## 1.数据库事务特征

(1)原子性（atomicity)：一个事务要么全部提交成功，要么全部失败回滚，不能只执行其中的一部分操作。

(2)一致性（consistency)：事务的执行不能破坏数据库数据的完整性和一致性，一个事务在执行之前和执行之后，数据库都必须处于一致性状态。

(3)隔离性（isolation）：事务的隔离性是指在并发环境中，并发的事务时相互隔离的，一个事务的执行不能不被其他事务干扰。不同的事务并发操作相同的数据时，每个事务都有各自完成的数据空间，即一个事务内部的操作及使用的数据对其他并发事务时隔离的，并发执行的各个事务之间不能相互干扰。

(4)持久性（durability）：一旦事务提交，那么它对数据库中的对应数据的状态的变更就会永久保存到数据库中。即使发生系统崩溃或机器宕机等故障，只要数据库能够重新启动，那么一定能够将其恢复到事务成功结束的状态。

## 2.标准SQL规范中，事务隔离级别

(1)Read uncommitted

读未提交，顾名思义，就是一个事务可以读取另一个未提交事务的数据。

事例：老板要给程序员发工资，程序员的工资是3.6万/月。但是发工资时老板不小心按错了数字，按成3.9万/月，该钱已经打到程序员的户口，但是事务还没有提交，就在这时，程序员去查看自己这个月的工资，发现比往常多了3千元，以为涨工资了非常高兴。但是老板及时发现了不对，马上回滚差点就提交了的事务，将数字改成3.6万再提交。

分析：实际程序员这个月的工资还是3.6万，但是程序员看到的是3.9万。他看到的是老板还没提交事务时的数据。这就是脏读。

(2)Read committed

读提交，顾名思义，就是一个事务要等另一个事务提交后才能读取数据。

事例：程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（程序员事务开启），收费系统事先检测到他的卡里有3.6万，就在这个时候！！程序员的妻子要把钱全部转出充当家用，并提交。当收费系统准备扣款时，再检测卡里的金额，发现已经没钱了（第二次检测金额当然要等待妻子转出金额事务提交完）。程序员就会很郁闷，明明卡里是有钱的。

分析：这就是读提交，若有事务对数据进行更新（UPDATE）操作时，读操作事务要等待这个更新操作事务提交后才能读取数据，可以解决脏读问题。但在这个事例中，出现了一个事务范围内两个相同的查询却返回了不同数据，这就是不可重复读。

(3)Repeatable read

重复读，就是在开始读取数据（事务开启）时，不再允许修改操作

事例：程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（事务开启，不允许其他事务的UPDATE修改操作），收费系统事先检测到他的卡里有3.6万。这个时候他的妻子不能转出金额了。接下来收费系统就可以扣款了。

分析：重复读可以解决不可重复读问题。写到这里，应该明白的一点就是，不可重复读对应的是修改，即UPDATE操作。但是可能还会有幻读问题。因为幻读问题对应的是插入INSERT操作，而不是UPDATE操作。

什么时候会出现幻读？

事例：程序员某一天去消费，花了2千元，然后他的妻子去查看他今天的消费记录（全表扫描FTS，妻子事务开启），看到确实是花了2千元，就在这个时候，程序员花了1万买了一部电脑，即新增INSERT了一条消费记录，并提交。当妻子打印程序员的消费记录清单时（妻子事务提交），发现花了1.2万元，似乎出现了幻觉，这就是幻读。

(4)Serializable 序列化

Serializable 是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。

注：大多数数据库默认的事务隔离级别是Read committed，比如Sql Server, Oracle。Mysql的默认隔离级别是Repeatable read。

## 3.SpringBoot启动原理

main方法上的注解：@SpringBootApplication

三个注解核心注解：@SpringBootConfiguration，@EnableAutoConfiguration和@ComponentScan。

(1)@SpringBootConfiguration

根据Javadoc可知，该注解作用就是将当前的类作为一个JavaConfig，然后触发注解。(2)@EnableAutoConfiguration

借助@Import的帮助，将所有符合自动配置条件的bean定义加载到IoC容器。

其中最重要的两个注解已经标注：

①@AutoConfigurationPackage

②@Import(AutoConfigurationImportSelector.class)：注册**当前启动类的根package**。

(3)@ComponentScan

扫描的 Spring 对应的组件，如 @Componet，@Repository。

我们可以通过 basePackages 等属性来细粒度的定制 @ComponentScan 自动扫描的范围，如果不指定，则默认Spring框架实现会从声明 @ComponentScan 所在类的package进行扫描，所以 SpringBoot 的启动类最好是放在根package下，我们自定义的类就放在对应的子package下，这样就可以不指定 basePackages。

全流程：

(1)创建一个StopWatch实例，用来记录SpringBoot的启动时间

(2)通过SpringFactoriesLoader加载listeners：比如EventPublishingRunListener

(3)发布SprintBoot开始启动事件（EventPublishingRunListener#starting()）

(4)创建和配置environment（environmentPrepared()）

(5)打印SpringBoot的banner和版本

(6)创建对应的ApplicationContext：Web类型，Reactive类型，普通的类型(非Web)

