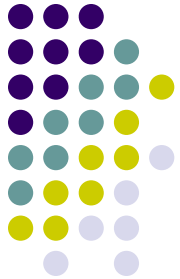
Java 语言程序设计

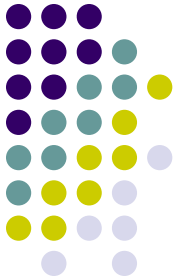
第一章
Java 语言基础知识



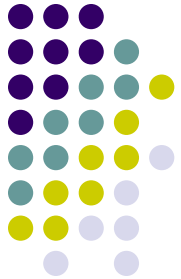
参考:

- CSDN,JAVA.CN,JAVA开源等网站
- Java.sun.com; http://www.oracle.com
- http://www.netbeans.org
- javaAPI_html_CN
- Java编程思想 中文第三版

….

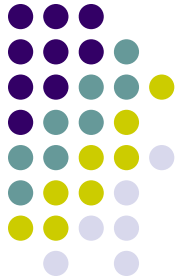


目录



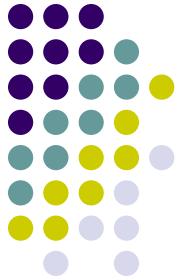
1.1 Java语言与面向对象的程序设计

- 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。



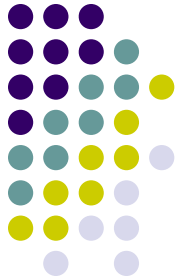
1.1.1 面向对象的程序设计思想

- 计算机程序设计
  - 对问题进行抽象
  - 用计算机语言表述，利用机器求解



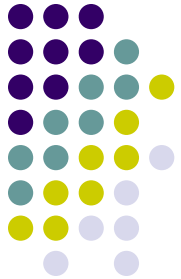
1.1.1 面向对象的程序设计思想(续)

- 程序设计语言发展的历程
  - 机器语言
  - 汇编语言
  - 高级语言
  - 面向对象的语言



1.1.1 面向对象的程序设计思想(续)

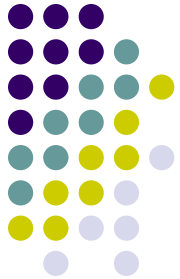
- 面向对象的思想
  - 将客观事物看作具有状态和行为的对象，通过抽象找出同一类对象的共同状态和行为，构成类。



1.1.1 面向对象的程序设计思想(续)

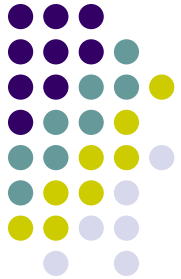
- 例：
  - 构建一个汽车类，需要提取所有汽车对象的共有的状态和行为。将状态用变量表示，行为用方法表示。

```
class Car {
    int color_number;
    int door_number;
    int speed;
    ……
    void brake() { … }
    void speedUp() {…};
    void slowDown() { …  };
    ……
}
```



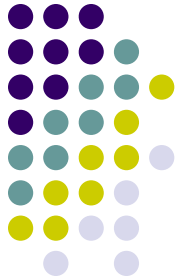
1.1.1 面向对象的程序设计思想(续)

- 面向对象技术给软件发展带来的益处
  - 可重用性
  - 可靠性
- 面向对象语言的基本特征
  - 抽象和封装
  - 继承性
  - 多态性



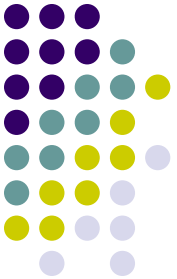
1.1.2 Java语言的特点

- 面向对象
- 安全性
  - Java不支持指针
  - Java的内部安全措施
- 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



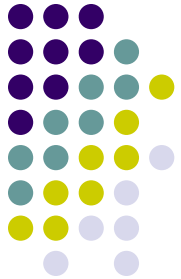
1.1.2 Java语言的特点(续)

- Java 语言的优点
  - 易于学习
  - 代码效率高
  - 代码质量高
  - 开发程序快
  - 体系结构中立，纯Java程序不依赖于平台
  - 一处编写，各处运行
  - 软件易于发布



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。(API)
- 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。

