第4章
C# 语言基础

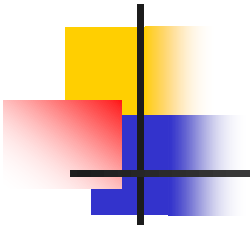
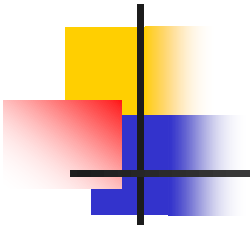
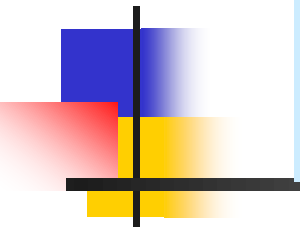


Table of contents

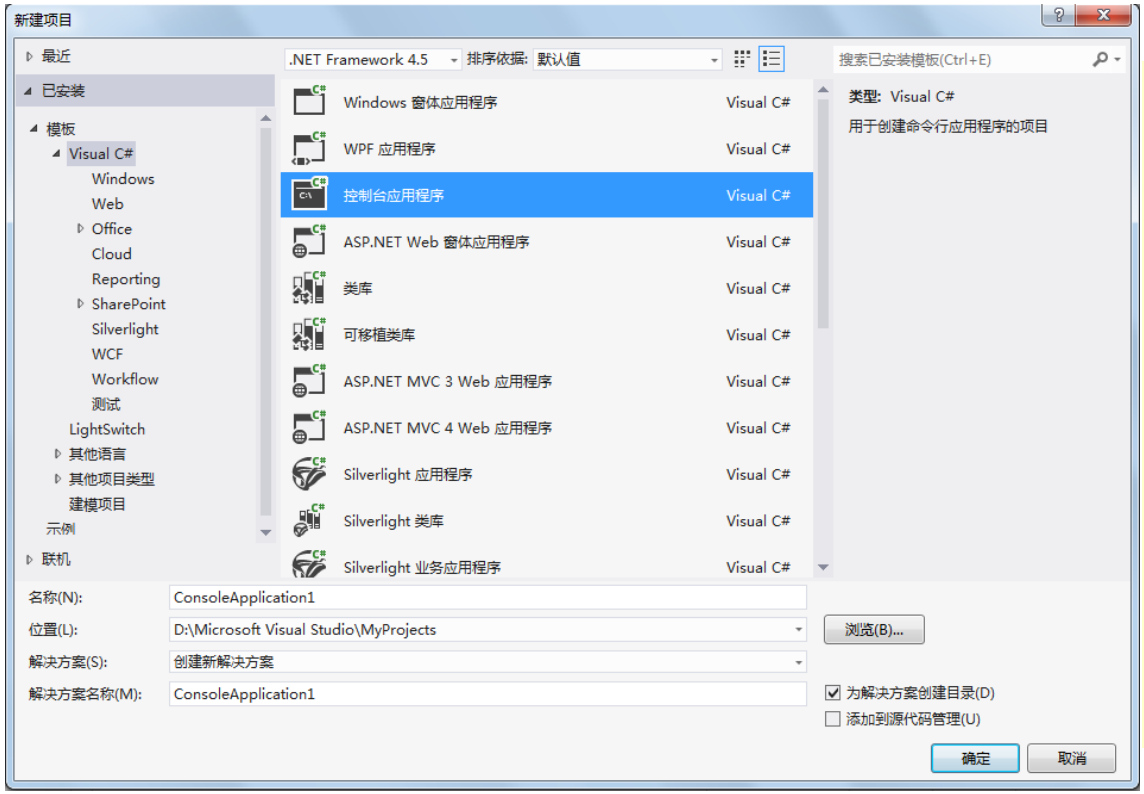


- C#（读作“C sharp”）是一种简单的、面向对象和类型安全的编程语言，是微软公司专门为.NET框架量身定做的编程语言，是最适合开发.NET应用的语言。
- C#的类型就是.NET框架所提供的类型，C#本身无类库，而是直接使用.NET所提供的类库。
- C#具有面向对象编程语言的一切特性。
- Visual C# 是 Microsoft 对 C# 语言的实现。



4.1 创建一个简单的C#程序

- 首先启动Visual C#，在菜单中选择“文件”→“新建项目”。在“新建项目”对话框中选择“控制台应用程序”，系统将自动创建一个控制台程序 Program.cs。




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

"Hello World" program in C#

```
class HelloWorld
{
static void Main( )
    {
    // Use the system console object
    System.Console.WriteLine("Hello World");
    }
}
```

A second program in C#

```
using System;
class Hello{
   public static void Main() {
      Console.WriteLine("Enter your name:");
      String name=Console.ReadLine( );
      Console.WriteLine("Welcome : {0}",name);
      }
}
```

A second program in C#

向控制台输出的几种方式

```
Console.WriteLine();     // 相当于换行
```

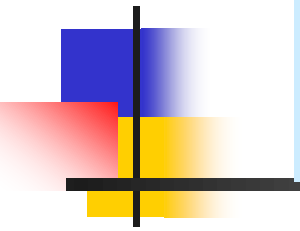
```
Console.WriteLine(要输出的值);     // 输出一个值
```

```
Console.WriteLine("格式字符串", 变量列表);
```


```
Console.WriteLine ("Hello World!");
```

```
string course = "C#";
Console.WriteLine(course);
Console.WriteLine("我的课程名称是：" + course);
Console.WriteLine("我的课程名称是：{0}",course);
```

