

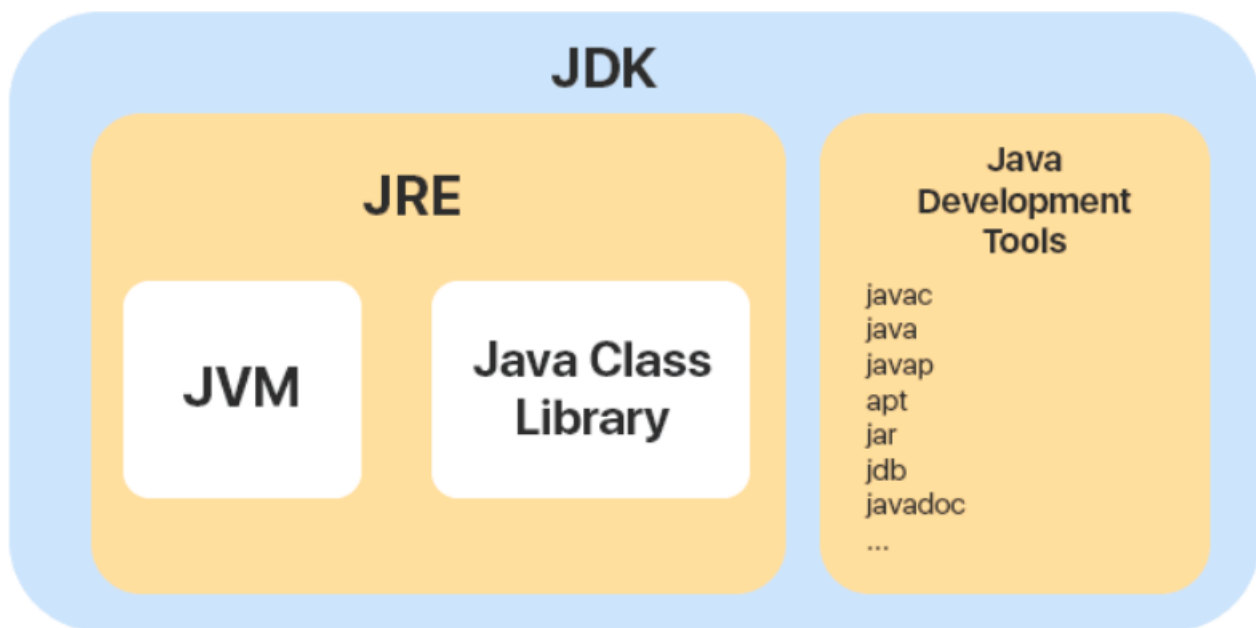
第01章：Java语言概述

1. 开发环境搭建

1.1. JVM、JRE和JDK之间的区别

1.1.1. 区别

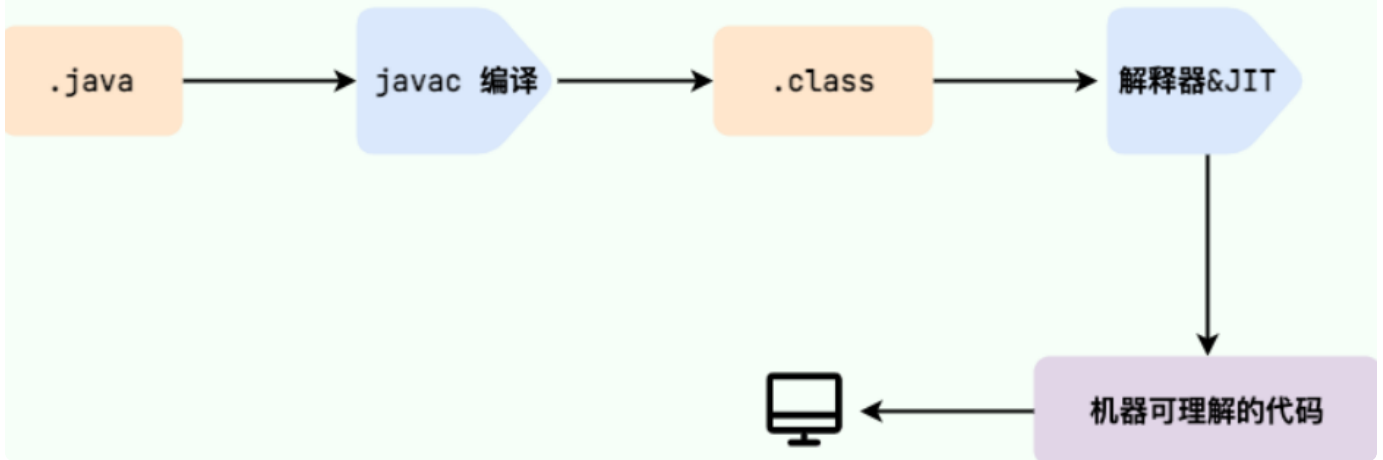
1. JVM是运行Java字节码的虚拟机, 针对不同的系统有不同的实现, 作用是将字节码文件翻译成适合操作系统运行的程序文件.
2. JRE是Java运行时环境, 包括JVM和基础类库, 作用是运行已编译的Java程序.
3. JDK是Java开发工具包, 包含JRE和编译Java源码的编译器Javac以及其他一些工具(如, javadoc文档注释工具, jdb调试器), 作用是创建, 编译和运行Java程序.



