|  |
| --- |
| 题目1【2021/7/20】 说明StringBuffer和StringBuilder的异同。  延申：HashMap&Hashtable; ArrayList&Vector; Hashtable&Properties; Synchronizied同步特性;Lock;java.util.concurrent;悲观锁和乐观锁;锁的级别;死锁;读写锁;锁池和等待队列。 StringBuffer和StringBuilder StringBuffer 和 String Builder 非常类似，都是可变字符串，StringBuffer是JDK1.0提供的类，线程安全，而StringBuilder是JDk1.5提供的类，是线程不安全的，大概率只要线程不安全就会比线程安全的效率要高。 HashMap和Hash table 而HashMap和Hashtable也是一个线程不安全，一个线程安全的集合，HashMap是新版JDK提供的接口，（map和serializable）继承的Abstractmap,HashTable是实现早期JDK的接口（map和Serializable）继承的Dictionary，HashTable 不允许null值，hashmap允许空值。 ArrayList和Vector ArrayList和Vector也是一样，Vector是线程安全的集合，Array List是不安全的，但是效率比Vector高，然后长度增长的时候，Vector默认增长一倍，Arraylist增长50%。 synchronized synchronized，必须是引用数据类型，不能是基本数据类型,而且同步代码块中可以发生CPU切换，但是后面的代码也没法执行，锁也是close关闭状态。做到了安全，但是效率低，而且会出现死锁。 死锁 死锁是由于多个线程共享多个资源，或者多个线程都需要其他线程的资源，每个线程无法放弃自己的资源。解决死锁的方法就是加锁 lock锁 说到加锁，除了有synchronized锁，还有lock锁，它是一个接口，轻量级锁，只能锁代码块，等待的线程可以响应中断，提前结束等待，发生异常不会自动释放锁，synchronized发生异常会自动释放锁，其中trylock可以通过返回值直到是否获取到了锁   锁的级别 为了在一定程度上减少获得锁和释放锁带来的性能消耗，在jdk6之后引入了“偏向锁”和“轻量级锁”，就有了4种状态，级别由高到低：无锁状态，偏向锁状态，轻量级锁，重量级锁 juc juc是concurrent包的简称，目的就是为了更好支持高并发，其中有一个经典的ABA问题，就是线程A要通过CAS操作把变量A修改成变量B，修改之前会判断值还是不是A，这个过程就可能发生线程B把A修改成了B，然后线程C把B又修改成了A，但是这个A已经不是最原来的那个A了，解决方法就是记录每次修改的时间，或者记录每次修改的版本。  synchronized 是有原子性，有序性，有序性。而volatile是一种稍弱的同步机制，volatile的读性能消耗和普通的变量几乎相同，但是在写操作稍慢，因为他需要在本地代码种插入许多内存屏障指令来保证处理器不发生乱序执行。  lock锁底层就是volatile+CAS来实现的  CAS是原子操作，是通过系统操作指令实现的原子操作， compare and swap(比较和交换) 悲观锁和乐观锁 乐观锁与悲观锁不是具体的什么类型的锁，而是看待并发同步的角度  悲观锁认为对于同一个数据是并发操作，一定是会发生修改的，哪怕是没有修改的 |

