方法区的类被回收的三个条件？

1该类的所有实例对象被回收

2加载该类的classLoader被回收

3没有任何引用指向该类的class对象

为什么要做一个大操作后要初始化连接？

1. mysql每一个连接都又一个管理连接的对象
2. 在执行操作时占用的临时内存放在这个对象在管理
3. 只有连接释放，管理该连接的对象才会释放，内存才会释放

为什么rows\_examined的大小不一定等于存储引擎实际扫描的行数？

这就要从rows\_examined的计算方式说起了，rows\_examined是指执行器扫描到的行数，是根据执行器调用一次存储引擎接口获取到的行数累加得到的。假设执行器调用存储引擎获取到一行，rows\_examined就会+1，但是这个过程中存储引擎可能扫描了多行数据。

什么是脏读、不可重复读、幻读？

脏读，读到了一些脏数据，本质受由于事务可以读取别的事务未提交的数据导致的。举个例子，事务A将某行的数据从1改为0，修改已经完成，但还没提交。此时事务B读取改行数据的这个值，发现值为0，但事务A提交之前出现了错误，事务进行了回滚，在该行该列的值回滚成了1，而事务B读取到的值为0，事务B将0返回给程序徐，程序会根据这个值为0做出逻辑判断。举个生活中的例子，假如我和我的朋友去买衣服，我们看上了同一件衣服，我朋友先拿到了这件衣服，去商店付款，付款完成后，商店系统会修改该衣服的库存，我以看库存减为0，基于这个判断，我于是去了另外一家店购买衣服。但是由于某些原因，朋友不想要这件衣服了，于是选择当场退货，这件衣服的库存又被回复成了1。而我去了别家店买衣服，造成了商店的损失，这个原因是我读了朋友购买衣服这个事务还没有提交的数据。

不可重复读：针对数据脏读，数据库使用了“读提交”的隔离机制，防止其它事务读取某个事务未提交的数据。但都提交隔离机制无法解决不可重复读的问题，因为某个事务读取c=1的值后，另外一个事务对c进行了修改c=2，并提交，此时该事务再读c是，会读到c=2。这就是不可重复读。 不可重复读的解决办法是使用行级锁，将读取的某行数据要锁定该锁，直到该事务将所有读取操作完成，才释放该锁，允许别的事务进行修改。

幻读是针对整张表，读取数据总量前后有区

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 错误 | 描述 | 产生原因 | 针对的级别 | 针对的操作 | 解决办法 |
| 脏读 | 读取到了脏数据 | 读取了其他事务未提交的数据 | 行 |  | 读提交 |
| 不可重复读 | 同一行，前后读取的数据不一致 | 在事务读的过程其它事务修改了该行并提交了 | 行 | Update | 行级锁 |
| 幻读 | 同一张表，前后读取的数据总量不一样 | 在事务读的过程中其它事务插入了会删除了该表中的某些行 | 表 | Insert  Delete | 表级锁 |

隔离性

读未提交、读提交、重复度、串行化

乐观锁是用了解决多个事务同时尝试修改同一条数据而导致数据不一致的问题，其核心思想是，在要更新数据时检查数据是否被更新过（当前版本号<=读出来的版本号），若被其它事务更新过，则当前事务的更新作废，要重新开始

介绍一下Object常见的方法？

Object类Java类层级最最顶部的类，所有类的父类，Object有一些方法：

1、hashcode方法，它的声明语句是：public native int hashcode(); 该方法会生成一个对象的hash值，是int类型。hashcode方法设计出来是为了方便使用哈希表。

2、equals方法，该方法声明语句是public boolean equals(Object obj); 该方法用于判断两个对象是否相等，默认的实现方式是返回两个对象的地址值是否相等

HashMap、HashSet等数据结构通常不会有equals来比较，而是使用compareTo或者compare方法。因为equals方法不是强制重写的，如果某个类没有重写equls方法，就会使用默认的equlas，着显然不太合理，因为默认的是比较地址值，对于需要其它比较方法的类，实现comparable或comparator接口会比较好一些。通过阅读HashSet的源码，因为一个set里面不能有重复的元素，发现对象的相等比较是通过compare或compareTo方法。

2、toString() 方法，该方法声明语句是public String toString();。默认是返回【类名@hashcode的十六进制表示】，在System.out.print()中，对象+字符串常量，对象+字符串变量，一些情况JVM会自动调用toString方法。该方法不是必须重写的，因为有默认实现。

// 下面几个是final修饰，不允许重写

3、getClass，public final native Class<?> getClass(); 返回该类的class对象 final修饰，不可以被重写。native修饰，由本地语言实现。

4、wati，由三个重载的方法，都用final修饰。

5、notify notiyAll final修饰，不允许重写

// 下面几个是protected，无法重写

6、clone

7、finalize

Object是所有Java类的父类，里面包含一些重要的方法

1. toString toString方法返回一个字符串，默认返回类名@hashcode的十六进制，在调用print函数或者一个对象与字符串相加时等情况JVM会自动调用toString方法。自定义类通常会重写这个方法以输出人方便阅读的信息。
2. equals和hashcode，equals的返回值是boolean 默认的实现方式是用==比较，即比较地址值是否相同，hashcode返回对象的hash值，这个值与对象的内存地址有关。在用hashset或hashmap存储自定义类是通常会重写该方法，以保证存储的结果符合预期。如果没有重写，在hashmap比较两个key是否重复是会先比较hashcode值，再比较equals，其中有一个为true就认为重复，所以当两个对象hashcode不同但equals结果相同时，不会认为时重复的，依然会存入hashset中，这就会造成困惑。所以要重写这两个方法以保证它们再比较两个对象是否相同时一致。如果两个对象相等，则hashcode值应该相同，如果不相同，则hashcode值也不同。String类的源码就同时重写这两个方法，Integer类也是，它们都重写了hashcode和equals方法以保证两个对象是否相同时的一致性
3. wait notify notifyall，中三个方法用于线程同步，wait方法由被上锁的对象调用，会导致当前调用该方法的线程进入等待状态，并释放锁，直到该对象调用notify或notifyall方法，这两个方法的区别是notify方法会重等待该对象的线程中随机挑选一个线程进入Runnable状态。notifyall放啊会把所以等待该对象的线程唤醒，让它们去抢占锁，抢到锁的线程会执行同步代码，没有抢到锁的线程会进入blocked状态。这三个方法加上synchronized关键字可以实现一个生产者消费者模型。生产者生产消息后会调用notify唤醒消费者，消费者消费完消息后会调用wait进入等待状态。同样当队列满后生产者会调用wait进入等待状态，当消费者消费消息后会notify唤醒生产者。

Java中Object类不是一个抽象类，为什么可以声明一个没有函数体的函数？

我们以hashcode函数为例，hashcode函数声明是public native int hascode();从形式上看上去是没有函数体，这是由于native关键字存在，它告诉JVM该函数的实现有本地语言C/C++，而不是Java语言来实现，所以看上去是没有函数体，实际上是有的。

为什么hashtable不能把空作为key？

因为hashtable必须要存储的对象实现hashcode和equals方法，该类会调用

为什么哈希表这个数据结构要使用到hashcode函数？

由hashtmap源码可知，在比较key是否重复是，会调用对象的hashcode函数计算hash值，如果哈希值映射到hash表上的索引以及存放了对象，会进一步比较这两个对象用hash值比较、==比较、equals比较

为什么byte类型的数据取值范围是[-128,127]？

简单一句话来说就是-128在8位带符号整数是没有直接的机器数表示的，-128是人们将-0作为8位带符号整数的最小值

什么是机器数、真值、原码、反码、补码？

机器数：数在计算机中二进制表示，用于计算机的存储和表示

真值：由于机器数符号位的存在，直接将机器数转换位十进制的结果不一定正确。真正就是考虑到符号位后机器数的真正的数值

原码=符号位+数值位

反码：正数：反码=原码；负数：反码=原码出符号位外按位取反

补码：正数：补码=原码；负数：补码=原码的反码 + 1

String a = “a” 和String b = new String(“b”)经历了什么过程？为什么没有像Integer那样，String.valueof()没有被调用？用图能否画出来？用文字能否描述一遍？

1、与Integer、Double、Float、Bollean、Long类型不同，String直接就是引用数据类型，没有对应的基本数据类型，所以不存在装箱和拆箱的操作，也就没有调用String.valueOf

2、字符串有常量池的概念，位于JVM内存的方法区中。



程序编译为字节码后被加入到JVM内存进行加载。程序执行到main方法时会分配一个mian方法的栈帧用于存放局部变量。执行到String a 时会在main栈中声明一个a的变量，然后程序进一步执行发现等号右边是一个字符串常量。这是JVM会直接去方法区的常量池中查找有么有字符串“a”，如果有就将其地址返回给变量a，如果没有，就在常量池中创建字符串常量”a”，并将地址范围。对于第二条语句，前面在main栈中创建b变量与第一条语句一致，不同的是等号执行到右边，JVM发现是new String，所以JVM会在堆中创建一个String类对象，这个String类对象里面有一个成员变量value，用private final 修饰，是一个字符串数组，value本身是一个数组引用，然后，JVM回去方法区中的常量池中查找有没有字符串”b”，如果没有就创建。将常量池中”b”串的地址值传给value，即value指向了常量池中字符串”b”。最后，将这个在堆中的String对象的地址值传给b。

什么是元数据，metadata？

元数据是指用于描述其它数据的数据。举个例子，一本书的内容就是数据，而描述这本书的出版日期、作者、ISBN号等这些就是元数据。一个类的成员包括变量和方法，变量和方法的具体内容就是数据，而一个类有哪些字段或方法是类信息(元数据)，类名、父类、实现的接口这些描述类的数据就是元数据

方法区用来存放哪些内容？

类信息、常量、静态变量

为什么说一个String类对象是保存一个字符串常量？

这个可以通过阅读String类的源码得知，String对象保存的数据本质上是一个字符数组，这个字符数组是这样定义的private final char value[]；关键字final修饰，表示这是一个字符串常量。

char a = ‘我’ 对不对？为什么？

这是对的，因为Java中字符使用Unicode编码，一个char类型占两个字节。

你能想到有哪些String类的构造器？



一些常用的String类构造器

值得注意的是，new String(String original) 这个构造函数会执行两个步骤

this.value=original.value

this.hash = original.hash

注意第二个，hash值也会传过来。所以两个不同地址的变量，hashcode值的结果是相同的，因为hash值会复制过来

实现comparable和comparator接口有什么区别？

comparable需要通过类对象调用comparTo方法。comparator可以用第三方类调用comapare方法比较两个类对象。

串行化是什么？为什么要串行化？为什么实现了串行化就可以在网络中传输？串行化的反义词是什么？

串行化是将对象变成字节流。在网络通信中，只有基本数据类型和字节流可以传输。原因：网络传输TCP/IP协议是以字节为基本传输形式，更高级的数据结构这些网络协议是无法识别的，它们只能发送或接收字节流，更底层数据链路层或物理层的协议连字节的概念都没有，只是发送bit流。而基本数据类型可以直接转换为字节，如btte是1个字节、short和char类型是2个字节、int类型是4个字节等 long类型是8个字节

实现charsequence接口有什么用？

实现charsequence可以统一对字符串序列的使用规范。该接口定义了length、charAt等方法

为什么说“字符串的本质是char数组”?

由String类的源码可知，真正存储字符串的是字符数组

final int[] v = {1, 2, 3} 为什么v[0] = 4可以运行？不可修改值的是什么不可修改？

final修饰变量表示该变量的值不能修改，这里v是只想堆中数组的引用，存储的是地址值，所以修改v的地址值，即指向一个新的int数组区域是不允许的，但用v[0]修改的是堆中某个区域的值，这不是产量，是可以改变的。所以这里的不可修改是指v的指向不能修改。

Java中声明数组有哪些方式？

int[] a;

int a[];

String类为什么hashcode相同的字符串，并不一定指向同一个内存地址即s1!=s2?

hashcode的计算通常与内存地址相关，如果hashcode相同，内存地址应该也相同。但是对于这个情况：String a1 = “a”; String a2 = new String(“a”); a1和a2的地址值是不同的，前者指向常量区、后者指向堆区。但在第二条构造语句执行时会执行this.hash = “a”.hash; 于是a1.hashcode是等于a2.hashcode值的。

String s = “hello” + “abc”; 创建了多少个对象？

JVM会帮我门优化，等价于String s = “helloabc”; 编译器会看一下，所以一个创建了一个对象。编译器会判断常量池中的对象是否有引用变量指向。

String a = “hello”; String b = “abc”; String c = a +b; 这个过程创建了多少个对象？

关键是分析String c = a + b是怎么执行的？这个过程可以debug一下(可以的)？

这个过程最终池中是3个对象

画出下面这段代码的内存动态图？做一个视频？





Java8之后方法区被metaspace代替?

