



JSP程序设计教程

第2章 JSP开发基础



第2章 JSP开发基础

- 2.1 Java语言基础 ✓
- 2.2 JavaScript脚本语言 ✓



2.1 Java语言基础

Java语言是由Sun公司于1995年推出的新一代编程语言。Java语言一经推出，便受到了业界的广泛关注，现已成为一种在Internet应用中被广泛使用的网络编程语言。它具有简单、面向对象、可移植、分布性、解释器通用性、稳健、多线程、安全及高性能等语言特性。另外，Java语言还提供了丰富的类库，方便用户进行自定义操作。



2.1 Java语言基础

2.1.1 面向对象程序设计 ✓

2.1.2 基本数据类型及基本数据类型间的转换 ✓

2.1.3 常量与变量 ✓

2.1.4 运算符的应用 ✓

2.1.5 流程控制语句 ✓

2.1.6 字符串处理 ✓

2.1.7 数组的创建与应用 ✓

2.1.8 集合类的应用 ✓

2.1.9 异常处理语句 ✓





2.2 JavaScript脚本语言

JavaScript是一种比较流行的制作网页特效的脚本语言，它由客户端浏览器解释执行，可以应用在JSP、ASP和PHP等网站中。同时，随着Web2.0和Ajax技术进入Web开发的主流市场，JavaScript已经被推到了舞台的中心。因此，熟练掌握并应用JavaScript对于网站开发人员非常重要。下面将详细介绍JavaScript的基本语法及常用对象。



2.2 JavaScript脚本语言

- 2.2.1 JavaScript脚本语言概述 ✓
- 2.2.2 在JSP中引入JavaScript ✓
- 2.2.3 JavaScript的数据类型与运算符 ✓
- 2.2.4 JavaScript的流程控制语句 ✓
- 2.2.5 函数的定义和调用 ✓
- 2.2.6 事件 ✓
- 2.2.7 JavaScript常用对象的应用 ✓





2.1.1 面向对象程序设计

面向对象程序设计是软件设计和实现的有效方法，这种方法可以提供软件的可扩充性和可重用性。客观世界中的一个事物就是一个对象，每个客观事物都有自己的特征和行为。从程序设计的角度来看，事物的特性就是数据，行为就是方法。一个事物的特性和行为可以传给另一个事物，这样就可以重复使用已有的特性或行为。当某一个事物得到了其他事物传给它的特性和行为，再添加上自己的特性和行为，就可以对已有的功能进行扩充。面向对象的程序设计方法就是利用客观事物的这种特点，将客观事物抽象成为“类”，并通过类的“继承”实现软件的可扩充性和可重用性。



2.1.1 面向对象程序设计

1. 类的基本概念

Java语言与其他面向对象语言一样，引入了类和对象的概念，类是用来创建对象的模板，它包含被创建的对象的状态描述和方法的定义。因此，要学习Java编程就必须学会怎样去编写类，即怎样用Java的语法去描述一类事物共有的属性和行为。属性通过变量来刻画，行为通过方法来体现，即方法操作属性形成一定的算法来实现一个具体的功能。类把数据和对数据的操作封装成一个整体。



2.1.1 面向对象程序设计

```
[修饰符] class <类名> [extends 父类名] [implements 接口列表]{  
}
```

修饰符：可选参数，用于指定类的访问权限，可选值为**public**、**abstract**和**final**。

类名：必选参数，用于指定类的名称，类名必须是合法的**Java**标识符。一般情况下，要求首字母大写。

extends父类名：可选参数，用于指定要定义的类继承于哪个父类。当使用**extends**关键字时，父类名为必选参数。

implements 接口列表：可选参数，用于指定该类实现的是哪些接口。当使用**implements**关键字时，接口列表为必选参数。



2.1.1 面向对象程序设计

(2) 类体

在类声明部分的大括号中的内容为类体。类体主要由两部分构成，一部分是成员变量的定义，另一部分是成员方法的定义。类体的定义格式如下：

```
[修饰符] class <类名> [extends 父类名] [implements 接口列表]{  
    定义成员变量  
    定义成员方法  
}
```



2.1.1 面向对象程序设计

3. 定义成员方法

Java中类的行为由类的成员方法来实现。类的成员方法由方法的声明和方法体两部分组成，其一般格式如下：

```
[修饰符] <方法返回值的类型> <方法名>([参数列表]) {  
    [方法体]  
}
```

修饰符：可选参数，用于指定方法的被访问权限，可选值为public、protected和private。

方法返回值的类型：必选参数，用于指定方法的返回值类型，如果该方法没有返回值，可以使用关键字void进行标识。方法返回值的类型可以是任何Java数据类型。



2.1.1 面向对象程序设计

```
[修饰符] <方法返回值的类型> <方法名>([参数列表]) {  
    [方法体]  
}
```

方法名：必选参数，用于指定成员方法的名称，方法名必须是合法的**Java**标识符。

参数列表：可选参数，用于指定方法中所需的参数。当存在多个参数时，各参数之间应使用逗号分隔。方法的参数可以是任何**Java**数据类型。

方法体：可选参数，方法体是方法的实现部分，在方法体中可以定义局部变量。需要注意的是，当方法体省略时，其外面的大括号一定不能省略。



2.1.1 面向对象程序设计

【例2-1】 在Fruit类中声明两个成员方法
grow()和harvest()



2.1.1 面向对象程序设计

4. 成员变量与局部变量

在类体中变量定义部分所声明的变量为类的成员变量，而在方法体中声明的变量和方法的参数则称为局部变量。成员变量和局部变量的区别在于其有效范围不同。成员变量在整个类内都有效，而局部变量只在定义它的成员方法内才有效。

(1) 声明成员变量

Java用成员变量来表示类的状态和属性，声明成员变量的基本语法格式如下：



2.1.1 面向对象程序设计

[修饰符] [static] [final] [transient] [volatile] <变量类型> <变量名>;

修饰符：可选参数，用于指定变量的被访问权限，可选值为public、protected和private。

static：可选，用于指定该成员变量为静态变量，可以直接通过类名访问。如果省略该关键字，则表示该成员变量为实例变量。

（2）声明局部变量

定义局部变量的基本语法格式同定义成员变量类似，所不同的是不能使用public、protected、private和static关键字对局部变量进行修饰，但可以使用final关键字。语法格式如下：



2.1.1 面向对象程序设计

[final] <变量类型> <变量名>;

final: 可选，用于指定该局部变量为常量。

变量类型: 必选参数：用于指定变量的数据类型，其值为**Java**中的任何一种数据类型。

变量名: 必选参数，用于指定局部变量的名称，变量名必须是合法的**Java**标识符。

【例2-2】 成员变量和局部变量示例

在**Fruit**类中声明3个成员变量，并且在其成员方法**grow()**中声明两个局部变量。



2.1.1 面向对象程序设计

5. 构造方法的概念及用途

构造方法是一种特殊的方法，它的名字必须与它所在类的名字完全相同，并且没有返回值，也不需要使用关键字**void**进行标识。构造方法用于对对象中的所有成员变量进行初始化，在创建对象时立即被调用。需要注意的是，如果用户没有定义构造方法，**Java**会自动提供一个默认的构造方法，用来实现成员变量的初始。



2.1.1 面向对象程序设计

6. 创建Java类对象

在Java中，创建对象包括声明对象和为对象分配内存两部分，下面分别进行介绍。

(1) 声明对象

对象是类的实例，属于某个已经声明的类。因此，在对对象进行声明之前，一定要先定义该对象的类。声明对象的一般格式如下：



2.1.1 面向对象程序设计

类名 对象名;

类名：必选，用于指定一个已经定义的类。

对象名：必选，用于指定对象名称，对象名必须是合法的**Java**标识符。

例如，声明**Fruit**类的一个对象**fruit**的代码如下：

```
Fruit fruit;
```

在声明对象时，只是在内存中为其建立一个引用，并置初值为**null**，表示不指向任何内存空间，因此，还需要为对象分配内存。



2.1.1 面向对象程序设计

(3) 为对象分配内存

为对象分配内存也称为实例化对象。在Java中使用关键字**new**来实例化对象，具体语法格式如下：

```
对象名=new 构造方法名([参数列表]);
```

对象名：必选，用于指定已经声明的对象名。

构造方法名：必选，用于指定构造方法名，即类名，因为构造方法与类名相同。

参数列表：可选参数，用于指定构造方法的入口参数。如果构造方法无参数，则可以省略。



2.1.1 面向对象程序设计

例如，在声明**Fruit**类的一个对象**fruit**后，可以通过以下代码为对象**fruit**分配内存：

```
fruit=new Fruit();
```

