# 数据类型概念

## Java中数据类型基本概念

1. 数据类型在计算机语言里面，是**对内存位置的一个抽象表达方式**，可以理解为针对内存的一种抽象的表达方式。
2. **Java是强类型语言**，强类型注重变量的数据类型不能混用，弱类型中变量可以混用几种数据类型。Java中将数据类型分为两种：**基本数据类型和引用数据类型**。
3. 基本类型：简单数据类型是不能简化的、内置的数据类型。由编程语言本身定义，**表示了真实的数字、字符和整数。**
4. 引用数据类型：Java 语言本身不支持 C++ 中的结构（struct）或联合（union）数据类型。**Java的复合数据类型一般都是通过类或接口进行构造**，类提供了捆绑数据和方法的方式，同时可以针对程序外部进行信息隐藏。比如String类，BigDecimal类。

## 八大基本数据类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 名称 | 内存大小 | 取值范围 | 作用 |
| 整型 | byte | 8位（1字节） | (-128,127) - 2^7-2^7-1 | 最小的数据类型 |
| short | 16位（2字节） | (-32768,32767) - 2^15-2^15-1 | 短整型 |
| int | 32位（4字节） | (-2147483648~2147483647)  - 2^31-2^31-1 | 整型 |
| long | 64位（8字节） | - 2^63-2^63-1 | 长整型 |
| 浮点型 | float | 32位（4字节） | 有效小数点只有6~7位 | 浮点型 |
| double | 64位（8字节） | 有效小数点有15~16位 | 双精度浮点型 |
| 字符型 | char | 16位（2字节） | 0~65535， | 存储单个字符 |
| 布尔型 | boolean | 8位（1字节） | 两个值，true或false | 判断真或假 |

1. 整型byte,short,int,long默认值都是0。浮点型float,double默认值都是0.0。字符型char默认值是 \。布尔型boolean默认值是false。
2. **类型转换都是小范围向大范围转换，大范围往小范围转化需要用到强制转换；**

例如：(1)int a=12;double b=a;(2)double a=3.0;int b=(int)a;

1. int 跟 char 之间的转换是通过 ASCII 转化的；

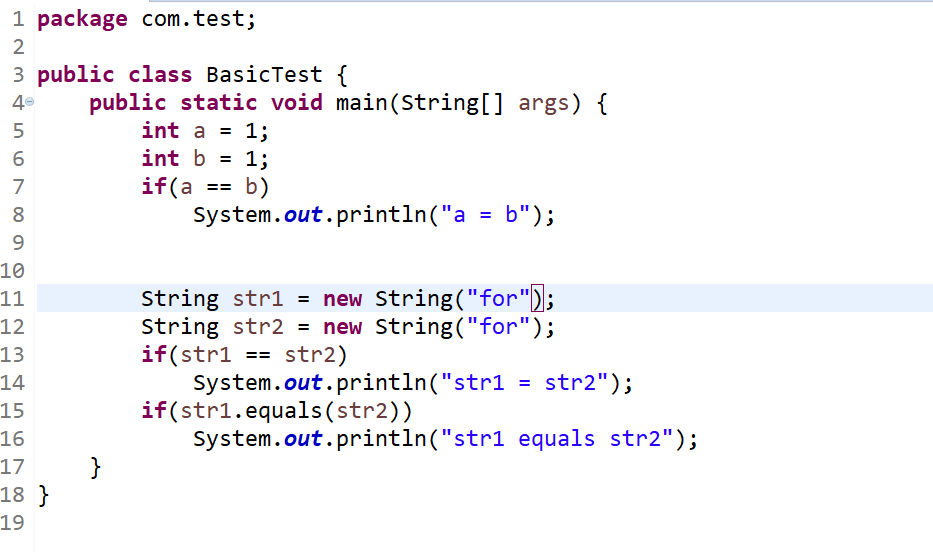
例如：char a ='A';int b=a;System.out.prientln(b); 最后输出的结果为 65；