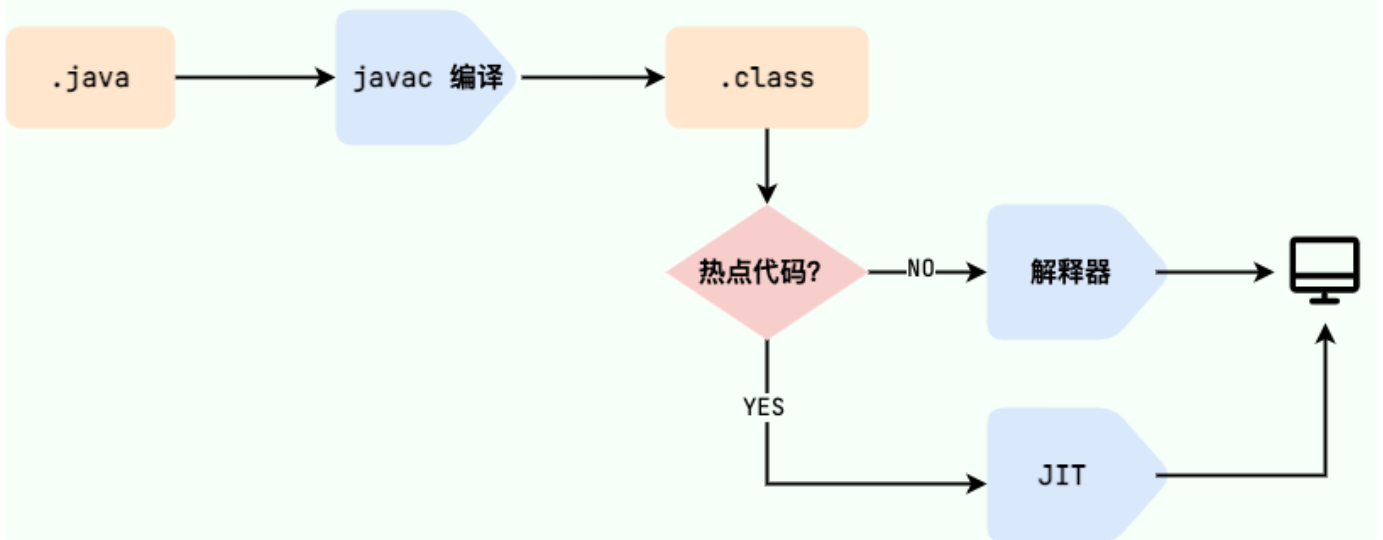
1.1.2. 字节码文件

字节码文件是不面向任何特定的操作系统, 只面向JVM的文件, 简单来说就是JVM可以理解的文件. 扩展名为 `.class` 文件.

由于字节码文件不针对一种特定的操作系统, 所以Java编译一次字节码后便可在不同的OS上运行.



我们需要格外注意的是 .class→机器码 这一步。在这一步 JVM 类加载器首先加载字节码文件，然后通过解释器逐行解释执行，这种方式的执行速度会相对比较慢。而且，有些方法和代码块是经常需要被调用的(也就是所谓的热点代码)，所以后面引进了 JIT (just-in-time compilation) 编译器，而 JIT 属于运行时编译。当 JIT 编译器完成第一次编译后，其会将字节码对应的机器码保存下来，下次可以直接使用。而我们知道，机器码的运行效率肯定是高于 Java 解释器的。这也解释了我们为什么经常会说 **Java 是编译与解释共存的语言**。

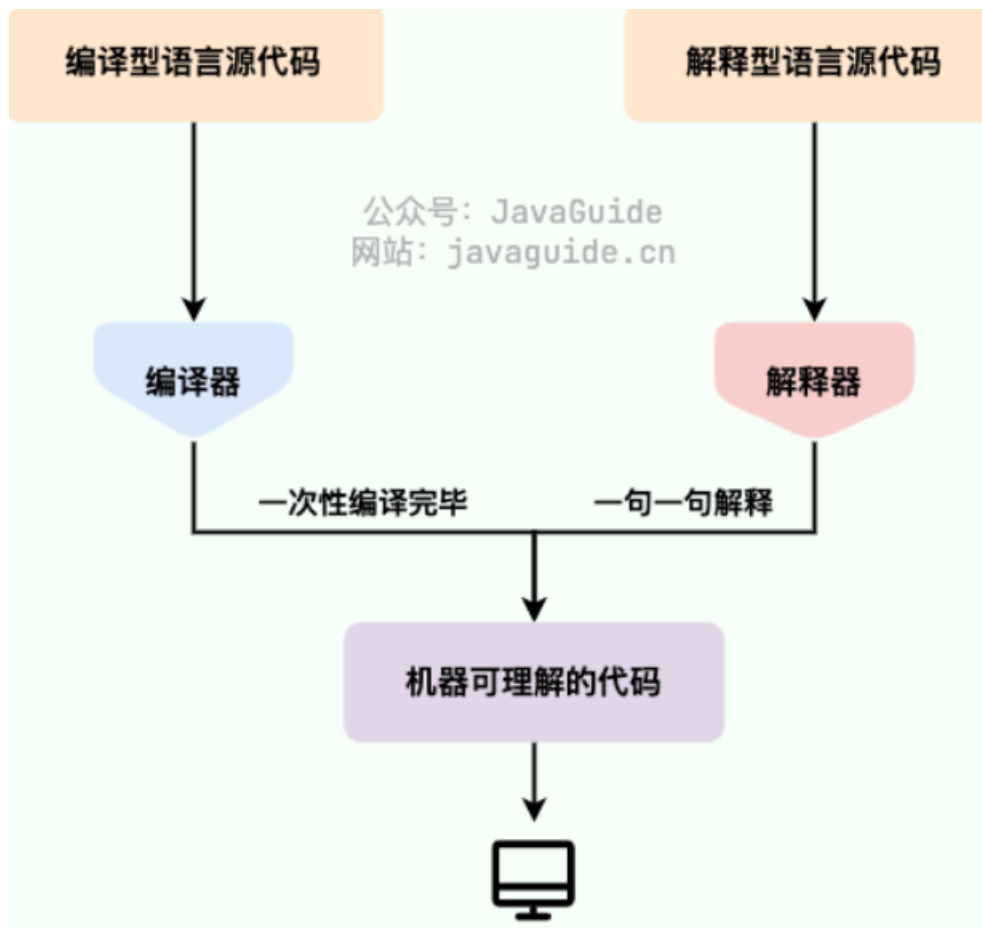


1.1.3. 为什么说 Java 语言“编译与解释并存”？

我们可以将高级编程语言按照程序的执行方式分为两种：

1. **编译型**：编译型语言会通过编译器将源代码一次性翻译成可被该平台执行的机器码。一般情况下，编译语言的执行速度比较快，开发效率比较低。常见的编译性语言有 C、C++、Go、Rust 等等。