在上面的代码中，由于**Fruit**类的构造方法无入口参数，所以省略了参数列表。

在声明对象时，也可直接为其分配内存。例如，上面的声明对象和为对象分配内存的功能也可以通过以下代码实现。

```
Fruit fruit=new Fruit();
```



2.1.1 面向对象程序设计

7. 对象的使用

创建对象后，就可以通过对象来引用其成员变量，并改变成员变量的值，而且还可以通过对象来调用其成员方法。通过使用运算符“.”实现对成员变量的访问和成员方法的调用。

【例2-3】 对象的使用方法



2.1.1 面向对象程序设计

8. 对象的销毁

在许多程序设计语言中，需要手动释放对象所占用的内存，但是，在Java中则不需要手动完成这项工作。Java提供的垃圾回收机制可以自动判断对象是否还在使用，并能够自动销毁不再使用的对象，收回对象所占用的资源。

Java提供了一个名为`finalize()`的析构方法，用于在对象被垃圾回收机制销毁之前，由垃圾回收系统调用。但是垃圾回收系统的运行是不可预测的。因此，在Java程序中，也可以使用析构方法`finalize()`随时来销毁一个对象。析构方法`finalize()`没有任何参数和返回值，每个类有且只有一个析构方法。



2.1.1 面向对象程序设计

9. 包的使用

包（**package**）是Java提供的一种区别类的名字空间的机制，是类的组织方式，是一组相关类和接口的集合，它提供了访问权限和命名的管理机制。Java中提供的包主要有以下3种用途：

- ① 将功能相近的类放在同一个包中，可以方便查找与使用；
- ② 由于在不同包中可以存在同名类，所以使用包在一定程度上可以避免命名冲突；
- ③ 在Java中，某些访问权限是以包为单位的。



2.1.1 面向对象程序设计

(1) 创建包

创建包可以通过在类或接口的源文件中使用 **package** 语句实现，**package** 语句的语法格式如下：

```
package 包名;
```

包名：必选，用于指定包的名称，包的名称为合法的 **Java** 标识符。当包中还有包时，可以使用“包1.包2.....包n”进行指定，其中，包1为最外层的包，而包n则为最内层的包。

package 语句通常位于类或接口源文件的第一行。例如，定义一个类 **SimpleH**，将其放入 **com.wgh** 包中的代码如下：



2.1.1 面向对象程序设计

```
package com.wgh;  
public class SimpleH{  
    ...    //此处省略了类体的代码  
}
```

(2) 使用包中的类

类可以访问其所在包中的所有类，还可以使用其他包中的所有public类。访问其他包中的public类可以有以下两种方法。

- 使用长名引用包中的类。



2.1.1 面向对象程序设计

使用长名引用包中的类比较简单，只需要在每个类名前面简单地加上完整的包名即可。例如，创建**Circ**类（保存在**com.wgh**包中）的对象并实例化该对象的代码如下：

```
com.wgh.Circ circ=new com.wgh.Circ();
```

■ 使用import语句引入包中的类

由于采用使用长名引用包中的类的方法比较繁琐，所以**Java**提供了**import**语句来引入包中的类。**import**语句的基本语法格式如下：



2.1.1 面向对象程序设计

```
import 包名1[.包名2....].类名|*;
```

当存在多个包名时，各个包名之间使用“.”分隔，同时包名与类名之间也使用“.”分隔。

*：表示包中所有的类。

例如，引入com.wgh包中的Circ类的代码如下：

```
import com.wgh.Circ;
```

如果com.wgh包中包含多个类，也可以使用以下语句引入该包下的全部类：

```
import com.wgh.*;
```



课件制作人：王国辉



2.1.1 面向对象程序设计

2. 定义类

在Java中定义类主要分为两部分：类的声明和类体。

(1) 类声明

在类声明中，需要定义类的名称、对该类的访问权限和该类与其他类的关系等。类声明的格式如下：

```
[修饰符] class <类名> [extends 父类名] [implements 接口列表]{  
}
```

修饰符：可选参数，用于指定类的访问权限，可选值为public、abstract和final。

2.1.2 基本数据类型及基本数据类型间的转换

1. 基本数据类型

Java基本数据类型主要包括整数类型、浮点类型、字符类型和布尔类型。其中整数类型又分为字节型（**byte**）、短整型（**short**）、整型（**int**）和长整型（**long**），它们都用来定义一个整数，唯一的区别就是它们所定义的整数所占用内存的空间不同，因此整数的取值范围也不同；Java中的浮点类型又包括单精度类型（**float**）和双精度类型（**double**），在程序中使用这两种类型来存储小数。

Java中的各种基本数据类型及它们的取值范围、占用的内存大小和默认值，如表下表所示。

2.1.1 面向对象程序设计

各种基本数据类型的取值范围、占用的内存大小及默认值

数据类型		关键字	占用内存	取值范围	默认值
整数类型	字节型	byte	8位	-128~127	0
	短整型	short	16位	-32768~32767	0
	整型	int	32位	-2147483648~2147483647	0
	长整型	long	64位	-9223372036854775808~9223372036854775807	0
浮点类型	单精度型	float	32位	1.4E-45~3.4028235E38	0.0f
	双精度型	double	64位	4.9E-324~1.7976931348623157E308	0.0d
字符型	字符型	char	16位	16位的Unicode字符，可容纳各国的字符集；若以Unicode来看，就是'\u0000'到'\uffff'；若以整数来看，范围在0~65535，例如，65代表'A'	'\u0000'
布尔型	布尔型	boolean	8位	true和false	false

2.1.2 基本数据类型及基本数据类型间的转换

2. 基本数据类型间的转换

在Java语言中，当多个不同基本数据类型的数据进行混合运算时，如整型、浮点型和字符型进行混合运算，需要先将它们转换为统一的类型，然后再进行计算。在Java中，基本数据类型之间的转换可分为自动类型转换和强制类型转换两种。

(1) 自动类型转换

从低级类型向高级类型的转换为自动类型转换，这种转换将由系统按照各数据类型的级别从低到高自动完成，Java编程人员无需进行任何操作。在Java中各基本数据类型间的级别如下图所示。

2.1.2 基本数据类型及基本数据类型间的转换

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

(2) 强制类型转换

如果把高级数据类型数据赋值给低级类型变量，就必须进行强制类型转换，否则编译出错。强制类型转换格式如下：

(欲转换成的数据类型)值

其中“值”可以字面常数或者变量，例如：

```
short    s1=65, s2;
```

```
char     c1='a', c2;
```

```
s2=(short)c1;
```

```
c2=(char)s1;
```

//将char型强制转换为short型，s2值为：97

//将short型强制转换为char型，c2值为：A.





2.1.3 常量与变量

1. 变量

变量是Java程序中的基本存储单元，它的定义包括变量名、变量类型和作用域几个部分。

(1) 变量名是一个合法的标识符，它是字母、数字、下划线或美元符“\$”的序列，Java对变量名区分大小写，变量名不能以数字开头，而且不能为关键字。合法的变量名如pwd、value_1、money\$等。非法的变量名如3Three、house#、final（关键字）。

(2) 变量类型用于指定变量的数据类型，可以通过int、float、double和char等关键字来指定。例如下面的代码：



2.1.3 常量与变量

```
int number;           //定义整型变量
long numberL;         //定义长整型变量
short numberS;        //定义短整型变量
float numberF;        //定义单精度变量
double numberD;       //定义双精度变量
char strC;            //定义字符变量
```

（3）变量的有效范围是指程序代码能够访问该变量的区域，若超出该区域访问变量，则编译时会出现错误。有效范围决定了变量的生命周期，变量的生命周期是指从声明一个变量并分配内存空间开始，到释放该变量并清除所占用内存空间结束。进行变量声明的位置，决定了变量的有效范围，根据有效范围的不同，可将变量分为以下两种。



2.1.3 常量与变量

(1) 成员变量：在类中声明，在整个类中有效。

(2) 局部变量：在方法内或方法内的某代码块（方法内部，“{”与“}”之间的代码）中声明的变量。在代码块中声明的变量，只在当前代码块中有效；在代码块外、方法内声明的变量，在整个方法内都有效。

通过以下代码可以了解成员变量和局部变量的声明及使用范围。

2.1.3 常量与变量

```
public class Game {  
    private int medal_All=800;           //成员变量  
    public void China(){  
        int medal_CN=100;               //方法的局部变量  
        if(true){                        //代码块  
            int gold=50;                 //代码块的局部变量  
            medal_CN+=50;                //允许访问  
        }                                //允许访问  
        medal_All-=150;  
        }  
        gold=100;                        //编译出错  
        medal_CN+=100;                   //允许访问  
        medal_All-=200;                  //允许访问  
    }  
    public void Other(){  
        medal_All=800;                   //允许访问  
        medal_CN=100;                   //编译出错，不能访问其他方法中的局部变量  
        gold=10;                         //编译出错  
    }  
}
```



