

Java开发环境



一、认识Java

- java语言之父：**詹姆斯·高斯林 (James Gosling)**
- 语言最开始只是Sun计算机 (Sun MicroSystems) 公司在1990年12月开始研究的一个内部项目。
- 工作小组：“**Green计划**”，研究下一代智能家电（如微波炉）的程序设计
- 最初，高斯林试图修改和扩展C++的功能，他自己称这种新语言为C++ --，但是后来他放弃了。他将要创造出一种全新的语言，被他命名为“**Oak**”（橡树），以他的**办公室外的橡树**命名。
- 由于商标搜索显示，Oak已被一家**显示卡制造商**注册。于是同年，Oak被改名为**Java**。当使用**十六进制编辑器**打开由Java源代码编译出的**二进制文件 (.class文件)**的话，最前面的32位将显示为CA FE BA BE，即词组“**CAFE BABE**”（**咖啡屋宝贝**）。
- **1995年5月23日，Java语言诞生**
- **1996年1月，第一个JDK-JDK1.0诞生 代号Oak (橡树)。**
- 1996年4月，10个最主要的操作系统供应商申明将在其产品中嵌入JAVA技术
- 1996年9月，约8.3万个网页应用了JAVA技术来制作
- **1997年2月18日，JDK1.1发布**
- 1997年4月2日，JavaOne会议召开，参与者逾一万人，创当时全球同类会议规模之纪录
- 1997年9月，JavaDeveloperConnection社区成员超过十万
- 1998年2月，JDK1.1被下载超过**2,000,000次**
- 1998年12月8日，JAVA2企业平台J2EE发布 **代号为Playground (操场)**
- **1999年6月，SUN公司发布Java的三个版本：标准版 (J2SE) 、企业版 (J2EE) 和微型版 (J2ME)**
- 2000年5月8日，JDK1.3发布 **开发代号为Kestrel (红隼)**
- 2000年5月29日，JDK1.4发布
- 2001年6月5日，NOKIA宣布，到2003年将出售1亿部支持Java的手机
- 2001年9月24日，J2EE1.3发布
- 2002年2月26日，J2SE1.4发布，自此Java的计算能力有了大幅提升 **开发代号为Merlin (隼)**

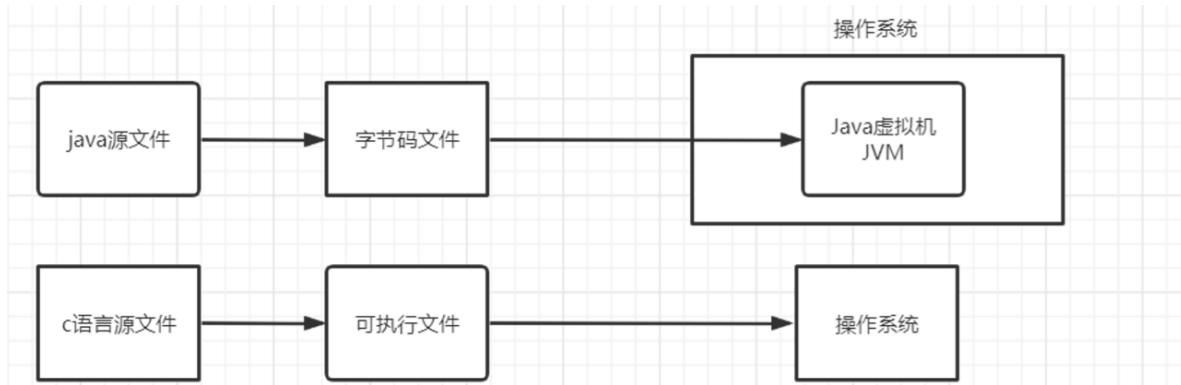
- 2004年9月30日18:00PM，代号为“Tiger”，J2SE1.5发布，成为Java语言发展史上的又一里程碑。为了表示该版本的重要性，J2SE1.5更名为Java SE 5.0(因为新特新增加特别多，所以起了个好听的名字！)，更新了包括泛型支持、基本类型的自动装箱、改进的循环、枚举类型、格式化I/O及可变参数。
- 2005年6月，JavaOne大会召开，SUN公司公开Java SE 6。此时，Java的各种版本已经更名，以取消其中的数字“2”：J2EE更名为Java EE，J2SE更名为Java SE，J2ME更名为Java ME
- 2006年12月，SUN公司发布JRE6.0 **代号为Mustang (野马)**
- 2009年12月，SUN公司发布Java EE 6
- 2010年11月，由于Oracle公司对于Java社区的不友善，因此Apache扬言将退出JCP
- 2011年7月28日，Oracle公司发布Java SE 7 **代号是Dolphin (海豚)**
- **2014年3月18日，Oracle公司发表Java SE 8(市场主流版本) 代号是Spider (蜘蛛)**
- 2017年9月21日，Oracle公司发表Java SE 9。
- 2018年开始，每6个月就会发布一个Java版本，以更快地引入新特性。
- 2018年3月21日，Java 10发布。
- 2018年9月25日，Java 11 LTS发布。
- 2019年2月Java 12发布
- 2019年9月Java 13发布
- 2020年3月17日，Java 14发布。
- 2020年9月15日，Java 15发布。
- 2021年3月16日，Java SE 16发布。
- 2021年9月14日，Java SE 17 LTS发布。

这个是继Java 11之后的一个长期支持版本，并且直接支持到了2029年9月。

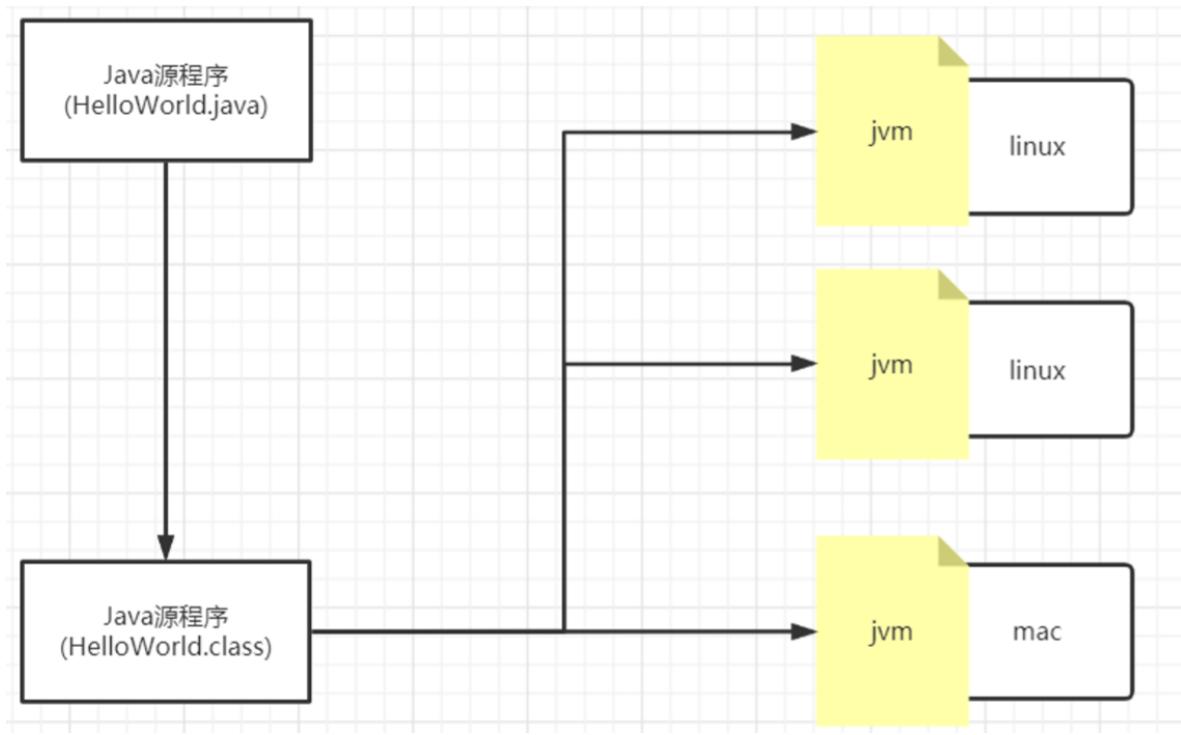
二、Java开发环境

1. Java编译过程

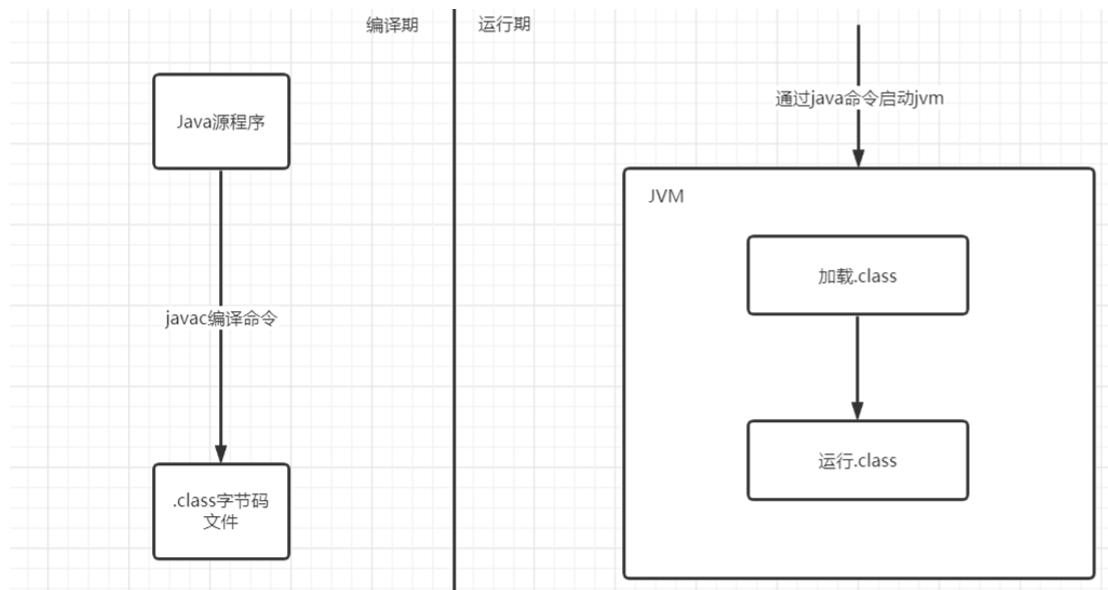
- Java语言编译原理
 - 程序员编写的Java源文件(.java)首先要经过编译，生成字节码文件(.class);
 - Java程序运行需要JVM的支持，JVM是安装在操作系统上的软件，为字节码文件提供运行环境。



- 一次编程到处使用-跨平台性
 - Java官方提供了针对不同平台的JVM软件，遵循着相同的规则，只要是class文件，就可以在不同的JVM上运行。



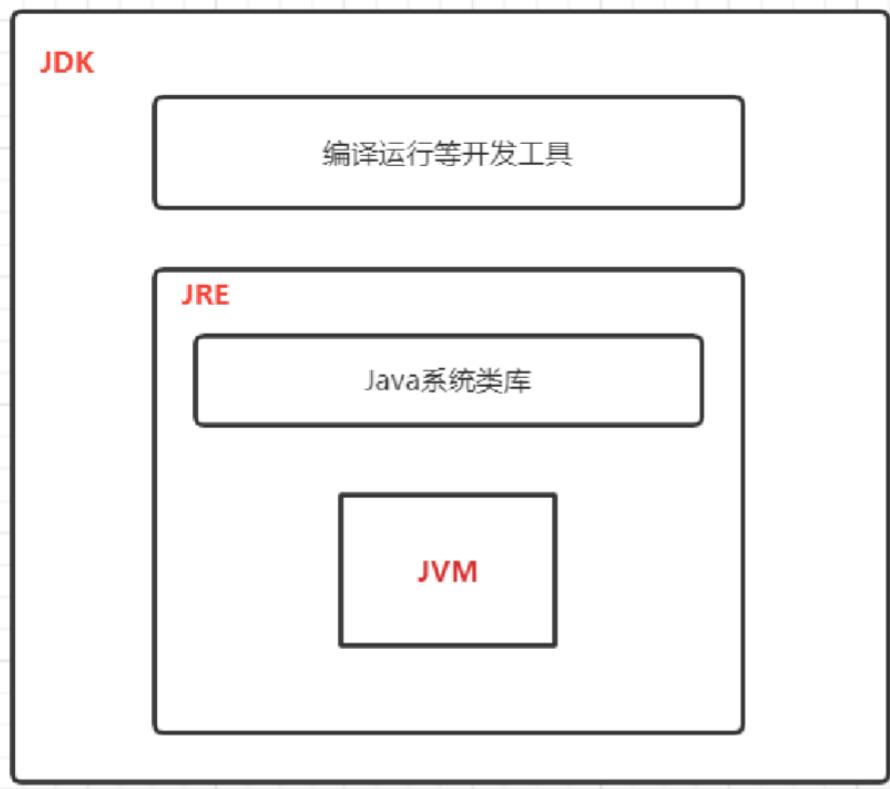
- Java运行时期：编译期 和 运行期



2.JVM、JRE、JDK之间的关系(面试)

- JDK-Java Development Kit(Java开发工具包)
- JRE-Java Runtime Environment(Java运行环境)
- JVM- Java Virtual Machines(Java 虚拟机)