与 Java 用法相同



4.2 C#基本语法

4.2.1 C#数据类型

- C# 的数据类型分为两大类：

  值类型(Value Types)

  引用类型(Reference Types)

- 值类型和引用类型的区别在于：

值类型变量直接包含它们的数据，而引用类型变量则存储对于对象的引用。

◆ **值类型包括：**

简单类型（simple type）
结构类型（struct）
枚举类型（enumeration）

◆ **引用类型包括：**

对象（Object）
类（Class）
指代（delegate）
接口（interface）
数组（array）
字符串 （String）

值类型

表4-1 C#的简单类型

类型	关键字	大小/精度	范围	.NET类型	后缀
整型	byte	无符号8位整数	0~255	System.Byte	无
	sbyte	有符号8位整数	-128~127	System.SByte	无
	short	有符号16位整数	-32,768~32,767	System.Int16	无
	ushort	无符号16位整数	0~65,535	System.UInt16	无
	int	有符号32位整数	-2,147,483,648~2,147,483,647	System.Int32	无
	uint	无符号32位整数	0~4,294,967,295	System.UInt32	U或u
	long	有符号64位整数	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	System.Int64	L或l
	ulong	无符号64位整数	0 ~ 0xffffffffffffffff	System.UInt64	UL
浮点型	float	32位浮点值，7位精度	$\pm 1.5 * 10^{-45}$ ~ $\pm 3.4 * 10^{38}$	System.Single	F或f
	double	64位浮点值，15~16位精度	$\pm 5.0 * 10^{-324}$ ~ $\pm 1.8 * 10^{308}$	System.Double	D或d
字符型	char	16位Unicode字符		System.Char	无
布尔型	bool	8位空间，1位数据	true或 false	System.Boolean	无
小数型	decimal	128位数据类型，28~29位精度	$\pm 1.0 \times 10^{-28}$ ~ $\pm 7.9 \times 10^{28}$	System.Decimal	M或m

// 简单类型（整型）示例

```
<%@page language="C#"%>
<script language="C#" runat="server">
void page_load(object serder, EventArgs e)
{
    labContent1.Text=(0x10).ToString();
    ushort i;
    int j;
    j=2147483647;
    i=(ushort)j;
    labContent2.Text=i.ToString();
    labContent3.Text=j.ToString();
}
</script>
<html>
<body>
<asp:label runat=server id=labContent1/><br>
<asp:label runat=server id=labContent2/><br>
<asp:label runat=server id=labContent3/><br>
</body>
</html>
```

结构类型（struct）

结构的定义：

```
struct Point {
    public Double x , y , z ;
}
```

结构类型的使用：

```
Point p ;
p.x=100 ;
p.y=200 ;
p.z=300 ;
```

结构类型可以包含数据成员和函数成员：

```
struct 结构名 {
   public 数据类型 域名;
            ... ...
   public void 方法名 {
            //方法的实现
   }
};
```

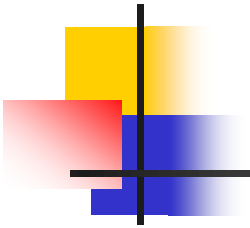
类和结构的主要区别

- **结构**是比类更简单的对象。与类一样，可以包含各种成员，也可以实现接口。
- 结构适合表示如点、矩形等简单的数据结构，这比使用类可以降低成本、效率更高。
- 结构和类最重要的差别在于，结构是“value types”，而不是引用类型，结构不支持继承。
- 结构的实例化可以使用new，也可以不使用new（所有的域默认为0，false，null等）。而类的实例化必须使用new。

枚举类型（Enumerations）

- 枚举类型是一组已命名的数值常量。
- C# 中的枚举包含与值关联的数字。默认情况下，将 0 赋给第一个元素，然后对每个后续的枚举元素按 1 递增
- 在初始化过程中可重写默认值

```
public enum WeekDays  {
            Monday,
            Tuesday,
            Wednesday=20,
            Thursday,
            Friday=5
}
```



引用类型

（1）对象类型（Object）

在C#中，所有的类型都可以看成是对象，对象类型object是一切类型的基类型。object类型对应的.NET系统类型是System.Object。

（2）类类型（Class）

类类型可以包含数据成员、函数成员和嵌套类型。数据成员为常量、字段和事件。函数成员包括方法、属性、索引、操作符、构造函数和析构函数。

一个类可以派生多重接口。

（3）指代（delegate）

指代类型可用于将方法用特定的签名封装。用户可以在一个指代实例中同时封装静态方法和实例方法。

指代的声明格式如下：

delegate 返回类型 代理名（参数列表）

例如，

Public delegate double MyDelegate(double x)； //声明了一个代理类型

MyDelegate d； //声明该代理类型的变量

对代理进行实例化:

MyDelegate d1 = new MyDelegate(System.Math.Sqrt);

MyDelegate d2 = new MyDelegate(obj.myMethod());

（4）接口（interface）

一个接口定义一个只有抽象成员的引用类型。该类型不能实例化对象，但可以从它派生出类。

```
interface 接口名 {
    //接口成员的定义
};
```

以下是一个接口定义例子。

```
Public interface MyInterf  {
      void showface();
}
```

（5）数组类型（Array）

- 数组是一组类型相同的有序数据。数组可以存储整数对象、字符串对象或任何一种用户提出的对象。
- 声明数组时并不需要明确指定其大小，这样反而会出现编译错误。声明多维数组时每一维之间要用逗号隔开。

```
string [ ]  myarray ={ “ab”,“aa”,“c”,“ddd”};   //定义一维数组
string [ , ]  twarray;                          //定义二维数组
```