2. **解释型**：[解释型语言open in new window](#)会通过[解释器open in new window](#)一句一句的将代码解释（interpret）为机器代码后再执行。解释型语言开发效率比较快，执行速度比较慢。常见的解释性语言有 Python、JavaScript、PHP 等等。



因为 Java 语言既具有编译型语言的特征，也具有解释型语言的特征。因为 Java 程序要经过先编译，后解释两个步骤，由 Java 编写的程序需要先经过编译步骤，生成字节码（.class 文件），这种字节码必须由 Java 解释器来解释执行。

1.2. JDK安装

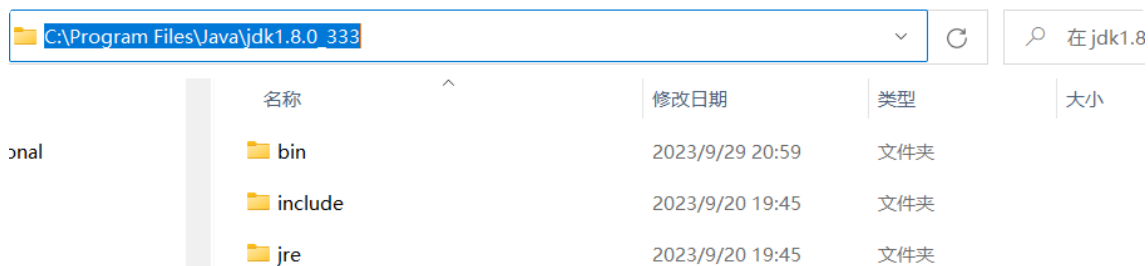
1.2.1. JDK8安装

1.2.1.1. JDK8软件

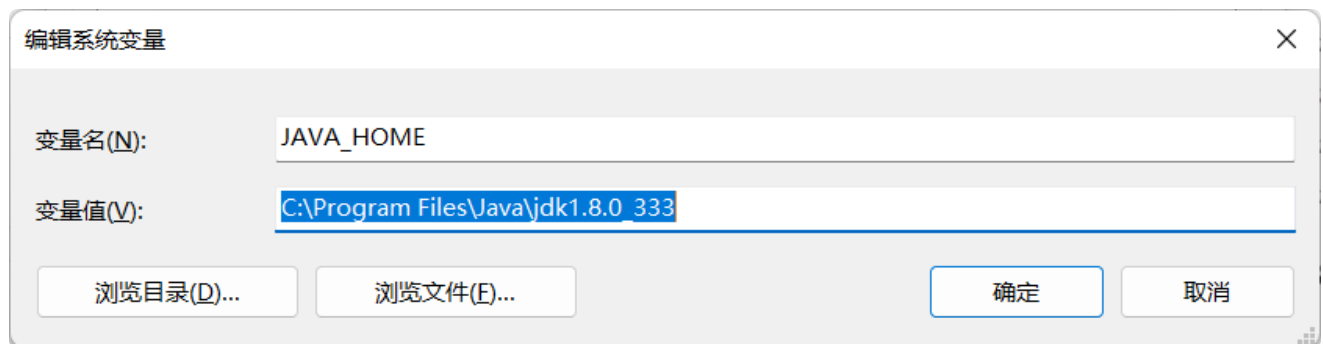
软件包地址：`C:\Users\20961\Documents\WPSTDrive\418342908\WPS云盘\myFiles\编程\常用文件\Java\JavaSE`

1.2.1.2. 配置环境变量

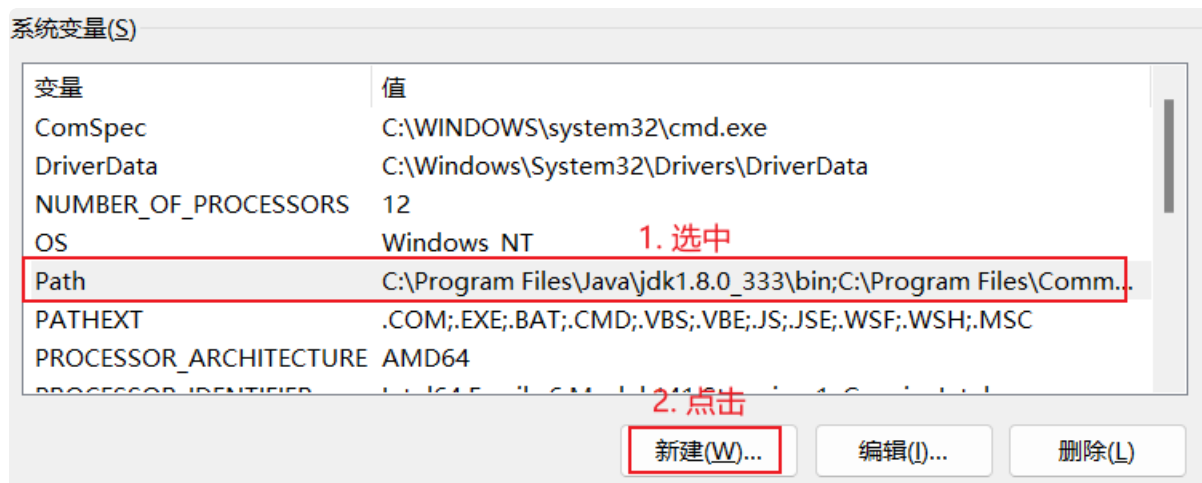
1. 复制JDK8文件夹所在的路径；



2. 在系统环境变量中新建一个 `JAVA_HOME`，然后JDK8的路径，即 `JAVA_HOME=C:\Program Files\Java\jdk1.8.0_333`

