## **常见问题汇总**

1. **Java**
2. **Java SE / Java EE**
3. **Java基础知识**
4. final 关键字的作用。

final关键字可以用来修饰引用、方法和类。

修饰引用：如果引用为基本数据类型，则该引用为常量，该值不可修改。如果引用为引用类型数据，如对象、数组，则对象、数组本身可以修改，但指向该对象或数组的地址不可修改。引用类型的变量必须当场赋值，否则会编译错误。

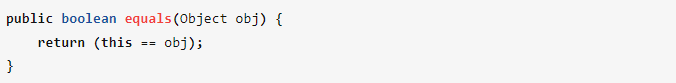
修饰方法：被final修饰的方式将成为最终方法，无法被子类重写，但是可以继承。

修饰类：被final修饰的类无法被继承，其方法会被隐式地定义成final方法。

1. java 中==和 equals 和 hashCode 的区别。

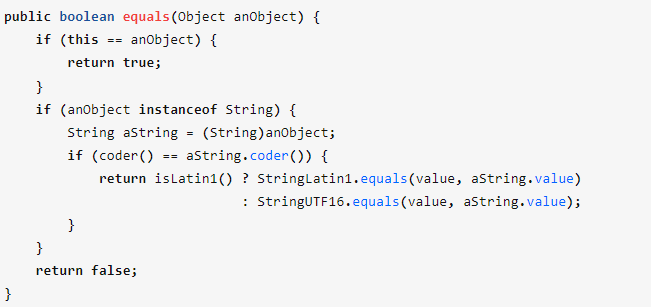
①==是运算符。equals，hashCode是Object类的方法，被所有java继承，允许被覆盖，用于比较引用类型的数据在内存中的地址是否相等。

②==用于比较值是否相等，基本数据类型（byte,short,char,int,long,float,double,boolean）的比较，由于基本数据类型在类加载时是被存储在了方法区，因此用==比较他们的值实际上是比较他们在内存中存储的值是否相等。equals与hashCode则是用于比较引用类型数据在内存中的地址是否相等。equals的实现方法也是用的==，只不过比较的是地址的值是否相等。hashCode则主要是用于集合的操作，其作用是返回一个离散的int型整数，便于提高查询速度，例如在HashMap和HashSet中的使用。



Object类中equals方法的实现

③在很多类中equals方法被重写了，例如String,Integer,Date。此时，equals方法比较的不再是对象在内存中的地址，其结果依据具体的实现不同而不同，比如String类重写equals方法后实际比较的是在内存中的值是否相等，其实现如下图。在重写equals方法时为了保证在equals相同的情况下hashCode值也必定相等这一原则，就必须要要重写hashCode方法，否则将导致出现两个没有关系的对象equals却相同的情况。



String类中重写equals方法

④equals和hashCode之间的关系。

1. 如果两个对象的equals相同，在java运行环境下会认为他们的hashCode也相同。
2. 如果两个对象的equals不相同，他们的hashCode可能相同。
3. 如果两个对象的hashCode相同，他们不一定equals。
4. 如果两个对象的hashCode不相同，他们一定不equals.

基于hashCode性能的优越性和equals的准确性以及它们两者之间的关系，在操作集合时会遍历集合中的所有对象，比较与将要插入对象的hashCode值是否相等，如果不相等，直接插入，如果相等则调用equals()方法比较是否相等，如果相等则不插入，反之则插入。这样即可以保证对集合操作的高效性，也能保证其正确性。

1. int、char、long 各占多少字节数。

byte ：1个字节，-128 ~ 127

short ： 2个字节16位

int ： 4个字节32位

long ： 8个字节64位

float ： 4个字节32位

double ： 8个字节64位

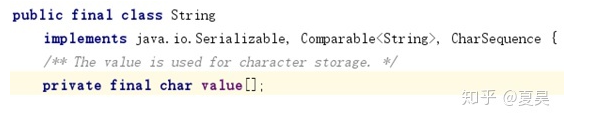
char : 2个字节16位

1. int 与 Integer 的区别。

①int是基本数据类型，Integer是引用类型，是int的封装类。②Integer变量必须要先实例化才能使用，而int不需要。③Integer实际是对象的引用，当new一个Integer对象时，实际是生成了一个引用，指向内存中的堆空间，对应其值，而int的值则直接被存储到了堆空间。④Integer的默认值是null，而int的默认值是0。