## 数据类型与内存关系

在 Java 中，每个存放数据的变量都是有类型的。不同类型的变量在内存中分配的字节数不同，同时存储方式也是不同的。**确定了变量的类型，即确定了数据需分配内存空间的大小，数据在内存的存储方式。**

* 1. 存储原理

1. 基本数据类型的存储原理：所有的简单数据类型不存在 “引用” 的概念，**基本数据类型都是直接存储在内存中的内存栈上的**，数据本身的值就是存储在栈空间里面，而 Java 语言里面八种数据类型是这种存储模型；
2. 引用类型的存储原理: 引用类型继承于 Object 类（也是引用类型）都是按照 Java 里面存储对象的内存模型来进行数据存储的，使用 Java 内存堆和内存栈来进行这种类型的数据存储，简单地讲，**“引用” 是存储在有序的内存栈上的，而对象本身的值存储在内存堆上的。**
3. 区别：基本数据类型是分配在栈上的，引用类型指针是分配在栈上，对象本身在堆上。
   1. “==”和equals()区别
4. “==”比较的是在栈上的区别，基本类型比较的本身值，引用类型比较的是指针的值。



## 基本类型存储原理

1. 整型（byte、short、int、long）

**整型用二进制表示时最高位为符号位 0 代表正数 1 代表负数。**

1. 浮点型（float、double）

double 型比 float 型存储范围更大，精度更高，所以通常的浮点型的数据在不声明的情况下都是 double 型的。

如果要表示一个数据是 float 型的，可以在数据后面加上 “F”。 浮点型的数据是不能完全精确的，所以有的时候在计算的时候可能会在小数点最后几位出现浮动，这是正常的。

