端发送完最后一个确认报文后，2MSL的时间可以使本次连接的所有报文都从网络中消失，这样下次连接中就不会出现本次链接的请求报文。

**为什么握手三次，挥手四次？**

建立连接的时候，服务器在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在同一个报文里发送给客户端，因此只有三次。

而关闭连接时，服务器收到对方的FIN报文时，仅表示对方不再发送数据了，但可以接受数据，自己可能还有一些数据要发送，所以需要在所有数据发送完成后再发送FIN报文表示同意关闭连接，因此，己方FIN和ACK一般是分开发送，这样就多了一次。

**如果连接建立后，客户端出故障了？**

TCP有一个保活计时器，服务器每次收到客户端请求后重置计时器，计时器一般设为2小时，若两个小时没有收到客户端的请求，就每个75秒发送探测报文，10个探测报文没有反应就认为客户端故障，关闭连接。

1. **待解决问题：**

后面写了几个IP问哪些是私有地址？TCP好好在那里，UDP呢？UDP是啥？因为说了UDP也挺好，他问UDP从你的理解上好在哪里？

Linux操作基本命令，kill -7与kill -9的区别  
Session、Cookie和Token的主要区

1. IPV6如何实现即插即用？

  即插即用是指无需人工干预，就可以将一个节点插入IPV6网络并在网络中启动，IPV6网络地址是用了两种不同的机制来支持即插即用网络连接：启动协议（BOOTstrap Protocol,BOOTP)和动态主机配置协议（DHCP）。

1. **TCP可靠传输的实现**
2. **Servlet = service applet，是运行在服务器上的java小程序，实际上是一个接口，程序员处理用户请求时，定义一个专门处理某个url请求的类，该类实现Servlet接口，并重写该接口的五个方法（下面提到了），再把这个类在web.xml文件中配置匹配的url，配置该类的路径，也就是将我们写的这个java类注册到web服务器中，供服务器调用。tomcat可以在第一次收到用户请求时，自动创建实例（只创建一次），然后每次接到请求就调用一次该类的service方法。**
3. **Servlet是线程安全的吗？**

**不是线程安全的，多线程并发读写会导致数据不同步的问题。**

以下我看不懂，先写在这：解决的办法是尽量不要定义类的成员变量，而是在doGet(), doPost()方法内分别定义局部变量。虽然使用synchronized（name）{}语句块可以解决问题，但会造成线程等待。

注意：多线程的并发读写Servlet类属性会导致数据不同步，但只是并发读取属性而不写入就没有问题。因此，Servlet里的只读属性最后定义为final类型的。

1. **Servlet接口中有哪些方法及Servlet生命周期探秘**

在JavaWeb程序中，Servlet主要负责接收用户请求HttpServletRequest，在doGet() doPost()方法中处理，并将回应HttpServletResponse反馈给用户。Servlet可以初始化参数，供Servlet内部调用。

Servlet定义了五个方法，前三个与生命周期有关。

1. Void init()
2. Void service()
3. Void destroy()
4. getServletInfo()
5. getServletConfig()

生命周期：

Web容器加载Servlet并将其实例化后（这个实例好像是在static代码块里创建的，也就是init方法之前，由于static代码块只运行一次，因此是单例模式），Servlet生命周期开始，容器运行init（）方法进行Servlet的初始化。

请求到达时调用service方法，这个方法会根据需要调用与请求对应的doGet或doPost方法。

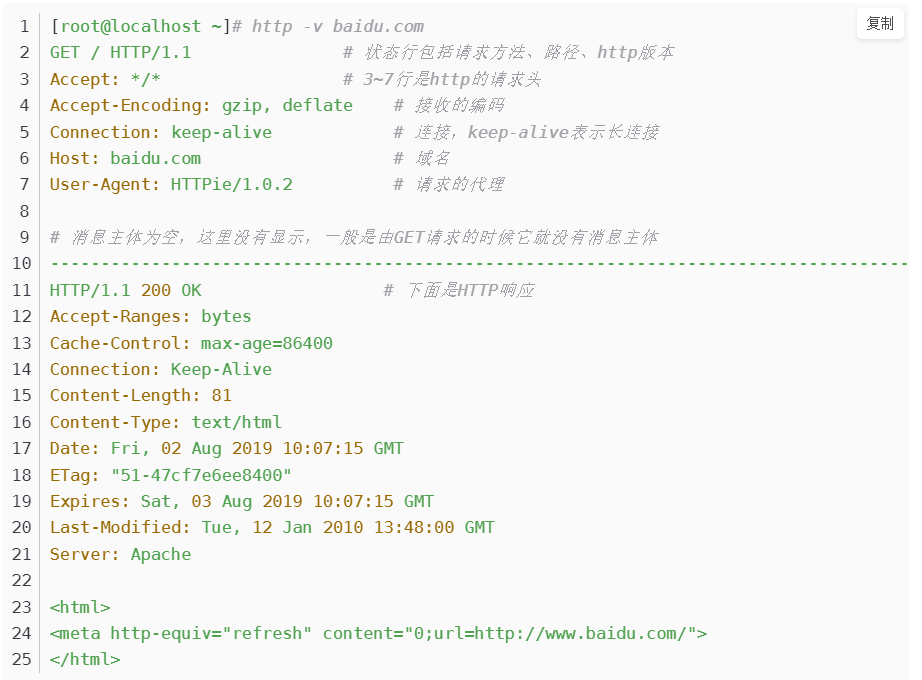
当服务器关闭或项目被卸载时服务器会销毁Servlet实例，此时调用destroy方法（在销毁实例之前）。

Init方法和destroy方法只调用一次，service方法在客户端每次请求Servlet时都会执行。

1. **Http请求流程**
2. 客户端调用请求函数向指定IP地址的服务器发出http请求
3. 服务器容器Tomcat收到http请求，寻找相应的Java Web编写的服务器应用程序，把请求分派给他处理
4. Java Web编写的服务器应用程序找到处理请求相应的Servlet实例，把请求分派给他处理
5. Servlet根据请求执行相应操作，将结果添加到应答中，发回客户端
6. 客服端收到应答，解析出结果。
7. **http的组成 https://www.cnblogs.com/wanghuaqiang/p/12093563.html**

请求的组成：状态行（请求方法、路径、http版本）、请求头、消息主体

响应的组成：状态行（HTTP版本、状态码）、响应头、响应正文



**提交一些表单的时候HTTP请求里面就会包含消息主体。**

请求头内容：

1. Accept
2. Accept-charset
3. Accept-Encoding
4. Connection
5. If-none-match
6. If-modified-since
7. Host
8. Refer
9. User-agent等

响应头内容：

1. Accept-range 是否接受对某对象的部分请求
2. Content-encoding
3. Content-length
4. Content-Type
5. Etag
6. Last-modified

响应实体（响应正文）

User-Agent是用户代理，是一个特殊字符串，服务器能够识别客户所使用的操作系统及版本号，以及浏览器相关信息。

1. **http协议的八种请求方法**

HTTP/1.1协议中共定义了八种方法（有时也叫“动作”），来表明Request-URL指定的资源不同的操作方式

1、OPTIONS  
返回服务器针对特定资源所支持的HTTP请求方法，也可以利用向web服务器发送‘\*’的请求来测试服务器的功能性  
2、HEAD  
向服务器索取与GET请求相一致的响应，只不过响应体将不会被返回。可以在不必传输整个响应内容的情况下，获取包含在响应消息头中的元信息。  
3、**GET  
向特定的资源发出请求**。它本质就是发送一个请求来取得服务器上的某一资源。资源通过一组HTTP头和呈现数据（如HTML文本，或者图片或者视频等）返回给客户端。GET请求中，永远不会包含呈现数据。  
4、**POST  
向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中**。POST请求可能会导致新的资源的建立和/或已有资源的修改。 Loadrunner中对应POST请求函数：web\_submit\_data,web\_submit\_form  
5、**PUT  
向指定资源位置上传其最新内容**  
6、**DELETE  
请求服务器删除Request-URL所标识的资源**  
7、TRACE  
回显服务器收到的请求，主要用于测试或诊断  
8、CONNECT  
HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

**http1.0、 1.1、 1.x、2.0、 3.0的区别 参考下文：**[**https://blog.csdn.net/glpghz/article/details/106063833**](https://blog.csdn.net/glpghz/article/details/106063833)

1. **HTTP1.0**
2. **使用非持久连接，客户端必须为每一个待请求的对象建立并维护一个连接。**一个页面可能存在多个对象，所以非持久连接可能使一个页面的下载十分缓慢，这种短连接增加了网络负担。
3. **http1.1和1.0的区别 （缓范码ho长）**
4. **在http1.0基础上引入了更多的缓存控制策略**，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Entity tag，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。
5. **引入请求范围设置，节省了带宽**，*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。
6. **错误通知的管理中新增了错误状态响应码**，在**HTTP1.1中新增了24个错误状态响应码**，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。
7. **增加Host头处理，可以传递主机名**，在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且共享一个IP地址。HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。
8. **支持长连接**，HTTP 1.1支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，在HTTP1.1中默认开启Connection： keep-alive，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点。

1. **http和https的区别**

|  |  |  |
| --- | --- | --- |
|  | http | https |
| 传输安全性 | http协议是超文本传输协议，信息是明文传输 | https协议是具有安全性的ssl加密传输协议 |
| 连接方式 | 连接方式简单，是无状态的 | 是由ssl+http协议构建的，可进行加密传输、身份认证 |
| 资源消耗 | 较少 | 由于加解密处理，消耗CPU和内存资源 |
| 端口 | 80 | 443 |
| 证书申请方式 | 免费申请 | 一般需要到ca交费申请 |

CA：Certificate Authority的简称，指电子商务认证授权机构

1. http是请求-响应协议，除此之外websocket协议（利用http建立连接），可实现浏览器-服务器之间全双工通信。
2. ARP协议 根据IP地址获取物理地址的一种TCP/IP协议。
3. 常见HTTP状态码：
4. 1xx：请求正在处理
5. 2XX：请求正常处理完毕
6. 3xx：重定向状态码
7. 4xx：客户端错误状态码
8. 5xx：服务器端错误状态码
9. **常见状态码及意义：**
10. 100 Continue 客户端应当继续发送请求的剩余部分
11. 200 OK 请求正常处理
12. 201 Created 请求已经被实现并且一个新资源已经依据请求的需要而建立
13. **202 Accepted** 服务器已接收请求，但尚未处理，该请求最终不一定会执行
14. 301 Moved Permanently 永久重定向，资源已永久分配新URI
15. 302 Found 临时重定向
16. **401 Unauthorized** 当前请求需要用户验证
17. 403 Forbidden 请求被拒绝
18. 404 Not Found 无法请求到资源
19. 500 Internal Server Error 服务器故障或web应用故障，无法完成对请求的处理，
20. **503 Service Unavailable** 服务器超负载或停机维护，无法处理请求，这个状态是临时的，将在一段时间后恢复
21. **HTTP的缓存机制** [**https://www.cnblogs.com/chenqf/p/6386163.html**](https://www.cnblogs.com/chenqf/p/6386163.html)

**重要！！！**

1. **POST和GET的区别：**
2. GET参数通过url’传递，而POST放在request body中。但是也可以给get请求加上request body，给post请求加上url参数，只是http协议不建议这样。
3. GET请求的参数暴露在url上，比较不安全。
4. Get请求提交的url长度有限，这个限制是浏览器服务器添加的，不是来自http协议，post请求则没有大小限制
5. **GET方法把http header 和data一同发送，服务器响应200ok，而POST分两步发送，服务器分别响应100 continue和200 ok。**

**后者的优势：**在网络环境好的情况下，二者时间差别很小，而网络环境差的情况下，两次包的TCP在验证数据包完整性上有很大优点。（不是所有浏览器都在POST方法发送两次包）

1. GET在浏览器回退时无害，而POST会再次提交请求
2. GET会被浏览器主动cache，而POST不会，除非手动设置
3. GET请求参数被完整保存在浏览器历史记录里，而POST参数不会被保留

所以，GET和POST本质上都是无差别的TCP连接，只是http的规定和浏览器服务器的限制导致二者体现出一些不同。

1. DNS域名解析过程：
2. 浏览器检查自身缓存中有没有解析过这个域名的ip地址，如果有则解析结束
3. 浏览器检查操作系统中有没有对该域名的解析结果。这个解析结果一般在/etc/hosts文件（Linux系统）里
4. 请求本地域名服务器（LDNS）来解析这个域名。LDNS一般在你的城市某角落，且性能很好，可以解决80%的域名解析
5. 如果LDNS没有命中，就跳到Root Server根域名服务器请求解析
6. 根域名服务器返回给LDNS一个**所查询域的主域名服务器（gTLD Server）地址**
7. 此时LDNS再发送请求给上一步返回的gTLD主域名服务器
8. 接收请求的主域名服务器查找并**返回这个域名对应的Name Server地址**，这个NameServer就**是网站注册的域名服务器**
9. NameServer根据映射关系找到目标ip返回给LDNS
10. LDNS缓存这个域名和对应ip
11. LDNS把解析结果返回给用户，用户根据TTL值缓存到本地系统缓存中。解析结束。
12. HTTP1.x优化（SPDY）
13. 多路复用，为多路复用设立了请求优先级；
14. 对header部分进行了压缩；
15. 引入了https加密传输；
16. 客户端可以在缓存中取到之前请求的内容。
17. Http2.0新特性（SPDY的升级版）
18. **基于二进制格式解析**（Binary Format），HTTP1.x的解析是基于文本。**文本的表现形式有多样性，很多场景下都需要考虑健壮性，二进制则不同，只认0和1的组合**。基于这种考虑HTTP2.0的协议解析决定采用二进制格式，实现方便且健壮。

**实现方法是：**在应用层(HTTP2.0)和传输层(TCP or UDP)之间增加一个二进制分帧层。   
在二进制分帧层上， HTTP 2.0 会将所有传输的信息分割为更小的消息和帧,并对它们采用二进制格式的编码 ，其中HTTP1.x的首部信息会被封装到Headers帧，而我们的request body则封装到Data帧里面。

1. **多路复用**（MultiPlexing），即连接共享，

基于二进制分帧层，HTTP2.0可以在共享TCP链接的基础上同时发送请求和接收响应。HTTP消息被分解为独立的帧，不破坏消息本身的语义，交错发出去，在另一端根据流标识符和首部将他们重新组装起来。

即每一个request都是是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面。

**http1.1是多个请求串行化单线程处理，2.0是并行执行，一个请求阻塞不影响其他请求。**

1. **首部（header）压缩**，HTTP 2.0 在客户端和服务器端使用“首部表”header fields来跟踪和存储之前发送的键-值对，通信期间几乎不会改变的通用键-值对(用户代理、可接受的媒体类型,等等)只需发送一次。事实上,如果请求中不包含首部(例如对同一资源的轮询请求),那么 首部开销就是零字节。此时所有首部都自动使用之前请求发送的首部。