(7)prepareContext

①准备ApplicationContext，Initializers设置到ApplicationContext(contextPrepared())

②打印启动日志，打印profile信息(如dev, test, prod)

③最终会调用到AbstractApplicationContext#refresh方法，实际上就是Spring IOC容器的创建过程，并且会进行自动装配的操作，以及发布ApplicationContext已经refresh事件，标志着ApplicationContext初始化完成(contextLoaded())

(8)afterRefresh hook方法

(9)stopWatch停止计时，日志打印总共启动的时间

(10)发布SpringBoot程序已启动事件(started())

(11)调用ApplicationRunner和CommandLineRunner

(12)最后发布就绪事件ApplicationReadyEvent，标志着SpringBoot可以处理就收的请求了(running())

## 4.Spring框架IOC/DI的理解

(1) IOC—Inversion of Control，即“控制反转”，是一种设计思想。对于Spring框架来说，就是由spring来负责控制对象的生命周期和对象间的关系。

举个简单的例子，我们是如何找女朋友的？常见的情况是，我们到处去看哪里有长得漂亮身材又好的mm，然后打听她们的兴趣爱好、qq号、电话号、ip号、iq号……，想办法认识她们，投其所好送其所要，然后嘿嘿……这个过程是复杂深奥的，我们必须自己设计和面对每个环节。传统的程序开发也是如此，在一个对象中，如果要使用另外的对象，就必须得到它（自己new一个，或者从JNDI中查询一个），使用完之后还要将对象销毁（比如Connection等），对象始终会和其他的接口或类藕合起来。

那么IOC是如何做的呢？有点像通过婚介找女朋友，在我和女朋友之间引入了一个第三者：婚姻介绍所。婚介管理了很多男男女女的资料，我可以向婚介提出一个列表，告诉它我想找个什么样的女朋友，比如长得像李嘉欣，身材像林熙雷，唱歌像周杰伦，速度像卡洛斯，技术像齐达内之类的，然后婚介就会按照我们的要求，提供一个mm，我们只需要去和她谈恋爱、结婚就行了。简单明了，如果婚介给我们的人选不符合要求，我们就会抛出异常。整个过程不再由我自己控制，而是有婚介这样一个类似容器的机构来控制。

Spring所倡导的开发方式就是如此，所有的类都会在Spring容器中登记，告诉Spring你是个什么东西，你需要什么东西，然后spring会在系统运行到适当的时候，把你要的东西主动给你，同时也把你交给其他需要你的东西。所有的类的创建、销毁都由 spring来控制，也就是说控制对象生存周期的不再是引用它的对象，而是spring。对于某个具体的对象而言，以前是它控制其他对象，现在是所有对象都被spring控制，所以这叫控制反转。

(2)DI—Dependency Injection，即“依赖注入”,是IOC的另外一种说法组件之间依赖关系由容器在运行期决定，形象的说，即由容器动态的将某个依赖关系注入到组件之中。

## 5.Spring框架AOP的理解

AOP即Aspect Oriented Programming，面向切面编程

AOP代理主要分为静态代理和动态代理

静态代理：AspectJ是静态代理的增强，采用编译时生成 AOP 代理类，因此也称为编译时增强，具有更好的性能。 缺点：但需要使用特定的编译器进行处理。

动态代理：

jdk动态代理实现AOP拦截

cglib动态代理实现AOP拦截

名词解释：

(1)切面（Aspect）：切面是通知和切点的结合。通知和切点共同定义了切面的全部内容。 在Spring AOP中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @AspectJ 注解来实现。

(2)连接点（Join point）：指方法，在Spring AOP中，一个连接点 总是 代表一个方法的执行。 应用可能有数以千计的时机应用通知。这些时机被称为连接点。连接点是在应用执行过程中能够插入切面的一个点。这个点可以是调用方法时、抛出异常时、甚至修改一个字段时。切面代码可以利用这些点插入到应用的正常流程之中，并添加新的行为。

(3)通知（Advice）：在AOP术语中，切面的工作被称为通知。

(4)切入点（Pointcut）：切点的定义会匹配通知所要织入的一个或多个连接点。我们通常使用明确的类和方法名称，或是利用正则表达式定义所匹配的类和方法名称来指定这些切点。

(5)引入（Introduction）：引入允许我们向现有类添加新方法或属性。

(6)目标对象（Target Object）： 被一个或者多个切面（aspect）所通知（advise）的对象。它通常是一个代理对象。也有人把它叫做 被通知（adviced） 对象。 既然Spring AOP是通过运行时代理实现的，这个对象永远是一个 被代理（proxied） 对象。

(7)织入（Weaving）：织入是把切面应用到目标对象并创建新的代理对象的过程。在目标对象的生命周期里有多少个点可以进行织入。

## 6.Servlet 生命周期

Servlet 生命周期可被定义为从创建直到毁灭的整个过程。

以下是 Servlet 遵循的过程：

(1)Servlet 通过调用 init () 方法进行初始化。

init 方法被设计成只调用一次。它在第一次创建 Servlet 时被调用，在后续每次用户请求时不再调用。

(2)Servlet 调用 service() 方法来处理客户端的请求。

service() 方法是执行实际任务的主要方法。Servlet 容器（即 Web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并把格式化的响应写回给客户端。

