



Java 语言程序设计

第一章

Java 语言基础知识

本课件部分内容取自清华大学郑莉老师的课件



目录

1.1 Java语言与面向对象的程序设计

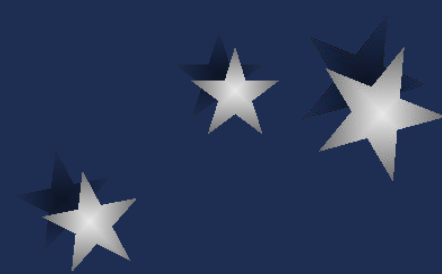
1.2 Java程序概述

1.3 基本数据类型与表达式

1.4 数组的概念

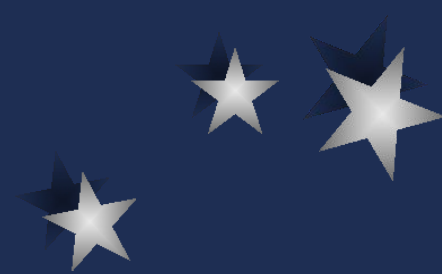
1.5 数组的创建和引用

1.6 本章小结



1.1 Java语言与面向对象的程序设计

- **Java语言**是一个面向对象的程序设计语言。
- 除了面向对象的特点以外，**Java语言**还在安全性、平台无关性、支持多线程、内存管理等许多方面具有卓越的优点。



1.1.2 Java语言的特点

- 面向对象
- 安全性

Java不支持指针，防止以不法手段访问变量的私有成员，避免指针操作中易犯错误。

Java的内部安全措施可以防止未经授权程序访问系统资源或危及客户机的安全性。

代码的验证和代码的执行功能分开，确保了代码执行是安全的（代码安全机制）

- 平台无关性

编译后的字节码对应于Java虚拟机，因此可在不同平台上运行

- 多线程

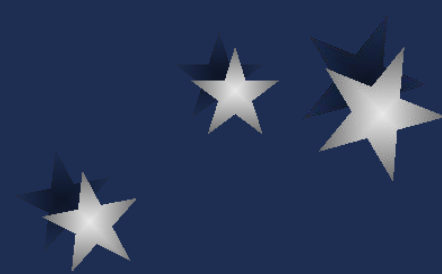
Java是第一个在语言级提供内置多线程支持的高级语言

- 内存管理

Java对内存自动进行管理并进行垃圾回收

概念说明--进程

- 进程通常被定义为一个正在运行的程序的实例，是一个程序在其自身的地址空间中的一次执行活动。
- 一个程序可以对应多个进程，例如可以同时打开多个记事本程序，从而创建多个进程。
- 设置一个进程要占用相当一部分处理器时间和内存资源。大多数操作系统不允许进程访问其他进程的内存空间，进程间的通信很不方便，编程模型比较复杂。



概念说明--线程

- 线程是系统分配处理器时间资源的基本单元，或者说进程之内独立执行的一个单元。对于操作系统而言，其调度单元是线程。操作系统为每一个运行线程安排一定的CPU时间时间片。
- 一个进程至少包括一个线程，通常将该线程称为主线程。一个进程从主线程的执行开始进而创建一个或多个附加线程，就是所谓基于多线程的多任务。通过多线程，一个进程表面上看同时可以执行一个以上的任务——并发。
- 创建线程比创建进程开销要小得多，线程之间的协作和数据交换也比较容易。
- **Java**是第一个支持内置线程操作的主流编程语言。

1.1.2 Java语言的特点(续)

- Java与C++的区别

Java中没有#include 和#define 等预处理功能，用import语来句包含其它类和包；

Java中没有structure，union及typedef；

Java中没有不属于类成员的函数，没有指针和多重继承，Java只支持单重继承；

Java中禁用goto，但goto还是保留的关键字；

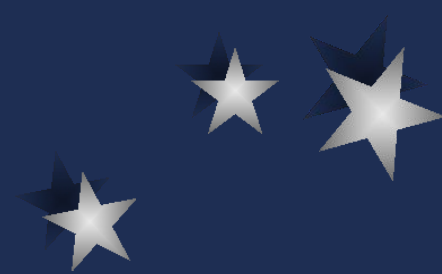
Java中没有操作符重载；

Java中没有全局变量，可以在类中定义公用、静态的数据成员实现相同功能；

.....

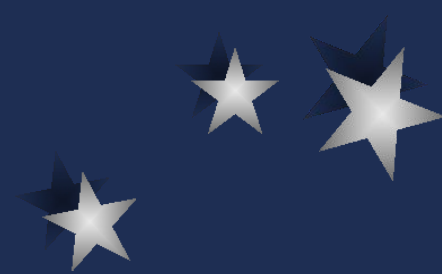
1.1.3 Java类库

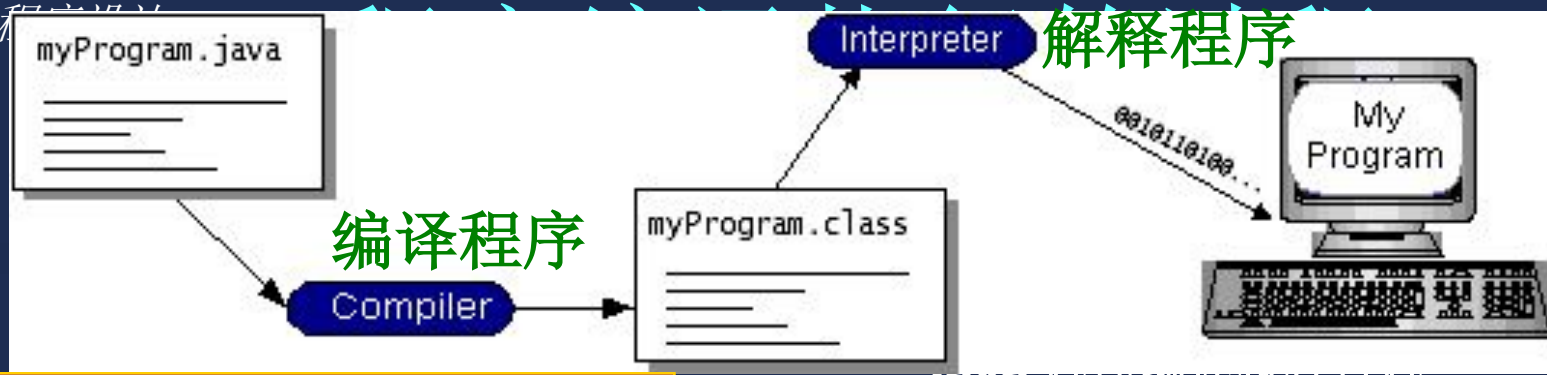
- 组成Java程序的最小单位是类，类封装了数据与处理数据的方法。
- 对于大多数常用的功能，有大量已经编译好、经过测试的类，这些类的集合就是Java类库。
- Java类库主要是随编译器一起提供，也有些类库是由独立软件开发商提供的。



1.2 Java程序概述

- Java 开发环境
- Application 举例
- Applet举例
- Servlet举例
- JSP举例





Java 程序编译执行过程

Java虚拟机（JVM）是Java Virtual Machine的缩写，它是一个虚构出来的计算机，Java字节码就像是在这台“计算机”上运行。实际在执行字节码时,JVM把字节码解释成具体平台上的机器指令执行。

它屏蔽了与具体硬件平台相关的信息,使得Java字节码可以在多种硬件平台上不加修改地运行。不同的硬件平台需要安装不同的Java虚拟机。



休息一下： 两个问题

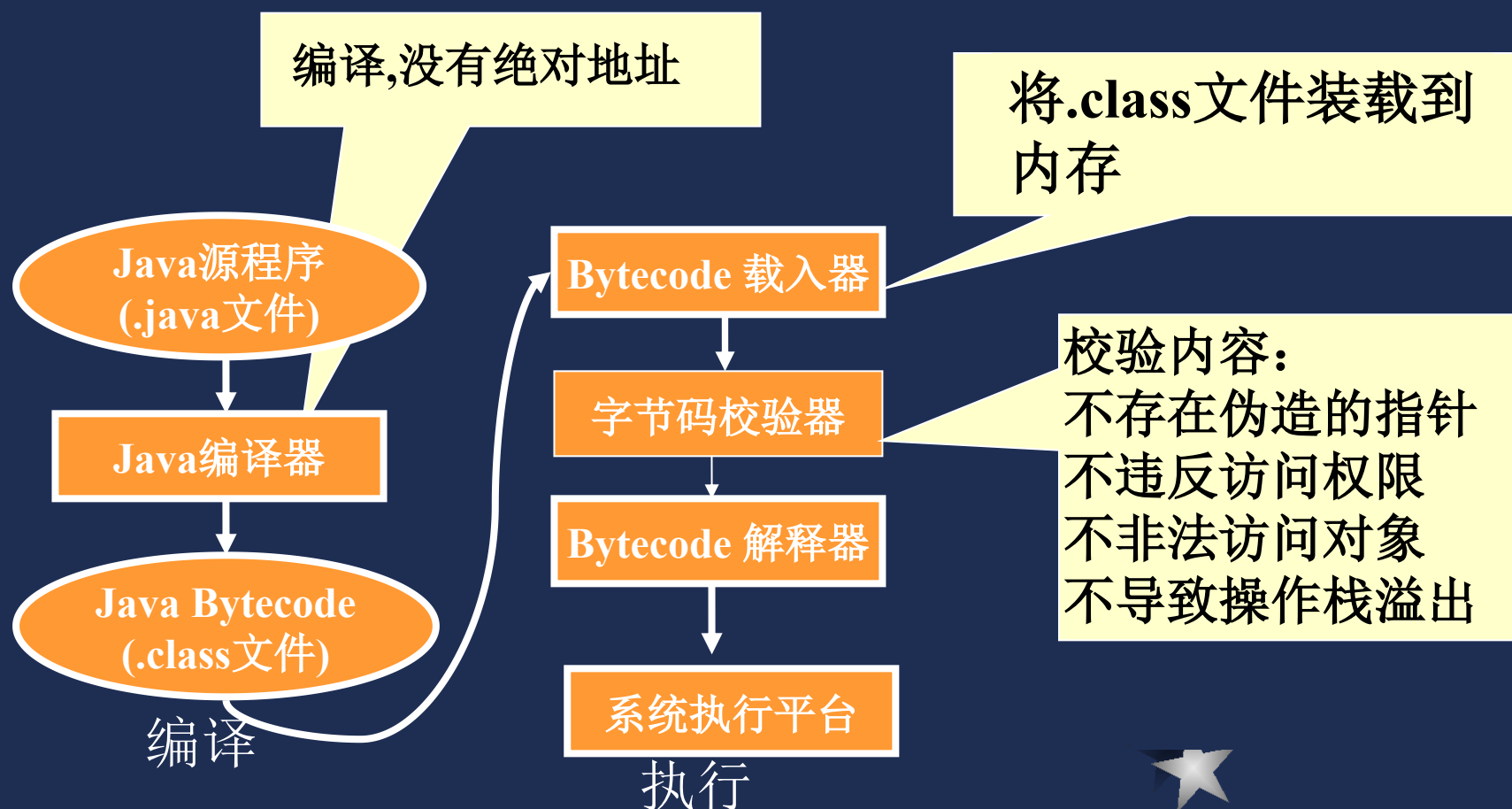
- Java是跨平台（平台无关）的？JVM呢？
- 如果要想Java程序在Unix操作系统上运行，需要怎么做？



