

C#基础

郑贵锋

课程目标



通过这一节课，可以使听众对.NET基本概况和C#开发语言基本结构和语法有一个概括性的认识。



大纲

1

.NET基础

2

语言基础

3

基本类型

4

流程控制

.NET基础

.NET的定义

定义

.NET技术是微软公司推出的一个全新概念，“它代表了一个集合、一个环境和一个可以作为平台支持下一代Internet的可编程结构。”

最终目标

.NET的最终目标就是让用户在任何地方、任何时间，以及利用任何设备都能访问所需的信息、文件和程序

.NET平台

.NET开发平台包括

编程语言(C# , Visual Basic , Visual C++)

.NET开发工具(Visual Studio .NET)

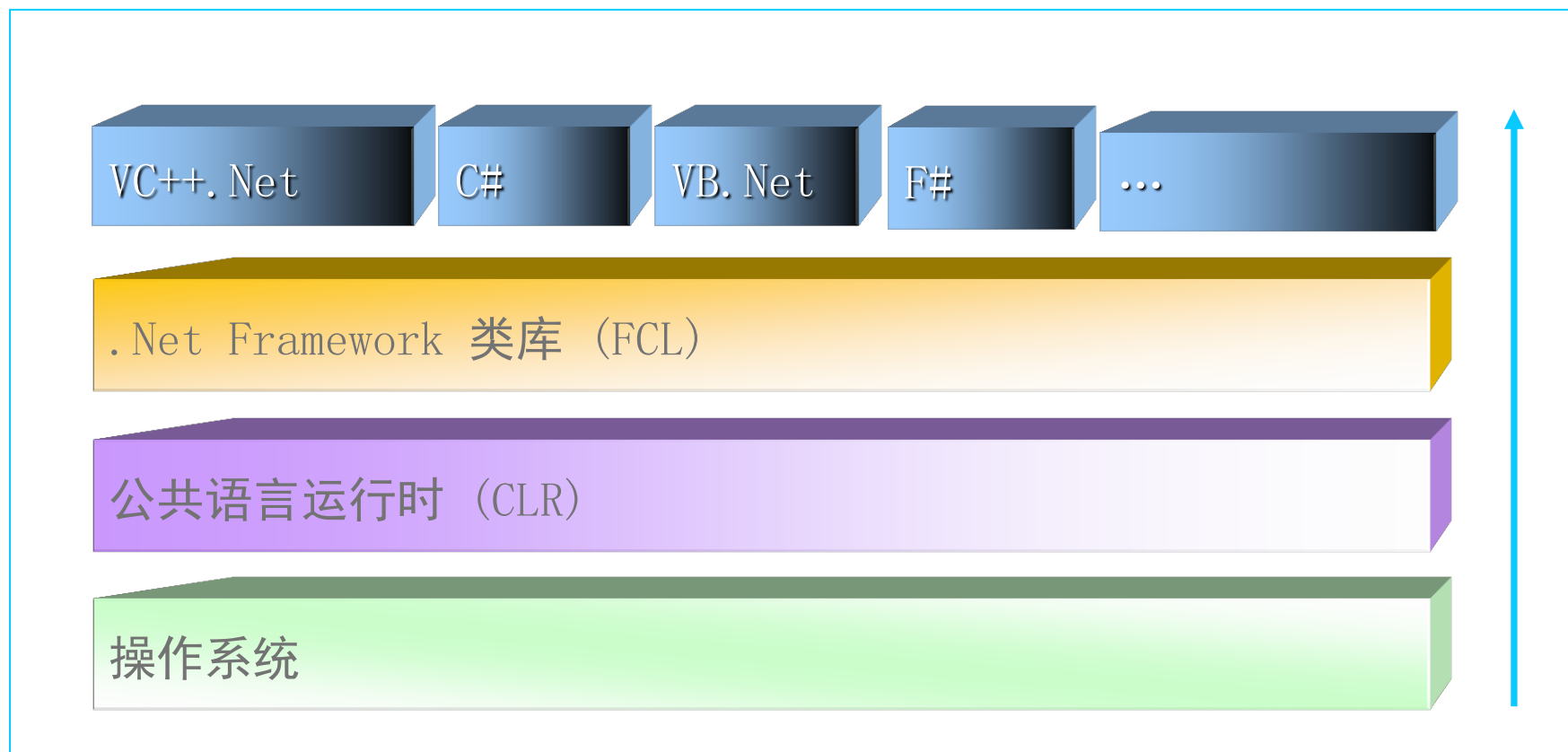
.NET框架(.NET Framework)

...

为什么选择.NET

- 可以同时使用**多种开发语言**进行开发
- 可以利用**方便的开发工具**
- 书写**更少的代码**
- 充分利用Windows系统的**应用程序服务功能**，如先进快速的事件处理和消息队列机制
- 软件服务的**发布**
- 良好的**继承性**
- 利用ADO.NET，数据访问更加简单

