**JSD2310一阶段模拟面试题库**

# 1、说说JDK、JRE、JVM的区别:

**JDK**（Java Development Kit）、**JRE**（Java Runtime Environment）、**JVM**（Java Virtual Machine）是Java开发和运行环境的三个重要组织部分。

* **JDK：java开发工具包。**其中包含JRE、解释器/加载器、编译器(javac)、归档器(jar)，文档生成器(javadoc)以及其他开发工具。编译器可以将**.**java文件编译成字节码(**.** Class)文件。
* **JRE：Java运行时环境。**主要包含JVM和Java的核心类库(Java API)。JRE不包含开发工具
* **JVM：java 虚拟机。**执行class文件。它能够将class文件中的**字节码指令**进行识别并调用操作系统上的 API 完成动作（调汇编指令，让cpu按照汇编指令去干活）JVM是跨平台特性的基础，因为不同的平台有不同的JVM，无论哪个平台class文件都是一样的。JVM在jre目录里的bin目录中。

**大小顺序：**JDK包含—>JRE包含—>JVM(bin里)

# 2、java的8种基本数据类型是什么?(简述java的8种基本数据类型)

**基本数据类型：8种**包括: **byte，short，int，long，float，double，boolean，char**

* + - **byte：**字节型，用于存储整数，占1个字节，范围-128到127
    - **short：**短整型，用于存储整数，占2个字节，范围-32768到32767。
    - **int：**最常用整型，用于存储整数，占4个字节，范围-231到231-1。
    - **long：**长整型，用于存储整数，占8个字节，范围-263到263-1。
    - **float：**单精度浮点，用于存储小数，占4个字节，不能表示精确值。
    - **double：**双精度浮点，最常用存储小数，占8个字节，不能表示精确值。
    - **boolean：**布尔型，用于存储true或false，占用1个字节。
    - **char：**字符型，采用Unicode字符编码格式，用于存储单个字符，占用2个字节

**作用：基本数据类型在Java中具有固定的大小和范围，并且它们是不可变的，它们可用于声明变量，方法参数和方法的返回值类型。**

**引用数据类型：类、接口、数组、枚举等。**

# 3、switch可以应用于哪些数据类型上?

在Java中，switch语句是一种条件控制语句，用于根据不同的条件执行不同的代码块。switch语句使用一个或多个case标签来匹配某个表达式的值，并根据匹配结果执行相应的代码。

**switch语句的执行流程如下：**

1、表达式会被计算出一个值。

2、switch语句会逐个比较case标签与表达式的值，如果匹配成功，则执行对应的代码块。

3、如果没有匹配的case标签，则执行default代码块 (可选)。

4、执行完匹配的代码块后，会通过break语句跳出switch语句，继续执行后续的代码

**需要注意的是:**

* 表达式的类型可以是byte、short、char、int、枚举类型或String类型
* case标签的值不能重复，且必须是常量表达式(即不能是变量)
* 每个case标签后面需要使用break语句来结束该分支的执行，否则会继续执行下一个case分支
* 可以使用default关键字来定义默认的代码块，当没有匹配的case标签时将执行该代码块。

# 4、什么是数组？有哪些特性？

**概念：**数组是一种用于存储相同类型元素的容器，它是一个固定长度的对象。

**特性：**

* 数组的长度是固定的，一旦被创建，大小就不能修改。
* 数组里存放的数据类型必须一致。
* 数组本身是引用数据类型，但数组中的元素可以是任何数据类型，包括基本类型和引用类型。
* 创建数组对象时会在内存中开辟一整块连续的内存空间，数组名代表的是数组在内存中的首地址。

# 5、面向对象编程语言的特性，各有什么优劣势

**基本特征：封装，继承，多态。**

**1）封装**

**定义：**将数据（属性）和操作数据的方法（行为）封装在一个类中。

**特点：**通过接口和访问修饰符来控制和访问类的内部成员，隐藏内部实现的细节。

**优势：**

**数据隐藏：**封装允许将数据隐藏在类的内部。通过定义私有（private）属性可以避免直接访问和修改对象的数据，提高数据的安全性和可靠性。

**隔离复杂性：**封装将类的实现细节隐藏起来，只需暴露必要的接口给外部使用者，减少对外部的依赖，降低了类与类之间复杂性。

**提供公共接口：**通过封装，可以定义公共的接口（public方法），对外提供统一的访问和操作方式，提供更好的交互方法，便于使用和维护。

**劣势：**对类内部成员的可见性需要仔细控制，使用不当可能导致对属性的过度保护或过度开放。

**2）继承**

**定义：**继承是指一个类型（子类/派生类）继承另一个类的类型（父类/基类/超类）。

**特点：**子类通过关键字extends来继承父类，子类可以直接使用父类的非私有的属性和方法。子类也可以覆盖（重写）父类的方法，从而实现自己的行为。

**优势**

**代码复用：**继承允许子类直接使用父类的属性和方法，避免了重复编写相同的代码，提高 了代码复用性。

**扩展性：**子类可以在继承父类的基础上进行修改和扩展，添加新属性和方法，满足不同的需求。

**逻辑清晰：**继承建立了对象之间的关系，通过继承，可以形成类的体系结构，使得程序逻辑更加清晰和有组织性。

**劣势**

* 如果继承结构过于复杂，层次过深，会增加类之间的关系和依赖，理解和维护代码可能会更加困难。
* 子类继承了父类的属性和方法，子类与父类之间存在较强的耦合关系，当父类需要进行修改时，可能会影响到所有的子类。
* Java只支持单一继承，一个类只能继承一个父类，无法同时继承多个类，在某些情况下会限制程序设计。