2.1.3 常量与变量

2. 常量

在Java中写下一个数值，这个数就称为字面常数。它会存储于内存中的某个位置，用户将无法改变它的值。Java中的常量值是用文字串表示的，它区分为不同的类型，如整型常量321、实型常量3.21、字符常量‘a’、布尔常量“true”和“false”及字符串常量“One World One Dream”。

在Java中，也可以用final关键字来定义常量。通常情况下，在通过final关键字定义常量时，常量名全部为大写字母。需要说明的是，由于常量在程序执行过程中保持不变，所以在常量定义后，如果再次对该常量进行赋值，程序将会出错。

课件制作人：王国辉



2.1.4 运算符的应用

在Java语言中表达各种运算的符号叫做运算符。Java运算符主要可分为：赋值运算符、算术运算符、关系运算符、逻辑运算符、位运算符及条件运算符，下面将分别进行介绍。

1. 赋值运算符

Java中的赋值运算符可以分为简单赋值运算和复合赋值运算。简单赋值运算是将赋值运算符（=）右边的表达式的值保存到赋值运算符左边的变量中，复合赋值运算是混合了其他操作（算术运算操作、位操作等）和赋值操作，如：



2.1.4 运算符的应用

`sum+=i;` `//等同于sum=sum+i;`

Java中的赋值运算符如下表所示。

运算符	说 明	运算符	说 明
=	简单赋值	&=	进行与运算后赋值
+=	相加后赋值	=	进行或运算后赋值
-=	相减后赋值	^=	进行异或运算后赋值
*=	相乘后赋值	<<=	左移之后赋值
/=	相除后赋值	>>=	带符号右移后赋值
%=	求余后赋值	>>>=	填充零右移后赋值



2.1.4 运算符的应用

2. 算术运算符

Java中的算术运算符包括：**+**（加号）、**-**（减号）、*****（乘号）、**/**（除号）和**%**（求余）。算术运算符支持整型和浮点型数据的运算，当整型与浮点型数据进行算术运算时，会进行自动类型转换，结果为浮点型。

Java中算术运算符的功能及使用方式如下表所示。

运算符	说 明	举 例	结果及类型
+	加法	1.23f+10	结果： 11.23 类型： float
-	减法	4.56-0.5f	结果： 4.06 类型： double
*	乘法	3*9L	结果： 27 类型： long
/	除法	9/4	结果： 2 类型： int
%	求余数	10%3	结果： 1 类型： int



2.1.4 运算符的应用

3. 关系运算符

通过关系运算符计算的结果是一个boolean类型值。对于应用关系运算符的表达式，计算机将判断运算对象之间通过关系运算符指定的关系是否成立，若成立则表达式的返回值为true，否则为false。

关系运算符包括： $>$ （大于）、 $<$ （小于）、 $>=$ （大于或等于）、 $<=$ （小于或等于）、 $==$ （等于）和 $!=$ （不等于）。其中等于和不等于运算符适用于引用类型和所有的基本数据类型，而其他的关系运算符只适用于除boolean类型外的所有基本数据类型。

Java中的关系运算符如下表所示。



2.1.4 运算符的应用

运算符	说 明	举 例	结 果	运 算 符	说 明	举 例	结 果
>	大于	'a'>'b'	false	<=	小于或等于	1.67f<=1.67f	true
<	小于	200>100	true	==	等于	1.0==1	true
>=	大于或等于	11.11>=10	true	!=	不等于	'天'!='天'	false

4. 逻辑运算符

逻辑运算符经常用来连接关系表达式，对关系表达式的值进行逻辑运算，因此逻辑运算符的运算对象必须是逻辑型数据，其逻辑表达式的运行结果也是逻辑型数据。

Java中的逻辑运算符如下表所示。



2.1.4 运算符的应用

运算符	意 义	运 算 结 果
&	逻辑与	true&true: true , false&>false: false, true&>false: false
	逻辑或	true!true: true , false!false: false , true!false: true
^	异或	true&true: false , false&>false: false, true&>false: true
	短路或	true&true: true , false&>false: false, true&>false: true
&&	短路与	true&true: true , false&>false: false, true&>false: false
!	逻辑反	!true: false, !false: true
==	相等	true==true: true , false==false: true, true==false: false
!=	不相等	true!=true: false , false!=false: false , true!=false: true



2.1.4 运算符的应用

5. 位运算符

位运算符用于对数值的位进行操作，参与运算的操作数只能是int或long类型。在不产生溢出的情况下，左移一位相当于乘以2，用左移实现乘法运算的速度比通常的乘法运算速度快。Java中的位运算符如下表所示。

2.1.4 运算符的应用

运算符	说 明	实 例
&	转换为二进制数据进行与运算	$1\&1=1, 1\&0=0, 0\&1=0, 0\&0=0$
	转换为二进制数据进行或运算	$1 1=1, 1 0=1, 0 1=1, 0 0=0$
^	转换为二进制数据进行异或运算	$1\wedge 1=0, 1\wedge 0=1, 0\wedge 1=1, 0\wedge 0=0$
~	进行数值的相反数减1运算	$\sim 50 = -50-1 = -51$
>>	向右移位	$15 \gg 1 = 7$
<<	向左移位	$15 \ll 1 = 30$
>>>	向右移位	$15 \ggg 1 = 7$
<<=	左移赋值运算符	$n \ll -3$ 等价于 $n = n \ll 3$
>>=	右移赋值运算符	$n \gg -3$ 等价于 $n = n \gg 3$
>>>=	无符号右移赋值运算符	$n \ggg -3$ 等价于 $n = n \ggg 3$



2.1.4 运算符的应用

6. 条件运算符

条件运算符是三元运算符，其语法格式如下：

`<表达式> ? a : b`

其中，表达式值的类型为逻辑型。若表达式的值为true，则返回a的值；若表达式的值为false，则返回b的值。

【例2-4】 应用条件运算符输出库存信息



2.1.4 运算符的应用

7. 自动递增、递减运算符

与C、C++相同，Java语言也提供了自动递增与递减运算符，其作用是自动将变量值加1或减1。它们既可以放在操作元的前面，也可以放在操作元的后面，根据运算符位置的不同，最终得到的结果也是不同的：放在操作元前面的自动递增、递减运算符，会先将变量的值加1，然后再使该变量参与表达式的运算；放在操作元后面的递增、递减运算符，会先使变量参与表达式的运算，然后再将该变量加1。例如：



2.1.4 运算符的应用

```
int n1=3;  
int n2=3;  
int a=2+(++n1);           //先将变量n1加1，然后再执行"2+4"  
int b=2+(n2++);           //先执行"2+3"，然后再将变量n2加1  
System.out.println(a);    //输出结果为： 6  
System.out.println(b);    //输出结果为： 5  
System.out.println(n1);   //输出结果为： 4  
System.out.println(n2);   //输出结果为： 4
```

说明：自动递增、递减运算符的操作元只能为变量，不能为字面常数和表达式，且该变量类型必须为整型、浮点型或Java包装类型。例如，`++1`、`(n+2)++`都是不合法的。





2.1.5 流程控制语句

Java语言中，流程控制语句主要有分支语句、循环语句和跳转语句3种。

1. 分支语句

所谓分支语句，就是对语句中不同条件的值进行判断，进而根据不同的条件执行不同的语句。在分支语句中主要有以下两个语句：

- If条件语句 ✓
- switch多分支语句 ✓



If...else语句

if...else语句是条件语句最常用的一种形式，它针对某种条件有选择地做出处理。通常表现为“如果满足某种条件，就进行某种处理，否则就进行另一种处理”。其语法格式如下：

```
if(条件表达式){  
    语句序列1  
}  
else{  
    语句序列2  
}
```

