第 6 章　程序调试与异常处理

本章要点

- 程序调试方法的使用
- 异常处理的方法
- 如何自行抛出异常

在详细学习训练了 Visual C#的基本语法及编程的基础知识基础上，我们训练了大量的 C#程序，也遇到了不少的问题，如何处理运行中的问题，就是我们接下来的应当讨论的问题—C#程序调试与异常处理。

无论是多么有经验的程序员，写代码的时候无论多么小心，也都会出错。因为人不是机器，肯定会有考虑不周全的时候，这样就会造成错误。Visual Studio 2012 环境中提供了基本的语法检查以及错误识别，对于一般的小错误，稍微有点编程经验的人员通过运行时的错误提示就可以轻松地化解。

但是，也有这样的时候：我们的程序本身看不出来有什么错误，运行时也能得到结果，可是结果却不是所预期的，这时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)。

异常(Exception)是运行时产生的错误。使用 Visual C#的异常处理子系统，我们能够以标准化并可控制的方式来处理运行时错误。C#异常处理的方式是 C++和 Java 所使用的方法的混合以及改进。因此，对于具有这些语言背景的读者，这可能是非常熟悉的。Visual C#异常处理的与众不同之处在于其更加清晰、直接的体现。

异常处理的主要优点是：它自动操作许多错误处理代码，而以前必须“手动”将它们输入到大型程序中。例如，在没有大型异常处理的计算机语言中，方法失败时必须返回错误代码，而且每次调用方法时都必须手动检验这些值。此过程不但繁杂，而且极其容易出错。异常处理通过允许程序定义代码块来简化错误处理，此代码块称为异常处理程序，出现错误时自动执行它。每次都手动检查特殊操作或方法调用是成功还是失败是不必要的。如果产生错误，那么可以通过异常处理自动地解决。

6.1 程序调试和调试方法

上面已经指出，程序在编写的过程中会因为种种原因而出现错误，总结起来，错误主要有以下 3 类。

（1）语法错误：语法错误是初学者常犯的错误，针对这种错误，在编译阶段程序可以自动地跳到错误之处，很容易修改。

（2）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，这种错误会造成程序的中断，可以使用 try-catch-finally 语句来解决。