每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。

(3)Servlet 通过调用 destroy() 方法终止（结束）。

destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。destroy() 方法可以让您的 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。

(4)最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收。

## 7.JVM内存回收机制

当JVM创建对象遇到内存不足的时候，JVM会自动触发垃圾回收garbage collecting（简称GC）操作，将不再使用但仍存在JVM内存中的对象当做垃圾一样直接清理掉，释放被占用的内存空间，供新创建的对象使用。

那么问题来了，要让系统能够自动实现不被引用对象的回收，有几个问题需要解决：

Who：哪些是不再使用要被当做“垃圾”回收处理的对象？也就是要确定垃圾对象。

Where：在哪里执行垃圾回收？明确要清理的内存区域。

When：什么时候执行GC操作？即JVM触发GC的时机。

How：怎么样进行垃圾对象处理？即GC的实现算法。

(1)判断对象是否可以被回收(判别算法或搜索算法)

①引用计数法

每个对象创建的时候，会分配一个引用计数器，当这个对象被引用的时候计数器就加1，当不被引用或者引用失效的时候计数器就会减1。任何时候，对象的引用计数器值为0就说明这个对象不被使用了，就认为是“垃圾”，可以被GC处理掉。

评价：

【优点】算法实现简单。

【缺点】不能解决对象之间循环引用的问题。有垃圾对象不能被正确识别，这对垃圾回收来说是很致命的，所以GC并没有使用这种搜索算法。

②根搜索算法

以一些特定的对象作为基础原始对象，或者称作“根”，不断往下搜索，到达某一个对象的路径称为引用链。

如果一个对象和根对象之间有引用链，即根对象到这个对象是可到达的，则这个对象是活着的，不是垃圾，还不能回收。例如，假设有根对象O，O引用了A对象，同时A对象引用了B对象，B对象又引用了C对象，那么对象C和根对象O之间的路径的可达的，C对象就不能当做垃圾对象。引用链为O->A->B->C。

反之，如果一个对象和根对象之间没有引用链，根对象到这个对象的路径是不可达的，那么这个对象就是可回收的垃圾对象。

评价：

【优点】可找到所有的垃圾对象，并且完美解决对象之间循环引用的问题。

【缺点】不可避免地要遍历全局所有对象，导致搜索效率不高。

(2) GC算法

①标记-清除算法（Mark-Sweep）

该算法分两步执行：

1）标记Mark：从GC ROOTS开始，遍历堆内存区域的所有根对象，对在引用链上的对象都进行标记。这样下来，如果是存活的对象就会被做了标记，反之如果是垃圾对象，则没做有标记。GC很容易根据有没有被做标记就完成了垃圾对象回收。

2）清除Sweep：遍历堆中的所有的对象（标记阶段遍历的是所有根节点），找到未被标记的对象，直接回收所占的内存，释放空间。

评价：

【优点】没有产生额外的内存空间消耗，内存利用率高。

【缺点】效率低，清除阶段要遍历所有的对象；回收的垃圾对象是在各个角落的，直接回收垃圾对象，导致存在不连续的内存空间，产生内存碎片。

标记-清除算法操作的对象是【垃圾对象】，对于活着的对象（被标记的对象），它则直接不理睬。

②复制算法（Copying）

复制算法把内存区间一分为二，有对象存在的一半区间称为“活动区间”，没有对象存在处于空闲状态的空间则为“空闲区间”。

当内存空间不足时触发GC，先采用根搜索算法标记对象，然后把活着的对象全部复制到另一半空闲区间上，复制算法的“复制”就来自这一操作。复制到另一半区间的时候，严格按照内存地址依次排列要存放的对象，然后一次性回收垃圾对象。

这样原来的空闲区间在GC后就变成活动区间，而且内存顺序齐整美观。原来的活动区间在GC后就变成了完全空的空闲区间，等待下一次GC把活的对象被copy进来。

评价：

【优点】GC后的内存齐整，不产生内存碎片。

【缺点】GC要使用两倍的内存，或者说导致堆只能使用被分配到的内存的一半,这个算法对空间要求太高！如果存活的对象较多，则意味着要复制很多对象并且要维护大量对象的内存地址，所以存活的对象数量不能太多，否则效率也会很低。

复制算法复制移动的对象是【活着的对象】，对于垃圾对象（不被标记的对象）则直接回收。

③标记-整理算法（Mark-Compact）

这个算法则是对上面两个算法的综合结果。也分为两个阶段：

1）标记：这个阶段和标记-清除Mark-Sweep算法一样，遍历GC ROOTS并标记存活的对象。

2）整理：移动所有活着的对象到内存区域的一侧（具体在哪一侧则由GC实现），严格按照内存地址次序依次排列活着的对象，然后将最后一个活着的对象地址以后的空间全部回收。