首先这个说法是错误的，要先弄清楚几个概念：

方法区：JVM在运行时管理的一块数据区域。方法区只是一个规范，一个定义。具体实现，不同版本的JVM由出入。

永久代：对于HotSpot虚拟机，在jdk1.8之前，方法区的具体实现叫做Permanent Generation

元空间：对于HotSpot虚拟机，从jdk1.8开始，PermG被舍去，方法区的具体实现叫做Metaspace

Metaspace存储class metadata，描述类的信息、方法的元数据、静态变量、常量池，字节码

存储已被虚拟机加载的类信息、常量、静态变量、即使编译器编译后的代码缓存。

你能不能手写一个泛型类？

可以。

Java Object wait方法有哪些重载？是用来干什么的？为什么要用它？为什么出现在Object类中？

wati(timeout) 这个是native的，由本地语言实现

wait(timeout nanoseconds)

wait() 实际调研wait(0) 表示无限期等待

该方法只能被上锁的对象调用，synchronized(对象){

对象.wati(); // 执行该同步代码的线程会主动进入wait状态，并释放锁，直到该对象发出notify通知或者notifiyAll通知

}

进程和线程的区别？

进程是程序的一次执行过程，或者是一个运行的程序就是进程。

代码(程序)---运行🡪 进程🡪占用内存空间。

线程：由进程创建的，它是进程的一个实体。一个进程可以拥有多个线程

进程是操作系统资源分配的最小单位，用于个理资源，比如内存资源，线程是CPU调度的最小单位，线程包含在进程中，是进程实际运作单位。

什么是并发和并行？

并发：单核CPU，同一个时刻只有一个线程在执行，不同时刻切换到不同的线程进行执行，但由于CPU太快了，人看起来是同时在执行，这就叫并发

并行：多核CPU，同一时刻有多个线程同时在执行，这就是并行

对于多核CPU，并发和并行也有可能是同时进行的

创建线程的方式？

Java中创建线程有两种方式

1、继承Thread类，重写run方法

2、实现Runnable接口，重写run方法

3、通过Callable接口和Furture类实现

4、通过线程池创建

为什么有了继承的方式创建多线程，还要发明接口的方式？

因为Java是单继承的，如果某个类已经继承了其他类，再使用继承的方式实现多线程就无法实现。

如何停止一个线程的运行？

1. 自动方式：线程完成自己的任务后就会自动退出
2. 通知方式：使用变量控制run方法退出，通知该线程此时需要结束run方法了

Runnable和Callable创建线程有什么区别？

1. Callable方式创建线程可以得到返回值，重写call方法是由返回值的，而Runnable重写的run方法没有返回值
2. Runnable里面重写run方法不能抛出异常，Callable可以
3. Runnable和Callable都需要用Thread.start启动线程
4. Callable需要用FutureTake.get获取返回值，当在主线程调用该方法时，主线程会阻塞，直到结果返回，不调用该方法，主线程不会阻塞。

Thread类和Runable接口的最大区别是什么？

1. 从本质上来将，继承Thread类的方式实现多线程和实现Runnable方式没有本质的区别，因为从Java源码可以发现，Thread类本身也实现了Runnable接口。
2. 实现Runnable接口更加适合多个线程共享一个资源的情况，并且避免了单继承的限制。

为什么说实现Runnable接口更加适合多个线程共享一个资源的情况？

凭什么说这句话，当多个线程需要共享同一个资源时，例如访问同一个文件、数据库连接、网络连接时，我们把这些资源作为一个对象的属性，并写入调用这些资源的方法，然后实现Runnable接口，在run方法里面调用上述对象的方法。只需要创建一个资源对象，当创建新的线程时，就把该对象传入。从而让多个线程共享同一个资源。

如果采用继承Thread类的方式，创建一个新的线程，就会new一个新的对象，没有线程拥有一份独立的内存空间，每个线程都会拥有自己的资源、而不是共享一份资源。会造成资源的浪费

线程的几种状态，以及状态的转换，生命周期？



什么是线程安全？为什么不安全？

线程安全不是指线程的安全，而是指内存的安全。这与操作系统有关，现在主流的操作系统都是多任务的，即多个进程同时运行，为了保证进程之间互不干扰，操作系统会给每个进程分配一块内存区域，每个进程只能访问自己的内存空间，而不能访问别人的内存空间。每个进程的内存空间会有一块特殊的公共区域，被称为堆，进程中的所有线程都可以访问，这就是造成问题的潜在原因。线程安全是指某块内存，或者某个对象，在被多个线程同时访问，线程间按照某种方式访问时，访问的结果总是正确的，就说明这个对象、这个内存时线程安全的。

1. 线程安全不是指线程的安全，而是指内存的安全或者对象的安全
2. 当多个线程同时访问同一个对象时，线程间按照某种调度方式访问，调用这个对象的行为都可以获得正确的结果，就称这个对象时线程安全的，反之称这个对象是线程不安全的。

为什么不安全：由于堆内存的数据可以被任何线程访问到，在没有限制的情况下存在被意外修改的风险，堆内存空间在没有保护机制的情况下，对多线程是不安全的，因为放进去的数据，可能被别的线程破坏。

线程安全不是指线程的安全，而是指内存的安全，更具体一点，是指对象的安全，因为JAVA中对象是存放在一块被称为堆的公共内存区域，任何线程都可以访问。如果没有保护机制，某个线程存放到堆中的数据有可能被其它线程修改的风险。而如果多线程并发访问同一个对象，按照某种方式进行访问，调用对象的行为都是正确的，我们就称这个对象是线程安全的。

一个类中有类变量、成员变量、局部变量，它们存放的位置分别是？

方法区、堆、栈内存

前两个是共享变量，需要考虑线程安全。最后一个是线程安全的，因为每一个线程，JVM都会分配一个线程栈，而局部变量存放在各个线程的栈空间中，线程之间是无法访问对方的栈内存的，栈内存空间是独立的空间，所以是线程安全的。

阻塞状态和等待状态的区别？

两者都是当前线程暂停执行的状态，阻塞状态时被动的、等待状态是主动的。更近一步，阻塞状态实在同步代码之外，等待某个对象的锁，等待状态是获取到某个对象的锁后，进入同步代码后主动执行wait函数，主动释放锁、等待其它进程使用完该对象后发出notify或notifyall，然后结束等待状态，进入runnable状态。

怎么实现线程安全？

1. 无状态实现。比如一个计算阶乘的静态方法、给的特定的输入就会输出特定的结果，不依赖外部状态，也不维护自身状态。因此，该方法是线程安全的。
2. 不可变实现。想要保证某个对象是线程安全的，在声明该对象的类时将所有的字段声明为privat 和 final 并且不提供setter。那么，该对象一定是线程安全的。
3. 线程私有ThreadLocal字段。

锁怎么实现线程安全？

以同步锁为例，java sychronized可以实现对对象加上同步锁，sychronized实现线程同步由两种方式：同步代码块、同步方法。以同步代码块为例，它的使用格式是synchronized (对象){} 要求多个线程上锁的对象保持一致、不然就失去了意义。花括号里面是执行同步代码，同步锁保证在任意一个时刻最多只有一个线程在执行该同步代码，因为对象只有一把锁，哪个线程获得了就可以进来执行，在当前线程执行完同步代码里面的任务时，其它线程只能被阻塞，其它线程也无法访问共享的内存空间、这是就保证了共享资源的线程安全。

多线程什么场景下会发生死锁 ， 有什么具体的办法可以避免死锁？

多个线程占用了对方锁资源，需要等待对方将锁释放后，自己才能释放锁，结果对方线程也是这样等待，大家都不想让，最终造成死锁。

首先是要在写业务代码的时候避免多个线程给对方的资源上锁，然后在发现死锁的时候让其中某个线程主动释放锁，先让别的线程完成任务。

避免死锁的方式：

1. 按顺序加锁
2. 获取锁的时限，获取某个锁加上时限，超过这个时间就释放锁
3. 死锁检测

wait和sleep的区别，没有说清楚，HR提醒用wait时会释放锁，sleep不会?

wait和sleep都会让线程进入Waiting状态或Timed\_Wait，如果wait加上timeout参数，则也会进入Timed\_Wait状态。而wait在让当前线程进入Waiting状态时会主动释放锁，sleep则不会。另外如果不加timeout参数，就会无限期的等待该对象调用notify

1. wait是Object里面的，所有对象都可以调用，Sleep是Thread类所有的
2. wait是非静态成员方法，需要通过对象调用，sleep是静态成员方法，可以通过类名调用
3. 调用wait可以不需要参数，调用sleep必须要参数
4. sleep会抛出interrput异常
5. wait让当前线程进入等待状态并释放锁，sleep让当前线程进入等待状态不会释放对象锁。

多线程了解吗，为什么要多线程？

为什么要多线程：第一，假设我们的程序只有一个线程，当该程序处理某个比较耗时的任务、如网络传输、文件读写时，其它任务会被阻塞，只能等待该任务执行完毕，在用户看来，程序就卡住了，不会执行其它请求，这对用户的体验非常不好。如果采用多线程， 用一个线程去处理比较耗时的任务而不会影响主线程的执行，程序还是可以接收用户请求完成其它任务，所以多线程可以提高程序的并发现和响应性。第二，由于硬件的发展，现在的CPU多是多核的，可以并行执行多个线程，使用多线程可以提高CPU的利用率。

总的来说，为什么要用多线程分为两个方面: 提升用户体验；提高CPU利用率。

如何启动一个线程？

1. 某个类继承Thread类或者实现Runnable接口
2. 重写run方法，这里面是业务方法
3. 创建该类对象
4. 如果是接口方式：创建Thread类对象，将该类对象传入
5. 使用该类对象调用start方法，如果是接口方式：使用Thread类对象调用start方法

为什么用start不直接用run？

启动一个线程、start、最终会调用run方法。如果直接在mian方法里面调用run方法，相当于直接用主线程栈里面调用方法，并没有真正实现多线程。run方法就是一个普通的方法，相当于是一个串行化的执行。而start才会真正地去启动线程。在start0里面用多线程机制调用run方法。 start调用native的start0后，start0会启动一个新的线程，并调用run方法，然后将该线程变成可运行状态，等待CPU时间片，run方法具体什么时候执行是由CPU决定的，而不是start0决定，由CPU统一调度。run方法只是一个普通方法。

什么叫同步机制？

从以下两个方面理解同步机制：

1. 同步机制是保证在任意一个时刻t，最多只有一个线程在访问某个资源。
2. 同步机制：当有一个线程在对内存进行操作时，其它线程都不可以对这个内存地址进行操作，直到该线程操作完成，其它线程才能对该地址进行操作。
3. 假设有三个线程t1,t2,t3，t1在操作某一个资源，这时如果t2和t3也想操作该资源，就会被阻塞，直到t1操作完成之后，t2或t3中某一个线程在被选择来操作该资源，这就叫同步

多线程环境下为什么要引入同步的机制？

1. 在多线程编程中，一些敏感数据不允许被多个线程【同时访问】，所以引入同步机制，保证数据在任意一个确定的时刻，最多只有一个线程访问，以保证数据的完整性。
2. 引入同步机制，是为了控制多线程对共享资源的访问，以保证数据的正确性和一致性。

知道了什么是同步，为什么要同步，现在问在JAVA里面如何实现同步呢？

1 最简单的方式，加Synchronized关键字。

Synchronized有两种情况：同步代码块、同步方法

同步代码块 synchronized (对象){ //得到对象的锁，才能够操作同步代码。

// 同步代码，任意一个确定的时刻，最多只有一个线程在执行该部分代码

}// 操作完成后，归还对象的锁。对象要求是同一个对象，如果对象一直在变，上锁就没用了。

同步方法

public synchronized void m(String name){

// 需要被同步的代码

}

在任意一个时刻，最有一个线程在执行该方法

同步方法：同步非静态方法、同步静态方法。同步非静态方法等价于给this对象加锁==synchronized(this){}。同步静态方法等价于给该类的class对象加锁

synchronized有两种实现方式：同步代码块、同步方法；但本质上只有一种：即同步代码块。

同步代码块：synchronized(对象) 要求不同线程的对象必须是同一个内存空间且

同步静态方法==synchronized(当前类.class){}

同步方法==synchronized(this){}

必须保证多个线程锁的对象是同一个对象

为什么要线程安全，举例子，说了数据库脏读的例子， Java方面不清楚，面试官给了非元子性的操作 i++的例子怎么实现线程安全？

线程安全本质是内存的安全，java中由于堆是一公共的内存区域，任何线程都可以访问，当某个对象被多个线程共享时，当前线程操作块内存区域时有可能其它线程也修改了内存区域，这导致对象的行为结果不确定或不正确。举个例子，Java实现一个售票系统，一共有3个售票窗口，开启了三个线程并发地卖票，这3个线程共享剩余票数这个变量，每个售票窗口在执行售票业务代码时会判断剩余票数是否大于0，如果小于等于0，就提示票已经售空，否走就执行售票业务代码，并将票数减一，若售票窗口1此时访问剩余票数这个变量，发现为1，于是判断成功，准备执行售票业务代码，但售票窗口2比售票窗口1先一步拿到，先将票数减1了，但售票窗口1已经进入了售票逻辑，最后剩余票数就会变成-1

i++操作为什么当i是成员变量时是线程不安全的？

1. 线程安全本质上是内存安全，多线程访问同一个内存空间时会导致线程安全问题
2. 如果i是局部变量，i会存放在栈中，而JVM会给每个线程分配一个栈，这个栈空间是相互独立的，线程之间无法访问，所以i如果是局部变量，就一定是线程安全的。
3. 当i是成员变量是，i所属的对象是存放在堆中的，这块内存是任意线程都可以访问的，所以可能存在线程安全问题。
4. 在3的基础上，i++不是一个原子性的操作，i本身是在堆中，i++会被看作i=i+1，当前线程有一个线程栈，首先会把在堆中的i的值读取到栈中tp=i，压入栈顶，然后将常量1压入栈顶，接着计算tp2 = tp +1，将tp2压入栈顶。最后把栈顶的tp2重新写回堆中的i。综上所述：当i是成员变量时，i++的过程可以看成3步：读取i的值、计算i+1、向i写入结果。
5. 所以i++不是一个原子操作🡪在线程1计算i++还没写入时，如果线程2就读取了一个脏数据，所以这些线程不安全的。

Java里面的锁有哪些？

1、互斥锁，synchronized(非公平锁，才释放锁的线程可以立即去抢锁)

java里面的线程和操作系统的线程一样吗？

不一样

Java中线程实在JVM上执行的线程，是一种轻量级线程，由java语言支持，并受到JVM控制。

操作系统线程是在操作系统中执行的线程，是一种重量级线程，由操作系统支持。并受到操作系统控制。

Java线程实在操作系统线程之上进行封装的，它们可以映射到操作系统线程上。但是Java线程不能直接访问操作系统线程。必须通过JVM间接访问。

wait-notify、线程的状态？

Waiting状态

notify 和 notifyAll 的区别？

某个对象调用notify和notifyAll都需要当前获取到该对象的锁。

对象.notify会随机唤醒一个在等待当前对像的线程，使其进入runnable状态

对象.notifyAll会唤醒所有在等待当前对像的线程，让他们竞争该对象的锁，没有竞争到的线程进入Blocked状态。

线程怎么才能称为一个对象监视器的所有者？

线程获取到对象的锁即可，synchronized(读写)、同步方法、同步静态方方法

调用某个对象的wait和notify/notifyAll方法都需要当前线程称为该对象的监视器所有者，不然就会报IllegalMonitorStateException异常

什么叫程序？

为了完成某种任务，用某种语言编写一组指令的集合。通俗地将，就是我们写的代码就叫程序。代码就是程序

什么是进程？

运行的程序叫做进程

什么是静态代理，什么是动态代理，他们有什么区别？

静态代理和动态代理都是设计模式中的代理模式。代理模式本质是：利用一个中间代理对象来控制原目标对象，以达到增强原目标对象的功能。

代理提供了对目标对象的额外访问方式，即通过代理对象访问目标对象，这样可以在不改变目标对象的前提下、提供额外的功能、拓展目标对象的功能。

静态代理：之所以叫静态，是因为目标对象和代理对象实现同样的接口，在编译阶段编译器就知道目标对象要代理的方法，不需要等到程序运行起来。优点是可以在不修改目标对象的前提下拓展目标对象的功能。缺点是：1 由于要代理类和目标类要实现同一个接口，会导致代理类的数量增多，冗余。2 不易维护，如果接口里面的方法变化或着要增加方法，那么，所有的实现类都要修改。

动态代理：又称为JDK代理或接口代理。要求目标对象必须实现接口。代理对象根据目标对象实现的接口动态的在内存中构造代理对象。

它们的区别在于：1、静态代理在编译阶段已经实现，编译完成后代理类是一个class文件。2 动态代理实在运行阶段动态生成的，编译后代理类没有class文件，而是动态生成字节码，加载到JVM中。

Java中new一个对象时，初始化过程是什么？

类加载器加载类信息、首先执行类层级别代码(静态成员变量、静态代码块)

