Java语言基础

Java语言概述

1，Java语言发展史

Java语言本质

- Java是面向对象的高级语言，它是由c和c++发展而来。

Java发展语言历史概述

- Java 是由 James Gosling 在 Sun公司领导的小组开发的。2010 年 Sun 公司被 Oracle 收购。Java 最初被称为 Oak (橡树)，是 1991 年为消费类电子产品的 嵌入式芯片而设计的，由于当时的消费类电子产品的面积小使得该语言无法被广泛推出。随着互联网浪潮的推动，Sun公司发现其在互联网中的运用空间，于1995 年更名为 Java, 并重新设计用于开发 Web应用程序。
- Java 一开始富有吸引力是因为 Java 程序可以在 Web 浏览器中运行。这种能在 Web浏览器中运行的 Java 程序称为 Java 小程序（ applet)。applet 使用现代的图形用户界面与 Web 用户进行交互，处理用户的要求，界面中包括按钮、文本字段、文本域、单选按钮等。applet使得 Web 更加具有响应性、交互性和趣味性。applet内嵌在 HTML 文件中。
- 现在，它不仅用于 Web 程序设计，Java 广泛用于开发服务器端的应用程序。这些应用程序处理数据、执行计算，并生成动态网页。许多商用网站后端都是采用 Java 进行开发的。Java 是一个功能强大的程序设计语言，可以用它来开发台式计算机、 服务器以及手持设备上的应用程序。用于安卓手机的软件也是采用Java 进行开发的。用它开发过与火星探测器通信并控制其在火星上行走的代码。

JDK版本发展

从1996年1月23日发表JDK1.0到目前JDK19，Java已经流行了27年。

1. JDK1.0：提供了一个解释执行的 Java 虚拟机；Applet 能在 Mozilla 浏览器中运行。
2. JDK1.1：引入JDBC（Java Database Connectivity）；支持内部类；引入Java Bean；引入 RMI（Remote Method Invocation）；引入反射（仅用于内省）。
3. JDK1.2：引入集合（Collection）框架；对字符串常量做内存映射；引入 JIT（Just In Time） 编译器；引入对打包的 Java 文件进行数字签名；引入控制授权访问系统资源的策略工具；引入 JFC（Java Foundation Classes），包括 Swing 1.0、拖放和 Java 2D 类库；引入 Java 插件；在 JDBC 中引入可滚动结果集、BLOB、CLOB、批量更新和用户自定义类型；在 Applet 中添加声音支持。
4. JDK1.3：引入Java Sound API；jar 文件索引；对 Java 的各个方面都做了大量优化和增强。
5. JDK1.4：XML 处理；Java 打印服务；引入 Logging API；引入 Java Web Start；引入 JDBC 3.0 API；引入断言；引入 Preferences API；引入链式异常处理；支持 IPv6；支持正则表达式；引入 Image I/O slot machine API。
6. **JDK1.5（2004年9月）**：引入泛型；增强循环，可以使用迭代方式；自动装箱与自动拆箱；类型安全的枚举；可变参数；静态引入；元数据（注解）；引入 Instrumentation。
7. JDK6：支持脚本语言；引入 JDBC 4.0 API；引入 Java Compiler API；可插拔注解；增加对 Native PKI(Public Key Infrastructure)、Java GSS(Generic Security Service)Kerberos 和 LDAP(Lightweight Directory Access Protocol) 的支持；继承 Web Services；做了很多优化。

8. JDK7：switch 语句块中允许以字符串作为分支条件；在创建泛型对象时应用类型推断；在一个语句块中捕获多种异常；支持动态语言；支持 try-with-resources；引入 Java NIO.2 开发包；数值类型可以用2进制字符串表示，并且可以在字符串表示中添加下划线；钻石型语法；null 值的自动处理。
9. **JDK8（2014年3月）**：Lambda 表达式；Pipelines 和 Streams；Date 和 Time API；Default 方法；Type 注解；
   Nashhorn JavaScript 引擎；并发计数器；Parallel 操作；移除 PermGen Error；TLS SNI。
10. JDK9：模块化 —— Jigsaw；交互式命令行 —— JShell；默认的垃圾回收器 —— G1；进程操作改进；竞争锁的性能优化；分段代码缓存；优化字符串占用空间。
11. JDK10：局部变量类型推断，类似JS可以通过var来修饰局部变量，编译之后会推断出值的真实类型；并行Full GC，来优化G1的延迟；允许在不执行全局VM安全点的情况下执行线程回调，可以停止单个线程，而不需要停止所有线程或不停止线程。
12. **JDK11（2018年9月）**：ZGC，ZGC可以看做是G1之上更细粒度的内存管理策略。由于内存的不断分配回收会产生大量的内存碎片空间，因此需要整理策略防止内存空间碎片化，在整理期间需要将对于内存引用的线程逻辑暂停，这个过程被称为"Stop the world"，只有当整理完成后，线程逻辑才可以继续运行。（并行回收）；Flight Recorder（飞行记录器），基于OS、JVM和JDK的事件产生的数据收集框架；对Stream、Optional、集合API进行增强。
13. JDK12：Shenandoah GC，新增的GC算法；switch 表达式语法扩展，可以有返回值；G1收集器的优化，将GC的垃圾分为强制部分和可选部分，强制部分会被回收，可选部分可能不会被回收，提高GC的效率。
14. JDK13：Socket的底层实现优化，引入了NIO；switch表达式增加yield关键字用于返回结果，作用类似于return，如果没有返回结果则使用break；ZGC优化，将标记长时间空闲的堆内存空间返还给操作系统，保证堆大小不会小于配置的最小堆内存大小，如果堆最大和最小内存大小设置一样，则不会释放内存还给操作系统；引入了文本块，可以使用"""三个双引号表示文本块，文本块内部就不需要使用换行的转义字符。
15. JDK14：instanceof类型匹配语法简化，可以直接给对象赋值，如if(obj instanceof String str),如果obj是字符串类型则直接赋值给了str变量；引入record类，类似于枚举类型，可以向Lombok一样自动生成构造器、equals、getter等方法；NullPointerException打印优化，打印具体哪个方法抛的空指针异常，避免同一行代码多个函数调用时无法判断具体是哪个函数抛异常的困扰，方便异常排查。
16. JDK15：隐藏类 hidden class；密封类 sealed class，通过sealed关键字修饰抽象类限定只允许指定的子类才可以实现或继承抽象类，避免抽象类被滥用。
17. JDK16：ZGC性能优化；instanceof模式匹配；record的引入。
18. **JDK17（2021年9月）**：正式引入密封类sealed class，限制抽象类的实现；统一日志异步刷新，先将日志写入缓存，然后再异步刷新。

    其中具有重大改变或长期支持的重要版本是JDK5，JDK8，JDK11和JDK17，其中JDK8是具有重大改变和长期支持的版本，也是目前企业使用最多的一个JDK版本。最近的JDK17版本实际上没有多大的改变，但是它是整合并正式在Java中引入前几个版本的特性。

2，JavaSE，JavaEE，JavaME

JavaSE

- 允许开发和部署在桌面、服务器、嵌入式环境和实时环境中使用的 Java 应用程序。Java SE 包含了支持 Java Web 服务开发的类，并为Java EE和Java ME提供基础。
- JavaSE可以开发客户端的应用程序也就是主要开发用于C/S端的应用程序，如果学完swing就可以开发图形化的软件，比如类似于桌面的QQ，微信，网易音乐等这样的应用。但是值得注意的是Java并不是更专业的开发语言在开发具有GUI界面的桌面软件领域，在该领域中主流的是.net,c#和c++，Java的专业领域应当在大型服务器，分布式和大数据领域。

JavaEE

- 企业版本帮助开发和部署可移植、健壮、可伸缩且安全的服务器端Java 应用程序。Java EE 是在 Java SE 的基础上构建的，它提供 Web 服务、组件模型、管理和通信 API，可以用来实现企业级的面向服务体系结构（service-oriented architecture，SOA）和 Web2.0应用程序。
- JavaEE可以用来开发服务器端的程序，该阶段是以Java语言为主，还需要学习数据库，前端的一些知识（HTML5，CSS3和javaScripts语言）以及各种框架（Spring，SSM，mybaits等）等，学完该阶段之后，可以结合一些必要的web前端的知识就能该法出像淘宝，京东，百度...这样基于B/S端的程序，所谓的B/S端的程序也就是当我们打开浏览器，输入网址之后呈现出来的各种各样的网站，这类基于浏览器的具有客户端和服务端交互的程序就是B/S程序，是目前主流的程序和互联网架构。

JavaME

- 为在移动设备和嵌入式设备（比如手机、PDA、电视机顶盒和打印机）上运行的应用程序提供一个健壮且灵活的环境。Java ME包括灵活的用户界面、健壮的安全模型、许多内置的网络协议以及对可以动态下载的连网和离线应用程序的丰富支持。基于 Java ME 规范的应用程序只需编写一次，就可以用于许多设备，而且可以利用每个设备的本机功能。
- JavaME也是基于JavaSE的基础上为开发移动设备，嵌入式设备提供的的一些必要类库，它和JavaEE是两个不同的方向，学完该阶段之后可以开发一些嵌入式的设备软件，比如目前的安卓环境下，也就是目前主流移动设备操作系统的应用软件都是基于Java语言开发的，虽然目前安卓也有自己专用的编程语言Kotlin，但是Kotlin使用的任然是Java的虚拟机并且它可以说是Java的一个衍生，在编写的过程中都可以看到Java的影子。

目前Java其他领域

- 除此之外，目前Java由于其优秀的分布式和数据计算能力，现在也渐渐的应用于大数据领域，目前大数据领域使用的编程语言的Scale，但是它也是使用Java的虚拟机并且和Java的差别不大。

3，JVM，JRE，JDK，API

JVM

- Java virtual machines，它的作用的加载.class文件并且运行.class文件。

JRE

- Java running environment，Java运行环境，它是Java程序运行的最小环境，如果我们使用PC端下载了一个Java应用程序的话，那么PC中就必须具有JRE，否则程序将无法运行，当然这只是个举例，实际上在下载大部分的Java应用程序的时候都会打包运行环境，不需要用户去自行安装，当然也有特殊情况没有打包。
- 它包含JVM和Java运行是必要的环境依赖，比如一些操作系统的依赖文件和Java的核心类库。

JDK

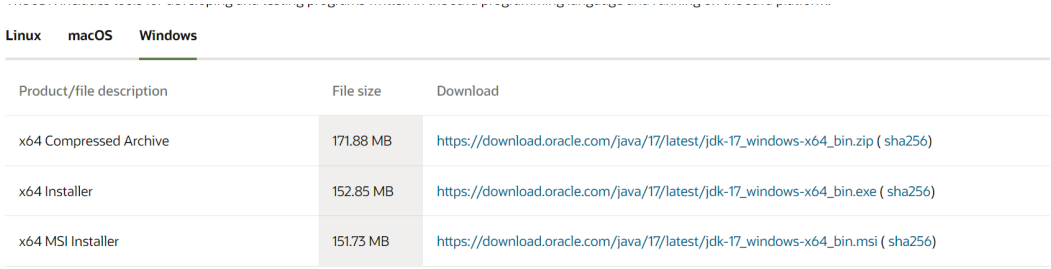
- Java Development kit，Java开发工具包，如果要开发一个Java应用程序的话就必须具有JDK，它是开发一个Java程序的最小单位，比如编译Java源程序的工具javac.exe和java.exe等。
- 它包含JRE和Java的一些编译，运行等开发工具。
- 在计算机中安装JDK

  对于我们来说，我们需要去开发Java程序，所以我们需要在我们的计算机中安装JDK，配置JDK的过程是：检查自己的计算机受否装有JDK，官网下载对应的JDK，配置环境变量，检验是否装好JDK。

  比如：在Windows系统中安装JDK

  - win+r打开命令窗口，输入cmd，回车进入DOS系统。

- 输入java此时如果出现无法找到该文件的意思就是计算机没有安装过JDK。
- 进入Oracle（甲骨文）官网下载https://www.oracle.com/java/technologies/java-se-glance.html，我们知道Java是sun公司开发的，但是sun公司在2010年被Oracle公司收购了。可以看到目前最新的版本是Java19，但是它并不是长期支持的版本，目前最近的长期支持的版本是JDK17，也可以下载目前市场上使用最广泛的JDK8。



Linux macOS Windows

Product/file description	File size	Download
x64 Compressed Archive	171.88 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256)
x64 Installer	152.85 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256)
x64 MSI Installer	151.73 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256)

