# 整理

# 题

### 类在内存中的唯一性如何确定

对于任意⼀个类，都需要由它的类加载器和这个类本身⼀同确定其在JVM中的唯⼀性。例如两个类的类名都是String，但是一个类的类全名为java.lang.String，另一个为com.study.String，这两个类不属于同一个。

public class MyTest {

@Test

public void dieLock(){

// sun.misc.Launcher$AppClassLoader@14dad5dc

System.out.println(MyTest.class.getClassLoader());

// sun.misc.Launcher$ExtClassLoader@6842775d

System.out.println(MyTest.class.getClassLoader().getParent());

// null

System.out.println(MyTest.class.getClassLoader().getParent().getParent());

}

}

### 说一下 JVM 调优的工具

JDK 自带了很多监控工具，都位于 JDK 的 bin 目录下，其中最常用的是 jconsole 和 jvisualvm 这两款视图监控工具。

jconsole用于对 JVM 中的内存、线程和类等进行监控；

jvisualvm是JDK 自带的全能分析工具，可以分析：内存快照、线程快照、程序死锁、监控内存的变化、gc 变化等。

### 说一下类加载的执行过程

加载：将 class 文件加载到内存中；

检查：检查加载的 class 文件的正确性；

准备：给类中的静态变量分配内存空间；

解析：虚拟机将常量池中的符号引用替换成直接引用的过程。符号引用就理解为一个标识，而直接引用是指直接指向内存中的地址；

初始化：对静态变量和静态代码块执行初始化工作。

### 说一下 JVM 有哪些垃圾回收算法

标记-清除算法：标记无用对象，然后进行清除回收。

标记-整理算法：标记无用对象，让所有存活的对象都向一端移动，然后直接清除掉端边界以外的内存。

复制算法：按照容量划分二个大小相等的内存区域，当一块用完的时候将活着的对象复制到另一块上，然后再把已使用的内存空间一次清理掉。

分代算法：根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，新生代基本采用复制算法，老年代采用标记整理算法。

### 什么是双亲委派模型

类加载器根据指定全限定名称将 class 文件加载到 JVM 内存，然后再转化为 class 对象。

启动类加载器（Bootstrap ClassLoader）：用来加载 lib 目录中的类库。

扩展类加载器（Extension ClassLoader）：负责加载 \lib\ext 目录中的类库。

应用程序类加载器（Application ClassLoader）：负责加载用户类路径（classpath）上的指定类库。

双亲委派模型是指如果一个类加载器收到了类加载的请求，它首先不会自己去加载这个类，而是把这个请求委派给父类加载器去完成，每一层的类加载器都是如此，这样所有的加载请求都会被传送到顶层的启动类加载器中，只有当父加载无法完成加载请求时，子加载器才会尝试去加载类。

### 说一下 JVM 的主要组成部分及其作用

组成部分：

类加载器、运行时数据区、执行引擎、本地库接口。

运行时数据区包括：方法区、堆区、本地方法栈、虚拟机栈、程序计数器。

组件的作用：

类加载器会将字节码文件加载到运行时数据区的方法区中，执行引擎将字节码翻译成底层系统指令，再交由 CPU 去执行，这个过程中需要调用其他语言的本地库接口来实现整个程序的功能。

### Redis怎么实现分布式锁

在系统里面占一个“坑”，其他程序也要占“坑”的时候，占用成功了就可以继续执行，失败了就只能放弃或稍后重试。

占坑一般使用 setnx(set if not exists) 指令，只允许被一个程序占有，使用完调用 del 释放锁。

### 说一下 MySQL 的行锁和表锁

MyISAM 只支持表锁，InnoDB 支持表锁和行锁，默认为行锁。

表级锁：开销小，加锁快，不会出现死锁。锁定粒度大，发生锁冲突的概率最高，并发量最低。

行级锁：开销大，加锁慢，会出现死锁。锁力度小，发生锁冲突的概率小，并发度最高。

### 说一下事务的四大特性ACID是什么

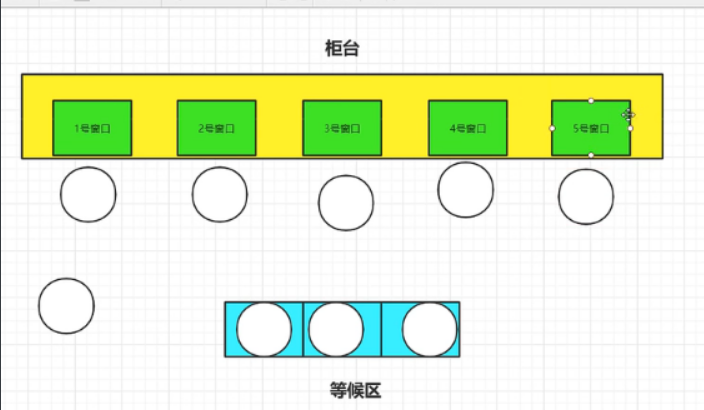
原子性：一个事务（transaction）中的所有操作，或者全部完成，或者全部不完成，不会结束在中间某个环节。