评价：

【优点】内存空间利用率高，消除了复制算法内存减半的情况；GC后不会产生内存碎片。

【缺点】需要遍历标记活着的对象，效率较低；复制移动对象后，还要维护这些活着对象的引用地址列表。

④分代回收算法（Generational Collecting）

举例：我是一个普通的java对象，我出生在Eden区，在Eden区我还看到和我长的很像的小兄弟，我们在Eden区中玩了挺长时间。有一天Eden区中的人实在是太多了，我就被迫去了Survivor区的“From”区，自从去了Survivor区，我就开始漂了，有时候在Survivor的“From”区，有时候在Survivor的“To”区，居无定所。直到我18岁的时候，爸爸说我成人了，该去社会上闯闯了。于是我就去了年老代那边，年老代里，人很多，并且年龄都挺大的，我在这里也认识了很多人。在年老代里，我生活了20年(每次GC加一岁)，然后被回收。

GC过程：

HotSpot JVM把年轻代分为了三部分：1个Eden区和2个Survivor区（分别叫from和to）。默认比例为8：1,为啥默认会是这个比例，接下来我们会聊到。一般情况下，新创建的对象都会被分配到Eden区(一些大对象特殊处理),这些对象经过第一次Minor GC后，如果仍然存活，将会被移到Survivor区。对象在Survivor区中每熬过一次Minor GC，年龄就会增加1岁，当它的年龄增加到一定程度时，就会被移动到年老代中。

　　因为年轻代中的对象基本都是朝生夕死的(80%以上)，所以在年轻代的垃圾回收算法使用的是复制算法，复制算法的基本思想就是将内存分为两块，每次只用其中一块，当这一块内存用完，就将还活着的对象复制到另外一块上面。复制算法不会产生内存碎片。

在GC开始的时候，对象只会存在于Eden区和名为“From”的Survivor区，Survivor区“To”是空的。紧接着进行GC，Eden区中所有存活的对象都会被复制到“To”，而在“From”区中，仍存活的对象会根据他们的年龄值来决定去向。年龄达到一定值(年龄阈值，可以通过-XX:MaxTenuringThreshold来设置)的对象会被移动到年老代中，没有达到阈值的对象会被复制到“To”区域。经过这次GC后，Eden区和From区已经被清空。这个时候，“From”和“To”会交换他们的角色，也就是新的“To”就是上次GC前的“From”，新的“From”就是上次GC前的“To”。不管怎样，都会保证名为To的Survivor区域是空的。Minor GC会一直重复这样的过程，直到“To”区被填满，“To”区被填满之后，会将所有对象移动到年老代中。

## 8.TCP和UDP的区别

TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。

UDP---用户数据报协议，是一个简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快。

(1)TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接

(2)TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付

(3)TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流；UDP是面向报文的，UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）

(4)每一条TCP连接只能是点到点的；UDP支持一对一，一对多，多对一和多对多的交互通信

(5)TCP首部开销20字节；UDP的首部开销小，只有8个字节

(6)TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

## 9.TCP三次握手

(1)第一次握手([SYN], Seq = x)

客户端发送一个SYN标记的包，Seq初始序列号x，发送完成后客户端进入SYN\_SEND状态。

(2)第二次握手([SYN,ACK], Seq = y, ACK = x + 1)

服务器返回确认包(ACK)应答，同时还要发送一个SYN包回去。ACK = x + 1,表示确认收到(客户端发来的Seq值 + 1)，Seq = y, 表示让客户端确认是否能收到。发送完成后服务端进入SYN\_RCVD状态。

(3)第三次握手([ACK], ACK = y + 1)

客户端再次发送确认包(ACK),ACK = y + 1, 表示确认收到服务器的包（服务端发来的Seq值 + 1）。客户端发送完毕后，进入ESTABLISHED状态，服务端接收到这个包，也进入ESTABLISHED状态, TCP握手结束。

## 10.TCP四次挥手

(1)TCP连接的断开需要发送四个包，所以称为四次挥手。

第一次挥手（[FIN], Seq = x）

客户端发送一个FIN标记的包，告诉服务器需要关闭连接，表示自己不用发送数据了，但是还可以接收数据。发送完成后，客户端进入FIN\_WAIT\_1状态。

(2)第二次挥手 ([ACK], ACK = x + 1)

服务端发送一个ACK的确认包，告诉客户端接收到关闭的请求，但是还没有准备好。发送完成后，服务端进入CLOSE\_WAIT状态，客户端收到这个包后，进入FIN\_WAIT\_2，等待服务器关闭连接。

(3)第三次挥手 ([FIN], Seq = y)

服务端准备好关闭连接时，发送FIN标记的包，告诉客户端准备关闭了。发送完成后，服务端进入LAST\_ACK状态，等待客户端确认。

(4)第四次挥手 ([ACK], ACK = y + 1)

