第 7 章　错误和异常处理

概述

本章将介绍异常处理的常见办法，以及应用程序的调试方法，以减少异常产生的概率，并尽量避免未处理的异常，从而使得应用程序健壮性、容错性增强。

重点及难点

异常及异常处理。

重点及难点学习指导建议

分析本章项目，结合本章知识点的介绍，查看 C＃中异常及异常处理项目中的应用，更好地掌握本章的重点及难点。

本章项目导引——控制台版 21 点游戏设计与实现

事实上，一个再优秀的程序员，一个再完善的应用程序都可能会存在或多或少的错误或异常。因为程序的使用者有时并不一定按照程序员的预想来使用程序，我们永远都无法预知用户的行为，异常可能发生在程序的任何阶段。本章将介绍异常处理的常见办法，以及应用程序的调试方法，以减少异常产生的概率，并尽量避免未处理的异常，从而使得应用程序健壮性、容错性增强。

7.1 项目构思

当使用Ctrl+F5执行程序时,是否经常产生“未处理的异常”这样的错误提示?或者使用F5执行程序,程序没有顺利地执行完毕,而是中断在其中的某一行,提示产生了异常?试想如果应用程序发布并已经提交给用户,这些异常会对产品造成怎样的影响!

异常的产生对于终端用户来说是非常不友好的,如何避免异常的产生呢?

7.2 项目分析

1.开发思路

应用程序中出现异常及异常处。

2.新知识

学会使用visual studio的调试。错误异常处理。

7.3 项目实施

1.两个整数相加的异常处理

内容:创建windows窗体应用程序,实现两个int类型数的相加。

实现:

(1)创建控制台应用程序Demo7-1,并保存到适当的位置。

(2)在Main()函数中添加如下代码:

```
static void Main(string[] args)
{
    Console.WriteLine("Please input two intergers:");
    Console.WriteLine("The first one:");
    int operator1 = Int32.Parse(Console.ReadLine());
    Console.WriteLine("Second one:");
    int operator2 = Int32.Parse(Console.ReadLine());
    Console.WriteLine(operator1 + operator2);
}
```

(3)使用F5执行程序,并输入非整型数据,可以得到如图7-1所示的错误提示。

图 7-1　输入非法字符引发的异常

(4)修改 Main(),函数的代码体,加入异常处理部分,具体如下:

```
static void Main(string[] args)
{
    int operator1=0;
    int operator2=0;

    Console.WriteLine("Please input two integers:");
    Console.WriteLine("The first one:");
    while (true)
    {
        try
        {
            operator1 = Int32.Parse(Console.ReadLine());
            break;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine("Please input the integer again!");
        }
    }
    Console.WriteLine("Second one:");
    while (true)
    {
        try
        {
            operator2 = Int32.Parse(Console.ReadLine());
            break;
```

```
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("Please input the integer again!");
            }
        } Console.WriteLine(operator1 + operator2);
    }
```

(5)使用 Ctrl+F5 执行程序,可以得到如图 7-2 所示的结果。

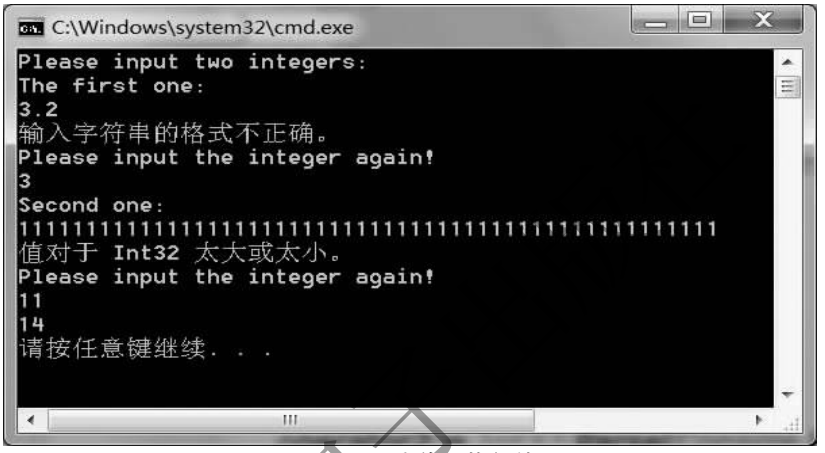

图 7-2 项目实施 1 执行结果

分析:

(1)用户可以输入任何值。但是定义用于存储用户输入和用于存储运算结果的变量均为 int 类型,当用户输入与定义的变量类型不符时就会导致异常。

(2)将被加数的值赋值为非整型数值,这时由于输入的是字符串而存储加数的变量为 int 类型,所以引发了 FormatException 类型的异常。

(3)在后台代码中添加异常处理程序,当程序发现被加数输入出现异常时,跳出 try 块然后执行 catch 块中的内容,提示用户输入不正确并请用户重新输入。当输入无误,则执行加法操作。

2.自定义异常

内容:自定义异常类,并测试该类。

实现:

(1)创建控制台应用程序 Demo7-2,并保存到适当的位置。

(2)在 Program.cs 文件中添加如下代码:

```
class Program
    {
        static void Main(string[] args)
        {
            TestClass c = new TestClass();
```

```
            c.TestMethod();
        }
    }
    /// <summary>
    /// 用户自定义异常类
    /// </summary>
    class TestException:ApplicationException
    {
        private string message;
        public TestException()
        {
            message = "这是一个用户自定义的异常类;
        }

        public override string Message
        {
            get
            {
                return message;
            }
        }
    }
    /// <summary>
    /// 用户自定义测试类
    /// </summary>
    class TestClass
    {
        public TestClass()
        {
            Console.WriteLine("这是一个测试类!");
        }

        public void TestMethod()
        {
            try
            {
                TestException e=new TestException ();
                throw e;
            }
            catch(TestException e)
            {
```

```
                Console.WriteLine(e.Message);
            }
        }
    }
```

(3)使用Ctrl+F5执行程序,可以得到如图7-3所示的结果。

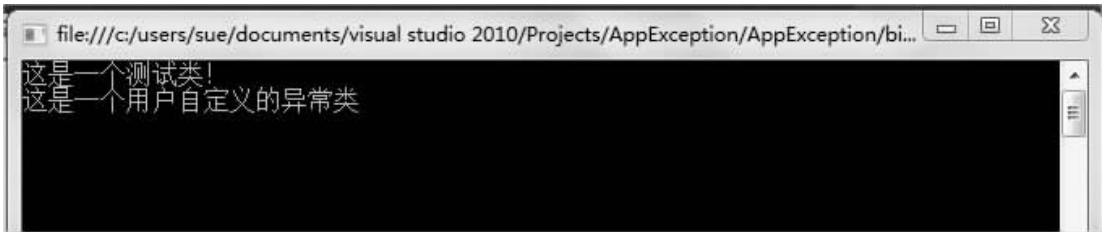

图7-3 自定义异常类实例运行结果

分析:

(1)在异常类TestException中重载了Exception类的Message属性,该属性的值与字段message的值一致。由于字段的值在构造函数中被初始化为字符串常数,因此该属性的值一直保持为“这是一个自定义的异常类”。

(2)在TestClass的TestMethod方法中创建了TestException类的实例,并通过throw语句将其抛出,从而产生了异常。在catch语句中捕获的异常就是该TestException类的实例。因此,输出的提示也是该类的Message属性值。

(3)Main()函数中的代码使TestMethod()方法得到了调用,因此控制台输出了消息“这是一个自定义的异常类”。

7.4 知识点详解

7.4.1 Visual Studio中的调试

当应用程序在执行过程中遇到异常的情况,它首先会抛出一个异常,然后终止该方法直到找到异常处理程序。这就说明如果当前运行的方法没有相应的异常处理,那么该方法将被终止,如果找不到对应的方法处理它,该异常则会交给公共语言运行时处理,它会将程序终止。