一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

隔离性：隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。

持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

### 线程池



将线程对象先创建好后，可以提高响应速度，如果刚开始线程池中只有三个线程，都用了，并且等待队列的任务也满了，此时再进来一个任务时，线程池会创建一个新的线程。直到达到了线程池的最大线程数，此时如果等待队列的任务满了，再进来一个新的任务，也无法执行该任务了。

### 说一下StackOverflowError和OutOfMemeryError的区别

StackOverflowError 栈溢出，一般由于递归过多，调用方法过多导致栈区内存不足。

OutOfMemeryError 堆内存溢出，即OOM，由于堆内存中没有被GC回收的对象过多导致。

内存映像分析工具

### 说一下java内存分配与垃圾回收

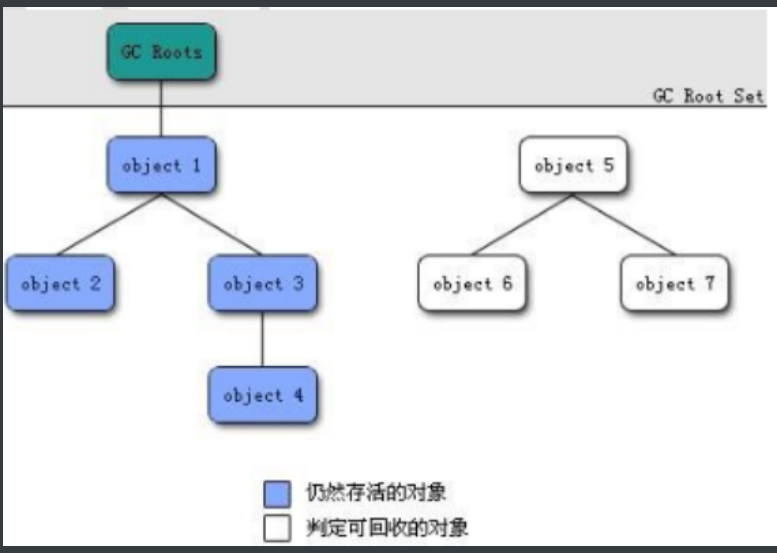
JVM的内存可以分为堆内存和非堆内存。

堆内存分为年轻代和老年代。

年轻代又可以进一步划分为一个Eden（伊甸）区和两个Survivor（幸存）区组成。

### 说一下如何判断一个对象应该被回收

该对象没有与 GC Roots 相连。



### 线程

Study.java

public class Study {

public static void main(String[] args) {

ExecutorService threadPool = Executors.newFixedThreadPool(50);

for (int i = 0; i < 100; i++) {

int r = i;

threadPool.execute(new Runnable() {

@Override

public void run() {

System.out.println(Thread.currentThread().getName() + "------>" + r );

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

}

System.out.println(1123);

}

}

### 线程池的拒绝策略

ThreadPoolExecutor.AbortPolicy:  
丢弃任务并抛出RejectedExecutionException异常。

ThreadPoolExecutor.DiscardPolicy：  
丢弃任务，但是不抛出异常。

ThreadPoolExecutor.DiscardOldestPolicy：  
丢弃队列最前面的任务，然后重新提交被拒绝的任务

ThreadPoolExecutor.CallerRunsPolicy：  
由提交任务的线程直接处理该任务

### 线程池满了，往线程池里提交任务会发生什么样的情况

线程池的饱和策略：当阻塞队列满了，且没有空闲的工作线程，此时继续提交任务，必须采取一种策略处理该任务，线程池提供4种策略

如果使用LinkedBlockingQueue（阻塞队列），也就是无界队列的话，继续添加任务到阻塞队列中等待执行，因为LinkedBlockingQueue可以近乎认为是一个无穷大的队列，可以无限存放任务；

如果使用的是有界队列如ArrayBlockingQueue，如果ArrayBlockingQueue满了，则会使用拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy。

