CHAPTER

第5章 异常处理

本章要点

- 什么是异常；
- 怎样捕获异常；
- 怎样处理异常；
- 通用异常类及自定义异常类。

虽然程序员尽其所能开发并完善程序，但是最后运行的程序还是会面临各种各样可能存在的缺陷与非正常情况需要处理。本章将讨论C#中的一个重要功能——异常处理。通过本章的学习将了解什么是异常，怎样捕获异常，怎样处理异常。最后本章将对已有的通用异常类做一个简单介绍，并介绍怎样编写自定义异常类，从而为程序员编写健壮的代码提供参考。

5.1 异常实例

【例 5-1】 字符串转换成整数时的异常处理。

首先按照下述方法创建一个 Windows 窗体应用程序（以后章节将具体讨论 Windows 窗体应用程序）：

在 Visual Studio 2008 的主窗口上选择“文件”→“新建”→“项目”命令，出现图 5-1 所示的“新建项目”对话框，并设定相应属性。

选择 Windows 窗体应用程序，并保存在 E:\Code 目录下，项目名为 ExceptTest，单击“确定”按钮之后，系统自动生成一个窗体 Form1，在 Form1 中放置 TextBox 控件和 Button 控件，如图 5-2 所示。

本例实现的功能是在 TextBox 输入框中输入一个整数，单击窗体上“确定”按钮之后，系统将把数字乘以 2，并显示在原来的输入框中。双击“确定”按钮并编写相应的代码如下：

```
using System;
using System.Collections.Generic;
```