可引发异常的情况大概有以下几种:代码中有错误、公共语言运行库出现意外、用户自定意抛出的异常、操作系统资源不可访问等。

程序编好后,用各种手段进行查错和排错的过程就是调试的过程。程序的正确性不仅仅表现在完成正常功能上,更重要的是能对意外情况进行正确的处理。在Visual Studio中,点击菜单栏的调试-启动调试就可调试当前程序,如图7-4所示。或单击 ▶ ,也可使用快捷健F5。

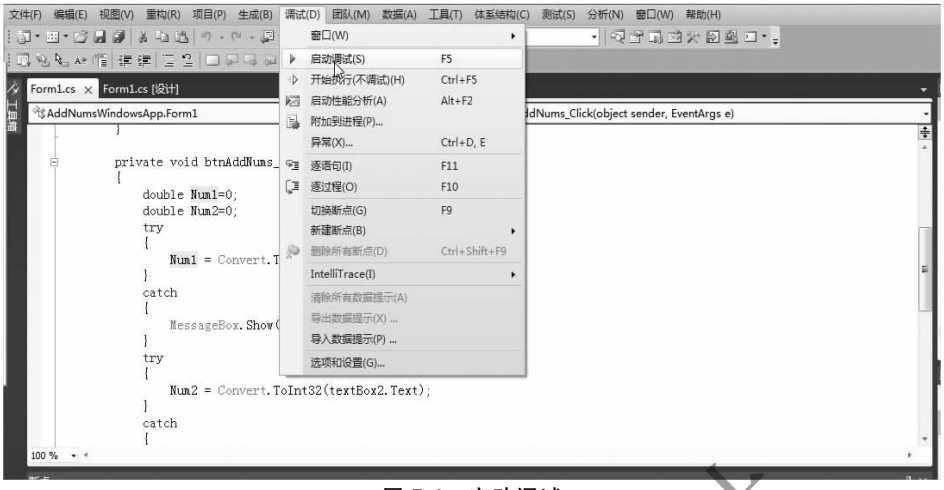

图 7-4　启动调试

在 C# 中调试的主要技术就是设置断点。断点是调试器的功能之一，可以让程序在需要的地方中断，从而方便其分析。

在代码左边的 5 毫米宽的竖条那里点左键就能加入断点，如图 7-5 所示，再点一下可以消去断点。或者点击调试-新建断点，如图 7-6 所示。

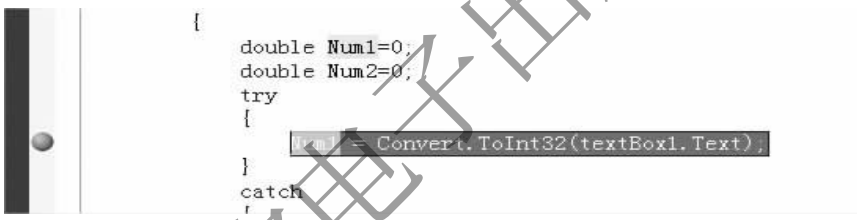

图 7-5　设置断点

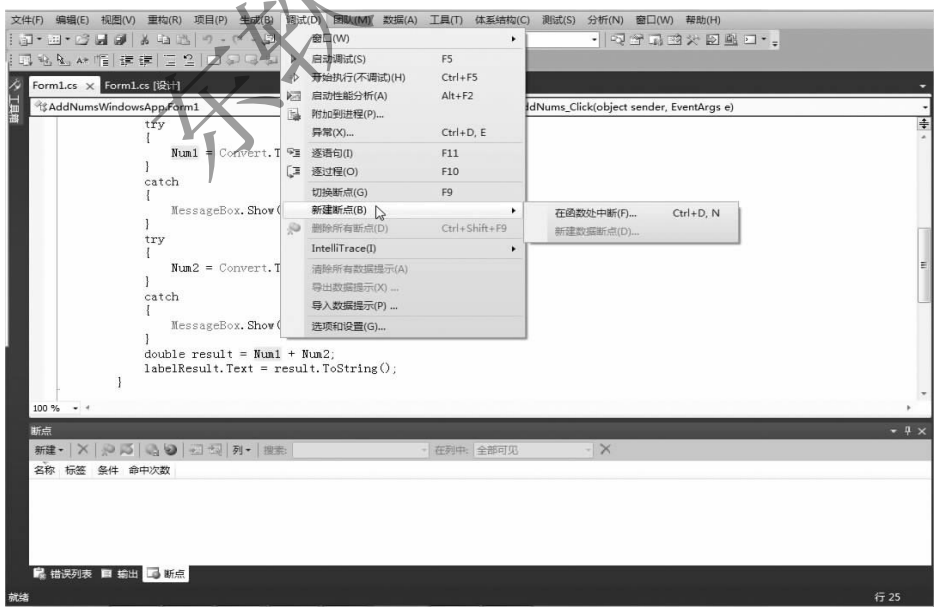