如上文中所言，对前面提到过HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。

1. **服务端推送**（server push），就是服务器可以对一个客户端发送多个响应。当浏览器请求一个html，服务器其实大概知道你是接下来要请求资源了，就将相应的资源推送，而不需要等待浏览器得到html解析页面后再发送资源请求。我们常用的内嵌图片也可以理解为一种强制的服务器推送：我请求html，却内嵌了张图。
2. **请求优先级**
3. 把HTTP消息分为很多独立帧之后，就可以通过优化这些帧的交错和传输顺序进一步优化性能。
4. 什么是请求优先级  
   每个流都可以带有一个31bit的优先值：0表示最高优先级；2的31次方-1表示最低优先级。
5. 请求优先级如何工作  
   客户端明确指定优先级，服务端可以根据这个优先级作为交互数据的依据，比如客户端优先设置为.css>.js>.jpg。服务端按此顺序返回结果更加有利于高效利用底层连接，提高用户体验。然而，在使用请求优先级时应注意服务端是否支持请求优先级，是否会引起队首阻塞问题，比如高优先级的慢响应请求会阻塞其他资源的交互。

2016.3 2015.12

**HTTP2.0多路复用提升了网页性能**

1. HTTP1中浏览器限制了同一域名下的请求数量（Chrome下是6个），当浏览器达到最大请求数量时，剩余资源需等待当前六个请求完成后。
2. Http2.0的多路复用可以只通过一个tcp连接就传输所有请求，绕过浏览器对同一域名请求数量的限制，提升网页性能。
3. **Http3.0（QUIC）特性**

QUIC( Quick UDP Internet Connection)

1. **线头阻塞(HOL)问题的解决更为彻底：**  
   基于TCP的HTTP/2，尽管从逻辑上来说，不同的流之间相互独立，不会相互影响，但在实际传输方面，数据还是要一帧一帧的发送和接收，一旦某一个流的数据有丢包，则同样会阻塞在它之后传输的流数据传输。而基于UDP的QUIC协议则可以更为彻底地解决这样的问题，让不同的流之间真正的实现相互独立传输，互不干扰。
2. **切换网络时的连接保持：**  
   基于TCP的协议，一旦切换网络导致ip地址改变，原先的连接就无法保持，而基于UDP的QUIC协议，使用与TCP不同的连接标识方法，在切换网络后仍可以恢复与服务器的连接。

当前移动端的应用环境，用户的网络可能会经常切换，比如从办公室或家里出门，WiFi断开，网络切换为3G或4G。基于TCP的协议，由于切换网络之后，IP会改变，因而之前的连接不可能继续保持。而基于UDP的QUIC协议，则可以内建与TCP中不同的连接标识方法，从而在网络完成切换之后，恢复之前与服务器的连接。

1. HTTP2.0的多路复用和HTTP1.x中的长连接复用有什么区别？
2. HTTP/1.\* 一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接；
3. HTTP/1.1 Pipeline解决方式为，若干个请求排队串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会，一旦有某请求超时等，后续请求只能被阻塞，毫无办法，也就是人们常说的线头阻塞；
4. HTTP/2多个请求可同时在一个连接上并行执行。某个请求任务耗时严重，不会影响到其它连接的正常执行；
5. 私有ip有哪些？

公网IP是在Internet使用的IP地址，而私有IP地址是在局域网中使用的IP地址。

A: 10.0.0.0~10.255.255.255 即10.0.0.0/8 #32-8=24 作为2的指数 就是ip地址数量

B:172.16.0.0~172.31.255.255即172.16.0.0/12

C:192.168.0.0~192.168.255.255 即192.168.0.0/16

1. **Ip地址与子网掩码**，及相关计算 参考：<https://blog.csdn.net/yinshitaoyuan/article/details/51782330>和<https://blog.csdn.net/twypx/article/details/87085306>

Ip地址共8\*4=32个二进制位（一般用4个十进制数表示），分ABC类，他们的网络位的基础数分别为8，16，24。子网掩码一定是前一部分全是1，后一部分全是0. 子网掩码与ip相与，得到的就是网络号（也就是子网掩码为1的部分对应网络号，子网掩码为0的部分对应主机号）。

给子网掩码，求子网的个数：子网掩码的1的个数减去对应的某一类的ip地址的基础位数，得到的数作为2的指数，结果就是子网个数。

求每个子网的主机个数：子网掩码的0的个数作为2的指数再-2，就是每个子网中可容纳的主机个数（减2是减去全0 或全1的主机号，这两个主机号不可用）

比如c类ip地址192.168.3.0，子网掩码为255.255.255.240. 子网掩码1 的位数为3\*8+4 = 28位，减去c类地址基础位数3\*8 = 24， 得到4，那么2^4 = 16就是子网个数。子网掩码中0的个数为12，那么每个子网可容纳的主机数就是 2^12-2.

一个地址可能表示为172.16.0.0/12，后面的/12就是子网掩码的1的位数，这样就把ip地址和子网掩码都表示了。

1. 工厂模式
2. Redis常用命令

<https://blog.csdn.net/qq_40140473/article/details/81201017>

1. **https加密过程**[**https://www.cnblogs.com/jtlgb/p/6760679.html**](https://www.cnblogs.com/jtlgb/p/6760679.html)
2. 客户端请求服务器，返回证书 公钥（第一次，第二次握手）
3. 客户端解析证书
4. 客户端生成随机对称密钥
5. 用公钥加密对称密钥
6. 将加密的密钥传递给服务器（第三次握手）
7. 服务器用私钥解密得到对称密钥
8. 接下来双方就可以使用对称密钥加密数据来进行传输。
9. 服务器使用对称密钥加密信息，发送给客户端
10. 客户端用对称密钥解密数据
11. 对称加密与非对称加密：

**对称密钥加密：加密和解密使用同一个密钥**，问题在于如何安全将密钥发送给对方。

**非对称加密：使用非对称密钥，即公钥和私钥**，公钥随意发布，私钥私有。发送方使用对方的公钥加密，接收方使用自己的私钥解密。

后者保证了安全性，但速度慢。

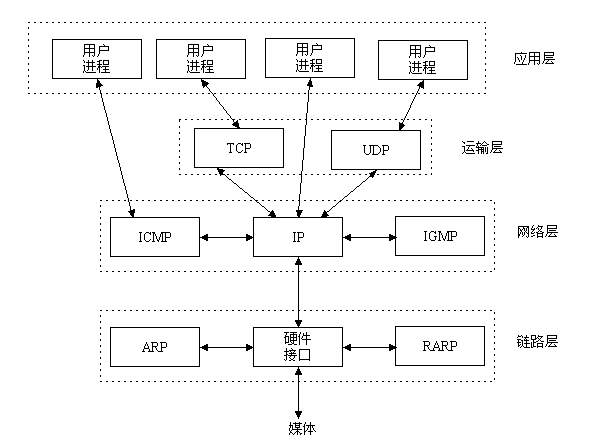
1. 什么是http2？http1 http1.1 http2的区别

http2可以提高网页性能。

http1中浏览器限制了同一域名下的请求数量

http2使用多路复用技术，可以只通过一个TCP连接就传输所有的请求数据。可以绕过浏览器的限制，提高网页性能。

1. **TCP协议：在IP协议提供的主机间通信功能基础上，完成两个主机上进程对进程的通信。给进程分配唯一的数字来标识它，就是端口号。**
2. **UDP不需要三次握手就可以直接发送数据，非面向连接，只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。**
3. TCP/IP即传输控制/网间协议，是一个工业标准的协议集，为广域网WANs设计的。
4. UDP是与TCP相对应的协议，属于TCPIP协议族里的一种。



1. TCPIP协议族包括运输层，网络层，链路层。
2. Socket（孔，插座）实现长连接：要区分一个数据包是心跳还是真实数据。



网络中进程之间如何通信？首要解决的问题是如何唯一标识一个进程，否则通信无从谈起！在本地可以通过进程PID来唯一标识一个进程，但是在网络中这是行不通的。其实TCP/IP协议族已经帮我们解决了这个问题，网络层的“**ip地址**”可以唯一标识网络中的主机，而传输层的“**协议+端口**”可以唯一标识主机中的应用程序（进程）。这样利用三元组（ip地址，协议，端口）就可以标识网络的进程了

1. **死锁:** 是操作系统或软件运行的一种状态：**两个或以上的进程在执行过程中，因争夺资源而造成的一种互相等待彼此资源释放的情况。**在多任务系统下，当一个或多个进程等待系统资源，而资源又被进程本身或其它进程[占用](https://baike.so.com/doc/496682.html)时，就形成了死锁。
2. 死锁的四个必要条件：
3. 互斥条件：一个资源每次只能被一个进程调用 (进程对分配到的资源进行排他性使用)
4. 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放（已获得资源但又提出新的资源请求而新资源被其他进程占用，请求阻塞，但又对自己已获得的资源保持不放）
5. 不剥夺条件：进程已获得的资源在未使用完之前不能强行剥夺（只能使用完后自己释放）
6. 环路等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。（A等B占用的资源，B等C，而C又在等A，此为环路）
7. 如何避免死锁：

在并发程序中，避免了逻辑中出现复数个线程互相持有对方线程所需要的独占锁的情况 就可避免。（独占锁就是一个线程对资源的排他性使用）

1. **解决死锁问题**

**可剥夺资源**：即当某进程新的资源未满足时，释放已占用资源

**资源有序分配**法：系统给每个资源赋予编号，每个进程按编号递增顺序请求资源，释放则相反。（破坏环路等待条件）

**资源一次性分配**：（破坏请求和保持条件）

**避免死锁：‘银行家算法’** 系统在进行资源分配前预先计算资源分配的安全性，安全则分配给进程，否则进程等待

**银行家算法的核心思想：**

1. 当进程**首次申请资源**时，要**测试**该进程对资源的**最大需求量**，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。
2. 当进程在执行中继续申请资源时，先测试该进程**已占用的资源数与本次申请的资源数之和**是否超过了该进程对资源的最大需求量。若超过则拒绝分配资源，若没有超过则再测试**系统现存的资源能否满足该进程尚需的最大资源量**，若能满足则按当前的申请量分配资源，否则也要推迟分配。

**消除死锁：终止进程或进程回滚**。

1. TCP流量控制：

由滑动窗口协议实现，接收方返回的ACK中会包含自己的接收窗口大小，并且利用大小来控制发送方的数据传送。

1. TCP拥塞控制

发送方维护一个拥塞窗口的状态变量，大小取决于网络拥塞程度，且动态变化，发送方的发送窗口取拥塞窗口和接收方接收窗口中较小的一个。

就是为了：防止过多的数据注入到网络中，避免出现网络负载过大的情况

1. 慢启动，拥塞避免
2. 快重传，快恢复
3. **慢启动和拥塞避免**

慢启动：不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小。

拥塞窗口一开始设为1 ，每收到一次确认，就让拥塞窗口变为原来的两倍，当窗口值为16时（慢启动门限），改为加法增大，每次+1，直到网络拥塞。拥塞时让新的慢启动门限设为拥塞时的一半，并把拥塞窗口置为1，再让他重复，这时一瞬间会将网络数据量大量降低。

拥塞避免：拥塞避免算法思路是让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。

1. 快重传和快恢复

快重传：接收方每收到一个失序的报文段（收完2后就收到了4说明3丢了）就立即发出包2的重复确认，这样可以让发送方尽早知道丢包了。

发送端连续收到三个重复确认就立即重传3

快恢复：发送方收到3个连续确认时，把慢启动门限减半，把拥塞窗口的值置为慢启动门限的一半，实行拥塞避免算法，每次确认收到后+1

1. 排序算法对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 平均 | 最好 | 最坏 | 辅助空间 | 稳定性 |
| 冒泡 | n2 | n | n2 | 1 | 稳 |
| 选择排序 | n2 | n2 | n2 | 1 | 不 |
| 插入排序 | n2 | n | n2 | 1 | 稳 |
| 堆排序 | n log n | n log n | n log n | 1 | 不 |
| 归并排序 | n log n | n log n | n log n | n | 稳 |
| 快速排序 | n log n | n log n | n2 | logn-n(递归) | 不 |
| 希尔排序 | n log n- n2 | n1.3 | n2 | 1 | 不 |

什么是稳定：就是能保证排序前2个相等的数其在序列的前后位置顺序和排序后它们两个的前后位置顺序相同。

**快排的最坏情况的时间复杂度如何避免？**

最坏情况指每次选择的枢纽分出的两部分其中一部分为空（一般数据有序或局部有序就会出现这种情况），最好的情况是每次都均分为两部分。

**避免**：每次都随机选择枢纽或者取多个随机数再取均值；事先扫描子数组，若有序就不用快排了。

**归并排序优化：**

归并前判断两边的数组是否一边最小大于另一边最大，这样就不需要归并了。

1. 哈希表简介

哈希表的特点：关键字与其在表中的位置之间存在一种确定的关系。

哈希函数：一般情况下，需要在关键字与它在表中的存储位置之间建立一个函数关系，以f(key)作为关键字为key的记录在表中的位置，通常称这个函数f(key)为哈希函数。

hash : 翻译为“散列”，就是把任意长度的输入，通过散列算法，变成固定长度的输出，该输出就是散列值。

这种转换是一种压缩映射，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来唯一的确定输入值。

           简单的说就是一种将任意长度的消息压缩到莫伊固定长度的消息摘要的函数。

hash冲突：就是根据key即经过一个函数f(key)得到的结果的作为地址去存放当前的key value键值对(这个是hashmap的存值方式)，但是却发现算出来的地址上已经有人先来了。

1. 哈希表散列冲突的解决方案：
2. 建立一个缓冲区（公共溢出区），把凡是拼音重复的放到缓冲区中。
3. 开放地址法（一旦冲突则去找下一个空的散列地址，只要散列表够大，空地址总能找到）：线性探测再散列，二次探测再散列，伪随机探测再散列
4. 再哈希法（算hashcode的方法不止一个，有重复再用另一个算法算）
5. 链地址法（最熟悉的）（Java Hashmap就是这种，冲突元素放到链表开头）
6. 开放地址法公式
7. 线性 fi(key) = (f(key)+di) MOD m （di=1,2,3,….m-1）
8. 二次 fi(key) = (f(key)+di) MOD m  (di = 1^2, -1^2, 2^2, -2^2,……, q^2, -q^2 (q <= m/2))
9. 随机 fi(key) = (f(key)+di) MOD m (di 为伪随机数列)

补充：伪随机数（计算机只会产生伪随机数，设置随机种子（默认从时钟获取），随机函数用一种计算方法根据随机种子可以生成一段随机数列，用相同的随机种子，得到的随机数列是相同的）

1. 树
2. 平衡二叉查找树是一个高度平衡的二叉树，也就是说树的高度差不能大于1，在删除的时候，可能需要多次调整，也就是左旋转、右旋转操作，在树的深度很大的情况下，删除效率会非常低.解决：**红黑树**
3. 红黑树的特性：
4. 红黑树基于二叉查找树（二叉搜索树），保证树的高度为log n 因此查插删操作都是 log n。
5. 是一颗基本平衡的二叉查找树；高效查插删且效率稳定。
6. 每个节点或者是黑色，或者是红色。
7. 根节点是黑色。叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]
8. 如果一个节点是红色的，则它的子节点必须是黑色的。
9. 从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。