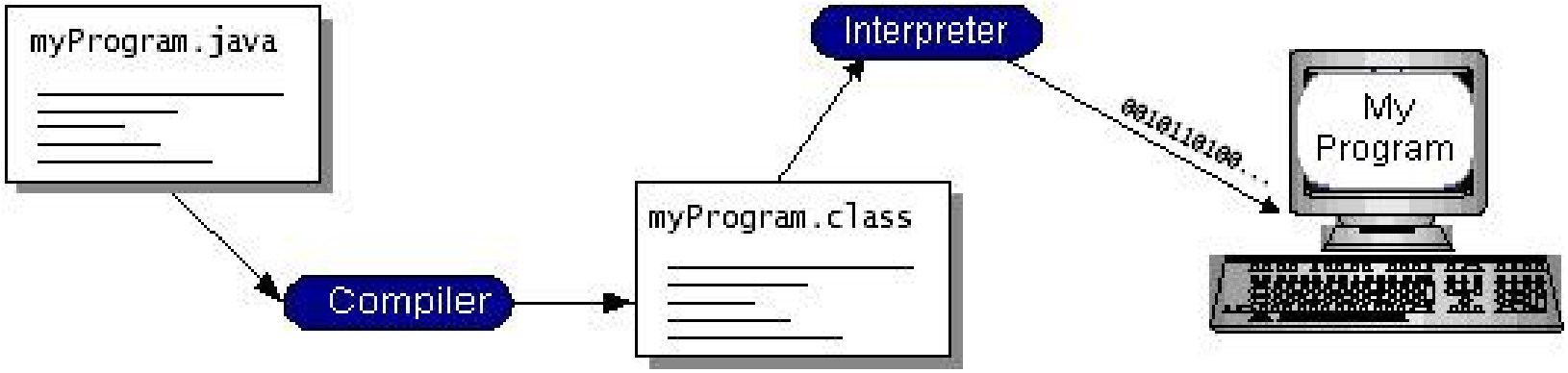


1.2 Java程序概述

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

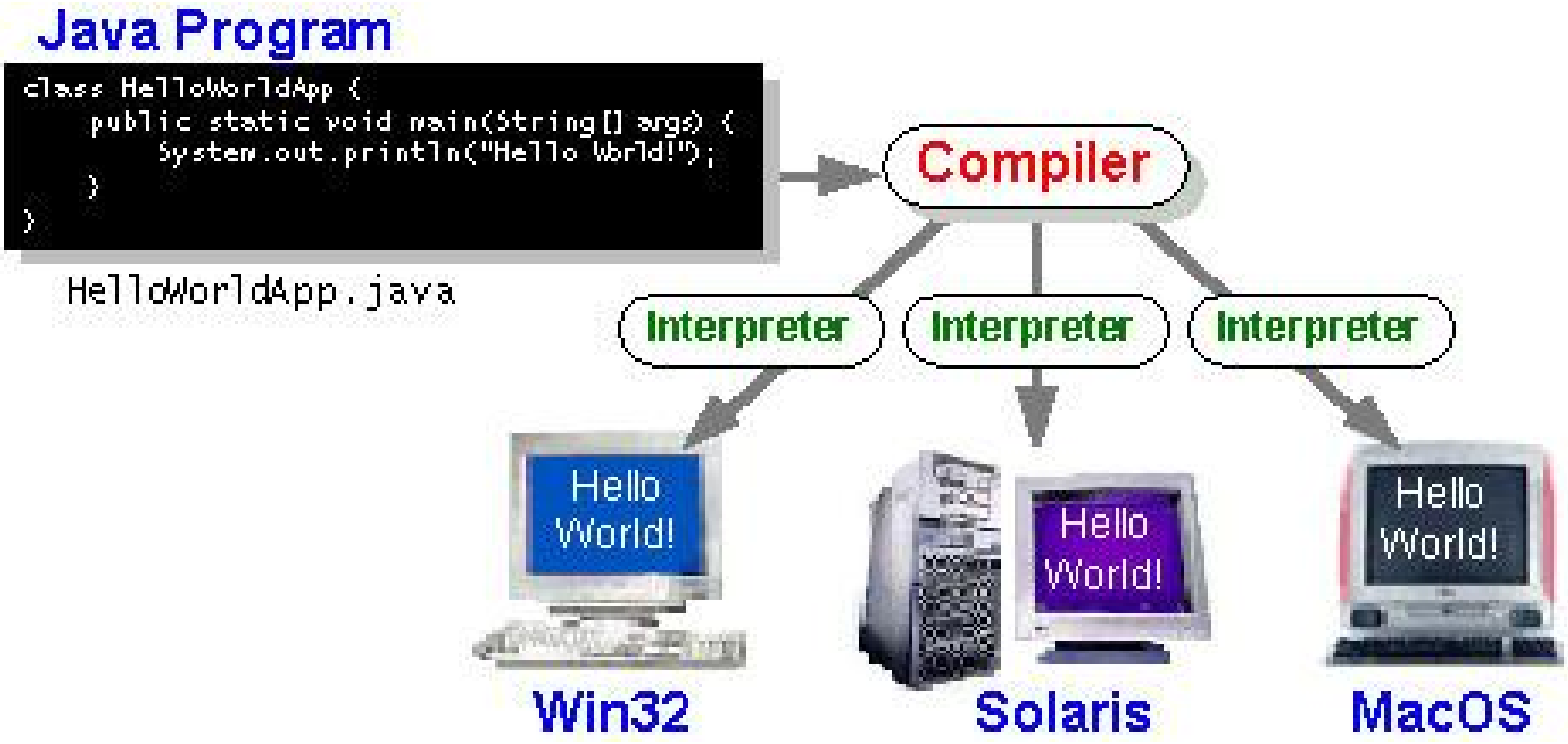
1.2.1 Java开发环境

Java程序编译执行的过程



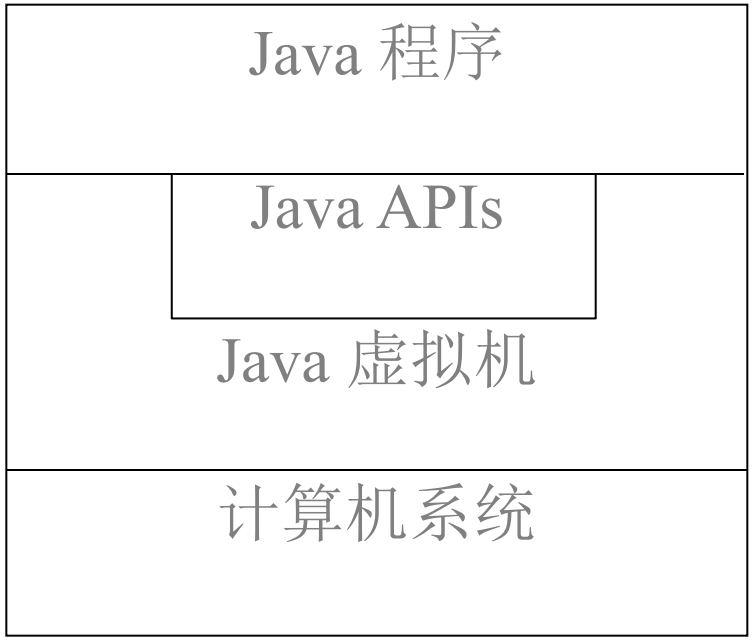
1.2.1 Java开发环境(续)

一次编写，各处运行

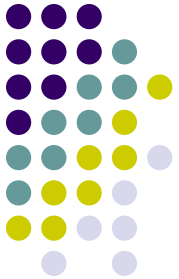


1.2.1 Java开发环境 (续)

Java 平台



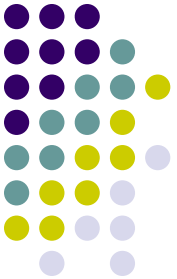
- Java APIs (应用程序接口)
  - 经过编译的，可在程序中使用的Java代码标准库。
- Java VM（虚拟机）--能运行Java代码的假想机器.
  - Java 程序由Java虚拟机程序执行（或解释执行）。



1.2.1 Java开发环境(续)

Java2 SDK(Software Development Kit)

- Standard Edition (J2SE)
- Enterprise Edition (J2EE)
- Java 2 Micro Edition (J2ME)小家电手机

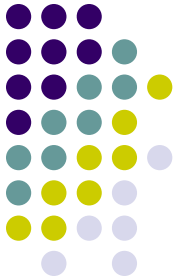


1.2.1 Java开发环境(续)

J2SE——J2SDK的核心部分

- 开发工具
  - 编译器
  - 调试器
  - 文档制作工具
- 运行环境
  - Java 虚拟机
  - 组成Java 2 平台API的类。
  - 帮助文档
- 附加库
- Java程序（Applets 和 Applications） 的演示

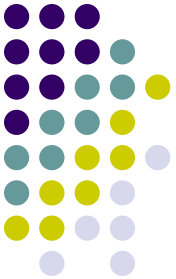
……



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。
- java：
  - Java解释器，执行已经转换成Bytecode的java应用程序。
- jdb：
  - Java调试器， 用来调试java程序。
- javap：
  - 反编译，将类文件还原回方法和变量。
- javadoc：
  - 文档生成器,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。