条件表达式：必要参数。其值可以由多个表达式组成，但是其最后结果一定是**boolean**类型，也就是其结果只能是**true**或**false**。



If...else语句

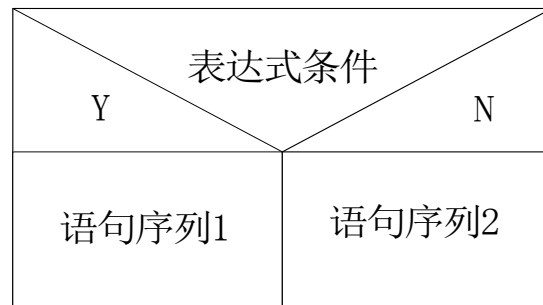
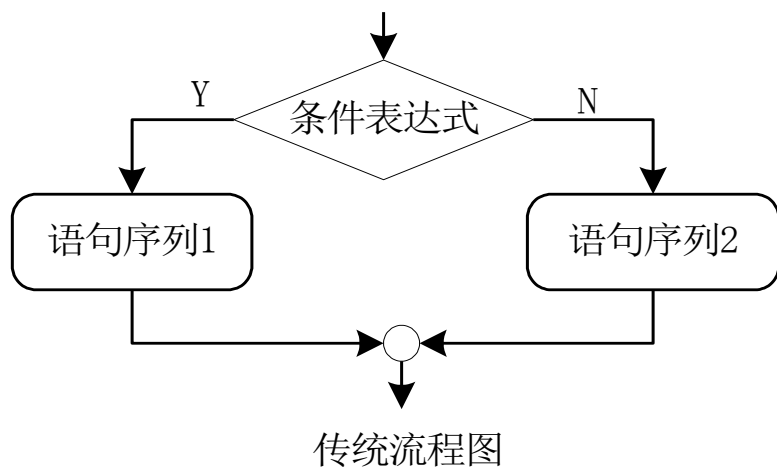
```
if(条件表达式){  
    语句序列1  
}else{  
    语句序列2  
}
```

语句序列1：可选参数。一条或多条语句，当表达式的值为**true**时执行这些语句。

语句序列2：可选参数。一条或多条语句，当表达式的值为**false**时执行这些语句。

if...else条件语句的执行过程如下图所示。

If...else语句



【例2-5】 if...else语句示例





switch多分支语句

switch语句是多分支选择语句，常用来根据表达式的值选择要执行的语句。switch语句的基本语法格式如下：

```
switch(表达式){  
    case 常量表达式1: 语句序列1  
        [break;]  
    case 常量表达式2: 语句序列2  
        [break;]  
    .....  
    case 常量表达式n: 语句序列n  
        [break;]  
    default: 语句序列n+1  
        [break;]  
}
```



switch多分支语句

表达式：必要参数。可以是任何byte、short、int和char类型的变量。

常量表达式1：如果有case出现，则为必要参数。该常量表达式的值必须是一个与表达式数据类型相兼容的值。

语句序列1：可选参数。一条或多条语句，但不需要大括号。当表达式的值与常量表达式1的值匹配时执行；如果不匹配则继续判断其他值，直到常量表达式n。

常量表达式n：如果有case出现，则为必要参数。该常量表达式的值必须是一个与表达式数据类型相兼容的值。



switch多分支语句

语句序列**n**：可选参数。一条或多条语句，但不需要大括号。当表达式的值与常量表达式**n**的值匹配时执行。

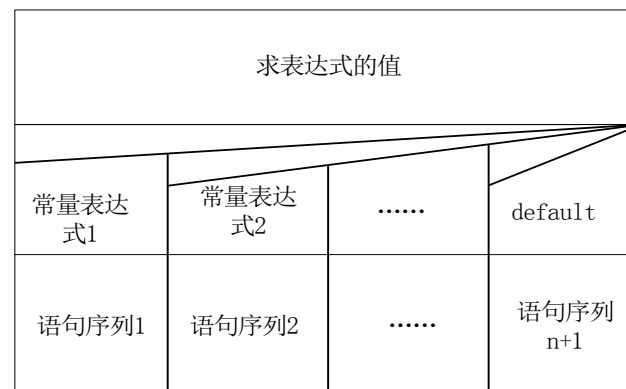
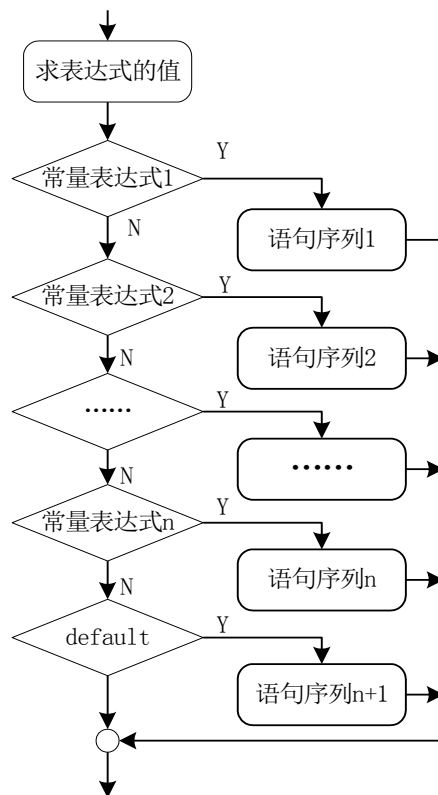
break;：可选参数。用于跳出**switch**语句。

default：可选参数。如果没有该参数，则当所有匹配不成功时，将不会执行任何操作。

语句序列**n+1**：可选参数。如果没有与表达式的值相匹配的**case**常量时，将执行语句序列**n+1**。

switch语句的执行流程如下图所示。

If...else语句



N-S结构化流程图

【例2-6】 switch语句示例





2.1.5 流程控制语句

2. 循环语句

所谓循环语句，主要就是在满足条件的情况下反复执行某一个操作。在Java中，提供了3种常用的循环语句，分别是：

- for 循环语句 ✓
- while 循环语句 ✓
- do...while循环语句 ✓



for循环语句

for循环语句也称为计次循环语句，一般用于循环次数已知的情況。**for**循环语句的基本语法格式如下：

```
for(初始化语句;循环条件;迭代语句){  
    语句序列  
}
```

初始化语句：为循环变量赋初始值的语句，该语句在整个循环语句中只执行一次。

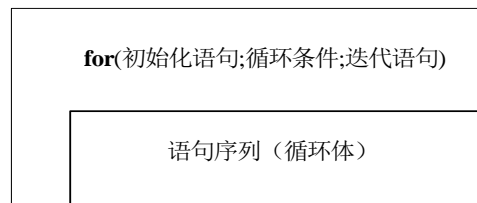
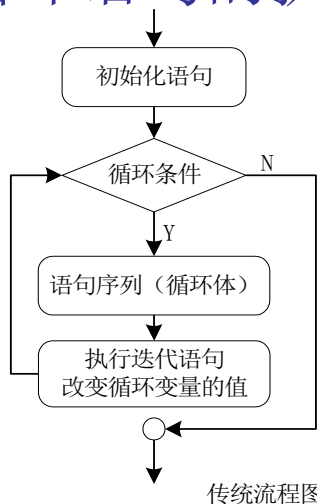
循环条件：决定是否进行循环的表达式，其结果为**boolean**类型，也就是其结果只能是**true**或**false**。

迭代语句：用于改变循环变量的值的语句。

语句序列：也就是循环体，在循环条件的结果为**true**时，重复执行。

for循环语句

for循环语句执行的过程是：先执行为循环变量赋初始值的语句，然后判断循环条件，如果循环条件的结果为**true**，则执行一次循环体，否则直接退出循环，最后执行迭代语句，改变循环变量的值，至此完成一次循环，接下来将进行下一次循环，直到循环条件的结果为**false**，才结束循环。**for**循环语句的执行流程如下图所示。



N-S结构化流程图





while循环语句

while循环语句也称为前测试循环语句，它的循环重复执行方式，是利用一个条件来控制是否要继续重复执行这个语句。**while**循环语句与**for**循环语句相比，无论是语法还是执行的流程，都较为简明易懂。**while**循环语句的基本语法格式如下：

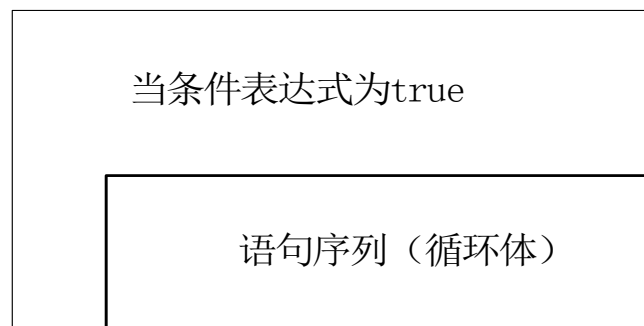
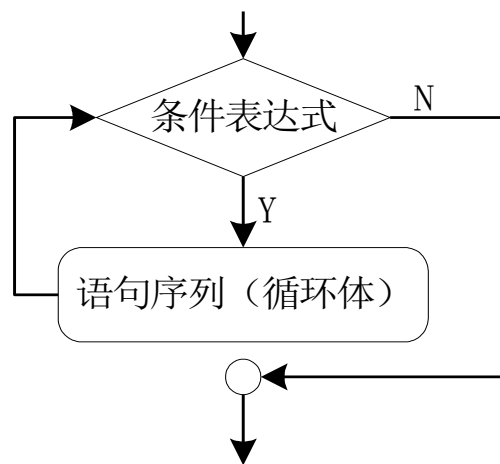
```
while(条件表达式){  
    语句序列  
}
```

