背概念题的时候，可以联系它的英文名字意思来记忆。

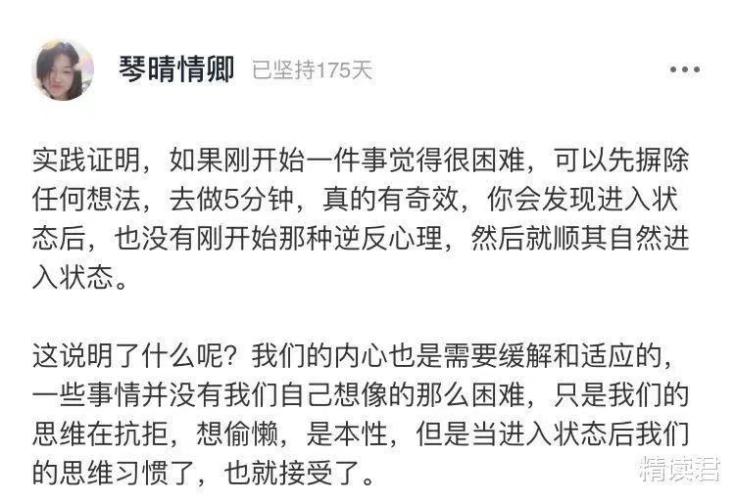
理解记忆为最好，有模糊的赶紧翻笔记去看！

不懂不理解的面试题可以去浏览器的收藏里去找找答案，可能有更易理解的答案。Interview knowledge 书签下看看。

早上是记忆的黄金时间，不要浪费了！

带着问题学习，是最有效的学习方式之一！

再完备的定义，都没有图直观！



# 0.操作系统与计算机网络

## **1.讲一下常见编码方式？**

****ASCII编码：****用来表示英文，它使用1个字节表示，其中第一位规定为0，其他7位存储数据，一共可以表示128个字符。

****拓展ASCII编码：****用于表示更多的欧洲文字，用8个位存储数据，一共可以表示256个字符

****GBK/GB2312/GB18030：****表示汉字。GBK/GB2312表示简体中文，GB18030表示繁体中文。

****Unicode编码：****包含世界上所有的字符，是一个字符集。

****UTF-8：****是Unicode字符的实现方式之一，它使用1-4个字符表示一个符号，根据不同的符号而变化字节长度。

## **Utf-8编码中的中文占几个字节？Int型占几个**

[英文字母和中文汉字在不同字符集编码下的字节数](http://www.cnblogs.com/lancidie/archive/2013/04/12/3017241.html)

英文字母：

字节数 : 1;编码：GB2312

字节数 : 1;编码：GBK

字节数 : 1;编码：GB18030

字节数 : 1;编码：ISO-8859-1

字节数 : 1;编码：UTF-8

字节数 : 4;编码：UTF-16

字节数 : 2;编码：UTF-16BE

字节数 : 2;编码：UTF-16LE

中文汉字：

字节数 : 2;编码：GB2312

字节数 : 2;编码：GBK

字节数 : 2;编码：GB18030

字节数 : 1;编码：ISO-8859-1

字节数 : 3;编码：UTF-8，中文标点占三个字节。一个英文字符等于一个字节，英文标点占一个字节。

字节数 : 4;编码：UTF-16

字节数 : 2;编码：UTF-16BE

字节数 : 2;编码：UTF-16LE

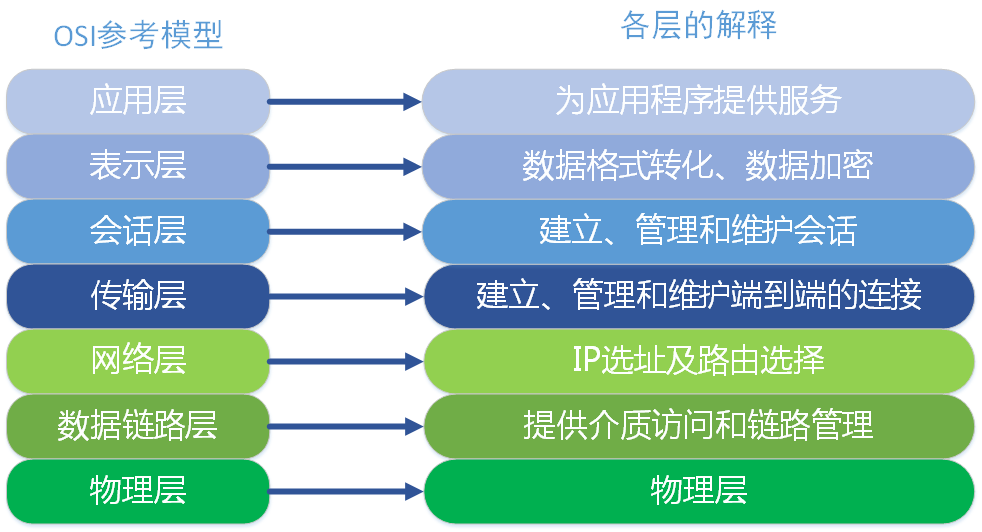
## **说说网络分层，OSI七层模型？**

 1、OSI的来源：OSI（Open System Interconnect），即开放式系统互联。 一般都叫OSI参考模型，是ISO（国际标准化组织）组织在1985年研究的网络互连模型。

 ISO为了更好的使网络应用更为普及，推出了OSI参考模型。其含义就是推荐所有公司使用这个规范来控制网络。这样所有公司都有相同的规范，就能互联了。

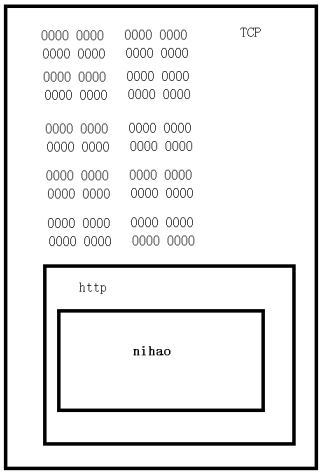
2、OSI七层模型的划分： OSI定义了网络互连的七层框架（物理层、数据链路层、网络层、传输层、会话层、表示层、应用层），即ISO开放互连系统参考模型。如下图。

物数网传会表应

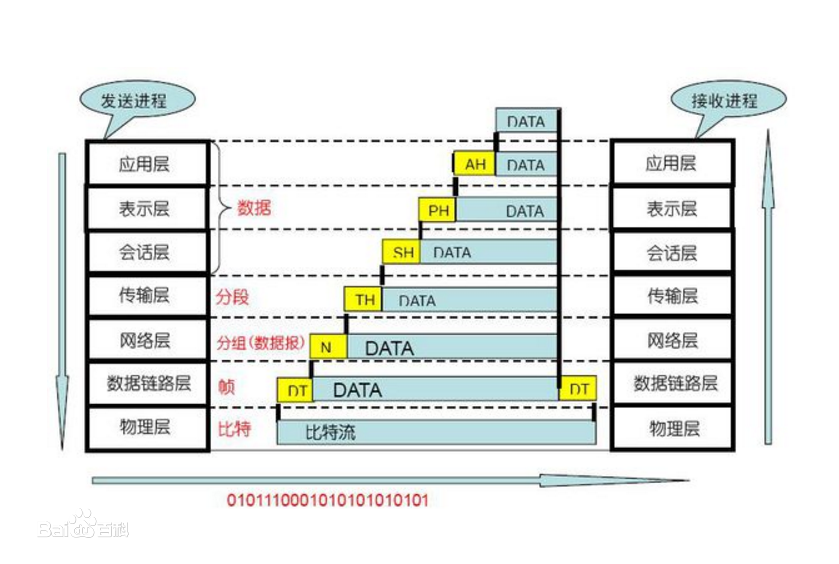


1. 客户端与服务器的连接过程：

01、举例：如下图，客户端发送文本“nihao”，文本在http的请求体里。TCP就会把http从应用层，层层包装到物理层（请求头里很多key:value就是包装的东西。），最后发送出去。这些包装的东西就是大家规范的连接规则。服务端就能识别你，接受你了。TCP包装了20多位，一位8个字节。



02、发送-接收顺序：



## **说说TCP/IP五层模型？**

**1.物理层：**负责光/电信号的传递方式. 比如现在以太网通用的网线(双绞 线)、早期以太网采用的的同轴电缆(现在主要用于有线电视)、光纤, 现在的wifi无线网使用电磁波等都属于物理层的概念。物理层的能力决定了最大传输速率、传输距离、抗干扰性等. 集线器(Hub)工作在物理层.

**2.数据链路层：** 负责设备之间的数据帧的传送和识别. 例如网卡设备的驱动、帧同步(就是说从网线上检测到什么信号算作新帧的开始)、冲突检测(如果检测到冲突就自动重发)、数据差错校验等工作. 有以太网、令牌环网, 无线LAN等标准. 交换机(Switch)工作在数据链路层.

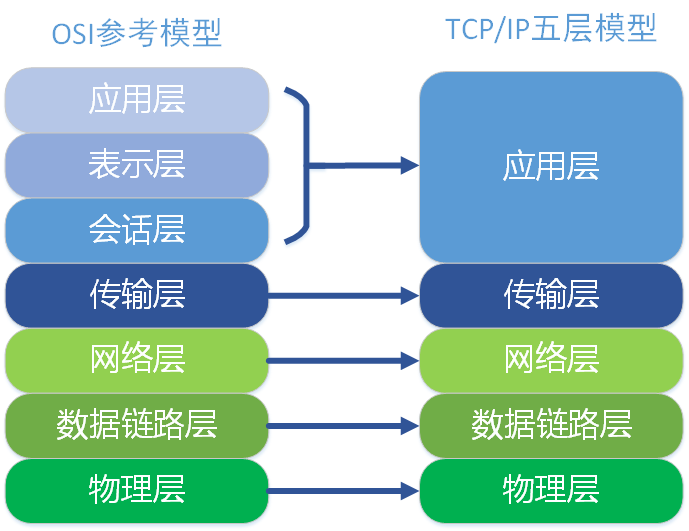
**3.网络层：**负责地址管理和路由选择. 例如在IP协议中, 通过IP地址来标识一台主机, 并通过路由表的方式规划出两台主机之间的数据传输的线路(路由). 路由器(Router)工作在网路层.

**4.传输层：** 负责两台主机之间的数据传输. 如传输控制协议 (TCP), 能够确保数据可靠的从源主机发送到目标主机.

**5.应用层：** 负责应用程序间沟通，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）等. 我们的网络编程主要就是针对应用层.

## **TCP/IP五层协议和OSI的七层协议对应关系？**

OSI更细分，更专业。



## **说说TCP的三次握手和四次挥手？**

**三次握手（传输层）**

1. 客户端创建套接字（即申请的内存区域），开始建立连接
2. 客户端发送控制为 SYN = 1，设置合适的传输序号（和ACK号一起做消息确认机制）和窗口大小（能够接受的最大数据量）
3. 服务端收到后，同样需要返回 SYN = 1表示接受，同时 ACK = 1 表示收到，设置合适的传输序号和窗口大小
4. 客户端收到服务端这个 SYN = 1 表示连接成功，同时需要发送一个 ACK = 1 的消息给服务器，这样才开始传输 (问：要是没有发送ACK怎么办？服务端就会再发送一次ACK，告诉他容量大小。)

**图解传输过程：**

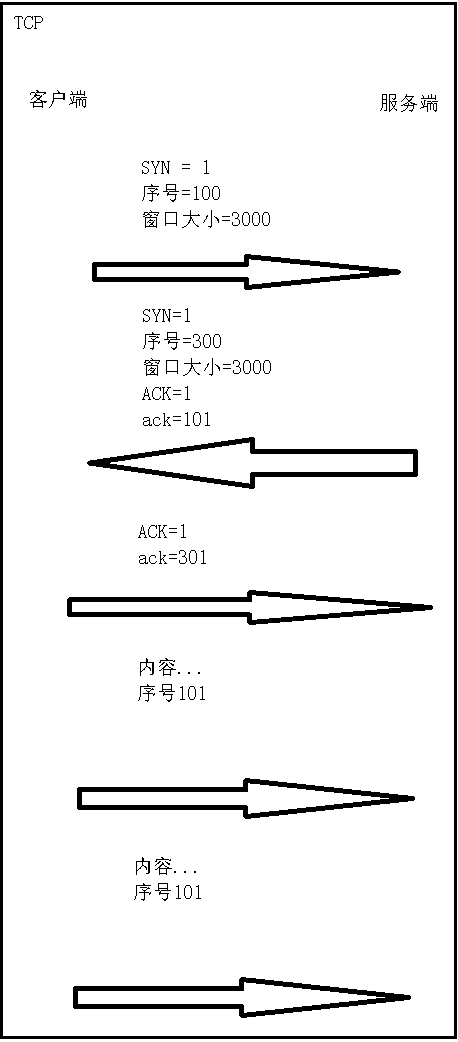
位码即tcp标志位（传输层二进制流01010101里的6位），有6种标示：SYN(synchronous建立联机) ACK(acknowledgement 确认) PSH(push传送) FIN(finish结束) RST(reset重置) URG(urgent紧急)

第一次握手：主机A发送位码为syn＝1，随机产生Sequence number(顺序号码)=100的数据包到服务器，主机B由SYN=1知道，A要求建立联机；

第二次握手：主机B收到请求后要确认联机信息，向A发送 Acknowledge number(确认号码)=(主机A的seq+1)，syn=1，ack=1，随机产生seq=300的包；

第三次握手：主机A收到后检查ack number是否正确，即第一次发送的seq number+1，以及位码ack是否为1，若正确，主机A会再发送ack number=(主机B的seq+1)，ack=1，主机B收到后确认seq值与ack=1则连接建立成功。

完成三次握手，主机A与主机B开始传送数据。



**四次挥手**

主要是TCP是全双工通信（互相同时发消息），一方结束通信另一方可能还在通信，所以通信结束需要四次挥手。

发送 FIN

发送 ACK 号

## **TCP为什么三次握手不是两次握手，为什么两次握手不安全？**

为了实现可靠数据传输， TCP 协议的通信双方， 都必须维护一个序列号， 以标识发送出去的数据包中， 哪些是已经被对方收到的。 三次握手的过程即是通信双方相互告知序列号起始值， 并确认对方已经收到了序列号起始值的必经步骤。

如果只是两次握手， 至多只有连接发起方的起始序列号能被确认， 另一方选择的序号则得不到确认。这样会导致服务端的接收容量大小不被客户端知道，发送超过容量大小的数据就不好了。所以有重试机制。

## **TCP与UDP（适用于直播）的区别与应用？**

TCP是有连接的，UDP无连接

Tcp头部20字节，UDP8字节，所以TCP有更多功能，拥塞控制，确认重传机制~

TCP是面向字节流的，UDP是面向数据报的有大小限制

TCP仅支持1对1

TCP适合对数据准确性要求高的场景

为什么TCP是可靠的，UDP早不可靠的?为什么UDP比TCP快?

因为TCP有很多机制，保证数据传输。UDP只管发送数据，不管是否有人接受。UDP没有三次握手。

## **Http位于TCP/IP模型中的第几层？为什么说Http是可靠的数据传输协议？**

应用层

## **HTTP链接的特点？**

1. 支持客户端/服务器端模式。
2. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。
3. 灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。
4. 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

## **http的get和post的区别？**

安全性上，post略安全一点

参数，一个放在url，一个放在请求体中

长度限制，Get受到浏览器限制url长度，Post不会，但协议没有规定长度限制

Get会被浏览器缓存，Post不会。

## **socket和http的区别？**

http 为短连接：客户端发送请求都需要服务器端回送响应.请求结束后，主动释放链接，因此为短连接。通常的做法是，不需要任何数据，也要保持每隔一段时间向服务器发送"保持连接"的请求。这样可以保证客户端在服务器端是"上线"状态。

HTTP连接使用的是"请求-响应"方式，不仅在请求时建立连接，而且客户端向服务器端请求后，服务器才返回数据。

TCP 三次握手：握手过程中并不传输数据，在握手后服务器与客户端才开始传输数据，理想状态下，TCP 连接一旦建立，在通讯双方中的任何一方主动断开连接之前 TCP 连接会一直保持下去。

Socket 是对 TCP/IP 协议的封装，Socket 只是个接口 API 不是协议，通过 Socket 我们才能使用 TCP/IP 协议，除了 TCP，也可以使用 UDP 协议来传递数据。

创建 Socket 连接的时候，可以指定传输层协议，可以是 TCP 或者 UDP，当用 TCP 连接，该Socket就是个TCP连接，反之。

## **HTTP1.0与2.0的区别？**

**HTTP1.0 HTTP1.1**

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。

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。

错误通知的管理，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。

Host头处理，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），并且它们共享一个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）。

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。

**HTTP2.0**

新的二进制格式（Binary Format），HTTP1.x的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，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。

多路复用（MultiPlexing），即连接共享，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。

header压缩，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。

服务端推送（server push），同SPDY一样，HTTP2.0也具有server push功能。

## **HTTP与HTTPS的区别以及如何实现安全性？**

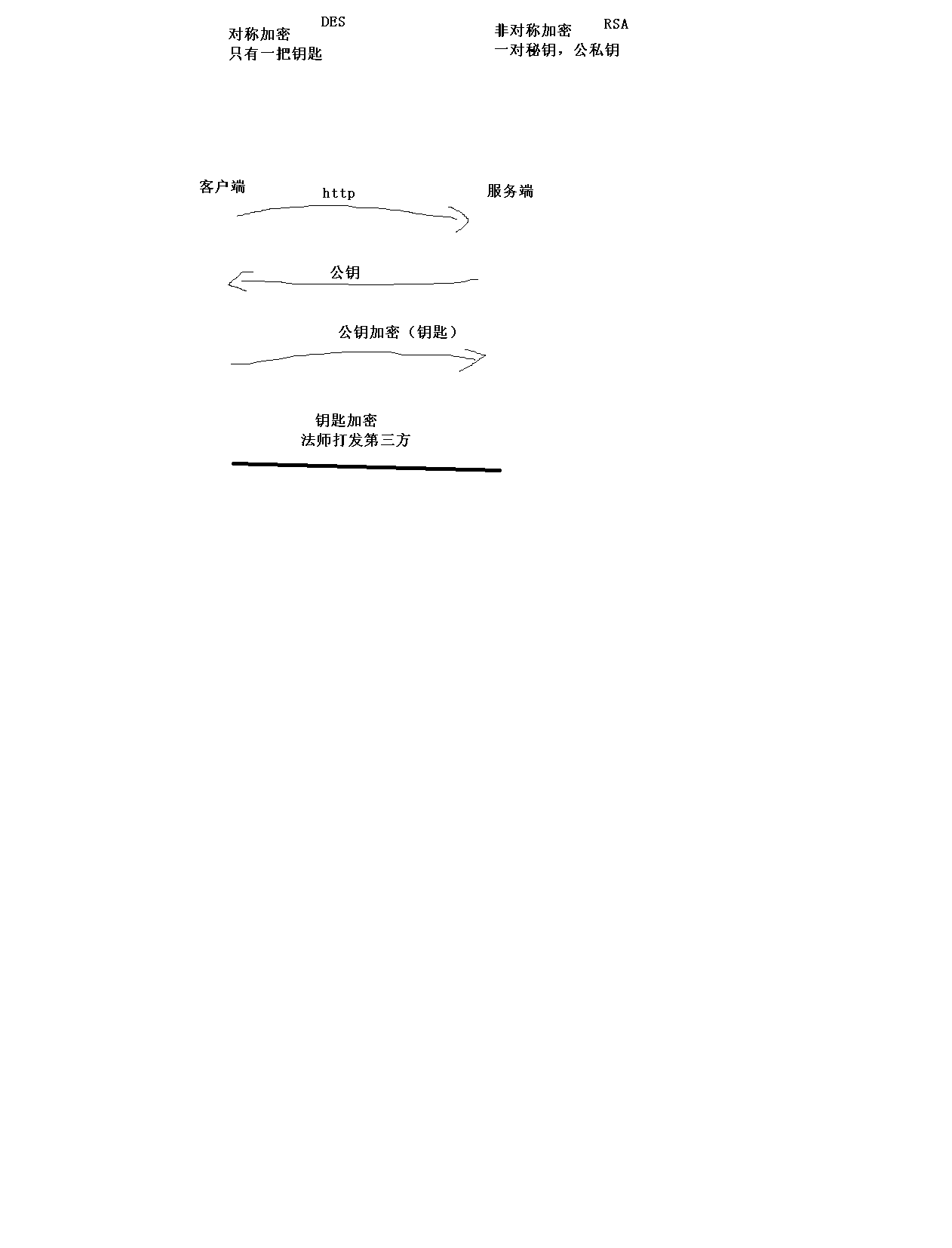
（MD5：不是加密，是生成哈希，特征值，不要搞混。他是不可逆的。

对称加密DES：只有一把钥匙。

非对称加密RSA：一对秘钥，公私钥。）

**在https中，加密过程大致如下：** 首先服务器将公钥给浏览器，浏览器拿到公钥之后，生成一个“会话密钥”，这个会话密钥属于对称加密，然后用公钥加密这个“会话密钥”发送给服务器，最后，在数据传输的过程中，就用这个会话密钥来加密数据。但只有客户端可以解密，服务端解密不了。即使服务端发送数据给多个客户端，其它客户端也解密不了，只有持有公钥的客户端才能解密。

上述的过程是在3次握手中完成，采用明文发送，握手完成以后，客户端和服务端就约定好了“会话密钥”（非对称加密），以后的数据传输，就采用这个会话密钥加密。



## **https中哪里用了对称加密，哪里用了非对称加密，对加密算法（如RSA）等是否有了解?**

传输数据包时使用了对称加密，连接协商加密方法时使用了非对称加密

client如何确定自己发送的消息被server收到?

使用 ACK 机制

## **谈谈你对WebSocket的理解？**

传统Web应用中浏览器与服务器进行数据交互通常需要经过以下几个步骤：

DNS查询

**TCP三次握手**

传送HTTP请求头

传送HTTP请求体(如果有)

服务器处理后传送响应头

服务器传送响应体

断开TCP连接

在WebSocket中进行交互通常为以下几个步骤：

DNS查询

**TCP三次握手**

WebSocket握手

浏览器发送请求

服务器发送响应

断开TCP连接

如果仅是一次通信，二者差异并不是很大，甚至WebSocket比普通方式还要多一次握手。但在需要频繁交互数据时，WebSocket的优势就显露出来了。

例如，当有10次数据交互时，前者要建立10个TCP连接(HTTP 1.0需要建立10次,HTTP 1.1可以通过长连接keep-alive复用TCP连接)，然后要发送10次请求头(包含Cookie等信息,可能会达到K级别)，接收的响应信息可能才几个字节(如某些心跳包)，这样会极大的浪费带宽等资源。试想，如果你在做一个聊天应用，想要获取当前在线人数，你需要向服务器发送你的全部cookie(至少要几百个字节)，除此之外HTTP头中还要包含其他信息，如URL、host等，这些都是必不可少的。最后服务器返回了几百个字节，但其中真正需要用到的只有不到10字节(只需要知道在线人数，其他信息都是无用的)。通过WebSocket，浏览器可以向服务器发送1~2字节的请求(不需要带上cookie验证身份，可以在握手时进行认证，一旦TCP连接建立，则在连接上的通信都是认证过身份的数据，这也是它的好处之一：便于服务端识别客户端的状态)，这个请求仅包含一个特定的控制码(由开发者实现的应用层协议指定)，服务器只需返回特定的返回码及数据即可，一切无用的字节都被省去。

WebSocket协议的开篇就说，本协议的目的是为了解决基于浏览器的程序需要拉取资源时必须发起多个HTTP请求和长时间的轮训的问题

## **WebSocket与socket的区别？**

WebSocket 是应用层协议，socket 是对传输层协议的封装

## **现代浏览器在与服务器建立了一个 TCP 连接后是否会在一个 HTTP 请求完成后断开？什么情况下会断开？**

不会，目前普遍使用 Http 1.1 会复用连接

## **一个 TCP 连接可以对应几个 HTTP 请求？**

1 如果tcp连接保持长连接，Connection:keep-alive && http(s) 1.1 只要在tcp连接（默认两小时）不断开，可以一直串行发送数量无上限；

2 如果tcp连接不保持长连接，Connection:close 只能发一次请求；

3 如果http2 ,采用多路复用技术Multiplexing，一个tcp可以并发多个http请求，同样也是无上限；

## **一个 TCP 连接中 HTTP 请求发送可以一起发送么（比如一起发三个请求，再三个响应一起接收）？**

可以 ，Http2.0 多路复用

## **为什么有的时候刷新页面不需要重新建立 SSL 连接？（https比http多了个s，就是ssl，用来加密的。）**

TCP 连接有的时候会被浏览器和服务端维持一段时间。TCP 不需要重新建立，SSL 自然也会用之前的

## **浏览器对同一 Host 建立 TCP 连接到数量有没有限制？**

假设我们还处在 HTTP/1.1 时代，那个时候没有多路传输，当浏览器拿到一个有几十张图片的网页该怎么办呢？肯定不能只开一个 TCP 连接顺序下载，那样用户肯定等的很难受，但是如果每个图片都开一个 TCP 连接发 HTTP 请求，那电脑或者服务器都可能受不了，要是有 1000 张图片的话总不能开 1000 个TCP 连接吧，你的电脑同意 NAT 也不一定会同意。

所以答案是：有。Chrome 最多允许对同一个 Host 建立六个 TCP 连接。不同的浏览器有一些区别。

如果图片都是 HTTPS 连接并且在同一个域名下，那么浏览器在 SSL 握手之后会和服务器商量能不能用 HTTP2，如果能的话就使用 Multiplexing 功能在这个连接上进行多路传输。不过也未必会所有挂在这个域名的资源都会使用一个 TCP 连接去获取，但是可以确定的是 Multiplexing 很可能会被用到。

如果发现用不了 HTTP2 呢？或者用不了 HTTPS（现实中的 HTTP2 都是在 HTTPS 上实现的，所以也就是只能使用 HTTP/1.1）。

那浏览器就会在一个 HOST 上建立多个 TCP 连接，连接数量的最大限制取决于浏览器设置，这些连接会在空闲的时候被浏览器用来发送新的请求，如果所有的连接都正在发送请求呢？那其他的请求就只能等等了

# JAVA基础

## **面向对象和面向过程的区别？**

* ****面向过程**** ：****面向过程性能比面向对象高。**** 因为类调用时需要实例化，开销比较大，比较消耗资源，所以当性能是最重要的考量因素的时候，比如单片机、嵌入式开发、Linux/Unix 等一般采用面向过程开发。但是，****面向过程没有面向对象易维护、易复用、易扩展。****
* ****面向对象**** ：****面向对象易维护、易复用、易扩展。**** 因为面向对象有封装、继承、多态性的特性，所以可以设计出低耦合的系统，使系统更加灵活、更加易于维护。但是，****面向对象性能比面向过程低****。

[面向过程 ：面向过程性能比面向对象高？？](https://github.com/Snailclimb/JavaGuide/issues/431)

这个并不是根本原因，面向过程也需要分配内存，计算内存偏移量，Java 性能差的主要原因并不是因为它是面向对象语言，而是 Java 是半编译语言，最终的执行代码是.class文件，并不是可以直接被 CPU 执行的二进制机械码。