1.2.1 Java开发环境(续)

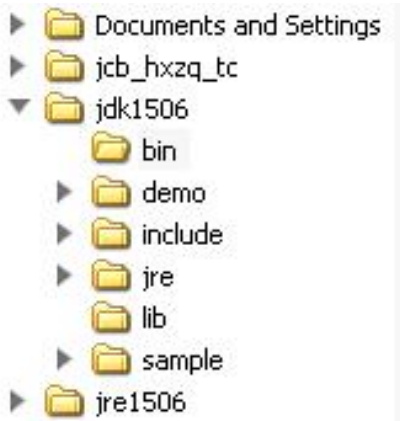
环境安装——以j2sdk1.4.0为例

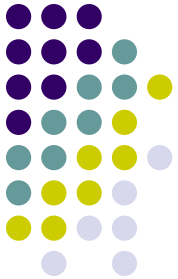
- 下载地址
  - http://java.sun.com
- 下载文件
  - j2sdk-1_4_0-win.exe
  - j2sdk-1_4_0-doc.zip
- 安装
  - 直接运行“j2sdk-1_4_0-win.exe”。
- 文档
  - 解开“j2sdk-1_4_0-doc.zip”。

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
- \demo目录：一些实例程序
- \lib目录：Java开发类库
- \jre目录： Java运行环境，包括Java虚拟机、运行类库等
- …

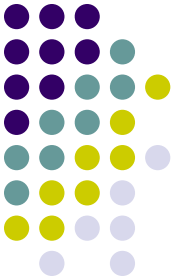




1.2.1 Java开发环境(续)

几种集成开发环境

- netBeans
- Jcreator PRO
- Borland JBuilder
- MyEclipse
- Microsoft Visual J++
- IBM : Visual Age for Java
- Sun ONE Studio



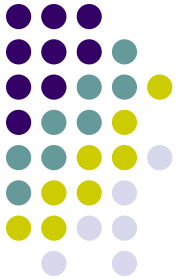
1.2.2 Application举例

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 Application举例(续)

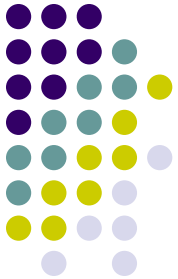
——例1-1



```
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 Application举例(续)

——例1-1 运行结果

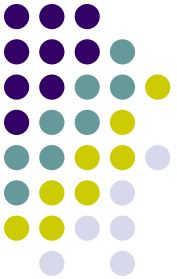


使用如下命令编译并运行程序：

```
javac   MyClass.java
java  MyClass
```

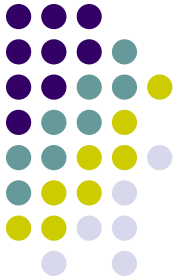
运行结果如下：

```
The sum is: 3
```



1.2.3 Applet举例

- Applet——小应用程序
  - 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和机器匹配的指令，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

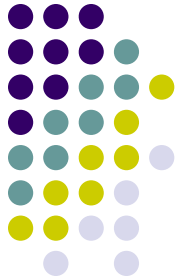


1.2.3 Applet举例(续)

- Applet的优点
  - Web 浏览器软件包括很多小应用程序运行所需的功能
- Applet的局限性
  - 在客户端主机的文件系统中读/写受限
  - 不能运行客户端主机的任何程序
  - 仅能在服务器和客户端之间建立联系

1.2.3 Applet举例(续)

——例1-2



JAVA Applet:

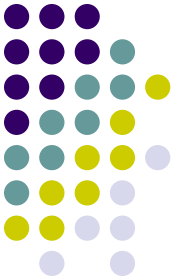
```
import java.awt.Graphics;
import java.applet.Applet;
public class MyApplet extends Applet
{
    public String s;
    public void init()
    {    s=new String("Hello World !");   }
    public void paint(Graphics g)
    {    g.drawString(s,25,25);   }
}
```

HTML:

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

1.2.3 Applet举例(续)

——例1-2注释

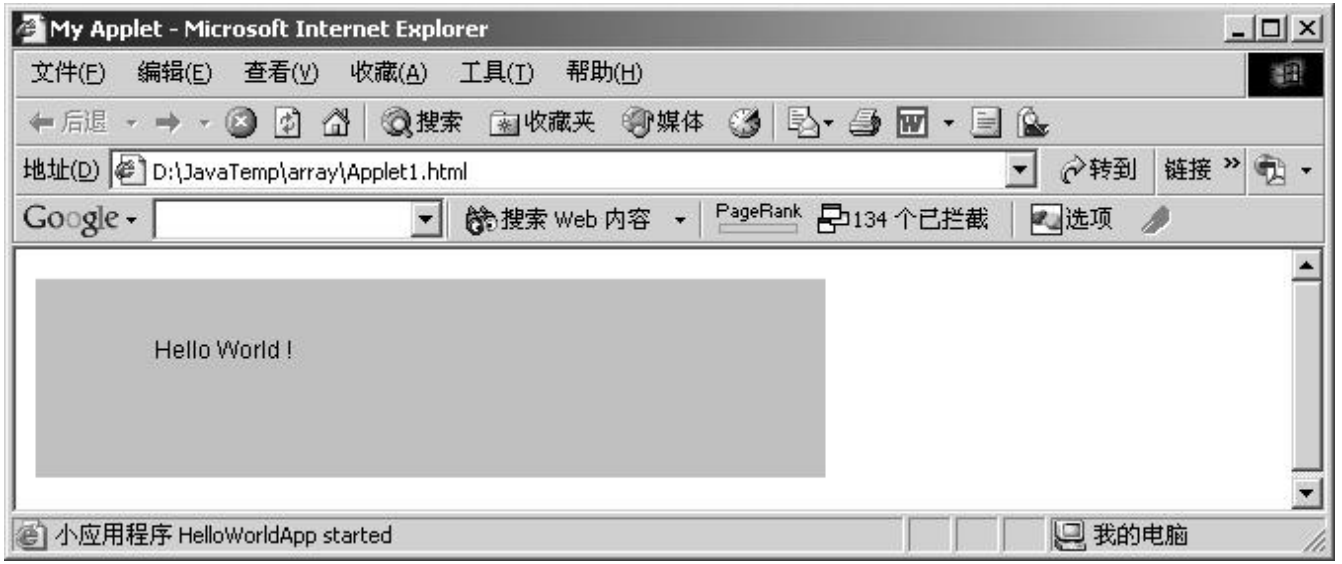


- Graphics类
  - 使得applet绘制直线、矩形、椭圆形、字符串等
- 方法init()
  - 初始化，实现了字符串的创建
- 方法paint() 中
  - g为Graphics类的对象。调用了Graphics的drawString方法绘制字符串。
  - 此方法执行的结果就是从坐标(60,40)开始绘制出字符串Hello World! 。

1.2.3 Applet举例(续)

——例1-2运行

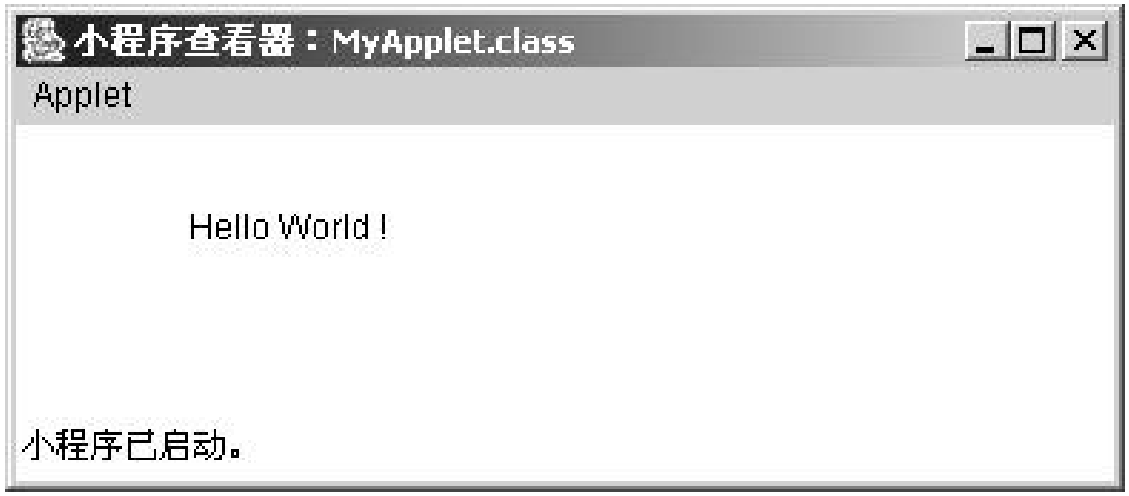
- 用支持Java的浏览器，比如IE6.0，打开Applet1.html

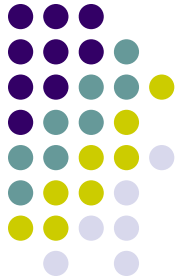


1.2.3 Applet举例(续)

——例1-2运行

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





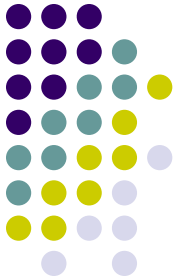
1.2.4 Servlet举例

Servlet

- 运行在服务器端，响应客户端请求，扩展了服务器的功能
- 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

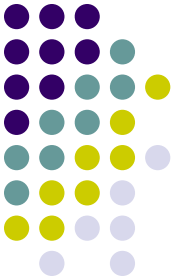


程序首先构建HttpServletRequest，并建立一个数据表单；点击submit按钮后，servlet再次被调用，并产生一个含有表单的网页。

