Java 基础 第 1 阶段：基本语法——尚硅谷学习笔记（含面试题） 2023 年

第 1 章 Java 语言概述

1.1 Java 基础阶段的学习内容

1.1.1 第 1 阶段：Java 基本语法

Java 概述、关键字、标识符、变量、运算符、流程控制（条件判断、选择结构、循环结构）、IDEA、数组。

1.1.2 第 2 阶段：Java 面向对象编程

- 类及类的内部成员。
- 面向对象的三大特征： 封装、继承、多态。
- 其它关键字的使用。

1.1.3 第 3 阶段：Java 语言的高级应用

异常处理、多线程、IO 流、集合框架、反射、网络编程、新特性、其它常用的 API 等。

> 书籍推荐：《Java 核心技术》、《Effective Java》、《Java 编程思想》。

1.2 软件开发相关内容

1.2.1 计算机的构成

硬件 + 软件。

1.2.2 软件

软件，即一系列按照特定顺序组织的计算机数据和指令的集合。有系统软件和应用软件之分。

- 系统软件，即操作系统，Windows、Mac OS 、Linux、Android、ios。
- 应用软件，即 OS 之上的应用程序。

1.2.3 人机交互方式

- 图形化界面（Graphical User Interface，GUI）。
- 命令行交互方式（Command Line Interface，CLI）。

熟悉常用的 DOS（Disk Operating System，磁盘操作系统）命令：

- *进入和回退*
    - *盘符名称:*
      盘符切换。E:回车，表示切换到 E 盘。
    - *dir*
      列出当前目录下的文件以及文件夹。
    - *cd 目录*
      进入指定目录。
    - *cd ..*
      回退到上一级目录。
    - *cd \ 或 cd /*
      回退到盘符目录。
- *增、删*
    - *md 文件目录名*
      创建指定的文件目录。
    - *rd 文件目录名*
      删除指定的文件目录（如文件目录内有数据，删除失败）。
- *其它*
    - *cls*
      清屏。
    - *exit*
      退出命令提示符窗口。
    - *↑↓*
      调阅历史操作命令。

1.2.4 计算机编程语言

- 语言的分代：
    - 第 1 代：机器语言
    - 第 2 代：汇编语言

        - 第 3 代：高级语言
    - 面向过程的语言：C
    - 面向对象的语言：C++、Java、C#、Python、Go、JavaScript

没有“最好”的语言，只有在特定场景下相对来说，最适合的语言而已。

1.3 Java 概述

1.3.1 Java 发展史

- 几个重要的版本：1996 年，发布 JDK1.0
    - 里程碑式的版本：JDK5.0、JDK8.0（2014 年发布）
    - JDK11（LTS）、JDK17（LTS）long term support

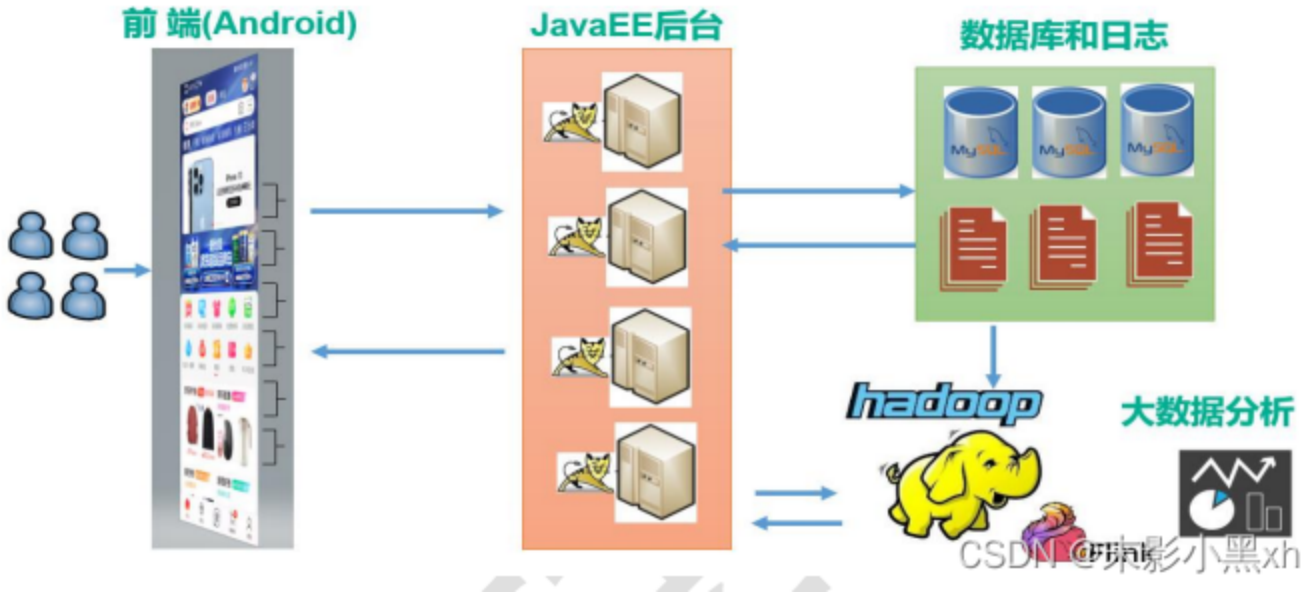
1.3.2 Java 之父

詹姆斯·高斯林

1.3.3 Java 具体的平台划分

- J2SE ——> JavaSE（Java Standard Edition）标准版
- J2EE ——> JavaEE（Java Enterprise Edition）企业版
- J2ME ——> JavaME（Java Micro Edition）小型版

Java 目前主要的应用场景：JavaEE 后端开发、Android 客户端的开发、大数据开发。



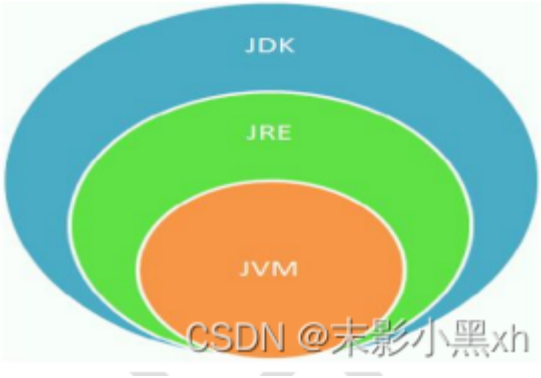
1.4. Java 环境的搭建

1.4.1 JDK、JRE、JVM 的基本概念

- JDK（Java Development Kit）：是 Java 程序开发工具包，包含 JRE 和开发人员使用的工具。
- JRE（Java Runtime Environment）：是 Java 程序的运行时环境，包含 JVM 和运行时所需要的核心类库。
- Java 虚拟机（Java Virtual Machine 简称 JVM）是运行所有 Java 程序的抽象计算机，是 Java 语言的运行环境，它是 Java 最具吸引力的特性之一。

1.4.2 JDK、JRE、JVM 三者之间的关系

- JDK = JRE + 开发工具集（例如 Javac 编译工具等）。
- JRE = JVM + Java SE 标准类库。



1.4.3 安装 JDK

- JDK 的下载：Oracle 官网（https://www.oracle.com）
- JDK 的安装：
  安装 jdk11 和 jdk17。
- 环境变量的配置：
  配置 JAVA_HOME + path。