图 5-1 “新建项目”对话框

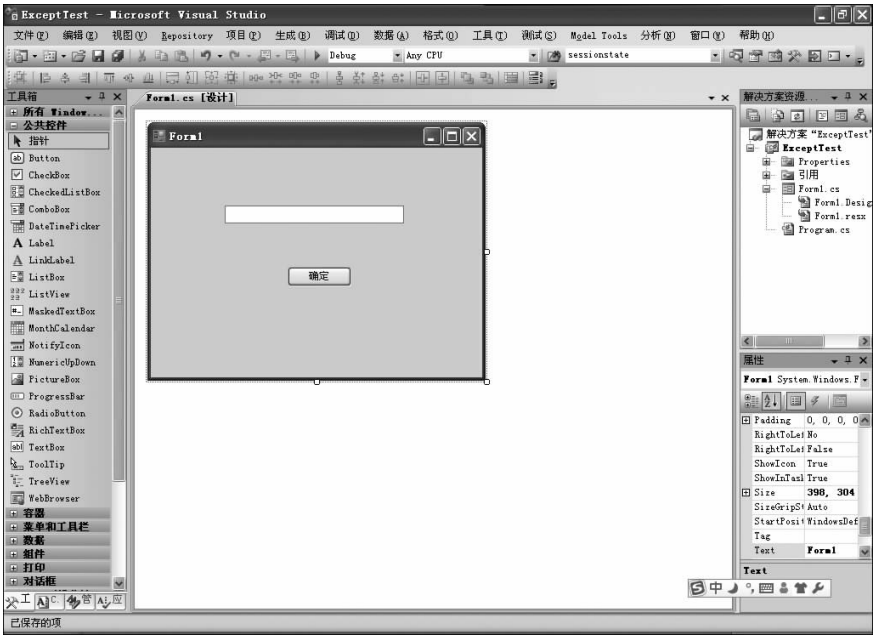

图 5-2 项目控件设定

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace ExceptTest
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            int tmp;
            try
            {
                tmp=Int32.Parse(textBox1.Text.Trim());
                                                    //将 TextBox 中的字符串转换成整数
                tmp=tmp * 2;
                textBox1.Text=tmp.ToString();   //将整数转换成 String
            }
            catch
            {  //如果不是整数形式的字符串,比如输入"ABC123",则产生异常情况
                textBox1.Text="你输入的不是整数";
            }
        }
    }
}
```

编译该程序并运行,在出现的窗体中输入 123,单击"确定"按钮测试一下效果,然后输入 123.45 或者字符串 abc 并点击"确定"按钮测试实际的效果。本例的代码非常简单,在 Button 的 Click 事件中首先获取了 TextBox 控件中的输入信息,然后转换成整数并乘以 2,最后把新的值显示在 TextBox 控件的输入框中。相信读者已经注意到里面有 try…catch…部分的代码,这就是下面要讲述的 C# 中的结构化异常处理功能。

在上面的例子中如果输入的是整数,程序能正常运行,如果输入的不是整数,那么系统将运行 catch 部分的代码,从而保证了程序的健壮性。

5.2 C# 异常处理过程

所谓的异常就是指应用程序运行时遇到的错误或者是一些意外的情况。在实际的程序开发中,无论代码编写质量如何好,程序还是需要能处理可能出现的错误。例如在连接网络数据库的情况下,可能发生网络问题导致数据库连接不上,此时必须处理有关的异常情况;或者在进行除法运算的时候可能会遇到除数为 0 的情况,此时也必须处理有关的异常情况。

前面的例子简单地演示了什么是异常以及异常的捕获与处理,在 C# 中提供了通用的异常处理结构,不过在详细讨论该结构之前,先来介绍一下 checked 和 unchecked 语句

的用法。

5.2.1 checked 与 unchecked 语句

在程序代码中往往会碰到有关的整数运算与转换，为了能更好地捕获该类型的异常，在 C# 中提供了 checked 和 unchecked 运算符。这两个运算符的作用是控制是否实施关于整型算术运算和转换的溢出检查；同时在 C# 中也提供了 checked 和 unchecked 语句；两者的功能完全等效，但是它们的作用范围却不一样：运算符只作用于表达式，而语句则作用于语句块。当有多行语句需要检查的时候，可以用 checked 语句来达到等效的功能。

C# 中这两个运算符的格式为：checked(表达式) 、unchecked(表达式)；而语句的格式则为：checked 语句块、unchecked 语句块。当整数运算的值太大超出系统规定的范围时(如超出了 32 位所能容纳的最大值)，此时就可以用 checked 与 unchecked 来控制不同的操作。

在 checked 上下文中，如果运算发生在一个常数表达式中，则会发生编译错误，否则当在运行时执行运算，则会抛出 System. OverflowException 异常；在 unchecked 上下文中计算的结果不被检查，系统自动对结果进行截断，也就是按照结果的位数，将有关的二进制位填入，其他溢出部分则不考虑，因此结果往往同我们的预期并不相同。在下面的示例代码中编译时并不报错，在实际运行的时候函数 F 将抛出异常，函数 G 则按照实际的结果取低 32 位，函数 H 则根据编译时设定的默认的溢出检查来处理(F 与 G 的结果中取其一)。

```
class Test
{
   static readonly int x=1000000;
   static readonly int y=1000000;
   static readonly int z=100000 * 100000;   //此处发生编译时错误
   static int F() {
      return checked(x * y);               //此处在运行时会抛出 OverflowException
   }
   static int G() {
      return unchecked(x * y);              //返回-727379968
   }
   static int H() {
      return x * y;              //取决于编译器的默认设置,默认情况下是 unchecked 的
   }
}
```

对于 checked 的处理可以加上 try-catch，以实现对溢出异常的处理。代码结构如下：

```
try {
    checked
{
⋮
```

```
}
    } catch(OverflowException e)
{  …   }
```

其中 checked 语句也可以换成 checked(表达式)的格式。

5.2.2 异常处理语句

前面的内容基本上描述了 C# 中异常处理语句的结构与使用方法,在 C# 中为了捕获与处理异常,系统提供了 3 个关键字: try、catch 和 finally;其典型的格式如下:

```
try {
    //要执行的代码,此代码可能产生异常
} catch(…)
{
    //异常处理语句
} catch(… )
{
    //异常处理语句
}
    //其他 catch 语句
finally
{
    //必须执行的语句
}
```

在本章开头的异常例子中,我们希望用户输入整数,但是可能会遇到输入为浮点数甚至是字符串的情况,当然我们可以在代码中加入 if 语句来判断是不是浮点数、是不是字符串等,但是这样会造成代码的繁琐,而且也达不到处理的通用性。

使用 try/catch 结构能有效地避免这种情况,通过将可能产生异常的代码放入到 try 语句块中,将捕获并处理异常的代码放入 catch 语句块中,能够达到非常好的效果。从上面异常的典型结构上也可以看出,在一段 try 代码之后可以有多个 catch 处理语句块,但是 try 语句块和 finally 语句块只能有一个。

在.NET 中所有的异常都是从 Exception 派生而来,异常从发生问题的代码区域引发,然后沿堆栈向上传递,直到应用程序处理它或程序终止。因此在实际的代码编写中,可以在 catch 语句中使用具体的异常类如 DivideByZeroException(除数为 0 的异常类);或者使用异常的基类 Exception 来简化代码的编写(如在信息管理系统中不管是网络问题还是数据库错误,我们只是简单捕获 Exception 然后给用户一个提示:不能连接系统)。

在上述典型结构中还有 finally 语句块,该语句块中的代码不管有没有发生异常都要执行,在实际的编码中往往用来释放一些系统资源,例如,关闭打开的文件或者释放系统的内存等。

增加了 try-catch-finally 之后,程序的流程是:首先进入 try 语句块,如果程序没有错误发生则执行 finally 中的语句;如果有错误则转入到 catch 语句块处理异常,异常处理完

成再执行 finally 中的代码。

5.2.3 throw 语句

在 C# 的异常处理结构中，throw 语句也是非常重要的，其基本功能是把捕获到的异常或者程序员指定的异常抛出。简单地说，throw 语句可以显式地引发异常或者将捕获到的异常再次抛出。查看下面取自 MSDN 的代码。

```
using System;
using System.IO;

public class ProcessFile
{
   public static void Main()
      {
      FileStream fs=null;
      try
      {
         //Opens a text tile.
         fs=new FileStream(@"C:\temp\data.txt", FileMode.Open);
         StreamReader sr=new StreamReader(fs);
         string line;

         //A value is read from the file and output to the console.
         line=sr.ReadLine();
         Console.WriteLine(line);
      }
      catch(FileNotFoundException e)
      {
         Console.WriteLine("[Data File Missing] {0}", e);
         throw new FileNotFoundException(@"data.txt not in c:\temp directory]",e);
      }
      finally
      {
         if (fs !=null)
            fs.Close();
      }
   }
}
```

该代码在捕获到 FileNotFoundException 异常的时候重新抛出了该异常，并添加了部分的新信息。程序员可以在自己的代码中抛出自定义的异常并做处理，后面还将介绍自定义异常类。

5.3 通用异常类

在 C# 中提供了一些通用的异常类，其部分类结构情况如图 5-3 所示。

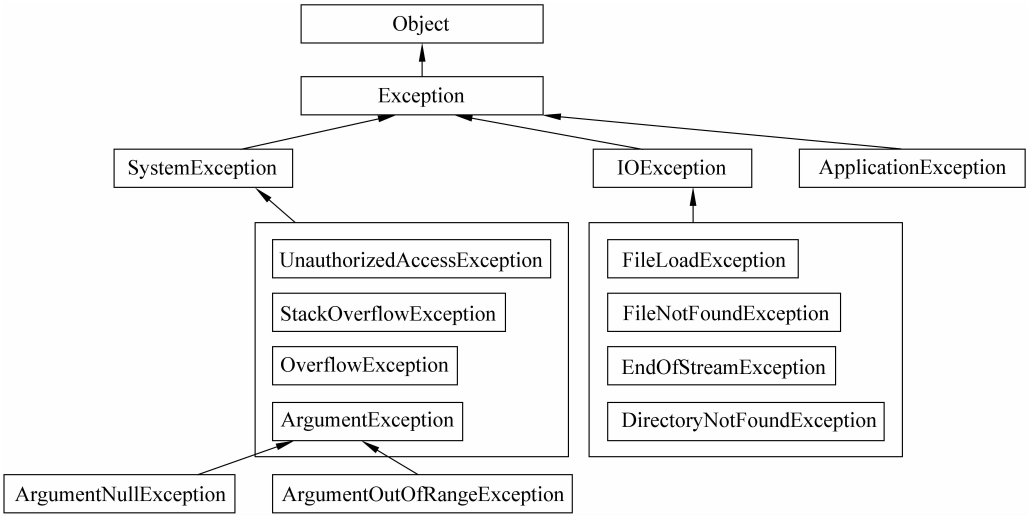

图 5-3 部分异常类结构图

所有的异常都是由 Exception 类派生而来，而 Exception 又是继承自 Object 类，.NET 中定义了大量的异常类，图 5-3 只是列举了其中一部分异常类，从图中所示的名字上面也能大概看出是何种类型的异常类，例如，IOException 与输入输出操作相关，里面细分了目录、文件操作时的一些异常类。

异常类 Exception 有几个属性，其中 Message 属性比较常用。对于常用的一些异常类，可以参考 MSDN 来查看更详细的介绍，程序员在实际的代码开发中可以只使用 Exception，当然也可以使用具体的异常类，甚至还可以使用自定义的异常类。

5.4 自定义异常类

如果通用的异常类不能满足实际的需要，程序员可以自定义异常类。创建自定义异常类时可以从 Exception 类继承或者从 Exception 的子类继承，具体的选择看实际的需要。下面编写一个自定义异常类并在代码中调用测试。

【例 5-2】 自定义异常类。

创建一个控制台应用程序新项目 ConsoleTest，新增一个类文件 ExceptTest，改写程序代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;

namespace ConsoleTest
{
    public class ExceptTest:Exception  //表示该类继承自 Exception 类
    {
        public ExceptTest()
        {
        }
        public ExceptTest(string message)
            : base(message)              //以 message 为参数调用基类的构造函数
        {
        }
        public ExceptTest(string message, Exception inner)
            : base(message, inner)      //以 message、inner 为参数调用基类的构造函数
        {
        }
    }
}
```

