Java 基础、集合框架、并发编程

什么是 JDK、JRE、JVM？三者的关系是什么？

JDK (Java Development Kit)：Java 开发工具包，包含开发 Java 程序所需的工具（如javac编译器、jdb调试器）和 JRE。是开发者必须安装的环境。

JRE (Java Runtime Environment)：Java 运行时环境，包含 JVM 和运行 Java 程序所需的核心类库（如rt.jar）。仅需运行 Java 程序时安装。

JVM (Java Virtual Machine)：Java 虚拟机，负责将字节码（.class文件）解释 / 编译为本地机器码并执行，是 “一次编译，到处运行” 的核心。

关系：JDK ⊇ JRE ⊇ JVM（JDK 包含 JRE，JRE 包含 JVM 和类库）

String 的不可变性、底层实现及频繁修改推荐类

String 不可变性的体现

不可变性指 String 对象创建后，其内部字符序列无法被修改，具体体现在：

字符存储容器不可变：String 底层用 private final char[] value（JDK8）或 private final byte[] value（JDK9+，更节省空间）存储字符，final 修饰数组意味着数组引用不能被重新赋值（即不能指向新数组），且 private 修饰保证外部无法直接操作数组。

无修改方法：String 类中没有提供修改字符的方法（如 “修改某索引字符”“删除字符”），看似 “修改” 的方法（如 substring()、replace()、concat()），实际是 创建新的 String 对象 并返回，原对象不变。

类本身被 final 修饰：public final class String 确保 String 不能被继承，避免子类重写方法破坏不可变性。

频繁修改字符串的推荐类：StringBuilder 或 StringBuffer

String 频繁修改会创建大量临时对象，导致内存浪费和 GC 压力；而 StringBuilder/StringBuffer 底层用 可变数组 存储字符，支持直接修改数组内容，避免创建新对象。

| **特性** | **String** | **StringBuilder** | **StringBuffer** |
| --- | --- | --- | --- |
| 可变性 | 不可变（底层 char 数组被 final 修饰） | 可变（动态扩容的 char 数组） | 可变（动态扩容的 char 数组） |
| 线程安全 | 安全（不可变天然线程安全） | 不安全（无同步锁） | 安全（方法加synchronized锁） |
| 性能 | 低（拼接会创建新对象） | 高（单线程首选） | 中（多线程场景） |

final、static、volatile 关键字的作用及 volatile 不保证原子性

（1）final 关键字：“不可变” 的修饰符

可修饰 类、方法、变量，作用不同：

修饰类：类不能被继承（如 String、Integer），避免子类扩展破坏原逻辑。

修饰方法：方法不能被重写，确保父类方法逻辑不被篡改（如工具类的核心方法）。

修饰变量：

基本类型变量：值不能修改（如 final int a = 10; a = 20; // 编译报错）；

引用类型变量：引用不能指向新对象（但对象内部属性可修改，如 final List list = new ArrayList(); list.add(1); // 合法）。

（2）static 关键字：“属于类” 的修饰符

可修饰 类、方法、变量、代码块，表示 “与类绑定，而非实例”(JVM中类加载器中加载)：

静态变量（类变量）：所有实例共享同一值，存储在 方法区（而非堆），通过 类名.变量名 访问（如 Math.PI）。

静态方法（类方法）：无 this 指针，不能访问非静态成员（非静态成员属于实例），通过 类名.方法名 调用（如 Arrays.sort()）。

静态代码块：类加载时执行一次，用于初始化静态变量（如加载配置文件）。

静态内部类：不依赖外部类实例，可直接创建（如 HashMap.Node），常用于工具类。

（3）volatile 关键字：“可见性” 和 “禁止重排序” 的保证

作用是解决 多线程下的内存可见性和指令重排序问题：

内存可见性：当一个线程修改 volatile 变量后，其他线程能 “立刻看到” 最新值（避免线程读取到本地缓存的旧值）；

禁止指令重排序：阻止 JVM 对 volatile 变量相关的指令进行优化重排（如避免 “双重检查单例” 中的空指针问题）。

（4）volatile 为什么不能保证原子性？

原子性指 “一个操作要么全部执行，要么全部不执行，中间不会被打断”。volatile 仅保证可见性和禁止重排序，不保证复合操作的原子性—— 因为复合操作（如 i++）本质是 “读取 - 修改 - 写入” 三步，多线程下可能被打断。