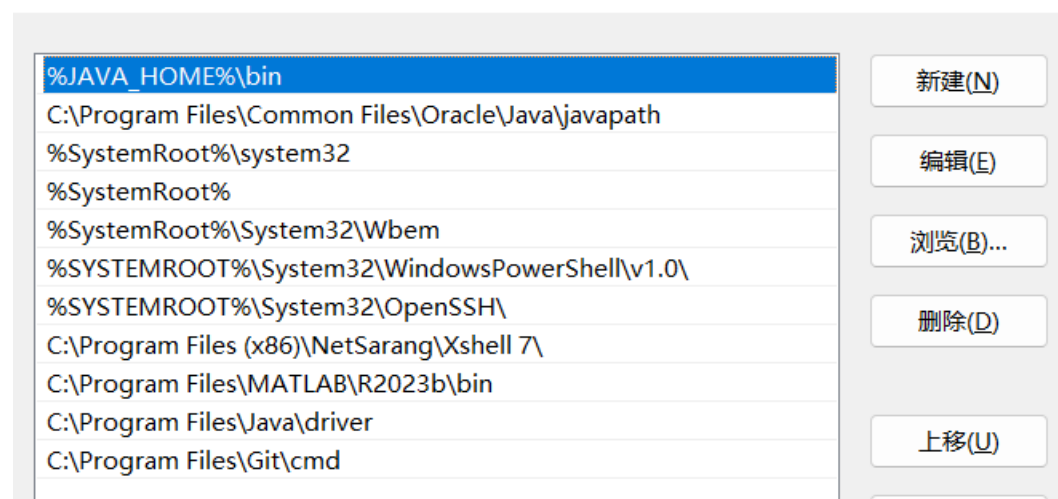


3. 在系统环境变量找到 `Path` 后，点击“新建”；



4. 将 `%JAVA_HOME%\bin` 添加到 `Path` 系统变量中；

5. 将该目录移到最顶部（因为系统查找文件时上从上到下搜索路径下的文件）。



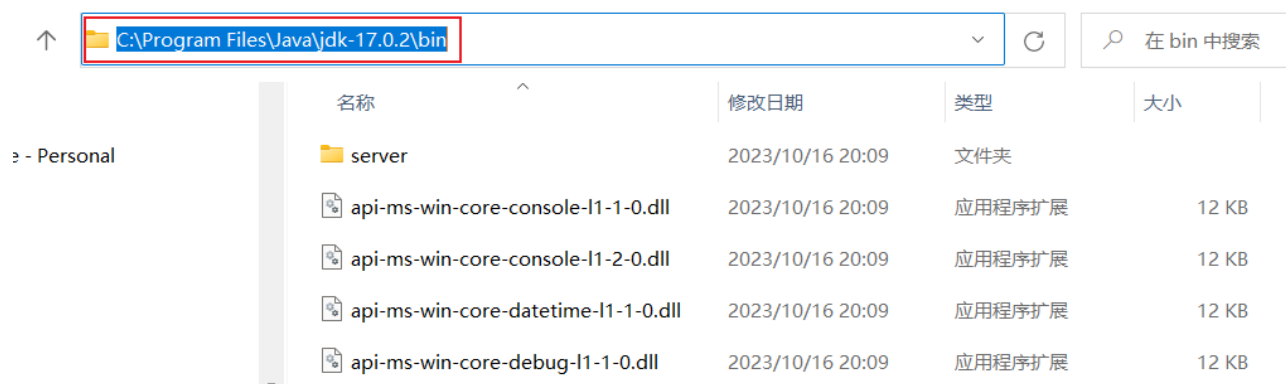
1.2.2. JDK17安装

1.2.2.1. JDK17软件

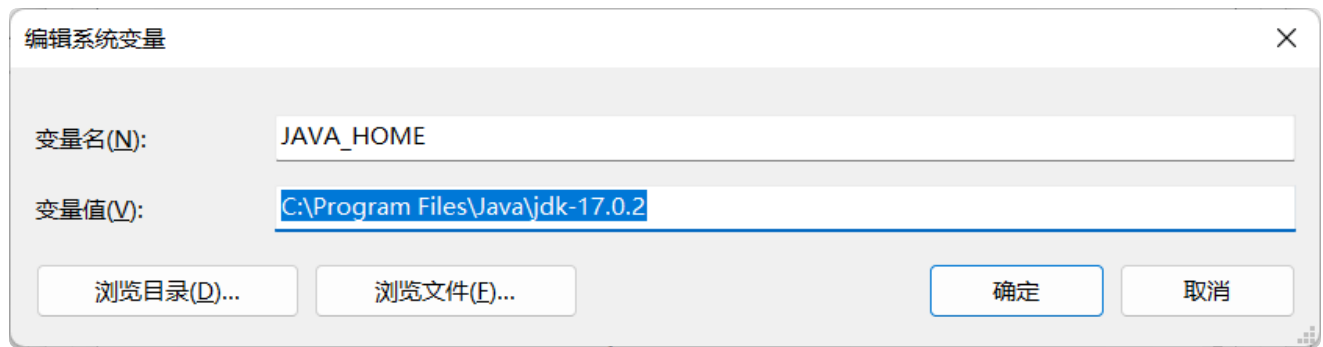
软件包地址：`C:\Users\20961\Documents\WPSDrive\418342908\WPS云盘\myFiles\编程\常用文件\Java\JavaSE`。