1.2.1 Java开发环境 (续)

再进一步来看看Java程序的执行过程吧

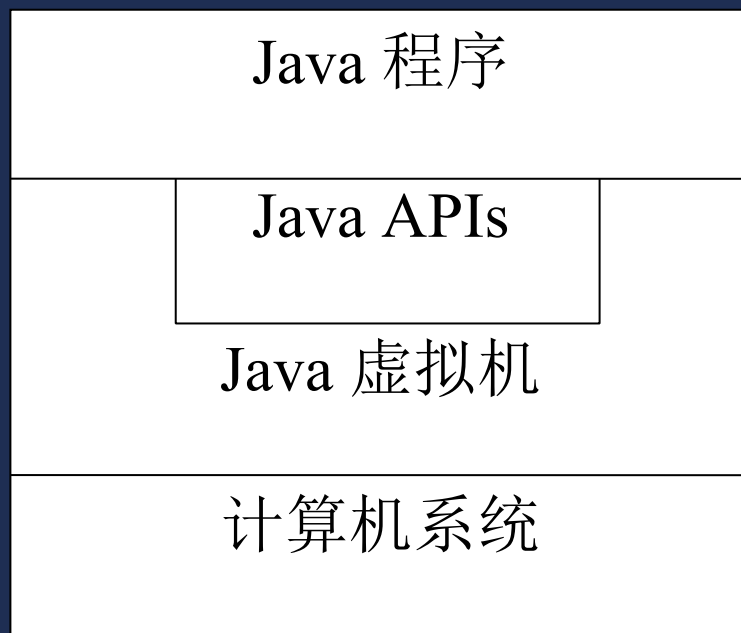
Java程序概述



1.2.1 Java开发环境 (续)

Java 程序概述

Java 平台



- **Java APIs (应用程序接口)**
 - 经过编译的，可在程序中使用的 **Java** 代码标准库。
- **Java VM (虚拟机)**
 - **Java** 程序由 **Java** 虚拟机程序执行（或解释执行）。



1.2.1 Java开发环境(续)

Java2 SDK(Software Development Kit)

Java软件开发工具包，包括3个版本：

Standard Edition (J2SE)

- 标准版，包含那些构成Java语言核心的类

Enterprise Edition (J2EE)

- 在J2SE基础上增加了企业应用方面的特性

Micro Edition (J2ME)

- 高度优化的Java运行环境，主要针对消费类电子设备



1.2.1 Java开发环境(续)

J2SE——J2SDK的核心部分，包含：

- **开发工具**
 - 编译器
 - 调试器
 - 文档制作工具
- **运行环境**
 - Java 虚拟机
 - Java API
 - 帮助文档
- **附加库**
- **Java程序（Applets 和 Applications）的演示**
 -



1.2.1 Java开发环境(续)

Java开发工具包括

- **Javac:**

Java编译器，用来将java程序编译成 Bytecode。

命令格式为：javac [选项] 源文件名

- **Java:**

Java解释器，执行已经转换成Bytecode的java应用程序。

命令格式为：java [选项] 类名 [参数]

- **Jdb:**

Java调试器，用来调试java程序。



1.2.1 Java开发环境(续)

Java开发工具包括(续)

- **Javap:**

反编译, 将.class类文件还原回方法和变量。

命令格式: `javap [选项] 类名`

- **Javadoc:**

文档生成器, 接受源文件 (后缀为.java) 输入, 然后自动生成一个HTML文件, 内容包括java源文件中的类、变量、方法、接口、异常等创建HTML文件。

- **Appletviewer:**

Applet解释器, 用来解释已经转换成Bytecode的java小应用程序。

命令格式: `appletviewer [选项] URL`

1.2.1 Java开发环境(续)

Java开发工具包括(续)

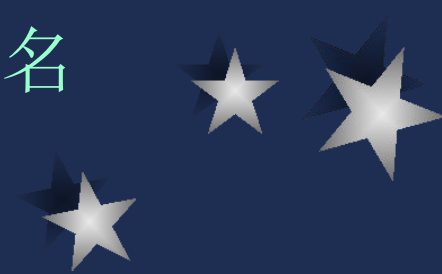
- **jar.exe**

压缩工具

- **javah.exe**

用以从java字节码上生成c语言头文件和源文件，这些文件用来在java的类中融入c语言的原生方法

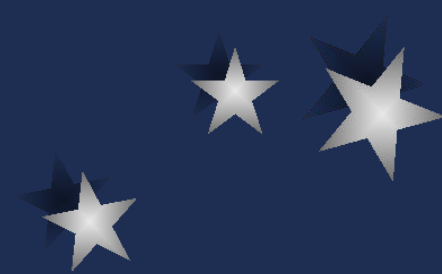
命令格式： `javah [选项] 类名`



1.2.1 Java开发环境(续)

环境安装——以j2sdk1.4.0为例

- 下载地址
<http://java.sun.com>
- 下载文件
j2sdk-1_4_0-win.exe
j2sdk-1_4_0-doc.zip
- 安装
直接运行“j2sdk-1_4_0-win.exe”。
- 帮助文档
解开“j2sdk-1_4_0-doc.zip”。



1.2.1 Java开发环境(续)

安装JDK后产生如下目录：

bin目录： **Java**开发工具，包括**Java**编译器、解释器等

lib目录： **Java**开发类库；

include目录： 包含了一些与c连接时所需的文件；

jre目录： **Java**运行环境，包括**Java**虚拟机（bin目录）、运行类库（lib目录）等

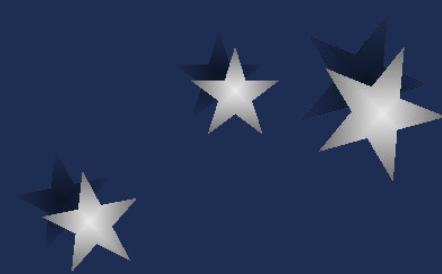
src.zip: JDK程序源代码压缩文件；

demo目录： 包含了许多Sun公司提供的Java小应用程序范例。

COPYRIGHT: JDK版本说明；

README.html: JDK的HTML说明文档；

README.txt: JDK基本内容及功能说明；



思考：
JDK和JRE的区别？



1.2.1 Java开发环境(续)

- 环境变量的配置

`java_home`

- jdk的安装路径，例如：`c:\j2sdk1.4.2`

`classpath`

- 用于指定java类文件的路径
- 一般配置如下：

```
.;%java_home%\lib;  
%java_home%\lib\tools.jar;.
```

`path`

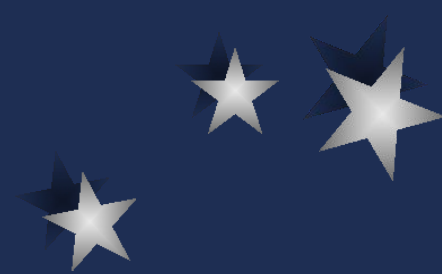
- 命令的搜索路径。如`java -version`
- 在原来的基础上增加：

```
%java_home%\bin;%java_home%\jre\bin
```



1.2.1 Java开发环境(续)

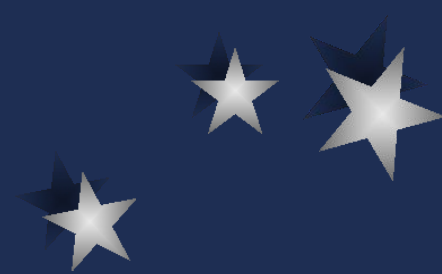
- 在MS-DOS命令提示符下，输入：
Java -version
出现 版本信息后，就为正确的安装。



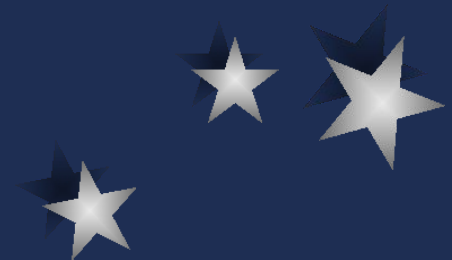
1.2.1 Java开发环境(续)

几种集成开发环境

- **JBuilder**
- **Sun ONE Studio**
- **Eclipse**
- **IntelliJ IDEA**



-
- **Java Application**
 - **Java Applet**
 - **Servlet**
 - **JSP**



1.2.2 Application 举例

Java 程序概述

Application

运行在客户端Java虚拟机上的Java程序

可在客户端机器中读写

可使用自己的主窗口、标题栏和菜单

程序可大可小

能够以命令行方式运行

主类必须有一个主方法`main()`，作为程序运行的入口。



1.2.3 Applica

使用如下命令编译并运行程序：

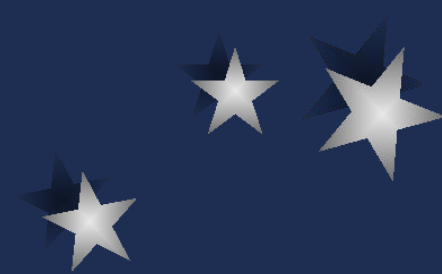
```
javac MyClass.java
```

```
java MyClass
```