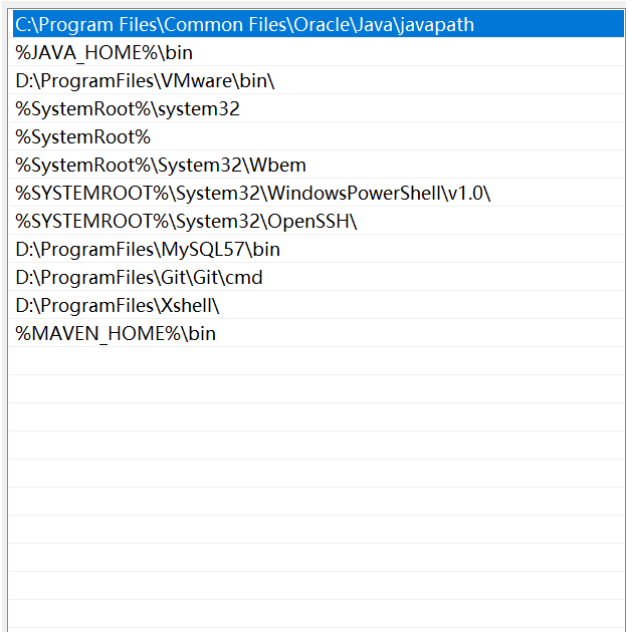
下载自己操作系统的JDK，这里 选择Windows版本的JDK17，可以看到目前只支持x64架构的CPU了，有Compressed Archive是压缩包版的，可以直接解压安装。Installer版本的是使用安装程序进行安装的应用程序版的，可以下载这个版本使用安装向导进行安装。MSI是使用微软格式安装程序，它不仅是安装程序而且在安装程序之后还具有容灾备份和管理安装的软件的应用。现在使用.exe的安装方式安装JDK17。下载，这里下载需要首先登陆到Oracle官网才能下载。

- 在非系统盘中新建文件夹并命名为JDK17，把下载好的.exe文件拖动到JDK17文件夹下，点击.exe文件进行安装，这里需要注意的是安装的目录需要需改为该JDK17文件夹为根目录的路径，等待安装即可。
- 重新打开DOS系统进行测试，此时如果输入java --version会出现找不到目录，当然使用.exe安装的即使自己没有配置它也能找到。但是我们还是走一遍老流程，手动配置JDK17的环境变量。
- 进入Windows环境变量的设置，点击此电脑，右键，属性，进入到属性中，点击高级系统设置，点击环境变量，可以看到用户变量框和系统变量框，我们需要在系统变量框中配置环境变量，点击新建，变量名为JAVA_HOME,变量值为JDK17文件夹所在的路径比如D:\ProgramFiles\JDK17,点击确定，然后找到Path，点击进入Path，然后点击新建，输入%JAVA_HOME%/bin，点击确定，回到环境变量界面，点击确定，再点击确定即可。
- 验证JDK，把原来的DOS窗口关闭，这是配置之前打开的窗口还没有同步设置，如果在这个窗口上验证还是无法查找，重新打开一个DOS窗口，然后输入java --version即可看到JDK的版本信息。

```
C:\Users\EMCred>java --version
java 17.0.5 2022-10-18 LTS
Java(TM) SE Runtime Environment (build 17.0.5+9-LTS-191)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.5+9-LTS-191, mixed mode, sharing)
```

- 为什么JDK17实际上不需要我们手动配置环境变量也能找到?

  这是因为从JDK11版本之后所下载的.exe的安装JDK的引导程序会自动的帮助我们在C盘中建立一个C:\Program Files\Common Files\Oracle\Java\javapath的文件夹，然后把该路径直接放置到系统环境变量中的path中。我们打开环境变量后可以看到。



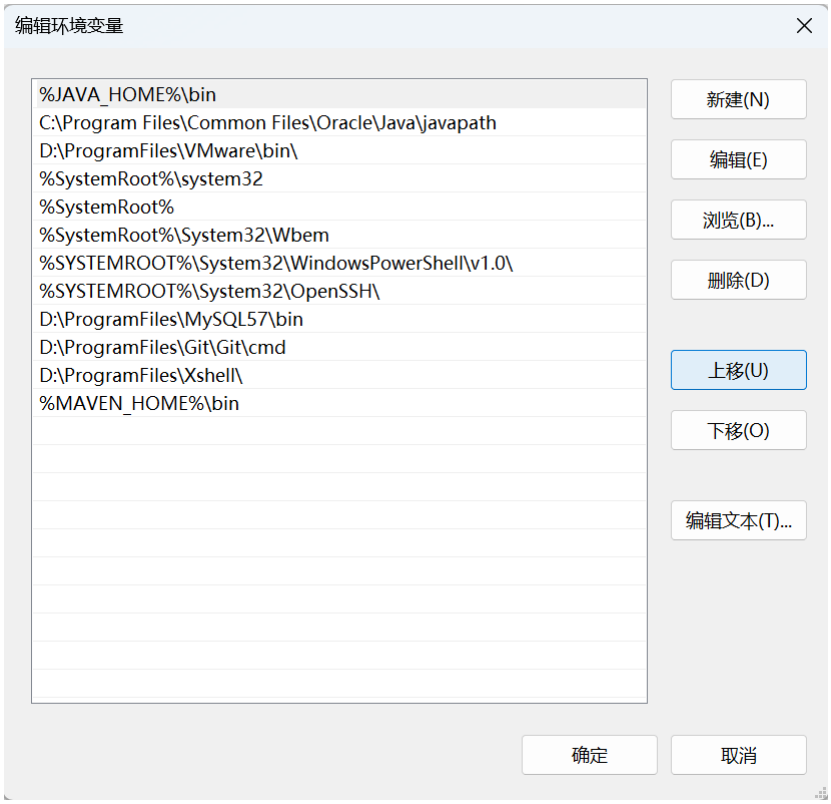
第一个就是JDK17的自动配置的环境变量。

在实际开发中目前使用最多的版本是JDK8的版本，为什么JDK已经升级到了17但是企业任然会广泛的使用JDK8的版本，首先JDK8的版本属于长期支持的版本，官网显示目前它可以支持到2030年左右。其次JDK8的版本稳定，之前企业大量的平台使用JDK8作为开发的Java版本，如果目前突然更换JDK的话，那么在平台中所使用的框架依赖的JDK版本出现不兼容的现象，导致线上出现错误。在IT行业所以的版本不是最新也不是很旧而是长久的版本和稳定的版本。所以我们也需要在计算机中安装JDK8版本。

对于JDK8版本安装过程就不再多叙，也是从官网下载.exe的安装引导程序进行安装，这里需要注意的点是：

- JDK8的安装引导程序和JDK17的有些不一样，我们需要注意的是在选择安装源程序，工具和公共JRE的选项中我们需要将公共JRE右键选择不安装，然后随意点击源程序更改安装的路径即可。
- 在JDK8安装到JDK8文件夹后，我们只需要去环境变量中把JAVA_HOME的值设置到JDK8的文件路径下即可。
- 当我们使用CMD测试的时候会发现输出的任然是JDK17的版本，此时如果想要换为JDK8的版本的话。

  我们需要进入到Path中，然后点击上移，让%JAVA_HOME%\bin位列最上位即可，也就是如下。



然后点击确定，确定，确定，重新打开CMD或powershell进行测试。输入命令**java -version**，即可看到版本如下所示。

实际上我们使用原来的方法配置JDK17也是可以的，我们只需要将%JAVA_HOME%\bin位列最上位即可。

- JDK目录文件说明

文件名	说明
bin	存放JDK的各种开发的命令工具，比如javac.exe和java.exe
conf	存放JDK的相关配置文件
include	存放与当前操作系统依赖相关的头文件
jre	JRE文件夹，包含Java虚拟机和运行时必要的一些类库（JDK17版本中可能没有发现有jre文件夹，需要进入cmd中使用命令bin\jlink.exe --module-path jmods --add-modules java.desktop --output jre就可以调用jre文件夹）
jmods	存放JDK的各种模块化工具（JDK8之前没有模块化封装，JDK9开始使用模块化封装技术），这里先了解一下这是JDK9的新特性。所谓的模块化就是代码和数据集合。 它可以包含Java代码和本地代码。 Java代码被组织为一组包含诸如类，接口，枚举和注解等类型的类。 数据可以包括诸如图像文件和配置文件的资源。它们改进了以前JDK/JAR的单体结构，实现JDK/JAR的模块化开发，同时能够强封装和可靠的配置。将Java的系统看成是由一个个的模块搭建而成的
legal	存放JDK的授权文档
lib	开发工具所使用的文件
其他文件	一些关于JDK的说明文件
javafx-src.zip	在JDK8之前，javafx是包含在JDK之中的，javafx是一种Java的GUI框架，它是基于MVC模式的是对Swing框架的一种升级，在JDK9之后它就作为一个单独的扩展包，需要我们自行下载导入

- 环境变量

  环境变量属于操作系统的相关知识，在这里需要知道的几点。

  - 什么是环境变量?

    一般是指在操作系统中用来指定操作系统运行环境的一些参数，如：临时文件夹位置和系统文件夹位置等。

    所谓变量，那么定义的格式应当是符合变量名=变量值，最明显的好处是变量名可以在多处使用，那么当变量值需要改变的时候只需要改变变量名所在的变量值的值即可，比如现在设置的JAVA_HOME="D:/PogramFiles/JDK8",然后它在Path中使用，当我想更换为JDK9的时候，只需要将JDK8该为JDK9即可。

  - 为什么要使用环境变量?

    为了能够让该程序可以在任何目录下启动运行。

    其实我们下载并解压好JDK8的时候JDK就已经存在了，但是当我们打开DOS去运行java --version的时候却不能找到，这是因为我们刚打开所进的DOS系统默认是在C盘的用户文件夹下,当我们输入指令java --version时它需要找到对应文件并调用程序，而Java指令在JDK8的bin目录下，它寻找当前文件夹中的内容并没有找到，因为用户文件夹下根本没有与JDK相关的文件，然后它会去看Path中是否设置过对应的文件，但是依然没有，所以它会找不到。

    如果我们进入到JDK8文件夹下，点击文件路径栏，输入cmd，可以看到此时进入的是当前文件夹所在的目录,此时输入java --version指令它就能出现相关的响应，所以它已经存在。

    此时我们在Path目录中配置了%JAVA_HOME%/bin,这里的%变量名%是取变量值的意思，也就是取该变量名所对应的变量值，这里翻译过来就D:/ProgramFiles/JDK8/bin。

    此时打开无论进入到任何的目录下，当输入java --version时都会有响应，这是因为即使当前文件夹中找不到，但是当去Path中就能够找到对应的文件。

我们之所以在这里使用环境变量是能够让操作系统找到并操作Java，因为我们需要开发Java，所以需要使用到开发工具也就是bin目录下的内容，为了能够在任何地方使用开发编译Java程序，所以需要将该目录设置到Path中，当然之后使用IDE进行开发也可以不配置Path，比如idea，但是需要选定对应的文件夹D:/ProgramFiles/JDK8，而且在JavaEE开发阶段的时候需要使用到Tomcat，这时就会出现Tomcat无法启动的问题，这是因为Tomcat依赖于Java，它就是Java开发的Web容器，它不能通过path找到对应的JDK就无法运行，所以最好还是配置Path。

- 用户变量和系统变量

  两者最大的区别是，在Windows上我们是使用对应的Microsoft用户登录使用计算机的，当前登录的用户如果只在该用户的Path中设置环境变量，那么其他用户登录的话就不能够使用之前用户的用户变量设置的内容，需要自己重新设置，而系统变量是无论是哪个用户登录都能使用。

  对于双系统的计算机，比如C盘安装Windows7，D盘安装了Windows10，此时如果在计算机的系统变量中进行了设置即使切换系统也能够正常使用Path中的内容。

4，Java语言的特点

Java 的快速发展以及被广泛接受都应归功于它的设计特性，特别是它的承诺：一次编写，任何地方都可以运行。跨平台运行是Java最大的特点。

Java跨平台运行的原理

- 什么是跨平台?