- 使用new关键字新建数组时，若指定了数组的大小，则花括号{ }中指定的元素个数必须相符，否则出错。若未指定大小，则根据{ }中元素个数自动分配大小。

例4-1 使用一维数组和二维数组（04-01.aspx）

```
<%@page language="C#"%>
<script language="C#" runat="server">
void page_load(object serder, EventArgs e)
{
        int[] myArray1=new int[5] {1,2,3,4,5};
        int[,] myArray2=new int[2,3] {{1,2,3},{4,5,6}};
        labContent1.Text=myArray1[1].ToString();
        labContent2.Text=myArray2[1,2].ToString();
}
</script>
<html>
<body>
<asp:label runat=server id=labContent1/><br>
<asp:label runat=server id=labContent2/><br>
</body>
</html>
```

（6）字符串类型（String）

- 字符串类型就是string类型。它是由一系列字符组成的。所有的字符串都是写在双引号中的。

  例如，“this is a book.”和“hello”都是字符串。

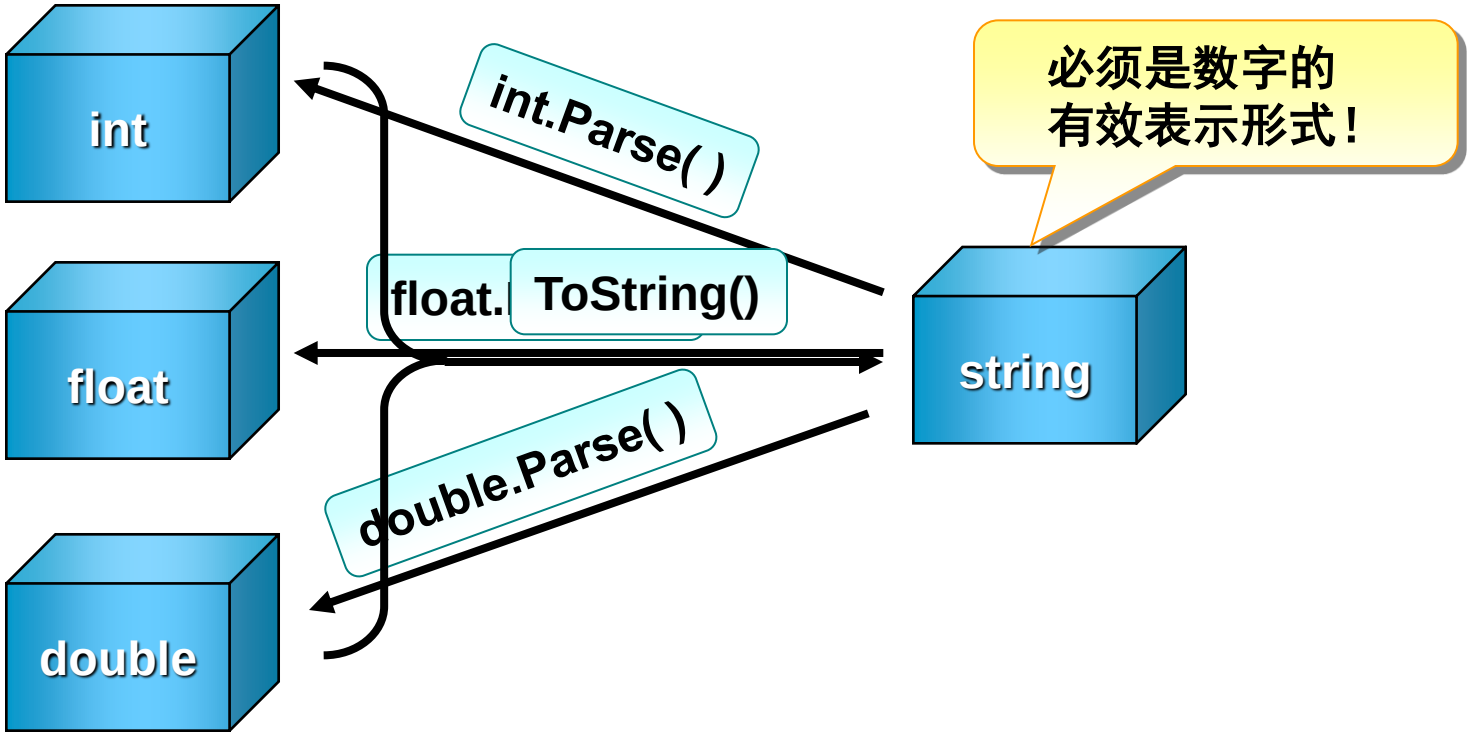
- "A"和’A’有本质的不同，前者是string类型，后者是char类型。

数据类型之间的转换

- **隐式转换**：数据转换的过程是自动进行的，不需要程序进行任何额外的工作。必须保证转换后不会导致数据精度的损失，否则不允许。
- **显式转换**：又称强制转换。它告知C#的编译器必须按照程序的要求进行这种类型转换，即使发生数据精度的损失也在所不惜。
- 相关原则：
  - 不能在数值类型和**bool**值之间进行转换。
  - 不允许转换的结果超出数据类型的表示范围。

