第一章 数据类型转换

Java程序中要求参与运算的数据，必须要保证数据类型的一致性

1.1 自动转换

将取值范围小的类型**自动提升**为取值范围大的类型

```
public static void main(String[] args) {
    int i = 1;
    byte b = 2;
    // int类型和byte类型运算，结果为int类型
    int j = b + i;
    System.out.println(j);
}
```

转换规则

`byte、short、char`运算时直接提升为`int`

```
byte、short、char -> int -> long -> float -> double
```

1.2 强制转换

将取值范围大的类型**强制转换**成取值范围小的类型

转换格式

`数据类型 变量名 = (数据类型) 被转换数据值;`

例: `int i = (int)1.5;`

注意事项

- 浮点数转成整数，直接取消小数点，可能会造成数据损失精度
- `int`强制转换成`short`砍掉两个字节，可能会造成数据丢失

1.3 ASCII编码表

在计算机内部都是二进制的0、1数据，让计算机能直接识别人类文字，这就是ASCII编码表

- **编码表**:将人类的文字和一个十进制数进行对应起来组成的一张表格

字符	数值
0	48
9	57
A	65
Z	90
a	97
z	122

- 将所有的英文字母、数字、符号都和十进制进行了对应，因此产生了世界上第一张编码表ASCII(American Standard Code for Informatica Interchange 美国标准信息交换码)

第二章 运算符

2.1 算数运算符

算数运算符	含义
`+`	加法运算，字符串连接运算
`-`	减法运算
`*`	乘法运算
`/`	除法运算
`%`	取模运算，两个数字相除取余数
`++`、`--`	自增自减运算

Java中，整数使用以上运算符，无论怎么计算，也不会得到小数

```
public static void main(String[] args) {
    int i = 1234;
    System.out.println(i/1000*1000); //计算结果是1000
}
```

- `++`:变量自己增长1，反之，`--`运算，变量自己减少1，用法与`++`一致
  - 独立运算:变量在独立运算时，`变量++`和`++变量`没有区别
  - 混合运算
    - `变量++`:变量先自增，再运算
    - `++变量`:变量先运算，再自增

```
public static void main(String[] args) {
    int a = 1;
    int b = ++a;
    System.out.println(a); //计算结果是2
    System.out.println(b); //计算结果是2
}
```

```
public static void main(String[] args) {
    int a = 1;
    int b = a++;
    System.out.println(a); //计算结果是2
    System.out.println(b); //计算结果是1
}
```

- `+`符号在字符串中表示连接、拼接的含义

```
public static void main(String[] args) {
    System.out.println("5+5=", + 5+5); //输出5+5=55
}
```

2.2 赋值运算符

赋值运算符就是将符号右边的值，赋给左边的变量

赋值运算符	含义
`=`	等于号
`+=`	加等于
`-=`	减等于
`*=`	乘等于
`/=`	除等于
`%=`	取模等于

```
public static void main(String[] args) {
    int i = 5;
    i += 5; //计算方式:i=i+5,变量i先加5，再赋值给变量i
    System.out.println(i); //输出结果是10
}
```

2.3 比较运算符

比较运算符，是两个数据之间进行比较的运算，运算结果都是布尔值`true`或者`false`

比较运算符	含义
==	比较符号两边数据是否相等，相等结果是true
<	比较符号左边的数据是否小于右边的数据，如果是，结果为true
>	比较符号左边的数据是否大于右边的数据，如果是，结果为true
<=	比较符号左边的数据是否小于或者等于右边的数据，如果是，结果为true
>=	比较符号左边的数据是否大于或者等于右边的数据，如果是，结果为true
!=	不等于符号，如果符号两边的数据不相等，结果是true

```
public static void main(String[] args) {
    System.out.println(1==1); //true
    System.out.println(1<2); //true
    System.out.println(3>4); //false
    System.out.println(3<=4); //true
    System.out.println(3>=4); //false
    System.out.println(5!=6); //true
}
```

2.4 逻辑运算符

逻辑运算符，是用来连接两个布尔类型结果的运算符，运算结果都是布尔值`true`或者`false`

逻辑运算符	含义
&& 短路与	1.两边都是true，结果是true 2.一边是false，结果是false 短路特点:符号左边是false，右边不再运算
|| 短路或	1.两边都是false，结果是false 2.一边是true，结果是true 短路特点:符号左边是true，右边不再运算
! 取反	1.!true结果是false 2.!false结果是true

```
public static void main(String[] args) {
    System.out.println(true && true); //true
    System.out.println(true && false); //false
    System.out.println(false && true); //false，右边不计算

    System.out.println(false || false); //false
    System.out.println(false || true); //true
    System.out.println(true || false); //true，右边不计算

    System.out.println(!false); //true
}
```

2.5 三元运算符

- 三元运算符格式:`数据类型 变量名 = 布尔类型表达式?结果1:结果2`
- 三元运算符计算方式
  - 布尔类型表达式结果是true，三元运算符整体结果为结果1，赋值给变量
  - 布尔类型表达式结果是false，三元运算符整体结果为结果2，赋值给变量

```
public static void main(String[] args) {
    int i = (1==2 ? 100 : 200);
    System.out.println(i); //200
    int j = (3<4 ? 500 : 600);
    System.out.println(j); //500
}
```