运行结果如下：

The sum is: 3

```
public class MyClass
{ private int val1,val2 ;
  public void myFun(int x,int y)
  {
    val1=x ;
    val2=y ;
    System.out.println("The sum is: "+ (val1+val2) ) ;
  }
  public static void main(String arg[])
  {
    MyClass MyObj=new MyClass();
    MyObj.myFun(1,2);
  }
}
```



1.2.3 Applet 举例

- *Applet*——小应用程序

一种嵌入HTML文档中的Java程序

运行于支持Java的Web浏览器中

浏览器的解释器把字节码转换成具体平台的机器指令，在网页中执行小程序

*Applet*和*Application*的差别：运行环境的不同，小应用程序总是放在Web浏览器的图形用户界面中

1.2.3 Applet 举例(续)

——例1-2

JAVA Applet: //表明Applet使用Graphics类实现直线、矩形、椭圆形、字符串等的绘制

```
import java.awt.Graphics;
import java.applet.Applet;
public class MyApplet extends Applet
{
    public String s; //声明一个字符串
    public void init() //当浏览器载入某个Applet时, 自动调用init方法
    {
        s=new String("Hello World!");
        //从坐标(60, 40)开始绘制出字符串Hello World!
    }
    public void paint(Graphics g)
    {
        g.drawString(s,60,40);
    }
}
```

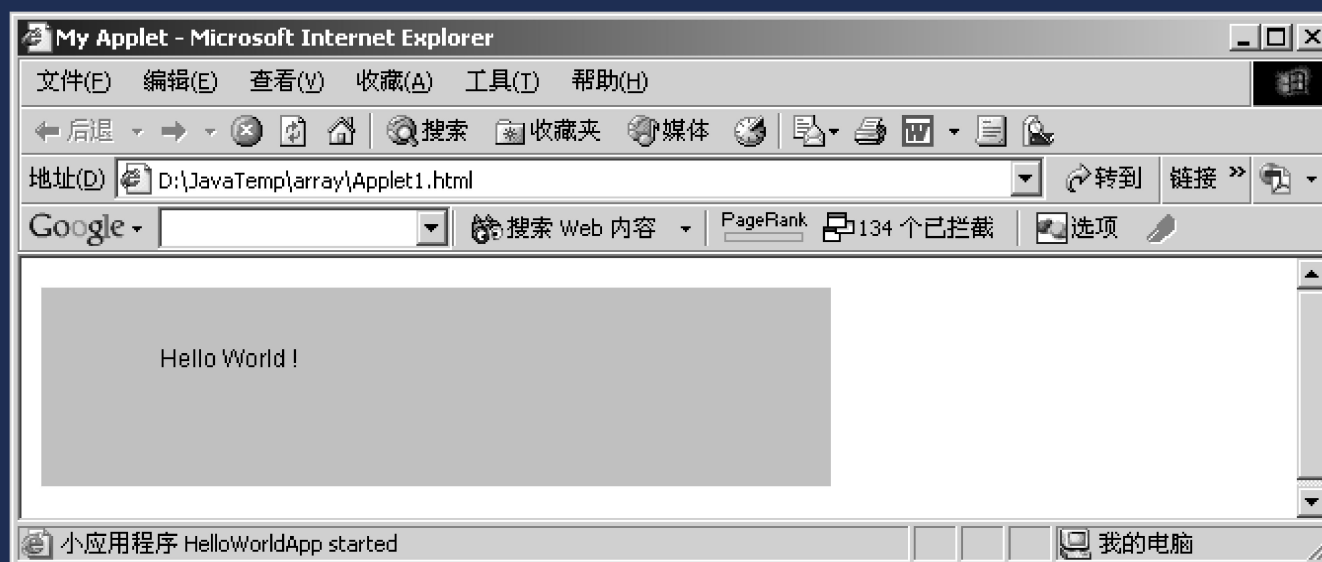
HTML:

```
<html>
<applet code= MyApplet.class width=400 height=400>
</applet>
</html>
```

1.2.3 Applet举例(续)

——例1-2运行

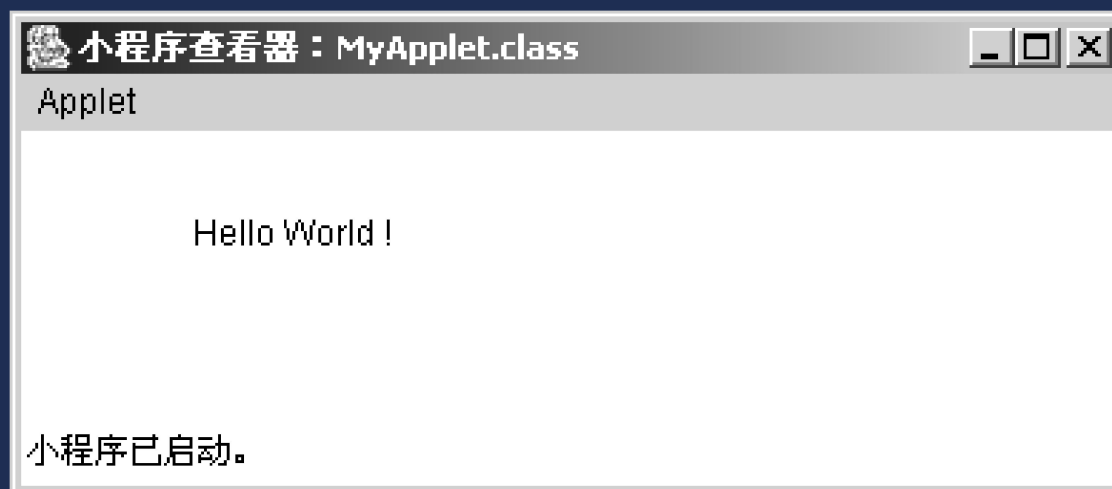
- 用支持Java的浏览器，比如IE6.0，打开Applet1.html



1.2.3 Applet举例(续)

——例1-2运行

- 用Java自带的appletviewer浏览
输入: `appletviewer Applet1.html`



1.2.3 Applet 举例

与Application相比，Applet具有明显的优点

- web浏览器提供了运行Applet所需要的许多功能
- Applet是在运行时通过网络从服务器端下载的，因而便于软件的发布和及时更新

Applet也有其局限性，

- 不能在客户机上读写本地文件
- 不能运行客户端主机的任何程序
- 也不能连接除它所在的服务器以外的其它机器

1.2.4 Servlet 举例

Servlet

介于浏览器（或其他HTTP客户端）与服务器之间，起到桥梁的作用。具体作用为：

- 读取客户端发送的数据
- 获取客户请求（**request**）中所包含的信息
- 产生响应结果，并将结果包含到一个文件中，比如HTML文件中
- 设置HTTP响应参数，比如告诉浏览器，文件类型为HTML
- 将文件返回给客户端

运行Servlet需要服务器的支持，需要在服务器中进行部署

Servlet用到的包在J2EE的API中能找到
所有的servlet都必须实现Servlet接口



1.2.5 Servlet 举例(续)

——例1-3

程序首先构建HttpServletRequest，并建立一个数据表单；点击submit按钮后，servlet再次被调用，并产生一个含有表单的网页。

```
public class EchoForm extends HttpServlet
{
    public void service(HttpServletRequest req,
        HttpServletResponse res) throws
        IOException{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        Enumeration flds = req.getParameterNames();
```

1.2.5 Servlet 举例(续)

——例1-3

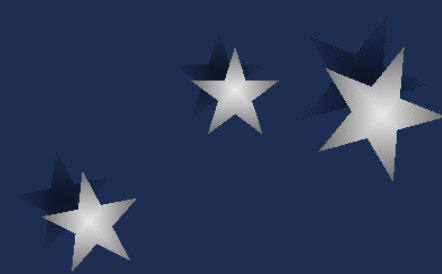
```
if(!flds.hasMoreElements())
{
    out.print("<html>");
    out.print("<form method=\"POST\" "
               + "action=\"EchoForm\">");
    for(int i = 0; i < 10; i++)
        out.print("<b>Field" + i + "</b> " +
                   "<input type=\"text\" " + " size=\"20\" " + "name=\"Field"
                   + i + "\" value=\"Value" + i + "\"><br>");
    out.print("<INPUT TYPE=submit name=submit "
               + "Value=\"Submit\"></form></html>");
}
```

1.2.5 Servlet 举例(续)

——例1-3

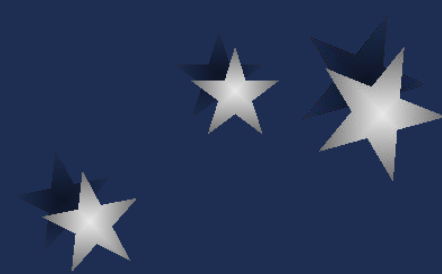
```

else
{
    out.print("<h1>Your form contained:</h1>");
    while(flds.hasMoreElements())
    {
        String field= (String)flds.nextElement();
        String value= req.getParameter(field);
        out.print(field + " = " + value+ "<br>");
    }
}
out.close();
}
    
```



1.2.6 JSP举例

- JSP就是嵌入了Java代码的HTML
- JSP和servlet同是服务器端的技术。实际上，JSP文档在后台被自动转换成servlet
- 使用JSP便于实现网页的动静分离
- 相对于Servlet，JSP在服务器的部署简单



1.2.6 JSP 举例(续)

```
<html>
```

```
<body>
```

```
<H1>The time in second is:
```

```
<%= System.currentTimeMillis()/1000 %>
```

```
</H1>
```

```
</body>
```

```
</html>
```