  这里的平台指的是操作系统可以说是跨操作系统运行。

  这里需要知道的特定的操作系统的底层指令集是不同的，比如Liunx操作系统的指令集和Windows操作系统的指令集是不一样，就算是Windows操作系统的指令集也是不一样的比如X86和X64，这就意味着在不同的操作系统上所开发的软件调用的底层指令是不同的，对同一个问题解决的代码形式也是不同的，这就意味着不同的操作系统所开发的软件只能在各自系统上运行，如果将软件下载到其他的操作系统就无法运行。

  比如，如果我们使用非跨平台的语言（如C/C++这类的编译型语言）开发QQ，在Windows平台下我们需要开发一款Windows版本的QQ，而在Liunx平台（Android的底层架构就是基于Liunx的）下我们需要开发一款Liunx版本的QQ，虽然我们可以看到QQ的实现功能差不多，但是我们需要在两台操作系统上使用不同的代码来实现同样的功能，并且如果我们将Windows版本的QQ发到我们的Liunx设备上它是无法运行的，如果出现了新的流行操作系统就需要在对应的操作系统上编写对应的代码来实现QQ。

  跨平台所实现的就是无论底层的操作系统是什么，随便在一款操作系统上编写的代码能够在其他操作系统上运行，也就是所说的一次编写到处运行。

  比如，我们使用跨平台语言（如Java这样的半编译半解释型语言或者是Python这样的解释型语言）开发QQ，我们只需要在Windows平台下开发好QQ，然后该QQ软件不仅可以在Windons环境下运行，也可以在Liunx环境下运行而不需要去写不同的代码，如果出现了新的流行的操作系统，那么我们也不用根据操作系统的特点去更改代码。

- Java如何实现跨平台的

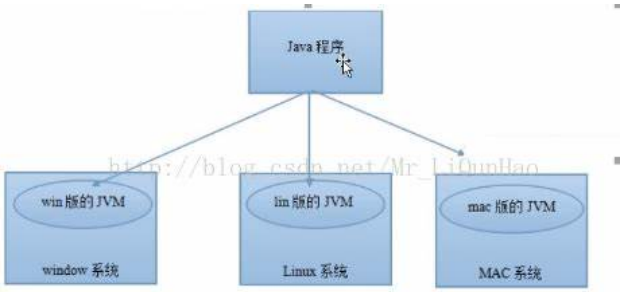
  Java实现跨平台的原理是它采用了一种虚拟机的机制，也就是JVM。

  所谓的虚拟机可以看作是一种逻辑上假想的一台计算机，我们现在使用的是物理上真真确确的计算机。虚拟机和一台计算机的功能有异曲同工之处。

  对于Java虚拟机来说，我们先不看JVM的实现机制，而是看JVM是如何实现跨平台的，Java的JVM是使用C/C++封装而成的，也就是说JVM是依赖于特定的操作系统的，Windows操作系统需要下载Windows的JRE，Liunx需要下载Liunx的JRE，刚刚我们在安装JDK的时候就会发现官网上需要我们针对自己的操作系统和具体的CPU采用的指令架构下载对应的JDK。

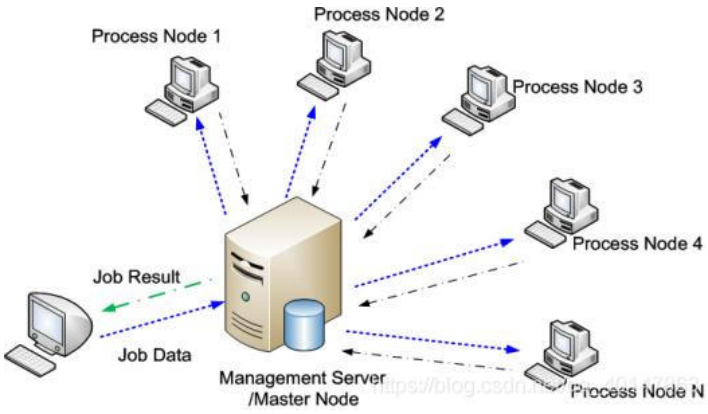
JVM的作用是加载.class文件并且运行.class文件的，而.class文件是由Java源代码经过特定平台javac的编译而来的，这个.class文件虽然经过特定平台的编译工具的编译，但是不同平台下编译出来的内容是一样的文件。不同平台的JVM识别.class文件 ，然后根据自己平台的指令集和平台特性翻译成对应的机器指令进行执行完成相同的功能。

可以这样理解，假如现在我有两台操作系统可以使用，一套是Windows操作系统，另一套是Liunx操作系统，我在这两套操作系统上下载了对应的JDK，当我在Windows操作系统下编写的java源代码经过编译后生成的.class文件放在Liunx平台下运行而不会出现任何错误并且最后的功能一致，而且如果我在Liunx平台下反编译出源代码并且进行修改后产生的新的.class文件放到Windows平台下也能运行并且功能一致。这样如果之后出现了主流的操作系统，只需要下载对应系统的JRE就能运行不同系统的Java程序。



Java的其他特性

- 简单：Java没有复杂指针的概念，也没有多继承的概念，也不需要担心内存泄漏的问题，它的GC会随机收取垃圾。
- 面向对象：Java是一种支持面向对象开发的语言，它其中内置了许多的关键字用于面向对象的开发。
- 跨平台性（体系结构中立）
- 可移植性：所谓的可移植性指的是同一个程序可以在不同的操作系统之间任意的进行部署，这样就减少了开发的难度，对于Java来说它是基于JVM实现的。
- 分布式：通过计算机网络将后端工作分布到多台主机上，多个主机一起协同完成工作。



比如：淘宝，它本身就是一个分布式系统，我们通过浏览器访问淘宝网站时，这个请求的背后就是一个庞大的分布式系统在为我们提供服务，整个系统中有的负责请求处理，有的负责存储，有的负责计算，最终他们相互协调把最后的结果返回并呈现给用户。

Java 语言支持 Internet 应用的开发，在基本的 Java 应用编程接口中有一个网络应用编程接口（java net），它提供了用于网络应用编程的类库，包括 URL、URLConnection、Socket、ServerSocket 等。Java 的 RMI（远程方法激活）机制也是开发分布式应用的重要手段。

- 多线程：Java支持采用单进行多线程的方式进行开发软件

所谓单进程多线程可以这样理解，我们平时打开一个软件就是打开了一个进程，很多功能是基于这个进程上的线程实现的。比如我们打开桌面上的QQ应用程序，首先我们点击QQ.exe然后就会看到登录界面，只有登录之后我们才能进入到我们的QQ账户中，在这个过程中，当我们点击QQ时，QQ这个进程就启动了，然后这个进程启动了一个登录的线程处理登录事件，当我们登录成功之后登录线程结束但是QQ进程还没有结束，QQ进程启动账户界面线程，当我们点击头像进行来聊天的时候，QQ进程启动聊天界面线程，同步线程来实现聊天和备份。

在Windows系统中我们可以按ctrl+alt+delete打开任务管理器，然后我们点击QQ，观察任务管理器的进程栏中的变化。

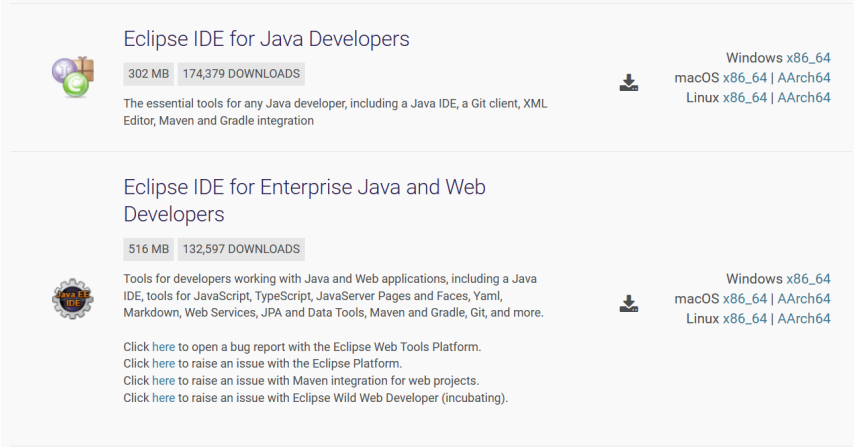
在 Java 语言中，线程是一种特殊的对象，它必须由 Thread 类或其子（孙）类来创建。通常有两种方法来创建线程：其一，使用型构为 Thread(Runnable) 的构造子类将一个实现了 Runnable 接口的对象包装成一个线程，其二，从 Thread 类派生出子类并重写 run 方法，使用该子类创建的对象即为线程。值得注意的是 Thread 类已经实现了 Runnable 接口，因此，任何一个线程均有它的 run 方法，而 run 方法中包含了线程所要运行的代码。线程的活动由一组方法来控制。Java 语言支持多个线程的同时执行，并提供多线程之间的同步机制（关键字为 synchronized）。

- 安全性：Java通常被用在网络环境中，为此，Java 提供了一个安全机制以防恶意代码的攻击。除了 Java 语言具有的许多安全特性以外，Java 对通过网络下载的类具有一个安全防范机制（类 ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查，并提供安全管理机制（类 SecurityManager）让 Java 应用设置安全哨兵。
- 解释型：Java的本质时属于解释型的语言。Java 程序在 Java 平台上被编译为字节码格式，然后可以在实现这个 Java 平台的任何系统中运行。在运行时，Java 平台中的 Java 解释器对这些字节码进行解释执行，执行过程中需要的类在联接阶段被载入到运行环境中。
- 动态性：Java的动态性体现在它使用反射机制。Java 语言的设计目标之一是适应于动态变化的环境。Java 程序需要的类能够动态地被载入到运行环境，也可以通过网络来载入所需要的类。这也有利于软件的升级。另外，Java 中的类有一个运行时刻的表示，能进行运行时刻的类型检查。
- 健壮性：Java 的强类型机制、异常处理、垃圾的自动收集等是 Java 程序健壮性的重要保证。对指针的丢弃是 Java 的明智选择。Java 的安全检查机制使得 Java 更具健壮性。
- 高性能：那些解释型的高级脚本语言相比，Java 的确是高性能的。事实上，Java 的运行速度随着 JIT(Just-In-Time）编译器技术的发展越来越接近于 C++。

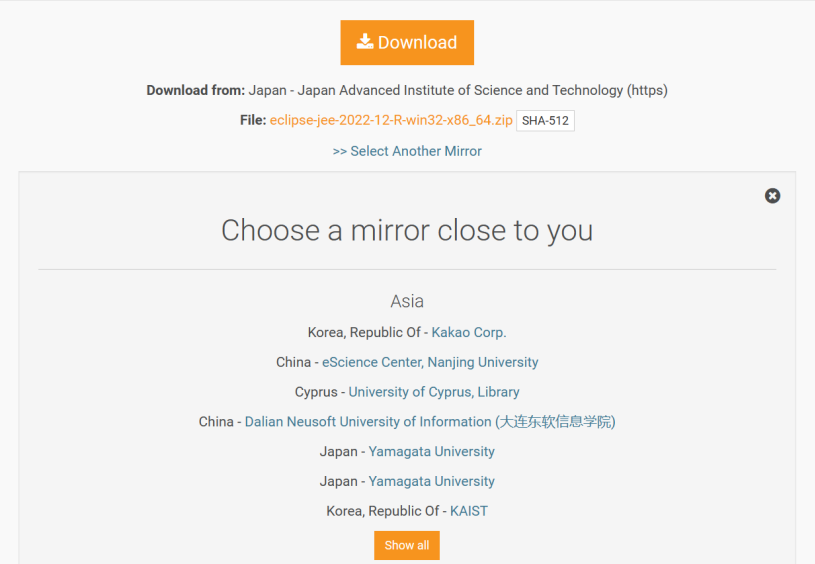
5，创建，编译和运行Java程序的过程

创建Java程序

- 这里所说的创建Java程序就是编写Java程序的源代码文件，所谓的源代码文件就是我们见到的.java文件。
- 创建Java程序可以借助于任何一个文本编辑器书写，比如Windows的记事本或者是NodePad编写或者Liunx下的vim。当然这种方式是最原始的开发方式，效率很低而且容易出错。
- “工欲善其事，必先利其器”，在现代的开发软件的过程中，集成开发环境IDE的使用使得软件的效率和质量得到大幅度的提升。所谓的集成开发环境就是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具。集成了代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件服务套。比如Visual Studio 2022,Visual Studio Code,IDEA,CLion,eclipse，sublime，IDA，webStrom，DeramWeaver，PyChorm，Navicat Premium，vc++6.0,DEVC++,CodeBlock，GoLang等。
- 对于开发Java语言来说最开始主流使用的是Eclipse，现在随着时代的发展目前IDEA被广泛的使用。所以现在我们需要了解Eclipse和IDEA的一些必要知识。
- Eclipse
    - 最初是由IBM公司开发的替代商业软件Visual Age for Java的下一代IDE开发环境，2001年11月贡献给开源社区，它由非营利软件供应商联盟Eclipse基金会（Eclipse Foundation）管理。
- 开源免费的，使用Java编写而成的IDE。所以如果要运行Eclipse就必须下载JRE。

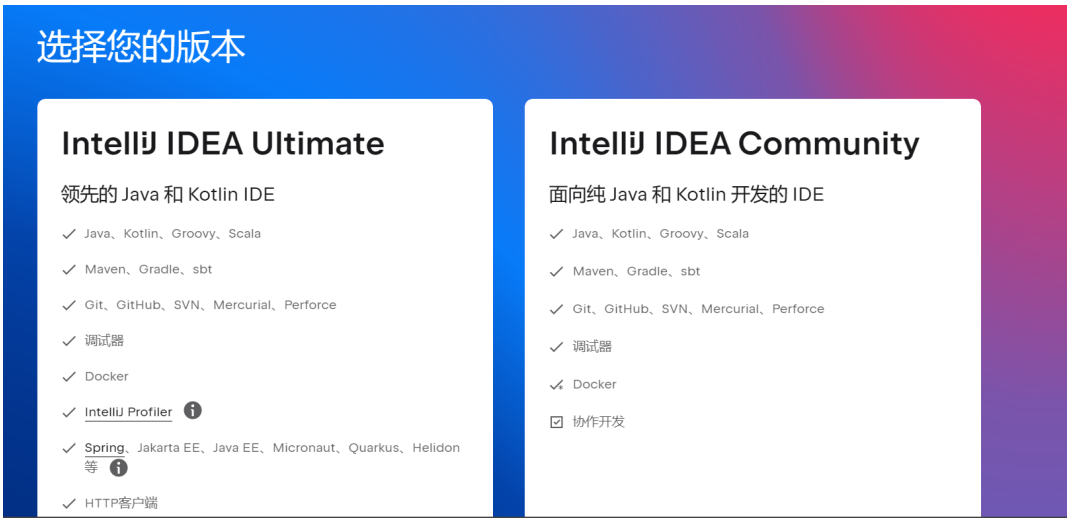
- 基于插件的可扩展的IDE，虽然大多数用户很乐于将 Eclipse 当作 Java 集成开发环境（IDE）来使用，但 Eclipse 的目标却不仅限于此。Eclipse 还包括插件开发环境（Plug-in Development Environment，PDE），这个组件主要针对希望扩展 Eclipse 的软件开发人员，因为它允许他们构建与 Eclipse 环境无缝集成的工具。由于 Eclipse 中的每样东西都是插件，对于给 Eclipse 提供插件，以及给用户提供一致和统一的集成开发环境而言，所有工具开发人员都具有同等的发挥场所。
- Eclipse属于绿色软件，所谓的绿色软件就是指的是它是基于压缩包的，只要解压之后就可以使用。删除压缩包即删除软件
- 一些软件，比如LOL这样的软件，当我们下载下来之后就会有安装引导程序指导我们去安装到Windows上，虽然我们可能安装的路径并不在系统盘上，但是我们会发现随着安装软件的数量越多，系统盘的存储量越少，原因就是这些软件运行的过程中会在系统盘理产生一些日志文件，临时文件，某些数据文件等，虽然它的主体并没有安装到系统盘中，但是如果作为安装软件，它在安装的时候需要将相关的信息写入到系统的注册表中。这样我们如果卸载的时候其实并不能够完全卸载干净。
- 下载和安装Eclipse。
- 进入官网https://www.eclipse.org/downloads/packages/
- 