条件表达式：决定是否进行循环的表达式，其结果为**boolean**类型，也就是其结果只能是**true**或**false**。

语句序列：也就是循环体，在条件表达式的结果为**true**时，重复执行。

while循环语句

while循环语句执行的过程是：先判断条件表达式，如果条件表达式的值为**true**，则执行循环体，并且在循环体执行完毕后，进入下一次循环，否则退出循环。**while**循环语句的执行流程如下图所示。





do...while循环语句

do...while循环语句也称为后测试循环语句，它的循环重复执行方式，也是利用一个条件来控制是否要继续重复执行这个语句。与**while**循环所不同的是，它先执行一次循环语句，然后再去判断是否继续执行。**do...while**循环语句的基本语法格式如下：

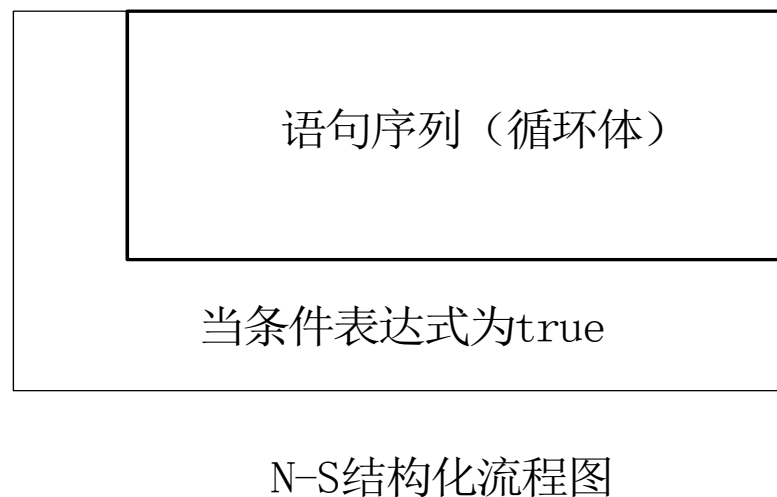
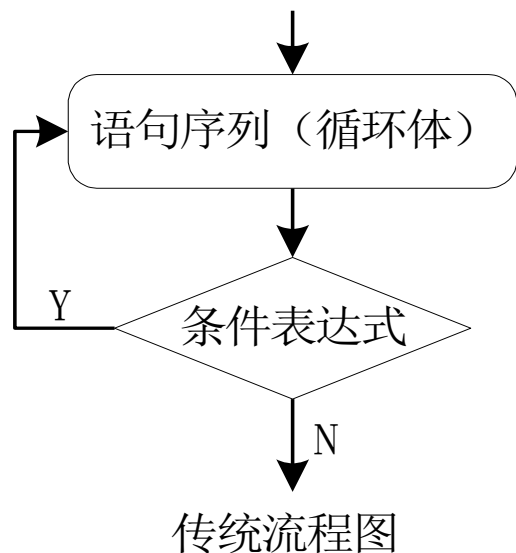
```
do{  
    语句序列  
} while(条件表达式);    //注意！语句结尾处的分号";"一定不能少
```

语句序列：也就是循环体，循环开始时首先被执行一次，然后在条件表达式的结果为**true**时，重复执行。

条件表达式：决定是否进行循环的表达式，其结果为**boolean**类型，也就是其结果只能是**true**或**false**。

Do...while循环语句

do...while循环语句的执行流程如下图所示。



【例2-7】 循环语句示例





2.1.5 流程控制语句

3. 跳转语句

Java语言中提供了3种跳转语句，分别是：

- **break**跳转语句

break语句可以应用在**for**、**while**和**do...while**循环语句中，用于强行退出循环，也就是忽略循环体中任何其他语句和循环条件的限制。

- **continue**跳转语句

continue语句只能应用在**for**、**while**和**do...while**循环语句中，用于让程序直接跳过其后面的语句，进行下一次循环。

- **return**跳转语句



2.1.5 流程控制语句

return语句可以从一个方法返回，并把控制权交给调用它的语句。它的语法格式如下：

```
return [表达式];
```

表达式：可选参数，表示要返回的值。它的数据类型必须同方法声明中的返回值类型一致，这可以通过强制类型转换实现。

return语句通常被放在被调用方法的最后，用于退出当前方法并返回一个值。当把单独的**return**语句放在一个方法的中间时，会产生“**Unreachable code**”的编译错误。但是可以通过把**return**语句用**if**语句括起来的方法，将**return**语句放在一个方法中间，用来实现在程序未执行完方法中的全部语句时退出。





2.1.6 字符串处理

字符串由一连串字符组成，它可以包含字母、数字、特殊符号、空格或中文字，只要是键盘能输入的文字都可以。它的表示方法是在文字两边加双引号，例如：“简单”或“world”等都是合法字符。**Java**以类型的方法来处理字符串。所有以双引号包围的字符串常数，**Java**的编译器都会将它编译为**String**类对象。

1. 字符串的声明

在**Java**中，对于字符串的处理，均由**Java.lang**包中的**String**类完成。下面是声明字符串变量的语法。

（1）初始化一个新创建的**String**对象，它表示一个空字符序列。声明代码如下：



2.1.6 字符串处理

String()

(2) 导入参数。声明代码如下：

String(String name)

该方法创建带有内容的字符串，使用双引号标识。利用new关键字，调用String类产生一个字符串对象，并设置字符串的值。例如下面的代码：

```
String name = new String("简单");
```

name是String类的对象，“简单”指的是字符串内容。例如，要在网页中输出文字“平平淡淡才是真！快快乐乐才是福！”，可以写成以下代码：



2.1.6 字符串处理

```
<%  
String str;  
    //定义字符串  
str = new String("平平淡淡才是真！");  
String str1 = new String("快快乐乐才是福！");  
out.println(str);  
out.println(str1);  
%>
```

（3）导入一个char[]数组。声明代码如下：

```
String(char[] value);
```



2.1.6 字符串处理

该方法产生的String对象，内含的是value参数（char[]类型）所代表的字符串内容。字符串是常量；它们的值在创建之后不能改变。字符串缓冲区支持可变的字符串。因为String对象是不可变的，所以可以共享它们。例如下面的代码：

```
String str = "abc";
```

等效于：

```
char data[] = {'a', 'b', 'c'};
```

```
String str = new String(data);
```

（4）导入一个char[]数组并决定元素值范围。声明代码如下：



2.1.6 字符串处理

`String(char[] value,int offset,int count)`

该方法产生的**String**对象内含的字符串内容，是由**value**字符数组中取出的字符所组成。在该字符串中，第一个字符的索引位置为0。例如下面的代码：

```
char[] myL = {'简','单','快','乐'};  
String str3 = new String(myL,1,2);  
System.out.println("Str3 = " + Str3);
```

输出结果：

Str3 = 单快

（5）导入一个**byte[]**数组。声明代码如下：



2.1.6 字符串处理

```
String(byte[] bytes)
```

该方法产生的**String**对象，其内含的是**bytes**参数（**byte[]**类型）代表的字符串内容，而一个英文字母是以一个**byte**表示，一个中文则以2个**byte**表示。

（6）导入一个**byte[]**数组并决定元素值范围。声明代码如下：

```
String (byte[] bytes,int offset,int length)
```

该方法产生的**String**对象，其包含的是字符串内容，是由**bytes**数组元素取出的一个**byte**类型的值所转化而成。由**offset**参数指定要从哪个默认值开始，**length**参数决定要取多少个元素。



2.1.6 字符串处理

(7) 导入一个StringBuffer对象。声明代码如下：

```
String(StringBuffer buffer)
```

该方法产生的String对象，其内含的字符串，等同于buffer参数（StringBuffer对象）所存放的字符串内容。

2. 字符串类的常用方法

String字符串类的常用方法及含义如表所示。

String字符串类的常用方法及含义

方法名称	方法含义
boolean endsWith(String suffix)	测试此字符串是否以指定的后缀结束
boolean equals(Object anObject)	比较此字符串与指定的对象
boolean equalsIgnoreCase(String anotherString)	将此String与另一个String进行比较，不考虑大小写
int indexOf()	返回指定字符串在另一个字符串中的索引位置
int lastIndexOf()	返回最后一次出现的指定字符在另一个字符串中的索引位置
int length()	返回此字符串的长度
String replace(char oldChar, char newChar)	返回一个新的字符串，它是通过用newChar替换此字符串中出现的所有oldChar而生成的
boolean startsWith(String prefix)	测试指定字符串是否以指定的前缀开始
String substring()	返回一个字符串的子串
char[] toCharArray()	将指定字符串转换为一个新的字符数组
String toLowerCase()	将指定字符串中的所有字符都转换为小写
String toUpperCase()	将指定字符串中的所有字符都转换为大写
String trim()	返回字符串的副本，忽略前导空白和尾部空白
static String valueOf(boolean b)	返回指定参数的字符串表示形式



