Java程序设计

2019春季

彭启民

pengqm@aliyun.com

Java语言基础

Java Scripts—不是Java,是脚本语言！

1 脚本语言,用于WWW动态页面

2 源程序嵌于HTML文档中

3 在HTML文档中用<Scripts>标记嵌入

4 与HTML文档一起被浏览器解释运行

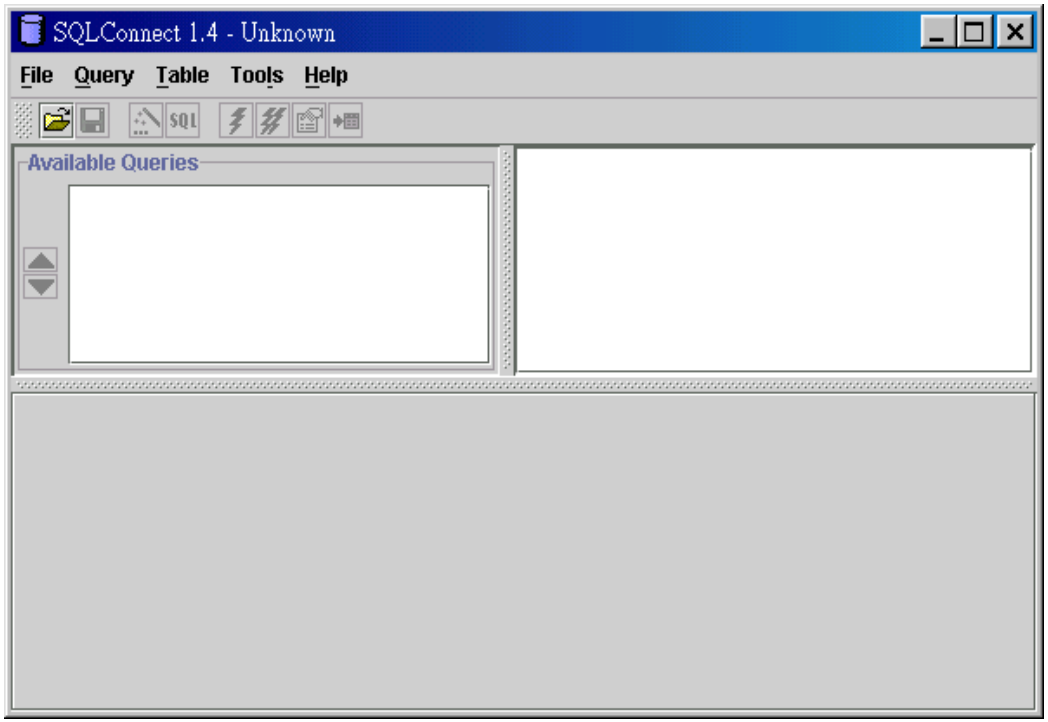
```
<script LANGUAGE="JavaScript">
<!--
        window.open("/tongzhi/0623.htm","name","width=550,height=550,scrol
lbars=yes")
//-->
</script>
```

(1) Java Application

1 独立应用程序

2 有一个或多个类

3 必须有一个类定义main()方法,
作为程序的入口.




```
class helloworld {
    public static void main(String args[]){
        System.out.println("aa");
    }
}
```