<https://open-hl.toutiao.com/a6802917448916926980/?utm_campaign=open&utm_medium=webview&utm_source=o_llq_api&req_id=20200313073256010012035146196B1B14&dt=OPPO+R9sk&crypt=5991&label=open_o_channel&a_t=4115924741504904790338914b4&gy=272f281757a7364d955d52e4017d019981ec36a96ba6f693cd9401ded82144fdd4f7e01933aff3b39b66fa66d8a2fd6253408b594c413f370cd108700a2efce2468fac98b4a45df36d6e24fb7429694491d2d190bdf451b95d7f2cc3248b1eef12ef740e46925b7a3d4f8eeaf259d161bc1d816694f8e4ad7538501f4b884baea161229f70f472131aaaa9f9a33e8ec97ae79af3e4f424c479b18aab8c1247a9569cad0a67a796e21d3bb7f0db7e6a6860943369e23c303ec885e1acb9a27e2a&item_id=6802917448916926980&__docId__=6802917448916926980&__barStyle__=1_1&__fromId__=__all__&__statParams__=sourceMedia&__source__=toutiao&sourceMedia=toutiao&__cmtCnt__=3&__styleType__=2&__publisher_id__=83571748439&openv=&__pf__=detail>

1. 同源策略需要满足：协议相同（http和https协议不同）；域名相同（test.com与admmiin.com域名不同）；端口相同（test.com与test.com:9000端口不同）
2. Session cookie 和token的区别

http协议本身是无状态的，即服务器无法判断用户身份。

1. Cookie：有web服务器保存在浏览器上的小文件（key-value格式），包含用户相关信息。如果服务器需要记录用户状态，就通过response向客户端浏览器颁发cookie，客户端再次请求时都携带cookie，服务器检查cookie来辨认用户身份。
2. Session：是服务器浏览器会话过程中，服务器分配的一块存储空间。服务器在浏览器cookie中设置session\_id，浏览器携带cookie请求时，服务器根据session\_id获取会话中存储的信息，确定会话的身份信息。
3. Token：服务器生成的字符串，作为客户端进行请求的令牌。客户端频繁的请求数据会导致服务器端频繁向数据库查询用户名和密码进行身份对比，为了减轻服务器压力，避免频繁查询数据库。服务器对客户端进行身份认证后返回token，客户端请求时带上token证明合法地位。
4. 我的理解：cookie里可以携带用户名密码等信息，但是容易被截获，进而使用session，将用户信息放在服务器的session中，cookie携带session\_id，服务器根据session\_id在session中找用户信息；session可以存在数据库或缓存中，为了避免反复查询，使用token。
5. Session和token的区别：
6. Session在服务器端，token在客户端
7. Session机制存在服务器压力大，CSRF跨站伪造请求攻击，扩展性不强等问题
8. Token提供认证和授权功能，作为身份认证，token安全性好
9. Session适用于客户端代码与服务器代码运行在同一台服务器上，token适用于项目级的前后端分离（前后端代码运行在不同的服务器下）。
10. 简述**cookie和session**的区别：
11. 前者在客户端，服务端能够知道其中的信息，后者在服务器端，客户端不知道其信息。
12. Session保存的是对象，cookie保存的是字符串
13. Cookie安全性差于session
14. Cookie保存的数据有限，session无限制。
15. Session的运行依赖session id，而session id是通过cookie传递的，也就是说，浏览器禁用cookie会使session失效。

注意：服务器端通过session存储用户登录状态，而请求如何找到对应的session呢？答：利用sessionID，sessionID存放在客户端的cookie里，请求附带cookie用来找到对应的session会话。如果客户端禁用了cookie，可以通过url传递sessionid。

1. Session保存在服务器上，访问增多时占用服务器性能。性能优化上尽量使用cookie。
2. Session的几种实现方式/如何实现session在服务器间共享？
3. 粘性session：将用户锁定到某一个服务器上，该用户的每次请求都会转发到对应的服务器上。（简单，不需要对session做任何处理/若服务器故障，用户请求转移到第二个服务器上时，session就失效了）
4. Session复制：任何一个服务器上session改变（增删改），都将该session序列化广播给其他节点，保证session同步。（可容错，服务器间session能实时响应/对网络负荷造成一定压力）
5. Session共享：将session存储在mencached、redis（要求redis或mencached必须是集群）
6. 粘性session处理方式: 不同的服务器指定访问不同的mencached，多个memcahced之间信息同步。用户访问时首先在服务器创建session，将session复制一份到对应的memcached，也就是memcached只起备份作用，session的读写都在服务器上，用户的粘性服务器挂掉之后，用户的访问定位到其他服务器上，根据cookie中存储的session’id寻找session，找不到就去相应的memcached中复制一份到服务器上。
7. 非粘性session处理方式：memcached做主从复制，写session都往从memcached上写，读取都从主memcached上读取，服务器本身不存储session。
8. Session持久化到数据库：用数据库专门存储session信息（服务器故障session不会丢失/访问量大的情况下会对数据库造成很大压力）
9. Terracotta实现session复制：Terracotta的基本原理是对于集群间共享的数据，当在一个节点发生变化的时候，Terracotta只把变化的部分发送给Terracotta服务器，然后由服务器把它转发给真正需要这个数据的节点。可以看成是对第二种方案的优化。
10. 几种常见的服务器认证机制：
11. Oauth（开放授权）
12. Cookie-session
13. Token
14. 简述输入url到显示页面的过程：

1、输入网址

2、DNS解析

3、建立tcp连接

4、客户端发送HTTP请求

5、服务器处理请求

6、服务器响应请求

7、浏览器展示HTML

8、浏览器发送请求获取其他在HTML中的资源。

1. **MVC相关**

模型-视图-控制器的缩写，一种软件设计规范,一种程序开发设计模式，用一种业务逻辑，数据，界面显示分离的方法组织代码，提高了程序的可维护性、可扩展性、可移植性、可重用性。

模型用来提供数据、视图用来展示数据、控制器接收请求并交给模型处理并返回视图。

**优点：**

**可以为一个模型在运行时同时建立和使用多个视图。变化-传播机制可以确保所有相关的视图及时得到模型数据变化，从而使所有关联的视图和控制器做到行为同步。   
2、视图与控制器的可接插性，允许更换视图和控制器对象，而且可以根据需求动态的打开或关闭、甚至在运行期间进行对象替换。   
3、模型的可移植性。因为模型是独立于视图的，所以可以把一个模型独立地移植到新的平台工作。需要做的只是在新平台上对视图和控制器进行新的修改。   
4、潜在的框架结构。可以基于此模型建立应用程序框架，不仅仅是用在设计界面的设计中。**

**不足：**

1. **增加了系统结构和实现的复杂性。对于简单页面使用MVC会增加结构的复杂性，降低运行效率**
2. **视图与控制器之间虽然相互分离，却联系紧密，妨碍了他们的独立重用**
3. **视图对模型数据的低效率访问。依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问损害操作性能**
4. 一般高级的界面工具或构造器不支持MVC模式

## 数据库

1. 关系型数据库与非关系型数据库的区别：<https://www.360kuai.com/pc/9a8eaf818d4147d3b?cota=4&kuai_so=1&tj_url=so_rec&sign=360_57c3bbd1&refer_scene=so_1>

关系就是二维表格，关系型数据库就是二维表及其之间的联系所组成的一个数据组织。

优点：

1. 易于维护，都是表结构，格式一致，字段固定
2. 支持复杂操作：支持sql，可以多表之间实现复杂查询

缺点：

1. 读写性能差，不支持高效率读写
2. 固定的表结构，灵活度不高
3. 高并发读写场景下，硬盘io是一个瓶颈。

菲关系型数据库严格来说不是数据库，而是就是一种数据结构化存储方法的集合，可以是文档或键值对。

优点：

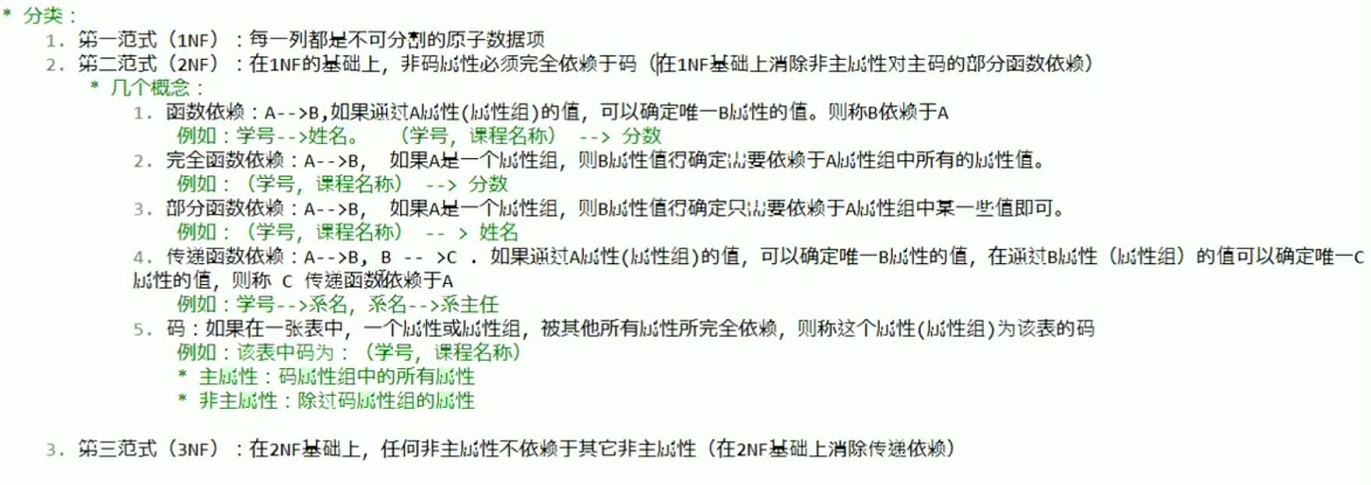
1. 格式灵活，速度快，高扩展性

缺点：

1. 不支持sql，无法进行复杂条件查询
2. 无事务处理

Redis应用场景：内容缓存，用于处理大量数据的高访问负载。查询速度快，且减小了数据库压力。

1. 数据库设计的三大范式：<https://www.cnblogs.com/linjiqin/archive/2012/04/01/2428695.html>
2. 确保每列的原子性，即每列都不可再分
3. 每列都与主键相关
4. 每列都与主键直接相关，而非间接相关



1. 数据库自增id越界后发生什么？

<https://blog.csdn.net/jeffrey_li/article/details/80320339>

可以将某个字段比如主键设置为自动增加（并且可以设置初始值）（属于缺省值，也就是插入数据时可以不设定值，会递增的自动分配一个值），插入数据时不必给出该字段的值，数据库会自动分配一个递增的值。

若插入数据将自增字段设置为某个值而该值越界，则插入时会报出提醒，实际插入值是距离最近的临界值（临界值要看设置的id范围是多大）

1. 前缀索引

<https://www.cnblogs.com/gomysql/p/3628926.html>

当索引是很长的字符序列时，这个索引将很占内存且很慢，这时要用到前缀索引。即取索引的前几个字符作为索引。前缀索引的关键在于要选择足够长的前缀以保证较高的选择性，同时又不能太长因为会占用内存空间。

1. **事务：**<https://blog.csdn.net/dengjili/article/details/82468576>

定义：由一系列语句构成的一个小的不可再分的逻辑工作单元，通常一个事务对应一个完整的业务（如银行账户转账业务）

一个完整的业务需要批量的DML语句（insert，update、delete）共同联合完成。

事务的四个特性（ACID）：

1. 原子性：事务是最小单位，不可再分，事务要求所有DML语句操作必须同时成功或失败
2. 一致性： 事务的执行不能破坏数据的完整性和一致性，即事务的执行结果必须使得数据库从一个一致性状态转移到另一个一致性状态。事务前后数据的完整性必须保持一致，举例，转账前后两个账户的总额是相同的。（这个很难理解）。
3. 隔离性：事务并发会导致各种问题，事务之间应存在一定隔离，即一个事务不能被其他事务干扰，隔离级别越高，数据安全性越高**。（涉及事务隔离的四个级别）**
4. 持久性：事务一旦提交，对数据库的改变就是永久性的（事务对内存数据的更改会永久保存到硬盘文件中）

**事务开始标志：任意一条DML语句执行**

**结束标志：提交或回滚；**

**提交：所有DML语句操作历史记录与硬盘数据同步。（只有事务成功结束时，才修改硬盘数据，否则只是在内存中保存历史操作记录）**

**回滚：清空所有DML语句操作历史记录**

1. 事务并发产生的问题：<https://www.cnblogs.com/huanongying/p/7021555.html>
2. 脏读：事务a读取了事务b更新的数据，而事务b回滚操作，那么事务a读取的是脏数据。**脏读就是指当事务A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B也访问这个数据，然后使用了这个数据。**
3. 不可重复读：事务a多次读取同一数据（注意：多次读取发生在一个事务内），而事务b在此过程中对数据进行了更改，导致事务a读取的数据不一致。

不可重复读的缺陷在于：事务读取到的数据随时可能变化，比如进行扣款100元时，查到账户余额100元，在将数据修改为0元之前，数据可能被其他事务更改，余额不再是100元了，那么就不应该改为0元。在查询到余额为100元到修改为0元之间，余额可能被修改，不再是100元，进行扣款修改成余额0元就是错误的操作。

1. 幻读：事务a对所有数据进行了修改，而事务b此时插入了一条未修改的数据，然后事务a发现还有数据未修改，好像出现幻觉一样。（比如将学生成绩修改为等级制，而新插入的数据成绩是分数制）
2. 事务或者叫数据库的**隔离级别（针对以上问题）**



默认设置是可重复读。（第二条应该翻译成 读已提交）

实现隔离级别的两种锁模式：共享锁和排他锁都属于悲观锁。

**共享锁：**读锁，允许大家一块读，但只能拿到锁的事务一个人写

**排他锁：**只能拿到锁的一个人处理数据，其他人不能读不能写。

解释：<https://blog.csdn.net/bamboo_cqh/article/details/81171625>

1. 读未提交：允许一个事务可以看到未提交的数据。

**实现**：不是完全没加锁，**读取数据时不加锁，数据更新时加行级共享锁，其他事务不可更改，但可以读取**，就可能导致脏读。事务结束时释放锁。

1. 读已提交：事务修改的数据提交后才能被另一个事务读取。（读取的数据可能被其他事务修改，导致不可重复读（见上文））

**实现：给读的数据加行级共享锁，读取完释放共享锁，给写的数据加行级排他锁，事务结束才释放排他锁。也就是在读数据时不允许数据被更改，在写数据时，直到事务提交前，都不允许数据被读取或修改，从而避免了脏读。**

1. 可重复读：除了保证事务无法读取未提交的数据外，还保证避免不可重复读。（mysql默认隔离级别）（该级别会锁定事务引用的所有行，这就保证了行记录在该事务结束前不会被其他事务修改。但由于只是锁定所引用的行，没有锁定表，所以对于新插入的数据导致的幻读是没有办法的。）