- 1.三者之间是包含关系
- 2.环境只需要安装JDK



3.安装JDK

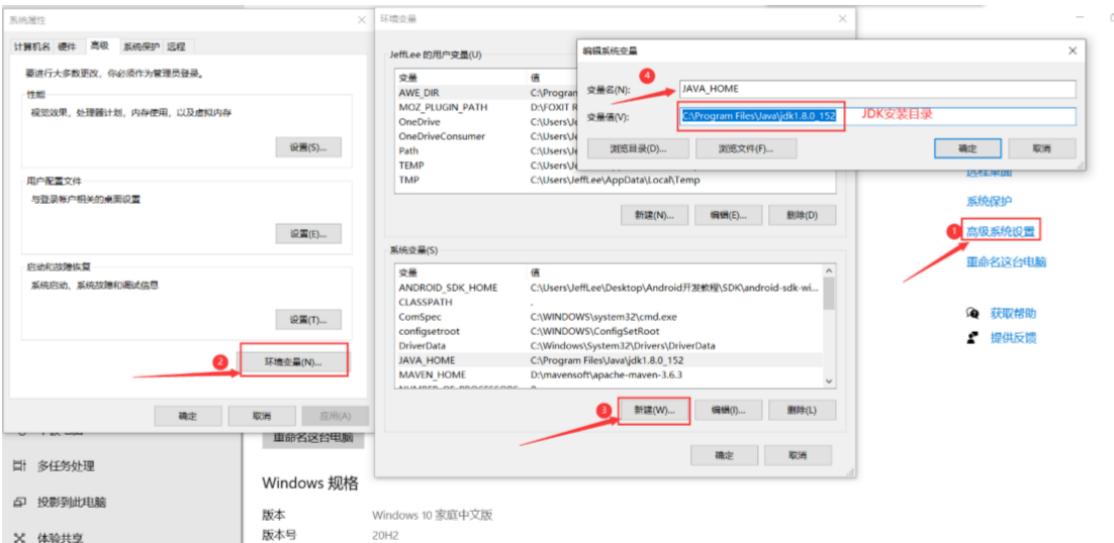
- 官方网址: www.oracle.com
- JDK下载: <https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html>

Solaris x64 (SVR4 软件包)	134.74 MB	jdk-8u311-solaris-x64.tar.Z
Solaris x64 压缩存档	92.76 MB	jdk-8u311-solaris-x64.tar.gz
Windows x86 安装程序	157.37 MB	jdk-8u311-windows-i586.exe
Windows x64 安装程序	170.57 MB	jdk-8u311-windows-x64.exe

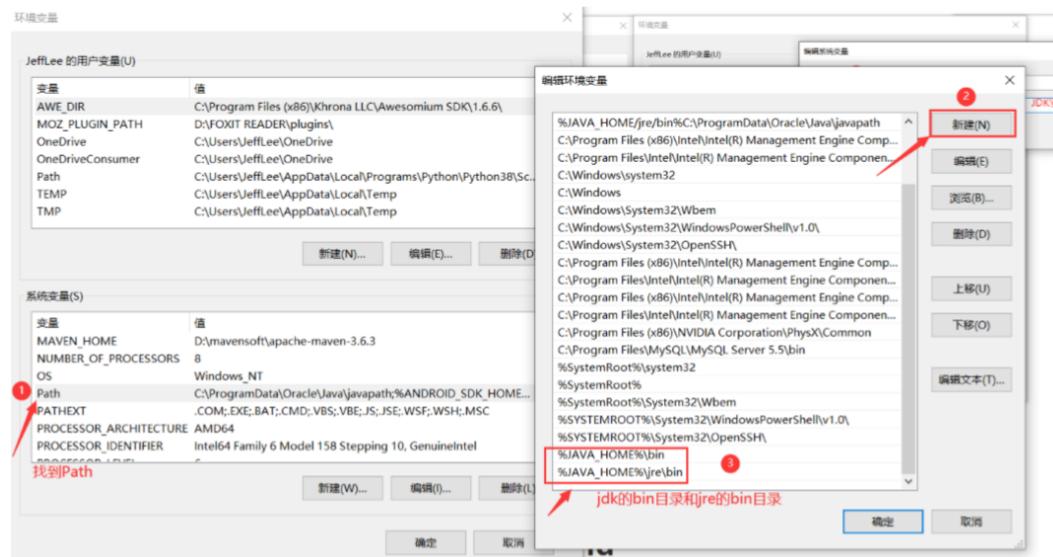
- 安装:傻瓜式安装
- 默认安装位置 : C:\Program Files\Java

4.配置环境变量

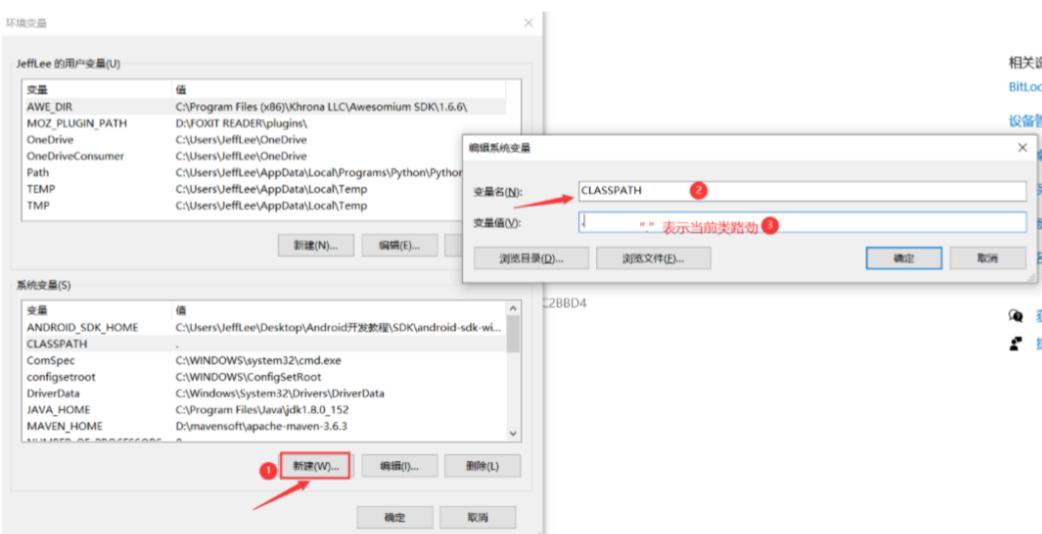
- 目的:让操作系统可以找到命令文件(jdk和jre 的bin目录)所在的目录。
- 环境变量下的系统变量中的Path去配置。
- 配置环境变量的步骤:
 - 我的电脑->右键->属性
 - 配置JAVA_HOME



○ 配置path



○ 配置CLASSPATH



5. 控制台的HelloWorld

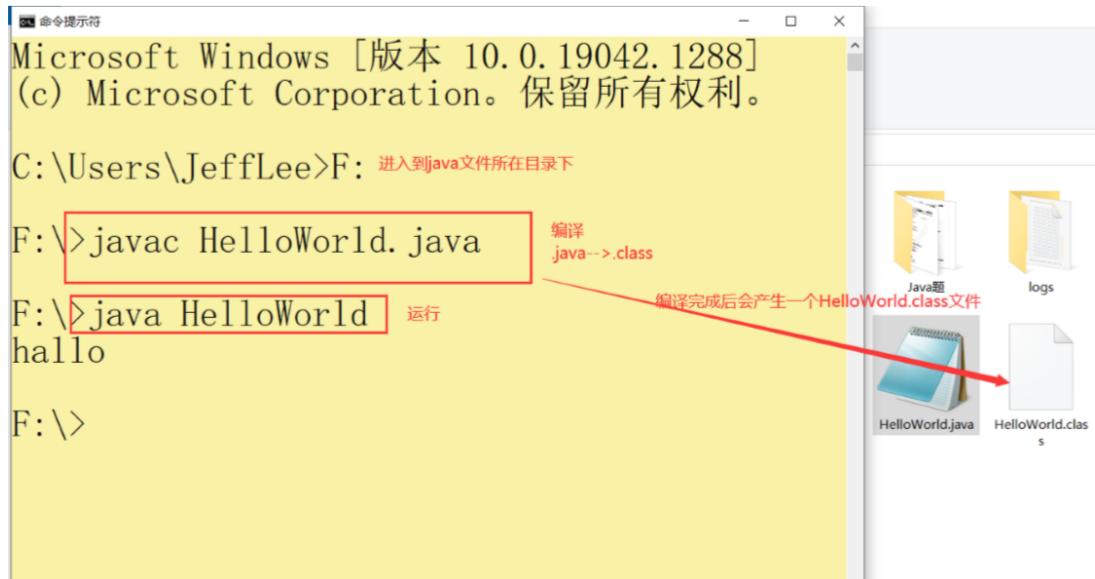
1. 在任意盘符下新建一个HelloWorld.txt

2. 输入代码如下：

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

3. 将HelloWorld.txt改为HelloWorld.java

4. 打开cmd命令提示符编译运行源程序



二、IntelliJ IDEA

1. IDEA简介

- IDE(Integrated Development Environment)集成开发环境，能为程序员开发简化步骤。集成的是编译、调试、提交、重构等。
- 官网:<https://www.jetbrains.com/>
- 下载地址：<https://www.jetbrains.com/idea/download/#section=windows>