1. 字符型（char）、

Java 的文本编码采用 Unicode 集，Java **字符 16 位无符号型数据**，一个字符变量在内存中占 2 个字节。

1. 布尔型（boolean）

Boolean没法转换为其他类型，boolean型就只能是boolean型。

* 1. 原码、反码、补码

1. 整数使用补码表示，最高位表示符号位（0代表正，1代表负）

**正数原码反码补码都一样。在计算机中，正反都使用补码存储。**

原码：符号位加上数字的二进制表示（假设为byte字节）

+7的原码为： 00000111

-7的原码为： 10000111

反码：负数符号位为1，其他位按位取反后得到反码。

+7的反码为： 00000111

-7的反码为： 11111000

补码：负数符号位为1，将反码最末位加1得到补码

+7的补码为： 00000111

-7的补码为： 11111001

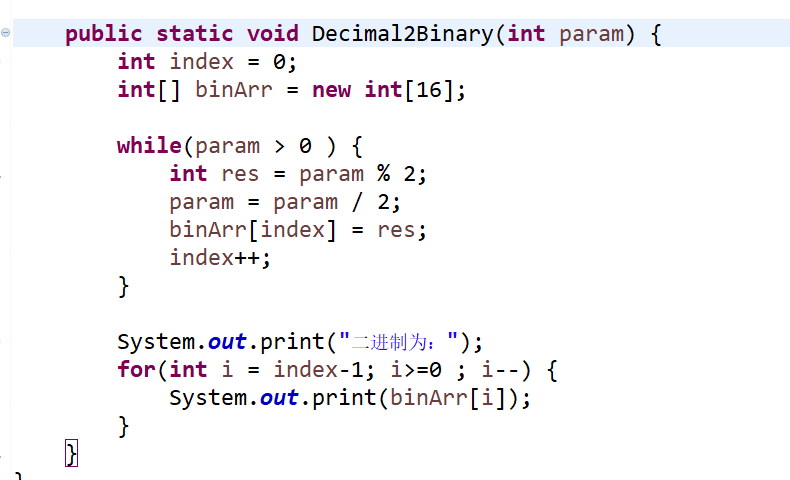
1. 负数原码和补码之间的转换

原码除了最高位按位取反后得到反码，反码最末位加1得到补码。

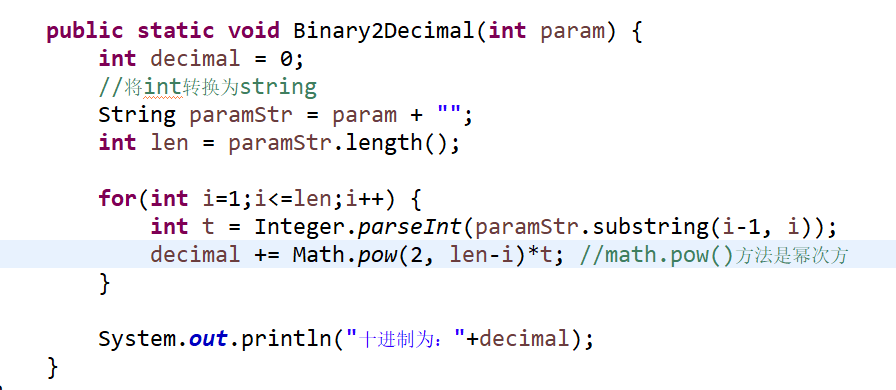
补码除了最高位安位取反后，再将最末位加1得到原码。

* 1. 二进制以及其他进制间的转换

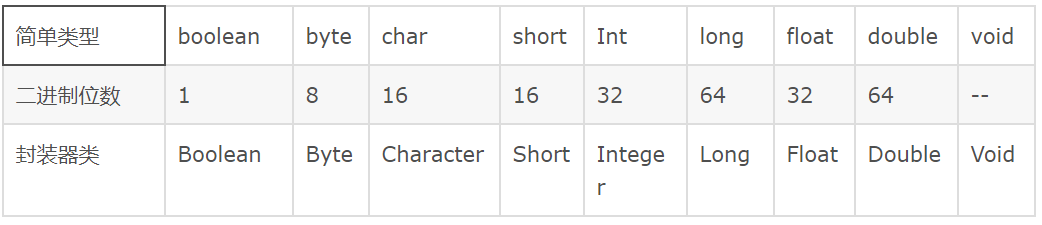
1. 二进制转换成十进制



1. 十进制转换为二进制



## 基本类型及其包装类



* 1. 包装类常量

1. 取值范围（Byte）

二进制位数：Byte.SIZE

最小值： Byte.MIN\_VALUE

最大值： Byte.MAX\_VALUE

1. 基本类型存储在栈中，包装类对象数据存储在堆中。
   1. 包装类和基本类型之间的转换
2. 同类型之间的转换（举例：float）

int i = 100; Integer I = new Integer(I);

public int intValue()可以将Integer转换为int; i = I.intValue()

1. 不同类型之间的转换（举例：int 和 float，double之间的转换）

int i = 100; Integer I = new Integer(I);

方法 xxValue()可以将Integer类型转换为其他的类型。

public double doubleValue() //返回一个double类型

public float floatValue() //返回一个float类型

public short shortValue() //返回一个short类型

**其他类型的包装类也有同样的方法。**

* 1. String和其他类型之间的转换

1. 其他类型X 转换成 String类

调用包装类的方法 X .toString()

**对所有包装类型进行操作（比如 toString()）要进行空指针判断，我们并不能保证这个值是有的。**

自动转换：其他类型 X + ””

调用String类的方法 String.valueOf(X)

1. String类 转换成 其他类型

先转换成相应的封装器实例, 再调用对应的方法转换成其它类型

构造方法(s)新建对应的包装类实例，再调用xxValue()方法

静态方法: parseXX()

String s = "1";

byte b = Byte.parseByte(s);

short t = Short.parseShort(s);

int i = Integer.parseInt(s);

long l = Long.parseLong(s);

float f = Float.parseFloat(s);

double d = Double.parseDouble(s);