```
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)

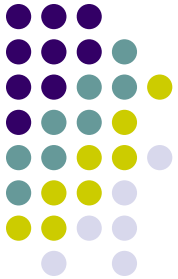
——例1-3



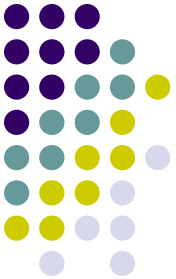
```
if(!flds.hasMoreElements())
{
  out.print("<html>");
  out.print("<form method=\"POST\""
                        +"action=\"EchoForm\">");
  for(int i = 0; i < 10; i++)
    out.print("<b>Field" + i + "</b> " +
      "<input type=\"text\""+" size=\"20\ “name=\"Field"
            + i + "\" value=\"Value"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

1.2.5 Servlet举例(续)

——例1-3

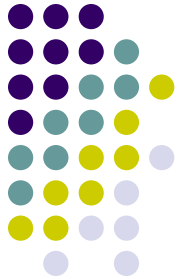


```
else
    {
      out.print("<h1>Your form contained:</h1>");
      while(flds.hasMoreElements())
      {
        String field= (String)flds.nextElement();
        String value= req.getParameter(field);
        out.print(field + " = " + value+ "<br>");
      }
    }
    out.close();
  }
}
```



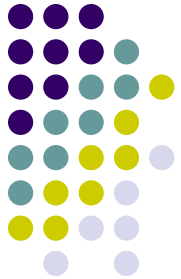
1.2.6 JSP举例

- 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



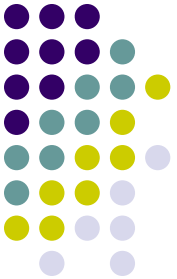
1.2.6 JSP举例(续)

```
<html>
<body>
<H1>The time in second is:
<%= System.currentTimeMillis()/1000 %>
</H1>
</body>
</html>
```



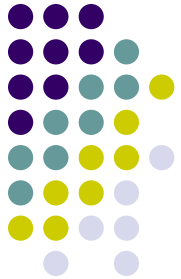
1.3 基本数据类型与表达式

- 变量与常量
- 基本数据类型
- 表达式与运算符
- 类型转换



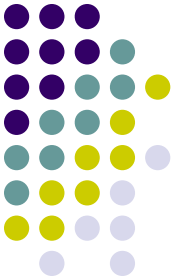
1.3.1 变量与常量

- 变量
  - 一个由标识符命名的项
  - 每个变量都有类型, 例如 int 类型或 Object类型, 变量还有作用域.
  - 变量的值可以被改变.
- 常量
  - 常量一旦被初始化以后就不可改变。



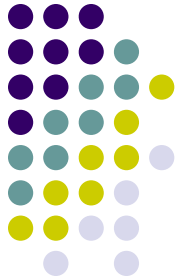
1.3.1 变量与常量(续)

- 标识符
  - 标识符是一个名称，与内存中的某个位置（地址）相对应
  - 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)



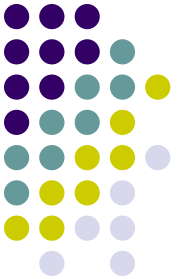
1.3.2 基本数据类型

- 整数
  - byte 8 bits -128 ~ +127
  - short 16 bits -32768~ + 32767
  - int 32 bits -2^{31}~$(2^{31}-1)$
  - long 64 bits -2^{63}~$(2^{63}-1)$
  - char 16 bits 0 ~ 65535
- ***boolean 8 bits true,false***



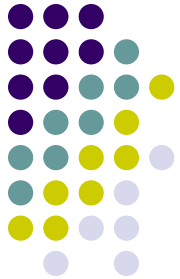
1.3.2 基本数据类型(续)

- 整数运算
  - 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 +



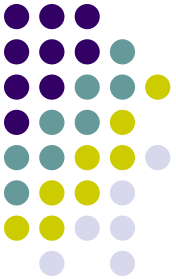
1.3.2 基本数据类型(续)

- 浮点数
  - float
    - 单精度浮点数
    - 32-bit
    - $-m \cdot 2^e \sim m \cdot 2^e$
      - 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-bit
    - $-m \cdot 2^e \sim m \cdot 2^e$
      - 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)

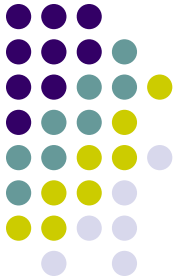
- 浮点运算
  - 比较运算符（关系运算符）
    - 算术比较运算符 <, <=, >, and >=
    - 算术相等比较运算符 == and !=
  - 算术运算符
    - 一元运算符 + 和 −
    - +,- *, /, 和 % （取余 ）
    - 自增/自减运算符 ++/−−
    - 移位运算符 <<, >>, and >>>
    - 位运算符 ~, &, |, and ^
  - 条件运算符？ ：
  - 类型转换运算符
  - 字符串连接运算符 +



1.3.2 基本数据类型(续)

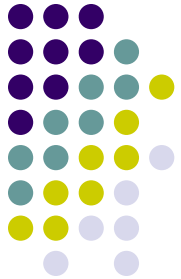
- 布尔类型和布尔值
  - 布尔类型表示一个逻辑量， 有两个取值：true和false
  - 例如:

```
boolean is_salaried;
boolean is_hourly;
is_salaried = true; //将 is_salaried设置为true
is_hourly = false; //将is_hourly设置为false
```



1.3.2 基本数据类型(续)

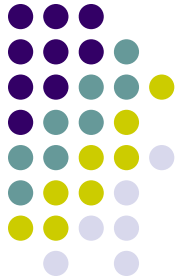
- 布尔运算符
  - 关系运算符 == and !=
  - 逻辑“非”运算符 !
  - 逻辑运算符 &, ^, 和 |
  - 条件“与” 和条件“或”运算符 && 和 ||
  - 条件运算符 ? :
  - 字符串连接运算符 +



1.3.2 基本数据类型(续)

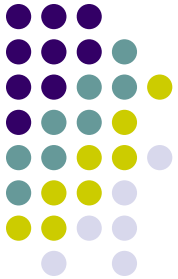
- String——字符串
  - String 是一个类
  - String类JDK标准类集合中的一部分

```
String animal = "walrus";
```



1.3.2 基本数据类型(续)

- 文字量(直接量)
  - 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
      如： 0xff
    - 八进制
      如： 0377

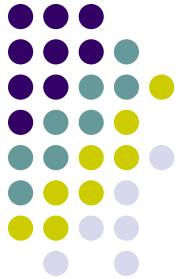


1.3.2 基本数据类型(续)

- 浮点文字量
  - 一个浮点文字量包括以下几个部分
    - 整数部分
    - 小数点
    - 小数部分
    - 指数 (e or E)
    - 类型后缀 (f or F for float, d or D for double)
  - float 类型文字量举例：

    1e 1f 2.f .3f 0f 3.1 4f
    6.022137e+23f
  - double 类型文字量举例：

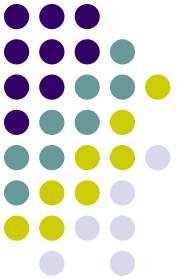
    1e1 2. .3 0.0 3.1 4 1e-9d
    1e137
- 布尔文字量
  - 布尔类型只有两个值，由文字量 true 和 false表示



1.3.2 基本数据类型(续)

- 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 \

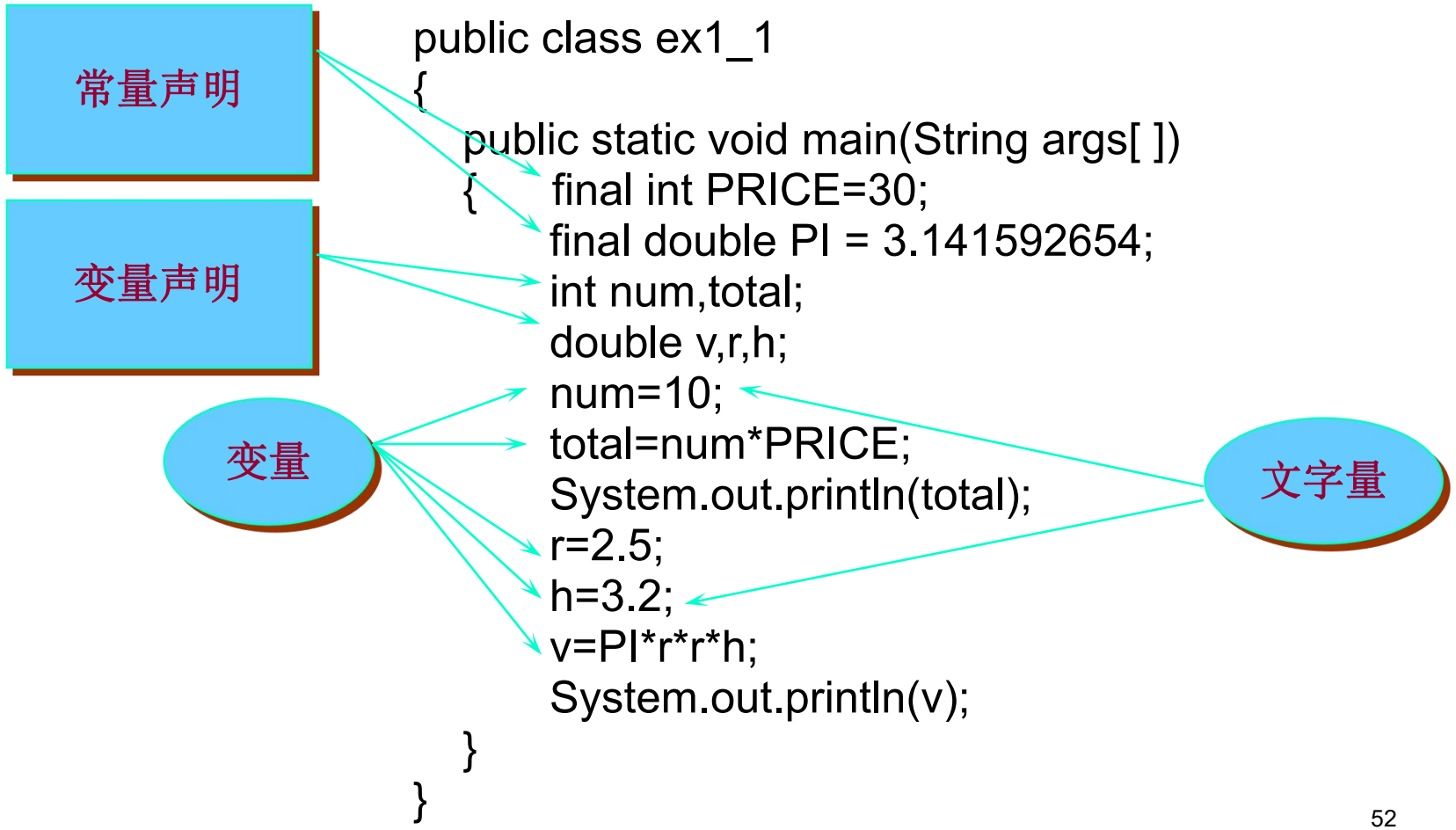


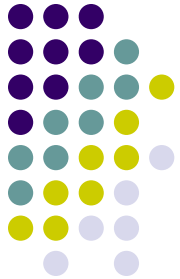
1.3.2 基本数据类型(续)

- 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
```

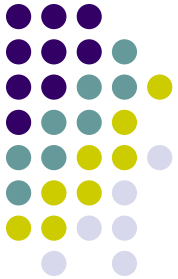
1.3.2 基本数据类型(续)