分配对象空间，并将对象中的成员变量设置为默认值

执行初始化代码块

执行成员变量初始化语句，（成员变量声明语句，没有初始化的用默认值，不会执行）

执行构造函数中的代码

一个类中由哪些东西？

类级：静态成员、静态代码块

对象级: 成员、代码块

构造方法

Java各个类各个部分的执行顺序是什么【代码块部分】？

类级别的部分在JVM加载类的时候就会执行，JVM加载类的时机有：调用该类的静态成员、声明该类的引用、构造该类的对象

对象级的部分只有在构造该类对象的时候才会调用

执行顺序：

类层级🡪成员变量默认值(分配对象空间是就会初始化一个默认的值)🡪成员变量初始化语句🡪构造方法

类层级🡪对象级🡪构造方法



Integer对象是不可变的？

是

所有的包装类都是不可变的吗？

是的，包括String类也是。

synchronized (number) {

if (number <= 0) {

System.out.println("售票结束...");

break;

}

try {

Thread.sleep(50);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("窗口" + Thread.currentThread().getName() + "售出一张票，剩余票数=" + (--number));

}

为什么这段上锁没有成功？

1. 上锁不是给引用上锁，而是给引用指向的内存区域上锁
2. Integer类是包装类，所有的包装类、String类都是不可变类，在Integer类真正存储数据的是pubic final int value，由final修饰，所以Integer类是不可变类
3. 当执行number++的时候：number会指向一个新的内存地址，其它线程进来一看，发现该内存地址没有上锁，于是对其进行操作。导致结果依然不正确。

Java使用synchronized上互斥锁时，什么时候会释放锁？

1. 当前线程的同步代码块、同步方法执行结束。
2. 在同步代码块、同步方法中遇到break、return语句。
3. 在同步代码块、同步方法中遇到未处理异常或者Error，导致异常退出。
4. 在在同步代码块、同步方法中执行了线程对象的wait方法，当前线程暂停，并释放锁。

什么时候不会释放锁？

Thread.sleep()

Thread.yield()

线程执行同步代码时，其它线程调用了该线程的suspend方法，也不会释放锁

挂起还是在Runnable状态

如何实现一个生产者-消费者模型？

1. synchronized关键字：解决线程安全问题，不会出现不正确的结果
2. synchronized无法解决忙等状态，就是生产者一直抢占cpu时间片判断队列满或消费者一直抢占cpu时间片判断队列是否空

什么是Spring？

Spring是一个轻量级、非侵入式的IOC和AOP框架。

轻量级：spring提供了很多服务，但这些提供给用户的服务完全由用户决定，想用什么服务，就用什么服务，而不是一股脑的全部给用户。

非侵入式：非侵入性是引入了框架(jar包)，对现有的类的结构没有影响。

IOC：控制反转

AOP：面向切面编程

从狭义的角度：spring就是指spring框架。



从广义的角度：spring生态体系。

如何能够使用spring？

spring本质也是Java语言写的，由很多Java类组成。相应使用spring的服务，就要引入相关jar包，jar包里面包含很多类。这些类提供了相应的服务。

引入jar包这个事情很麻烦，可以通过maven管理jar包，只需创建一个maven工程。

需要用到什么服务就引入对应的依赖

spring发展历程？

1.X 支持IoC AOP

2．5X 支持注解

3.X 基于java 5

4.X 支持Spring boot 1.x

5.X 基于java8



Spring 有哪些特性？

1. IoC和DI的支持：Spring核心就是一个大的容器工厂，可以维护所有对象的创建和依赖关系。Spring工厂用于生成Bean，并且管理Bean的声明周期，实现高内聚，低耦合的设计理念。高内聚是指模块内、低耦合是指模块间
2. AOP编程的支持，Spring提供了面向切片编程，可以方便实现对程序进行权限拦截、运行监控等切面功能
3. 声明式事务的支持，支持通过配置来完成对声明式事务的支持，只要声明某个方法是事务，不需要通过硬编码方式。以前重复的一些事务提交、回滚的JDBC代码，都可以不用自己写了。
4. 快捷测试支持：Spring提供对Junit支持，可以通过注解快捷地测试Spring程序
5. 快速集成功能：方便集成各种优秀框架，Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架的直接支持
6. 复杂API封装：Spring对Java EE开发中非常难用的API 如JDBC JavaMail、远程调用等都提供了模块化封装，这些封装API使得使用难度大大降低。

Spring 有哪些模块呢？

Spring Core：Spring框架核心模块，提供了IoC和DI特性

Spring Context：Spring上下文容器，它是BeanFactory加强的一个子接口

Spring Web 提供web开发支持

Spring MVC 针对Web开发中MVC思想实现

Spring AOP 面向切面编程，提供了与AOP联盟兼容的编程实现。

Spring 有哪些常用注解呢？

1. Web
2. 容器
3. AOP：
4. 事务：

@Transactional：该注解作用于要开启事务的方法，即声明式事务开启



Spring 中应用了哪些设计模式呢？



工厂模式：Spring本质是一个大工厂，使用工厂模式通过BeanFactory、ApplicationContext创建bean对象

代理模式：Spring AOP就是通过代理模式实现的，分为静态代理和动态代理

单例模式：Spring中的Bean默认都是单例的，这样有利于容器对Bean的管理

模板模式：Spring JdbcTemplate, RestTemplate等以template结尾对数据库、网络连接进行操作的模板类就用到了模板模式

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的增强或通知使用了适配器模式

策略模式：Spring中有一个Resource接口，它的不同实现类会根据不同策略取访问资源。

说一说什么是 IOC？什么是 DI?

什么是IOC：

1. IOC全称是Inverse of Control，IOC是一种设计理念。
2. 它是指将对象的控制权交由第三方管理。由代理人负责创建和管理对象，消费者通过代理人获取对象。
3. Ioc的目的是降低对象之间直接耦合
4. 加入IoC容器将对象统一管理，让对象关联变为弱耦合

什么是DI依赖注入：

1、IoC是设计理念，是宏观目标

2、Dependence Injection是实现IoC理念的具体的技术实现，是微观实现

3、将某个类所依赖的对象实例注入给这个类。

4、DI在java中是利用反射技术来实现运行时对象的创建与关联。

所谓IoC就是将我们对象的控制权，由程序本身转嫁给第三方，这个第三方通过一个容器，统一地对我们系统中的对象进行管理，这就是IoC的理念。

spring IoC容器的职责？

对象的控制权交由第三方统一管理(IoC控制反转)

利用Java反射技术实现运行时对象的创建于关联(DI 依赖注入)

基于配置提高应用程序的可维护性和可扩展性。

什么是反射？

反射技术是指在程序运行过程中可以拿到一个对象的信息。

如何对spring IoC进行配置，Spring IoC核心配置文件是什么？

在resource目录下创建application.xml文件，这是spring IoC核心配置文件。所有对象的创建以及对象之间如何关联就是在这个文件中设置的。

XML需要有一个xml头，XML声明，schema定义，定义xml里面的标签以及格式。有了这些头部内容，spring框架就可以去解析该xml文件了。

程序中绝大部分静态的信息不用硬编码到程序中，而是采用配置的形式放到xml文件中，一旦这些信息发生变化，不需要重新编译，而是直接修改xml文件就行了

idea一个project里面有module，一个module里面有facets，这个facets是什么？

facets是一个额外的框架特定的配置。

聊一聊spring实例化对象？

spring实例化对象是通过IoC容器，IoC容器的实现机制在程序运行阶段，利用反射技术获取对象所有信息，然后动态地进行实例化。实现IoC容器有三种方式：基于XML配置文件，基于注解 基于Java Config类。总体上分为两种：基于XML配置方式，基于注解方式。每个方式都各有优劣，使用XML配置方式将对象的创建于关联写入配置文件，便于后期的改进于维护，也便于项目切分。而使用注解方式则摆脱了繁琐的XML配置，对于程序员编码更加方便。

spring管理Bean的方式有哪些？

基于XML配置文件

基于注解

基于Java代码config配置

spring如何通过xml方式管理对象(Java Bean)？

引入依赖

创建applicationContext.xml

引入xml头

使用bean标签 id class属性

property子标签 name value ref属性

Application 是一个接口，有很多实现类，ClassPathxmlApplicationContext是实现类，会读取applicationContext.xml文件并初始化IoC容器并创建对象。这个contex就是ioc容器

三种XML实例化Bean的方式？

1. 基于构造方法实例化对象（大多数常见）
   1. 基于默认构造方法实例化对象
   2. 基于有参构造方法实例化对象

constructor-arg 子标签

property子标签是调用Bean的set方法，没有走构造函数。

在IoC容器创建对象时去调用对象的有参构造函数构造对象

constructor-arg 有name属性 index属性

index是通过位置设置对应构造方法参数的值

根据constructor-arg标签数量去调用对应的构造方法

有多个构造方法时会根据name、数量、type来查找调用哪个构造方法

xml只能写字符串，调用对应的构造方法时会自动转换

尽量按名称设置构造方法的参数，尽量不要按照索引

使用constructor-arg标签表示使用带参构造，有两种该方式设置参数：名称和索引。具体调哪个构造函数与参数数量、name、type有关

1. 基于工厂（很少用）
   1. 基于静态工厂实例化对象

静态工厂通过静态方法创建对象，隐藏创建对象的细节‘

bean标签里面class写上静态工厂类，加上factory-method对象

* 1. 基于工厂实例的实例化方法实例对象

1. IoC容器对工厂实例化
2. 工厂实例化对象创建对象

需要两个bean，第一个bean指向工厂本身， 第二个bean是apple，属性有factory bean 和factory method

这两种方法最大的区别就是创建对象的方法是不是static的，如果不是，还需要多加一步，spring ioc创建实例化工厂实例，所以要多一个bean标签

基于工厂的方式还是会把new对象写入源代码中，这和原来的方式有什么区别吗？

作为工厂来说，其职责就是通过工厂创建对象，隐藏创建细节。其主要目的是在IoC容器之外，通过程序的方法帮助我们组织对象，这样做有一个好处，就是扩展spring IoC容器的功能

如何从IoC容器中获取Bean？

从ApplicationContex的实现类对象调用getBean方法

getBean可以传入类，也可以不传入后面用强转。

Bean标签 id 和 name属性的相同点和不同点？

相同的：

1. 作为id和name都是为对象设置在IoC容器中设置唯一标识，本质来讲都是一样的。
2. 两者在配置文件中都不允许重复，即使id与name之间也不能重复。
3. 如果有多个配置文件，是会允许重复的，后加载的对象会覆盖先加载对象，但指向它们的id name不会变，堆中的对象位置不会改变，内容会被覆盖

不同点：

一个对象可以有多个不同的name，但只能有一个id

当一个Bean标签既没有id也没有name，我们该如何获取呢？

要想从IoC容器中获取该bean，需要在getBean中添加该bean所属类的完整名称，getBean(“com.imooc.spring.ioc.entity.Apple”，Apple.class)

什么是工厂实例？

工厂实例是和静态工厂相对的，静态工厂的方法是用static修饰属于类层级。而工厂实例需要创建工厂实例对象在调用实例方法创建对象。

工厂模式是什么？

隐藏创建对象的细节，通过一个额外的工厂类组织要创建的对象

聊一聊IoC容器中加载配置文件时所使用的路径表达式？

什么是路径表达式：指在我们加载spring ioc配置xml文件是传入的特殊格式的字符串—“classpath:applicatonContext.xml”

作为ApplicationContex接口的实现类，ClassPathXmlApplicationContext类需要传入一个路径表达式告知这个类去哪里找到xml配置文件。

classpath 类路径是指编译后，有一个classes文件夹，里面有Java类顶层包以及其它文件，这就是类路径，Idea会把resource路径下的自动文件复制一份到类路径下。这就是路径表达式的意义所在

|  |  |
| --- | --- |
| 路径表达式 | 说明 |
| classpath:config.xml | 扫描当前项目的classpath路径下 |
| classpath\*:config.xml | 扫描当前项目的classpath路径下和所有jar包下的classpath路径下 |
| classpath:com/imooc/config.xml | 扫描当前项目的classpath路径下com/imooc路径下 |
| classpath:com/imooc/\*.xml | 扫描当前项目的classpath路径下com/imooc路径下所有以xml结尾的的文件 |
| classpath:com/\*\*/config.xml | 扫描当前项目的classpath路径下com路径下：任意子路径，一层或多层子路径 |
| file:c:/config.xml | file表示采用文件系统路径  c盘根路径下的config.xml |

能简单说一下 Spring IOC 的实现机制吗？

IoC是在运行时利用反射技术，在运行时，动态地将对象实例化，以及利用反射技术，对属性进行注入

IoC容器本质是一个map键值对对象，将beanId与对应的对象进行绑定，如果需要创建对象，就利用Class.forName获取要创建对象的类的类对象，然后调用newInstacne，使用默认的构造函数构造对象。如果遇到proerty标签，则利用method对象进行方法的动态调用。

说说 BeanFactory 和 ApplicantContext?