### 线程池的线程数量怎么确定

如果是CPU密集型应用，则线程池大小设置为N+1。

如果是IO密集型应用，则线程池大小设置为2N+1。

### CAS思想

是一种乐观锁。

包含三个操作数：内存位置（V）、预期原值（A）、新值（B）

执行CAS操作时，将内存位置的值与预期原值进行比较，如果匹配，处理器会自动将该位置的值更新为新值。否则不做任何操作。

场景：当一个线程要修改共享变量的值，先取出共享变量的值赋给A,然后基于A的基础进行计算得到B,执行完毕后需要更新共享变量时，就可以调用CAS方法更新变量的值。

### 多线程并发或线程安全问题如何解决

1）通过volatile 关键字修饰变量，可以实现线程之间的可见性，避免变量脏读的出现，底层是通过限制jvm指令的重排序来实现的。

2）通过synchronized锁(任意对象)来实现线程同步。

3）通过lock锁的机制，进行手动lock和unlock。

4）通过线程安全的集合类，可以解决并发问题，如ConcurrentHashMap，底层使用的是cas机制（乐观锁），可以解决并发问题。

5）使用线程池来创建和管理线程，也可以一定程度上解决并发问题

6）使用ThreadLocal来修饰变量，可以解决并发问题。多个线程会复制一份threadLocao变量的副本进行操作，互不影响。

### JVM中的垃圾收集器有了解吗？

CMS收集器：

是一种以获取最短回收停顿时间为目标的收集器。

G1收集器：

是一款面向服务器的垃圾收集器,主要针对配备多颗处理器及大容量内存的机器，以极高概率满足GC停顿时间要求的同时,还具备高吞吐量性能特征。

### JVM的引用类型有哪些？

强引用：

普通存在，P p = new P()，只要强引用存在，垃圾收集器永远不会回收掉被引用的对象。当内存不足的时候，JVM宁可出现OutOfMemoryError错误停止，也需要进行保存，并且不会将此空间回收，在引用期间和栈有联系就无法被回收。

软引用：

当内存不足的时候，进行对象的回收处理，往往用于高速缓存中；mybatis就是

弱引用：

不管内存是否紧张，只要有垃圾了就立即回收。

虚引用：

也称为幽灵引用,和没有引用是一样的

### 说明内存泄漏和内存溢出的区别和联系，Java程序中如何检测？如何解决？

内存溢出：

程序申请内存时，没有足够的内存，就会报错OutOfMemory；

内存泄漏：

垃圾对象无法回收，比如缓存中有一个对象但是一直不使用这个对象，再比如通过 Connection 对象连接数据库时，使用完毕后未关闭该连接。可以使用 Memory Analyzer 等工具排出内存泄漏。

### 说一下什么是上下文切换

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。

### 你知道Java内存模型中的原子性、有序性、可见性是什么吗

并发程序正确地执行，必须要保证原子性、可见性以及有序性。只要有一个没有被保证，就有可能会导致程序运行不正确。

原子性：一个操作或多个操作要么全部执行完成且执行过程不被中断，要么就不执行。

可见性：当多个线程同时访问同一个变量时，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。

有序性：程序执行的顺序按照代码的先后顺序执行

### Volatile如何保证内存可见性

当写一个volatile变量时，JVM会把该线程对应的本地内存中的共享变量刷新到主内存。

当读一个volatile变量时，JVM会把该线程对应的本地内存置为无效。线程接下来将从主内存中读取共享变量。

### 什么是ThreadLocal类，怎么使用它

ThreadLocal为每个线程维护一个本地变量。为每一个使用该变量的线程提供一个副本，每个线程都可以独立地改变自己的副本，而不会和其他线程的副本冲突。

ThreadLocal类中维护一个Map，用于存储每一个线程的变量副本，Map中元素的键为线程对象，而值为对应线程的变量副本。

ThreadLocal在Spring中发挥着巨大的作用，在管理Request作用域中的Bean、事务管理、任务调度、AOP等模块都出现了它的身影。Spring中绝大部分Bean都可以声明成Singleton作用域，采用ThreadLocal进行封装，因此有状态的Bean就能够以singleton的方式在多线程中正常工作了。

### TCP/IP协议有了解吗

TCP/IP网络协议包含：

网络接口层，主机必须使用某种协议与网络相连

网络层，使主机可以把分组发往任何网络，并使分组独立地传向目标。IP协议

传输层，使源端和目的端机器上的对等实体可以进行会话。TCP和UDP协议

应用层，包含所有的高层协议，FTP/TELNET/SMTP/DNS/NNTP/HTTP

### Spring 中的设计模式

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。。

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。

### SpringMVC中的适配器模式

当有请求访问时，DispatchServlet会执行doDispatch()方法，

里面调用 HandlerExecutionChain mappedHandler = getHandler(processedRequest);

再调用HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());

再调用ModelAndView mv = ha.handle(processedRequest, response, mappedHandler.getHandler());