类型转换

- 字符串和数值型的互相转换



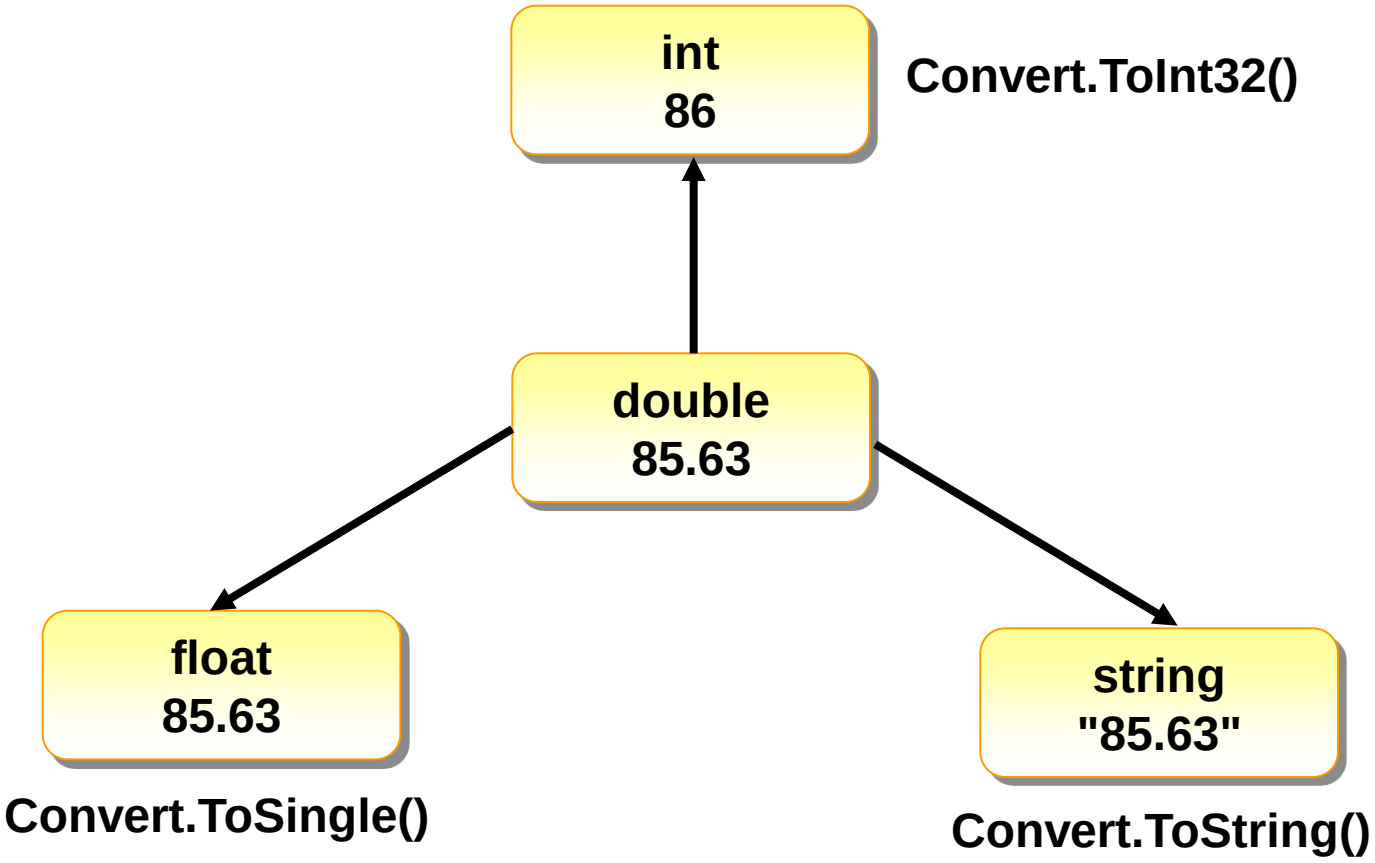
ToString()把数据转换成字符串

```
1:<%@page language="C#"%>
2:<script language="C#" runat="server">
3:void page_load(object serder, EventArgs e)
4:{
5:  double d=123.4;
6:  bool b=true;
7:  DateTime dt=DateTime.Now;
8:  labContent1.Text=d.ToString();
9:  labContent2.Text=b.ToString();
10: labContent3.Text=dt.ToString()+"<br>";
11: labContent3.Text=labContent3.Text+dt.ToString("d")+"<br>";
12: labContent3.Text=labContent3.Text+dt.ToString("D")+"<br>";
13: labContent3.Text=labContent3.Text+dt.ToString("G")+"<br>";
14: labContent3.Text=labContent3.Text+dt.ToString("T")+"<br>";
15: labContent3.Text=labContent3.Text+dt.ToString("t");
16:}
17:</script>
18:<html>
19:<body>
20:<asp:label runat=server id=labContent1/><br>
21:<asp:label runat=server id=labContent2/><br>
22:<asp:label runat=server id=labContent3/><br>
23:</body>
24:</html>
```

不同的日期输出格式

类型转换

- 使用 Convert： Convert.ToXxx(object value)



使用Convert类实现数据转换

函数	功能
Convert.ToBoolean()	转换成bool型，字符串必须为true或false
Convert.ToChar()	转换成为char型
Convert.ToDateTime()	转换成为日期型
Convert.ToDecimal()	转换成为Decimal型
Convert.ToInt32()	转换成为int型

