Java 多线程 50 题

1) 什么是线程?

线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对运算密集型任务提速。比如，如果一个线程完成一个任务要 100 毫秒，那么用十个线程完成改任务只需 10 毫秒。

2) 线程和进程有什么区别?

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。更多详细信息请点击这里。

3) 如何在 Java 中实现线程?

java.lang.Thread 类的实例就是一个线程但是它需要调用 java.lang.Runnable 接口来执行，由于线程类本身就是调用的 Runnable 接口所以你可以继承 java.lang.Thread 类或者直接调用 Runnable 接口来重写 run()方法实现线程。

4) Thread 类中的 start()和 run()方法有什么区别?

这个问题经常被问到，但还是能从此区分出面试者对 Java 线程模型的理解程度。start()方法被用来启动新创建的线程，而且 start()内部调用了 run()方法，这和直接调用 run()方法的效果不一样。当你

调用 run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

5) Java 中 Runnable 和 Callable 有什么不同?

Runnable 和 Callable 都代表那些要在不同的线程中执行的任务。Runnable 从 JDK1.0 开始就有了，Callable 是在 JDK1.5 增加的。它们的主要区别是 Callable 的 call()方法可以返回值和抛出异常，而 Runnable 的 run()方法没有这些功能。Callable 可以返回装载有计算结果的 Future 对象。

6) Java 内存模型是什么?

Java 内存模型规定和指引 Java 程序在不同的内存架构、CPU 和操作系统间有确定性地行为。它在多线程的情况下尤其重要。Java 内存模型对一个线程所做的变动能被其它线程可见提供了保证，它们之间是先行发生关系。

- 线程内的代码能够按先后顺序执行，这被称为程序次序规则。
- 对于同一个锁，一个解锁操作一定要发生在时间上后发生的另一个锁定操作之前，也叫做管程锁定规则。
- 前一个对 Volatile 的写操作在后一个 volatile 的读操作之前，也叫 volatile 变量规则。
- 一个线程内的任何操作必需在这个线程的 start()调用之后，也叫作线程启动规则。
- 一个线程的所有操作都会在线程终止之前，线程终止规

则。

- 一个对象的终结操作必需在这个对象构造完成之后，也叫对象终结规则。
- 可传递性

7) Java 中的 volatile 变量是什么？

volatile 是一个特殊的修饰符，只有成员变量才能使用它。在 Java 并发程序缺少同步类的情况下，多线程对成员变量的操作对其它线程是透明的。volatile 变量可以保证下一个读取操作会在前一个写操作之后发生，就是上一题的 volatile 变量规则。

8) 什么是线程安全？Vector 是一个线程安全类吗？

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。一个线程安全的计数器类的同一个实例对象在被多个线程使用的情况下也不会出现计算失误。很显然你可以将集合类分成两组，线程安全和非线程安全的。Vector 是用同步方法来实现线程安全的，而和它相似的 ArrayList 不是线程安全的。

9) Java 中如何停止一个线程？

Java 提供了很丰富的 API 但没有为停止线程提供 API。JDK1.0 本来有一些像 stop()，suspend()和 resume()的控制方法但是由于潜在的死锁威胁因此在后续的 JDK 版本中他们被弃用了，之后 JavaAPI 的设计者就没有提供一个兼容且线程安全的方法来停止一个线程。

当 run()或者 call()方法执行完的时候线程会自动结束，如果要手动结束一个线程，你可以用 volatile 布尔变量来退出 run()方法的循环或者是取消任务来中断线程。

10)一个线程运行时发生异常会怎样?

如果异常没有被捕获该线程将会停止执行。

Thread.UncaughtExceptionHandler 是用于处理未捕获异常造成线程突然中断情况的一个内嵌接口。当一个未捕获异常将造成线程中断的时候 JVM 会使用 Thread.getUncaughtExceptionHandler()来查询线程的 UncaughtExceptionHandler 并将线程和异常作为参数传递给 handler 的 uncaughtException()方法进行处理。

11)如何在两个线程间共享数据?