1.2.2.2. 配置环境变量

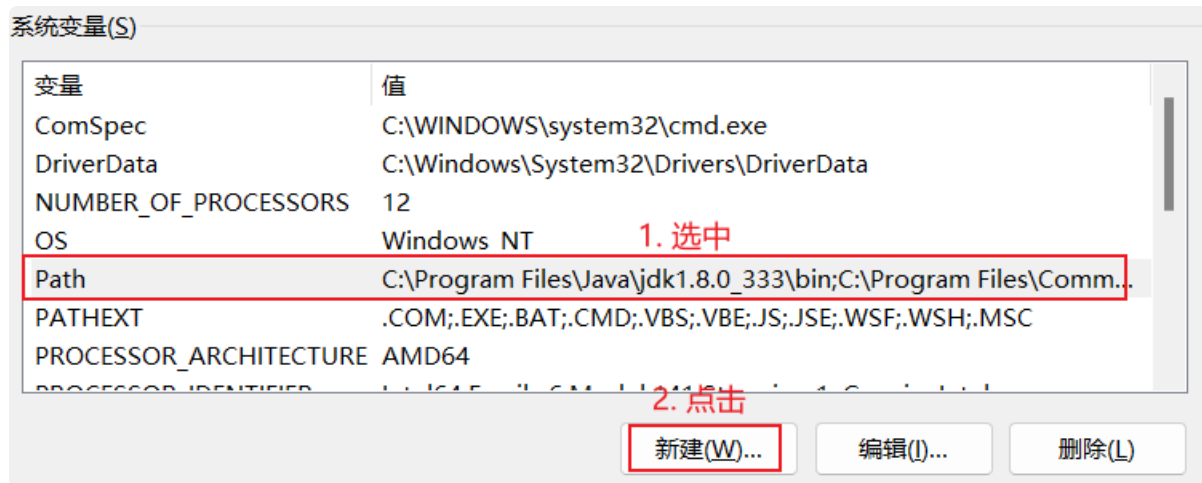
1. 复制JDK17文件夹所在的路径；



2. 在系统环境变量中新建一个 `JAVA_HOME`，然后JDK8的路径，即 `JAVA_HOME=C:\Program Files\Java\jdk-17.0.2`

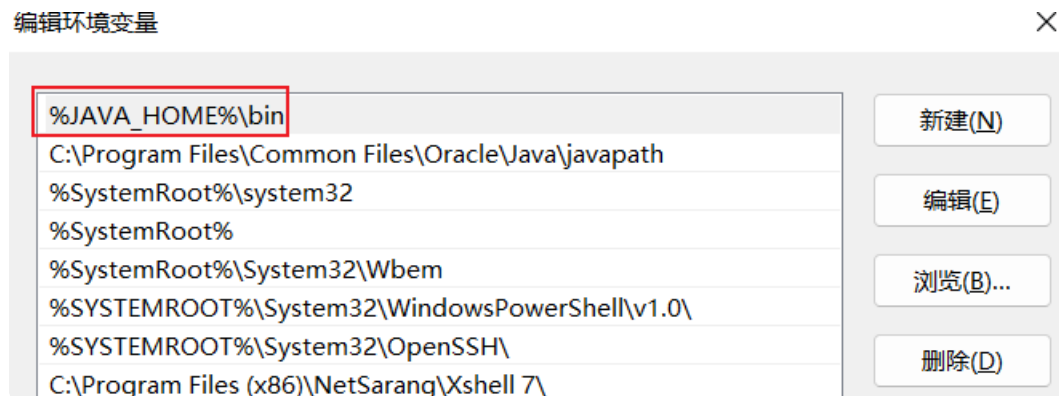


3. 在系统环境变量找到 **Path** 后，点击“新建”；



4. 将 **%JAVA_HOME%\bin** 添加到 **Path** 系统变量中；

5. 将该目录移到最顶部（因为系统查找文件时上从上到下搜索路径下的文件）。



1.2.3. JDK8安装与JDK17安装的区别

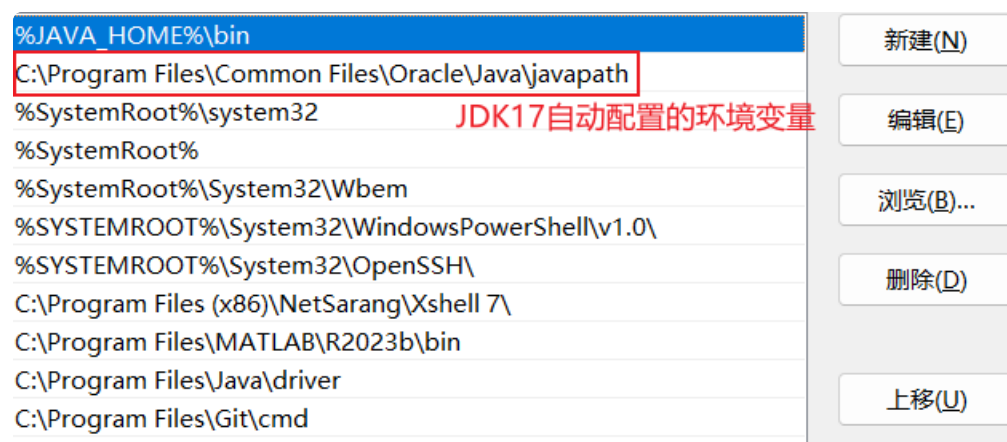
1. JDK17没有了JRE文件夹，而是将JRE中的文件分散到了JDK其他文件夹中了；

Windows-SSD (C:) > Program Files > Java > jdk-17.0.2

在jdk-17.0.2中搜索

名称	无jre文件夹	修改日期	类型	大小
bin		2023/10/16 20:09	文件夹	
conf		2023/10/16 20:09	文件夹	
include		2023/10/16 20:09	文件夹	
jmods		2023/10/16 20:09	文件夹	
legal		2023/10/16 20:09	文件夹	
lib		2023/10/16 20:09	文件夹	