Convert类的使用

```
<script language="C#" runat="server">
void page_load(object serder, EventArgs e)  {
    int i=0, j=-3;
    string str="fAlSe11";
    bool b1,b2,b3;
    b1=Convert.ToBoolean(i);
    labContent1.Text=b1.ToString();
    b2=Convert.ToBoolean(j);
    labContent2.Text=b2.ToString();
    b3=Convert.ToBoolean(str);
    labContent3.Text=b3.ToString();
}
</script>
<asp:label runat=server id=labContent1/><br>
<asp:label runat=server id=labContent2/><br>
<asp:label runat=server id=labContent3/><br>
```

4.2.2 运算符和表达式

- 算术操作符 Mathematical Operators
- 赋值操作符 Assignment Operator
- 比较运算符 Relational Operators
- 逻辑操作符 Logical Operators
- 条件运算符 Conditional Operators
- 位运算符

算术运算符（op：操作数）

运算符	说明	表达式
+	加法运算（若操作数是字符串，则为字符串连接符）	op1 + op2
-	减法运算	op1 - op2
*	乘法运算	op1 * op2
/	除法运算	op1 / op2
%	求余数	op1 % op2
++	将操作数加 1	op++，++op
--	将操作数减 1	op-- ，--op
~	将一个数按位取反	~op

赋值运算符（op：操作数）

运算符	说明	表达式
=	给变量赋值	op1 = op2
+=	运算结果op1 = op1 + op2	op1 += op2
-=	运算结果op1 = op1 - op2	op1 -= op2
*=	运算结果op1 = op1 * op2	op1 *= op2
/=	运算结果op1 = op1 / op2	op1 /= op2
%=	运算结果op1 = op1 % op2	op1 %= op2

逻辑运算符

运算符	描述
&&	逻辑与（AND）
||	逻辑或（OR）
!	逻辑非（NOT）

位运算符

运算符	描述	运算符	描述
&	位与	~	位非
|	位或	<<	左移
^	位异或	>>	右移

条件运算符

C# 只有一个条件运算符，即三元操作符（?:），它是if-else语句的缩写。

形式如下：

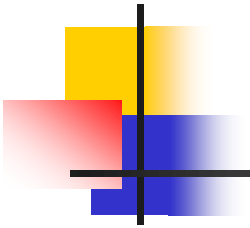
条件表达式？语句1：语句2

例4-2 位运算符（04-02.cs）

```
class 04-02
{
        public static void Main{} {
                byte byte1 = 0x9a;   //binary 10011010,decimal 154
                byte byte2 = 0xdb;   //binary 11011011,decimal 219
                byte result;
                System.Console.WriteLine("byte1 = " + byte1);
                System.Console.WriteLine("byte2 = " + byte2);
                result = (byte)(byte1 & byte2);      //bitwise AND
                System.Console.WriteLine("byte1 & byte2 = " + result);
                result = (byte)(byte1 | byte2);      //bitwise OR
                System.Console.WriteLine("byte1 | byte2 = " + result);

                result = (byte)(byte1 ^ byte2);      //bitwise exclusive OR
                System.Console.WriteLine("byte1 ^ byte2 = " + result);

                result = (byte)~byte1;               //bitwise NOT
                System.Console.WriteLine("~byte1 = " + result);

                result = (byte)(byte1 << 1);         //left shift
                System.Console.WriteLine("byte1 << 1 = " + result);

                result = (byte)(byte1 >> 1);         //right shift
                System.Console.WriteLine("byte1 >> 1 = " + result);
        }
}
```

运算符的优先级（由高到低）

优先级	类别	运算符
1	基本	（x） x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked
2	单目	+ − ! ~ ++x --x (type)x
3	乘法与除法	* / %
4	加法与减法	+ −
5	移位	<< >>
6	关系和类型检测	< > <= >= is as
7	相等	= = !=
8	位与	&
9	位异或	^
10	位或	|
11	逻辑与	&&
12	逻辑或	||
13	三元	?:
14	赋值	= *= /= %= += -= <<= >>= &= ^= |=



4.2.3 程序控制结构

- 分支语句：if 语句，switch 语句
- 循环语句：do--while循环，while循环，
  for循环，foreach循环
- 跳转语句：break，continue，return

if 语句

```
if （条件）{
    执行语句1;
}
else {
   执行语句2;
}
```

```
if （条件）
    执行语句1;
elseif
   执行语句2;
else
   执行语句3;
```

switch…case 语句

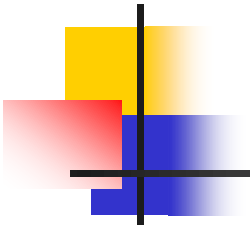
```
switch （控制表达式）{
    case 常量表达式1：
                语句1;
    case 常量表达式2：
                语句2;
    ……
    case 常量表达式n：
                语句n;
    default ：  语句;
}
```

例4-3 if语句示例（04-03.aspx）

```
<%@page language="C#"%>
<script language="C#"runat="server">
void page_load(object serder, EventArgs e)
 {
int intNowHour;
intNowHour=DateTime.Now.Hour;
if (intNowHour<12)
            labContent1.Text = "Good morning,Cindy!";
if (intNowHour=12)
            labContent1.Text = "Good noon,Cindy!";
if (intNowHour>12)
            labContent1.Text = "Good afternoon,Cindy!";
}
</script>
<html>
<body>
<asp:label runat=server id=labContent1 /><br>
</body>
</html>
```

例4-4 switch语句示例（04-04.cs）

```
using System;
class Sample{
    public static void Main()  {
       int myage=10;
       string mystr;
       switch (myage) {
         case 10: mystr="还是小孩!"; break;
         case 25: mystr="可以结婚了!"; break;
         default: mystr="不对吧！你到底多大！"; break;
       }
       Console.WriteLine("小子，你{0}",mystr);
    }
}
```