第三章 方法入门

将一个功能抽取出来，把代码单独定义在一个大括号内，形成一个单独的功能。当我们需要的这个功能的时候，就可以去调用，这样即实现了代码的复用性，也解决了代码冗余的现象。

3.1 方法的定义

定义格式:

```
修饰符 返回值类型 方法名 (参数列表) {
    代码...
    return 返回值;
}
```

- 修饰符:目前固定写法`public static`
- 返回值类型:目前固定写法`void`，表示返回空值
- 方法名:为我们定义的方法起名，满足标识符的规范，用来调用方法
- 参数列表:目前无参数
- return:方法的结束。因为返回值类型是void，return则可以省略不写

```
public static void methodName() {
    System.out.println("这是一个方法");
}
```

3.2 方法的调用

方法在定义完毕后，方法不会自己运行，必须经过调用才能执行。我们可以在主方法main中来调用我们自己定义好的方法。在主方法中，只要写上要调用的方法名字就可以调用了。

```
public static void main(String[] args) {
    //调用定义的方法method
    method();
}
//定义方法，被main调用
public static void method() {
    System.out.println("自己定义的方法，需要被main调用运行");
}
```

3.3 调用练习

将三元运算符代码抽取到自定义的方法中，并调用

```
public class Demo {
    public static void main(String[] args) {
        //调用定义的方法operator
        operator();
    }
    //定义方法，方法中定义三元运算符
    public static void operator() {
        int i = 0;
        i = (1==2 ? 100 : 200);
        System.out.println(i);
        int j = 0;
        j = (3 <= 4 ? 500 : 600);
        System.out.println(j);
    }
}
```

- 注意事项:
    - 方法必须定义在一类中方法外
    - 方法不能定义在另一个方法的里面

```
//正确写法
public class Demo {
    public static void main(String[] args) {}
    public static void method() {}
}
```

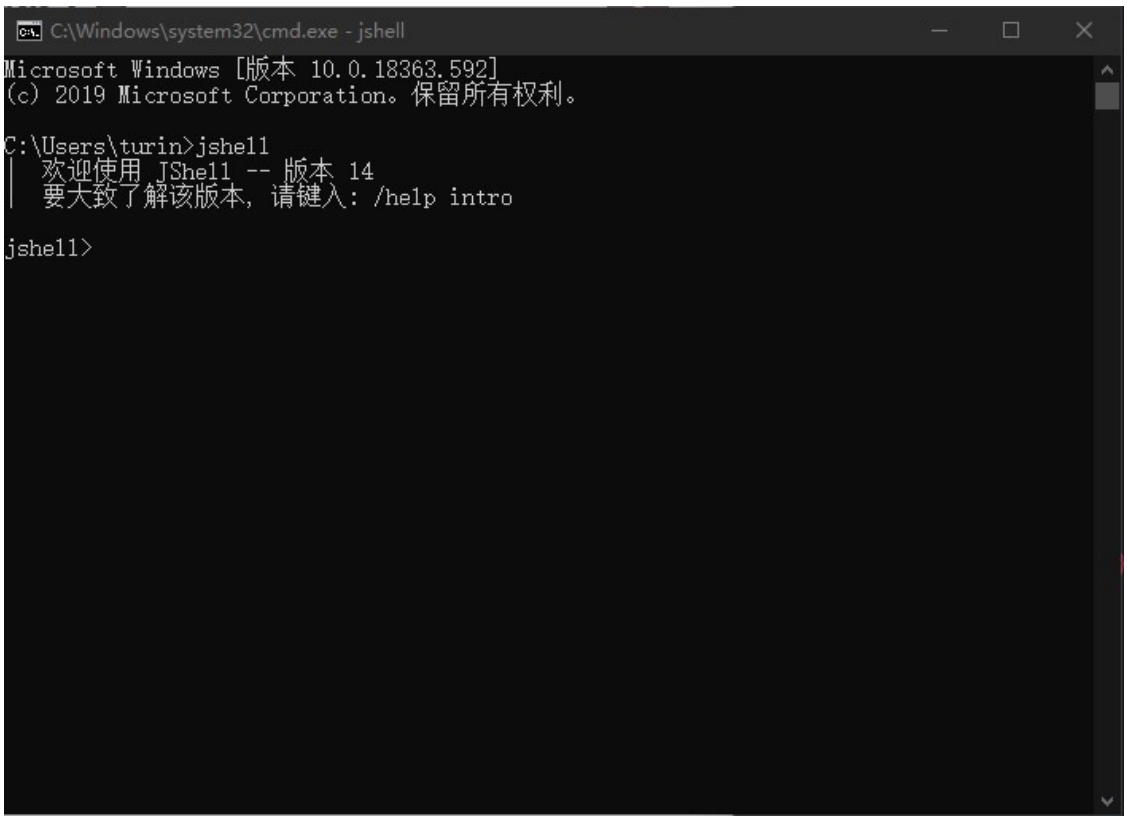
```
//错误写法
public class Demo {
    public static void main(String[] args) {
        public static void method() {}
    }
}
```