2. 如果只是想单纯的配置JDK17的环境变量，那么在安装完JDK17后，无需再配置环境变量，因为在安装JDK17时已经自动配置相关的环境变量，具体如下：



1.2.4. 为什么使用 `JAVA_HOME` 变量的形式配置环境变量

之所以使用 `JAVA_HOME` 这种常量的方式，是为了后面学习高级知识用到，例如Tomcat会使用到 `JAVA_HOME` 这个常量

2. 编译与运行Java程序

2.1. 大体操作步骤

Java 程序开发三步骤：编写、编译、运行，具体如下：

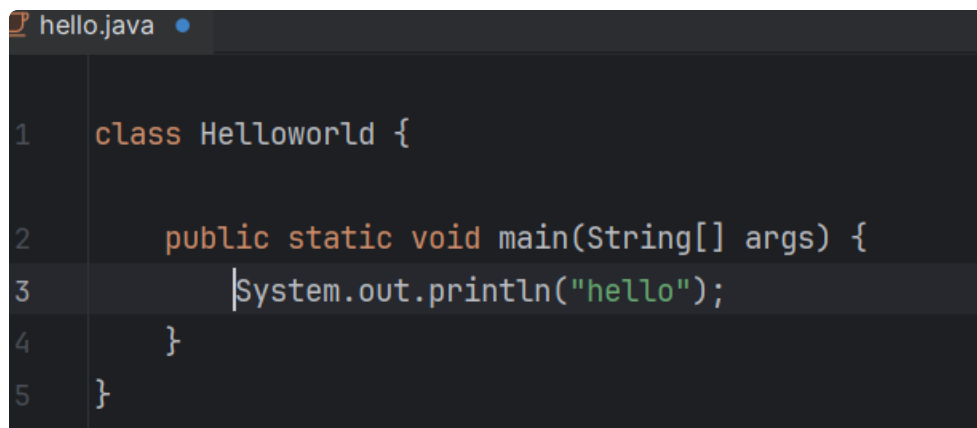
1. 将 Java 代码编写到扩展名为 .java 的源文件中；

2. 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；
3. 通过 `java.exe` 命令对生成的 `class` 文件进行**运行**

2.2. 具体操作步骤

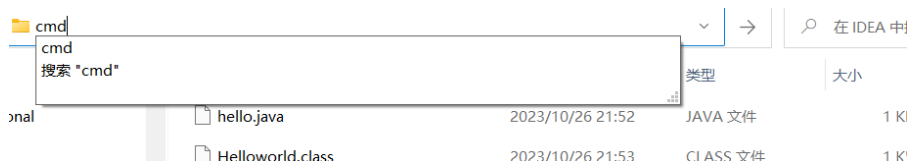
简单编写和运行Java程序，操作步骤如下：

1. 编写 `hello.java`



```
1 class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("hello");
4     }
5 }
```

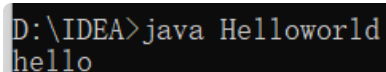
2. 在当前文件夹输入 `cmd` 调出当前文件夹所在命令行窗口；



3. 在命令行窗口编译 `hello.java` 文件，即 `javac hello.java`，得到 `Helloworld.class` 文件；

hello.java	2023/10/26 21:52	JAVA 文件	1 KB
Helloworld.class	2023/10/26 21:53	CLASS 文件	1 KB

4. 运行 `Helloworld.class` 文件，即 `java HelloWorld`。注意：不带 `class` 后缀。



```
D:\IDEA>java HelloWorld
hello
```

2.3. 总结

2.3.1. 源文件名与类名是否与类名一致？

1. 如果这个类不是 `public`，那么源文件名可以和类名不一致，但是不利于代码维护；

2. 如果这个类是public，那么要求源文件名必须与类名一致。否则报编译报错；



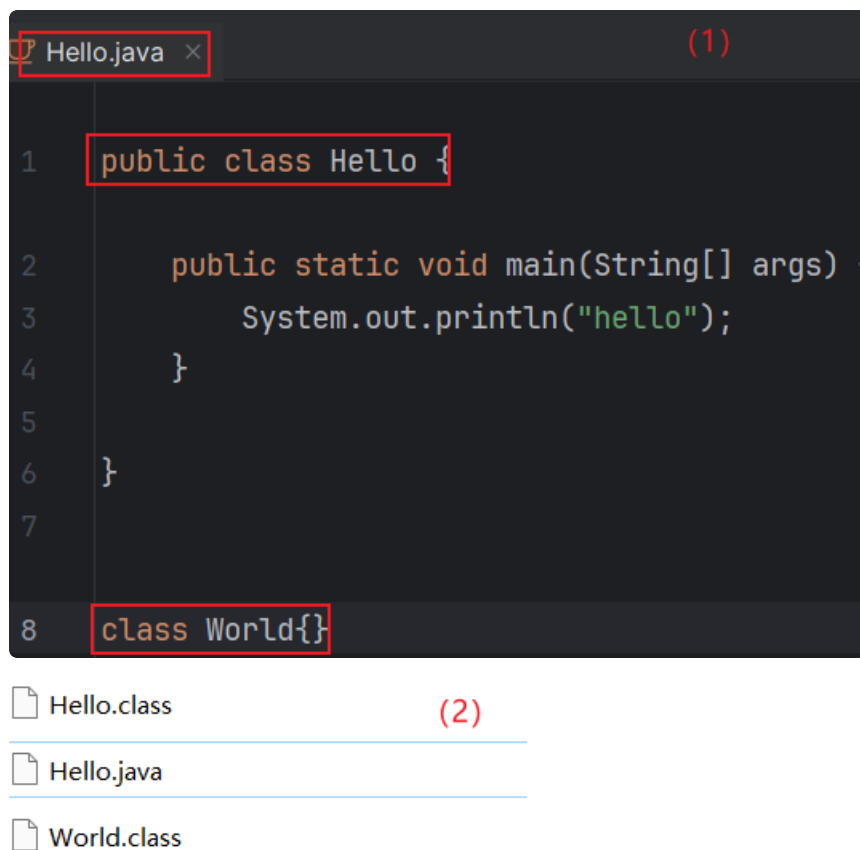
```
hello.java x (1)
1 public class Hello {
2     public static void main(String[] args) {
3         System.out.println("hello");
4     }
5 }

C:\IDEA>javac hello.java
hello.java:1: 错误: 类 Hello 是公共的, 应在名为 Hello.java 的文件中声明
public class Hello {
               (2)
```

3. 建议不是否为public，都要与源文件名保持一致，而且一个源文件尽量只写一个类，目的是为了好维护。

2.3.2. 一个源文件中，是否可以有多个类？

1. 一个源文件中可以有多个类，编译后会生成多个 `.class` 字节码文件，但是一个源文件只能有一个public类。



```
Hello.java x (1)
1 public class Hello {
2     public static void main(String[] args) {
3         System.out.println("hello");
4     }
5 }
6
7
8 class World{}
```