**解决方案：**为了克服继承的劣势，可以采用其他面向对象编程机制，如接口、组合等来替代继承。

**3）多态**

**定义**：同一个类型的变量引用不同类型的对象，然后在运行时根据对象的实际情况动态的执行相应的方法。

**条件：**

* 存在继承关系，即子类继承父类。
* 子类重写父类的方法，即子类覆盖了父类的方法。
* 使用父类类型的引用变量来引用子类对象。

**优势**

* 代码的灵活性
* 代码的可扩展性
* 可以实现接口的透明性
* 可能实现方法的回调
* 提高代码的可读性和可维护性。

**劣势**

* 简单的功能过度抽象化
* 增加运行时额外的开销。
* 引入复杂性，增加代码阅读和理解的困难。

# 6、重写(overriding)与重载(overloading)的区别:

**重写：发⽣在父子类中，⽅法名相同，返回值与参数列表相同。**

**好处：**子类可以根据需要实现父类的方法。

**注意：**重写的方法不能抛出超父类的异常，如父类申明了一个IO异常，子类重写时不能抛出Exception异常。只能抛出IO异常的子类异常。

**重载：发⽣在同⼀类中，⽅法名相同，参数列表不同与返回值类型无关**。

**注意：**每一个重载的方法 ，都必须有一个独一无二的参数列表。

**重写与重载的区别：**重写是子父类中多态的表现，重写是一个类中多态的表现。

# 7、向上转型与向下转型：

**向上转型：**自动转型，父类引用指向子类对象。

**特点：**调用的方法如果没被子类重写，则调用父类方法，如果该方法已被子类重写，则调用重写后的方法。

**向下转型：**强制类型转换，本质是子类已经发生向上转型，再次转回来而已。

**特点：**向下转型是因为向上转型之后，要使用子类有而父类没有的方法，所以要转回来。

# 8、实例变量与静态变量的区别:

实例变量和静态变量都属于成员变量

**实例变量：**是属于对象的，在创建对象时存储在内存堆中，创建多少个对象，则实例变量就会在内存中存在多少份，需要通过引⽤变量来访问。

**静态变量：**是属于类的，在类被加载时存储在内存⽅法区中，⽆论创建多少个对象，静态变量在内存中都只有⼀份，常常通过类名点来访问

# 9、抽象类和接⼝的区别:

**抽象类:**

* 由abstract修饰
* 可以包含变量、常量、构造⽅法、普通⽅法、静态⽅法、抽象⽅法
* 派⽣类通过extends继承
* 只能继承⼀个(单⼀继承)
* 抽象类中的成员，任何访问权限都可以(默认为默认权限(同包中))

**接⼝:**

* 由interface定义
* 可以包含常量、抽象⽅法、静态⽅法(1.8后)、默认⽅法(1.8后)、私有方法(1.9后)
* 实现类通过implements实现
* 可以实现多个(体现多实现)
* 接⼝中的成员，访问权限只能是public(默认public权限)

# 10、说说Java中的值传递含义

* 对于基本类型⽽⾔，传递的是具体的值的副本
* 对于引⽤类型⽽⾔，传递的是具体的地址的副本

# 11、==和equals()的区别:

**==：**比较的是变量(栈)内存中存放的对象的(堆)内存地址，用来判断两个对象的地址是否相同，即是否指向同一个对象。比较的是真正意义上的指针操作。

* 比较的是操作符两端的操作数是否指向同一个对象。
* 两边的操作数必须是同一类型的(可以是父类之间)，编译才能通过。
* 比较的是地址，如果是具体的数字比较，值相等则为true。如int a=10与long b=10L与double c=10.0都是相同的(即为true)，因为他们都指向的地址为10的堆。

**equals：** equals用来比较的是两个对象的内容是否相等，由于所有的类都是继承于java.lang.Object类的，所以适用于所有对象，如果对象中没有重写equals方法，则调用的是Object类中的equals方法，与==的判断是等价的。

**注意：**String重写equals方法是一个字符一个字符的比较。

# 12、Java中的this与super关键字

**this：是一个特殊的关键字，它代表当前对象（实例）。**

**运用场景：**

* **引用当前对象：**

在一个实例方法中，this关键字用于引用当前调用该方法的对象。

当需要访问当前对象的成员变量或调用当前对象的其他方法时，也可以使用this关键字来引用

* **解决命名冲突：**

当方法内部存在与成员变量同名的局部变量时，使用this关键字可以解决全名冲突。

this关键字指明了要访问的是当前对象的成员变量，而不是局部变量。

* **在构造方法中引用其他构造方法：**

当一个类中定义了多个构造方法时，可以使用this关键字来引用 其他构造方法。

当需要在一个构造 方法中调用其他构造方法时，可以使用this关键字并传递相应的参数。

**注意：**this关键字只能在实例方法和构造方法中使用，不能在静态方法中使用。因为静态方法不依赖于具体的对象实例，所以无法引用this关键字。

**super：super是一个特殊的关键字，用于引用父类(超类)的成员或调用父类的构造方法。**

**运用场景：**

* **引用父类的成员：**

当子类和父类中有同名的成员变量或方法时，可以使用super关键字来引用 父类的成员。

通过super关键字，可以明确指定要访问父类的成员，而不是子类的成员。

* **调用父类的构造方法：**

当子类继承父类时，可以使用super关键字来调用父类的构造方法。

在子类的构造方法中，使用super关键字调用父类的构造方法，可以完成对父类的初始化操作。

**注意：this()是调用本类构造，super()调用父类的构造**

**this()** 和 **super()**不能出现在同一个构造器中, 否则编译不通过。

**this()** 和**super()** 在 Java 构造器中的作用是调用其他构造器。它们都必须出现在构造器的第一行。

**this()** 不能调用无参构造，会报递归构造函数调用错误。

# 13、static关键字的作用

在Java中，static关键字有以下几种用法

* **修饰变量**：使用static关键字修饰变量，表示该变量为静态变量（类变量），属于类，不属于实例，静态变量在程序启动时就会被初始化，且所有该的实例将共享同一个静态变量。可以通过类名直接访问静态变量。例如：public static int count=0;
* **修饰方法**：使用static关键字修饰的方法，表示该方法为静态方法，属于类，不属实例。静态方法可以通过类名直接调用，无需创建实例。静态方法只能直接访问静态变量和调用静态方法。