常量声明
变量声明
变量
文字量
public class ex1_1
{
public static void main(String args[ ])
{
final int PRICE=30;
final double PI = 3.141592654;
int num,total;
double v,r,h;
num=10;
total=num*PRICE;
System.out.println(total);
r=2.5;
h=3.2;
v=PI*r*r*h;
System.out.println(v);
}
}



1.3.3 表达式与运算符

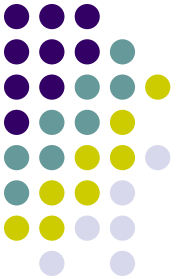
- 表达式是由一系列变量、运算符、方法调用构成的，表达式可以计算出一个值来
- 程序中的很多工作是通过计算表达式的值来完成的。
  - 有时需要的是表达式的副作用，例如赋值表达式将数值赋给变量
  - 更多时候起作用的是表达式的值，这个值可以用作方法的参数，或更大的表达式的操作数，或者影响语句的执行顺序



1.3.3 表达式与运算符(续)

- 算术运算符
  - 运算符 ++ 和 --

    例如： i++; --j;
  - 一元运算符 + 和 –
  - 加法运算符 + 和 -
  - 乘法运算符 *, /, 和 %



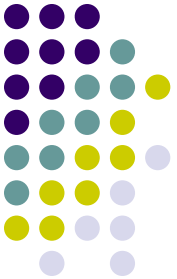
1.3.3 表达式与运算符(续)

- 赋值运算符
  - 简单赋值运算符 =
  - 复合赋值运算符

    *=　　/=　　%=　　+=　　-=　　<<=　　>>=　　>>>=

    &=　　^=　　|=

    E1 op= E2 等效于 *E1* = (*T*) ((*E1*) *op* (*E2*)), 其中*T* 是*E1*的类型



1.3.3 表达式与运算符(续)

- 举例

  a=5表达式的值为 5

  a=b=c=5　表达式的值以及 a,b,c 的值都是 5

  a=5+(c=6) 表达式的值是 11, a 是 11, c 是 6

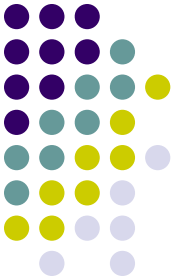
  a=(b=4)+(c=6) 表达式的值是 10，

  a是10，b是4，c是6

  a=(b=10)/(c=2)　表达式的值是 5，

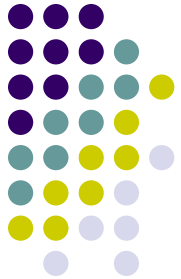
  a是5，b是10，c是2

  a+=a-=a*a 等效于 a=a+(a=a-a*a)



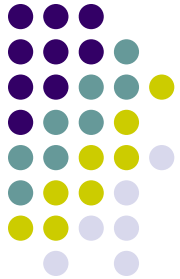
1.3.3 表达式与运算符(续)

- 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



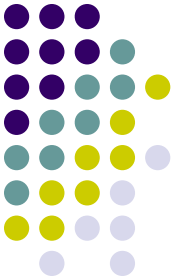
1.3.3 表达式与运算符(续)

- 相等关系运算符
  - 数字相等运算符 ==， !=
  - 布尔相等运算符 ==， !=
  - 引用相等运算符 ==， !=



1.3.3 表达式与运算符(续)

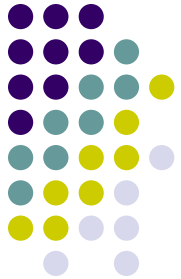
- 逻辑运算符
  - “与”运算 &&
    - 如果两个操作数的值都为true运算结果为true; 否则，结果为false.
  - “或”运算 ||
    - 如果两个操作数的值都为false运算结果为false;否则，结果true
  - “非”运算符！
    - 操作数的类型必须是布尔类型
    - 如果操作数的结果为 false，则表达式的结果为 true ，如果操作数的结果为 true则表达式的结果为 false



1.3.3 表达式与运算符(续)

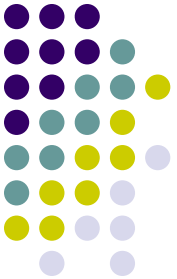
条件运算符 (表达式1？表达式2：表达式3)

- 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



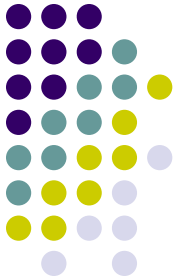
1.3.4 类型转换

- 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



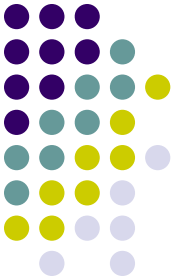
1.3.4 类型转换(续)

- 扩展转换
  - byte, short, int, long, float, double
    char
  - 从一种整数类型到另一种整数类型，或者从float到double的转换不损失任何信息
  - 从整数类形向float或double转换，会损失精度
- 窄化转换
  - double, float, long, int, short, byte,char
  - 窄化转换可能会丢失信息



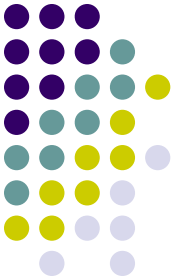
1.3.4 类型转换(续)

- 字符串转换
  - 任何类型（包括null类型）都可以转换为字符串类型



1.3.4 类型转换(续)

- 赋值转换
  - 将表达式类型转换为制定变量的类型
- 方法调用转换
  - 适用于方法或构造方法调用中的每一个参数
- 强制转换
  - 将一个表达式转换为指定的类型
  - 例如 (float)5.0
- 字符串转换
  - 只当一个操作数是String类型时，适用于+运算符的操作数



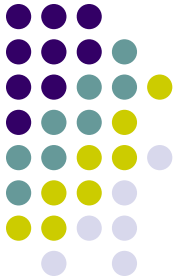
1.3.4 类型转换(续)

- 数字提升

  将算术运算符的操作数转换为共同类型

  - **一元数字提升**
    - 如果一个操作数是 byte, short, 或 char类型, 一元数字提升通过扩展转换将它转换为int类型
  - **二元数字提升**
    - 二元数字提升作用在特定操作符的操作数上

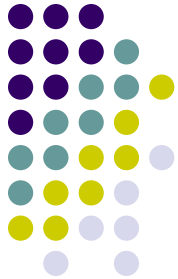
      *, /, %, +, -, <, <=, >, >=, ==, !=, &, ^, | and?:
    - 在必要时使用扩展转换来转换操作数类型



1.3.4 类型转换(续)

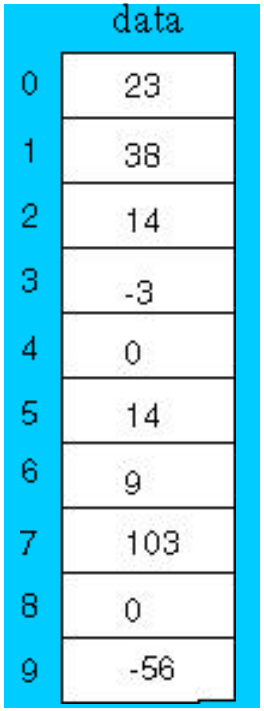
- 标准输入输出简介
  - 标准输入流 System.in
  - 标准输出流 System.out
  - 例如