ApplicationContext继承BeanFactory接口，所以ApplicationContext用于BeanFactory接口的所有的功能

BeanFactory是使用懒加载方式，意味着只有在getBean的时候才会区实例化对象，而ApplicationContext接口在容器启动阶段就会去加载scope为singleton的对象

类对象的getResource(name)方法，name加 /和不加 /的区别

如果加/：会从classpath路径下开始查询

如果不加/：会把name看作时一个相对路径，相对于当前类所在的文件夹

聊一聊 spring IoC注解？

除了用XML文件配置Spring IoC容器外，Spring还支持使用注解和JavaConfig方式配置IoC容器。这三种方式只是表现形式不同，其本质原理都是一样的。

如何利用注解配置IoC容器？

注解是对所描述的类或者方法、变量进行额外的功能拓展或者增强。作为Spring，提供了大量的注解

注解的优势：摆脱繁琐的XML形式的bean与依赖注入的配置

注解是写在源代码中的配置信息，基于声明式的原则。因为注解本质上是一种声明，在代码中使用注解时，本质上是指声明一些信息，注解本身不会完成它所声明的功能，而是实现类利用反射机制，扫描某个类、方法、变量上有什么样的注解，再进一步执行功能代码。所以所注解的本质是声明。

利用注解可以让代码的可读性变得更好

Spring中，按照注解功能划分可以划分为哪几类注解？

1. 组件类注解：声明当前类的功能与自责
2. 自动装配注解：更加属性特性自动注入对象、或者注入值
3. 元数据注解：更细化辅助IoC容器管理对象

|  |  |  |
| --- | --- | --- |
| 类型 | 注解 | 说明 |
| 组件类注解 | @Component | 组件注解，通用注解，告知spring，当前类应该被IoC容器管理并实例化 |
| @Controller | 说明当前类是MVC引用中的控制类，Controller层 |
| @Service | 说明当前类用于服务层，Service层，是日常开发中用于实现功能的最核心代码，核心业务代码放在service层 |
| @Repository | 说明当前类用于数据持久层，Dao层 |
|  |  |  |
|  |  |  |
|  |  |  |

需要再spring IoC配置文件中开启注解扫描才可以使用注解

context：component-scan标签 base-package属性

component-scan会再IoC容器初始化时会自动扫描四种类型的组件类型注解

组件类型的注解默认的beanId是默认类名首字母小写，当然也可以进行手动设置

你知道 Spring 容器启动阶段会干什么吗？

容器启动阶段首先加载配置文件，然后利用对象的默认构造方法创建对象，然后注入对象的属性，然后对象执行init-method方法，容器启动完毕

能说一下 Spring Bean 生命周期吗？



将IoC容器中Bean的生命周期就是值在IoC容器，哪个阶段，Bean做了什么事情。

|  |  |  |
| --- | --- | --- |
| Bean生命周期 | | |
| 时间 | 容器行为 | 对象行为 |
| t1 | IoC容器准备初始化，解析XML配置文件 |  |
| t2 |  | 对象实例化(包括字面量和Bean对象)，执行构造方法 |
| t3 | 为对象注入属性 |  |
| t4 |  | 调用init初始化方法,有bean标签inti-method属性指定，在对象注入属性值后，基于这个属性值完成对象的初始化工作 |
| t5 | IoC容器初始化完毕 |  |
| t6 |  | 执行业务代码 |
| t7 | IoC容器准备销毁 |  |
| t8 |  | 调用destroy-method方法，释放资源 |
| t9 | IoC容器销毁完毕 |  |

Bean表进行属性设置的时候如何让程序自动计算属性之间的结果？

在init-method方法中计算，如果是单例模式，该方法只会执行1次

Bean 定义和依赖定义有哪些方式？

xml

注解

JavaConfig

有哪些依赖注入的方法？

依赖注入是指运行时将容器内的对象利用反射赋给其它对象的操作

依赖注入说白了就是将两个对象关联起来

1. 利用setter方法实现对象依赖注入

property标签

* 1. 是静态数值的输入 name value

“123”、“红色”、字符串、包装类

<property name="tittle" value="红富士"/>

首先创建”红富士”String对象里面的value引用指向常量区中的”红富士”，然后将这个对象用setTittle方法赋给tittle属性实现了字符串对象的依赖注入

1.2 对象注入

name ref

首先创建两个不想干的Bean到IoC容器中

然后通过调用set方法将其中一个Bean赋给另一个Bean

1. 利用构造方法实现对象依赖注入

contructor-arg标签

1、name value 静态

2、name ref 动态

Spring IoC如何注入集合对象？

bean标签的子标签property的子标签list可以设置对象

list标签分为value子标签或ref子标签分别对应静态值(字面值)和动态bean对象

和前面对象依赖注入意义 name：字面值，ref：动态bean对象

如何查看Spring IoC容器内有多少个对象？

查看Spring IoC有多少个对象、有哪些类型：

contex.getBeanDefinitionNames 获取IoC容器中所有的对象的名称

1、内部bean标签是不会出现在结果中的，虽然Spring IoC会创建该对象并赋给map的value

2、如果bean标签没有写id或者name，就会用类名名称#0 作为id com.imooc.spring.ioc.entity.Apple#0

3、com.imooc.spring.ioc.entity.Apple#1，如果有多个相同类型的匿名Bean，会在后面编号进行区别

4、如果有多个相同类型的匿名bean，用类名进行获取，会默认返回第一个匿名bean，要想获取后面的，必须在类名后面加#再加编号

通过cotext.getBean(beanName).getClass.getName得到类的名称，得到运行时类型，这是javase的内容

获取内容，执行打印对象即可，如果重写了toString就可以查看内容

Spring 有哪些自动装配的方式？

自动装配：指的是再IoC容器运行的过程中自动地为某个属性注入数据。自动装配注解就是为了依赖注入所存在的

|  |  |  |
| --- | --- | --- |
| 分类 | 注解 | 说明 |
| 按类型装配：不关心beanId是什么，把与属性相同类型的对象注入 | @Autowired | 按容器内对象动态注入属性 |
| @Inject |  |
| 按名称装配 | @Named | 按属性名自动装配属性 |
| @Resource |  |

注：使用xml配置文件中，property标签的name ref是按照bean的名称进行注入

@Autowired注解属性是，变量名必须与容器内的beanId匹配吗？

不需要，因为@Autowired是按照类型装配，即根据要注入的属性的类型，从容器中找到向匹配的类型进行装配，与变量名没有关系。所以不是必须的

@Autowire在IoC容器中找到多个对象与当前属性匹配怎么办，会装配容器内哪个对象注入到该属性中？

当@Autowire在IoC容器中找到多个对象与当前属性匹配时，会抛出一个异常，因为spring也不知道选哪个，与变量名称、beanId名称没有关系。无论怎么修改要注入属性的名称都没有用。此时的解决办法有两个：以实@Primary注解，在找到的多个类中选一个加上该注解，spring会优先注入该类的对象

1. 如果@Autowired 在IoC容器中只找到一个bean与当前要注入的属性类型相同，则与属性名称没有关系，属性名称就可以任意
2. 如果@Autowired 在IoC容器中找到多个bean与当前要注入的属性类型相同，则属性名称必须与对应类的首字母小写，如果对应类注解时没有指定beanId
3. 解决注入属性找到多个符合的对象，使用@Primary注解和@Qualifier注解

解决方式：

1 修改变量名称

2 注解：@Primary @Qualifier

Spring 中的 Bean 的作用域有哪些?

bean scope属性用于决定对象何时被创建，以及作用范围是什么

bean scope决定容器内的bean数量

默认值为 singleton 即单例模式，全局共享一个对象实例

bean scope 属性

|  |  |
| --- | --- |
| bean标签的scope属性 | 说明 |
| singleton(默认设置) | 单例模式，只有一个对象，全局共享 |
| prototype | 多例模式，每次要使用对象时，IoC容器会自动创建一个全新的对象 |

为什么Spring IoC容器会默认使用单例模式？

如果每一次在需要某个对象的时候都去创建一个新的bean，会额外创建内存空间，同时创建对象也会占用cpu资源，由于频繁创建对象所带来的资源损耗必须要考虑在内，而IoC使用单例模式可以有效解决创建对象时占用资源的问题，因为全局只用创建一次就行了。

Spring 中的单例 Bean 会存在线程安全问题吗？

会

要看这个Bean是不是无状态的，如果是无状态的，即没对自身的成员变量以及外界变量、对象、数组进行修改，则是线程安全的，如果依赖于外界的状态，则不是线程安全的。

singleton的线程安全问题：a.setnum(1)🡪a对象num属性为1🡪程序调用🡪sout(num)🡪1

但在这个过程中其它线程修改num的值就会导致线程1看起来结果不正确

这个过程可以通过线程同步方式，用synchronized给同步代码加锁

还有一个解决方法就是给每一个线程创建一个对象，在使用getBean获取对象时，每个线程都会创建一个新的Bean从而解决线程安全问题，prototype在容器中会创建多个实例，每个线程使用到某个Bean就会创建一个新对对象。每次使用IoC容器里面的对象，就会创建一个新对实例，不会产生线程安全问题

singleton和prototype对比

|  |  |  |
| --- | --- | --- |
|  | singleton | prototype |
| 对象数量 | 全局唯一 | 存在多个 |
| 对象实例化 | IoC容器启动时 | 要使用对象时：getBean或者依赖注入时都会产生一个新对象 |
| 线程安全 | 存在 | 不存在 |
| 执行效率 | 高 | 低 |

装配注解@AutoWired在成员变量上和在方法上有什么不同？

如果装配注解@AutoWired放在方法上，则自动按类型或者名称对方法参数进行注入，并自动调用该方法。

如果装配注解@AutoWired放在字段上，SpringIoC容器会通过反射技术将字段的访问修饰符修改成public，然后直接进行赋值，最后在复原，不执行setter方法。

为什么controller service dao层的类会设成单例？

因为这写层的类没有修改自身字段的函数，setter方法，所以这些类对象本身时线程安全的，至于这些对象调用的方法里面用到的对象、内存空间是不是线程安全的与单例还是多例已经无关。如果方法会修改某一块公共的内存地址，则该内存不是线程安全的，与调用该方法的类是不是单例已经没有关系了。所以这几个层的类基本上都是单例模式，根源在于这些类本身的字段不会修改。

service层，dao层，controller为什么采用单例模式，单例对象是线程安全的吗，开发所这些层的类的字段不会提供setter方法进行修改，所以是线程安全的。而且，使用@Autowired注解，Spring IoC容器不会通过setter方法进行属性的注入，而是将要注入的变量的访问修饰符修改为public然后赋值，单例模式在IoC启动的时候只执行一次，后面多线程访问时访问修饰符又变回public，所以是线程安全

@AutoWired可以注解在方法上，会对形参进行注入并执行方法

如何理解无状态？

无状态：不依赖于自身或者外界状态

1. 函数无状态：指函数执行时不会对外界变量、对象、数组进行修改，输入与输出是一对一映射的，每一个线程调用该函数都只会操作线程栈内部的局部变量，而这些变量是线程安全的。所以无状态函数是线程安全的
2. Bean无状态：没有字段或不提供修改字段的方法，bean里面的成员方法都是无状态的，则bea时无状态的，多个线程调用都不会修改Bean，这个对象就是线程安全的
3. Spring 单例模式的Bean如果提供了setter方法，或者bean的有无状态的成员方法，就不是线程安全的，因为在多线程调用时会修改Bean的状态

聊一聊基于Java Config配置IoC容器？

Spring IoC容器的配置方式有三种：基于XML、基于注解、基于Java Config

基于Java Config是在spring 3.0之后提出的一种全新的配置方式，它的主要原理是使用Java代码来替代传统的XML文件：

1. 基于Java Config可以完全摆脱XML的束缚，使用独立Java类管理对象与依赖
2. 注解配置相对分散，利用Java Config可对配置集中管理
3. 可以在编译阶段进行依赖检查，不容易出错

Java Config核心注解

|  |  |
| --- | --- |
| 注解 | 说明 |
| @Configuration | 放在配置类，说明当前类是Java Config配置类，作用：完全替代XML文件 |
| @Bean | 用在方法上，方法返回的对象有Spring IoC容器管理，返回的对象的beanId默认为方法名 |
| @ImportResource | 放在类上，加载静态文件，可使用@Value进行获取 |
| @ComponentScan | 放在类上，与context:componentscan是一样的，用于扫描项目中四种组件类型：@Repository，@Service，@Component，@Controller  mapper有mapperScan |

@Bean注入方法参数跟@Autorwired一样，@Bean也可以和@Primary配合使用，也可以加上Scope等等以前用XML标签完成的事情

@Autowired 的实现原理？

1、@Autowired时按类型进行装配

2、@Autowired放在字段上

3、@Autowired放在方法上

Spring IoC有哪几类注解？

1. 组件类注解
2. 自动装配类注解：按类型、按名称
3. 元数据注解

|  |  |
| --- | --- |
| 元数据注解 | 说明 |
| @Primary | 按类型装配是，如果有多个bean匹配，则有@Primary类的bean会优先注入 |
| @PostConstruct | 相当于xml配置中bean标签的init-method属性 |
| @PreDestroy | 相当于xml配置中bean标签的destroy-method属性 |
| @Scope | 设置对象的scope属性  singleton，prototype |
| @Value | 为属性注入静态数据 |

@Value注解实现原理？

与@Autowired一样，@Value注解会将字段的访问修饰符通过反射机制修改为public，然后直接赋值，并不会调用setter方法，也没有setter方法提供调用。需要在Spring IoC的xml配置文件中引入properties文件，通过property-placeholder引入，然后@Value中写入el表达式形式的字符串，${}里面填上设置的key，就可以得到value

使用Java Config与使用XML配置对比？

|  |  |  |
| --- | --- | --- |
|  | XML | Java Config |
| 配置文件 | xml文件 | Java类 |
| 加载配置文件 | ClassPathXmlApplicationContext | AnnotationConfig  ApplicationContext |
| Junit4 | @ContextConfiguration  value默认属性 | @ContextConfiguration  classes属性 |
|  | bean标签 | bean注解 |
|  | property | set方法 |
|  | ref | 形参 |
|  | bean scope | @Scope |
|  | 扫描组件类注解 | @ComponentScan |
| 加载properties或properies文件 | context:property-placeholder |  |

Java Config可以完全替代XML！

spring IoC阶段注解总结：



说说什么是 AOP？

面对切面编程：在软件运行过程中，在执行前，或者执行后，都可以去额外地增加相应的扩展功能，而这个扩展功能称之为切面

切面有点类似浏览器中的插件

Spring中的可插拔组件技术

根据面对对象的思想：万物皆对象，一个功能也是对象，抽象为功能类，称之为切面类。 切面放在执行前、执行后、或者同时都有（surround）



什么是面向过程编程和面向对象编程？

面向过程编程是一种以过程为中心的编程思想，该思想首先会分析出解决问题所需步骤，然后用函数一步一步把这些步骤实现。面向过程的编程思想是一种基础的顺序思维的方式，它将程序分为数据和操纵数据两部分，其核心问题是数据结构和算法的开发和优化。

面向对象编程思想是按照人认识客观世界的方式，采用基于对象的概念建立模型，模拟客观世界分析设计实现。面对对象编程思想把所有的事务都当成对象，处理问题过程中所属思考的不再是用怎样的数据结构描述问题，而是考虑重现问题中各个对象之间的关系。

说说你平时有用到 AOP 吗？

有

aop 切点的execution?



访问修饰符如果时public，是可以省略的

形参列表那里是写类型名

(..,String)

(..,Str\*)

(Fl\*,In\*,..,String)

(int,float,..)

AOP 五种通知类型？

@Before

@AfterReturning ：执行成功，返回对象后通知

@AfterThrowing：运行出错后通知，与@AfterReturning 互斥

@After： 类似try catch finally中的finally，不管执行成功还是失败，一定会通知的

@Around：自定义通知时间，甚至可以决定目标方法执不执行

说说 JDK 动态代理和 CGLIB 代理 ？

JDK动态代理:

代理类和委托类需要实现同一个接口， 通过反射机制，代理类可以在运行是动态生成，不用像动态代理那样手写代理类。

JDK动态代理需要实现InvocationHandler接口，这是JDK提供的反射类，用于在JDK动态代理中实现功能拓展增强，类似于环绕通知

实现 invoke方法：对目标方法进行增强的部分

方法返回值：Object，目标方法运行后的返回值

方法参数：Object proxy，代理类对象，JDK动态代理自动生成

Method method：目标方法的方法类对象

Object args：目标方法传入的实际参数

throwable： 目标方法抛出的异常

调用method.invoke(target,实参列表)，等价于proceedingJointPoint.proceed()

Proxy是JDK反射包下，newProxyInstance用于动他创建代理类对象

需要传入委托类的类加载器、委托类的接口对象，invocationHandler

JDK反射机制实现动态代理，强制要求委托类实现接口

CGLib实现对类的增强：不要求目标类实现接口

CGLib Code Generated Library 运行时代字节码增强技术

CGLib 会在运行时，生成目标基层继承类字节码的方式进行拓展

本质原理是运行时多态

如果目标类 实现了接口，则Spring底层优先使用JDK动态代理技术来实现动态代理

如果目标类没有实现接口，则Spring会利用CGLib技术，继承的方式实现目标类的拓展

AOP 实现原理？

核心是基于设计模式中的代理模式实现功能的拓展

如果目标类实现了接口：通过JDK动态代理实现功能拓展

如果目标类没有实现接口：通过CGLib第三方组件实现功能拓展

什么是代理模式：指通过代理对象实现对原对象的功能拓展

静态代理：实现Runnable接口的类，用Thread作为代理类，调用start方法

动态代理：

客户类，代理类，委托类

代理类和委托类实现同一个接口

客户通过代理类完成需要的功能





静态和动态的区别在于：代理类是在哪个阶段生成的，如果是编译阶段，则是静态代理，如果是运行阶段则是动态代理。静态代理需要手动地创建代理类，开发起来虽然很简单，但是很麻烦

说说 Spring AOP 和 AspectJ AOP 区别?