(2) Java Applet

1 小应用程序

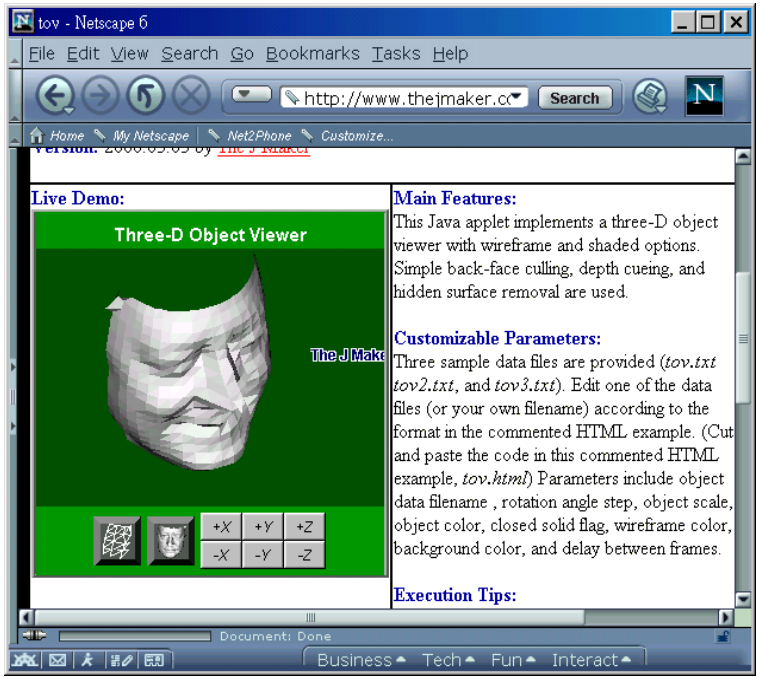
2 有一个类必须是applet的子类

3 用于WWW动态页面的开发

Oracle提醒：即将没有浏览器可以运行Applets了。各家主流浏览器厂家抱怨：插件机制使他们对性能和安全问题毫无掌控之力。

4 在HTML文档中用<applet>标记指明class文件名开发者们，很快就

```
<body><h1>this is an example of java applet.
<applet
code=helloworldapplet
width=600
height=300>
</applet></body>
```



```
import java.applet.*;
import java.awt.*;
public class helloworldapplet extends Applet{
    public void init() {}
    public void paint(Graphics g){
        g.setColor(Color.red);
        g.drawString("helloworld",10,200);
    }
}
```

(3) JSP(Java Server Page)

1 动态网页技术

2 在HTML 文档中加入Java程序段和JSP标签,构成JSP网页(*.jsp)

3 server接收到JSP网页访问请求后,首先执行程序段,将执行结果以html的格式返回给客户端.

```
<BODY>
<%
String test1="My  first jsp";
out.print(test1);
<%>
</body>
```

(54 Java Servlet

在服务器端执行的Java程序，没有 main 方法，不能够独立运行，需要Tomcat 等JSP/Servlet 容器的支持，由容器管理从创建到销毁的整个过程。

可以象applet一样作为插件嵌入到WEB中.

第一种方法:

```
<form method="get"action="......./myservlet>
```

第二种方法:

```
<SERVLET NAME="myservlet" CODE="myservlet.class .....>
```

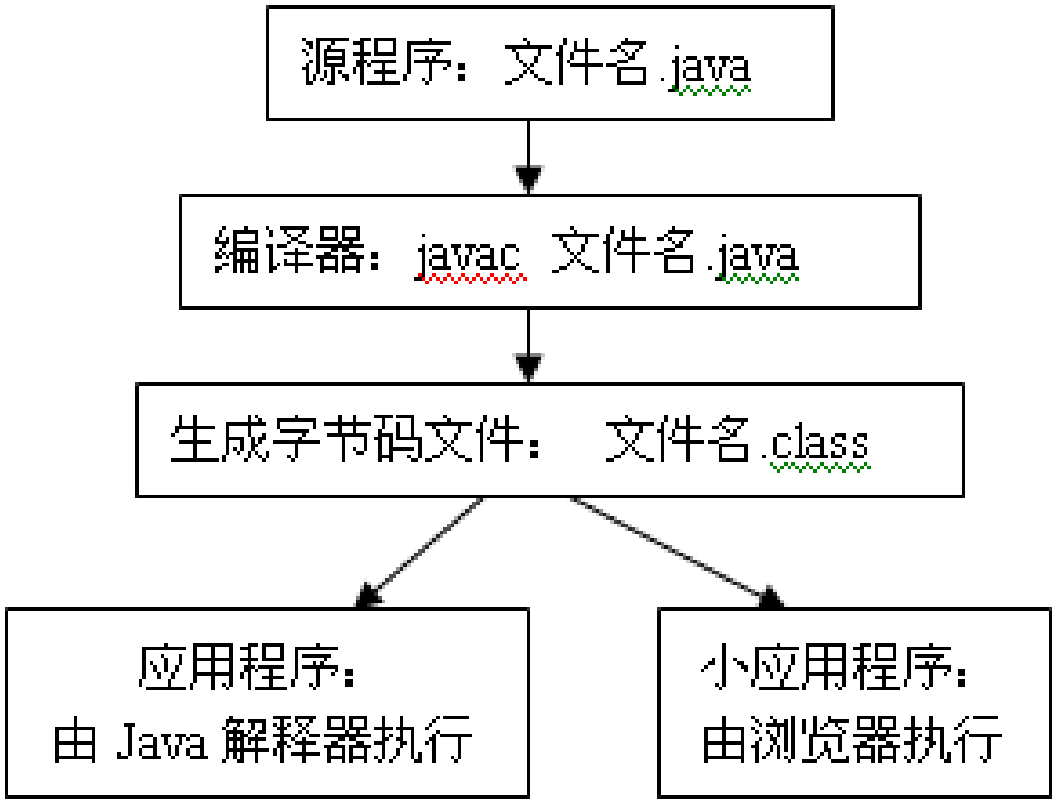
(5) JavaBeans

1 Java的软件组件模型,可以扩充Java程序的功能.

2 实现代码的重复利用

(6) shell

Java程序开发过程


源程序：文件名.java
编译器：javac 文件名.java
生成字节码文件： 文件名.class
应用程序：
由 Java 解释器执行
小应用程序：
由浏览器执行