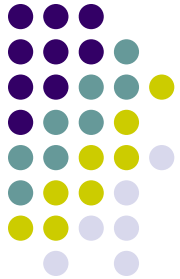
    System.out.println("Hello world!");



1.4 数组的概念

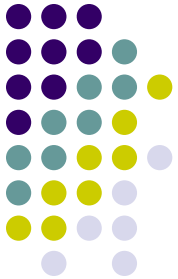
- 数组由同一类型的一连串对象或者基本数据组成，并封装在同一个标识符（数组名称）下。
- 数组是对象
  - 动态初始化
  - 可以赋值给Object类型的变量
  - 在数组中可以调用类Object 的所有方法





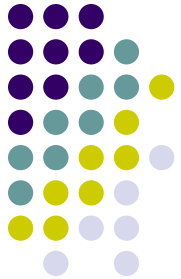
1.4 数组的概念(续)

- 数组元素
  - 数组中的变量被称作数组的元素
  - 元素没有名字，通过数组名字和非负整数下标值引用数组元素。
  - 每个数组都有一个由 public final 修饰的成员变量：*length*，即数组含有元素的个数（*length*可以是正数或零）



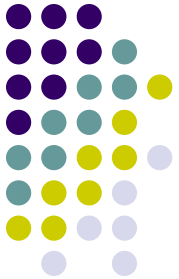
1.5 数组的创建和引用

- 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

- 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
  - 不能直接使用，必须经过初始化分配内存后才能使用



1.5.1 数组的声明(续)

Type[ ] arrayName;

例如：

int[] intArray;

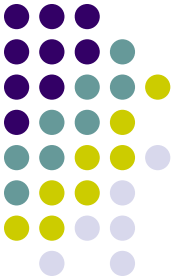
String[] stringArray;

Type arrayName[ ];

例如：

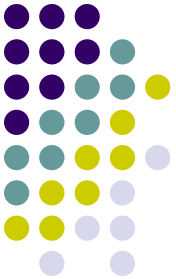
int intArray[];

String stringArray[];



1.5.2 数组的创建

- 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的引用



1.5.2 数组的创建(续)

arryName=new Type[componets number];

- 例如：

  int[] ai;　ai=new int[10];

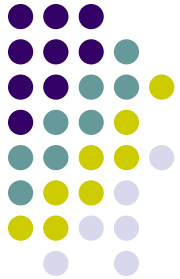
  String[] s;　s=new String[3];

- 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];

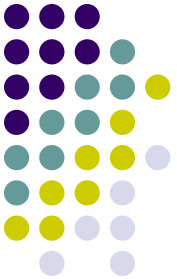


1.5.3 数组元素的初始化

- 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

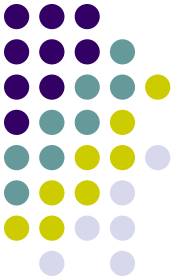
  int a[]={22, 33, 44, 55};

- 创建数组的时，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

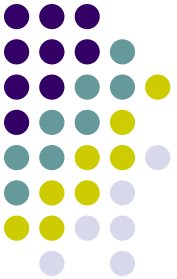
- 通过下面的表达式引用数组的一个元素：
  arrayName[index]
- 数组下标必须是 int , short, byte, 或者 char.
- 下标从零开始计数.
- 元素的个数即为数组的长度，可以通过arryName.length引用
- 元素下标最大值为 **length – 1，如果超过最大值，将会产生数组越界异常（ArrayIndexOutOfBoundsException）**



1.5.4 数组的引用(续)

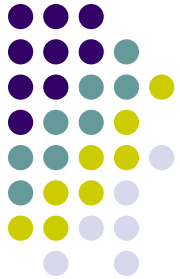
```
int[] data = new int[10];
```

- 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



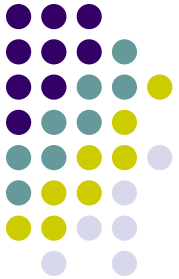
1.5.4 数组的引用(续)

```
int values[] = new int[7];
int index;
index  = 0;
values[ index ] = 71;
index  = 5;
values[ index ] = 23;
index  = 3;
values[ 2+2 ] = values[ index-3 ];
```



1.5.4 数组的引用(续)

```
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
    }
}
```

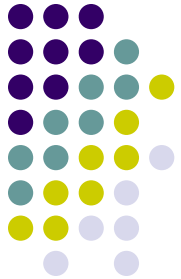


1.5.4 数组的引用(续)

```
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



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)

运行结果：

a1[0] = 2

a1[1] = 3

a1[2] = 4

a1[3] = 5

a1[4] = 6

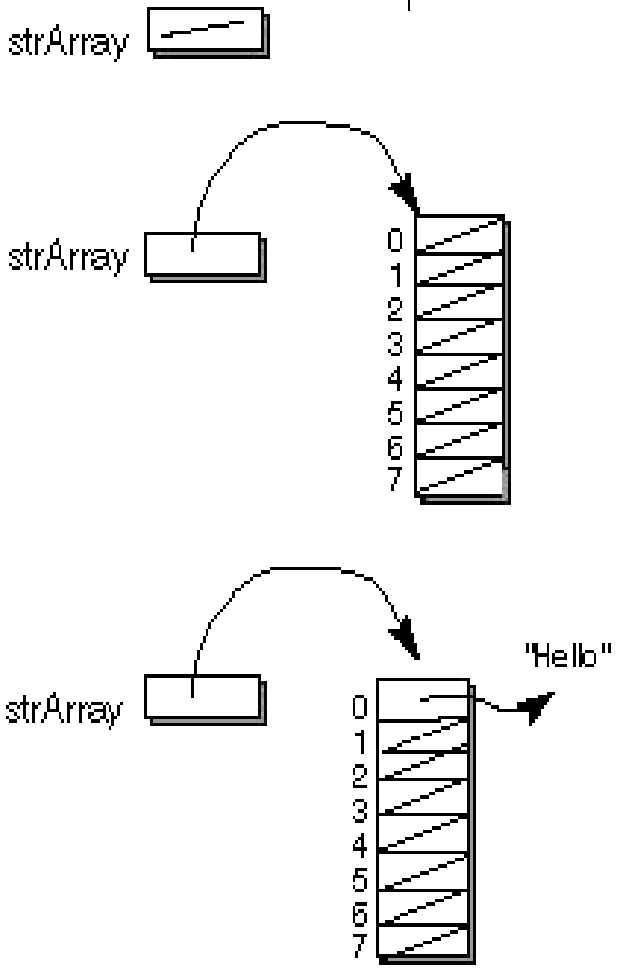
1.5.4 数组的引用(续)

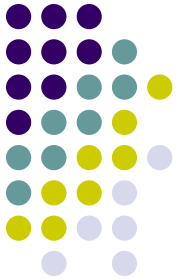
- 字符串引用构成的数组：

String[] strArray;

strArray = new String[8] ;

strArray[0]= "Hello" ;





1.5.4 数组的引用(续)

- 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());
      }
   }
}
```