|  |
| --- |
| 题目2【2021/7/21】 说明IO流的特性（方向，单位，功能），介绍常用的IO流接口及类型（类名，功能）。  延申： 装饰器模式。Socket&ServerSocket。BIO/NIO/AIO。Servlet&SpringMVC。XML（良构，合法）。 特性 按流的方向分类  输入流：数据流向是数据源到程序  输出流：数据流向是程序都目的地  输入、输出流的划分是相对程序而言的，并不是相对数据源  按处理的数据单元分类  字节流：以字节为单位获取数据，命名上以Stream结尾的流一般是字节流，  字符流：以字符为单位获取数据，  按处理对象不同分类  节点流，可以从数据源或者目的地读写数据  处理流：不直接连接到数据源或目的地，是“处理流的流”  序列化：将对象的状态信息转换为可以存储或传输的形式的过程 BIO/NIO/AIO BIO 同步并阻塞，服务器实现模式为一个连接一个线程，即客户端有连接请求时，服务器就需要启动一个线程进行处理，如果这个连接步做任何事情会造成不必要的线程开销  这种方式对服务器资源要求比较高  NIO 同步非阻塞，服务器实现模式为一个线程处理多个请求，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有I/O请求就进行处理  使用于连接数目多且连接比较短的架构  AIO 异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的I/o请求都是由OS先完成了再通知服务器应用去启动线程进行处理，一般适用于连接较多且连接时间较长的应用，AIO相当于NIO的升级版本2.0（只处理有效请求）  适用于连接数目多且连接比较长的架构  NIO有三大核心部分：Channel(通道)，Buffer(缓冲区)，Selector(选择器)  通道：就像火车的轨道  Buffer 缓冲区，作用就是存放数据的，底层就是一个数组  Selector ：列车的调度员  注意：只有通道是不可以传递数据的，只可以配合缓冲区一块进行数据的传递  TCP和UDP的区别  连接方面：TCP面向连接 UDP是无连接的  安全方面：TCP提供可靠服务，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尽可能努力交付，即不保证可靠交付  传输效率区别：  TCP传输效率相对较低  UDP传输效率高，适用于高速传输和实时性有较高的通信或广播通信  连接对象数量的区别：  TCP连接是能是点对点，一对一的  UDP支持一对一，一对多，多对一，和多对多的交互通信  XML 格式良好的文档必须遵循  元素正确嵌套  xml文件时第一行必须是xml声明  xml文件只能有一个根节点  英文字符的大小写时有差异的  开始的控制标记与结束的控制标记缺一不可  属性值的设置必须被包围起来  有效的xml文档  首先必须时格式是良好的  使用DTD和XSD定义语义约束  Servlet对象是被多线程共享的一个对象，servlet在服务期间只会被创建一次，而tomcat服务器每接受一个请求就会创建一个线程来调用Servlet对象中的service方法来处理请求，Servlet的声明周期是有容器管理的，有5个阶段，  类加载🡪new:实例化🡪init 初始化🡪service 执行服务🡪destroy回收销毁  ServletContext，当容器启动时，会创建ServletContext对象，并一直缓存该对象，直到容器关闭后该对象生命周期结束，ServletContext对象的声明周期非常长  请求转发： 能实现资源的共享  请求转发是一次请求内，地址栏地址不变  请求转发内的servlet共享request对象  请求转发是tomcat内部的行为，用户看不到  可以使用setAttribute和getAttribute实现请求转发内的Servlet的数据的流转  重定向：重定向两次请求，地址栏信息改变 不能实现数据的共享  重定向的地址是浏览器接受响应后，自主的直接跳转了，用户不参与  两次请求，request对象是不同的  能跨域访问外部资源  不能访问WEB-INF安全目录下的资源  同源策略是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响， 可以说WEB是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。 同源策略会阻止一个域的Javascript脚本和另一个域的内容进行交互。所谓的同源就是两个页面具有相同的协议，主机，端口号，  HttpSession和Cookie的区别：  cookie数据是存放在客户的浏览器或系统的文件汇总的，而HttpSession中的数据存放在服务器中，压力分配不一样  cookie不安全，而HttpSession是安全的  单个保存的数据不超过4k， 而HttpSession没有容量以及数量的限制  四大域对象  Page域 对应 pageContext 也叫页面域 就这个页面有效，出了页面也无效  Request域 对应 HttpServlettRequest对象 也叫请求域 当下请求有效  Session域 对应 HTTP Session对象 也叫会话域 这次会话有效  Application 对应 ServletContext对象 也叫应用域 项目创建直到项目关闭  MVC模式 V：视图层 controller 控制层 M：service层mapper层 实体类  耦合性低  重用性高  开发效率提高，人员职责明确  耦合度低，可维护性高  有利软件工程化管理  JSP 执行过程  JSP的执行过程大致可以分为两个时期：转译时期和请求时期  转译期间：jsp网页转译成servlet 生成java文件，然后进行编译生成class字节码文件  请求时期：运行.class字节码文件，处理请求  springMVC就是将servlet进行了封装，提供一个公共的Servlet。该servlet可以根据请求动态的调用对应的逻辑方法完成请求处理  springMVC和struts2的区别  共同点：它们都是表现层框架，都是基于MVC模型编写的  它们的底层都离不开原始ServletAPI  它们处理请求的机制都是一个核心控制器  区别：springMVC入口时servlet，而sturts2是Filter  springMVC是基于方法设计的，而Sturts2是基类，Struts2每次执行都会创建一个动作类，所以springMVC会稍微比Struts2快一些  springMVC执行流程   1. 一个请求匹配前端控制器DispatcherServlet的请求映射路径（在web.xml）WEB容器将该请求转交给DispatcherServlet处理 2. Dispatcher Servlet接收到请求后，将根据请求信息交给处理器映射器（HandlerMapping） 3. HandlerMapping根据用户的url请求，查找匹配该url的Handler，并返回一个执行链 4. DispatcherServlet再请求处理器适配器（Handler Adapter）调用相应的Handler进行处理并返回ModelAndView给DispatcherServlet 5. DispatcherServlet将ModelAndView请求ViewReslover（视图解析器）解析，返回具体View 6. DispatcherServlet将View进行渲染视图 7. Dispatcherservet将页面响应给用户   springMVC中的异常处理  系统中异常包括两类，预期异常和运行时异常RuntimeException,系统的mapper service controller出现都通过throws exception 向上抛出，最后由springMVC前端控制器交由异常处理器进行异常处理，   1. 使用@ExceptionHandler注解处理异常，只能处理当前Controller中的异常 2. @ControllerAdvice+@ExceptionHandler 全局异常处理器 3. 使用SimpleMappingExceptionResolver 全局异常 4. HandlerExceptionResolver 自定义异常处理器   用@CrossOrigin来处理跨域问题 |