1.5. HelloWorld

1.5.1 开发步骤

Java 程序开发三步骤：编写、编译、运行。

- 将 Java 代码编写到扩展名为 .java 的源文件中。
- 通过 javac.exe 命令对该 java 文件进行编译，生成一个或多个字节码文件。格式：
  javac 源文件名.java
- 通过 java.exe 命令对生成的 class 文件进行运行。格式： java 字节码文件名

1.5.2 开发说明

- 格式：

```
类
{
    方法
    {
        语句;
    }
}
```

- 代码：

```
 HelloWorld.java
```

```
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

- 说明：
  ① class：关键字，表示“类”，后面跟着类名。
  ② main() 方法的格式是固定的。表示程序的入口。
  ③ Java 程序，是严格区分大小写的。
  ④ 从控制台输出数据的操作：
  System.out.println() : 输出数据之后，会换行。
  System.out.print() : 输出数据之后，不会换行。
  ⑤ 每一行执行语句必须以;结束。
  ⑥ 编译以后，会生成 1 个或多个字节码文件。每一个字节码文件对应一个 Java 类，并且字节码文件名与类名相同。
  ⑦ 我们是针对于字节码文件对应的 Java 类进行解释运行的。
  ⑧ 一个源文件中可以声明多个类，但是最多只能有一个类使用 public 进行声明，且要求声明为 public 的类的类名与源文件名相同。

1.6 注释

源文件中用于解释、说明程序的文字就是注释。

1.6.1 Java 中的注释的种类

单行注释、多行注释、文档注释（Java 特有）。

1.6.2 单行注释