**实现：给读的数据加行级共享锁，给写的数据加行级排他锁，均为事务结束后释放。在事务结束之前，所读取的数据不会被修改，从而避免了不可重复读。**

1. 串行化：代价最高最可靠的事务隔离级别。事务被处理为顺序执行。（会锁定整表，从而解决幻读）

**实现：事务读数据加表级共享锁，写数据加表级排他锁。事务结束后释放。**

1. MVCC 多版本并发控制

提高数据库的并发性能，处理读写操作的冲突。

基本特征：

* 每行数据都存在一个版本，每次数据更新都更新该版本
* 该版本修改时copy出当前版本随意修改，事务之间无干扰
* 保存时比较版本号，，如果成功（commit）则覆盖原纪录否则回滚

InnoDB引擎MVCC的实现策略：

在每一行数据中保存两个隐藏的列，一个是当前行创建时的的版本号和和删除时的版本号，事务都有唯一的id，是事务开始时申请的且递增，事务对数据修改时会将事务id赋值给数据版本。

MVCC下的InnoDB增删改查

1. 插入记录操作，记录该行记录的创建版本为当前事务id，，删除版本为空
2. 更新，先标记旧的行记录为已删除，删除版本号为当前事务id，再创建新的记录，新记录的创建版本也是当前事务id
3. 删除操作，把事务id作为删除版本号
4. 查询，符合以下两个条件才能阿才能被查询出来
5. 删除版本号未指定或大于当前事务id，即查询事务开启时要读的行记录行记录还没有被删除
6. 创建版本号小于或等于当前事务id

优点：

1. 并发读写时，写操作不用阻塞读操作，读操作也不用阻塞写操作，提高了数据库并发读写的性能。
2. 解决脏读、幻读、不可重复读等问题，但不能解决更新丢失的问题。
3. 数据库索引

B-Tree是最常用的用于索引的数据结构，有序的，增删改查都是logm n

哈希索引无序，比如’<40的’，无法查询

数据库索引是创建在表的某列上的，并且存储了这一列的所有值，存储着指向表的指针

在经常需要搜索的列上创建索引

修改性能和检索性能是互相矛盾的，减少索引时，会提高修改性能，降低检索性能

三种索引：唯一索引 主键索引 普通索引

唯一索引：不允许其中任何两行具有相同索引值的索引，允许有空值

**主键索引：数据表经常有一列或列组合，其值唯一标识表的每一行，该列称为表的主键，**主键索引是唯一索引的特定类型，该索引要求主键中的每个值都唯一，不允许有空值。

聚集索引与非聚集索引：

聚焦索引：InnoDB中存在两种索引，聚集索引：如果定义了主键，就默认使用主键做索引，否则选择第一列非空uniqueu列做索引，否则建立隐藏字段做索引，总之聚集索引必须要有。聚集索引只有一个，它用于把数据库中无序的数据变成B+树结构的数据，因此称为聚集索引。

表中行的物理顺序与键值的逻辑（索引）顺序相同，一个表只能包含一个聚焦索引，如果某索引不是聚焦索引，则表中行的物理顺序与键值的逻辑顺序不匹配。

1. mysql查询不使用索引的情况：
2. 估计**使用索引比全表扫描更慢**的情况
3. 如果使用MEMORY/HEAP表，并且where条件中不使用“=”进行索引列，那么不会用到索引，head表只有在“=”的条件下才会使用索引
4. **用or分隔开的条件**，如果or条件中的一个列有索引，其他的列没有索引，那么涉及到的索引都不会被用到，例如：select \* from table\_name where key1='a' or key2='b';如果在key1上有索引而在key2上没有索引，则该查询也不会走索引
5. **复合索引，如果索引列不是复合索引的第一部分，**则不使用索引（即不符合最左前缀），例如，复合索引为(key1,key2),则查询select \* from table\_name where key2='b';将不会使用索引。
6. 如**果like是以‘%’开始的**，则该列上的索引不会被使用。例如select \* from table\_name where key1 like '%a'；该查询即使key1上存在索引，也不会被使用。
7. **如果字段为字符串，则where条件中必须将字符常量值加引号，**否则即使该列上存在索引，也不会被使用。例如,select \* from table\_name where key1=1;如果key1列保存的是字符串，即使key1上有索引，也不会被使用。
8. mysql会对sql语句做优化， in 后面的条件不超过一定数量仍然会使用索引。mysql 会根据索引长度和in后面条件数量判断是否使用索引。另外，如果是in后面是子查询，则不会使用索引
9. 数据库索引底层实现大部分是B树索引（B+树是B树的一种），原因：B树相比于AVL（自平衡二叉搜索树），RBT（红黑树），高度更低（B树是m叉树，搜索复杂度为log m（n），而且插入删除操作不容易改变树结构，不会频繁进行平衡维护），与hash相比，hash存在hash冲突，且利用率有限（需要扩容）。而且，m叉树保证插入时很少改变树高，减少平衡次数，适用于数据库中大量查询插入操作。
10. AVL树中每个节点都需要额外空间保存平衡值。自平衡二叉树
11. mySql主从同步：当主库（master）发生变化时，实时同步到从库（slave）上。
12. 主从同步原理：

Master机器上主从同步事件写到特殊log文件中（binary-log）

Slave连接master时，master为slave开启binlog dump线程，该线程通知slave binlog的变化并传递binlog。

Slave开启2个线程。I/O线程连接master，接受binlog，并写入本地replay log；SQL线程读取本地replay log，对slave数据库执行相应操作。

1. **悲观锁与乐观锁**

悲观锁：对数据库中某条数据进行修改前，使用数据库提供的锁机制对该数据加锁以防止并发。

此时一般认为**并发修改的概率较大**，对数据的修改抱有悲观态度。

缺点：加锁使数据库产生额外开销，增加产生死锁的风险，降低了并行性（若数据被锁定，则需要等待）。

乐观锁：

乐观锁假设数据一般不会造成冲突，在数据提交更新时才正式对数据的冲突与否进行检测，如果发现冲突，则返回给用户错误信息，让用户决定如何做。

不使用数据库的锁机制，实现乐观锁一般方式是记录数据版本。（也就是行记录有个字段为version，用来记录版本）

此时一般认为数据竞争概率小。

**悲观锁流程**：修改数据前加锁；加锁失败则等待或抛出异常；加锁成功则进行操作。

使用悲观锁要关闭mysql中自动提交的属性（set autocommit=0）。

**乐观锁具体实现细节**：冲突检测和数据更新。典型实现方式是Compare and Swap（CAS）技术：多个线程同时更新同一个变量时，仅一个成功，其他失败，失败的线程不会挂起，而是被告知失败，并可以再次尝试。

存在的问题：高并发环境下会产生大量失败，所以要根据情况减小乐观锁的粒度。粒度把控是一门学问。

Vs

1. 乐观锁并未真正加锁**，效率高，一旦锁的粒度掌握不好，更新失败概率就高**，易发生业务失败。
2. 悲观锁依赖数据库锁，**效率低，更新失败的概率低**。

悲观锁使用的越来越少，尤其是并发量大的场景。

1. NoSQL数据库：MongoDB，Redis。
2. 二者适用哪些场景？
3. Redis和memcache有什么区别？redis为什么更有优势？
4. Redis的数据结构
5. Redis的list zset底层实现
6. Solr和mongodb的区别，存数据为什么不用solr？
7. 分布式session的共享方案有哪些？有什么优劣势？
8. MongoDB应用场景：解决海量数据的访问效率。
9. 数据库优化查询方法：外键，索引，联合查询，选择特定字段等。
10. 数据表student中有name, id, score, city字段，其中name可能有重复，需要消除重复行，写sql语句： select distinct name from student
11. 5条常用语句

show databases

show tables

desc 表名：显示表的字段详细信息

select \* from 表名

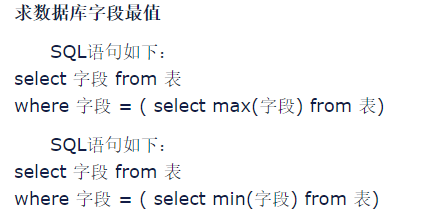
delete from 表名 where id=5

update students set gender=0,hometown=’北京’ where id=5

select \* from 表名 order by id

insert into table\_name(field1,field2) values(value1,value2)

**选取字段最小值：**



**排序：order by age desc；order by age asc；**

**Limit m,n；m为偏移量，默认为0，n为取从偏移量开始的n条数据。**

**Limit m，n：返回第m行开始的n行数据；limit n；返回前n行数据。**

**所以取字段最值也可写成：select \* from 表名 order by age desc limit 1.**

**查询字段为空的数据** select \* from ABCD where XYZ='' 或者 XYZ is null

**创建索引：create [**UNIQUE|CLUSTERED**] index 索引名 on 表名(列名)**

**创建联合索引：create [索引类型] index 索引名 on 表名(列名1，列名2)**

**从table1中选择所有grade和name字段重复的数据id：**

**Select id from tablename group by (grade,name) having count(\*)>1**

1. **数据库分页实现及性能优化：**

**详细参考：**<https://blog.csdn.net/qq_36276335/article/details/73824243>

1. limit命令可以实现数据库分页，但是limit查询偏移量越大，查询速度越慢；
2. 优化方法：见链接文章。
3. 范围查询并不是一定不使用索引，这要看查询的数据量级，如果数据量级大，使用索引比全表扫描更慢，就不会使用索引。
4. 简述mysql与redis的区别：

Redis:内存型非关系型数据库，数据保存在内存中，速度快

Mysql：关系型数据库，数据保存在磁盘中，检索有一定的io操作，访问速度相对较慢。

1. 场景**mysql数据存储引擎：**
2. InnoDB：支持**事务处理，外键，崩溃修复和并发控制**。如果对事物的完整性要求高（如银行），实现并发控制（如售票），则选择InnoDB。如需要频繁更新、删除操作，也可以选择InnoDB，因为支持事务的提交（commit）和回滚（rollback）
3. MyISAM：插入数据快，空间和内存使用低。用于需要频繁插入和读出记录，或对应用的完整性，并发性要求低时。
4. MEMORY：所有数据都在内存中**，处理速度快但安全性不高**。用于需要很快的读写速度但对安全性要求不高的情况。
5. **数据库引擎：**

数据库引擎是数据库底层软件组织，数据库管理系统使用数据库引擎来创建修改查询删除数据。不同的存储引擎提供不同的存储机制、索引技巧、锁定水平等功能，不同的存储引擎还可以获得特定的功能。

存储引擎也可以称为表类型，即存储和操作此表的类型。

如创建一个InnoDB类型的表：

1. CREATE TABLE `brand` (
2. `id` int(11) NOT NULL AUTO\_INCREMENT,
3. `brand\_name` varchar(64) NOT NULL,
4. `brand\_logo` varchar(255) NOT NULL,
5. `description` varchar(255) NOT NULL,
6. `parent\_id` int(11) NOT NULL,
7. `layer` int(11) NOT NULL DEFAULT '0',
8. `enabled` int(11) NOT NULL DEFAULT '0',
9. `path` varchar(255) DEFAULT NULL,
10. PRIMARY KEY (`id`),
11. UNIQUE KEY `brand\_name` (`brand\_name`)
12. ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

**InnoDB**

支持事务，实现了四种隔离级别；

提供行级锁，写操作不会锁定全表（但不是绝对的，如果MySQL不确定扫描范围，同样会锁全表），适合处理多重并发的更新请求

支持外键；

可以从灾难中恢复（利用bin-log日志等）

不支持全文类型的索引，没有保存表的行数，所以执行count(\*) 时会扫面全表。

**总结：**事务，外键，行级锁，灾难恢复，不保存总行数，不支持全文索引

**MyISAM**

不支持事务，外键，行级锁（需要锁整表），表损坏后无法恢复；（该引擎不支持事务，但可以在应用层根据自己的业务需求进行相应的控制）

查询速度快，存储了表的行数，适合读操作远多于写操作且不需要数据库事务支持的场景。

**总结**：不支持事务，外键，行级锁，灾难恢复，支持全文索引，保存总行数，（查询快，适合读多于写且不需要事务支持的场景）

**MEMORY**

存储于内存中，访问非常快，默认使用hash索引，但也可以使用B树索引。

适用于访问频繁，变化不频繁，数据量较小（存于内存，不能太多）且突然丢失也没有太大关系的数据。

**总结**：存储于内存，速度快，使用hash索引（适用于访问频繁且变化不大，数据量小的场景）

**InnoDB与MyISAM的区别：**

1. 前者支持事务，后者不支持，这是InnoDB为默认引擎的原因
2. 前者支持外键，后者不支持，无法将一个有外键约束的InnoDB表转为MYISAM表
3. MyIASM引擎的索引是B+树，叶子节点存储的是数据地址而不是数据，因此是非聚集索引。

InnoDB引擎的索引是B+树，有聚集索引和非聚集索引之分，聚集索引的叶子节点就是行记录，而其他非聚集索引的叶子节点是主键值。

1. 前者不保存表的行数，执行count(\*)会全表扫描，而后者用一个变量保存了行数
2. 前者最小粒度是行锁，后者最小粒度是表锁。
3. 前者缓存索引和数据，对内存要求较高，后者只缓存索引。

问：表有自增id主键，当insert了17条记录后，删除15、16、17，重启MySQL，再插入一条新数据，问该数据id是多少？

答：如果是InnoDB，为15，因为InnoDB表只在内存中记录自增主键的最大id，重启数据库会导致最大id丢失。

 但是，如果我们使用表的类型是MylSAM，那么这条记录的ID就是18。因为MylSAM表会把自增主键的最大ID**记录到数据文件里面**，重启MYSQL后，自增主键的最大ID也不会丢失。

如果在这17条记录里面删除的是中间的几个记录（比如删除的是**10,11,12三条记录），重启MySQL数据库后，insert一条记录后，ID都是18。因为内存或者数据库文件存储都是自增主键最大ID**

问：varchar与char？

答：相同点：char(n) varchar(n)的n都代表字符的个数；超过最大长度n的限制后，字符串会被截断。

不同点：

1. Char一定会占用n个字符空间，varchar只会占用实际字符空间加1或加2个字节（用来保存实际长度数值length，加1 字节说明length<255=2^8-1,加2 表示length>255）
2. Char 的存储上限为255字节。
3. Char在存储时会截断尾部空格，varchar不会。
4. 索引分类

数据结构

1. B+树索引
2. Hash索引
3. 全文索引
4. R-Tree索引

物理存储（仅适用于InnoDB引擎）

1. 聚集索引
2. 非聚集索引

逻辑角度

1. 主键索引：特殊的唯一索引，不允许有null
2. 普通索引或单列索引
3. 联合索引：多个字段共同组成的索引
4. 唯一索引：每个值唯一，可以为null
5. MySQL瓶颈及应对措施

<https://www.cnblogs.com/liugx/p/9935749.html>

当单表数据量达到千万级别以上时，无论如何对MySQL优化，查询如何简单，MySQL性能都会显著降低。

措施：

1. 增加MySQL配置中buffer和Cache的数值，增加服务器CPU数量和内存大小。（硬件和服务器优化是效果最显著，成本最低的）
2. 对数据库进行分区分表，减小单表体积
3. 使用NoSQL等辅助解决方案，如redis mencached
4. 使用中间件做数据拆分和分布式部署，典型案例有阿里开源数据库中间件Cobar
5. 使用数据库连接池技术：