而面向过程语言大多都是直接编译成机械码在电脑上执行，并且其它一些面向过程的脚本语言性能也并不一定比 Java 好。

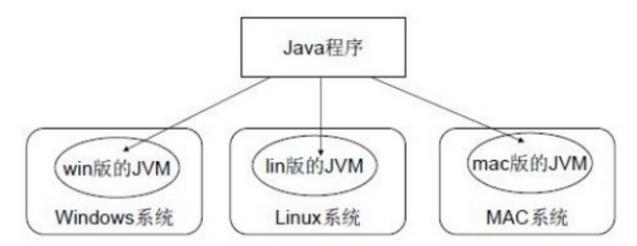
## **Java 语言有哪些特点?**

1. 简单易学；
2. 面向对象（封装，继承，多态）；
3. 平台无关性（ Java 虚拟机实现平台无关性）；
4. 可靠性；
5. 安全性；
6. 支持多线程（ C++ 语言没有内置的多线程机制，因此必须调用操作系统的多线程功能来进行多线程程序设计，而 Java 语言却提供了多线程支持）；
7. 支持网络编程并且很方便（ Java 语言诞生本身就是为简化网络编程设计的，因此 Java 语言不仅支持网络编程而且很方便）；
8. 编译与解释并存；

## 说说Java虚拟机——JVM？

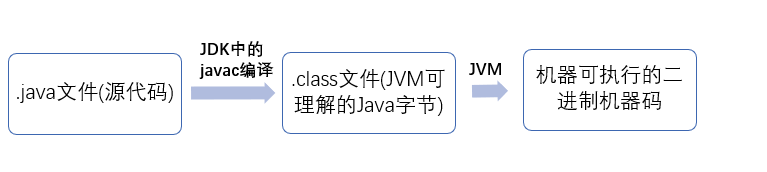
JVM（Java Virtual Machine ）：Java虚拟机简称JVM，是运行Java字节码的虚拟机。

跨平台：任何软件的运行，都必须要运行在操作系统之上，而我们用Java编写的软件可以运行在任何的操作系统上，这个特性称为Java语言的跨平台特性。该特性是由JVM实现的，我们编写的程序运行在JVM上，而JVM运行在操作系统上。



如图所示，Java的虚拟机本身是不具备跨平台功能的，每个操作系统下都有不同版本的虚拟机。

Java程序到源代码到运行一般有下面3步：



我们需要格外注意的是 .class->机器码 这一步。在这一步 JVM 类classloader加载器首先加载字节码文件，然后通过解释器逐行解释执行，这种方式的执行速度会相对比较慢。而且，有些方法和代码块是经常需要被调用的(也就是所谓的热点代码)，所以后面引进了 JIT 编译器，而 JIT 属于运行时编译。当 JIT 编译器完成第一次编译后，其会将字节码对应的机器码保存下来，下次可以直接使用。而我们知道，机器码的运行效率肯定是高于 Java 解释器的。这也解释了我们为什么经常会说 Java 是编译与解释共存的语言。

## 说说JRE 和 JDK？

JRE (Java Runtime Environment) ：是Java程序的运行时环境，它是运行已编译Java程序所需的所有内容的集合，包括jvm、Java类库，java命令和其他的一些基础构件。但是，它不能用于创建新程序。

JDK (Java Development Kit)：是Java程序开发工具包，包含JRE 和开发人员使用的编译器和工具。它能够创建和编译程序。

如果你只是为了运行一下 Java 程序的话，那么你只需要安装 JRE 就可以了。如果你需要进行一些 Java 编程方面的工作，那么你就需要安装 JDK 了。但是，这不是绝对的。有时，即使您不打算在计算机上进行任何 Java 开发，仍然需要安装 JDK。例如，如果要使用 JSP 部署 Web 应用程序，那么从技术上讲，您只是在应用程序服务器中运行 Java 程序。那你为什么需要 JDK 呢？因为应用程序服务器会将 JSP 转换为 Java servlet，并且需要使用 JDK 来编译 servlet。

JVM的堆栈

栈：在jvm中栈用来存储一些对象的引用、局部变量以及计算过程的中间数据，在方法退出后那么这些变量也会被销毁。它的存储比堆快得多，只比CPU里的寄存器慢；

堆：用来存储程序中的一些对象，比如你用new关键字创建的对象，它就会被存储在堆内存中，但是这个对象在堆内存中的首地址会存储在栈中。

## Java1.8 新特性？

Lambda 表达式 − lambda表达式本质上是一段匿名内部类，也可以是一段可以传递的代码，允许把函数作为参数传递进方法中。它的作用就是大大的简化了代码的开发。

函数式接口—就是只定义了一个抽象方法的接口（Object类的public方法除外），就是函数式接口。为了给Lambda表达式的使用提供更好的支持。

方法引用 − 可以直接引用已有Java类的方法或构造器。是lambda表达式的另外一种表现形式并且其语法比lambda表达式更加简单。

默认方法 − 就是在接口里面有了一个实现的方法。

Stream API −新添加的Stream API（java.util.stream） 把真正的函数式编程风格引进来。

Date Time API − 新添加的时间日期API ，加强对日期与时间的处理。

## ****什么是字节码?采用****字节****码的好处是什么?****

在 Java 中，JVM 可以理解的代码就叫做字节码（即扩展名为 .class 的文件），它不面向任何特定的处理器，只面向虚拟机。Java 语言通过字节码的方式，在一定程度上解决了传统解释型语言执行效率低的问题，同时又保留了解释型语言可移植的特点。所以 Java 程序运行时比较高效，而且，由于字节码并不针对一种特定的机器，因此，Java 程序无须重新编译便可在多种不同操作系统的计算机上运行。

## 类什么时候被初始化？

1）创建类的实例，也就是 new 一个对象

2）访问某个类或接口的静态变量，或者对该静态变量赋值

3）调用类的静态方法

4）反射（Class.forName("com.lyj.load")）

5）初始化一个类的子类（会首先初始化子类的父类）

6）JVM 启动时标明的启动类，即文件名和类名相同的那个类

## **Java 和 C++的区别?**

我知道很多人没学过 C++，但是面试官就是没事喜欢拿咱们 Java 和 C++ 比呀！没办法！！！就算没学过 C++，也要记下来！

* 都是面向对象的语言，都支持封装、继承和多态
* Java 不提供指针来直接访问内存，程序内存更加安全
* Java 的类是单继承的，C++ 支持多重继承；虽然 Java 的类不可以多继承，但是接口可以多继承。
* Java 有自动内存管理机制，不需要程序员手动释放无用内存
* ****在 C 语言中，字符串或字符数组最后都会有一个额外的字符‘\0’来表示结束。但是，Java 语言中没有结束符这一概念。****

## Java面向对象三大特性以及你对这些特性的理解？

1. 封装：是把一个对象的属性私有化，同时提供一些可以被外界访问的属性的方法。
2. 继承：是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类。通过使用继承我们能够非常方便地复用以前的代码。

关于继承如下 3 点请记住：

1. 子类拥有父类对象所有的属性和方法（包括私有属性和私有方法），但是父类中的私有属性和方法子类是无法访问，****只是拥有****。
2. 子类可以拥有自己属性和方法，即子类可以对父类进行扩展。
3. 子类可以用自己的方式实现父类的方法。。

多态性：就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量到底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。

在 Java 中有两种形式可以实现多态：继承（多个子类对同一方法的重写）和接口（实现接口并覆盖接口中同一方法）。

## Java为什么不能多继承?

因为可能导致错误，当一个子类继承两个父类，且两个父类都有相同的行为，子类没有重写的情 况下调用此方法，就不知道调用哪个了

## **Java 应用程序与小程序之间有哪些差别?**

简单说应用程序是从主线程启动(也就是 main() 方法)。applet 小程序没有 main() 方法，主要是嵌在浏览器页面上运行(调用init()或者run()来启动)，嵌入浏览器这点跟 flash 的小游戏类似。

## **Object 类的常见方法总结？**

Object 类是一个特殊的类，是所有类的父类。它主要提供了以下 11 个方法：

public final native Class<?> getClass()//native方法，用于返回当前运行时对象的Class对象，使用了final关键字修饰，故不允许子类重写。

public native int hashCode() //native方法，用于返回对象的哈希码，主要使用在哈希表中，比如JDK中的HashMap。public boolean equals(Object obj)//用于比较2个对象的内存地址是否相等，String类对该方法进行了重写用户比较字符串的值是否相等。

protected native Object clone() throws CloneNotSupportedException//naitive方法，用于创建并返回当前对象的一份拷贝。一般情况下，对于任何对象 x，表达式 x.clone() != x 为true，x.clone().getClass() == x.getClass() 为true。Object本身没有实现Cloneable接口，所以不重写clone方法并且进行调用的话会发生CloneNotSupportedException异常。

public String toString()//返回类的名字@实例的哈希码的16进制的字符串。建议Object所有的子类都重写这个方法。

public final native void notify()//native方法，并且不能重写。唤醒一个在此对象监视器上等待的线程(监视器相当于就是锁的概念)。如果有多个线程在等待只会任意唤醒一个。

public final native void notifyAll()//native方法，并且不能重写。跟notify一样，唯一的区别就是会唤醒在此对象监视器上等待的所有线程，而不是一个线程。

public final native void wait(long timeout) throws InterruptedException//native方法，并且不能重写。暂停线程的执行。注意：sleep方法没有释放锁，而wait方法释放了锁 。timeout是等待时间。

public final void wait(long timeout, int nanos) throws InterruptedException//多了nanos参数，这个参数表示额外时间（以毫微秒为单位，范围是 0-999999）。 所以超时的时间还需要加上nanos毫秒。

public final void wait() throws InterruptedException//跟之前的2个wait方法一样，只不过该方法一直等待，没有超时时间这个概念

protected void finalize() throws Throwable { }//实例被垃圾回收器回收的时候触发的操作

## **字符型常量和字符串常量的区别?**

1. 形式上: 字符常量是单引号引起的一个字符; 字符串常量是双引号引起的若干个字符
2. 含义上: 字符常量相当于一个整型值( ASCII 值),可以参加表达式运算; 字符串常量代表一个地址值(该字符串在内存中存放位置)
3. 占内存大小 字符常量只占 2 个字节; 字符串常量占若干个字节。

## Java 的基本数据类型都有哪些各占几个字节？

如下表所示：



## String 是基本数据类型吗？

String 是引用类型，底层用 char 数组实现的。

## short s1 = 1; s1 = s1 + 1; 有错吗?short s1 = 1; s1 += 1 有错吗？

前者不正确，后者正确。对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结果也是 int 型，需要强制转换类型才能赋值给 short 型。而 short s1 = 1; s1 += 1;可以正确编译，因为 s1+= 1;相当于 s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

## **说说自动装箱与拆箱？**

为了能够将这些基本数据类型当成对象操作，Java 为每一个基本数据类型都引入了对应的包装类型（wrapper class），从 Java 5 开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

* ****装箱****：将基本类型用它们对应的引用类型包装起来；
* ****拆箱****：将包装类型转换为基本数据类型；

Java 为每个原始类型提供了包装类型：

- 原始类型: boolean，char，byte，short，int，long，float，double

- 包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

## 为什么要提供包装类呢？？？ 一是为了在各种类型间转化，通过各种方法的调用。否则 你无法直接通过变量转化。 比如，现在int要转为String int a=0; String result=Integer.toString(a); 在java中包装类，比较多的用途是用在于各种数据类型的转化中。 例如：通过包装类来实现转化的 int [num](https://www.baidu.com/s?wd=num&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)=Integer.valueOf("12"); int [num](https://www.baidu.com/s?wd=num&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)2=Integer.parseInt("12"); double [num](https://www.baidu.com/s?wd=num&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)3=Double.valueOf("12.2"); double num4=Double.parseDouble("12.2");

## **整型包装类值的比较**

所有整型包装类对象值的比较必须使用equals方法。

先看下面这个例子：

Integer x = 3;

Integer y = 3;

System.out.println(x == y);// true

Integer a = new Integer(3);

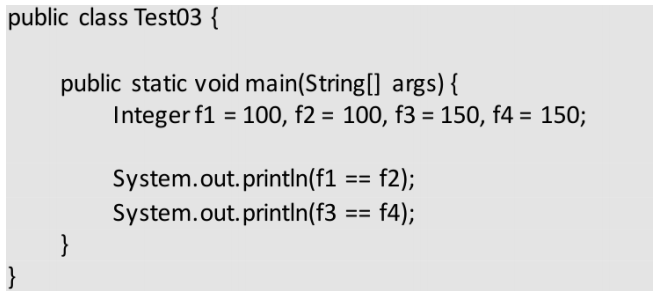
Integer b = new Integer(3);

System.out.println(a == b);//false

System.out.println(a.equals(b));//true

当使用自动装箱方式创建一个Integer对象时，当数值在-128 ~127时，会将创建的 Integer 对象缓存起来，当下次再出现该数值时，直接从缓存中取出对应的Integer对象。所以上述代码中，x和y引用的是相同的Integer对象。

## Integer 类型的数值比较输出的结果为？



首先需要注意的是 f1、f2、f3、f4 四个变量都是 Integer 对象引用，所以下面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个 Integer 对象赋一个 int 值的时候，会调用 Integer 类的静态方法 valueOf。

简单的说，如果整型字面量的值在-128 到 127 之间，那么不会 new 新的 Integer 对象，而是直接引用常量池中的 Integer 对象，所以上面的面试题中 f1==f2 的结果是 true，而 f3==f4 的结果是 false。

## **说说 &和&&、|和||的区别？**

共同点：两者都可做逻辑运算符。它们都表示运算符的两边都是 true 时，结果为 true；

不同点: &也是位运算符。& 表示在运算时两边都会计算，然后再判断；&&表示先运算符号左边的东西，然后判断是否为 true，是 true 就继续运算右边的然后判断并输出，是 false 就停下来直接输出不会再运行后面的东西。

|和||：

共同点：两者都可做逻辑运算符。它们都表示运算符的两边任意一边为 true，结果为 true，两边都不是 true，结果就为 false；

不同点：|也是位运算符。| 表示两边都会运算，然后再判断结果；|| 表示先运算符号左边的东西，然后判断是否为 true，是 true 就停下来直接输出不会再运行后面的东西，是 false 就继续运算右边的然后判断并输出。

例子：

3 | 9=0011（二进制） | 1001（二进制）=1011（二进制）=11（十进制）

## **在一个静态方法内调用一个非静态成员为什么是非法的?**

由于静态方法可以不通过对象进行调用，因此在静态方法里，不能调用其他非静态变量，也不可以访问非静态变量成员。

## **说一下泛型原理，并举例说明？**

原理是参数化类型，举例，定义方法声明形式参数，调用方法的时候传实参，泛型就是定义的时候 参数的数据类型也定义为参数，在实际调用的时候才传入。

## **泛型中extends和super的区别**

代表泛型的上下边界，类型实参只准传入某种类型的父类或某种类型的子类。

## **string 转换成 integer 的方式及原理**

可以使用Integer.paserInt() 方法内部   
1.对 String 判空  
2.判断字符符号+-   
3.循环遍历每个字符十进制值   
4.拼接每一个字符

## **Interger 转为 String ，在数组后面拼接空串和使用String.valueOf的区别?**

1.拼空串的际上经过了两个步骤，首先进行了i.ToString()把 i 转换为 字符串，然后再进行字符串加法运算，由于字符串是不可变的，要需要new一个内存空间来存放新的字符串，字符串加法运算影响了效率。  
2.String.valueOf() 只产生一个对象,调用了 Interge.valueof  
注意Interge缓冲池的问题

## **在 Java 中定义一个不做事且没有参数的构造方法的作用？**

Java 程序在执行子类的构造方法之前，如果没有用 super()来调用父类特定的构造方法，则会调用父类中“没有参数的构造方法”。因此，如果父类中只定义了有参数的构造方法，而在子类的构造方法中又没有用 super()来调用父类中特定的构造方法，则编译时将发生错误，因为 Java 程序在父类中找不到没有参数的构造方法可供执行。解决办法是在父类里加上一个不做事且没有参数的构造方法。

## **构造器 Constructor 是否可被 override?**

Constructor 不能被 override（重写）,但是可以 overload（重载）,所以你可以看到一个类中有多个构造函数的情况。

## 重载和重写的区别？

#### **重载**

发生在同一个类中，方法名必须相同，参数类型、参数个数、参数顺序，方法返回值和访问修饰符可以不同。

#### **重写**

重写是子类对父类的允许访问的方法的实现过程进行重新编写,发生在子类中，方法名、参数列表必须相同，返回值范围小于等于父类，抛出的异常范围小于等于父类，访问修饰符范围大于等于父类。另外，如果父类方法访问修饰符为 private 则子类就不能重写该方法。****也就是说方法提供的行为改变，而方法的外貌并没有改变。****

## **接口和抽象类的区别是什么？**

1. 接口的方法默认是 public，所有方法在接口中不能有实现，而抽象类可以有非抽象的方法。
2. 接口中除了 static、final 变量，不能有其他变量，而抽象类中则不一定。
3. 一个类可以实现多个接口，但只能实现一个抽象类。接口自己本身可以通过 extends 关键字扩展多个接口。
4. 接口方法默认修饰符是 public，抽象方法可以有 public、protected 和 default 这些修饰符（抽象方法就是为了被重写所以不能使用 private 关键字修饰！）。
5. 从设计层面来说，抽象是对类的抽象，是一种模板设计，而接口是对行为的抽象，是一种行为的规范。

## **成员变量与局部变量的区别有哪些？**

1. 从语法形式上看:成员变量是属于类的，而局部变量是在方法中定义的变量或是方法的参数；成员变量可以被 public,private,static 等修饰符所修饰，而局部变量不能被访问控制修饰符及 static 所修饰；但是，成员变量和局部变量都能被 final 所修饰。
2. 从变量在内存中的存储方式来看:如果成员变量是使用static修饰的，那么这个成员变量是属于类的，如果没有使用static修饰，这个成员变量是属于实例的。而对象存在于堆内存，局部变量则存在于栈内存。
3. 从变量在内存中的生存时间上看:成员变量是对象的一部分，它随着对象的创建而存在，而局部变量随着方法的调用而自动消失。
4. 成员变量如果没有被赋初值:则会自动以类型的默认值而赋值（一种情况例外:被 final 修饰的成员变量也必须显式地赋值），而局部变量则不会自动赋值。

## **创建一个对象用什么运算符?对象实体与对象引用有何不同?**

new 运算符，new 创建对象实例（对象实例在堆内存中），对象引用指向对象实例（对象引用存放在栈内存中）。一个对象引用可以指向 0 个或 1 个对象;一个对象可以有 n 个引用指向它。

## **什么是方法的返回值?返回值在类的方法里的作用是什么?**

方法的返回值是指我们获取到的某个方法体中的代码执行后产生的结果！（前提是该方法可能产生结果）。返回值的作用:接收出结果，使得它可以用于其他的操作！

## **一个类的构造方法的作用是什么? 若一个类没有声明构造方法，该程序能正确执行吗? 为什么?**

主要作用是完成对类对象的初始化工作。可以执行。因为一个类即使没有声明构造方法也会有默认的不带参数的构造方法。

## **构造方法有哪些特性？**

1. 名字与类名相同。
2. 没有返回值，但不能用 void 声明构造函数。
3. 生成类的对象时自动执行，无需调用。

## **静态方法和实例方法有何不同？**

1. 在外部调用静态方法时，可以使用"类名.方法名"的方式，也可以使用"对象名.方法名"的方式。而实例方法只有后面这种方式。也就是说，调用静态方法可以无需创建对象。
2. 静态方法在访问本类的成员时，只允许访问静态成员（即静态成员变量和静态方法），而不允许访问实例成员变量和实例方法；实例方法则无此限制。

## **在 Java 中，为什么不允许从静态方法中访问非静态变量？**

因为静态方法静态成员变量会在类加载的时候初始化，但是普通的成员变量只有实例初始化的时候 才会存在，如果静态方法可以访问不存在的非静态变量就会出错

## **静态属性和静态方法是否可以被继承？是否可以被重写？以及原因？**

静态属性和静态方法可继承，不可被重写。 原因，因为静态方法从程序开始运行后就已经分配了内存，也就是说已经写死了。所有引用到该方法的对象（父类的对象也好子类的对象也好）所指向的都是同一块内存中的数据，也就是该静态方法。子类中如果定义了相同名称的静态方法，并不会重写，而应该是在内存中又分配了一块给子类的静态方法，没有重写这一说。不支持多态，动态绑定和静态绑定相关。

## **对象的相等与指向他们的引用相等,两者有什么不同?**

对象的相等，比的是内存中存放的内容是否相等。而引用相等，比较的是他们指向的内存地址是否相等。

## **在调用子类构造方法之前会先调用父类没有参数的构造方法,其目的是?**

帮助子类做初始化工作。

## **== 与 equals 的区别?**

****==**** : 它的作用是判断两个对象的地址是不是相等。即，判断两个对象是不是同一个对象(基本数据类型==比较的是值，引用数据类型==比较的是内存地址)。

****equals()**** :

* 情况 1：类没有覆盖 equals() 方法。等价于通过“==”比较这两个对象。
* 情况 2：类覆盖了 equals() 方法。是比较两个对象的内容是否相等；若它们的内容相等，则返回 true 。

## **说说 hashCode（）？**

hashCode() 的作用是获取哈希码，也称为散列码；它实际上是返回一个 int 整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。hashCode() 定义在 JDK 的 Object.java （所有类的父类）中，这就意味着 Java 中的任何类都包含有 hashCode() 函数。

散列表存储的是键值对(key-value)，它的特点是：能根据“键”快速的检索出对应的“值”。这其中就利用到了散列码！（可以快速找到所需要的对象）

## 你重写过 **hashcode** 和 equals 么，为什么重写 equals 时必须重写 hashCode 方法？

**我们先以“HashSet 如何检查重复”为例子来说明为什么要有 hashCode：** 当你把对象加入 HashSet 时，HashSet **会先计算对象的 hashcode 值来判断对象加入的位置**，同时也会与其他位置已经加入的对象的 hashcode 值作比较，如果没有相符的 hashcode，HashSet 会假设对象没有重复出现。但是如果发现有相同 hashcode 值的对象，这时会调用 equals()方法来检查 hashcode 相等的对象是否真的相同。如果两者相同，HashSet 就不会让其加入操作成功。如果不同的话，就会重新散列到其他位置。。这样我们就大大减少了 equals 的次数，相应就大大提高了执行速度。

## **hashCode（）与 equals（）的相关规定？**

* 如果两个对象相等，则 hashcode 一定也是相同的
* 两个对象相等,对两个对象分别调用 equals 方法都返回 true
* 两个对象有相同的 hashcode 值，它们也不一定是相等的
* 因此，equals 方法被覆盖过，则 hashCode 方法也必须被覆盖
* hashCode()的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode()，则该 class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）

## 为什么两个对象有相同的 hashcode 值，它们也不一定是相等的？

因为计算出来的hashcode值，表示对象在hash表上的索引位置，这个值是有限的，而我们要表示的数却是无穷尽的。所以如果在一个大范围下，哈希值是可能相同的。但是对象却不相同！

## 为什么Java中只有值传递?

**按值调用表示方法接收的是调用者提供的值，而按引用调用表示方法接收的是调用者提供的变量地址。一个方法可以修改传递引用所对应的变量值，而不能修改传递值调用所对应的变量值。**

**Java 程序设计语言总是采用按值调用。方法得到的是所有参数值的一个拷贝，也就是说，方法不能修改传递给它的任何参数变量的内容。**

## 是否可以继承 String ？

String 类是 final 类，不可以被继承。

继承 String 本身就是一个错误的行为，对 String 类型最好的重用方式是关联关系（Has-A）和依赖关系（UseA）而不是继承关系（Is-A）。

## String s=new String("xyz");创建了几个String Object？二者之前的区别是什么？

两个。第一个对象是字符串常量"xyz" 第二个对象是new String("xyz")的时候产生的，在堆中分配内存给这个对象，只不过这个对象的内容是指向字符串常量"xyz" 另外还有一个引用s，指向第二个对象。这是一个变量，在栈中分配内存。

## Math.round(11.5)等于多少？ Math.round(-11.5)等于多少？

Math类中提供了三个与取整有关的方法：ceil、floor、round，这些方法的作用

 \* 与它们的英文名称的含义相对应，例如，ceil的英文意义是天花板，该方法就表示向上取整，

 \* 所以，Math.ceil(11.3)的结果为12,Math.ceil(-11.3)的结果是-11；floor的英

 \* 文意义是地板，该方法就表示向下取整，所以，Math.floor(11.6)的结果为11,Math.floor(-11.6)

 \* 的结果是-12；最难掌握的是round方法，它表示“四舍五入”，算法为Math.floor(x+0.5)，即将原来的

 \* 数字加上0.5后再向下取整，所以，Math.round(11.5)的结果为12，Math.round(-11.5)的结果为-11。

## 用java写一个单例类？/单例模式有几种方式？

单例模式的三个注意点：

将构造方法私有化（保证外部不能直接构造）。

有一个静态属性指向实例

提供一个公有的静态方法向外面提供这个实例。

饿汉式：

public class Singleton {

private Singleton() {

}

private static Singleton instance = new Singleton();

public Singleton getInstance() {

return instance;

}

这种写法就是所谓的**饥饿模式**，每个对象在没有使用之前就已经初始化了。这就可能带来潜在的性能问题：如果这个对象很大呢？没有使用这个对象之前，就把它加载到了内存中去是一种巨大的浪费。

　 针对这种情况，我们可以对以上的代码进行改进，使用一种新的设计思想——延迟加载**（Lazy-load Singleton）**。

懒汉式：

public class Singleton {

private Singleton() {

}

private static Singleton singleton;

private static synchronized Singleton getInstance() {

if (singleton == null) {

singleton = new Singleton();

}

return singleton;

}

这种写法就是所谓的**懒汉模式**。它使用了延迟加载来保证对象在没有使用之前，是不会进行初始化的。

## 简述几种JAVA设计模式？

一、工厂模式 :

我个人的理解是这样的，JAVA多态的一种体现，比如定义了一个接口或者抽象类的变量，可以由它的子类或者实现者来初始化它，但是我们在使用其中的抽象方法的时候，实际上是调用它的子类或者实现者重写的方法。

客户端不再与具体的子类联系在一起。

package org.test;

interface Fruit{

    void eat();

}

class Apple implements Fruit{

    public void eat(){

       System.out.println("吃苹果");

    }

}

class Orange implements Fruit{

    public void eat(){

       System.out.println("吃橘子");

    }

}

class Factory{

    public static Fruit getInstance(String name){

       if(name.equals("Apple")){

           return new Apple();

       }

       if(name.equals("Orange")){

           return new Orange();

       }

return null;

    }

}

public class TestDemo {

    public static void main(String[] args) {

       Factory.getInstance("Apple");

    }

}

1. 观察者模式 :

介绍

意图：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

主要解决：一个对象状态改变给其他对象通知的问题，而且要考虑到易用和低耦合，保证高度的协作。

何时使用：一个对象（目标对象）的状态发生改变，所有的依赖对象（观察者对象）都将得到通知，进行广播通知。

如何解决：使用面向对象技术，可以将这种依赖关系弱化。

关键代码：在抽象类里有一个 ArrayList 存放观察者们。

三、模板模式 :

四、单例模式 :

  定义：在内存中始终只有一个对象

  饿汉式

  特点:当前某一个一加载,就会创建一个对象

  需要将该类的无参构造私有化

  在成员变量创建该类的实例

需要提供公共的访问方法(实际开发中使用这种方式)

  懒汉式

  特点:并不是加载某个类就直接创建对象,而是需要的时候在创建对象

  1)懒加载(延迟加载)

2)线程的安全问题

  线程安全问题检验标准:

  a:是否是多线程环境

  b:是否有共享数据

   c:是否有多条语句对共享数据进行操作!