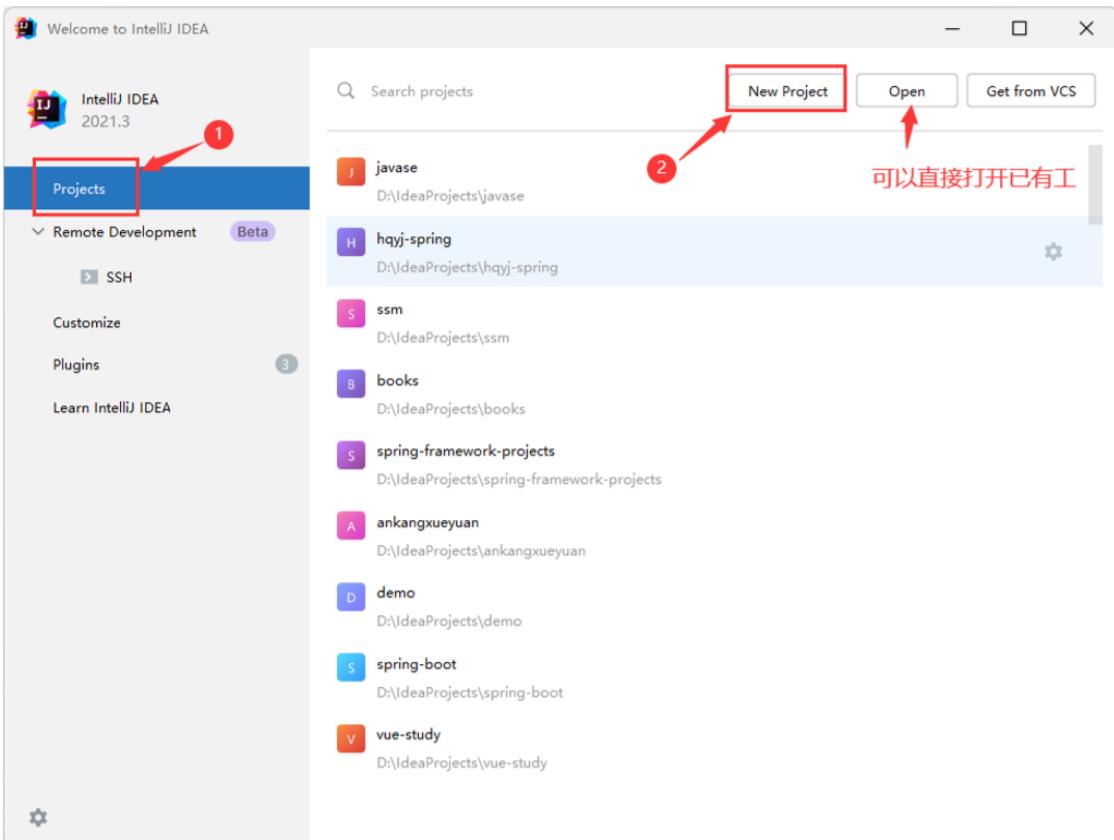
2. 使用IDEA开发一个HelloWorld。

IDEA目录简介

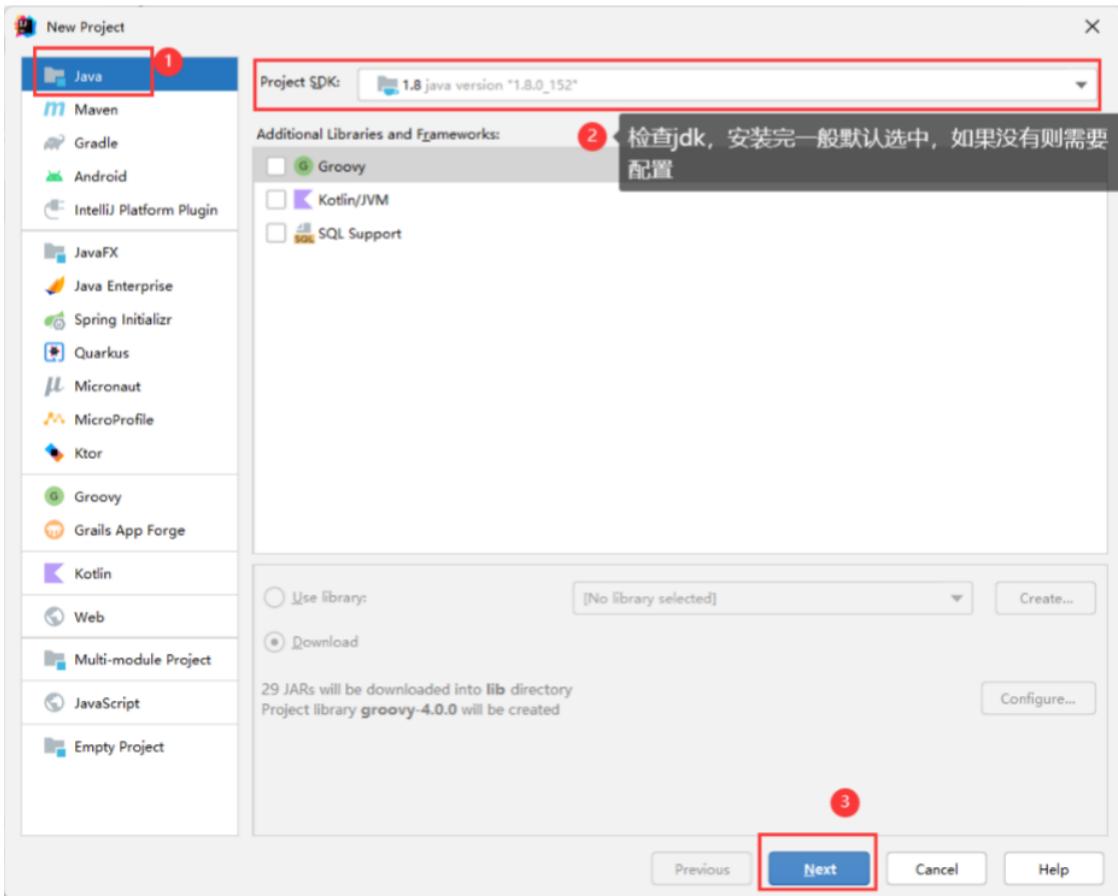
一、工程创建

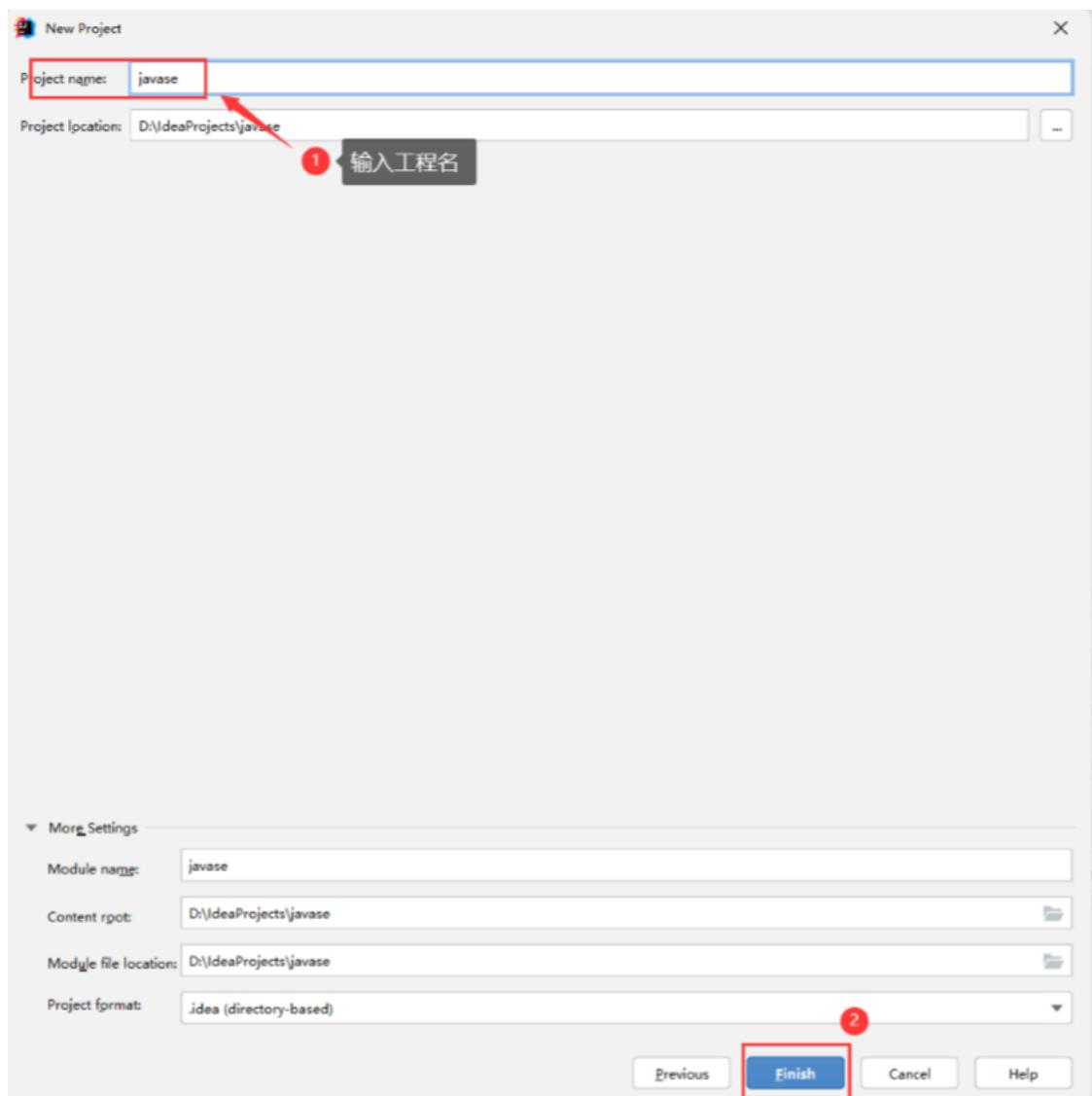
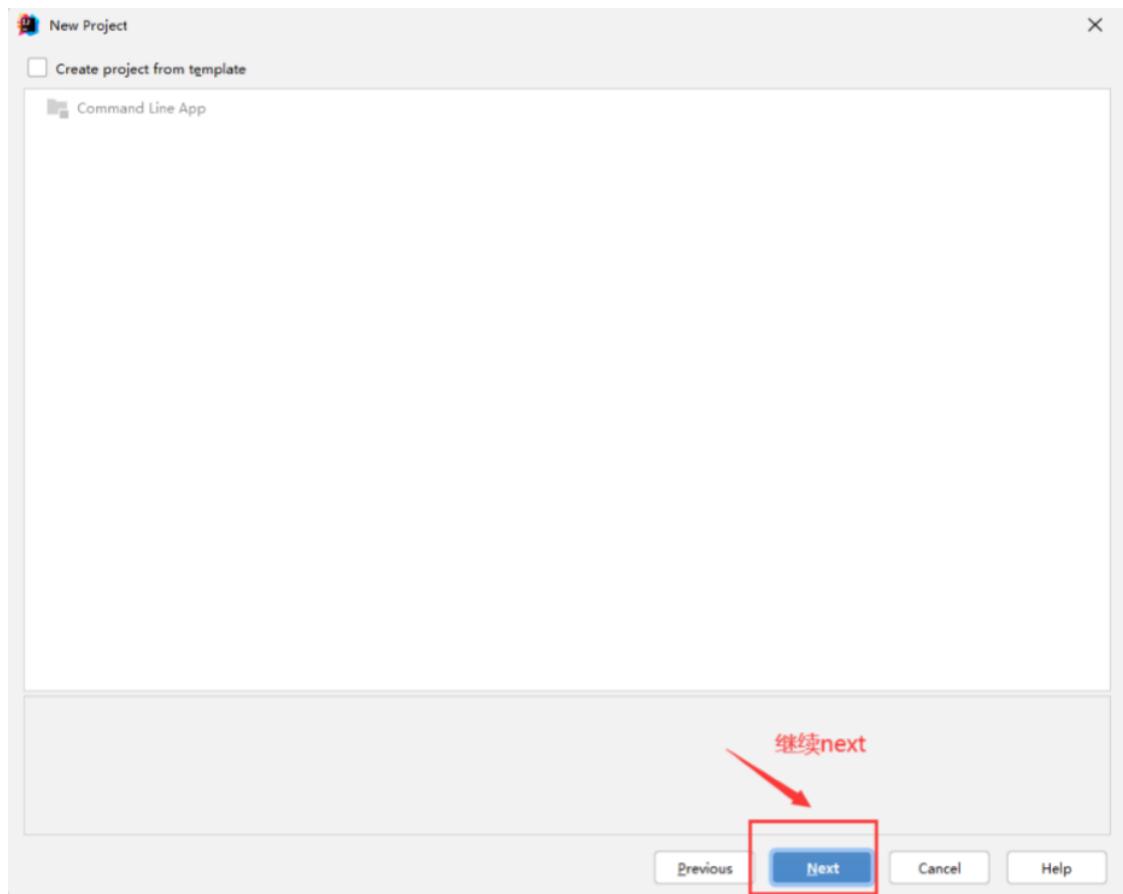
1. 创建工程