程序设计语言的一些基本概念

标识符

数据类型

变量

常量

数据成分

表达式

运算成分

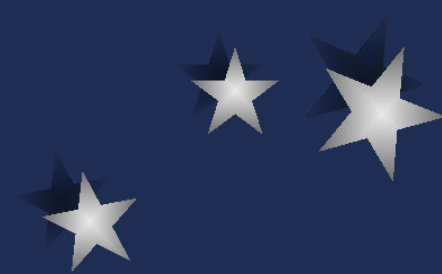
输入与输出

传输成分



1.3 基本数据类型与表达式

- 标识符、关键字、变量与常量
- 基本数据类型[1]
- 表达式与运算符
- 类型转换
- 标准输入输出简介



标识符

标识符、变量与常量

标识符是由程序员定义的单词，用来给程序中的数据、方法和其他用户自定义对象命名。

标识符的第一个字符必须是下列字符之一：

- 大写字母 (A-Z)、小写字母 (a-z)
- 下划线(_)、美元符号 (\$)

标识符的第二个字符及后继字符必须是：

- 上述列表中的任意字符
- 数字字符 (0-9)

例子：

- 合法标识符： `identifier`, `userName`, `User_Name`, `_sys_val`, `$change`
- 非法标识符： `2mail`, `room#`, `class`



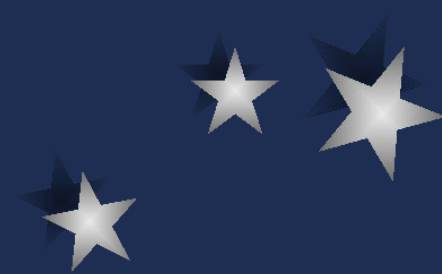
标识符(续)

标识符、
变量与常量

Java语言对标识符区别大小写，Myfirst与myfirst分别代表不同的标识符

使用标识符应该在一定的程度上反应出它所表示的变量、常量、类或对象和含义。

标识符不能与关键字（保留字）同名。



关键字

标识符、
变量与常量

程序设计语言本身会定义一些专用名称，称之为**保留字**或**关键字**，它们具有特定含义，程序员不能另做他用。

java语言中的所有保留字：

abstract, break, byte, boolean, catch, case, class, char, continue, default, double, do, else, extends, false, final, float, for, finally, if, import, implements, int, interface, instanceof, long, length, native, new, null, package, private, protected, public, return, switch, synchronized, short, static, super, try, true, this, throw, throws, threadsafe, transient, void, while

java语言中的保留字均用小写字母表示。

变量

标识符、
变量与常量

- 程序运行时，通过内存地址对内存数据进行读写；
- 程序员写源程序时，如果也是直接使用内存地址来读写内存数据，会很麻烦！
- 如果用名字来代替地址，在程序中出现的代表内存空间的名字，让编译器来跟踪数据实际存放的物理内存地址，就可以简化编程——**变量**；

变量（续）

标识符、
变量与常量

- 变量用来代表内存存储空间，该存储空间用来存放被加工的数据或处理的结果。源程序中对变量的操作（读和赋值）实际上是对存储空间的读写操作。
- 存储在变量中的值可以改变。
- 变量具有三个重要属性：名称、值和数据类型。
- 需要掌握：变量定义、变量赋值、变量读取

变量（续）

- 变量定义

使用变量来对内存数据进行存取前，必须先定义变量，目的是建立变量和存储空间的对对应关系。变量必须先定义后使用。

变量定义需指明变量的名称和数据类型，必要时还可以指定变量的初始数值。

JAVA语言的变量的定义语句格式为：

数据类型 变量名表；



变量（续）

- 变量赋值

把一个值写入变量代表的存储空间。

变量赋值格式：变量名=表达式

示例：

```
int age; //变量定义
```

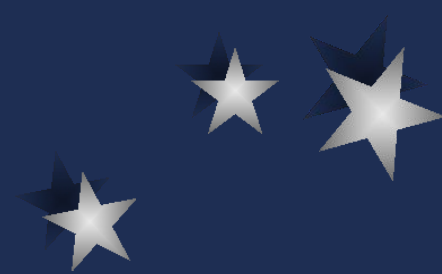
```
age = 18; //变量赋值
```



常量

标识符、
变量与常量

- 和变量类似，常量也是内存存储空间的名字，代表一片内存存储空间，但一旦被初始化以后就不允许程序去改变该存储空间中的数据。
- 如： `final double PI=3.14159;`



1.3.2 基本数据类型

数据是计算机处理的对象。

数据依据其本身的特点可以归为不同的类：**整数、小数、字符、字符串**等。

(高级程序语言特点：按照人对于数据的表达方式)

程序设计语言应该具备区分各种类型数据的能力，于是引入了数据类型的概念。

数据类型包括两层含义：定义了一系列的值(**属于该类型的数据能够取值的范围**)以及能应用于这些值上的一系列操作(**数据操作**)。

1.3.2 基本数据类型

数据类型的分类



1.3.2 基本数据类型

- 整型

类型名	比特位	表达范围
byte	8 bits	-128 ~ +127
short	16 bits	-32768 ~ + 32767
int	32 bits	$-2^{31} \sim (2^{31}-1)$
long	64 bits	$-2^{63} \sim (2^{63}-1)$
char	8 bits	0 ~ 65535

注：在声明长整型时，在整数后要加“L”或“l”以示区别。

1.3.2 基本数据类型(续)

基本数据类型与表达式

● 整型数据运算

比较运算符（关系运算符）

- 算术比较运算符 <, <=, >, >=
- 算术相等比较运算符 ==, !=

算术运算符

- 一元运算符 + 和 -
- +, -, *, /, 和 %（取余）
- 自增/自减运算符 ++/--
- 移位运算符 <<, >>, >>>
- 位运算符 ~, &, |, ^

条件运算符 ? :

类型转换运算符

字符串连接运算符 +



1.3.2 基本数据类型(续)

基本数据类型与表达式

● 浮点数值型

float

- 单精度浮点数
- 32-bits (4字节)
- $-m \cdot 2^e \sim m \cdot 2^e$

m 是一个小于 2^{24} 的正整数

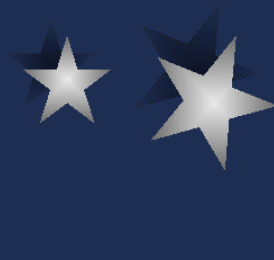
e 是一个介于 -149和104之间 (含) 的整数

double

- 双精度浮点数
- 64-bits (8字节)
- $-m \cdot 2^e \sim m \cdot 2^e$

m是一个小于 2^{53} 的正整数

e是一个介于 -1045 和 1000之间 (含) 的整数



1.3.2 基本数据类型(续)

基本数据类型与表达式

- 浮点数运算

比较运算符（关系运算符）

- 算术比较运算符 <, <=, >, and >=
- 算术相等比较运算符 == and !=

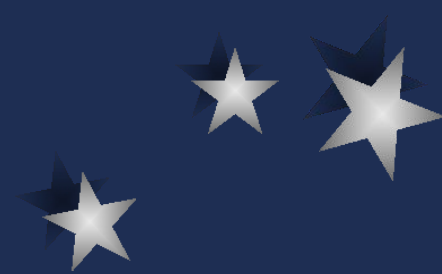
算术运算符

- 一元运算符 + 和 -
- +, -, *, /, 和 %（取余）
- 自增/自减运算符 ++/--
- 移位运算符 <<, >>, and >>>
- 位运算符 ~, &, |, and ^

条件运算符 ? :

类型转换运算符

字符串连接运算符 +



1.3.2 基本数据类型(续)

基本数据类型与表达式

- 布尔类型和布尔值

布尔类型表示一个逻辑量， 有两个取值：
true和false

例如：

```
boolean is_salaried;//定义布尔类型的变量  
boolean is_hourly;  
is_salaried = true;  
is_hourly = false;
```



1.3.2 基本数据类型(续)

基本数据类型与表达式

- 布尔运算符

关系运算符 == , !=

逻辑“非”运算符 !

逻辑“与”和逻辑“或”运算符: && 和 ||

条件运算符 ? :

字符串连接运算符 +



1.3.2 基本数据类型(续)

- 字符型

一个字符在内存中占据**16**位存储空间
不允许字符类型和整型数据之间进行相互替代和相互转换。



1.3.2 基本数据类型(续)

基本数据类型与表达式

- **String——字符串**

String 是一个类

是JDK标准类库的类

- `String animal = "walrus";`
- `char data[] = {'w', 'a', 'l', 'r', 'u', 's'};`
`String animal = new String(data);`

字符串是常量，它们的值在创建之后不能更改。

1.3.2 基本数据类型(续)

基本数据类型与表达式

- 文字量

直接出现在程序中并被编译器直接使用的值.

整数文字量

- 十进制

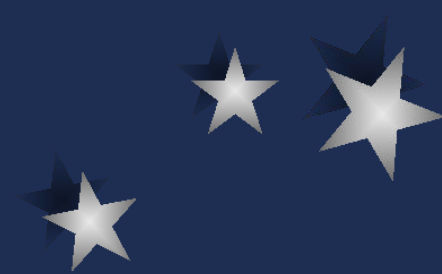
如: 15

- 十六进制 (以0x开头)

如: 0xff

- 八进制 (以0开头)

如: 0377



1.3.2 基本数据类型(续)

浮点型文字量

- ▣ 十进制数形式：由数字和小数点组成，且必须有小数点，如0.123, 1.23, 123.0
- ▣ 科学计数法形式：如123e3或123E3，其中e或E之前必须有数字，且e或E后面的指数必须为整数。
- ▣ 默认是double型。
- ▣ float型（单精度浮点型）的值,必须在数字后加f或F,如1.23f。

文字量

1.3.2 基本数据类型(续)

布尔文字量

- 布尔类型只有两个值，由文字量 `true` 和 `false` 表示

字符文字量

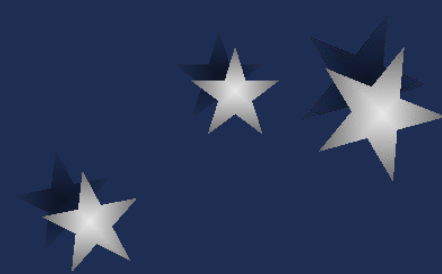
- 一个字符文字量表示为一个字符或者一个转义序列，用单引号括起

例如 `'a'` `'z'` `'@'`

- 格式字符

```
\ b    backspace BS
\ t    horizontal tab HT
\ n    linefeed LF
\ f    form feed FF
\ r    carriage return CR
\ "    double quote "
\ '    single quote '
\ \    backslash \
```