.NET架构



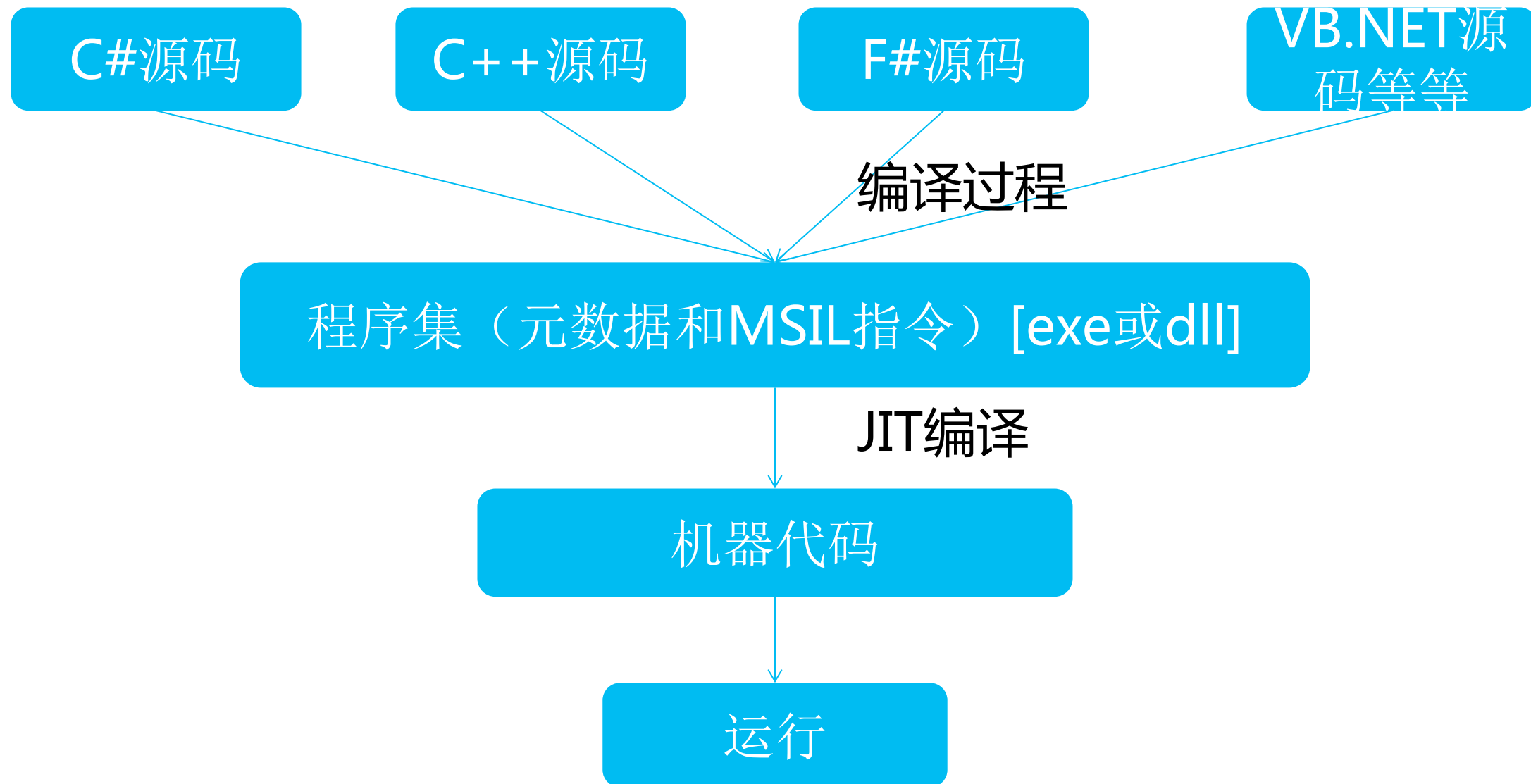
CLR能提供什么？

- CLR是Common Language Runtime的简写，中文翻译是公共语言运行。全权负责托管代码的执行(主要有内存管理和垃圾收集)，是.NET的基石。
- CLR两个基础核心：
 - 元数据：依赖元数据进行内存管理和垃圾收集等等
 - MSIL中间语言：使得.NET具有跨语言的集成的能力。CLR所支持的语言都由相对应的编译器编译为MSIL指令。再由CLR中的JIT组件编译MSIL为机器语言，最后由CLR托管执行。

.NET Framework

- 包含一个非常大的代码库，可以在客户语言(如C#)中通过面向对象的编程技术来使用这些代码。这个库分为不同的模块，可以根据需要来使用其中的各个部分。如，一个模块包含windows应用程序的构件，另一个模块包含web开发的代码块等等。
- 定义了一些基本数据类型，以便使用.NET Framework在各种语言之间进行交互作用，这称为通用类型系统(Common Type System,CTS)

.NET程序编译运行流程



语言基础

简介

- C#是一种简洁、现代、面向对象且类型安全的编程语言。
- C#特性：
 - 垃圾回收 (Garbage collection) 将自动回收不再使用的对象所占用的内存
 - 异常处理 (exception handling) 提供了结构化和可扩展的错误检测和恢复方法
 - 类型安全 (type-safe) 的语言设计则避免了读取未初始化的变量、数组索引超出边界或执行未经检查的类型强制转换等情形
- C# 是面向对象的语言，然而 C# 进一步提供了对面向组件编程的支持。

注释（一）

- 注释就是写在源代码中的描述信息，用来帮助开发人员阅读源代码的。
- 注释信息会在编译过程中自动过滤掉，不会出现在程序集中。
- C#支持三种注释格式：
 - 单行注释：以“//”开始，此行后续任何文本都作为注释内容。
 - 多行注释：以“/*”开始，“*/”结束。可跨越多行。
 - XML注释：以“///”开始，后面紧跟XML样式元素，用来描述类型
- 方法，属性，事件，索引器等等信息， Visual Studio中智能提示的描述信
- 息依赖此注释，也可在编译时期导出这些XML格式的注释到一个XML文档

注释（二）

```
class Program
```

```
{
```

```
    /// <summary>
```

```
    /// Main方法
```

```
    /// </summary>
```

```
    /// <param name="args">命令行参数</param>
```

```
    static void Main(string[] args)
```

```
    {
```

```
        /*
```

```
        声明并初始化一个变量
```

```
        *
```

```
    */
```

```
    string info = "hello world";
```

```
    //打印hello world
```

```
    System.Console.WriteLine(info);
```

```
}
```

```
}
```



XML格式注释



多行注释 /*注释内容*/



单行注释 //注释内容

Hello World 程序（一）

```
using System;  
namespace Notepad  
{  
    class HelloWorld  
    {  
        public static void Main()  
        {  
            Console.WriteLine("Hello World");  
        }  
    }  
}
```

导入 System 命名空间

声明命名空间 Notepad

声明 HelloWorld 类

程序入口点，Main 的返回类型为 void

控制台类的 WriteLine() 方法用于显示输出结果

将文件保存为 HelloWorld.cs

Hello World 程序（二）

命名空间：C# 程序是利用命名空间组织起来的。一种 “逻辑文件夹” 的概念。开发人员可以定义自己的命名空间。
常用的命名空间如下：

命名空间	说明
System	一些基本数据类型
System.Data	处理数据存取和管理，在定义 ADO.NET 技术中扮演重要角色
System.IO	管理对文件和流的同步和异步访问
System.Windows	处理基于窗体的窗口的创建
System.Reflection	包含从程序集读取元数据的类
System.Threading	包含用于多线程编程的类
System.Collections	包含定义各种对象集的接口和类

标识符（一）

- 标识符是指标识某一个东西的一个名字符号
 - 比如：变量名，类型名，参数名等等。
- 标识符以字母或者下划线（_）开头，其余部分允许出现数字和Unicode 转义序列。关键字在以@为前缀的情况下也可以作为标识符。
- C#严格区分字母大小写。
 - 如Age和age是不同的标识符。

标识符（二）

示例	是否有效	说明
123	否	不能以数字开头
n123	是	字母开头，混合数字
N123	是	大些字母N，所以和n123是不同的标识符
_n123	是	下划线加字符和数字
int	否	int是关键字
@int	是	@做前缀加关键字
n\u0061me	是	支持Unicode转义序列

关键字

- 关键字是一组特殊的“标识符”，由系统定义，供开发者使用。因而我们不能再次定义关键字为标识符（以 @ 字符开头时除外）。
- 比较常用的有using、class、static、public、get*、set*、var*等等。
- 其中加 “*” 的比较特殊些，称作上下文关键字，这些关键字只有在特殊的位置才会有意义。如get和set只有在属性中才有意义、var只能用在局部变量环境下。