- 新建一个Java工程

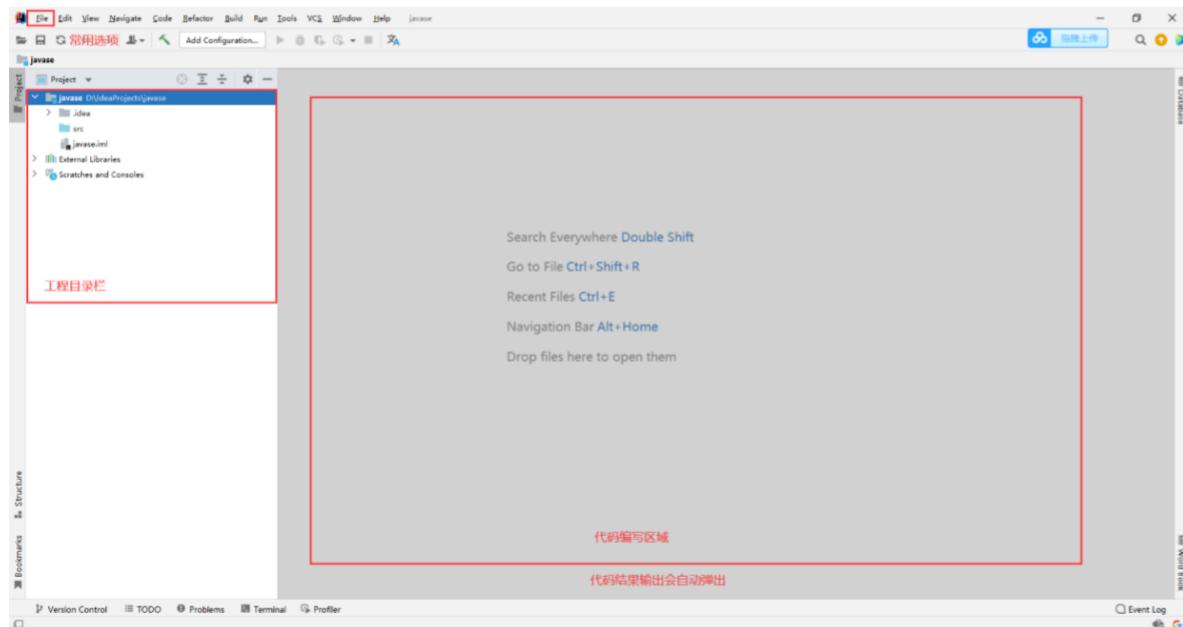


- 选择创建工程类型及jdk



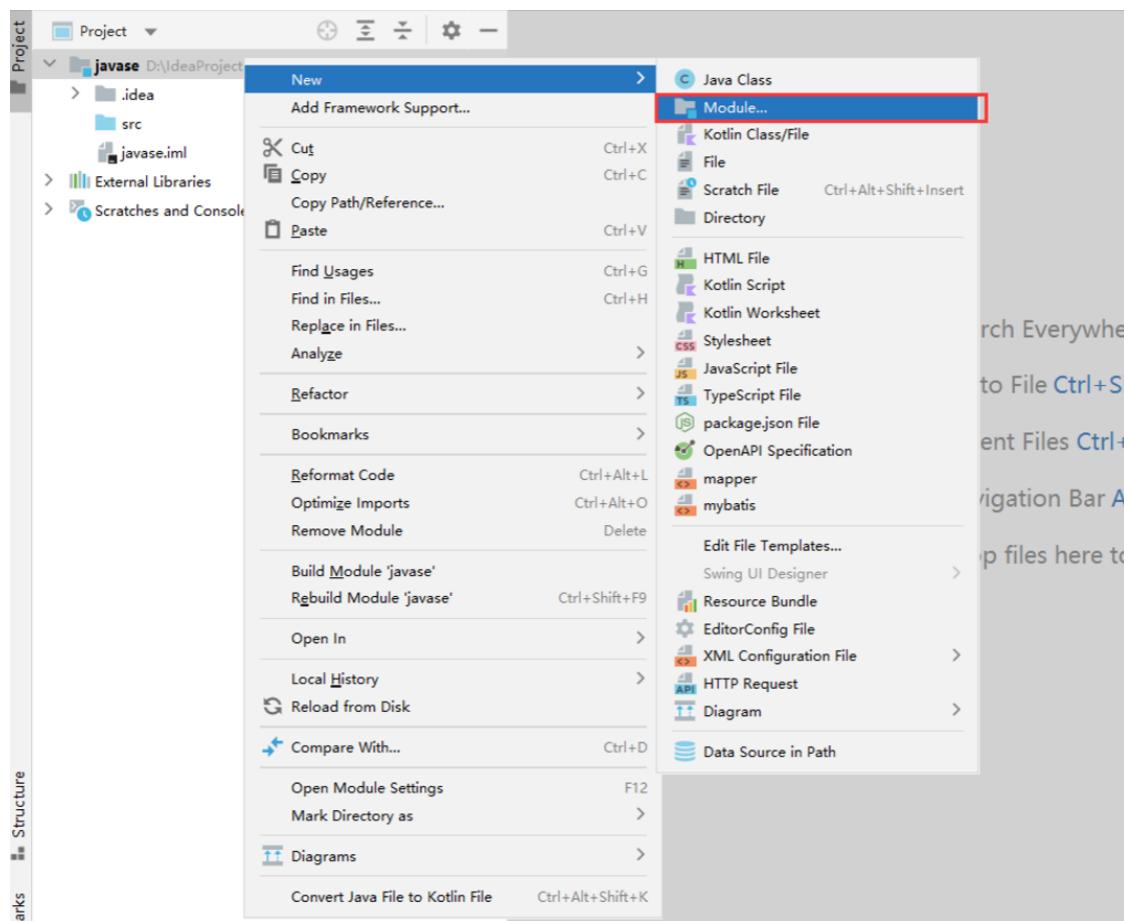


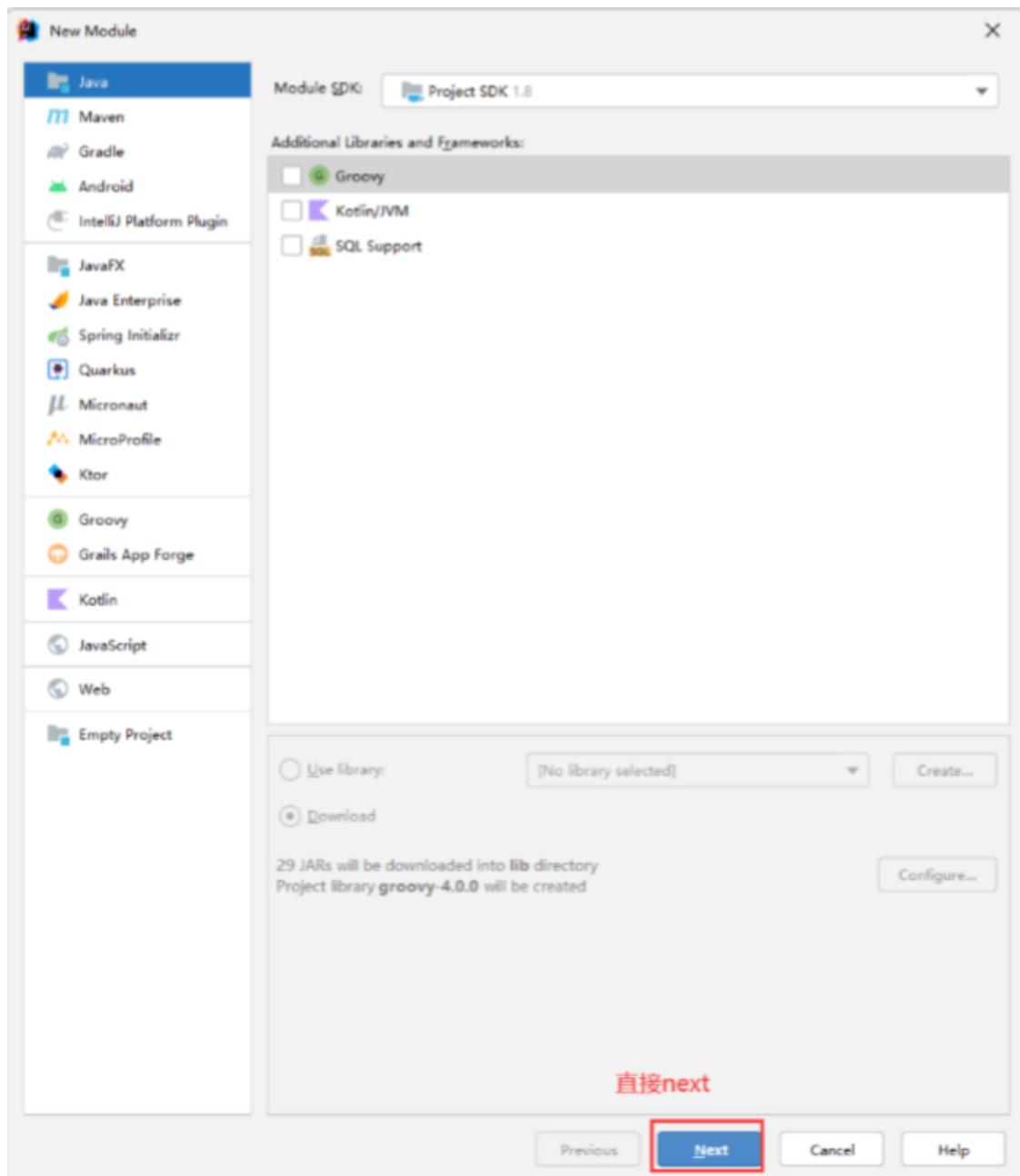
2.认识IDEA界面

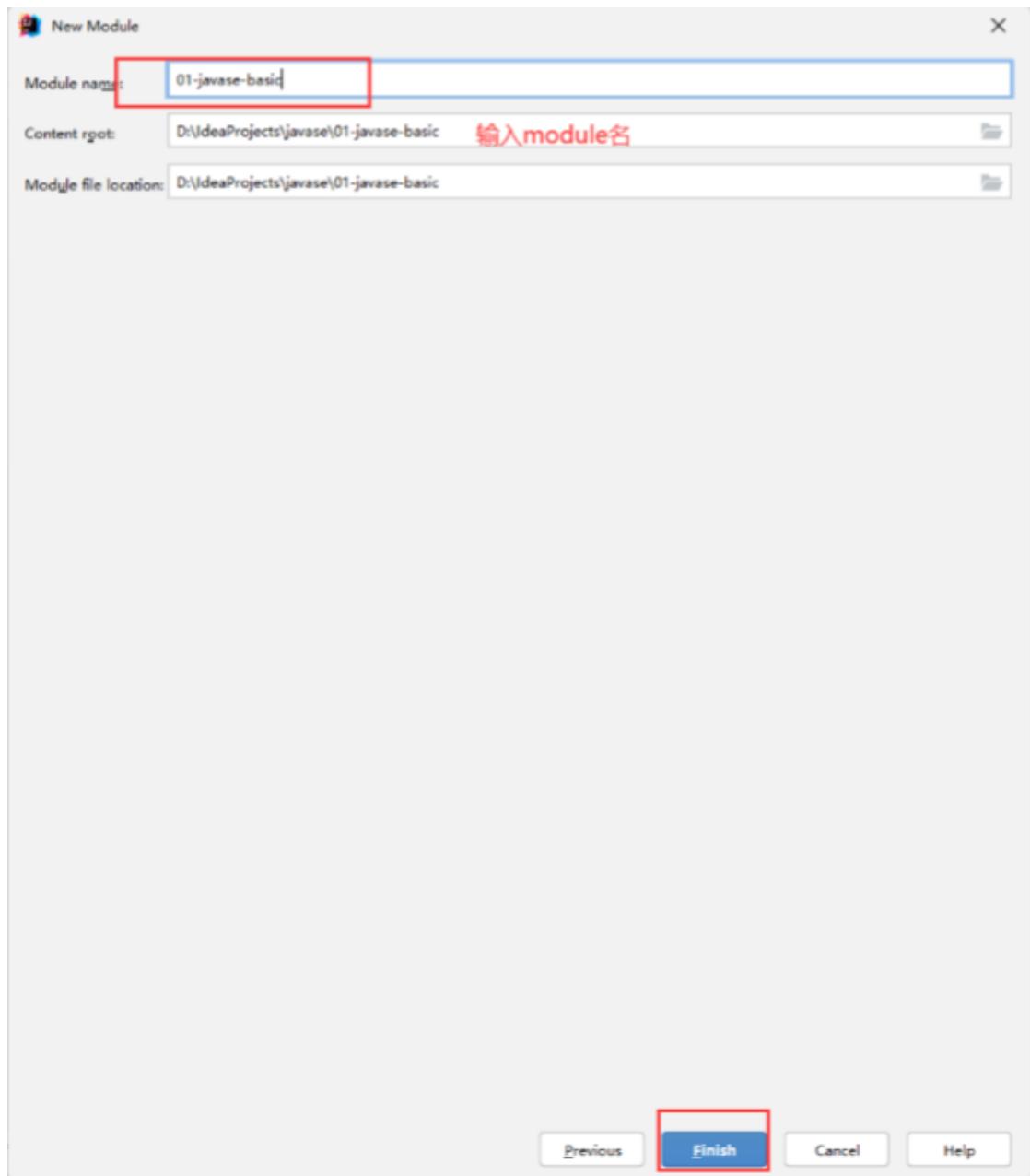


3.创建module

- 选中工程新建module



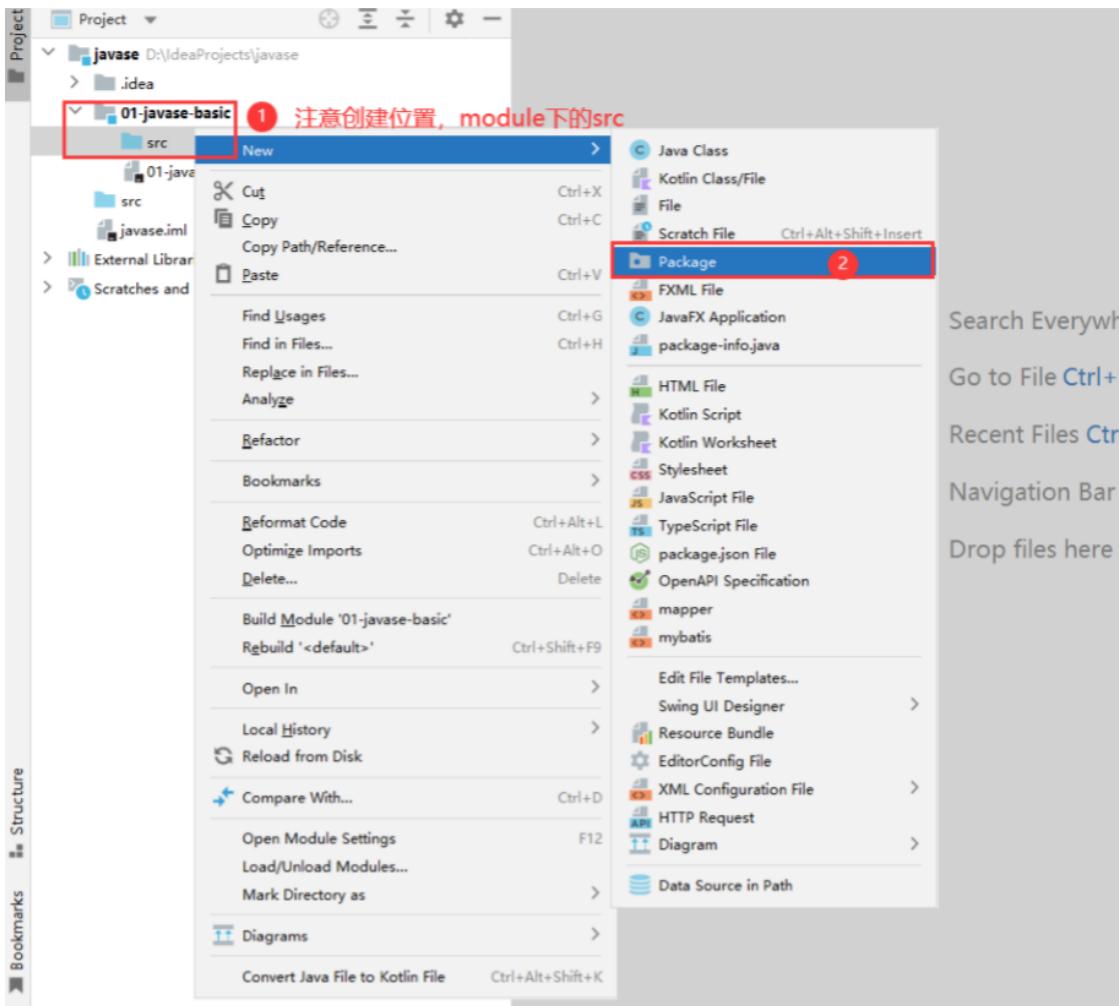




- 工程创建完毕

4. 创建包

- 右键module中的src



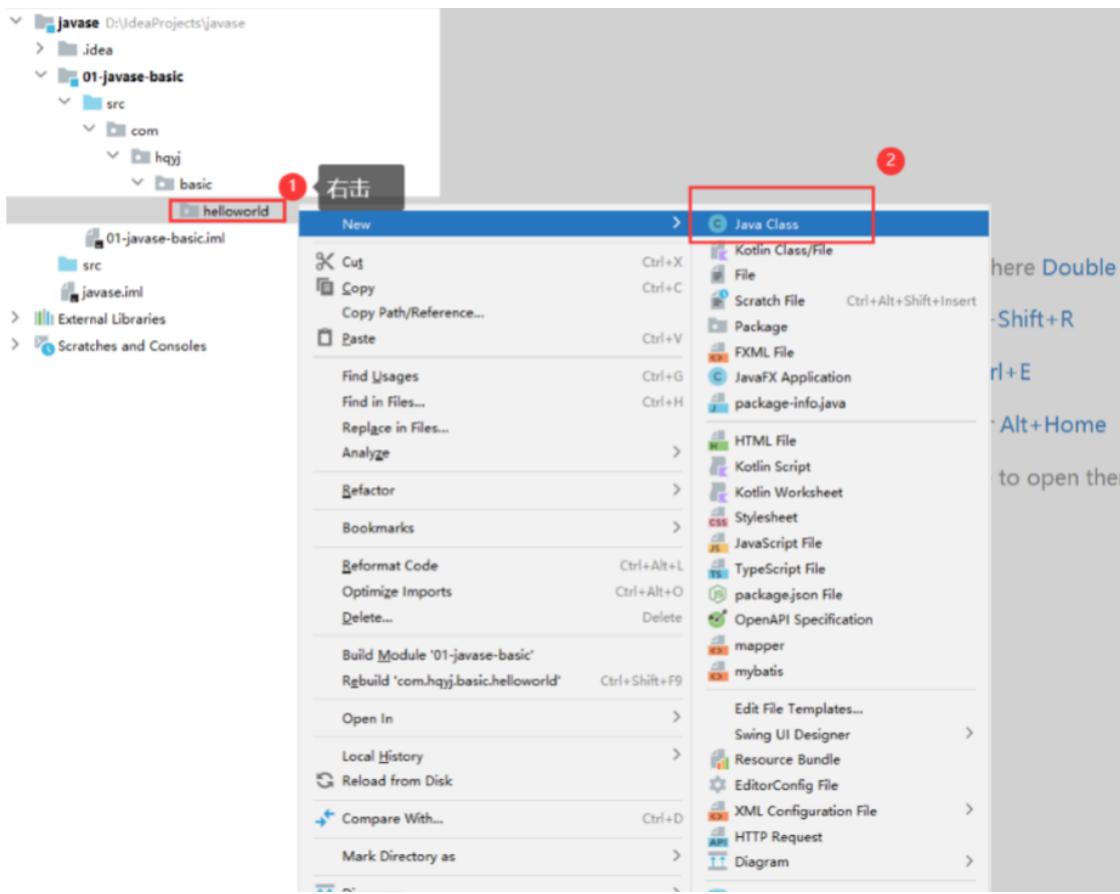
- 包的命名规范-域名反写(全小写)

```
com.hqyj.basic.helloworld
```

- 比如: www.baidu.com 反写为 com.baidu.www
 - com:通用域名格式, com为commercial简称, 表示商业性质的, com结尾的域名为最早通用的顶级域名。
 - .net (一般为网络公司注册此种域名)
 - .org (非盈利组织或协会用此种域名)
 - hqyj:公司名称 华清远见
 - basic:项目名称 比如现阶段学习Java基础
 - helloworld:模块名称
- 包的创建在本地是一个多级文件目录

