第二章 标识符、常量、变量、类型转换

1.注释

> 程序中的注释，用来说明某段代码的作用，其增强了代码的**可读性**，让程序员快速的理解代码的含义和系统设计思路，同时便于系统交付后对代码日常维护以及后续迭代升级。

Java中的注释，分为三种：

- 单行注释
- 多行注释
- 文档注释（后续课程讲解）

1）单行注释

单行注释是最常用的注释方式，其注释内容从 "//"开始到本行末尾。

例如：

```
package com.briup.chap02;

//类名：Test01_LineNote
//作用：单行注释测试
public class Test01_LineNote {
    //main方法是程序的执行入口，且写法是固定的
    public static void main(String[] args){
        //向控制台中输出指定字符串内容
        System.out.println("hello world");
```

```
10
11          //下面一句代码被注释掉了，不会被执行
12          //System.out.println("hello Java");
13      }   //main方法结束
14  }//Test01_LineNote类定义结束
```

2）多行注释

多行注释以"/*" 开始，到 "*/" 结束，其可以注释一行，也可以注释多行

例如：

```
1   package com.briup.chap02;
2
3   /*
4       类名：Test01_MoreNote
5       作用：多行注释测试类
6    */
7   public class Test01_MoreNote {
8       /* main方法，固定写法，程序入口 */
9       public static void main(String[] args){
10          /* 向控制台中输出 */
11          System.out.println("hello world");
12
13          /* 下面多行代码被会被注释掉了，不会执行 */
14          /*
15              int a = 1;
16              System.out.println(a);
17          */
18      }   /* main方法结束 */
19  }   /* Test01_MoreNote类定义结束 */
```

> **注意**： 多行注释不能嵌套，否则会报错

下面的写法是**错误的**，编译会报错：

```
/*
    /*多行注释里面又嵌套了多行注释，编译报错*/
*/
public static void test(){
    //...
}
```

3）文档注释

格式：/** 文档注释内容 */

后续课程再给大家补充讲解！

4）困惑解答

思考：Java源代码中如果添加了注释，会不会导致字节码文件会臃肿变大？

回答：不会的。因为**源代码文件**Xxxx.Java 通过编译生成**字节码文件**Xxx.class的过程中，编译器会忽略掉源码中的注释部分。

> 注释是一个程序员必须要具备的编程习惯，初学者写程序可以养成习惯：先把实现思路用注释写下来，然后再补全代码。

2.关键字

关键字概述：

被Java语言赋予**特定含义的单词**

关键字特点：

组成关键字的**字母全部小写**，常用的代码编辑器对关键字都有高亮显示

常见关键字：

```
public  static  void  class
```

Keywords in Java：

◆ abstract	do	implements	private	throw
◆ boolean	double	import	protected	throws
◆ break	else	instanceof	public	transient
◆ byte	extends	int	return	true
◆ case	false	interface	short	try
◆ catch	final	long	static	void
◆ char	finally	native	super	volatile
◆ class	float	new	switch	while
◆ continue	for	null	synchronized	default
◆ if	this	package	const	goto

注意：

- const 和 goto 是Java中的保留字（暂时没用，后续可扩展使用）
- true 和 false 不是关键字，是boolean类型的字面值，但是也不能用做自定义标识符

3.标识符

> 在Java中给类、方法、变量起的名字，就是标识符，其可以用来标识这个类、方法、变量

命名规则：

- 标示符可以由字母、数字、下划线_ 、美元符号$组成
- 标示符开头不能是数字
- 标识符中的字符大小写敏感
- 标识符的长度没有限制
- 标示符不能使用Java中的关键字或保留字

案例展示：

合法标示符	非法标示符
try1	try#
GROUP_1	1GROUP
helloworld	hello-world
_int	int
$int	$-int

命名约定：

采用**驼峰命名法**，具体要求如下：

- **类和接口**

不管是1个还是多个单词，每个单词的首字母都大写（大驼峰）

```
public class Account { }
```

```
public interface UserService { }
```

- **方法和变量**

  首字母小写，第二个单词开始每个单词的首字母大写（小驼峰）

  ```
  public String getName(){ }
  ```

  ```
  int totalNum = 1;
  ```

- **常量**

  全部字母大写，如果是多个单词，单词与单词间使用下划线分隔

  ```
  public static final int MAX_NUM = 10;
  ```

- 尽量做到**见名知意**，使用有意义的名字

  ```
  int numOfStudent = 10;
  ```

  ```
  String userName = "tom";
  ```

4.常量

概念：

在程序运行过程中，**其值不可以发生改变的量**，称为常量

常量分类：

- **字面值常量**
- **自定义常量** (面向对象部分讲解)

字面值常量：

- 字符串常量

  用双引号括起来的多个字符（可以包含0个、一个或多个）

  例如："、"a"、"abc"、"中国"等

- 整数常量

  整数值，例如：-10、0、88等

- 小数常量

  小数值，例如：-5.5、1.0、88.88等

- 字符常量

  用单引号括起来的一个字符

  例如：'a'、'5'、'B'、'中'等

- 布尔常量

  布尔值，表示真假，只有两个值：`true、false`

- 空常量

  一个特殊的值，空值：`null`

注意事项：

- 除空常量外，其他常量均可使用输出语句直接输出
- 大家可以好好揣摩下`字面值常量`这个词，这个字面值是固定、不可改变的，即常量值

案例展示：

```java
package com.briup.chap02;

public class Test04_Constant {
    //输出各种类型的常量值
    public static void main(String[] args) {
        System.out.println(10);     // 输出整数常量
        System.out.println(5.5);    // 输出小数常量
        System.out.println('a');    // 输出字符常量
        System.out.println(true);   // 输出boolean常量值true
        System.out.println("欢迎来到杰普软件"); // 输出字符串常量
    }
}
```