图 7-6　设置断点

也可以打开断点窗口，点击调试→窗口→断点。断点窗口如图7-7所示。在这里可以搜索断点，删除断点等。

图7-7 断点窗口

程序在运行过程中如果遇到断点，一般是要查看变量值。这是只需把鼠标放到该变量的名称上，就会出现一个显示该变量值的方框，这个方框还可以被扩展，显示变量的详细信息。如图7-8所示。

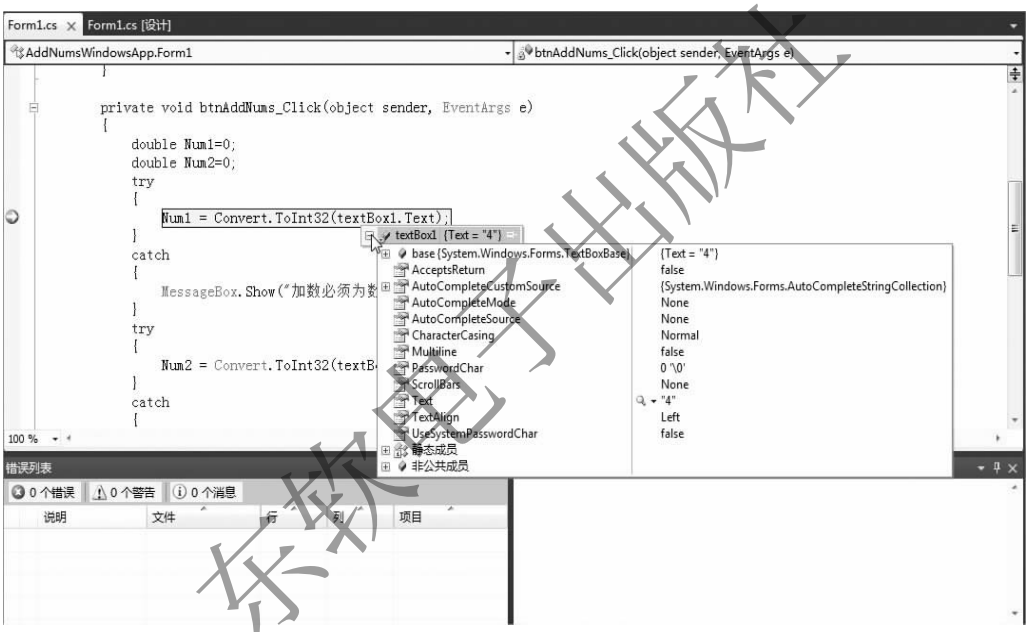

图7-8 显示变量的详细信息

在Visual Studio中还提供了许多快捷键以方便程序员调试程序，表7-1介绍了较为常用的几种快捷键：

表7-1 几种常用的快捷键

命令名称	快捷键	说明
启动调试	F5	自动附加调试器，从项目-属性对话框中指定启动项目要运行的应用程序。如果当前为中断模式，则更改为继续
逐过程调试	F10	执行单行代码，遇到函数直接跳过到一下行，不进入函数内部
逐语句调试	F11	执行单行代码，如果遇到函数直接进入函数内部
执行不调试	Ctrl + F5	在不调用调试器的情况下运行代码