客户端接收到服务端的关闭请求，再发送ACK标记的确认包，进入TIME\_WAIT状态，等待服务端可能请求重传的ACK包。

服务端接收到ACK包后，关闭连接，进入CLOSED状态。

客户端在等待固定时间(两个最大段生命周期)后，没有接收到服务的ACK包，认为服务器已关闭连接，自己也关闭连接，进入CLOSED状态。

## 11.Spring Cloud常用组件

Spring Cloud Netflix Eureka：服务注册中心。

Spring Cloud Zookeeper：服务注册中心。

Spring Cloud Consul：服务注册和配置管理中心。

Spring Cloud Netflix Ribbon：客户端负载均衡。

Spring Cloud Netflix Hystrix：服务容错保护。

Spring Cloud Netflix Feign：声明式服务调用。

Spring Cloud OpenFeign(可替代 Feign)：OpenFeign 是 Spring Cloud 在 Feign 的基础上支持了 Spring MVC 的注解，如 @RequesMapping等等。OpenFeign 的 @FeignClient 可以解析 SpringMVC 的 @RequestMapping 注解下的接口，并通过动态代理的方式产生实现类，实现类中做负载均衡并调用其他服务。

Spring Cloud Netflix Zuul：API 网关服务，过滤、安全、监控、限流、路由。

Spring Cloud Gateway(可替代 Zuul)：Spring Cloud Gateway 是 Spring 官方基于 Spring 5.0，Spring Boot 2.0 和 Project Reactor 等技术开发的网关，Spring Cloud Gateway 旨在为微服务架构提供一种简单而有效的统一的 API 路由管理方式。Spring Cloud Gateway 作为 Spring Cloud 生态系中的网关，目标是替代 Netflix Zuul，其不仅提供统一的路由方式，并且基于 Filter 链的方式提供了网关基本的功能，例如：安全，监控/埋点，和限流等。

Spring Cloud Security：安全认证。

Spring Cloud Config：分布式配置中心。配置管理工具，支持使用 Git 存储配置内容，支持应用配置的外部化存储，支持客户端配置信息刷新、加解密配置内容等。

Spring Cloud Bus：事件、消息总线，用于在集群（例如，配置变化事件）中传播状态变化，可与 Spring Cloud Config 联合实现热部署。

Spring Cloud Stream：消息驱动微服务。

Spring Cloud Sleuth：分布式服务跟踪。

Spring Cloud Alibaba Nacos：阿里巴巴开源产品，一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。

Spring Cloud Alibaba Sentinel：面向分布式服务架构的轻量级流量控制产品，把流量作为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。

Spring Cloud Alibaba RocketMQ：一款开源的分布式消息系统，基于高可用分布式集群技术，提供低延时的、高可靠的消息发布与订阅服务。

Spring Cloud Alibaba Dubbo：Apache Dubbo™ 是一款高性能 Java RPC 框架，用于实现服务通信。

Spring Cloud Alibaba Seata：阿里巴巴开源产品，一个易于使用的高性能微服务分布式事务解决方案。

## 12.高并发解决思路

扩容：水平扩容、垂直扩容

缓存：Redis、Memcache、GuavaCache等

队列：Kafka、RabitMQ、RocketMQ等

应用拆分：服务化Dubbo与微服务Spring Cloud

限流：Guava RateLimiter使用、常用限流算法、自己实现分布式限流等

服务降级与服务熔断：服务降级的多重选择、Hystrix

数据库切库，分库分表：切库、分表、多数据源

高可用的一些手段：任务调度分布式elastic-job、主备curator的实现、监控报警机制

## 13.常用排序算法

(1) 简单选择排序

常用于取序列中最大最小的几个数时。

(如果每次比较都交换，那么就是交换排序；如果每次比较完一个循环再交换，就是简单选择排序。)

遍历整个序列，将最小的数放在最前面。

遍历剩下的序列，将最小的数放在最前面。

重复第二步，直到只剩下一个数。

代码实现：

首先确定循环次数，并且记住当前数字和当前位置。

将当前位置后面所有的数与当前数字进行对比，小数赋值给key，并记住小数的位置。

比对完成后，将最小的值与第一个数的值交换。

重复2、3步。

1 public void selectSort(int[] a){

2 int len=a.length;

3 for(int i=0;i<len;i++){//循环次数

4 int value=a[i];

5 int position=i;

6 for(int j=i+1;j<len;j++){//找到最小的值和位置

7 if(a[j]<value){

8 value=a[j];

9 position=j;

10 }

11 }

12 a[position]=a[i];//进行交换

13 a[i]=value;

14 }

15 }

(2)冒泡排序

很简单，用到的很少，据了解，面试的时候问的比较多！

将序列中所有元素两两比较，将最大的放在最后面。

将剩余序列中所有元素两两比较，将最大的放在最后面。

重复第二步，直到只剩下一个数。

代码实现：

设置循环次数。

设置开始比较的位数，和结束的位数。

两两比较，将最小的放到前面去。

重复2、3步，直到循环次数完毕。

1 public void bubbleSort(int[] a){

2 int len=a.length;

3 for(int i=0;i<len;i++){

4 for(int j=0;j<len-i-1;j++){//注意第二重循环的条件

5 if(a[j]>a[j+1]){

6 int temp=a[j];

7 a[j]=a[j+1];

8 a[j+1]=temp;