File Explorer:

- Hello.class (2)
- Hello.java
- World.class

2.3.3. Java程序的入口

1. 在运行 `class` 文件时，其包含的类必须由相关的程序运行的入口，即 `main` 方法，具体就是 `public static void main(String[] args)`。

2.3.4. 两种输出语句

2.3.4.1. 简介

1. 换行输出语句：`System.out.println(输出内容);`。输出内容，完毕后进行换行
2. 直接输出语句：`System.out.print(输出内容);`。输出内容，完毕后不做任何处理

2.3.4.2. 总结

1. 换行输出语句，括号内可以什么都不写，只做换行处理
2. 直接输出语句，括号内什么都不写的话，编译报错

2.3.5. 程序格式

1. Java程序严格区分大小写；
2. 结构格式

结构：

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

格式：

(1) 每一级缩进一个 Tab 键

(2) {}的左半部分在行尾，右半部分单独一行，与和它成对的"{"的行首对齐

3. 注释

3.1. 注释分类

1. 单行注释：`//`

2. 多行注释: `/**/`
3. 文档注释 (Java特有): `/** */`

3.2. 注释作用

3.2.1. 单行和多行注释

1. 对程序中代码进行解释说明;
2. 对程序进行调试。

3.2.2. 文档注释

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

1. 操作方式: `javadoc -d mydoc -author -version Hello.java`

```
D:\IDEA>javadoc -d mydoc -author -version Hello.java
正在加载源文件Hello.java...
正在构造 Javadoc 信息...
正在创建目标目录: "mydoc\"
正在构建所有程序包和类的索引...
标准 Doclet 版本 17.0.2+8-LTS-86
正在构建所有程序包和类的树...
正在生成mydoc\Hello.html...
Hello.java:10: 警告: args没有 @param
public static void main(String[] args) {
```

2. 进入生成的 `mydoc` 文件夹, 打开 `index.html`, 结果如下:

名称	修改日期	类型	大小
(1) 生成的mydoc文件夹			
mydoc	2023/10/27 20:28	文件夹	



3.3. 总结

1. 单行注释和多行注释中声明的信息，不参与编译，即编译以后的字节码文件不包含单行注释和多行注释中的信息；
2. 多行注释不能嵌套使用，即不能出现 `/*/**/*/`

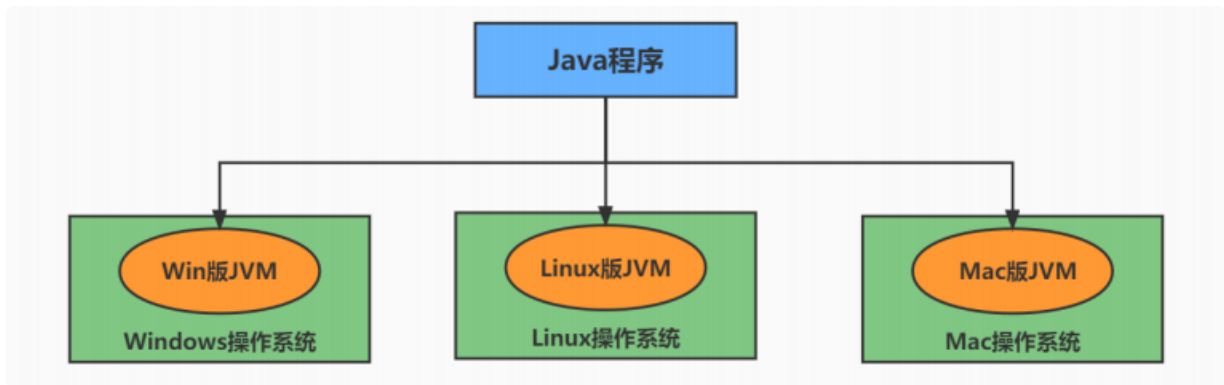
4. Java17 API文档

- [Java17 API在线文档](#)

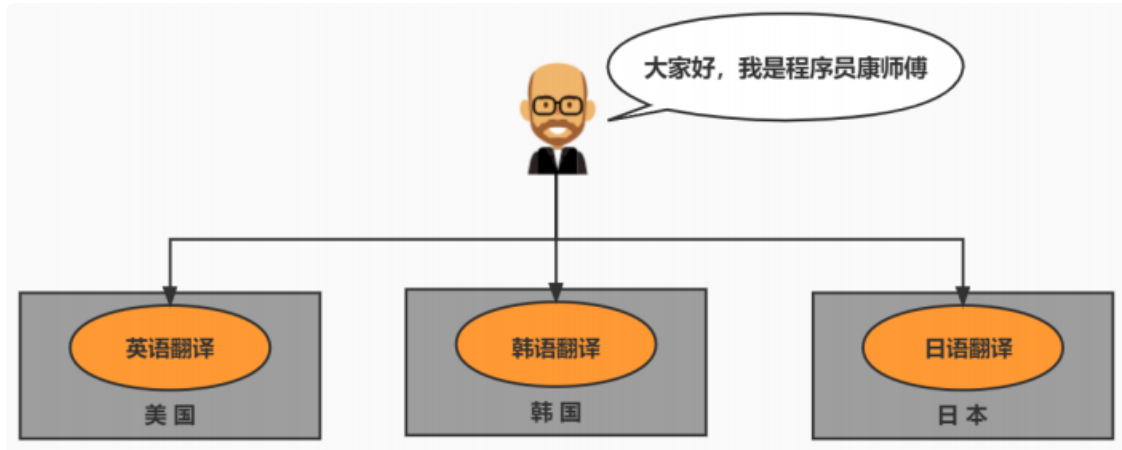
5. Java核心机制：JVM

5.1. Java语言的优点

1. 跨平台。
 - a. 简介：通过 Java 语言编写的应用程序在不同的系统平台上都可以运行。“*Write once , Run Anywhere*”。
 - b. 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。