例如：public static void Count(){……}

* **静态代码块**：使用static关键字定义静态代码块，在类加载时执行且只执行一次。静态代码块常用于进行类的初始化操作，如加载配置文件，初始化静态变量等。例如：static{……}
* **静态内部类**：使用static关键字定义静态内部类，静态内部类与外部类的实例无关，可以直接通过外部类名访问。静态内部类不能访问外部类的非静态成员，但可以访问外部类的静态成员。

例如：public static class InnerClass{……}

# 13-1、程序绑定

绑定是指一个方法的调用与该方法所在的类（方法主体）关联起来。对于Java来说，绑定分为**静态绑定**和**动态绑定**（**前期绑定**和**后期绑定**）

**静态绑定：**在程序执行前就知识该方法所属的类，即在编译前该方法已经绑定。例如：final，static，private和构造方法都是前期绑定。

**结论：**如果一个方法不可被继承或者继承后不可被覆盖，那么这个方法就采用的静态绑定。

**动态绑定：**也叫运行时绑定，是指在程序运行时根据对象类型进行绑定，也叫后期绑定。

# 14、final、finally、finalize的区别：

* **final：**是关键字。final修饰的变量叫常量，必须初始化，不能被修改；final修饰的方法不能被子类重写；final修饰的类不能被继承。final修饰的参数不能在方法内部修改。final不能修饰抽象类。
* **finally：**是异常处理时的finally块，不管有没有异常被抛出、捕获，finally块都会被执行，在return前执行。
* **finalize：**是个方法名，是在Object类中定义的，因此所有的类都继承了它。finalize()方法是在垃圾回收器删除对象之前对这个方法进行调用。

# 15、说说Java中的内部类？

**定义：**内部类是指定义在其他类内部的类。

根据内部类的定义位置和访问权限，Java中的内部类可分为以下几种类型：​

* **成员内部类：**成员内部类是定义在另一个类的内部的类。它可以访问外部类的成员变量和方法，包括私有成员。成员内部类的实例化需要先实例化外部类的对象才能访问，即成员内部类的对象与外部类对象是有关联的。​
* **静态内部类：**静态内部类是定义在另一个类内部并且使用 static 修饰的类。它与普通的静态成员类似，不依赖于外部类的实例化对象而存在。静态内部类可以访问外部类的静态成员，但不能访问外部类的非静态成员。​
* **局部内部类：**方法内部类是定义在方法内部的类。它的作用域仅限于该方法内部，只能在该方法内使用。方法内部类可以访问外部类的成员变量和方法，包括私有成员，但是只能访问方法中的 final 局部变量。​
* **匿名内部类：**匿名内部类是没有命名的内部类，它通常作为实现接口或继承父类的方式来创建一个对象。匿名内部类的定义和创建都在同一个语句中完成，使用时一般作为参数传递给方法或者作为局部变量使用。匿名内部类可以访问外部类的成员变量和方法，但需要注意外部类成员变量需要为 final 类型或 effectively final 才能被匿名内部类访问。

# 16、Object类常用方法及应用场景？

Obiect类是所有类的根类，在Java中，每个类都直接或间接继承自Obiect类。

* **equals(Object obj):** 用于比较两个对象是否相等。该方法默认比较两个对象的引用是否相等，但可以根据需要进行重写，实现自定义的相等判断。
* **hashCode():** 返回对象的哈希码。可以用于自定义对象的散列算法，如将对象放入哈希表(HashMap) 中进行查找。
* **toString():** 返回对象的字符串表示，通常用于将对象转换为我们期望的字符串的格式，便于输出和调试。建议在自定义类中重写该方法，以便更好地展示对象的信息。
* **getClass():** 返回对象的运行时类，可以通过该方法获取对象所属的类的信息，例如获取类的名称、包名等。
* **finalize():** 当垃圾回收器确定不再有对该对象的引用时，会调用该方法进行资源的释放，一般不推荐使用该方法，因为它的具体调用时间不确定。
* **clone():** 创建并返回当前对象的一个拷贝(即克隆)。使用该方法，对象所属的类必须实现Cloneable接口。可以用于实现深拷贝，即复制对象及其引用。
* **wait()、notify()和notifyAl():** 用于线程间的通信和同步。这些方法配合synchronized关键字使用，实现线程的等待和唤醒机制。
* **instanceof:** 判断对象是否属于某个类或其子类的实例。可以用于类型检查，以确保对象在进行类型转换之前是安全的。

# 17、String常用的方法有哪些？

* length():获取字符串的长度(字符个数)
* trim():去除两边的空白字符
* toUpperCase()/toLowerCase():将英文部分给转换为全大写字母/全小写字母
* startsWith()/endsWith():判断是否是以???开始/结束的
* charAt():根据下标找字符
* indexOf()/lastIndexOf():查找字符串第一次出现/最后一次出现的下标
* substring():截取字符串
* 静态方法valueOf():将其它类型的数据转换为字符串
* matches():验证匹配(正则表达式)
* replaceAll():替换(正则表达式)
* split():拆分(正则表达式)

# 18、String、StringBuilder、StringBuffer的区别:

**String、StringBuffer**和**StringBuilder**是Java中用于处理字符串的三个类，他们之间的区别为

1. **可变性：**

* **String**是不可变的类，底层是final修饰的字符数组，即一旦创建了字符串对象，就不能修改其内容。对字符串进行拼接、插入或删除等操作都会创建新的字符串对象。
* **StringBuffer**和**StringBuilder**是可变的类，StringBuffer底层是一个可变数组。所以它们支持对字符串进行动态修改。如拼接、插入、删除和替换等操作，不会创建新的对象。

1. **线程安全性：**

* **String**是线程安全的，适用于多线程环境。这是因为String的不可变性，不会出现多个线程同时修改同一个字符串对象的情况。
* **StringBuffer**是线程安全的，它的方法使用了synchronized关键字修饰，保证了多线程环境下的安全性。
* **StringBuilder**是非线程安全的，它的方法没有进行同步处理。适合单线程环境下使用。

1. **性能：**

* **String**的不可变性给它带来了一些性能上的优化，比如字符串常量 池的使用。但是在频繁的字符串拼接操作中，会创建大量的临时对象，性能也会受到一定的影响。
* **StringBuffer**和**StringBuilder**的可变性使得它们在对字符串频繁操作中性能更好，特别是**StringBuilder**，因为它不需要进行同步的操作。