文字量



1.3.2 基本数据类型(续)

字符串文字量

- 由零个或多个字符组成，以双引号括起
- 每一个字符都可以用转义序列来表示
- 例如：

`""` // 空字符串

`"\""` // 只包含 `"` 的字符串

`"This is a string"` // 有16个字符的字符串

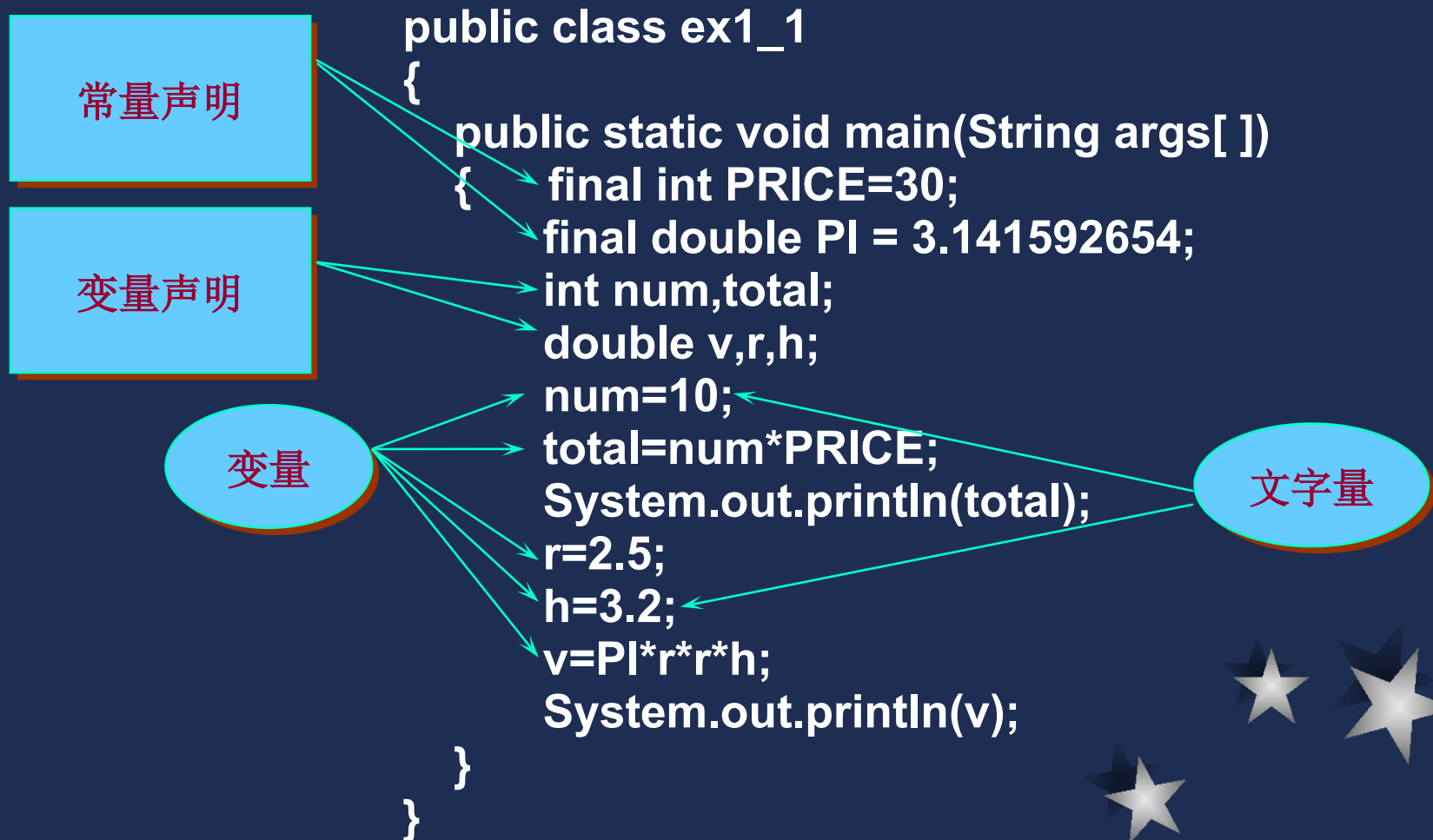
`"This is a " + "string"`

//字符串常量表达式, 由两个字符串常量组成

文
字
量

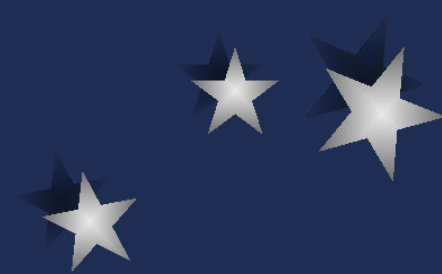
1.3.2 基本数据类型(续)

基本数据类型与表达式



1.3.3 表达式与运算符

- ◆ 表达式是由运算符、操作数和括号组成的、计算求值的基本单位。
- ◆ 操作数可以是：常量、变量、函数调用和表达式；
- ◆ 运算符种类：
 - 算术运算符
 - 赋值运算符
 - 比较运算符（关系运算符）
 - 逻辑运算符
 - 位运算符



1.3.3 表达式与运算符(续)

- 算术运算符

`+`, `-`, `*`, `/`, `%`, `++`, `--`

例如: `3+2`; `a-b`; `i++`; `--i`;

注意:

- (1) “`+`”除有字符串连接的功能外,还能将字符串与其他的数据类型相连组成一个新的字符串,条件是表达式中至少有一个字符串。如:“`a`”+68;结果是“a68”。
- (2) `++a`是在变量参与参与运算之前自增1,然后用新值参与运算;而`a++`则是先用原来的值参与运算,然后再将自己加1。`--a`与`a--`与上述相类似。
- (3) 除号“`/`”,整数除和小数除是有区别:整数之间做除法时,只保留整数部分而舍弃小数部分。

1.3.3 表达式与运算符(续)

- 赋值运算符

=, 及其扩展赋值运算符如+=, -=, *=, /=等

例如: `i=3; i+=3;`

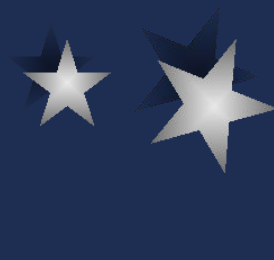
- 关系运算符

关系表达式的类型永远是布尔类型 (`bool`) .

算术比较运算符 `<`, `<=`, `>`, and `>=`

类型比较运算符 `instanceof`

- 例如: `e instanceof Point` // `Point` 是一个类



1.3.3 表达式与运算符(续)

- 逻辑运算符

“与”运算 &&

- 如果两个操作数的值都为true运算结果为true; 否则, 结果为false.

“或”运算 ||

- 如果两个操作数的值都为false运算结果为false;否则, 结果true

“非”运算符!

- 操作数的类型必须是布尔类型
- 如果操作数的结果为 false, 则表达式的结果为 true, 如果操作数的结果为 true则表达式的结果为 false

1.3.3 表达式与运算符(续)

条件运算符 (表达式1? 表达式2: 表达式3)

首先计算表达式1

如果表达式1的值为 true, 则选择表达式2的值

如果表达式1的值为 false, 则选择表达式3的值

例如: `result=(sum==0 ? 1 : num/sum);`



String.valueOf:将基本类型数据转换为String对象

```
public class DataType {
```

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

```
        //算术表达式
```

```
        System.out.println("the value of:102/20 is: "+String.valueOf( 102/20 ));
```

```
        System.out.println("the value of:102%20 is: "+String.valueOf( 102%20 ));
```

```
        System.out.println("the value of:1/2 is: "+String.valueOf( 1/2 ));
```

```
        System.out.println("the value of:1.0/2 is: "+String.valueOf( 1.0/2 ));
```

```
        System.out.println("the value of:'A'+32 is: "+String.valueOf( 'A'+32 ));
```

```
        //关系表达式
```

```
        System.out.println("the value of:(2+3)<10 is: "+String.valueOf( (2+3)<10 ));
```

```
        System.out.println("the value of:(2+3)==10 is: "+String.valueOf( (2+3)==10 ));
```

```
        DataType theObj = new DataType();
```

```
        System.out.println("the value of:theObj instanceof of DataType is: "+String.valueOf( theObj instanceof DataType ));
```

```
        //逻辑表达式
```

```
        int a=1,b=2,c=4;
```

```
        System.out.println("a=1,b=2,c=4");
```

```
        System.out.println("the value of:a<b&& b<c is: "+String.valueOf( a<b && b<c ));
```

```
        System.out.println("the value of:a>b||b>c is: "+String.valueOf( a>b || b>c ));
```

```
        System.out.println("the value of:!(a>b) is: "+String.valueOf( !(a>b) ));
```

```
        //条件表达式
```

```
        System.out.println("the value of:a>b? 1: 0 is: "+String.valueOf( a>b?1:0 ));
```

```
    }
```

```
}
```

表达式练习运行结果

the value of:102/20 is: 5

the value of:102%20 is: 2

the value of:1/2 is: 0

the value of:1.0/2 is: 0.5

the value of:'A'+32 is: 97

the value of:(2+3)<10 is: true

the value of:(2+3)==10 is: false

the value of:theObj instanceof of DataType is: true

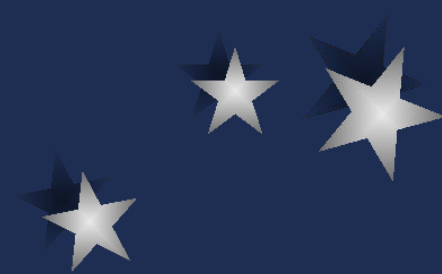
a=1,b=2,c=4

the value of:a<b&& b<c is: true

the value of:a>b||b>c is: false

the value of:!(a>b) is: true

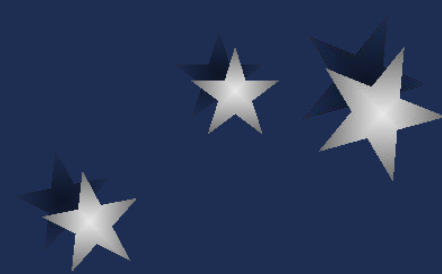
the value of:a>b? 1: 0 is: 0



1.3.4 类型转换

基本数据类型与表达式

- 每个表达式都有类型
- 如果表达式的类型对于上下文不合适
 - 有时可能会导致编译错误
 - 有时语言会进行隐含类型转换
- 数据类型的转换有自动转变和强制转换两种形式