可以通过共享对象来实现这个目的，或者是使用像阻塞队列这样并发的数据结构。用 wait 和 notify 方法实现了生产者消费者模型。

12)Java 中 notify 和 notifyAll 有什么区别?

notify()方法不能唤醒某个具体的线程，所以只有一个线程在等待的时候它才有用武之地。而 notifyAll()唤醒所有线程并允许他们争夺锁确保了至少有一个线程能继续运行。

13)为什么 wait,notify 和 notifyAll 这些方法不在 thread 类里面?

JAVA 提供的锁是对象级的而不是线程级的，每个对象都有锁，通过线程获得。如果线程需要等待某些锁那么调用对象中的 wait()方法就有意义了。如果 wait()方法定义在 Thread 类中，线程

正在等待的是哪个锁就不明显了。简单的说，由于 wait，notify 和 notifyAll 都是锁级别的操作，所以把他们定义在 Object 类中因为锁属于对象。

14)什么是 ThreadLocal 变量?

ThreadLocal 是 Java 里一种特殊的变量。每个线程都有一个 ThreadLocal 就是每个线程都拥有了自己独立的一个变量，竞争条件被彻底消除了。它是为创建代价高昂的对象获取线程安全的好方法，比如你可以用 ThreadLocal 让 SimpleDateFormat 变成线程安全的，因为那个类创建代价高昂且每次调用都需要创建不同的实例所以不值得在局部范围使用它，如果为每个线程提供一个自己独有的变量拷贝，将大大提高效率。首先，通过复用减少了代价高昂的对象的创建个数。其次，你在没有使用高代价的同步或者不变性的情况下获得了线程安全。线程局部变量的另一个不错的例子是 ThreadLocalRandom 类，它在多线程环境中减少了创建代价高昂的 Random 对象的个数。

15)什么是 FutureTask?

在 Java 并发程序中 FutureTask 表示一个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等方法。只有当运算完成的时候结果才能取回，如果运算尚未完成 get 方法将会阻塞。一个 FutureTask 对象可以对调用了 Callable 和 Runnable 的对象进行包装，由于 FutureTask 也是调用了 Runnable 接口所以它可以提交给 Executor 来执行。

16)为什么 wait 和 notify 方法要在同步块中调用?

主要是因为 JavaAPI 强制要求这样做，如果你不这么做，你的代码会抛出 IllegalMonitorStateException 异常。还有一个原因是为了避免 wait 和 notify 之间产生竞态条件。

17)为什么你应该在循环中检查等待条件?

处于等待状态的线程可能会收到错误警报和伪唤醒，如果不在循环中检查等待条件，程序就会在没有满足结束条件的情况下退出。因此，当一个等待线程醒来时，不能认为它原来的等待状态仍然是有效的，在 notify()方法调用之后和等待线程醒来之前这段时间它可能会改变。这就是在循环中使用 wait()方法效果更好的原因。

18)Java 中堆和栈有什么不同?

为什么把这个问题归类在多线程和并发面试题里？因为栈是一块和线程紧密相关的内存区域。每个线程都有自己的栈内存，用于存储本地变量，方法参数和栈调用，一个线程中存储的变量对其它线程是不可见的。而堆是所有线程共享的一片公用内存区域。对象都在堆里创建，为了提升效率线程会从堆中弄一个缓存到自己的栈，如果多个线程使用该变量就可能引发问题，这时 volatile 变量就可以发挥作用了，它要求线程从主存中读取变量的值。

19)什么是线程池? 为什么要使用它?

创建线程要花费昂贵的资源和时间，如果任务来了才创建线

程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从 JDK1.5 开始，JavaAPI 提供了 Executor 框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

20)如何避免死锁?

死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。这是一个严重的问题，因为死锁会让你的程序挂起无法完成任务，死锁的发生必须满足以下四个条件：

- 互斥条件：一个资源每次只能被一个进程使用。
- 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。
- 不剥夺条件：进程已获得的资源，在末使用完之前，不能强行剥夺。
- 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。

避免死锁最简单的方法就是阻止循环等待条件，将系统中所有的资源设置标志位、排序，规定所有的进程申请资源必须以一定的顺序（升序或降序）做操作来避免死锁。

21)怎么检测一个线程是否拥有锁?

在 java.lang.Thread 中有一个方法叫 holdsLock()，它返回 true 如果当且仅当当前线程拥有某个具体对象的锁。

22)JVM 中哪个参数是用来控制线程的栈堆栈小的

-Xss 参数用来控制线程的堆栈大小。

23)Java 中 synchronized 和 ReentrantLock 有什么不同?

