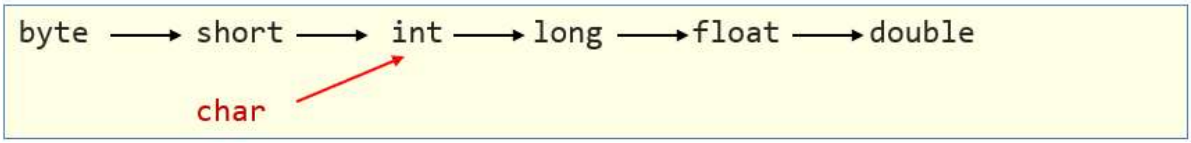
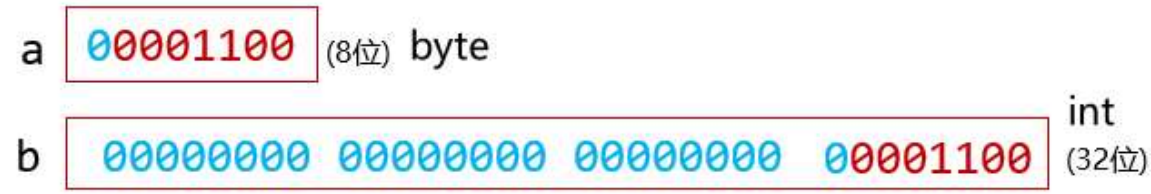
一、类型转换

1.0 自动类型转换

类型范围小的变量，可以直接赋值给类型范围大的变量

自动类型转换的其他形式





2.0 自动类型转换运算

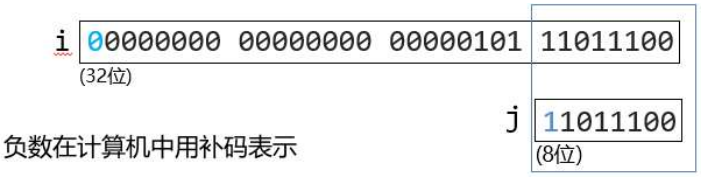
1. 在表达式中，小范围类型的变量会自动转换成当前较大范围的类型再运算。
2. 表达式的最终结果类型由表达式中的最高类型决定。
3. 在表达式中，byte、short、char 是直接转换成int类型参与运算的
4. **字面量**是整数且没有后缀L、l的当成是int来运算
5. **字面量**是小数且没有F、f作为后缀的当成是double运算


```
byte a = 100;
byte b = 90;
//byte short char 其中一个或多个在进行运算时自动转成int，再运算
int i = a + b;
System.out.println(i);
```

3.0 强制类型转换

可以 强行将类型范围大的变量、数据赋值给类型范围小的变量

```
int i = 1500;
byte j =  (byte)i;
System.out.println(j); // -36
```