- 注重看上面的两个版本，第一个版本的官方解释是The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration，第二个版本的官方解释是Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more，也就是说第二个版本比第一个版本多了一些开发web应用的支持，可以实现JavaEE的开发，所以我们下载第二个版本，当然下载第一个版本也是完全够用的。
- 进入到下载界面中，选择select Another Mirror，也就是选择其他的镜像源作为下载源，因为Eclipse是由外国的公司所维护的一款IDE，它在美洲，如果直接下载的话没有VPN或者是加速器的情况下将会很慢，所以选择其他的镜像源进行下载，网站会根据你所在的州给你推荐离你最近的镜像源，我们选择China - eScience Center, Nanjing University（中国-电子科技中心，南京大学）或者是China - Dalian Neusoft University of Information (中国-大连东软信息学院)



点击链接之后就会下载，之后会出现让你捐款的界面，当然这根据你自己的情况决定，虽然Eclipse是开源免费的，但是它的发展始终离不开经济的支持。

- IDEA
  - IDEA 全称 IntelliJ IDEA，是java编程语言的集成开发环境。IntelliJ在业界被公认为最好的Java开发工具，尤其在智能代码助手、代码自动提示、重构、JavaEE支持、各类版本工具(git、svn等)、JUnit、CVS整合、代码分析、 创新的GUI设计等方面的功能可以说是超常的。IDEA是JetBrains公司的产品，这家公司总部位于捷克共和国的首都布拉格，开发人员以严谨著称的东欧程序员为主。它的旗舰版还支持HTML，CSS，PHP，MySQL，Python等。免费版只支持Java,Kotlin等少数语言。
  - IDEA也可以扩展插件使用，它是一款安装软件。
- 它是目前最主流的开发Java产品的IDE，它不像Eclipse那样是免费开源的，它是由JetBrains公司所开发和维护的，所以在一些专业版本中它是需要钱的而且只能是按年租用，很贵。但是它的智能化和提示化的开发方式的确要比其他IDE甚至是Eclipse要强很多。
- 下载安装IDEA首先进入到官网中进行下载https://www.jetbrains.com/zh-cn/idea/
- 找到以下页面



第一个就是旗舰版的（专业版）它支持多种语言的开发，如果要进行JavaEE开发或者安卓开发，那么只需要一个IDEA即可，但是它是付费的，当然它有30天的免费试用，它的费用是按年收取的，个人版的在$168。第二个版本是社区版（免费版）这个版本的IDEA不能进行JavaEE的开发，它只能够支持JavaSE开发和基本的安卓开发。

- 就目前JavaSE这个阶段的话可以安装Eclipse的JavaEE版本或者是JavaSSE版本进行学习，也可以下载IDEA的免费版进行学习，当到JavaEE阶段的时候，使用最多的还是IDEA专业版。当然现在安装IDEA免费版到IDEA专业版的使用就会更熟练一些。

编译和执行Java程序

- 计算机语言

  这里我们只需要了解一些必要的关于计算机语言的知识就可以了，计算机语言作为一种交叉学科，阐述本质的学科是编译原理和操作系统。

  - 什么是计算机语言?

    可以这样理解，人与人之间的沟通需要通过语言我们才知道对方所说的意思和对方想请我们去做的事是什么和该如何去规范的做这件事。现在的计算机我们可以看到，它并不是传统意义上的计算机，它是一种能够完成特定功能，智能化的，科技化的，类人化的，不局限于计算功能的事物，所以它也称为电脑。为了让电脑能够完成我们想要的一些业务功能比如能够远程的聊天，我们就需要使用一种电脑听得懂的话和电脑交流告诉他如何实现这个功能，需要它调用什么功能和执行什么命令，这就是计算机语言。

  - 计算语言的发展

    下面所说的高级和低级并不是指的是这个语言的等级，它指的是离计算机硬件底层的距离。低级表示该语言的开发离计算机的底层硬件很近，依赖于底层的硬件。

    - 机器语言/低级语言（第一代计算机语言）

      是最早的计算机语言，它是一系列二进制数字组成的。面向机器的语言。

      最早的计算机时非常庞大的，它是由电子管或晶体管组成的，所占面积有两个足球场那么大，运行起来也非常容易出现损坏，计算速度很慢，当时的计算机可以说只能被一些专家使用在像航空航天，原子弹等这样的高精尖领域中，而且它的编程方式是使用打孔机，在纸袋上打孔，可以说是用纯二进制数字的方式编写代码，也就是010101.....。看起来如同天书一般，如果对机器的硬件特点和结构不是很了解的话是无法看懂这一串串的010101.....的，把打孔的纸送入到机器的激光点录入到机器的内存中进行运行，机器通过激光点判断0101的组合，纸带的每一行规定了8个孔位，每个孔位代表1bit，可以打孔也可以不打，打孔代表1，不打代表0。那是的编程人员需要穿戴着旱冰靴左右来回移动的进行工作。不同的机器所写的机器指令是不同的，所以当时很多专家可能只会对一种计算机类型进行编程。很难使用和掌握。

      虽然说机器语言的编程方式非常的麻烦，但是目前我们现代所使用的无论是笔记本，台式机还是大到超级计算机，服务器，小到各种嵌入式的设备，智能手表，智能手机，汽车的单片机等它们都是基于二进制的机器。

      也就是说计算机只能识别机器语言，无论我们使用汇编，C,C++或者是Java，python编写的程序最终都会以机器语言（机器指令，机器码）的方式被计算机执行。

    - 汇编语言/低级语言（第二代计算机语言）

      第二代计算机编程语言，它采用助记符（一些具有含义的英文单词）的方式编写程序。面向机器的程序设计语言。

      由于二进制语言使用纯二进制数字编写，可以说是特别的难记和难以被常人理解，当出现错误的时候非常难以改正。计算机也就非常难以被推广和使用，为了解决这个状态，自然而然的就出现了汇编语言。它采用助记符的方式书写程序，每一个助记符代表一串特定的二进制机器语言。比如下面所展示的。可以看到具体机器的机器指令和汇编语句的对应关系，其实这种写法只是为了展示一条汇编语句对应一串特定的机器指令。

      ```
      机器指令：   1000100111011000
      汇编指令：   MOV  AX，BX
      上述的机器指令和汇编指令是一一对应的，它们操作的含义都是：把寄存器 BX 中的内容送到 AX中。
      ```

      在实际的反汇编查看中是按照十六进制展示的。

```
08048423 <main>:
 8048423:       55                      push   %ebp
 8048424:       89 e5                   mov    %esp,%ebp
 8048426:       83 e4 f0                and    $0xfffffff0,%esp
 8048429:       83 ec 10                sub    $0x10,%esp
 804842c:       c7 04 24 14 85 04 08    movl   $0x8048514,(%esp)
 8048433:       e8 ec fe ff ff          call   8048324 <puts@plt>
 8048438:       e8 b7 ff ff ff          call   80483f4 <f1>
 804843d:       c7 04 24 1a 85 04 08    movl   $0x804851a,(%esp)
 8048444:       e8 db fe ff ff          call   8048324 <puts@plt>
 8048449:       b8 00 00 00 00          mov    $0x0,%eax
 804844e:       c9                      leave
 804844f:       c3                      ret
```

它使用英文单词作为特定的助记符号，在一定程度上使得程序容易理解和显示其含义，相对于机器语言来说它更加接近于我们人类的阅读习惯。

实际上汇编语言是通过汇编语言编译器将汇编语言转化为机器语言，特定的汇编语言和特定的机器语言指令集是一一对应的，它是面向机器的语言，也就是说它的依赖于具体机器的硬件进行编写的，不同机器的汇编指令的语法和实现结构是不同的，所以在不同的机器之间是不能够直接进行移植的。并且使用汇编语言进行开发的人员必须对于特定机器的存储结构，整个硬件的架构和组成原理都掌握了解才能使用汇编语言开发一些应用程序。

目前虽然高级语言的飞速发展使得汇编语言在一定程度上并不常见，但是汇编语言在目前常常用来编写底层的硬件上的一些程序，比如我们计算机中最重要的BIOS，51单片机和一些针对于芯片的开发比如Intel和AMD，同时它也是实现一些操作系统的底层架构的重要语言 。可以说它目前来讲在做一些背后的支撑工作。

- 高级语言（第三代编程语言）

  第三代计算机编程语言，它是一种独立于具体机器的，参考数学语言设计的近似我们人类能够读懂的日常语言。

  随着时代的发展，从集成电路到大规模集成电路在计算机中的应用，计算机已经变得非常的小巧便携，从台式机到笔记本再到智能手机，以及目前的智能穿戴设备，VR，以及更低价格更高性能更便携的计算机和电子设备的出现，再加上网络的大力发展，可以说现在的计算机已经深入到社会的各行各业同时渗透到我们生活的方方面面，成为我们生活的必须品。而这样的景象来自于软件带来的生活和工作的巨大便利和推动社会的革新，软件就是计算机的灵魂，没有灵魂的计算机是毫无意义的，各种各样的不仅限于专业计算的方便我们生活的软件被开发出来，而现在的编程是使用各种高级语言进行开发。

  高级语言是最接近于我们人类所能够理解和读懂的语言，比如Java，C，C++,Python.....这样的程序设计语言，同时可以看到高级语言不是一种语言而是一些列语言的集合。比如下面进行加法运算，使用高级语言开发的语句。

  ```
  int i = 2;
  int sum = i+2;
  ```

  作为用户层面的编程工具，用户并不需要去了解硬件的结构，而是去用逻辑的语言去实现想要的目标。基本脱离了机器的硬件系统，用人们更易理解的方式编写程序。编写的程序称之为源程序。

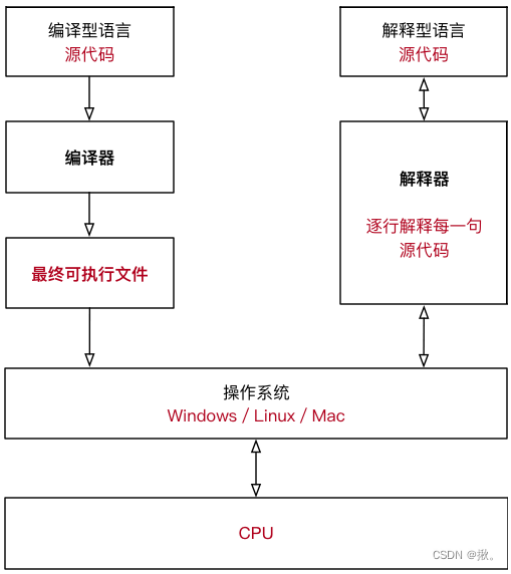
实际上使用高级语言编写的程序最后在计算机中执行都是要转换为机器语言才能够被执行的，只是这个转化的过程并不是由我们操作而是通过编译器或者解释器操作的，这背后是一些列的操作系统资源的调用，CPU和内存，硬盘的交互操作的复杂机制实现的。所以可以说高级语言"看不见"机器硬件结构。由于这一特性高级语言不能直接用于编写访问机器硬件资源的系统软件或设备控制软件，并且运行速度比汇编程序要低，同时因为高级语言比较冗长，所以代码的执行速度也要慢一些。为了能够访问底层有些高级语言提供汇编接口，有些使用相当的内存模型进行映射和提高底层功能进行操作等来实现对底层硬件的访问。