synchronized 和 Lock 的区别及死锁避免

| **对比维度** | **synchronized** | **ReentrantLock（Lock 接口实现类）** |
| --- | --- | --- |
| 锁的获取 / 释放 | 自动（进入同步块时获取，退出时释放，包括异常退出） | 手动（lock() 获取，unlock() 释放，必须在 finally 中释放，避免死锁） |
| 锁类型支持 | 非公平锁（默认），不可设置为公平锁 | 支持公平锁（构造器传 true）和非公平锁（默认） |
| 可中断性 | 不可中断（一旦进入阻塞，只能等锁释放或程序终止） | 可中断（lockInterruptibly() 方法，允许线程在等待锁时响应中断） |
| 超时获取锁 | 不支持（等待时无超时，可能永久阻塞） | 支持（tryLock(long time, TimeUnit unit)，超时未获取则返回 false，避免永久阻塞） |
| 条件变量（Condition） | 不支持（仅能通过 wait()/notify() 通信） | 支持（newCondition() 方法，可创建多个 Condition，实现更精细的线程通信） |
| 性能 | JDK6+ 优化后（如偏向锁、轻量级锁），性能接近 ReentrantLock | 性能稳定，在高并发下略优于未优化的 synchronized |

死锁的产生条件：资源互斥、持有并等待、不可剥夺、循环等待，破坏任一条件即可避免死锁

按固定顺序获取锁：所有线程获取多个锁时，按统一的顺序（如按锁的哈希值从小到大）获取，避免 “循环等待”。

超时释放锁：用 ReentrantLock 的 tryLock(timeout) 方法，若超时未获取锁则释放已持有锁，避免 “持有并等待”。

使用 ReentrantLock 的可中断特性：线程等待锁时，若检测到死锁风险（如通过 JVM 工具），可调用 interrupt() 中断线程，释放已持有锁，破坏 “不可剥夺”。

MyBatis 中 #{} 和 ${} 的区别及 SQL 注入防护

（1）#{} 和 ${} 的核心区别

| 对比维度 | #{} | ${} |
| --- | --- | --- |
| 底层处理方式 | 预编译参数：将 #{} 替换为 ?，通过 PreparedStatement 设参（setString()） | 字符串拼接：直接将 ${} 替换为参数值，无预编译 |
| SQL 注入防护 | 支持（参数值会被转义，如单引号 ' 转义为 ''） | 不支持（直接拼接，参数中若有恶意 SQL 会被执行） |
| 适用场景 | 传递参数（如查询条件、插入值，如 where id = #{id}） | 动态拼接标识符（如表名、列名，如 select \* from ${tableName}） |

Redis常见数据结构

| **数据结构** | **核心特点** | **应用场景** | **示例** |
| --- | --- | --- | --- |
| String | 存储字符串、数字（int/float），支持自增 / 自减、拼接 | 1. 缓存单个值（如用户信息、商品详情）；2. 计数器（如文章阅读量、接口调用次数）；3. 分布式锁（setnx 命令）。 | set user:100 "{name:'张三', age:20}"；incr article:read:100 |
| Hash | 存储键值对集合（field-value），适合存储对象 | 1. 缓存对象（如用户信息，field 为属性名，value 为属性值）；2. 购物车（用户 ID 为 key，商品 ID 为 field，数量为 value）。 | hset user:100 name 张三 age 20；hget user:100 name |
| List | 有序列表（双端链表），支持头部 / 尾部插入 / 删除 | 1. 消息队列（lpush 生产，rpop 消费）；2. 排行榜（如最新评论列表）；3. 栈 / 队列（lpush + lpop 是栈，lpush + rpop 是队列）。 | lpush msg:queue "消息1"；rpop msg:queue |
| Set | 无序集合（无重复元素），支持交集、并集、差集 | 1. 去重（如用户标签去重）；2. 好友关系（如共同好友：交集）；3. 抽奖（srandmember 随机取元素）。 | sadd user:tags:100 篮球 音乐；sinter user:friends:100 user:friends:200（共同好友） |
| ZSet | 有序集合（按 score 排序，无重复元素） | 1. 排序排行榜（如商品销量榜、用户积分榜）；2. 范围查询（如查询积分前 10 的用户）。 |  |