```
// 这是单行注释。
```

1.6.3 多行注释

```
/*
这是多行注释。
*/
```

1.6.4 文档注释

```
/**
 @author 指定 Java 程序的作者
 @version 指定源文件的版本
*/
```

文档注释内容可以被 JDK 提供的工具 javadoc 所解析，生成一套以网页文件形式体现的该程序的说明文档。

1.7 API 文档

API （Application Programming Interface，应用程序编程接口）是 Java 提供的基本编程接口。

Java 语言提供了大量的基础类，因此 Oracle 也为这些基础类提供了相应的说明文档，用于告诉开发者如何使用这些类，以及这些类里包含的方法。

Java API 文档，即为 JDK 使用说明书、帮助文档。

1.8 Java 核心机制：JVM

1.8.1 Java 语言的优点

- 跨平台性：
    - 原理：只要在需要运行 java 应用程序的操作系统上，先安装一个 Java 虚拟机(JVM ， Java Virtual Machine) 即可，由 JVM 来负责 Java 程序在该系统中的运行。
- 面向对象性：
  面向对象是一种程序设计技术，非常适合大型软件的设计和开发。面向对象编程支持封装、继承、多态等特性，让程序更好达到高内聚，低耦合的标准。
- 健壮性：
  吸收了 C/C++ 语言的优点，但去掉了其影响程序健壮性的部分（如指针、内存的申请与释放等），提供了一个相对安全的内存管理和访问机制。
    - 安全性高：
      Java 适合于网络/分布式环境，需要提供一个安全机制以防恶意代码的攻击。如：安全防范机制（ClassLoader 类加载器），可以分配不同的命空间以防替代本地的同名类、字节代码检查。
- 简单性：
  Java 就是 C++ 语法的简化版，我们也可以将 Java 称之为“C++ - -”。比如：头文件，指针运算，结构，联合，操作符重载，虚基类等。
    - 高性能：
      Java 最初发展阶段，总是被人诟病“性能低”；客观上，高级语言运行效率总是低于低级语言的，这个无法避免。Java 语言本身发展中通过虚拟机的优化提升了几十倍运行效率。比如，通过 JIT(JUST IN TIME)即时编译技术提高运行效率。

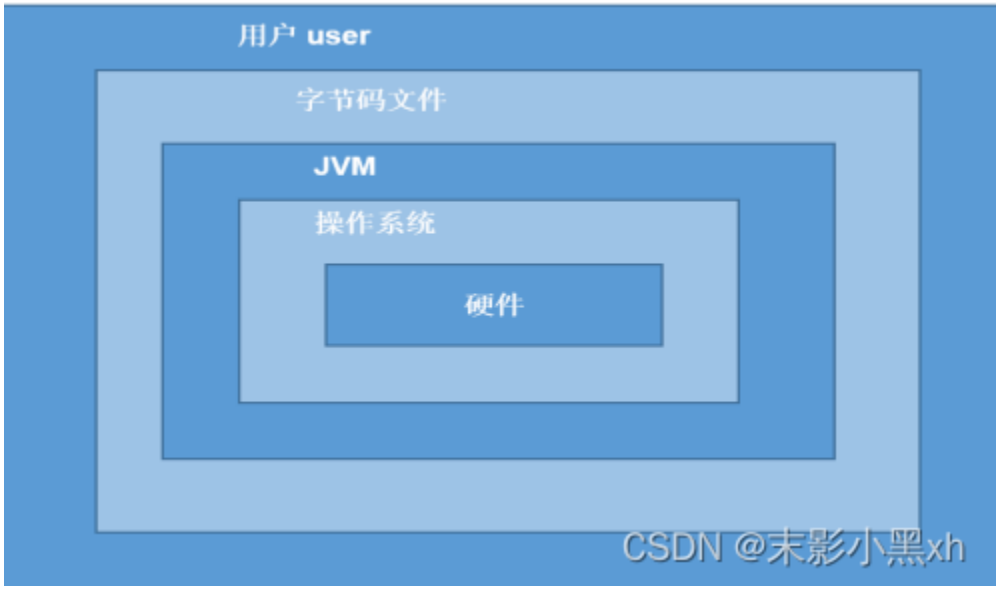
1.8.2 Java 语言的缺点

- 语法过于复杂、严谨，对程序员的约束比较多，与 python、php 等相比入门较难。
- 一般适用于大型网站开发，整个架构会比较重，对于初创公司开发和维护人员的成本比较高。

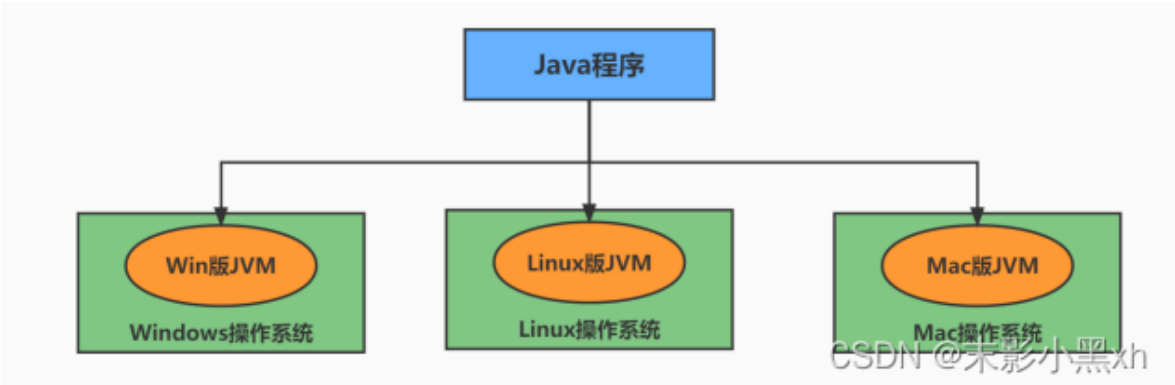
- 并非适用于所有领域。比如，Objective C、Swift 在 iOS 设备上就有着无可取代的地位；浏览器中的处理几乎完全由 JavaScript 掌控；Windows 程序通常都用 C++ 或 C#编写；Java 在服务器端编程和跨平台客户端应用领域则很有优势。

1.8.3 JVM 功能说明

JVM（Java Virtual Machine ，Java 虚拟机）：是一个虚拟的计算机，是 Java 程序的运行环境。JVM 具有指令集并使用不同的存储区域，负责执行指令，管理数据、内存、寄存器。



- 功能 1：实现 Java 程序的跨平台性 。我们编写的 Java 代码，都运行在 JVM 之上。正是因为有了 JVM，才使得 Java 程序具备了跨平台性。



- 功能 2：自动内存管理（内存分配、内存回收）
    - Java 程序在运行过程中，涉及到运算的数据的分配、存储等都由 JVM 来完成。
    - Java 消除了程序员回收无用内存空间的职责。提供了一种系统级线程跟踪存储空间的分配情况，在内存空间达到相应阈值时，检查并释放可被

释放的存储器空间。
    - GC 的自动回收，提高了内存空间的利用效率，也提高了编程人员的效率，很大程度上减少了因为没有释放空间而导致的内存泄漏。

1.9 企业真题

1. 一个“.java”源文件中是否可以包括多个类？有什么限制?
   答:
   一个源文件中可以声明多个类，但是最多只能有一个类使用 public 进行声明。且要求声明为 public 的类的类名与源文件名相同。
2. Java 的优势有哪些?
   答：跨平台型、安全性高、简单性、高性能、面向对象性、健壮性。
3. 常用的几个命令行操作都有哪些?
   答:
   ① 盘符名称:
   盘符切换。E:回车，表示切换到 E 盘。
   ② dir
   列出当前目录下的文件以及文件夹。
   ③ cd 目录
   进入指定目录。
   ④ cd ..
