基础面试题

笔记本:	My Notebook		
创建时间:	2023/9/8 20:58	**更新时间:**	2023/9/12 11:06
作者:	pe8jmft6		
URL:	https://c.binjie.fun/#/chat/1691719279859		

1.equals和==的区别?

==
如果是基本数据类型，==表示判断它们的值是否相等
如果是引用数据类型，==表示判断两个对象指向的内存地址是否相同
equals用于比较对象的内容是否相等，可以通过重写来自定义比较规则

2.final, finally, finalize的区别

final： 用于修饰属性,方法和类, 分别表示属性不能被重新赋值, 方法不可被覆盖, 类 不可被继承.
finally： 异常处理语句结构的一部分，一般以 try-catch-finally 出现，finally 代码块表示总被执行.
finalize： Object 类的一个方法，该方法一般由垃圾回收器来调用，当我们调用 System.gc() 方法的时候，由垃圾回收器调用 finalize()方法，回收垃圾，JVM 并不保证此方法总被调用

3.重载和重写的区别

Overload 重载 过载 发生在同一个类当中 两个方法名字相同 但是参数列表不同
Override 覆盖 重写 发生在有继承关系的两个类的子类中 子类对从父类继承来的方法不满意 从而进行了重新实现

4. 两个对象 hashCode()相同，则 equals()是否也一定为 true?

两个对象 equals 相等，则它们的 hashcode 必须相等，如果两个对象的hashCode()相同，则 equals()不一定为 true

5. 接口和抽象类的区别

接口当中定义的变量默认就是常量：public static final 抽象类当中定义的变量是每个这个类型的对象都有一份的属性 接口当中定义的方法默认就是公共的抽象方法：public abstract 抽象类当中定义的方法可以是普通方法 也可以定义抽象方法

6.BIO、NIO、AIO 有什么区别?

BIO：线程发起 IO 请求，不管内核是否准备好 IO 操作，从发起请求起，线程一直阻塞，直到操作完成。
NIO：线程发起 IO 请求，立即返回；内核在做好 IO 操作准备之后，通过调 用注册✂回调函数通知线程做 IO 操作，线程开始阻塞，直到操作完成。
AIO：线程发起 IO 请求，立即返回；内存做好 IO 操作准备之后，做 IO 操 作，直到操作完成或者失败，通过调用注册回调函数通知线程做 IO 操作完 成或者失败。

7.String，StringBuffer，StringBuild 的区别?

第一个没有缓冲空间 后面两个都有 第二个底层大量的 synchronized 修饰 第三个没有

StringBuffer和StringBuilder之间的区别?

- StringBuffer同一时间允许一个线程进行访问 效率较低 但是不会出现并发错误
- StringBuilder同一时间允许多个线程进行访问 效率较高 但是可能会出现并发错误

8.Comparator 与 Comparabl有什么区别?

Comparable 接口是在自身类内部实现，用于定义对象的自然排序规则；Comparator 接口是一个独立的比较器接口，用于定义对象的定制排序规则。Comparable 适用于对单一类的对象进行

排序，Comparator 适用于对多个类的对象进行排序。在使用时，需要根据具体需求和场景选择合适的接口来实现对象的比较和排序

9. JAVA 中的几种基本数据类型

int long short byte char float double boolean

10.String类能被继承吗，为什么?

String是一个final修饰的类，final修饰的类不可以被继承
不能被继承的原因：
1.效率性，String类作为最常用的类之一，禁止被继承和重写，可以提高效率
2.安全性，String类中有很多调用底层的本地方法，调用了操作系统的API，如果方法可以重写，可能被植入恶意代码，破坏程序

11. 说说 Java 中多态的实现原理

多态机制包括静态多态（编译时多态）和动态多态（运行时多态）
静态多态比如说重载，动态多态一般指在运行时才能确定调用哪个方法。
我们通常所说的多态一般指运行时多态，也就是编译时不确定究竟调用哪个具体方法，一直等到运行时才能确定。
多态实现方式：子类继承父类（extends）和类实现接口（implements）
多态核心之处就在于对父类方法的改写或对接口方法的实现，以取得在运行时不同的执行效果。
Java 里对象方法的调用，依靠类信息里的方法表实现，对象方法引用调用和接口方法引用调用的大致思想一样。当调用对象的某个方法时，JVM 查找该对象类的方法表以确定该方法一直引用地址，有了地址后才真正调用该方法。