第四章 扩展知识点

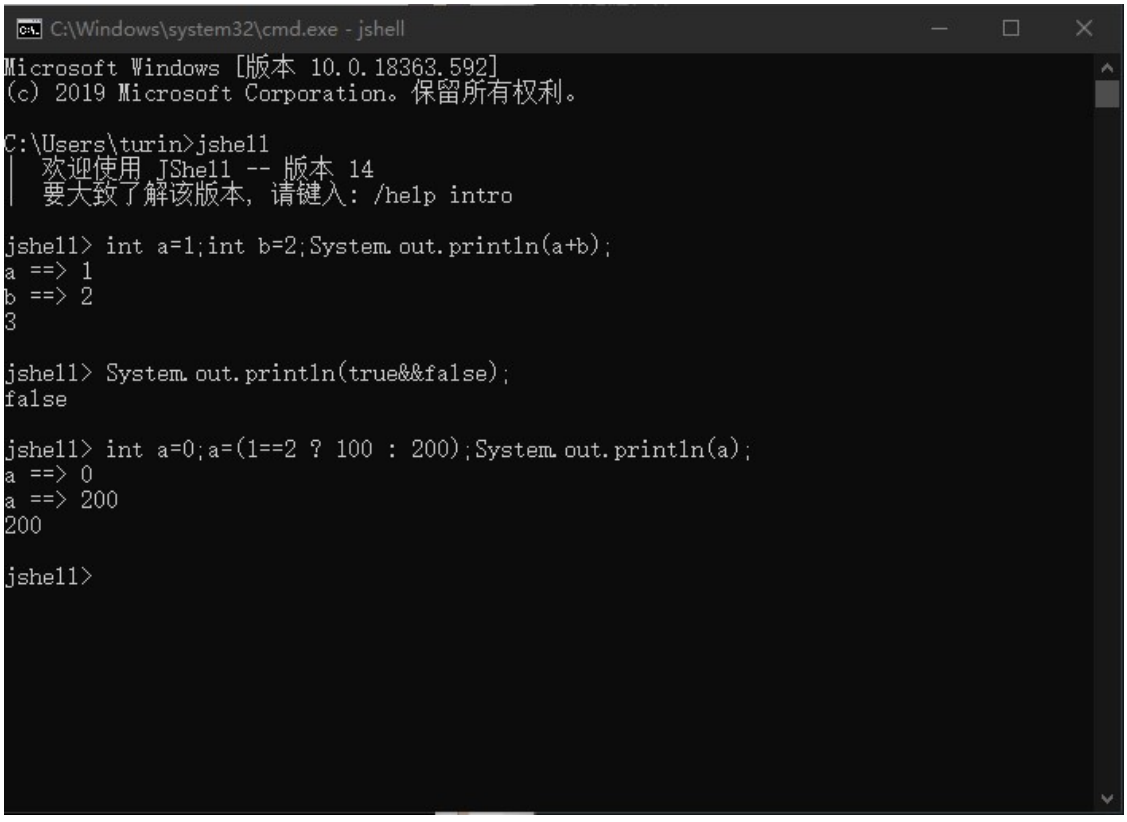
4.1 Jshell脚本工具

Jshell脚本工具是JDK9之后版本才有的新特性，当我们编写的代码非常少，又不愿意编写类，main方法，也不愿意去编译和运行，这个时候就可以使用JShell工具

- 启动Jshell工具，在DOS命令行直接输入JShell命令



- 接下来可以编写Java代码，无需编写类和方法，直接写方法中的代码即可，同时无需编译和运行，直接回车即可



- JShell工具，只适合片段代码的测试，开发更多的内容，建议编写在方法中

4.2 `+=`符号的扩展

```
public static void main(String[] args) {
    short s = 1;
    s += 1;
    System.out.println(s);
}
```

分析:
`s += 1`逻辑上看作是`s = s + 1`计算结果被提升为int类型，再向short类型赋值时发生错误，因为不能将取值范围大的类型赋值到取值范围小的类型。但是，**`s = s + 1`进行两次运算，`+=`是一个运算符，只运算一次，并带有强制转换的特点，也就是说`s += 1`就是`s = (short)(s + 1)`**，因此程序没有问题，编译通过，运算结果是2

4.3 常量和变量运算的扩展

```
public static void main(String[] args) {
    byte b1=1;
    byte b2=2;
    byte b3=1+2;
    byte b4=b1+b2;
    System.out.println(b3);
    System.out.println(b4);
}
```

分析:
`b3=1+2`，1和2是常量，为固定不变的数据，在编译的时候，编译器已经确定了结果没有超过byte的取值范围，可以赋值给变量`b3`，因此`b3=1+2`是正确的
`b4=b1+b2`，b2和b3是变量，变量的值是可能变化的，在编译的时候，编译器不确定`b1+b2`的结果是什么，因此会将结果以int类型进行处理，又int类型不能赋值给byte类型，因此编译失败