|  |
| --- |
| 题目3【2021/7/22】 说明JavaSE中的集合结构，常用接口、实现。底层数据结构。接口特性。  延申： Set|Map集合-Object中常用方法。TreeXxx涉及到排序算法-Comparable|Comparetor。所有的集合都有数据结构。Map-Redis。Queue-MQ。Redis底层数据结构（字典）。  Collect接口下有list和Set接口 ，List下有ArrayList和LinkedList，set下有Hash Set和Treeset  map接口和Collect是并排的，但是平常一说集合都说list和set和map三种。Map下有Hashmap和Treemap  其中List接口存储内容可重复，有顺序  Set接口存储的内容不可重复，没有顺序  ArrayList 查询效率比较高，在内存中分配连续的空间，实现了长度可变的数组  LinkedList 采用双向链表存储方式，插入和删除效率高  Arraylist底层就是一个长度可以动态增长的Object数组，（StringBuilder底层就是一个长度可以动态增长的char数组）。jdk1.7中使用无参构造方法创建ArrayList对象是，默认底层长度是10 jdk1.8中，默认底层数组长度是0 第一次添加元素，在扩容。  容量不足时进行扩容，默认扩容50%，如果扩容50%还不足容纳新增元素，就扩容为能容纳新增元素的最小数量。  linkedList 实现了Deque接口，所以除了可以作为线性表来使用外，还可以当作队列和栈来使用  Set 特点：无序，唯一  HashSet 采用Hashtable哈希表存储结构，添加速度快，查询速度快，删除速度快  缺点：无序  LinkHashset采用哈希表存储结构，同时使用链表维护次序 有序  Treeset 采用二叉树（红黑树）的存储结构  优点：有序 查询速度比list快  缺点：查询速度没有HashSet快  散列函数的构造方法   1. 直接定址法：这种方法时取关键字值的某个线性函数作为散列函数 H(k)=ak+b 直接定址法比较适合于关键字值的分布基本连续，或者关键字值有一定规律的情况 2. 数字分析法：当关键字的位数大于散列地址码的位数是，对关键字值的各位数字进行分析，从中取出于散列地址位数相同的位，这种使用于所有关键字都已经知道的情况下 3. 平方取中法：就是先平方取中间的几位，适用于关键字值中每一位都不够分散的情况 4. 叠加法：和平方取中类似，加起来取后面几位， 5. 基数转换法：就是把原来的10进制数当作q进制数，然后再转换成10进制数 6. 除留余数法：最常用的方法，取模运算，在Hashmap中用的是右移动一位做位运算。 7. 随机数法：这种不确定性因素太多，不建议使用的（计算散列函数所需要的时间，关键字长度，散列表大小，关键字分布情况，记录的查找概率）   处理冲突的方法：   1. 开放定址法：是将散列表中的“空”地址向处理冲突开放，   D = (H(K)+d)MOD m  D是要算出的那个地址 H(k)是散列函数构造方法算出来的地址， m是数组的长度  d有三种取法，第一种：线性探测再散列 就从1 2 3 挨个取值  第二种：二次探测再散列 d= 1²，-1²，2²，-2²，3²。。。。。这样取  第三种：随机探测再散列   1. 再散列法 在发生冲突的时候，用不同的散列函数再求得新的散列地址，直到不发生冲突为止。 2. 链地址法：就是相同地址后，用一个链表存储，称为桶， 3. 建立一个公共溢出区   ConcurrentHashmap 在jdk1.7的时候 分段锁+lock锁  在jdk1.8的时候，利用volatile+CAS 实现无锁化操作  CopyOnWriteArrayList CopyOnWrite+Lock锁 CopyOnWrite原理：写时复制，就是先复制一份，所以希望写入的数据马上能读到，就不要用CopyOnWrite容器  CopyOnWriteArraySet CopyOnWirte + lock锁  redis数据类型  String 字符串 Hash哈希表 List列表 Set集合 Sorted Set有序集合  Stream （新类型，存二进制数组） Geo类型 （存经纬度坐标）  Redis 每次启动时都会从硬盘存储文件中把数据读取到内存中，运行过程中操作的数据都是内存中的数据  Redis一共包含两种持久化策略  RDB：Redis DataBase 默认模式，可以在指定的时间间隔内生成数据快照，默认保存到dump.rdb文件中，当redis重启后，会自动加载dump.rdb文件中内容到内存中。。  AOF：原理，监听执行的命令，如果发现执行了修改数据的操作，同时直接同步到数据库文件中。即使突然出现问题，由于日志文件中已经记录命令，下一次启动时也可以按照日志进行恢复数据，由于内存数据和硬盘数据实时同步，即使出现意外情况也不需要担心。  RDB比AOF快， |