MySQL 的锁机制还不够完善，同时程序中的一些问题都有可能导致 MySQL 数据库连接阻塞，在并发大的情况下，就会浪费很多连接资源和反复连接的消耗。使用数据库连接池，让连接进行排队和复用，一定程度上可以缓解高并发下的连接压力

1. 迁移到其他数据库。

Oracle 的性能瓶颈在 2 亿数据量的级别，远远优于 MySQL。 不仅如此，在关联查询和内置函数等功能上，Oracle 都是完胜 MySQL 数据库的。

1. **数据库优化** <https://www.cnblogs.com/liliuguang/p/11015964.html>
2. 选择**合适的字段属性**；

数据表越小查询越快，应将表中字段设置的尽可能小，比如邮政编码设为char（6）即可

1. 使用**join**代替子查询；

Sql从4.1支持子查询，子查询的结果可以加入主查询的条件中。效率高的原因是不需要创建临时表存储子查询的结果。

Join用法<https://blog.csdn.net/qq1332479771/article/details/62104624>

1. 使用**union**代替手动创建的临时表；

可以把使用临时表的多个select查询合并到一个查询中，查询会话结束时，临时表自动删除；使用方法：union连接多个select语句，这些语句中select的字段数要相同

1. 使用**事务**保持数据的一致性和完整性（也就是保证多条语句同时成功或失败）。

BEGIN；DML语句；COMMIT；

事务还可以利用锁机制为用户提供安全的访问方式。

1. 锁定表

LOCK TABLE表名 WRITE SELECT（select语句）。。。。；write和select之间无分号

UNCLOCKTABLES

1. 使用**外键**保证数据的关联性；

比如删除班级时，里面的学生也会自动删除；没有合法班级id的学生不会被插入。

1. 使用**索引**；

在取最值或orderby命令中效率提升明显。

语句：CREATE [UNIQUE|CLUSTERED|NONCLUSTERED] INDEX 索引名 ON TABLE\_NAME(列名) ；  
其中UNIQUE和CLUSTERED为[可选项](http://www.so.com/s?q=%E5%8F%AF%E9%80%89%E9%A1%B9&ie=utf-8&src=internal_wenda_recommend_textn)，分别是建立[唯一索引](http://www.so.com/s?q=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&ie=utf-8&src=internal_wenda_recommend_textn)和[聚簇索引](http://www.so.com/s?q=%E8%81%9A%E7%B0%87%E7%B4%A2%E5%BC%95&ie=utf-8&src=internal_wenda_recommend_textn)

创建索引应选择 选择性高的 字段，也就是 重复少的字段。

DROP INDEX 索引名 ON 表名

1. 优化查询语句

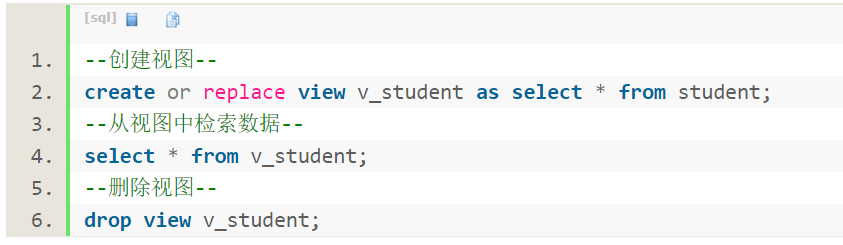
最好在相同类型字段间进行比较；

在建有索引的字段上尽量不要使用函数；

Like查询会比较每一条记录，最好改为 大于小于；比如like “mysql%” 改为 >=”mysql” and <”mysqm”

1. **数据库视图** [**https://blog.csdn.net/u011649691/article/details/78998907**](https://blog.csdn.net/u011649691/article/details/78998907)

**视图是从一个或几个基本表（或视图）中导出的虚拟的表，可以将视图看成一个窗口，通过他可以看到感兴趣的数据。创建视图并不会复制数据，而是共享基本表的数据。对视图的操作同其他表一样，通过视图修改数据实际是在改变基表的数据，相反的，基表数据的改变也会反映在视图中 。（视图都支持查询，但只有单表视图用于修改，会改变基本表的数据，多表视图的修改，不会改变基表的数据）**



**一段模拟面试：**

1. 数据库有千万数据，如何快速查询？答：加索引
2. 索引有哪些数据结构？答：hash B+
3. 为什么哈希表，完全平衡二叉树、B树、B+树都可以优化查询，为何mysql用B+树？

答：哈希表可以快速的精确查询，但无法范围查询（可以直接找到id=1的记录，但无法找到id>x的记录）

哈希表适合哪些场景？答：等值查询的场景，就只有key value的情况，如Redis、Memcached这些NoSQL的中间件。

你说的是无序的哈希表，有没有有序的数据结构？

答：有序数组，在等值查询和范围查询都有优势。

那它的缺点呢？

答：有序的适合存储静态数据，因为如果我们插入、删除、修改数据的时候就会以很大的复杂度改变它的结构。

它的用处呢？

答：可以用来做静态存储引擎，保存静态数据，例如去年的账单或购物记录，都不会发生变化的数据。

二叉树呢？（回答中的二叉树默认是二叉搜索树，否则没有意义）

答：二叉树是有序的，所以支持范围查询，但是时间复杂度是logn，为了维持这个复杂度，它更新的时间复杂度也是logn，那就得保证这个树是完全平衡二叉树了。

平衡二叉树做索引还不错？

答：不，索引不只是在内存里存储，还有落盘持久化的。树高会随数据增加，查询成本会随树高增加。为了节约成本很多公司采用机械硬盘，千万级别的查询差不多要10秒了。

如果用B树呢？

答：B树（基本平衡的多路搜索树）每个节点有M个子节点，复杂度为logmn，比二叉树矮。

**B树补充：**

**1） 任意非叶子节点最多有M个儿子（M>2）**

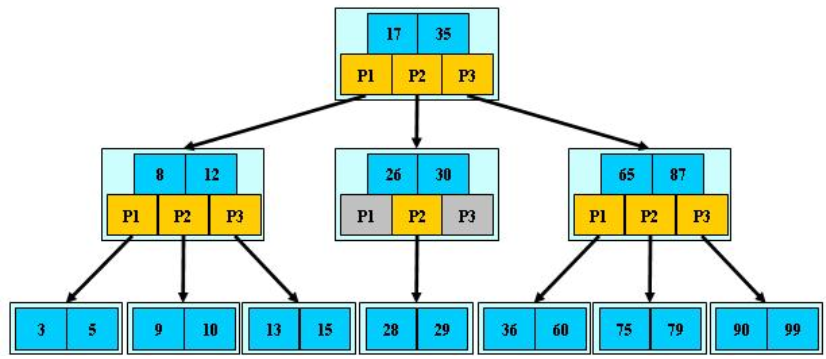
**2) 根节点儿子数为[2,M]**

**3) 除根节点以外非叶子节点儿子数为[M/2,M]**

**4) 非叶子节点存放的关键字个数=儿子数-1**

**5）所有叶子节点位于同一层。**

**例如M=3：**



**基于以上定义，有如下特性：**

1. **关键字分布在整棵树中；2.任意关键字只出现在一个节点中；3.搜索可能在非叶子节点结束。**

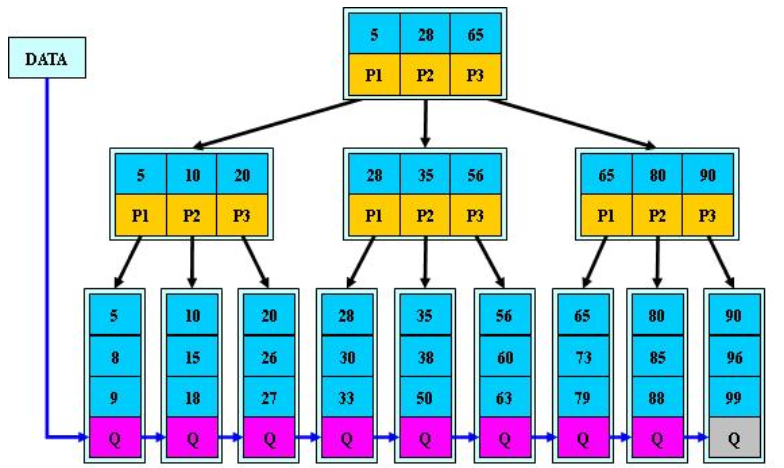
**B+树补充：**

**1） 与B树基本相同，除了**

**2) 关键字向子节点冗余，因此非叶子节点的儿子与关键字个数相同，最终所有关键字都在叶子节点中**

**3) 所有叶子节点增加一个链指针**

**例如M=3**



**基于以上定义，有如下特性：**

1. **所有关键字都出现在叶子节点的链表中，且链表中的关键字有序；2. 不可能在非叶子节点命中；（.非叶子节点相当于叶子节点的索引，叶子节点相当于存储数据的数据层；适合文件索引系统）**

B+树有什么优势？

答：hash不支持范围查询，B树查找效率logmn优于二叉树，且树高更低，且增删操作很少改变树高，平衡操作更少。只有B树跟B+有的一比。

B+树是B树的升级版，只是把非叶子节点向子节点冗余一下，所有关键字都存在与叶子节点中，且叶子节点有链指针连接。这么做是为了**提高范围查找的效率**。

1. B+树的磁盘读写代价更低。B+树的**内部结点并没有指向关键字具体信息的指针**，其内部结点比B树小，盘块能容纳的结点中关键字数量更多，一次性读入内存中可以查找的关键字也就越多，相对的，IO读写次数也就降低了。
2. B树**元素遍历的效率**低下。B+树**只要遍历叶子节点就可以实现整棵树的遍历**。而且**在数据库中基于范围的查询是非常频繁的**，而B树效率很低。

所以，MySQL使用B+树做索引，可以提高查询索引时的磁盘IO效率，并且可以提高范围查询的效率，并且B+树里的元素也是有序的。

B+树中一个节点存多少元素合适？一个节点到底多大合适？

答：为一页或页的倍数最合适。

为什么？

答：因为如果一个节点的大小小于1页，那么读取这个节点的时候也会读出一页，造成资源浪费。如果大于1页，那么读取这个节点的时候会读出2页，也会造成资源浪费。

为了避免浪费，最好把一个节点的大小控制在等倍数页。

提到了页的概念，能简单说一下吗？

答：MySQL的基本结构是页，（记录都存在页里边）。**各个数据页**可以组成一个**双向链表，**数据页中的记录可以组成一个**单向链表**。

每个数据页会为其中的记录生成一个**页目录**。

通过**主键**查找某记录时，先在页目录中使用**二分法**快速定位到对应的槽，再遍历该槽对应分组，得到指定记录。

通过**普通索引字段**查找时，先通过普通索引找到主键,再通过主键找行记录（回表）

通过**非主键、非索引**字段搜索时，只能**依次遍历单向链表中的每条记录。**

例如执行select\* from user where username = ‘name’ 这样的没有任何优化的sql语句时，默认这样做：

-遍历双向链表找到所在页；-遍历所在页的单向链表。

很明显，在数据量很大时，查找**很慢。**

问：回表你聊一下。

答：回表大概就是我们有个主键为id的索引，和一个普通字段name的索引，我们在普通字段上搜索：select \* from table where name = ’xxx’，执行的流程是先查询name索引上的‘xxx’，找到它的id值，然后去主键索引，找到id值对应的记录。

回到主键索引树搜索的过程，就是**回表**。不过也有方法避免回表，那就是**覆盖索引**。

**补充 回表：参考**[**https://www.jianshu.com/p/8991cbca3854**](https://www.jianshu.com/p/8991cbca3854)

**InnoDB的索引分两大类：聚集索引（以主键创建的索引）；普通索引（以其他字段创建的索引）。**

**聚集索引树（B+树）的叶子节点存行记录，因此有且只有一个聚集索引。聚集索引名字的来源就是因为该索引将无序的数据整理成了B+树结构。**

**普通索引的叶子节点存放主键值。**

**聚集索引的生成：若定义了PK，则PK就是聚集索引；没有PK则第一个非空unique列为聚集索引；否则创建隐藏字段作聚集索引。（所以pk查询很快，直接定位行记录）**

**而普通索引的索引树的叶子节点存储主键值。**

**因此，根据普通字段的索引去查行记录时，要先从普通索引找到对应pk值，再从聚集索引找行记录，这就是回表。**

问：哦？那你再说一下**覆盖索引**？

答：覆盖索引（或者索引覆盖）是指所需的字段从索引中就能获取，不必读取行记录，不用回表，速度更快。（也就是说查询列被索引覆盖）。

覆盖索引可以减少树的搜索次数，提升性能，也是实际开发过程中经常用来优化查询效率的手段。很多联合索引的建立，都是为了支持覆盖索引。

刚才我们是select \*查询所有，我们如果只查询id，其实在name字段的索引上就已经有了，那就不需要回表了。比如 select id, name from table where name = ‘xxx’。（由于只查id和name，而id和name通过name字段的普通索引树的叶子节点就能得到，无需回表）

问：索引的**最左匹配原则**知道么？（又叫最左前缀原则）， 参考：<https://www.cnblogs.com/lanqi/p/10282279.html>

我的解释：最左匹配是针对联合索引来说的。若干个字段可以组成联合索引，在联合索引中，对于任意字段来说，只有当其左边的字段确定的情况下，该字段才是有序的，才可以使用索引。因此查找数据时会在联合索引的字段中从左到右依次匹配。

举个例子，有联合索引（abcd），查找条件a=1,b=2,c>3,d=4,查找时会依次命中ab，到c时由于是范围查询，c的值不确定，后续就退化为线性查找， d无法使用索引。

答：索引可以简单如一个列，也可以复杂如多个列，即**联合索引。**如果是联合索引，那么key也由多个列组成。同时，索引只能查找key是否存在（相等），遇到范围查询（>、 <、 between、 like左匹配等）就不能进一步匹配了，后续退化为线性查找。因此，**列的排列顺序决定了可命中索引的列数**。

例如：有索引（a, b, c, d），查询条件a=1 and b=2 and c>3 and d=4,则会在每个节点依次命中abc，无法命中d（c已经是范围查找了，d肯定是排不了序了）（因此要尽量选择区分度高的列做索引）

问：为什么B+索引会降低新增、修改、删除的速度？

答：B+树是一棵平衡树，进行新增、修改、删除会破坏他的原有结构，所以要花额外的时间去维护索引维持平衡。这些操作导致索引会降低新增、修改、删除的速度。

问：**为什么MongoDB用B树而MySQL用B+树？**

答：参考：<https://zhuanlan.zhihu.com/p/107228878>

**B树：每个节点都存储数据；叶子节点无指针相连；B+树：数据存储在叶子节点，用链指针相连。**

**结合以上特点：B树单一数据查询效率不稳定，平均效率低；但不适合遍历操作。B+树单一数据查询效率稳定；适合做遍历查询。**

MongoDB是非关系型数据库，常进行单一数据查询，而MySQL是关系型数据库常进行遍历操作。

例如：数据库中有班级和学生两个逻辑实体，是一对多的关系，关系数据库用两张表存储，若选择指定班级的所有学生，则需要：在班级表中根据班级名称找到班级id，（若班级名称这个字段建立了索引就找班级id就方便了，否则要遍历），在学生表中**遍历**查找符合班级id的学生记录。（为什么这么多遍历操作？因为给出的条件字段没有建立索引）