这个是自定义异常类，在其中仅简单地实现了几个构造函数。ConsoleTest 主文件代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleTest
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                throw new ExceptTest("这是自定义异常类测试程序");
                                                        //手动抛出一个自定义异常类
            }
            catch (ExceptTest ee)
            {
                Console.WriteLine(ee.Message.ToString());
            }
        }
    }
```

```
}
```

代码实现上非常简单，在 try-catch 中构造了一个 ExceptTest 的对象，并通过 throw 将该异常对象抛出，在 catch 语句块中我们捕获了 ExceptTest 异常，并通过 WriteLine 将里面的 Message 信息打印出来。运行程序看看实际的效果。

5.5 小结

本章简单讲解了异常的基本概念及 C＃中对异常的处理过程，通过使用. NET 提供的通用异常类能完成大部分的工作，对于有特定要求的异常类，可以通过自定义异常类来实现，在实际的代码编写中一定要好好利用异常处理，从而编写健壮的程序代码。

习题

1. 选择题

(1) 在 C＃中，通用的异常处理结构为 try-catch(　　)。

A. exception　　B. error　　C. finally　　D. other

(2) 在 C＃中，所有的异常类继承自(　　)类。

A. IOException　　B. SystemException

C. ApplicationException　　D. Exception

(3) 在 C＃中，程序使用(　　)语句可以抛出异常类。

A. catch　　B. throw　　C. try　　D. finally

(4) 阅读下面的 C＃代码：

```
public class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("try 语句");
        }
        catch (Exception e)
        {
            Console.WriteLine("catch 语句");
        }
        finally
        {
            Console.WriteLine("finally 语句");
        }
    }
```

```
}
```

该代码的运行结果是(　　)。

A.	B.	C.	D.
try 语句 catch 语句 finally 语句	try 语句 finally 语句	try 语句 catch 语句	try 语句

(5) 阅读下面的 C# 代码：

```
public class Program
{
      static void Main(string[] args)
      {
          int x=100000;
          int y=100000;
          int z;
          try
          {
              z=checked(x * y);
              Console.WriteLine("输出 z 的值");
          }
          catch (System.Exception ex)
          {
              Console.WriteLine("系统异常");
          }
      }
}
```

该代码的运行结果是(　　)。

A. 输出 z 的值　　B. 系统异常

C. 程序发生编译错误　　D. 程序运行时没有任何输出

2. 编程题

(1) 参考例 5-1,编程模拟一个信息管理系统的登录过程,在上面放置两个按钮,一个为“登录成功”,一个为“登录异常”,单击“登录成功”时用 MessageBox 输出“系统登录成功”信息,单击“登录异常”时用 MessageBox 输出“网络故障,系统无法登录”。

(2) 参考 5.2.3 节 throw 语句中的 MSDN 代码及例 5-2,编程实现一个自定义的文件读取异常类,类名为 MyFileException,并分别测试读取文件正常与异常的结果,结果的输出为“xxx 文件读取成功”、“xxx 文件读取不成功”。