1. String、StringBuffer、StringBuilder 区别。

①String是java中一个特殊的类，也是我们平时开发中用得最多的类之一。它是一个不可变的字符串，底层的实现是一个用final修饰的字符串数组。



String类的底层实现

String对象一经赋值就会缓存在字符串常量池中，字符串常量池是堆空间中存在的一个特殊的空间，在创建String对象时，会优先判断字符串常量池中是否存在缓存对象，如果存在则返回其引用给这个创建的对象。对String对象进行操作时，如”+”操作，会优先在字符串常量池中创建要追加的部分，再将拼接后的对象缓存在字符串常量池中，并将String对象的引用指向拼接后的内存地址。但是如果是使用new的方式来创建String对象，那么String对象的引用不会指向拼接后的内存地址，会在堆空间中（非字符串常量池）存储拼接对后的值，并将String对象的引用指向该内存。无论是哪种方式，其效率都是比较低的，对内存空间的浪费也比较大。

②StringBuffer、StringBuilder的存在就是为了解决这些问题。StringBuffer与StringBuilder一样，都继承于AbstractStringBuilder，底层的实现没有采用用final修饰的字符串数组，在对字符串进行拼接时，会在原有的内存上进行拼接，不会浪费空间，效率也更高。StringBuffer同StringBuilder的区别是：StringBuiler是线程不安全的，其执行效率优于StringBuffer，安全性低于StringBuffer；StringBubber是线程安全的，其执行效率低于StringBuilder，安全性高于StringBuilder。

③因此，在字符串操作较多的时候应优先使用StringBuffer和StringBuilder，要求线程安全时优先使用StringBuffer，不要求线程安全时使用StringBuilder。

1. 什么是内部类？内部类的作用。

定义：将一个类定义在一个类或者一个方法里，这样的类就称之为内部类。

分类：成员内部类、静态内部类、匿名内部类、局部内部类。

①成员内部类：最普通的一种内部类，可以调用外部类的所有属性和方法，外部类要调用内部类的方法则必须先实例化内部类。成员内部类不能包含静态属性和方法。

②静态内部类：与成员内部类的区别是给内部类多加了一个static关键字，静态内部类只能访问外部类的静态成员变量和静态方法。

③匿名内部类：匿名内部类是没有名称的内部类。当一个内部类需要继承一个类或者实现一个接口（必要条件），并且只使用一次的时候，可以考虑使用这种方式。调用的时候直接使用父类的无参构造，并重写父类的方法，从而达到优化代码的目的。

④局部内部类：定义在代码块里的类，比如定义在一个方法中。起作用范围仅限于它所在的代码块中，局部内部类不能被public、protected、private、static修饰，但是可以被final修饰。

作用：①内部类可以很好地实现隐藏，非内部类是不可以被private、protected修饰的，但是内部类可以，从而达到了隐藏的作用。同时可以将一定逻辑关系的类组织在一起，增强可读性。②间接实现多继承。每个内部类都可以独立地继承类，实现接口，外部类继承类跟实现接口的情况与内部类无关。

1. 泛型中 extends 和 super 的区别？

<? extends T>表示包括T在内的任何T的子类。

<? supper T> 表示包括T在内的任何T的父类。

1. final，finally，finalize 的区别。

final：final是java的一个关键字，可以用来修饰变量、方法以及类。被final修饰的变量在使用中不可改变，必须在声明的时候给定初始值，如果为引用类型的变量，则其引用的地址不会发生改变，其属性是可以发生改变的。被final修饰的方法，可以被继承，但是不可以被重写。被final修饰的类无法被继承，其方法也会隐式地定义为final。

finally：在处理异常时提供finally块来执行任何清除操作。通常在处理一些资源的时候，会使用finally来释放资源。一般情况下finally代码块一定会被执行，除了以下几种情况：①在finally代码块中语句存在异常。②在前面的代码中使用了System.exit()退出程序。③程序所在的线程死亡。④关闭CPU。

finalize：finalize()是一个方法，在Object类中定义了，因此所有类都继承了这个方法。在垃圾收集器将对象从内存中清除出去之前做必要的清理工作时会调用这个方法以确定这个对象有没有被引用，任何对象在这个方法中都存在复活的可能，但这个方法只会被调用一次，一般情况下不建议重写这个方法。