声明&初始化

声明一个变量的语法：

数据类型 变量名；//变量名须为有效标识符

如：

```
string name;
```

声明并初始化一个变量：

数据类型 变量名=初始化值；

如：

```
string name= "张三" ;
```

运算符（一）

常用的运算符：

运算符类型	常用运算符	示例
算数运算符	+ - * / %	int i=1,j=2; i+j;//结果3
关系运算符	> < >= <= == !=	i>j ; //结果false
赋值运算符	= += -= *= /= %=	i+=j;//结果3[i=i+j ; 的简写形式]
自运算符	前置：++ -- 后置：++ --	int n=1;int m; 前置：m=++n;//结果m=2,n=2 后置：m=n++;//结果m=1,n=2
成员访问运算符	. []	.：调用对象成员 []:访问数组元素或索引器
逻辑运算符	! &&	bool a=true; !a;//结果false

运算符（二）

- 大多数运算符都可以重载 (overload)。运算符重载允许指定用户定义的运算符实现来执行运算，这些运算的操作数中至少有一个，甚至所有操作数都属于用户定义的类型或结构类型。
- 运算符是有优先级的，优先级高的先运算。

表达式

- 表达式 由操作元 (operand) 和运算符 (operator) 构成。
 - 运算元可以是常数、对象、变量、常量、字段等等。
 - 运算符可以是上节提到的一些运算符。
- 当表达式包含多个运算符时，运算符的优先级 (precedence) 控制各运算符的计算顺序。
 - 例如，表达式 $x + y * z$ 按 $x + (y * z)$ 计算。

Hello World

演示

基本类型

变量（一）

- 变量（ variable ），言外之意即是可变的，用来存储程序所需的数据。
- 声明一个变量的语法结构如下：
//变量名必须是有效的标识符
数据类型 变量名；
- 也可以在声明的同时初始化该变量：
//变量名必须是有效的标识符
//值必须是与变量声明的数据类型兼容。
数据类型 变量名=值；

变量（二）

```
class Program
{
    static void Main(string[] args)
    {
        //声明变量
        int age;
        //为变量age赋值
        age = 18;
        //声明name并初始化为李四
        string name = "李四";
    }
}
```

常量（一）

- 常量：一经初始化就不会再次被改变的“变量”，在程序的整个运行过程中不允许改变它的值。
- 编译时常量：
 - `const` 数据类型 常量名=值；
 - 编译时常量做为类成员时总是作为static成员出现。不允许自己加static关键字。
 - 编译时常量的值必须是在编译时期能确定下来的，只支持一些基本数据类型。
- 运行时常量：
 - `readonly` 数据类型 常量名=值；
 - 运行时常量可以弥补编译时常量不能定义复杂数据类型的缺点。

(下页)

常量 (二)

```
class Program
{
    //正确, string为基本数据类型
    const string NAME_CONST = "const string";
    //错误, Program为自定义复杂类型
    const Program PROGRAM_CONST = new Program();
    //正确,
    readonly string NAME_READONLY = "readonly string";
    //正确, 可定义任意数据类型
    readonly Program PROGRAM_READONLY = new Program();
}
```

结构（一）

- 结构 (struct) 是能够包含数据成员和函数成员的数据结构。
- 结构类型的变量直接存储该结构的数据。
- 所有结构类型都隐式地从类型System.ValueType继承。
- System.ValueType继承自System.Object。
- 结构是值类型，不需要在堆分配。
- 结构类型不允许继承。

结构（二）

//用struct修饰，表示一个结构类型

```
struct Point
```

```
{
```

```
    public int x;
```

```
    public int y;
```

```
    public Point(int x, int y)
```

```
    {
```

```
        this.x = x;
```

```
        this.y = y;
```

```
    }
```

```
}
```

枚举（一）

- 枚举 (enum) 是具有一组命名常量的独特的值（结构）类型
- 每个枚举类型都有一个相应的整型类型，称为该枚举类型的基础类型 (underlying type)。没有显式声明基础类型的枚举类型所对应的基础类型是 int。枚举类型的存储格式和取值范围由其基础类型确定。
- 所有枚举类型默认继承自 System.Enum 类型，System.Enum 继承自 System.ValueType。故枚举为结构类型。

枚举（二）

//:long表示基础类型是long

//如果不写则默认为int

```
enum Alignment : long
```

```
{
```

 //=也可不写，默认则是以0开始，依次加1

```
    Left = -1,
```

```
    Center = 0,
```

```
    Right = 1
```

```
}
```

枚举（三）

```
public class Test
{
    static void Main()
    {
        //初始化一个枚举变量
        Alignment alignment = Alignment.Left;
        //输出: Left
        Console.WriteLine(alignment);
        //输出: -1
        Console.WriteLine((long)alignment);
        //获得基础类型
        Type underlyingType = Enum.GetUnderlyingType(typeof(Alignment));
        //输出: System.Int64
        //解释: [long关键字映射的类型为System.Int64]
        Console.WriteLine(underlyingType.FullName);
    }
}
```

数组（一）

- 数组 (array) 是一种包含若干变量的数据结构，这些变量都可以通过计算索引进行访问。数组中包含的变量（元素 (element)）具有相同的类型，该类型称为数组的元素类型 (element type)。
- 数组类型为引用类型，因此数组变量的声明只是为数组实例的引用留出空间。在运行时使用 new 运算符动态创建（须指定长度），长度在该实例的生存期内是固定不变的。数组元素的索引范围从 0 到 Length - 1。new 运算符自动将数组的元素初始化为它们的默认值，例如将所有数值类型初始化为零，将所有引用类型初始化为 null。

数组（二）

- C#支持一维、多维、交错数组。
- 数组下标一般是从0开始。也提供有其他方式支持非从0下标开始的数组。
- `System.Array` 类型是所有数组类型的抽象基类型。
- 访问数组元素使用下标方式：`array[索引]`

数组（三）

```
public class Test
{
    static void Main()
    {
        //元素个数为3的int类型数组
        int[] ages = new int[3];
        //3 X 3的多维数组
        int[,] i = new int[3, 3];
        //交错数组
        int[][] j = new int[2][];
        j[0] = new int[2] { 3, 4 };
        j[1] = new int[3];
        //输出：4
        Console.WriteLine(j[0][1]);
    }
}
```

字符串处理（一）

- 写程序中很大一部分的时间都是在和字符串打交道。微软给出的.NET类库中也给出了一些字符串处理的类型。
- C#中的常用字符串处理类：
 - System.String
 - System.Text.StringBuilder
- 利用String类可以进行字符串的创建，截取，替换，合并等等操作。也可以用“+”方便的进行字符串的合并。
- 大写String与小写string是完全相同的，大写是指.NET类库中的String类型，小写是C#关键字，也是对应到String这个类型上去的。比如在C#中int和Int32也是这样对应的。