9 }

10 }

11 }

12 }

(3) 快速排序

要求时间最快时。

选择第一个数为p，小于p的数放在左边，大于p的数放在右边。

递归的将p左边和右边的数都按照第一步进行，直到不能递归。

1 public void quickSort(int[]a,int start,int end){

2 if(start<end){

3 int baseNum=a[start];//选基准值

4 int midNum;//记录中间值

5 int i=start;

6 int j=end;

7 do{

8 while((a[i]<baseNum)&&i<end){

9 i++;

10 }

11 while((a[j]>baseNum)&&j>start){

12 j--;

13 }

14 if(i<=j){

15 midNum=a[i];

16 a[i]=a[j];

17 a[j]=midNum;

18 i++;

19 j--;

20 }

21 }while(i<=j);

22 if(start<j){

23 quickSort(a,start,j);

24 }

25 if(end>i){

26 quickSort(a,i,end);

27 }

28 }

29 }

## 14.常用查找算法

(1)顺序查找

描述：从表中的第一个或者是最后一个记录开始，将表中记录的关键字和给定的值进行逐个比较，若某个记录的关键字和给定值相等，则查找成功，若表中所记录的关键字和给定值都不相等，则查找失败。

算法实现：

/\*\*

\* 顺序查找

\*

\* @param searchKey 要查找的值

\* @param array 数组（从这个数组中查找）

\* @return 查找结果（数组的下标位置）

\*/

public static int orderSearch(int searchKey, int[] array) {

if (array == null || array.length < 1)

return -1;

for (int i = 0; i < array.length; i++) {

if (array[i] == searchKey) {

return i;

}

}

return -1;

}

(2)二分查找

描述：折半查找的前提条件是在一个有序的序列汇总，首先确定待查询记录所在的区间，然后逐步的缩小范围区间直到找到或者找不到该记录为止，与数学中的二分法一样。

算法实现：

/\*\*

\* 二分查找(折半查找)，它是一种效率较高的查找方法

\* 二分查找要求：1.必须采用顺序存储结构

\* 2.必须按关键字大小有序排列

\*

\* @param array 有序数组 \*

\* @param searchKey 查找元素 \*

\* @return searchKey的数组下标，没找到返回-1

\*/

public static int binarySearch(int[] array, int searchKey) {

int low = 0;

int high = array.length - 1;

while (low <= high) {

int middle = (low + high) / 2;

if (searchKey == array[middle]) {

return middle;

} else if (searchKey < array[middle]) {

high = middle - 1;

} else {

low = middle + 1;

}

}

return -1;

}

## 15.接口和抽象类

(1)抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract方法；

(2)抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；

(3)接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；

(4)一个类只能继承一个抽象类，而一个类却可以实现多个接口。

## 16.get请求和post请求的区别

GET在浏览器回退时是无害的，而POST会再次提交请求。

GET产生的URL地址可以被收藏，而POST不可以。

GET请求会被浏览器主动缓存，而POST不会，除非手动设置。

GET请求只能进行url编码，而POST支持多种编码方式。

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

GET请求在URL中传送的参数是有长度限制的，而POST么有。

对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。

GET参数通过URL传递，POST放在Request body中。

## 17.HashMap底层数据结构

JDK1.7：数组+单向链表

JDK1.8：数组+单向链表+红黑树

数组是HashMap的主体，链表则是主要为了节解决哈希碰撞(两个对象调用的hashCode方法计算的哈希码值一致导致计算的数组索引值相同)而存在的（“拉链法”解决冲突）。

当**链表长度大于阈值**（或者红黑树的边界值，默认为8）并且当前**数组的长度大于64**时，此时此索引位置上的所有数据改为使用红黑树存储。

数组里面都是key-value的实例，在JDK1.8之前叫做Entry，在JDK1.8之后叫做Node。

将链表转换成红黑树前会判断，如果阈值大于8，但是数组长度小64，此时并不会将链表变为红黑树。而是选择进行数组扩容。

这样做的目的是因为数组比较小，尽量避开红黑树结构，这种情况下变为红黑树结构，反而会降低效率，因为红黑树需要进行左旋，右旋，变色这些操作来保持平衡。同时数组长度小于64时，搜索时间相对快一些。所以综上所述为了提高性能和减少搜索时间，底层在阈值大于8并且数组长度大于64时，链表才转换为红黑树。

## 18.SQL优化

在表中建立索引，优先考虑where、group by使用到的字段。

尽量避免使用select \*，返回无用的字段会降低查询效率。

尽量避免使用in 和not in，会导致数据库引擎放弃索引进行全表扫描。

尽量避免使用or，会导致数据库引擎放弃索引进行全表扫描。

尽量避免在字段开头模糊查询，会导致数据库引擎放弃索引进行全表扫描。

尽量避免进行null值的判断，会导致数据库引擎放弃索引进行全表扫描。

尽量避免在where条件中等号的左侧进行表达式、函数操作，会导致数据库引擎放弃索引进行全表扫描。

当数据量大时，避免使用where 1=1的条件。通常为了方便拼装查询条件，我们会默认使用该条件，数据库引擎会放弃索引进行全表扫描。

## 19.建立索引的原则

定义主键的数据列一定要建立索引。

定义有外键的数据列一定要建立索引。

对于经常查询的数据列最好建立索引。

对于需要在指定范围内的快速或频繁查询的数据列;