2.1.6 字符串处理

【例2-8】 字符串应用实例





2.1.7 数组的创建与应用

数组是由多个元素组成的，每个单独的数组元素，就相当于一个变量，可用来保存数据，因此可以将数组视为一连串变量的组合。根据数组存放元素的复杂程度，可将数组依次分为一维数组、二维数组及多维（三维以上）数组。

1. 一维数组

Java中的数组必须先声明，然后才能使用。声明一维数组有以下两种格式：

```
数据类型 数组名[] = new 数据类型[个数];  
数据类型[] 数组名 = new 数据类型[个数];
```



2.1.7 数组的创建与应用

当按照上述格式声明数组后，系统会分配一块连续的内存空间供该数组使用，例如，下面的两行代码都是正确的：

```
String any[] = new String[10];  
String[] any = new String[10];
```

这两个语句实现的功能都是创建了一个新的字符串数组，它有10个元素可以用来容纳String对象，当用关键字new来创建一个数组对象时，则必须指定这个数组能容纳多少个元素。

对于一维数组的赋值，语法格式如下：



2.1.7 数组的创建与应用

数据类型 数组名[] = {数值1,数值2,...,数值n};

数据类型[] 数组名 = {数值1,数值2,...,数值n};

括号内的数值将依次赋值给数组中的第1到n个元素。另外，在赋值声明时，不需要给出数组的长度，编译器会按所给的数值个数来决定数组的长度，例如下面的代码：

```
String type[] = {"乒乓球","篮球","羽毛球","排球","网球"};
```

在上面的语句中，声明了一个数组**type**，虽然没有特别指名**type**的长度，但由于括号里的数值有5个，编译器会分别依次为各元素指定存放位置，例如，**type[0]**="乒乓球"，**type[1]**="篮球"。



2.1.7 数组的创建与应用

2. 二维数组

在Java语言中，实际上并不存在称为“二维数组”的明确结构，而二维数组实际上是指数组元素为一维数组的一维数组。声明二维数组语法格式如下：

```
数据类型 数组名[][] = new 数据类型[个数][个数];
```

例如下面的代码：

```
int array[][] = new int [5][6];
```

上述语句声明了一个二维数组，其中[5]表示该数组有（0~4）5行，每行有（0~5）6个元素，因此该数组有30个元素。



2.1.7 数组的创建与应用

对于二维数组元素的赋值，同样可以在声明时进行，例如：

```
int number[][] = {{20,25,26,22},{22,23,25,28}};
```

在上面的语句中，声明了一个整型的**2行4列**的数组，同时进行赋值，结果如下：

```
number[0][0] = 20; number[0][1] = 25;  
number[0][2] = 26; number[0][3] = 22;  
number[1][0] = 22; number[1][1] = 23;  
number[1][2] = 25; number[1][3] = 28;
```





2.1.8 集合类的应用

集合类的作用和数组类似，也可以保存一系列数据，但是集合类的优点是可以方便地对集合内的数据进行查询、增加、删除和修改等操作。本节将介绍**List**集合类。

List集合为列表类型，列表的主要特征是存放其中的对象以线性方式存储。**List**集合包括**List**接口以及**List**接口的所有实现类。**List**接口的常用实现类有**ArrayList**、**Vector**和**LinkedList**，这3个类拥有以下两个特征：

- (1) 允许内容有重复的元素存在；
- (2) 内部元素有特定的顺序。

在使用**List**集合时，通常情况下，声明为**List**类型，实例化时根据实际情况的需要，实例化为**ArrayList**或**LinkedList**，例如：



2.1.8 集合类的应用

```
List<String> l = new ArrayList<String>();  
List<String> l2 = new LinkedList<String>();
```

1. ArrayList类

ArrayList类实现了List接口，由ArrayList类实现的List集合采用数组结构保存对象。数组结构的优点是便于对集合进行快速地随机访问，如果经常需要根据索引位置访问集合中的对象，使用由ArrayList类实现的List集合的效率较好。ArrayList类的常用方法如下表所示。



ArrayList类的常用方法

方法名称	方法含义
add(int index, Object obj)	用来向集合的指定位置添加元素，其他元素的索引位置相对后移一位。索引位置从0开始。
addAll(int, Collection coll)	用来向集合的指定索引位置添加指定集合中的所有对象
remove(int index)	用来清除集合中指定索引位置的对象
set(int index, Object obj)	用来将集合中指定索引位置的对象修改为指定的对象
get(int index)	用来获得指定索引位置的对象
indexOf(Object obj)	用来获得指定对象的索引位置。当存在多个时，返回第一个的索引位置；当不存在时，返回 -1
lastIndexOf(Object obj)	用来获得指定对象的索引位置。当存在多个时，返回最后一个的索引位置；当不存在时，返回 -1
listIterator()	用来获得一个包含所有对象的ListIterator型实例
listIterator(int index)	用来获得一个包含从指定索引位置到最后的ListIterator型实例



2.1.8 集合类的应用

【例2-9】 ArrayList类示例

2. Vector类

Vector类是一元集合，可以加入重复数据，它的作用和数组类似，可以保存一系列数据，它的优点是可以很方便地对集合内的数据进行查找、增加、修改和删除等操作。下面介绍**Vector**类的常用方法。**Vector**类的常用方法如下表所示。



Vector类的常用方法

方法名称	方法含义
add(int index,Object element)	在指定的位置添加元素
addElementAt(Object obj,int index)	在Vector类的结尾添加元素
size()	返回Verctor类的元素总数
elementAt(int index)	取得特定位置的元素，返回值为整型
setElementAt(object obj,int index)	重新设定指定位置的元素
removeElementAt(int index)	删除指定位置的元素

【例2-10】 Vector类示例





2.1.9 异常处理语句

在Java语言中，处理异常的语句有4种：try...catch语句、finally语句、throw语句及throw语句。

1. try...catch语句

在Java语言中，用try...catch语句来捕获异常，代码格式如下：

```
try{
    /*可能出现异常状况的代码*/
}catch (IOException e){
    /*处理输出输入出现的异常*/
}catch(SQLException e){
    /*处理操作数据库出现的异常*/
}
```



2.1.9 异常处理语句

在上述代码中，**try**块用来监视这段代码运行过程中是否发生异常，若发生则产生异常对象并抛出；**catch**用于捕获异常并处理它。

2. **finally**语句

由于异常将程序中断执行，这会使得某些不管在任何情况下都必须执行步骤被忽略，从而影响程序的健壮性。**finally**语句的作用就是不管捕获的异常是否出现，都会执行**finally**代码块。

3. **throw**语句



2.1.9 异常处理语句

当程序发生错误而无法处理时，会抛出对应的异常对象。除此之外，在某些代码中，可能会要自行抛出异常。例如，在捕捉异常并处理结束后，再将异常抛出，让下一层异常处理区块来抛出；另一个情况是重新包装异常，将捕捉到的异常以自己定义的异常对象加以包装抛出。如果要自行抛出异常，可以使用**throw**关键字，并生成指定的异常对象。例如下面的代码：

```
throw new ArithmeticException();
```

4. throws语句



2.1.9 异常处理语句

```
返回类型 方法名（参数表） throws 异常类型表{  
    方法体  
}
```

一个方法可能会出现多种异常，throws子句语句允许声明抛出多个异常，例如下面的代码：

```
public void methodServlet(int number) throws  
    NumberFormatException, IOException{  
  
    .....  
}
```



2.1.9 异常处理语句

异常声明是接口（这里的接口是指概念上的程序接口）的一部分。根据异常声明，方法调用者了解到被调用方法可能抛出的异常，从而采取相应的措施：捕获异常并处理异常或者声明继续抛出异常。

如果不确定这个方法会抛出哪种异常，那么可以直接抛出**Exception**异常，例如，下面的代码：

```
public void methodServlet(int number) throws Exception{  
.....  
}
```

注意：throw和throws关键字尽管只有一个字母之差，却有着不同的用途，注意不要将两者混淆。





2.2.1 JavaScript脚本语言概述

JavaScript是一种基于对象和事件驱动并具有安全性能的解釋型脚本语言，在Web应用中得到了非常广泛的应用。它不但可以用于编写客户端的脚本程序，由Web浏览器解释执行，而且还可以编写在服务器端执行的脚本程序，在服务器端处理用户提交的信息并动态地向浏览器返回处理结果，通常在JSP中应用JavaScript编写客户端脚本程序。