循环语句

- 循环语句有4种格式：
  - do--while： 直到条件为 True 时循环。
  - while： 当条件为 True 时循环。
  - for： 指定循环次数，使用计数器重复运行语句。
  - foreach： 对于集合中的每项或数组中的每个元素，重复执行。

do-while 循环

```
Do {
    <循环体>
}
While( <条件> );
```

```
using System;
class test{
    public static void Main()    {
       int sum=0;    //初始值设置为0
       int i=1;          //加数初始值为1
       do {
            sum+=i;
            i++;
        } while (i<=100);
       Console.WriteLine("从0到100的和是{0}",sum);
    }
}
```

while 循环

```
While (<条件>)
{
    <循环体>
}
```

```
using System;
class Sample{
    public static void Main()    {
        int sum=0;
        int i=1;
        while (i<=100) {
            sum+=i;
            i++;
        }
        Console.WriteLine("从0到100的和是{0}",sum);
    }
}
```

for循环

```
for（初始化部分；条件部分；更新部分）
{
    <执行语句>;
}
```

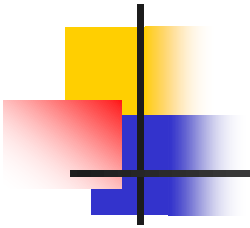
```
using System;
class test{
    public static void Main() {
       int sum = 0;
      for (int i= 1; i <= 100;  i++) {
          sum += i;   }
       Console.WriteLine("从0到100的和是{0}\n",sum);
   }
}
```

foreach循环

```
foreach (type 变量名 in 集合)
    <执行语句>;
```

The **foreach** statement allows you to iterate through all the items in an array or other collection, examining each item in turn.

```
public class Test
{
static void Main( ) {
        int[] intArray;
        intArray = new int[5];
                foreach (int i in intArray) {
                   Console.WriteLine( i.ToString( ) );
                }
        }
}
```



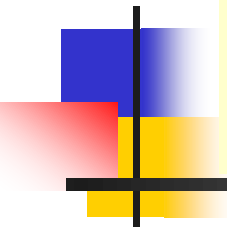
跳转语句

（1）break语句

break语句跳出包含它的switch，while，do，for或for-each语句。

（2）continue语句

continue语句用于结束本次循环，继续下一次循环，但是并不退出循环体。



4.3 类和对象

4.3.1 类和对象的创建

- 类定义对象的属性和行为。类可以认为是一个模板，通过它创建了对象。
- 类的声明

```
public class Car {
  public string model;    //定义域（类的数据成员）
  public void Start() {    //定义方法（类的函数成员）
     System.Console.WriteLine(model + " started");
  }
}
```

- 创建对象：类是创建对象的模板，一旦创建了类，就可以创建那个类的对象。

```
Car myCar;
myCar = new Car();
```

例4-11 类的定义和使用（04-11.cs）

```
using System;
public class Car {
public string model;  //定义域
public string color;
public int yearBuilt;
public void Start() {  //定义方法
    System.Console.WriteLine(model + " started.");
}
public void Stop() {
    System.Console.WriteLine(model + " stopped."); }
}
public class Tester {
   static void Main( ) {
   Car myCar;              //声明一个叫myCar的Car对象的引用
   myCar = new Car();      //创建Car对象, 将它的内存地址保存到myCar中
   myCar.model="Toyota";   //给Car对象的域赋值
   myCar.color="red";
   myCar.yearBuilt =2010;
   System.Console.WriteLine("myCar.model = " + myCar.model);
   System.Console.WriteLine("myCar.color = " + myCar.color);
   System.Console.WriteLine("myCar.yearBuilt = " + myCar.yearBuilt);
   myCar.Start();
   myCar.Stop();
  }
}
```

4.3.2 属性和方法

- 属性的定义通过get和set关键字来实现，get用来定义读取属性时的操作，set用来定义设置属性时的操作。
- 声明方法时，需要指定访问权限、返回值类型、方法名、使用的参数等。
- 通过方法的重载，可以在类中定义方法名相同而参数不同的方法。参数不同指的是参数的个数不同，或参数的类型不同。当一个重载方法被调用时，C# 会根据调用该方法的参数自动调用具体的方法来执行。

4.3.3 构造函数和析构函数

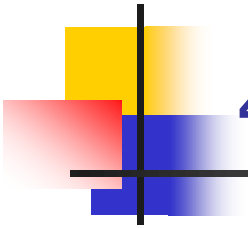
- **构造函数**用于执行类的实例的初始化。每个类都提供一个默认的构造函数。

- 使用构造函数请注意以下几个问题：
  - 一个类的构造函数通常与类名相同
  - 构造函数不声明返回类型
  - 构造函数总是public类型的
  - 构造函数可以重载

- **析构函数**是实现销毁一个类的实例的方法成员。析构函数不能有参数，不能加任何修饰符而且不能被调用。由于析构函数的目的与构造函数相反，就加前缀“~”以示区别。

