基本数据类型

基本类型及其大小

Java共有8种基本数据类型，它们分别如下表所示：

基本数据类型	类型	占用空间大小	取值范围	默认值
byte	数值型	8位(1字节)	有符号,最小值是 **-128（-2^7）** 最大值是 **127（2^7-1）**	0
short	数值型	16位(2字节)	有符号,最小值是 **-32768（-2^15）**最大值是 **32767（2^15 - 1）**	0
int	数值型	32位(4字节)	有符号,最小值是 **-2^31**, 最大值是 **2^31 - 1**	0
long	数值型	64位(8字节)	有符号,最小值是 **-2^63**, 最大值是 **2^63 - 1**	0L
float(单精度)	数值型	32位(4字节)	有符号,最小值为**-2^128**, 最大值为**2^128**	0.0f
double(双精度)	数值型	64位(8字节)		0.0d
char	字符型	16位(2字节)	单一的 Unicode 字符, 最小值是 **\u0000**（即为0）； 最大值是 **\uffff**（即为65,535）；	\u0000
boolean	布尔型		true和false两个字面量值	false

其中boolean类型只能表示1位的信息量，根据官方文档的描述,它的**大小并没有精确地定义。**(根据虚拟机的规范来决定)

在switch-case 语句中的变量类型可以是： byte、short、int 或者 char。从 Java SE 7 开始，switch 支持字符串 String 类型了，同时 case 标签必须为**字符串常量或字面量**。枚举类enum也可以适配switch-case

基本类型的运算和相互转换

boolean类型数据可以进行逻辑运算（&&、||、！），其他的基本类型都可以进行数值计算（+、-、*、/、%等）。逻辑运算比较简单易懂，完全与逻辑数学的规则一致。

数值运算涉及到运算后的结果的类型问题，稍微比较复杂一点。一般来说，**数值运算最终结果的类型与表达式中的最大（占用空间最大）的类型一致。**

```
long l = 1 + 2L; // 与2L的类型一致
int i = 1 + 2L; // 编译不通过
float f = 1 + 2 + 1.2f; // 与1.2f的类型一致
double d = 1 + 2 + 1.2; // 与1.2的类型一致
```

如果两种相同的类型的数据进行运算，按理来说，运算结果应该还是那个类型。但事实上，`byte`、`char`、`short`等类型是不满足这个结论的。

```
// 编译不通过，编辑器报：Type mismatch: cannot convert from int to byte 。
byte s1 = 1;
byte s2 = 1;
byte s = s1 + s2;

// 编译不通过，编辑器报：Type mismatch: cannot convert from int to char 。
char s1 = 1;
char s2 = 1;
char s = s1 + s2;

// 编译不通过，编辑器报：Type mismatch: cannot convert from int to short 。
short s1 = 1;
short s2 = 1;
short s = s1 + s2;
```

从字面上来看，1+1=2绝对没有超过这个类型的范围。**Java中的数值运算最低要求是int类型**，如果参与运算的变量类型都没有超过int类型，则它们都**会被自动升级为int类型再进行运算**，所以它们运算后的结果类型也是int类型(但是short类型不能自动转换成int类型)而编译器会直接将`byte s1 = 1 + 1`编译成`byte s1 = 2`，这个表达式在编译器就可以确定是合法表达式，故可以通过编译。可以通过字节码来进行佐证。

类型转换

Java中除了boolean类型之外，其他7中类型相互之间可以进行转换。转换分为自动转换和强制转换。对于自动转换（隐式），无需任何操作，而强制类型转换需要显式转换，即使用转换操作符（type）。7种类型按照其占用空间大小进行排序：

`byte < (short=char) < int < long < float < double`

类型转换的总则是：**小可直接转大、大转小会失去精度**。小转大是Java帮我们自动进行转换的，与正常的赋值操作完全一样；大转小需要进行强制转换操作，其语法是`target-type var = (target-type) value`。

为什么占用空间大的long可以转成占用空间小的float

因为long类型在计算机中存储的形式就是补码, 但是float类型的32位中,最高位是符号位(S),**接下来的8位是指数域(阶码E),代表10的多少次方,**最后的23位，是小数域（尾数M）对于一个二进制数，我们的表示应该是1.XXX* 2^X，由于第一位永远都是1所以直接省去，因此表示为S.M*2^E。**所以float的取值范围是-2^128到2^128**，远远大于long的最大值。所以java中long类型可以自动转换为float类型。

包装类

Java中每一种基本类型都会对应一个唯一的包装类，基本类型与其包装类都可以通过包装类中的静态或者成员方法进行转换。每种基本类型及其包装类的对应关系如下，值得注意的是，所有的包装类都是`final`修饰的，也就是它们都是**无法被继承和重写**的。

基本数据类型	包装类
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

包装类与基本类型的转换

从源代码的角度来看，基础类型和包装类型都可以通过赋值语法赋值给对立的变量类型

```
Integer a = 1;
int a = new Integer(1);
```

这种语法是可以通过编译的。但是，Java作为一种强类型的语言，对象直接赋值给引用类型变量，而基础数据只能赋值给基本类型变量，这个是毫无异议的。那么基本类型和包装类型为什么可以直接相互赋值呢？这其实是Java中的一种“语法糖”。通常来说使用语法糖能够增加程序的可读性，从而减少程序代码出错的机会。换句话说，这其实是一种障眼法，那么实际上是怎么样的呢？下面是`Integer a = 1;`语句编译的字节码。

```
0: iconst_1
1: invokestatic  #2                  // Method java/lang/Integer.valueOf:
(I)Ljava/lang/Integer;
4: astore_1
```

首先，生成一个常量1，然后调用`Integer.valueOf(int)`方法返回`Integer`对象，最后将对象的地址（引用）赋值给变量a。`Integer a = 1;`其实相当于`Integer a = Integer.valueOf(1);`。

其他的包装类都是类似的，下表是所有包装类中的类型转换方法。

包装类	包装类转基本类型
Byte	Byte.valueOf(byte)
Short	Short.valueOf(short)
Integer	Integer.valueOf(int)
Long	Long.valueOf(long)
Float	Float.valueOf(float)
Double	Double.valueOf(double)
Character	Character.valueOf(char)
boolean	Boolean.valueOf(booleann)

两个包装类引用相等性

在Java中，“==”符号判断的**内存地址所对应的值得相等性**，具体来说，基本类型判断值是否相等，引用类型判断其指向的地址是否相等。看看下面的代码，两种类似的代码逻辑，但是得到截然不用的结果。

```
Integer a1 = 1;
Integer a2 = 1;
System.out.println(a1 == a2); // true

Integer b1 = 222;
Integer b2 = 222;
System.out.println(b1 == b2); // false
```

这个必须从源代码中才能找到答案。`Integer`类中的`valueOf()`方法的源代码如下：

```
public static Integer valueOf(int i) {
    if (i >= IntegerCache.low && i <= IntegerCache.high) // 判断实参是否在可缓存范围内，默认为[-128, 127]
        return IntegerCache.cache[i + (-IntegerCache.low)]; // 如果在，则取出初始化的Integer对象
    return new Integer(i); // 如果不在，则创建一个新的Integer对象
}
```

由于1属于[-128, 127]集合范围内，所以valueOf()每次都会取出同一个Integer对象，故第一个判断结果为true；而222不属于[-128, 127]集合范围内，所以valueOf()每次都会创建一个新的Integer对象，由于两个新创建的对象的地址不一样，故第二个判断结果为false。