AspectJ AOP是Eclise实现的，新语言

Spring AOP 使用AspectJ中的aspectjweaver实现切点的织入

Spring AOP 利用Java语言，代理模式实现

AspectJ 是Eclipse提供的，是一种面向切面编程的语言，AspectJ有一套完整的体系，可以在运行时实现AOP的理念。Spring AOP依赖于AspectJ，但不完全使用AspectJ实现AOP

Spring AOP利用AspectJ AOP实现AspectJWeaver实现类于方法的匹配

在引入Spring-contex之后会自动引入spring aop，但是还无法满足全部要求，还必须引入aspectjweaver依赖，其核心作用就是，圈定切点的作用范围。

作为Spring AOP，并不是完全依赖AspectJ AOP，在运行时对目标方法进行功能拓展时是由Spring AOP来实现的，其核心原理时利用代理模式。

AOP中的概念

|  |  |
| --- | --- |
| Aspect | 具体的可插拔组件功能类，通常一个切面只实现与一个通用方法，一个切面类就是一个标准的Java类 |
| Targe Class/Method | 真正要执行的业务代码，目标方法、类、对象可以通过JointPoint方法，利用反射机制，动态获取。 |
| PointCut | 切点，使用execution表达式说明切面要作用域系统的哪些方法上，说明了切面要作用的范围 |
| JointPoint | 连接点，切面运行过程中包含了目标方法、目标类元数据的对象 |

切点决定了 地点

通知决定了 时间

方法决定了 事情

AOP配置过程

1. 引入依赖 spring-context, aspectjweaver
2. 实现切面类，方法
3. 配置Aspect Bean
4. 定义PointCut
5. 配置Advice

JointPoint连接点有哪些方法？

getTarget: 获取IoC容器中目标类对象

getSignature: 获取目标方法

getArgs: 获取目标方法的参数

使用spring IoC、spring AOP、spring事务的最小依赖？

spring IoC： spring-context

spring AOP： spring-context aspectjweave

spring事务：

为什么Spring Ioc可以降低对象之间的耦合？

首先要理解耦合的概念：在软件工程中，耦合是指一个模块与另一个模块的关联程度，这个模块可以是类或方法、对象。如果一个模块的改变会影响到另一个模块，我们就说这两个模块关联程度较低，也就是高耦合的。反之，如果一个模块的改变很少影响另一个模块，则说明这两个模块关联程度较低，也就是低耦合的。

两个对象之间什么时候会产生耦合呢，就是当两个对象之间是依赖关系是。依赖关系是指一个对象中需要使用另一个对象，具体由三种不同的情况：另一个对象作为局部变量，另一个对象作为方法参数，向另一个对象发送消息。当两个对象由依赖关系时，这两个对象是耦合的，spring ioc通过将容器来管理这种依赖关系，降低耦合读。举个例子，加上对象A需要使用对象B的功能，在没有IoC的时候，需要在A里面创建B的实例，如果B的实现方法发生变化，A也会跟着改变，此时A和B是高耦合的。而如果用IoC实现，只需在A中声明要使用B实例，至于B的实例是如何创建的，A不需要关系，只需要关注自己的业务代码，后面如果B的实现方式改变，也不会影响A的代码，这极大降低了两个对象之间的耦合度。

依赖关系和关联关系的区别？

A依赖B

依赖关系：B的实例对象作为方法参数出现在A中、B的实例对象作为局部变量出现在A中，A中向B发送了消息 、B的实例对象作为返回参数出现在A中

关联关系比依赖关系更强，A关联B，B的实例对象作为A类的成员变量出现在A中

依赖关系是一种临时关系，只有在方法调用的时候会产生，而关联关系是一种长期的关系，主要体现在成员变量，只要对象一声明，这种关系就存在。

关联关系本质上也是依赖，只是强度更强。

什么是Java Bean？

Bean就是Java中的类，是符合一定规范的类。

规范1：字段是私有的

规范2：通过setter getter读写字段

规范3：必须由默认的构造函数

如何使用spring整合的测试框架？

传统程序员管理对象方式有哪些不足？

1. 类属性发生改变时需要修改源代码
2. 如果要增加或减少对象的数量也需要修改源码
3. 对象之间时硬关联，在编译阶段就将对象之间的关系确定了，这是一件非常死板的事情。

修改源码需要走繁琐的审批流程、重新编译上线，这是非常麻烦的。

需要在修改源代码的情况下才能对程序进行调整

所以对象的管理交由第三方管理可以避免上述问题。通过引入IoC容器的方式、对系统中的所有对象进行管理，并且根据一定规则来约定对象是如何创建的以及如何关联的。

Spring服务最小的依赖范围？

spring-context，只有引入了它，才能使用spring最核心的IoC容器，才能使用由这个特性引发出来的其他功能。

引入spring-contex会引入

spirng-aop：面向切面编程模块

spring-beans：对对象继续管理的模块，对象实例化或者设置管理

spring-context：通过ApplicatContex对象可以通过代码对spring IoC容器创建。

spring-core: 所有spring最核心的代码,

spring-expression：spring内置的表达式模块，spring有自己的表达式语言，通过它进行解析和处理。

spring-jcl：这是spring与日志进行交互的模块，在运行过程中产生的底层日志由它处理。

这是引入spring最小的服务范围，这些模块共同完成了Spring IoC的工作。

junit4和junit5的区别？

引入时包不同

注解不同

spring如何使用junit4进行单元测试？

引入spring-test依赖

引入junit4依赖

创建测试类

@Runwith注解 这是junit4的注解

@ContextConfiguration 这是spring的注解，org.springframework.test.context

如何用spring来管理对象的创建？

1. 创建一个maven工程
2. 引入spring-context
3. 创建Java Bean
4. 在resource目录下创建applicationContext文件
5. 写入xml头部内容，去官网复制
6. 通过bean标签声明要创建一个对象

id:对象的名称

class：创建哪个类的对象

bean标签里面由property标签，通过property设置对象的属性

原来创建对象，需要将这些信息硬编码程序源代码中，现在只需要使用配置文件，以后对象属性改变、对象数量改变、对象间关系改变时，只需要修改配置文件，而无需修改源代码。配置文件的修改不用重新编译，审批。极大地缩减了流程。

spring如何启动IoC容器？

在项目入口，即main方法里面需要新建ApplicationContext接口的实现类对象，这个对象就是一个IoC容器。

classpath是什么路径？

是项目编译后，classes路径下面，包顶层所在路径，resource目录下的文件，idea会自动复制一份过去。

springIoC容器的applicationContext文件中，如何关联两个对象？

在bean标签里面的property标签使用ref关联相应的bean id

spring IoC容器在启动过程中根据applicationContex，进行动态实列化以及动态绑定，这是利用java反射技术实现的

为什么要用配置文件来进行对象创建以及关系绑定呢？

这是为了程序上线后可维护性考虑的，采用硬编码的方式，如果对象需要改变源代码，但利用配置文件，不需要调整任何源代码，不需要重新编译即可完成对象修改。

利用IoC容器可以使得对象与对象解耦。

hashset是如何比较对象是否重复的？

首先比较hash值、hash值通过对象的hashcode函数计算，如果hashcode函数不同则不会再去比较equals方法，如果equals相同，不会会认为是重复的key。所以，为了保证hashset或hashmap在使用自定义类作为key时的正确性，要确保hashcode不同时，equlas的结果也是不同的。如果hashcode结果不同，而equals相同，JVM会认为这两个对象不是重复的。所以，在重写是，通常需要对这两个方法都重写：hashcode和equals

utf8和utf8mb4的区别？

utf8mb4是utf8的超集，能够表示更多特殊的符号，如果只表示unicode编码格式下的字符，utf8就够了

什么是事务？

事务是一组对数据库的操作，事务是原子性的，即这一组对数据库的操作要么都成功，要么都失败。事务是以一种可靠的，一致性的方式访问和操作数据库的程序单元。

事务依赖于底层数据库的实现，mysql通过事务区作为数据缓冲地带，当所有操作完成后提交，就可以将事务区中的数据一次性写入实际的表中，如果出现了问题就进行回滚。

Spring 事务的种类？

spring通过Jdbc中的TransactionManager事务管理器实现事务控制

TransactionManager提供commit和rollback实现事务的提交与回滚。

1. 编程式事务（手动式事务）

编程式事务指通过代码手动编程实现提交回滚事务的事务控制方法

1. 声明式事务（注解型事务,自动式事务）

注解的本质是声明。

声明式事务的底层原理就是AOP，它是AOP理念的一个典型应用

1. 当目标方法执行成功时机：自动提交事务
2. 当目标方法遇到运行时异常时：自动回滚事务

声明式事务是利用AOP中的环绕通知解决，完成应用程序拓展，在不修改源代码的前提下实现事务配置

配置TransactionManager事务管理器

配置事务通知与事务属性

为事务绑定PointCut切点，限定事务通知的范围

Spring 的事务传播机制？

事务传播行为：是指多个拥有事务的方法，在嵌套调用时，事务控制方式。

propagation来配置





声明式事务实现原理了解吗？

声明式事务其根本是一个Spring AOP的环绕通知

什么是包装类？

包装类是对Java中八大基本数据类型的一种拓展

装箱：基本数据类型🡪包装类

拆箱：包装类🡪基本数据类型

自动装箱底层调用的是valueOf静态方法

自动拆箱底层调用的是intValue()实例方法

Jdk1.5之前需要手动拆箱与装箱

Jdk1.5及以后就可以自动拆箱与装箱了

Integer integer = 1; 🡺 Integer integer = Integer.valueOf(1);

int i = integer; 🡺 int i = integer.intValue();

包装类和String类都是用常量来存储真正的值

所以包装类和String类的对象都是不可变的

private final int value // 非引用变量，存储值

private final char[] value // 引用变量，存储地址

Object obj1 = true ? Integer.valueOf(1):Double.valueOf(2.0)，打印obj会输出什么？

1.0

首先三元表达式，为true选第一个

其次Integer和Double在一个表达式中，Integer的值会自动提升为Double类型

所以结果不是1，而是1.0

关键：一真大师，三元运算夫看成一个整体，与if else语句不同，后者是分别计算，不会提升优先级。

自动调用对象toString方法的场景？

与字符串运算

打印

String.valueOf()

包装类和String类相互转化有哪些方式？

包装类🡪String类

与字符串做运算 （自动转）

toString方法

打印包装类对象 （自动转）

String.valueOf

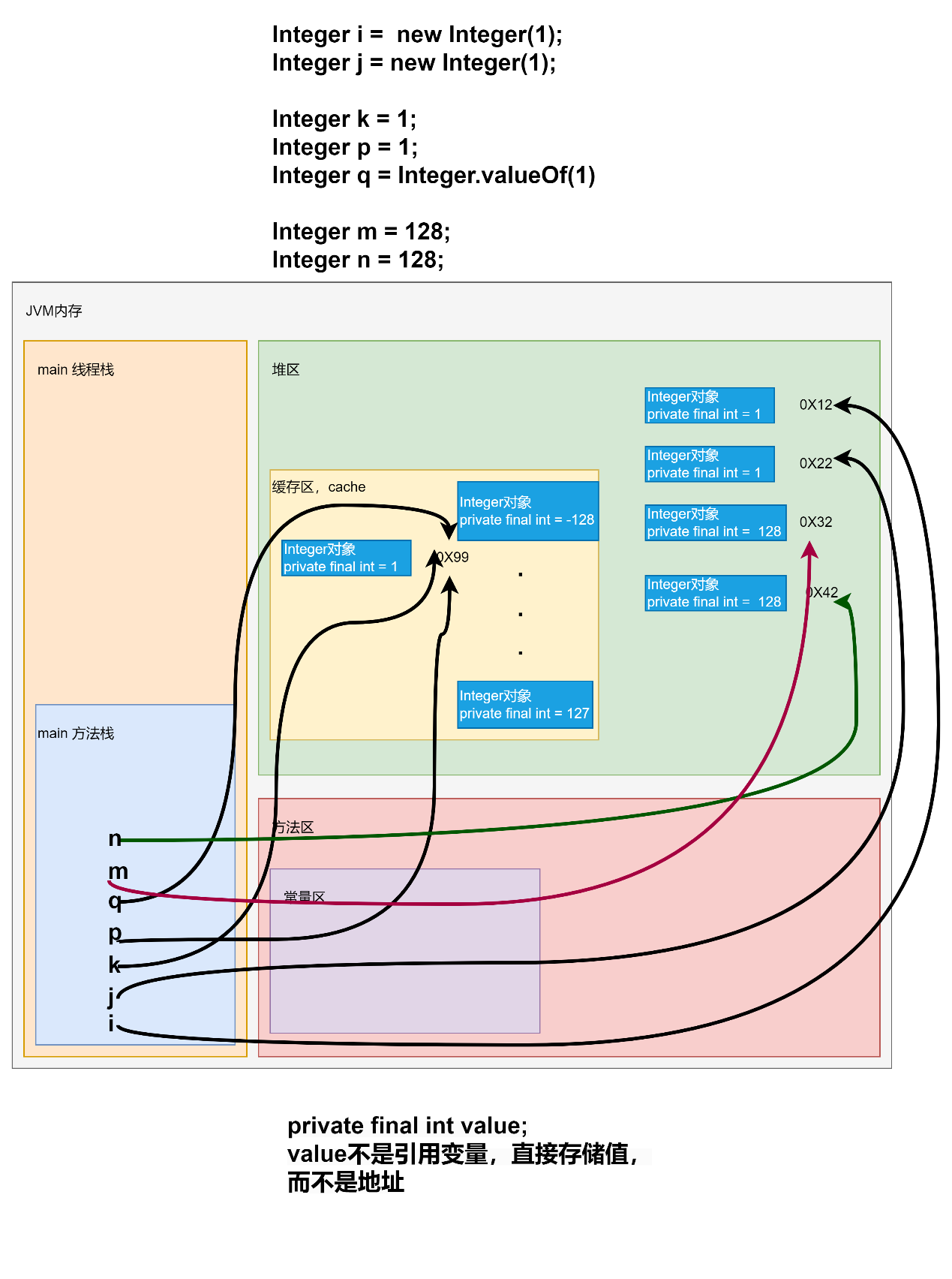
这几种方式本质上都在调用对象的toString方法

String类🡪包装类

构造方法

parse方法

new Integer(112)和Integer.valueOf(112)的区别？



new Integer(112)会直接在堆中创建一个Integer对象，该对象有一个私有final修饰的int变量，其值会被赋为112，Ineger.valueOf(112)会先判断112是不是在缓存池范围中，缓存池的范围为-128到127，112在里面，会直接返回缓存池中的一个创建好的Integer对象，而不会再创建一个新的Integer对象。