例4-14 构造函数和析构函数（04-14.cs）

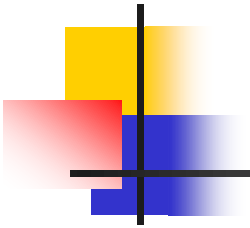
```
using System;
class Desk {
  public Desk() {   //构造函数和类名一样
          Console.WriteLine("Constructing Desk");
          weight=6;
          high=3;
          width=7;
          length=10;
          Console.WriteLine("{0},{1},{2},{3}",weight,high,width,length);
  }
  ~Desk() {     //析构函数，前面加~
          Console.WriteLine("Destructing Desk ");
   }
   protected int weight, high, width, length;
   public static void Main() {
             Desk aa=new Desk();
             Console.WriteLine("back in main() ");
   }
};
```



4.3.4 继承和多态

C# 中的继承符合下列规则

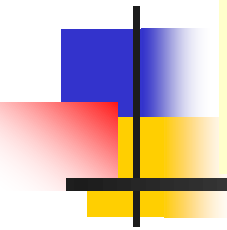
- 继承是可传递的。如果A是基类，B 从A 中派生，C 从B 中派生，那么C 不仅继承了B 中声明的成员，同样也继承了A 中的成员，Object 类作为所有类的基类。
- 派生类是对基类的扩展，它可以添加新的成员，但不能除去已经继承的成员的定义。
- 构造函数和析构函数不能被继承。
- 派生类如果定义了与继承而来的成员同名的新成员，就可以覆盖已继承的成员。



类的多态性

- 通过继承实现的不同对象调用相同的方法，表现出不同的行为，称为多态。

- C#支持两种类型的多态性：编译时的多态，运行时的多态。
  - 编译时的多态是通过重载来实现的，如方法重载和操作符重载。
  - 运行时的多态是直到系统运行时，才根据实际情况决定实现何种操作。



4.4 字符串

4.4.1 使用字符串

- 使用Length属性从字符串中读取单个字符
- 使用ToString方法把数据转换成字符串

```
int age=25;
string strAge = age.ToString();
```

- 使用ToString方法格式化数字

```
double a= 17688.658
string str=a.ToString("C")      //返回 ￥17688.658
str=a.ToString("C2")            //返回 ￥17688.65
str=a.ToString("F2")            //返回 17688.65
```

C，c— 货币，可指定小数点后位数

F，f— 定点计数法，指定小数位的位数

X— 十六进制

ToString函数 - 格式化日期和时间

- DateTime dt= DateTime.Now
  - t=dt.ToString(“D”)　‘Thursday,September 8,2005
  - t=dt.ToString(“d”)　‘9/8/2005
  - t=dt.ToString(“T”)　‘9:32:34 AM
  - t=dt.ToString(“t”)　‘9:32 AM
  - t=dt.ToString(“f ”)　‘Thursday,September 8,2005 9:26 PM
  - t=dt.ToString(“yyyy年MM月dd日”)　‘2005年09月8日

D—长日期，d—短日期
T—长时间， t—短时间
F—长日期和时间，f—短日期和时间
M,m—月和日
Y,y—月和年

Methods for the string class

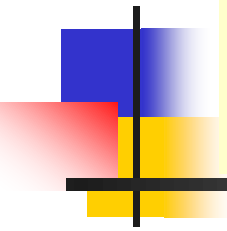
Method	Explanation
Compare()	Compares two strings
Concat()	creates a new string from one or more strings
Copy()	creates a new string by copying another
Equals()	determines if two strings have the same value
Format( )	formats a string using a format specification
Intern()	returns a reference to the specified instance of a string
Join()	concatenates a specified string between each element of a string array
Equals()	Determines if two strings have the same value
Insert()	Returns a new string with the specified string inserted
LastIndexOf()	Reports the index of the last occurrence of a specified character or string within the string
Remove()	Deletes the specified number of characters
Split()	Returns the substrings delimited by the specified characters in a string array
StartsWith()	Indicates if the string starts with the specified characters
Substring()	Retrieves a substring

4.4.2 创建动态字符串

- 使用System.Text.StringBuilder类可以创建动态字符串。同String对象的一般字符串不同，动态字符串的字符可以被直接修改。
- 创建StringBuilder对象

```
StringBuilder myStringBuilder1 = new StringBuilder();
int capacity=50;
StringBuilder myStringBuilder2 = new StringBuilder(capacity);  //指定初始容量
```

```
StringBuilder sb=new StringBuilder("Hello World! ");   //初始化字符串sb
sb.Insert(6, "Beautiful ");          //将字符串"Beautiful "添加到当前指定位置
Console.WriteLine(sb);               //输出"Hello Beautiful World! "
sb.Remove(0, sb.Length);             //移除整个字符串
```



4.5 集合编程

- ArrayList
- 哈希表（Hash Table）
- 队列（Queues）
- 堆栈（Stacks）

4.5.1 ArrayList

- ArrayList可以理解为一种特殊的数组，ArrayList集合可以动态地添加或删除所存储的元素。

```
    ArrayList myArrayList = new ArrayList(); //创建一个ArrayList对象
myArrayList.Add(“Hello ”) ； //添加元素
for (int i=0; i<myArrayList.Count; i++ )   //读取ArrayList中的元素
{
     Console.WriteLine(myArrayList[counter]);
}
```

4.5.2 哈希表（Hash Table）

- 哈希表表示一个关键码（Key）和值（Value）相关联的集合，也就是说，每一个关键码都与一个值相对应，即“Key-Value”对。

```
Hashtable myHashtable = new Hashtable(); //建立一个哈希表
myHashtable.Add(“cn”,”China”);          //添加元素
myHashtable.Add(“hk”,”Hongkong”);    //添加元素
String myCountry = (string) myHashtable[“hk”];
//查找关键码对应的值
```

