**面试题总结**

## **java中四种引用类型**

在JDK 1.2以前的版本中，若一个对象不被任何变量引用，那么程序就无法再使用这个对象。也就是 说，只有对象处于可触及（reachable）状态，程序才能使用它。从JDK 1.2版本开始，把对象的引 用分为4种级别，从而使程序能更加灵活地控制对象的生命周期。这4种级别由高到低依次为：强引 用、软引用、弱引用和虚引用。

**⑴ 强引用**（StrongReference）

强引用是使用最普遍的引用。**如果一个对象具有强引用，那垃圾回收器绝不会回收它**。 当内存空 间不足，Java虚拟机宁愿抛出OutOfMemoryError错误，使程序异常终止， 也不会靠随意回收 具有强引用的对象来解决内存不足的问题。  ps：强引用其实也就是 我们平时A a = new A()这个意 思。

**⑵ 软引用**（SoftReference）

**如果一个对象只具有软引用，则内存空间足够，垃圾回收器就不会回收它；如果内存空间不足了，就 会回收这些对象的内存。**只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现 内存敏感的高速缓存（下文给出示例）。软引用可以和一个引用队列（ReferenceQueue）联合使用， 如果软引用所引用的对象被垃圾回收器回收，Java虚拟机就会把这个软引用加入到与之关联的引用队 列中。

**⑶ 弱引用**（WeakReference）

弱引用与软引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。**在垃圾回收器线程扫描它 所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回 收它的内存。**不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱 引用的对象。弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象 被垃圾回收，Java虚拟机就会把这个弱引用加入到与之关联的引用队列中。

**⑷ 虚引用**（PhantomReference）

“虚引用”顾名思义，就是形同虚设，与其他几种引用都不同，虚引用并不会决定对象的生命周期。 **如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。**

## **union 和 union all的区别**

UNION 指令的目的是将两个 SQL 语句的结果合并起来,可以查看你要的查询结果.

select employee\_id,job\_id from employees

union

select employee\_id,job\_id from job\_history

union 并集，表中的所有数据，并且去除重复数据（工作中主要用到的是这个）；**去重且排序**

union all，表中的数据都罗列出来；**不去重不排序**

**从效率上说**，UNION ALL 要比UNION快很多，所以，如果可以确认合并的两个结果集中不包含重 复的数据的话，尽量使用union all，因为union需要进行排序，去除重复记录，效率低

## **minus**

在进行两个表格或者两个查询结果的时候，

返回在第一个表格/查询结果中与第二个表格/查询结果不相同的记录

## **已上线项目出现BUG！**

在测试环境尽力重现问题，找出解决方案，从线上拉取fix分支，还原bug，修复，测试通过，合并 分支

## **存储金钱用什么数据类型**

BigDecimal

## **SQL优化**

对查询进行优化，要尽量避免全表扫描

查询语句中不要使用 \*

or 的查询尽量用 union或者union all 代替

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

尽量减少子查询，使用关联查询（left join,right join,inner  join）替代

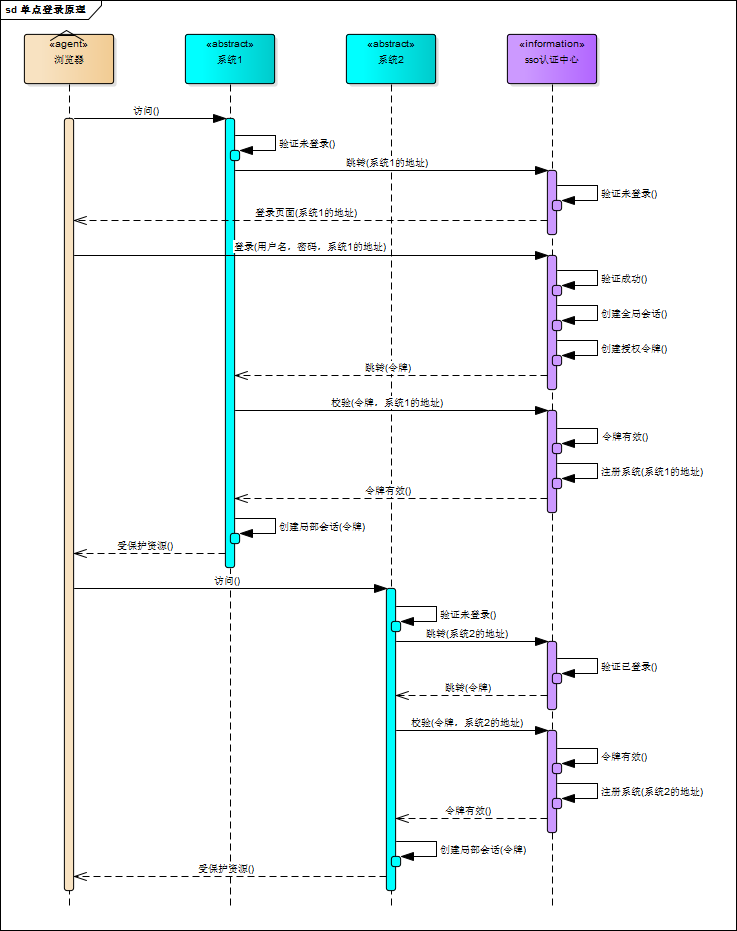
修改 like 程序，去掉前置百分号。like语句却因为前置百分号而无法使用索引