### 数据库设计的三范式

第一范式：做到每列不可拆分：

中国北京可以拆分成两个字段中国和北京

第二范式：确保一个表只做一件事情

id name sex address grade

1 tom male beijin 80

对于grade可以专门定义一张成绩表

第三范式：没有传递依赖

id name number price totalprice

1 风扇 20 200 4000

可以将totalprice字段删除，节省空间。

### sql 优化

**索引优化：**

索引是一种已经排好序的数据结构。

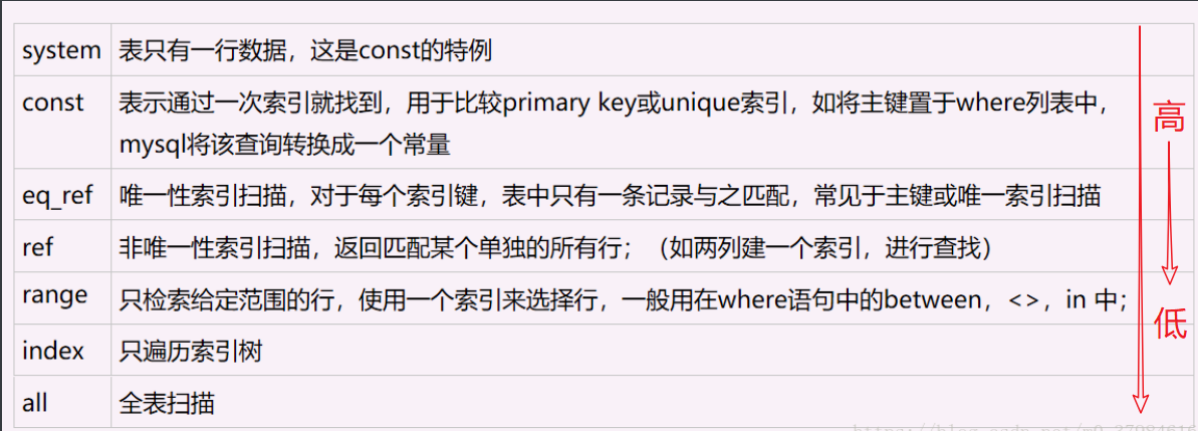
对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

B+树与B树的不同在于：

1）所有关键字存储在叶子节点，非叶子节点不存储真正的data

2）为所有叶子节点增加了一个链指针

使用 explain + sql 语句，通过返回的 type 类型，可以发现是否使用索引：



索引失效情况：

where 字段a is null / is not null

where 字段a like '%OQC%'

where 字段a != 0

where 字段a + 0 = 0

函数也会使索引失效

or 也会使函数失效

**避免使用 select \***

多查出来的数据，通过网络IO传输的过程中，也会增加数据传输的时间。

**用 union all 代替 union**

去重的过程需要遍历、排序和比较，它更耗时，更消耗cpu资源。

**右边数据少用 in，右边数据多用 exists**

in

select \* from A where id in(select id from B)

底层会查出 B 表所有的 id 并放入缓存中，循环遍历表 A 中每一个 id，如果 A 的 id 与 B 表中缓存的所有 id 中的某一个 id 有相等，就将记录放到结果集中。

如果 A 表有10000条记录，B 表有1000000条记录,那么最多有可能遍历 10000\*1000000次。B 表数据越多，效率越差。

exists

select a.\* from A a where exists(select 1 from B b where a.id=b.id)

底层会执行 A 表 id 个数次 sql 查询（ select 1 from B b where b.id = a.id and a.id=A 表某个具体 id ）。

不管 B 表有多少次记录，最终执行的 sql 次数都是 A 表 的 id 的个数个，但是这种方式会频繁进行IO操作，降低性能。

结论：exists() 适合 B 表比A 表数据大的情况。

**使用连接查询代替子查询**

执行子查询时MYSQL需要创建临时表，查询完毕后再删除这些临时表，所以子查询的速度会受到一定的影响，这里多了一个创建和销毁临时表的过程。

**join 表的数量尽量超过3个**

如果join太多，mysql在选择索引的时候会非常复杂，很容易选错索引。

**能在 group by 之前就可以过滤的数据要在 where 处过滤掉**

分组是一个相对耗时的操作，尽量在分组前就过滤掉无效数据。

### @RequestBody和@RequestParam区别

**@RequestParam**

用来处理Content-Type: 为 application/x-www-form-urlencoded编码的内容。

Http协议中，如果不指定Content-Type，则默认传递的参数就是application/x-www-form-urlencoded类型。

**@RequestBody**

处理HttpEntity传递过来的数据，一般用来处理非Content-Type: application/x-www-form-urlencoded编码格式的数据。