字符串处理（二）

- String的特别之处：
 - 不变性；
 - 读共享，写复制；
 - 字符串驻留技术；
- String是引用类型，但其值确是不可变的，即是指已经赋值就不能再改变。针对字符串的一些操作（如合并、截取）都会产生出新的String对象。
- 由于写复制的特性，在一些需要大量合并字符串的场合就会产生出很多临时性的String对象，然后又被丢弃，浪费掉不少内存。所以类库中有另一个System.Text.StringBuilder类型来高效的拼接字符串。

字符串处理（三）

```
public class Test
{
    static void Main()
    {
        String name = "[ 小明";
        //合并字符串
        name = name + " 20岁";
        name = name + " 男生    ]";
        Console.WriteLine(name); // [ 小明 20岁 男生    ]
        //替换空格为"-"
        name = name.Replace(' ', '-');
        Console.WriteLine(name); // [-小明-20岁-男生-----]
    }
}
```


字符串处理（四）

```
public class Test
{
    static void Main()
    {
        System.Text.StringBuilder stringBuilder =
            new System.Text.StringBuilder();
        for (int i = 0; i < 1000; i++)
        {
            //追加字符串
            stringBuilder.Append(i.ToString() + "|");
        }
        //输出: 1|2|3|4|5.....999|
        Console.WriteLine(stringBuilder.ToString());
    }
}
```

委托（一）

- 委托类型 (delegate type) 表示对具有特定参数列表和返回类型的方法的引用。通过委托，我们能够将方法作为实体赋值给变量和作为参数传递。委托类似于在其他某些语言中的函数指针的概念，但是与函数指针不同，委托是面向对象的，并且是类型安全的。
- 委托声明定义一个从 `System.Delegate` 类派生的类。委托实例封装了一个调用列表，该列表列出了一个或多个方法，每个方法称为一个可调用实体。对于实例方法，可调用实体由该方法和一个相关联的实例组成。对于静态方法，可调用实体仅由一个方法组成。用一个适当的参数集来调用一个委托实例，就是用此给定的参数集来调用该委托实例的每个可调用实体。

委托（二）--特性

- 将方法作为参数传递
 - 通常传递的是变量(字段),委托则是传递方法
- 回调方法
 - 底层代码定义方法签名的类型（委托），定义委托成员
 - 上层代码创建方法，创建委托实例，让需要调用的方法传给底层
 - 底层通过调用委托，调用上层方法
- 多路广播
 - 可以同时维持多个方法的引用（+=、-=）
- 委托是类型安全的
 - DelegateA da ; DelegateB db ; 即使函数签名相同，也不能执行da=db ;
- 委托类型都是密封的(sealed)
 - 不能继承

委托（三）--重要成员

- Target
 - object类型的属性，指向回调函数所属的对象实例（对于实例方法来言）
 - 引用的方法是静态方法时,Target为null
- Method
 - System.Reflection.MethodInfo类型的属性，指向回调函数
- Invoke
 - 函数，同步执行委托
- BeginInvoke
 - 开始异步执行委托
- EndInvoke
 - 完成异步执行

委托（四）-- 运算操作

```
myDelegate += new MyDelegate(AddNumber.add2);
```

- 将一个委托A与另一个委托B连接，将连接后的新委托，在赋给原委托A
- 实质是使用的System.Delegate的静态方法Combine

```
myDelegate=(MyDelegate)Delegate.Combine(myDelegate, new  
MyDelegate(AddNumber.add2));
```

```
myDelegate -= new MyDelegate(AddNumber.add2);
```

- 一个委托A的调用列表中移除另一个委托B的最后一个调用列表，将移除后的新委托，再赋给原委托A
- 实质是使用的System.Delegate的静态方法Remove

```
myDelegate = (MyDelegate)Delegate.Remove(myDelegate, new  
MyDelegate(AddNumber.add2));
```

委托（五）

```
class Test
{
    //声明一个委托
    delegate double Function(double x);
    static void Main()
    {
        //创建一个委托对象
        Function f = new Function(Square);
        //利用f携带的“Square”方法，所以可以用f进行间接调用Square
        //也可以写f(5)，这是对f.Invoke(5)的语法简化
        double result = f.Invoke(5);
        System.Console.WriteLine(result); //25
    }
    static double Square(double x)
    {
        return x * x;
    }
}
```

事件（一）

- .NET的事件模型建立在委托的机制之上。定义事件成员的类型允许类型(或者类型的实例)在某些特定事件发生时通知其他对象。
- 事件为类型提供了以下三种能力：
 - 允许对象登记该事件；
 - 允许对象注销该事件；
 - 允许定义事件的对象维持一个登记对象的集合，并在某些特定的事件反生时通知这些对象。

事件（二）

//声明一个委托

```
public delegate void ComingEventHandler  
                        (object sender, EventArgs e);
```

// 老鼠

```
public class Mouse  
{
```

//此方法原型与ComingEventHandler委托匹配

```
public void Speak(Object sender, EventArgs e)  
{
```

```
    Console.WriteLine("猫来了，我要逃跑了！");
```

```
}
```

```
}
```


事件（三）

```
//猫
public class Cat
{
    //声明一个事件
    public event ComingEvevtHandler Coming;
    //触发事件
    public void OnComing(EventArgs e)
    {
        if (Coming != null)
        {
            Coming(this, e);
        }
    }
}
```

事件（四）

```
public class Test
{
    static void Main()
    {
        //初始化一只猫
        Cat cat = new Cat();
        //初始化一只老鼠
        Mouse mouse = new Mouse();

        //注册事件
        cat.Coming += new ComingEventHandler(mouse.Speak);
        //猫来了
        //调用注册的方法，输出：猫来了，我要逃跑了！
        cat.OnComing(EventArgs.Empty);
    }
}
```