4.5.3 队列（Queues）

- 队列是一个遵循“先进先出”（First In First Out）原则的集合。
- 在一端输入数据（称为加队，Enqueue），在另一端输出数据（称为减队，Dequeue）。

```
Queue myQueue = new Queue();  //创建一个Queue对象
myQueue.Enqueue(“This”); //添加元素
myQueue.Enqueue(“is”);
myQueue.Enqueue(“a”);
myQueue.Dequeue();   //删除队列头的元素
Console.WriteLine(myQueue.Peek());  //读取队列中最前面的元素
int numElements = myQueue.Count;     //获得队列的元素个数
```

4.5.4 堆栈（Stacks）

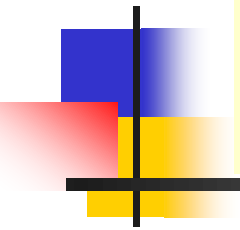
- 堆栈是一种遵循“后进先出”（Last In First Out，LIFO）原则的数据集合，简称为栈。
- 栈有一个固定的栈底和一个浮动的栈顶。所有对堆栈的操作都是针对栈顶元素进行的。

对堆栈进行操作主要有如下方法：

（1）void Push(object item)：在堆栈顶部添加一个元素，也叫入栈。

（2）object Pop()：删除栈顶的元素，并返回该元素，也叫出栈。

（3）object Peek()：返回栈顶元素，但不删除它。

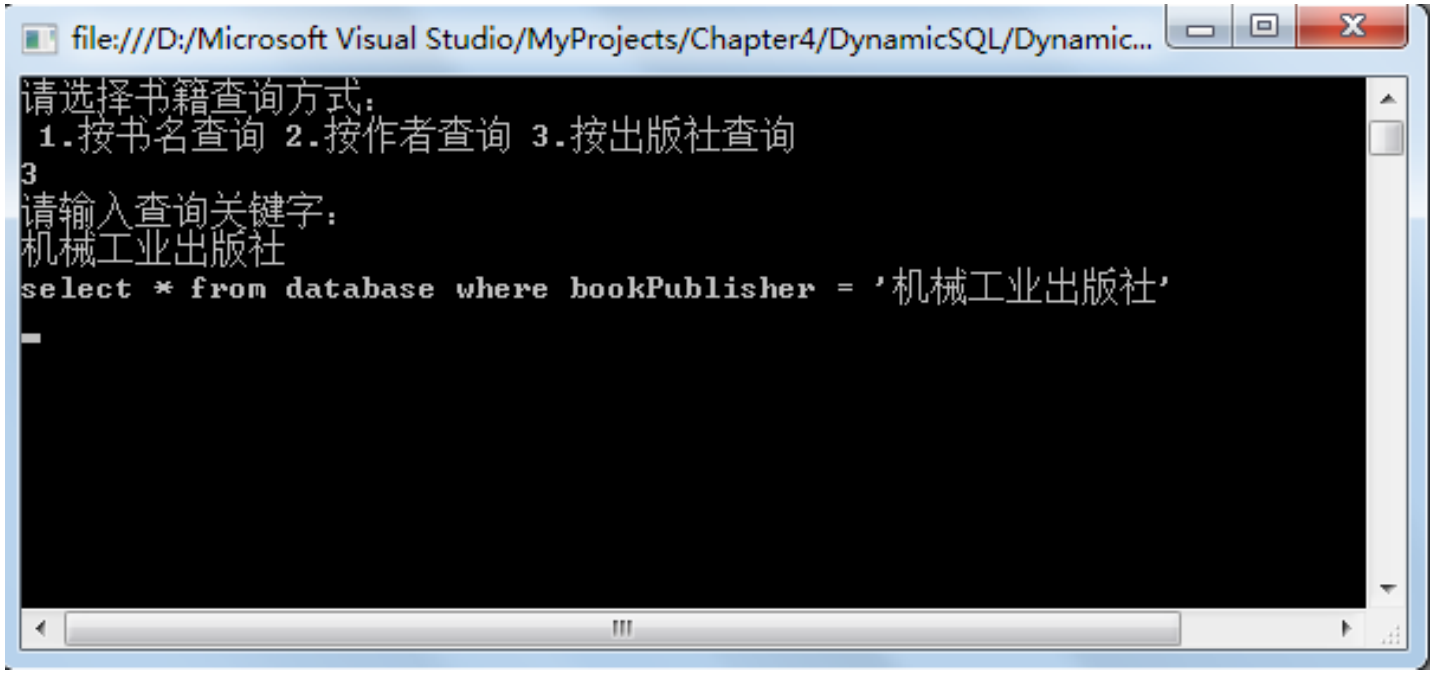


4.6 实例：用C#动态创建SQL数据查询语句

4.6.1 设计说明

当用户进行信息检索时，可能用到不同的查询方式，如精确查询或模糊查询、单项查询或组合查询等。例如，在图书检索时，可以按书名查询、按作者查询、按出版社查询等精确查询，或者进行模糊查询等。为了实现这些查询，用户需要首先在客户端界面上选择查询方式，然后输入查询关键字，提交服务器后，系统根据用户输入项动态地生成一个SQL查询语句

4.6.2 程序实现


file:///D:/Microsoft Visual Studio/MyProjects/Chapter4/DynamicSQL/Dynamic...
请选择书籍查询方式:
1.按书名查询 2.按作者查询 3.按出版社查询
3
请输入查询关键字:
机械工业出版社
select * from database where bookPublisher = '机械工业出版社'