高级语言分类

对于高级语言我们常常按照语言的编程思想进行分类，下面列举几种目前主流的高级语言。

- 面向过程的编程语言：C
- 面向对象的编程语言：C++,Java，C#，Python，Swift，Object-c，GoLang，Rust，Lua

- 编译型语言，解释型语言，动态语言，静态语言 ，动态类型语言，静态类型语言强类型语言，弱类型语言



- 编译型语言（compare）
  - 原理

    通过专门的编译器，将所有源代码一次性转换成特定平台（Windows、Linux 等）执行的机器码（以可执行文件的形式存在）。
  - 特点

    编译型语言经过编译器转成可执行程序后，可执行程序里面包含的就是机器码。只要拥有可执行程序，就可以随时运行，不用再重新编译了，即 一次编译，无限次运行。

    运行可执行程序时，不再需要源代码和编译器，因此编译型语言可以脱离开发环境运行。

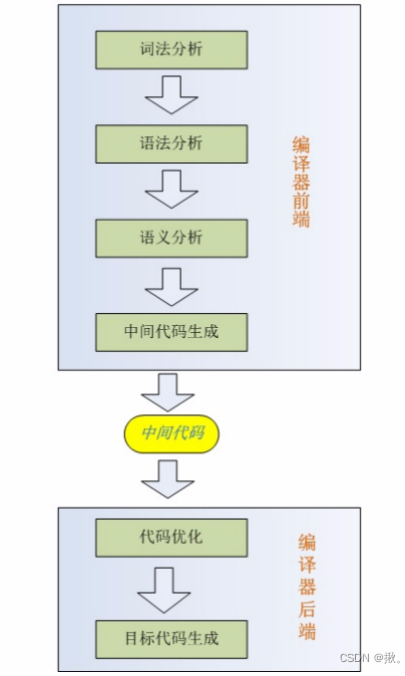
    编译型语言一般不能跨平台，即不能在不同的操作系统之间随意切换。
  - 优点

    编译一次后，脱离了编译器也可以运行，并且运行效率高。
  - 缺点

    可移植性差，不够灵活。
  - 类型

前端编译器：编译器的分析阶段也称为前端，它将程序划分为基本的组成部分，检查代码的语法、语义和语法，然后生成中间代码。分析阶段包括词法分析、语义分析和语法分析。

后端编译器：编译器的合成阶段也称为后端，优化中间代码，生成目标代码。合成阶段包括代码优化器和代码生成器。



- 编译型语言总结

参数机制	Compare
程序步骤	1、创建代码 2、Compile将解析或分析所有语言语句的正确性。如果不正确，则抛出错误 3、如果没有错误，编译器将把源代码转换为机器码 4、它将不同的代码文件链接到一个可运行的程序(称为exe) 5、运行程序
输入 (Input)	整个程序
输出 (Output)	生成中间目标代码
工作机制	编译在执行之前完成
存储	存储编译后的机器代码在机器上
执行	程序执行与编译是分开的，它只在整个输出程序编译后执行
生成程序	生成可以独立于原始程序运行的输出程序(以exe的形式)
修改	如果需要修改代码，则需要修改源代码，重新编译
运行速度	快
内存	内存需求更多的是由于目标代码的创建
错误	编译器在编译时显示所有错误和警告。因此，不修正错误就不能运行程序，当程序中出现错误时，它会停止翻译，并在删除错误后重新翻译整个程序。
错误监测	难，编译器同时显示所有错误，很难检测错误
编程语言	C、C++、Golang、Pascal（Delphi）

- 不可跨平台表现
  - 可执行程序不能跨平台

    编译器将编译型语言编译生成可执行程序，这个可执行程序不能跨平台。因为不同操作系统对可执行程序的内部结构有着截然不同的要求，彼此之间也不能兼容。 比如，不能将 Windows 下的可执行程序拿到 Linux 下使用，也不能将 Linux 下的可执行程序拿到 Mac OS 下使用（虽然它们都是类 Unix 系统）。

    相同操作系统的不同版本之间也不一定兼容。 比如不能将 x64 程序（Windows 64 位程序）拿到 x86 平台（Windows 32 位平台）下运行。但是反之一般可行，因为 64 位 Windows 对 32 位程序作了很好的兼容性处理
  - 源代码不能跨平台

    不同平台支持的函数、类型、变量等都可能不同，基于某个平台编写的源代码一般不能拿到另一个平台下编译。以C语言为例。

示例	Windows平台	Linux 平台
“睡眠”函数	Sleep(x)，x为毫秒	sleep(x)，x为秒
long 类型的长度	（64 位）占 4 字节	（64 位）占8 字节

编译型语言虽然是高级语言，却与平台联系比较紧密（操作系统是用编译型语言和汇编编写的），针对不同平台（操作系统）可能有其特定的编写规则。因此编译型语言不能像解释型语言一样，实现源代码跨平台

- 解释型语言（Interpreter）
  - 原理

    由专门的解释器，根据需要将部分源代码临时转换成特定平台的机器码。
  - 特点

    解释型语言，每次执行程序都需要一边转换一边执行。

    因为每次执行程序都需要重新转换源代码，所以无法脱离开发环境，所以解释型语言的执行效率天生就低于编译型语言，甚至存在数量级的差距。

    解释型语言 能跨平台，即实现“一次编写，到处运行”。
  - 优点

    跨平台性好，通过不同的解释器，将相同的源代码解释成不同平台下的机器码。
  - 缺点

    一边执行一边转换，效率很低。
  - 解释型语言总结

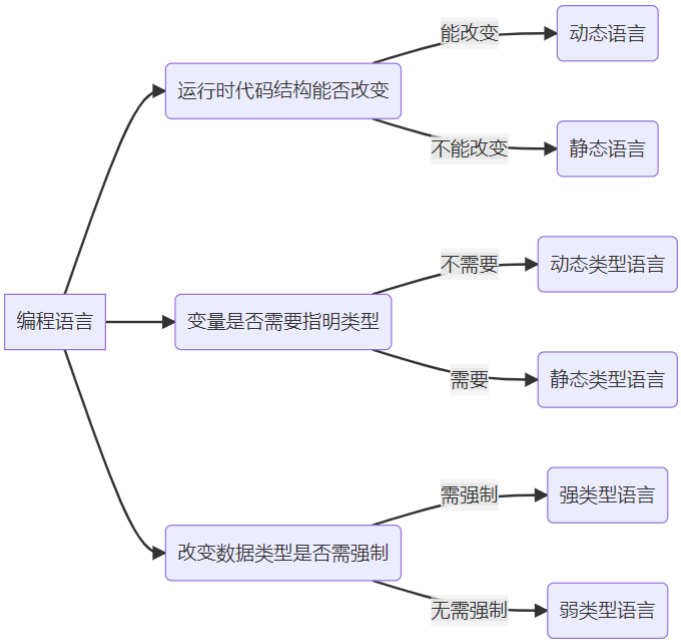
参数机制	Compare
程序步骤	1、创建代码 2、没有文件链接或机器代码生成 3、源语句在执行过程中逐行执行
输入 (Input)	每次读取一行
输出 (Output)	不保存任何机器代码
工作机制	编译和执行同时进行
存储	存储编译后的机器代码在机器上
执行	程序执行是解释过程的一部分，因此是逐行执行的
生成程序	不生成输出程序，所以在每次执行过程中都要评估源程序
修改	直接修改就可运行
运行速度	慢
内存	它需要较少的内存，因为它不创建中间对象代码
错误	解释器读取一条语句并显示错误(如果有的话)。你必须纠正错误才能解释下一行，当解释器中发生错误时，它会阻止其翻译，在删除错误后，翻译将继续
错误监测	容易，解释器则逐个显示每条语句的错误，更容易检测错误
编程语言	Python、JavaScript、PHP、Shell、MATLAB

- 跨平台表现

  解释型语言的跨平台，指的是源代码跨平台，而不是解释器跨平台。

  解释器是一个将源代码转换成机器码的可执行程序，可执行程序是不能跨平台的。因此官方需要针对不同平台开发不同的解释器，这些解释器必须要能够遵守同样的语法，识别同样的函数，完成同样的功能。

  因为有了解释器这个中间层，解释器帮助我们屏蔽了不同平台之间的差异。在不同的平台下，不同的解释器会将相同的源代码转换成不同的机器码，所以才让同样的代码在不同平台的执行结果相同。



- 动态语言（Dynamic Programming Language）
  - 什么是?

    一类在运行时可以改变其结构的语言。强调改变代码结构，例如新的函数、对象、甚至代码可以再运行时被引进，已有的函数可以在运行时被删除或是其他结构上的变化。
  - 代表语言

    Object-C、C#、JavaScript、PHP、Ruby、Python、Erlang。
- 静态语言（Statical Programming Language）
  - 什么是?

    与动态语言相对应的，运行时结构不可变的语言就是静态语言。强调改变代码结构。
  - 代表语言

    Java、C、C++。
- 动态类型语言（Dynamically Typed Language）
  - 什么是?

    在运行时，确认数据类型的语言，变量在使用之前无需申明类型，通常变量的值是被赋值的那个值的类型。强调检查数据类型。
  - 举例

    ```
    var s="hello";
    s=1;
    s=true;
    ```
  - 优点

    思维不受束缚，可以任意发挥，把更多的精力放在产品本身。集中思考业务逻辑实现，思考过程即实现过程。
  - 缺点

    代码运行期间有可能会发生与类型相关的错误。
  - 代表语言

PHP、ASP、JavaScript、Python、Perl、SQL、Ruby、ABAP、Unix Shell

- 静态类型语言（Statically Typed Language）
  - 什么是?

    在编译时，变量的数据类型就可以确定的语言。多数静态类型语言要求在使用变量之前必须声明数据类型。强调检查数据类型。

    - 举例

      ```
      String s = "hello";
      int i=0;
      double d = 3.14;
      boolean b=true;
      ```

    - 优点

      由于类型的强制声明，IDE(集成开发环境)有很强的代码感知能力，因此，在实现复杂的业务逻辑，开发大型商业系统，以及那些生命周期很长的应用中，依托IDE对系统的开发很有保障。
      由于静态类型语言相对比较封闭，使得第三方开发包对代码的侵害性可以降到最低。

    - 缺点

      开发代码的时候，需要格外注意变量的类型。过多的类型声明会增加更多的代码。

    - 代表语言

      C、C++、Java、Delphi、C#

    - 静态语言=编译型语言，动态语言=解释型语言?

      既然编译型语言需要将所有源代码一次性转换成可执行程序，解释型语言是逐行解释每一句源代码，是否意味着编译型语言就是静态语言，代码结构无法改变；而解释型语言因为是逐行运行可以随意改变未运行源代码的代码结构呢?
      反例：Object-C 是编译型语言，但是他是动态语言。得益于特有的 run time 机制（准确说 run time 不是语法特性是运行时环境，这里不展开）OC 代码是可以在运行的时候插入、替换方法的。

- 强类型语言（Strongly Typed Language）
  - 什么是?

    强制数据类型定义的语言。
    一旦一个变量被指定了某个数据类型，如果不经过强制转换，那么它就永远是这个数据类型。强调转换数据类型。

    - 举例

      不能把一个整形变量当成一个字符串来处理。也就是说

      ```
      int s = 1;
      String s = "hello";//编译错误，S已经定义为了int类型的数据
      ```

    - 代表语言

      Java、C#、Python、Object-C、Ruby

- 弱类型语言（Weakly Typed Language）
  - 什么是?

数据类型可以被忽略，一个变量可以赋不同数据类型的值。强调转换数据类型。

- 举例

  一个变量可以赋不同数据类型的值。

```
  var s="hello";//此时s为String类型
s = 1;//此时s为number类型
```

- 代表语言

  JavaScript、PHP、python。

- 与强类型语言比较

  强类型定义语言在速度上可能略逊色于弱类型定义语言，但是强类型定义语言带来的严谨性能够有效的避免许多错误。

- 静态类型语言=强类型语言，动态类型语言=弱类型语言?

  一个语言是不是强类型语言和是不是动态类型语言也没有必然联系。 Python 是动态类型语言，是强类型语言。JavaScript 是动态类型语言，是弱类型语言。Java 是静态类型语言，是强类型语言。

- 什么是计算机BUG

  - 计算机BUG是指计算机系统的硬件，系统软件或者是应用软件中隐藏的一些错误或者是漏洞，在运行的时候可能会危害到计算机的正确性和安全性，使计算机产生逻辑或物理上的故障。
  - BUG在英文单词中是指虫子的意思，为什么它会成为计算机术语呢，因为第一代计算机是由许多庞大且昂贵的真空管组成，并利用大量的电力来使真空管发光。可能正是由于计算机运行产生的光和热，引得一只Bug钻进了一支真空管内，导致整个计算机无法工作。研究人员费了半天时间，总算发现原因所在，把这只Bug从真空管中取出后，计算机又恢复正常。后来，Bug这个名词就沿用下来，表示电脑系统或程序中隐藏的错误、缺陷，漏洞或问题。
  - Debug可以在多的IDE中见到，我们平常也会经常使用，它的英文原意是捉虫子，也就是人们发现错误并且及时纠正的过程。在实际的工程开发中这类工作由程序员，用户和软件测试人员逐步完成。

- 为什么是二进制计算机

  这部分的知识可以参考计算机组成原理，数字电路和模拟电路的相关知识。

  目前我们所使用的电子设备或者说是计算机都是采用二进制的方式的方式来处理各种各样的数据。那么为什么要这样设计呢？我们人类在日常生活中使用的数制度是10进制的数制，也就是0-9而计算机采用的是0和1，这和计算机的本质有关。