课件制作人：王国辉



2.2.2 在JSP中引入JavaScript

通常情况下，在JSP中引入JavaScript有以下两种方法，一种是在JSP页面中直接嵌入JavaScript，另一种是链接外部JavaScript。

1. 在页面中直接嵌入JavaScript

在Web页面中，可以使用<script>...</script>标记对封装脚本代码，当浏览器读取到<script>标记时，将解释执行其中的脚本。

在使用<script>标记时，还需要通过其language属性指定使用的脚本语言。例如，在<script>中指定使用JavaScript脚本语言的代码如下：



2.2.2 在JSP中引入JavaScript

```
<script language="javascript">...</script>
```

2. 链接外部JavaScript

在JSP中引入JavaScript的另一种方法是采用链接外部JavaScript文件的形式。如果脚本代码比较复杂或是同一段代码可以被多个页面所使用，则可以将这些脚本代码放置在一个单独的文件中，该文件的扩展名为.js，然后在需要使用该代码的Web页面中链接该JavaScript文件即可。在Web页面中链接外部JavaScript文件的语法格式如下：

```
<script language="javascript" src="javascript.js"></script>
```

说明：在外部JS文件中，不需要将脚本代码用<script>和</script>标记括起来。



2.2.3 JavaScript的数据类型 与运算符

- 数据类型 ✓
- 变量 ✓
- 运算符 ✓



数据类型

JavaScript有6种数据类型，如下表所示。

类 型	含 义		说 明	示 例
int	数值	整型	整数，可以为正数、负数或0	17, -80, 0
float		浮点型	浮点数，可以使用实数的普通形式或科学计数法表示	3.14159.27, 6.16e4
string	字符串类型		字符串，是用单引号或双引号括起来的一个或多个字符	'wgh', "平平淡淡才是真"
boolean	布尔型		只有true或false两个值	true, false
object	对象类型			
null	空类型		没有任何值	
undefined	未定义类型		指变量被创建，但未赋值时所具有的值	



变量

变量是指程序中一个已经命名的存储单元，它的主要作用就是为数据操作提供存放信息的容器。在JavaScript中，可以使用命令**var**声明变量，语法格式如下：

```
var variable;
```

在声明变量的同时也可以对变量进行赋值：

```
var variable=11;
```

由于JavaScript采用弱类型的形式，所以在声明变量时，不需要指定变量的类型，而变量的类型将根据其变量赋值来确定。例如：

```
var variable=17;
```

//数值型

```
var str="爱护地球";
```

//字符型



变量

但是变量命名必须遵循以下规则：

（1）必须以字母或下划线开头，中间可以是数字、字母或下划线，但是不能有空格或加号、减号等符号。

（2）不能使用JavaScript中的关键字。JavaScript的关键字如下表所示。

abstract	continue	finally	instanceof	private	this
boolean	default	float	int	public	throw
break	do	for	interface	return	typeof
byte	double	function	long	short	true
case	else	goto	native	static	var
catch	extends	implements	new	super	void
char	false	import	null	switch	while
class	final	in	package	synchronized	with



变量

注意：关键字同样不可用作函数名、对象名及自定义的方法名等。





运算符

在JavaScript中提供了算术运算符、关系运算符、逻辑运算符、字符串运算符、位操作运算符、赋值运算符和条件运算符等7种运算符。下面进行详细介绍。

(1) 算术运算符

算术运算符等同于数学运算，即在程序中进行加、减、乘、除等运算。在JavaScript中常用的算术运算符如下表所示。

运算符

运算符	描 述	示 例
+	加运算符	1+6 //返回值为7
-	减运算符	5-2 //返回值为3
*	乘运算符	7*3 //返回值为21
/	除运算符	9/3 //返回值为3
%	求模运算符	6%4 //返回值为2
++	自增运算符。该运算符有两种情况：i++（在使用i之后，使i的值加1）；++i（在使用i之前，先使i的值加1）	i=1; j=i++ //j的值为1，i的值为2 i=1; j=++i //j的值为2，i的值为2
--	自减运算符。该运算符有两种情况：i--（在使用i之后，使i的值减1）；--i（在使用i之前，先使i的值减1）	i=6; j=i-- //j的值为6，i的值为5 i=6; j=--i //j的值为5，i的值为5





运算符

(2) 关系运算符

关系运算符的基本操作过程是：首先对操作数进行比较，这个操作数可以是数字也可以是字符串，然后返回一个布尔值true或false。JavaScript支持的常用关系运算符与Java中的常用关系运算符相同。

(3) 逻辑运算符

逻辑运算符返回一个布尔值，通常和比较运算符一起使用，用来表示复杂的比较运算，常用于if、while和for语句中。JavaScript中常用的逻辑运算符如下表所示。



运算符

运算符	描 述	运算符	描 述	运算符	描述
!	逻辑非	&&	逻辑与		逻辑或

(4) 字符串运算符

字符串运算符是用于两个字符型数据之间的运算符，除了比较运算符外，还可以是+和+=运算符。其中，+运算符用于连接两个字符串（例如，"World"+"Dream"），而+=运算符则连接两个字符串，并将结果赋给第一个字符串（例如，`var a="One";a+="Dream";`）。

(5) 赋值运算符



运算符

最基本的赋值运算符是等于号“=”，用于对变量进行赋值，而其他运算符可以和赋值运算符“=”联合使用，构成组合赋值运算符。JavaScript支持的常用赋值运算符与Java中的常用赋值运算符相同。

(6) 位操作运算符

位操作运算符用于对数值的位进行操作，如向左或向右移位等。JavaScript中常用位操作运算符如下表所示。

运算符	描 述	运算符	描 述	运算符	描 述
&	与运算符		或运算符	^	异或运算符
<<	左移	>>	带符号右移	>>>	填0右移



运算符

(7) 条件运算符

条件运算符是JavaScript支持的一种特殊的3目运算符，同Java中的3目运算符类似，其语法格式如下：

操作数?结果1:结果2

如果“操作数”的值为true，则整个表达式的结果为“结果1”，否则为“结果2”。



2.2.4 JavaScript的流程控制语句

- if条件判断语句 ✓
- for循环语句 ✓
- while循环语句 ✓
- do...while循环语句 ✓
- switch语句 ✓





if条件判断语句

对变量或表达式进行判定并根据判定结果进行相应的处理，可以使用if语句。if语句的语法格式如下：

```
if(条件表达式){  
    语句序列1 //条件满足时执行  
}  
else{  
    语句序列2 //条件不满足时执行  
}
```

执行上述if语句时，首先计算“条件表达式（任意的逻辑表达式）”的值，如果为**true**，就执行“语句序列1”，执行完毕后结束该if语句；否则执行“语句序列2”，执行后同样结束该if语句。





for循环语句

for语句是JavaScript语言中应用比较广泛的条件语句。通常for语句使用一个变量作为计数器来执行循环的次数，这个变量就称为循环变量。for语句的语法格式如下：

```
for(循环变量赋初值;循环条件;循环变量增值){  
    循环体  
}
```

循环变量赋初值：一条初始化语句，用来对循环变量进行初始化赋值。

循环条件：一个包含比较运算符的表达式，用来限定循环变量的边限。如果循环变量超过了该边限，则停止该循环语句的执行。

循环变量增值：用来指定循环变量的步幅。





for循环语句

for语句可以使用break语句来中止循环语句的执行。
break语句默认情况下是终止当前的循环语句。





while循环语句

while循环语句是另一种基本的循环语句，其结构和**for**循环语句有些类似，但是**while**语句不包含循环变量的初始化及循环变量的步幅。其语法格式如下：

```
while (条件表达式){  
    循环体  
}
```

使用**while**语句时，必须先声明循环变量并且在循环体中指定循环变量的步幅，否则**while**语句将成为一个死循环。





do...while循环语句

do...while循环语句和while循环语句非常相似，所不同的是它是在循环底部检测循环表达式，而不是像while循环语句那样在循环顶部进行检测。这就保证了循环体至少被执行一次。do...while语句的语法格式如下：

```
do{  
    循环体  
} while (条件表达式);
```

【例2-11】 分别利用for、while和do...while循环语句将数字7格式化为00007，并输出到页面上。





switch语句

switch是典型的多路分支语句，其作用与嵌套使用**if**语句基本相同，但**switch**语句比**if**语句更具有可读性，而且**switch**语句允许在找不到一个匹配条件的情况下执行默认的一组语句。**switch**语句的语法格式如下：

```
switch (expression){  
    case judgement1:  
        statement1;  
        break;  
    case judgement2:  
        statement2;  
        break;  
    ...  
    default:  
        defaultstatement;  
        break;  
}
```



switch语句

expression: 任意的表达式或变量。

judgement: 任意的常数表达式。当**expression**的值与某个**judgement**的值相等时，就执行此**case**后的**statement**语句，如果**expression**的值与所有的**judgement**的值都不相等时，则执行**default**后面的**defaultstatement**语句。

break: 用于结束**switch**语句，从而使JavaScript只执行匹配的分支。如果没有了**break**语句，则该**switch**语句的所有分支都将被执行，**switch**语句也就失去了使用的意义。