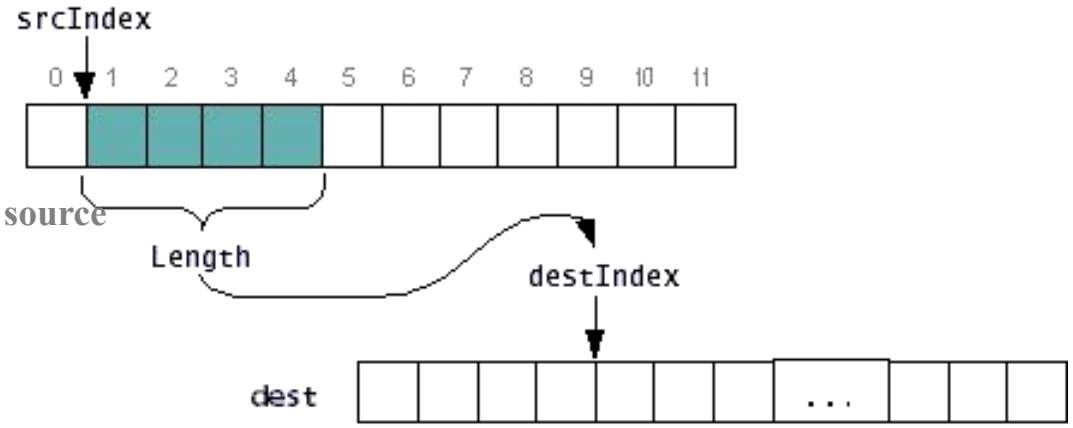
运行结果：

string one
string two
string three

1.5.4 数组的引用(续)

- 数组的复制：

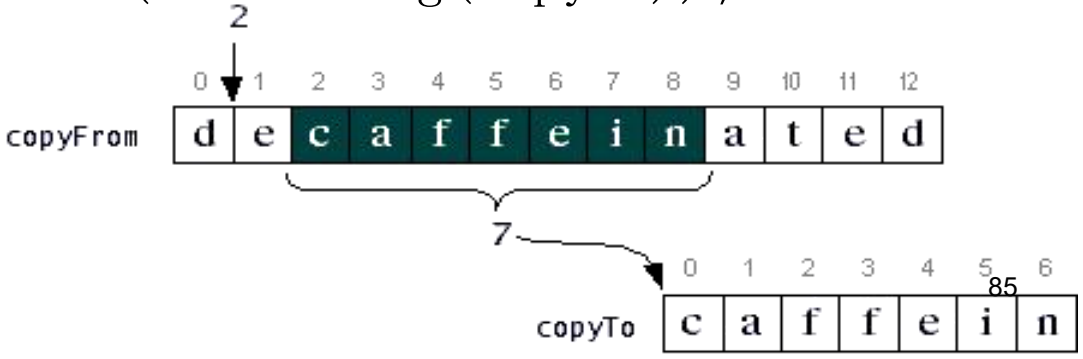
  public static void arraycopy(Object *source* , int *srcIndex* , Object *dest* , int *destIndex* , int *length*)



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'};
     char[] copyTo = new char[7];
     System.arraycopy(copyFrom, 2, copyTo, 0, 7);
     System.out.println(new String(copyTo));
   }
}
```

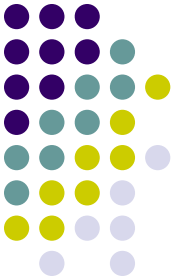


1.5.5 多维数组

```
int[][] gradeTable;
……
```

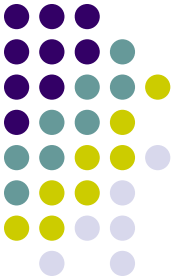
gradeTable[0][1]为
gradeTable[3][4]为
gradeTable[6][2]为

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)

- 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。



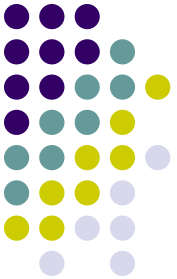
1.5.5 多维数组(续)

- 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：

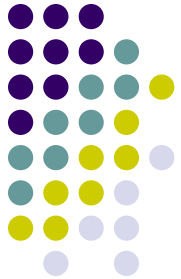
Length is: 3



1.5.5 多维数组(续)

- 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 );
  }
}
```

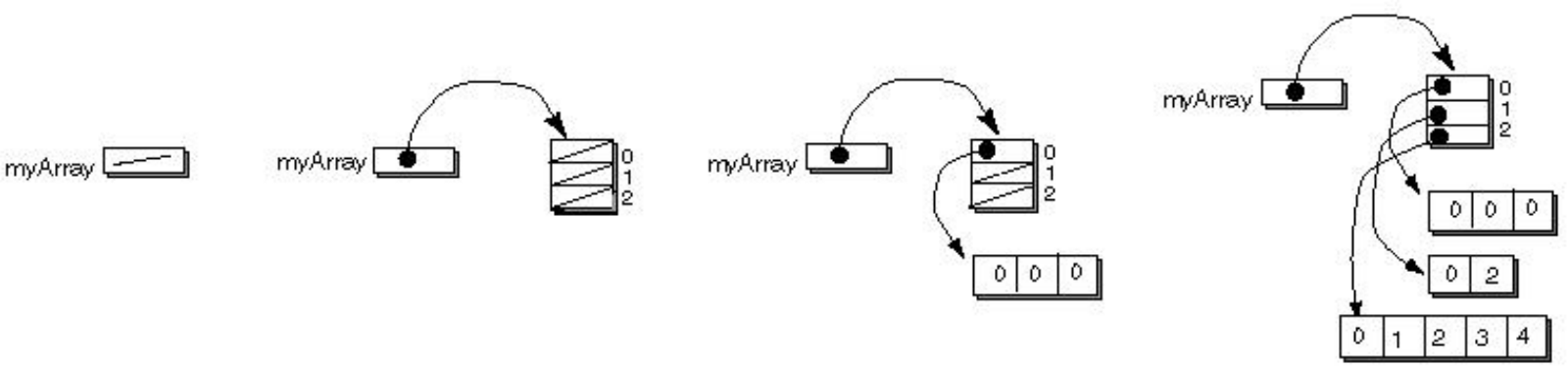
运行结果：

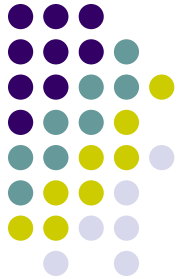
```
Length of array is: 3
Length of row[0] is: 3
Length of row[1] is: 2
Length of row[2] is: 5
```

1.5.5 多维数组(续)

```
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