   回退到上一级目录。
   ⑤ cd \ 或 cd /
   回退到盘符目录。
   ⑥ md 文件目录名
   创建指定的文件目录。
   ⑦ rd 文件目录名*
   删除指定的文件目录（如文件目录内有数据，删除失败）。
   ⑧ cls
   清屏。
   ⑨ exit
   退出命令提示符窗口。
   ⑩ ↑↓
   调阅历史操作命令。
4. Java 中是否存在内存溢出、内存泄漏？如何解决？举例说明。

答：Java 中存在内存溢出和内存泄漏问题。

内存溢出指的是程序申请的内存超出了 JVM 所能分配的内存大小，导致程序崩溃。解决方法包括：

① 增加 JVM 内存限制：通过修改 JVM 启动参数，增加内存限制，例如 -Xmx。

② 优化代码：检查代码中是否存在大量的无用对象或者内存泄漏，及时释放资源。

③ 分析内存使用情况：使用工具分析内存使用情况，找到内存占用过多的地方，及时优化。

内存泄漏指的是程序中的对象在使用完毕后没有及时释放，导致内存占用不断增加。解决方法包括：

① 手动释放资源：在代码中手动释放资源，例如关闭文件、数据库连接等。

② 使用 try-with-resources：使用 try-with-resources 管理资源，确保资源被及时释放。

③ 检查代码：检查代码中是否存在内存泄漏的地方，及时优化。例如，如果在循环中创建对象并没有及时释放，就会导致内存泄漏。

5. 如何看待 Java 是一门半编译半解释型的语言?

答：Java 是一门半编译半解释型的语言，这意味着 Java 具有一定的优点和缺点。

优点：

① 跨平台性强：Java 的半编译半解释型特性使得它能够在不同的操作系统上运行，只需在不同平台上安装 Java 虚拟机即可。

② 安全性高：Java 的编译过程中会进行严格的类型检查和边界检查，减少了程序出错的可能性，同时 Java 还具有内存自动管理机制，防止了一些常见的安全漏洞。

③ 灵活性好：Java 的半编译半解释型特性使得它能够在运行时进行动态加载和更新，增强了程序的灵活性。

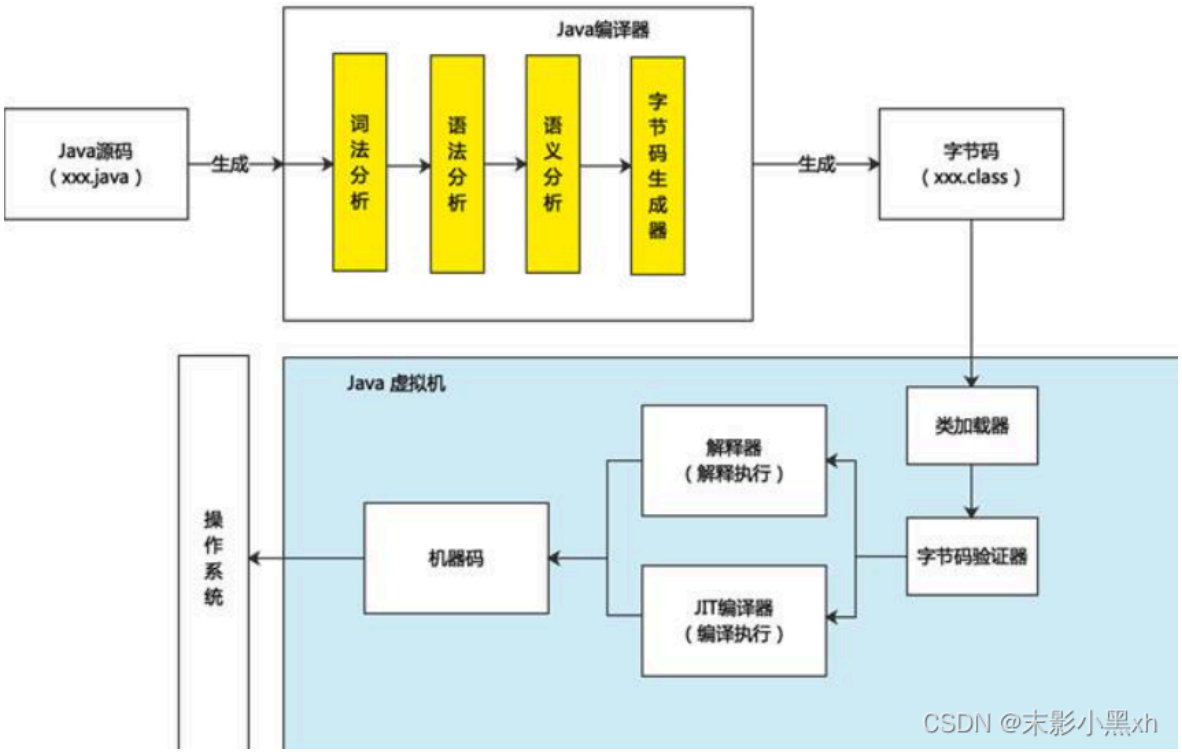
缺点：

① 执行效率低：由于 Java 是半编译半解释型的语言，需要在运行时进行解释和编译，导致执行效率较低。

② 内存占用大：Java 在运行时需要加载虚拟机和类库，占用的内存较大。

③ 资源消耗多：Java 的编译和解释过程需要占用较多的 CPU 和内存资源，对于一些资源受限的设备来说可能会造成困扰。

综上所述，Java 的半编译半解释型特性使得它具有跨平台性和安全性等优点，但也存在执行效率低、内存占用大和资源消耗多等缺点。在实际应用中需要根据具体情况进行权衡和选择。



第 2 章 变量与运算符

2.1 关键字、保留字

- 关键字：被 Java 赋予特殊含义的字符串。
  官方规范中有 50 个关键字，true、false、null 虽然不是关键字，但是可以当做关键字来看待。
- 保留字：goto、const

2.2 标识符

标识符：凡是可以自己命名的地方，都是标识符。
比如：类名、变量名、方法名、接口名、包名、常量名等。

标识符命名的规则：

- 由 26 个英文字母大小写，0 - 9，或 $ 组成。
- 数字不可以开头。

- 不可以使用关键字和保留字，但能包含关键字和保留字。
- Java 中严格区分大小写，长度无限制。
- 标识符不能包含空格。

标识符命名的规范：

- 包名：多个单词组成时所有字母都小写，
  例如：java.lang、com.MYXH.bean
- 类名、接口名：多个单词组成时，所有单词的首字母大写，例如：HelloWorld，String，System 等。
- 变量名、方法名：多个单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写，例如：age, name, bookName, main, binarySearch, getName 等。
- 常量名：所有字母都大写。多单词时每个单词用下划线连接，例如：MAX_VALUE, PI, DEFAULT_CAPACITY 等。

2.3 变量

变量的概念：内存中的一个存储区域，该区域的数据可以在同一类型范围内不断变化。

变量的构成包含三个要素：数据类型、变量名、存储的值。

Java 中变量声明的格式：数据类型 变量名 = 变量值

代码：

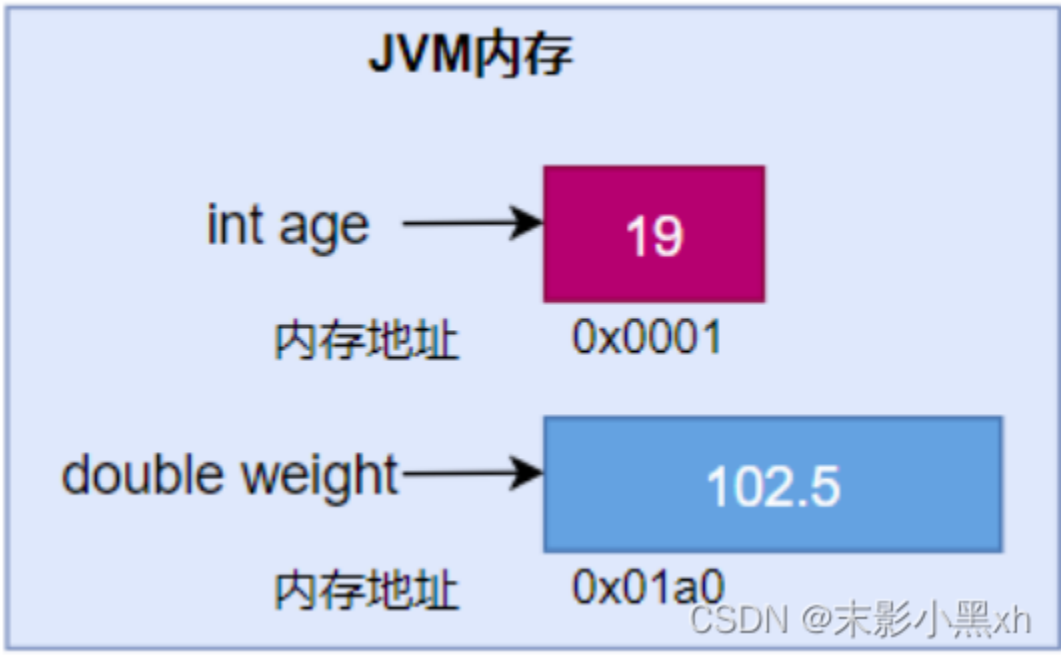
```
// 定义变量的方式1：
char gender;    // 过程1：变量的声明
gender = '男';    // 过程2：变量的赋值（或初始化）
// 定义变量的方式2：声明与初始化合并
int age = 21;