**总结：**如果需要对字符串进行频繁操作和修改，多线程环境下使用**StringBuffer**，单线程下使用**StringBuilder**。如果多线程环境且不需要修改，应选择不可变的**String**类。

# 19、String s = new String("hello");创建了几个对象?

**答：**首先去字符串常量池去查找，如果字符串常量池中不存在字面量hello，则创建**两个对象**，分别是hello这个字符串常量，以及new String这个实例对象。如果字符串常量池存在hello字符串常量（字面量），则只会创建**一个对象**。

# 20、字符串存储的机制是什么

字符串存储在字符串常量池中，并且可以被共享使用。当给字符串常量赋值时，会首先在常量池查找是否已经存在相同的字符串，其判断依据是string类equals()方法。若存在则直接引用该地址。若没有则首先创建这个对象，然后把它加入常量池，再将他的引用返回。

# 21、数组中的length与String类中的length有什么区别

* 数组中没有length()这个方法，但是数组中有length这个属性，用来表示数组的长度；
* String中有length()这个方法。用来得到字符串的长度；

# 22、什么是自动装箱和拆箱？

* 自动装箱是Java编译器将基本数据类型与它们对应的包装类之间进行的自动转换。

如：int转换为Integer使用方法 Integer s = Integer.valueOf(i); i为int类型。

* 自动拆箱即将包装类型的对象转换为其对应的基本类型。int I = Integer **.** valueOf(10)；

# 23、ArrayList和LinedList的区别？

ArrayList和LinkedList是在Java中用于存储和操作集合的两种常见实现方式，它两都是不同步的，不保证线程安全。它们之间的区别如下:

**1、底层数据结构:**

ArrayList是使用数组来存储元素的，内部维护一个连续的内存空间。可以根据索引速访问和修改元素

LinkedList是使用链表实现的，内部的元素通过节点相互连接。插入和删除操作效率较高，但访问元素需要遍历整个链表。

**2、插入和删除操作:**

ArrayList在末尾进行插入和删除操作的效率较高，时间复杂度为O(1)。但是中间或开头进行插入和删除操作时，需要进行元素的位移，效率较低，时间复杂度为O(n)。

LinkedList在任意位置进行插入和删除操作的效率都很高，时间复杂度为o(1)。这是因为它只需要修改向上或向下两个节点的指向，而不需要移动元素

**3、访问元素的效率:**

ArrayList可以通过索引直接访问元素，时间复杂度为O(1)。适用于频繁访问和随机访问的场景。

LinkedList需要从链表头或尾开始遍历，直到找到目标位置的元素。时间复杂度为O(n)，适用于按顺序访问的场景

**4、空间消耗:**

ArrayList在创建时会分配一块连续的内存空间，并根据实际需要进行动态扩容。如果元素数量较多，可能会浪费一些空间

LinkedList在存储元素时还需要额外的空间用于存储节点之间的链接关系。所以，相同数量的元素，inkedList通常占用的空间更多。

# 24、说说Java中的泛型及应用?

泛型提供了编译时类型安全检测机制。该机制允许程序员在编译时检测到非法的类型。泛型的本质是参数化类型，它允许在类、接口、方法在定义中使用类型参数，使得我们可以在编译时期指定集合、类或方法的数据类型，以提高代码的安全性和可读性。

**Java中的泛型主要有以下几个要点**

* **泛型类和接口：**泛型类或接口通过使用尖括号<>来指定类型参数。
* **泛型方法：**泛型方法是定义在普通类或接口中的方法，并在方法返回值类型前使用尖括号<>来指定类型参数
* **类型通配符：**当我们需要在泛型中使用未知的类型时，可以使用通配符来表示。通配符有两种形式：<?> 和 <? extends Type>。例如，List<?> 表示一个末知类型的列表，而 List<? extends Number> 表示一个可以存放任意 Number 类型或其子类的列表。 List<? super Integer> 表示一个可以存放Integer 类型或其类型的父类类型。
* **类型推断：** Java 7 引入了类型推断，使得在使用泛型时可以省略类型参数的声明。例如，使用 diamond 语法可以简化创建泛型实例的代码如：List<String> list= new ArrayList<>();

**总结：**使用泛型的好处是可以提高代码的类型安全性，减少类型转换的错误，并目使代码更加通用和可读。通过使用泛型，可以在编译时期对代码进行更严格的类型检查，并提供更好的代码提示。

# 25、Java中什么是泛型类型擦除

Java使用的是JVM来执行运行时的代码，JVM中没有泛型类型的概念，所以在Java开发过程中，当[编译器](https://www.zhihu.com/search?q=%E7%BC%96%E8%AF%91%E5%99%A8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7b:,:2915020960%7d)对带有泛型的Java代码进行编译时，会去执行类型检查和类型推断，然后将泛型类型转换为它们的原始类型（Raw type）或是替换为Object类型，这种字节码可以被一般的Java[虚拟机](https://www.zhihu.com/search?q=%E8%99%9A%E6%8B%9F%E6%9C%BA&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7b:,:2915020960%7d)接收并执行，这种技术被称为擦除。

**擦除原则：**

1、所有[参数化容器](https://www.zhihu.com/search?q=%E5%8F%82%E6%95%B0%E5%8C%96%E5%AE%B9%E5%99%A8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7b:,:2915020960%7d)类都被擦除成非参数化的：如List<E>、List<List<E>>都被擦除成List

2、所有[参数化数组](https://www.zhihu.com/search?q=%E5%8F%82%E6%95%B0%E5%8C%96%E6%95%B0%E7%BB%84&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7b:,:2915020960%7d)都被擦除成非参数化的数组：如List<E>[]，被擦除成List[]

3、Raw type（原始类）的容器类，被擦除成其自身，如List 被擦除成List

4、原生类型（int,String还有wrapper（包装）类）都擦除成他们的自身

5、参数类型E，被擦除成Object

6、所有约束参数，如<? Extends E>、<X extends E>都被擦除成E

7、如果有多个约束，擦除成第一个，如<T extends Object & E>，则擦除成Object

# 26、说说你对Java中泛型的理解（泛型的特点）

Java中的泛型是一种在编译时期进行类型检查和类型约束的机制，它的特点有:

**1、类型安全性：**泛型可以在编译阶段检查类型的正确性，避免了在运行时发生类型转换错误的问题。通过使用泛型，可以在编写代码时就能够确定对象的类型，并目编译器会确保在使用对象时类型的一致性。

**2、代码重用性：**泛型可以使代码更加通用和可重用，因为可以将类型参数化，使得同一段代码可以适用于不同类型的数据。例如，可以定义一个通用的集合类，可以存储不同类型的对象，提高代码的灵活性。

**3、可读性和安全性提高：**有了泛型，代码更加清晰和易读，因为泛型可以提供更具描述性的类型信息。通过使用泛型，可以减少类型转换和强制类型检查的需要，使得代码可读性更强，同时也减少了出错的可能性

**4、避免了强制类型转换：**在使用非泛型的集合类时，需要进行强制类型转换以获取存储在集合中的对象。而泛型可以省略这一步骤，使得代码更加简洁和安全

# 27、Java中创建对象的方式

1、**使用new关键字：** 这是最常见的创建对象的方式，在堆内存中创建一个新的对象实例。例如：MyObject obj = new MyObiect();

2、**通过反射机制：**利用Java反射机制中的Class类里的newInstance0方法来创建一个类的实例。这种方式需要无参构造方法。例如:MyObject obj = MyClass.class.newInstance();

3、**使用Constructor类的newInstance()方法：** 同样是通过反射机制，我们可以使用Constructor类的newInstance0)方法来创建一个类的实例。这种方式可以使用带参构造方法创建对象。例如: Constructor<MyClass> constructor =MyClass.class.getConstructor(String.class);

MyClass obj =constructornewInstance("parameter"):

4、**使用clone()方法：**如果一个类实现了Cloneable接口，我们可以使用clone()方法来创建一个对象的副本，使用clone()方法创建对象时，实际上是调用了对象的clone0)方法，并返回一个新的对象实例。例如: MyObiect obi2 =(MyObiect)obj.clone();

5、**使用反序列化：**如果一个类实现了Serializable接口，我们可以使用反序列化的方式来创建对象。通过将对象转换成字节序列，然后再将字节序列转换回对象，来实现对象的创建。

例如: ObjectInputStream in = new ObjectInputStream(newFileInputStream("file.ser"));

MyObject obj =(MyObject) in.readObject();

**总结：**无论使用哪种方式创建对象，最终都会在内存中为对象分配空间，并调用构造方法来进行初始化。每种方式都有其适用的场景和注意事项，开发者应根据具体需求来选择适合的方式来创建对象。

# 28、什么是IO流，IO流是怎么分类的

IO是Input（输入）和Output（输出）的首字母缩写。

I（输入Input）：指向Java程序中输入数据，即Java程序从外部获取数据。  
O（输出Output）：指的是Java程序向外部输出数据，即Java程序向外部发送数据。

**按流的方向：**

* **输入流（InputStream）**：从文件读入到内存。只能进行读操作。
* **输出流（OuputStream）**：从内存读出到文件。只能进行写操作。

注：输出流可以帮助我们创建文件，而输入流不会。

**根据操作单元：**

* **字节流**：以字节为单位读写数据，常用于处理二进制数据。可以读任何类型数据，图片、文件、音乐视频等。 (Java代码接收数据只能为byte数组，每次操作8位数据)
* **字符流**：以字符为单位读写数据，常用于处理文本数据。内部会根据指定的字符编码进行转换 (Java代码接收数据为一般为char数组，也可以是别的，每次操作16位数据)

**根据角色：**

* **节点流**：直接与数据源或目标进行相连。例FileInputstream 和FileQutputstream
* **处理流**：也叫**包装流**，是通过包装其他流来提供额外功能的流。如 BufferedInputStream 和 BufferedOutputStream 是增加了缓冲功能的处理流，它们可以提高读写的效率。

**注意：**为什么要有处理流？主要作用是在读入或写出时，对数据进行缓存，以减少I/O的次数。

# 29、BIO和NIO、AIO的区别

**1、BIO同步并阻塞：**一个连接一个线程。

JDK1.4之前，建立网络连接的时候采用BIO模式，需要先在服务端启动socket，然后在客户端启动socket来对服务端进行通信。客户端发送请求后，先判断服务端是否有线程响应，如果没有则会一直等待或者遭到拒绝请求，如果有的话会等待请求结束后才继续执行。

**2、NIO同步非阻塞：**一个请求一个线程。

NIO主要是想解决BIO的大并发问题，BIO是每一个请求分配一个线程，当请求过多时，每个线程占用一定的内存空间，服务器瘫痪了。

JDK1.4开始支持NIO，适用于连接数目多且连接比较短的架构，比如聊天服务器，并发局限于应用中。

**3、AIO异步非阻塞：**一个有效请求一个线程。

JDK1.7开始支持AIO，适用于连接数目多且连接比较长的结构，比如相册服务器，充分调用OS参与并发操作。

# 30、什么是深拷贝与浅拷贝

**深拷贝 (Deep Copy)** 和**浅拷贝 (Shallow Copy)** 是用来复制对象的两种不同的方式。

**浅拷贝：**是创建一个新对象，并将原始对象的引用复制到新对象中。这意味着新对象和原始对象将共享相同的内存地址，修改一个对象的引用地址会影响到另一个对象。

**深拷贝：**是创建一个新对象，并将原始对象的所有属性的副本复制到新对象中。这意味着新对象和原始对象是完全独立的，修改一个对象的引用地址不会影响到另一个对象

**注意：**

* 在深拷贝中，如果原始对象的属性是值类型 (例如整数、浮点数等)，则会直接复制它们的值。如果原始对象的属性是引用类型 (例如列表、字典、自定义类等)，则会递归地复制它们的内容，以确保完全独立的新对象
* 在浅拷贝中，无论原始对象的属性是值类型还是引用类型，都只是复制它们的引用.这意味着新对象和原始对象之间共享相同的引用类型属性，修改一个对象的属性会影响到另一个对象。