强制类型转换可能造成数据(丢失)溢出；
浮点型强转成整型，直接丢掉小数部分，保留整数部分返回

二、toString、valueOf、(String)的区别

- 代码体现

```
package api_object;

import java.util.Calendar;

/**
 * ClassName: StringDemo1 <br/>
 * Description: <br/>
 * date: 2022/3/21 16:40<br/>
 *
 * @author fgcy<br />
 * @since JDK 1.8
 */
public class StringDemo1 {
    public static void main(String[] args) {
        int a1 = 10;
        Integer a2 = 10;
        byte b1 = 12;
        Byte b2 = 12;
        short ss1 = 2;
        Short ss2 = 2;
        Long l1 = 888L;
        long l2 = 888L;
        char c1 = 'a';
        Character c2 = 'a';
        Float f1 = 1.29f;
        float f2 = 1.29f;
        double d1 = 9.9;
        Double d2 = 9.8;

        //编译报错，基本数据类型及其包装类不能强转为String类
        //同样的的String类也不能强转为基本数据类型及其包装类
//        String s1 = (String) a1;
//        String s1 = (String) a2;
//        String s1 = (String) b1;
//        String s1 = (String) b2;
//        String s1 = (String) ss1;
//        String s1 = (String) ss2;
//        String s1 = (String) l1;
//        String s1 = (String) l2;
//        String s1 = (String) c1;
//        String s1 = (String) c2;
//        String s1 = (String) f1;
//        String s1 = (String)f2;
//        String s1 = (String) d1;
//        String s1 = (String) d2;

        //基本数据类型没有toString方法
//      a1.toString();
        System.out.println(a2.toString());//10

        //null.toString()会报错
        a2 = null;
        //println()底层使用了String.valueOf()方法，源码如下：
            //return (obj == null) ? "null" : obj.toString();
        System.out.println(a2);//null
```

```
//      System.out.println(a2.toString());

        //基本数据类型及其包装类转String类型推荐做法是String.valueOf()
        //如果是null会返回字符串null
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(a1));//10
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(a2));//null
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(b1));//12
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(b2));//12
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(c2));//a
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(c1));//a
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(ss1));//2
        System.out.println("通过String.valueOf(x)，使得基本数据类型及其包装类变为
String: " + String.valueOf(ss2));//2

        //通过拼接字符串的方法，将基本数据类型及其包装类变为String
        //基本数据类型转String除了拼接空字符就是String.valueOf(x)
        System.out.println("通过拼接空字符串将基本数据类型及其包装类变为String: " + b1 +
"");//12
        System.out.println("通过拼接空字符串将基本数据类型及其包装类变为String: " + b2 +
"");//12
        System.out.println("通过拼接空字符串将基本数据类型及其包装类变为String: " + ss2
+ "");//2
        System.out.println("通过拼接空字符串将基本数据类型及其包装类变为String: " + ss1
+ "");//2


        //包装类型与String 无论谁强转转谁 都报错，编译就出错
//        Integer integer = (Integer) "aaa";


        //用Object类型的变量(接基本数据类型的字面值)，强制转换为String(向下转型)，编译不报错
        Object o = 12;
        //用Object类型的变量接包装类(向上转型)，强制转换为String(向下转型)，编译不报错
        Object object = a2;
        //运行的时候就报错，类型转换异常
        //java.lang.ClassCastException: java.lang.Integer cannot be cast to
java.lang.String
//      String rs = (String) o;
        Object obj = "a";
        Integer r = obj instanceof Integer ? (Integer) obj : -1;
        System.out.println("String类与包装类强制类型转换: " + r);//-1
    }
}
================================================
10
null
通过String.valueOf(x)，使得基本数据类型及其包装类变为String: 10
通过String.valueOf(x)，使得基本数据类型及其包装类变为String: null
通过String.valueOf(x)，使得基本数据类型及其包装类变为String: 12
通过String.valueOf(x)，使得基本数据类型及其包装类变为String: 12
通过String.valueOf(x)，使得基本数据类型及其包装类变为String: a
```

```
通过String.valueOf(x)，使得基本数据类型及其包装类变为String：a
通过String.valueOf(x)，使得基本数据类型及其包装类变为String：2
通过String.valueOf(x)，使得基本数据类型及其包装类变为String：2
通过拼接空字符串将基本数据类型及其包装类变为String：12
通过拼接空字符串将基本数据类型及其包装类变为String：12
通过拼接空字符串将基本数据类型及其包装类变为String：2
通过拼接空字符串将基本数据类型及其包装类变为String：2
String类与包装类强制类型转换：-1
```

- 总结

```
1、toString()，可能会抛空指针异常
    在这种使用方法中，因为java.lang.Object类里已有public方法toString()，所以java对象都可以调用此方法。但在使用时要注意，必须保证object不是null值，    否则将抛出NullPointerException异常。采用这种方法时，通常派生类会覆盖Object里的toString()方法。

2、String.valueOf()，推荐使用，因为当对象为null时，返回字符串“null”
    String.valueOf()方法是推荐使用的，因为它不会出现空指针异常，而且是静态的方法，直接通过String调用即可，只是有一点需要注意，就是上面提到的，如果为    对象为null，String.valueOf()返回结果是字符串“null”。而不是null。

3、(String)强转，不推荐使用
    （String）是标准的类型转换，将Object类型转为String类型，使用(String)强转时，最好使用instanceof做一个类型检查，以判断是否可以进行强转，否则容易抛出    ClassCastException异常。需要注意的是编写的时候，编译器并不会提示有语法错误，所以这个方法要谨慎的使用

4.基本数据类型及其包装类与String 无论谁【强转】转谁 都报错，编译就出错
5.基本数据类型没有toString方法，对象独有
6.println()底层使用了String.valueOf()方法，源码如下：return (obj == null) ? "null" : obj.toString();
```

三、常见运算符

1.0 算数运算符

```
*  /  %
+  -
如果两个整数做除法，其结果一定是整数，因为最高类型是整数。10/3=3  10*1.0/3=3.333335
```

- 数值拆分

```
    @Test
    public void testBasic() {
        int score = 123;
        //取模运算符可以得到最尾端的数字
        System.out.println("个位：" + score % 10 + " 十位:" + score / 10 % 10 + "百位" + score / 100);
    }
```