// 在同一个作用域内，不能声明两个同名的变量
// char gender = '女';

byte b1 = 127;
// b1 超出了 byte 的范围，编译不通过。
// b1 = 128;
```

说明：
定义变量时，变量名要遵循标识符命名的规则和规范。
① 变量都有其作用域。变量只在作用域内是有效的，出了作用域就失效了。
② 在同一个作用域内，不能声明两个同名的变量。
③ 定义好变量以后，就可以通过变量名的方式对变量进行调用和运算。
④ 变量值在赋值时，必须满足变量的数据类型，并且在数据类型有效的范围内变化。

变量的内存结构如图：



2.4 基本数据类型

2.4.1 基本数据类型

- 整型：byte、short、int、long
- 浮点型：float、double
- 字符型：char
- 布尔型：boolean

2.4.2 引用数据类型

- 类(class)
- 数组(array)

- 接口(interface)
- 枚举(enum)
- 注解(annotation)
- 记录(record)

如果在开发中，需要极高的精度，需要使用 BigDecimal 类替换浮点型变量。

2.5 基本数据类型变量间的运算规则

2.5.1 自动类型提升

规则：将取值范围小（或容量小）的类型自动提升为取值范围大（或容量大）的类型 。

2.5.2 强制类型转换

规则：

- 如果需要将容量大的变量的类型转换为容量小的变量的类型，需要使用强制类型转换。
- 强制类型转换需要使用强转符：()。在()内指明要转换为的数据类型。
- 强制类型转换过程中，可能导致精度损失。

2.6 基本数据类型与 String 的运算

2.6.1 字符串类型：String

String 类，属于引用数据类型，俗称字符串。

String 类型的变量，可以使用一对""的方式进行赋值。

String 声明的字符串内部，可以包含 0 个，1 个或多个字符。

2.6.2 运算规则

- String 与基本数据类型变量间只能做连接运算，使用“+”表示。
- 运算的结果是 String 类型。

- String 类型不能通过强制类型()转换，转为其他的类型。

2.7 进制的认识

2.7.1 进制的分类

二进制（以 0B、0b 开头）、十进制、八进制（以 0 开头）、十六进制（以 0x 或 0X 开头）。

2.7.2 二进制的理解

- 正数：原码、反码、补码三码合一。
- 负数：原码、反码、补码不相同。了解三者之间的关系。
    - 负数的原码：把十进制转为二进制，然后最高位设置为 1。
    - 负数的反码：在原码的基础上，最高位不变，其余位取反（0 变 1,1 变 0）。
    - 负数的补码：反码 + 1。
- 计算机数据的存储使用二进制补码形式存储，并且最高位是符号位。
    - 正数：最高位是 0。
    - 负数：最高位是 1。
- 熟悉：二进制与十进制之间的转换。
- 了解：二进制与八进制、十六进制间的转换。

2.8 运算符

运算符是一种特殊的符号，用以表示数据的运算、赋值和比较等。

- 运算符的分类：
    - 按照功能分为：算术运算符、赋值运算符、比较（或关系）运算符、逻辑运算符、位运算
      符、条件运算符、Lambda 运算符。
    - 按照操作数个数分为：一元运算符（单目运算符）、二元运算符（双目运算符）、三元运算符 （三目运算符）。

2.8.1 算术运算符

运算符	运算	范例	结果
+	正号	+3	3
-	负号	b=4; -b	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取模(取余)	7%5	2
++ ++	自增（前）：先运算后取值 自增（后）：先取值后运算	a=2;b=++a; a=2;b=a++;	a=3;b=3 a=3;b=2
- - - -	自减（前）：先运算后取值 自减（后）：先取值后运算	a=2;b=- -a a=2;b=a- -	a=1;b=1 a=1;b=2
+	字符串连接	“He”+”llo”	“Hello”

2.8.2 赋值运算符

- 符号：=
  - 当“=”两侧数据类型不一致时，可以使用自动类型转换或使用强制类型转换原则进行处理。
  - 支持连续赋值。
- 扩展赋值运算符： +=、 -=、*=、 /=、%=

2.8.3 比较（关系）运算符

运算符	运算	范例	结果
==	相等于	4==3	false
!=	不等于	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	true
instanceof	检查是否是类的对象	“Hello” instanceof String	true

2.8.4 逻辑运算符

a	b	a&b	a&&b	a|b	a||b	!a	a^b
true	true	true	true	true	true	false	false
true	false	false	false	true	true	false	true
false	true	false	false	true	true	true	true
false	false	false	false	false	false	true	false

逻辑运算符，操作的都是 boolean 类型的变量或常量，而且运算得结果也是 boolean 类型的值。

- 运算符说明：
  - & 和 &&：表示"且"关系，当符号左右两边布尔值都是 true 时，结果才能为 true。否则，为 false。
  - | 和 ||：表示"或"关系，当符号两边布尔值有一边为 true 时，结果为 true。当两边都为 false 时，结果为 false。
  - ！：表示"非"关系，当变量布尔值为 true 时，结果为 false。当变量布尔值为 false 时，结果为 true。
  - ^：当符号左右两边布尔值不同时，结果为 true。当两边布尔值相同时，结果为 false。
    理解：异或，追求的是“异”！

2.8.5 位运算符

位运算符		注意：无<<<
运算符	运算	范例
<<	左移	3 << 2 = 12 --> 3*2*2=12
>>	右移	3 >> 1 = 1 --> 3/2=1
>>>	无符号右移	3 >>> 1 = 1 --> 3/2=1
&	与运算	6 & 3 = 2
|	或运算	6 | 3 = 7
^	异或运算	6 ^ 3 = 5
~	取反运算	~6 = -7

位运算符的细节	
<<	空位补0，被移除的高位丢弃，空缺位补0。
>>	被移位的二进制最高位是0，右移后，空缺位补0； 最高位是1，空缺位补1。
>>>	被移位二进制最高位无论是0或者是1，空缺位都用0补。
&	二进制位进行&运算，只有1&1时结果是1，否则是0;
|	二进制位进行 | 运算，只有0 | 0时结果是0，否则是1;
^	相同二进制位进行 ^ 运算，结果是0；1^1=0 , 0^0=0 不相同二进制位 ^ 运算结果是1。1^0=1 , 0^1=1
~	正数取反，各二进制码按补码各位取反 负数取反，各二进制码按补码各位取反

- 左移：<<
  运算规则：在一定范围内，数据每向左移动一位，相当于原数据 * 2。
- 右移：>>
  运算规则：在一定范围内，数据每向右移动一位，相当于原数据 / 2。

2.8.6 条件运算符

- 条件运算符格式：(条件表达式)? 表达式 1 : 表达式 2

- 说明：
  ① 条件表达式的结果是 boolean 类型。
  ② 如果条件表达式的结果是 true，则执行表达式 1。否则，执行表达式 2。
  ③ 表达式 1 和表达式 2 需要是相同的类型或能兼容的类型。

2.8.7 运算符的优先级

- 如果想体现优先级比较高，使用()
- 我们在编写一行执行语句时，不要出现太多的运算符。

2.8.8 字符集

- 编码与解码：
  计算机中储存的信息都是用二进制数表示的，而我们在屏幕上看到的数字、英文、标点符号、汉字等字符是二进制数转换之后的结果。按照某种规则，将字符存储到计算机中，称为编码 。反之，将存储在计算机中的二进制数按照某种规则解析显示出来，称为解码 。
- 字符编码（Character Encoding）：就是一套自然语言的字符与二进制数之间的对应规则。
- 字符集：也叫编码表。是一个系统支持的所有字符的集合，包括各国家文字、标点符
  号、图形符号、数字等。
- ASCII 码、ISO-8859-1 字符集、GBxxx 字符集、Unicode 码、UTF-8。

2.9 企业真题

1. 怎么高效计算 2 * 8 的值?
   答：
   2 << 3 或 8 << 2
2. &和&&的区别?
   答：
   区分“&”和“&&”：
   相同点：如果符号左边是 true，则二者都执行符号右边的操作。
   不同点：
   & ： 如果符号左边是 false,则继续执行符号右边的操作。

&&：如果符号左边是 false,则不再继续执行符号右边的操作（短路与）。
建议：开发中，推荐使用 &&。

3. Java 中的基本类型有哪些？String 是最基本的数据类型吗?
   答:
   基本数据类型（8 种）。
   整型：byte、short、int、long
   浮点型：float、double
   字符型：char
   布尔型：boolean
   String 类，属于引用数据类型，俗称字符串。
4. Java 开发中计算金额时使用什么数据类型?
   答:
   不能使用 float 或 double，因为精度不高。
   使用 BigDecimal 类替换，可以实现任意精度的数据的运算。
5. char 型变量中能不能存储一个中文汉字，为什么?
   答:
   可以。char c1 = ‘中’；char c2 = ‘a’。
   因为 char 使用的是 unicode 字符集，包含了世界范围的所有的字符。
6. 代码分析。

```
short s1=1;
s1=s1+1;   // 有什么错?   答：= 右边是 int 类型，需要强转。
short s1=1;
s1+=1;     //有什么错?   答：没错。
```

7. int i = 0; i = i++ 执行这两句话后，变量 i 的值为?
   答:
   变量 i 的值为 0。
8. 如何将两个变量的值互换?
   String s1 = “abc”；
   String s2 = “123”；
   答:

```
String temp = s1;
s1 = s2;
s2 = temp;
```

9. boolean 占几个字节?
   答:
   在编译时不谈占几个字节。
   但是 JVM 在给 boolean 类型分配内存空间时，boolean 类型的变量占据一个槽位(slot，等于 4 个字节)。
   细节：true:1 false:0

   > 拓展:
   > 在内存中，byte、short、char、boolean、int、float : 占用 1 个 slot。
   > double、long :占用 2 个 slot。

10. 为什么 Java 中 0.1 + 0.2 结果不是 0.3?
    在代码中测试 0.1 + 0.2，你会惊讶的发现，结果不是 0.3，而是 0.3000……4。这是为什么?
    答:
    几乎所有现代的编程语言都会遇到上述问题，包括 JavaScript、Ruby、Python、Swift 和 Go 等。引发这个问题的原因是，它们都采用了 IEEE 754 标准。
    IEEE 是指“电气与电子工程师协会”，其在 1985 年发布了一个 IEEE 754 计算标准，根据这个标准，小数的二进制表达能够有最大的精度上限提升。但无论如何，物理边界是突破不了的，它仍然不能实现“每一个十进制小数，都对应一个二进制小数”。正因如此，产生了 0.1 + 0.2 不等于 0.3 的问题。
    具体的:
    整数变为二进制，能够做到“每个十进制整数都有对应的二进制数”，比如数字 3，二进制就是 11；再比如，数字 43 就是二进制 101011，这个毫无争议。
    对于小数，并不能做到“每个小数都有对应的二进制数字”。举例来说，二进制小数 0.0001 表示十进制数 0.0625 （至于它是如何计算的，不用深究）；二进制小数 0.0010 表示十进制数 0.125；二进制小数 0.0011 表示十进制数 0.1875。看，对于四位的二进制小数，二进制小数虽然是连贯的，但是十进制小数却不是连贯的。比如，你无法用四位二进制小数的形式表示 0.125 ~ 0.1875 之间的十进制小数。
    所以在编程中，遇见小数判断相等情况，比如开发银行、交易等系统，可以采用四舍五入或者“同乘同除”等方式进行验证，避免上述问题。

第 3 章 流程控制语句

3.1 流程控制结构

- 顺序结构
- 分支结构
    - if-else
    - switch-case
- 循环结构
    - for
    - while
    - do-while

3.2 分支结构之一：if-else

在程序中，凡是遇到了需要使用分支结构的地方，都可以考虑使用 if-else。

3.2.1 基本语法

- 结构 1：单分支条件判断：if