（3）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，或者是与我们所预期的不一样。

下面通过经典的例子调试来演示调试的方法。

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();
        }
```

程序运行结果如图 6.1 所示。

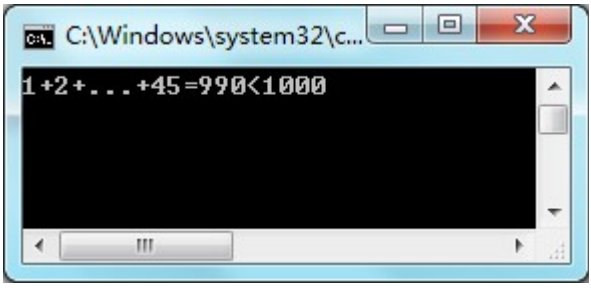

图 6.1 程序运行结果

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，下面来探讨一下如何通过调试来查询错误。

在代码中设置 3 个断点。断点设置的方式是：单击所需断点语句最左边，则会出现实心圆点。再次单击，圆点则会消失。如图 6.2 所示，显示设置了 3 个断点，单击【运行】按钮，则程序会停在第一个断点上，箭头停靠的行表示当前运行的行，图中的小窗口可以通过菜单“调试(D)”下面的“窗口(W)/自

动窗口(A)”命令来打开，按 F11 键可逐语句地运行，这时便可在窗口显示运行到某阶段时变量的值。

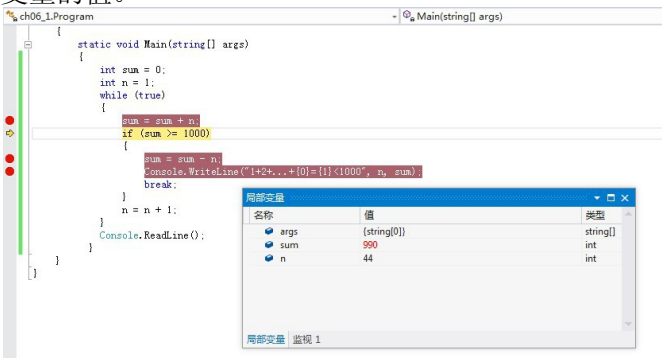

图 6.2　设置断点

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，所以在代码 sum=sum-n;后面添加代码 n=n−1;就可以了。

修改之后程序的运行结果如图 6.3 所示。

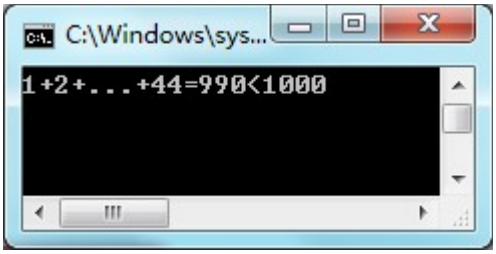

图 6.3　例 6.1 修改之后的运行结果

以上只是介绍了众多调试方法的一种，还有很多不正规的方法，比如用 MessageBox 显示当前变量的值，也很明确，读者可以在编程过程中多思考，面对这些错误时就可以迎刃而解了。

6.2 异 常 处 理

本章的上一节已经介绍了如何在应用程序的开发过程中查询和更改错误，以便使我们的程序能够正确运行。另外，还可以进行错误预料，以使程序更加健壮，可以处理错误代码，而不必中断程序的进行。这就是异常处理的目的，下面我们简要介绍一下异常处理的特点以及处理它们的方式。

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，大概地了解每个关键字在异常处理中的作用是有益的，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。

6.2.1 异常处理的注意事项

在 C#的异常处理中应该注意下列事项：

- 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符时，抛出 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。

6.2.2 异常处理中使用的语句

1. 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。

例 6.2 两个数组的元素相除，输出结果，并且捕捉异常。

【步骤 1】新建 windows 窗体应用程序，程序界面如图 6.4 所示。

图 6.4 程序界面

【步骤 2】窗体和窗体上各控件的属性设置，如表 6.1 所示：

表 6.1 控件属性列表

控件类型	控件名称	属性	设置结果
Form	Form1	Text	整数相除
Label	Label1	Text	/

	Label2	Text	为空
TextBox	TextBox1	Name	txtA
	TextBox2	Name	txtB
	TextBox3	Name	txtC
		ReadOnly	true
Button	Button1	Name	btnCal
		Text	=

修改属性后的窗体如图 6.5 所示：

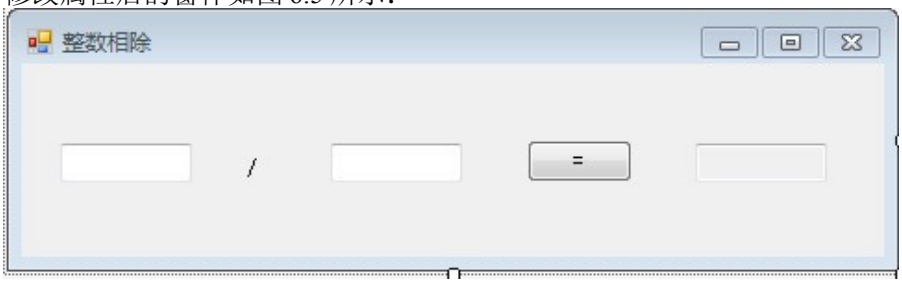

图 6.5 修改属性后的程序界面

【步骤 3】编写按钮的单击（Click）事件，代码如下。