## **单点登录**

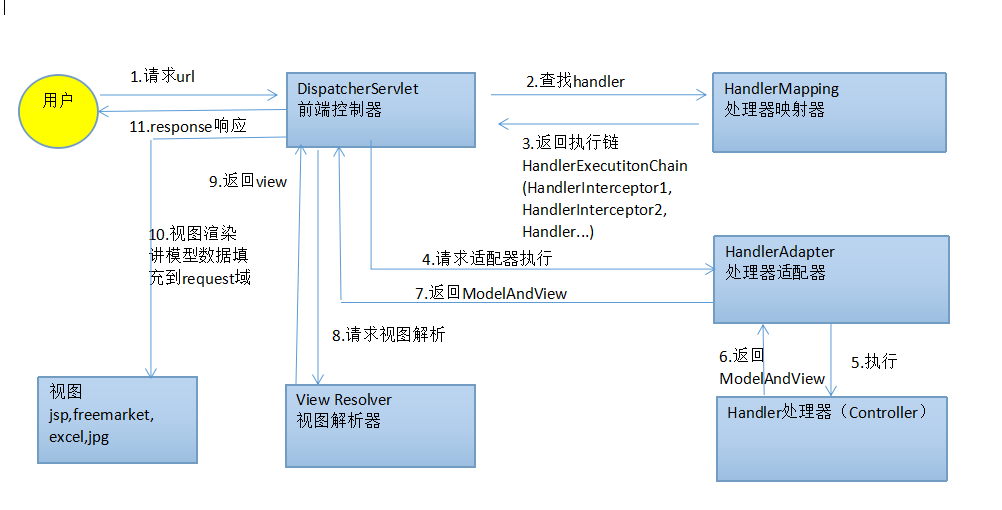
全称Single Sign On（以下简称SSO），是指在多系统应用群中登录一个系统，便可在其他所有系 统中得到授权而无需再次登录，包括单点登录与单点注销两部分

相比于单系统登录，sso需要一个独立的认证中心，只有认证中心能接受用户的用户名密码等安全信 息，其他系统不提供登录入口，只接受认证中心的间接授权。间接授权通过令牌实现，sso认证中心 验证用户的用户名密码没问题，创建授权令牌，在接下来的跳转过程中，授权令牌作为参数发送给各 个子系统，子系统拿到令牌，即得到了授权，可以借此创建局部会话，局部会话登录方式与单系统的 登录方式相同。这个过程，也就是单点登录的原理，用下图说明

详情：https://www.cnblogs.com/ywlaker/p/6113927.html



## **SpringMVC执行流程**

****

1.用户发送请求至前端控制器DispatcherServlet

2.DispatcherServlet收到请求调用处理器映射器HandlerMapping。

3.处理器映射器根据请求url找到具体的处理器，生成处理器执行链HandlerExecutionChain并返回给前端控制器DispatcherServlet。

4.前端控制器DispatcherServlet根据处理器Handler获取处理器适配器HandlerAdapter执行HandlerAdapter处理一系列的操作，如：参数封装，数据格式转换，数据验证等操作