1. String 转换成 Integer 的方式及原理。

内部默认是使用的十进制，可以通过parseInt(s, radix)方法中的radix参数来自定义转换进制；正常判断Null，进制范围，length等；判断第一位是否为符号位；循环遍历每个字符的十进制值；通过\*=和-=进行计算拼接，判断是否为负值并返回结果。

1. **面向对象编程**
2. 抽象类和普通类的区别，接口和抽象类的区别。
3. 父类的静态方法能否被子类重写？静态属性和静态方法是否可以被继承？
4. **AOP**
5. **多线程**

线程池有哪些类型？

concurrentHashMap分段锁原理，java8和java7实现的区别。

Java锁有了解吗？Synchronized和ReentrantLock区别？说说如何ReentrantLock如何实现超时锁的等待？

进程和线程的区别。

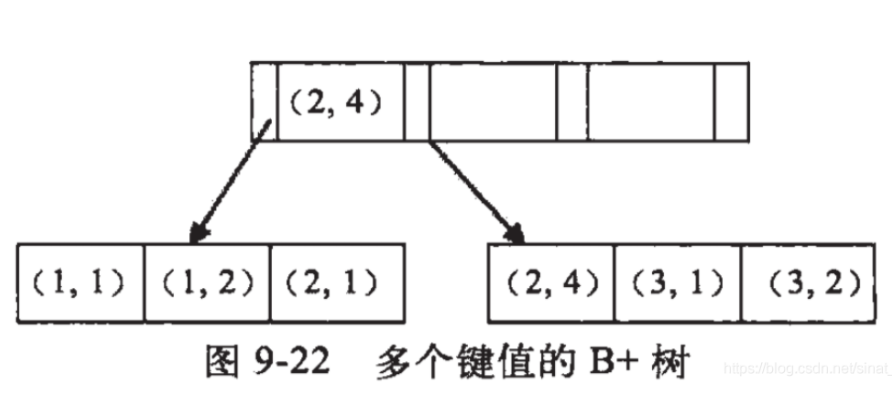
1. **IO**
2. **Java Web**
3. **Web编程原理**
4. **Servlet**
5. **MVC模型**
6. **Web**
7. **Web 基础**
8. **脚本语言**
9. **HTML**
10. **JavaScript**
11. **jQuery**
12. **主流框架**
13. **Vue**
14. **Angular**
15. **数据库**
16. **关系型数据库**
17. **MySQL**
18. MySQL索引的原理是什么。

B+树，本质上它是将瘦长的平衡二叉树，改造成一个矮扁的平衡多叉树从而减少IO查询次数，平均查询复杂度O(logN),另外与B树不同的是，除了叶子结点外，其他结点均不存储数据，另外叶子结点之间，用[链表](https://www.nowcoder.com/jump/super-jump/word?word=%E9%93%BE%E8%A1%A8" \t "https://www.nowcoder.com/discuss/_blank)方式相连以加速查询。当然视面试官的反应，可以再补充B+树的插入和删除操作。【牛客】

1. MySQL建立索引时，应该注意什么？

①对区分度高的列作索引，对于那种只有两三个值的字段做索引没有太多意义。②建立联合索引时要满足最左匹配原理。只对需要作为查询条件的列做索引，过度索引会拖慢插入速度。【牛客】

最左匹配原理：是针对联合索引的。索引的底层是一个B+数，联合索引自然也是一个B+数，只不过联合索引的键值数量是多个。由于构建一颗B+数只能根据一个值来构建，因此数据库依据联合索引的最左字段来构建B+树。假如我们创建一个联合索引(a,b)，如下图：



从上图我们可以发现，a的值是有序的1,1,2,2,3,3，而b的值是无序的，所以当查询条件为a = 1时，索引是生效的，而查询条件仅为b = 2时索引是无效的，因为a的有序性和b的无序性决定了这一特点。但当a的值为定值时，我们发现，b的值是有序的，如上图，当a = 1时，b是有序的1,2；当a = 2时，b为1,4。因此当查询条件为a = 1 and b = 2时，索引是生效的，由于feature优化器的存在，当查询条件为b = 2 and a = 1时，索引同样生效，因为优化器会自动优化条件顺序。但是例如查询条件为a > 1 and b = 2时，a的字段可以匹配上索引，但是b不可以，因为当a是一个范围是，b是无序的。

假设建立联合索引(a,b,c):

1、全值匹配查询时

①select \* from table\_name where a = ‘1’ and b = ‘2’ and c = ‘3’