  首先我们应该了解的是以下几点

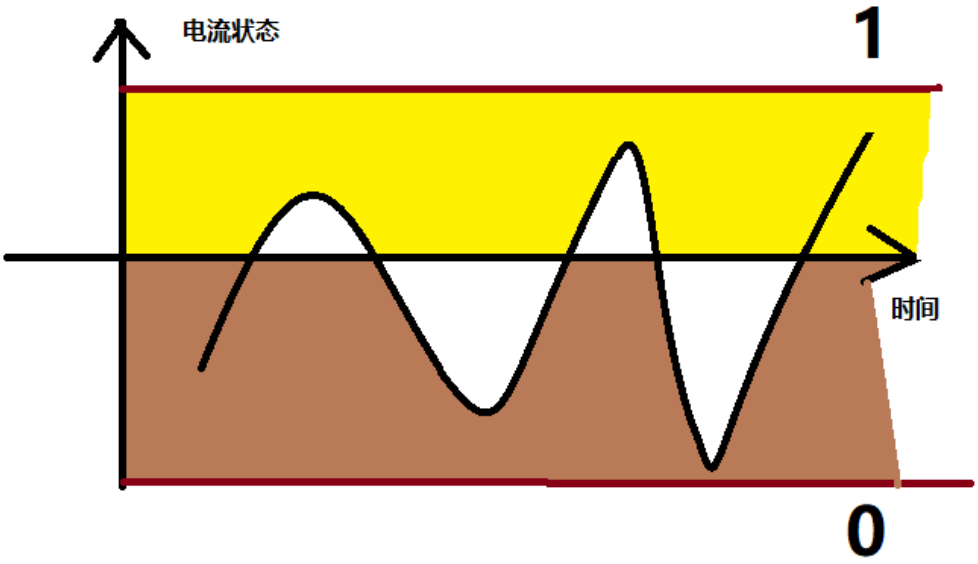
  - 实际上计算机也不认识0和1,0和1是用来表示计算机的一种状态，计算机只有放电和充电的两个过程，放电为0充电为1,1010的就组合成了多种的形态。
  - 那么计算机是如何利用充电放电原理的呢？计算机中有CPU，它其中有一个震荡发生器，比如Intel core i7- ....GHz,GHz指的就是0和1之间频繁的切换频率，切换速度越快逻辑电路翻转越快，计算速度越快。
  - 一位二进制只能表示2种组合，只能是0和1，那我想表示其他的数字该如何表示？我们可以用扩展位数的方式，如果我扩展位两位去表示，那它就有4种表示形式00,01,10,11,这也符合逻辑，00->0,01->1,10->2,11->3，那么100->4,101->5,110->6，111->7，1000->8。
  - 我们看到使用二进制也可以以这样的计算方式不断的表示数字，那能有一个界限吗？我们规定8位二进制称为一个字节，这表示一个界限,比如：在Java中int型数据的取值范围是-2^31~2^31，它有4个字节的长度界限。

- 既然存在正数，那如何表示其负数形式呢？通常最高位表示它的符号位其他的7位表示数字位，因此这就解释了为什么一个字节能表示的最大值是127而不是128。

  接下来我们可以总结到为什么是二进制的计算机

  - 第一，现在的计算机都是采用电作为能源的设备，我们知道电的使用有通电，断电，这就正好符合了二进制所对应的0和1。
- 第二，计算机是由逻辑电路组成的，而逻辑电路只有两个状态开关的接通和断开，这种状态正好可以使用二进制的0和1表示。
  - 第三，计算机最主要的作用就是运算和处理各种数据，计算机可以进行逻辑运算，算数运算，位移运算，二进制的0和1组成的序列可以很方便的转换为八进制，十进制和十六进制；二进制的0和1，可以表示逻辑代数的true和false；二进制的0和1可以根据二进制的计算特定+1位移，实现位移运算。
- 第四，二进制更简单，它只有0和1两个数值组成，实现起来简单，在CPU中的运算器的结构可以得到简化，有益于提高运算的性能。
  - 上述的四条是为什么计算机要使用二进制的本质原因。除此之外还有一条就是。
- 第五，抗干扰能力强，不易造成数据的损失，我们知道电流会受到周围的磁场的影响，这会使得电流并不是恒稳定的，电流只是在一个范围之内上下波动，而设定1表示电流向上，0表示电流波动向下，这样电流被限定在一个大的范围中。

既然如此那么现在思考计算机可以使用其他进制来进行表示吗？



- 因为上述图像可以看到电流是呈现出波动状态如果我在芯片中使用更加精密的探测仪器经过各种环境下的电流波动测试，选定缩小到一定的范围表示0，1，2，3，甚至于-1，-2，0，1，2，3，4等等状态但是这就要求更精确的仪器和更好的融错机制，以及更多的经济投入。特别是在大规模集成电路的时代这样的要求几乎的不可能实现的。而且不同进制的出现，那么当前的编程方式将不能够满足，整个计算机体系将会发生颠覆性的改变。其实历史上存在过三进制的计算机-1，0，1，是由前苏联开发的，但是由于各种政治原因和二进制计算机的流行使得二进制计算机成为一种标准，所以三进制的计算机并没有再深入研究和推广。
- 目前计算机科技的最前沿已经开始向着量子计算机，分子计算机，光子计算机和纳米计算机的方向进行研究。
- 目前最火热的是量子计算机，它通过量子力学规律以实现数学和逻辑运算，处理和储存信息。在量子计算机中其硬件的各种元件的尺寸达到原子或分子的量级。量子计算机是一个物理系统，它能存储和处理用量子比特表示的信息量子计算机和许多计算机一样都是由许多硬件和软件组成的，软件方面包括量子算法、量子编码等，在硬件方面包括量子晶体管、量子存储器、量子效应器等。如果它能够实现，那么它和传统计算机的最大

区别就是在量子计算机中，基本信息单位是量子比特（qubit），用两个量子态|0>和|1>代替经典比特状态0和1。量子比特相较于比特来说，有着独特的存在特点，它以两个逻辑态的叠加态的形式存在，这表示的是两个状态是0和1的相应量子态叠加。同时可以利用量子纠缠的特性使得量子计算机的安全性更高。

- 低代码和人工智能

这里对低代码和人工智能辅助开发进行相关的认识。计算机行业的发展速度是飞快的，虽然真正意义上的计算机诞生至今不超过百年的时间，但是计算机在近20年的发展是不可估量的，特别是互联网的发展和web2.0的时代，当时的马云，马化腾抓住浪潮，阿里，淘宝，腾讯一跃成为互联网公司巨头，百度，360等也抓住机遇顺势而上，随着通信技术和软件产业的革新发展，不少人靠着互联网的机会一跃而上，跨越阶层，京东，拼多多，各种教育，金融等网站相继而起，许多的行业，银行，会计，建筑，机械等都需要借助于计算机的辅助设计和开发属于自己业务的软件系统，我们可以对比从2003年到2013年的发展，我们所看到的景象和我们所经历的这10年是什么状态。再看从2013年到2023年的这10年间我们的生活状态 和我们的生活所发生的变化，这是巨大的变化。现在计算机行业已经成为了各行各业所必须的专业配置，计算机行业的高速发展引领着更多的商业形式和技术领域，现在人工智能，机器学习，深度学习，神经网络，云计算，大数据等领域正在大力发展，元宇宙 （web3.0），区块链等新概念正在被相继实现。

- 低代码

究竟需要什么样的新技术，才能真正解放IT生产力，加速社会数字化转型，Make The World Great Again？我认为是低代码（Low-Code）。基于经典的可视化和模型驱动理念，结合最新的云原生与多端体验技术，低代码能够在合适的业务场景下实现大幅度的提效降本，为专业开发者提供了一种全新的高生产力开发范式（Paradigm Shift）。另一方面，低代码还能让不懂代码的业务人员成为所谓的平民开发者（Citizen Developer），弥补日益扩大的专业人才缺口，同时促成业务与技术深度协作的终极敏捷形态（BizDevOps）。

- 什么是低代码

低代码开发平台（LCDP）本身也是一种软件，它为开发者提供了一个创建应用软件的开发环境。也就是说它和我们平时使用的IDE是一类的，但是它不同于传统的IDE，它提供的是更高维和易用的可视化IDE。

大多数情况下，开发者并不需要使用传统的手写代码方式进行编程，而是可以通过图形化拖拽、参数配置等更高效的方式完成开发工作。也就是说如果我作为一名会计，为了更高效的完成公司的报账任务，我需要搭建一款财务管理系统，此时要么请专业的外包公司进行开发，要么经过基本的培训使用低代码，通过图形化的拖拉拽的方式搭建一款财务管理系统。

低代码的概念并不是近几年才出现的，早在2014年的时候低代码这个概念就被提出了。

- 低代码优势

低代码开发平台能够降低业务应用的开发成本。一方面，低代码开发在软件全生命周期流程上的投入都要更低（代码编写更少、环境设置和部署成本也更简单）；另一方面，低代码开发还显著降低了开发人员的使用门槛，非专业开发者经过简单的IT基础培训就能快速上岗，既能充分调动和利用企业现有的各方面人力资源，也能大幅降低对昂贵专业开发者资源的依赖。

低代码开发平台能够实现业务应用的快速交付。也就是说，不只是像传统开发平台一样“能”开发应用而已，低代码开发平台的重点是开发应用更“快”。更重要的是，这个快的程度是颠覆性的：根据Forrester在2016年的调研，大部分公司反馈低代码平台帮助他们把开发效率提升了5-10倍。有理由相信，随着低代码技术、产品和行业的不断成熟，这个提升倍数还能继续上涨。

- 低代码的核心能力

全栈可视化编程：可视化包含两层含义，一个是编辑时支持的点选、拖拽和配置操作，另一个是编辑完成后所及即所得（WYSIWYG）的预览效果。传统代码IDE也支持部分可视化能力（如早年Visual Studio的MFC/WPF），但低代码更强调的是全栈、端到端的可视化编程，覆盖一个完整应用开发所涉及的各个技术层面（界面/数据/逻辑）。

全生命周期管理：作为一站式的应用开发平台，低代码支持应用的完整生命周期管理，即从设计阶段开始（有些平台还支持更前置的项目与需求管理），历经开发、构建、测试和部署，一直到上线后的各种运维（e.g. 监控报警、应用上下线）和运营（e.g. 数据报表、用户反馈）。

低代码扩展能力：使用低代码开发时，大部分情况下仍离不开代码，因此平台必须能支持在必要时通过少量的代码对应用各层次进行灵活扩展，比如添加自定义组件、修改主题CSS样式、定制逻辑流动作等。一些可能的需求场景包括：UI样式定制、遗留代码复用、专用的加密算法、非标系统集成。

- 为什么需要低代码？

  低代码是什么可能并没那么重要，毕竟在这个信息爆炸的世界，永远不缺少新奇而又短命的事物。大部分所谓的新技术都只是昙花一现：出现了，被看到了；大部分人“哦”了一声，已阅但表示不感兴趣；小部分人惊叹于它的奇思妙想，激动地点了个赞后，回过头来该用什么还是什么。真正决定新技术是否能转化为新生产力的，永远不是技术本身有多么优秀和华丽，而是它是否真的被需要。

  “互联网+”和“数字化转型”的时代，企业越来越需要通过应用（App）来改善企业内部的信息流转、强化与客户之间的触点连接。然而，对于我国来说诞生还不太久的IT信息时代，也正面临着与我国社会主义初级阶段类似的供需关系矛盾：落后的软件开发生产力跟不上人民日益增长的业务需求。2021年应用开发需求的市场增长将至少超过企业IT交付能力的5倍。面对如此巨大的IT缺口，如果没有一种革命性的“新生产力”体系，很难想象仅凭现有传统技术体系的发展延续就能彻底解决问题。

  虽然软件行业一直在高速发展，新的语言、框架和工具层出不穷，但作为从业者我们不得不承认：软件开发仍处于手工作坊阶段，效率低、人力成本高、质量不可控。项目延期交付已成为行业常态，而瓶颈几乎总是开发人员（对机器能解决的问题都不是问题）；优秀的开发人才永远是稀缺资源，没有几个大的公司雇的起；软件质量缺陷始终无法收敛，线上故障频发资损不断。

  相比而言，传统制造业经过几百年工业革命的发展，大部分早已摆脱了对“人”的强依赖：从原料输入到制品输出，中间是各种精密仪器和自动化流水线的稳定支撑，真正实现生产的标准化和规模化。虽然信息化号称是人类的第三次工业革命，但以软件行业目前的状况，远远还没到达成熟的“工业化”阶段。

  - 应用软件开发工业化：

    低代码正在将应用软件开发过程工业化：每个低代码开发平台都是一个技术密集型的应用工厂，所有项目相关人员都在同一条产线内紧密协作。借助应用工厂中各种成熟的基础设施、现成的标准零件、自动化的装配流水线，开发者只需要专注于最核心的业务价值即可。即便是碰到非标需求，也可以随时自己动手，用最灵活的手工定制（代码）方式来解决各种边角问题。它能让业务人员实现自助式（self-service）应用交付，既解决了传统IT交付模式下的任务堆积（backlog）问题，避免稀缺的专业开发资源被大量简单、重复性的应用开发