5.变量

变量的定义有2种格式，分别如下：

- **格式1**

  `数据类型 变量名;`

  `变量名 = 数据值;`

- **格式2**（推荐）

  `数据类型 变量名 = 数据值;`

注意事项：变量一定要求先声明、再赋值、之后才能使用

案例展示：

```java
package com.briup.chap02;
```

```java

public class Test05_Variable {
    public static void main(String[] args) {
        // 第一种定义格式
        //变量声明
        int a;
        //变量赋值
        a = 10;
        //使用变量
        System.out.println(a);

        // 第二种定义格式：声明变量的同时就进行赋值（初始化）
        int b = 20;
        System.out.println(b);
    }
}
```

定义格式理解：

数据类型：`Java是强类型语言`，每个常量或变量都有对应的数据类型（后面章节专门介绍）

变量名：就是一个标识符，用来标识内存上的一块空间

数据值：即变量值，该值会存储到变量名标识的那块空间中

内存基础知识：

理解变量内存结构之前，我们需要补充一点内存区域的基础知识，大家**了解即可**，详见下表：

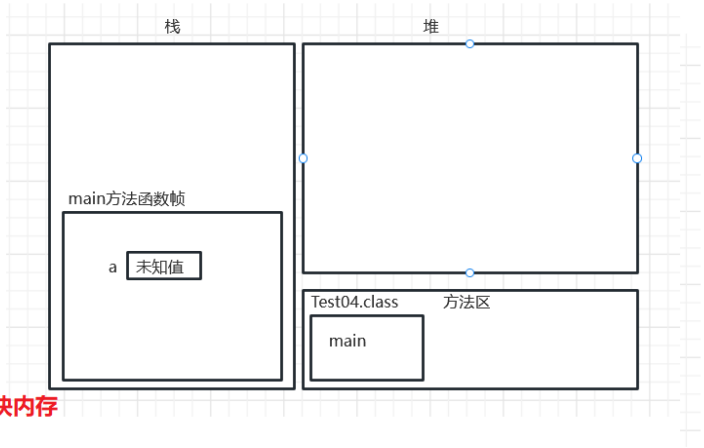
区域名称	作用
寄存器	给CPU使用，~~和我们开发无关，忽略~~
本地方法栈	JVM在使用操作系统功能的时候使用，~~和我们开发无关，忽略~~
方法区	存储可以运行的class文件
方法栈	方法运行时使用的内存，比如main方法运行，进入方法栈中执行
堆内存	存储数组或对象，new来创建的，都存储在堆内存（后续课程介绍）

程序执行流程：

1. 执行命令`java 测试类名`，系统将`测试类.class文件`内容装入内存中的方法区
2. 接下来找到里面的`main方法`（程序入口）
3. 然后在栈空间申请一块区域（函数帧）来保证main方法的执行
4. 最后顺序执行main方法中的代码
5. 遇到main方法的`"}"或return语句`，方法执行结束，main方法函数帧内存释放

单个变量内存理解：

```java
public class Test04{
    public static void main(String[] args) {
        // 第一种定义格式
        int a;
        a = 10;
        //如果没有给变量a赋初值,
        //则不能访问变量，编译报错 error
        System.out.println(a);
    }
}
```



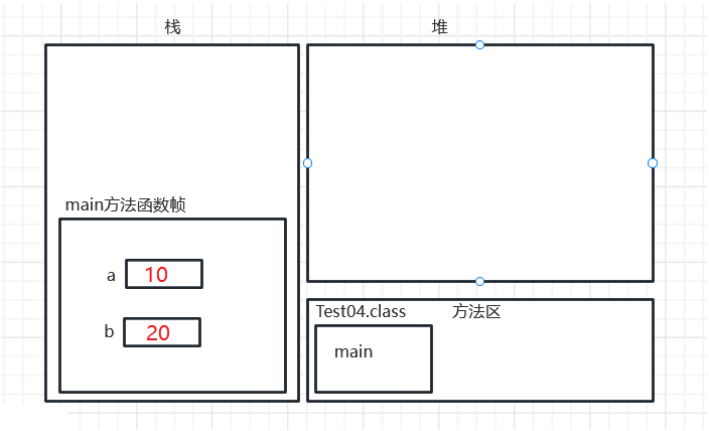
注意事项：

1. a为变量名，是标识符中的一种，用来标识一块内存

2. 变量定义以后必须赋初值，否则不能使用

多个变量内存理解：

```java
public class Test04 {
    public static void main(String[] args) {
        // 第一种定义格式
        int a;
        // 给变量赋值
        a = 10;
        // 运行正常，从a标识的空间中取出10并输出
        System.out.println(a);

        // 第二种定义格式
        int b = 20;
        System.out.println(b);
    }
}
```