## **break 和 continue 的区别？**

break 和 continue 都是用来控制循环的语句。

break 用于完全结束一个循环，跳出循环体执行循环后面的语句。

continue 用于跳过本次循环，执行下次循环。

## **String 、StringBuilder 、StringBuffer 的区别？**

****可变性****

String 类中使用 final 关键字修饰字符数组来保存字符串，private final char value[]，所以 String 对象是不可变的。

而 StringBuilder 与 StringBuffer 也是使用字符数组保存字符串char[]value 但是没有用 final 关键字修饰，所以这两种对象都是可变的。

****线程安全性****

String 中的对象是不可变的，也就可以理解为常量，线程安全。StringBuffer 对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。StringBuilder 并没有对方法进行加同步锁，所以是非线程安全的。

****性能****

每次对 String 类型进行改变的时候，都会生成一个新的 String 对象，然后将指针指向新的 String 对象。StringBuffer 每次都会对 StringBuffer 对象本身进行操作，而不是生成新的对象并改变对象引用。相同情况下使用 StringBuilder 比使用 StringBuffer 能获得 10%~15% 左右的性能提升，但却要冒多线程不安全的风险。

****所以：****

1. 操作少量的数据: 适用 String
2. 单线程操作字符串大量数据: 适用 StringBuilder
3. 多线程操作字符串大量数据: 适用 StringBuffer

## String s = "Hello";s = s + " world!";这两行代码执行后，原始的 String 对象中的内容到底变了没有？

没有。因为 String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。在这段代码中，s 原先指向一个 String 对象，内容是 "Hello"，然后我们对 s 进行了“+”操作，那么 s 所指向的那个对象是否发生了改变呢？

答案是没有。这时，s 不指向原来那个对象了，而指向了另一个 String 对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的话会引起很大的内存开销。因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。

例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始化，把它设置为初始值，应当这样做：

1．public class Demo {

2． private String s;

3． ...

4． s = "Initial Value";

5． ...

6．}

而非

1．s = new String("Initial Value");

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个 String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性 s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。

而用关键字 new 调用构造器，总是会创建一个新的对象，无论内容是否相同。 至于为什么要把 String 类设计成不可变类，是它的用途决定的。其实不只 String，很多 Java 标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以 Java 标准类库还提供了一个可变版本，即 StringBuffer。

## 谈谈你对闭包的理解.

闭包就是一个函数引用另外一个函数的变量，因为变量被引用着所以不会被回收，因此可以用来封装一个私有变量。这是优点也是缺点，不必要的闭包只会徒增内存消耗！另外使用闭包也要注意变量的值是否符合你的要求，因为他就像一静态私有变量一样。

## Java中把100转为字符串,把”100”转为整数

java中有很多方法可以将字符型转化为字符串。

方法1：字符串拼接

任何类型变量都可以和字符串相加拼接，加一个[空字符串](https://www.baidu.com/s?wd=%E7%A9%BA%E5%AD%97%E7%AC%A6%E4%B8%B2&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)，就可以得到它转化为字符串的结果。。

String str1 = a + "";

方法2：String类的静态方法

String str5 = String.valueOf(b);

java中，String字符串转化为数字的方法有：

（1）Integer.parseInt(String s) ，代码示例如下：

public class Test {

public static void main(String args[]){

String s = "123";

int num = Integer.parseInt(str);

}}

## 系列化是干什么的？什么情况下会用到序列化？

1系列化是干什么的？  
序列化简单来说就保存对象在内存中的状态也可以说是实例化变量。，一种保存对象状态的机制。只有实现了serializable接口的类的对象才能被实例化。

2什么情况下会用到序列化？  
1当你想把内存中的对象写入到硬盘时

2当你想用套接字在网络上传输对象时

3当你想通过RMI调用对象时

（RMI是什么东西？）：RMI总结来说就是远程调用对象，在一个jvm上调用另一个jvm的对象。

## **Java 序列化中如果有些字段不想进行序列化，怎么办？**

对于不想进行序列化的变量，使用 transient 关键字修饰。

transient 关键字的作用是：阻止实例中那些用此关键字修饰的的变量序列化；当对象被反序列化时，被 transient 修饰的变量值不会被持久化和恢复。transient 只能修饰变量，不能修饰类和方法。

## **获取用键盘输入常用的两种方法**

方法 1：通过 Scanner

Scanner input = new Scanner(System.in);String s = input.nextLine();（nextline处理换行）.

input.close();

方法 2：通过 BufferedReader

BufferedReader input = new BufferedReader(new InputStreamReader(System.in));String s = input.readLine();

## 抽象的(abstract)方法是否可同时是静态的(static), 是否可同时是本地方法(native)，是否可同时被 synchronized？

都不能。

抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。

本地方法是由本地代码（如 C 代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。

synchronized 和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

## 阐述静态变量和实例变量的区别？

静态变量: 是被 static 修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；

实例变量: 必须依存于某一实例，需要先创建对象然后通过对象才能访问到它。静态变量可以实现让多个对象共享内存。

## 说说Java 中IO流？

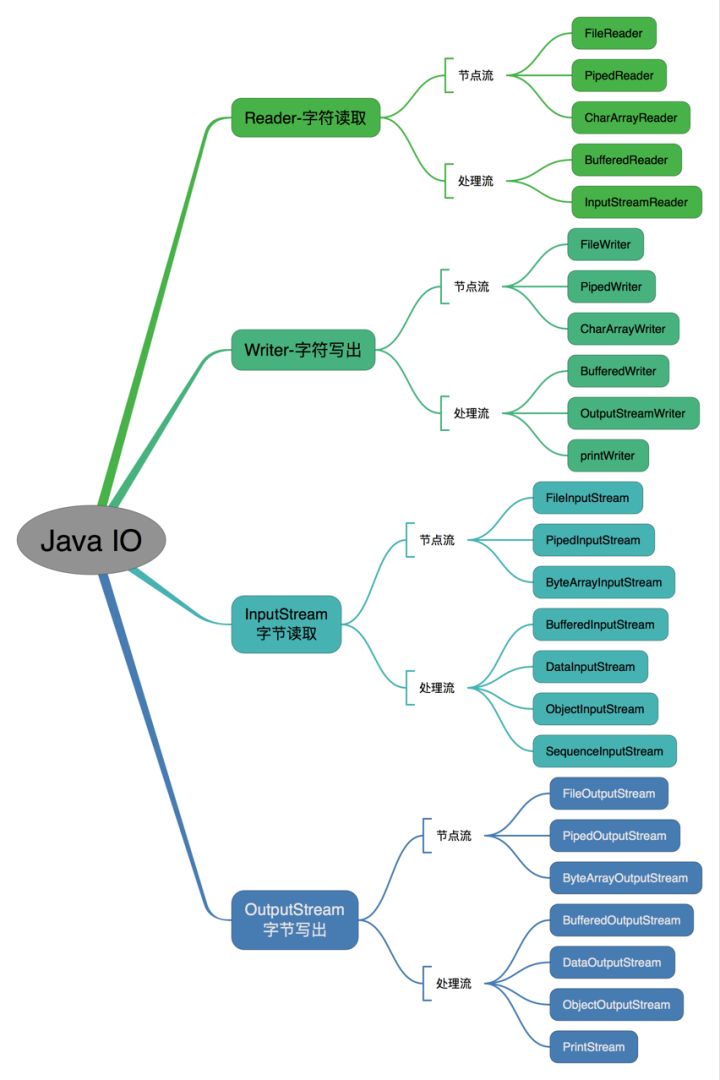
### **Java 中 IO 流分为几种?**

* 按照流的流向分，可以分为输入流和输出流；
* 按照操作单元划分，可以划分为字节流和字符流；
* 按照流的角色划分为节点流和处理流。

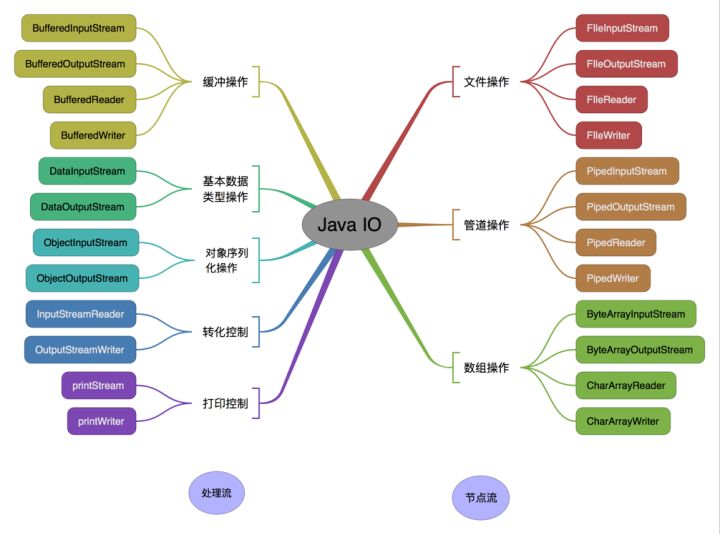
Java Io 流共涉及 40 多个类，这些类看上去很杂乱，但实际上很有规则，而且彼此之间存在非常紧密的联系， Java I0 流的 40 多个类都是从如下 4 个抽象类基类中派生出来的。

* InputStream/Reader: 所有的输入流的基类，前者是字节输入流，后者是字符输入流。
* OutputStream/Writer: 所有输出流的基类，前者是字节输出流，后者是字符输出流。

按操作方式分类结构图：



按操作对象分类结构图：



## 既然有了字节流,为什么还要有字符流?

问题本质想问：****不管是文件读写还是网络发送接收，信息的最小存储单元都是字节，那为什么 I/O 流操作要分为字节流操作和字符流操作呢？****

回答：字符流是由 Java 虚拟机将字节转换得到的，问题就出在这个过程还算是非常耗时，并且，如果我们不知道编码类型就很容易出现乱码问题。所以， I/O 流就干脆提供了一个直接操作字符的接口，方便我们平时对字符进行流操作。如果音频文件、图片等媒体文件用字节流比较好，如果涉及到字符的话使用字符流比较好。

## **说说 [final 关键字](https://snailclimb.gitee.io/javaguide/" \l "/docs/java/basic/final,static,this,super?id=final-%e5%85%b3%e9%94%ae%e5%ad%97)？**

****final关键字主要用在三个地方：变量、方法、类。****

1、当用final修饰一个变量时，如果是基本数据类型的变量，则其数值一旦在初始化之后便不能更改；如果是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象。

2、当用final修饰一个类时，表明这个类不能被继承。final类中的所有成员方法都会被隐式地指定为final方法。

3、当用final修饰方法的原因有两个。第一个原因是把方法锁定，以防任何继承类修改它的含义；第二个原因是效率，在早期的Java实现版本中，final方法会带来性能提升。

在早期的Java实现版本中，会将final方法转为内嵌调用。但是如果方法过于庞大，可能看不到内嵌调用带来的任何性能提升（现在的Java版本已经不需要使用final方法进行这些优化了）。类中所有的private方法都隐式地指定为final。

## **[static 关键字](https://snailclimb.gitee.io/javaguide/" \l "/docs/java/basic/final,static,this,super?id=static-%e5%85%b3%e9%94%ae%e5%ad%97)**

****static 关键字主要有以下四种使用场景：****

1. ****修饰成员变量和成员方法:**** 被 static 修饰的成员属于类，不属于单个这个类的某个对象，被类中所有对象共享，可以并且建议通过类名调用。被static 声明的成员变量属于静态成员变量，静态变量 存放在 Java 内存区域的方法区。调用格式：类名.静态变量名 类名.静态方法名()
2. ****静态代码块:**** 静态代码块定义在类中方法外, 静态代码块在非静态代码块之前执行(静态代码块—>非静态代码块—>构造方法)。 该类不管创建多少对象，静态代码块只执行一次.
3. ****静态内部类（static修饰类的话只能修饰内部类）：**** 静态内部类与非静态内部类之间存在一个最大的区别: 非静态内部类在编译完成之后会隐含地保存着一个引用，该引用是指向创建它的外围类，但是静态内部类却没有。没有这个引用就意味着：1. 它的创建是不需要依赖外围类的创建。2. 它不能使用任何外围类的非static成员变量和方法。
4. ****静态导包(用来导入类中的静态资源，1.5之后的新特性):**** 格式为：import static这两个关键字连用可以指定导入某个类中的指定静态资源，并且不需要使用类名调用类中静态成员，可以直接使用类中静态成员变量和成员方法。

## **[this 关键字](https://snailclimb.gitee.io/javaguide/" \l "/docs/java/basic/final,static,this,super?id=this-%e5%85%b3%e9%94%ae%e5%ad%97)？**

this关键字用于两个地方：

* 访问当前类实例的变量。
* 调用当前类实例的方法。

## **[super 关键字](https://snailclimb.gitee.io/javaguide/" \l "/docs/java/basic/final,static,this,super?id=super-%e5%85%b3%e9%94%ae%e5%ad%97)？**

super关键字用于从子类访问父类的变量和方法。

****使用 this 和 super 要注意的问题：****

* 在构造器中使用 super（） 调用父类中的其他构造方法时，该语句必须处于构造器的首行，否则编译器会报错。另外，this 调用本类中的其他构造方法时，也要放在首行。
* this、super不能用在static方法中。

****简单解释一下：****

被 static 修饰的成员属于类，不属于单个这个类的某个对象，被类中所有对象共享。而 this 代表对本类对象的引用，指向本类对象；而 super 代表对父类对象的引用，指向父类对象；所以,****this和super是属于对象范畴的东西，而静态方法是属于类范畴的东西****。

## 说说反射？

JAVA 反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为 java 语言的反射机制。是另外一种调用类、方法、属性等的调用方式。

普通调用：对象.方法名(实参)

反射调用：操作方法对象的方式来调用的。

## **静态编译和动态编译？**

* \*\*静态编译：\*\*在编译时确定类型，绑定对象
* \*\*动态编译：\*\*运行时确定类型，绑定对象

## 说说获取 Class 对象的两种方式？

如果我们动态获取到这些信息，我们需要依靠 Class 对象。Class 类对象将一个类的方法、变量等信息告诉运行的程序。Java 提供了两种方式获取 Class 对象:

1.知道具体类的情况下可以使用：类名.class;

2.通过 Class.forName()传入类的路径获取：

Class alunbarClass1 = Class.forName("cn.javaguide.TargetObject");

3.实例对象.getClass()。

## 三种方式的区别？

1、类名.class

说明：JVM将使用类装载器, 将类装入内存(前提是:类还没有装入内存),不做类的初始化工作.返回Class的对象；

2、Class.forName("类名字符串")  （注：类名字符串是包名+类名）

说明：装入类,并做类的静态初始化，返回Class的对象；

3、实例对象.getClass()

说明：返回引用o运行时真正所指的对象(因为:子对象的引用可能会赋给父对象的引用变量中)所属的类的Class的对象。

## 说说反射机制优缺点？

* ****优点：**** 运行期类型的判断，动态加载类，提高代码灵活度。
* ****缺点：**** 1,性能瓶颈：反射相当于一系列解释操作，通知 JVM 要做的事情，性能比直接的 java 代码要慢很多。2,安全问题，让我们可以动态操作改变类的属性同时也增加了类的安全隐患。

## 反射的应用场景？

****反射是框架设计的灵魂。****

在我们平时的项目开发过程中，基本上很少会直接使用到反射机制，但这不能说明反射机制没有用，实际上有很多设计、开发都与反射机制有关，例如模块化的开发，通过反射去调用对应的字节码；动态代理设计模式也采用了反射机制，还有我们日常使用的 Spring／Hibernate 等框架也大量使用到了反射机制。

举例：

1. 我们在使用 JDBC 连接数据库时使用 Class.forName()通过反射加载数据库的驱动程序；
2. Spring 框架的 IOC（动态加载管理 Bean）创建对象以及 AOP（动态代理）功能都和反射有联系；
3. 动态配置实例的属性；

## 什么是动态代理？

一．代理类不存在;

二．需要使用某些API，动态的生成一个代理类；

三．然后创建代理类的对象，之后再调用代理类对象的方法。

## 说说动静态代理的区别，什么场景使用？

1. 静态代理通常只代理一个类，动态代理是在程序运行时，通过反射机制动态生成。是代理一个接口下的多个实现类。
2. 静态代理事先知道要代理的是什么，而动态代理不知道要代理什么东西，只有在运行时才知道。
3. 静态代理类：由程序员创建或者由第三方工具生成，再进行编译；在程序运行之前，代理类的.class文件已经存在了。
4. 动态代理是实现 JDK 里一个接口的 invoke 方法，但注意代理的是接口，也就是你的业务类必须要实现接口，通过 newProxyInstance 方法得到代理对象。

还有一种动态代理 CGLIB，代理的是类，不需要业务类实现接口，通过派生的子类来实现代理。通过在运行时，动态修改字节码达到修改类的目的。

AOP 编程就是基于动态代理实现的，比如著名的 Spring 框架、Hibernate 框架等等都是动态代理的使用例子。

## 说说动态代理的作用？

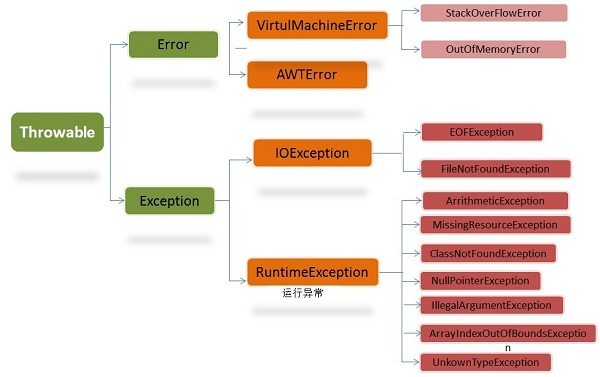
动态代理的优势是实现无侵入式的代码扩展，也就是方法的增强；让你可以在不用修改源码的情况下，增强一些方法；在方法的前后你可以做你任何想做的事情（甚至不去执行这个方法就可以）。

动态代理是设计模式中的代理模式：

定义：为其它对象提供一种代理以控制对这个对象的访问控制；在某些情况下，客户不想或者不能直接引用另一个对象，这时候代理对象可以在客户端和目标对象之间起到中介的作用。

## 说说Java中的异常处理？

Java异常类层次结构图



在 Java 中，所有的异常都有一个共同的祖先 java.lang 包中的 ****Throwable 类****。Throwable： 有两个重要的子类：****Exception（异常）**** 和 ****Error（错误）**** ，二者都是 Java 异常处理的重要子类，各自都包含大量子类。

****Error（错误）:是程序无法处理的错误****，表示运行应用程序中较严重问题。大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM（Java 虚拟机）出现的问题。例如，Java 虚拟机运行错误（Virtual MachineError），当 JVM 不再有继续执行操作所需的内存资源时，将出现 OutOfMemoryError。这些异常发生时，Java 虚拟机（JVM）一般会选择线程终止。

这些错误表示故障发生于虚拟机自身、或者发生在虚拟机试图执行应用时，如 Java 虚拟机运行错误（Virtual MachineError）、类定义错误（NoClassDefFoundError）等。这些错误是不可查的，因为它们在应用程序的控制和处理能力之 外，而且绝大多数是程序运行时不允许出现的状况。对于设计合理的应用程序来说，即使确实发生了错误，本质上也不应该试图去处理它所引起的异常状况。在 Java 中，错误通过 Error 的子类描述。

****Exception（异常）:是程序本身可以处理的异常****。Exception 类有一个重要的子类 ****RuntimeException****。RuntimeException 异常由 Java 虚拟机抛出。****NullPointerException****（要访问的变量没有引用任何对象时，抛出该异常）、****ArithmeticException****（算术运算异常，一个整数除以 0 时，抛出该异常）和 ****ArrayIndexOutOfBoundsException**** （下标越界异常）。

****注意：异常和错误的区别：异常能被程序本身处理，错误是无法处理。****

### **Throwable 类常用方法**

* ****public string getMessage()****:返回异常发生时的简要描述
* ****public string toString()****:返回异常发生时的详细信息
* ****public string getLocalizedMessage()****:返回异常对象的本地化信息。使用 Throwable 的子类覆盖这个方法，可以生成本地化信息。如果子类没有覆盖该方法，则该方法返回的信息与 getMessage（）返回的结果相同
* ****public void printStackTrace()****:在控制台上打印 Throwable 对象封装的异常信息

### **异常处理总结**

* ****try 块：**** 用于捕获异常。其后可接零个或多个 catch 块，如果没有 catch 块，则必须跟一个 finally 块。
* ****catch 块：**** 用于处理 try 捕获到的异常。
* ****finally 块：**** 无论是否捕获或处理异常，finally 块里的语句都会被执行。当在 try 块或 catch 块中遇到 return 语句时，finally 语句块将在方法返回之前被执行。

****在以下 4 种特殊情况下，finally 块不会被执行：****

1. 在 finally 语句块第一行发生了异常。 因为在其他行，finally 块还是会得到执行
2. 在前面的代码中用了 System.exit(int)已退出程序。 exit 是带参函数 ；若该语句在异常语句之后，finally 会执行
3. 程序所在的线程死亡。
4. 关闭 CPU。

## Throw 和Throws 区别?

1. throws出现在方法函数头；而throw出现在函数体。  
   2、throws表示出现异常的一种可能性，并不一定会发生这些异常；throw则是抛出了异常，执行throw则一定抛出了某种异常。  
   3、两者都是消极处理异常的方式（这里的消极并不是说这种方式不好），只是抛出或者可能抛出异常，但是不会由函数去处理异常，真正的处理异常由函数的上层调用处理。

## final、finally、finalize的区别

1. final修饰符（关键字）。被final修饰的类，就意味着不能再派生出新的子类，不能作为父类而被子类继承。将变量或方法声明为final，可以保证他们在使用的过程中不被修改。被声明为final的变量必须在声明时给出变量的初始值，而在以后的引用中只能读取。被final声明的方法也同样只能使用，即不能方法重写。

2、finally是在异常处理时提供finally块来执行任何清除操作。不管有没有异常被抛出、捕获，finally块都会被执行。try块中的内容是在无异常时执行到结束。catch块中的内容，是在try块内容发生catch所声明的异常时，跳转到catch块中执行。finally块则是无论异常是否发生，都会执行finally块的内容，所以在代码逻辑中有需要无论发生什么都必须执行的代码，就可以放在finally块中。

1. finalize是方法名。java技术允许使用finalize（）方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。它是在object类中定义的，因此所有的类都继承了它。子类覆盖finalize（）方法以整理系统资源或者被执行其他清理工作。finalize（）方法是在垃圾收集器删除对象之前对这个对象调用的。

## 说说工具类Collections中的常用方法?

下面是集合中常用的方法

//static void sort(List<T> list) :按照列表中元素的自然顺序进行排序

//static void swap(List list, int i, int j) :将指定列表中的两个索引进行位置互换

//static void shuffle(List list): 随机置换

//static void reverse(List list)  :反转

## **说说Arrays类的常见操作？**

1. 排序 : sort()
2. 查找 : binarySearch()
3. 比较: equals()
4. 填充 : fill()
5. 转列表: asList()
6. 转字符串 : toString()
7. 复制: copyOf()

## [GC是什么? 为什么要有GC?](https://www.cnblogs.com/guweiwei/p/6641016.html)

GC是垃圾收集的意思（Gabage Collection）：将内存管理，尤其是内存空间的释放实现自动化，这就是GC（Garbage Collection）

GC术语定义

1，垃圾：

所谓垃圾（Garbage）,就是需要回收的对象。作为编写程序的人，是可以做出“这个对象已经不需要了“这样的判断，但是计算机是做不到的。因此如果程序（通过某个变量等等）可能会直接或间接的引用一个对象，那么这个对象就被视为“存活“；与之相反，已经引用不到的则被视为“死亡“。将这些死亡对象找出来，然后作为垃圾进行回收，者就是GC的本质。

2，根

所谓的根（Root），就是判断对象是否被引用的起始点。至于哪里的才是根，不通的语言和编译器都有不通的规定，但基本上是将变量和运行栈空间作为根。

GC是垃圾收集器。安全性考虑；减少内存泄露；减少程序员工作量。

## 说说如何确定垃圾 ？

引用计数法

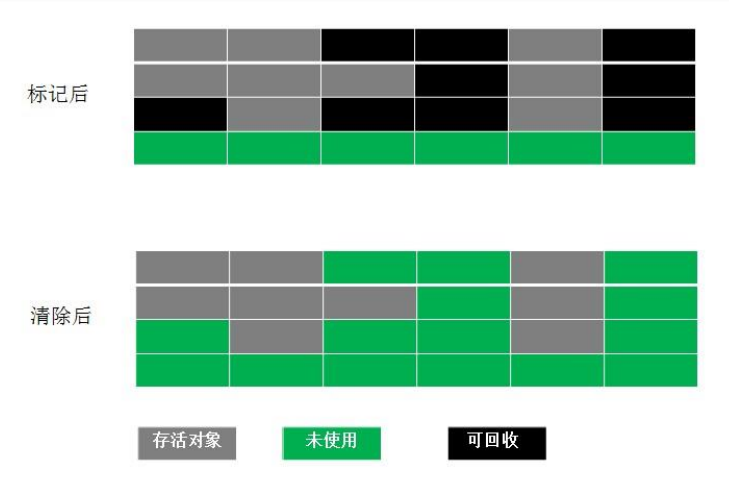
在 Java 中，引用和对象是有关联的。如果要操作对象则必须用引用进行。因此，很显然一个简单 的办法是通过引用计数来判断一个对象是否可以回收。简单说，即一个对象如果没有任何与之关联的引用，即他们的引用计数都不为 0，则说明对象不太可能再被用到，那么这个对象就是可回收对象。

可达性分析

为了解决引用计数法的循环引用问题，Java 使用了可达性分析的方法。通过一系列的“GC roots” 对象作为起点搜索。如果在“GC roots”和一个对象之间没有可达路径，则称该对象是不可达的。要注意的是，不可达对象不等价于可回收对象，不可达对象变为可回收对象至少要经过两次标记 过程。两次标记后仍然是可回收对象，则将面临回收。

