第 6 章 程序调试与异常处理

主要内容

- 程序调试方法的使用
- 异常处理的方法
- 如何自行抛出异常

调试的必要性

- 俗话说得好：“人无完人”。程序也是如此，再有经验的程序员，写代码的时候再小心，也会出错。因为人不是机器，肯定会有考虑不周全的时候或者粗心，这样就会造成错误。 Visual C#.NET 2005 环境中提供了基本的语法检查以及错误识别，小错误稍微有点程序经验的人员通过运行时的错误提示就可以轻松化解。
- 但是，也有的时候我们的程序本身看不出来有什么错误，运行时也会能得到结果，可是结果不是我们预期的，这时我们就需要通过跟踪代码，通过窗口输出协助以及设置断点等方式，进行查找。这种错误叫逻辑错误，是最难查出的错误。

6.1 程序调试的介绍以及调试方法的使用

- 程序在编写的过程中错误主要类型：
  - 语法错误

    语法错误常见的是发生在初学者身上，这种错误在编译阶段程序可以自动跳到错误之处，很容易修改。
  - 运行时错误

    运行时错误是用户在执行应用程序时，因为输入类型不符或者是被除数为 0 或者数组越界造成的，这个错误会造成程序的中断，可以使用 try-catch-finally 语句解决。
  - 逻辑错误

    逻辑错误是最困难的错误，尤其在大型程序中最为明显。程序在执行过程中不提示错误信息，也会有运行结果，但是结果不符合逻辑，或者是跟我们预期的不一样。

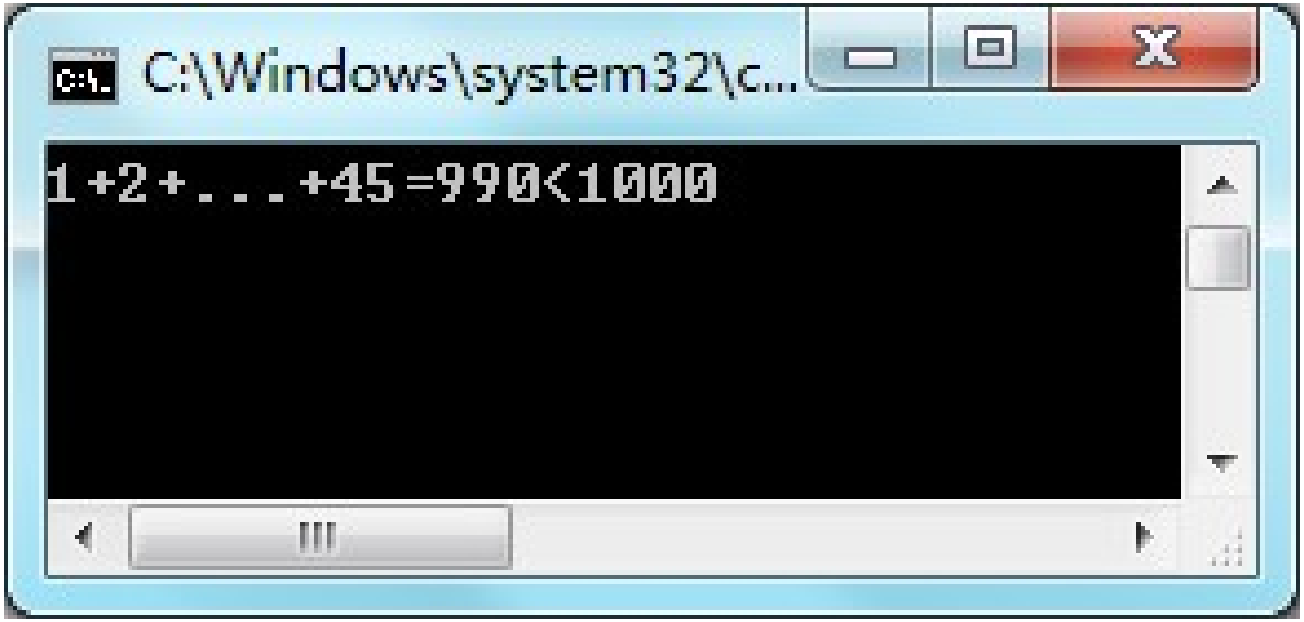
调试演示

- **例 6.1** 求 1+2+3+…+n <1000 的最大 n 值 。

  程序代码：

```
static void Main(string[] args)
        {
            int sum = 0;
            int n = 1;
            while (true)
            {
                sum = sum + n;
                if (sum >= 1000)
                {
                    sum = sum - n;
                    Console.WriteLine("1+2+...+{0}={1}<1000", n, sum);
                    break;
                }
                n = n + 1;
            }
            Console.ReadLine();
        }
```