（续表）

命令名称	快捷键	说明
运行到光标处	Ctrl + F10	在中断模式下，从当前语句继续执行代码，直到到选定语句为止
断点	Ctrl + D，Ctrl + B	显示“断点”对话框，可以在其中添加和修改断点
启用断点	Ctrl + F9	将断点从禁用状态切换到启用状态
切换断点	F9	在当前行设置或移除断点
全部中断	Ctrl + Alt + Break	临时停止执行调试会话中的所有进程。仅可用于运行模式
重新启动	Ctrl + Shift + F5	结束当前调试的程序，重新生成并从头开始运行应用程序 可用于运行模式和中断模式
停止调试	Shift + F5	停止运行程序中的当前应用程序 可用于中断模式和运行模式。

7.4.2　异常类的定义和使用

一个好的应用程序应包含大量处理异常的方法，当异常出现时就会自动创建一个包含有利于问题跟踪信息的对象。程序员既可以自己创建异常类也可以使用预定义的异常类。

1.异常基类(Exception)

在.NET 中，Exception 类表示异常的基类，其他异常均是从 Exception 类中继承而来的。

使用如下代码定义异常：

Exception e；

Exception 有四种构造函数，表 7-2 给出了这四种构造函数及含义：

表 7-2　　Exception 构造函数

构造函数	说明
Exception()	详细信息为 null 的新异常
Exception(String message)	带指定详细信息的新异常
Exception (String message, Exception innerException)	带指定详细消息和原因的新异常
Exception(Exception innerException)	根据指定的原因和（innerException == null ? null : innerException. toString()）的详细消息构造新异常

2.异常类

C#除了 Exception 类之外还有许多其他派生类。表 7-3 列出可能遇到异常类：

表 7-3 常用异常类

异常类	说明
SystemException	该类是 System 命名空间中所有其他异常类的基类
ApplicationException	当应用程序发生非致命错误时会引发该异常
ArgumentException	当处理参数无效时会引发该异常
FormatException	当处理参数格式错误时会引发该异常
MemberAccessException	当访问类的成员失败时会引发该异常(失败原因可能是要访问成员不存在或没有足够权限)
IndexOutOfException	当下标超出了数组长度时会引发该异常
ArrayTypeMismatchException	当在数组中存储了数据类型不正确的元素时会引发该异常
RankException	当维数错误时会引发该异常
IOException	当进行文件输入输出操作时会引发该异常

3.抛出异常

抛出异常与由于代码出现错误而出现异常的现象是相同的。在程序中使用 throw 语句来抛出异常,throw 用于发出在程序执行期间出现异常情况的信号。该异常可以在程序中进行捕获和处理,也可以在调用方法中进行捕获和处理。

throw 语句的形式为:throw 异常类的实例

执行代码“throw new Exception();”将得到如图 7-9 的结果。

图 7-9 抛出异常运行结果

应用程序在执行过程中遇到 throw 关键字时会出现包含异常名称以及异常类型的提示。

4.捕捉异常

在程序中利用异常或异常处理程序可以很好的将主逻辑代码与错误代码区分开。

(1)try 块和 catch 块

将代码放到 try 块中,当程序运行时,会首先尝试执行 try 里的所有语句,如果这些语句都没有异常再逐一运行这些语句,直到全部运行完。如果发现异常,则跳出 try 块,进入 catch 块中执行。我们可以在 try 后写一个或多个 catch 块来处理异常,try 中任何语句发生错误时都会抛出一个异常,然后程序将检查 catch 块中的程序并选择一个匹配的程序来处理。