## 说说主要GC实现方法？

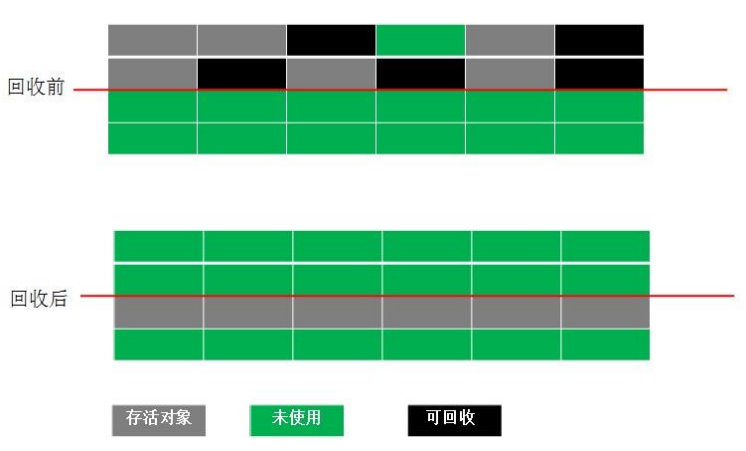
1. 标记清除（Mark and Sweep）：最基础的垃圾回收算法，分为两个阶段，标注和清除。标记阶段标记出所有需要回收的对象，清除阶段回收被标记的对象所占用的空间。如图：



从图中我们就可以发现，该算法最大的问题是内存碎片化严重，后续可能发生大对象不能找到可利用空间的问题。

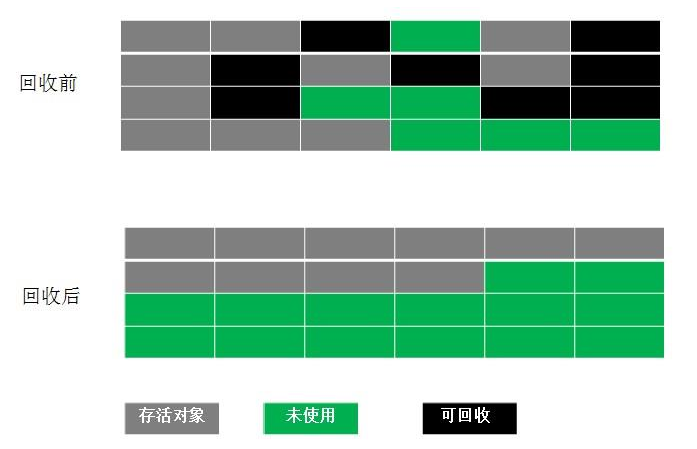
1. 复制回收（Copy recycle）：为了解决 Mark-Sweep 算法内存碎片化的缺陷而被提出的算法。按内存容量将内存划分为等大小的两块。每次只使用其中一块，当这一块内存满后将尚存活的对象复制到另一块上去，把已使用

的内存清掉，如图：



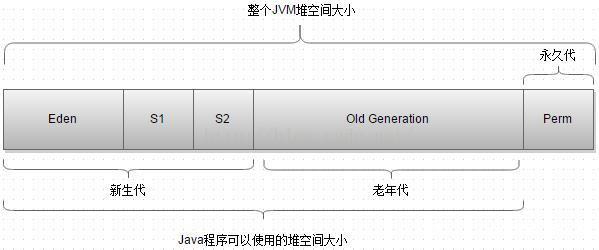
这种算法虽然实现简单，内存效率高，不易产生碎片，但是最大的问题是可用内存被压缩到了原本的一半。且存活对象增多的话，Copying 算法的效率会大大降低。

引用计数方式：结合了以上两个算法，为了避免缺陷而提出。标记阶段和 Mark-Sweep 算法相同，标记后不是清理对象，而是将存活对象移向内存的一端。然后清除端边界外的对象。如图：



## 说说新生代和老年代？

我们先来看一下 Java 堆的结构。



Java 堆空间分成了三部分，这三部分用来存储三类数据：

刚刚创建的对象；

存活了一段时间的对象；

永久存在的对象。

也就是说，常规的 Java 堆至少包括了 新生代 和 老年代 两块内存区域，而且这两块区域有很明显的特征：

新生代：存活对象少、垃圾多；

老年代：存活对象多、垃圾少。

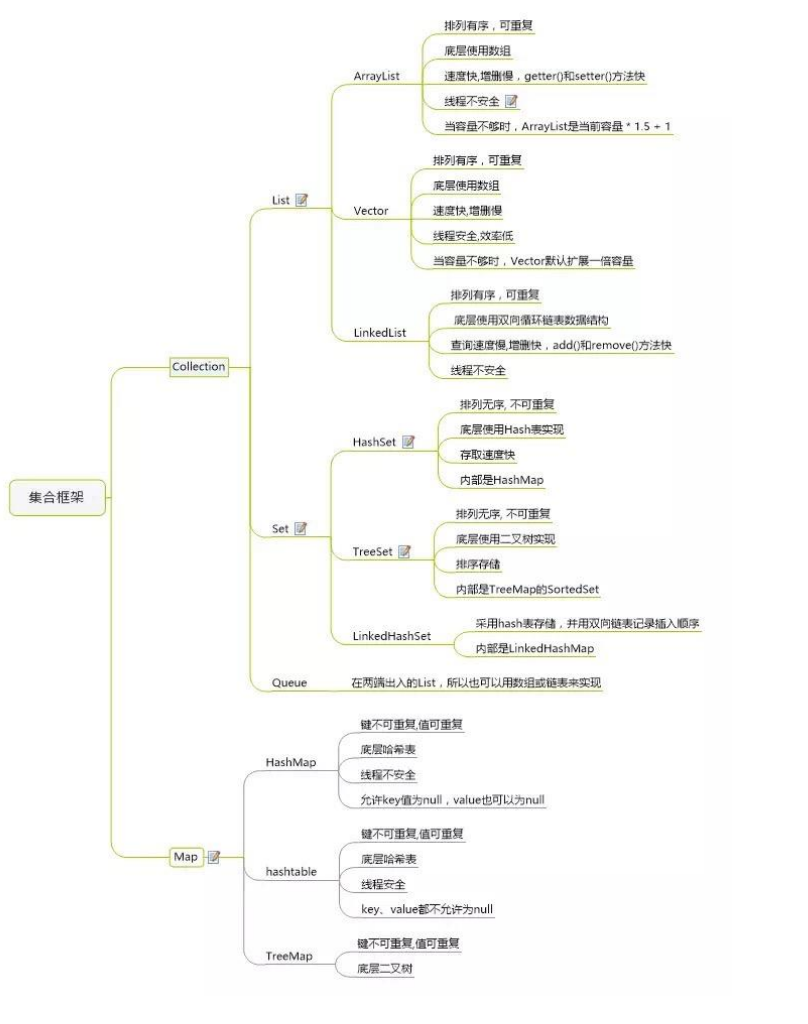
针对这种特点，我们有如下的几种方案;

(1)新生代－复制 回收机制

对于新生代区域，由于每次 GC 都会有大量新对象死去，只有少量存活。因此采用 复制 回收算法，GC 时把少量的存活对象复制过去即可。

（2）老年代－标记整理 回收机制

老年代的特点是：存活对象多、垃圾少。因此，根据老年代的特点，这里仅仅通过少量地移动对象就能清理垃圾，而且不存在内存碎片化。也就是标记整理的回收机制。既然是标记整理算法，而且老年代内部也不存在着内存划分，所以只需要根据标记整理的具体步骤进行垃圾回收就好了。



# Java集合

### 什么是 Set

Set 继承于 Collection 接口，是一个不允许出现重复元素，并且无序的集合，主要 HashSet 和 TreeSet 两大实现类。

在判断重复元素的时候，HashSet 集合会调用 hashCode()和 equal()方法来实现；TreeSet 集合会调用compareTo方法来实现。

补充：有序集合与无序集合说明

有序集合：集合里的元素可以根据 key 或 index 访问 (List、Map)

无序集合：集合里的元素只能遍历。（Set）

### HashSet **和 TreeSet 底层数据结构？**

****HashSet**** 是哈希表结构，主要利用 HashMap 的 key 来存储元素，计算插入元素的 hashCode 来获取元素在集合中的位置；

****TreeSet**** 是红黑树结构，每一个元素都是树中的一个节点，插入的元素都会进行排序；

### **什么是List**

在 List 中，用户可以精确控制列表中每个元素的插入位置，另外用户可以通过整数索引（列表中的位置）访问元素，并搜索列表中的元素。 与 Set 不同，List 通常允许重复的元素。 另外 List 是有序集合而 Set 是无序集合。

**List的常见实现类**

****ArrayList**** 是一个数组队列，相当于动态数组。它由数组实现，随机访问效率高，随机插入、随机删除效率低。

****LinkedList**** 是一个双向链表。它也可以被当作堆栈、队列或双端队列进行操作。LinkedList随机访问效率低，但随机插入、随机删除效率高。

****Vector**** 是矢量队列，和ArrayList一样，它也是一个动态数组，由数组实现。但是ArrayList是非线程安全的，而Vector是线程安全的。

****Stack**** 是栈，它继承于Vector。它的特性是：先进后出(FILO, First In Last Out)。

### **说说List,Set,Map三者的区别？**

List 和 Set 是存储单列数据的集合，Map 是存储键和值这样的双列数据的集合；

* ****List(对付顺序的好帮手)：**** 中存储的数据是有顺序，并且允许重复；
* ****Set(注重独一无二的性质):**** 中存储的数据是无序的，且不允许有重复。
* ****Map(用Key来搜索的专家):**** Map 中存储的数据是没有顺序的，其键是不能重复的，它的值是可以有重复的。

### Array和ArrayList有什么区别？什么时候应该使用Array而不是ArrayList？

Array：

它是数组，申明数组的时候就要初始化并确定长度，长度不可变。而且它只能存储同一类型的数据，比如申明为String类型的数组，那么它只能存储String类型数据。可以包含基本类型和对象类型.

ArrayList：

它是一个集合，需要先申明，然后再添加数据，长度是根据内容的多少而改变的，ArrayList可以存放不同类型的数据，在存储基本类型数据的时候要使用基本数据类型的包装类。只能包含对象类型.

当能确定长度并且数据类型一致的时候就可以用数组，其他时候使用ArrayList.

### 数组有没有 length() 方法？String 有没有 length() 方法？

数组没有 length()方法，而是有 length 的属性。String 有 length()方法。JavaScript 中，获得字符串的长度是通过 length 属性得到的，这一点容易和 Java 混淆。

### List a=new ArrayList()和 ArrayList a =new ArrayList()的区别？

List list = new ArrayList(); 这句创建了一个 ArrayList 的对象后把上溯到了 List。此时它是一个 List 对象了，有些ArrayList 有但是 List 没有的属性和方法，它就不能再用了。而 ArrayList list=new ArrayList();创建一对象则保留了ArrayList 的所有属性。 所以需要用到 ArrayList 独有的方法的时候不能用前者。

### **Arraylist 与 LinkedList 区别?**

****1. 是否保证线程安全：**** ArrayList 和 LinkedList 都是不同步的，也就是不保证线程安全；

****2. 底层数据结构：**** Arraylist 底层使用的是 **Object**数组****；LinkedList 底层使用的是 ****双向链表**** 数据结构。

****3. 插入和删除是否受元素位置的影响：**** ① **ArrayList**采用数组存储，所以插入和删除元素的时间复杂度受元素位置的影响。**** （数组结构的原因，插入/删除元素都要执行向后位/向前移一位的操作。） ② **LinkedList**采用链表存储，所以对于**add(E e)**方法的插入，删除元素时间复杂度不受元素位置的影响。（列表数据结构的原因，插入/删除只需要修改结点上标注下一个元素的坐标即可。）****

****4. 是否支持快速随机访问：**** LinkedList 不支持高效的随机元素访问，而 ArrayList 支持。快速随机访问就是通过元素的序号快速获取元素对象(对应于get(int index) 方法)。

****5. 内存空间占用：**** ArrayList的空间浪费主要体现在在list列表的结尾会预留一定的容量空间，而LinkedList的空间花费则体现在它的每一个元素都需要消耗比ArrayList更多的空间（因为要存放直接后继和直接前驱以及数据）。

### hashing(散列法或哈希法)的概念？

**将任意长度的二进制值串映射为固定长度的二进制值串**的方法，称为散列法，也叫哈希法。由于通过更短的哈希值比用原始值进行数据库搜索更快，这种方法一般用来在数据库中建立索引并进行搜索，同时还用在各种解密算法中。

### 说说HashMap有什么特点？

实现Map接口，使用hash算法,底层数据结构是基于数组+链表来实现的，它可以接收null的键值和值，非同步。

默认情况下HashMap的容量是16，加载因子为0.75. 但是，如果用户通过构造函数指定了一个数字作为容量，那么Hash会选择大于该数字的第一个2的幂作为容量。(3->4、7->8、9->16)。

### 何为加载因子？

加载因子是表示Hsah表中元素的填满的程度.若:加载因子越大,填满的元素越多,好处是,空间利用率高了,但:冲突的机会加大了.反之,加载因子越小,填满的元素越少,好处是:冲突的机会减小了,但:空间浪费多了.

### 为什么默认初始化桶数组大小为16，为什么加载因子的大小为0.75,这两个值的选取有什么特点？

我们都知道桶数组如果扩容，会申请内存空间，然后把原桶中的元素复制进新的桶数组中，这是一个比较耗时的过程。

如果桶初始化桶数组设置太大，就会浪费内存空间，16是一个折中的大小，既不会像1，2，3那样放几个元素就扩容，也不会像几千几万那样可以只会利用一点点空间从而造成大量的浪费。

加载因子设置为0.75而不是1，是因为设置过大，桶中键值对碰撞的几率就会越大，同一个桶位置可能会存放好几个value值，这样就会增加搜索的时间，性能下降，设置过小也不合适，如果是0.1，你放两个键值对就要扩容，太浪费空间了。

### 说说HashMap原理,为什么是线程不安全的？

简述HashMap工作原理

 Hashmap是一个数组，数组中每一个元素是链表。HashMap调用put（）方法存元素进去的时候，会调用hashcode（）方法来计算key的hash散列值来获取到一个index索引值，根据index索引值找到数组的位置，进行元素插入。新来的元素会插入到链表头部。

两个不同的关键字，由于hash散列值相同，因而映射到同一链表位置上。该现象就叫hash冲突。Hash冲突的时候会调用equals方法来进行比较两个对象是否相同。相同则不存。不相同则存储起来。

为什么是线程不安全的

（hash碰撞与扩容导致）多线程环境下put（）导致的扩容，容易形成循环链表，这时有线程来get（）就会陷入死循环

### 为什么hashmap要设置初始值？

设置一个合理的初始化容量可以有效的提高性能。如果我们没有设置初始容量大小，随着元素的不断增加，HashMap会发生多次扩容，而HashMap中的扩容机制决定了每次扩容都需要重建hash表，会非常影响性能。

### 如何保证线程安全的使用Hashmap？

1. **使用HashTable：**

它的底层和HashMap一样，只是在方法上都加上了synchronized关键字，是线程安全版本的HashMap。当一个线程访问HashTable的同步方法时，其他线程如果也要访问同步方法，会被阻塞住。举个例子，当一个线程使用put方法时，另一个线程不但不可以使用put方法，连get方法都不可以，所以效率很低，基本不会选择它。

1. **使用ConcurrentHashMap：**

它可以看作是一个二重hashMap，首先concurrentHashMap是一个segment分段数组，每个分段数组都是一个继承了ReentrantLock可重入锁的类，这样就可以方便地在各个分段数组里面加锁，所以每次锁住的是一个数组，这样只要保证每个数组是线程安全的，也就实现了全局的线程安全。

哦哦还要注意，这个最外面的Segment[]数组，是不可以扩容的！

然后进到Segment内部，会发现，每个Segment可以看作一个HashMap。也就是在一个Segment里面，有个HashEntry[]数组，然后这个数组是一个个桶，桶里面是单向链表。

### **HashMap 和 Hashtable 的区别？**

1. ****线程是否安全：**** HashMap 是非线程安全的，HashTable 是线程安全的；HashTable 内部的方法基本都经过synchronized 修饰。（如果你要保证线程安全的话就使用 ConcurrentHashMap 吧！）；
2. ****效率：**** 因为线程安全的问题，HashMap 要比 HashTable 效率高一点。另外，HashTable 基本被淘汰，不要在代码中使用它；
3. ****对Null key 和Null value的支持：**** HashMap 中，null 可以作为键，这样的键只有一个，可以有一个或多个键所对应的值为 null。。但是在 HashTable 中 put 进的键值只要有一个 null，直接抛出 NullPointerException。
4. ****初始容量大小和每次扩充容量大小的不同 ：**** ①创建时如果不指定容量初始值，Hashtable 默认的初始大小为11，之后每次扩充，容量变为原来的2n+1。HashMap 默认的初始化大小为16。之后每次扩充，容量变为原来的2倍。②创建时如果给定了容量初始值，那么 Hashtable 会直接使用你给定的大小，而 HashMap 会将其扩充为2的幂次方大小（HashMap 中的tableSizeFor()方法保证，下面给出了源代码）。也就是说 HashMap 总是使用2的幂作为哈希表的大小。
5. ****底层数据结构：**** JDK1.8 以后的 HashMap 在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为8）（将链表转换成红黑树前会判断，如果当前数组的长度小于 64，那么会选择先进行数组扩容，而不是转换为红黑树）时，将链表转化为红黑树，以减少搜索时间。Hashtable 没有这样的机制。

### 说说HashSet？

HashSet实现了Set接口，它不允许集合中有重复的值，当我们提到HashSet时，第一件事情就是在将对象存储在HashSet之前，要先确保对象重写equals()和hashCode()方法，这样才能比较对象的值是否相等，以确保set中没有储存相等的对象。如果我们没有重写这两个方法，将会使用这个方法的默认实现。

用add(Object o)方法来在Set中添加元素，当元素值重复时则会立即返回false，如果成功添加的话会返回true。

HashSet实现原理？

对于HashSet而言，它是基于HashMap实现的，HashSet底层使用HashMap来保存所有元素，因此HashSet 的实现比较简单，相关HashSet的操作，基本上都是直接调用底层HashMap的相关方法来完成。

### **HashSet如何检查重复**

当你把对象加入HashSet时，HashSet会先计算对象的hashcode值来判断对象加入的位置，同时也会与其他加入的对象的hashcode值作比较，如果没有相符的hashcode，HashSet会假设对象没有重复出现。但是如果发现有相同hashcode值的对象，这时会调用equals()方法来检查hashcode相等的对象是否真的相同。如果两者相同，HashSet就不会让加入操作成功。

### HashMap和HashSet的区别？

| **HashMap** | **HashSet** |
| --- | --- |
| 实现了Map接口 | 实现Set接口 |
| 存储键值对 | 仅存储对象 |
| 调用put()向map中添加元素 | 调用add()方法向Set中添加元素 |
| HashMap使用键（Key）计算Hashcode | HashSet使用成员对象来计算hashcode值，对于两个对象来说hashcode可能相同，所以equals()方法用来判断对象的相等性， |

**集合框架底层数据结构总结**

**Collection**

**1. List**

Arraylist： Object数组

Vector： Object数组

LinkedList： 双向链表(JDK1.6之前为循环链表，JDK1.7取消了循环)

**2. Set**

HashSet（无序，唯一）: 基于 HashMap 实现的，底层采用 HashMap 来保存元素；

LinkedHashSet：LinkedHashSet 继承于 HashSet，并且其内部是通过 LinkedHashMap 来实现的。有点类似于我们之前说的LinkedHashMap 其内部是基于 HashMap 实现一样，不过还是有一点点区别的；

TreeSet（有序，唯一）： 红黑树(自平衡的排序二叉树)。

1. **Map**

HashMap： JDK1.8之前HashMap由数组+链表组成的，数组是HashMap的主体，链表则是主要为了解决哈希冲突而存在的（“拉链法”解决冲突）。JDK1.8以后在解决哈希冲突时有了较大的变化，当链表长度大于阈值（默认为8）（将链表转换成红黑树前会判断，如果当前数组的长度小于 64，那么会选择先进行数组扩容，而不是转换为红黑树）时，将链表转化为红黑树，以减少搜索时间。

LinkedHashMap： LinkedHashMap 继承自 HashMap，所以它的底层仍然是基于拉链式散列结构即由数组和链表或红黑树组成。另外，LinkedHashMap 在上面结构的基础上，增加了一条双向链表，使得上面的结构可以保持键值对的插入顺序。同时通过对链表进行相应的操作，实现了访问顺序相关逻辑。

Hashtable： 数组+链表组成的，数组是 HashMap 的主体，链表则是主要为了解决哈希冲突而存在的。

TreeMap： 红黑树（自平衡的排序二叉树）。

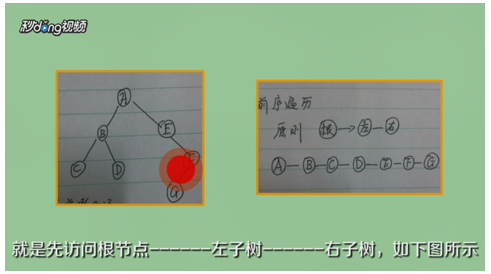
### 什么是二叉树？

每个节点最多只有2个子节点的树，这两个节点分别是左子节点和右子节点。

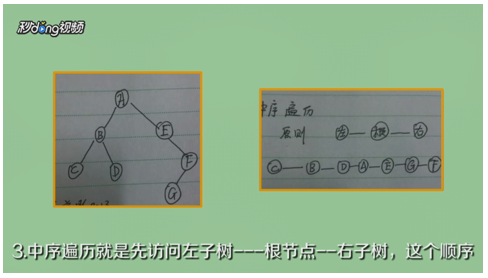
### 说说平衡二叉树的排序？

二叉树排序的基本原理：先构建一颗空树，使用第一个元素作为根节点，如果之后的元素比第一个小，则放到左子树，否则放到右子树，之后按中序遍历。  
时间复杂度：nlog2(n)。

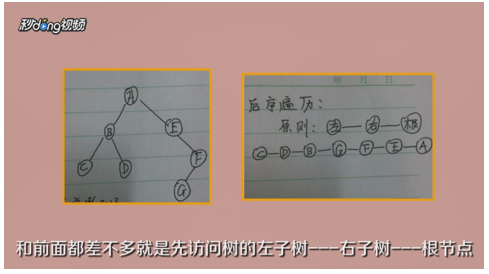
前序遍历：先访问根节点——左子树——右子树。



中序遍历：先访问左子树——根节点——右子树，按照这个顺序。



后序遍历：和前面差不多，先访问树的左子树——右子树——根节点。



### 说说常用的排序算法？

**一、冒泡排序：**列表每两个相邻的数比较大小，如果前边的比后边的大，那么这两个数就互换位置。就像是冒泡一样。

**二、选择排序：**首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。

**三、插入排序：**元素被分为有序区和无序区两部分。最初有序区只有一个元素。每次从无序区中选择一个元素，插入到有序区的位置，直到无序区变空。

还有快速排序、堆排序、归并排序。

# Java并发

## 线程、进程

### 简述线程、进程的基本概念。以及他们之间关系是什么?

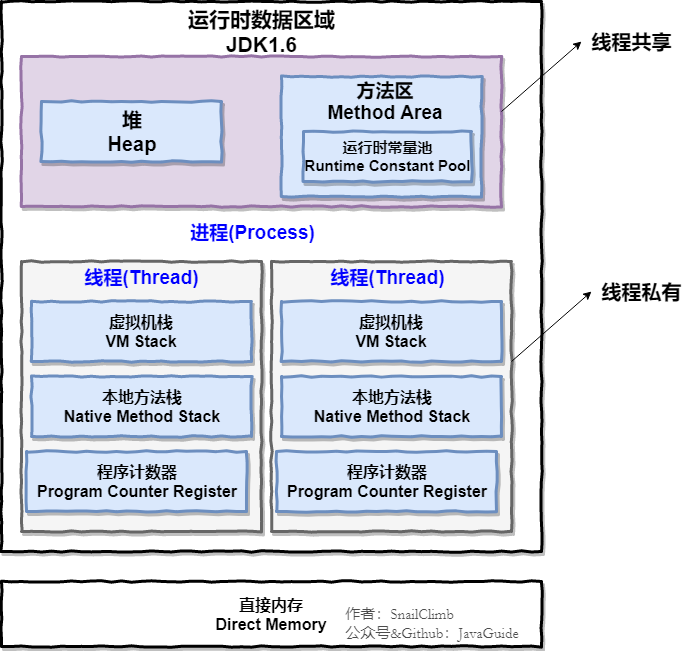
****线程****与进程相似，但线程是一个比进程更小的执行单位。一个进程在其执行的过程中可以产生多个线程。与进程不同的是同类的多个线程共享同一块内存空间和一组系统资源，所以系统在产生一个线程，或是在各个线程之间作切换工作时，

****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。简单来说，一个进程就是一个执行中的程序，它在计算机中一个指令接着一个指令地执行着，同时，每个进程还占有某些系统资源如 CPU 时间，内存空间，文件，输入输出设备的使用权等等。

****从 JVM 角度说进程和线程之间的关系****

**1. 图解进程和线程的关系**

下图是 Java 内存区域，通过下图我们从 JVM 的角度来说一下线程和进程之间的关系。

[](https://camo.githubusercontent.com/a66819fd82c6adfa69b368edf3c52b1fa9cdc89d/68747470733a2f2f6d792d626c6f672d746f2d7573652e6f73732d636e2d6265696a696e672e616c6979756e63732e636f6d2f323031392d332f4a564de8bf90e8a18ce697b6e695b0e68daee58cbae59f9f2e706e67)

从上图可以看出：一个进程中可以有多个线程，多个线程共享进程的****堆****和****方法区 资源，但是每个线程有自己的程序计数器****、****虚拟机栈**** 和 ****本地方法栈****。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。

### **进程间的几种通信方式说一下？**

1. ****管道（pipe）****：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有血缘关系的进程间使用。进程的血缘关系通常指父子进程关系。管道分为 pipe（无名管道）和 fifo（命名管道）两种，有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间通信。
2. ****信号量（semophore）****：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它通常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。
3. ****消息队列（message queue）****：消息队列是由消息组成的链表，存放在内核中 并由消息队列标识符标识。消息队列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。消息队列与管道通信相比，其优势是对每个消息指定特定的消息类型，接收的时候不需要按照队列次序，而是可以根据自定义条件接收特定类型的消息。
4. ****信号（signal）****：信号是一种比较复杂的通信方式，用于通知接收进程某一事件已经发生。
5. ****共享内存（shared memory）****：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问，共享内存是最快的 IPC 方式，它是针对其他进程间的通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量配合使用，来实现进程间的同步和通信。
6. ****套接字（socket）****：socket，即套接字是一种通信机制，凭借这种机制，客户/服务器（即要进行通信的进程）系统的开发工作既可以在本地单机上进行，也可以跨网络进行。也就是说它可以让不在同一台计算机但通过网络连接计算机上的进程进行通信。也因为这样，套接字明确地将客户端和服务器区分开来。

### **线程间的几种通信方式知道不？**

1、锁机制

* 互斥锁：提供了以排它方式阻止数据结构被并发修改的方法。
* 读写锁：允许多个线程同时读共享数据，而对写操作互斥。
* 条件变量：可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件测试是在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。

2、信号量机制：包括无名线程信号量与有名线程信号量

3、信号机制：类似于进程间的信号处理。

线程间通信的主要目的是用于线程同步，所以线程没有象进程通信中用于数据交换的通信机制。

### 一句话简单了解堆和方法区？

堆和方法区是所有线程共享的资源，其中堆是进程中最大的一块内存，主要用于存放新创建的对象 (所有对象都在这里分配内存)，方法区主要用于存放已被加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