变量小结：

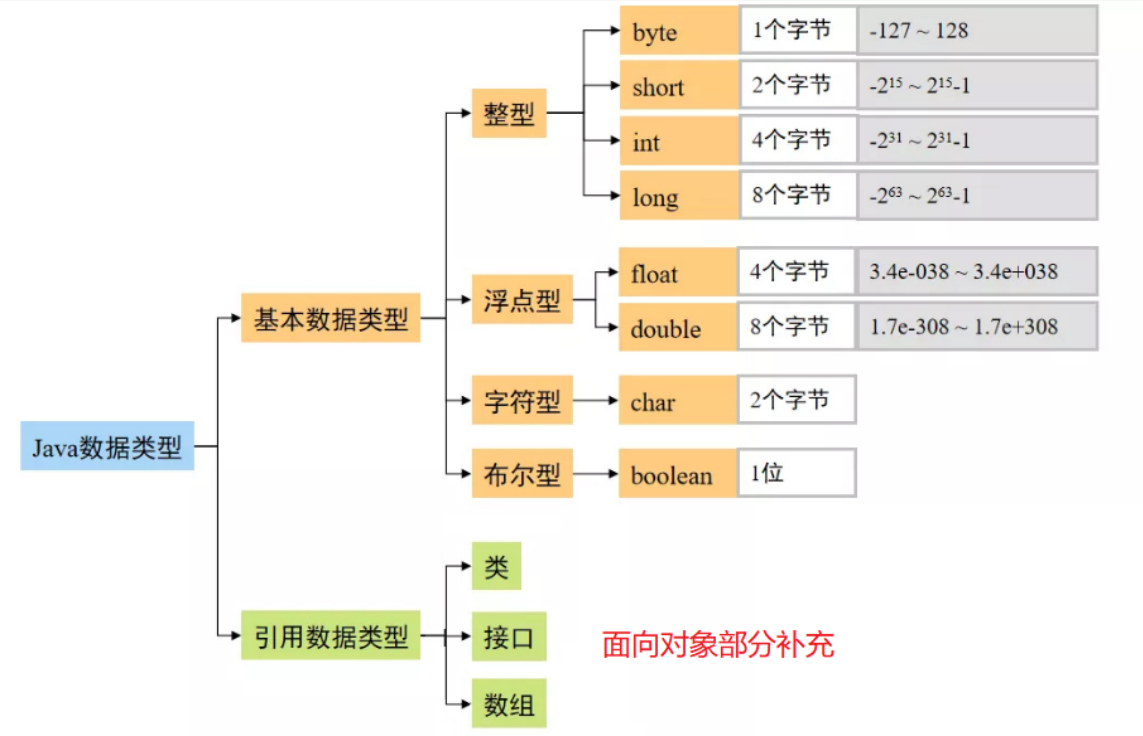
- 变量名是标识符中的一种，用来标识一块内存区域
- 变量定义后没有赋初值，不能直接使用，编译报错
- 变量值的类型应该跟变量的类型一致（关于数据类型，后续会补充）

6.数据类型

Java语言是强类型语言，每一种数据都定义了明确的数据类型，不同类型变量占用内存大小不同，取值范围不同。

Java数据类型可以分为两大类：

- **基本数据类型（本章重点讲解）**
  - 整形 byte、short、int、long
  - 浮点型 float、double
  - 字符类型 char
  - 布尔类型 boolean
- **引用数据类型（面向对象部分补充）**
  - 数组
  - 类
  - 接口



6.1 字节

比特位： bit，是计算机存储信息的最小单位，存放一位二进制数，即 `0 或 1`

字节： byte，是计算机**分配内存的最小单位**，通常用大写字母 `B` 表示，一个字节包含了8个比特位，即：`1byte == 8bits`

大家可看下自己硬盘中存储的文件，也是以字节为单位存储的：

计算器存储容量的单位，最小单位为字节，其他常见的还有**KB、M、G、T**，换算过程如下：

- 1KB (Kilobyte 千字节) = 1024B，其中1024=2^10 (2 的10次方)
- 1MB (Megabyte 兆字节 简称“兆”) = 1024KB
- 1GB (Gigabyte 吉字节 又称“千兆”) = 1024MB
- 1TB (Trillionbyte 万亿字节 太字节) = 1024GB

6.2 基本类型

前面的章节中我们大致认识了下基本数据类型，知道其可以分为4大类8小种，且每种表示范围不同。

思考：为什么数据类型有这么多种呢？

- 不同据类型占用的内存空间是不同的
- 不同的内存空间，所存储的数据的大小范围是不一样的

数据类型	关键字	内存占用	取值范围
字节型	byte	1个字节	-128~127 -2^7 ~ 2^7-1
短整型	short	2个字节	-32768~32767 -2^15 ~ 2^15-1
整型	int（默认）	4个字节	-2的31次方~2的31次方-1
长整型	long	8个字节	-2的63次方~2的63次方-1
单精度浮点数	float	4个字节	1.4013E-45~3.4028E+38
双精度浮点数	double（默认）	8个字节	4.9E-324~1.7977E+308
字符型	char	2个字节	0-65535 0 ~ 2^16-1
布尔类型	boolean	1个字节	true，false

Java中的默认类型：整数类型是`int`、浮点类型是`double`。

额外说明：

e+38表示是乘以10的38次方，同样，e-45表示乘以10的负45次方。

> 4大类基本数据类型介绍如下

1）整数型

`byte short int long`

四种数据类型，都可以表示有符号的整形数，但是表示范围不同。

```
byte  a1 = 1;   //(内存中占8位) 1字节
short a2 = 1;   //(内存中占16位)2字节
int   a3 = 1;   //(内存中占32位)4字节
long  a4 = 1L;  //(内存中占64位)8字节
```

注意：使用long类型数据的时候，后面要加大写L或者小写l,建议加上大写的L。

2）浮点型

`float double`，可以表示小数，但表示范围不同。

```
//后面加f或者F
float f = 10.5f;

double d1 = 10.5;
//后面也可以加d或者D
double d2 = 10.5D;
```

3）布尔型

`boolean`，该类型变量的取值只能是`true或false`

```
boolean f1 = true;
boolean f2 = false;
```

4）字符型

`char`，用于表示一个16位的Unicode字符（包括中文汉字），其类型值用一般`''`括起来单个字符，范围是0-65535。

```
package com.briup.chap02;

public class Test062_Char {
    public static void main(String[] args) {
        char c1 = 'a';
        char c2 = 'Z';
        char c3 = ' ';
        char c4 = '1';
        char c5 = '中';
```

```
10
11          System.out.println(c1);
12          System.out.println(c2);
13          System.out.println(c3);
14          System.out.println(c4);
15          System.out.println(c5);
16      }
17  }
18  //输出结果：
19  a
20  Z
21
22  1
23  中
```

错误用法：

```
1  //至少要包含1个字符
2  char c1 = '';
3
4  //只能包含1个字符，多个不行
5  char c2 = 'abc';
```

剩余char类型内容，放到下一章节专门讨论。

综合案例展示：

定义变量时，我们应该根据变量可能的取值范围，来选择合适的数据类型。


```java
String name = "高启强";
int age = 32;
char gender = '男';
double height = 175.6;
boolean isStudyJava = true;
```