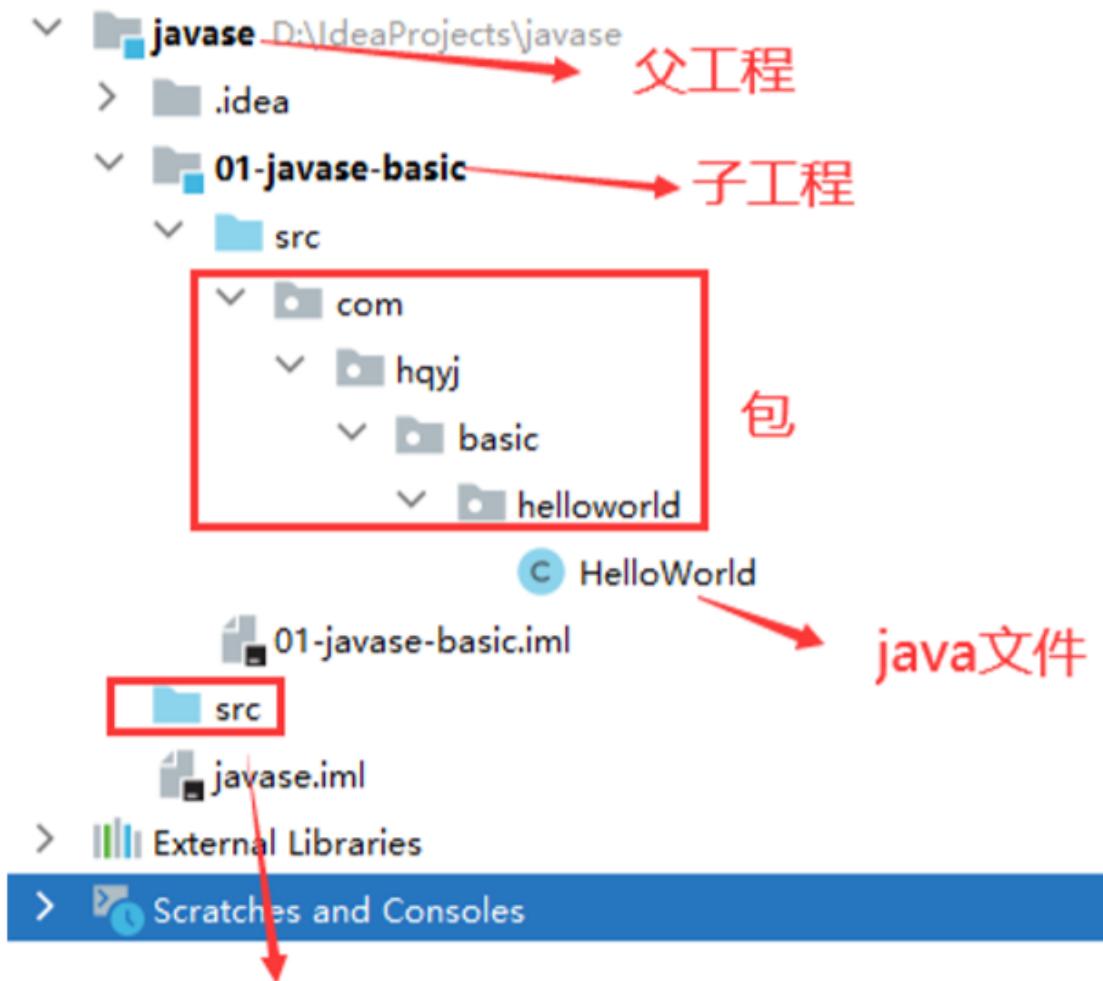
5. 创建类

- 在包下创建Class



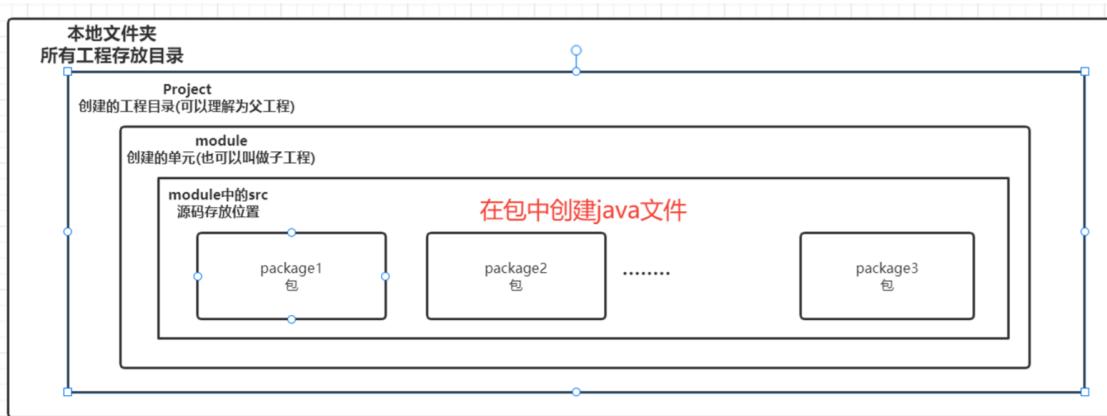
二、工程目录

- 创建完成后工程目录如下：



父工程的src可以删掉不用
现阶段可以这样理解父工程：
就是存放module的一个文件夹！

- 工程目录图【理解】



编程前的认识

一、注释

- 对代码的解释和说明，不参与编译。
- 编写注释是对代码的负责,要求必须为自己代码添加注释，养成良好习惯
 - 梳理编程思路
 - 合作开发，让别人能快速读懂你的代码
- 三种注释方式
 - 单行注释(注释某行代码)

```
//注释内容
```

- 多行注释(注释某段代码)

```
/* 注释内容*/  
或  
/*  
注释内容  
*/
```

- 文档注释(用于生产api文档，结合javadoc命令)

```
/** 注释内容*/  
或  
/**  
注释内容  
*/
```

二、主方法

- 程序执行的入口
- 注意：
 - 牢记：没有主方法的程序不能运行！！！
- 主方法

```
public void static main(String[] args) {  
}
```

Java变量

什么是变量？

在内存中开辟的存储空间，用于存放运算过程中需要用到的数据。

```
int a = 5; int b = 6; int c = a + b;
```

一、变量的声明

格式 :变量类型 变量名

1.未经声明的变量不能使用

- 变量的使用必须声明
- 常见的拼写错误，是错误变量没有被声明。

2.一条语句中声明多个类型的变量

- 中间用逗号隔开

```
int a = 1,b=2;
```

二、变量的命名

1.命名需要符合标识符的语法

- 由字母、数字、_和\$组成
- 首字符不能以数字开头
- 大小写敏感
- 不能使用Java保留字 goto
- 不能使用关键字
- 中文可以命名但不建议

2.命名遵循规则

- 小驼峰命名法: 第一个单词首字母小写, 其他单词首字母大写
- "见名之意": 使用英文单词或英文单词缩写

```
int goSchool;  
String userName;
```

三、变量的初始化

1.在第一次使用变量前初始化

- 确保一定是在第一次使用变量之前初始化的。

```
int sum;  
  
sum= 0; (第一次使用前初始化的)  
  
sum = sum+100;
```

四、变量的访问

1. 可以对变量中的值进行存取、操作

- 变量是存放数据的空间，可以对其赋值、更改和操作；
- 对变量的操作其实是对变量中存储的数值的操作。

```
int a = 100;  
a = a+100;  
//将变量a中的值加上100后所得的结果存储在变量a中。
```

2. 变量操作必须类型匹配(重点)

- 变量在指定声明类型后，Java编译器会检测对该变量的操作是否类型匹配，如果不匹配则编译错误。

```
int salary;  
//salary = 500.0;//小数不能赋值给整数类型  
double d= 123.56;  
//int n = d % 2;//d%2一定为double
```

Java的基本类型

8种基本类型

类型名称	字节空间	使用场景
byte	1字节(8bit位)	存储字节数据
short	2字节(16)	兼容性考虑
int	4字节(32)	存储普通整数
long	8字节(64)	长整数
float	4字节(32)	存储浮点数
double	8字节(64)	双精度浮点数
boolean	1字节(8)	逻辑true/false
char	2字节(16)	存储一个字符

一、 int

- 整数的直接量为int。
- 存储范围: -2147483648~2147483647
- 公式: $-2^{n-1} \sim 2^{n-1} - 1$
 - int的取值范围: $-2^{31} \sim 2^{31} - 1$
 - long的取值范围: $-2^{63} \sim 2^{63} - 1$

- byte的取值范围: $-2^{7\sim}2^{7-1}$ (-128~127)

1. 整数的直接量为int

- 直接量(literal)表示直接写出的整数。

```
int a = 100; //100就是直接写出整数，类型默认为int。
```

- 整数直接量的数字必须在int的范围内。
- 除了进制书写形式外，整数的直接量也经常写为16进制数值，以0X或0x或者8进制的0开头。

```
int c = 0x10;//16进制整数直接量
int d = 020;//8进制整数直接量
System.out.println(c);
System.out.println(d);
```

2. 整型数据的除法运算中的取整

- 除法运算保留整数部分，舍弃小数部分

```
//2. 整数除法运算中的取整
int q = 5/3;//1
System.out.println(q);
int total = 100;
int error = 20;
int percent = error / total * 100;
System.out.println(percent + "%"); //0
percent = 100 * error / total;
System.out.println(percent + "%");//20
```

3. 运算时要防止溢出发生

- 整数运算的溢出:两个整数进行运算时，其结果可能会超出整数的范围而产生溢出。

```
//3. 防止整数运算发生溢出
byte w = (byte)(127 + 1);
System.out.println(w); //-128
```

二、long

- 在表示较大整数的时候，int范围不够的情况下，范围: $-2^{63\sim}2^{63-1}$.
- 表示long类型的直接量，需要以L或者l结尾。

```
long l1 = 2147483647;//对的
long l2 = 2147483648;//编译错误，超出整数直接量的范围，也就是int值得范围。
long l3 = 2147483648L;
```

1. 使用long类型进行较大整数的运算

- 较大整数:超过int范围的整数

```
long distance1 = 1000*365*24*299792458;//使用较大整数运算
/*
* 4个int类型的值相乘结果一定为int，且超出了int的范围,
* 再将值赋值给long类型，则结果时错误的
*/
System.out.println(distance1); //4个int相乘结果一定为int,则超出int范围导致溢出!
long distance2 = 1000*365*24*299792458L;
System.out.println(distance2); //4个int乘long, 结果一定为long
```

2.通过时间毫秒数来存储日期和时间

- JDK提供了一个方法，返回1970年1月1日0点0分0秒到此时此刻所经历的毫秒数，其数据类型为long。

```
long time = System.currentTimeMillis();
System.out.println(time);
```

- 案例-计算程序运行时间

```
long time = System.currentTimeMillis();
//一个程序
try {
    Thread.sleep(5000);
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
long time2 = System.currentTimeMillis();
System.out.println(time2 - time);
```

三、double

- 浮点数，小数，包括float、double
- double类型的精度是float的两倍，因此double叫双精度浮点数。

1.浮点数的直接量为double类型

- 小数的写法有以下几种:
 - 通常写法:3.14、314.0
 - 科学计数法: 1.25E2、1.25e2、1.25E-2
 - eg:1.25E2 1.25*10^2=125.0
- 默认浮点直接量为double类型。如果需要表示float类型，则必须在末尾加F或者f。

```
float f = 1.25F;
```

2.double运算时会出现舍入误差(面试)

- 2进制中无法精确地表示1/10，就和10进制无法精确表示1/3一样。
- 二进制表示10进制可能会有误差。

```
double money = 3.0;
double price = 2.9;
System.out.println(money - price); //输出结果为0.100000000009 舍入误差!
```

四、char

- 本质16位无符号的整数。对应的是字符集的编码。
- 采用的编码是Unicode编码(世界通用的定长字符集16位)。

```
//字符型
char c1 = '中';
char c2 = '\u4e2d';
char c3 = (char)(c1 + 1);
int i = '中' + 1; // '中' 20013
//0000 0000 0001 0000 == 16
int z = '中'; //20013
char c4 = '1';
char c5 = 1;
System.out.println(c1); //中
System.out.println(c2); //中
System.out.println(c3); //不认识的汉字
System.out.println(i); //20014
System.out.println(z); //20013
System.out.println(c4); //1
System.out.println(c5); //1
```

1.对char型变量进行赋值

- 整型变量：0~65535之间的整数数值
- 字符直接量：用单引号括起来的内容就为字符的实际内容 'A' '1'
- Unicode形式：'\u004e'、'\u4e2d', Unicode 的16进制形式

```
//1.char型变量赋值
char c1 = 65;
char c2 = 'A';
char c3 = '\u0041';
System.out.println(c1); //A
System.out.println(c2); //A
System.out.println(c3); //A
```

2.使用转义字符

- 对于不方便输出的字符采用转义

转义	含义
'\n'	回车符
'\r'	换行符
''	单引号
""	双引号
'\'	单斜杠\

五、boolean

- boolean是逻辑运算类型
- 赋值只有 true 或者 false
- 经常用于存储关系运算的结果值

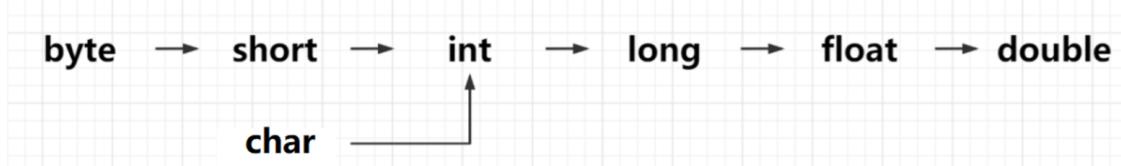
```
//2.逻辑运算
int age = 18;
boolean ischild = age > 16;
System.out.println(ischild); //true
```

六、(重点)基本类型间的转换

1. 基本类型转换

- 自动类型转换(隐式类型转化): 小类型 --> 大类型

可以自动转换的，且大小关系如下图：



- 强制类型转换：大类型 --> 小类型

需要转换符,格式为: (需要转化的类型) 变量

注意:这样转化可能 精度丢失或者 溢出。

2. 强制转化时的精度丢失和溢出

```
//3.精度丢失和溢出
int a = 100;
int b = 200;
long c = a + b;//自动类型转换
System.out.println(c); //300
long l1 = 1024L;
int i = (int)l1;//强制类型转换
System.out.println(i); //1024 没有超出int的范围
long l = 1024L*1024*4*1024;
int j = (int)l;//0
System.out.println(j);
double pi = 3.141592654789456132;
```

```
float f = (float)pi;//3.1415927 造成精度丢失  
System.out.println(f);
```

精度丢失:转化后存储的小数位不是原来的小数位。

3.数值运算时的自动转换

- 多种基本类型参与的表达式运算中，运算结果会自动向较大类型转换。

```
//3.多种基本类型参与运算会向较大类型转换  
long l3 = 123*456*789L;//int*int*long 向long转化  
double d = 500-599.0;//-99.0  
double persent2 = 80/100.0;//0.8
```

4.byte、short、char转化为int

- byte、short、char实际存储数值都为整数
- 在参与运算时，会将byte、short、char都转化为int在进行运算。

```
short x = 12;  
byte y = 3;  
char z = 'A';  
int res = x + y + z;//65+3+12  
System.out.println(res);//80
```

运算符

一、算术运算

1.使用%运算符

- 取余(取模)，意为取余数，可用于整数、char和浮点数。

```
//1.取余  
int n = 255;  
char n2 = 'A';  
double n3 = 16.8;  
System.out.println(n3%8);//0.8000000000000000
```

2.自增和自减运算

- 前置自增自减:先自增自减后运算
- 后置自增自减:先运算后自增自减

```

int a = 10,b=20;
int c1 = a++;
int c2 = ++ b;
System.out.println("a=" + a + ",b= "+b+",c1=" + c1 + ",c2=" + c2);
//11 21 10 21

```

- 结论:
 - 变量无论是前置还是后置运算其结果一定为+1
 - 只在使用它的时候考虑其前置和后置规则

二、关系运算符

- 包或 > < <= >= == !=

```

int max = 10;
int num = 9;
boolean b1 = max > 15;//false
boolean b2 = num%2 == 1;//true

```

三、逻辑运算符

1.逻辑运算 &&逻辑与 || 逻辑或 !逻辑非.