|  |
| --- |
| 题目4【2021/7/23】 介绍JDBC访问数据库的6个基本步骤及事务管理方式。JDBC的默认事务是提交还是回滚？  延申：MyBatis框架，MyBatis框架默认事务提交还是回滚，JDBC注册驱动的底层逻辑，MyBatis框架中的executor执行器种类，executor执行器的特性，executor执行器中使用的JDBC Statement类型，Spring中事务管理的方式，spring xml配置事务管理方式，spring注解事务管理方式，spring中的事务传播行为，数据库事务隔离级别。  JDBC6个基本实现步骤：   1. 加载一个Driver驱动 2. 创建数据库连接 （Connection） 3. 创建SQL命令发送器Statement 4. 通过Statement发送SQL命令并得到结果 5. 处理结果 （Select语句） 6. 关闭数据库资源 ResultSet Statement Connection   PreparedStatement和statment的关系和区别  PreparedStatement 安全性高，可以避免SQL注入 ： （就是再密码处拼接恒成立表达式）  PreparedStatement 简单不繁琐，不用进行字符串拼接  PreparedStatement 性能高，用在执行多个相同数据库DML操作时  JDBC事务操作缺省是自动提交  注册原理：一开始通过反射技术，把Driver类加载进内存，而且Driver里面有个内置的静态代码块，在进入内存时会Driver类会初始化，静态代码块里的代码也会被执行  Driver内置静态代码块去调用Java.sql.DriverManager的方法registerDriver(Driver driver)完成注册驱动，DriverManager里就会有个newDriver()的对象。最后通过驱动管理器Driver Manager的getConnection方法去获取Connection与数据库建立连接  Mybatis本身不做事务管理，而是交给其他框架去处理事务  控制事务传播行为 propagation  REQUIRED（默认值） 如果当前有事务，就在事务中执行，如果当前没有事务，新建一个事务  SUPPORTS 如果当前有事务就在事务中执行，如果没有事务，就在非事务下执行  MANDATORY 必须在事务内部执行，如果当前有事务就在事务中执行，如果没有事务，报错  REQUIRES\_NEW 必须在事务中执行，如果当前没有事务，新建事务，如果当前有事务，把当前事务挂起  NOT\_SUPPORTED 必须在非事务下执行，如果当前没有事务，正常执行，如果当前有事务，把当前事务挂起  NEVER：必须是在非事务状态，如果当前没有事务，正常执行，如果当前有事务，报错  NESTED：必须在事务状态下执行，如果没有事务，新建事务，如果当前有事务，创建一个嵌套事务  事务的隔离级别 isolation  脏读，不可重复读，幻读， |