流程控制

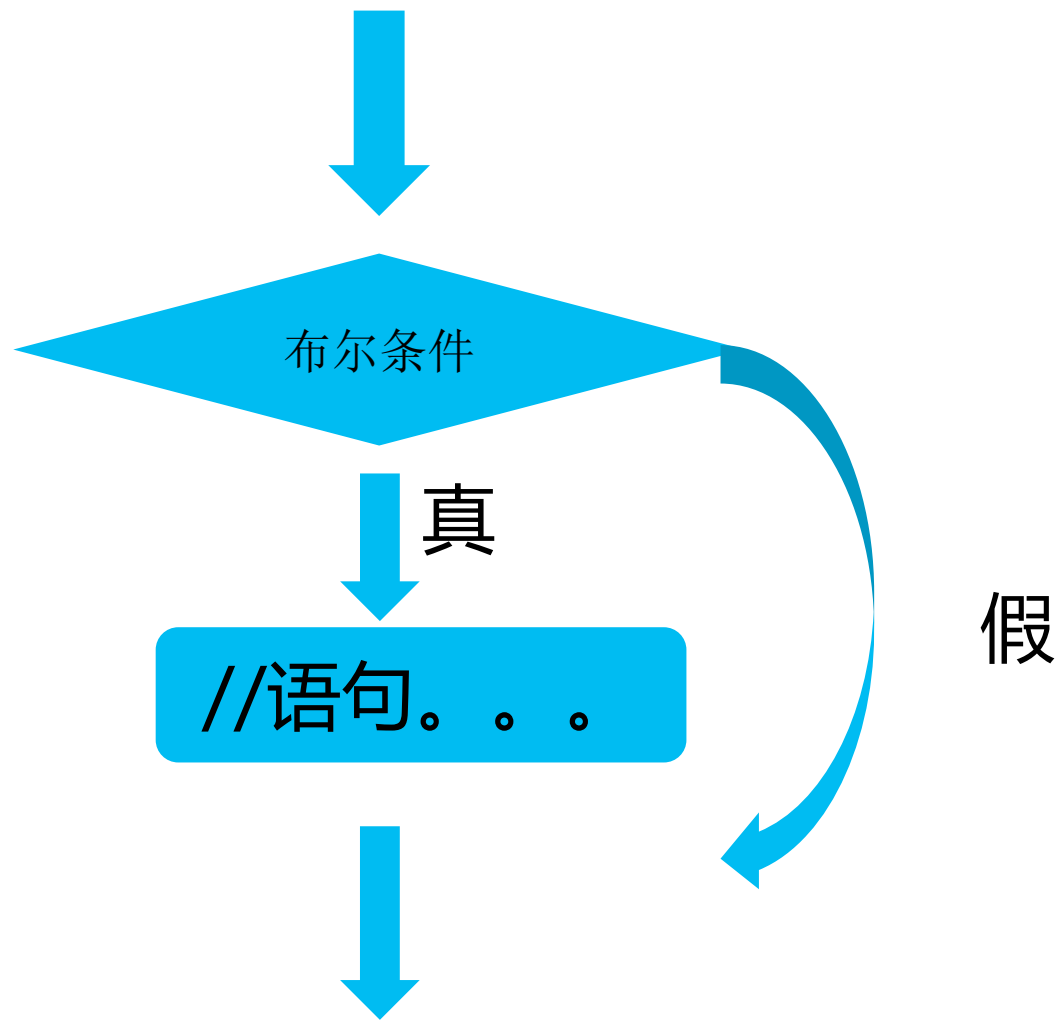
三种基本流程控制

分类	常用
顺序	普通代码
分支	If else、 switch case
循环	For、 do while、 while

If else (一)

If语句语法：

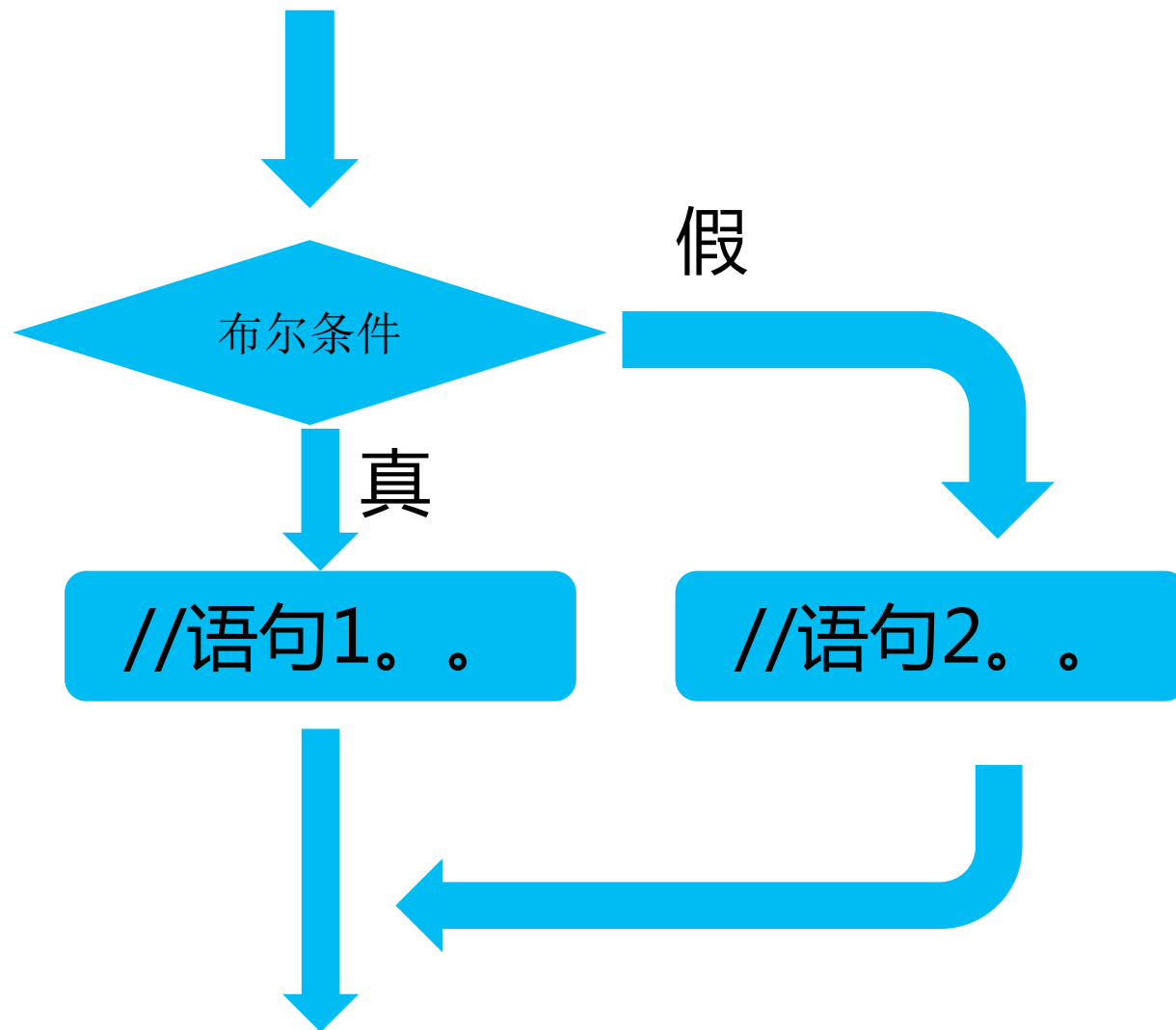
```
if(布尔条件)
{
    //语句。。。
}
```



If else (二)