经常用在WHERE子句中的数据列。

经常出现在关键字order by、group by、distinct后面的字段，建立索引。如果建立的是复合索引，索引的字段顺序要和这些关键字后面的字段顺序一致，否则索引不会被使用。

对于那些查询中很少涉及的列，重复值比较多的列不要建立索引。

对于定义为text、image和bit的数据类型的列不要建立索引。

对于经常存取的列避免建立索引。

限制表上的索引数目。对一个存在大量更新操作的表，所建索引的数目一般不要超过3个，最多不要超过5个。索引虽说提高了访问速度，但太多索引会影响数据的更新操作。

对复合索引，按照字段在查询条件中出现的频度建立索引。在复合索引中，记录首先按照第一个字段排序。对于在第一个字段上取值相同的记录，系统再按照第二个字段的取值排序，以此类推。因此只有复合索引的第一个字段出现在查询条件中，该索引才可能被使用,因此将应用频度高的字段，放置在复合索引的前面，会使系统最大可能地使用此索引，发挥索引的作用。

## 20.MyBatis与Hibernate区别

(1)相同点

Hibernate与MyBatis都可以是通过SessionFactoryBuider由XML配置文件生成SessionFactory，然后由SessionFactory 生成Session，最后由Session来开启执行事务和SQL语句。

其中SessionFactoryBuider，SessionFactory，Session的生命周期都是差不多的。Hibernate和MyBatis都支持JDBC和JTA事务处理。

(2) 不同点

①hibernate是全自动，而mybatis是半自动

hibernate完全可以通过对象关系模型实现对数据库的操作，拥有完整的JavaBean对象与数据库的映射结构来自动生成sql。而mybatis仅有基本的字段映射，对象数据以及对象实际关系仍然需要通过手写sql来实现和管理。

②hibernate数据库移植性远大于mybatis

hibernate通过它强大的映射结构和hql语言，大大降低了对象与数据库（Oracle、MySQL等）的耦合性，而mybatis由于需要手写sql，因此与数据库的耦合性直接取决于程序员写sql的方法，如果sql不具通用性而用了很多某数据库特性的sql语句的话，移植性也会随之降低很多，成本很高。

③hibernate拥有完整的日志系统，mybatis则欠缺一些

hibernate日志系统非常健全，涉及广泛，包括：sql记录、关系异常、优化警告、缓存提示、脏数据警告等；而mybatis则除了基本记录功能外，功能薄弱很多。

④mybatis相比hibernate需要关心很多细节

hibernate配置要比mybatis复杂的多，学习成本也比mybatis高。但也正因为mybatis使用简单，才导致它要比hibernate关心很多技术细节。mybatis由于不用考虑很多细节，开发模式上与传统jdbc区别很小，因此很容易上手并开发项目，但忽略细节会导致项目前期bug较多，因而开发出相对稳定的软件很慢，而开发出软件却很快。hibernate则正好与之相反。但是如果使用hibernate很熟练的话，实际上开发效率丝毫不差于甚至超越mybatis。

⑤sql直接优化上，mybatis要比hibernate方便很多

由于mybatis的sql都是写在xml里，因此优化sql比hibernate方便很多。而hibernate的sql很多都是自动生成的，无法直接维护sql；虽有hql，但功能还是不及sql强大，见到报表等变态需求时，hql也歇菜，也就是说hql是有局限的；hibernate虽然也支持原生sql，但开发模式上却与orm不同，需要转换思维，因此使用上不是非常方便。总之写sql的灵活度上hibernate不及mybatis。

⑥缓存机制上，hibernate要比mybatis更好一些

MyBatis的二级缓存配置都是在每个具体的表-对象映射中进行详细配置，这样针对不同的表可以自定义不同的缓存机制。并且Mybatis可以在命名空间中共享相同的缓存配置和实例，通过Cache-ref来实现。而Hibernate对查询对象有着良好的管理机制，用户无需关心SQL。所以在使用二级缓存时如果出现脏数据，系统会报出错误并提示。

## 21.Hibernate的缓存机制

(1)一级缓存

使用一级缓存的目的是为了减少对数据库的访问次数，从而提升hibernate的执行效率；（当执行一次查询操作的时候，执行第二次查询操作，先检查缓存中是否有数据，如果有数据就不查询数据库，直接从缓存中获取数据）；

Hibernate中的一级缓存，也叫做session的缓存，它可以在session范围内减少数据库的访问次数，只在session范围内有效，session关闭，一级缓存失败；