介绍一下自动装箱和自动拆箱？

装箱：基本数据类型🡪包装类

拆箱：包装类🡪基本数据类型

自动装箱底层调用的是valueOf静态方法

自动拆箱底层调用的是intValue()实例方法

Jdk1.5之前需要手动拆箱与装箱

Jdk1.5及以后就可以自动拆箱与装箱了

Integer integer = 1; 🡺 Integer integer = Integer.valueOf(1);

int i = integer; 🡺 int i = integer.intValue();

当包装类和非包装类运算、比较时，会自动拆箱

Integer创建对象的几种方式和区别？

new

Integer.valueOf()

自动装箱

int与Integer的区别？

int是基本数据类型，Integer是包装类类型，是int类型的拓展

Integer类型量必须实例化后才能使用，int类型变量不需要

Intger类型变量默认值为null，int类型默认值为0

Integer类型的变量是引用变量，存储的是对象的地址值，执行堆中的一个Integer对象，int类型变量是非引用变量，存储的是实际数值

1. int是java中的基本数据类型，Integer是int的包装类，是对int类型的拓展
2. Integer类型变量必须实例化之后才能使用，int类型变量不需要
3. Integer类型变量默认值为null，int类型变量默认值为0
4. Integer类型变量是一个引用变量，存储的是地址值，指向堆中一个Integer对象，而int变量是非引用变量，存储的是实际的数值

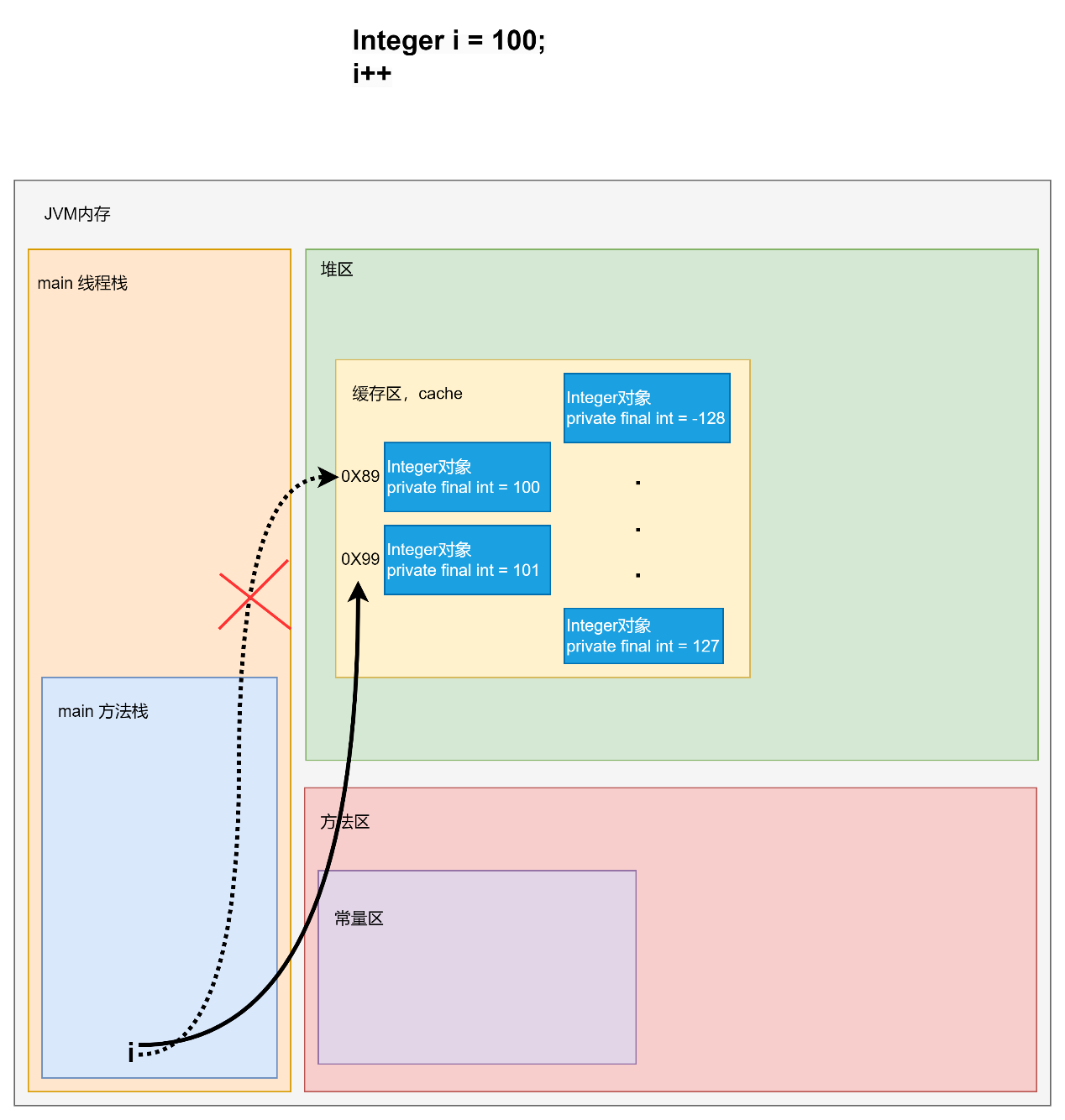
Integer.valueOf() 与new Integer()的结果一定不是同一个对象吗？

是的，首先new Intege()的结果一定不在缓存池中。然后两new Integer()的对象一定不是同一个对象。

而Integer.valueOf(xxx)，当xxx属于[-128,127]时，Integer.valueOf(xxx)的对象再缓存池中，满足第一种情况，不是同一个对象。当xxx不在这个范围时，会new Integer(xxx)，满足第二种情况，不是同一个对象。

综上所述，Integer.valueOf() 与new Integer()的结果一定不是同一个对象

Integer i = 100; i++, 画出内存执行图?



i是Integer类型

i++ 🡺 i = i + 1 🡺 i = 拆箱 + 1 🡺 i = 101 🡺 装箱

如果i是int类型

i = 100;

i++ 🡺 i = i + 1 🡺

1. i是int但i是局部变量，i++是线程安全的。
2. i是成员变量，i在堆中，i++不是线程安全的。

i++不是原子操作 + i不是局部变量🡺线程不安全

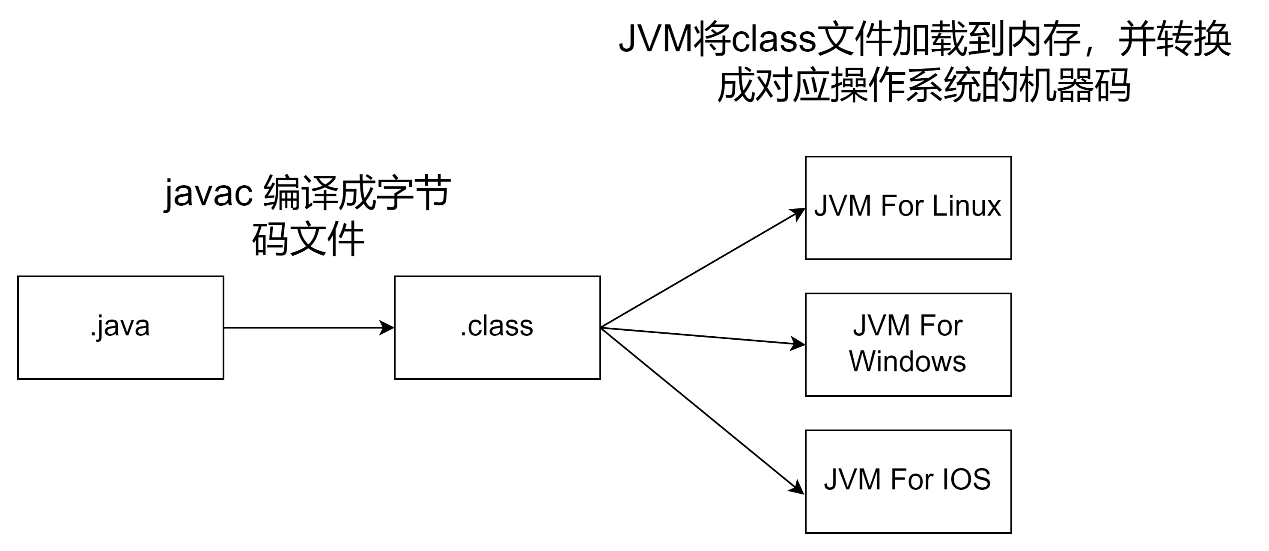
为什么说Java是平台无关性语言？

Java中有一句话，Compile once，Run anywhere，这是如何实现的呢：

java分为编译时和运行时两个阶段

编译时：javac 指令，全称时java compile，即将源码.java文件编译成字节码，并存入对应的.class文件中。class文件就是java文件翻译成的二进制字节码文件，java文件中类的属性、方法、常量信息等会被存储在.class文件中，除了这些，还会添加一个共有的静态常量属性，Class类的示例，这个属性记录了类的相关信息以及类型信息。真正加载class文件用到虚指令，通过javap -c xxx xxx是class文件名，会出现一些指令。

JVM将字节码加载进内存，并将字节码转换为本操作系统可识别的机器码去执行。将对应的class文件拷贝到其他装有相同版本的jdk或这jre的操作系统，也可以执行。

java源码首先被编译成.class文件，再由不同平台的JVM解析。Java在不同平台运行时不需要重新编译。JVM在执行字节码文件时，会将字节码转换成对应平台的机器码

Java源文件首先被编译成class文件，再由不同平台的JVM解析，JVM解析时将字节码文件加载进内存，并将其转换为对应操作系统的机器码，所以java只需要编译一次，就可以在不同平台上运行，这也就是为什么java是平台无关性语言。