(2)finally 块

不管是否捕捉到了异常,finally 块中的代码都会被执行。例如:在执行数据库操作时数据库已经被打开,这时发生了错误,当在 try 块中捕捉到异常后,会直接跳转到 catch 块中执行,那么数据库就一直没有关闭,这就造成了资源浪费或其余步骤不能继续进行。

Finally 与 try 的退出方式没有关系,用于保证语句的执行。

Finally 既可以与 try 块配对使用也可以与 try…catch 共同使用。

(3)使用多个 catch 处理异常

不同的程序可能会抛出不同类型的异常,程序员可以编写多个 catch 处理程序来解决这个问题,当引发异常时,程序会按顺序来查看每个 catch 块,并执行第一个与异常类型匹配的语句,其它的 catch 块将被忽略。需要注意的是:在一系列的 catch 语句中,异常子类应该位于其父类之前。多重 catch 语法如下:

```
try
{
    //正常程序处理语句
}
catch(派生异常类 1 参数)
{
    //一种派生类异常
}
catch(派生异常类 2 参数)
{
    //一种派生类异常
}
… …
catch(派生异常类 n 参数)
{
    //另一种派生类异常
}
catch(Exception e)
{
    //基类异常
}
```

(4)try块的嵌套

try块是可以嵌套使用的,代码如下所示:

```
try
    {
        Console.WriteLine("A");
        try
        {
              Console.WriteLine("B");
        }
        catch
        {
              Console.WriteLine("C");
        }
        finally
        {
            Console.WriteLine("D");
        }
        Console.WriteLine("E");
        }
        catch
        {
            Console.WriteLine("F");
        }
        finally
        {
            Console.WriteLine("G");
    }
```


如果异常出现在外层try块中,并且在输出A或E的语句中,那么该异常将由外层catch捕获,执行输出F,然后执行外层finally块,输出G。

如果异常出现在内层try块中,并且有内层中有合适的catch块来处理它,这时程序将在内层处理该异常,然后执行内层中的finally块,再继续执行外层的try块,输出E。

如果异常出现在内层try块中,但是内层中没有合适的catch块来处理它,这时就要执行内层finally块,输出D。然后退出内层的try块,继续在外层的catch块中搜索合适的处理程序,如果外层catch块中有合适的处理程序就执行该程序,然后执行外层finally块。如果外层的catch块中没有合适的处理程序,控制权将交还给.NET运行库。

5.自定义异常类

程序员可以直接使用系统已经定义的异常类,也可以自定义异常类。当创建一个异常类时,应该是从System.ApplicationException类派生我们的异常类,而不System.Exception类。

6.异常属性

异常也有许多常用的属性，这些属性对开发人员判断异常发生的原因有很大帮助。表 7-4 列出了几种常用的异常类：

表 7-4　　几种常用异常类的属性

属性	说明
Data 属性	此属性可以以键值对的形式来保存任意数据，给异常提供了额外的信息，方便后续处理
HelpLink 属性	此属性可以保存能提供引起异常原因的帮助文件
Message 属性	此属性可以提供描述错误的文本
Source 属性	此属性可以提供导致异常的对象名或应用程序
StackTrace 属性	此属性包含堆栈跟踪，用来确定错误发生的位置

7.5　本章小结

本章介绍了 Visual Studio 中的调试技术，包括如何设置断点、单步执行调试等技术。介绍了异常及异常处理方法，如何使用 try{}catch{}finally{}程序结构处理异常，并介绍了异常类、抛出异常、捕捉异常、异常属性等知识点，通过本章学习，学生可以章无程序调试、及异常处理技术。

7.6　习题

(1)在调试中在单步执行模式下，F10，F11 键的作用是什么？

(2)简述 try{}catch{}finally{}程序结构中每段结构的作用。

(3)用什么语句抛出异常？如果在原方法中不处理异常，该异常在哪里处理？

(4)给出四种异常常用的属性，并简述其含义。