一、在spring中有哪些设计模式经常被采用，会在哪些地方使用?

1. 单例模式：在创建bean的时候默认使用单例模式
2. 简单工厂：如BeanFactory
3. 工厂模式：如FactoryBean
4. 模板模式：各种模板方法，如JdbcTemplate、TransactionTemplate
5. 代理模式：在Aop的实现中用到了JDK和CGLIB的动态代理
6. 策略模式：Resource接口就是策略模式的典型应用，调用ApplicationContext实例的getResource方法，ApplicationContext会去自动确认使用Resource的某个实现类。另外在Aop中选择使用JDK或者CGLIB的动态代理也用到了策略模式。
7. 观察者模式：如ApplictionEvent和ApplicationListener就是用的观察者模式

二、数据库连接池的工作机制是什么？解决什么问题？

解决资源频繁分配、释放所造成的问题。传统的数据库操作，每次都要创建、释放数据库连接，频繁的创建和释放非常费时间，占内存。数据库连接池负责创建、分配和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接。数据库连接池需要设置一个连接最小值min、连接最大值max和最大空闲时间time。在数据库连接池初始化时就会创建min个数据库连接，一直维护，并根据应用程序的数据库连接请求数动态的管理数据库连接。

1. spring提供了哪两种事务处理方式，采用了java的什么技术

编程式和注解式，使用了java的动态代理技术。其中编程式通过TransactionTemplate手动管理事务

1. 12306 售卖模型/秒杀模型，怎么解决瞬时大并发的情况？
2. 前端做控制，防止同一用户短时间内的重复提交；
3. 后端用redis做控制，防止同一用户短时间内的重复提交；
4. 后端用redis存储一个count值，记录商品库存数（可以比实际库存数略大），再声明一个count大小的消息队列。当一个合法请求过来时，count-1，并将请求放到消息队列中。当count为0时，不再接收请求。
5. 平稳的从消息队列中取数据，然后插入数据库中，同时减数据库中的库存。当数据库操作成功时，提醒用户秒杀成功，否则秒杀失败。
6. 抽象类和接口的区别与联系？

抽象类 ：1.抽象类中可以构造方法   
2.抽象类中可以存在普通属性，方法，静态属性和方法。   
3.抽象类中可以存在抽象方法。   
4.如果一个类中有一个抽象方法，那么当前类一定是抽象类；抽象类中不一定有抽象方法。   
5.抽象类中的抽象方法，需要有子类实现，如果子类不实现，则子类也需要定义为抽象的。

6.JDK1.8之前，抽象类方法默认为protected，JDK1.8中，抽象类方法默认为default  
7.抽象类不能被实例化，抽象类和抽象方法必须被abstract修饰

关键字使用注意： 抽象类中的抽象方法（其前有abstract修饰）不能用private、static、synchronized、native访问修饰符修饰。

接口 ：1.在接口中只有方法的声明，没有方法体。   
2.在接口中只有常量，因为定义的变量，在编译的时候都会默认加上public static final   
3.在接口中的方法，永远都被public来修饰。   
4.接口中没有构造方法，也不能实例化接口的对象。（所以接口不能继承类）   
5.接口可以实现多继承   
6.接口中定义的方法都需要有实现类来实现，如果实现类不能实现接口中的所有方法则实现类定义为抽象类。   
7.接口可以继承接口，用extends

1. 什么是GC，什么时候会触发GC。

GC是用来监控堆中的对象的，在对象死亡后（通过GC root算法来判断），在合适的时间对死亡对象进行回收。堆中区域分为年轻代（new）和老年代（old），年轻代包括1个Eden区和2个Survivor区（from和to）（默认空间比例为8:1:1）。当我们声明一个对象时，比如声明一个大小为100的数组，此时会先去Eden区尝试去初始化一块连续内存空间。如果空间足够，则内存申请成功。如果空间不够，则会触发一次minor GC，使用标记复制算法，Eden区所有存活的对象被移动到Survivor区的to区。Survivor区的from区的存活对象中，年龄达到阈值（-XX:MaxTenuringThreshold）的，复制到老年区。年龄未达到的复制到Survivor区的to区，并且年龄+1。如果to区空间满了，会触发一次Full GC，对老年区的对象使用标记整理算法，回收老年区的死亡对象。再将to区的对象复制到老年区，如果老年区空间不够，则抛出java.lang.OutOfMemoryError异常。回到申请内存空间的过程，如果GC以后，Eden区能存放要申请的数组空间，则内存申请成功。如果Eden区存放不下，则在老年区尝试申请空间。如果老年区空间不够，则触发一次Full GC。Full GC后，如果老年区能存放要申请的数组空间，则内存申请成功，否则抛出java.lang.OutOfMemoryError异常。还有以下情况会触发GC：1.GC守护线程获取到CPU时。2.程序员显示调用System.gc(),建议JVM进行GC，但是是否会GC，以及何时GC是不确定的。