变量b1	变量b2	b1&&b2	b1 b2	!b1
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

结论: 与运算:一假必假 或运算:一真必真

2.使用逻辑运算符

```

//4.逻辑运算符
//与
int score = 80;
boolean res1 = score >= 60 && score <=90;//true
//或
boolean flag = true;
int i = 200;
boolean res2 = flag || (i>=0 && i <10);//true
//非
res2 = !flag || (i>=0 && i <10);//false

```

3.“短路逻辑”

- 与运算: 一假必假 指:当第一个表达式为假时后面的表达式就不会再取运算了
- 或运算: 一真必真 指:当第一表达式为真时后面的表达式不参与运算

```

int i = 100, j = 200;
boolean b1 = (i > j) && (i++ > 100);
System.out.println(b1); // f
System.out.println(i); // 100
boolean b2 = i > 0 || j++ > 200;
System.out.println(b2); // t
System.out.println(j); // 200

```

四、赋值运算

1. 使用“=”进行赋值运算

- 赋值运算符一定是从右边表达式看向左边。

i=1; 表达式将1赋值给i。

```

//赋值运算符
int x,y,c;
x = y = c = 100;
//将100赋值给c --> b = (c = 100)

```

2. 使用扩展赋值表达式

运算符	表达式	计算	结果 (假设X=10)
+=	X += 5	X = X + 5	15
-=	X -= 5	X = X - 5	5
*=	X *= 5	X = X * 5	50
/=	X /= 5	X = X / 5	2
%=	X %= 5	X = X % 5	0

五、字符串连接符

- “+”可以实现字符串的连接。
- 也可以实现字符串与其他类型相连接。

```

//字符串连接符
int in = 100;
String msg = "in = " + in;
System.out.println(msg); //in = 100
msg = "" + 100 + 200;
//String s = "" + 100; String s="100";
//s + 200; // String s = 100200
System.out.println(msg); //100200 拼接
msg = 100 + "" + 200;
System.out.println(msg); //100200 拼接
msg = 100 + 200 + "";
System.out.println(msg); //300 int类型先运算后拼接
msg = 100 + 200 + "" + (300 + 400); //300700 先运算两边再拼接
System.out.println(msg);

```

- 结论:

1. 主要看字符串的位置，字符串与其他类型或字符串类型，那么“+”一定为拼接
2. 如果是同类型先相加，那么“+”为运算符，再去拼接其他字符串。

六、三目运算符(重点-源码里能见到)

1. 使用三目运算符

- 三目运算符(条件运算符) : boolean 表达式? 表达式1:表达式2;
- boolean 表达式为true则运行表达式1，否则运行表达式2。

```
String s = 65 > 60 ? "及格" : "不及格"; //及格
```

2. 三目的嵌套

- 三目运算符可以嵌套使用，表达式1和表达式2的结果可以为嵌套三目的boolean表达式。

```
int a = -3;
String r = a > 0 ? "正数" : (a == 0 ? "0" : "负数"); //
```

- 案例一接受用户输入年份，判断是否为闰年。
 - 年份能被4整除，且不能被100整除的是闰年。
 - 年份能被400整除的是闰年。

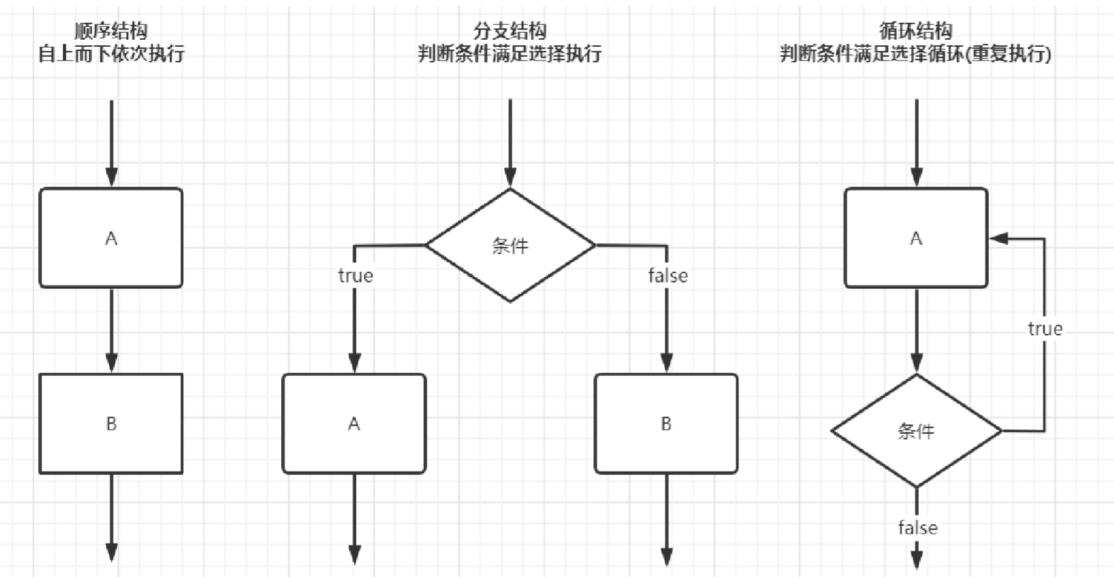
```
//1.接收年份
System.out.print("请输入年份:");
Scanner scan = new Scanner(System.in);
//2.获取年份
int year = scan.nextInt();
//3.判断条件
String result = (year % 4 == 0 && year % 100 != 0) ||
year % 400 == 0 ? "闰年" : "平年";
//4.输出结果
System.out.println(result);
```

分支结构

一、if语句

1. 什么是流程控制

- 任何程序逻辑都是由顺序、分支、和循环组成



- 程序在运行过程中，根据不同条件执行不同的结果
 - 当条件满足时，执行if括号中的结果
 - 条件不满足时则不执行。

2.if执行逻辑

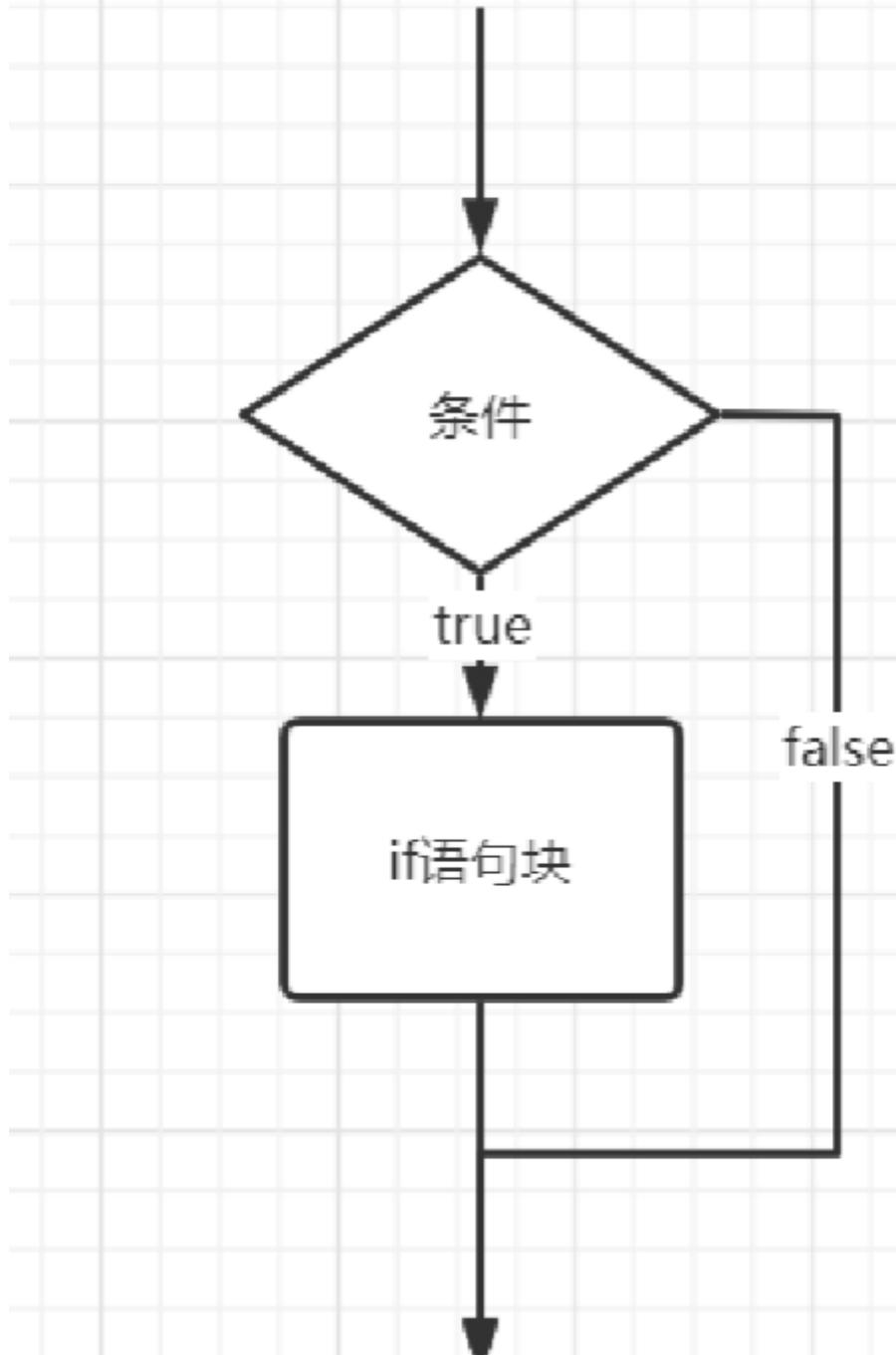
```

if( 100 > 20) {
    语句1;
}
语句2;
//执行语句1和语句2，原因是100>20，则执行语句1，如果不满足则不执行语句1

```

3.if流程图

if流程图



4. 注意：if语句不要省略“{}”

- 当if语句中的{}中只有1行代码时可以省略{}

```
if(100 > 20) {  
    System.out.println("true");  
}  
//<==>  
if(100 > 20) System.out.println("true");
```

- 不建议省略括号，原因时当语句数量过多时，有可能会造成粗心错误，

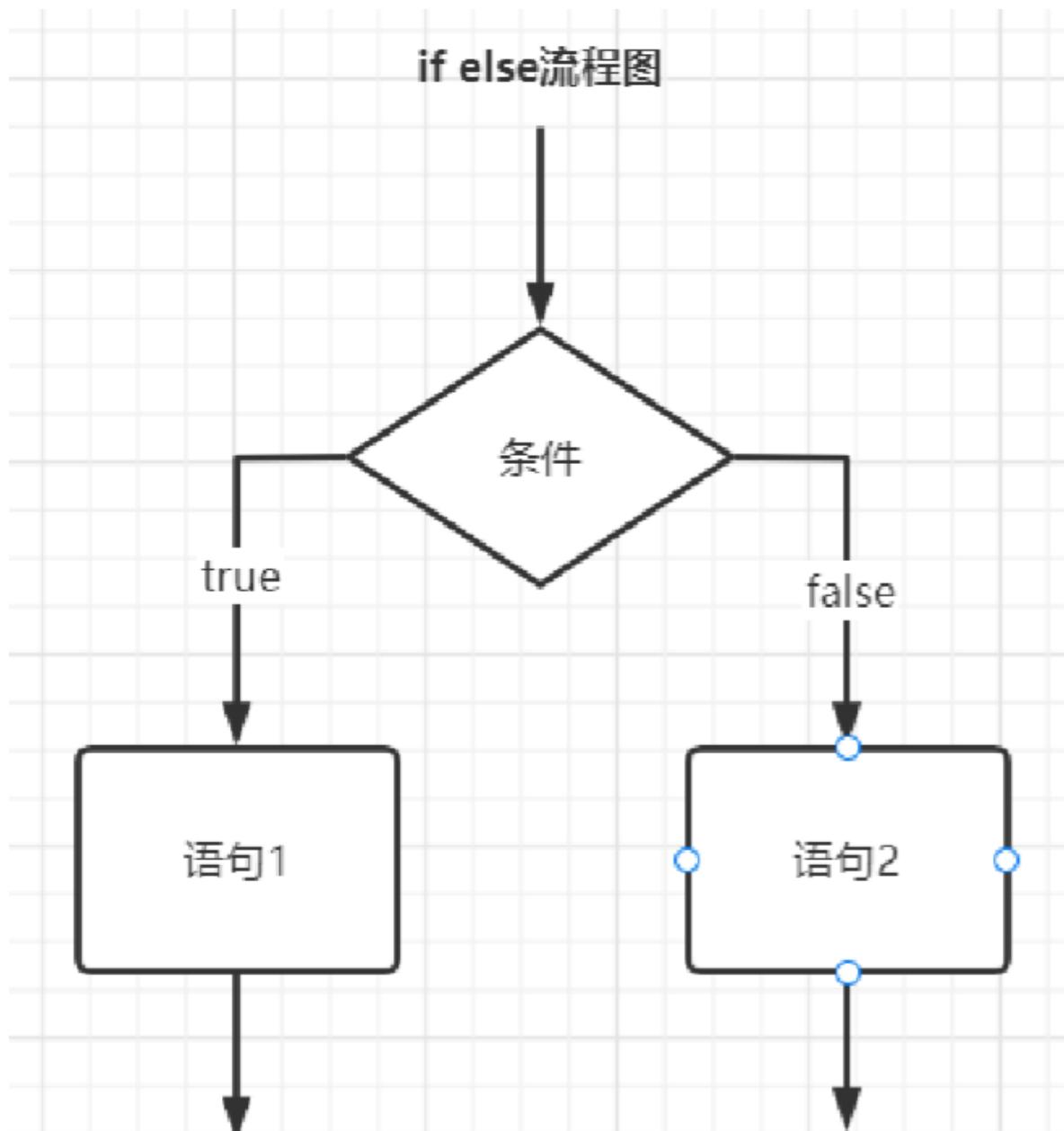
- if后面必须跟{}!