结果显示

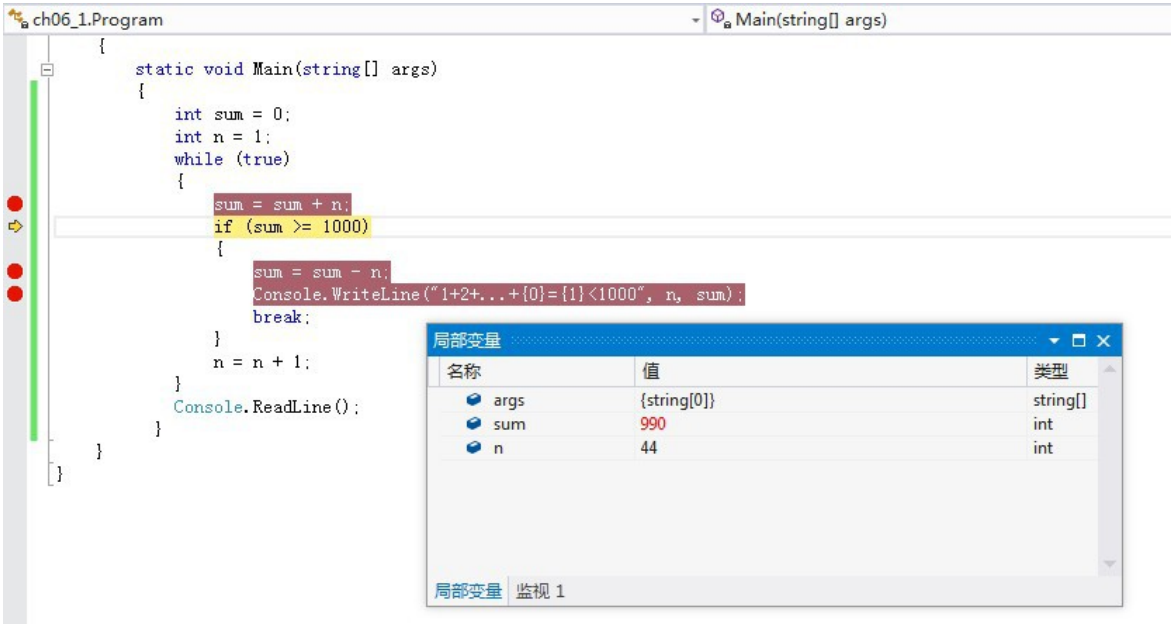


错误发现

- 上面的程序运行正常，也产生了结果，但是我们发现 1+2+…+45 并不等于 990，而是等于 1035，这就说明我们的程序存在逻辑错误，下面我们一起来探讨一下如何通过调试来查询错误。

断点设置

- 在代码中设置 3 个断点
-



错误分析过程

- 通过自动窗口我们发现，在上面的例子中，n=44 时 result 的值就已经是 990 了，所以我们可以在代码 sum = sum - n ; 后面添加代码 n = n - 1; 就可以了。

6.2 异常处理

- C# 中的异常处理由 4 个关键字来管理： Try 、 Catch 、 Throw 和 Finally 。它们构成了一个相关子系统，在这个子系统中，一个关键字的使用隐含地使用另一个关键字。本节中将详细地分析每个关键字，但是开始时，大概了解每个关键字在异常处理中的作用是有作用的，也是有帮助的。 try 语句块主要包含我们要监视的是否产生异常的程序语句，如果 try 语句块内的语句发生异常，那么就要抛出（ Throw ）异常，然后使用 Catch 语句捕捉此异常，并以合理的方式处理它。 C# 运行时系统会自动抛出系统产生的异常。要手动抛出异常，则使用关键字 Throw 。从 try 模块退出时绝对需要执行的代码放置在 Finally 语句中。