1.3.4 类型转换（续）

- 自动类型转换

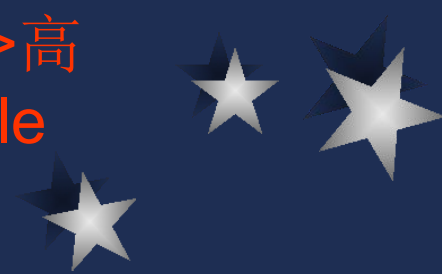
整型, 实型, 字符型数据可以混合运算。运算中, 不同类型的数据先转化为同一类型, 然后进行运算, 转换从低级到高级。

自动类型转变, 需要同时满足两个条件, 第一是两种类型彼此兼容, 第二是目标类型的取值范围大于源类型。

例如, 当byte型向int型转换时, 由于int型的取值范围大于byte型, 就会发生自动转换。

不同类型数据间的优先关系如下:

低----->高
byte, short, char -> int -> long -> float -> double



1.3.4 类型转换（续）

- 强制类型转换

当两种类型彼此不兼容，或者目标类型的取值范围小于源类型，自动转换无法进行，这时就需要进行强制类型转换。强制类型转换的通用格式如下：

目标类型 变量 = (目标类型) 值；

高级数据要转换成低级数据，需用到强制类型转换

如： `int i = 256; byte b = (byte) i; b = ?`

- 表达式数据类型的自动提升

如果在JAVA应用程序表达式中出现数据类型不一致的情形，那么JAVA运行时系统首先自动将低优先级类型的数据转换成高优先级类型的数据，然后才进行表达式值的计算。

1.3.5 标准输入输出简介

基本数据类型与表达式

标准输入流 `System.in`

标准输出流 `System.out`

例如

```
System.out.println("Hello world!");
```



System.in是程序启动时由Java系统自动创建的流对象，它是原始的字节流，不能直接从中读取字符，需要对其进行进一步的处理
以System.in作为参数，进一步创建了一个InputStreamReader流对象，它相当于字节流和字符流之间的一座桥梁，读取字节并将其转换为字符
BufferedReader用于对InputStreamReader处理后的信息进行缓冲，以提高效率

```
import java.io.*;
public class testInput {
    public static void main(String[] args)
        throws IOException{
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));

        int age;
        float height;
        char sex;
        boolean isSick;
        String name;
        System.out.println("input your name"); //读取字符串
        name=in.readLine();//读取一个文本行
        System.out.println("input your age");
        age=Integer.parseInt(in.readLine()); //将输入字符串转换为整数
```

1.3.5 标准输入输出简介

```

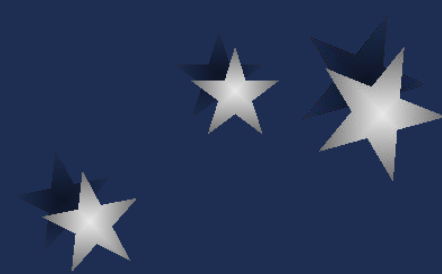
System.out.println("input your height");
height=Float.parseFloat(in.readLine()); //输入字符串转换为浮点
数
System.out.println("input weather you are sick");
isSick=Boolean.parseBoolean(in.readLine()); //输入字符串转换
为布尔数据
System.out.println("input your sex");
sex=(char)System.in.read (); //读取1个字节并转换为字符
System.out.println("your name is:"+name);
System.out.println("your age is:"+age);
System.out.println("your height is:"+height);
System.out.println("you are sick?" +isSick);
System.out.println("your sex is:"+sex);
}
}
    
```



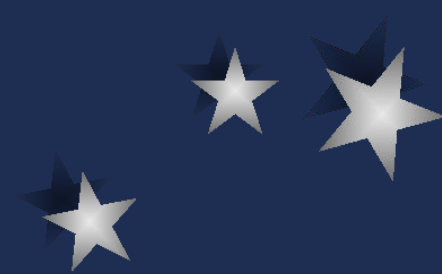
应用Scanner类实现输入输出

Scanner类：把给定的字符串（来源可以是输入流、文件、字符串）解析成Java的各种基本数据类型，用于分解字符串的默认的分隔符是空格，当然也可以定制

```
import java.util.Scanner;  
public class scanners {  
    public static void main(String[] args) {  
        int age;  
        float height;  
        char sex;  
        boolean isSick;  
        String name;  
  
        Scanner sc = new Scanner(System.in);
```



```
try{
    System.out.println("input your name");
    name=sc.nextLine(); //读取一个字符串
    System.out.println("input your age:");
    age=sc.nextInt(); //读取一个整数
    System.out.println("input your height:");
    height=sc.nextFloat(); //读取一个浮点数
    System.out.println("input weather you are sick:");
    isSick=sc.nextBoolean();//读取一个布尔值
    System.out.println("input your sex");
    sex=(char)System.in.read(); //读取一个字符
    System.out.println("name:"+name);
    System.out.println("age:"+age);
```



```
System.out.println("height:"+height);  
System.out.println("isMale:"+isSick);  
System.out.println("sex:"+sex);
```

```
/*应用Scanner读取一批整数*/
```

```
System.out.println("input numbers");  
while(sc.hasNextInt()){  
    System.out.println(sc.nextInt());
```

```
}
```

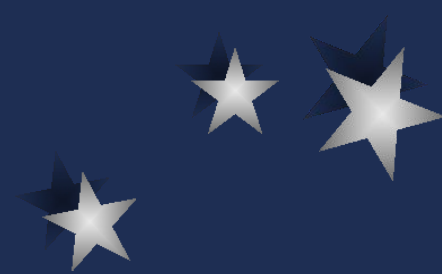
```
}
```

```
catch( Exception e){
```

```
    System.out.println(e.getMessage());
```

```
}
```

```
}
```



练一练

- 例子:

编写程序用于计算圆的面积;
接受用户的输入, 读取半径值;
利用下面的公式计算面积:

- $\text{Area} = \text{radius} * \text{radius} * 3.14159$

显示面积area。



请输入半径:

3.5

圆的面积是:

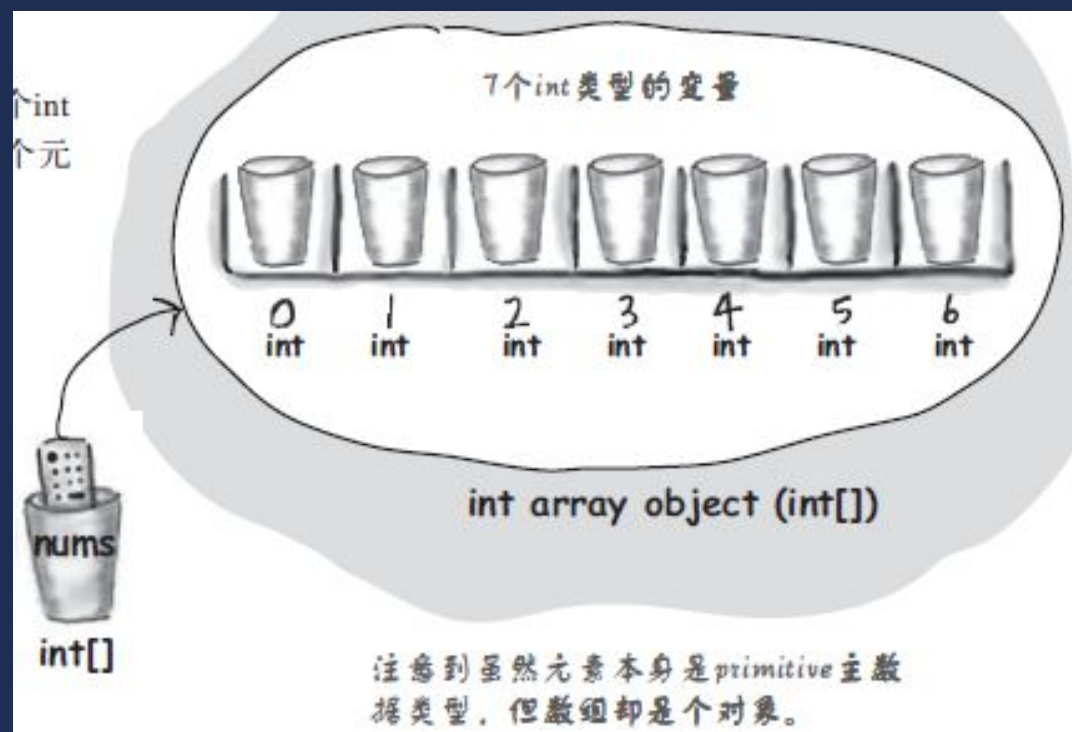
38.484478

```
import java.io.*;  
import java.lang.Float;
```

```
public class Area {  
    public static void main(String[] args)  
        throws IOException{  
        BufferedReader in= new BufferedReader(  
                                new InputStreamReader(System.in));  
        float itsRadius, itsArea;  
  
        System.out.println("请输入半径: ");  
        itsRadius = Float.parseFloat( in.readLine() );//获取半径  
        itsArea = itsRadius * itsRadius * 3.14159f; //计算面积  
        System.out.println("圆的面积是: "+ itsArea); //输出结果  
    }  
}
```

1.4 数组的概念

- 数组是对象。存放同一类型的一连串对象或者基本类型数据。



```
//1.声明数组变量
int[] nums;
//2.创建数组对象
nums = new int[7];
//3.保存数据到数组
nums[0]=6;
num[1]=7;
```