运行时：JVM将.class文件加载进内存，将其转换为对应操作系统的机器码。

JVM这一块主要有哪些点要掌握？

1、类加载

2、运行时内存模型

3、垃圾回收

我们写的Java代码到底是如何运行起来的？

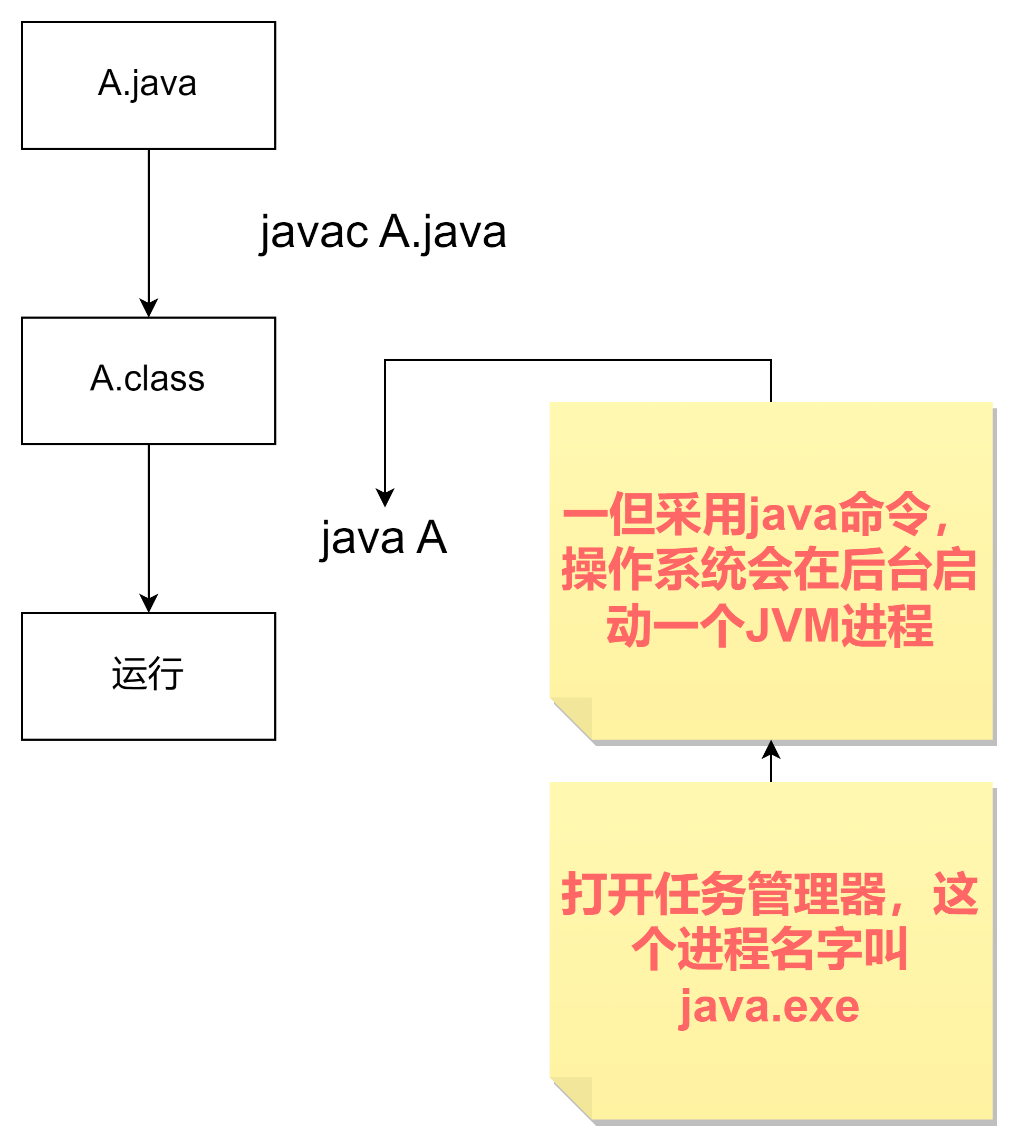
jar包和war包有什么区别？

jar全称是java archive，它是基于zip文件格式的归档文件，用于将多个class文件整合成一个文件

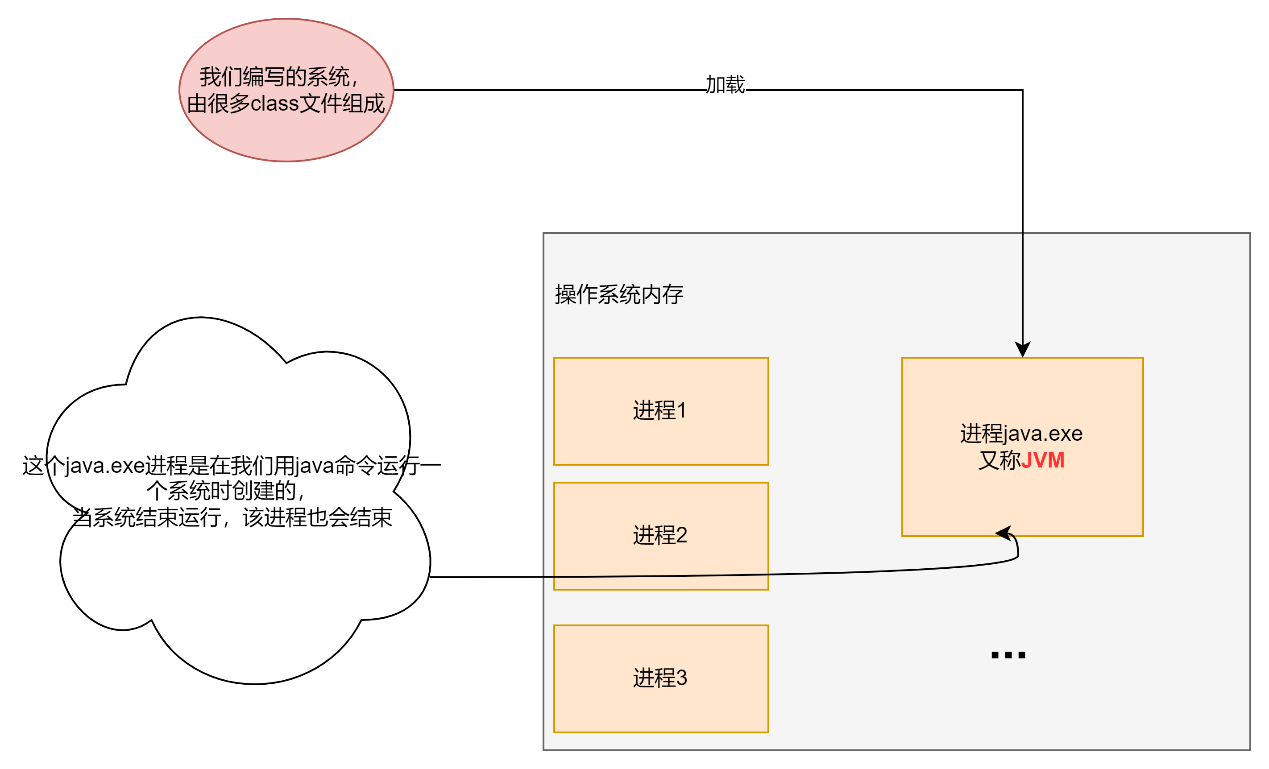
war的全称是web application archive，是用于分发和部署web应用程序的jar文件格式，把war包放在tomcat的webapp目录下，tomcat服务器在启动时会解压war包，将class文件加载进内存。

jar包中通常只包含class文件和一些部署文件，但war包里面就很全了，包括class文件、配置文件、静态文件…

对于编译好的这些“.class”字节码，是怎么让他们运行起来的呢？



每运行一个java程序(可以是一个文件、一个项目…)，操作系统就会启动一个java.exe的进程，这个进程就是JVM，负责加载对应的class文件，JVM会根据本机操作系统的类型，将class文件翻译为对应的机器码。所以启动一个java程序或者java项目，本质上是启动了一个JVM，再本质上是启动了一个名为java.exe的进程，这个进程负责运行这台机器上运行的这个系统。



所以一个JVM==一个系统进程

我们用java编写的应用程序启动并不是该程序进程启动了，而是启动了一个JVM进程，然后改JVM进程将应用程序加载进内存，由该JVM负责这个应用程序的运行，内存分布、垃圾回收等等各种

JVM运行机制是怎么样的？

1、.java文件编译为.class文件

2、启动一个java.exe进程，或者JVM进程

3、将class文件加载到JVM内存中，这个加载是懒加载

4、执行类中的代码，需要用到哪个类再加载哪个类

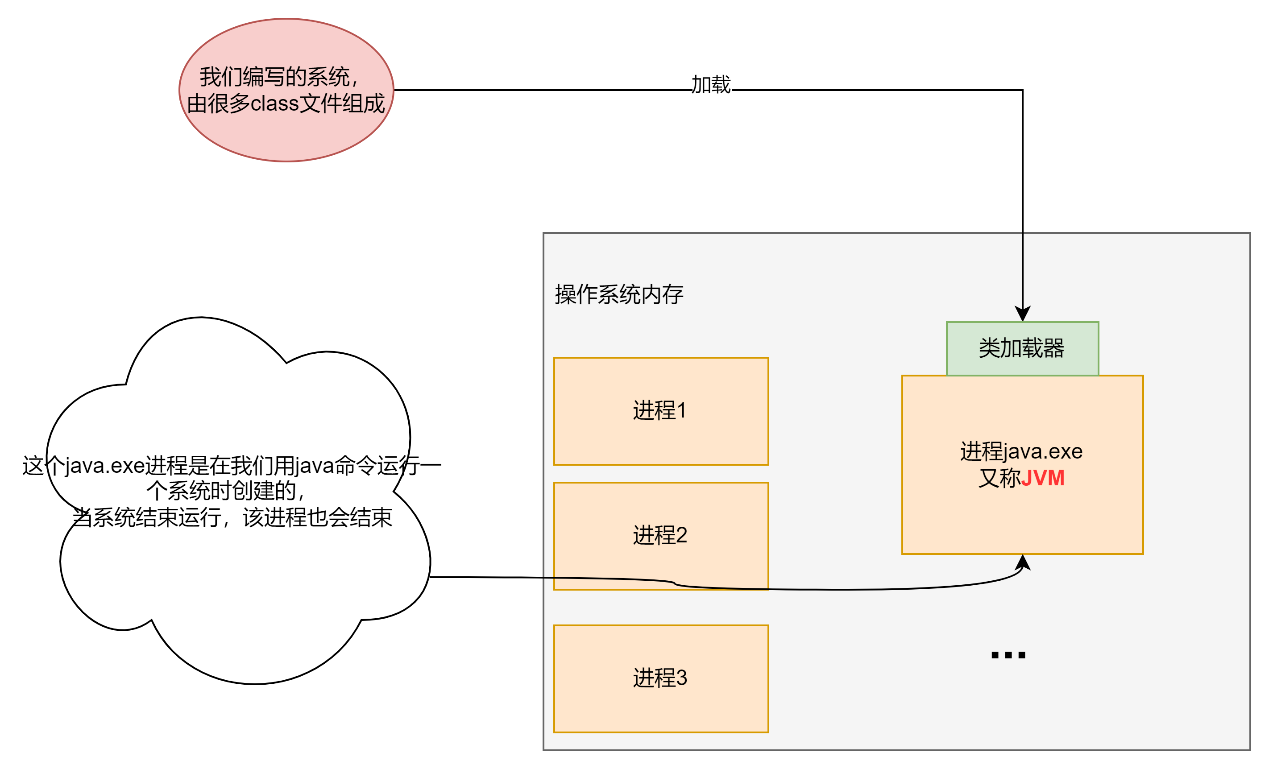
为什么会有类加载器这个东西？

class文件🡪内存

一个java文件中，有多少个外部类，就有多少个class文件。所以我们可以这样说，一个class文件就代表一个类。

1. 一个class文件代表一个类
2. 把class文件加载进内存==把类加载进内存

而把类加载进内存有一个专门的模块，叫做类加载器



所以什么叫做类加载：把class文件加载进内存就叫做类加载

什么叫类加载器：负责类加载的模块叫做类加载器

这个加载的过程时懒加载，不会一股脑的把我们编写好的系统中的所有class文件全部加载，而是需要哪个就加载哪个。

JVM是个什么东西？

JVM本质上是操作系统中一个名为java.exe的进程，会在操作系统中使用java命令运行一个class文件或者jar包时启动。

JVM跟我们平时运行在机器上的系统之间是什么关系？

我们编写的，平时运行在机器上的系统本质上不是这个系统本身启动了一个进程，而是启动了一个JVM，由JVM管理该系统的运行。

类加载器的概念?

类加载器：JVM负责类加载的模块

类加载：将class文件加载进内存

class文件本质就是一个类，这就是类加载名字的来由。

针对加载进内存的类进行代码的执行？

类加载器采用的是懒加载模式，即需要用到哪个类才会去加载对应的类，执行到哪一步，遇到要使用某个类，才回去加载。

如何保证class文件的安全，不被反编译回源文件？

1、加密：在编译阶段可以通过一些工具给字节码文件加密

2、解密：在类加载的时候采用自定义的类加载器来解密字节码文件

这样就可以保证源码不被窃取。

JVM在什么情况下会加载一个类？

需要用到一个类的时候:

1. 包含main方法的主类，一定会在JVM进程启动后加载到内存，然后从main方法里面开始执行代码。

其他情况：

1. 调用某个类的静态属性、静态方法
2. 构造某个类的对象
3. 声明某个类的变量

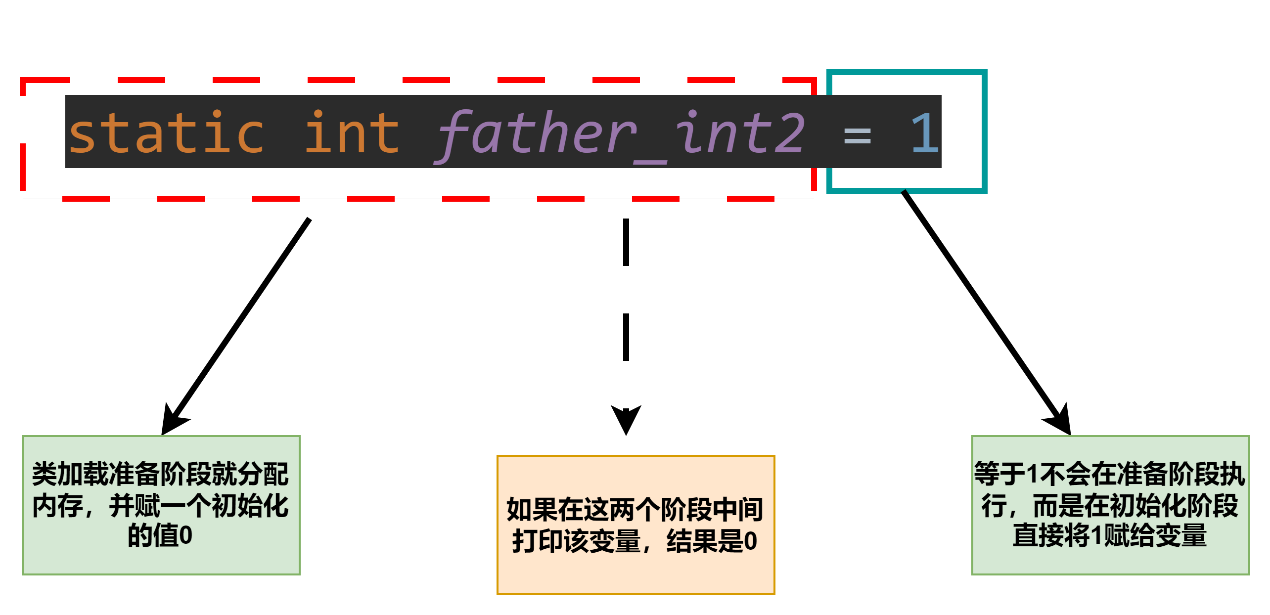
为什么class文件经过编译阶段已经检查过了符合java语法规范，在类加载时还需要验证阶段？

因为编译后的class文件有被篡改的风险，从而不符合JVM规范，如果没经过验证直接给JVM运行可能会直接出错。

初始化语句在类加载准备阶段会分配内存空间吗？

根据调试的结果，初始化语句会在类加载准备阶段分配内存，在类加载初始化阶段，执行初始化语句直接初始化，无需再分配内存。

类加载准备阶段：所有的类变量，不管有没有初始化语句，只要声明了，就根据类型分配内存空间，然后进入类加载解析阶段，将字符引用改为直接引用，然后进入类加载初始化阶段，执行初始化语句。



实例化过程中实例变量也是遵循一样的规则，分为准备阶段和初始化阶段

父类加载准备->父类加载初始化->子类加载准备->子类加载初始化->父类实例化准备->父类实例化初始化->父构->子类实例化准备->子类实例化初始化->子构

java中有类加载器？

类加载：将class加载到内存中

类加载器：JVM中负责类加载的模块，JVM本身就是一个java程序。类加载器就是负责类加载的类

java中负责类加载的类：

|  |  |  |
| --- | --- | --- |
| 类型 | 英文名称 | 说明 |
| 启动类加载器 | BootStrap ClassLoader | 负责将Java按照目录下的核心类加载到内存中。一旦JVM启动，这些类就会自动加载进内存，保证运行我们编写的系统所需的依赖 |
| 扩展类加载器 | Extension ClassLoader | lib\ext,这里面一些类也是用来支持系统运行，JVM启动时也会自动加载 |
| 应用程序类加载器 | Application Classloader | 这个类加载器负责去加载ClassPath环境变量所指定的路径中的类。可以通过-classpath或-cp指定路径 |