注意：String为字符串类型，它不是基本数据类型，是类类型。

String.class

```java
public final class String   JDK提供的类类型
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];
```

6.3 进制基础

概念：

进制就是进位制，是人们规定的一种进位方法，二进制逢2进1，八进制是逢8进1，十进制逢10进1，十六进制逢16进1。

不同进制形式：

- 二进制　　**0b|B开头**，由0和1组成
- 八进制　　**0开头**，0、1...6、7组成

- 十进制　　常见整数，0、1...8、9组成
- 十六进制　**0x或0X开头**，0、1...8、9、a、b、c、d、e、f组成，大小写不区分

案例描述：

```
package com.briup.chap02;

public class Test063_Binary {
    public static void main(String[] args) {
        byte b1 = 0b01100001;   //二进制
        byte b2 = 97;           //十进制
        byte b3 = 0141;         //八进制
        byte b4 = 0x61;         //十六进制

        //打印出来结果全是97，为什么？
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        System.out.println(b4);
    }
}
```

注意事项：不论什么类型的数据值，在计算机的底层存储时，统一按照二进制形式存储！

上述案例中，0b01100001、97、0141、0x61在计算机底层存储时，都是按二进制存储的，其值按照十进制表示，都是97。

思考：如何将2进制、8进制、10进制、16进制数进行转换呢？

6.4 进制转换

额外提醒：6.4、6.6、6.7三个小章节大家学习时重在理解，进而熟悉，没必要死记硬背！

1）任意进制转换为10进制

转换方式：

`结果值 = 系数*基数的权次幂相加`

系数：每一位上的数据

基数：X进制，基数就是X

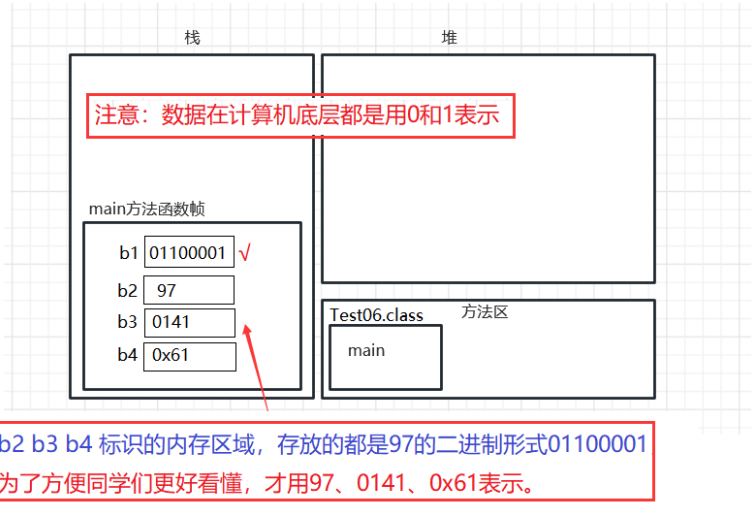
权：最右边那位对应0，每左移一位加1

案例展示：

```
// 97：系数为9和7，基数为10，权是0和1
97: 7*10^0 + 9*10^1 = 7 + 90 = 97

0b01100001: 1*2^0 + 0*2^1 + 0*2^2 + 0*2^3 + 0*2^4 + 1*2^5 + 1*2^6
    = 1 + 32 + 64 = 97

0141：1*8^0 + 4*8^1 + 1*8^2 = 1 + 32 + 64 = 97

// 0x61：系数为6和1，基数为16，权是0和1
0x61：1*16^0 + 6*16^1 = 1 + 96 = 97
```

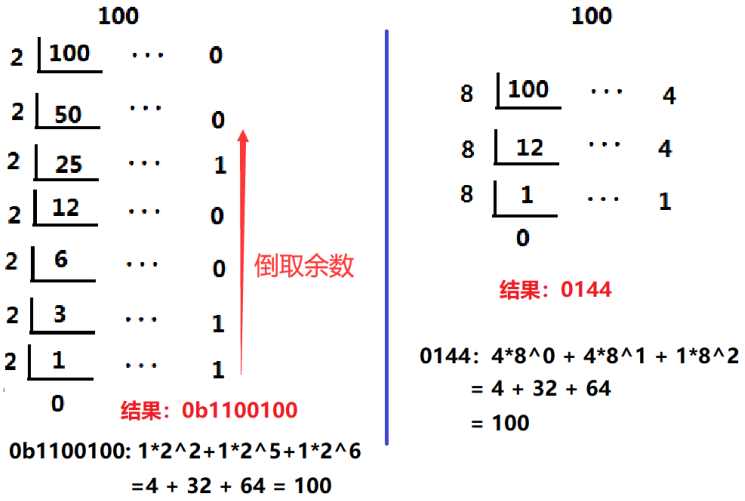
变量内存理解：

```java
public class Test06 {
    public static void main(String[] args) {
        byte b1 = 0b01100001; //二进制
        byte b2 = 97;         //十进制
        byte b3 = 0141;       //八进制
        byte b4 = 0x61;       //十六进制
    }
}
```



2）十进制转换为其他进制

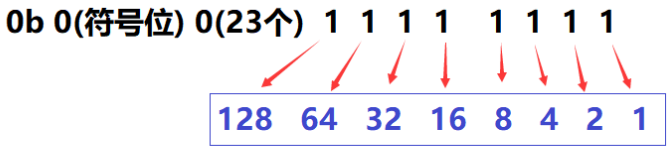
除积倒取余，具体计算过程如下：



16进制也是一样的换算方式。

3）十进制到二进制的快速转换

对于不太大的正整数，我们可以采用一种快捷的方式来获取其2进制形式，具体可参考下图：



97: 64 + 32 + 1 = 0b 0 0(23) 0110 0001

277: 256 + 16 + 4 +1 = 0b 0 0(15) 0000 0001 0001 0101

上图运算步骤解析:

- 拆解正整数，将其分解为2的指数倍相加
- 找出2的指数倍 对应的 二进制1
- 根据变量类型确定占用的字节数及比特位，组合得到最终的二进制形式

4）二进制转换为8、16进制

- **2进制转化为8进制**

  从最低位开始，每3位分一组，不足3位的话高位补0

  将得到的数字组合到一起，最前面以0开头

案例展示：

```
//定义变量
byte b = 126;

//获取其二进制形式
// 126 = 127 - 1 或 64 + 32 + 16 + 8 + 4 + 2
//  0b 0111 1110
//2进制 -> 8进制
//a.从最低位开始，每3位分一组，不足3位则高位补0
//  001 111 110
//   1   7   6
//b.最后组合到一起，最前面以0开头
//结果：0176
```

```
byte b2 = 0176;
System.out.println(b == b2);
```

- **2进制转换为16进制**

  从最低位开始，每4位分一组，不足4位的话高位补0

  将得到的数字组合到一起，最前面以0x开头

案例展示：

```
//定义变量
byte b = 126;

//二进制形式：0b 0111 1110
//2进制 -> 16进制
//a.从最低位开始，每4位分一组，不足4位则高位补0
//  0000 0111 1110
//    0    7   e(14)        其中a:10 b:11 c:12 d:13 e:14 f:15
//b.最后组合到一起，最前面以0x开头
//结果：0x07e
byte b2 = 0x07e;
System.out.println(b == b2);
```

6.5 字符类型

> char类型用于表示一个占2个字节（16位）的Unicode字符（包括中文汉字），其是基本数据类型，取值范围为0-65535。

1）问题引入

一般情况下，我们使用**单引号**括起来单个字符来表示字符值。

但也可以用其他方式表示字符值，具体见下面案例：

```java
package com.briup.chap02;

public class Test065_Char {
    public static void main(String[] args) {
        //int字面值常量也可以直接赋值给char变量
        char c1 = 48;
        char c2 = 65;
        char c3 = 97;

        //思考1：为什么c1、c2、c3输出值为字符 '0' 'A' 'a'
        System.out.println(c1); //0
        System.out.println(c2); //A
        System.out.println(c3); //a

        System.out.println("-------------");

        //使用char字面值常量也可以赋值给int变量
        int num = '中';
        //思考2：为什么输出 20013
        System.out.println(num);    //20013

        //思考3：为什么c4的输出结果为 '中'
        char c4 = 20013;
        System.out.println(c4);     //中
    }
}
```

观察并运行上述案例。

思考上面提出的三个疑问：

1. 为什么c1、c2、c3输出值为字符 `'0' 'A' 'a'`
2. 为什么输出num结果为 20013
3. 为什么输出c4 结果为`'中'`

2）字符编码知识补充

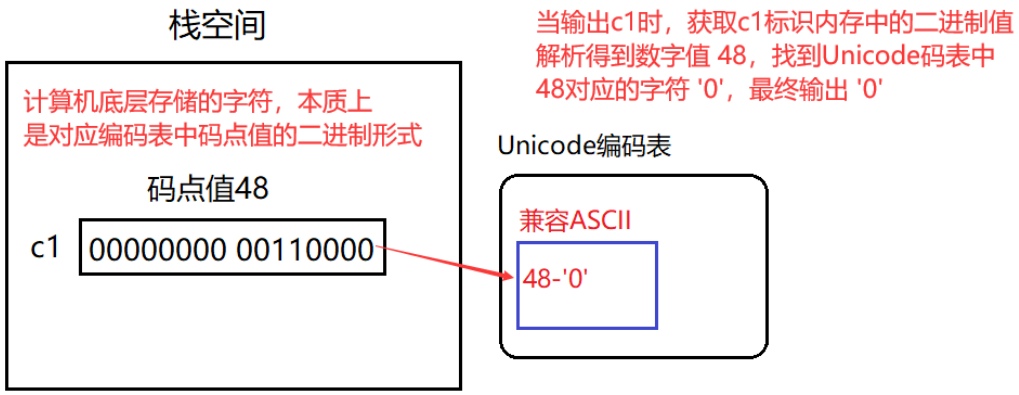
- **字符编码则定义了字符与字节之间的转换关系**；
- Java 表示内部字符采用 Unicode 字符集；
- Unicode 是一种国际字符集，旨在为世界上几乎所有的字符提供唯一的数字编码，它定义了每个字符与一个唯一的数字值之间的映射关系，这个数字值称为 **Unicode 码点**（code point）。

问题：上述案例中`char c1 = 48;`为什么输出c1结果为`'0'`

解答：

Java采用 Unicode 编码，其兼容 ASCII 编码，在ASCII编码表中，**码点是48对应字符为`'0'`**。

内存理解：



ASCII编码表如下图：

ASCII表

（ American Standard Code for Information Interchange 美国标准信息交换代码 ）

高四位 / 低四位		ASCII控制字符												ASCII打印字符											
		0000						0001						0010		0011		0100		0101		0100		0111		
		0						1						2		3		4		5		6		7		
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl
0000	0	0		^@	NUL	\0	空字符	16	▶	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH		标题开始	17	◀	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX		正文结束	19	‼	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK		肯定应答	22	▬	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	\a	响铃	23	↨	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[	ESC	\e	溢出	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	\f	换页	28	∟	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI		移入	31	▼	^-	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	⌂	^Backspace 代码：DEL

注：表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入。

3）其他常见编码（了解即可）

- **ASCII**（American Standard Code for Information Interchange）

  ASCII 是最早的字符编码标准，使用 7 位（8 位的扩展 ASCII）来表示字符，包括英文字母、数字、标点符号等，共计 128（或 256）个字符。

- **Unicode**

  Unicode 是一种国际字符集，旨在为世界上几乎所有的字符提供唯一的数字编码。

  它使用 16 位（UTF-16）或 32 位（UTF-32）来表示字符，**包括 ASCII 字符**、各种语言字符、符号、表情符号等，共计超过 1,100,000 个字符。