### **说说并发与并行的区别?**

* ****并发：**** 同一时间段，多个任务都在执行 (单位时间内不一定同时执行)；
* ****并行：**** 单位时间内，多个任务同时执行。

### **为什么要使用多线程呢?**

先从总体上来说：

* ****从计算机底层来说：**** 线程可以比作是轻量级的进程，是程序执行的最小单位,线程间的切换和调度的成本远远小于进程。另外，多核 CPU 时代意味着多个线程可以同时运行，这减少了线程上下文切换的开销。
* ****从当代互联网发展趋势来说：**** 现在的系统动不动就要求百万级甚至千万级的并发量，而多线程并发编程正是开发高并发系统的基础，利用好多线程机制可以大大提高系统整体的并发能力以及性能。

再深入到计算机底层来探讨：

* ****单核时代：**** 在单核时代多线程主要是为了提高 CPU 和 IO 设备的综合利用率。举个例子：当只有一个线程的时候会导致 CPU 计算时，IO 设备空闲；进行 IO 操作时，CPU 空闲。我们可以简单地说这两者的利用率目前都是 50%左右。但是当有两个线程的时候就不一样了，当一个线程执行 CPU 计算时，另外一个线程可以进行 IO 操作，这样两个的利用率就可以在理想情况下达到 100%了。
* ****多核时代:**** 多核时代多线程主要是为了提高 CPU 利用率。举个例子：假如我们要计算一个复杂的任务，我们只用一个线程的话，CPU 只会一个 CPU 核心被利用到，而创建多个线程就可以让多个 CPU 核心被利用到，这样就提高了 CPU 的利用率。

### **使用多线程可能带来什么问题?**

并发编程的目的就是为了能提高程序的执行效率，提高程序运行速度。但是并发编程并不总是能提高程序运行速度的，而且并发编程可能会遇到很多问题，比如：内存泄漏、上下文切换、死锁还有受限于硬件和软件的资源闲置问题。

### Java 线程有五种基本状态？

**新建状态（New）：**当线程对象对创建后，即进入了新建状态，如：Thread t = new MyThread();

**就绪状态（Runnable）：**当调用线程对象的 start()方法（t.start();），线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待 CPU 调度执行，并不是说执行了 t.start()此线程立即就会执行；

**运行状态（Running）：**当 CPU 开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。

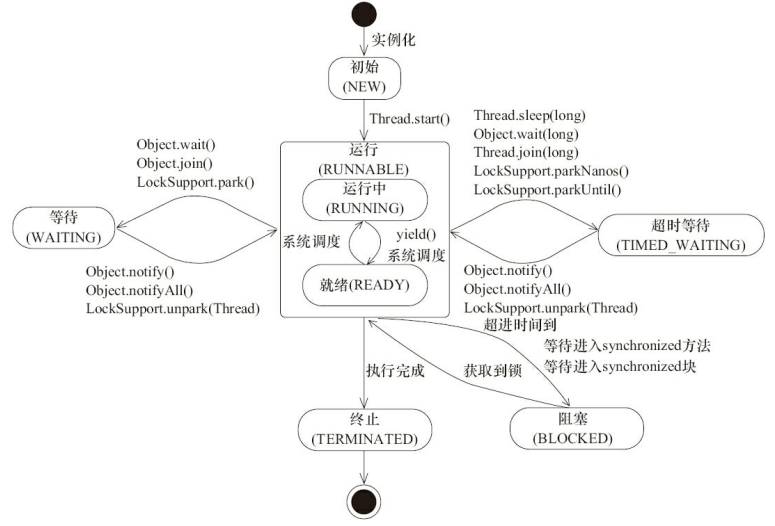
注：就绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

**阻塞状态（Blocked）：**处于运行状态中的线程由于某种原因，暂时放弃对 CPU 的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被 CPU 调用以进入到运行状态。

**死亡状态（Dead）：**线程执行完了或者因异常退出了 run()方法，该线程结束生命周期。

### **说说线程的生命周期和状态?**

线程在生命周期中并不是固定处于某一个状态而是随着代码的执行在不同状态之间切换。Java 线程状态变迁如下图所示：

[](https://camo.githubusercontent.com/e518e038e37c2d27abb394b00b438d347466c90c/68747470733a2f2f6d792d626c6f672d746f2d7573652e6f73732d636e2d6265696a696e672e616c6979756e63732e636f6d2f31392d312d32392f4a6176612b2545372542412542462545372541382538422545372538412542362545362538302538312545352538462539382545382542462538312e706e67)

由上图可以看出：线程创建之后它将处于 ****NEW（新建）**** 状态，调用 start() 方法后开始运行，线程这时候处于 ****READY（可运行）**** 状态。可运行状态的线程获得了 CPU 时间片（timeslice）后就处于 ****RUNNING（运行）**** 状态。当线程执行 wait()方法之后，线程进入 ****WAITING（等待）**** 状态。进入等待状态的线程需要依靠其他线程的通知才能够返回到运行状态，而 ****TIME\_WAITING(超时等待)**** 状态相当于在等待状态的基础上增加了超时限制，比如通过 sleep（long millis）方法或 wait（long millis）方法可以将 Java 线程置于 TIMED WAITING 状态。当超时时间到达后 Java 线程将会返回到 RUNNABLE 状态。当线程调用同步方法时，在没有获取到锁的情况下，线程将会进入到 ****BLOCKED（阻塞）**** 状态。线程在执行 Runnable 的run()方法之后将会进入到 ****TERMINATED（终止）**** 状态。

### **什么是上下文切换?**

多线程编程中一般线程的个数都大于 CPU 核心的个数，而一个 CPU 核心在任意时刻只能被一个线程使用，为了让这些线程都能得到有效执行，CPU 采取的策略是为每个线程分配时间片并轮转的形式。当一个线程的时间片用完的时候就会重新处于就绪状态让给其他线程使用，这个过程就属于一次上下文切换。

概括来说就是：当前任务在执行完 CPU 时间片切换到另一个任务之前会先保存自己的状态，以便下次再切换回这个任务时，可以再加载这个任务的状态。****任务从保存到再加载的过程就是一次上下文切换****。

上下文切换通常是计算密集型的。也就是说，它需要相当可观的处理器时间，在每秒几十上百次的切换中，每次切换都需要纳秒量级的时间。所以，上下文切换对系统来说意味着消耗大量的 CPU 时间，事实上，可能是操作系统中时间消耗最大的操作。

Linux 相比与其他操作系统（包括其他类 Unix 系统）有很多的优点，其中有一项就是，其上下文切换和模式切换的时间消耗非常少。

### **说说 sleep() 方法和 wait() 方法区别和共同点?**

* 两者最主要的区别在于：****sleep 方法没有释放锁，而 wait 方法释放了锁**** 。
* 两者都可以暂停线程的执行。
* Wait 通常被用于线程间交互/通信，sleep 通常被用于暂停执行。
* wait() 方法被调用后，线程不会自动苏醒，需要别的线程调用同一个对象上的 notify() 或者 notifyAll() 方法。sleep() 方法执行完成后，线程会自动苏醒。或者可以使用 wait(long timeout)超时后线程会自动苏醒。

### **为什么我们调用 start() 方法时会执行 run() 方法，为什么我们不能直接调用 run() 方法？**

new 一个 Thread，线程进入了新建状态;调用 start() 方法，会启动一个线程并使线程进入了就绪状态，当分配到时间片后就可以开始运行了。 start() 会执行线程的相应准备工作，然后自动执行 run() 方法的内容，这是真正的多线程工作。 而直接执行 run() 方法，会把 run 方法当成一个 main 线程下的普通方法去执行，并不会在某个线程中执行它，所以这并不是多线程工作。

****总结： 调用 start 方法方可启动线程并使线程进入就绪状态，而 run 方法只是 thread 的一个普通方法调用，还是在主线程里执行,并不是多线程的工作。****

### 多线程的实现方式有哪些?

多线程的形式上实现方式主要有两种，一种是继承Thread类，一种是实现Runnable接口。

### 多线程的原理？

同一时间内，CPU只能处理1条线程，只有1条线程在工作（执行）;多线程并发（同时）执行，其实是CPU快速地在多条线程之间调度（切换）。如果CPU调度线程的时间足够快，就造成了多线程并发执行的假象。

### Thread和Runnable的区别？

* 适合多个相同程序代码的线程去共享同一个资源。
* 可以避免java中单继承的局限性。
* 增加程序的健壮性，实现解耦操作，代码可以被多个线程共享，代码和线程独立。(类与类的弊端 -- 耦合 : 父类坏了,子类都坏了.)
* 线程池只能放入实现Runable或Callable类线程，不能直接放入继承Thread的类。

## 锁机制

### 说说线程安全问题，什么是线程安全？如何保证线程安全？

**线程安全**：就是多线程访问时，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读取完，其他线程才可使用。不会出现数据不一致或者数据污染。 线程不安全就是不提供数据访问保护，有可能出现多个线程先后更改数据造成所得到的数据是脏数据。

**如何保证**：

使用线程安全的类；

使用synchronized同步代码块，或者用Lock锁；

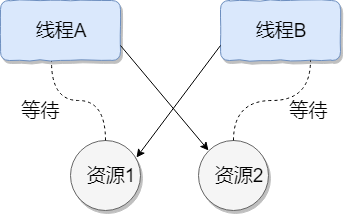
多线程并发情况下，线程共享的变量改为方法局部级变量；

### **什么是线程死锁?如何避免死锁?**

1. 认识线程死锁

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

如下图所示，线程 A 持有资源 2，线程 B 持有资源 1，他们同时都想申请对方的资源，所以这两个线程就会互相等待而进入死锁状态。

[](https://camo.githubusercontent.com/3903a4dc24008be52f72bad23498808b5a743c35/68747470733a2f2f6d792d626c6f672d746f2d7573652e6f73732d636e2d6265696a696e672e616c6979756e63732e636f6d2f323031392d342f323031392d34254536254144254242254539253934253831312e706e67)

### **死锁的条件？**

1. 互斥条件：该资源任意一个时刻只由一个线程占用。
2. 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。
3. 不剥夺条件:线程已获得的资源在末使用完之前不能被其他线程强行剥夺，只有自己使用完毕后才释放资源。
4. 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

### **如何避免**线程**死锁?**

我上面说了产生死锁的四个必要条件，为了避免死锁，我们只要破坏产生死锁的四个条件中的其中一个就可以了。现在我们来挨个分析一下：

1. ****破坏互斥条件**** ：这个条件我们没有办法破坏，因为我们用锁本来就是想让他们互斥的（临界资源需要互斥访问）。
2. ****破坏请求与保持条件**** ：一次性申请所有的资源。
3. ****破坏不剥夺条件**** ：占用部分资源的线程进一步申请其他资源时，如果申请不到，可以主动释放它占有的资源。
4. ****破坏循环等待条件**** ：靠按序申请资源来预防。按某一顺序申请资源，释放资源则反序释放。破坏循环等待条件。

### **说一说自己对于 synchronized 关键字的了解？**

synchronized关键字解决的是多个线程之间访问资源的同步性，synchronized关键字可以保证被它修饰的方法或者代码块在任意时刻只能有一个线程执行。

### **说说自己是怎么使用 synchronized 关键字？**

****synchronized关键字最主要的三种使用方式：****

* ****修饰实例方法:**** 作用于当前对象实例加锁，进入同步代码前要获得当前对象实例的锁
* ****修饰静态方法:**** 也就是给当前类加锁，会作用于类的所有对象实例，因为静态成员不属于任何一个实例对象，是类成员（ static 表明这是该类的一个静态资源，不管new了多少个对象，只有一份）。所以如果一个线程 A 调用一个实例对象的非静态 synchronized 方法，而线程 B 需要调用这个实例对象所属类的静态 synchronized 方法，是允许的，不会发生互斥现象，****因为访问静态 synchronized 方法占用的锁是当前类的锁，而访问非静态 synchronized 方法占用的锁是当前实例对象锁****。
* ****修饰代码块:**** 指定加锁对象，对给定对象加锁，进入同步代码库前要获得给定对象的锁。

****总结：**** synchronized 关键字加到 static 静态方法和 synchronized(class)代码块上都是是给 Class 类上锁。synchronized 关键字加到实例方法上是给对象实例上锁。尽量不要使用 synchronized(String a) 因为JVM中，字符串常量池具有缓存功能！

### **为何要使用同步？**

java 允许多线程并发控制，当多个线程同时操作一个可共享的资源变量时（如数据的增删改查），将会导致数据不准确，相互之间产生冲突，因此加入同步锁以避免在该线程没有完成操作之前，被其他线程调用，从而保证了该变量的唯一性和准确性。

### **如何实现**线程**的同步？**

线程同步有5 种方式：

1. 同步方法 2.同步代码块 3.使用特殊域变量(volatile) 4.使用重入锁（自己可以获取自己的内部锁。当线程请求自己持有的对象锁时，如果锁是重入锁，线程请求成功。） 5.使用局部变量

### **说说volatile 实现原理？**

使用valatile修饰的成员变量，就是告知程序任何对该变量的访问均需要从共享内存中获取，而对它的改变必须同步刷新回共享内存，它能保证所有线程对变量访问的可见性。 原理就是强制把修改的数据写回内存。

### 请说明一下锁和同步的区别？

**用法上的不同：**  
synchronized既可以加在方法上，也可以加在特定代码块上，而lock需要清楚地指定起始位置和终止位置。  
synchronized是托管给JVM执行的，lock的锁定是通过代码实现的，它有比synchronized更精确的线程语义。  
**性能上的不同：**  
lock接口的实现类ReentrantLock，不仅具有和synchronized相同的并发性和内存语义，还多了超时的获取锁、定时锁、等候和中断锁等。  
在竞争不是很激烈的情况下，synchronized的性能优于ReentrantLock，竞争激烈的情况下synchronized的性能会下降的非常快，而ReentrantLock则基本不变。  
**锁机制不同：**  
synchronized获取锁和释放锁的方式都是在块结构中，当获取多个锁时，必须以相反的顺序释放，并且是自动解锁。而Lock则需要开发人员手动释放，并且必须在finally中释放，否则会引起死锁。

### 说说CAS无锁的概念、乐观锁和悲观锁？

**概念：**

CAS（compare and swap）是一种比较交换技术，可以用来鉴别线程技术，一旦检测到冲突就充当当前操作指导直到解决冲突。CAS机制中使用了3个基本操作数：内存地址V，旧的预期值A，要修改的新值B。更新一个变量的时候，只有当变量的预期值A和内存地址V当中的实际值相同时，才会将内存地址V对应的值修改为B。

**乐观锁：**

大多基于版本来实现。将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为提交的数据是过期数据。

**悲观锁：**

对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

### Java 8并法包下常见的并发类？

ConcurrentHashMap、CopyOnWriteArrayList等。

### 偏向锁、轻量级锁、重量级锁、自旋锁的概念？

**偏向锁：**如果一个线程获得了锁，那么锁就进入了偏向模式。当这个线程再次请求锁时，无需再做任何同步操作。

**轻量级锁：**简单的将对象头部作为指针，指向持有锁的线程堆栈的内部，来判断一个线程是否持有对象锁。

**重量级锁：**

**自旋锁：**轻量级锁就会膨胀为重量级锁后，虚拟机为了避免线程真实的在操作系统层面挂起，虚拟机还会在做最后的努力

### 请描述同步锁和异步锁的区别？

异步锁：同一进程内的，多个线程间有互斥关系。只有等一个线程运行结束才能运行运行另一个进程。

同步锁：多个线程运行一个方法，因为方法上加了同了同步，一次只有一个线程运行，其他线程进入竞争进制。

   同步是一种更为复杂的互斥，而互斥是一种特殊的同步。

本质：等待响应

### 有没有用过单体锁？

（在Google markups书签下有详细说明）

由JDK官方提供的锁，只能在一个JVM进程内有效，我们把这种锁叫单体应用锁。单体应用锁的局限性，它只能在一个JVM内加锁，无法跨JVM、跨进程，不能从整个应用层面去加锁。

那么分布式锁的定义就出来了，分布式锁就是可以跨越多个JVM、跨越多个进程的锁，这种锁就叫做分布式锁。通过第三方组件实现分布式锁。

### 说**说Java 中实现分布式锁的常见方式？**

我们用简单的比喻说明了锁的本质：****同一时间只允许一个用户操作****。所以理论上，能够满足这个需求的工具我们都能够使用 (就是其他应用能帮我们加锁的)：

1. ****基于 MySQL 中的锁****：MySQL 本身有自带的悲观锁 for update 关键字，也可以自己实现悲观/乐观锁来达到目的；
2. ****基于 Zookeeper 有序节点****：Zookeeper 允许临时创建有序的子节点，这样客户端获取节点列表时，就能够当前子节点列表中的序号判断是否能够获得锁；
3. ****基于 Redis 的单线程****：由于 Redis 是单线程，所以命令会以串行的方式执行，并且本身提供了像 SETNX(set if not exists) 这样的指令，本身具有互斥性；

### 什么是线程池?

就是一个容纳多个线程的容器，其中的线程可以反复使用，省去了频繁创建线程对象的操作，可以降低资源消耗。

### 线程池作用？

第一：降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。  
第二：提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。  
第三：提高线程的可管理性。线程是稀缺资源，如果无限制地创建，不仅会消耗系统资源，  
还会降低系统的稳定性，使用线程池可以进行统一分配、调优和监控。但是，要做到合理利用。

### **对象池技术的基本原理**

对象池技术基本原理的核心有两点：缓存和共享，即对于那些被频繁使用的对象，在使用完后，不立即将它们释放，而是将它们缓存起来，以供后续的应用程序重复使用，从而减少创建对象和释放对象的次数，进而改善应用程序的性能。事实上，由于对象池技术将对象限制在一定的数量，也有效地减少了应用程序内存上的开销。

### 说说线程池初始化的方式有几种？

Java通过Executors提供四种线程池：

**newCachedThreadPool** 创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。  
**newFixedThreadPool** 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。  
**newScheduledThreadPool** 创建一个定长线程池，支持定时及周期性任务执行。  
**newSingleThreadExecutor** 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序, 优先级执行。

Executors  
Executors是个静态工厂类。它通过静态工厂方法返回ExecutorService、ScheduledExecutorService、ThreadFactory 和 Callable 等类的对象。

# JavaWEB基础

### session怎么共享

1.以cookie加密的方式保存在客户端.优点是减轻服务器端的压力.

2.使用主-从服务器的架构，当用户在主服务器上登录后，通过脚本或者守护进程的方式，将session信息传递到各个从服务器中，这样，用户访问其它的从服务器时，就可以读到session信息。

3.提供一个群集保存session共享信息.其他应用统统把自己的session信息存放到session集群服务器组。当应用系统需要session信息的时候直接到session群集服务器上读取。目前大多都是使用Memcache来对Session进行存储。

4.把Session持久化到数据库,其它应用可以从数据库中查出session信息。所使用的数据库一般为mysql。

### Cookie和Session的区别

* 1. *定义：*

Cookie：是浏览器访问某个服务器时，服务器会创建一部分数据（以消息头set-cookie的方式)发送给浏览器，浏览器会将这部分数据保存下来。当浏览器再次访问该服务器时，会将这部分数据发送给服务器。也是在客户端（浏览器）维护用户状态的一种状态管理技术。

Session：在服务器端维护用户状态的一种状态管理技术。当浏览器访问服务器时，服务器会创建一个对象(session对象),同时，还会生成一个标识该对象的唯一的字符串(sessionId),服务器在默认情况下，使用cookie机制将sessionId发送给浏览器。浏览器下次访问服务器时，会将sessionId携带给服务器，服务器会使用sessionId查找对应的session对象。通过这种方式，来维护用户的状态。

*(2) cookie 和session 的区别：*

① cookie数据存放在客户的浏览器上，session数据放在服务器上。

② 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 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.

如果考虑到减轻服务器性能方面，应当使用COOKIE。

④ 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。

### http 常见的状态码有哪些？

200 OK //客户端请求成功

302 found 重定向

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

401 Unauthorized //请求未经授权

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的 URL

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

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

### **GET 和 POST 的区别？**

**从表面现像上面看 GET 和 POST 的区别：**

**1. GET 请求的数据会附在 URL 之后（就是把数据放置在 HTTP 协议头中），以?分割 URL 和传输数据，参数之间以&相连，如：login.action?name=zhagnsan&password=123456。POST 把提交的数据则放置在是 HTTP 包的包体中。**

**2. GET 方式提交的数据最多只能是 1024 字节，理论上 POST 没有限制，可传较大量的数据。其实这样说是错误的，不准确的：**

**“GET 方式提交的数据最多只能是 1024 字节"，因为 GET 是通过 URL 提交数据，那么 GET 可提交的数据量就跟URL 的长度有直接关系了。而实际上，URL 不存在参数上限的问题，HTTP 协议规范没有对 URL 长度进行限制。这个限制是特定的浏览器及服务器对它的限制。IE对URL长度的限制是2083字节(2K+35)。对于其他浏览器，如Netscape、FireFox 等，理论上没有长度限制，其限制取决于操作系统的支持。**

**3. POST 的安全性要比 GET 的安全性高。注意：这里所说的安全性和上面 GET 提到的“安全”不是同个概念。上面“安全”的含义仅仅是不作数据修改，而这里安全的含义是真正的 Security 的含义，比如：通过 GET 提交数据，用户名和密码将明文出现在 URL 上，因为(1)登录页面有可能被浏览器缓存，(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用 GET 提交数据还可能会造成 Cross-site request forgery 攻击。**

**Get 是向服务器发索取数据的一种请求，而 Post 是向服务器提交数据的一种请求，在 FORM（表单）中，Method默认为"GET"，实质上，GET 和 POST 只是发送机制不同，并不是一个取一个发！**

### **网关是啥?公司还有个网关吗？不是指自己的ip地址？**

**网关不是这个意思 可能说的是网段。就是你的IP地址是哪个段。**

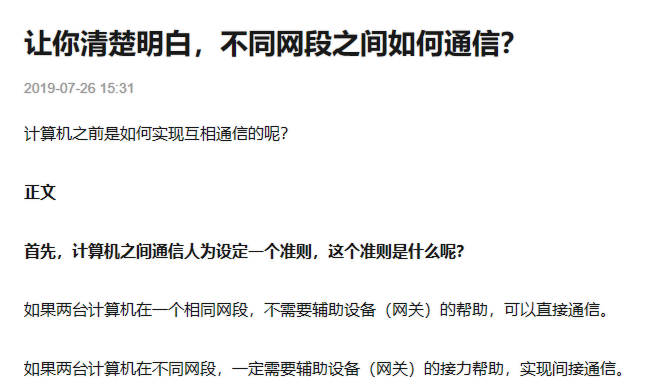
**网段:用来区分网路上的主机是否在同一网路区段内,在局域网中,每台电脑**

**只能和自己同一网段的电脑互相通讯.IP地址是192.168.123.250,说明它处於192.168.123.X的网段(X代表1-255之间的任意值).若您的IP地址是192.168.1.X或是其他位址,说明二者不处於同一网段,则它们之间无法相互连接。**

**192.168.123.250**

**192.168.1.X**

**这两个就不是同一个网段的无法通信。**

****

### **会话追踪？**

会话跟踪常用的4种方法：URL重写，隐藏表单域，cookie,sesion

URL重写技术就是在URL结尾添加一个附加数据以标识该会话，把会话ID通过URL的信息传递过去，以便在服务端进行识别不同的用户，

隐藏表单域：将会话ID添加到HTML表单元素中提交到[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/shuye_gen/article/details/_blank)，此表单不再客户端显示，

cookie,Cookie是Web服务器发送给客户端的一小段信息，客户端请求时可以读取该信息发送到服务器端，进而进行用户的识别。对于客户端的每次请求，服务器都会将Cookie发送到客户端,在客户端可以进行保存,以便下次使用。

 session:

在服务器端会创建一个session对象，产生一个sessionID来标识这个session对象，然后将这个sessionID放入到Cookie中发送到客户端，下一次访问时，sessionID会发送到服务器，在服务器端进行识别不同的用户 ,

Session是依赖Cookie的，如果Cookie被禁用，那么session也将失效

session默认的会话时长为30分钟.

### 阐述常见HTTP安全和攻防策略.

Web攻击的常用方式.

1. 下载数据库.2.上传漏洞.3.SQL注入.4.旁注

防御 :

1. 防止数据库下载. 在IIS中添加一个.mdb的解释器.
2. 单一站点,单一用户,防止旁注.
3. 将上传目录权限更改为”无”,防止asp脚本木马被执行.
4. 采用预编译语句集，它内置了处理SQL注入的能力.

