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。接触每种语言的时候，都会存在数据类型的认识，有复 杂的、简单的。各种数据类型都需要在学习初期去了解，Java是强类型语言，所以Java对于数据类型的规范相对严格。数据类型是语言的抽象原子概念，可 以说是语言中最基本的单元定义，在Java里面，本质上数据类型分为两种：简单类型和复杂类型。

　　简单类型：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，它表示了真实的数字、字符和整数。

　　复杂类型：Java语言本身不支持C++中的结构（struct）或联合（union）数据类型，它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。

**i.Java中的基本类型：**

　　1)概念：

　　Java中的简单类型从概念上分为四种：实数、整数、字符、布尔值。但是有一点需要说明的是，Java里面有八种原始类型，其列表如下：

　　实数：double、float

　　整数：byte、short、int、long

　　字符：char

　　布尔值：boolean

　　复杂类型和基本类型的内存模型本质上不一样，简单数据类型的存储原理：所有的简单数据类型不存在“引用”概 念，简单数据类型直接存储在内存中的内存栈上，数据本身的值存储在栈空间里面，而Java语言里面只有这八种数据类型是这种存储模型；而其他的只要是继承 于Object类的复杂数据类型都是按照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。

　　2)原始类型特征：

　　Java的简单数据讲解列表如下：

　　int：int为整数类型，在存储的时候，用4个字节存储，范围为-2,147,483,648到2,147,483,647，在变量初始化的时候，int类型的默认值为0。

　　short：short也属于整数类型，在存储的时候，用2个字节存储，范围为-32,768到32,767，在变量初始化的时候，short类型的默认值为0，一般情况下，因为Java本身转型的原因，可以直接写为0。

　　long：long也属于整数类型，在存储的时候，用8个字节存储，范围为-9,223,372,036,854,775,808到9,223,372,036, 854,775,807，在变量初始化的时候，long类型的默认值为0L或0l，也可直接写为0。

　　byte：byte同样属于整数类型，在存储的时候，用1个字节来存储，范围为-128到127，在变量初始化的时候，byte类型的默认值也为0。

　　float：float属于实数类型，在存储的时候，用4个字节来存储，范围为32位IEEEE 754单精度范围，在变量初始化的时候，float的默认值为0.0f或0.0F，在初始化的时候可以写0.0。

　　double：double同样属于实数类型，在存储的时候，用8个字节来存储，范围为64位IEEE 754双精度范围，在变量初始化的时候，double的默认值为0.0。

　　char：char属于字符类型，在存储的时候用2个字节来存储，因为Java本身的字符集不是用ASCII码来进行存储，是使用的16位Unicode字符集，它的字符范围即是Unicode的字符范围，在变量初始化的时候，char类型的默认值为'u0000'。

　　boolean：boolean属于布尔类型，在存储的时候不使用字节，仅仅使用1位来存储，范围仅仅为0和1，其字面量为true和false，而boolean变量在初始化的时候变量的默认值为false。

　　——[$]提供一个字面量赋值的例子——

package org.susan.java.basic;

public class AssignTester {

public static void main(String args[]){

int x,y;//定义x，y变量

float f = 12.34f; //定义float类型的变量并赋值

double w = 1.234;//定义double类型变量并且赋值

boolean flag = true ; //指定变量flag为boolean型，且赋初值为true

char c ; //定义字符型变量c

String str ; //定义字符串变量str

String str1 = " Hi " ; //指定变量str1为String型，且赋初值为Hi

c = 'A' ; //给字符型变量c赋值'A'

str = " bye " ; //给字符串变量str赋值"bye"

x = 12 ; //给整型变量x赋值为12

y = 300; //给整型变量y赋值为300

}

}

　　3)自动拆箱（AutoBox）：

Java里面，每一种原始类型都对应着相应的包装类型，在JDK1.5之前（不包含JDK1.5），当包装类和原始类型进行相互转换的时候，需要调用包装类型的方法进行转换，不能通过操作符进行直接的计算。下边是一个原始类型和包装类型的一个对应表：



简单看看下边这段代码：

package org.susan.java.basic;

public class AutoBoxTester {

public static void main(String args[]){

Integer integer = new Integer(12);

int integer2 = 33;

System.out.println(integer + integer2);

}

}

　　这段代码在JDK 1.5版本以上可以通过编译，而且不会报错，运行结果如下输出：

45

　 　但是如果这段代码在JDK 1.4上边编译就会有问题了，因为在JDK 1.4的规范里面Integer属于一个包装类型，而int是原始类型，如果一个包装类型和原始类型要进行想对应的运算的时候，需要进行转换操作，直接将 Integer类型转换称为原始类型操作，否则二者是不允许相加的，可以试试将上边代码用1.4版本进行编译：

javac -source 1.4 AutoBoxTester.java

　　就会收到下边的异常：

AutoBoxTester.java:5: operator + cannot be applied to java.lang.Integer,int

System.out.println(integer + integer2);