5.执行处理器Handler(Controller，也叫页面控制器)。

6.Handler执行完成返回ModelAndView

7.HandlerAdapter将Handler执行结果ModelAndView返回到DispatcherServlet

8.DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9.ViewReslover解析后返回具体View

10.DispatcherServlet对View进行渲染视图（即将模型数据model填充至视图中）。

11.DispatcherServlet响应用户。

## **日志级别**

等级由低到高：debug < info < warn < Error < Fatal ；

debug 级别最低，可以随意的使用于任何觉得有利于在调试时更详细的了解系统运行状态的东东；

info 重要，输出信息：用来反馈系统的当前状态给最终用户的；

后三个，警告、错误、严重错误，这三者应该都在系统运行时检测到了一个不正常的状态。

warn, 可修复，系统可继续运行下去；

Error, 可修复性，但无法确定系统会正常的工作下去;

Fatal, 相当严重，可以肯定这种错误已经无法修复，并且如果系统继续运行下去的话后果严重。

## **Spring执行原理**

IOC AOP

## **spring的三种注入方式是什么?**

setter（set方法注入）

interface（注解注入）

constructor（构造注入）

## **Java：注解（Annotation）自定义注解入门**

**元注解：**

　　元注解的作用就是负责注解其他注解。Java5.0定义了4个标准的meta-annotation类型，它 们被用来提供对其它 annotation（注解）类型作说明。Java5.0定义的元注解：

　　　　1.@Target：说明了注解所修饰的对象范围

　　　　2.@Retention：定义了该注解被保留的时间长短

　　　　3.@Documented：用于描述其它类型的注解应该被作为被标注的程序成员的公共API

　　　　4.@Inherited：是一个标记注解，@Inherited阐述了某个被标注的类型是被继承的

**自定义注解：**

[使用@interface自定义注解时，自动继承了java.lang.annotation.Annotation接口，由编译 程序自](mailto:使用@interface自定义注解时，自动继承了java.lang.annotation.Annotation接口，由编译程序自)动完成其他细节。在定义注解时，不能继承其他的注解或接口。@interface用来声明一 个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回 值类型就是参数的类型（返回值类型只能是基本类型、Class、String、enum）。可以通过default 来声明参数的默认值。

**定义注解格式：**  
　　 public @interface 注解名 { 定义体 }

## **Java的原子操作**

 "原子操作(atomic operation)是不需要synchronized"，这是Java多线程编程的[老生常谈](https://www.baidu.com/s?wd=%E8%80%81%E7%94%9F%E5%B8%B8%E8%B0%88&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/chuck_kui/article/details/_blank)了。 所谓原子操作是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束， 中间不会有任何 context switch （切[1]  换到另一个线程）。

## **在并发编程中，原子性，可见性，有序性**

**原子性：**

一个操作或者多个操作要么全部执行并且执行的过程不会被任何因素打断，要么就都不执行。

Java内存模型保证了基本读取和赋值是原子性操作，如果要实现更大范围操作的原子性，可以通过synchronized 和Lock来实现。由于synchronized和Lock能够保证任一时刻只有一个线程执行该代码块，那么自然就不存在 原子性问题了，从而保证了原子性。

**可见性：**

是指当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到 修改的值。

对于可见性，Java提供了volatile关键字来保证可见性。当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。

**有序性：**

即程序执行的顺序按照代码的先后顺序执行。

在Java内存模型中，允许编译器和处理器对指令进行重排序，但是重排序过程不会影响到单线程程序的执行，却 会影响到多线程并发执行的正确性。

　　在Java里面，可以通过volatile关键字来保证一定的“有序性”（具体原理在下一节讲述）。另外可以通过 synchronized和Lock来保证有序性，很显然，synchronized和Lock保证每个时刻是有一个线程执行同步代 码，相当于是让线程顺序执行同步代码，自然就保证了有序性。