Example java program

```
//   From TIJ by Bruce Eckel

import java.util.*;
public class HelloDate {
    public static void main(String[] args) {
        System.out.println("Hello, it's: ");
        System.out.println(new Date());
    }
}
```

source file:
HelloDate.java

All code must be inside a named class & the filename must match the classname **exactly** (case-sensitive)

Java程序的构成

- Java语言的源程序代码由一个或多个编译单元组成，每个编译单元可包含三个要素：
  - （1）一个包声明（package statement，可选）；
  - （2）任意数量引入语句（import statements）；
  - （3）类的声明（class declarations）和接口声明（interface declarations）。
- 该三要素必须以上述顺序出现。也就是说任何引入语句出现在所有类定义之前；如果使用包声明，则包声明必须出现在类和引入语句之前。每个Java的编译单元可包含多个类或接口，但是每个编译单元最多只能有一个类或者接口是公共的。

- 开发一个Java程序必须遵循下述基本原则：
  - Java区别大小写，即Public 和public是不同的标识符。
  - 用花括号｛｝将多个语句组合在一起，语句之间必须用分号隔开。

关于Java文件名

- 当类被 public 修饰时，此时文件名必须和该类保持一致！
  - （例如 public class A{} 只能放在 A.java 文件中，不然在用 javac 编译时会提示错误：类 A 是公共的，应在名为 A.java 的文件中声明）。也就是说在同一个 .java 文件中，不应该出现2个或2个以上的 public class 。
  - 如果有一个public class 类，就应该让文件名和此类名相同；
  - 如果没有public class 类， 即所有的 class 都没有修饰符，那么可以给该文件随便起名字，甚至可以不和任意一个类同名，哪怕是汉字名称都可以，但是扩展名必须为 .java 。
  - javac 进行编译的时候，最终会生成多个 .class 文件，每一个类对应一个 .class 文件。

- 一个可执行的应用程序必须包含下述基本框架：

```
public class Test
{
        public static void main(String args[])
        {
                ...;  //程序代码
        }
}
```

- 用文件名Test.java保存起来，即文件名必须与public class 后的类名相同（包括相同的大小写），并使用java作为扩展名。

- 编译源文件
  - 在命令控制台窗口中，输入编译命令javac：

```
javac Test.java
```

  - 按回车键确认编译，如果编译成功，编译器就在包含Test.java文件的同一个目录下建立一个Test.class字节码文件。

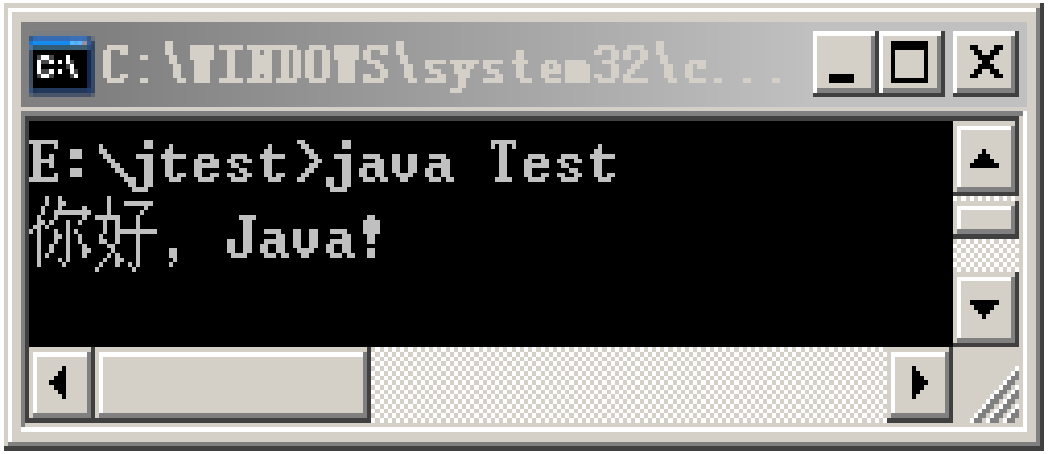
must include the .java extension

- 多个版本间的兼容：
  - 编译选项：

    javac -source 1.5 - target 1.5 test.java
- 编译时可用参数控制生成代码的大小：
  - 默认：code source.info 行序号
  - 调试：-g 默认+本地变量表
  - 最小：-g none 仅代码

- 执行字节码文件
  - Java编译器并不直接产生一个执行代码，因而不能直接在操作系统环境下执行。
  - 通过Java解释器命令：

```
java Test
```



Deconstructing a Java program

// From TIJ by Bruce Eckel

There are 3 kinds of comments in Java

1. //
2. /* … */

Implementation comments: tells a programmer reading your code about your implementation

3. /** … */

Documentation comments: used by javadoc to generate info for users of your classes