在非关系型数据库中，会把学生放在所属班级中，这样只需要根据班级名称找到对应的班级记录，其中自然包含所有学生。（因此不用遍历所有学生，另外如果班级名称建立了索引，就不用遍历班级了）

ORM（Object-Rational Mapping）:关系型数据库与对象之间的映射，操作数据库时不需SQL语句，只需像对象一样操作即可。

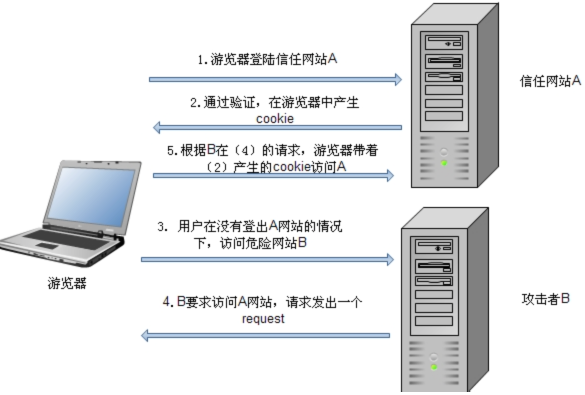
**SQL注入攻击：**SQL注入即是指web应用程序**对用户输入数据的合法性没有判断或过滤不严，**攻击者可以在web应用程序中**事先定义好的查询语句的结尾上添加额外的SQL语句，**在管理员不知情的情况下实现非法操作，以此来实现欺骗数据库服务器执行非授权的任意查询，从而进一步得到相应的数据信息。

因为服务器会使用参数去拼接sql语句，如果参数后面有一个分号和一个完整sql语句，注入语句就会被执行。或者是输入 or 1=1 ,这样条件就一定为真，就可以避开密码等条件判断。

**CSRF（跨站请求伪造）攻击**[**https://blog.csdn.net/lamp\_yang\_3533/article/details/79959071**](https://blog.csdn.net/lamp_yang_3533/article/details/79959071)

**：**攻击者通过跨站请求，以合法的用户身份进行非法操作（如转账或发帖等）。CSRF的原理是利用浏览器的Cookie或服务器的Session，盗取用户身份。其原理如下图所示。

XSS攻击？



## Redis面试题

参考：<https://zhuanlan.zhihu.com/p/134104400>

1. Redis是什么

定义和特点：是C语言开发的开源的（遵从BSD协议）高性能键值对的内存型非关系型数据库，可以用作数据库，缓存，消息中间件等。

特点：

* 性能优秀，数据在内存中，读写速度非常快
* 线程安全，单进程单线程，采用IO多路复用机制
* 丰富的数据类型，支持字符串，散列，列表，集合，有序集合等
* 支持数据持久化
* 主从复制，哨兵，高可用。
* 可以用作分布式锁
* 可以作为消息中间件使用，支持发布订阅。

1. 支持的五种数据类型？
   1. String类型：是最基本的数据类型，一个key对应一个value。Value可以是String或数字。（String是二进制安全的，意思是String类型可以包含任何数据，包括图片或序列化对象，一个String对象最大存储512M。
   2. Hash是一个键值的集合，相当于字典。Redis的Hash是一个String的key和value的映射表，Hash特别适合存储对象。常用命令：hget hset hgetall
   3. List列表是简单的字符串列表，按插入顺序排序。可以从首尾添加或移出元素。常用命令：lpush rpush lpop rpop lrange(获取列表片段)

数据结构：List就是链表，可以当消息队列使用，提供了push和pop操作，还可以直接查询或删除列表的某一段

实现方式：双向链表

* 1. Set：String类型的无序集合。通过hashtable实现的。常用命令：sadd，spop，smembers，sunion
  2. Zset：String类型的有序无重复集合，常用命令：zadd，zrange，zrem，zcard

插入有序，即它是自动排序的，并且为用户提供了一个优先级（score）参数为成员排序。若需要一个有序且不重复的集合列表，可以使用Zset.

Score是一个double类型的权重参数，使集合元素能按score有序排列。Redis对成员按score进行从小到大的排序。

实现方式<https://www.cnblogs.com/yuanfang0903/p/12165394.html>

：使用HashMap和跳跃表（skipList）来保证数据的存储和有序，HashMap存放的是成员到score的映射。跳跃表存放所有成员，排序依据是HashMap 里的score

1. 实际项目中会遇到什么问题？

缓存和数据库数据的一致性问题：分布式环境下容易出现缓存和数据库数据不一致，如果项目对缓存的要求是强一致性的，就不要使用缓存。

只能采取合适的策略降低缓存和数据库的数据不一致的概率，但无法保证强一致性。

合适的策略包括合适的缓存更新策略：更新数据库后及时更新缓存，缓存失败时增加重试机制。

1. Redis雪崩
2. 电商首页及热点数据都会做缓存，一般缓存都是定时任务去刷新，或者查不到之后去更新缓存，定时任务刷新就有一个问题。
3. 举例：如果首页所有key的失效时间都是12小时，中午12点刷新（0点失效），而0点大量用户涌入，由于缓存失效，所以大量的请求都落在数据库上，如果没有特别方案处理，重启数据库又会被新的流量打死。也就是说，**redis同一时间大面积缓存失效，redis就等于没有，大量的请求落到数据库上，数据库就会扛不住。**（如果挂（崩溃）的是一个用户服务的库，那依赖该库的其他库的所有接口几乎都会报错，如果没做熔断等策略，基本上就是瞬间挂一片的节奏，无论怎么重启，用户都会把你打挂）
4. 处理？

批量往redis里存数据时，**每个key的失效时间随机设置**，就可以保证数据不会大面积同时失效。

或者**设置热点数据永不过期，有更新操作就更新缓存**（比如运维更新了首页商品，那你就刷下缓存就好了，不要设置过期时间）

如果redis是集群部署，将热点数据均匀分布在不同的redis库中也能避免全部失效。

1. 缓存击穿

跟缓存雪崩有点类似，但缓存雪崩是大面积缓存失效，打崩了DB。而缓存击穿是指**一个key非常热点，不停地扛着大量请求，大并发集中对这个点进行访问，当这个key失效的时候，持续的大并发直接落到数据库上，就在这个key上击穿了缓存**。

解决：**设置热点数据永不过期，或者去数据库查询时加上互斥锁，或者使用分级缓存，（一级缓存过期时间短，线程发现一级缓存过期后加锁，去数据库读取，读取后更新一级二级缓存，其他线程从二级缓存中读取）**。<https://blog.csdn.net/sanyaoxu_2/article/details/79472465>

1. 缓存穿透

**指的是缓存和数据库都没有的数据，而用户（黑客）不断发起请求。**举例：我们数据库的id都是从1递增的，如果请求id=-1或不存在的数据，不断的攻击会导致数据库压力很大，严重的会击垮数据库。

解决**：在接口层增加校验，比如用户鉴权，参数做校验，不合法的直接return。**另一个方法是redis里有一个高级用法布隆过滤器，利用高效的数据结构和算法判断你这个key是否在数据库中，不存在就return，存在就去查DB刷新KV再return。

最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被 这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。另外也有一个更为简单粗暴的方法（我们采用的就是这种），如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

散列表（又叫哈希表，Hash table）的数据结构可以通过一个Hash函数将一个元素映射成一个位阵列（Bit Array）中的一个点。这样一来，我们只要看看这个点是不是 1 就知道可以集合中有没有它了。这就是布隆过滤器的基本思想。

算法：  
1. 建立k个hash表，需要k个hash函数，每个函数可以把key散列成为1个整数  
2. 初始化时，需要一个长度为n比特的数组，每个比特位初始化为0  
3. 某个key加入集合时，用k个hash函数计算出k个散列值，并把数组中对应的比特位置为1  
4. 判断某个key是否在集合时，用k个hash函数计算出k个散列值，并查询数组中对应的比特位，如果所有的比特位都是1，认为在集合中。

优点：不需要存储key，节省空间

优点：时间：插入查询都是O1复杂度，空间：不需要存储关键字本身，在某些保密性要求高的场合有优势。

缺点：随着元素增加，误算率也会增加。不支持删除操作。

1. Redis为何这么快？

Redis是基于内存的，cpu不是它的瓶颈，它的瓶颈在内存大小和网络带宽。

* 1. Redis完全**基于内存**，绝大部分请求是纯粹的内存操作，非常迅速。
  2. **数据结构简单**，对数据的操作也简单
  3. 采用**单线程，避免了不必要的上下文切换和竞争条件**，不存在多线程导致的CPU切换和锁问题，**不存在加锁释放锁操作**，没有死锁导致的性能消耗。
  4. 使用**多路复用IO模型，非阻塞IO**。

1. 六种淘汰策略
   1. 从**已设置过期时间**的KV中优先淘汰**最近最少使用的数据**（Less Recently Used）（volatile-lru）
   2. 从**已设置过期时间**的KV中优先淘汰**剩余时间短的数据**（Time To Live）(volatile-ttl)
   3. 从**已设置过期时间**的KV中**随机选择**数据淘汰 (volatile-random)
   4. 从**所有**KV中优先淘汰**最近最少使用的数据**（LRU） (allkeys-lru)
   5. 从**所有**KV中**随机淘汰** (allkeys-random)
   6. **不淘汰策略**，若超过最大内存则返回错误信息 (noeviction)
   7. Redis4.0加入了LFU(least frequency use) 最少访问频率策略，包括volatile-lru和allkeys-lfu。

## Python基础

1. Python为什么是解释性的语言？

没有编译过程（编译成二进制代码），没有静态检查，语句是通过解释器一条一条执行的，可以交互运行。

1. 面向对象和面向过程的区别？？
2. 面向对象是把构成问题的事务分解成各个对象，将某个事务在解决问题的步骤中的行为定义为对象的方法，不同的事务对象使用各自的方法来解决问题；
3. 面向过程是分析出解决问题所需要的步骤，将其封装为函数，使用的时候按顺序调用。
4. Python的优势劣势：

解释型，语法简单易懂，可读性强

可调用很多库，可以让我们站在巨人的肩膀上简单的实现想要的功能

可扩展，和其他编程语言或其他软件有可连接的接口

免费开源、可移植

自动内存管理，让程序员可以专注于代码的实现

1. 缺点：

可解释特征使其运行速度变慢

动态语言的特点可能会增加运行时错误。

1. 1行代码实现1-100求和：sum( range(1,101) )
2. 5个python标准库：os（提供与操作系统相关联的函数）；sys(通常用于命令行参数)；re(正则匹配)；math(数学运算)；datetime(处理日期时间)
3. 字典如何删除键，如何合并两个字典：

删除键：del dict[‘name’] 合并：dict1.update(dict2) （如果dict1和dict2中有相同的key，则dict1的key会被覆盖，因为被update了）

1. **Python序列化库：pickle cpickle**

<https://blog.csdn.net/xufeng0991/article/details/40105893>

class Bird:

name = ‘a’

id = 1

objectname = Bird()

pickledString = pickle.dump(objectname)

objectname = pickle.load(pickledString)

**cpickle这个库是c语言写的效率更高，与pickle用法一样。**

1. **Python表示其他进制数：**

**Jin\_2 = 0b10**

**Jin\_8 = 0o10**

**Jin\_16=0x10**

1. **read，readline 和readlines：**
2. 读取整个文件
3. 读取下一行，使用**生成器**方法 读取到的依次是 ‘a\n’ ,’\n’, ‘b\n’
4. 读取整个文件到一个**迭代器**里，以供遍历。[‘a\n’, ’\n’, ’b\n’]

**需要注意的是，print（'\n’），会出现两个空行，一个空行是字符串‘\n’造成的，后一个空行是print函数自带的。去掉自带的空行可以使用print(‘\n’,end=’’)**

**字符串.strip()函数将会去除字符串前后的空格以及最后的换行符。**

**可以这样来读取文件：**

**with** open***(*'./Java/MyLruCache.java'**,encoding=**'utf8'*)* as** f:  
 a=f.readline***()*****while*(***a***)***:  
 print***(***a.strip***())*** a=f.readline***()***

1. Python的GIL：

是python的全局解释器锁，同一进程中若有多线程运行，一个线程在运行python程序时会霸占python解释器（加了一把锁即GIL），使其他线程无法运行，只能等待该线程运行结束。如果线程运行中遇到操作如io，则解释器锁解开，使其他线程运行。所以在多线程中，线程运行仍然有先后顺序，不是同时运行。

所以对于io密集型任务，python多线程起作用，对于CPU密集型任务，python多线程几乎没有优势。

多进程中每个进程都能被系统分配资源，相当于每个进程都有python解释器，所以多进程可以同时运行，缺点是进程系统资源开销大。

1. 简述多进程，多线程

进程：

1. 操作系统进行资源调度和分配的基本单位，多个进程之间独立
2. 稳定性好，一个进程崩溃不影响其他，但进程消耗资源大，开启的进程数量有限

线程：

1. CPU进行地缘分配和调度的基本单位，是进程的一部分，比进程更小的能独立运行的基本单位，进程下的多线程共享该进程所有资源
2. 若IO操作密集，使用多线程提高效率，缺点是一个线程崩溃则进程崩溃。

应用：

1. IO密集用多线程，在用户输入或sleep时切换其他线程执行，减少等待
2. 多核CPU用多进程。因为在多线程并不是并发运行，运行线程会霸占GIL，使其他线程无法运行。
3. 列表去重：先转集合再转回列表
4. Range 和xrange的区别？python2中，前者返回列表，后者是一个生成器。
5. Python2 和python3 中range(100)的区别：2 返回列表3返回生成器，节约内存。
6. 一句话解释什么语言能使用装饰器：函数可以作为参数传递的语言。
7. Python内建数据类型：整型 布尔型 字符串 列表 元组 字典。
8. **简述\_\_new\_\_ 和\_\_init\_\_的区别：**
9. \_\_init\_\_是初始化方法，创建实例后立刻调用，\_\_new\_\_先于\_\_init\_\_执行，返回一个实例。
10. \_\_new\_\_ 至少要输入一个参数cls，代表当前类。
11. \_\_new\_\_必须要有返回值，返回实例化出来的实例（这点在自己写\_\_new\_\_时要注意），可以 return super().\_\_new\_\_(cls) 来返回实例，即通过父类的\_\_new\_\_产生实例。若无返回值则\_\_init\_\_不会调用。
12. \_\_init\_\_有一个参数self就是\_\_new\_\_返回的实例，\_\_init\_\_ 在\_\_new\_\_的基础上进行其他的初始化操作，不需要返回值。
13. 如果\_\_new\_\_创建的是当前类的实例（保证return语句里调用的\_\_new\_\_函数的第一个参数是cls），则自动调用\_\_init\_\_函数；如果中参数不是cls而是其他类名则返回其他类的实例，\_\_init\_\_和其他类的\_\_init\_\_都不会被调用。

补充解释：先从\_\_new\_\_(cls[,...])的参数说起，\_\_new\_\_方法的第一个参数是这个类，而其余的参数会在调用成功后全部传递给\_\_init\_\_方法初始化，也就是new先于init执行。