If else语句语法：

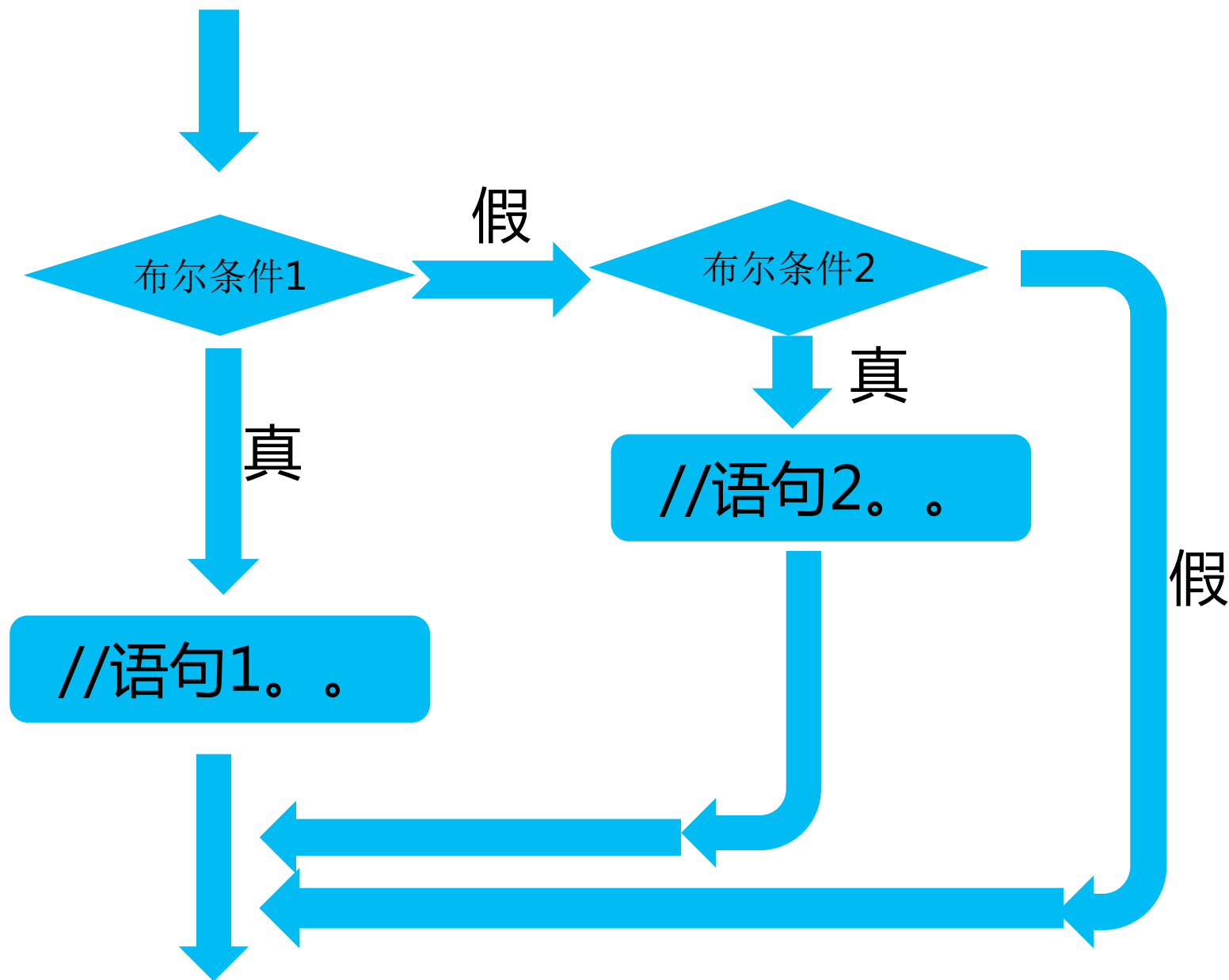
```
if(布尔条件)
{
    //语句1。。
}
else{
    //语句2。。
}
```



If else (三)

If else语句语法：

```
if(布尔条件1)
{
    //语句1。。
}
else if(布尔条件2){
    //语句2。。
}
```



If else (三)

```
static void Main(string[] args)
{
    bool isTrue = true;
    if (isTrue)
    {
        //执行
        System.Console.WriteLine("true");
    }
    if (!isTrue)
    {
        //不执行
        System.Console.WriteLine("true");
    }
    else if(true) {
        //执行
        System.Console.WriteLine("false");
    }
}
```


Switch case (—)

- Switch case是多分支选择语句，用来实现多分支选择结构。
- 适合于从一组互斥的分支中选择一个来执行。
- 类似于if语句，但switch语句可以一次将变量与多个值进行比较，而不是仅比较一个。
- switch参数后面跟一组case 子句,如果switch参数中的值与某一个case 后面的判断式相等,就执行case 子句中的代码。执行完后用break语句标记每个case 代码的结尾,跳出switch语句;

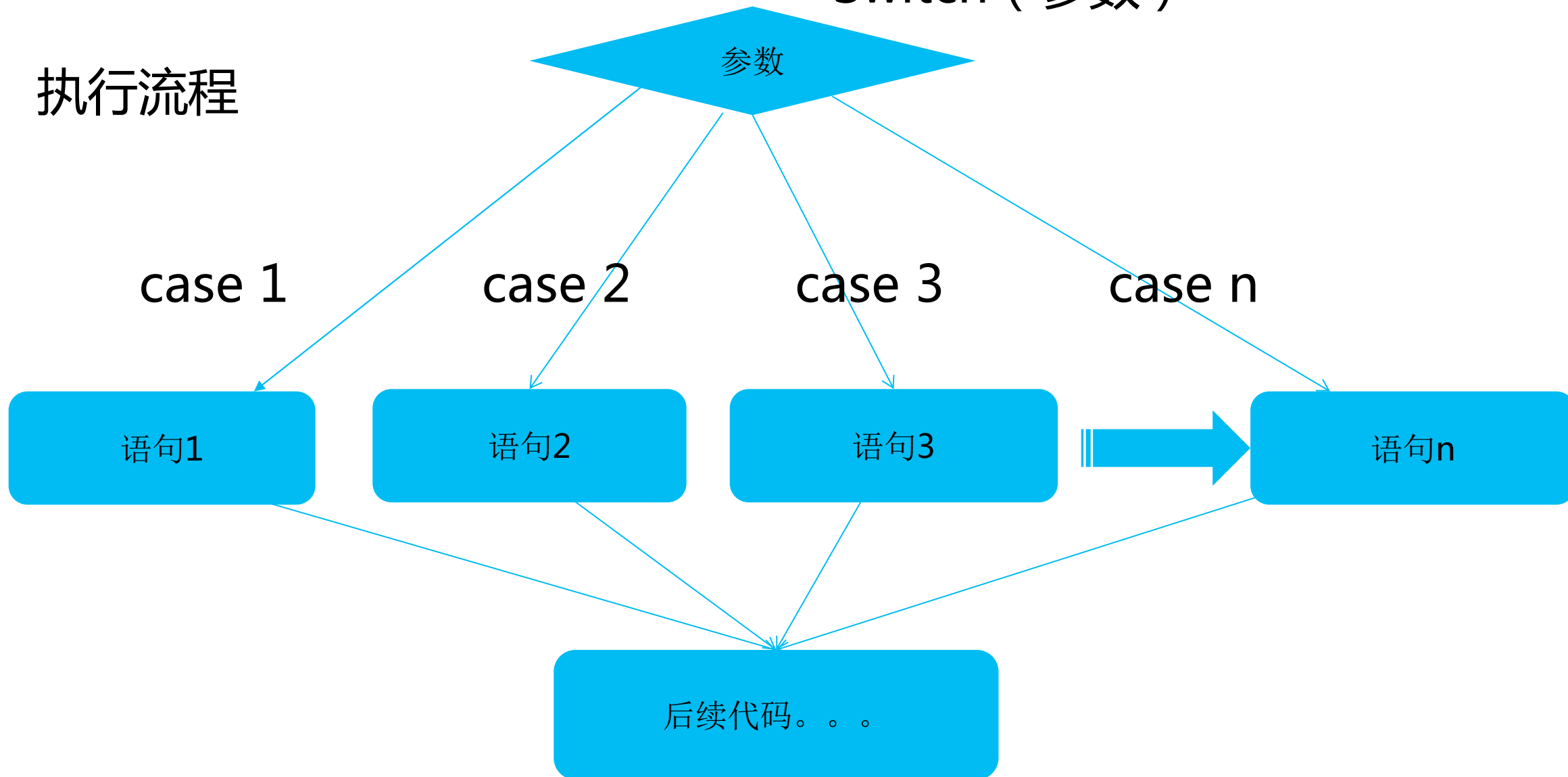
Switch case (二)