|  |
| --- |
| 题目5【2021/7/25】 介绍Servlet结构，并简介Servlet继承树中各类型|接口的特性。说明JSP和Servlet关系，阐述JSP的生命周期。  延申：Filter，Listener。责任链设计模式，观察者设计模式。Filter和Interceptor的异同。Spring AOP概念。代理设计模式，（动态代理|静态代理）。    Servlet 接口  方法：init() 创建Servlet对象后立即调用该方法完成初始化工作。  service，处理客户端请求，执行业务操作，利用响应对象响应客户端请求  destrory 在销毁Servlet对象之前调用方法，释放资源  getServletConfig ServletConfig是容器向servlet传递参数的载体  getServletInfo() 获取servlet相关信息  GenericServlet抽象类  定义了一个通用的，不依赖于具体协议的servlet，给出了除service（）方法外的其余四个方法实现  我们平常重写的是第二个init方法  HttpServlet  继承自GenericServlet，针对处理HTTP协议的请求所定制  jsp的本质就是一个Servlet（就是类），单实例多线程的程序。jsp负责页面，是一种动态页面技术，servlet负责跳转，两个都是在服务器端执行。  JSP执行过程  JSP的执行过程大致可以分为三个时期，转译（翻译），编译，执行  转译：JSP网页转译成Servlet，生成.java文件，然后进行编译生成.class字节码文件  编译：JSP网页转译成Servlet，生成.java文件，然后进行编译生成.class字节码文件  运行： 运行.class字节码文件，处理请求  Spring AoP 为了解决不修改原有功能逻辑的基础上完成功能拓展  AOP 面向切面编程，通过预编译方式和运行期间动态代理实现程序功能的统一维护的一种技术，利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。  filter 和 interceptor 技术层面不一样  filter是servlet的技术，interceptor是spring的技术  filter是过滤servlet层面上的，而servlet是javaweb开发最底层的技术，客户端发的任意请求，过滤器都会处理。  interceptor只能拦截spring容器内的，就是dispatchservlet才拦截。 |

