Java的特性

▼ Java是 强类型 , 静态编译 的语言

强类型 Strongly Typed：

> 变量必须先声明类型，且类型不能随意改变，类型不匹配的操作会直接报错

- 每一个变量都必须有**明确的数据类型**，比如：`int`, `String`, `boolean`
- 不允许**隐式的类型转换**，比如不能把一个 `String` 自动当作 `int` 使用
- 类型不匹配时，**编译阶段**就会报错

✅ 示例：

```
int number = 10;
number = "hello";  // ❌ 错误！String类型不能赋值给int
```

而在像 Python 这样的弱类型语言中：

```
x = 10
x = "hello"  # ✅ 合法，x 的类型可以随时改变
```

静态编译 Statically Compiled

> 程序在运行之前，编译器会把代码编译成字节码/机器码, 并在这个过程中进行类型检查和语法检查

Java 的静态编译过程：

1. 写好 `.java` 文件
2. 用 `javac` 编译器编译为 `.class` 字节码文件
3. JVM（Java 虚拟机）在运行时解释 / 即时编译这些字节码

✅ 特点：

- 编译阶段就能发现大部分语法和类型错误（安全性高）
- 程序运行前必须编译（不同于 Python 直接解释运行）

🔍 总结一句话：

> Java 被称为强类型静态编译语言，因为它在编写时要求变量必须声明类型（强类型），并且在运行前需要先经过编译器检查和转换为字节码（静态编译），这让它在程序运行前就能发现很多错误，提升了代码的安全性和稳定性

▼ Java 的跨平台性能

📦 第一步：计算机的底层是怎么运行程序的？

1.1 什么是机器码（Machine Code）？

- 是计算机唯一能直接“看懂”和“执行”的语言
- 机器码由 **0 和 1** 组成（二进制），每一条机器码指令控制 CPU 干一件事，比如加法、跳转、读内存等

举个例子：

```
10110000 01100001 ⇒ 表示“把数字 97 放进寄存器”
```

1.2 操作系统的角色：

- 你不可能直接写机器码，太难了！
- 所以需要一个中间人：**操作系统（OS）**

- 操作系统帮你管理 CPU、内存、磁盘等硬件资源
- 它本身就是一套运行在机器码之上的“控制系统”

1.3 应用程序怎么运行？

你写的程序（比如 Java、C）不能直接运行，必须：

1. **被编译成机器码**（比如 C/C++）
2. 或 **通过某种“运行环境”翻译成机器码**（比如 Java）

🧱 第二步：Java 不直接编译成机器码，而是编译成字节码（Bytecode）

2.1 什么是字节码？

- 是一种**中间语言**，介于源码和机器码之间
- Java 编译器（`javac`）把 `.java` 文件变成 `.class` 字节码文件
- 字节码不是操作系统能直接运行的，但它是 **跨平台的**

🧱 第三步：JVM（Java Virtual Machine）

3.1 什么是 JVM？

> JVM 是一种虚拟的计算机，可以理解字节码，把它翻译成机器码，再交给操作系统去运行

🌍 假设写了一个 Java 程序：

```
System.out.println("Hello World!");
```

你在 Windows 上运行没问题，现在你把这个 `.java` 文件：

1. 用 `javac` 编译成 `HelloWorld.class` 字节码文件

2. 然后你复制这个 `.class` 文件到：
    - macOS 电脑
    - Linux 服务器
    - Android 手机（特例，有自己的虚拟机）

✅ 只要这些设备上装了对应的 JVM，程序就能运行，**不需要重新编译，不需要改代码** 👉这就是跨平台

🔧 为什么能跨平台：

1. Java 编译器（javac）

把 `.java` 源代码 ➜ 编译成 `.class` 字节码文件（跟平台无关）

2. Java 虚拟机（JVM）

每个平台都有专门的 JVM，比如：

- Windows 的 JVM
- macOS 的 JVM
- Linux 的 JVM

这个 JVM 会把 Java 的字节码转成平台自己的机器码来执行

所以只要你换平台，就换 JVM，不用动代码

🔁 类比：JVM 就像一层“中间平台”

层级	内容	比喻
你写的 Java 程序	HelloWorld.java	英文手稿
编译后的字节码	HelloWorld.class	通用翻译稿（字节码）
JVM	Java 虚拟机	各国翻译官（JVM for Windows, Linux, macOS）
操作系统	Windows/Linux/macOS	不同国家
机器码 & 硬件	CPU、内存	真正的电路、机器

每个平台上都可以有一个自己的“JVM 翻译官”，你写一次 Java 程序，**所有平台的 JVM 都能翻译成各自的机器码来运行**

这就是“**一次编写，到处运行**”的实现机制

🧠 从底层到 JVM：

```
CPU（硬件） ← 只能执行机器码（0/1）

↑ 操作系统（OS）
  - 管理 CPU、内存、磁盘
  - 接收来自程序的机器码请求

↑ Java 虚拟机（JVM）
  - 加载字节码（.class）
  - 翻译成机器码交给 OS 执行
  - 自动管理内存、线程等

↑ Java 字节码（.class）
  - 由 javac 编译器生成

↑ Java 源码（.java）
  - 开发者编写
```