## **volatile关键词**

Java提供了volatile关键字来保证可见性。

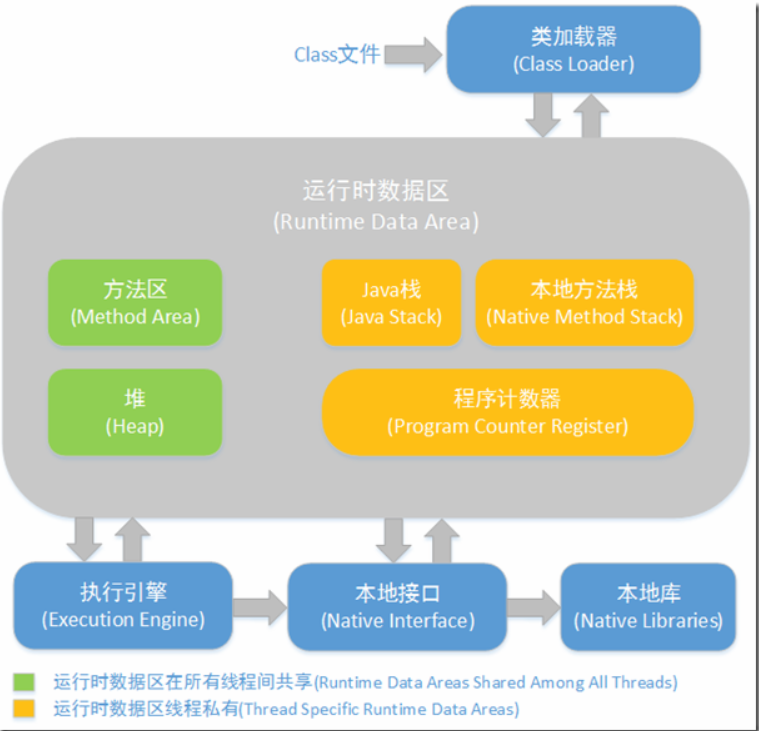
　　当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取 时，它会去内存中读取新值。

　　而普通的共享变量不能保证可见性，因为普通共享变量被修改之后，什么时候被写入主存是不确定的， 当其他线程去读取时，此时内存中可能还是原来的旧值，因此无法保证可见性。

　　另外，通过synchronized和Lock也能够保证可见性，synchronized和Lock能保证同一时刻只有 一个线程获取锁然后执行同步代码，并且在释放锁之前会将对变量的修改刷新到主存当中。因此可以 保证可见性。

## **JVM内存结构**

JVM内存结构主要有三大块：堆内存、方法区 和 栈。



**Java堆（Heap）**

Java堆是被所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对 象实例，几乎所有的对象实例都在这里分配内存。

**方法区（Method Area）**

方法区（Method Area）与Java堆一样，是各个线程共享的内存区域，它用于存储已被虚拟机加 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。虽然Java虚拟机规范把方法区描 述为堆的一个逻辑部分，但是它却有一个别名叫做Non-Heap（非堆）

**程序计数器（Program Counter Register）**

程序计数器（Program Counter Register）是一块较小的内存空间，它的作用可以看做是当前线程 所执行的字节码的行号指示器。

**JVM栈（JVM Stacks）**

与程序计数器一样，Java虚拟机栈（Java Virtual Machine Stacks）也是线程私有的，它的生命周 期与线程相同。虚拟机栈描述的是Java方法执行的内存模型：每个方法被执行的时候都会同时创建 一个栈帧（Stack Frame）用于存储局部变量表、操作栈、动态链接、方法出口等信息。每一个方法 被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。

**本地方法栈（Native Method Stacks）**

本地方法栈（Native Method Stacks）与虚拟机栈所发挥的作用是非常相似的，其区别不过是虚拟 机栈为虚拟机执行Java方法（也就是字节码）服务，而本地方法栈则是为虚拟机使用到的Native方 法服务。

## **Java内存模型**

Java内存模型看上去和Java内存结构（JVM内存结构）差不多，很多人会误以为两者是一回事儿， 这也就导致面试过程中经常答非所为。

在前面的关于JVM的内存结构的图中，我们可以看到，其中Java堆和方法区的区域是多个线程共享 的数据区域。也就是说，多个线程可能可以操作保存在堆或者方法区中的同一个数据。这也就是我们 常说的“Java的线程间通过共享内存进行通信”。

Java内存模型是根据英文Java Memory Model（JMM）翻译过来的。其实JMM并不像JVM内存 结构一样是真实存在的。他只是一个抽象的概念。JMM是和多线程相关的，他描述了一组规则或规 范，这个规范定义了一个线程对共享变量的写入时对另一个线程是可见的。