需求所侵占，也能让业务人员真正按自己的想法去实现应用，摆脱交由他人开发时不可避免的。

- 扩大应用开发劳动力

  通过让大部分开发工作可以仅通过简单的拖拽与配置完成，低代码（包括零代码）显著降低了使用者门槛，让企业能够充分利用前面所提到的平民开发者资源。

- 加强开发过程的沟通协作

  多方调查结果显示，软件项目失败的最主要原因之一就是缺乏沟通（poor communication）。传统开发模式下，业务、产品、设计、开发、测试与运维人员各司其职，且各有一套领域内的工具和语言，长久以来很容易形成一个个“竖井”（silos），让跨职能的沟通变得困难而低效。这也是为什么当前热门的敏捷开发和DevOps都在强调沟通（前者是协同Biz与Dev，而后者是协同Dev和Ops），而经典的DDD领域驱动设计也主张通过“统一语言”来减少业务与技术人员之间的沟通不一致。

  有了低代码后，这一状况将得到根本改善：上述各角色都可以在同一个低代码开发平台上紧密协作（甚至可以是同一个人），这种全新的协作模式不仅打破了职能竖井，还能通过统一的可视化语言和单一的应用表示（页面/数据/逻辑），轻松对齐项目各方对应用形态和项目进度的理解，实现更终极的敏捷开发模式，以及在传统DevOps基础之上更进一步的BizDevOps。

- 低代码和专业开发者

  虽然零代码确实是设计给非专业开发者用的，但其所能支撑的业务场景确实有限，无法真正革新传统开发模式，替代那些仍需专业开发者参与的复杂业务场景。而狭义上的低代码却有潜力做到这一点，因为它天生就是为专业开发者而量身定制的。Gartner最近的一项调研报告显示，“66%的低代码开发平台用户都是企业IT部门的专业开发者”。这充分说明了，专业开发者比平民开发者更需要低代码。

  这里专业的开发者就会发问：低代码都不怎么写代码了，怎么能算是为我们程序员服务呢？”。虽然程序员讨厌重复自己，但重要的事情还是得多说一遍：开发 ≠ 写代码。1万年前蹲在洞穴里的原始人，在用小石子画远古图腾；100年前坐在书桌前的徐志摩，在用钢笔给林徽因写情书；而今天趴在屏幕前的很多人，相信都已经开始用上手写板或iPad涂涂写写了。千百年来，人类使用的工具一直在演进，但所从事活动的本质并没有多大改变。无论是用小石子还是小鼠标，写作绘画的本质都是创造与表达，最终作品的好坏并不取决于当时你手中拿着什么；同样地，应用开发的本质是想法和逻辑，最终价值的高低也不取决你实现时是用的纯代码还是低代码。

  而相比纯代码而言，低代码极有可能成为更好的下一代生产力工具。

  - 减少不必要的工作量

    可视化拖拽与参数配置的极简开发模式，结合模型驱动的代码自动生成机制，可以消灭绝大部分繁琐和重复的boilerplate代码；一站式的部署和运维管理平台，无需自己搭建CI/CD流水线、申请环境资源、配置监控报警；一次搭建同时生成、构建和发布多端应用，免去人工同步维护多个功能重复的端应用；开箱即用的组件库、模板库、主题库、连接器等，让最大化软件复用成为可能。总而言之，低代码能够让专业开发者更专注于创新性、有价值、有区分度的工作，而不是把宝贵开发时间都耗费在上面那些不必要的非业务核心工作上。

  - 强大的平台能力支撑

    虽然上面列的技术支撑性工作并不直接产生业务价值，但却会直接影响业务的性能、成本、稳定性、安全性、可持续发展能力等。因此在简化和屏蔽底层技术细节的同时，也会尽可能把自己所cover的部分做到最好（至少能和纯代码开发方式一样好），包括但不限于：

- 现代化的技术架构和实现

  现代化的低代码开发平台，在支撑用户应用时所选择的技术架构与实现方案，也会是现代化且符合业界最佳实践的，例如，前端基于主流的HTML5/CSS3标准和React框架，后端基于成熟的Java语言、SpringBoot框架和MySQL数据库，部署环境基于云原生的Docker镜像、CI/CD流水线、K8s集群和Service Mesh技术。

- 零成本的技术升级和维护

  低代码的高维抽象开发方式，让应用的核心业务逻辑与底层技术细节彻底解耦。开发者在大部分情况下都不需要关心底层技术选型，同时也无需亲自跟进这些技术的版本升级与漏洞修复，免费享受与时俱进的技术红利和应用安全性提升。即便遇到某些底层技术或工具需要进行彻底更换（比如不再维护的开源项目），开发者也完全不必感知；技术迁移再费劲再难搞，平台自己努力就行，对开发者来说只要服务一直在线，岁月就依然静好；事后可能还会惊喜地发现，应用访问突然就变得更快了

- 一体化生态能力复用

  复用（Reuse）是提升软件开发效率和工程质量的最有效途径。传统的代码开发模式下，开发者可以通过提取公共类/函数、引用共享库、调用外部API服务、沉淀代码片段和模板等方式实现复用。在低代码的世界里，平台也可以提供对应的多层次多粒度复用手段，比如页面组件库、逻辑函数库、应用模板库等。

  但更重要的是，低代码平台还可以充分发挥其一体化的生态优势，提供强大易用的可复用能力（资产）的发现、集成与共享体系：以页面组件为例，你可以直接用系统组件，也可以在平台自带的组件市场上搜索和引用更合适的组件，还可以自己用代码开发一个自定义组件并发布到市场中。平台的生态体系越大，积累的可复用能力就越多，应用的开发成本也会越低。

  相比而言，虽然传统代码世界整体生态更庞大和深厚，但由于各类技术不互通、缺乏统一平台与市场、代码集成成本高等原因，一直以来都没有形成有类似规模潜力的生态能力复用体系，导致重复造轮子和低水平重复建设的现象司空见惯，还美名为“新基建”。

低代码虽然是一场应用开发生产力革命，但并不会革掉优秀程序员的饭碗。它去掉的只是难懂的编程语法、繁琐的技术细节和一切可自动化的重复性工作，并没有也无法去掉应用开发最核心的东西：严谨的业务逻辑、巧妙的算法设计、良好的工程风格等。对于优秀的程序员，即使剥去他一层又一层的编程语言和工具熟练度技能外壳，最终剩下的仍然是一个有价值的硬核开发者。

当然，如果你坚持要用纯粹的写代码方式来改变世界，也不至于失业（不一定）。要么，你可以选择那些低代码暂时不太适用的领域，比如底层系统驱动、3D游戏引擎、火箭发射程序；或者，你也可以选择去写低代码中那一部分不可或缺的自定义代码扩展，为平民开发者提供高质量的积木。

- 人工智能

  我们需要知道的。

  人工智能并不是最近才流行起来的，我们或许很早就听说过人工智能大战围棋冠军的故事，我们或许都有这样的体验，当我们下载了一款象棋游戏并进入人机对战模式，可能初级的人机对战我们也无法获胜。除此之外，身份识别，自动驾驶，战争机器人，机器女友等已经是耳熟能详的名词了。

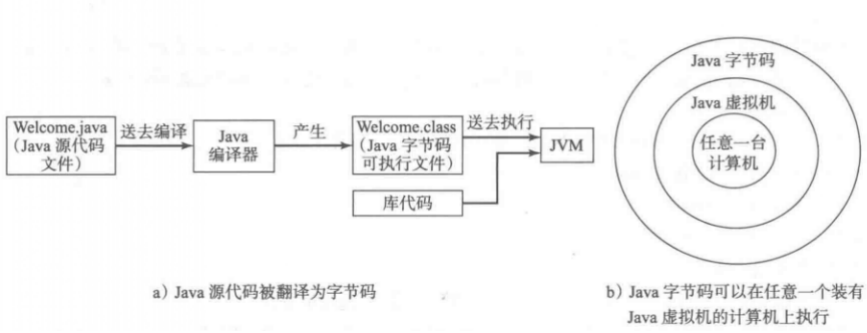
人工智能已经开始对浏览器进行改革了，传统的浏览器比如Google，我们要访问网站的时候，要么在浏览器的URL栏中输入网站的相关域名，要么就在搜索栏中输入一些关键字让浏览器进行爬取，而人工智能比如前段时间基于OpenAI发布的Chat GPT，当我们输入需求的时候，比如输入用python写一个愤怒的小鸟游戏，ChatGPT就会以对话的方式，以文字的形式在显示器中边讲解，边用python代码甚至于注释写出愤怒的小鸟的游戏，把代码复制并按照它的讲解进行部署之后就能真正运行这个游戏，除此之外，它还能根据需求写出需求文档，根据问题给出想要的答案，相比于传统浏览器来说它更简洁和智能，甚至于请了个专家进行解答。相信在不久的将来它将改革浏览器的访问方式。

人工智能辅助开发者，Github和OpenAI已经联合开发了一款插件Complit，这款插件可以直接输入对应的需求在开发中生成代码，虽然它目前还是针对于一些比较固定的业务，比如生成注册网页，使用Java实现登录逻辑等。低代码还是需要人工手动拖拽实现，而人工智能的开发方式将是通过人工智能模型生成一些代码甚至是部署到项目中，在未来，随着人工智能的大力发展，人工智能有望实现输入需要的业务即可生成并部署好相应的软件。曾经有人用人工智能和人类在算法大赛上进行对比实验，人工智能能够碾压冠军。

当然目前人工智能的技术还是需要时间的证明才能投入到市场当中，并且人工智能还可能与人伦等社会学问题有关，需要研究解除矛盾。

物竞天择，适者生存，没有永恒不变的时空，过去很多行业的大量人力需求出现的高成本，低效率也已经被现在的低成本高效率的设备所代替，没有永远的敌人，也没有永远的朋友，只有永远的利益，在利益经济体的驱动下，时代的变革下，人无完人，作为开发者我们不应该局限于开发更应该学习更多的管理和新型技术领域的知识以及巩固自身的基础，有不断学习，不断经历的过程。这个过程是不局限于书本而是任何经验和知识的载体（网络，经历等）能够达到我们的目标即可，也就是不管黑猫，白猫抓到老鼠的就是好猫。

- 编译和执行Java程序
  - Java的编译和执行过程



Java的编译过程可以总结为两个时期：编译期和运行期。

编译期：将.java的源文件通过javac命令编译生成.class字节码文件

运行期：通过java命令启动JVM并把.class文件加载到JVM中进行运行

接下来使用文本编辑器来开发第一个Java程序。

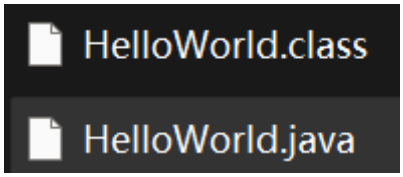
首先在一个文件夹中新建一个文本文档，命名为HelloWorld并将后缀名改为.java。

右键，选择打开方式为使用记事本打开。在打开的记事本中输入以下代码。

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

ctrl+s保存文档。在下面文件夹的地址路径栏上输入cmd，进入Windows的DOS界面中。

首先我们需要编译.java的文件生成编译后的.class字节码文件，使用命令javac HelloWorld.java。如果程序中没有错误，那么编译通过后是如下的样子。



回到文件夹可以看到在文件夹中出现了字节码文件。

```
D:\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>javac HelloWorld.java

D:\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>
```

该字节码文件可以放在任何具有JVM的平台下进行运行，这体现了Java的跨平台特性（体系结构中立），字节码文件是类似于机器指令一样的二进制文件，JVM和真正的计算机的功能以及结构是差不多的，所以它在一定程度上模拟了计算机，我们知道计算机只能读懂二进制的文件。然后我们需要运行该字节码文件所以需要启动JVM并加载该字节码文件，使用 命令java HelloWorld就可以看到输出的”HelloWorld“。

```
D:\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>javac HelloWorld.java

D:\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>java HelloWorld
HelloWorld
```

- 那么Java属于编译型语言还是解释型语言呢？属于静态语言还是动态语言？属于强类型语言还是弱类型语言？

  Java属于半编译半解释型语言，本质还是解释型语言Java的编译过程中是将Java源代码编译为Java字节码，然后由JVM通过加载运行字节码解释给操作系统，操作系统调用主机资源进行运行，所以在这个过程中它即用到了编译器，也用到了解释器。除了Java之外C#也是半编译半解释型语言。

  它的源代码经过一次编译就一次性的转换为了字节码文件这是它的编译型语言的特性。它的字节码文件通过JVM的加载和运行解释给操作系统进行资源的调度，这是它的解释型语言特性。

  它的本质还是解释型的语言，因为它最终是通过JVM解释字节码文件给操作系统之后调度资源的。实际上它要比纯粹的解释型语言的运行速度要快都快赶上编译型语言的运行速度了。最主要的就是它的字节码文件是二进制的，而JVM的加载解释二进制文件给操作新系统，就相当于使用了一种机器语言编写了代码，通过JVM的映射使得操作系统更高效的执行。纯粹的解释型语言，比如python，它的源代码是一句一句被解释给操作系统的，也就是直接解释源代码，解释一句运行一句，如果要再运行的话还是需要重新解释一句运行一句。

- 编译和执行Java程序时可能会出现的问题
  - 字符编码

    什么是字符编码?

    我们可以看到现在我们所使用的计算机大多数时候并没有和数值计算这类功能打过交道，相反我们常常和一些非数值计算打交道比如文字，表格，图片，视频，音频等，其中最主要接触的就是文字。

    文字作为人机交互中最常见的一种方式，在计算机领域中的地位是非常重要的，可以说没有文字的交互就没有计算机的巨大发展。我们可以看到，在计算机中的文字是一个个的字符表示出来的，而这一个个的字符的本质是什么呢?