Java 在过去很长一段时间只能通过 synchronized 关键字来实现互斥，它有一些缺点。比如你不能扩展锁之外的方法或者块边界，尝试获取锁时不能中途取消等。Java5 通过 Lock 接口提供了更复杂的控制来解决这些问题。ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义且它还具有可扩展性。

24)有三个线程 T1，T2，T3，怎么确保它们按顺序执行?

在多线程中有多种方法让线程按特定顺序执行，你可以用线程类的 join()方法在一个线程中启动另一个线程，另外一个线程完成该线程继续执行。为了确保三个线程的顺序你应该先启动最后一个(T3 调用 T2，T2 调用 T1)，这样 T1 就会先完成而 T3 最后完成。

25)Thread 类中的 yield 方法有什么作用?

yield 方法可以暂停当前正在执行的线程对象，让其它有相同优先级的线程执行。它是一个静态方法而且只保证当前线程放弃 CPU 占用而不能保证使其它线程一定能占用 CPU，执行 yield()的

线程有可能在进入到暂停状态后马上又被执行，所以 yield()不会释放锁。

26)如果你提交任务时，线程池队列已满。会时发会生什么？

事实上如果一个任务不能被调度执行那么 ThreadPoolExecutor’ssubmit()方法将会抛出一个 RejectedExecutionException 异常。

27)Java 线程池中 submit()和 execute()方法有什么区别？

两个方法都可以向线程池提交任务，execute()方法的返回类型是 void，它定义在 Executor 接口中，而 submit()方法可以返回持有计算结果的 Future 对象，它定义在 ExecutorService 接口中，它扩展了 Executor 接口，其它线程池类像 ThreadPoolExecutor 和 ScheduledThreadPoolExecutor 都有这些方法。

28)什么是阻塞式方法？

阻塞式方法是指程序会一直等待该方法完成期间不做其他事情，ServerSocket 的 accept()方法就是一直等待客户端连接。这里的阻塞是指调用结果返回之前，当前线程会被挂起，直到得到结果之后才会返回。此外，还有异步和非阻塞式方法在任务完成前就返回。

29)如果同步块内的线程抛出异常会发生什么？

无论你的同步块是正常还是异常退出的，里面的线程都会释放锁，所以对比锁接口我更喜欢同步块，因为它不用我花费精力去释放锁，该功能可以在 finallyblock 里释放锁实现。

30)单例模式的双检锁是什么？

它其实是一个用来创建线程安全的单例的老方法，当单例实例第一次被创建时它试图用单个锁进行性能优化。

31)如何在 Java 中创建线程安全的 Singleton?

可以利用 JVM 的类加载和静态变量初始化特征来创建 Singleton 实例，或者是利用枚举类型来创建 Singleton。

32)如何强制启动一个线程?

这个问题就像是如何强制进行 Java 垃圾回收，目前还没有觉得方法，虽然你可以使用 System.gc()来进行垃圾回收，但是不保证能成功。在 Java 里面没有办法强制启动一个线程，它是被线程调度器控制着且 Java 没有公布相关的 API。

33)Java 多线程中调用 wait()和 sleep()方法有什么不同?

Java 程序中 wait 和 sleep 都会造成某种形式的暂停，它们可以满足不同的需要。wait()方法用于线程间通信，如果等待条件为真且其它线程被唤醒时它会释放锁，而 sleep()方法仅仅释放 CPU 资源或者让当前线程停止执行一段时间，但不会释放锁。

34)什么是多线程编程？什么时候使用?