1. **Else不仅可以与if连用，还可以与while else， for else，try except else**，Python中的for、while是在循环体内没有break语句、没有return语句，或者没有异常出现时才执行else语句，try..except…else是在try中无异常产生时执行，注意不包括continue。
2. **单例模式：##########**

一种常用软件设计模式，在它的核心结构中包含一个被称为**单例类的特殊类**。**通过单例模式可以保证系统中一个类只有一个实例，且该实例易于外界访问，从而方便控制实例个数并节约系统资源。**如果希望在系统中某个类的对象只能存在一个，单例模式是最好的解决方案。

class Singleton (object):

def \_\_new\_\_(cls, \*args, \*\*kw):

if not hasattr(cls, ‘\_instance’):

orig = super(Singleton, cls)

cls.\_instance = orig.\_\_new\_\_(cls, \*args, \*\*kw)

return cls.\_instance



单例模式另一种写法：



1. Python中的作用域：

一个变量的作用域由被赋值的地方决定。当python遇到一个变量的话会按照以下顺序搜索：

本地作用域-当前作用域被嵌入的本地作用域-全局作用域-内置作用域。

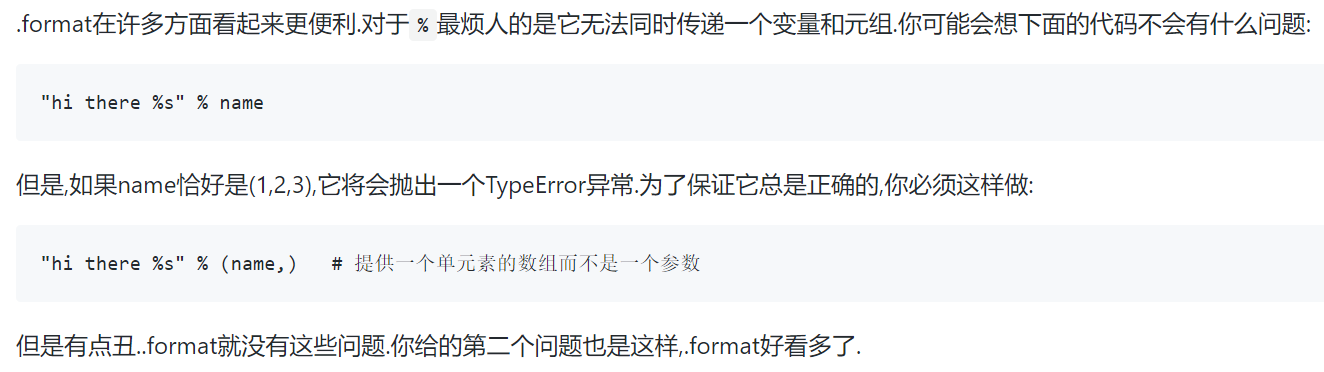
1. 单下划线和双下划线：

\_\_foo\_\_一种约定的python内部的名字，用来区别其他用户自定义的命名，如\_\_init\_\_() \_\_call\_\_()这些特殊方法

\_foo:一种约定，指定变量私有，不能from module import \*导入可以像公有一样访问，但不建议。

\_\_foo:这个有真正的意义，解释器用\_类名\_\_foo来代替这个名字，以区别和其他类相同的名字，无法像公有一样直接访问，只能通过对象名.\_类名\_\_xxx对该方法访问。

1. 字符串格式化 %和.format：



1. With方法打开文件做了什么？

常规方法打开文件需要写try except finally进行异常判断并最后关闭文件，with方法帮我们实现了文件的close。

1. Python生成随机整数、小数，0-1之间随机小数。

random.randint(a,b) 生成区间内的整数；np.random.randn(a)生成a个小数

random.random()（不设置参数） 生成0-1随机小数

1. **re正则匹配**

s=re.compile(r’正则匹配字符串’) 正则匹配字符串的预编译为s

res=s.match(待匹配字符) 匹配则返回匹配对象，否则返回None

re.spilt(r’正则串’，待匹配) 将待匹配字符串按正则串分离，返回一个包含所有分离子串的列表。

提取子串：m=re.match(r’^(\d{3})-(\d{3,8})$’,’010-12345’) 在正则式上加括号

>>>m.group()

‘010-12345’

>>>m.group(1)

‘010’

A=”张明 98分 8分”

Res=re.sub( r”\d+”, “100”, A) #将所有数字替换为“100”

>>>Res

张明 100分 8分

正则匹配中文：###########神奇

Title=”你好， hello， 世界”

Pattern=re.compile(r”[\u4e00-\u9fa5]+”)

Res=pattern.findall(title)

Print res [ “你好”, “世界” ]

1. **url中提取日期**

url=<https://sycm/taobao.com/bda/get_summary.json?dateRange=2020-03-06%7c2020-03-06&dateType=recent&device=1&/>

res = re.findall ( r’dateRange=(.\*?)%7c(.\*?)&’ , url )

#### 用括号提取需要的内容，注意findall这个函数好像不需要从头开始匹配，也不需要匹配到末尾，但re.match一定要从头匹配，否则匹配不成功，match可以不匹配到末尾。

print ( res )

如果不加括号，findall函数会将匹配部分全部提取出来。

1. (.\*) 和(.\*?)的区别：前者贪婪匹配，会把满足正则的尽可能多的向后匹配，后者是尽可能少匹配。默认匹配原则是贪婪匹配，加个？就是取消贪婪匹配。
2. python 断言：

assert（）方法，断言成功则程序继续，否则报错

1. **python2与python3的区别**
2. **print：**python3 的print必须用括号包括输出内容
3. **range**函数在前者返回列表，后者返回生成器
4. **编码**：前者ascii编码，后者utf-8编码
5. **序列表示：**前者unicode表示字符串序列，str表示字节序列。后者str表示字符串序列，byte表字节序列
6. 前者中为正常显示中文，引入coding声明，后者不需要
7. **Input**: 前者是raw\_input()函数 后者是input()函数
8. （不）可变数据类型

不可变：数值型，字符串型和元组：不允许变量的值发生变化，一旦变化就新建了对象；对于相同值的对象，内存中只存在一个（例如 a=3 b=3 那么变量a 和b的id相同）

可变：列表，字典（实例也是这样，但实例不属于数据类型）：允许变量值变化，使用append、+=操作后改变对象的 值但不会创建新对象，变量引用的地址也不会变；（列表切片是新建对象）相同值的不同对象，内存中会存在不同对象（例如：a=[1,2] b=[1,2] a和b的id是不同的）

1. S=’ajijiclahohefiha’,去重并从小到大排序

S=set(s)

S=list(s)

s.sort(reverse=False)

res=””.join(s)

print( res )

1. 对字典根据键从小到大排序：

lis=sorted( dictory.items() , key=lambda i: i[0] ) lis是列表，有多个元组，每个元组是一对键值构成。

补充：上一句可以拆成：lis=list(dictory) lis.sort(key=lambda i:i[0])

res = dict(lis)

1. collections.Counter(string) 统计string中每个字符出现的次数返回一个字典
2. list1.extend( list2 ) 将列表2中的元素全部加到列表1 中，不会去重。不会产生新对象。
3. list1=list1+list2 会生成一个新对象
4. 删除文件：

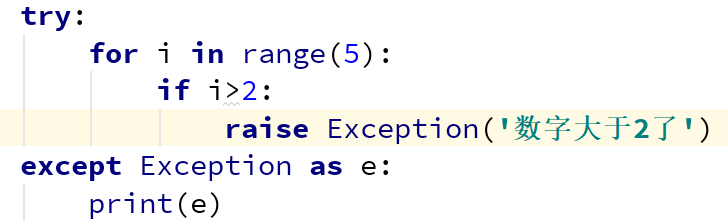
Python: os.remove(文件名) Linux: rm 文件名

1. A=datetime.datetime.now().strftime( “%Y-%m-%d %H:%M:%S ” )

C=str(A)

Print(C) 2020-02-01 12:23:33

1. 自定义抛出异常：



1. 简述Django的ORM

ORM：object-ration-mapping 对象关系映射：

实现了数据模型与数据库的解耦，通过简单配置就可以轻松更换数据库，不需要修改代码，只需要面对对象编程，orm本质上会根据对接的数据库引擎翻译成对应的sql语句。Django项目无需关心使用的数据库是什么，数据库迁移只需更换Django的数据库引擎即可。

1. A=[[1,2],[3,4],[5,6]] b=[ for i in A for j in i ] (b=[1,2,3,4,5,6])
2. a=”abc” b=”def” c=[‘d’, ’e’, ’f’]

m=a.join(b) ;n=a.join(c) (join参数为可迭代对象，将a插入可迭代对象中，返回字符串)

结果m=n=”dabceabcf”

1. zip(a, b, c) (a,b,c为可迭代对象，可以不等长) 返回一个生成器

>>>list( zip( [1,2], [3, 4, 5], “efdg” ) )

[ (1, 3,’e’ ), (2, 4, ‘f’) ]

1. 将字符串string转为bytes类型：string.encode()

如果字符串string仅包含ascii字符（也就是说没有汉字），可以直接b’string’。

1. 列表排序不用sort（）：原列表求min值，删去该值，加入新列表，不断重复。
2. **提高python效率：**
3. 使用生成器，节约内存
4. 循环代码优化，避免重复代码
5. 核心模块用Cython PyPy等提高效率
6. 多进程，多线程，协程
7. 多个if条件判断，把最可能发生的放前。
8. 遇到bug咋整
9. 如果是细节错误， print打印
10. 涉及第三方框架，查官方文档或技术博客
11. 对于bug的管理和归类总结，一般测试将测试出的bug放到teambin等bug管理工具进行记录，逐条修改
12. 导包问题，城市定位多音字造成的显示错误问题。
13. 保留两位小数：b=round( float a, x) b是保留x位小数的a。
14. print(‘%.nf’ %a),将数字a保留n位小数输出。不够n位小数就自动补0.
15. all() any()两个函数：前者：迭代器中所有元素为真才真，后者存在元素为真则真。
16. python中什么元素为假？（0，None，False，空字符串、列表、字典、元组）
17. 删除dict中的name字段：dict.remove(‘ name’)或 del dict [‘name’]
18. **Python中copy.copy(x)和copy.deepcopy(x)的区别：深拷贝和浅拷贝**
19. 复制不可变数据类型：copy和deepcopy和=赋值结果一样，都是同一个对象，同一个id
20. 复制可变对象：deepcopy：完全复制独立
21. copy：若对象中无复杂子对象，则相对于创建新对象（id不同，改变指相互无影响）
22. copy：若**对象中有复杂子对象**（例如，列表中的一个子元素是列表），改变复杂子对象的值相互影响，但改变其他值不会影响。
23. 几种魔法方法：
24. \_\_init\_\_:对象初始化方法
25. \_\_new\_\_:创建对象时执行的方法，单列模式会用到
26. \_\_str\_\_:当print对象时，会打印这个函数的返回值
27. \_\_del\_\_:删除对象执行的方法
28. 命令行启动并传参 python 1.py 2 3，则print（sys.argv）会输出：[ ‘1.py’, ‘2’, ‘3’ ]
29. Sort和sorted函数区别：前者是在原列表上修改，不会生成新列表，后者返回排序后的新列表。
30. [3, 4, -2, 0, -1].sort(key=lambda x: ( x<0, abs(x)) ) =[0,3,4,-1,-2]注意：匿名函数传了两个条件，系统会先按第一个条件排序，然后第一条件无法确定顺序的元素再按第二个条件排序。
31. Python字典与json字符串、转换：dict=json.loads(x) x=json.dumps(dict)
32. 统计字符串string中字符s的个数：num = string.count(s)
33. 字符串转化大小写：str.lower() str.upper()
34. 两种方法去空格：res=str.replace(“ ”, ””) list=str.spilt(“ ”); res=””.join(list)
35. **Python内存管理**
36. **引用计数：每个对象都包含一个头部信息，包括类型标识符和引用计数器。**

当一个变量保存了对象的引用时，此对象的引用计数加1，当使用del删除变量指向的引用对象时，引用计数减1。

引用计数增加：1创建新变量指向对象；2对象放入容器中（比如列表）；3对象作为参数传递给函数时。

引用计数减少：1 变量指向其他对象或被显式销毁（del 变量名）；移出容器或容器销毁；一个本地引用离开了其作用域（比如f（x）函数结束后，x指向的对象引用计数减一）

1. **垃圾回收：python会检查引用计数为0的对象，清除其所占的内存空间。**

**垃圾回收时，python不能进行其他任务，因此频繁垃圾回收会降低工作效率。**

**只有在特定情况下才会启动垃圾回收。**

**何时启动：Python运行时会记录其中分配对象的次数和取消分配的次数，当二者差值到一定阈值时才会启动垃圾回收，阈值默认是700.**

**分代回收：以空间换时间的操作方式，python将所有对象分为012三代，新建对象为0代，每一代对应一个链表，若经过某对象一次垃圾回收依然存活，则归入下一代对象。**

**每进行10次0代的垃圾回收会进行一次1代垃圾回收，而每10次1代垃圾回收会进行一次2代垃圾回收。（认为存活时间越长的对象越不可能被删除，因此减少对这些对象的回收次数）**

**当0代对象内存池满后，进行垃圾回收**

**使用标记清除来回收循环引用：只有容器才会造成循环引用的问题，数值和字符串不会。因为容器中可能包含对其他对象的引用，容器对象都有一个PyGC\_head。**

<https://blog.csdn.net/yw_1207/article/details/100017640>

**垃圾回收器从根对象开始遍历（根对象指分配在堆内存之外，可以直接访问到的对象，一般指静态/全局变量以及Thread-Local变量），将可以访问到的对象标记为可达对象（标记在对象的头部信息）；垃圾回收器对堆内存进行线性遍历，读取对象的header信息，将没有标记为可达的对象回收。**

**标记清除阶段会暂停整个程序。**

1. **内存池机制：**
2. **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。Python对象（如整数 浮点和list等）都有其独立的私有内存池。**
3. **Python所有小于256个字节的对象使用pymalloc实现的分配器，而大的对象使用系统的malloc。**
4. **Python 垃圾回收机制：**

Python垃圾回收以引用计数为主，标记-清除和分代回收为辅的机制，其中标记-清除和分代回收主要为了处理循环引用的难题。

引用计数算法：当一个变量保存了对象的引用时，此对象的引用计数加1，当使用del删除变量指向的引用对象时，引用计数减1，若引用计数为0则真的把对象删除（此时对象中的\_\_del\_\_函数会被调用）。

1. **提升代码效率的方式：**

1、应尽量减少函数的嵌套定义，因为函数嵌套定义会导致内部函数反复定义而影响效率；

2、每个import语句只导入一个模块，最好按照标准库、扩展库、自定义库的顺序导入。如果只用到库函数的某一个功能，则尽量使用from导入；

3、选择合适的数据类型；

4、充分利用and/or惰性求值的特点，合理组织条件顺序，把容易False的条件放在前面；

5、**利用生成器对象惰性求值的特点**，避免将其转换为列表、元组等；

6、减少内循环中的无关计算，尽量往外层提取(这是因为内循环的语句会在每次循环中执行，增加无用代码的执行次数，会降低CPU效率)；

7、Python**内部实现对元组做了大量优化，**访问速度比列表快；

8、**字典和集合的速度远快于列表和元组**，这是因为字典和集合都采用了哈希表存储元素；

9、在函数递归调用的情况中，一个函数执行尚未结束就又调用了自己，原来的栈帧还没释放就又分配了新栈帧，会占用大量的栈空间。**所以递归的深度如果太大，可能会导致栈空间不足而导致程序崩溃。**