- **UTF-8（Unicode Transformation Format-8）**

  UTF-8 是一种变长编码方案，使用 1 到 4 个字节来表示字符。

  它**能够表示 Unicode 字符集中的所有字符，并且向后兼容 ASCII 编码。**

UTF-8 是互联网上最常用的字符编码方式。

- UTF-16（Unicode Transformation Format-16）

  UTF-16 是一种定长编码方案，使用 2 个或 4 个字节来表示字符。

  它能够表示 Unicode 字符集中的所有字符，但在表示一些特殊字符时需要使用 4 个字节。

- GB2312

  GB2312 是中国国家标准，用于表示简体中文字符。

  它使用 2 个字节来表示一个中文字符，共计包含了 6,763 个常用汉字。

- GBK（Guo Biao Kan）

  GBK 是汉字编码标准之一，主要用于表示中文字符。

  它是对 GB2312 编码的扩展，使用 2 个字节来表示一个中文字符，共计包含了 21,692 个汉字。

- ISO-8859-1

  ISO-8859-1 是一种单字节编码，也被称为 Latin-1。

  它能够表示西欧语言中的字符，包括英文字母、西班牙语、法语、德语等字符。

注意：每一种字符编码，都有一个与之对应字符编码表，其中包含了每个字符对应的码点值。

4）特殊字符值

给字符变量赋值，我们除了可以用10进制的码点值，还可以使用2、8、16进制码点值，也可以使用Unicode编码值，具体见下面案例：

```
1  public static void main(String[] args) {
2      // 普通字符赋值
3      char c1 = 'a';
4      // 使用10进制表示字符
```

```
5       char c2 = 97;
6       // 使用8进制表示字符
7       char c3 = 0141;
8       // 使用16进制表示字符
9       char c4 = 0x0061;
10
11      //使用Unicode编码值表示字符a
12      char c5 = '\u0061';
13
14      System.out.println(c1 + " " + c2 + " " + c3 + " " + c4 + "
    " + c5);
15  }
16  //输出结果：
17  a a a a a
```

注意：实际开发时，我们用码点给字符变量赋值的情况很少，大多数情况下还是使用常规字符赋值。

5）转义字符

在 Java 中，转义字符是一种特殊的字符序列，用于表示一些特殊字符或具有特殊含义的字符。

转义字符以反斜杠（`\`）开头，后面跟着一个或多个字符。

常用转义字符	效果
\n	换行符，将光标定位到下一行的开头
\r	回车，把光标移动到行首(和环境有关)
\t	垂直制表符，将光标移到下一个制表符的位置
\\	反斜杠字符，\是一个特殊含义字符，第一个\用于去除第二个\的特殊含义
\'	单引号字符
\"	双引号字符
\uXXXX	Unicode 转义，用于表示 Unicode 字符，其中 `XXXX` 是四位十六进制数。

案例展示1：

```java
//转义字符案例
public static void main(String[] args) {
    char c1 = '\t';
    System.out.println("c1: " + c1);
    //查询ASCII表可知，\t 对应 码点 9
    System.out.println(c1 == 9);

    System.out.println("---------------");

    char c2 = '\n';
    //查询ASCII表可知，\n 对应 码点 10
    System.out.println("c2: " + c2);
    System.out.println(c2 == 10);

    System.out.println("---------------");
```

```
16
17      // \可以去除后面'的特殊含义，仅标识字符 '
18      char c3 = '\'';
19      System.out.println("c3: " + c3);
20  }
21
22  //程序输出：
23  c1:
24  true
25  ---------------
26  c2:
27
28  true
29  ---------------
30  c3: '
```

案例展示2：

在字符串中使用转义字符。

```
1   public static void main(String[] args) {
2       String str1 = "Hello, \"World\"!"; // 使用转义字符表示双引号
3       String str2 = "I\'m Java programmer."; // 使用转义字符表示单
    引号
4       String str3 = "C:\\path\\to\\file.txt"; // 使用转义字符表示反
    斜杠
5       String str4 = "First line\nSecond line"; // 使用转义字符表示
    换行
6       String str5 = "Hello\tworld!"; // 使用转义字符表示制表符
7       String str6 = "Unicode character: \u03A9"; // 使用 Unicode
    转义表示特殊字符
8
9       System.out.println(str1);
10      System.out.println(str2);
11      System.out.println(str3);
```

```
12        System.out.println(str4);
13        System.out.println(str5);
14        System.out.println(str6);
15    }
```

运行效果：

```
public class Test065_Char {
    public static void main(String[] args) {
        String str1 = "Hello, \"World\"!"; // 使
        String str2 = "I\'m Java programmer."; //
        String str3 = "C:\\path\\to\\file.txt";
        String str4 = "First line\nSecond line";
        String str5 = "Hello\tworld!"; // 使用转义
        String str6 = "Unicode character: \u03A9"

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(str5);
        System.out.println(str6);
```

```
<terminated> Test065_Char [Java Application] C:\Program Files\Jav
Hello, "World"!
I'm Java programmer.
C:\path\to\file.txt
First line
Second line
Hello    world!
Unicode character: Ω
```

6.6 原反补码

学习这个知识点之前，我们先来看一个题目：写出10的二进制形式

答案及解读：

> `0b 0 0(23个) 0000 1010`
>
> 10对应的类型为int，在计算机底层占4字节，需要32个比特位表示
>
> 其中最高位为符号位，0表示正数，1表示负数
>
> 剩下的31位，其中23位都为0，低8位为0000 1010 = 8 + 2 = 10
>
> 连到一起，结果为正整数10

思考：如何表示-10的二进制形式呢？

如果要表示负整数的二进制形式，则必须学习原码、反码、补码。

注意事项：计算机底层运算时，以变量的补码方式进行

1）原码

就是二进制定点表示法，即最高位为符号位，“0”表示正，“1”表示负，其余位表示数值的大小

```
1    10的原码：0 0(23) 0000 1010
2   -10的原码：1 0(23) 0000 1010
```