- 也可在switch语句中包含一个default语句，当所有case 中的常量表达式的值都没有与switch中表达式的值相等,就执行default子句中的代码。
- default子句可有可无，一个switch语句中有且仅有一个default分支。
- case后的值必须是常量表达式,不允许使用变量。
- case 子句的排放顺序无关紧要;
- default子句也可放到最前;任何两个case 的值不能相同.

Switch case (三)

Switch (参数)


执行流程



Switch case (四)

```
class Program
{
    static void Main(string[] args)
    {
        DateTime now = DateTime.Now;
        DayOfWeek week = now.DayOfWeek;
        switch (week)
        {
            case DayOfWeek.Saturday:
                Console.WriteLine("休息");
                break;
            case DayOfWeek.Sunday:
                Console.WriteLine("休息");
                break;
            default:
                Console.WriteLine("上班");
                break;
        }
    }
}
```

合并两个
case

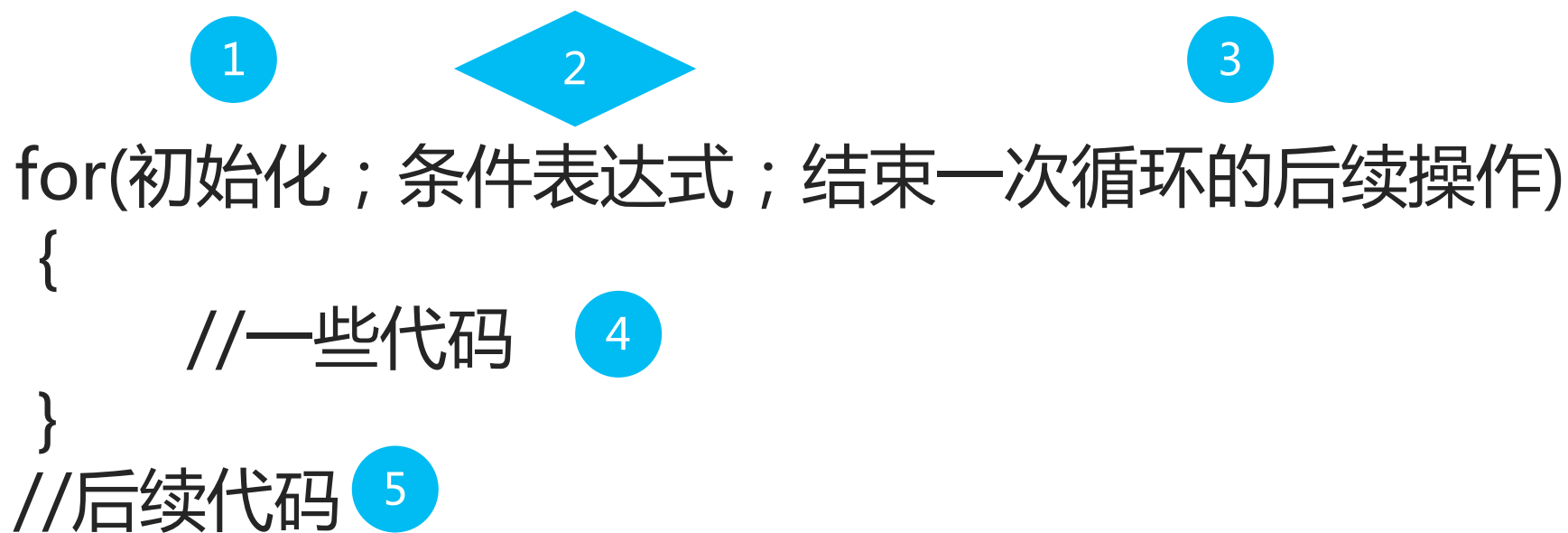


```
class Program
{
    static void Main(string[] args)
    {
        DateTime now = DateTime.Now;
        DayOfWeek week = now.DayOfWeek;
        switch (week)
        {
            case DayOfWeek.Saturday:
            case DayOfWeek.Sunday:
                Console.WriteLine("休息");
                break;
            default:
                Console.WriteLine("上班");
                break;
        }
    }
}
```

for (—)