```
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";
            }
        }
```

【步骤 4】运行程序，结果如下：

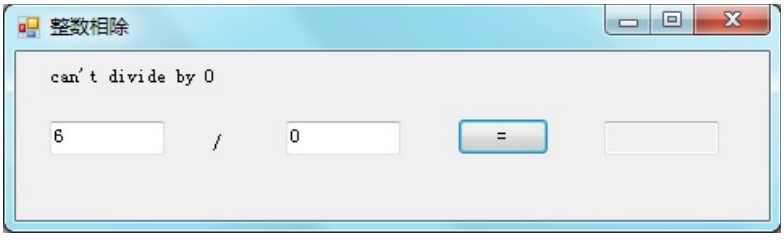

图 6.6 程序运行结果

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。对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。

2. 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。

例 6.3 使用 try-finally 语句清除异常。

程序代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ch06_3
{
    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();
        }
    }
}
```

程序运行结果如图 6.7 所示。

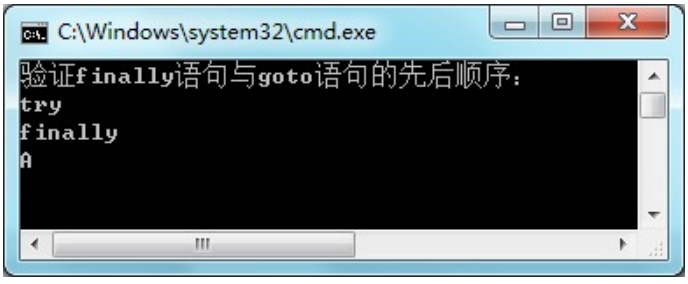

图 6.7 程序运行结果

从上面程序的运行结果来看，finally 语句块总是被执行，所以可以利用 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又发生了另一个异常，那么这个异常将会被丢失。

3. 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。所需做的只是在出错处理代码中使用 try-catch-finally 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，在 finally 块中清除输入的数据，等待多次输出。

【步骤 1】新建 windows 窗体应用程序，程序界面如图 6.8 所示。

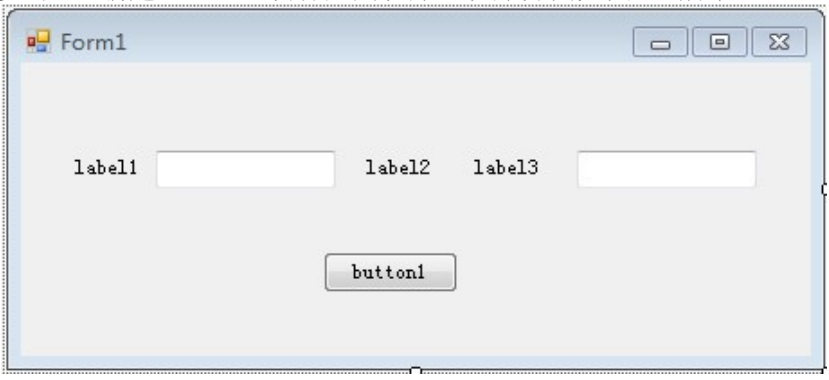

图 6.8 程序界面

【步骤 2】窗体和窗体上各控件的属性设置，如表 6.2 所示：

表 6.2 控件属性列表

控件类型	控件名称	属性	设置结果
Label	Label1	Text	a[
	Label2	Text	]
	Label3	Text	=
TextBox	TextBox1	Name	txtNum
	TextBox2	Name	txtResult
Button	Button1	Name	btnOutPut
		Text	输出

修改属性后的窗体如图 6.9 所示：

图 6.9 修改属性后的程序界面

【步骤 3】编写按钮的单击（Click）事件，代码如下。

```
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();
            }
        }
```

【步骤 4】运行程序，结果如图 6.10 所示。

图 6.10 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 命名空间中。

抛出异常需要的是 throw 语句和一个适当的异常类。表 6.3 提供了运行时的标准异常。

表 6.3 常见标准异常及说明

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

通过使用 throw 语句可以手动抛出异常。

同样地，我们可以创建自己的异常。一般情况下，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：

- 用 Exception 结束类名。
- 它实现了所有 3 个被推荐的通用构造方法。

下面是创建自己异常类 ExceptionExample 的源代码：

```
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)
        { }
    }
}
```