二、if else语句

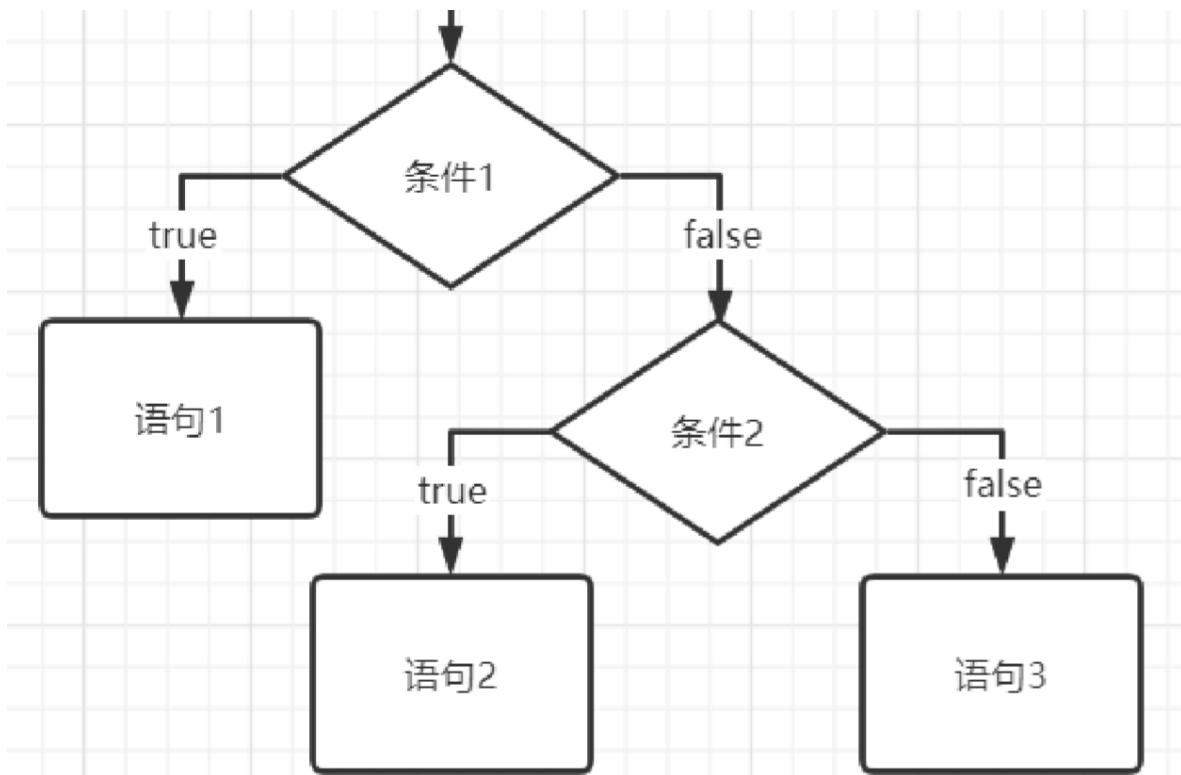
1. if else语句的执行逻辑

```
if(100 > 20) {  
    语句体1;  
} else {  
    语句体2;  
}  
//执行了语句体1. 当条件满足为true时, 执行语句体1, 否则执行语句2
```

2. 流程图



三、else if语句



1. if else 的嵌套

```
//嵌套ifelse
int score = 61;
if(score >= 90) {
    System.out.println("A");
} else {
    if(score < 90 && score >= 80) {
        System.out.println("B");
    } else {
        if(score < 80 && score >= 60) {
            System.out.println("C");
        } else {
            System.out.println("D");
        }
    }
}
```

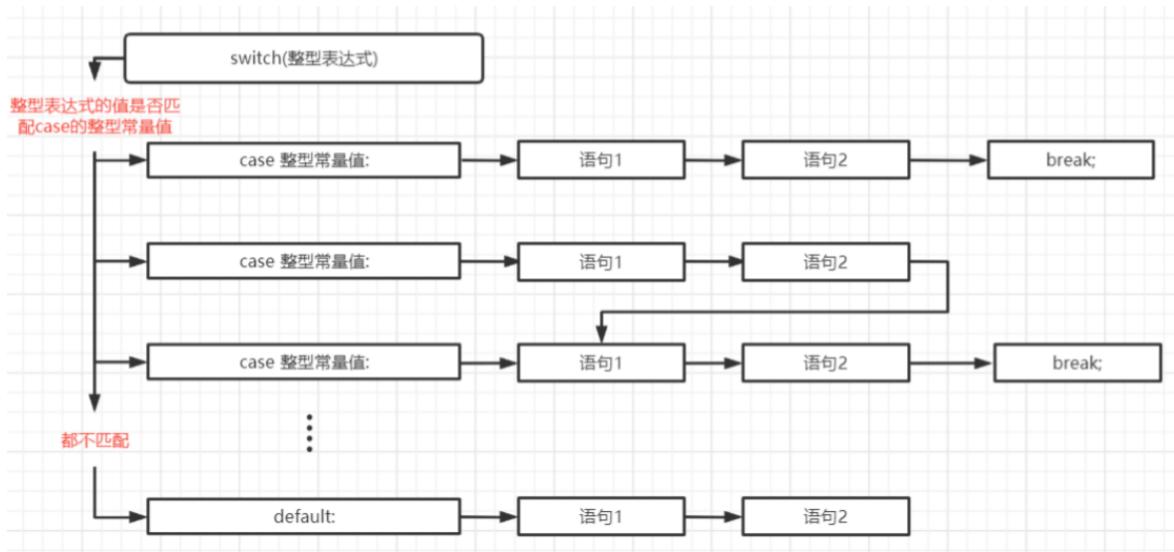
2. else if 的执行逻辑

```
//else if的执行逻辑 -- 简化了if else的嵌套
int score = 61;
if(score >= 90) {
    System.out.println("A");
} else if(score < 90 && score >= 80) {
    System.out.println("B");
} else if(score < 80 && score >= 60) {
    System.out.println("C");
} else {
    System.out.println("D");
}
```

四、switch case语句(重点)

1.switch-case的语句执行逻辑

- 一种特殊的分支，运行通道与switch(整型表达式)中的整型表达式有关。



- 当传入整型常量值时，执行对象case通道。
- 且每一个case通道都必须要和break关键字配合使用，如果不加break，就会执行下一个case，直到遇到break为止或执行完全部的case。
- !!!case后面必须跟整型常量值。
- switch(整数表达式)中整数表达式的类型只能为：
 - byte、int、short
 - char
 - String表达式 (1.7更新后)
- case 整型常量值中整型常量值的取值范围只能为：
 - 必须为整型-不能超出int的取值范围。
 - 必须为常量或常量表达式(1/1、5-4)。

```
switch(x) { //x只能为byte、short、int、char、String表达式
    case 整型常量或常量表达式(1/1、5-4): //整型常量不能超出int的范围
}
```

2.switch-case分支演示

```
int i = 1;
switch(i){
    case 0: System.out.println("0"); break;
    case 1: System.out.println("1"); break;
    case 2: System.out.println("2"); break;
}
//执行结果为1
```

- 与if else结果相比优势：
 - 效率更高
 - 结构更清晰
- JDK从1.7版本后才开始支持字符串表达式的。

循环结构

一、while循环

1.什么是循环

- 一组相同或相似的语句被有规律的重复执行
- 循环要素：
 - 循环条件
 - 循环体

2.while的执行逻辑

```
while(boolean 表达式) { //当boolean表达式的值为true时执行循环体  
    循环体;  
}
```

3.while用于处理循环逻辑

```
//计算0-100的和  
int i = 0; //循环变量  
int sum = 0; //结果  
while(i <= 100) {  
    sum += i; //累加  
    i++; //循环变量自增  
}
```

4.while循环和break配合使用

- break关键字一般用于跳出循环

```
//计算0-50的和  
int i = 0; //循环变量  
int sum = 0; //结果  
while(i <= 100) {  
    sum += i; //累加  
    i++; //循环变量自增  
    if(i == 50) {break;}  
}
```

二、do while循环

1.do while的执行逻辑

```
do{
    循环体
}while(boolean表达式); // 当boolean表达式为true, 会执行循环体
```

2.do while用于处理循环逻辑

```
int pwd;
do{
    System.out.println("请输入密码");
    pwd = scan.nextInt();
}while(pwd != 123456); // 判断输入的密码是否为123456, 直到输入为止
```

3.do while和while的区别

- while是先判断后执行的，而dowhile是先执行后判断
- do while无论条件满足或者不满足都会先执行一次循环体

三、for循环(重)

1.for语句的执行逻辑

```
for(表达式1;表达式2;表达式3) {
    循环体
}
1. 表达式1 只执行第一次, 后续的循环从表达式2开始
2. 表达式2 如果为true 如果为false 则不执行直接跳出
3. 循环体
4. 表达式3
```

2.for用于固定次数的循环

```
//0-100的累加
int sum = 0;
for(int i = 0; i <= 100; i++) {
    sum += i;
}
```

3.for循环的三要素

- 循环变量的定义 --- int i = 0
- 循环条件 -- i <= 100
- 循环变量的变化 i ++
- 这三个要素可以变形

```
//第一种
int i = 0;
for(; i <= 100 ; i++) {
    循环体
}
```

```

//第二种
int i = 0;
for(; i <= 100 ; ) { //!!循环变量二不能变形
    循环体
    i++;
}
//for循环的死循环
for(;;){
    循环体
}
总结: for循环必须要有两个分号";"

```

4. for循环中的break和continue关键字

- for的break搭配使用，一般情况下需要和条件分支结构搭配使用。
- break和continue的区别：
 - break:跳出所有循环
 - continue：跳出本次循环

```

int sum = 0;
for(int i=1; i<=100; i++){
    if( i % 10 == 3) {
        continue;
    }
    sum += i;
} //累加，跳过了个位数位3的所有数字

```

四、循环问题

1. 循环问题考虑的三个部分：

- 循环变量的初始值问题
- 循环的条件(难点)
- 循环变量的改变

2. 循环问题逻辑

- “当”型循环采用while
- “直到”型循环采用do-while
- “固定次数循环”采用for

3. 循环嵌套问题

- 案例：99乘法表

```

for (int i = 1; i <= 9; i++) {//行数
    for (int j = 1; j <= i; j++) {//列数
        System.out.print(i+"*" + j + "=" + (i*j) + " ");
    }
    System.out.println();
}

```

结论: 外层循环控制行数、内层循环控制列数

数组(重点)



一、什么是数组

- 程序 = 算法 + 数据结构
- 数据结构: 把数据按照某种特定的结构保存, 设计一个合理的数据结构是解决问题的关键。
- 数组:最基本的数据结构
- 数组的特定结构: !!!相同类型组成元素集合!!!。
- 通过元素的下标进行访问, 且下标从0开始, 最大元素下标为[数组长度-1]。

二、数组的定义

声明数组

- 数据类型[] 数组名 = new 数据类型[数组长度];

```
int[] i = new int[5];  
含义：  
int：数组的元素类型必须为int  
i：数组的名称  
new：创建了一个数组对象  
5：数组的长度  
1. 初始值为int的默认值  
2. 元素为{0, 0, 0, 0, 0}
```

三、数组的初始化

1. 基本类型数组默认元素值

- 整型：初始值为0
- 浮点型：初始值为0.0
- 字符型：空字符
- 布尔型：初始值 false

2. 三种初始化方式

- `int[] i = new int[5]; //长度为5`
- `int[] i = new int[]{0,1,2,3,4}; //长度即为元素个数`
- `int[] i = {0,1,2,3,4}; //直接初始化元素值。长度为元素的个数`
 - 1. 这种写法只能再声明时对其初始化不能用于赋值
 - 2. `int[] arr; arr={1,2,3,4};`；编译错误
 - 3. 对于数组的赋值只能对其元素进行赋值，不能对数组定义直接赋值。
 - 4. 可以采用第二种方式先用new创建空间，再赋值 `arr = new int[]{1,2,3,4}`

四、数组的访问

1. 获取数组的长度

```
int[] a = new int[]{3,6,7,9};  
int len = a.length; //数组的 length 属性  
System.out.println(len); //4
```

2. 通过下标访问数组元素

```
int[] b = {4,6,9,8,7}; //b[0] b[1] b[2]...b[4]  
//0. 获取最后一位元素值。  
System.out.println(b[b.length-1]); //7  
System.out.println("-----");  
//1. 获取元素下标为奇数的元素  
//循环问题 条件：数组的长度 变量：第一个元素 改变条件：%2==1  
for (int i = 0; i < b.length; i++) {
```

```

        if(i % 2 == 1) {
            System.out.println(b[i]);
        }
    }
//2.交换元素下标为2和3的元素位置
int temp = b[2];
b[2] = b[3];
b[3] = temp;
System.out.println(b[2]);
System.out.println(b[3]);