其实不难想到，计算机只能识别二进制的机器码，它并不知道这一个个的字符，那么这些一个个的字符就是由一串串二进制的机器码所表示出来的。

我们经常在这样的情境下，现在我在打字，在显示器上看到的是一个个的具体的字符，其实计算机并不知道这些字符，而是通过解码的软件比如文本编辑器通过我调用的二进制位串找到对应的字符显示出来的。

这种二进制位串和具体的字符的对应关系就是字符编码，字符编码可以理解为一种规则，这个规则将字符映射到唯一的一串二进制机器码。就像是以前发电报时的密码本，里面的内容是“字符—电码”，发送对应的电码之后就能还原出情报。

- 常见的字符编码
  - **ASCII编码：**

    使用7位2进制数表示一个字符，7位2进制数可以表示出2的7次方个字符，共128个字符。由于计算机是按照字节的方式存储数据的，所以一个字符在计算机中也是按照字节的方式进行存储的。即便ASCII码仅有7位的二进制，但是它依然是要 按照字节也就是8位的方式进行存储，最高位为0。

    ASCII码包括32个不能显示出来的控制符号，我们需要知道的是ASCII码中还包括有阿拉伯字母0-9，小写英文字母a-z和大写英文字母A-Z。上述的符号的ASCII码是连续的，比如0-9中我们只需要知道0的ASCII码为48，就可以推出1的为49。a的为97，A的为65。

    在英语中使用128个字符就可以表示所有的英文符号，但是如果要表示其他的语言128个字符是不够的。这也表明英语是适合于计算机的表达的，因为无论多么复杂的英语单词它的本体都是由26个字母中的一些排列组成的，大小写一共52个。

    如何理解ASCII码的对应关系呢，比如‘A’的ASCII是65，那么在机器中'A'-01000001,我们现在所看到的字符是我所通过键盘输入的，其实现在来看，如果我们用放大镜或者是将文档放得很大的话它是有像素的，那么也就是说我们所看的显示器上展示出来的文字其实就是图片显示出来的。这里只是浅层次的理解一下文字是如何显示出来的，如果要专业一点说的话文字是通过点阵图或者是矢量图显示出来的，这里可以参考的专业书籍是计算机图形学的内容。

    ASCII码作为国际通用的编码，也就是说无论使用任何其他的编码它都要兼容ASCII码，所谓的兼容就是说给定的字符编码和ASCII的字符编码一致，也就是说一个国家在制作自己的编码的时候要先为ASCII码留出它原来的位置，然后扩展自己的其他编码。

  - ISO-8859-1编码：

    这套编码规则由ISO组织制定。是在 ASCII 码基础上又制定了一些标准用来扩展ASCII编码，即 00000000（0）~ 01111111（127）与ASCII的编码一样，10000000（128）~ 11111111（255）这一段进行了新的编码。

    通过最高位的不同充分利用了一字节。现在一字节可以表示256个字符编码。

    但是即便是256个字符也不能够表示出其他的语言，就比如如果要表示我们国家的文字，我们知道汉语是一种象形文字，它的形态不同，而且数量远远的甚至是几百倍的超过了256个字符达到万数之巨。而且还分为简体中文和繁体中文，数量巨大。

  - **GBK编码：**

    GBK编码是用双字节的编码方式。所谓的双字节编码方式也就是说扩展为2个字节对文字符号进行编码，使用16位二进制。

如果使用16位二进制进行编码的话，理论上可以编辑的文字码为0-65535，但是采用双字节编码为了兼容ASCII码和节省存储容量，双字节编码的使用是为了变长，也就是说同一个编码里面的有些字符是用单字节表示而有些字符使用双字节表示的，所以实际上GBK编码所能表示的字符数为23940个字符，能表示21003个汉字。

GBK编码是我国编写的适合于我们大陆使用的编码，它主要包含我们平常使用的简体中文。

- Big5编码：

  Big5编码也是采用双字节编码的，需要知道的是该编码是中文繁体字的编码标准，收录了13060个中文编码。

  该标准的编码一般在台湾地区流行使用，这属于历史遗留问题，并且该标准和GBK是不兼容的。

- **Unicode编码：**

  Unicode编码是国际通用的编码，它使用多字节编码方式进行编码。它使用4个字节表示一个字符，在一定程度上浪费了很大的空间，但是它所包含的字符可以说是迄今为止世界上所有字符的合集，它的本质可以说就是一张完整的字符表。

  它通过码点的方式来唯一的记录一个对应的字符Unicode只是一个符号集。为什么说它就是一张表呢？它只规定的字符所对应的码点，并没有指定如何存储，如何进行存储有不同的编码方案，关于Unicode编码方案主要规定有两条主线：UCS和UTF。UTF主线由Unicode Consortium进行维护管理，UCS主线由ISO/IEC进行维护管理。这和它的历史有关。

  对于Java语言来说，我们可以说Java支持Unicode的前两个字节的编码。

  实际上Unicode 码一开始被设计为 16 位的字符编码。基本数据类型 char 试图通过提供一种能够存放任意字符的简单数据类型来利用这个设计。但是，一个 16 位的编码所能产生的字符只有 65 536 个，它是不足以表示全世界所有字符的。因此，Unicode 标准被扩展为 1112064 个字符。这些字符都远远超过了原来 16 位的限制，它们称为补充字符（supplementary character），Java 也支持这些补充字符。但是我们平常使用最多的还是Unicode的前两个字节的内容。

- **UTF-8编码：**

  UTF-8编码是Unicode编码的具体实现。它是遵从Unicode编码中的UTF主线规则设计的是一种变长编码方式，使用1-4个字节进行编码。UTF-8完全兼容ASCII，对于ASCII中的字符，UTF-8采用的编码值跟ASCII完全一致。

  UTF-8是在互联网上使用最广的一种Unicode的编码规则，因为这种编码有利于节约网络流量（因为变长编码，而非统一长度编码）。

  这里需要结合理解的是，我们说到Unicode实际上是一张表，它没有实际的存储在计算机中应用，它只是规定了如果采用Unicode编码应该实现什么样的方案，UTF-8实现了Unicode的编码方案。

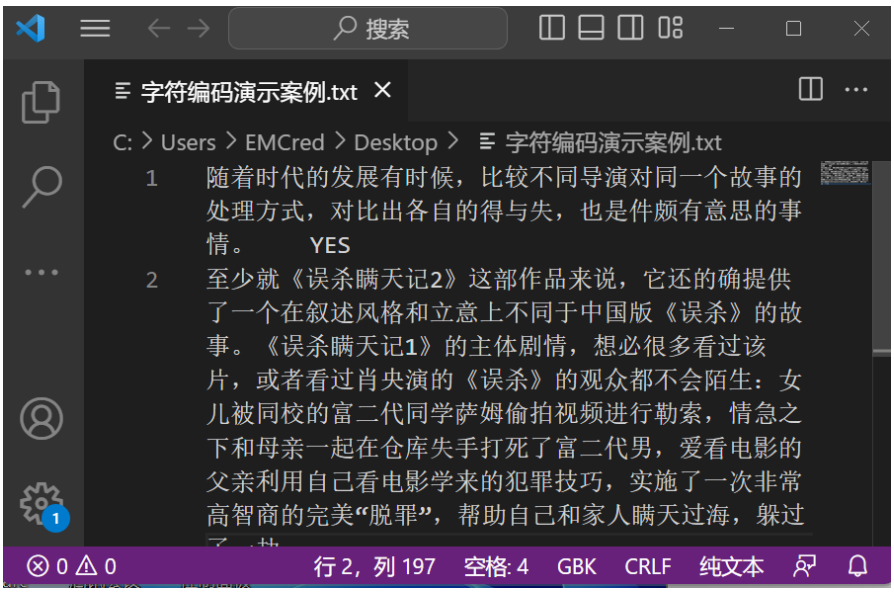
  现在，对于Java来说，它可以支持Unicode的前两个编码也就意味着它能够使用Unicode的具体实现方案UTF-8,UTF-8也是目前编程中常常使用的字符集。

- 解决字符编码问题
  - 乱码问题

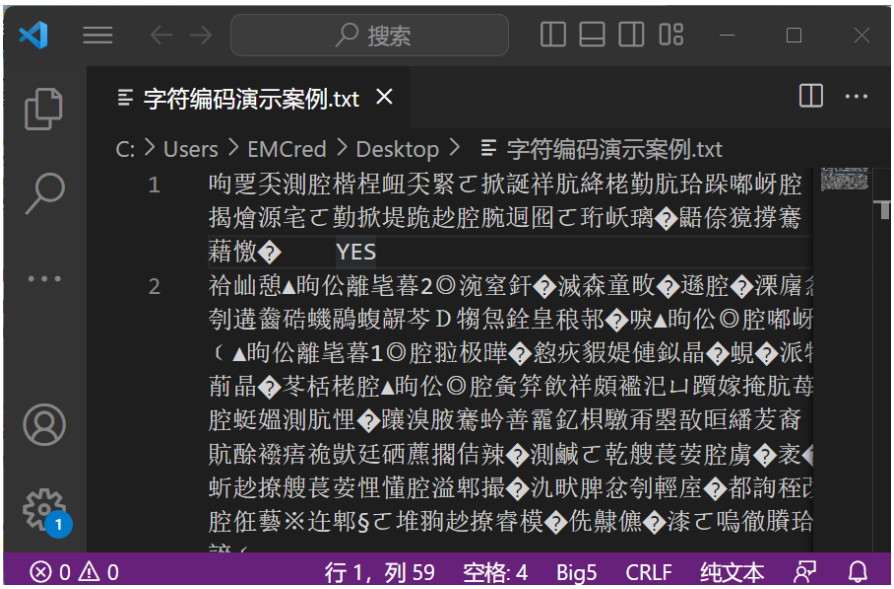
什么是乱码，可以这样理解，随着互联网的发展，我们经常要上网，如果我们要上网搜索文章，那么网页中所展示的主要内容就是文字，在很早的时期，我国使用的编码是GBK编码，所以我们在上网的时候，网页所采用的编码默认是GBK编码，而台湾同胞一直使用的是繁体字，所以他们使用的编码是Big5编码进行上网，然后日本使用的是日本的编码，欧洲使用的是欧洲的编码，也就是说我们所采用的编码方式是不同的。假如我想搜索的一篇文章是来自于台湾的某个网站，那么当网页加载出来的时候，是一片乱码。本质计算机通过网络传输的是二进制的位串，台湾的网站中我所访问的页面内容是使用它们的big5的字符编码所编辑的，所以传输的时候是按照big5的字符编码的二进制串传输的，当传到我的计算机上的时候，我的浏览器默认按照GBK编码进行解析，此时由于“字符编码（二进制位串）— 字符"不一致就会出现本来要传的字是"你好"变成"?@"这样的乱码现象，这时网页中除了英文和数字是正常的，其他的都是一些不能识别的字符。

为了更好的理解乱码的现象，我在这里做个实验。

首先使用编辑器，这里使用VScode编写一段文本。



可以看到我使用的是GBK编码所编写的文本文档。它是正常显示的，当我使用”通过编码重新打开“，选择Big5编码打开之后就会看到。



可以看到出现了许多的乱码，但是YES并没有乱码，这是因为它是ASCII码的基本组成部分。

为了解决乱码，它的最主要的方式就是将全世界的字符录入到一起，这样任何国家都能够通用并且不会出现乱码问题，这就是Unicode组织，它的具体实现UTF-8是当今网页或者是代码编辑时的指定字符集。

- 使用命令编译Java程序时出现的乱码及解决

  我们在文件夹中新建一个.java的文件，输入以下代码。

```
public class HelloWorld_1{
    public static void main(String[] args){
        System.out.println("你好");
    }
}
```

这里我把要在屏幕上输出的内容换成了中文的”你好“，然后我们使用javac HelloWorld_1.java进行编译，然后使用java HelloWorld_1进行运行之后，输出的内容是”浣犲ソ“。

```
D:\BaiduSyncdisk\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>javac HelloWorld_1.java

D:\BaiduSyncdisk\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>java HelloWorld_1
浣犲ソ
```

此时就出现了乱码现象，这是因为调用javac进行编译的时候，javac所使用的字符编码如果不指定的话是默认调用操作系统的默认字符编码进行解析的，对于Windows操作系统来说，由于我们的地区是在中国，所以Windows的默认字符编码是GBK，所以此时它会调用GBK作为解析去解析源代码，我们的源代码是使用UTF-8编写的，其中的代码是由英文的内容组成的，而英文的内容都被定义在了ASCII码中，UTF-8和GBK都认识，所以代码被正确的解析了，但是UTF-8中的字符"你""好"和GBK的所对应的字符编码不同，所以按GBK的解析之后就出现了乱码。

为了正确的解析出来我们想要的内容，使用：

```
javac -encoding UTF-8 HelloWorld_1.java
```

此时重新编译并运行后可以看到：

```
D:\BaiduSyncdisk\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>javac -encoding UTF-8 He
lloWorld_1.java

D:\BaiduSyncdisk\EMCredFile\DevelopRD_Java\1,JavaSE\1,Java-Languag_basic\codes>java HelloWorld_1
你好
```

正确的显示出了”你好“。