```
if(条件表达式)
{
    语句块;
}
```

- 结构 2：双分支条件判断：if...else

```
if(条件表达式)
{
    语句块 1;
}
else
{
    语句块 2;
}
```

- 结构 3：多分支条件判断：if...else if...else

```
if (条件表达式 1)
{
    语句块 1;
}
else if (条件表达式 2)
{
    语句块 2;
}

...

}
else if (条件表达式 n)
{
    语句块 n;
}
else
{
    语句块 n+1;
}
```

3.3 分支结构之二：switch-case

在特殊的场景下，分支结构可以考虑使用 switch-case。

3.3.1 基本语法

分支结构之 switch-case

```
switch(表达式)
{
    case 常量1:
      // 执行语句 1
      // break;
      case 常量2:
      // 执行语句 2
      // break;

    ...

    default:
      // 执行语句 n
      // break;
}
```

3.3.2 case 的穿透性

在 switch 语句中，如果 case 的后面不写 break，将出现穿透现象，也就是一旦匹配成功，不会在判断下一个 case 的值，直接向后运行，直到遇到 break 或者整个 switch 语句结束，执行终止。

3.4 循环结构之一：for

凡是循环结构，都有 4 个要素：
① 初始化条件
② 循环条件（是 boolean 类型）
③ 循环体
④ 迭代条件

3.4.1 基本语法

循环结构之一：for 循环

```
for(① 初始化条件;② 循环条件;④ 迭代条件)
{
    ③ 循环体
}
```

说明：

- 我们可以在循环结构中使用 break。一旦执行 break，就跳出（或结束）当前循环结构。
- 如何结束一个循环结构？
    - 方式 1：循环条件不满足。（即循环条件执行完以后是 false）
    - 方式 2：在循环体中执行了 break。
- 如果一个循环结构不能结束，那就是一个死循环！我们开发中要避免出现死循环。

3.5 循环结构之二：while

凡是循环结构，就一定会有 4 个要素：
① 初始化条件
② 循环条件（是 boolean 类型）
③ 循环体
④ 迭代部分

3.5.1 基本语法

循环结构之一：while 循环

```
① 初始化条件
while(② 循环条件)
{
    ③ 循环体
    ④ 迭代部分
}
```

3.6 循环结构之三：do-while

凡是循环结构，就一定会有 4 个要素：
① 初始化条件
② 循环条件（是 boolean 类型）
③ 循环体
④ 迭代部分

3.6.1 基本语法

循环结构之一：do-while 循环

```
① 初始化条件
do
{
    ③ 循环体
    ④ 迭代部分
}
while(② 循环条件);
```

3.7 “无限”循环

3.7.1 基本语法