一级缓存的特点，只在session范围有效，作用时间短，效果不是特别明显，在短时间内多次操作数据库，效果比较明显。

当调用session的save/saveOrUpdate/get/load/list/iterator方法的时候，都会把对象放入session缓存中；

session的缓存是由hibernate维护的，用户不能操作缓存内容；如果想操作缓存内容，必须通过hibernate提供的evict/clear方法操作

缓存相关的方法（在什么情况下使用上面方法呢？批量操作情况下使用，如Session.flush();先与数据库同步，Session.clear();再清空一级缓存内容）：

session.flush();让一级缓存与数据库同步；

session.evict();清空一级缓存中指定的对象；

session.clear();清空一级缓存中所有的对象；

(2)二级缓存

基于应用程序级别的缓存，作用在SessionFactory范围内的， 可以跨多个session，即不同的session都可以访问缓存。 Hibernate提供的二级缓存有默认的实现，且是一种可插配的缓存框架！如果用户想用二级缓存，只需要在hibernate.cfg.xml中配置即可； 不想用，直接移除，不影响代码。

在执行各种条件查询时，如果所获得的结果集为实体对象的集合，那么就会把所有的数据对象根据ID放入到二级缓存中。

当Hibernate根据ID访问数据对象的时候，首先会从Session一级缓存中查找，如果查不到并且配置了二级缓存，那么会从二级缓存中查找，如果还查不到，就会查询数据库，把结果按照ID放入到缓存中。

删除、更新、增加数据的时候，同时更新缓存。

注意：

在通常情况下会将具有以下特征的数据放入到二级缓存中：

很少被修改的数据。

不是很重要的数据，允许出现偶尔并发的数据。

不会被并发访问的数据。

常量数据。

不会被第三方修改的数据

而对于具有以下特征的数据则不适合放在二级缓存中：

经常被修改的数据。

财务数据，绝对不允许出现并发。

与其他应用共享的数据。

## 22.多线程实现

有三种使用线程的方法：

(1)实现 Runnable 接口；

需要实现 run() 方法。

通过 Thread 调用 start() 方法来启动线程。

public class MyRunnable implements Runnable {

public void run() {

// ...

}

}

public static void main(String[] args) {

MyRunnable instance = new MyRunnable();

Thread thread = new Thread(instance);

thread.start();

}

(2)实现 Callable 接口；

与 Runnable 相比，Callable 可以有返回值，返回值通过 FutureTask 进行封装。

public class MyCallable implements Callable<Integer> {

public Integer call() {

return 123;

}

}

public static void main(String[] args) throws ExecutionException, InterruptedException {

MyCallable mc = new MyCallable();

FutureTask<Integer> ft = new FutureTask<>(mc);

Thread thread = new Thread(ft);

thread.start();

System.out.println(ft.get());

}

实现 Runnable 和 Callable 接口的类只能当做一个可以在线程中运行的任务，不是真正意义上的线程，因此最后还需要通过 Thread 来调用。可以说任务是通过线程驱动从而执行的。

(3)继承 Thread 类。

同样也是需要实现 run() 方法，因为 Thread 类也实现了 Runable 接口。

当调用 start() 方法启动一个线程时，虚拟机会将该线程放入就绪队列中等待被调度，当一个线程被调度时会执行该线程的 run() 方法。

public class MyThread extends Thread {

public void run() {

// ...

}

}

public static void main(String[] args) {

MyThread mt = new MyThread();

mt.start();

}

23.MVC流程

MVC是三个单词的首字母缩写，它们是Model（模型）、View（视图）和Controller（控制）。

(1)用户发送请求至 前端控制器DispatcherServlet。

(2)前端控制器DispatcherServlet收到请求后调用处理器映射器HandlerMapping。

(3)处理器映射器HandlerMapping根据请求的Url找到具体的处理器，生成处理器对象Handler及处理器拦截器HandlerIntercepter（如果有则生成）一并返回给前端控制器DispatcherServlet。

(4)前端控制器DispatcherServlet通过处理器适配器HandlerAdapter调用处理器Controller。

(5)执行处理器（Controller，也叫后端控制器）

(6)处理器Controller执行完后返回ModelAnView。

(7)处理器映射器HandlerAdapter将处理器Controller执行返回的结果ModelAndView返回给前端控制器DispatcherServlet。

(8)前端控制器DispatcherServlet将ModelAnView传给视图解析器ViewResolver。

(9)视图解析器ViewResolver解析后返回具体的视图View。

(10)前端控制器DispatcherServlet对视图View进行渲染视图（即：将模型数据填充至视图中）

(11)前端控制器DispatcherServlet响应用户。

## 23.过滤器Filter和拦截器Interceptor的区别

过滤器是基于函数回调的，而拦截器则是基于Java反射的。

过滤器依赖于Servlet容器，而拦截器不依赖于Servlet容器。

过滤器对几乎所有的请求起作用，而拦截器只能对action请求起作用。

拦截器可以访问Action的上下文，值栈里的对象，而过滤器不能。

在action的生命周期里，拦截器可以被多次调用，而过滤器只能在容器初始化时调用一次。

过滤器可以修改request，而拦截器不能；

过滤器需要在servlet容器中实现，拦截器可以适用于javaEE，javaSE等各种环境；

拦截器可以调用IOC容器中的各种依赖，而过滤器不能；

过滤器只能在请求的前后使用，而拦截器可以详细到每个方法。