Deconstructing a Java program

```
//  From TIJ by Bruce Eckel
```

The main program **must** have this **exact** header:

```
public static void main(String[] args) {
```

public – the main method is publicly visible

static – allows the main program to be called without creating an object (details to follow)

void – no value is returned

String[] – array of Strings allows info to be given to the main program (String args[] is also OK)

关键字与保留字

- 正确识别java语言的关键字（keyword）和保留字（reserved word）是十分重要的。
- Java的关键字对java的编译器有特殊的意义，用来表示一种数据类型，或者表示程序的结构等。
- 保留字在当前版本没有启用，但在以后的升级版本中有可能启用。

关键字

- Java 关键字列表 (52个,依字母排序):
  abstract, assert（1.4）,boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum（1.5）,extends, final, finally, float, for, if, implements, import, instanceof, int, interface, long, module, native, new, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while
- 逐渐在增加…

保留字

- Java 保留字列表 (14个,依字母排序)
  - byValue, cast, false, future, generic, inner, operator, outer, rest, true, var， goto ，const,null
- 启用了4个值（true, false, null，var）
  - true、false和null为小写，而不是象在C++语言中那样为大写。
  - 它们不是关键字，而是文字。

- Java10启用var
  - 局部变量类型推断，被认为是Java 10中最值得注意的特性，简化了Java应用程序的编写，减少与编写Java相关的冗长度，同时保持对静态类型安全性的承诺。
  - 从JDK 5引进泛型，到JDK 7的〈〉操作符允许不绑定类型而初始化List，再到JDK 8的Lambda表达式，再到现在JDK 10的局部变量类型推断，Java类型推断正大刀阔斧的向前发展。
  - 仅限如下使用场景：
    - 局部变量、初始化for循环内部索引变量、传统的for循环声明变量
  - 不能用于：
    - 方法参数、构造函数参数、方法返回类型、字段捕获表达式（或任何其他类型的变量声明）

```
var
List<String> list = new ArrayList<String>();

var
Stream<String> stream = getStream();
```

- 无sizeof运算符，因为所有数据类型的长度和表示是固定的，与平台无关，不是象在C语言中那样数据类型的长度根据不同的平台而变化。这正是Java语言的一大特点。

- 标识符
  - java中的包、类、方法、参数、变量等的名字
  - 由大小写字母(可以是汉字、日文等任何国家文字的字符，但不建议使用)、数字、下划线、$(美元)组成。
  - 不能以数字开头。
  - 不能是Java中的关键字(用关键字作为标识符编译器会提示错误)。
  - 例：Inner$1、123、java hello、user_name、user-name

- 标识符命名习惯
  - 类、接口名：每个词的首字母都要大写
  - 方法、变量名：首字母要小写，其它单词首字母要大写
  - 常量：字母全部大写，单词之间用下划线分隔

课后：打开一个java程序，观察其中的标识符命名，区分类或接口名，变量或方法名。

JAVA的数据类型

- 基本数据类型
  - 整数、浮点数等
- 扩展类型/引用数据类型
  - 类、数组、接口

注:Java是静态类型的。在一个Java程序中，必须定义所用对象(数字、字符、数组等)的类型。这有助于编程人员很快发现问题，程序编译时可以检测类型错误。

Java的基本数据类型

数据类型	大小	最小值	最大值
byte	8bits	–128	127
short	16 bits	–32768	32767
int	32 bits	–2147483648	2147483647
long	64 bits	–9223372036854775808	9223372036854775807
float	32 bits	±1.40239846E-45	±3.40282347E+8
double	64 bits	±4.94065645841246544 E-324	±1.797693134862315 70E+308
char	16 bits	\u0000	\uFFFF
boolean	n/a	true 或false	

整型

- Java语言提供了多种整型数据类型
  - byte、short、int、long
  - 都是定义了一个整数，唯一的区别就是它们能够表示数据的范围。

- 能够表示数据的范围越大，占用的内存空间也就越大，因此，在程序设计中应该选择最合适的类型来定义整数。
  - int是最基本的，占用32位；
  - long，长的，也就是比int还长，占用64位；
  - short，短的，也就是比int还短，占用16位；
  - byte，字节，8位组成一个字节，占8位；
  - 数字默认是int型，后面加L强制为long型。

- 整型数示例。

```
public class IntegerDemo {
    public static void main(String [] args)
    {
        int a = 18;   //声明并初始化变量
        short x,y=10,z=20;  //同时声明多个变量
        x = (short)( y + z );        //这里为什么要强制转换?
        System.out.println("x的值是:"+x);
        long b = 12345678987654321L;  /*常量数值  后为何要加
           ‘L’字符?  */
        System.out.println("b的值是:"+b);
        b = a;                  //隐式类型转换
        byte c;
        c = (byte)a;            //强制类型转换
        System.out.println("c的值是:"+c);
    }
}
```