```
while(true) {}  或  for(;;) {}
```

- 开发中，有时并不确定需要循环多少次，需要根据循环体内部某些条件，来控制循环的结束（使用 break）。
- 如果此循环结构不能终止，则构成了死循环！开发中要避免出现死循环。

3.8 关键字 break、continue

break 和 continue 关键字的使用

- 使用范围：在循环结构中的作用
- 相同点：
    - break：循环结构中结束（或跳出）当前循环结构。
    - continue：循环结构中结束（或跳出）当次循环 。

3.9 Scanner 类的使用

如何从键盘获取不同类型（基本数据类型、String 类型）的变量：使用 Scanner 类。

键盘输入代码的四个步骤：
① 导包：import java.util.Scanner;
② 创建 Scanner 类型的对象：Scanner scan = new Scanner(System.in);
③ 调用 Scanner 类的相关方法（next()、nextXxx()），来获取指定类型的变量。
④ 释放资源：scan.close();

3.10 获取随机数

如何获取一个随机数?
① 可以使用 Java 提供的 API:Math 类的 random() 。
② random() 调用以后，会返回一个[0.0,1.0)范围的 double 型的随机数。

3.11 企业真题

1. break 和 continue 的作用?
   答：
   使用范围：在循环结构中的作用。
   相同点：
   break：循环结构中结束（或跳出）当前循环结构。
   continue：循环结构中结束（或跳出）当次循环
2. if 分支语句和 switch 分支语句的异同之处?
   答：
   if-else 语句优势：
   if 语句的条件是一个布尔类型值，if 条件表达式为 true 则进入分支，可以用于范围的判断，也可以用于等值的判断，使用范围更广。
   switch 语句的条件是一个常量值（byte、short、int、char、枚举、String），只能判断某个变量或表达式的结果是否等于某个常量值，使用场景较狭窄。
   switch 语句优势：
   当条件是判断某个变量或表达式是否等于某个固定的常量值时，使用 if 和 switch 都可以，习惯上使用 switch 更多。因为效率稍高。
   当条件是区间范围的判断时，只能使用 if 语句。
   使用 switch 可以利用穿透性，同时执行多个分支，而 if-else 没有穿透性。
3. switch 语句中忘写 break 会发生什么?

答：
如果在 switch 语句中忘记写 break，程序将会继续执行下一个 case 语句，直到遇到 break 或者 switch 语句结束。这种情况被称为“穿透”（fall-through），因为程序“穿透”了一个 case 语句并继续执行下一个 case 语句。这可能会导致程序出现意外行为，因为程序可能会执行不应该执行的代码。因此，在编写 switch 语句时，应该始终记得写上 break 来避免出现这种情况。

4. Java 支持哪些类型循环?
   答：
   for；while；do-while；
   增强 for 循环（for-each）。
5. while 和 do while 循环的区别?
   答：
   while 循环和 do while 循环都是用于重复执行某个代码块的结构，但它们之间存在一些区别：
   ① while 循环是先判断条件是否成立，再决定是否执行循环体，如果条件不成立，则一次都不执行；而 do while 循环是先执行一次循环体，再判断条件是否成立，所以至少会执行一次循环体。
   ② while 循环的循环体可能一次都不执行，因为条件不成立；而 do while 循环的循环体至少会执行一次。
   ③ while 循环是入口判断循环，即在循环开始前就判断条件是否成立；而 do while 循环是出口判断循环，即在循环结束后判断条件是否成立。
   ④ 在循环条件不成立的情况下，while 循环不会执行循环体，而 do while 循环会执行一次循环体。
   总的来说，while 循环适合在条件不成立时不需要执行循环体的情况下使用；而 do while 循环适合在至少需要执行一次循环体的情况下使用。

第 4 章 IDEA 的安装与使用

4.1 IDEA 的认识

- IDEA（集成功能强大、符合人体工程学）
- Eclipse

4.2 IDEA 的下载、安装、卸载

- 卸载：使用控制面板进行卸载，注意删除 C 盘指定目录下的两个文件目录：jetbrains。
- 下载：从官网IDEA 官网https://www.jetbrains.com/进行下载：旗舰版。
- 安装：傻瓜式的安装、注册。

4.3 工程结构

- project(工程)、module(模块)、package(包)、package(包)等概念。
- 掌握：如何创建工程、如何创建模块、如何导入其他项目中的模块、如何创建包、如何创建类、如何运行。

4.4 企业真题

1. 开发中你接触过的开发工具都有哪些?
   答:
   IDEA、Visual Studio Code、Eclipse。
2. 谈谈你对 Eclipse 和 IDEA 使用上的感受?
   答:
   IDEA 集成功能强大、符合人体工程学，Eclipse 不够人性化。

第 5 章：数组

5.1 数组的概述

数组(Array)：就可以理解为多个数据的组合。

程序中的容器：数组、集合框架（List、Set、Map）。

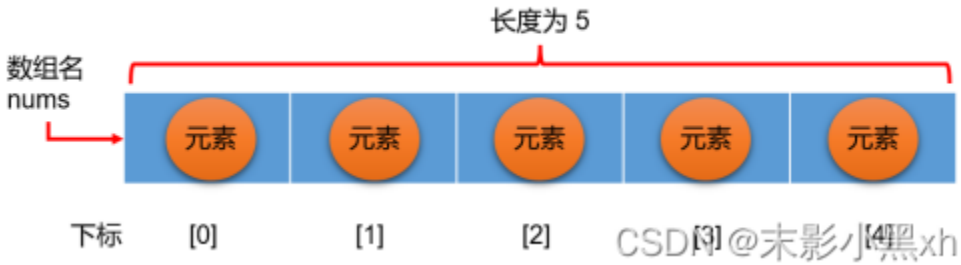
数组中的概念：

- 数组名

- 下标（或索引）
- 元素
- 数组的长度

数组存储的数据的特点：

- 依次紧密排列的、有序的、可以重复的。
- 数组的其它特点：
    - 一旦初始化，其长度就是确定的、不可更改的。



5.2 一维数组

5.2.1 数组的声明和初始化

代码示例：

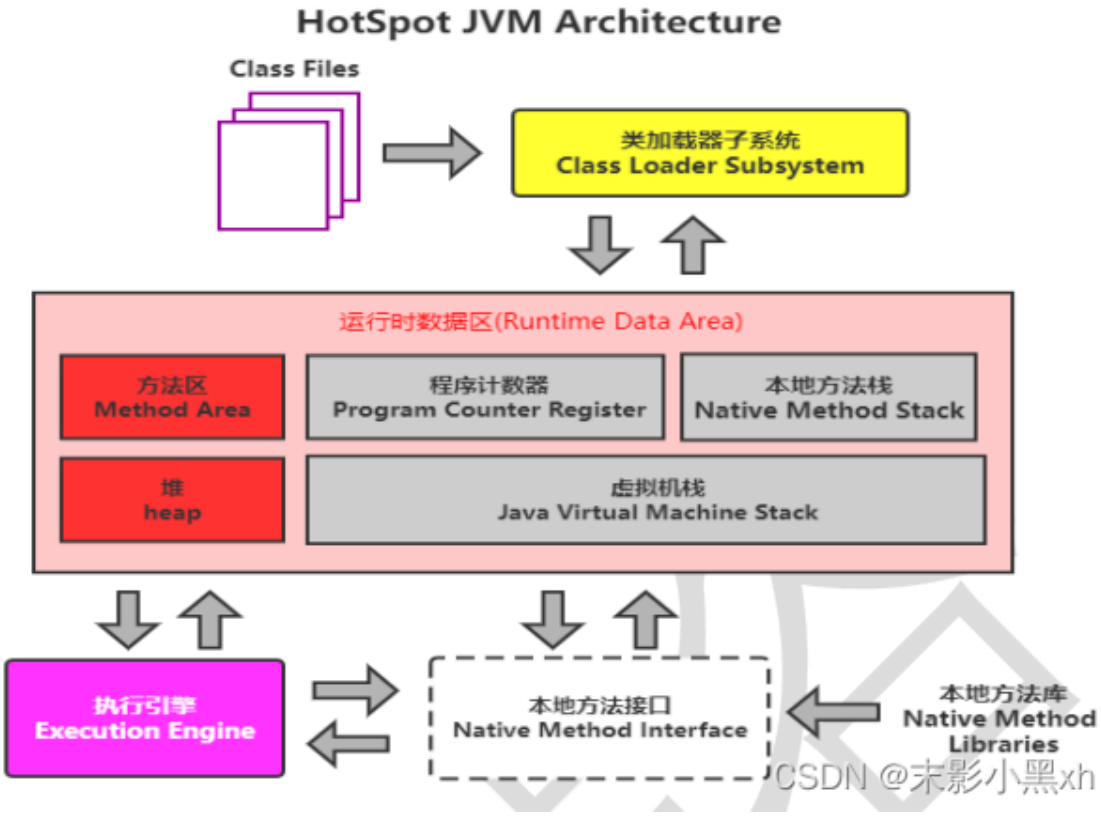
```
int[] arr1 = new int[10];
String[] arr2 = new String[]{"Tom","Jerry"};
```

5.2.2 数组的使用

- 调用数组的指定元素：使用角标、索引、index。
- index 从 0 开始。
- 数组的属性：length，表示数组的长度。
- 数组的遍历。
- 数组元素的默认初始化值。

5.2.3 一维数组内存分析

- 虚拟机栈：main() 作为一个栈帧，压入栈空间中。在 main() 栈帧中，存储着 arr 变量。arr 记录着数组实体的首地址值。
- 堆：数组实体存储在堆空间中。
- Java 虚拟机的内存划分：



5.3 二维数组

二维数组本质上是元素类型是一维数组的一维数组。

5.4 数组的常用算法

- 数值型数组的特征值的计算：最大值、最小值、总和、平均值等。
- 数组元素的赋值。
- 数组的复制、赋值。
- 数组的反转。
- 数组的扩容、缩容。

- 数组的查找：
    - 线性查找。
    - 二分法查找（前提：数组有序）。
- 数组的排序：
    - 冒泡排序（最简单）。
    - 快速排序（最常用）。

5.5 Arrays 工具类的使用

- java.util.Arrays 类即为操作数组的工具类，包含了用来操作数组（比如排序和搜索）的各种方法。
- toString() 、 sort()、 binarySearch()。

5.6 数组中的常见异常

- 下标越界异常：ArrayIndexOutOfBoundsException
- 空指针异常：NullPointerException

5.7、企业真题

1. 数组有没有 length()这个方法? String 有没有 length()这个方法?
   答：
   数组没有 length()，有 length 属性。
   String 有 length()。
2. 有数组 int[] arr，用 Java 代码将数组元素顺序颠倒?
   答：
   可以使用两个指针，一个指向数组的第一个元素，另一个指向数组的最后一个元素，交换它们的值，然后继续向中间靠拢，直到两个指针相遇。

```java
public static void reverseArray(int[] arr)
{
    int left = 0;
    int right = arr.length - 1;

    while (left < right)
    {
        int temp = arr[left];

        arr[left] = arr[right];
        arr[right] = temp;
        left++;
        right--;
    }
}

public static void main(String[] args)
{
    int[] arr = {1, 2, 3, 4, 5};    // 定义一个数组