### HTTP与HTTPS的区别？

****HTTP和HTTPS使用的是完全不同的连接方式。****

****HTTP是不安全的，HTTPS是安全的。HTTP标准端口是80，而HTTPS标准端口是443；在网络模型中，HTTP工作于应用层，而HTTPS工作于传输层。而HTTP无需证书，HTTPS需要认证证书。****

### **说一下转发(Forward)和重定向(Redirect)的区别？**

****转发是服务器行为，重定向是客户端行为。****

****转发（Forword）**** 通过 RequestDispatcher 对象的forward（HttpServletRequest request,HttpServletResponse response）方法实现的。RequestDispatcher 可以通过HttpServletRequest 的 getRequestDispatcher()方法获得。跳转到所需页面。

****重定向（Redirect）**** 是利用服务器返回的状态码来实现的。客户端浏览器请求服务器的时候，服务器会返回一个状态码。服务器通过 HttpServletRequestResponse 的 setStatus(int status)方法设置状态码。如果服务器返回 301 或者 302，则浏览器会到新的网址重新请求该资源。

1. 从地址栏显示来说：forward 是服务器请求资源，服务器直接访问目标地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器。浏览器根本不知道服务器发送的内容从哪里来的，所以它的地址栏还是原来的地址。redirect 是服务端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址。所以地址栏显示的是新的 URL。
2. 从数据共享来说：forward：转发页面和转发到的页面可以共享 request 里面的数据。redirect：不能共享数据。
3. 从运用地方来说：forward：一般用于用户登陆的时候，根据角色转发到相应的模块。redirect：一般用于用户注销登陆时返回主页面和跳转到其它的网站等。
4. 从效率来说：forward：高。redirect：低。

### **在浏览器中输入 url 地址到显示主页的过程,整个过程会使用哪些协议**

总体来说分为以下几个过程:

1. DNS 解析
2. TCP 连接
3. 发送 HTTP 请求
4. 服务器处理请求并返回 HTTP 报文
5. 浏览器解析渲染页面
6. 连接结束

### **IP 地址与 MAC 地址的区别**

IP 地址是指互联网协议地址（Internet Protocol Address）IP Address 的缩写。IP 地址是 IP 协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。

MAC 地址又称为物理地址、硬件地址，用来定义网络设备的位置。网卡的物理地址通常是由网卡生产厂家写入网卡的，具有全球唯一性。MAC 地址用于在网络中唯一标示一个网卡，一台电脑会有一或多个网卡，每个网卡都需要有一个唯一的 MAC 地址。

### **HTTP 请求,响应报文格式**

HTTP 请求报文主要由请求行、请求头部、请求正文 3 部分组成。

HTTP 响应报文主要由状态行、响应头部、响应正文 3 部分组成。

### 什么是 jsp，什么是Servlet？jsp 和Servlet 有什么区别？

jsp 本质上就是一个 Servlet，它是 Servlet 的一种特殊形式（由 SUN 公司推出），每个 jsp 页面都是一个 servlet实例。

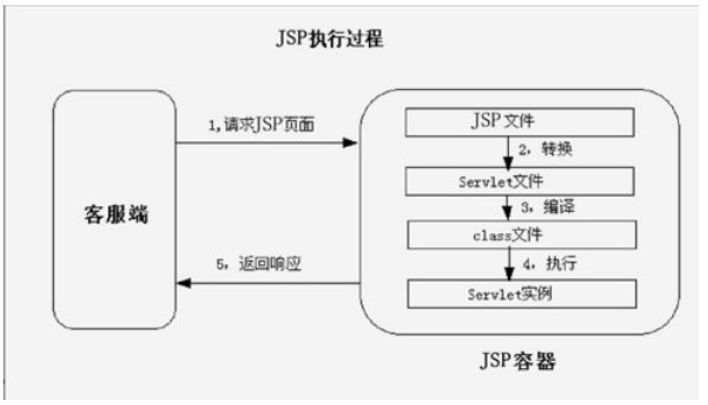
Servlet 是由 Java 提供用于开发 web 服务器应用程序的一个组件，运行在服务端，由 servlet 容器管理，用来生成动态内容。一个 servlet 实例是实现了特殊接口 Servlet 的 Java 类，所有自定义的 servlet 均必须实现 Servlet 接口。

**区别：**

jsp 是 html 页面中内嵌的 Java 代码，侧重页面显示；

Servlet 是 html 代码和 Java 代码分离，侧重逻辑控制，mvc 设计思想中 jsp 位于视图层，servlet 位于控制层。

Jsp 运行机制：如下图



JVM 只能识别 Java 类，并不能识别 jsp 代码！web 容器收到以.jsp 为扩展名的 url 请求时，会将访问请求交给tomcat 中 jsp 引擎处理，每个 jsp 页面第一次被访问时，jsp 引擎将 jsp 代码解释为一个 servlet 源程序，接着编译servlet 源程序生成.class 文件，再有 web 容器 servlet 引擎去装载执行 servlet 程序，实现页面交互。

### 谈谈你对 ajax 的认识？

Ajax 是一种创建交互式网页应用的的网页开发技术；Asynchronous JavaScript and XML”的缩写。

Ajax 的优势：

通过异步模式，提升了用户体验。

优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占用。

Ajax 引擎在客户端运行，承担了一部分本来由服务器承担的工作，从而减少了大用户量下的服务器负载。

Ajax 的最大特点：

可以实现局部刷新，在不更新整个页面的前提下维护数据，提升用户体验度。

### 说一下常用的 Linux 命令？

1. 目录切换命令

* cd ..（或cd../）： 切换到上一层目录
* cd /： 切换到系统根目录
* cd ~： 切换到用户主目录

1. 目录的操作命令(增删改查)

* mkdir 目录名称： 增加目录 rmdir 删除目录
* ls或者ll（ll是ls -l的别名，ll命令可以看到该目录下的所有目录和文件的详细信息）

1. 查看目录信息

* find 目录 参数： 寻找目录（查）
* mv 目录名称 新目录名称： 修改目录的名称（改）
* mv 目录名称 目录的新位置： 移动目录的位置---剪切（改）
* cp -r 目录名称 目录拷贝的目标位置： 拷贝目录（改），-r代表递归拷贝
* rm [-rf] 目录: 删除目录（删）

1. 文件的操作命令（增删改查）

* touch 文件名称: 文件的创建（增）
* cat/more/less/tail 文件名称 文件的查看（查）
* head -n 10 文件名 查看前十行
* tail -n 10 文件名 查看后十行
* vim 文件： 修改文件的内容（改）
* rm -rf 文件： 删除文件（删）

1. 压缩文件的操作命令

* tar -zcvf 打包压缩后的文件名 要打包压缩的文件
* tar [-xvf] x：代表解压

6. 修改文件/目录的权限的命令

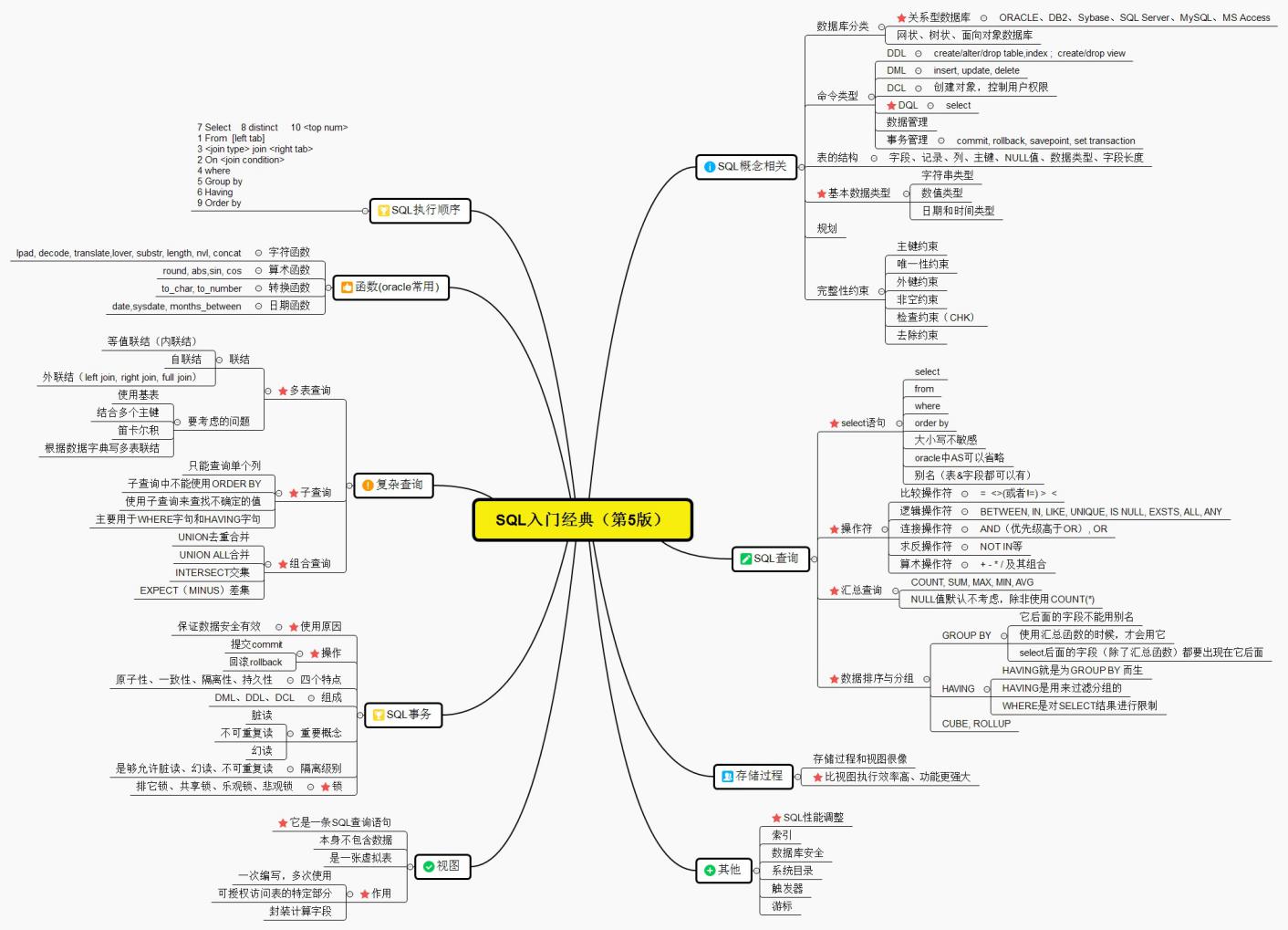
* Chmod

7. 其他常用命令

* pwd： 显示当前所在位置
* grep 要搜索的字符串 要搜索的文件 --color： 搜索命令，--color代表高亮显示
* ps -ef/ps -aux： 这两个命令都是查看正在运行进程状态，两者的区别是展示格式不同。
* 查看特定的进程：ps aux|grep redis 。如果直接用ps（（Process Status））命令，会显示所有进程的状态，通常结合grep命令查看某进程的状态。 ps -ef|grep tomcat （那一竖是“管道符号”）
* kill -9 进程的pid： 杀死进程（-9 表示强制终止。）先用ps查找进程，然后用kill杀掉。
* top：用于实时显示系统的动态信息。
* free：查看系统内存使用情况。

1. 网络通信命令

* 查看当前系统的网卡信息：ifconfig
* 查看与某台机器的连接情况：ping



# **数据库**

### 什么是MySQL?

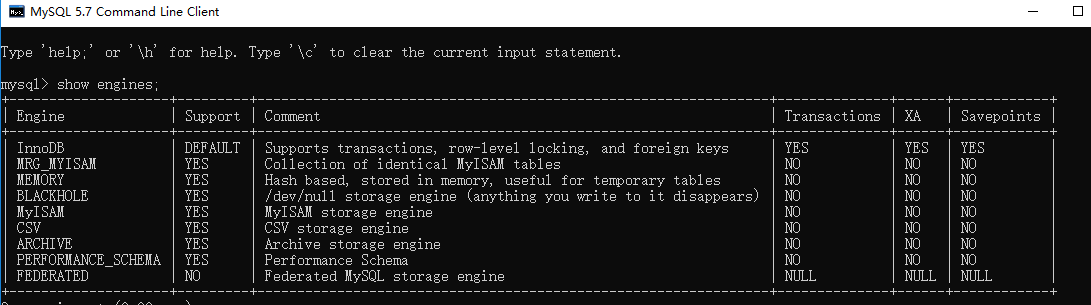
MySQL 是一种关系型数据库，在Java企业级开发中非常常用，因为 MySQL 是开源免费的，并且方便扩展。阿里巴巴数据库系统也大量用到了 MySQL，因此它的稳定性是有保障的。MySQL是开放源代码的，因此任何人都可以在 GPL(General Public License) 的许可下下载并根据个性化的需要对其进行修改。MySQL的默认端口号是****3306****。

### 说说存储引擎？

#### **一些常用命令**

****查看MySQL提供的所有存储引擎****

mysql> show engines;

[](https://camo.githubusercontent.com/34b24ff87145208403dce01554932f05bbe3f167/68747470733a2f2f6d792d626c6f672d746f2d7573652e6f73732d636e2d6265696a696e672e616c6979756e63732e636f6d2f323031392d362f6d7973716c2d656e67696e65732e706e67)

从上图我们可以查看出 MySQL 当前默认的存储引擎是InnoDB,并且在5.7版本所有的存储引擎中只有 InnoDB 是事务性存储引擎，也就是说只有 InnoDB 支持事务。

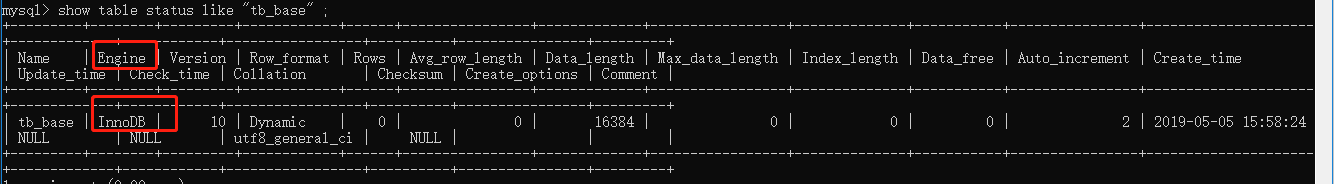
****查看MySQL当前默认的存储引擎****

我们也可以通过下面的命令查看默认的存储引擎。

mysql> show variables like '%storage\_engine%';

****查看表的存储引擎****

show table status like "table\_name" ;



### **MyISAM和InnoDB区别**

MyISAM是MySQL的默认数据库引擎（5.5版之前）。虽然性能极佳，而且提供了大量的特性，包括全文索引、压缩、空间函数等，但MyISAM不支持事务和行级锁，而且最大的缺陷就是崩溃后无法安全恢复。不过，5.5版本之后，MySQL引入了InnoDB（事务性数据库引擎），MySQL 5.5版本后默认的存储引擎为InnoDB。

大多数时候我们使用的都是 InnoDB 存储引擎，但是在某些情况下使用 MyISAM 也是合适的比如读密集的情况下。（如果你不介意 MyISAM 崩溃恢复问题的话）。

****两者的对比：****

1. ****是否支持行级锁**** : MyISAM 只有表级锁(table-level locking)，而InnoDB 支持行级锁(row-level locking)和表级锁,默认为行级锁。
2. ****是否支持事务和崩溃后的安全恢复： MyISAM**** 强调的是性能，每次查询具有原子性,其执行速度比InnoDB类型更快，但是不提供事务支持。但是****InnoDB**** 提供事务支持事务，外部键等高级数据库功能。 具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。
3. ****是否支持外键：**** MyISAM不支持，而InnoDB支持。

关于两者的总结

MyISAM 更适合读密集的表，而 InnoDB 更适合写密集的表。 在数据库做主从分离的情况下，经常选择 MyISAM 作为主库的存储引擎。

一般来说，如果需要事务支持，并且有较高的并发读取频率(MyISAM 的表锁的粒度太大，所以当该表写并发量较高时，要等待的查询就会很多了)，InnoDB 是不错的选择。如果你的数据量很大（MyISAM 支持压缩特性可以减少磁盘的空间占用），而且不需要支持事务时，MyISAM 是最好的选择。

### **字符集及校对规则？**

字符集指的是一种从二进制编码到某类字符符号的映射。校对规则则是指某种字符集下的排序规则。MySQL中每一种字符集都会对应一系列的校对规则。

MySQL采用的是类似继承的方式指定字符集的默认值，每个数据库以及每张数据表都有自己的默认值，他们逐层继承。比如：某个库中所有表的默认字符集将是该数据库所指定的字符集。

### 什么是事务?事务传播行为?

事务事务就是被绑定在一起作为一个逻辑工作单元的SQL 语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上有个节点。为了确保要么执行，要么不执行，就可以使用事务。

(ACID:四个特性. 原子性,一致性,隔离性,持久性.)

事务传播行为就是多个事务方法相互调用时，事务如何在这些方法间传播。当事务方法被另一个事务方法调用时，必须指定事务应该如何传播。

### 说说事物的四大特性(ACID)？



1. ****原子性（Atomicity）：**** 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（Consistency）：**** 执行事务前后，数据保持一致，多个事务对同一个数据读取的结果是相同的；
3. ****隔离性（Isolation）：**** 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（Durability）：**** 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。

### **说说事务的隔离级别？**

* ****脏读（Dirty read）:**** 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。
* ****丢失修改（Lost to modify）:**** 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。
* ****不可重复读（Unrepeatableread）:**** 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。
* ****幻读（Phantom read）:**** 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。

****不可重复读和幻读区别：****

不可重复读的重点是修改比如多次读取一条记录发现其中某些列的值被修改，幻读的重点在于新增或者删除比如多次读取一条记录发现记录增多或减少了。

### **说说分布式事务？**

随着业务规模不断地扩大，电商网站一般都要面临拆分之路。就是将原来一个单体应用拆分成多个不同职责的子系统。比如以前可能将面向用户、客户和运营的功能都放在一个系统里，现在拆分为订单中心、代理商管理、运营系统、报价中心、库存管理等多个子系统。

**拆分首先要面临的是什么呢？**

最开始的单体应用所有功能都在一起，存储也在一起。比如运营要取消某个订单，那直接去更新订单表状态，然后更新库存表就 ok 了。因为是单体应用，库在一起，这些都可以在一个事务里，由关系数据库来保证一致性。

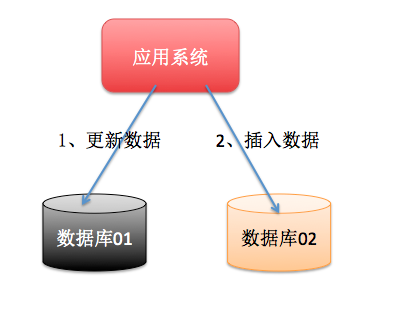
但拆分之后就不同了，不同的子系统都有自己的存储。比如订单中心就只管理自己的订单库，而库存管理也有自己的库。那么运营系统取消订单的时候就是通过接口调用等方式来调用订单中心和库存管理的服务了，而不是直接去操作库。这就涉及一个『分布式事务』的问题。

就是一次大的操作由不同的小操作组成，这些小的操作分布在不同的服务器上，且属于不同的应用，分布式事务需要保证这些小操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同数据库的数据一致性。

### 说说分布式事务的产生的原因？

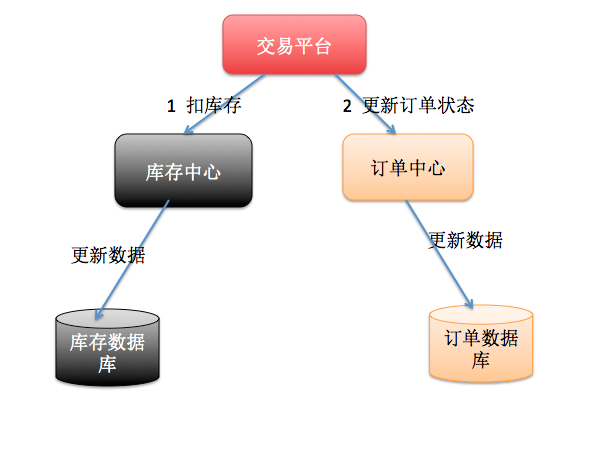
1、数据库分库分表

当数据库单表一年产生的数据超过1000W，那么就要考虑分库分表，具体分库分表的原理在此不做解释，以后有空详细说，简单的说就是原来的一个数据库变成了多个数据库。这时候，如果一个操作既访问01库，又访问02库，而且要保证数据的一致性，那么就要用到分布式事务。



2、应用SOA化

所谓的SOA化，就是业务的服务化。比如原来单机支撑了整个电商网站，现在对整个网站进行拆解，分离出了订单中心、用户中心、库存中心。对于订单中心，有专门的数据库存储订单信息，用户中心也有专门的数据库存储用户信息，库存中心也会有专门的数据库存储库存信息。这时候如果要同时对订单和库存进行操作，那么就会涉及到订单数据库和库存数据库，为了保证数据一致性，就需要用到分布式事务。



以上两种情况表象不同，但是本质相同，都是因为要操作的数据库变多了！

### 说说分布式事务的应用场景？

1、支付

最经典的场景就是支付了，一笔支付，是对买家账户进行扣款，同时对卖家账户进行加钱，这些操作必须在一个事务里执行，要么全部成功，要么全部失败。而对于买家账户属于买家中心，对应的是买家数据库，而卖家账户属于卖家中心，对应的是卖家数据库，对不同数据库的操作必然需要引入分布式事务。

2、在线下单

买家在电商平台下单，往往会涉及到两个动作，一个是扣库存，第二个是更新订单状态，库存和订单一般属于不同的数据库，需要使用分布式事务保证数据一致性。

### 说说常见的分布式事务解决方案？

1、基于XA协议的两阶段提交

XA是一个分布式事务协议。XA中大致分为两部分：事务管理器和本地资源管理器。其中本地资源管理器往往由数据库实现，比如Oracle、DB2这些商业数据库都实现了XA接口，而事务管理器作为全局的调度者，负责各个本地资源的提交和回滚。

XA协议比较简单，而且一旦商业数据库实现了XA协议，使用分布式事务的成本也比较低。但是，XA也有致命的缺点，那就是性能不理想，特别是在交易下单链路，往往并发量很高，XA无法满足高并发场景。XA目前在商业数据库支持的比较理想，在mysql数据库中支持的不太理想。

2、消息事务

所谓的消息事务就是基于消息中间件的两阶段提交，本质上是对消息中间件的一种特殊利用，它是将本地事务和发消息放在了一个分布式事务里，保证要么本地操作成功成功并且对外发消息成功，要么两者都失败，开源的RocketMQ就支持这一特性，

基于消息中间件的两阶段提交往往用在高并发场景下，将一个分布式事务拆成一个消息事务（A系统的本地操作+发消息）+B系统的本地操作，其中B系统的操作由消息驱动，只要消息事务成功，那么A操作一定成功，消息也一定发出来了，这时候B会收到消息去执行本地操作，如果本地操作失败，消息会重投，直到B操作成功，这样就变相地实现了A与B的分布式事务。

虽然上面的方案能够完成A和B的操作，但是A和B并不是严格一致的，而是最终一致的，我们在这里牺牲了一致性，换来了性能的大幅度提升。当然，这种玩法也是有风险的，如果B一直执行不成功，那么一致性会被破坏，具体要不要用，还是得看业务能够承担多少风险。

3、TCC编程模式

所谓的TCC编程模式，也是两阶段提交的一个变种。TCC提供了一个编程框架，将整个业务逻辑分为三块：Try、Confirm和Cancel三个操作。以在线下单为例，Try阶段会去扣库存，Confirm阶段则是去更新订单状态，如果更新订单失败，则进入Cancel阶段，会去恢复库存。总之，TCC就是通过代码人为实现了两阶段提交，不同的业务场景所写的代码都不一样，复杂度也不一样，因此，这种模式并不能很好地被复用。

总结

分布式事务，本质上是对多个数据库的事务进行统一控制，按照控制力度可以分为：不控制、部分控制和完全控制。不控制就是不引入分布式事务，部分控制就是各种变种的两阶段提交，包括上面提到的消息事务+最终一致性、TCC模式，而完全控制就是完全实现两阶段提交。部分控制的好处是并发量和性能很好，缺点是数据一致性减弱了，完全控制则是牺牲了性能，保障了一致性，具体用哪种方式，最终还是取决于业务场景。作为技术人员，一定不能忘了技术是为业务服务的，不要为了技术而技术，针对不同业务进行技术选型也是一种很重要的能力！

### **你有**了解 **mysql 的隔离级别吗？mysql 默认的隔离级别是什么？**

数据库事务的隔离级别有四种，隔离级别高的数据库的可靠性高，但并发量低，而隔离级别低的数据库可靠性低，但并发量高，系统开销小。

* ****READ-UNCOMMITTED(读取未提交)：**** 最低的隔离级别，允许读取尚未提交的数据变更，****可能会导致脏读、幻读或不可重复读****。
* ****READ-COMMITTED(读取已提交)：**** 允许读取并发事务已经提交的数据，****可以阻止脏读，但是幻读或不可重复读仍有可能发生****。
* ****REPEATABLE-READ(可重复读)：**** 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，****可以阻止脏读和不可重复读，但幻读仍有可能发生****。
* ****SERIALIZABLE(可串行化)：**** 最高的隔离级别，完全服从ACID的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，****该级别可以防止脏读、不可重复读以及幻读****。

| **隔离级别** | **脏读** | **不可重复读** | **幻影读** |
| --- | --- | --- | --- |
| READ-UNCOMMITTED | √ | √ | √ |
| READ-COMMITTED | × | √ | √ |
| REPEATABLE-READ | × | × | √ |
| SERIALIZABLE | × | × | × |

MySQL InnoDB 存储引擎的默认支持的隔离级别是 ****REPEATABLE-READ（可重读）****。我们可以通过SELECT @@tx\_isolation;命令来查看

mysql> SELECT @@tx\_isolation;

+-----------------+

| @@tx\_isolation |

+-----------------+

| REPEATABLE-READ |

+-----------------+

这里需要注意的是：与 SQL 标准不同的地方在于 InnoDB 存储引擎在 ****REPEATABLE-READ（可重读）**** 事务隔离级别下使用的是Next-Key Lock 锁算法，因此可以避免幻读的产生，这与其他数据库系统(如 SQL Server) 是不同的。所以说InnoDB 存储引擎的默认支持的隔离级别是 ****REPEATABLE-READ（可重读）**** 已经可以完全保证事务的隔离性要求，即达到了 SQL标准的 ****SERIALIZABLE(可串行化)**** 隔离级别。因为隔离级别越低，事务请求的锁越少，所以大部分数据库系统的隔离级别都是 ****READ-COMMITTED(读取提交内容)**** ，但是你要知道的是InnoDB 存储引擎默认使用 ****REPEAaTABLE-READ（可重读）**** 并不会有任何性能损失。

InnoDB 存储引擎在 ****分布式事务**** 的情况下一般会用到 ****SERIALIZABLE(可串行化)**** 隔离级别。

### 说说 锁机制与InnoDB锁算法？

****锁**** 是一种用来解决多个执行线程 ****访问共享资源**** 错误或数据不一致问题的工具。

****MyISAM和InnoDB存储引擎使用的锁：****

* MyISAM采用表级锁(table-level locking)。
* InnoDB支持行级锁(row-level locking)和表级锁,默认为行级锁
* 其实还有个页级锁(page-level locking)，数据库有三种锁。

****表级锁和行级锁对比：****

* ****表级锁：**** MySQL中锁定 ****粒度最大**** 的一种锁，对当前操作的整张表加锁，实现简单，资源消耗也比较少，加锁快，不会出现死锁。其锁定粒度最大，触发锁冲突的概率最高，并发度最低，MyISAM和 InnoDB引擎都支持表级锁。
* ****行级锁：**** MySQL中锁定 ****粒度最小**** 的一种锁，只针对当前操作的行进行加锁。 行级锁能大大减少数据库操作的冲突。其加锁粒度最小，并发度高，但加锁的开销也最大，加锁慢，会出现死锁。

****InnoDB存储引擎的锁的算法有三种：****

* Record lock：单个行记录上的锁
* Gap lock：间隙锁，锁定一个范围，不包括记录本身
* Next-key lock：record+gap 锁定一个范围，包含记录本身

### 说说大表优化？

当MySQL单表记录数过大时，数据库的CRUD性能会明显下降，一些常见的优化措施如下：

**1. 限定数据的范围**

务必禁止不带任何限制数据范围条件的查询语句。比如：我们当用户在查询订单历史的时候，我们可以控制在一个月的范围内；

**2. 读/写分离**

经典的数据库拆分方案，主库负责写，从库负责读；

**3. 垂直分区**

****根据数据库里面数据表的相关性进行拆分。**** 例如，用户表中既有用户的登录信息又有用户的基本信息，可以将用户表拆分成两个单独的表，甚至放到单独的库做分库。

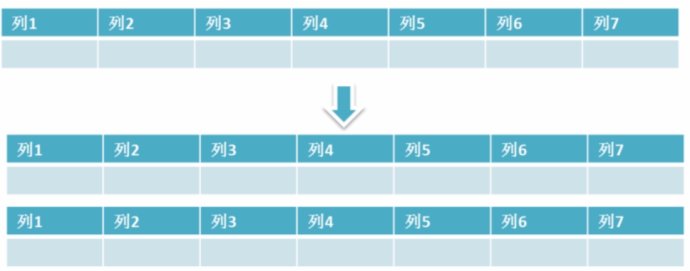
****简单来说垂直拆分是指数据表列的拆分，把一张列比较多的表拆分为多张表。**** 如下图所示，这样来说大家应该就更容易理解了。



* ****垂直拆分的优点：**** 可以使得列数据变小，在查询时减少读取的Block数，减少I/O次数。此外，垂直分区可以简化表的结构，易于维护。
* ****垂直拆分的缺点：**** 主键会出现冗余，需要管理冗余列，并会引起Join操作，可以通过在应用层进行Join来解决。此外，垂直分区会让事务变得更加复杂；