# 31、进程和线程的区别

进程是资源分配的最⼩单位，线程是CPU调度的最⼩单位。操作系统执⾏程序时候，按照进程分配内存等资源，⽽执⾏程序时候以线程为单位执⾏程序中的指令。⼀个进程内部包含多个并发执⾏的线程。在进程内部多个线程共享⼀个进程的内存资源。

# 32、线程的run()和start()有什么区别

* run方法内定义了线程执行的任务逻辑，重写run方法可以实现自定义的线程行为。
* start方法用来启动线程，并使其进入就绪状态，等待被调度执行；
* 需要并行处理的代码放在run方法中，start方法启动线程后自动调用run方法，该方法会在当前线程中同步执行，而不会启动一个新的线程。
* run方法必须是public的[访问权限](https://so.csdn.net/so/search?q=%E8%AE%BF%E9%97%AE%E6%9D%83%E9%99%90&spm=1001.2101.3001.7020" \t "_blank)，返回类型为void。此方法可以重复调用。
* start只能调用一次start方法是异步执行的，即调用后会立即返回，并不会阻塞当前线程。

# 33、创建线程的几种方式是什么？

* **通过继承 Thread 类创建线程类。**

通过继承 java.lang.Thread 类并重写其 run() 方法，然后创建该子类的实例并调用 start() 方法来启动线程。

* **实现 Runnable 接⼝创建线程类。**

通过实现 java.lang.Runnable 接口并重写其 run() 方法，然后将该实现类的实例作为参数传递给 Thread 类的构造方法，并调用 start() 方法来启动线程。

# 34、实现 Runnable 接口比继承 Thread 类所具有的优势

* 适合多个相同的程序代码的线程去处理同一个资源
* 可以避免 java 中的单继承的限制
* 增加程序的健壮性，代码可以被多个线程共享，代码和数据独立
* 线程池只能放入实现 Runable 或 callable 类线程，不能直接放入继承 Thread 的类
* runnable 实现线程可以对线 程进行复用，因为 runnable 是轻量级的对象，重复 new 不会耗 费太大资源，而 Thread 则不然，它是重量级对象，而且线程执行完就完了，无法再次利用

# 35、Java线程对象都有哪些状态？

在Java中，线程对象可以处于不同的状态，这些状态反映了线程在不同阶段的执行情况。具体状态说明如下:

**1、新建状态 (New) :**

* 新建状态表示线程对象创建后尚未启动，即尚未调用start()方法。
* 在新建状态下，线程对象的一些初始化操作可以进行，如分配内存、初始化变量等.

**2、就绪状态 (Runnable) :**

* 就绪状态表示线程对象已经准备好执行，并且等待获取CPU时间片。
* 当线程对象被调度器选中，并分配到CPU时间片时，它会从就绪状态切换到运行状态。

**3、运行状态 (Running) :**

* 运行状态表示线程对象正在执行run()方法中的代码。
* 在运行状态下，线程会不断占用CPU资源，执行自己的任务

**4、阻塞状态 (Blocked) :**

* 阻塞状态表示线程对象由于某些原因暂时无法执行，需要等待唤醒或满足特定条件。
* 当线程处于阻塞状态时，它会释放持有的锁，让其他线程有机会获取锁并执行。线程被唤醒后会从阻塞状态0切换到就绪状态。

**5、等待状态 (Waiting) :**

* 等待状态表示线程对象正在等待其他线程的特定操作，无法被唤醒。
* 线程进入等待状态的常见情况包括调用了wait()方法、join(方法、park()方法等。
* 当满足某个条件后，其他线程可以通知等待状态的线程，并将其唤醒。被唤醒后的线程会从等待状态切换到就绪状态。

**6、超时等待状态 (Timed Waiting)**

* 超时等待状态与等待状态类似，但是在等待一定时间后会自动返回。
* 线程进入超时等待状态的常见情况包括调用了带有超时参数的wait()方法、sleep()方法、join()方法等。
* 当时间到达或满足某个条件后，线程会从超时等待状态切换到就绪状态。

**7、终止状态 (Terminated) :**

* 终止状态表示线程对象已经执行完run()方法，或者出现了异常而终止了执行。
* 在终止状态下，线程对象的执行已经结束，不再具有了执行能力。

# 36、如何理解Java中的守护线程

在Java中，线程分为两种类型：**用户线程和守护线程**。

**用户线程：**是程序中普通的线程，当所有的用户线程都执行完毕后，Java虚拟机会退出。

**守护线程：**是一种特殊的线程，主要用于提供后台服务和支持性工作。守护线程的特点如下:

* 守护线程是在程序中被创建的，和用户线程类似，可通过setDaemon(true)方法设置为守护线程。
* 当所有的用户线程结束时，即使守护线程还未执行完毕，Java虚拟机也会退出，不会等待守护线程完成
* 守护线程通常用于执行一些非核心的、非必要的任务，例如垃圾回收线程就是一个守护线程。在实际开发中，守护线程可以用于执行一些后台服务或监控工作。例如，你可以创建一个守护线程用于定期清理临时文件，或者创建一个守护线程用于监控某个资源的变化。守护线程的作用是为用户线程提供支持，而不会阻止程序的退出。

# 37、什么是死锁，产⽣死锁的四个条件

* **互斥条件：**⼀个资源每次只能被⼀个进程使⽤。
* **请求与保持条件：**⼀个进程因请求资源⽽阻塞时，对已获得的资源保持不放。
* **不剥夺条件：**进程已获得的资源，在末使⽤完之前，不能强⾏剥夺。
* **循环等待条件：**若⼲进程之间形成⼀种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发⽣死锁，这些条件必然成⽴，⽽只要上述条件之⼀不满⾜，就不会发⽣死锁。

# 38、请细说一下synchronized关键字?

在Java中，synchronized关键字用于实现线程之间的同步。它可以修饰**方法**、**代码块**以及**静态方法**，来保证在多线程访问**共享资源**时的安全性。synchronized的使用方式和特点:

**1、修饰方法：**

synchronized可以修饰普通方法，使用该关键字修饰的方法称为同步方法。在同步方法中，当线程访问该方法时，会自动获得该方法所在对象的监视器(即锁)，其他线程则被阻塞，直到该线程执行完该方法，释放锁。同步方法是隐式获取锁的方式，它的锁是该方法所在对象的实例锁(即对象锁)。

**2、修饰代码块：**

synchronized还可以修饰代码块，使用该关键字修饰的代码块称为同步块。在同步块中，需要指定获取锁的对象或者类。当线程执行到同步块时，会尝试获取指定对象 (或类)的监视器 (即锁)，其他线程则被阻，直到该线程执行完同步块，释放锁。同步块的锁可以是任意对象，也可以使用this关键字表示当前对象的实例锁，还可以使用类名.class表示类锁。

**3、特点：**

* synchronized保证了同一时间只有一个线程可以执行同步代码块或同步方法，从而保证了共享资源的安全访问。
* synchronized关键字具有原子性，即在同步代码块或同步方法内的操作不会被其他线程打断
* synchronized还具有可见性，即一个线程对共享变量的修改会立即对其他线程可见
* synchronized关键字是重量级锁，获取和释放锁涉及到用户态和内核态之间的切换，会带来一定的性能开销。

# 39、说说你对序列化与反序列化的理解？（什么是序列化与反序列化用什么作用）

**定义：**

* **序列化：**把对象转化为字节序列的过程称为序列化。
* **反序列化：**把字节序列还原为对象的过程称为反序列化。

**作用：**

**序列化**是为了使对象可以被存储，传输、持久化及网络传输。序列化后的字节流可以保存到文件、数据库中，或者通过网络传输到远程主机中。

**反序列化**是将字节流重新转换为原始的对象，可以将序列化对象反复利用，使对象的状态可以在不同的运行环境中恢复。

**使用场景：**

* 对象在进行网络传输（比如远程方法调用 RPC 的时候）之前需要先被序列化，接收到序列化的对象之后需要再进行反序列化。
* 将对象存储到文件之前需要进行序列化，将对象从文件中读取出来需要进行反序列化。
* 将对象存储到数据库时可以将对象以二进制形式存储。将对象从缓存数据库中读取出来需要反序列化。
* 将对象存储到内存之前需要进行序列化，从内存中读取出来之后需要进行反序列化。
* 在分布式系统中，可以通过序列化和反序列化将对象在不同的节点之间传递。
* 在Web开发中，可以将对象转换为JSON或XML格式进行传输。

**使用机制：**

* java 实现序列化很简单，只需要被序列化对象类实现Serializable 接⼝，然后使⽤对象流进⾏序列化和反序列化。
* 使⽤ObjectOutputStream 进⾏对象序列化。
* 使⽤ObjectInputStream 进⾏对象反序列化。

# 40、反射技术的理解

反射技术是Java中一种机制，java反射（Java Reflect）是指程序在运⾏时动态地获操作类的信息。通过反射，可以在运行时利用获取类的名称、方法、字段等信息动态的创建对象，调用方法，访问和修改字段及执行其他与类相关的操作。

**反射技术的核心是java.lang.reflect包中的一组类和接口。**常用的反射类和接口有哪些

* Class类：代表一个类或接口，在运行时可以通过它获取和操作类的信息，例如获取类的名称、父类、实现的接口、构造函数、方法、字段等。
* Constructor类：代表类的构造函数，在运行时可以通过它创建对象
* Method类：代表类的方法，在运行时可以通过它调用方法
* Field类：代表类的字段，在运行时可以通过它访问和修改字段的值。

**反射机制有哪作用：**

**1、获取对象**

* Class.forName(全路径类名)：Class<?> c1 = Class.forName("com.yzq.Student");
* 类名.class ：Class<Student> c2 = Student.class;
* 实例对象,.getClass()：

Student s = new Student();

Class<? extends Student> c3 = s.getClass();

**2、动态创建对象:**

通过Class类的newInstance()方法可以创建类的实例，相当于调用了该类的无参构造函数.

**3、动态调用方法:**

通过 Method 类的 nvoke() 方法可以调用类的方法，可以传递参数并获取返回值。

**4、动态访问和修改字段：**

通过 field 类可以获取并修改类的字段的值

**总结：**使用反射技术可以在运行时动态地创建对象、调用方法、访问和修改字段，这使得程序具有更大的灵活性和扩展性。同时，反射技术也带来一定的性能开销，因此在性能敏感的场景下需要谨慎使用。

# 41、HashMap特点及应用场景?

HashMap是Java中常用的哈希表实现类，它实现了Map接口，用于存储建值对数据。HashMap具有以下几个特点和应用场景:

**1、高效的查找和插入操作：**HashMap的底层是通过哈希表实现的，使用了哈希函数将键转换为数组索引。因此，在查找和插入元素时，HashMap具有较高的效率。

**2 、键值对存储：**HashMap可以存储键值对类型的数据，每个键都是唯一的。通过键可以快速查找对应的值。HashMap中的键和值都可以为null，但一个HashMap只能有一个null键。

**3、无序的元素：**HashMap中元素的存储顺序是不固定的，取决于哈希值的计算结果。如果需要有序的元素，可以考虑使用LinkedHashMap。

**4、线程不安全：**HashMap是非线程安全的数据结构。如果在多线程环境下使用HashMap，可能会导致数据不一致的问题。如果需要在多线程环境下使用HashMap，可以考虑使用ConcurrentHashMap。

**5、可变大小：**HashMap的大小是可以动态调整的，根据实际的元素数量自动扩容和收缩.

**6、应用场景：**HashMap适用于需要根据键快速查找对应值的场景。例如，常见的应用场景包括缓存、数据索引、字典等。通过将键与对应的值进行关联，可以快速地进行查找和操作。

# 42、LinkedHashMap特点及应用场景?

LinkedHashMap是Java中的一种Map实现类，它基于哈希表和双向链表的数据结构，可以保持插入顺序或者访问顺序的有序性。LinkedHashMap具有以下几个特点和应用场景:

**1、有序性：**LinkedHashMap可以按照元素的插入顺序或者访问顺序进行遍历。在插入顺序模式下，元素的顺序与插入的顺序一致;在访问顺序模式下，元素会根据访问的顺序进行排序。

**2、高效的检索操作：**由于LinkedHashMap基于哈希表的实现，在读取操作时具有较高的性能。通过哈希表的散列算法，可以快速定位到元素所在的位置，提高了检索的效率。

**3、可变大小：**LinkedHashMap的大小是可以动态调整的，可以根据实际元素数量自动扩容和收缩。这样可以节省内存空间，并提高性能。

**4、迭代器的一致性：**LinkedHashMap的迭代器可以保证在遍历过程中，元素的顺序与插入或访问的顺序一致。这对于需要保持顺序的遍历操作非常有用。

**5、应用场景：**LinkedHashMap适用于需要有序存储和遍历的场景。例如，在LRU (Least Recently Used) 缓存实现中，可以使用LinkedHashMap来维护缓存的顺序，把最近使用的元素放在链表的头部，最少使用的元素放在链表的尾部。同时，LinkedHashMap不可以用于实现LRU近似算法的缓存淘汰策略，通过设置accessOrder为true，使得最近访问的元素被放在链表的尾部，从而实现近似的LRU策略。

# 43、说说TCP协议以及其特点?

TCP(传输控制协议)是一种可靠的传输层协议，用于在互联网上可靠地传输数据。TCP协议的特点是:

**1、可靠性：**TCP提供数据传输的可靠性，通过使用确认机制、席列号和超时重传来确保数据的可靠传递。TCP会对每个发送的数据包进行确认，如果发送方没有收到确认，会重新发送数据包，直到对方确认接收。

**2、有序性：**TCP保证数据的有席传输。每个TCP报文段都有一个序列号，接收方根据序列号对接收的数据进行按序重组，保证数据的正确排序。

**3、流量控制：**TCP使用滑动窗口机制来进行流量控制，确保发送方发送的速度不会超过接收方的处理能力。接收方会通知发送方可接收的数据量，发送方根据接收方的反馈进行调整，以避免数据丢失和网络拥塞。

**4、拥塞控制：**TCP使用拥塞控制算法来避免网络拥塞。通过动态调整发送速率和传输窗口大小，TCP可以适应网络的变化，保持网络的稳定性和公亚性

**5、面向连接：**TCP是一种面向连接的协议，通信前需要建立一个连接。连接的建立使用了三次握手的过程，即发送方和接收方都需要确认连接的建立。通信结束后，需要进行四次挥手的过程来正常关闭连接。

**6、全双工通信：**TCP支持全双工通信，即发送方和接收方可以同时发送和接收数据，实现双向的数据传输.

**7、基于字节流：**TCP以字节流方式传输数据，将数据分割成较小的TCP报文段进行传输。

# 44、三次握⼿四次挥⼿

**三次握手：**

**第一次：**Client将SYN置1，随机产生一个初始序列号seq发送给Server，进入SYN\_SENT状态；

**第二次：**Server收到Client的SYN=1之后，知道客户端请求建立连接，将自己的SYN置1，ACK置1，产生一个认可帐号acknowledge number=sequence number+1，并随机产生一个自己的初始序列号，发送给客户端；进入SYN\_RCVD状态；

**第三次：**客户端检查acknowledge number是否为序列号+1，ACK是否为1，检查正确之后将自己的ACK置为1，产生一个acknowledge number=服务器发的序列号+1，发送给服务器；进入ESTABLISHED状态；服务器检查ACK为1和acknowledge number为序列号+1之后，也进入ESTABLISHED状态；完成三次握手，连接建立。

**简化理解 ：**

1. 客户端给服务端发送信息，让服务端知道自己的发件能力。
2. 服务端收到后知道了客户端的发件能力和自己的收件能力是可以的，回信给客户端。
3. 客户端收到之后明白自己的发件能力与服务端的收件能力是可以的，并且客户端的发件能力和自己的收件能力也是可以的。此时服务端还不知道自己的发件能力和客户端的收件能力是否可以，所以再进行最后一次反馈，如果服务端收到，则证明服务端的发件能力与客户端的收件能力是可以的，连接完成。

**四次挥手：**

**第一次：**Client将FIN置为1，发送一个序列号seq给Server；进入FIN\_WAIT\_1状态；

**第二次：**Server收到FIN之后，发送一个ACK=1，acknowledge number=收到的序列号+1；进入CLOSE\_WAIT状态。此时客户端已经没有要发送的数据了，但仍可以接受服务器发来的数据。

**第三次**：Server将FIN置1，发送一个序列号给Client；进入LAST\_ACK状态；

**第四次**：Client收到服务器的FIN后，进入TIME\_WAIT状态；接着将ACK置1，发送一个acknowledge number=序列号+1给服务器；服务器收到后，确认acknowledge number后，变为CLOSED状态，不再向客户端发送数据。客户端等待2\*MSL（报文段最长寿命）时间后，也进入CLOSED状态。完成四次挥手。

**简化理解：**

(1)、[TCP](https://baike.baidu.com/item/TCP/33012?fromModule=lemma_inlink" \t "_blank)客户端发送一个FIN，用来关闭客户到服务器的[数据传送](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E4%BC%A0%E9%80%81/500685?fromModule=lemma_inlink" \t "_blank)。

(2)、服务器收到这个FIN，发回一个[ACK](https://baike.baidu.com/item/ACK/3692629?fromModule=lemma_inlink" \t "_blank)，确认序号为收到的序号加1。

(3)、服务器关闭客户端的连接，发送一个FIN给客户端。

(4)、客户端发回ACK报文确认，并将确认序号设置为收到序号加1。

# 45、tcp为什么要三次握手，两次不行吗？为什么？

因为客户端和服务端都要确认连接，两次无法保证A能收到B的数据

* 客户端请求连接服务端；
* 针对客户端的请求确认应答，并请求建立连接；
* 针对服务端的请求确认应答，建立连接；