    System.out.println("原数组: " + Arrays.toString(arr));    // 输出原数组
    reverseArray(arr);    // 调用方法将数组元素顺序颠倒
    System.out.println("颠倒后的数组: " + Arrays.toString(arr));    // 输出颠倒后的数组
}
```

运行该主函数，输出结果如下：

```
原数组: [1, 2, 3, 4, 5]
颠倒后的数组: [5, 4, 3, 2, 1]
```

3. 为什么数组要从 0 开始编号，而不是 1？
   答：
   数组的索引，表示了数组元素距离首地址的偏离量。因为第 1 个元素的地址与首地址相同，所以偏移量就是 0，所以数组要从 0 开始。
4. 数组有什么排序的方式，手写一下？
   答：
   常见的数组排序方式有冒泡排序、选择排序、插入排序、快速排序、归并排序等。
   冒泡排序：
   冒泡排序的思路是从第一个元素开始，依次比较相邻的两个元素，如果前一个元素比后一个元素大，则交换它们的位置。这样一轮下来，最大的元素就会被移动到最后一个位置。然后再从第一个元素开始，继续进行比较和交换，直到所有元素都被

排序。

```
public class BubbleSort
{
    public static void main(String[] args)
    {
        int[] arr = {3, 9, 1, 8, 2, 5, 7};

        bubbleSort(arr);

        for(int i = 0; i < arr.length; i++)
        {
            System.out.print(arr[i] + " ");
        }
    }

    public static void bubbleSort(int[] arr)
    {
        int n = arr.length;

         for(int i = 0; i < n - 1; i++)
         {
             for(int j = 0; j < n - i - 1; j++)
             {
                 if(arr[j] > arr[j + 1])
                 {
                     int temp = arr[j];
                     arr[j] = arr[j + 1];
                     arr[j + 1] = temp;
                 }
             }
         }
    }
}
```

冒泡排序的时间复杂度为 O(n^2)，空间复杂度为 O(1)。

快速排序：

快速排序的思路是选取一个基准元素，将数组分为左右两部分，左半部分的元素均小于等于基准元素，右半部分的元素均大于等于基准元素。然后对左右两部分分别进行快速排序，直到整个数组有序。在上面的代码中，partition 方法用于实现分区，将数组分为左右两部分。quickSort 方法用于实现快速排序，递归调用自身对左右两部分进行排序。

```java
public class QuickSort
{
    public static void main(String[] args)
    {
        int[] arr = {5, 2, 9, 3, 7, 6, 1, 8, 4};

        quickSort(arr, 0, arr.length - 1);

        for (int i : arr)
        {
            System.out.print(i + " ");
        }
    }

    public static void quickSort(int[] arr, int left, int right)
    {
        if (left < right)
        {
            int pivot = partition(arr, left, right);

            quickSort(arr, left, pivot - 1);
            quickSort(arr, pivot + 1, right);
        }
    }

    public static int partition(int[] arr, int left, int right)
    {
        int pivot = arr[left];

        while (left < right)
        {
            while (left < right && arr[right] >= pivot)
            {
                right--;
            }

            arr[left] = arr[right];

            while (left < right && arr[left] <= pivot)
            {
                left++;
            }

            arr[right] = arr[left];
```

```
        }

        arr[left] = pivot;

        return left;
    }
}
```

快速排序的时间复杂度为 O(nlogn)，空间复杂度为 O(logn)。

5. 二分算法实现数组的查找?

   答:

   二分查找思路:

   ① 首先确定要查找的数组的范围，即左右边界;

   ② 计算中间位置，即中间索引值;

   ③ 判断中间值是否等于要查找的值，如果是，则返回中间索引值;

   ④ 如果中间值大于要查找的值，则在左半部分继续查找，即将右边界设为中间索引值减一;

   ⑤ 如果中间值小于要查找的值，则在右半部分继续查找，即将左边界设为中间索引值加一;

   ⑥ 重复 ②-⑤ 步骤，直到找到要查找的值或左右边界重合，此时返回-1 表示未找到。

```java
public class BinarySearch
{
    public static int binarySearch(int[] arr, int key)
    {
        int low = 0;
        int high = arr.length - 1;

        while (low <= high)
        {
            int mid = (low + high) / 2;

            if (key < arr[mid])
            {
                high = mid - 1;
            }
            else if (key > arr[mid])
            {
                low = mid + 1;
            }
            else
            {
                return mid;
            }
        }

        return -1;
    }

    public static void main(String[] args)
    {
        int[] arr = {1, 3, 5, 7, 9};
        int key = 3;
        int index = binarySearch(arr, key);

        if (index == -1)
        {
            System.out.println("找不到指定的元素");
        }
        else
        {
            System.out.println("指定元素的索引为: " + index);
        }
    }
}
```

复杂度分析：
时间复杂度为 O(log n)，因为每次查找都将查找范围缩小一半，最坏情况下需要查找 log n 次，其中 n 为数组长度。
空间复杂度为 O(1)，因为只需要常数个额外变量存储查找范围的左右边界和中间索引值。

6. 怎么求数组的最大子序列和?
   答：
   以下是一个使用 Java 实现的求解最大子序列和的示例代码：
   这个算法的思路是使用动态规划的思想。
   我们从左到右遍历整个数组，使用两个变量 maxSum 和 currentSum 来记录最大子序列和和当前子序列和。
   对于当前遍历到的元素 nums[i]，我们可以有两种选择：
   将 nums[i] 加入当前子序列中，即 currentSum = currentSum + nums[i]；
   以 nums[i] 作为新的起点开始一个新的子序列，即 currentSum = nums[i]。
   我们需要比较这两种选择哪个更优，即选择 currentSum + nums[i] 或选择 nums[i] 中的较大值作为当前子序列的和 currentSum。同时，我们需要比较当前子序列的和 currentSum 和最大子序列和 maxSum 哪个更大，即选择 Math.max(maxSum, currentSum)作为新的最大子序列和 maxSum。
   最后，遍历完成后 maxSum 就是最大子序列和。

```java
public class MaxSubArraySum
{
    public static int maxSubArraySum(int[] nums)
    {
        int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.length; i++)
        {
            currentSum = Math.max(currentSum + nums[i], nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }

        return maxSum;
    }

    public static void main(String[] args)
    {
        int[] nums = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        int maxSum = maxSubArraySum(nums);

        System.out.println("最大子序列和为: " + maxSum);
    }
}
```

输出：

```
最大子序列和为: 6
```

解释：最大子序列为[4, -1, 2, 1]，和为 6。

7. Arrays 类的排序方法是什么？如何实现排序的?

   答：

   Arrays 类提供了多种排序方法，包括：

   ① sort(Object[] a)：对数组 a 进行升序排序，元素类型必须实现 Comparable 接口。

   ② sort(Object[] a, Comparator c)：对数组 a 进行排序，使用自定义的 Comparator 比较器进行比较。

   ③ parallelSort(Object[] a)：对数组 a 进行并行排序，效率更高。

   排序的实现原理主要是基于快速排序和归并排序，具体实现方式根据元素类型和排序方法不同而不同。

在 sort(Object[] a) 方法中，对于实现了 Comparable 接口的元素类型，通过 compareTo() 方法进行比较，并且使用快速排序实现；对于未实现 Comparable 接口的元素类型，则会抛出 ClassCastException 异常。
在 sort(Object[] a, Comparator c) 方法中，通过传入自定义的 Comparator 比较器进行比较，也使用快速排序实现。
在 parallelSort(Object[] a) 方法中，使用 Fork/Join 框架实现并行排序，将数组拆分成多个小数组进行排序，最后再合并起来。