**4. 水平分区**

****保持数据表结构不变，通过某种策略存储数据分片。这样每一片数据分散到不同的表或者库中，达到了分布式的目的。 水平拆分可以支撑非常大的数据量。****



### SQL 的 **select** 语句完整的执行顺序？

SQL Select 语句完整的执行顺序：

1、from 子句组装来自不同数据源的数据；

2、where 子句基于指定的条件对记录行进行筛选；

3、group by 子句将数据划分为多个分组；

4、计算所有的表达式；

5、select 的字段；

6、使用 order by 对结果集进行排序。

### 大数据量的情况下,需要删除掉一批数据要怎么做?

select \* from person where id in (select id from person where id >10 limit 100000,10 ),此处子查询中使用了id的索引，可以大大降低查询时间；

### in走不走索引？可以用哪个关键字代替？

发现 MySQL 4.1 以上版本的 IN 是走索引的， 但4.0及其以下版本是不走索引的。

原来是这样，4.0以下确实是不走索引的，难怪之前有人说。

可以用exist代替in，exit的执行效率要高。

in是子查询。而exist是通过id关联，组成一张虚表，操作的是同一张表。

### 什么是主键?什么是外键?

主键是表格里的(一个或多个)字段，只用来定义表格里的行;主键里的值总是唯一的。外键是一个用来建立两个表格之间关系的约束。这种关系一般都涉及一个表格里的主键字段与另外一个表格(尽管可能是同一个表格)里的一系列相连的字段。那么这些相连的字段就是外键。

### 什么是触发器?SQL Server 2000有什么不同类型的触发器?

触发器是一种专用类型的存储过程，它被捆绑到SQL Server 2000的表格或者视图上。在SQL Server 2000里，有INSTEAD-OF和AFTER两种触发器。INSTEAD-OF触发器是替代数据操控语言(Data Manipulation Language，DML)语句对表格执行语句的存储过程。例如，如果我有一个用于TableA的INSTEAD-OF-UPDATE触发器，同时对这个表格执行一个更新语句，那么INSTEAD-OF-UPDATE触发器里的代码会执行，而不是我执行的更新语句则不会执行操作。

AFTER触发器要在DML语句在数据库里使用之后才执行。这些类型的触发器对于监视发生在数据库表格里的数据变化十分好用。

### 触发器的作用？

  答：触发器是一中特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

### 什么是存储过程？用什么来调用？

答：存储过程是一个预编译的SQL 语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次SQL ，使用存储过程比单纯SQL 语句执行要快。可以用一个命令对象来调用存储过程。

### 什么是索引?SQL Server 2000里有什么类型的索引?

简单地说，索引是一个数据结构，用来快速访问数据库表格或者视图里的数据。

在SQL Server里，它们有两种形式:聚集索引和非聚集索引。聚集索引在索引的叶级保存数据。这意味着不论聚集索引里有表格的哪个(或哪些)字段，这些字段都会按顺序被保存在表格。由于存在这种排序，所以每个表格只会有一个聚集索引。非聚集索引在索引的叶级有一个行标识符。这个行标识符是一个指向磁盘上数据的指针。它允许每个表格有多个非聚集索引。

### 索引的作用？和它的优点缺点是什么？

答：索引就一种特殊的查询表，数据库的搜索引擎可以利用它加速对数据的检索。它很类似与现实生活中书的目录，不需要查询整本书内容就可以找到想要的数据。索引可以是唯一的，创建索引允许指定单个列或者是多个列。缺点是它减慢了数据录入的速度，同时也增加了数据库的尺寸大小。

### 如何创建索引？

1. 通常，创建表的时候就能为表创建索引。 例如，以下语句创建一个新表，并创建了是由两列c2和c3组成的索引。

CREATE TABLE t(

c1 INT PRIMARY KEY,

c2 INT NOT NULL,

c3 INT NOT NULL,

c4 VARCHAR(10),

INDEX (c2,c3) );

1. 要为列或一组列添加索引，可以使用**CREATE INDEX**语句，如下所示：

CREATE INDEX index\_name ON table\_name (column\_list)

要为列创建索引，要指定索引名称，索引所属的表以及列。

例如，要为列c4添加新索引，请使用以下语句：

CREATE INDEX idx\_c4 ON t(c4);

### Mysql 索引主要使用的哪两种数据结构？

哈希索引：对于哈希索引来说，底层的数据结构就是哈希表，简单地说，哈希索引就是采用一定的哈希算法，把键值换算成新的哈希值，检索时不需要类似B+树那样从根节点到叶子节点逐级查找，只需一次哈希算法即可立刻定位到相应的位置，速度非常快。

BTree 索引：Mysql 的 BTree 索引使用的是 B 树中的 B+Tree。B+树是一个平衡的多叉树，从根节点到每个叶子节点的高度差值不超过1，而且同层级的节点间有指针相互链接。

在B+树上的常规检索，从根节点到叶子节点的搜索效率基本相当，不会出现大幅波动，而且基于索引的顺序扫描时，也可以利用双向指针快速左右移动，效率非常高。

### B+树索引和哈希索引的区别？

1. 如果是等值查询，那么哈希索引明显有绝对优势，因为只需要经过一次算法即可找到相应的键值；当然了，这个前提是，键值都是唯一的。如果键值不是唯一的，就需要先找到该键所在位置，然后再根据链表往后扫描，直到找到相应的数据；
2. 从示意图中也能看到，如果是范围查询检索，这时候哈希索引就毫无用武之地了，因为原先是有序的键值，经过哈希算法后，有可能变成不连续的了，就没办法再利用索引完成范围查询检索；
3. 同理，哈希索引也没办法利用索引完成排序，以及like ‘xxx%’ 这样的部分模糊查询（这种部分模糊查询，其实本质上也是范围查询）；
4. 在HEAP表中（内存表使用哈希散列索引把数据保存在内存中，因此具有极快的速度），如果存储的数据重复度很低（也就是说基数很大），对该列数据以等值查询为主，没有范围查询、没有排序的时候，特别适合采用哈希索引

例如这种SQL：  
SELECT … FROM t WHERE C1 = ?; — 仅等值查询

1. B+树索引的关键字检索效率比较平均，不像B树那样波动幅度大，在有大量重复键值情况下，哈希索引的效率也是极低的，因为存在所谓的哈希碰撞问题。

在大多数场景下，都会有范围查询、排序、分组等查询特征，用B+树索引就可以了。

### 什么是覆盖索引?

如果一个索引包含（或者说覆盖）所有需要查询的字段的值，我们就称 之为“覆盖索引”。我们知道在 InnoDB 存储引擎中，如果不是主键索引，叶子节点存储的是主键+列值。最终还是要“回表”，也就是要通过主键再查找一次,这样就会比较慢。覆盖索引就是把要查询出的列和索引是对应的，不做回表操作！

### 索引这么多优点，为什么不对表中的每一个列创建一个索引呢？

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

### 索引是如何提高查询速度的？

将无序的数据变成相对有序的数据（就像查目录一样）。

### 说一下在什么地方创建索引？

**哪些列适合建立索引**

1. 在经常需要搜索的列上创建索引，增加搜索速度。
2. 在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构。
3. 在经常用在连接的列上，这些列主要是外键，可以加快连接速度。
4. 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，所以其指定的范围是连续的。
5. 在经常需要排列的列上创建索引，因为索引已经排序，可以加快查询的速度。
6. 在经常需要使用WHERE子句的列上创建索引，加快条件的判断速度。

**哪些列不利于创建索引**

1. 对于查询中很少使用或者参考列不应该创建索引。这是因为，既然这些列很少使用，那么创建和不创建都没有效果，反而由于增加了索引，降低了系统的维护和增大了空间的需求。
2. 对于那些是有很少数据值的列不应添加索引。由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据很大比例，即需要在表中搜索的数据行比例很大，增加缩影并不能明显的加快搜索速度。
3. 对于那些定义为text,image,bit数据类型的列不应添加索引，这是因为这些数据要么太大，要么取值很小。
4. 当修改性能大于检索性能的时候不应添加索引。这是因为，修改性能和检索性能是相互矛盾的，当增加索引时，会提高检索性能，但是会降低修改性能。当减小索引时，会提高修改性能，降低索引性能。因此，当修改性能远远大于检索性能时，不应该添加索引。

### 性别适合创建索引吗？名字和地址呢？

重复率高的字段不适合建索引。性别不适合，名字和地址可以！

当然凡事不是绝对，如果把性别字段设为表的聚集索引，那么就肯定能加快大约一半该字段的查询速度了。聚集索引指的是表本身中数据按哪个字段的值来进行排序。因此，聚集索引只能有一个，而且使用聚集索引不会付出额外IO开销。当然你得能舍得把聚集索引这么宝贵资源用到性别字段上。

### Like模糊查询是否走索引？

1. 后通配 走索引

前通配 走全表

例如：如果是用like '数学%' --这种模糊查询的是可以走范围索引的，如果开头有%号是不走索引的。

### mysql 索引失效的场景有哪些？

1、where条件用in或or 不会走索引；

2、 WHERE 条件里使用了函数（如： WHERE DAY(colu mn)=…）， MYSQL 将无法使用索引。

3、WHERE 条件里有不等于号（WHERE column!=…），MYSQL 将无法使用索引。

### 什么是内存泄漏？

答：一般我们所说的内存泄漏指的是堆内存的泄漏。堆内存是程序从堆中为其分配的，大小任意的，使用完后要显示释放内存。当应用程序用关键字new 等创建对象时，就从堆中为它分配一块内存，使用完后程序调用free 或者delete 释放该内存，否则就说该内存就不能被使用，我们就说该内存被泄漏了。

### 什么叫视图？游标是什么？

答：视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

  游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

### 你能向我简要叙述一下SQL Server 2000中使用的一些数据库对象吗?

包括这样一些对象:表格、视图、用户定义的函数，以及存储过程;如果他们还能够提到像触发器这样的对象就更好了。

### 子查询？

即一个select语句中嵌套了另外的一个或者多个select语句

### union all和union的区别？

如果我们需要将两个select语句的结果作为一个整体显示出来，我们就需要用到union或者union all关键字。

union和union all的区别是,

union会自动压缩多个结果集合中的重复结果，同时进行默认规则的排序；

union all则将所有的结果全部显示出来，不管是不是重复。不进行排序；

### 内连接查询, 左外连接讲一下?

内连接查询

查询多表中一定有关联的数据，叫内连接查询。

左外连接

查询出来左表的全部数据，以及右表相关联的数据表与表之间。

### 说说MySQL高性能优化规范建议？

数据库命令规范

•所有数据库对象名称必须使用小写字母并用下划线分割；

•所有数据库对象名称禁止使用 MySQL 保留关键字（如果表名中包含关键字查询时，需要将其用单引号括起来）；

•数据库对象的命名要能做到见名识意，并且最后不要超过 32 个字符；

•临时库表必须以 tmp\_为前缀并以日期为后缀，备份表必须以 bak\_为前缀并以日期 (时间戳) 为后缀；

•所有存储相同数据的列名和列类型必须一致（一般作为关联列，如果查询时关联列类型不一致会自动进行数据类型隐式转换，会造成列上的索引失效，导致查询效率降低）。

数据库基本设计规范

1. 所有表必须使用 Innodb 存储引擎

没有特殊要求（即 Innodb 无法满足的功能如：列存储，存储空间数据等）的情况下，所有表必须使用 Innodb 存储引擎（MySQL5.5 之前默认使用 Myisam，5.6 以后默认的为 Innodb）。

Innodb 支持事务，支持行级锁，更好的恢复性，高并发下性能更好。

2. 数据库和表的字符集统一使用 UTF8

兼容性更好，统一字符集可以避免由于字符集转换产生的乱码，不同的字符集进行比较前需要进行转换会造成索引失效，如果数据库中有存储 emoji 表情的需要，字符集需要采用 utf8mb4 字符集。

3. 所有表和字段都需要添加注释

使用 comment 从句添加表和列的备注，从一开始就进行数据字典的维护

4. 尽量控制单表数据量的大小,建议控制在 500 万以内。

500 万并不是 MySQL 数据库的限制，过大会造成修改表结构，备份，恢复都会有很大的问题。

可以用历史数据归档（应用于日志数据），分库分表（应用于业务数据）等手段来控制数据量大小

5. 谨慎使用 MySQL 分区表

分区表在物理上表现为多个文件，在逻辑上表现为一个表；

谨慎选择分区键，跨分区查询效率可能更低；

建议采用物理分表的方式管理大数据。

6.尽量做到冷热数据分离,减小表的宽度

MySQL 限制每个表最多存储 4096 列，并且每一行数据的大小不能超过 65535 字节。

减少磁盘 IO,保证热数据的内存缓存命中率（表越宽，把表装载进内存缓冲池时所占用的内存也就越大,也会消耗更多的 IO）；

更有效的利用缓存，避免读入无用的冷数据；

经常一起使用的列放到一个表中（避免更多的关联操作）。

**7. 禁止在数据库中存储图片,文件等大的二进制数据**

通常文件很大，会短时间内造成数据量快速增长，数据库进行数据库读取时，通常会进行大量的随机 IO 操作，文件很大时，IO 操作很耗时。

通常存储于文件服务器，数据库只存储文件地址信息

### 说说SQL语句优化技巧和表设计优化？

-- 针对SQL语句的优化：

1、使用group by 分组后，还会排序，可能会降低速度，在group by 后面增加 order by null 就可以防止排序.

explain select \* from emp group by deptno order by null;

2、减少使用 \* ，用具体的字段列表代替“\*”。

3、减少使用不含字段列表的 INSERT 语句

如： insert into values ('a','b','c');

应使用： insert into t(c1,c2,c3) values ('a','b','c');

4、减少子查询，使用关联查询（left join,right join...）替代。

**子查询性能差的原因：**

子查询的结果集无法使用索引，通常子查询的结果集会被存储到临时表中，不论是内存临时表还是磁盘临时表都不会存在索引，所以查询性能会受到一定的影响。特别是对于返回结果集比较大的子查询，其对查询性能的影响也就越大。

由于子查询会产生大量的临时表也没有索引，所以会消耗过多的 CPU 和 IO 资源，产生大量的慢查询。

5、减少使用IN或者NOT IN ,否则会导致全表扫描，使用exists，not exists或者关联查询语句替代

6、or 的查询尽量用 union或者union all 代替  
(在确认没有重复数据或者不用剔除重复数据时，union all会更好)。

7、应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

select id from t where num is null

二、表设计优化：

1、建表的时候能使用数字类型的字段就使用数字类型（type,status...），数字类型的字段作为条件查询比字符串的快。

2、对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

3、最好不要给数据库留 NULL，尽可能的使用 NOT NULL 填充数据库。备注、描述、评论之类的可以设置为 NULL，其他的，最好不要使用 NULL。

4、用varchar代替char，如果有null值就不会占用空间。

不要以为 NULL 不需要空间，比如：char(100) 型，在字段建立时，空间就固定了， 不管是否插入值（NULL 也包含在内），都是占用 100 个字符的空间的，如果是 varchar 这样的变长字段， null 不占用空间。

可以在 num 上设置默认值 0，确保表中 num 列没有 null 值，然后这样查询：

select id from t where num = 0.

### **有了解过大数据层面的分库分表吗？Mysql执行计划有什么作用？**

分库

通过 Mycat 结点来管理不同服务器上的数据库，每个表最多存 500 万条记录分表垂直切割，水平切割。

MySql的执行计划 提供了 EXPLAIN 语法用来进行查询分析，在 SQL 语句前加一个"EXPLAIN"即可。 mysql 中的 explain 语法可以帮助我们改写查询，优化表的结构和索引的设置，从而最大地提高查询效率。

### **查询某一个类别有多少本书的sql语句？和在最新列表展示最新上架的书籍的sql语句？**

Select 类别，count（\*）from 书籍表 group by 某一类别。

Select 类别，count（\*）from 书籍表 group by 某一时间段。

# **框架**

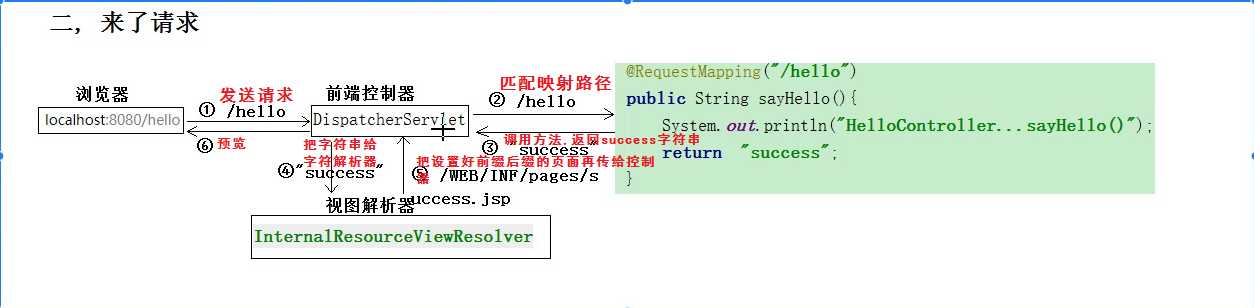
## Spring MVC

### 什么是Spring MVC ？简单介绍下你对springMVC的理解?

Spring MVC是一个实现了MVC设计模式的轻量级Web框架，并且它天生与 Spring 框架集成。把后端项目分为Controller层(控制层，返回数据给前台页面)、 Service层（处理业务）、Dao层（数据库操作）、Entity层（实体类）。把复杂的web应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

### [SpringMvc框架的工作流程](https://www.cnblogs.com/glorywzm/p/6503141.html)？

SpringMVC核心处理流程：



****流程说明：****

1. 客户端（浏览器）发送请求，直接请求到 DispatcherServlet。
2. DispatcherServlet 根据请求信息调用 HandlerMapping，解析请求对应的 Handler。
3. 解析到对应的 Handler（也就是我们平常说的 Controller 控制器）后，开始由 HandlerAdapter 适配器处理。
4. HandlerAdapter 会根据 Handler 来调用真正的处理器开处理请求，并处理相应的业务逻辑。
5. 处理器处理完业务后，会返回一个 ModelAndView 对象，Model 是返回的数据对象，View 是个逻辑上的 View。
6. ViewResolver 会根据逻辑 View 查找实际的 View。
7. DispaterServlet 把返回的 Model 传给 View（视图渲染）。
8. 把 View 返回给请求者（浏览器）

### SpringMVC前端到后端的servlet是啥？

是DispatcherServlet。

### SpringMVC怎么样设定重定向和转发的？

1. 转发：在返回值前面加"forward:"，譬如"forward:user.do?name=method4"
2. 重定向：在返回值前面加"redirect:"，譬如"redirect:http://www.baidu.com"

### 如何开启注解处理器和适配器？

我们在项目中一般会在 springmvc.xml 中通过开启 <mvc:annotation-driven>来实现注解处理器和适配器的开启。

### SpringMVC 传参的的时候要注意什么?

如果是基本类型或者 String 类型：

要求我们的参数名称必须和控制器中方法的形参名称保持一致。 (严格区分大小写) .

如果是 POJO 类型( 实际就是普通JavaBeans,使用POJO名称是为了避免和EJB混淆起来)，或者它的关联对象：

要求表单中参数名称和 POJO 类的属性名称保持一致。并且控制器方法的参数类型是 POJO 类型 .

如果是集合类型,有两种方式： 第一种：要求集合类型的请求参数必须在 POJO 中。在表单中请求参数名称要和 POJO 中集合属性名称相同。给 List 集合中的元素赋值， 使用下标。给 Map 集合中的元素赋值， 使用键值对。第二种：接收的请求参数是 json 格式数据。需要借助一个注解实现 。

### SpringMVC常用注解和参数绑定常用的注解？

@Component控制类 @Service服务类 @Repository通常用于数据类

@RequestMapping 处理请求地址映射，可用于类或方法上。

@Autowired 用于依赖注入。

请求参数必须和形参类型一致.如果不一致我们可以加个注解@RequestParam().

### springMVC常用的传参注解？

在springMVC的Controller中常用到

@RequestParam,@RequestBody,@ResponseBody和@PathVariable。

RequestParam

RequestParam注解通过名称获取一个请求参数，类似于request.getParameter()，如果方法的参数类型不一致，会自动进行格式转换。

RequestBody

RequestBody将request请求的整个body转换为对象。XHR(XMLHttpRequest)不允许get生成body，所以只有Post请求才会有request body，body的格式有form,json,xml等。不能和RequestParam混合使用。如下：

public void updateUser(@RequestBody User user){

}

RequestBody通过HttpMessageConverter将request body转换为对象。

ResponseBody

和RequestBody相似，用于将一个对象转换成json字符串并写入response中。

public @responseBody User getUser(Integer Id){

returnnew User("zhangsan")

}

PathVariable

转换一个请求路径中的变量，如：

@RequestMapping(path = "/{day}", method = RequestMethod.GET)

Public Map<String, Appointment> getForDay(@PathVariable

@DateTimeFormat(iso=ISO.DATE) Date day, Model model) {

return appointmentBook.getAppointmentsForDay(day);

}

## Spring

### 谈谈你对 Spring 的理解？

Spring 是一种轻量级开发框架，旨在提高开发人员的开发效率以及系统的可维护性。他解决的是业务逻辑层和其他各层的松耦合问题,因此它将面向接口的编程思想贯穿整个系统应用.Spring是一个分层的一站式轻量级开源框架。

### 说说Spring 的主要核心？

1. 控制反转（IOC），作用是将对象的创建反转给spring框架来创建和管理。削减计算机程序的耦合(重要的是配置与使用分离的原则，解除代码中的依赖关系，这个思想才是IOC的精髓所在)。

IOC容器是工厂模式。

2. 依赖注入（DI）：IOC的别名，就是由IOC容器在运行期间，动态地将某种依赖关系注入到对象之中。

3. 面向切面编程（AOP）：就是把我们程序重复的代码抽取出来，在需要执行的时候，使用动态代理的技术，在不修改源码的基础上，对我们的已有方法进行增强。

### Spring如何创建bean对象？

有三种方式。（在Google markups书签下有详细说明）。

1. 通过构造函数创建对象。二、通过静态方法创建对象。三、通过工厂方法创建对象。

### IOC里的Bean是不是线程安全的？

是不是线程安全跟IOC没有关系，我们想让它是就是，不是就不是。

### IOC跟DI是不是一样的？

不是同一类东西，因为IOC是对象创建。DI是controller写了service层的属性，我们先要去写service层的实例，然后在xml文件中开启注解扫描。扫描到controller层下有autowired这个注解，它才会把实例注入到在controller层的service属性上。

### SpringIOC反转的是什么？

​ 它反转的是对象的创建工作。 平常我们都是使用new 的方式来直接创建对象，现在有了spring之后，我们就再也不new对象了，而是把对象创建的工作交给spring容器去维护。我们只需要问spring容器要对象即可。

### IOC的优缺点是什么？

优点：

1.资源集中管理，实现资源的可配置和易管理。

2.降低了程序的耦合度。

缺点：

1.对象是通过反射机制实例化出来的，因此对系统的性能有一定的影响。

2.创建对象的流程变得复杂。

### 说说Spring的优点？

1.方便解耦，简化开发

​ 通过Spring提供的IoC容器，我们可以将对象之间的依赖关系交由Spring进行控制，避免硬编码所造成的过度程序耦合。有了Spring，用户不必再为单实例模式类、属性文件解析等这些很底层的需求编写代码，可以更专注于上层的应用。

2.AOP编程的支持

​ 通过Spring提供的AOP功能，方便进行面向切面的编程，许多不容易用传统OOP实现的功能可以通过AOP轻松应付。

3.声明式事务的支持

​ 在Spring中，我们可以从单调烦闷的事务管理代码中解脱出来，通过声明式方式灵活地进行事务的管理，提高开发效率和质量。

4.方便程序的测试

​ 可以用非容器依赖的编程方式进行几乎所有的测试工作，在Spring里，测试不再是昂贵的操作，而是随手可做的事情。例如：Spring对Junit4支持，可以通过注解方便的测试Spring程序。

5.方便集成各种优秀框架

​ Spring不排斥各种优秀的开源框架，相反，Spring可以降低各种框架的使用难度，Spring提供了对各种优秀框架（如Struts,Hibernate、Hessian、Quartz）等的直接支持。

6.降低Java EE API的使用难度

​ Spring对很多难用的Java EE API（如JDBC，JavaMail，远程调用等）提供了一个薄薄的封装层，通过Spring的简易封装，这些Java EE API的使用难度大为降低。

### **Spring 框架中用到了哪些设计模式？**

* ****工厂设计模式**** : Spring使用工厂模式通过 BeanFactory、ApplicationContext 创建 bean 对象。
* ****代理设计模式**** : Spring AOP 功能的实现。
* ****单例设计模式**** : Spring 中的 Bean 默认都是单例的。
* ****模板方法模式**** : Spring 中 jdbcTemplate、hibernateTemplate 等以 Template 结尾的对数据库操作的类，它们就使用到了模板模式。
* ****包装器设计模式**** : 我们的项目需要连接多个数据库，而且不同的客户在每次访问中根据需要会去访问不同的数据库。这种模式让我们可以根据客户的需求能够动态切换不同的数据源。
* ****观察者模式:**** Spring 事件驱动模型就是观察者模式很经典的一个应用。
* ****适配器模式**** :Spring AOP 的增强或通知(Advice)使用到了适配器模式、spring MVC 中也是用到了适配器模式适配Controller。
* ......

### IOC实现原理？/什么是Ioc容器？/原理实现过程？

控制反转IOC是通过反射机制实现的。

可以把IOC容器的工作模式看做是工厂模式的升华，把IOC容器看作是一个工厂，你只需要在 spring 配置文件中配置对应的 bean 以及设置相关的属性，让 spring 容器来生成类的实例对象以及管理对象。

在 spring 容器启动的时候，spring 会把你在配置文件中配置的 bean 都初始化好，然后在你需要调用的时候，就把它已经初始化好的那些 bean 分配给你需要调用这些 bean 的类。也就是把工厂和对象生成这两者独立分隔开来，目的就是提高灵活性和可维护性。

##### **工厂模式解耦思路？/什么是工厂？**

​ 在实际开发中我们可以把三层的对象都使用配置文件配置起来，当启动服务器应用加载的时候， 让一个类中的方法通过读取配置文件，把这些对象创建出来并存起来。在接下来的使用的时候，直接拿过来用就好了。那么，这个读取配置文件， 创建和获取三层对象的类就是工厂。

Javabean简单的讲就是实体类,用来封装对象,这个类里面全部都是属性值,和get,set方法。

### **请描述一下 Spring 的事务？**

****Spring支持两种方式的事务管理：****

****编程式事务管理： 通过写代码手动管理事务，实际应用中很少使用；****

****使用XML配置声明式事务： 推荐使用（代码侵入性最小），实际是通过AOP实现****。

声明式事务管理的定义：用在 Spring 配置文件中声明式的处理事务来代替代码式的处理事务。这样的好处是，事务管理不侵入开发的组件，具体来说，业务逻辑对象就不会意识到正在事务管理之中。如果想要改变事务管理策划的话，也只需要在定义文件中重新配置即可，这样维护起来极其方便。

****Spring并不直接管理事务，而是提供了多种事务管理器。****Spring事务管理器的接口是： **PlatformTransactionManager** ，通过这个接口，Spring为各个平台如JDBC、Hibernate等都提供了对应的事务管理器，但是具体的实现就是各个平台自己的事情了。

### AOP **执行原理简单说下？**

Spring AOP 底层使用动态代理，实现有两种方案。 JDK的动态代理和cglib 的动态代理

​ 使用JDK的动态代理。这一种主要是针对有接口实现的情况。 它的底层是创建接口的实现代理类，实现扩展功能。也就是我们要增强的这个类，实现了某个接口，那么我就可以使用这种方式. 而另一种方式cglib 的动态代理，这种主要是针对没有接口的方式，那么它的底层是创建目标类的子类，实现扩展功能.