12. int 和 Integer 有什么区别

int 是 Java 的基本数据类型，而 Integer 是 int 的包装类，属于引用类型， int 默认值为 0 ，而 interger 默认值为 null， Interger 使用需要判空处理

13. 说说反射的用途及实现原理，Java 获取反射的三种方法

反射是一种在运行时动态获取类的信息、调用类的方法和操作类的属性的能力。Java 的反射机制提供了一个强大的 API，可以在运行时检查和修改类、接口、字段和方法等信息，使得程序能够在运行时动态地加载类、创建对象、调用方法和访问属性
反射的主要用途如下：

1. 动态加载类：使用反射可以在运行时动态地加载需要使用的类，而不是在编译时固定地依赖于某个具体的类。
2. 创建对象：通过反射可以实例化一个类的对象，即使在编译时并不知道该类的具体名称，从而实现对象的动态创建。
3. 调用方法和访问属性：通过反射可以在运行时动态地调用类的方法和访问类的属性，甚至是私有的方法和属性。
4. 解析注解：反射可以用于解析注解，在运行时获取类、方法、属性上的注解信息，并根据注解的内容进行相应的处理。
5. 动态代理：反射可以实现动态代理，即在运行时创建一个实现了特定接口的代理对象，用于拦截并处理方法调用。

实现原理： Java 的反射机制是基于 Java 的反射 API 实现的，它主要依赖于以下两个类：

1. Class 类：Class 类是反射的核心类之一，它表示运行时类的信息。Java 在加载类时会将类的信息保存在内存中，并在运行时提供对类的信息的访问。

2. java.lang.reflect 包：这个包中提供了一系列的接口和类，用于操作类的信息。主要包括 Field、Method、Constructor 等类，它们提供了对类的字段、方法、构造函数等信息的访问和操作。

具体实现原理如下：

1. 获取 Class 对象：首先需要获取要操作的类的 Class 对象。可以通过类名.class、对象.getClass() 或 Class.forName() 等方式获取。

2. 获取类的信息：通过 Class 对象可以获取类的各种信息，如类的名称、父类、接口、字段、方法等。

3. 创建对象：通过 Class 对象的 newInstance() 方法可以创建类的实例对象。

4. 调用方法和访问属性：通过 Method 类的 invoke() 方法可以调用类的方法，通过 Field 类可以对类的属性进行操作。

5. 其他操作：还可以通过 Constructor 类来创建对象的实例，通过 Annotation 类获取注解信息等。

Java 获取反射的三种方法：
第一种，使用 Class.forName 静态方法。
第二种，使用类.class 方法
第三种，使用实例对象 getClass() 方法

14.面向对象的三大特征

封装
继承
多态

1. 封装（Encapsulation）：封装是将数据和操作数据的方法封装在一起，形成一个类（class）或对象（object）。通过封装，可以隐藏实现的细节，只暴露必要的接口，提高代码的安全性和可维护性。封装还可以实现信息隐藏，保护数据，让外部无法直接访问和修改对象的内部状态，只能通过定义的方法进行操作。

2. 继承（Inheritance）：继承是指一个类（子类/派生类）可以继承另一个类（父类/基类）的属性和方法。子类继承了父类的特性，并可以扩展或修改父类的行为。继承可以实现代码的重用，减少重复编写相同的代码，提高代码的可维护性和扩展性。通过继承还可以建立类之间的层次关系，形成多态的基础。

3. 多态（Polymorphism）：多态是指同一种操作或接口可以以多种形式实现或被不同类型的对象调用。多态可以通过继承和接口实现。多态提供了更灵活和通用的编程方式，允许在不确定对象类型的情况下，以统一的方式调用方法。通过多态，可以实现方法的重写（Override）和方法的重载（Overload），增强代码的可扩展性和灵活性。

15.&和&&的区别

按位与, a&b 表示a 和 b 都转换成二进制数，再进行与运算；

&和&&都是逻辑运算符号，&&又叫短路运算符 逻辑与，a&& b ，a&b 都表示当且仅当两个操作数均为 true 时，其结果才 为true，否则为 false。 逻辑与跟短路✂差别✂非常巨大✂，虽然二者都要求运算符左右两端✂布尔值都 ✂ true，整个表达式✂值才✂ true。但✂，&&之所以称为短路运算，✂因为如 果&&左边✂表达式✂值✂ false，右边✂表达式会被直❿短路掉，不会进行运算