浮点型

- 在Java语言中有两种浮点数类型：
  - float是单精度型，占用32位内存空间
  - double是双精度型，占用64位内存空间。
- 默认情况下Java以double方式处理浮点数，如果要指定某个浮点数的存储方式是float而不是double，需要在书写具体数值时追加一个类型标志符F ，如123.321F。
- 浮点数
  - 浮点数这个名称是相对于定点数而言的，这个点就是小数点。浮点数就是指小数点可以根据需要改位置。

浮点数示例

```
public class FloatDemo {
    public static void main(String [] args)
    {
        double pi = 3.14159;
        float f = 2.7F;// 注意这里指定以float类型存储
        System.out.println("pi = " + pi);
        System.out.println("f = " + f);
        int n = 15, d = 4;
        f = n/d;                          //f里面存储的值是什么?
        System.out.println("f的值是: " + f);
        int radius = 10;
        double area = pi * radius * radius;
        //area里面存储的值是什么?
        System.out.println("area = " + area);
    }
}
```

程序运行结果：

```
pi = 3.14159
f = 2.7
f的值是： 3.0
area = 314.159
```

布尔类型

- 布尔类型（Boolean），用来表示布尔值。布尔变量的取值只可以是true或者是false。
- 布尔类型使用示例。

```
public class BooleanDemo {
    public static void main(String[] args)
    {
        boolean t = true;
        System.out.println("t 的值是 " + t);
        int x = 2;
        boolean y = (x > 2);
        System.out.println("y 的值是 " + y);
        x=x+1;
        y=(x>2);
        System.out.println("在x加1以后，y是"+y);
    }
}
```

程序运行结果：

t 的值是 true

y 的值是 false

在x加1以后，y是true

说明：在Java中，布尔数据类型不是一个整型值，不能将一个整数数赋值给布尔变量。

```
// y = x; // 不能编译!
```

char型

- char型是用来表示字母的，它仅能表示一个单一的字母。通常char型常量必须使用单引号括起来，以与数字区分开来。

- char 型变量的实例：

  char letter=’a’;

char型

- 在Java中，字符数据类型是用16位来表示的，可以表示Unicode字符集，这个字符集可以处理国际通用的字符。
- 一个字符可以被作为无符号的整数值来处理，因此可以进行一些算术操作，如比较两个字符值的大小等

- 在描述一个字符时，我们需要给该字符加上一对单引号。例如，’A’，这样系统就以字符方式识别A（注意：如果用双引号，则系统将识别为字符串String，而不是单个字符）。有些字符是不可打印的，可以用转义符（\）来表示。
- 如果需要以Unicode值的方式指定某个特定的字符可以使用转义符\u加上Unicode字符值。（注：该字符值以16进制方式来表示）。例如’\uF9A4’。

常用转义字符

转义字符	含义
\t	tab键
\b	backspace退格键
\n	newline换行
\r	carriage return回车
\’	single quote单引号
\”	double quote 双引号
\\	backslash反斜杠

char数据类型的使用示例。

```
public class CharDemo
{
   public static void main(String[] args) {
      char a = 'A';
      char b = (char) (a + 1);
      System.out.println(a + b);
      System.out.println("a + b is " + a + b);
      int x = 75;
      char y = (char) x;
      char omega = '\u03A9';
      System.out.println("y is " + y + " omega is
        " + omega);
   }
}
```

程序运行结果：

```
131
a + b is AB
y is K and omega is Ω
```

数值类型的互相转换

- 变量从一种数据类型转换到另一种数据类型，称为类型转换。
- 自动类型转换，表达式类型的自动提升
  - 整形、实型、字符型数据可以进行混合运算。运算时，不同类型的数据先转化为同一类型，字符型会转换为数值型，然后进行运算。
    - 表达式的类型为存储长度最大，精度最高的数据类型。
    - 当把级别低的表达式的值赋给级别高的变量时，系统会自动完成数据类型的转换。而级别高的表达式的值不能直接赋给级别低的变量。
    - 例：float f; f=34*12;f=f+23.5;
- 强制类型转换
  - 一般情况下，当精度高的类型向精度低的类型转换时，要明确指明，如：
  - double k=6;
  - float b=(float)k;

数据类型转换

- 凡是把占用比特数较少的数据转换成占用比特数较多的数据，可以由编译系统自动完成，这种转换称为隐式转换。
- 如果要把占用比特数多的数据转换成占用比特数少的数据，可以采用强制类型转换。

- 例如：

  double x = 9.99;

  int nx=(int)x; //这时nx就是一个int类型的数据了，并且它的值是9，后面的部分在造型过程中被丢掉了。如果想把一个浮点数舌入成“最接近”的整数，可以使用Math.round方法。在使用Math.round方法舌入后，可能还要使用造型来进行类型转换。