异常处理的注意事项

- C# 异常处理中应该注意以下事项：
  - 抛出异常时，需要提供一些有价值的文本信息；
  - 只有在真正需要异常处理时才可以使用抛出异常。也就是说当一个正常的返回值
  - 不满足条件时才能使用；
  - 当传递给方法或属性的参数有错误时，使用一个 ArgumentException 异常；
  - 当操作无意识地与对象当前状态不相符时，要抛出一个 InvalidOperationException
  - 异常；
  - 要引发合适的异常；
  - 要使用链接的异常，它们允许用户跟踪异常树结构；
  - 不要在流程的正常控制中使用异常处理；
  - 不要用异常来控制程序的运行走向；
  - 不要在函数中引发 NullReferenceException 或 IndexRangeExcepption 异常。

异常处理中使用的语句

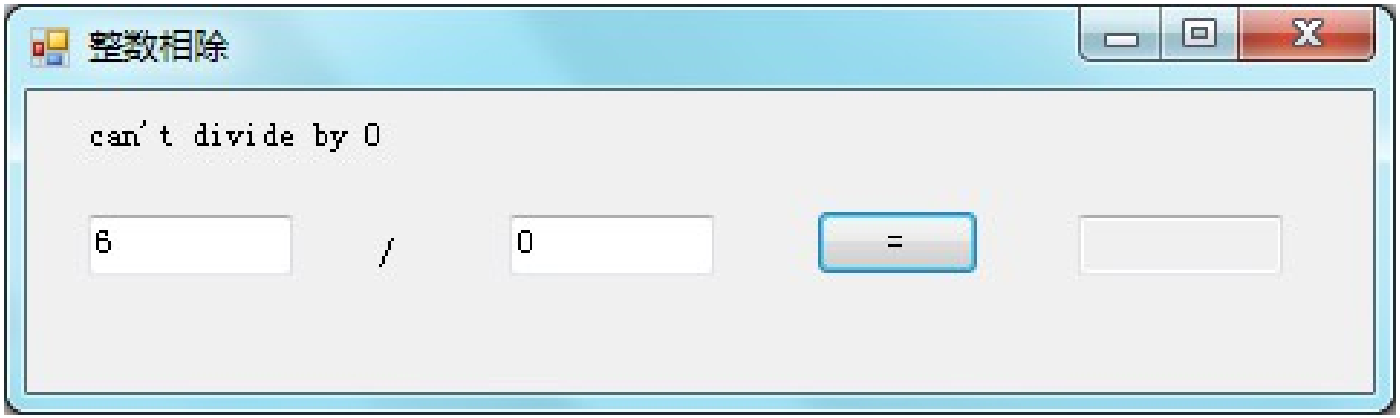
- 使用 Try 和 Catch 捕获异常
  - 使用 Try 语句和 Catch 语句，可以使程序在发生异常时不仅不会提示给用户比较讨厌的异常信息．还会继续执行程序。
  - Try 语句包括可能产生异常的部分，而 Catch 语句可以处理一个存在的异常。
- 使用 Try 和 Finally 清除异常
  - 使用 Try 和 Finally 语句，可以清除异常。 Finally 代码块可以用于清除 Try 代码块中分配的任何资源，以及运行任何即使在发生异常时也必须执行的代码。 Catch 语句用于处理语句块中出现的异常，而 Finally 语句用于保证代码语句块的执行。控制总是传递给 Finally 语句块，而与 Try 语句块的退出方式无关。也就是说， Finally 代码块总是会被执行到。 Finally 关键字既可以与 Try 关键字单独配对使用，也可以与 Try—Catch 语句共同使用。
- 使用 Try 、 Catch 和 Finally 处理所有的异常
  - 应用程序最有可能的途径是合并前面两种错误处理技术：捕获错误、清除并继续执行应用程序。所有你要做的是在出错处理代码中使用 Try-Catch-Finally 语句。

使用 Try 和 Catch 捕获异常的一个小例子

- **例 ：**两个数组的元素相除，输出结果，并且捕捉异常。
  程序代码：

```
private void btnCal_Click(object sender, EventArgs e)
        {
            int a = int.Parse(txtA.Text.Trim());
            int b = int.Parse(txtB.Text.Trim());
            try
            {
                txtC.Text = (a / b).ToString();
            }
            catch (DivideByZeroException)
            {
                label2.Text = "can't divide by 0";
            }
        }
```

程序运行结果



程序代码解析

- 上面的例子体现了异常处理的优点，它允许程序响应错误并且继续运行，当出现了除数为 0 的时候，就会产生 DivideByZeroException 异常。此异常因为我们进行了捕捉，所以并不会终止程序，而是会报告错误信息而后继续执行。