1. 文件打开模式
2. r:只读方式打开，指针放在开头，默认模式
3. w:只用于写入，文件已存在则覆盖，不存在则创建
4. rb, wb:以二进制方式
5. r+:打开文件用于读写，指针在开头
6. w+:打开文件用于读写，已存在则覆盖，否则创建新文件
7. rb+，wb+
8. a：打开文件用于追加，指针在文档结尾，不存在则创建



1. 函数参数是引用传递，（不是值传递），对于不可变类型，运算不会影响变量自身，对于可变类型，运算可能更改传入的变量。
2. 交并差集合：

List（ set(a).intersection( set(b) ) ） 或 [ I for I in a if I in b]

List( set(a).union( set(b) ))

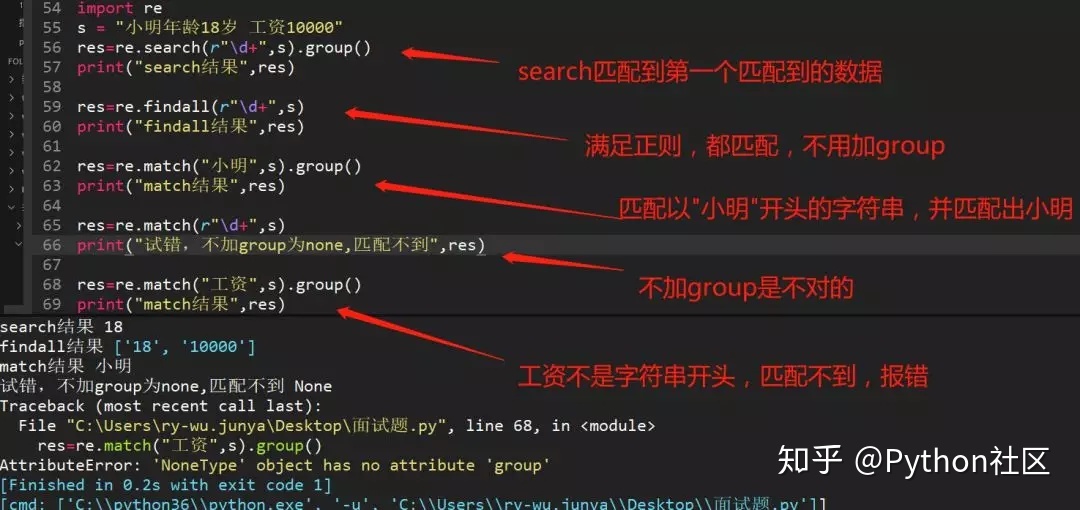
List( set(a).difference( set(b) ))

1. 单引号，双引号，三引号的用法

单双引号没有区别，但单（双）引号内若有单（双）引号，则内部的单（双）引号需要加\转义

三引号可以书写多行，用于大篇幅字符串

1. 打开excel文件 ：res=pandas.read\_excel( “1.xlsx” )



1. **Python中的堆和队列：**

队列：

import queue

q=queue.Queue()

优先队列： q=queue.PriorityQueue()

以下为常用函数q.get() q.put(item) q.empty() q.qsize()

堆：属于完全二叉树（从上到下从左到右）

Import heapq

q=[]

heapq.heappop(q) 弹出并返回最小值

heapq.heappush(q,item) 将item入堆

q = heapq.heapify(list) 将列表转为堆

**另外，heapq默认生成最小堆，如果用最大堆，则需要将元素取负再入堆，出堆时再取负。同样优先队列默认先pop最小值。**

**堆和优先队列：二者插入 查询都是logn，取最值为1.**

**而remove操作前者是logn，后者是n。**

**Python里优先队列的底层是用heapq实现的，python里的堆是通过在列表中维护堆的性质实现的，不是建立二叉树，（是在列表中按照层级遍历顺序存储节点值）。当然优先队列考虑了线程安全的问题。**

另外：队列和栈完全可以用python的列表来实现，不借助第三方。

**Python实现最大最小堆及堆排序：**<https://blog.csdn.net/qq_40587575/article/details/89290135>

1. 鸭子类型：一只动物只要叫起来 看起来，游起来像鸭子，我们就把他看作鸭子，我们只关心行为，不关心类型。比如list.extend() 方法中，不管参数是不是list，只要是可迭代对象如tuple、dict、字符串、生成器、就可以。
2. 新式类与旧式类：新式类早在python2.2就出现了，python3 全是新式类。二者一个区别是，在多类继承上，新式类采用C3算法，旧式类采用从左到右，深度优先原则。
3. Python读取不定行数据：

（strip()用于去掉末尾的换行符）



读取某一行：input()或raw\_input(),前者若读入正确的python表达式会直接读入结果。

Print(‘ ’,end=’’)去掉自动换行

**字符串.split(字符串)**

**字符串,join(字符串的列表)**

1. **位运算**：与& 或| 异或^ 左移位res = num<<位数 右移位res = num>>位数以上均可以直接对整型数字操作.
2. 判断一个**字符**是字母还是数字：string.isdigit() string.isalpha()
3. 协程 coroutine

子程序（即函数）都是层级调用（A调用B，B调C，C、B、A依次返回），因此是通过栈实现的，一个线程就是执行一个子程序。

协程是指子程序运行时可能中断 去执行其他子程序，但不是函数调用而是类似CPU的中断。 有点像多线程但是协程一个线程执行。

优势：极高的执行效率。（是子程序切换，没有线程切换的开销）

不需要多线程的锁机制。（仅一个线程 不存在同时写变量冲突，在协程中控制共享资源不加锁，只需判断状态）

如何利用多核CPU？多进程+协程

1. Python对协程的支持通过generator实现：在generator中，我们不但可以通过for循环来迭代，还可以不断调用next()函数获取由yield语句返回的下一个值。但是Python的yield不但可以返回一个值，它还可以接收调用者发出的参数。
2. 整个流程无锁，由一个线程执行，produce和consumer协作完成任务，所以称为“协程”，而非线程的抢占式多任务（摘自廖雪峰python教程，协程）。子程序是协程的一种特例。
3. 序列化：把内存中的数据结构变成可传输或可存储字节流（字符串）的过程。

## Linux

1. **Linux常用指令**
2. **cd <路径>** change directory，切换当前用户所在路径，可以使绝对路径或相对路径

cd / 切换到根目录； cd ../ 切换到上一目录； cd /system/bin

1. **ls<参数><路径>** list，后面可以跟路径或参数或无，表示列出路径或当前目录下所有文件信息，最常用的是ls –l。

ls / 显示根目录所有文件及文件夹；ls –l /data显示/data路径下所有文件及文件夹的详细信息；ls –l显示当前路径所有文件及文件夹的详细信息

####统计指定目录下文件个数：<https://www.cnblogs.com/yongjieShi/p/8075281.html>

<https://www.cnblogs.com/zeze/p/6839230.html>



1. **查看端口被哪个进程占用？**

<https://www.cnblogs.com/CEO-H/p/7794306.html>

lsof –i [:端口号]

1. **cat <文件>** concatenate，读取文件内容

cat /sys/test.txt 读取

1. **rm <文件>** 或rm –r <文件夹> remove，删除文件或文件夹，常用参数-r -f,-r表示删除目录，也可用于删除文件， -f表强制删除不需确认。

rm –rf path；rm test.txt

rm -f file1 删除一个叫做 'file1' 的文件'   
rmdir dir1 删除一个叫做 'dir1' 的目录'   
rm -rf dir1 删除一个叫做 'dir1' 的目录并同时删除其内容   
rm -rf dir1 dir2 同时删除两个目录及它们的内容

1. **mkdir** <文件夹> make directory，创建文件夹

mkdir /data/path 在/data路径下创建path文件夹

1. **cp** <文件/文件夹><目标文件/文件夹> copy，复制文件或文件夹。

cp /data/logs /data/local/logs 复制/data路径下的logs到/data/local路径下

cp 1.sh /sdcard/ 复制当前路径下的1.sh到/sdcard下

1. **grep**

grep ‘Aug’ /var/log/messages 在文件 '/var/log/messages'中查找关键词"Aug"   
grep ‘^Aug’ /var/log/messages 在文件 '/var/log/messages'中查找以"Aug"开始的词汇   
grep ‘[0-9]’ /var/log/messages 选择 '/var/log/messages' 文件中所有包含数字的行   
grep ‘Aug’ -R /var/log/\* 在目录 '/var/log' 及随后的目录中搜索字符串"Aug"

1. **pwd** 显示工作路径
2. **mv**  mv dir1 new\_dir 重命名/移动 一个目录
3. **查看文件内容：**

**cat     由第一行开始显示内容，并将所有内容输出**

**tac     从最后一行倒序显示内容，并将所有内容输出**

**more    根据窗口大小，一页一页的显示文件内容**

**less    和more类似，但其优点可以往前翻页，而且可以搜索字符**

**head    只显示头几行**

**tail    只显示最后几行**

nl      类似于cat -n，显示时输出行号

tailf   类似于tail -f

cat的功能是将文件从第一行开始连续的将内容输出在屏幕上。但是cat并不常用，原因是当文件大，行数比较多时，屏幕无法全部容下时，只能看到一部分内容。

tac实际上是cat反过来写。这个命令也不常用。

more的功能是将文件从第一行开始，根据输出窗口的大小，适当的输出文件内容。当一页无法全部输出时，可以用“回车键”向下翻行，用“空格键”向下翻页。退出查看页面，请按“q”键。

**less与more类似，但是可以向前翻页。还可以搜索想找的内容。**

cat file1 从第一个字节开始正向查看文件的内容   
tac file1 从最后一行开始反向查看一个文件的内容   
**more** file1 查看一个长文件的内容   
less file1 类似于 'more' 命令，但是它允许在文件中和正向操作一样的反向操作   
head -2 file1 查看一个文件的前两行   
tail -2 file1 查看一个文件的最后两行   
tail -f /var/log/messages 实时查看被添加到一个文件中的内容

1. Find

find / -name file1 从 '/' 开始进入根文件系统搜索文件和目录   
find / -user user1 搜索属于用户 'user1' 的文件和目录   
find /home/user1 -name \\*.bin 在目录 '/ home/user1' 中搜索带有'.bin' 结尾的文件   
find /usr/bin -type f -atime +100 搜索在过去100天内未被使用过的执行文件   
find /usr/bin -type f -mtime -10 搜索在10天内被创建或者修改过的文件   
find / -name \\*.rpm -exec chmod 755 '{}' \; 搜索以 '.rpm' 结尾的文件并定义其权限   
find / -xdev -name \\*.rpm 搜索以 '.rpm' 结尾的文件，忽略光驱、捷盘等可移动设备

1. tar

tar -cvf archive.tar file1 创建一个非压缩的 tarball   
tar -cvf archive.tar file1 file2 dir1 创建一个包含了 'file1', 'file2' 以及 'dir1'的档案文件   
tar -tf archive.tar 显示一个包中的内容   
tar -xvf archive.tar 释放一个包   
tar -xvf archive.tar -C /tmp 将压缩包释放到 /tmp目录下

1. gzip

gunzip file1.gz 解压一个叫做 'file1.gz'的文件   
gzip file1 压缩一个叫做 'file1'的文件   
gzip -9 file1 最大程度压缩   
rar a file1.rar test\_file 创建一个叫做 'file1.rar' 的包   
rar a file1.rar file1 file2 dir1 同时压缩 'file1', 'file2' 以及目录 'dir1'   
unrar x file1.rar 解压rar包

1. help
2. whalis
3. exit
4. ping
5. who 当前登录的用户名
6. su
7. uname
8. 文件夹创建 删除

mkdir

rmdir

1. **tree 以树状图列出指定目录下的文件及子目录的文件**
2. **touch 文件创建，若已存在则刷新访问时间；修改文件或目录的时间属性，包括存在时间和更改时间，若文件不存在则新建文件**
3. **echo 将输入的字符串送往标准输出**
4. df 显示磁盘使用情况 df -h 显示已经挂载的分区列表
5. ps 显示当前系统的进程情况，有哪些进程，及其 id
6. top CPU占用情况从 从大到小
7. shutdown –h now 关闭系统 shutdown –r now 重启系统（reboot）

* 文件管理
* cat tac
* chmod
* chown
* cp
* find
* head tail
* more less
* ln
* locate
* mv
* rm
* touch
* vim
* whereis
* which
* 文档编辑
* grep
* wc
* 磁盘管理
* cd
* df
* du
* ls
* mkdir
* rmdir
* pwd
* 网络通信
* ifconfig
* iptables
* ping
* netstat
* telnet
* 系统管理
* date
* free
* kill
* ps
* rpm
* top
* yum
* 备份压缩
* bzip2
* gzip
* tar
* unzip
* 如何测试一个登陆界面：<http://www.51testing.com/html/99/15137599-3709032.html>
* 功能测试：
* 输入正确的用户和密码，能否成功登录
* 登录后能否跳转到正确的界面
* 输入错误的用户和密码，能否出现错误提示
* 用户名及密码；太短或太长，有空格，有特殊字符的情况能否正确处理
* 密码是否以星号显示
* 记住密码功能测试
* 验证码能否手动刷新（如果有验证码）
* 界面测试
* 性能测试
* 打开登录界面需要几秒
* 登录后跳转需要几秒
* 安全性测试
* 错误登录次数限制
* 用户和密码是否加密传输
* 输入框屏蔽sql注入攻击
* 考虑是否支持同一设备多用户登录
* 考虑是否支持同一用户在多台设备上登录

Springboot的bean生命周期，如何被创建的

介绍netty框架特点，io多路复用

Springboot自动装配原理

ReentrantLock底层原理

1. 大文件问题海量数据问题 <https://blog.csdn.net/wypersist/article/details/80114709>
   * 1. 大文件中有多个字符串如何查重？

建立若干个小文件，将大文件中的每个字符串通过hash算法写入对应的小文件中，这样就保证相同的数据一定在同一个小文件中，小文件的大小一定要保证是内存可读的，然后对每一个小文件进行查重即可。（可以排序，也可以使用集合）

* + 1. 大文件中数据排序

将大文件切分成若干个小文件，将小文件进行排序，然后对小文件进行归并排序，不断生成更大的新文件。注意归并排序的过程中，每次只是读取两个小文件每一行，然后写入新文件，不必将数据保存在内存里，因此不会出现内存不够的情况。

* + 1. 大文件的数据取前k个频率最高的

同样是新建若干个小文件，将大文件中的数据通过hash算法写入不同的小文件中，这样相同的数据肯定在同一个小文件中，然后统计每个小文件里面前k个出现频率最高的，依次遍历每个小文件，通过堆排序找出所有数据中前k个出现频率最高的。