2）反码

- 正数的反码与其原码相同
- 负数的反码，在原码的基础上，保留符号位，其他位逐位取反

```
1    10的反码：    跟10的原码相同          0 0(23) 0000 1010
2   -10的反码：    拿到-10的原码，         1 0(23) 0000 1010
3                 保留符号位其他位取反    1 1(23) 1111 0101
```

3）补码

- 正数的补码与其原码相同
- 负数的补码是在其反码的末位加1

```
1    10的补码：    跟10的原码相同          0 0(23) 0000 1010
2   -10的补码：    拿到-10的反码，         1 1(23) 1111 0101
3                 在反码基础上加1         1 1(23) 1111 0110
```

案例验证：

```
模拟计算机底层进行运算：-10 + 10;
计算机底层通过补码进行运算，先获取-10补码：1 1(23) 1111 0110
                                        +
                                再获取 10补码：0 0(23) 0000 1010
                                        =
                                结果：       1【0 0(23) 0000 0000】
结果分析：两个int类型数据相加后结果值类型仍旧是int,其中int类型表示范围为4字节32个比特位，所以上述结果中第33位的那个1被自动抛弃，只保留低32位数值，0 0(23) 0000 0000，即 0。
所以：-10 + 10 == 0
```

课堂练习：

请写出-123的原反补码。

6.7 扩展内容

前面我们学过byte类型变量的取值范围是[-128，127]，char类型的取值范围是[0，65535]，请用二进制、原反补码推导出来（**理解即可，不用完全掌握推导过程**）

```
//char取值范围推导过程
1.char类型只能表示0或正整数，其最高位不表示符号位
2.char类型占用2个字节，16个比特位
3.最小值 0000 0000 0000 0000 ->  0
4.最大值 1111 1111 1111 1111 ->  65535（大家可借助计算器，也可以手算）

//byte取值范围推导过程
```

```
1.byte类型可以表示正负整数和0
2.byte类型占用1个字节，8个比特位
3.8个比特位 1111 1111 能够表示的数字有 256个
4.最大值 0111 1111 -> 127
5.中间值 0000 0000 -> 0
6.其他值 8个bits能够表示256个数，其中[0,127]占用了128个，剩余的数都是小于0的，共128个，结果可推理出来为[-128,-1]
7.总结，表示范围[-128,127]
```

大家理解上述推导过程后，**记下几个整形的取值范围即可**：

类型	字节	比特位	能否负值	取值范围
byte	1byte	8bits	能	[-128,127] [-2^7,2^7-1]
short	2byte	16bits	能	[-2^15,2^15-1]
char	2byte	16bits	否	[0,65535] [0,2^16-1]
int	4byte	32bits	能	[-2^31,2^31-1]
long	8byte	64bits	能	[-2^63,2^63-1]

注意：浮点型的二进制计算方式不同于整形数，是另一种计算方式，大家感兴趣的话，课后自行了解，开发及面试一般不需要。

7.常量补充

观察之前的常量案例，思考其中常量的数据类型

```java
package com.briup.chap02;

public class Test07_Constant {
    //输出各种常量值
    public static void main(String[] args) {
        System.out.println(10);     // 整数常量 默认类型int
        System.out.println(5.5);    // 小数常量 默认类型double
        System.out.println('a');    // 字符常量 默认类型char
        System.out.println(true);   // boolean常量值true
        System.out.println("欢迎来到杰普软件"); // 输出字符串常量 String
    }
}
```

观察下面案例，思考案例中常量的数据类型

注意：整形字面值，不论是二进制、八进制还是十进制、十六进制，默认都是int类型常量。

```java
//输出各种常量值
public static void main(String[] args) {
    System.out.println(10);     // int
    System.out.println(0b0110); // ?
    System.out.println(026);    // ?
    System.out.println(0x1c);   // ?
    System.out.println(5.5);    // double
    System.out.println('a');    // char
    System.out.println(true);   // boolean
    System.out.println("欢迎来到杰普软件"); // String
}
```

思考：long、float等类型的常量，该如何书写?

- **整形数后面加'L'或'l'，就表示long类型字面值常量**
- **小数后面加'F'或'f'，就表示float类型字面值常量**

```
//输出各种常量值
public static void main(String[] args) {
    System.out.println(10);     // int
    //整形数后面加'L'或'l'，就表示long类型字面值常量
    System.out.println(0x1cl);  //long
    System.out.println(10L);    //long

    System.out.println(5.5);    // double
    //小数后面加'F'或'f'，就表示float类型字面值常量
    System.out.println(3.14F);  //float
    System.out.println(-2.0f);  //float

    //下面用double类型常量值给f1赋值，编译报错，为什么？
    //float f1 = 5.5;
    float f2 = 5.5F;

    int num = 20;
    //下面给num赋值编译报错，为什么？
    //num = 1000000000000L;
    System.out.println(num);
    //下面给count赋值编译运行正常，为什么？
    long count = 1000000000000L;
    System.out.println(count);
}
```

扩展内容：思考下面代码中的为什么编译报错

```
package com.briup.chap02;

public class Test07_Extend {
    //输出各种常量值
```

```
5       public static void main(String[] args) {
6           //1.为什么下面用double类型常量值给f1赋值，会编译报错？
7           //float f1 = 5.5;
8           float f2 = 5.5F;
9
10          int num = 20;
11          //2.为什么下面给num赋值，会编译报错？
12          //num = 1000000000000L;
13          System.out.println(num);
14
15          //3.为什么下面给count赋值，编译运行正常？
16          long count = 1000000000000L;
17          System.out.println(count);
18      }
19  }
```

8.类型转换

基本数据类型表示范围大小排序：

$$\text{byte} < \begin{matrix}\text{short} \\ \text{char}\end{matrix} < \text{int} < \text{long} < \text{float} < \text{double}$$