运行结果如图 6.11 所示。

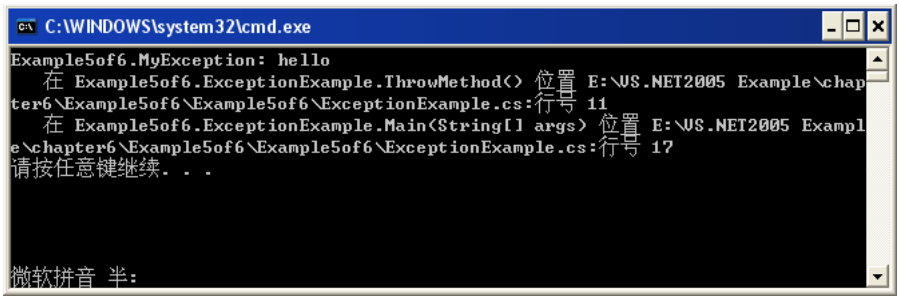

图 6.11 创建自己异常类 ExceptionExample 的运行结果

可以在 catch 语句中使用我们自己创建的异常类，来代替系统已经定义的异常类，可能抛出的新异常的客户代码可以按规定的 catch 代码发挥作用。

当使用自己的命名空间编写一个类库时，也要把异常放在该命名空间。尽管它并没有出现在这个例子中，还是应该使用适当的属性，为扩展了的错误信息扩充你的异常类。

6.4 案 例 实 训

1. 案例说明

使用 try…catch 结构实现数组的累加。

2. 程序代码

程序代码如下：

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

```
            int[] a = new int[10];
            int sum = 0;
            for (int i = 0; i < 10; i--)
            {
                a[i] = i * i;
                sum = sum + a[i];
            }
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
```

3. 运行结果

程序运行结果如图 6.12 所示。

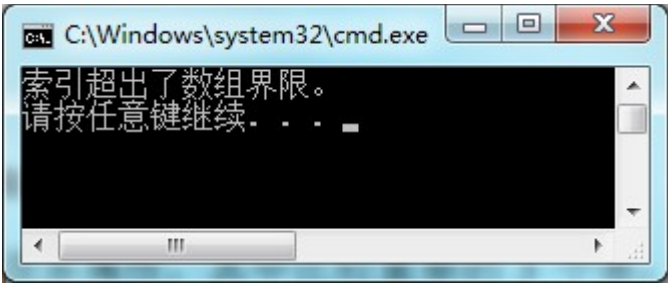

图 6.12　案例的运行结果

6.5 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的几种调试方法的演示。掌握调试的技巧，对于程序员来说是必不可少的。在本章中，重点讲述了用设置断点方法进行的调试，当然还有其他很多不太规范的方法，读者可以自己多摸索，积累经验，就可以快速准确地调试好自己的代码了。

本章最后还讲了异常处理的注意事项以及如何正确地抛出异常。异常的抛出在调试程序的过程中起着指路灯的作用。

6.6 习　　题

1. 选择题

(1) 异常类对象均为（ ）类的对象。

A.System.Exception　　B.System.Atttribute

C.System.Const　　D.System.Reflection

(2) 最难以排查的错误属于（ ）

A.编译错误　　B.运行时错误　　C.逻辑错误　D.系统错误

(3) 当访问索引超出数组界限的数组元素时引发的异常类是（ ）

A.AccessException　　B. ArgumentException

C.ArgumentOutOfRangeException

D.IndexOutOfRangeException

(4) 允许用户在名称列输入一个变量或者是表达式，在使用断点逐步执行程序时，可以全程监视变量或表达式的变化情况，这种调试工具是（ ）

A.断点　　B.临时表达式

C.单步运行　　D.监视