使用 Try-Finally 语句清除异常示例

- **例**　使用 Try-Finally 语句清除异常示例。

  程序代码：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Example3of6
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" 验证 finally 语句与 goto 语句的先后顺序： ");
            try
            {
                Console.WriteLine("try");
                goto A;
            }
            finally
            {
                Console.WriteLine("finally");
            }
        A:
            Console.WriteLine("A");
            Console.ReadLine();
        }
    }
}
```

程序运行结果


C:\Windows\system32\cmd.exe
验证finally语句与goto语句的先后顺序:
try
finally
A

程序代码解析

- 由上面程序的运行结果看， Finally 语句块总是被执行，所以我们可以利用 Try-Finally 语句来清除异常。如果在执行 Finally 语句块时抛出了一个异常，那么这个异常会被传播到下一轮 Try 语句中，如果在异常传播的过程中友发生了另一个异常，那么这个异常将被丢失。

使用 Try-Catch-Finally 异常处理语句

- **例 6.4** 使用 Try-Catch-Finally 异常处理语句输入下标越界的情况 。
  程序代码：

```
private void btnOutPut_Click(object sender, EventArgs e)
        {
            try
            {
                int[] a = new int[10];
                for (int i = 0; i < 10; i++)
                    a[i] = i;
                int n = int.Parse(txtNum.Text.Trim());
                txtResult.Text = a[n].ToString();
            }
            catch
            {
                txtResult.Text = " 数组上溢 ";
            }
            finally
            {
                txtNum.Text = "";
                txtNum.Focus();
            }
        }
```

程序运行结果

6.3 抛出异常

- C# 中的异常封装成了一个 Exception 类，这个类位于 System 命名空间之中。抛出异常需要的是 Throw 语句和一个适当的异常类。

运行时的标准异常类型

异常类型	说明
Exception	所有异常对象的基类
SystemException	运行时产生的所有错误的基类
IndexOutOfRangeException	当一个数组的下标超出范围时，运行时引发
NullReferenceException	当一个空对象被引用时引发
InvalidOperationException	当对方法的调用对对象的当前状态无效时，由某些方法引发所有参数异常的基类
ArgumentException	所有参数异常的基类
ArgumentNullException	在参数为空（不允许）的情况下，由方法引起