多线程一般用于当一个程序需要同时做一个以上的任务。多线程通常用于 GUI 交互程序。一个新的线程被创建做一些耗时的工作，当主线程保持界面与用户的交互。

35)可以重载 start()方法么？

可以重载，重载后还要重载 run（）方法。

36)编译运行下面的代码会发生什么？（D）

```
public class Bground extends Thread {
    public static void main(String argv[]) {
        Bground b = new Bground();
        b.run();
    }
    public void start(){
        for (int i = 0; i <10; i++) {
            System.out.println("Value of i = " + i);
        }
    }
}
```

A. 编译错误，Thread 类中的 run 方法没有定义

B. 运行时错误，Thread 类中的 run 方法没有定义

C. 编译无错，打印 0 到 9

D. 编译无错，不打印任何值

37)进程和线程之间有什么不同？

一个进程是一个独立(self contained)的运行环境，它可以被看作一个程序或者一个应用。而线程是在进程中执行的一个任务。Java 运行环境是一个包含了不同的类和程序的单一进程。线程可以被称为轻量级进程。线程需要较少的资源来创建和驻留在进程中，并且可以共享进程中的资源。

38)可以直接调用 Thread 类的 run()方法么？

当然可以，但是如果我们调用了 Thread 的 run()方法，它的行为就会和普通的方法一样，为了在新的线程中执行我们的代码，必须使用 Thread.start()方法。

39)你对线程优先级的理解是什么？

每一个线程都是有优先级的，一般来说，高优先级的线程在运行时会具有优先权，但这依赖于线程调度的实现，这个实现是和操作系统相关的(OS dependent)。我们可以定义线程的优先级，但是这并不能保证高优先级的线程会在低优先级的线程前执行。线程优先级是一个 int 变量(从 1-10)，1 代表最低优先级，5 代表中间优先级，10 代表最高优先级。

40)在多线程中，什么是上下文切换(context-switching)？

上下文切换是存储和恢复 CPU 状态的过程，它使得线程执行能够从中断点恢复执行。上下文切换是多任务操作系统和多线程环境的基本特征。

41)为什么线程通信的方法 wait(), notify()和 notifyAll()被定义在 Object 类里？

Java 的每个对象中都有一个锁(monitor，也可以成为监视器) 并且 wait()，notify()等方法用于等待对象的锁或者通知其他线程对象的监视器可用。在 Java 的线程中并没有可供任何对象使用的锁和同步器。这就是为什么这些方法是 Object 类的一部分，这样 Java 的每一个类都有用于线程间通信的基本方法。

42) 为什么 wait(), notify()和 notifyAll()必须在同步方法或者同步块中被调用？

当一个线程需要调用对象的 wait()方法的时候，这个线程必须拥有该对象的锁，接着它就会释放这个对象锁并进入等待状态直到其他线程调用这个对象上的 notify()方法。同样的，当一个线程需要调用对象的 notify()方法时，它会释放这个对象的锁，以便其他在等待的线程就可以得到这个对象锁。由于所有的这些方法都需要线程持有对象的锁，这样就只能通过同步来实现，所以他们只能在同步方法或者同步块中被调用。

43) 为什么 Thread 类的 sleep()和 yield()方法是静态的？

Thread 类的 sleep()和 yield()方法将在当前正在执行的线程上运行。所以在其他处于等待状态的线程上调用这些方法是没有意义的。这就是为什么这些方法是静态的。它们可以在当前正在执行的线程中工作，并避免程序员错误的认为可以在其他非运行线程调用这些方法。

44) 什么是阻塞队列？如何使用阻塞队列来实现生产者-消费者模型？

java.util.concurrent.BlockingQueue 的特性是：当队列是空的时，从队列中获取或删除元素的操作将会被阻塞，或者当队列是满时，往队列里添加元素的操作会被阻塞。阻塞队列不接受空值，当你尝试向队列中添加空值的时候，它会抛出 NullPointerException。阻塞队列的实现都是线程安全的，所有的查询方法都是原子的并且使用了内部锁或者其他形式的并发控制。BlockingQueue 接口是 java

collections 框架的一部分，它主要用于实现生产者-消费者问题。

45)Java 中多线程同步是什么？

在多线程程序下，同步能控制对共享资源的访问。如果没有同步，当一个 Java 线程在修改一个共享变量时，另外一个线程正在使用或者更新同一个变量，这样容易导致程序出现错误的结果。

46)Thread.start ()与 Thread.run ()有什么区别？

Thread.start ()方法(native)启动线程，使之进入就绪状态，当 cpu 分配时间该线程时，由 JVM 调度执行 run ()方法。

47)为什么需要 run ()和 start ()方法，我们可以只用 run ()方法来完成任务吗？

我们需要 run ()&start ()这两个方法是因为 JVM 创建一个单独的线程不同于普通方法的调用，所以这项工作由线程的 start 方法来完成，start 由本地方法实现，需要显示地被调用，使用这俩个方法的另外一个好处是任何一个对象都可以作为线程运行，只要实现了 Runnable 接口，这就避免因继承了 Thread 类而造成的 Java 的多继承问题。