## 那么，简单总结下，Java的多线程之间是通过共享内存进行通信的，而由于采用共享内存进行通信， 在通信过程中会存在一系列如可见性、原子性、顺序性等问题，而JMM就是围绕着多线程通信以及 与其相关的一系列特性而建立的模型。JMM定义了一些语法集，这些语法集映射到Java语言中就是 volatile、synchronized等关键字。

## **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？**

当调用start()方法时你将创建新的线程，并且执行在run()方法里的代码。但是如果直接调用run() 方法，它不会创建新的线程也不会执行调用线程的代码。

## **什么是多线程中的上下文切换？**

单核CPU也支持多线程执行代码，CPU通过给每个线程分配CPU时间片来实现这个机制。时间片是CPU分配给各个线程的时间，因为时间片非常短，所以CPU通过不停地切换线程执行，让我们感觉多个线程时同时执行的，时间片一般是几十毫秒（ms）。

CPU通过时间片分配算法来循环执行任务，当前任务执行一个时间片后会切换到下一个任务。但是，在切换前会保存上一个任务的状态，以便下次切换回这个任务时，可以再次加载这个任务的状态，从任务保存到再加载的过程就是一次上下文切换。

## **线程池**

**ThreadPoolExecutor**

**https://www.cnblogs.com/dolphin0520/p/3932921.html**

## **Mysql插入300万条数据(存储过程，64秒)**

**--创建MyISAM模式表方便批量跑数据**

**CREATE TABLE `logs1` (**

**`id` int(11) NOT NULL AUTO\_INCREMENT,**

**`logtype` varchar(255) DEFAULT NULL,**

**`logurl` varchar(255) DEFAULT NULL,**

**`logip` varchar(255) DEFAULT NULL,**

**`logdz` varchar(255) DEFAULT NULL,**

**`ladduser` varchar(255) DEFAULT NULL,**

**`lfadduser` varchar(255) DEFAULT NULL,**

**`laddtime` datetime DEFAULT NULL,**

**`htmlname` varchar(255) DEFAULT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=MyISAM AUTO\_INCREMENT=1811 DEFAULT CHARSET=utf8 COMMENT='日志表';**

**--创建存储过程**

**DROP PROCEDURE IF EXISTS my\_insert;**

**CREATE PROCEDURE my\_insert()**

**BEGIN**

**DECLARE n int DEFAULT 1;**

**loopname:LOOP**

**INSERT INTO `logs1`(`logtype`,`logurl`,`logip`,`logdz`,`ladduser` ,`lfadduser`,`laddtime`,`htmlname`) VALUES ( 2, '/index', '0:0:0:0:0:0:0:1', null, null, 'null', '2018-05-03 14:02:42', '首页');**

**SET n=n+1;**

**IF n=3000001 THEN**

**LEAVE loopname;**

**END IF;**

**END LOOP loopname;**

**END;**

**--执行存储过程**

**CALL my\_insert();**

**--数据插入成功后修改表模式InnoDB 时间稍微久点**

**alter table `logs1` engine=InnoDB;**

## **Springmvc和Struts的区别？**

\* 1: Struts2框架是类级别的拦截，每次来了请求就创建一个Action，然后调用get和set方法把request中的数据注入；Struts2中一个action对应一个requset上下文.  
\* 2 :SpringMVC是方法级别的拦截，拦截到方法后根据参数的注解，把requset数据注入进去springmvc容器中，一个方法对应一个request上下文.  
\* 3 :springmvc方法之间基本独立，独享requset和response之间的数据。请求数据通过参数获取，处理结果通过model Map交回给框架。（方法之间不共享变量）  
\* Struts虽然方法之间也是独立的，当时action变量是可以共享的  
\* 4 :处理ajax的请求很方便，只需一个注解2responseBody ,直接返回响应文本。  
\* 5 :SpingMVC的实现是servlet.Struts2是filter。

\* 6 ：intercepter的实现机制。Struts 有自己的拦截器机制.interceptor,Springmvc用的是独立的AOP方式,导致Springmvc的配置文件比struts2的配置文件多.