ArgumentOutOfRangeException	当参数不在一个给定范围之内时，由方法引发
InteropException	目标在或发生在 CLR 外面环境中的异常的基类
ComException	包含 COM 类的 HRESULT 信息的异常
SEHException	封装 win32 结构异常处理信息的异常

异常的几点 说明

- 通过使用 Throw 语句，我们可以手动抛出异常，实际上在上述的例子中我们已经使用 Throw 语句了，当满足某条件时，抛出了 ArithmeticException （）异常。
- 同样地，我们可以创建自己的异常。一般情况下，我们使用的都是预先定义好了的异常类，但是在实际的应用中，创建自己的异常类可能会更方便，可以允许异常类的使用者根据该异常类采取不同的手段。

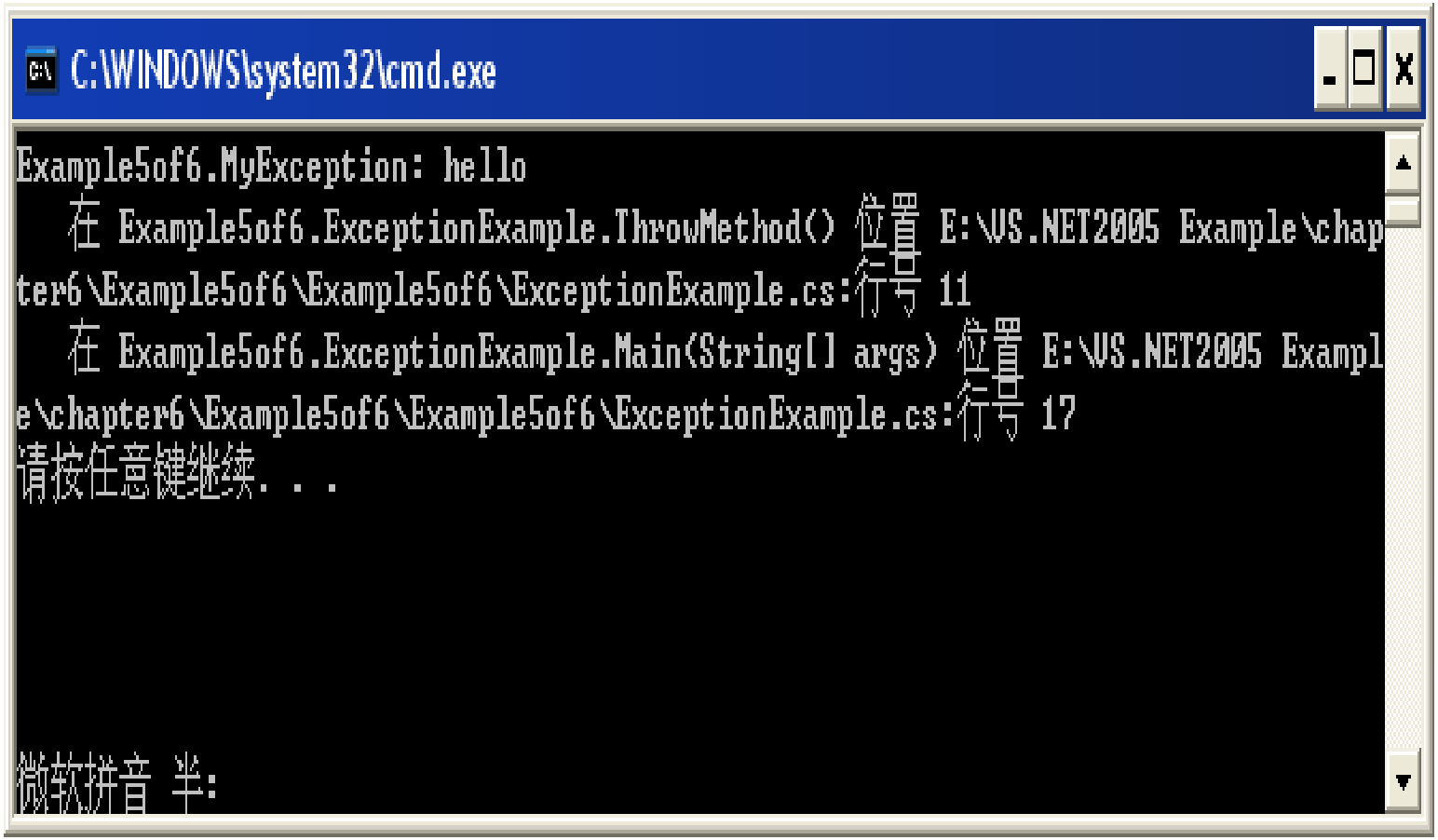
  创建自己的异常类要遵循两个规则：
  - （1）用 Exception 结束类名；
  - （2）它实现了所有三个被推荐的通用构造方法。

创建自己异常类 ExceptionExample

- 源代码

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Example5of6
{
    class ExceptionExample
    {
        public static void ThrowMethod()
        {
            throw new MyException("hello");
        }
        public  static void Main(string[] args)
        {
            try
            {
                ExceptionExample.ThrowMethod();
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
    }
    public class MyException:Exception
    {
        public MyException()
            :base()
        { }
        public MyException(string message)
            : base(message)
        { }
        public MyException(string message, Exception e)
            : base(message, e)
        { }
    }
}
```

运行结果


C:\WINDOWS\system32\cmd.exe
Example5of6.MyException: hello
在 Example5of6.ExceptionExample.ThrowMethod() 位置 E:\VS.NET2005 Example\chapter6\Example5of6\Example5of6\ExceptionExample.cs:行号 11
在 Example5of6.ExceptionExample.Main(String[] args) 位置 E:\VS.NET2005 Example\chapter6\Example5of6\Example5of6\ExceptionExample.cs:行号 17
请按任意键继续. . .
微软拼音 半:

小结

- 本章在前半部分主要讲述了程序调试的方法以及常见几种调试方法的演示，掌握了调试的技巧，对于我们广大程序员来说，是必不可少的。在本章中，重点讲述了设置断点方法进行的调试，当然还有其余很多不太规范的方法，读者们自己多摸索，积累经验，就可以快速准确地调试好自己的代码了。
- 之后，我们讲了异常处理的注意事项以及如何正确地抛出异常。异常的抛出对于我们在调试程序的过程中起着指路灯的作用。

谢谢！