Java三大特性

封装：封装是将对象的属性（数据）和方法（操作）捆绑在一起，并对外部隐藏内部实现细节的机制。通过访问修饰符（如private、public、protected）控制类的成员是否可被外部访问，仅暴露必要的接口。

继承：继承允许一个类（子类）继承另一个类（父类）的属性和方法，实现代码复用和扩展。子类可以重写父类的方法以适应自身需求，体现 "is-a" 的关系。

多态：多态指同一操作作用于不同对象时，产生不同的行为。

重载和重写

| **对比维度** | **方法重载（Overload）** | **方法重写（Override）** |
| --- | --- | --- |
| 定义 | 同一类中，方法名相同、参数列表不同（个数 / 类型 / 顺序） | 子类中，重写父类的方法（方法名、参数、返回值完全一致） |
| 返回值 | 可不同（仅参数不同即可） | 必须相同或为父类返回值的子类（协变返回，JDK 1.5+） |
| 访问修饰符 | 可修改 | 子类修饰符权限≥父类（如父类protected，子类可public） |
| 异常抛出 | 可修改 | 子类抛出的异常≤父类（范围更小或同类） |
| 作用场景 | 同一功能的不同实现（参数差异） | 子类对父类功能的扩展或修改 |

Java中的值传递和引用传递

Java 只有值传递，不存在引用传递。“值传递” 的定义是：方法调用时，传递的是参数变量的 “副本”（基本类型副本是值本身，引用类型副本是内存地址），原变量不会被方法内的修改直接改变

Java自动装箱和拆箱原理

| **维度** | **基本数据类型（如 int、double）** | **包装类（如 Integer、Double）** |
| --- | --- | --- |
| 存储方式 | 直接存储值（栈或局部变量表） | 存储对象引用（堆中存对象） |
| 默认值 | 有默认值（如 int 为 0） | 默认值为 null |
| 能否参与泛型 | 不能（泛型仅支持对象类型） | 能（如 List<Integer>） |
| 方法 / 属性 | 无（不能调用方法） | 有（如 Integer.parseInt()） |

自动装箱：基本类型（如 int）自动转为包装类（如 Integer），如 Integer i = 10;（编译时转为 Integer.valueOf(10)）；

自动拆箱：包装类自动转为基本类型，如 int num = i;（编译时转为 i.intValue()）。

常见坑及案例：

NullPointerException（NPE）：  
包装类可能为 null，拆箱时会调用 intValue() 等方法，若包装类为 null 则抛 NPE：

Integer i = null; int num = i; // 编译通过，运行时抛NullPointerException

场景：数据库查询字段为 null（若用包装类接收，后续拆箱未判空则出问题）。

缓存机制导致的 “==” 判断误区：  
JVM 对 Byte、Short、Integer、Long（-128~127）、Character（0~127）做了缓存，valueOf() 会优先返回缓存对象；超出范围则新建对象。用 “==” 比较时，范围外的包装类会返回 false（因为比较的是地址）：

Integer a = 127;Integer b = 127;System.out.println(a == b); // true（用缓存对象，地址相同）

Integer c = 128;Integer d = 128;System.out.println(c == d); // false（新建对象，地址不同）System.out.println(c.equals(d)); // true（equals比较的是值，包装类重写了equals）

结论：包装类比较值必须用 equals()，不能用 ==。

接口和抽象类

| **特性** | **接口** | **抽象类** |
| --- | --- | --- |
| 继承 | 一个类可以实现多个接口 | 一个类只能继承一个抽象类 |
| 方法 | 只能有抽象方法（Java 8 前），Java 8 后可以有默认方法和静态方法 | 可以有抽象方法和具体方法 |
| 变量 | 只能有静态常量（public static final） | 可以有各种类型的成员变量 |
| 构造方法 | 没有构造方法 | 有构造方法，但不能直接实例化 |
| 访问修饰符 | 方法默认是 public，不能有其他修饰符 | 方法可以有各种访问修饰符 |

String s = new String("abc") 和 String s = "abc" 的区别？

JVM 处理方式：

String s = "abc"：JVM 先检查字符串常量池，若存在 "abc" 则直接返回引用；若不存在则在常量池创建 "abc"，再将引用赋值给s（仅可能创建 1 个对象）。