2.2.5 函数的定义和调用

在JavaScript中，函数可以分为定义和调用两部分。

1. 函数的定义

在JavaScript中，定义函数最常的方法是通过function语句实现，其语法格式如下：

```
function functionName([parameter1, parameter2,...]){  
    statements  
    [return expression]  
}
```



2.2.5 函数的定义和调用

functionName: 必选，用于指定函数名。在同一个页面中，函数名必须是唯一的，并且区分大小写。

parameter1, parameter2, ...: 可选，用于指定参数列表。当使用多个参数时，参数间使用逗号进行分隔。一个函数最多可以有**255**个参数

statements: 必选，是函数体，用于实现函数功能的语句

return expression: 可选，用于返回函数值。
expression为任意的表达式、变量或常量



2.2.5 函数的定义和调用

2. 函数的调用

函数的调用比较简单，如果要调用不带参数的函数，则使用函数名加上括号即可；如果要调用的函数带参数，则在括号中加上需要传递的参数，如果包含多个参数，各参数间用逗号分隔。

如果函数有返回值，那么可以使用赋值语句将函数值赋给一个变量。

说明：在JavaScript中，由于函数名区分大小写，所以在调用函数时，也需要注意函数名的大小写。





2.2.6 事件

1. 事件概述

JavaScript与Web页面之间的交互是通过用户操作浏览器页面时触发相关事件来实现的。例如，在页面载入完毕时，将触发load（载入）事件；当用户单击按钮时，将触发按钮的click事件等。

用于响应某个事件而执行的处理程序称为事件处理程序，例如，当用户单击按钮时，将触发按钮的事件处理程序onClick。事件处理程序有以下两种分配方式。

（1）在JavaScript中分配事件处理程序




2.2.6 事件

在JavaScript中分配事件处理程序，首先需要获得要处理对象的引用，然后将要执行的处理函数赋值给对应的事件处理程序。例如：

```

<script language="javascript">
var img=document.getElementById("img_download");
img.onclick=function(){
    alert("单击了图片");
}
</script>
```

在页面中加入这段代码并运行，则当单击图片时，将弹出“单击了下载图片”对话框。



2.2.6 事件

注意：在JavaScript中分配事件处理程序时，事件处理程序名称必须小写，才能正确响应事件。


(1) 在JavaScript中分配事件处理程序

(2) 在HTML中分配事件处理程序

在HTML中分配事件处理程序，只需要在HTML标记中添加相应的事件处理程序的属性，并在其中指定作为属性值的代码或是函数名称即可。例如：

```

```

在页面中加入上面的代码，并运行，则当单击图片时，将弹出“您单击了图片”对话框。

2.2.6 事件

2. 事件类型

多数浏览器内部对象都拥有很多事件，下面的表中将给出常用的事件、事件处理程序及何时触这些事件处理程序。

事件	事件处理程序	何时触发
blur	onblur	元素或窗口本身失去焦点时触发
change	onchange	选中<select>元素中的选项或其他表单元素失去焦点时，并且在其获取焦点后内容发生过改变时触发
click	onclick	单击鼠标左键时触发
focus	onfocus	任何元素或窗口本身获得焦点时触发
keydown	onkeydown	键盘键被按下时触发，如果一直按着某键，则会不断触发；当返回false时，取消默认动作
load	onload	页面完全载入后，在window对象上触发；所有框架都载入后，在框架集上触发；标记指定的图像完全载入后，在其上触发；或<object>标记指定的对象完全载入后，在其上触发
select	onselect	选中文本时触发
submit	onsubmit	单击提交按钮时，在<form>上触发
unload	onunload	页面完全卸载后，在window对象上触发；或者所有框架都卸载后，在框架集上触发

2.2.7 JavaScript常用对象的应用

JavaScript提供了一些内部对象，下面将介绍最常用的String，Date和window对象。

JavaScript提供了一些内部对象，下面将介绍最常用的String，Date和window对象。

1. String对象

String对象是动态对象，需要创建对象实例后才能引用它的属性和方法。在创建一个String对象变量时，可以使用new运算符与来创建，也可以直接将字符串赋给变量。例如strValue="hello"与strVal=new String("hello")是等价的。

String对象的常用属性和方法如下表所示。

2.2.7 JavaScript常用对象的应用

属性/方法	说 明
length	用于返回String对象的长度
split(separator,limit)	用separator分隔符将字符串划分成子串并将其存储到数组中，如果指定了limit，则数组限定为limit给定的数，separator分隔符可以是多个字符或一个正则表达式，它不作为任何数组元素的一部分返回
substr(start,length)	返回字符串中从startIndex开始的length个字符的子字符串
substring(from,to)	返回以from开始、以to结束的子字符串
replace(searchValue,replaceValue)	将searchValue换成replaceValue并返回结果
charAt(index)	返回字符串对象中的指定索引号的字符组成的字符串，位置的有效值为0到字符串长度减1的数值；一个字符串的第一个字符的索引位置为0，第二个字符位于索引位置1，依次类推；当指定的索引位置超出有效范围时，charAt方法返回一个空字符串
toLowerCase()	返回一个字符串，该字符串中的所有字母都被转换为小写字母
toUpperCase()	返回一个字符串，该字符串中的所有字母都被转换为大写字母

2.2.7 JavaScript常用对象的应用

2. Date对象

Date对象是一个有关日期和时间的对象。它具有动态性，即必须使用**new**运算符创建一个实例。例如：

```
mydate=new Date();
```

Date对象没有提供直接访问的属性，只具有获取和设置日期和时间的方法。**Date**对象的方法如下表所示。

2.2.7 JavaScript常用对象的应用

获取日期和 时间的方法	说 明	设置日期和 时间的方法	说 明
getFullYear()	返回用4位数表示的年份	setFullYear()	设置年份，用4位数表示
getMonth()	返回月份（0~11）	setMonth()	设置月份（0~11）
getDate()	返回日数（1~31）	setDate()	设置日数（1~31）
getDay()	返回星期（0~6）	setDay()	设置星期（0~6）
getHours()	返回小时数（0~23）	setHours()	设置小时数（0~23）
getMinutes()	返回分钟数（0~59）	setMinutes()	设置分钟数（0~59）
getSeconds()	返回秒数（0~59）	setSeconds()	设置秒数（0~59）
getTime()	返回Date对象的内部毫秒表示	setTime()	使用毫秒形式设置Date对象

2.2.7 JavaScript常用对象的应用

3. window对象

window对象是浏览器（网页）的文档对象模型结构中最高级的对象，它处于对象层次的顶端，提供了用于控制浏览器窗口的属性和方法。由于window对象使用十分频繁，又是其他对象的父对象，所以在使用window对象的属性和方法时，JavaScript允许省略window对象的名称。

window对象的常用属性如下表所示。



Window对象的常用属性

属 性	描 述
frames	表示当前窗口中所有frame对象的集合
location	用于代表窗口或框架的Location对象，如果将一个URL赋予给该属性，那浏览器将加载并显示该URL指定的文档
length	窗口或框架包含的框架个数
history	对窗口或框架的History对象的只读引用
name	用于存放窗口的名字
status	一个可读写的字符，用于指定状态栏中的当前信息
parent	表示包含当前窗口的父窗口
opener	表示打开当前窗口的父窗口
closed	一个只读的布尔值，表示当前窗口是否关闭；当浏览器窗口关闭时，表示该窗口的window对象并不会消失，不过它的closed属性被设置为true



Window对象的常用方法

方 法	描 述
alert()	弹出一个警告对话框
confirm()	显示一个确认对话框，单击“确认”按钮时返回true，否则返回false
prompt()	弹出一个提示对话框，并要求输入一个简单的字符串
close()	关闭窗口
focus()	把键盘的焦点赋予给顶层浏览器窗口，在多数平台上，这将使用窗口移到最前边
open()	打开一个新窗口
setTimeout(timer)	在经过指定的时间后执行代码
clearTimeout()	取消对指定代码的延迟执行
resizeBy(offsetx,offsety)	按照指定的位移量设置窗口的大小
print()	相当于浏览器工具栏中的“打印”按钮
setInterval()	周期执行指定的代码
clearInterval()	停止周期性地执行代码

2.2.7 JavaScript常用对象的应用

【例2-12】 window对象示例

通过按钮打开一个新窗口，并在新窗口的状态栏中显示当前年份。