  Java还允许将一种类型的变量值赋给另一个变量，同时不进行显示造型，同样可以进行某些特定的赋值转换，允许的转换包括：

  buty->short->int_>long->float->double char->int

  当进行赋值的时候，会按照上面的顺序从左向右转换

不是所有的数据类型都允许隐含自动转换。例如，下面的语句把long型数据赋值给int型数据，在编译时就会发生错误：

```
long  i=10;
int  j=i;
```

这是因为当把占用位数较长的数据转化成占用位数较短的数据时，会出现信息丢失的情况，因而不能够自动转换。这时就需要利用强制类型转换，执行非兼容类型之间的类型转换。上面的语句写成下面的形式就不会发生错误：

```
long  i=10;
int  j=(int)i;
```

强制类型转换的格式是：

（数据类型）变量名

经过强制类型转换，将得到一个在“（）”中声明的数据类型的数据，该数据是从指定变量所包含的数据转换而来的。值得注意的是，指定变量本身不会发生任何变化。

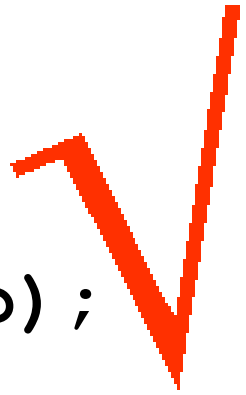
将占用位数较长的数据转化成占用位数较短的数据时，可能会造成数据超出较短数据类型的取值范围，造成“溢出”。如：

```
long  i=10000000000;
int  j=(int)i;
```

因为转换的结果已经超出了int型数据所能表示的最大整数（4294967295），造成溢出，产生了错误。

```
short a,b,c;
a=2;
b=3;
c=a+b;
```

```
short a,b,c;
a=2;
b=3;
c=(short)(a+b);
```



数组

- 数组，是相似元素的有序集合。在一个数组中的所有元素必须是相同类型的。

- 定义数组的方式有三种：
  - 零数组：表明数组存在，但没有值。
    - int firstArray[ ];
  - 预留空间：为数组预留一些存储空间，但并未马上赋值。
    - int secondArray[ ]=new int[5];
    - 在这里，我们定义了一个数组secondArray，并为其预留了够存放5个int型数的空间。
  - 一次性完成定义与赋值：
    - int intArray[ ]={30,20,10,-10}; //int intArray[ ]=new int[]{30,20,10,-10};

- 注意：
- 　　在Java语言中，数组一经定义之后就不能够改变其大小，这与其它程序设计语言中不同，在编程时一定要注意这点。

- 实例　源程序：arrayTest.java

```
public class arrayTest
{
        public static void main(String args[])
        {
                int intArray[ ]={30,20,10,-10};
                String stringArray[ ]={“first”,”second”,”third”};
                System.out.println(intArray[0]);
                System.out.println(intArray[1]);
                System.out.println(intArray[2]);
                System.out.println(intArray[3]);
                System.out.println(stringArray.length);
        }
}
```

- 两个变量声明：

  int intArray[ ]={30,20,10,-10};

  （由四个int型常量：30、20、10、-10组成）

  String stringArray[ ]={“first”,”second”,”third”}

（由三个String型常量“first”、“second”、“third”组成）

- 访问数组
  - 数组名[下标]

```
System.out.println(intArray[0]);
System.out.println(intArray[1]);
System.out.println(intArray[2]);
System.out.println(intArray[3]);
```

- 下标从0开始。 intArray[0]就是数组intArray中的第1个值：30，intArray[1]则是数组intArray的第2个值……依此类推。

- 数组元素不为基本原生数据类型时，存放的是引用类型，而不是对象本身。当生成对象之后，引用才指向对象，否则引用为null。
- 二维数组

  数组的数组, 有两种基本的定义形式：

  int[][] i = new int[2][3];（推荐）

  int i[][] = new int[2][3];

  二维数组的每个元素都是一个一维数组，且这些数组不一定都是等长的，数组空间不是连续分配的。

声明二维数组的时候可以只指定第一维大小，空缺出第二维大小，之后再指定不同长度的数组（注意，第一维大小不能空缺）。

```
public class ArrayTest4
{
    public static void main(String[] args)
    {
        //二维变长数组
        int[][] a = new int[3][];
        a[0] = new int[2];  //从最高维开始，分别为每一维分配空间：
        a[1] = new int[3];
        a[2] = new int[1];

        //Error: 不能空缺第一维大小
        //int[][] b = new int[][3];
    }
}
```

二维数组也可以在定义的时候初始化，使用花括号的嵌套完成（不必指定两个维数的大小，根据初始化值的个数不同，可以生成不同长度的数组元素）。

```
public class ArrayTest5
{
    public static void main(String[] args)
    {

        int[][] c = new int[][]{{1, 2, 3},{4},{5, 6, 7, 8}};

        for(int i = 0; i < c.length; ++i)
        {
            for(int j = 0; j < c[i].length; ++j)
            {
                System.out.print(c[i][j]+" ");
            }

            System.out.println();
        }

    }
}
```

- 数组的长度是指数组中元素的个数。
- 求数组长度
  - arrayTest.java：

```
System.out.println(stringArray.length);
```

打印出字符串数组stringArray的长度。根据前面的定义，可知输出值是3。

字符串

- Java中的string类不是基本数据类型，它以类的形式存在，封装了操作string的一些常用方法。
- Java中的任何对象（不仅仅是内建类型）都可以转换成一个字符串对象，因为Java中的每个对象都有一个toString()方法，在输出时会被自动调用.