String s = new String("abc")：JVM 先在常量池创建 "abc"（若不存在），再在堆中创建新的 String 对象（内容与常量池的 "abc" 相同），最后将堆对象的引用赋值给s（最多创建 2 个对象：常量池 1 个 + 堆 1 个）。

核心区别：前者指向常量池对象，后者指向堆对象；前者可能复用常量池已有对象，后者每次调用都会在堆中生成新对象。

| 类型 | 本质作用 | 比较维度 | 典型场景 |
| --- | --- | --- | --- |
| == | 运算符 | 基本类型比 “值”，引用类型比 “内存地址” | 判断是否为同一个对象 |
| equals() | 方法（可重写） | 默认比 “内存地址”，重写后比 “内容”，不能比较基本类型 | 判断对象内容是否相同（如字符串） |
| hashCode() | 方法（需与equals()联动） | 返回对象的哈希码 | 哈希表（HashMap/HashSet）中定位对象 |

java异常机制

异常是程序运行时发生的非预期事件（如空指针、数组越界、文件不存在等），会中断正常执行流程。Java 中所有异常都是Throwable类的子类，主要分为两大类：

Error（错误）：  
表示 JVM 自身的严重问题（如内存溢出OutOfMemoryError），程序无法处理，通常不需要捕获。

Exception（异常）：  
程序可以处理的异常，又分为：

受检异常（Checked Exception）：编译期必须处理的异常（如IOException、ClassNotFoundException），不处理会编译报错。

非受检异常（Unchecked Exception）：运行时异常（RuntimeException及其子类），编译期不强制处理（如NullPointerException、ArrayIndexOutOfBoundsException）。

JDK版本

JDK5：解决 “代码繁琐、类型不安全”

泛型：强制集合 / 类的类型约束（如List<String>），避免运行时类型转换错误，让代码更安全。

注解：用@Override/@Deprecated等标记代码意图，减少人工注释的歧义，还能支持框架自动解析（如 Spring 的@Component）。

自动装箱 / 拆箱：无需手动写Integer.valueOf(1)或intValue()，简化基本类型与包装类的转换，减少冗余代码。

可变参数：用void method(int... args)替代 “数组传参”，简化多参数场景（如日志打印、批量处理）。

JDK8：开启 “函数式编程、高效数据处理”

Lambda 表达式：用(a,b)->a+b替代匿名内部类（如new Comparator<Integer>(){...}），大幅简化多线程、集合排序等代码，让逻辑更聚焦。

Stream API：用 “声明式” 代码处理集合（如list.stream().filter(x->x>10).collect(Collectors.toList())），替代嵌套的 for 循环，提升代码可读性和效率。

接口默认方法：允许接口新增方法时不强制所有实现类重写（如interface A { default void method() {} }），解决 “接口升级导致全量改代码” 的问题。

java.time 包：替代线程不安全的Date/Calendar，提供LocalDate/Instant等类，安全处理日期时间（如计算 “3 天后” 直接用LocalDate.now().plusDays(3)）。

JDK11：聚焦 “性能提升、开发便捷”

HTTP Client 正式版：完善 JDK 9 的预览功能，提供稳定的网络请求能力，无需依赖 OkHttp 等第三方库。

单文件运行：直接用java Hello.java运行单文件代码（无需先javac编译），简化小工具开发和测试流程。

ZGC 垃圾收集器：实现 “毫秒级停顿” 的垃圾回收，即使堆内存达 TB 级，也不影响高并发应用（如电商秒杀、金融交易）的响应速度。

String 增强：isBlank()判断空字符串（含全空格）、strip()去除前后空白（比trim()支持更多 Unicode 空白字符）、repeat(n)重复字符串，减少自定义工具类。

反射和映射

| **维度** | **反射（Reflection）** | **映射（Mapping）** |
| --- | --- | --- |
| 本质 | Java 的****运行时机制**** | 数据的****关联转换思想**** |
| 作用层面 | 操作类的结构（属性、方法、构造器） | 处理数据的结构对应与转换 |
| 典型场景 | 框架底层、动态代理、工具类 | ORM（对象 - 数据库）、OOM（对象 - 对象） |
| 依赖 | 基于 Java 原生 API（java.lang.reflect包） | 依赖框架 / 工具（如 MyBatis、MapStruct） |