在Java中，&（逻辑与运算符）和&&（短路逻辑与运算符）是用于布尔类型的逻辑运算符，它们有以下区别：

1. 执行方式：
   - &：无论左边的表达式结果是true还是false，都会执行右边的表达式。
   - &&：如果左边的表达式结果为false，则不会执行右边的表达式，因此存在短路的情况。
2. 短路特性：
   - &：不具有短路的特性，无论左边的表达式结果是true还是false，都会对右边的表达式进行判断和求值。
   - &&：具有短路的特性，如果左边的表达式结果为false，则不会对右边的表达式进行判断和求值。
3. 使用场景：
   - &：常用于需要对两个表达式都进行求值的情况，无论左右表达式的结果如何。
   - &&：常用于提高代码的效率，在某些条件下可以避免对后续表达式的求值，例如在条件判断中。

总结：& 运算符始终会对左右两边的表达式进行求值，而 && 运算符在左边的表达式结果为 false 的情况下会短路，不再对右边的表达式进行求值

16.Java 中IO 流分为几种?

1.字节流

2.字符流

主要区别如下：

1. 数据单位：字符流以字符为单位进行读写，字节流以字节为单位进行读写。
2. 处理对象：字符流适用于处理文本数据，如文本文件、字符串等；字节流适用于处理二进制数据，如图像、音频、视频等非文本数据。
3. 编码处理：字符流会自动处理字符编码，可以进行字符集转换，而字节流处理原始的字节表示，不涉及字符编码的转换。
4. 功能特点：字符流提供了高级别的字符处理功能，如按行读取、字符缓冲等；字节流较为底层，提供了基本的读写操作。

17.类实例化顺序，比如父类静态数据，构造函数， 子类静 态数据，构造函数。

类实例化顺序为： 父类静态代码块/静态域->子类静态代码块/静态域 - > 父类非静态代码块 -> 父类构造器 -> 子类非静态代码块->子类构造器

18. Java 创建对象有几种方式

Java 创建对象有 5 种方式

1.用 new 语句创建对象。

2. 使用反射，使用 Class.newInstance()创建对象/调用类对象的构造方法—— Constructor

3.调用对象的 clone()方法。

4. 运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject()方法.

5.使用 Unsafe

19. 守护线程是什么？用什么方法实现守护线程

守护线程，它是一种专门为用户线程提供服务的线程，它的生命周期依赖于用户线程。 只有JVM中仍然还存在用户线程正在运行的情况下，守护线程才会有存在的意义。 否则，一旦JVM进程结束，那守护线程也会随之结束。 也就是说，守护线程不会阻止JVM的退出。但是用户线程会！ 守护线程和用户线程的创建方式是完全相同的，我们只需要调用用户线程里面的setDaemon方法并 且设置成true，就表示这个线程是守护线程。 因为守护线程拥有自己结束自己生命的特性，所以它适合用在一些后台的通用服务场景里面。 比如JVM里面的垃圾回收线程，就是典型的使用场景。 这个场景的特殊之处在于，当JVM进程技术的时候，内存回收线程存在的意义也就不存在了。 所以不能因为正在进行垃圾回收导致JVM进程无法技术的问题。 但是守护线程不能用在线程池或者一些IO任务的场景里面，因为一旦JVM退出之后，守护线程也会 直接退出。 就会可能导致任务没有执行完或者资源没有正确释放的问题。

20.`notify()` 和 `notifyAll()` 的区别

- `notify()`：唤醒一个正在等待该对象锁的线程，如果有多个线程等待，由系统决定唤醒哪一个线程。
- `notifyAll()`：唤醒所有正在等待该对象锁的线程。