在变量赋值及算术运算的过程中，经常会用到**数据类型转换**，其分为两类：

- **隐式类型转换**
- **显式类型转换**

8.1 隐式类型转换

- 赋值过程中，小数据类型值或变量可以直接赋值给大类型变量，类型会自动进行转换

案例展示：

```
package com.briup.chap02;

public class Test081_ImplicitTrans {
    public static void main(String[] args) {
        // int类型值 赋值给 long类型变量
        long num = 10;
        System.out.println(num);

        // float类型值 赋值给 double类型变量
        double price = 8.0F;
        System.out.println(price);

        char c = 'a';
        // char 赋值给 int
        int t = c;
        System.out.println(t);

        // 下面会编译报错
        //float pi = 3.14;
        //int size = 123L;
        //int length = 178.5;
    }
}
```

- byte、short、char**类型**的数据在进行算术运算时，会**先自动提升为**int，然后再进行运算
- 其他类型相互运算时，表示**范围小的会自动提升为范围大**的，然后再运算

案例展示：

```
public static void main(String[] args) {
    byte b = 10;
    short s = 5;
    // (byte -> int)  +  (short -> int)
    //          int   +   int
    //      结果为    int
    int sum = b + s;

    // 下一行编译报错，int 无法自动转换为 short进行赋值
    //short sum2 = b + s;

    System.out.println(sum);
}
```

案例展示：

```
public static void main(String[] args) {
    byte b = 10;
    short s = 5;
    double d = 2.3;
    //      (byte10->int10 - 5) * (short->int5) -> 5 * 5 = 25
    //              int25   +  double2.3
    //           double25.0 +  double2.3
    //      结果：double 27.3，必须用double变量来接收该值
    double t = (b - 5) * s + d;

    // double赋值给float，编译报错
    // float f = (b - 5) * s + d;
    System.out.println(t);
}
```

8.2 显式类型转换

赋值过程中，大类型数据赋值给小类型变量，编译报错，此时必须通过强制类型转换实现。

固定格式：

```
数据类型 变量名 = (目标数据类型)(数值或变量或表达式);
```

案例展示：

```java
package com.briup.chap02;

public class Test082_ExplicitTrans {
    public static void main(String[] args) {
        // 1.数据赋值 强制类型转换
        float f1 = (float)3.14;
        System.out.println(f1);     //3.14

        // 1.数据赋值 强制类型转换
        int size = (int)123L;
        System.out.println(size);   //123

        double len = 178.5;
        // 2.变量赋值 强制类型转换
        int length = (int)len;
        System.out.println(length); //178

        byte b = 10;
        short s = 5;
        double d = 2.3;
        // 3.表达式赋值 强制类型转换
        float f2 = (float)((b - 5) * s + d);
```

```
        System.out.println(f2);     //27.3
    }
}
```

8.3 特殊情况

观察代码，思考为什么那些代码编译成功了

```
package com.briup.chap02;

public class Test083_SpecialTrans {
    public static void main(String[] args) {
        // 为什么 int -> byte 成功了？
        byte b = 10;
        System.out.println(b);

        // 为什么 int -> short 成功了？
        short s = 20;
        System.out.println(s);

        // 为什么 int -> char 成功了？
        char c = 97;
        System.out.println(c);

        // 为什么 b2赋值成功，b3却失败了？
        byte b2 = 127;
        System.out.println(b2);
        //byte b3 = 128;
        //System.out.println(b3);

        // 为什么 s2赋值成功，s3却失败了？
        short s2 = -32768;
        System.out.println(s2);
        //short s3 = -32769;
```

```
27          //System.out.println(s3);
28
29          // 为什么 c2赋值成功，c3却失败了？
30          char c2 = 65535;
31          System.out.println("c2: " + c2);
32          //char c3 = 65536;
33          //System.out.println(c3);
34      }
35  }
```

结论：整形常量优化机制

使用**整形字面值常量**赋值时，系统会先判断该字面值**是否超出赋值类型的取值范围**，如果没有超过范围则赋值成功，如果超出则赋值失败。

```
1
2   public static void main(String[] args) {
3       // 为什么 int -> byte 成功了？
4       // 字面值常量10，在赋值给b的时候，常量优化机制会起作用：
5       //  系统会先判断10是否超出byte的取值范围，如果没有，则赋值成功，如果超出(比如128)，则赋值失败
6       byte b = 10;
7       System.out.println(b);
8
9       // 为什么 int -> short 成功了？
10      //  系统会先判断20是否超出short的取值范围，结果是没有，则赋值成功
11      short s = 20;
12      System.out.println(s);
13
14      //  如果使用-32769赋值，超出short类型取值范围，则赋值失败
15      //short s3 = -32769;
16      //System.out.println(s3);
17  }
```

注意事项:

- **常量优化机制只适用于常量**，如果是变量，则不可以
- **该优化机制只针对int类型常量**，对于long、float、double等常量，则不可以

案例展示:

```
public static void main(String[] args) {
    // 下面3个赋值语句都 编译失败
    byte b = 10L;
    short s = 3.14F;
    int n = 2.3;

    // 下面2个赋值语句 ok
    byte b1 = 2;
    short s2 = 3;

    // 变量赋值或表达式赋值，都编译失败
    byte b3 = s2;
    byte b4 = b1 + s2;
}
```

8.4 面试题

面试题1:

指出下面程序有问题的地方

- **byte b1 = 3;**
- **byte b2 = 4;**
- **byte b3 = 3 + 4;**
- **byte b4 = b1 + b2;**
- **byte b5 = b1 + 4;**

面试题2：

下面程序运行输出结果是多少？

```
public static void main(String[] args) {
    byte b1 = 126;
    byte b2 = (byte)(b1 + 3);

    //输出 -127，而非 129，建议用二进制补码运算验证结果
    System.out.println(b2);
}
```

注意事项：强制类型转换时，得到的结果值可能会与期望值不同，大家一定要注意