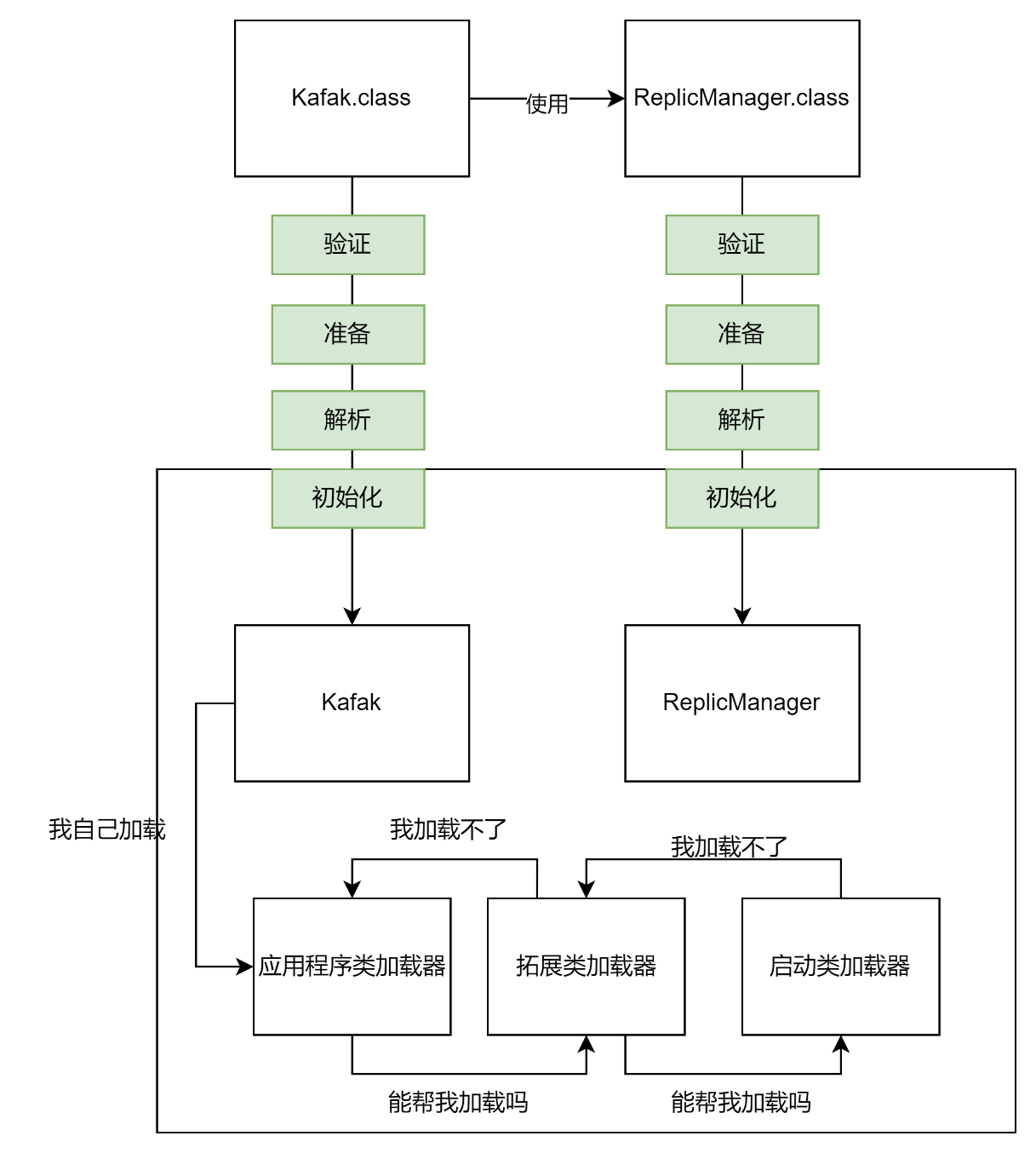
什么是双亲委派机制？

1. JVM的类加载器有亲子结构
2. 双亲委派机制基于这个亲子结构

一句话说明：先找父亲加载，不行再有儿子加载。

应用程序类加载器要想加载一个类会先去让扩展类加载

扩展加载之前会先去找启动类加载器加载，如果启动类加载器没有找该类，则会让扩展类加载器加载，如果扩展类加载器没找到该类，则会下放给应用程序类加载器加载。



Tomcat的类加载机制应该怎么设计，才能把我们动态部署进去的war包中的类，加载到Tomcat自身运行的JVM中，然后去执行哪些我们写好的代码？

Tomcat本身就是Java写的，一个运行的tomcat本质上就是一个JVM进程

Tomcat可以自定义一个类加载器专门负责webapp目录下类加载，这样通过双亲委派机制，父类加载器在各自作用范围均找不到类时就会下放给自定义类加载器进行加载。

JVM中有哪些内存区域，分别都是用来干嘛的？

什么是JVM内存区域划分？

使用java命令运行一个java程序或者项目时，系统会启动一个名为java.exe的进程，这个进程也叫JVM进程。JVM会分配一块独立内存区域，别的进程无法访问。对于这个块内存区域的划分、使用由JVM负责

1. 需要使用到某个类时，会把该class字节码文件加载进内存成为一个类，这个类需要有地方存储
2. 调用类的方法时，方法有一个方法栈存储局部变量
3. 实例化对象时，成员变量需要有地方存储

为什么要程序计数器？

.java文件编译称.class文件，这个class文件是字节码文件，这个字节码文件里面是一条条字节码指令，对应一条条机器指令。计算机只有读到这种机器指令，才知道要干什么。字节码指令通过一个叫字节码引擎的东西来执行，在执行一条条指令过程中，字节码引擎需要知道指令执行到哪一条了，程序计数器就是起到这个作用，帮助字节码引擎记录目前程序执行到那一条字节码指令了。

什么是内存?

1. 内存是一个硬件
2. cpu有cache可以存储数据
3. 速度：cache>内存>硬盘

容量: cache<内存<硬盘

1. 内存用于暂时存储CPU的数据，以及与硬盘等外部存储器或者设备交换的数据。
2. 内存是CPU能直接寻址的存储空间
3. 计算机中所有程序都运行在内存中
4. 操作系统本质是一个程序（软件），也会在内存中运行，随着计算机的启动，操作系统首先加载进内存，占用一定的内存空间，操作系统占用的内存空间是保护起来的，不允许别的进程访问

方法区中存储的类信息是什么？

1. 类变量
2. 描述类的元数据



描述这段程序JVM运行过程？

首先Kafa.java通过javac命令编译成class文件，会得到两个class文件，Kafa.class和ReplicaManager.class。

使用java命令运行系统，操作系统会在后台启动一个java.exe的进程，并为其分配一定的内存，这个进程就是JVM，JVM负责我们编写的系统的运行。

JVM首先查找main方法程序入口，发现它在Kafa类中，于是类加载器会首先加载Kafka类，然后启动一个main线程，为这个main线程分配一个mian线程栈，在该线程中调用main方法，该线程会在线程栈中为main方法分配一个栈帧，保存main方法信息和局部变量。同时JVM字节码引擎会将字节码指令一条条指向，JVM会为main线程分配一个程序计数器，用于记录本线程的字节码指令执行到了那一条。开始执行main方法里面的代码，发现需要使用到ReplicaManager类，于是类加载器将ReplicaManager通过同样的验证、准备、解析、初始化阶段加载进内存，在main线程栈的main方法栈帧中会压入一个局部变量replicaManager，该变量是一个引用变量，存储的是地址，指向堆中的new出来的ReplicaManager对象。

该引用变量调用loadReplicaFromDisk方法，会在main线程栈中为该方法分配栈帧，栈帧中压入 replicCount局部变量，当该方法运行结束后，会出栈。

我们在Java堆内存中分配的那些对象，到底会占用多少内存？一般怎么来计算和估算我们系统创建对象内存占用的一个压力？

一个对象占用的空间=对象本身的信息占用空间+对象的实例变量占用空间

什么是垃圾回收机制？

JVM进程自身由垃圾回收机制，它是一个后台自动运行的线程，只要启动一个JVM，就会自带一个垃圾回收的线程。这个线程会在后台不断检查Java堆内存中的各种实例对象，是否有人引用，如果一个对象没有任何一个局部变量指向它，也没有任何静态变量，包括常量等指向它，那么这个垃圾回收线程，就会把这个没人指向的对象给回收掉，从内存中清楚掉，让他不占用任何资源。

垃圾就是指Java堆中没有任何变量指向的对象。

垃圾回收机制：垃圾回收线程会在后台定期回收垃圾，不断释放内存资源

为什么需要垃圾回收机制？

当某个方法执行完退出栈帧后，里面局部变量也会出栈，而局部变量执行的堆内存中的对象不会自己消失，这些对象没有任何局部变量、静态变量、常量引用，是不可操作的对象，被称为垃圾。如果没有垃圾回收对象，这些不能被操作的对象会 越来越多，而内存资源是有限的，最终导致堆区堆满垃圾对象，而无法在new对象。垃圾回收机制就应运而生，它是JVM自带的一个线程，随着JVM启动自动在后台启动，他会的查看堆中的对象是否由变量引用，如果没有就视为垃圾，然后定期的进行清理，以保证内存资源合理的利用。

Java堆区中的对象会被垃圾回收，那么加载到方法区中的类会被垃圾回收机制回收吗，什么时候回收呢？

会回收。满足以下三个条件时：

1. 该类的所有实例化对象都已经从Java堆中回收
2. 该类的类加载器已经被回收
3. 该类的class对象没有任何变量引用

初始化子类，父类会初始化吗？

父类会初始化，但此时父类和子类都是初始化同一快内存区域，所以初始化两次（或多次），只会产生一个对象。

为什么要双亲委派机制？

双亲委派机制可以防止类加载器重复加载类。

Tomcat支持动态部署和静态部署，阐述这两者的区别？

动态部署，又称热部署，指在Tomcat运行时部署或更新应用。

静态部署，指在Tomcat启动时加载和部署应用。

如果配置的热部署，则直接将war包放入webapp目录下，tomcat会自动检测到新的系统，自动部署或更新应用，无需重新启动tomcat。

如果配置的时静态部署，更新或部署新的项目时，需要停止tomcat容器，将新系统放在webapp目录下之后再启动容器。

热部署：运行时，无需重启tomcat

静态部署：启动时，需要重启tomcat

什么叫内存泄漏？

内存泄漏：程序没有释放已经不再使用的内存。

内存泄漏常发生在创建一个对象，不再使用这个对象后却没有删除它。

之所以叫泄漏，可以理解为对于程序员来说，随着没有释放的垃圾内存越来越多，好像可使用的内存在减少，就像泄漏一样。

内存泄漏会导致内存溢出问题

类加载器有三层，如果在第二层的类加载器可以加载这些类的话，就没有必要往上去找他的父类加载吗？

不对，第二层如果不是顶层类加载器的话，不管自己能不能加载，都会让父级类加载器先去加载，知道传递到顶层类加载器，如果顶层类加载器能够加载，就直接加载，如果不能够加载就下放，重复这个过程。

什么是年轻代、老年代、永久代？

JVM内存的一个分代模型

在我们代码中创建的对象，大部分都是存货周期很短的，随着方法调用的结束，该对象没有任何变量指向，成为垃圾，被回收。

有存活很短的对象：局部变量指向

也要存活很久的对象：静态变量指向

由于编写代码方式不同，创建和使用对象的方式不同，不同对象有不同的生命周期。所以JVM将堆内存区域划分成了两块区域：老年代，年轻代

老年代：长期存在的对象

年轻代：很快要被收回的对象

永久代：方法区，存放类信息

首先，大部分新创建的对象都在年轻代，经过一段时间的运行，一些对象没有任何变量引用，变成垃圾，而少部分对象任然被引用，这通常是被静态变量引用，于是JVM会将这部分对象从年轻代移动至老年代，并对年轻代的对象做一次清理。

为什么要区分老年代和年轻代呢？

这跟垃圾回收机制有关，因为对于不同生命周期的对象有不同的垃圾回收算法，所以通过区域来管理对象，方便算法的执行。年轻代和老年代分别进行垃圾回收。

对象什么时候会进入年轻代，什么时候会进入老年代？

1、大部分的正常对象，都是优先在年轻代分配内存

垃圾回收机制触发条件？

当年轻代里面的对象没有任何引用时，并不会立马进行清理，当年轻代堆积了大量的实例对象，然后程序员想在new一个对象时，JVM发现年轻代里面的内存不够了，此时就会让垃圾回收线程做一次垃圾回收，这个对年轻代里面的垃圾回收叫做Young GC或者Minor GC。它会尝试把新生代中没有人引用的对象给回收掉。我们在代码中创建的对象大多时生命周期很短的对象，大量对象创建引发新生代内存空间不足，然后垃圾回收机制进行Young GC，在这个过程中，长期存活的对象会躲过多次垃圾回收。

随着系统的启动运行，不停在创建对象，年轻代很快变满，然后再垃圾回收一次，大量对象被回收。但一些长期存活的对象在多次垃圾回收中都没有被回收掉，因为有变量一直在引用，需要一直使用。所以JVM规定，如果一个实例对象在新生代，成功地在15次垃圾回收之后，还没有被回收掉，就说明这个对象已经15岁了。

对象的年龄怎么定义？

如果每垃圾回收一次，这个对象没有被回收掉，年龄就会加1

年轻代和老年代分别进行垃圾回收是如何实现的？

Java虚拟机栈要进行垃圾回收吗？

不会、JVM垃圾回收机制时针对年轻代、老年代、永久代。方法一旦执行完毕，就会栈帧出栈，局部变量就直接从内存中清理出去了

JVM 内存分配核心参数?

|  |  |
| --- | --- |
| 参数 | 说明 |
| -Xms | Java堆内存大小；这是JVM启动时堆内存的大小 |
| -Xmx | Java堆内存最大大小；随着程序的运行堆内存可能不够，这里设置了允许扩大的最大内存大小。 |
| -Xmn | 年轻代内存大小;  堆内存=年轻代内存+老年代内存  老年代内存 = 堆内存 – 年轻代内存 |
| -XX:PermSize | 永久代内存大小 |
| -XX:MaxPermSize | 永久代最大大小 |
| -Xss | 每个线程栈内存大小 |

你的对象在JVM中是如何分配，如何流转的？

平时在代码中创建的对象分为两种：

1. 短期存活的对象
2. 长期存活的对象

老年代会垃圾回收吗？

会

Tomcat、Spring boot启动项目时输入设置JVM参数？

声明式事务在哪些情况下会失效？

Spring 的事务隔离级别？

AOP 原理

AOP 用到了 Java 中的哪些技术 ？

IOC AOP 说说自己的看法

Spring 中 Bean 的生命周期（Bean 的加载步骤）

Spring 的常用注解

Spring 注入的几种方式

Spring 装配 Bean 的几种方式

如何使用 IOC

通常如何获取 Bean

Bean 的扫描路径

AutoWired 是按照什么规则来获取 Bean 的 （Resource 呢？）

Spring 如何解决循环依赖

为什么要用三级缓存，二级缓存不行？

BeanFactory 和 FactoryBean 的区别？

SpringBoot 自动配置怎么实现的？

Spring 事务管理怎么实现的？

Spring 哪里用到了反射？

说说循环依赖?

那 Spring 怎么解决循环依赖的呢？

为什么要三级缓存？⼆级不⾏吗？

MVC Spring MVC 的核心组件？

Spring MVC 的工作流程？

SpringMVC Restful 风格的接口的流程是什么样的呢？

Spring Boot介绍一下 SpringBoot，有哪些优点？

SpringBoot 自动配置原理了解吗？

如何自定义一个 SpringBoot Srarter?

Springboot 启动原理？

Spring Cloud对 SpringCloud 了解多少？

volatile的作用和原理？

什么是原子性？

volatile能不能保证原子性？

你说你的项目用到了ThreadLocal，好好讲讲ThreadLocal是怎么回事吧？

ThreadLocal是什么?为什么要使用ThreadLocal ？

一个ThreadLocal的使用案例 ？

ThreadLocal的原理 ？

为什么不直接用线程id作为ThreadLocalMap的key ？

为什么会导致内存泄漏呢？是因为弱引用吗？

Key为什么要设计成弱引用呢？强引用不行？

InheritableThreadLocal保证父子线程间的共享数据 ？

ThreadLocal的应用场景和使用注意点？

你能跟我说说它隔离有什么用，会用在什么场景么？

除了源码里面使用到ThreadLocal的场景，你自己有使用他的场景么？一般你会怎么用呢？

嗯嗯，你回答得很好，那你能跟我说说他底层实现的原理么？

ThreadLocalMap底层结构是怎么样子的呢？

为什么需要数组呢？没有了链表怎么解决Hash冲突呢？

能跟我说一下对象存放在哪里么？

那么是不是说ThreadLocal的实例以及其值存放在栈上呢？

如果我想共享线程的ThreadLocal数据怎么办？

怎么传递的呀？

spring TransactionSynchronizationManager类？