②select \* from table\_name where b = ‘2’ and a = ‘1’ and c = ‘3’

③select \* from table\_name where c = ‘3’ and a =‘1’ and b = ‘2’

因为优化器的存在，索引均生效

2、匹配左边的列时

①select \* from table\_name where a = ‘1’

②select \* from table\_name where a = ‘1’ and b = ‘2’

③select \* from table\_name where c = a =‘1’ and b = ‘2’ and ‘3’

符合最左原理，索引均生效

④select \* from table\_name where a = ‘1’ and c = ‘3’

不连续，a的索引生效，c的索引不生效。

3、 匹配列前缀

①select \* from table\_name where a like ‘As%’ //前缀都是排序好的，走索引

②select \* from table\_name where a like ‘%As’ //全表查询

③select \* from table\_name where a like ‘As%’ //全表查询

4、匹配范围值

①select \* from table\_name where a > 1 and a < 3 //索引生效

②select \* from table\_name where a > 1 and a < 3 and b > 3 //a索引生效，b索引无效。

③select \* from table\_name where a =1 and b > 3 //a,b索引均生效。

5、排序

①select \* from table\_name order by a,b,c //索引生效

②select \* from table\_name order by b,a,c //调到顺序，违反最左匹配原理，索引不生效。

③select \* from table\_name where a = 1 order by b,c //索引生效

1. Mysql用过对吧？说说Mysql索引建立策略？假如我给你三个字段, 性别, 年龄和身份证号, 哪个字段更适合做索引？
2. **Oracle**
3. **非关系型数据库**
4. **Redis**
5. Redis为什么这么快？

①单行程。②所有操作都在内存中完成。③采用足够简单的数据结构。【牛客】

1. [redis](https://www.nowcoder.com/jump/super-jump/word?word=redis" \t "https://www.nowcoder.com/discuss/_blank)设置过期时间的原理是怎样的？

①惰性删除：只有在get的时候去检查它是否expire，如果过期，就删除。②定时删除：每一秒重复10次，随机选择20个key，删除所有过期的key，如果25%的key都过期了，就回到步骤一再做一次。【牛客】

1. **Neo4j**
2. **数据库优化**
3. MySQL的优化。
4. **数据结构与算法**
5. **数据结构基础**
6. B+树相比于哈希索引有什么优点？
7. hashMap原理,put和resize过程。
8. B-树和B+树区别，数据库索引原理，组合索引怎么使用？阐述最左匹配的原理。
9. hashMap原理。
10. **数据操作**
11. **常见算法实现**
12. **框架**
13. **Spring**
14. Spring生命周期，几种scope区别，aop实现有哪几种实现，接口代理和类代理会有什么区别？
15. BeanFactory 和 ApplicationContext 有什么区别？
16. Spring Bean 的生命周期。
17. Spring IOC 如何实现。
18. Spring AOP 实现原理。
19. Spring MVC 运行流程。
20. Spring 事务底层原理。
21. Spring 的单例实现原理。
22. **Spring MVC + MyBatis / Hibernate**
23. Mybatis 的 ${} #{}的区别是什么？
24. Spring MVC中，@RequestMapping的实现原理？
25. Spring MVC 启动流程。
26. **Spring Boot**
27. springboot的启动原理（说了main函数，类加载器，主类的注解等）。
28. **Spring Cloud**
29. **微服务**
30. **设计模式**
31. **常见设计模式**
32. **常见设计模式代码实现**
33. **JVM**
34. **JVM原理**
35. G1 和 CMS 的区别。
36. 类加载过程（类的生命周期），解释各个阶段的意义。
37. 你们垃圾回收器用的什么？ G1有哪些特点？G1如何实现可预测的停顿时间？漏标问题如何解决的？介绍下三色标记？说说STAB[算法](https://www.nowcoder.com/jump/super-jump/word?word=%E7%AE%97%E6%B3%95" \t "https://www.nowcoder.com/discuss/_blank)？
38. 哪些情况下的对象会被垃圾回收机制处理掉？
39. 对象不可达，一定会被垃圾收集器回收么？
40. Java中实现多态的机制是什么？
41. **JVM优化**
42. **计算机网络**
43. **通讯原理及主流通讯协议**
44. **通讯代码实现**
45. **Socket**
46. **WebSocket**
47. **MQ**
48. **操作系统**
49. **Windows**
50. **Linux**
51. **项目经验**