48)Java 中的同步集合与并发集合有什么区别？

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。在 Java1.5 之前程序员们只有同步集合来用且在多线程并发的时候会导致争用，阻碍了系统的扩展性。Java5 介绍了并发集合像 ConcurrentHashMap，不仅提供线程安全还用锁分离和内部分区等现代技术提高了可扩展性。

49)Java 中活锁和死锁有什么区别?

活锁和死锁类似，不同之处在于处于活锁的线程或进程的状态是不断改变的，活锁可以认为是一种特殊的饥饿。一个现实的活锁例子是两个人在狭小的走廊碰到，两个人都试着避让对方好让彼此通过，但是因为避让的方向都一样导致最后谁都不能通过走廊。简单的说就是，活锁和死锁的主要区别是前者进程的状态可以改变但是却不能继续执行。

50)Java 中 synchronized 和 ReentrantLock 有什么不同?

Java 在过去很长一段时间只能通过 synchronized 关键字来实现互斥，它有一些缺点。比如你不能扩展锁之外的方法或者块边界，尝试获取锁时不能中途取消等。Java 5 通过 Lock 接口提供了更复杂的控制来解决这些问题。 ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义且它还具有可扩展性。

51)ThreadLocal(线程变量副本)

Synchronized 实现内存共享，ThreadLocal 为每个线程维护一个本地变量。

采用空间换时间，它用于线程间的数据隔离，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。

52)Java **内存模型**：

Java 虚拟机规范中将 Java 运行时数据分为六种。

- **程序计数器**：是一个数据结构，用于保存当前正常执行的程序的内存地址。Java 虚拟机的多线程就是通过线程轮流切换并分配处理器时间来实现的，为了线程切换后能恢复到正确的位置，每条线程都需要一个独立的程序计数器，互不影响，该区域为“线程私有”。
- **Java 虚拟机栈**：线程私有的，与线程生命周期相同，用于存储局部变量表，操作栈，方法返回值。局部变量表放着基本数据类型，还有对象的引用。
- **本地方法栈**：跟虚拟机栈很像，不过它是为虚拟机使用到的 Native 方法服务。
- **Java 堆**：所有线程共享的一块内存区域，对象实例几乎都在这分配内存。
- **方法区**：各个线程共享的区域，储存虚拟机加载的类信息，常量，静态变量，编译后的代码。
- **运行时常量池**：代表运行时每个 class 文件中的常量表。包

括几种常量：编译时的数字常量、方法或者域的引用。

53)你能不能谈谈，java GC 是在什么时候，对什么东西，做了什么事情？”

在什么时候：

- 新生代有一个 Eden 区和两个 survivor 区，首先将对象放入 Eden 区，如果空间不足就向其中的一个 survivor 区上放，如果仍然放不下就会引发一次发生在新生代的 minor GC，将存活的对象放入另一个 survivor 区中，然后清空 Eden 和之前的那个 survivor 区的内存。在某次 GC 过程中，如果发现仍然又放不下的对象，就将这些对象放入老年代内存里去。
- 大对象以及长期存活的对象直接进入老年区。
- 当每次执行 minor GC 的时候应该对要晋升到老年代的对象进行分析，如果这些马上要到老年区的老年对象的大小超过了老年区的剩余大小，那么执行一次 Full GC 以尽可能地获得老年区的空间。

对什么东西：

- 从 GC Roots 搜索不到，而且经过一次标记清理之后仍没有复活的对象。

做什么：

- 新生代：复制清理；
- 老年代：标记-清除和标记-压缩算法；
- 永久代：存放 Java 中的类和加载类的类加载器本身。

54)GC Roots 都有哪些?