- 可以使用两种方法来定义一个字符串：
  - 使用变量定义的方式，例如：

    String programOut=”Hello World!!”;
  - 使用new操作符，例如：

    String programOut=new String(”Hello World!!”);

- 通常使用第1种方法来定义String变量。

- 字符串合并：

  fullName=firstName+” “+lastName;
- 通过这个表达式，可以完成字符串合并。在本例中，实现了将字符串firstName与lastName合并，并在中间加上一个空格，形成一个新的字符串fullName。

- 实例 源程序：test605.java

```
public class test605
{
    public static void main(String args[])
    {
        String firstName = “Mike”;
        String lastName = “Joeden”;
        String fullName = firstName+” “+lastName;
        System.out.println(fullName);
    }
}
```

字符串的演示示例

```
public class StringDemo
{
   public static void main(String [] args)
   {
      String first = “zhang”, last = “san”;
      String name = first + “ “ + last;
      System.out.println(“Name = “ + name);
      double pi = 3.14159;
      String s = “Hello, “ + first;
      System.out.println(s + pi + 7);
      System.out.println(pi + 7 + s);
   }
}
```

- 运行结果如下：

```
Name = zhang san
Hello, zheng3.141597
10.14159Hello, zhang
```

求字符串长度：

- 在Java语言中提供了一个方法length()可以满足这个需求。它在Java API中定义为：

public int length()

其中，public代表任何外部类都可以访问和使用它；而int则表示这个函数将返回一个整数，即字符串的长度值。

- 实例 源程序：test606.java

```
public class test606
{
        public static void main(String args[])
        {
                String programOut = “I’m enjoy java program world”;
                int outlen = programOut.length();
                System.out.println(outlen);
        }
}
```

- 在字符串中查找：
- 在一个字符串中查找某一个单词，可以使用Java语言中的一个方法indexOf（）来满足这个需求。它在Java API中的定义是：

  public int indexOf(String findMe)
- public代表任何外部类可以访问它，而int表示它将返回一个整数值，也就是在字符串中包含要查找的单词findMe的第一个出现位置。

- 实例

```
public class test607
{
    public static void main(String args[])
    {
        String programOut = “I’m enjoy java program world”;
        int index = programOut.indexOf(“java”);
        System.out.println(index);
    }
}
```

- 求子串方法
- 子串是字符串的一部分。在一些字符串操作的程序中，经常会需要求一个字符串的子串。可以使用Java语言提供的求子串方法substring来实现。在Java API中是这样定义它的：

  public String substring(int startIndex)
  public String substring(int startIndex,endIndex)

- 它是一个可供所有外部类调用的方法（public指定）；它将返回一个字符串（返回值类型是String）；这个方法在调用时必须给参数，参数有两种形式：
  - 给出开始位置，表示从指定的开始位置开始，直到字符串结束；
  - 给出开始位置和结束位置，表示从指定的开始位置开始，直到结束位置。

- 实例

```
public class test608
{
    public static void main(String args[])
    {
        String programOut = “I’m enjoy java program world”;
        String preHalf=programOut.substring(0,14);
        String backHalf=prgramOut.substring(14);
        System.out.println(preHalf);
        System.out.println(backHalf);
    }
}
```

- String.getBytes()的问题
  - 本方法将返回该操作系统**默认的编码格式**的字节数组。如果使用这个方法时不考虑到这一点，会发现在一个平台上运行良好的系统，放到另外一台机器后会产生意想不到的问题。

- 实例
- class TestCharset
- {
- public static void main(String[] args)
- {
- new TestCharset().execute();
- }
- private void execute() {
- String s = "Hello!你好！ ";
- byte[] bytes = s.getBytes();
- System.out.println("bytes lenght is:" + bytes.length);
- }
- }

- 在中文WindowsXP系统下行结果为：

  bytes lenght is:12

- 英文UNIX环境下运行：

  $ java TestCharset

  bytes lenght is:9

- 请课后查阅Java中如何解决字符串编码问题的相关资料。

JAVA正则表达式

- 正则表达式就是一个字符串，但和普通的字符串不同的是，正则表达式是对一组相似字符串的抽象，如下面的几个字符串：

a98b　c0912d　c10b　a12345678d　ab

[ac]\\d*[bd]