七、数据库四大特性（ACID）、三种问题、四种隔离级别。

四大特性：

**原子性**（Atomicity）：事务的操作要么全部完成，要么全部失败。比如A向B转账，要么转账成功，要么转账失败。不能有A的钱减少，B的钱没有增加的情况。

**一致性**（Consistency）：事务必须使数据库从一个一致性状态到另一个一致性状态。比如无论A、B之间不论如何转账，账号的钱数总和是一定的。

**隔离性**（Isolation）：事务之间不相互干扰。

**持久性**（Durability）：事务一旦成功，对数据库的改变就是永久性的。

三种问题：

**脏读**：一个事务读取了另一个事务中未提交的数据。比如B事务修改数据库某一个值从50到100，A事务读取到值100，B事务回滚值到50，A事务读到的100就是脏读数据。

**不可重复读**：一个事务多次查询同一个数据，返回了不同的值。比如A事务读取数据库某个值为50，B事务修改50为100成功，A事务再读时发现值变成了100。

**幻读**：一个事务多次查询同一批数据，返回的数据数量不一样。比如A事务读取表中有50条数据，B事务插入一条数据成功，A事务再读时发现数据变成了51条。

四种隔离级别：

**读未提交**(Read uncommitted )：最低级别，不做任务约束，脏读、不可重复读、幻读都会发生。

**读已提交**(Read committed)：避免了脏读。约束事务不可读取另一个事务未提交的数据。B事务修改数据库某一个值从50到100未提交，A事务读取到值仍然是50。

**可重复读**(Repeatable read)：避免了脏读和不可重复读。A事务读取数据库某个值为50，B事务修改50为100成功，A事务再读时值仍然是50。A只有再开一个新事务才能查到最新值100。

**串行化**(Serializable ):事务操作时，锁表，避免了所有问题，但是也造成了性能损耗。

1. HTTP请求过程

DNS域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

1. ArrayList和LinkedList的区别

ArrayList的实现底层为动态数组，继承自AbstractList，实现了RandmoAccess 、Cloneable 、Serializable 接口。默认初始容量为10，每次扩容变为原来的1.5倍。LinkedList为双向链表，实现了List接口和Deque接口，可以当做双端队列来使用，有1个队头指针和1个队尾指针。

ArrayList各操作时间复杂度如下：

add(E e):插到列表尾部，O(1)；

add(int index, E element)：指定位置插入，O(n)；

remove(int index)：删除指定位置元素，O(n)；

remove(Object o)：删除指定元素，O(n)；

set(int index, E element)：修改指定位置元素，O(1)；

get(int index)：获取指定位置的元素，O(1)；

indexOf(Object o)：查找指定元素的位置，O(n)；

LinkedList各操作时间复杂度如下：

add(E e)、addLast(E e):插到链尾，O(1)；

push(E e)、addFirst(E e)：插到链头，O(1)；

add(int index, E element)：指定位置插入，O(n)；

pop()、remove()、removeFirst()：删除队头元素，O(1)；

removeLast():删除队尾元素，O(1)；

remove(int index)：删除指定位置元素，O(n)；

remove(Object o)：删除指定元素，O(n)；

set(int index, E element)：修改指定位置元素，O(n)；

get(int index)：获取指定位置的元素，O(n)；

indexOf(Object o)：查找指定元素的位置，O(n)；

getFirst()、element()、peek()、peekFirst()：获取队头元素，O(1)；

getLast()、 peekLast() ：获取队尾元素，O(1)；