- 虚拟机栈中的引用的对象
- 方法区中静态属性引用的对象，常量引用的对象
- 本地方法栈中 JNI（即一般说的 Native 方法）引用的对象。

55)谈谈 Synchronized 与 Lock

Synchronized 与 Lock 都是可重入锁，同一个线程再次进入同步代码的时候.可以使用自己已经获取到的锁。Synchronized 是悲观锁机制，独占锁。而 Locks.ReentrantLock 是，每次不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止。

ReentrantLock 适用场景:

- 某个线程在等待一个锁的控制权的这段时间需要中断。
- 需要分开处理一些 wait-notify，ReentrantLock 里面的 Condition 应用，能够控制 notify 哪个线程，锁可以绑定多个条件。
- 具有公平锁功能，每个到来的线程都将排队等候。

56)基本概念

- **同步**：就是一个任务的完成需要依赖另外一个任务，只有等待被依赖的任务完成后，依赖任务才能完成。
- **异步**：不需要等待被依赖的任务完成，只是通知被依赖的任务要完成什么工作，只要自己任务完成了就算完成了，被依赖的任务是否完成会通知回来。

- **阻塞：**CPU 停下来等一个慢的操作完成以后，才会接着完成其他的工作。
- **非阻塞：**非阻塞就是在这个慢的执行时，CPU 去做其他工作，等这个慢的完成后，CPU 才会接着完成后续的操作。非阻塞会造成线程切换增加，增加 CPU 的使用时间能不能补偿系统的切换成本需要考虑。

57) 线程池的作用：

在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程

- 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。
- 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。
- 提高线程的可管理性。

58) Spring IOC （控制反转，依赖注入）

Spring 支持三种依赖注入方式，分别是属性（Setter 方法）注入，构造注入和接口注入。

在 Spring 中，那些组成应用的主体及由 Spring IOC 容器所管理的对象被称之为 Bean。

Spring 的 IOC 容器通过反射的机制实例化 Bean 并建立 Bean 之间的依赖关系。

简单地讲，Bean 就是由 Spring IOC 容器初始化、装配及被管

理的对象。

获取 Bean 对象的过程，首先通过 Resource 加载配置文件并启动 IOC 容器，然后通过 getBean 方法获取 bean 对象，就可以调用他的方法。

Spring Bean 的作用域：

- Singleton：Spring IOC 容器中只有一个共享的 Bean 实例，一般都是 Singleton 作用域。
- Prototype：每一个请求，会产生一个新的 Bean 实例。
- Request：每一次 http 请求会产生一个新的 Bean 实例。

59)SpringMVC 运行原理

- 客户端请求提交到 DispatcherServlet
- 由 DispatcherServlet 控制器查询 HandlerMapping，找到并分发到指定的 Controller 中。
- Controller 调用业务逻辑处理后，返回 ModelAndView。
- DispatcherServlet 查询一个或多个 ViewResoler 视图解析器，找到 ModelAndView 指定的视图 。
- 视图负责将结果显示到客户端

60)HashMap 与 HashTable 的区别。

1、HashMap 是非线程安全的，HashTable 是线程安全的。

2、HashMap 的键和值都允许有 null 值存在，而 HashTable 则不行。

3、因为线程安全的问题，HashMap 效率比 HashTable 的要高。

HashMap 的实现机制：

1. 维护一个每个元素是一个链表的数组，而且链表中的每个节点是一个 Entry[]键值对的数据结构。

2. 实现了数组+链表的特性，查找快，插入删除也快。

3. 对于每个 key,他对应的数组索引下标是 int i = hash(key.hashcode)&(len-1);

4. 每个新加入的节点放在链表首，然后该新加入的节点指向原链表首

61)Hibernate 持久层框架

Hibernate 的一级缓存是由 Session 提供的，因此它只存在于 Session 的生命周期中，当程序调用 save(),update(),saveOrUpdate()等方法 及调用查询接口 list,filter,iterate 时，如 Session 缓存中还不存在相应的对象，Hibernate 会把该对象加入到一级缓存中，当 Session 关闭的时候缓存也会消失。

Hibernate 的一级缓存是 Session 所内置的，不能被卸载，也不能进行任何配置一级缓存采用的是 key-value 的 Map 方式来实现的，在缓存实体对象时，对象的主关键字 ID 是 Map 的 key，实体对象就是对应的值。

Hibernate 二级缓存：把获得的所有数据对象根据 ID 放入到第二级缓存中。Hibernate 二级缓存策略，是针对于 ID 查询的缓存策略，删除、更新、增加数据的时候，同时更新缓存。