被唤醒不等于就能执行了，需要得到锁对象才能有权利继续执行，而锁只有一个， 所以多个线程被唤醒时需要争取该锁。

21. Java 异常层次结构

异常的体系结构
Throwable
Error Exception
RuntimeException
错误和异常的区别： 错误通常是指由于硬件环境或者系统原因导致的程序员通过代码无法解决的问题 相对较严重，异常是指程序运行过程当中出现的例外情况而已。
运行时异常和非运行时异常的区别：
非运行时异常在编译的时候就需要进行异常处理，否则编译都无法通过，它们都直接继承 Exception
运行时异常 编译的时候不需要给出解决方案 运行时异常直接体现 它们继承RuntimeException
常见的运行时异常
运算符：
ArithmeticExcpetion 算术异常
数组：
NegativeArraySizeException 负数数组大小异常
ArrayIndexOutOfBoundsException 数组索引值超出边界异常
字符串：
NullPointerException 空指针异常
StringIndexOutOfBoundsException 字符串索引值超出边界异常
NumberFormatException 数字格式化异常

类型造型：
ClassCastException 类型造型异常
集合：
IllegalArgumentException 非法参数异常
IndexOutOfBoundsException 索引值超出边界异常
IllegalStateException 非法状态异常
ConcurrentModificationException 并发修改异常

22.throw 和 throws 的区别

throw 用在方法体当中 在没有异常的情况下 主动制造异常出现的场景 throws 用在方法签名的最后 表达本方法当中出现指定种类的异常 方法不做处理 抛还给调用的上级进行处理

23. 静态内部类与非静态内部类有什么区别

静态内部类可以有静态成员(方法，属性)，而非静态内部类则不能有静态成员(方法，属性)。
静态内部类只能够访问外部类的静态成员和静态方法,而非静态内部类则可以访问外部类的所有成员(方法，属性)。
实例化静态内部类与非静态内部类的方式不同
调用内部静态类的方法或静态变量,可以通过类名直接调用

24.string类new和不new之间的区别?

- 不new的方式会涉及到常量池查找机制 优先去常量池查看 常量池里面如果有的话 直接拿出来使用 如果常量池里面没有 缓存一份
- new的方式会在堆内存里面开辟空间 与此同时在常量池里面查看 有的话 不再缓存 如果没有的话 会在常量池里面缓存一份

25.反射中，Class.forName 和 ClassLoader的区别

- `Class.forName` 是一个静态方法，通过类名加载类并返回 `Class` 对象，会执行静态代码块。
- `ClassLoader` 是一个抽象类，提供动态加载类的机制，可以自定义类加载器，并按照层次结构进行加载。
- `Class.forName` 会使用默认的类加载器加载类，如果未指定类加载器，则使用当前线程的上下文类加载器。而 `ClassLoader` 可以显式指定类加载器。
- `Class.forName` 在加载类时会触发静态代码块的执行，而 `ClassLoader` 只负责加载字节码文件，不会执行静态代码块。

26.深拷贝和浅拷贝的区别

深拷贝和浅拷贝是用来描述对象或者对象数组这种引用数据类型的复制场景的。
浅拷贝，就是只复制某个对象的指针，而不复制对象本身。
这种复制方式意味着两个引用指针指向被复制对象的同一块内存地址.
深拷贝，会完全创建一个一模一样的新对象，新对象和老对象不共享内存，也就意味着对新对象的修改不会影响老对象的值。

27.String 类常用方法都有那些呢?

length（）得到字符串里面的字符个数

getBytes（）将一个字符串转换成字节数组

toCharArray（）将一个字符串转换成字符数组

split（）将一个字符串按照指定的内容劈开

equals（）区分大小写的判断两个字符串的内容是否一模一样

equalsIgnoreCase（）忽略大小写的比较字符串的内容是否一样

contains（）判断一个字符串里面是否出现了某个内容

startsWith（）判断一个字符串是否以指定的内容开头

endsWith（）判断一个字符串是否以指定的内容结尾

toUpperCase（）将一个字符串全部转换成大写

toLowerCase（）将一个字符串全部转换成小写

replace（）将字符串里面的某个内容全部替换成指定内容

replaceAll（）将字符串里面的某个内容全部替换成指定内容，支持正则表达式

replaceFirst（）将字符串里面第一次出现的某个内容替换成指定的内容

trim（）去掉字符串的前后空格

substring（int x，int y）从下标x一直截取到下标y-1对应的元素

substring（int x）从下标x一直截取到字符串的最后

charAt（）找到指定下标对应的元素

indexOf（）找到某个内容第一次出现的下标

lastIndexOf（）找到某个内容最后一次出现的下标

28.spring中的设计模式

工厂模式

代理模式

包装模式

策略模式

观察者模式

命令模式

29.什么是值传递和引用传递

- 值传递是指复制实际参数的值，传递给方法或赋值给变量。对基本数据类型进行值传递，方法内部修改不会影响原始值。
- 引用传递是指复制实际参数的地址，传递给方法或赋值给变量。对引用数据类型进行引用传递，方法内部修改会影响到原始对象。