|  |
| --- |
| 题目6【2021/7/28】 数据库开发过程中，SQL编写的注意事项有什么（SQL优化）？索引的使用注意事项有什么（索引特征，种类（单字段，复合|聚合，全文），特点（唯一，稠密，稀疏））？  延申：索引底层数据结构，索引的限制（什么样的SQL不使用索引，字段数量限制等），MySQL各种常用的函数。  SQL优化 ：   1. 对查询进行优化，应尽量避免全表扫描，首先应考虑在where和order by涉及的列上加索引 2. 应尽量避免在where子句种对字段进行null值判断，否则将导致引擎放弃使用索引而进行全表扫描 3. 应尽量避免在where子句中使用！=或<>操作符，和进行函数操作，否则将引擎放弃使用索引 进行全表扫描 4. 对于连续的数值能用between，就不要用in或者not in 不要用模糊搜索 5. 在使用索引字段作为条件时，如果该索引时符合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致 6. 很多时候exists代替in是一个好的选择， 7. 任何地方都不要使用\*，用具体的字段来代替\*， 8. 尽可能用varchar代替char。   多行函数  COUNT 统计表中记录的数目  SUM 计算指定字段值的总和  AVG 计算指定字段值的平均值  MAX 统计指定字段值的最大值  MIN 统计指定字段的最小值  B树和B-树是同一种树，假如用平衡二叉树实现索引，效率已经很高了，查找一个节点所作的IO次数是这个节点所处的树的高度，因为我们无法把整个索引都加载到内存，并且节点数据在磁盘中不是顺序排放的，所以最坏的情况，磁盘的IO次数为树的高度  虽然平衡二叉树查找效率确实很高，但是频繁的IO才是阻碍提高性能的瓶颈，怎样减少IO次数 呢。前辈们很聪明的提出了局部性原理，分为时间局部性原理，即假如你查询ID为1的用户数据，过一段时间你还会查询di为1的数据，所以会将这部分数据缓存下来，  空间局部性原理，当你查询di为1的用户数据的时候，你有很大的概率会去查询id为 2 3 4 的用户的数据，所以会一次性把id为1 2 3 4 的数据都读到内存中，这个最小的单位就是页 |

|  |
| --- |
| 题目7【2021/7/29】 简述MyBatis框架开发流程，细致说明MyBatis中的SqlSessionFactory创建过程。  延申：SqlSession创建方式，Mapper接口的动态代理要求，XML配置文件中的动态SQL标签（sql,include,set,trim,where,if,foreach），SqlSession中执行SQL语句的执行器Executor类型，默认类型，MyBatis底层封装的是JDBC，MyBatis框架单独使用的时候默认事务管理方式是什么。  Mybatis有三种基本的Executor执行器，SimpleExecutor ReuseExecutor,BatchExecutor  SimpleExecutor 每执行一次update或select 就开启一个statement对象，用完立刻关闭statement对象  ReuseExecutor执行update或select 以sql作为key查找statement对象，存在就使用，不存在就创建，用完后，不关闭statement对象，而是放置于Map内，供下一次使用，简言之，就是重复使用statement对象  BatchExecutor执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()）等待统一执行executeBatch() 它缓存了多个Statement对象，每个Statement对象都是addBatch完毕后，等待逐一执行executebatch()批处理，与JDBC批处理相同  MyBatis仅支持association关联对象和collection关联集合对象的延迟加载  association指的就是一对一，collection指的就是一对多查询，在MyBatis配置文件中，可以配置启用延迟加载。它的原理是使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，拦截器发现方法是null值的话，就会单独发送事先保存好的查询关联兑现的sql，把结果查出来，然后属性就不是null值， 这就是延迟加载的基本原理。  】  #{ }和${ } 的区别  #{} 是占位符，预编译处理 ${ } 是拼接符，字符串替换，没有预编译处理  Mybatis在处理#{}时，#{ } 传入参数是以字符串传入，会将SQL中的#{}替换为？号，调用PreparedStatement的set方法来赋值  变量替换后，#{}对应的变量自动加上单引号， 变量替换后，${ }对应的变量不会加上单引号  #{ }可以有效的防止SQL注入，提供系统的安全性，${}不能防止SQL注入  二级缓存是以namespace为标记的缓存，可能要借助磁盘，磁盘上的缓存。可以是由一个SqlSessionFactory 创建的SqlSession之间共享缓存数据 |