2.0 + 做连接符

```
能算则算，不能算就在一起。（计算机很聪明）
int a = 5 ;
System.out.println(“abc” + ‘a’); // abca
System.out.println("abc" + a);  // abc5
System.out.println(5 + a);   // 10
System.out.println("abc" + 5 + ‘a’);//abc5a
System.out.println(15 + "abc" + 15);//15abc15
System.out.println(a + 'a’);//102
System.out.println(a + "" +'a’);//5a
System.out.println(a + 'a'+" itheima ");//102itheima
System.out.println("itheima"+ a + 'a’);//itheima5a
System.out.println("itheima"+ ( a + 'a' ));//itheima102
```

3.0 自增自减运算符

- 概念

```
1. ++ 、-- 只能操作变量，不能操作字面量的
2. 放在变量的前面，先对变量进行+1、-1，再拿变量的值进行运算。
3. 放在变量的后面，先拿变量的值进行运算，再对变量的值进行+1、-1
```

- 代码体现

```
 @Test
    public void testBasic1() {
        int c = 10;
        int d = 5;
        int rs3 = c++ + ++c - --d - ++d + 1 + c--;
        System.out.println(rs3);
    }
=======================================
26
```

4.0 赋值运算符

- 基本赋值运算符

```
=
```

扩展赋值运算符　　自动强制类型转换

符号	作用	说明
+=	加后赋值	a+=b 等价于 a = (a的数据类型)(a+b); 将a + b的值给a
-=	减后赋值	a-=b 等价于 a = (a的数据类型)(a-b); 将a - b的值给a
*=	乘后赋值	a*=b 等价于 a = (a的数据类型)(a*b); 将a * b的值给a
/=	除后赋值	a/=b 等价于 a = (a的数据类型)(a/b); 将a / b的商给a
%=	取余后赋值	a%=b 等价于 a = (a的数据类型)(a%b); 将a % b的商给a

注意：扩展的赋值运算符隐含了强制类型转换。

```
    @Test
    public void testBasic2() {
        int c = 10;
        int d = 5;
        c = c + d;
        c += d;//等价于 c = (int)(c + d)

        byte a = 30;
        byte b = 120;
        a += b;//等价于  a = (byte)(a + b)
        System.out.println(a);//精度丢失
    }
=========================================
15
-106
```

5.0 关系运算符

- 概念

是对数据进行条件判断的符号，最终会返回一个比较的布尔结果（false,true）

- 形式

```
== < <= >= !=
```

6.0逻辑运算符

符号	介绍	说明
&	逻辑与	必须都是true，结果才是true; 只要有一个是false，结果一定是false。
|	逻辑或	只要有一个为true、结果就是true
!	逻辑非	你真我假、你假我真。 !true=false 、 !false= true
^	逻辑异或	如果两个条件都是false或者都是true则结果是false。两个条件不同结果是true。

短路逻辑运算符

符号	介绍	说明
&&	短路与	判断结果与“&”一样。过程是**左边为 false，右边则不执行。**
||	短路或	判断结果与“|”一样。过程是**左边为 true， 右边则不执行。**

注意：逻辑与 “&” 、逻辑或“|”： 无论左边是 false还是 true，右边都要执行。

- 代码体现

```
    @Test
```

```
    public void testBasic3() {
        int c = 10;
        int d = 5;
        System.out.println("短路与: " + (c >= 99 && d++ < 100) + " d=" + d);
        System.out.println("逻辑与: " + (c >= 99 & d++ < 100) + " d=" + d);
        System.out.println("----------------------------------------");
        System.out.println("短路或: " + (c <= 99 || d++ < 100) + " d=" + d);
        System.out.println("逻辑或: " + (c >= 99 | d++ < 100) + " d=" + d);
    }
}
==================================================
短路与: false d=5
逻辑与: false d=6
----------------------------------------
短路或: true d=6
逻辑或: true d=7
```

- 小结

```
逻辑与：同真为真
逻辑异或：不同为真
逻辑或：同假为假
短路性能更好
```

7.0三元运算符（嵌套）

```
    @Test
    public void testBasic4() {
        int c = 120;
        int d = 5;
        int b = 78;
        System.out.println(c > d ? (c > b ? c : b ):( d > b ? d : b));
    }
==========================================
120
```

8.0运算符优先级

优先级	运算符
1	()
2	!、-、++、--
3	*、/、%
4	+、-
5	<<、>>、>>>
6	<、<=、>、>=、instanceof
7	==、!=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	=、+=、-=、*=、/=、%=、&=、

```
()
! - ++ --
* / %
+ -
<< >> >>>
< <= > >= instanceof
== !=
&
^
|
&&
||
?:
= += =+ *= /= %= &=
```

算数运算符 》 位移运算符 》 关系运算符 》 逻辑运算符 》 三元运算符 》 赋值运算符

算一官，罗三赋