30. object 中定义了哪些方法?

getClass(); 获取类结构信息

hashCode() 获取哈希码

equals(Object) 默认比较对象的地址值是否相等，子类可以重写比较规则

clone() 用于对象克隆

toString() 将对象转变成字符串

notify() 多线程中唤醒功能

notifyAll() 多线程中唤醒所有等待线程✂功能

wait() 让持有对象锁的线程进入等待

wait(long timeout) 让持有对象锁的线程进入等待，设置超时毫秒数时间

wait(long timeout, int nanos) 让持有对象锁的线程进入等待，设置超时纳秒数时间

finalize() 垃圾回收前执行的方法

31.for-each 与常规 for 循环的效率对比

- 在遍历数组或集合时，for-each 循环通常比常规 for 循环更简洁和易读。
- 在性能上，常规 for 循环可能会比 for-each 循环稍微高效一些。这是因为 for-each 循环需要通过迭代器来获取每个元素，而常规 for 循环直接通过索引访问元素，少了迭代器的开销。
- Java 虚拟机（JVM）对 for-each 循环进行了一些优化，尽可能减少了迭代器的性能损失。在大多数情况下，两种循环的性能差异并不明显，而可读性更重要。

32.运行时异常。

运算符：
　　ArithmeticExcpetion　　算术异常
数组：
　　NegativeArraySizeException　负数数组大小异常
　　ArrayIndexOutOfBoundsException　数组索引值超出边界异常
字符串：
　　NullPointerException　　空指针异常
　　StringIndexOutOfBoundsException　字符串索引值超出边界异常
　　NumberFormatException　　数字格式化异常
　类型造型：
　　ClassCastException　　类型造型异常
　集合：
　　IllegalArgumentException　非法参数异常
　　IndexOutOfBoundsException　索引值超出边界异常
　　IllegalStateException　　非法状态异常
　　ConcurrentModificationException　并发修改异常

33.final 关键字修饰类，方法，变量的作用

final 修饰的类 最终类 不能被继承

final 修饰的方法 最终方法 不能被覆盖

final 修饰的变量 一旦赋值就再也不能修改

34.堆和栈的认识，以及使用

1. 堆（Heap）：

    - 堆是动态分配的内存区域，用于存储动态创建的对象和数据。
    - 在堆中分配的内存需要手动进行释放，否则会导致内存泄漏。
    - 堆的大小一般较大，可以容纳更多的数据。
    - 堆的分配和释放由程序员负责管理，需要手动调用相应的内存分配和释放函数（如`new`和`delete`）。
    - 堆中的对象可以被多个线程共享，并且可以在任何地方被访问。
    - 堆适合存储动态分配的数据，如对象实例、大数组等。

2. 栈（Stack）：

    - 栈是静态分配的内存区域，用于处理函数调用、局部变量和程序执行的上下文。
    - 栈的大小通常较小，有限制（如几百KB或几MB）。
    - 栈的分配和释放是自动完成的，由编译器和程序运行时管理。
    - 栈中的数据遵循先进后出（LIFO）的原则，每当调用一个函数时，该函数的局部变量和参数将被压入栈中，在函数返回时又会从栈中弹出。

- 栈上的数据只能在其所属函数的范围内访问，函数执行结束后，栈上的数据将被自动释放。
- 栈适合存储局部变量、函数调用和程序执行的上下文等。

35.this 和super 关键字✂作用
this： 对象内部指代自身的引用， 解决成员变量和局部变量同名问题 ，可以调用成员变量，不能调用局部变量，可以调用成员方法， 在普通方法中可以省略 this， 在静态方法当中不允许出现 this 关键字
super：调用父类成员或者方法
调用父类的构造方法
36. synchronized实现原理
`synchronized` 是一个关键字，用于实现线程同步和访问共享资源的互斥性。在多线程编程中，当多个线程同时访问临界区（共享资源）时，可能会导致数据不一致或竞态条件等问题。使用 `synchronized` 可以确保同一时间只有一个线程可以进入同步块或方法，从而保证线程安全性