### **说说AOP应用场景？**

场景一： 记录日志 ；

场景二： 监控方法运行时间 （监控性能） ；

场景三： 权限控制 ；

场景四： 缓存优化 （第一次调用查询数据库，将查询结果放入内存对象，第二次调用， 直接从内存对象返回，不需要查询数据库 ） ；

场景五： 事务管理 （调用方法前开启事务， 调用方法后提交关闭事务 ）。

### AOP 在事务管理方面是怎么实现的？

AOP 在事务管理方面，Spring 使用 AOP 来完成事务管理，有声明式事务和 编程式事务两种方式。编程式事务是自己写事务处理的类，然后调用。声明式事务是在配置文件中配置，一般搭配在框架里面使用！

开发中，方便代码编写，很多时候都是在 spring 配置文件中配置事务管理器并开启事务控制注解。在业务类或业务类方法中添加@Transactional 实现事务控制。

### **Transactional(rollbackFor = Exception.class)注解了解吗？**

我们知道：Exception分为运行时异常RuntimeException和非运行时异常。事务管理对于企业应用来说是至关重要的，即使出现异常情况，它也可以保证数据的一致性。

当@Transactional注解作用于类上时，该类的所有 public 方法将都具有该类型的事务属性，同时，我们也可以在方法级别使用该标注来覆盖类级别的定义。如果类或者方法加了这个注解，那么这个类里面的方法抛出异常，就会回滚，数据库里面的数据也会回滚。

在@Transactional注解中如果不配置rollbackFor属性,那么事物只会在遇到RuntimeException的时候才会回滚,加上rollbackFor=Exception.class,可以让事物在遇到非运行时异常时也回滚。

### Spring Aop代理是怎么选择的？

在spring中，框架会根据目标类是否实现了接口来决定采用哪种动态代理的方式。有接口,用jdk方式.没有接口,用cglib方式.但这不是我们关心的方式.它自己会切换.

### 在 Spring AOP 中，关注点和横切关注的区别是什么？

关注点是应用中一个模块的行为，一个关注点可能会被定义成一个我们想实现的一个功能。横切关注点是一个关注点，此关注点是整个应用都会使用的功能，并影响整个应用，比如日志，安全和数据传输，几乎应用的每个模块都需要的功能。因此这些都属于横切关注点。

### 什么是连接点？

被拦截到的点，因为 Spring 只支持方法类型的连接点，所以在 Spring 中连接点指的就是被拦截到的方法，实际上连接点还可以是字段或者构造器。

### 什么是切点？

切入点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。

### 什么是目标对象?

被一个或者多个切面所通知的对象。它通常是一个代理对象。也指被通知（advised）对象。

### 什么是代理?

代理是通知目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。

### Spring 的通知是什么？有哪几种类型？

通知是个在方法执行前或执行后要做的动作，实际上是程序执行时要通过 SpringAOP 框架触发的代码段。

Spring 切面可以应用五种类型的通知：

1）before：前置通知，在一个方法执行前被调用。

2）after: 在方法执行之后调用的通知，无论方法执行是否成功。

3）after-returning: 仅当方法成功完成后执行的通知。

4）after-throwing: 在方法抛出异常退出时执行的通知。

5）around: 在方法执行之前和之后调用的通知。

### Spring 配置文件有什么作用？

Spring 配置文件是个 XML 文件，这个文件包含了类信息，描述了如何配置它们，以及如何相互调用。

### 怎样开启注解装配？

注解装配在默认情况下是不开启的，为了使用注解装配，我们必须在 Spring 配 置 文 件 中 配置 <context:annotation-config/>元素。

### **简单介绍一下 bean;知道 Spring 的 bean 的作用域与生命周期吗?**

在 Spring 中，那些组成应用程序的主体及由 Spring IOC 容器所管理的对象，被称之为 bean。简单地讲，bean 就是由 IOC 容器初始化、装配及管理的对象，除此之外，bean 就与应用程序中的其他对象没有什么区别了。而 bean 的定义以及 bean 相互间的依赖关系将通过配置元数据来描述。

### **@**Component **和 @Bean 的区别是什么？**

1. 作用对象不同: @Component 注解作用于类，而@Bean注解作用于方法。
2. @Component通常是通过类路径扫描来自动侦测以及自动装配到Spring容器中（我们可以使用 @ComponentScan 注解定义要扫描的路径从中找出标识了需要装配的类自动装配到 Spring 的 bean 容器中）。@Bean 注解通常是我们在标有该注解的方法中定义产生这个 bean,@Bean告诉了Spring这是某个类的示例，当我需要用它的时候还给我。
3. @Bean 注解比 Component 注解的自定义性更强，而且很多地方我们只能通过 @Bean 注解来注册bean。比如当我们引用第三方库中的类需要装配到 Spring容器时，则只能通过 @Bean来实现。

### **将一个类声明为**Spring**的 bean 的注解有哪些?**

我们一般使用 @Autowired 注解自动装配 bean，要想把类标识成可用于 @Autowired 注解自动装配的 bean 的类,采用以下注解可实现：

* @Component ：通用的注解，可标注任意类为 Spring 组件。如果一个Bean不知道属于哪个层，可以使用@Component 注解标注。
* @Repository : 对应持久层即 Dao 层，主要用于数据库相关操作。
* @Service : 对应服务层，主要涉及一些复杂的逻辑，需要用到 Dao层。
* @Controller : 对应 Spring MVC 控制层，主要用户接受用户请求并调用 Service 层返回数据给前端页面。

## **MyBatis**

### **说说MyBatis**框架？

​ **mybatis 是一个优秀的基于 java 的持久层框架，它内部封装了 jdbc，==使开发者只需要关注 sql 语句本身，而不需要花费精力去处理加载驱动、创建连接、创建 statement 等繁杂的过程。**

​ mybatis 通过xml 或注解的方式将要执行的各种statement 配置起来，并通过java 对象和statement 中sql的动态参数进行映射生成最终执行的 sql 语句，最后由 mybatis 框架执行 sql并将结果映射为 java 对象并返回。

​ 采用 ORM 思想解决了实体和数据库映射的问题，对jdbc 进行了封装，屏蔽了jdbc api 底层访问细节，使我们不用与 jdbc api打交道，就可以完成对数据库的持久化操作。

### MyBatis的缓存分为两种：

一级缓存，一级缓存是SqlSession级别的缓存，自带的不可卸载的.

二级缓存，二级缓存是Mapper级别的缓存，定义在Mapper文件的<cache>标签中并需要开启此缓存.

### 持久层框架为什么选择 mybatis？

互联网项目**追求的是高性能**，尤其是数据库查询的时候，在**并发量高**的情况下需要**对** **sql 语句进行调优**。**Mybatis 就是面向 sql 语句的**，所以操作起来笔记容易入手。所以一般互联网项目中大多采用 mybatis 作为持久层框架。

### Mybatis 中#和$的区别？

* ${}是 Properties 文件中的变量占位符，它可以用于标签属性值和 sql 内部，属于静态文本替换，

比如${driver}会被静态替换为com.mysql.jdbc.Driver。

* #{}是 sql 的参数占位符，Mybatis 会将 sql 中的#{}替换为?号，在 sql 执行前会使用 PreparedStatement 的参数设置方法，按序给 sql 的?号占位符设置参数值。

比如 ps.setInt(0, parameterValue)，#{item.name} 的取值方式为使用反射从参数对象中获取 item 对象的 name 属性值，相当于 param.getItem().getName()。

### Mybatis 持久层sql语句等于号后面的符号是什么？

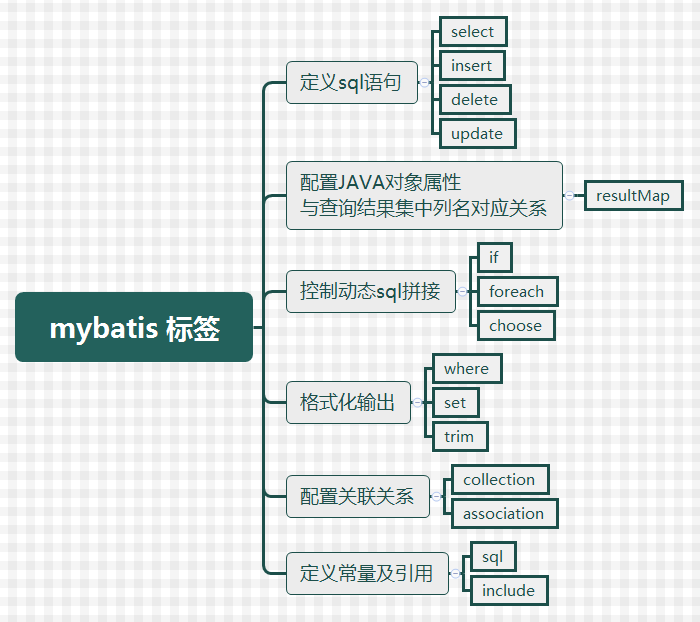
是井号（#）。

### Mybatis 中控制变化参数的标签？

If标签，where/if标签，foreach标签.

### **Xml 映射**文件**中，除了常见的 select|insert|updae|delete 标签之外，还有哪些标签？**

还有很多其他的标签，<resultMap>、<parameterMap>、<sql>、<include>、<selectKey>，加上动态sql的 9 个标签，trim|where|set|foreach|if|choose|when|otherwise|bind等。其中为 sql 片段标签，通过<include>标签引入 sql 片段，<selectKey>为不支持自增的主键生成策略标签。

****

### **最佳实践中，通常一个 Xml 映射文件，都会写一个 Dao 接口与之对应，请问，这个 Dao 接口的工作原理是什么？Dao 接口里的方法，参数不同时，方法能重载吗？**

答：Dao 接口的工作原理是 JDK 动态代理，Mybatis 运行时会使用 JDK 动态代理为 Dao 接口生成代理 proxy 对象，代理对象 proxy 会拦截接口方法，转而执行MappedStatement所代表的 sql，然后将 sql 执行结果返回。

Dao 接口，就是人们常说的 Mapper接口，接口的全限名，就是映射文件中的 namespace 的值，接口的方法名，就是映射文件中MappedStatement的 id 值，接口方法内的参数，就是传递给 sql 的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为 key 值，可唯一定位一个MappedStatement。

Dao 接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

### **Mybatis 动态 sql 是做什么的？都有哪些动态 sql？能简述一下动态 sql 的执行原理不？**

答：Mybatis 动态 sql 可以让我们在 Xml 映射文件内，以标签的形式编写动态 sql，完成逻辑判断和动态拼接 sql 的功能，Mybatis 提供了 9 种动态 sql 标签 trim|where|set|foreach|if|choose|when|otherwise|bind。

其执行原理为，使用 OGNL 从 sql 参数对象中计算表达式的值，根据表达式的值动态拼接 sql，以此来完成动态 sql 的功能。

### **Mybatis 是如何将 sql 执行结果封装为目标对象并返回的？都有哪些映射形式？**

答：第一种是使用<resultMap>标签，逐一定义列名和对象属性名之间的映射关系。第二种是使用 sql 列的别名功能，将列别名书写为对象属性名，比如 T\_NAME AS NAME，对象属性名一般是 name，小写，但是列名不区分大小写，Mybatis 会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成 T\_NAME AS NaMe，Mybatis 一样可以正常工作。

有了列名与属性名的映射关系后，Mybatis 通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

## **SpringBoot**

### 什么是微服务？

微服务架构是一种架构模式或者说是一种架构风格，它提倡将单一应用程序划分成一组小的服务，每个服务运行在其独立的自己的进程中，服务之间互相协调、互相合作，最终对外提供服务。服务之间采用轻量级的通信机制互相沟通。每个服务都围绕着具体业务进行构建，并且能够被独立地部署到生产环境、类生产环境等。SpringBoot是目前最流行的微服务框架，Spring Cloud是基于Spring Boot搭建的分布式微服务系统解决方案。

### **为什么要用微服务？**

1.、 微服务架构强调业务系统需要彻底的组件化和服务化，一个组件就是一个产品，可以独立对外提供服务。

2、 微服务强调每个微服务都有自己独立的运行空间，包括数据库资源。

3.、 微服务架构是 SOA 架构思想的一种扩展，更加强调服务个体的独立性、拆分粒度更小。

### 微服务之间是如何通信？

第一种：远程过程调用，即RPC（Remote Procedure Invocation）直接通过远程过程调用来访问别的service，如：REST、Apache。第二种：消息使用异步消息来做服务间通信。服务间通过消息管道来交换消息，从而通信。如：Apache Kafka、RabbitMQ

### 微服务之间是如何独立通讯的？

1.远程过程调用（Remote Procedure Invocation）：

也就是我们常说的服务的注册与发现，直接通过远程过程调用来访问别的service。

优点：简单，常见,因为没有中间件代理，系统更简单

缺点：

只支持请求/响应的模式，不支持别的，比如通知、请求/异步响应、发布/订阅、发布/异步响应。

降低了可用性，因为客户端和服务端在请求过程中必须都是可用的。

2.消息：

使用异步消息来做服务间通信。服务间通过消息管道来交换消息，从而通信。

优点:

把客户端和服务端解耦，更松耦合。

提高可用性，因为消息中间件缓存了消息，直到消费者可以消费。

支持很多通信机制比如通知、请求/异步响应、发布/订阅、发布/异步响应。

 缺点:

消息中间件有额外的复杂。

### 什么是 Spring Boot？

Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件一站式解决方案，主要是简化了使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

### Spring Boot有哪些优点？

1. Spring Boot引入自动配置的概念，快速创建独立运行的spring项目与主流框架集成 。它并不是用来替代Spring的解决方案，而是和Spring框架紧密结合用于提升Spring开发者体验的工具。
2. 它集成了大量常用的第三方库配置(例如Jackson, JDBC, Mongo, Redis, Mail等等)，Spring Boot应用中这些第三方库几乎可以零配置的开箱即用，大部分的Spring Boot应用都只需要非常少量的配置代码，开发者能够更加专注于业务逻辑。
3. SpringBoot提供服务单独运行的能力，而且内嵌容器如Tomcat，SpringBoot的项目在配置上做到了极简，通过spring自动配置让你解脱对项目配置依赖的过分关注，同时，SpringBoot的注解方式让我们从纷繁复杂的XML配置中彻底的解放了出来。

缺点：因为简单所以屏蔽了背后复杂的实现，这让不熟悉Spring的开发者在后续问题的解决上变得困难。

### Spring Boot 的配置文件有哪几种格式？

\*\*\*.properties 和 \*\*\*.yml，它们的区别主要是书写格式不同。.yml是一种人类可读的数据序列化语言。properties 是属性配置文件；

在配置文件中添加复杂的属性，yml文件可以分层配置数据，就更加结构化，而且更少混淆。 .yml 格式不支持 @PropertySource 注解导入配置。

### Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？

启动类注解是@SpringBootApplication，它也是 Spring Boot 的核心注解 主要组合包含了以下 3 个注解：  
1，@SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。  
2，@EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能：@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。  
3，@ComponentScan：Spring组件扫描。

### 运行 Spring Boot 有哪几种方式？

1）打包用命令或者放到容器中运行；

2）用 Maven/ Gradle 插件运行；

3）直接执行 main 方法运行。

### Spring Boot 支持的日志框架？

Spring Boot 支持 Java Util Logging, Log4j2, Lockback 作为日志框架，如果你使用 Starters 启动器，Spring Boot 将使用 Logback 作为默认日志框架。

### SpringBoot 实现热部署有哪几种方式？

主要有两种方式：

使用Spring Loaded配置pom.xml文件。

使用devtools工具包。

### Spring Boot 如何配置多环境？

提供多套配置文件：

applcation.yml

application-dev.yml  
application-test.yml  
application-pro.yml

通过改变applcation里的profiles.active值来加载对应的环境。

spring:  
 profiles:  
 active: dev

### **说说SpringBoot和SpringCloud的区别？**

SpringBoot是Spring推出用于解决传统框架配置文件冗余,装配组件繁杂的基于Maven的解决方案,旨在快速搭建单个微服务。  
而SpringCloud专注于解决各个微服务之间的协调与配置,服务之间的通信,熔断,负载均衡等。  
技术维度并不相同,并且SpringCloud是依赖于SpringBoot的,而SpringBoot并不是依赖与SpringCloud,甚至还可以和Dubbo进行优秀的整合开发。

总结:

SpringBoot专注于快速方便的开发单个个体的微服务。

SpringCloud是关注全局的微服务协调整理治理框架,整合并管理各个微服务,为各个微服务之间提供,配置管理,服务发现,断路器,路由,事件总线等集成服务。

SpringBoot不依赖于SpringCloud,SpringCloud依赖于SpringBoot,属于依赖关系。

SpringBoot专注于快速,方便的开发单个的微服务个体,SpringCloud关注全局的服务治理框架。

## **SpringCloud**

### 什么是SpringCloud？

spring cloud 是一系列框架的有序集合。它利用 spring boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 spring boot 的开发风格做到一键启动和部署。

### spring cloud 断路器的作用是什么？

在分布式架构中，断路器模式的作用也是类似的，当某个服务单元发生故障（类似用电器发生短路）之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。

### spring cloud 的核心组件有哪些？

Eureka：服务注册于发现。

Feign：基于动态代理机制，根据注解和选择的机器，拼接请求 url 地址，发起请求。

Ribbon：实现负载均衡，从一个服务的多台机器中选择一台。

Hystrix：提供线程池，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题。

Zuul：网关管理，由 Zuul 网关转发请求给对应的服务。

### 说说SpringCloud和Dubbo的区别？

SpringCloud和Dubbo都是现在主流的微服务架构  
SpringCloud是Apache旗下的Spring体系下的微服务解决方案  
Dubbo是阿里系的分布式服务治理框架  
从技术维度上,其实SpringCloud远远的超过Dubbo,Dubbo本身只是实现了服务治理,而SpringCloud现在以及有21个子项目以后还会更多  
所以其实很多人都会说Dubbo和SpringCloud是不公平的  
但是由于RPC以及注册中心元数据等原因,在技术选型的时候我们只能二者选其一,所以我们常常为用他俩来对比  
服务的调用方式Dubbo使用的是RPC远程调用,而SpringCloud使用的是 Rest API,其实更符合微服务官方的定义  
服务的注册中心来看,Dubbo使用了第三方的ZooKeeper作为其底层的注册中心,实现服务的注册和发现,SpringCloud使用Spring Cloud Netflix Eureka实现注册中心,当然SpringCloud也可以使用ZooKeeper实现,但一般我们不会这样做  
服务网关,Dubbo并没有本身的实现,只能通过其他第三方技术的整合,而SpringCloud有Zuul路由网关,作为路由服务器,进行消费者的请求分发,SpringCloud还支持断路器,与git完美集成分布式配置文件支持版本控制,事务总线实现配置文件的更新与服务自动装配等等一系列的微服务架构要素

**从技术选型上讲~**

目前国内的分布式系统选型主要还是Dubbo毕竟国产,而且国内工程师的技术熟练程度高,并且Dubbo在其他维度上的缺陷可以由其他第三方框架进行集成进行弥补  
而SpringCloud目前是国外比较流行,当然我觉得国内的市场也会慢慢的偏向SpringCloud,就连刘军作为Dubbo重启的负责人也发表过观点,Dubbo的发展方向是积极适应SpringCloud生态,并不是起冲突

**Rest和RPC对比**

其实如果仔细阅读过微服务提出者马丁福勒的论文的话可以发现其定义的服务间通信机制就是Http Rest  
RPC最主要的缺陷就是服务提供方和调用方式之间依赖太强,我们需要为每一个微服务进行接口的定义,并通过持续继承发布,需要严格的版本控制才不会出现服务提供和调用之间因为版本不同而产生的冲突  
而REST是轻量级的接口,服务的提供和调用不存在代码之间的耦合,只是通过一个约定进行规范,但也有可能出现文档和接口不一致而导致的服务集成问题,但可以通过swagger工具整合,是代码和文档一体化解决,所以REST在分布式环境下比RPC更加灵活  
这也是为什么当当网的DubboX在对Dubbo的增强中增加了对REST的支持的原因

**文档质量和社区活跃度**

SpringCloud社区活跃度远高于Dubbo,毕竟由于梁飞团队的原因导致Dubbo停止更新迭代五年,而中小型公司无法承担技术开发的成本导致Dubbo社区严重低落,而SpringCloud异军突起,迅速占领了微服务的市场,背靠Spring混的风生水起  
Dubbo经过多年的积累文档相当成熟,对于微服务的架构体系各个公司也有稳定的现状

### SpringCloud有什么优势？

优点：

1、服务拆分粒度更细，有利于资源重复利用，有利于提高开发效率；

2、可以更精准的制定优化服务方案，提高系统的可维护性；

3、微服务架构采用去中心化思想，服务之间采用Restful等轻量级通讯，比ESB更轻量；

4、适于互联网时代，产品迭代周期更短。

缺点：

1、微服务过多，治理成本高，不利于维护系统；

2、分布式系统开发的成本高（容错，分布式事务等）对团队挑战大；

总的来说优点大过于缺点，目前看来SpringCloud是一套非常完善的分布式框架，目前很多企业开始用微服务。

### SpringCloud如何实现服务的注册和发现？

1. 服务发布时，指定对应的服务名,将服务注册到 注册中心(eureka zookeeper)  
   2.注册中心加@EnableEurekaServer,服务用@EnableDiscoveryClient，然后用ribbon或feign进行服务直接的调用发现。

### Eureka和ZooKeeper都可以提供服务注册与发现的功能,请说说两个的区别？

著名的CAP理论指出，一个分布式系统不可能同时满足C(一致性)、A(可用性)和P(分区容错性)。由于分区容错性P在是分布式系统中必须要保证的，因此我们只能在A和C之间进行权衡。

1.ZooKeeper保证的是CP,Eureka保证的是AP

ZooKeeper在选举期间注册服务瘫痪,虽然服务最终会恢复,但是选举期间不可用的

Eureka各个节点是平等关系,只要有一台Eureka就可以保证服务可用,而查询到的数据并不是最新的

自我保护机制会导致

Eureka不再从注册列表移除因长时间没收到心跳而应该过期的服务

Eureka仍然能够接受新服务的注册和查询请求,但是不会被同步到其他节点(高可用)

当网络稳定时,当前实例新的注册信息会被同步到其他节点中(最终一致性)

Eureka可以很好的应对因网络故障导致部分节点失去联系的情况,而不会像ZooKeeper一样使得整个注册系统瘫痪

2.ZooKeeper有Leader和Follower角色,Eureka各个节点平等

3.ZooKeeper采用过半数存活原则,Eureka采用自我保护机制解决分区问题

4.Eureka本质上是一个工程,而ZooKeeper只是一个进程。

### Eureka自我保护机制是什么?

当Eureka Server 节点在短时间内丢失了过多实例的连接时（比如网络故障或频繁启动关闭客户端）节点会进入自我保护模式，保护注册信息，不再删除注册数据，故障恢复时，自动退出自我保护模式。

### 什么是Ribbon？

ribbon是一个负载均衡客户端，可以很好的控制htt和tcp的一些行为。feign默认集成了ribbon。

### 什么是Feigin？它的优点是什么？

1.feign采用的是基于接口的注解  
2.feign整合了ribbon，具有负载均衡的能力  
3.整合了Hystrix，具有熔断的能力

使用:  
1.添加pom依赖。  
2.启动类添加@EnableFeignClients  
3.定义一个接口@FeignClient(name=“xxx”)指定调用哪个服务

### 什么是Ribbon和Feign，他们的区别？

1.Ribbon和Feign都是用于调用其他服务的，不过方式不同。

2.启动类使用的注解不同，Ribbon用的是@RibbonClient，Feign用的是@EnableFeignClients。

3.服务的指定位置不同，Ribbon是在@RibbonClient注解上声明，Feign则是在定义抽象方法的接口中使用@FeignClient声明。

4.调用方式不同，Ribbon需要自己构建http请求，模拟http请求然后使用RestTemplate发送给其他服务，步骤相当繁琐。

Feign则是在Ribbon的基础上进行了一次改进，采用接口的方式，将需要调用的其他服务的方法定义成抽象方法即可。

### 什么是Spring Cloud Bus?

spring cloud bus 将分布式的节点用轻量的消息代理连接起来，它可以用于广播配置文件的更改或者服务直接的通讯，也可用于监控。  
如果修改了配置文件，发送一次请求，所有的客户端便会重新读取配置文件。  
使用:  
1.添加依赖  
2.配置rabbimq

### 什么是服务熔断？什么是服务降级?

服务直接的调用，比如在高并发情况下出现进程阻塞，导致当前线程不可用，慢慢的全部线程阻塞，导致服务器雪崩。

服务熔断：相当于保险丝，出现某个异常，直接熔断整个服务，而不是一直等到服务超时。通过维护一个自己的线程池，当线程到达阈值的时候就启动服务降级，如果其他请求继续访问就直接返回fallback的默认值。

### 什么是Hystrix(断路器)？它如何实现容错？

Hystrix是一个延迟和容错库，旨在隔离远程系统，服务和第三方库的访问点，当出现不可避免的故障时，停止级联故障并在复杂的分布式系统中实现弹性。

在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等， Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

### 什么是SpringCloudConfig?

在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件spring cloud config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。在spring cloud config 组件中，分两个角色，一是config server，二是config client。

使用：  
1、添加pom依赖  
2、配置文件添加相关配置  
3、启动类添加注解@EnableConfigServer

### 架构？

在微服务架构中，需要几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、智能路由、配置管理等，由这几个基础组件相互协作，共同组建了一个简单的微服务系统

在Spring Cloud微服务系统中，一种常见的负载均衡方式是，客户端的请求首先经过负载均衡（zuul、Ngnix），再到达服务网关（zuul集群），然后再到具体的服。，服务统一注册到高可用的服务注册中心集群，服务的所有的配置文件由配置服务管理，配置服务的配置文件放在git仓库，方便开发人员随时改配置。

### 什么是 zuul路由网关?

Zuul 包含了对请求的路由和过滤两个最主要的功能：

其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础.Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。

注意：Zuul服务最终还是会注册进Eureka

提供=代理+路由+过滤 三大功能

### 什么是SpringCloud Config分布式配置中心？

SpringCloud Config为微服务架构中的微服务提供集中化的外部配置支持，配置服务器为各个不同微服务应用的所有环境提供了一个中心化的外部配置。

### 分布式配置中心能干嘛？

1)，集中管理配置文件不同环境不同配置，动态化的配置更新，分环境部署比如dev/test/prod/beta/release

2)，运行期间动态调整配置，不再需要在每个服务部署的机器上编写配置文件，服务会向配置中心统一拉取配置自己的信息

3)，当配置发生变动时，服务不需要重启即可感知到配置的变化并应用新的配置将配置信息以REST接口的形式暴