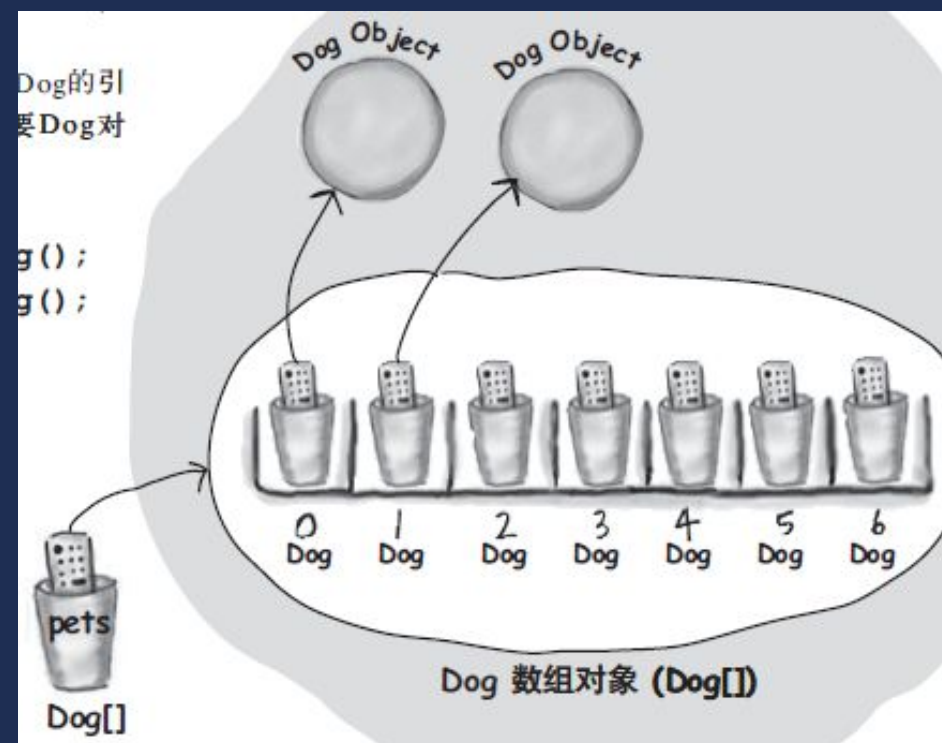


//1.声明数组变量

Dog[] pets;

//2.创建数组对象

pets = new Dog[7];



//3.创建dog对象并将
其引用保存到数组

pets[0] = new Dog();
pets[1] = new Dog();

1.5 数组的创建和引用

- 数组的声明
- 数组的创建
- 数组元素的初始化
- 数组的引用
- 多维数组



1.5.1 数组的声明

数组的创建和引用

- 声明 (Declaration)

声明数组时无需指明数组元素的个数，也不为数组元素分配内存空间

```
Type[] arrayName;
```

例如: int[] intArray;
 String[] stringArray;

```
Type arrayName[ ];
```

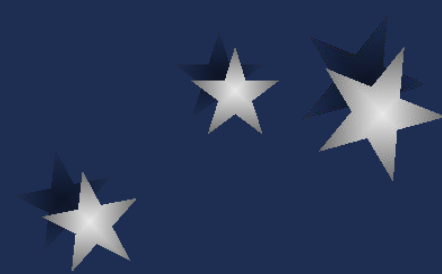
例如: int intArray[];
 String stringArray[];

不能直接使用，必须经过初始化分配内存后才能使用

1.5.2 数组的创建

数组的创建和引用

- 用关键字new构成数组的创建表达式，可以指定数组的类型和数组元素的个数。元素个数可以是常量也可以是变量
- 基本类型数组的每个元素都是一个基本类型的变量；引用类型数组的每个元素都是对象的引用



1.5.2 数组的创建(续)

数
组
的
创
建
和
引
用

```
arrayName=new Type[componets number];
```

例如:

```
int[] ai; ai=new int[10];
```

```
String[] s; s=new String[3];
```

或者可以将数组的声明和创建一并执行

```
int ai[]=new int[10];
```

可以在一条声明语句中创建多个数组

```
String[] s1=new String[3], s2=new String[8];
```



1.5.3 数组元素的初始化

数组的创建和引用

- 声明数组名时，给出了数组的初始值，程序便会利用数组初始值创建数组并对它的各个元素进行初始化

```
int a[]={22, 33, 44, 55};
```

- 创建数组时，如果没有指定初始值，数组便被赋予默认值初始值。

基本类型数值数据，默认的初始值为0;

boolean类型数据，默认值为false;

引用类型元素的默认值为null。

- 程序也可以在数组被构造之后改变数组元素值

1.5.4 数组的引用

数组的创建和引用

- 通过下面的表达式引用数组的一个元素：
`arrayName[index]`
- 数组下标必须是 `int` , `short`, `byte`, 或者 `char`.
- 下标从零开始计数.
- 元素的个数即为数组的长度，可以通过 `arrayName.length` 引用
- 元素下标最大值为 `length - 1`，如果超过最大值，将会产生数组越界异常
(`ArrayIndexOutOfBoundsException`)

1.5.4 数组的引用(续)

数组的创建和引用

```
int[] data = new int[10];
```

- `data[-1]` 非法的
- `data[10]` 非法的
- `data[1.5]` 非法的
- `data[0]` 合法的
- `data[9]` 合法的



1.5.4 数组的引用(续)

数组的创建和引用

```
public class MyArray {
    public static void main(String[] args){
        int myArray[];           //声明数组
        myArray=new int[10];      //创建数组
        System.out.println("Index\t\tValue");
        for(int i=0; i<myArray.length;i++)
            System.out.println(i+"\t\t"+myArray[i]);
        //证明数组元素默认初始化为0
        //myArray[10]=100;        //将产生数组越界异常
    }
}
```



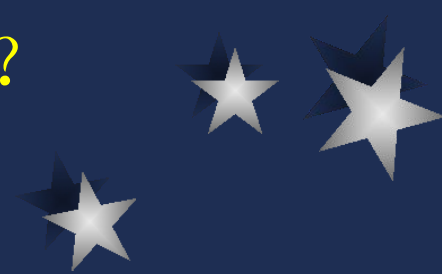
1.5.4 数组的引用(续)

数组的创建和引用

```
class Gauss
{ public static void main(String[] args)
  { int[ ] ia = new int[101];
    for (int i = 0; i < ia.length; i++)
      ia[i] = i;
    int sum = 0;
    for (int i = 0; i < ia.length; i++)
      sum += ia[i];
    System.out.println(sum);
  }
}
```

输出:
5050

动动脑：这段程序的功能？



1.5.4 数组的引用(续)

- 数组名是一个引用：
例子

```
public class Arrays
{
    public static void main(String[] args)
    {
        int[] a1 = { 1, 2, 3, 4, 5 };
        int[] a2;
        a2 = a1;
        for(int i = 0; i < a2.length; i++)
            a2[i]++;
        for(int i = 0; i < a1.length; i++)
            System.out.println( "a1[" + i + "]"
                               = " + a1[i]);
    }
}
```

1.5.4 数组的引用(续)

数组的创建和引用

运行结果:

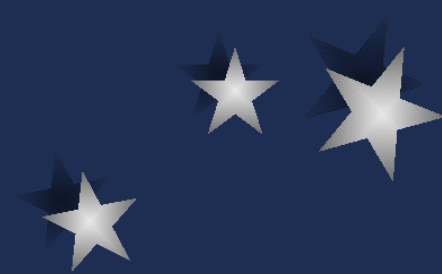
`a1[0] = 2`

`a1[1] = 3`

`a1[2] = 4`

`a1[3] = 5`

`a1[4] = 6`



1.5.4 数组的引用(续)

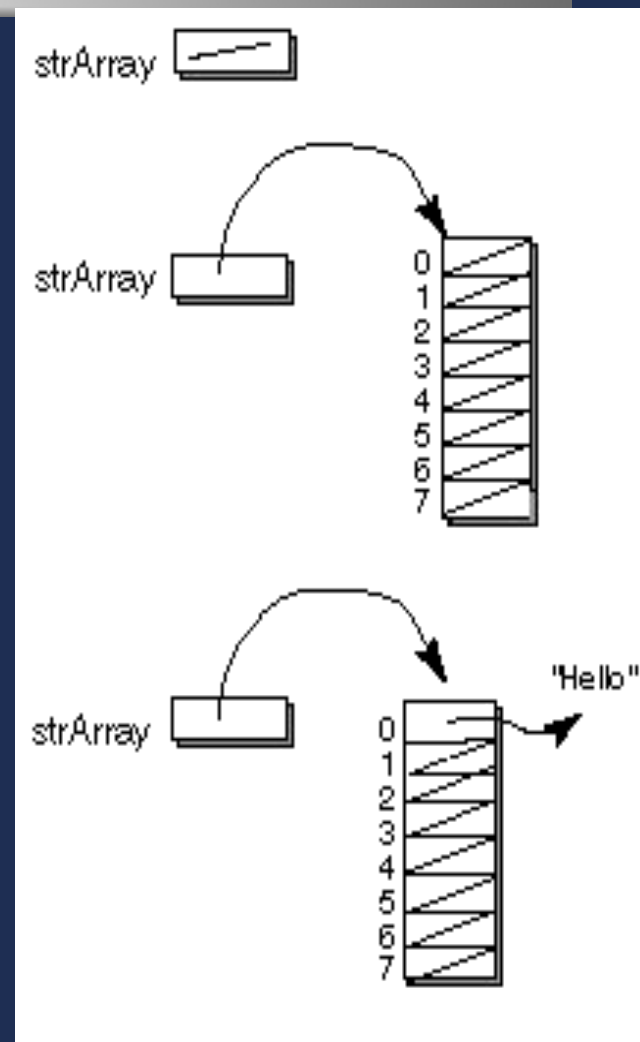
数组的创建和引用

- 字符串引用构成的数组:

```
String[] strArray;
```

```
strArray = new String[8];
```

```
strArray[0]= "Hello" ;
```



1.5.4 数组的引用(续)