|  |
| --- |
| 题目8【2021/7/30】 Elasticsearch服务器中的文档读写底层流程是什么？  涉及到：内存，缓存（操作系统缓存，虚拟内存），文件，IO，多线程 |

|  |
| --- |
| 题目9【2021/8/2】 介绍Spring框架核心概念（IoC、AOP）。并解释IoC执行流程（ApplicationContext初始化流程）。  延申：动态代理、静态代理。工厂（静态工厂，实例工厂，工厂方法）。单例（Spring容器默认管理单例对象）。Spring容器对Bean对象的管理方案（lazy-init, scope(singleton,prototype)）。BeanFactory和ApplicationContext的关系（异同）。  spring框架的核心 IOC容器和AOP模块，通过IOC容器管理pojo对象以及他们之间的耦合关系；通过AOP以动态非侵入的方式增强服务  IOC让相互协作的组件保持松散的耦合，而AOP编程允许你把遍布于应用各层的功能分离出来形成可重用的功能组件。 |

|  |
| --- |
| 题目10【2021/8/3】 使用MySQL数据库，如果表格中的数据量非常大，如何处 aging）？如：数据条数是1000万条，分页查询的时候，每页查询10条数据，现在查询第10万页，如何写SQL？  简单来说就是后面的分页参数limit 100，50 limit 10000，50 limit 100000，50  随着第一个参数（即偏移量）的增加，查询的速度呈指数增长趋势。  要解决这个问题，解决方案就是减少偏移量  我们知道按照Id升序排序，我们在查询上一页的时候返回id字段，当查询下一页时我们将上一页最后那条数据的id传递。 |

|  |
| --- |
| 题目11【2021/8/4】 什么是双写一致？实现双写一致的方式有哪些（各种实现方案的特点）？  概念： cache aside pattern  定义：当访问数据库执行写操作的时候，有缓存介入到整体执行流程中时，先写数据库保证成功的前提下，淘汰（删除）缓存中对应的数据。查询数据的时候，有缓存介入到整体执行流程中时，先查询缓存，如果有数据直接返回；如果没有数据，查询数据库，并将数据库查询结果保存到缓存中，再返回。  正式开发项目中，一般写操作分为保存（新增和修改）和删除。保存的时候，是同步缓存数据；删除的时候，是淘汰缓存数据。目的是，提升缓存的应用效率。  边路缓存思想中的淘汰目的： 1、保证缓存的数据绝对正确；2、没有查询，就没有缓存的必要。  边路缓存中，先写数据库，后淘汰缓存的目的：保证缓存中不会有脏数据。如：线程1删除某数据，先淘汰缓存中的数据，CPU到期，终止运行，等待下一个CPU时间片；线程2查询线程1要删除的数据，缓存中没有数据，访问数据库，查询到结果，并缓存；线程1获取CPU时间片，访问数据库，删除数据，执行结束。缓存中有脏数据。  数据库的访问时间一定比缓存访问时间长，可能出现脏数据的情况概率要低很多。  真实开发中，一定是先访问数据库，后访问缓存 ，如果考虑极限情况可能出现脏数据，则在访问缓存的时候，增加分布式锁或者分布式事务。  分布式事务相对来说是比较慢的，但是可以并行处理，  分布式锁是相对来说是比较快的，但是有可能出现串行化  到底用什么，要具体情况具体分析 |

|  |
| --- |
| 题目12【2021/8/5】 解释Spring Boot启动器自动装配原理，如何自定义启动器。 |

|  |
| --- |
| 题目13【2021/8/6】 解释Redis底层数据结构-字典。说明字典的结构，扩容方式。  延申：**字典（hash表），树（二叉树），链表（单向链表），栈（基础栈）。多数据结构的实现方案。** |

|  |
| --- |
| 题目14【2021/8/11】 说明：SpringMVC的核心组件和执行流程。 |

|  |
| --- |
| 题目15【2021/8/12】 SQL优化。就是开发时，编写SQL语句的注意事项，使用索引的注意事项。  回顾 case when then else end 语法。 |