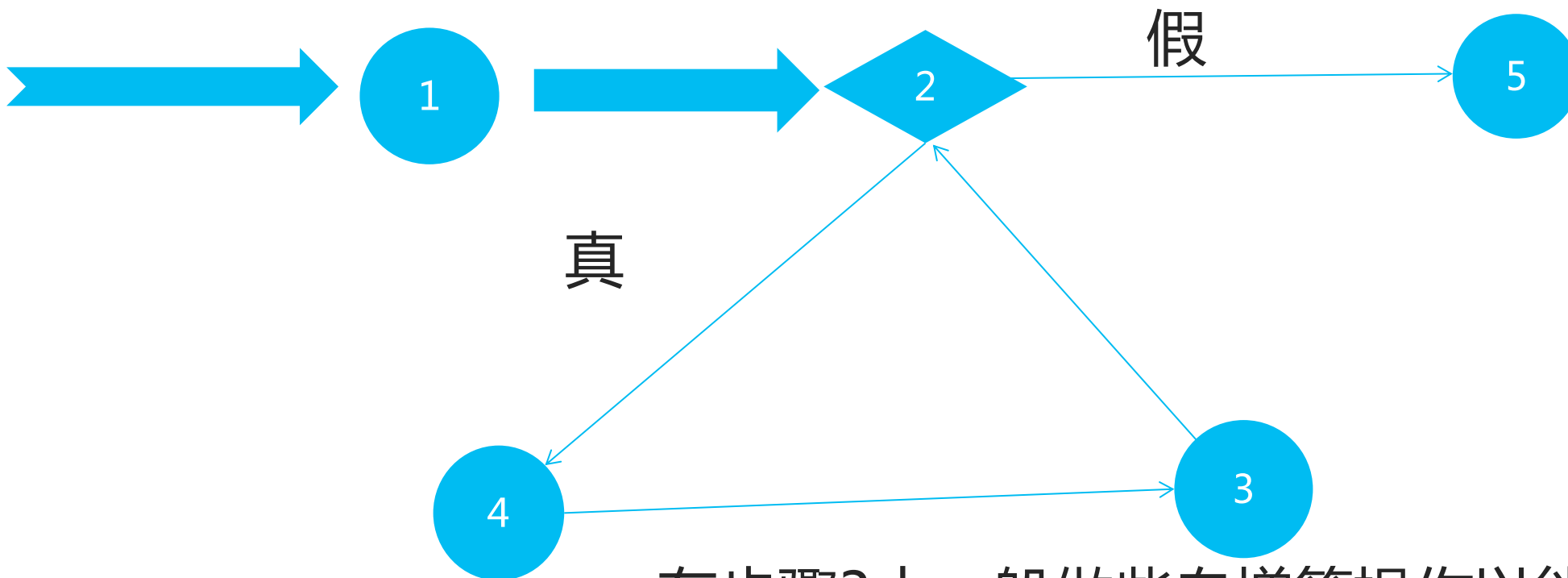
for 语句用来依据特定条件来多次重复执行某些代码。

语法如下：



for (二)

执行流程：



在步骤3中一般做些自增等操作以能影响步骤2的操作，使循环得以继续或者终止

for (三)

```
class Test
{
    static void Main()
    {
        for (int i = 0; i < 100; i++)
        {
            System.Console.WriteLine(i);
        }
    }
}
```

while & do while (—)

- while循环的一般形式为:

```
while(条件) //这个条件为布尔表达式
{
    循环体语句 ;
}
```

- 执行流程:
 - 先判断条件(即布尔表达式的值),如为真便重复执行循环体语句 ; 直到条件为假时才结束循环 , 并继续执行循环程序外的后续语句。

while & do while (二)

```
class Test
{
    static void Main()
    {
        int i = 0;
        while (i < 10)
        {
            System.Console.WriteLine(i);
            i++;
        }
    }
}
```

while & do while (三)

- do-while循环的一般格式为：

```
do
{
    //循环体语句；
}while(测试条件);
```

- 执行流程：
 - 先执行循环体语句，然后测试while中的条件，如果测试条件为true,就再次执行循环体语句，直到测试结果为false时，就退出循环。

while & do while (四)

```
class Test
{
    static void Main()
    {
        int i = 0;
        do
        {
            System.Console.WriteLine(i);
            i++;
            //这个while条件后面是有分号的,是必须的
        } while (i < 10);
    }
}
```

while & do while (五)

- do-while与while的不同之处在于：
 - do-while它是先执行循环中的语句，然后再判断条件是否为真，如果为真则继续循环，如果为假则终止循环。因此对于do-while语句来说至少要执行一次循环语句。
 - 而while语句是先判断条件是否为真，为真则执行循环语句，若不为真，则终止循环。因此对于while语句来说可能一次也不会执行循环体语句。

break&continue (一)

- break 语句退出直接封闭它的switch、while、do while或for语句。
- 当多个 switch、while、do while或for语句彼此嵌套时，break语句只应用于最里层的语句。直接跳出当前循环。
- continue 语句开始直接封闭它的 while、do while或for语句的一次新迭代。进入下一次循环。
- 当多个 while、do while或for语句互相嵌套时，continue语句只应用于最里层的语句。

break&continue (二)

```
public class Test
{
    static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            if (i==5)
            {
                continue;
            }
            if (i==8)
            {
                break;
            }
            Console.Write(i);
        }
        //输出结果: 0123467
    }
}
```

return (一)

- return 语句将控制返回到出现 return 语句的函数成员的调用方。结束当前方法。跳转回到调用位置。
- 不带表达式的return语句只能用在不计算值的函数成员中，即只能用在返回类型为void的方法、属性或索引器的set 访问器、事件的add和remove 访问器、实例构造函数、静态构造函数或析构函数中。
- 带表达式的 return 语句只能用在计算值的函数成员中，即返回类型为非 void 的方法、属性或索引器的get 访问器或用户定义的运算符。必须存在一个隐式转换，使得该表达式兼容于包含它的函数的返回类型。

return (二)

```
public class Test
{
    void NoReturnValue()
    {
        //return后面不能写表达式, 因为此方法返回类型为void
        //返回void的方法可以不写return语句
        return;
    }
    int ReturnInt()
    {
        //123为是int类型, 正确的return
        return 123;
        //123.00为是double类型, 不能隐式转换为int, 错误的return
        //return 123.00;
    }
}
```


总结

- **.NET基础**

.NET Framework和CLR

- **语言基础**

注释，标识符，关键字，声明&初始化，运算符和表达式

- **基本类型**

变量，常量，结构，枚举，数组，字符串，委托，事件

- **流程控制**

if else, switch case, for, while&do while, break&continue, return

资源

- [Visual C#](#)

- <http://msdn.microsoft.com/zh-cn/library/kx37x362.aspx>

- [《C#高级编程》](#)

- <http://product.china-pub.com/197224>

- [《CLR 框架设计》](#)

- <http://product.china-pub.com/28146>

- [《深入理解C#》](#)

- <http://product.china-pub.com/198866>

Q&A

© 2011 Microsoft Corporation。保留所有权利。Microsoft、Windows、Windows Vista 及其他产品名称是或者可能是在美国和/或其他国家/地区的注册商标和/或商标。

此处包含的信息仅供参考，并代表 Microsoft Corporation 截至本演示文稿发布之日的最新观点。由于 Microsoft 必须响应不断变化的市场条件，所以不应将本文视为 Microsoft 一方的承诺，Microsoft Corporation 也无法保证所提供信息在本文发布之后的准确性。MICROSOFT 对本演示文稿中包含的信息不做任何明示、暗示或法定的担保。

The Microsoft logo is centered on a solid blue background. It features the word "Microsoft" in a white, bold, sans-serif font. The background is decorated with several faint, light blue squares of various sizes, some of which are slightly offset from the main grid, creating a subtle pattern.

© 2011 Microsoft Corporation。保留所有权利。Microsoft、Windows、Windows Vista 及其他产品名称是或者可能是在美国和/或其他国家/地区的注册商标和/或商标。
此处包含的信息仅供参考，并代表 Microsoft Corporation 截至本演示文稿发布之日的最新观点。由于 Microsoft 必须响应不断变化的市场条件，所以不应将本文视为 Microsoft 一方的承诺，Microsoft Corporation 也无法保证所提供信息在本文发布之后的准确性。MICROSOFT 对本演示文稿中包含的信息不做任何明示、暗示或法定的担保。