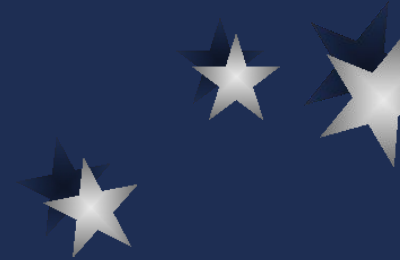
数组的创建和引用

● 例子

```
public class ArrayOfStringsDemo
{ public static void main(String[] args)
    { String[] anArray =
        { "String One", "String Two", "String
Three"};
        for (int i = 0; i < anArray.length; i++)

        { System.out.println(anArray[i].toLowerCase());
          }
    }
}
```

运行结果:
string one
string two
string three

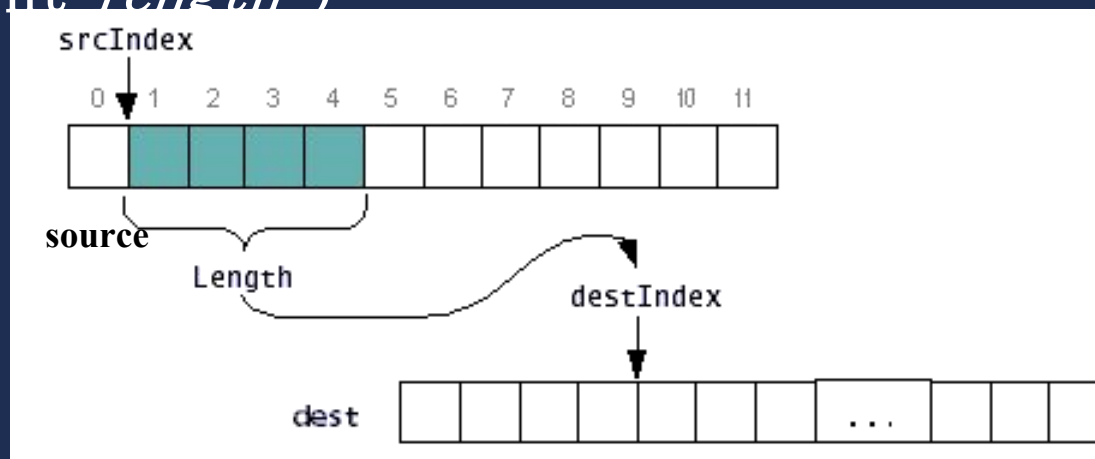


1.5.4 数组的引用(续)

数组的创建和引用

- 数组的复制:

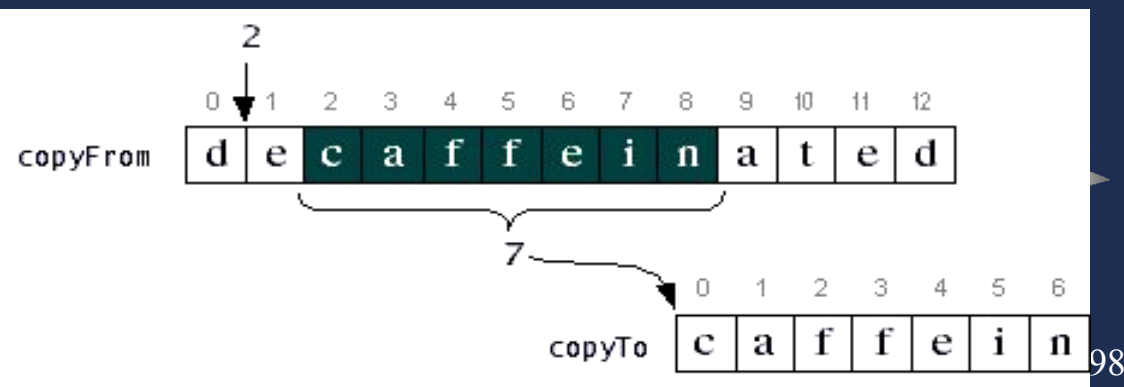
```
public static void arraycopy(Object source ,
int srcIndex , Object dest , int destIndex ,
int length )
```



1.5.4 数组的引用(续)

例子

```
public class ArrayCopyDemo
{ public static void main(String[] args)
  { char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd',
    'f', 'f', 'e', 'i', 'n', 'a', 't', 'e',
    'd' };
    char[] copyTo = new char[7];
    System.arraycopy(copyFrom, 2, copyTo, 0,
7);
    System.out.
  }
}
```



1.5.5 多维数组

数组的创建和引用

```
int[][] gradeTable;  
.....
```

```
gradeTable[ 0 ][ 1 ]  
    为42
```

```
gradeTable[ 3 ][ 4 ]  
    为93
```

```
gradeTable[ 6 ][ 2 ]  
    为78
```

Student	Week				
	0	1	2	3	4
0	99	42	74	83	100
1	90	91	72	88	95
2	88	61	74	89	96
3	61	89	82	98	93
4	93	73	75	78	99
5	50	65	92	87	94
6	43	98	78	56	99

1.5.5 多维数组(续)

数组的创建和引用

● 二维数组的声明和构造

```
int[ ][ ] myArray ;
```

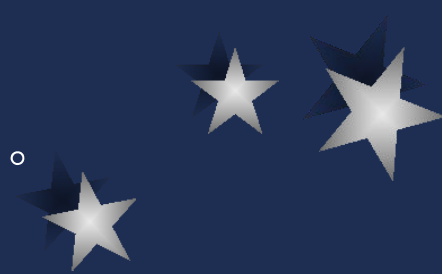
- myArray 可以存储一个指向2维整数数组的引用。其初始值为null。

```
int[ ][ ] myArray = new int[3][5] ;
```

- 建立一个数组对象，把引用存储到myArray。这个数组所有元素的初始值为零。

```
int[ ][ ] myArray = { {8,1,2,2,9},  
                      {1,9,4,0,3}, {0,3,0,0,7} };
```

- 建立一个数组并为每一个元素赋值。



1.5.5 多维数组(续)

数组的创建和引用

- 二维数组的长度

```
class UnevenExample2
{ public static void main( String[ ] arg )
    { int[ ][ ] uneven =
        { { 1, 9, 4 },
          { 0, 2},
          { 0, 1, 2, 3, 4 } };
      System.out.println("Length is: " +
        uneven.length );
    }
}
```

运行结果：
Length is: 3

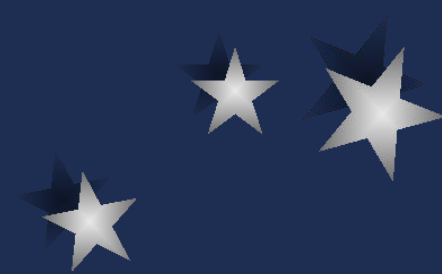


1.5.5 多维数组(续)

数组的创建和引用

- 每行的长度:

```
class UnevenExample3
{
    public static void main( String[] arg )
    {
        // 声明并构造一个2维数组
        int[ ][ ] uneven =
            { { 1, 9, 4 },
              { 0, 2 },
              { 0, 1, 2, 3, 4 } };
    }
}
```



1.5.5 多维数组(续)

数组的创建和引用

// 数组的长度 (行数)

```
System.out.println("Length of array is: " +
    uneven.length );
```

// 数组每一行的长度 (列数)

```
System.out.println("Length of row[0] is: " +
    uneven[0].length );
```

```
System.out.println("Length of row[1] is: " +
    uneven[1].length );
```

```
System.out.println("Length of row[2] is: " +
    uneven[2].length );
```

```
}
}
```

运行结果:

Length of array is: 3

Length of row[0] is: 3

Length of row[1] is: 2

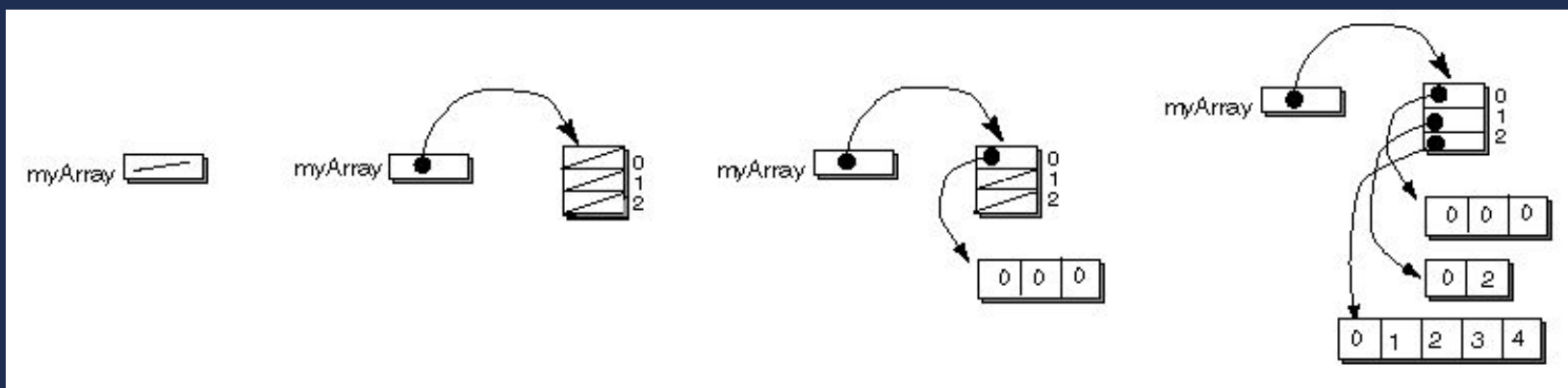
Length of row[2] is: 5

1.5.5 多维数组(续)

数组的创建和引用

```
int[ ][ ] myArray;  
myArray = new int[3][ ];  
myArray[0] = new int[3];
```

```
int[ ] x = {0, 2};  
int[ ] y = {0, 1, 2, 3, 4};  
myArray[1] = x;  
myArray[2] = y;
```



1.6 本章小结

- 本章内容

 - Java开发环境

 - Java语言的特点

 - 基础语法

- 复习要求

 - 下载、安装J2se

 - 熟悉命令行方式编译、运行Java程序

 - 熟悉一种集成开发环境