JAVA正则表达式

- 正则表达式在字符串处理上有着强大的功能，sun在jdk1.4加入了对它的支持
  - java.util.regex中的两个类：Pattern和Matcher
- 在String中有四个方法可以使用正则表达式，它们是matches、split、replaceAll和replaceFirst。

- **matches方法**

  matches方法可以判断当前的字符串是否匹配给定的正则表达式。如果匹配，返回true，否则，返回false。matches方法的定义如下：

  public boolean matches(String regex)

  如上面给出的正则表达式我们可以用如下程序验证。

```
String[] ss = new String[]{"a98b", "c0912d",  "c10b",  "a12345678d",  "ab"};
for(String s: ss)
    System.out.println(s.matches("[ac]\\d*[bd]"));
```

- 输出结果：

```
true
true
true
true
true
```

- **split方法**

  split方法使用正则表达式来分割字符串，并以String数组的形式返回分割结果。split有两种重载形式，它们定义如下：

  public String[] split(String regex)
  public String[] split(String regex, int limit)

- 用split的第一种重载形式来分割HTTP请求头的第一行：

```
String s = "GET /index.html HTTP/1.1";
String ss[] = s.split(" +");
for(String str: ss)
System.out.println(str);
```

- 输出结果：

```
GET
/index.html
HTTP/1.1
```

- **replaceAll 和 replaceFirst方法**

  public String replaceAll(String regex, String replacement)
  public String replaceFirst(String regex, String replacement)

  这两个方法用replacement替换当前字符串中和regex匹配的字符串。

关于Java版本更新

- Java 的版本发布周期变更为每六个月一次，严格遵循时间点，将在每年的 3 月份和 9 月份发布。
- Java 10 是采用新发布周期的第一个版本，版本号是 18.3，提供了 109 项新特性。

- Java 10 的 12 项关键新特性：
  - JEP 286: 局部变量的类型推断。
  - JEP 296: 将 JDK 的多个代码仓库合并到一个储存库中
  - JEP 304: 垃圾收集器接口。
  - JEP 307: 向 G1 引入并行 Full GC
  - JEP 310: 应用类数据共享，以允许应用类放置在共享存档中
  - JEP 312: 线程局部管控（线程本地握手）。在线程上执行回调的新方法，允许停止单个线程，而不是只能启用或停止所有线程。
  - JEP 313: 移除 Native-Header Generation Tool (javah)
  - JEP 314: 额外的 Unicode 语言标签扩展。包括：cu (货币类型)、fw (每周第一天为星期几)、rg (区域覆盖)、tz (时区) 等
  - JEP 316: 在备用内存设备上分配堆内存。允许 HotSpot 虚拟机在备用内存设备上分配 Java 对象堆
  - JEP 317: 基于 Java 的 JIT 编译器（试验版本）
  - JEP 319: 根证书。开源 Java SE Root CA 程序中的根证书
  - JEP 322: 基于时间的版本发布模式。“Feature releases” 版本将包含新特性，“Update releases” 版本仅修复 Bug

Java12.0的消息

- JDK 12 已于2018年12月进入 Rampdown Phase One 阶段，所有新的功能特性被冻结，不会再加入更多的 JEP 。该阶段将持续一个月，主要修复 P1-P3 级错误。JDK 12 定于2019年3月19日正式发布。

- Java 12 的8 项关键新特性：
  - 189: Shenandoah: A Low-Pause-Time Garbage Collector (Experimental) ：新增一个名为 Shenandoah 的垃圾回收器，它通过在 Java 线程运行的同时进行疏散 (evacuation) 工作来减少停顿时间。
  - 230: Microbenchmark Suite：新增一套微基准测试，使开发者能够基于现有的 Java Microbenchmark Harness（JMH）轻松测试 JDK 的性能，并创建新的基准测试。
  - 325: Switch Expressions (Preview) ：对 switch 语句进行扩展，使其可以用作语句或表达式，简化日常代码。
  - 334: JVM Constants API ：引入一个 API 来对关键类文件 (key class-file) 和运行时工件的名义描述（nominal descriptions）进行建模，特别是那些可从常量池加载的常量。
  - 340: One AArch64 Port, Not Two ：删除与 arm64 端口相关的所有源码，保留 32 位 ARM 移植和 64 位 aarch64 移植。
  - 341: Default CDS Archives ：默认生成类数据共享（CDS）存档。
  - 344: Abortable Mixed Collections for G1 ：当 G1 垃圾回收器的回收超过暂停目标，则能中止垃圾回收过程。
  - 346: Promptly Return Unused Committed Memory from G1 ：改进 G1 垃圾回收器，以便在空闲时自动将 Java 堆内存返回给操作系统。

- 课后：
  - 有应用需要的同学查找相关资料，了解Java中正则式的表达形式。
  - 了解不同Java版本的差异。