```

五、遍历数组元素

1. 基本for循环

```

int[] a2 = new int[]{3,6,7,9};
//基本for循环
for (int i = 0; i < a2.length; i++) {
    System.out.print(a2[i] + " ");
}
//jdk1.8提供了增强for循环
for (int i : a2) {
    System.out.print(i + " ");
}

```

2. 增强for循环

```

//jdk1.5提供了增强for循环
for (int i : a2) {
    System.out.print(i + " ");
}

```

3. 案例

- 产生一个随机数组，元素个数为10，将数组正序和逆序输出

```

//1.定义数组，长度为10
int[] num = new int[10];
//2.通过元素下标赋值随机数
for (int i = 0; i < num.length; i++) {
    Random ran = new Random();
    num[i] = ran.nextInt(100);
}
//3.正序遍历
for (int i = 0; i < num.length; i++) {
    System.out.print(num[i] + " ");
}
System.out.println();
//4.逆序遍历
for (int i = num.length-1; i >= 0; i--) {

```

```
        System.out.print(num[i] + " ");
    }
```

六、数组的复制

1. System.arraycopy()方法

- public static void arrayCopy(Object src,int srcPos, Object dest,int desPos,int length)
 - src: 源数组
 - srcPos: 源数组的起始位置
 - dest: 目标数组
 - destPos: 目标数组的起始位置
 - length: 复制的长度

```
int[] c1 = {10,30,65,45};
int[] c2 = new int[4];
System.arraycopy(c1, 1, c2, 0, 3);
System.out.println(Arrays.toString(c2));
```

- 如果目标数组小于需要copy的长度时会报出ArraysIndexOutOfBoundsException

2. Arrays.copyOf()方法

- java.util.Arrays类实现了数组复制的方法Arrays.copyOf(类型[] src,int length)
 - src: 源数组
 - length: 复制的长度

```
int[] c1 = {10,30,65,45};
System.out.println("s:" + c1.length);
c1 = Arrays.copyOf(c1, 3);
System.out.println(Arrays.toString(c1)); //截取
System.out.println("t:" + c1.length);
```

- 结论:
 - 当复制的长度小于新数组 则截取
 - 当复制的长度大于新数组 则扩容
- copyOf方法使用时源数组长度变化的原因:
 - 源数组长度不变
 - 源码中可以看出时创建了一个新数组使用System.arraycopy方法复制了数组

```
public static int[] copyOf(int[] original, int newLength) {
    int[] copy = new int[newLength];           新建了需要复制长度的新数组
    System.arraycopy(original, 0, copy, 0,
                     Math.min(original.length, newLength));
    return copy;
}
```

七、数组的排序

1. 数组的排序(深究)

- 排序是最常见的算法之一。
- 排序: 将数组元素按照从小到大或者从大到小的顺序排列。
- 常见的排序算法:[冒泡排序](#)、[选择排序](#)、[插入排序](#)、[希尔排序](#)、[快速排序](#)、[归并排序](#)等。

2. 冒泡排序

- 算法核心: 比较相邻两个元素, 将较大的元素互换。

```

System.out.println("请输入产生数组的长度:");
Scanner scan = new Scanner(System.in);
int length = scan.nextInt();
System.out.println("请输入你要产生的数组元素范围:");
int localtion = scan.nextInt();
int[] arr = new int[length];
for (int i = 0; i < arr.length; i++) {
    Random ran = new Random();
    arr[i] = ran.nextInt(localtion);
}
System.out.println("排序前:" + Arrays.toString(arr));
//冒泡排序
for (int i = 0; i < arr.length - 1; i++) {//排序次数
    for (int j = 0; j < arr.length - i - 1; j++) {//交换次数
        if(arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
System.out.println("排序后:" + Arrays.toString(arr));

```

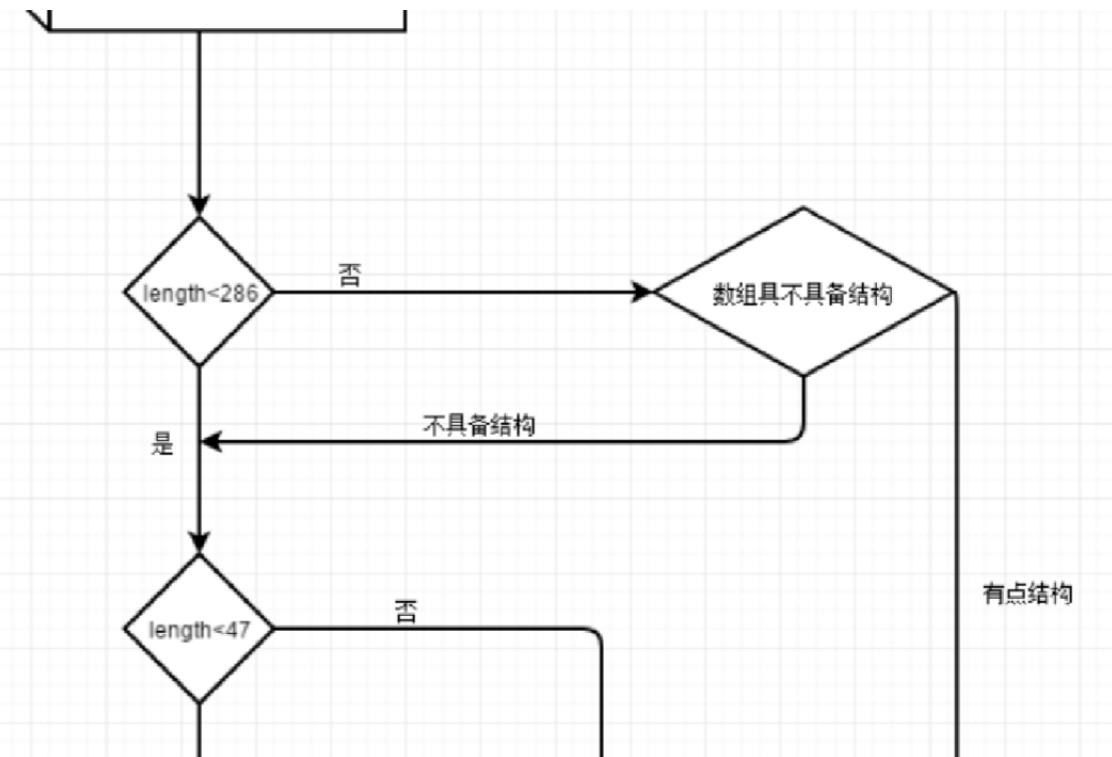
- `Arrays.sort`方法

```

System.out.println("排序前:" + Arrays.toString(arr));
//冒泡排序
Arrays.sort(arr);
System.out.println("排序后:" + Arrays.toString(arr));

```

- `sort()`方法原理结构



- 参考:<https://www.cnblogs.com/baichunyu/p/11935995.html>

方法(函数)

一、方法的定义

1. 定义方法(函数、过程)的功能

- 方法是一段封装特定逻辑功能的代码
- 方法可以被反复的调用，减少代码的重复性，便于程序的维护和优化。

```
//main函数 main方法 主方法 主函数
public static void main(String[] args) {
}

分析：
public static : [修饰词] 可以省略 不写表示当前所有修饰词为默认的。
void : 返回值类型 当前这个方法在使用后返回值的类型 void:无返回值类型
main : 方法名字 符合标识符命名规范且一般情况下采用小驼峰命名法。
(String[] args) : 参数列表 多个参数中间用逗号隔开 形式参数，当前方法调用时所需要传入的参数类型
{} : 方法的逻辑体
```

- 方法的5个要素: 修饰词 返回值类型 方法名 参数列表 方法体

2. 定义参数和返回值

- 在方法调用时，会将实际的参数值传递给方法的参数变量，必须保证参数类型和个数符合方法的声明。
- 方法在使用时，可以返回一个数据，称之为返回值。

- 方法在定义时必须指定返回值的类型
 - 有返回值时，写具体的返回值的类型
 - 无返回值时，写void

二、方法的调用

1.return语句

- 有返回值时，返回具体的返回值结果

```
public static double sum2(double a,int b) { //一定要声明合适的返回值的类型
    return a + b; //返回方法的返回值
}
```

- 返回值类型为void时，可以使用return；

```
public static void testfor() {
    for (int i = 0; i <= 1000; i++) {
        if(i == 999) {
            return; //立即结束方法，返回被调处
        }
        System.out.println(i);
    }
}
```

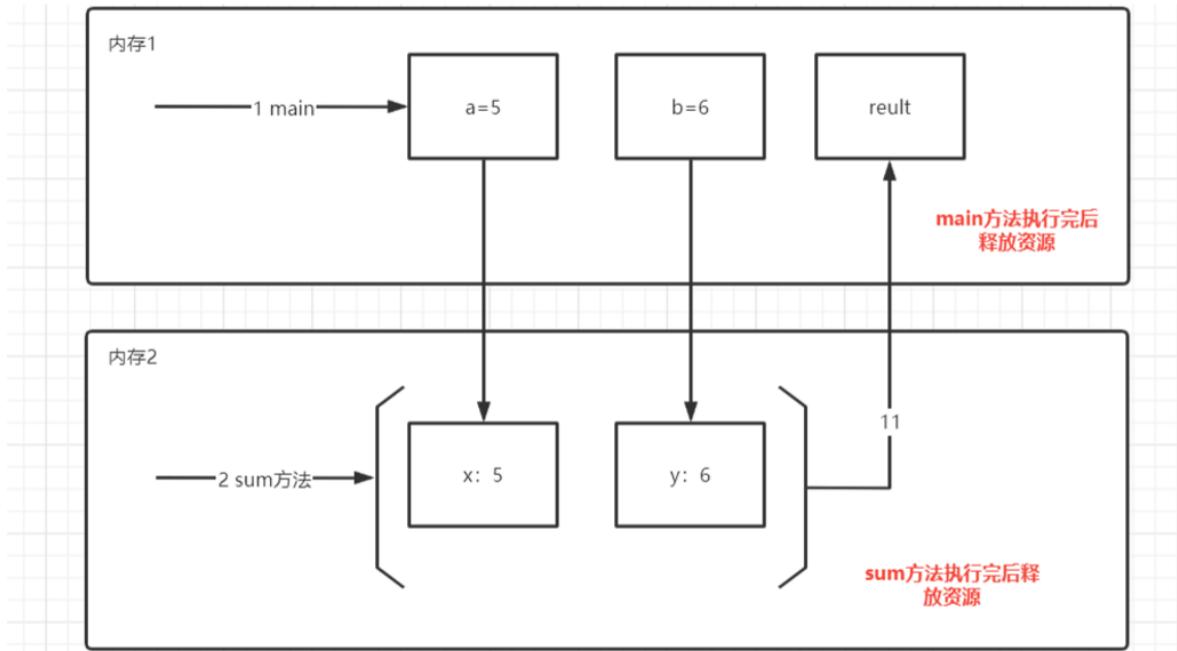
2.调用方法时的参数传递

```
package com.hqyj.basic.helloworld;

public class Test01 {

    public static int sum(int x , int y) {
        return x + y;
    }

    public static void main(String[] args) {
        int a = 5;
        int b = 6;
        int result = sum(a,b);
    }
}
```



综合案例

- 需求: 有猜数字游戏, 其游戏规则为: 1.(程序随机0-100以内随机整数)作为猜测的结果, 由玩家来猜测此数。2.(玩家可以猜测多次, 每猜测一次, 则由系统提示结果, 猜大了或者猜小了。如果猜测的完全正确, 则游戏结束, 计算玩家的游戏得分并输出;
- 演示: 本案例要求使用交互的方式实现此游戏: 由玩家在控制台输入所猜测的数字, 如果所猜测的数字与结果不相同, 则在界面输出比较后的结果及得分, 并提醒玩家继续猜测。交互过程如图所示:

***** Guessing *****

请输入0-100内的整数: 50

猜大了

25

猜小了

35

猜大了

30

猜对了

您一共猜了4次。

得分为: 88分。

由图, 每次猜测后, 程序将比较玩家所输入的数字, 然后提示结果: 猜大或猜小,

玩家终于猜测正确后, 游戏结束, 并给出游戏得分(每猜一次扣5分, 总100分)。

最后，如果玩家在控制台录入-1，则游戏中止，程序结束。交互过程如图所示：

```
***** Guessing *****
请输入0-100内的整数: -1
game over
```

- 方案

分析猜字母游戏可以看出，此程序需要存储随机生成的数字、系统产生的随机数、玩家猜测的总次数、玩家最后的得分，设计如下变量来存储此游戏中需要用到的相关数据：

- int 类型变量 randomNumber：用于记录系统产生的随机数；
- int类型变量inputNumber：用于保存用户猜测的数据。
- int类型变量count：用于记录用户所猜总次数；
- int类型变量score：用户的分数。

- 代码实现

```
package com.hqyj.basic.numbergame;

import java.util.Scanner;

public class GuessingGame {

    //主方法
    public static void main(String[] args) {
        //0 提示信息
        System.out.println("***** Guessing *****");
        System.out.print("请输入0-100内的整数:");
        //1 获取系统产生的随机数
        int randomNumber = random();
        //Scanner:输入语句的类，定义这个类
        Scanner scanner = new Scanner(System.in);
        //定义次数及得分
        int count = 0;
        int score = 100;

        //2 循环
        while(true) {
            //3 获取用户输入的数字
            int inputNumber = scanner.nextInt();
            //6 退出游戏，不想玩了
            if (inputNumber == -1) {
                System.out.println("game over");
                break;
            }
            //4 判断
            if (inputNumber > randomNumber) {
                System.out.println("猜大了");
                count++;
            } else if (inputNumber < randomNumber) {
                System.out.println("猜小了");
                count++;
            }
        }
    }
}
```

```
        } else {
            System.out.println("猜对了");
            count++;
            //5 猜对跳出循环
            System.out.println("您一共猜了" + count + "次。");
            System.out.println("得分为:" + (score-count*3) + "分。");
            break;
        }
    }

//随机生成一个0-100内的整数
public static int random() {
    return (int)(Math.random()*100+1);
}
```