　　为什么呢？其实编译器给的信息很明显，使用JDK 1.5进行编译可以直接通过而且不会报错，是因为JDK 1.5提供了自动拆箱和自动装箱的功能，而JDK 1.4里面如果要使得上边的代码段可以编译通过，必须做一个简单的修改：

public class AutoBoxTester {

public static void main(String args[]){

Integer integer = new Integer(12);

int integer2 = 33;

System.out.println(integer.intValue() + integer2);

}

}

　　改成上边代码段了过后，在JDK 1.4平台下就可以得到输出：

45

　　从上边的例子可以看出，在JDK 1.5之前，如果要针对包装类进行数值计算，必须要将包装类直接转化称为原始类型，否则操作符本身是不会支持包装类的操作的，但是在JDK 1.5以及以后就没有这个限制了。

　　【简 单总结：自动拆箱的意思就是不需要经过用户手工编程，编译器会直接识别包装类和原始类型相互之间的转换以及运算，并且把包装类型拆成原始类型进行代码里面 规定的数值运算或者其他操作，这功能JDK的最低支持版本是1.5。其实对Java语言本身而言，Integer这种封装类实际上就是Java里面继承于 Object的类的对象实例，只是在1.4之前，必须调用方法xxxValue()来完成手工拆箱的操作，只是这个在JDK 1.5不会有此限制。】

4)类型转换：

　　Java里面的类型转换包括两种：自动转换（隐式转换）；强制转换（显示转换）

　　[1]自动转换：

　　条件：A.这两种类型是兼容的；B.目的类型数的范围（位数）比来源类型的大

　　当以上2个条件都满足的时候，拓宽转换（widening conversion）就会自动发生，例如，int类型范围比所有byte类型的合法范围大，因此不要求显示的强制转换语句。对于拓宽转换，兼容程度可以看下边的继承树：

　　java.lang.Object

　　|—java.lang.Boolean

　　|—java.lang.Character

　　|—java.lang.Number

　　　　|—java.lang.Byte

　　　　|—java.lang.Float

　　　　|—java.lang.Integer

　　　　|—java.lang.Long

　　　　|—java.lang.Short

　　　　|—java.lang.Double

　　从上边的继承树可以可以看到，Boolean类型、Character类型、Number类型是两两不兼容的，所以在隐式转换的过程，不兼容的类型是不能进行自动类型转换的

　　[2]强制转换：

　 　尽管自动类型转换是很有帮助的，但并不能满足所有的编程需要。如果2种类型是兼容的，那么Java 将自动地进行转换。例如，把int 类型的值赋给long 类型的变量，总是可行的。然而，不是所有的类型都是兼容的，因此，不是所有的类型转换都是可以隐式实现的。例如，没有将double 型转换为byte 型的定义。幸好，获得不兼容的类型之间的转换仍然是可能的。要达到这个目的，你必须使用一个强制类型转换，它能完成两个不兼容的类型之间的显式变换。它的 通用格式如下：

(target-type)value

　　[3]关于类型的自动提升，遵循下边的规则：

　　所有的byte、short、char类型的值将提升为int类型；

　　[4]转换附加：

　　当两个类型进行自动转换的时候，需要满足条件：【1】这两种类型是兼容的，【2】目的类型的数值范围应该比源转换值的范围要大。（上边已经讲过了）而拓展范围就遵循上边的自动类型转换树，当这两个条件都满足的时候，拓展转换才会发生，而对于几个原始类型转换过程，根据兼容性boolean和char应该是独立的，而其他六种类型是可以兼容的，在强制转换过程，唯独可能特殊的是char和int是可以转换的，不过会使用char的ASCII码值比如：

int a = (int)'a';

　　a的值在转换过后输出的话，值为97；

　　[5]基础类型的几个常见的代码例子：

　　——[$]附加转换和精度丢失——

package org.susan.java.basic;

public class CastingNumbers {

public static void main(String args[]){

double above = 1.7;

double below = 0.4;

System.out.println("above:" + above);

System.out.println("below:" + below);

System.out.println("(int)above:" + (int)above);

System.out.println("(int)below:" + (int)below);

System.out.println("(char)('a' + above):" + (char)('a' + above));

System.out.println("(char)('a' + below):" + (char)('a' + below));

}

}

　　可以分析上边这段代码的输出：

above:1.7

below:0.4

(int)above:1

(int)below:0

(char)('a' + above):b

(char)('a' + below):a

　 　从上边的输出结果可以知道，在double到int的强制转换中，会发现精度丢失的情况，而且这种精度丢失是不考虑四舍五入的，是直接将浮点部分舍弃 掉，从上边的输出就可以知道了；而且当int以及double转化称为char的时候，会使用上边提及的附加转换的结果，当'a'加了1过后进行char 转换会生成b。

　　如果有一个操作数是long类型，计算结果是long类型；

　　如果有一个操作数是float类型，计算结果是float类型；

　　如果有一个操作数是double类型，计算结果是double类型；

　　自动类型转换图如下：

　　byte->short(char)->int->long->float->double

　　如果是强制转换的时候，就将上边的图反过来