跨平台



与跨平台相关的翻译问题

2. 面向对象

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。

3. 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。

4. 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），可以分配不同的命名空间以防替代本地的同名类、字节代码检查。

5. 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。

6. 高性能

- a. 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。
- b. Java 低性能的短腿，已经被完全解决了。业界发展上，我们也看到很多C++应用转到 Java 开发，很多 C++程序员转型为 Java 程序员。

5.2. JVM

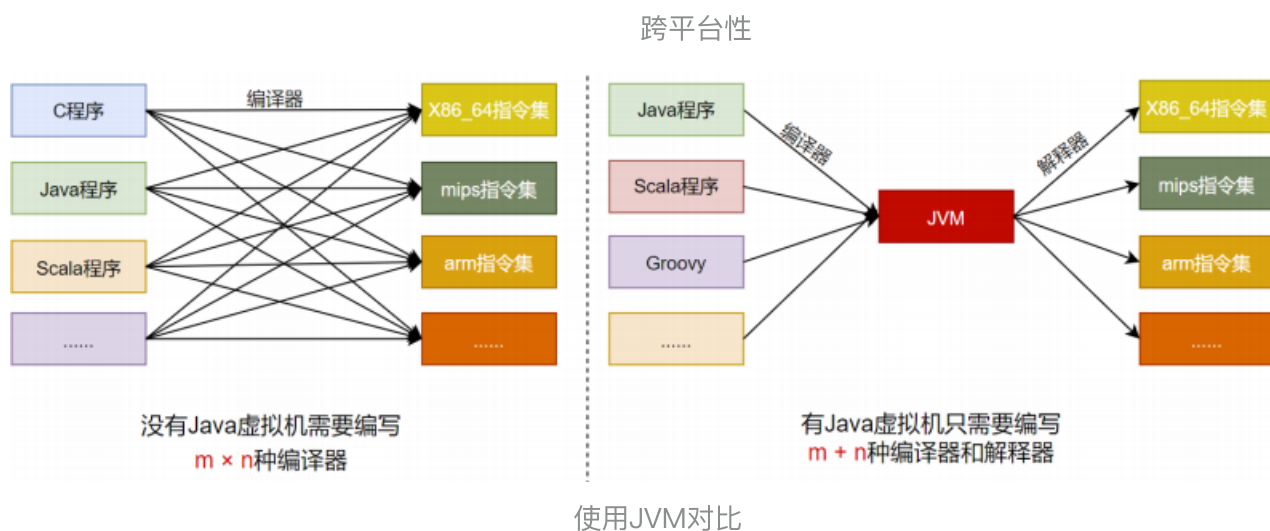
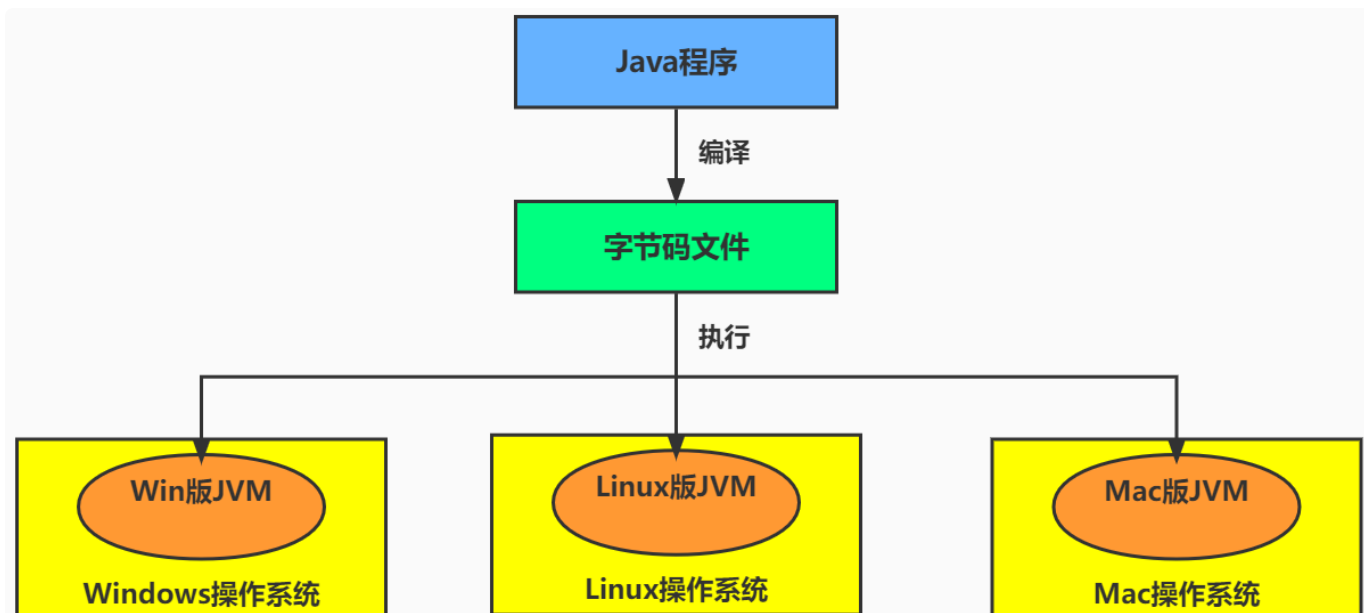
5.2.1. 简介

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。



JVM

5.2.2. 功能一：实现Java程序的跨平台性



5.2.3. 功能二：自动内存管理（内存分配、内存回收）

5.2.3.1. 简介

1. 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；
2. 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；
3. 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。

5.2.3.2. 面试题01：Java程序还会出现内存溢出吗？

会的。如果一个系统的内存只有2G，创建的程序过多，而且这些程序并不是垃圾程序，那么JVM就不会清理这些程序，最终导致内存溢出。

5.2.3.3. 面试题01：Java程序还会出现内存泄漏吗？

会的。内存泄漏即垃圾程序所占的内存没有被回收。JVM回收内存是根据相关的算法，并不能识别出所有的垃圾程序或及时识别出垃圾程序，所以还是会出现内存泄漏。