第5章 异常处理

主要内容：程序的健壮性和稳定性是应用程序的基本要求。C#提供了强有力的异常处理能力，为程序的健壮性和稳定性奠定了技术基础。本章介绍了异常的概念、异常捕获和处理的基本原理，详细介绍了基于 try-catch 结构及相关结构进行异常捕获和处理的多种方法，并介绍了异常的抛出、重写以及用户自定义异常的相关技术。

教学目标：熟练掌握各种异常捕获和处理的方法以及它们之间的区别与联系，能够根据实际情况自定义满足需要的异常。

5.1 一个产生异常的简单程序

5.1.1 程序代码

创建一个 C#控制台应用程序 ExceptionPro，该程序能够捕获产生的异常，并进行相应的处理。代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ExceptionPro
{
    class Program
    {
        static void Main(string[] args)
        {
            int n, m;
            string s = Console.ReadLine();
            n = 0;
            try
            {
                m = Convert.ToInt16(s);                    //产生异常的语句
                Console.WriteLine("m = {0}", m);
            }
            catch (Exception e)                            //捕获异常
            {
```

```
                Console.WriteLine("产生异常:{0}", e.Message);     //处理异常
            }
            Console.ReadKey();
        }
    }
}
```

运行该程序,并输入相关字符串,如图 5.1 所示。

图 5.1　程序 ExceptionPro 的运行结果

5.1.2　异常处理过程分析

在上述程序运行过程中,由于输入了“12345abc78”,结果函数 Convert.ToInt16()试图将其转换为整数时产生了异常。但由于该异常被捕获,因而没有导致程序执行的非正常中止,而是在产生该异常时自动转向执行下列语句。

```
Console.WriteLine("产生异常：{0}", e.Message);
```

该语句的作用是输出产生异常的原因。

显然,可能出现异常的代码放在 try 块中,处理异常的代码则放在 catch 块中。当程序在运行过程中产生异常时,则会转向执行 catch 块中的代码,从而避免因异常的产生而导致程序运行的非正常中止。

由此可见,这种程序具有较强的错误处理能力,使得程序更加健壮和稳定。而这就是 try-catch 结构的作用。除了这种结构以外,还有 try-catch-catch、try-catch-catch-finally 等多种异常处理结构,它们的使用方法不尽相同,作用也不完全一样。下面将介绍异常的概念及它们的使用方法。

5.2　异常的捕获与处理

5.2.1　异常的概念

异常是指程序在运行过程(而非编译过程)中产生的错误。编译过程中的错误可以通过代码调试来避免,而对于一个中大规模的程序来说,异常一般是不能避免的(只能是减少)。而如何实现对难以预测的异常进行捕获和处理,这是一个健壮、稳定的程序所必须解决的问题。

C#从 Java 语言中引入异常处理的概念,并对其进行了扩展,从而形成了 try-catch 及

其相关结构。

5.2.2 try-catch 结构

最简单的异常处理结构是 try-catch 结构，其格式如下：

```
try
{
    //可能产生异常的代码
}
catch [(异常类 对象名)]
{
    //处理异常的代码
}
```

【说明】

(1) 在 try 块中编写可能产生异常的代码；在 catch 块中编写用于处理异常的代码。一旦在 try 块中有某一条语句执行时产生异常，程序立即转向执行 catch 块中的代码，而不会再执行该语句后面的其他语句。当然，如果 try 块中的语句都不产生异常，那么就不会有任何的 catch 块被执行。

(2) "异常类"用于决定要捕获的异常的类型，不同的异常类能捕获和处理不同的异常。常用的异常类如表 5.1 所示，其中 Exception 是所有其他异常类的基类，即其他异常类都是 Exception 类的派生类。显然，用 Exception 类可以捕获所有类型的异常，该类有两个常用的属性。

表 5.1 常用的异常类

ArithmeticException	在进行算术运算时可能会产生的异常，是 DivideByZeroException 和 OverflowException 的基类
ArrayTypeMismatchException	当由于存储元素的类型与数组元素的类型不匹配而导致存储失败时会产生此异常
DivideByZeroException	当用零除一个整型数据时会产生此异常
Exception	是所有异常类的基类，它可用于捕获所有类型的异常
FormatException	参数格式错误而引发的异常
IndexOutOfRangeException	当用一个小于零或大于数组边界的下标来访问一个数组元素时会产生此异常
IOException	该类用于处理进行文件输入输出操作时所引发的异常
NullReferenceException	当试图以 null 作为对象名来引用对象的成员时会产生此异常
OutOfMemoryException	当使用 new 来申请内存而失败时会产生此异常
OverflowException	当选中的上下文中所进行的算术运算、类型转换等而导致存储单元溢出时会产生此异常
SqlException	SQL 操作引起的异常
TypeInitializationException	当一个静态构造函数抛出一个异常且没有任何 catch 结构来捕获它时会产生此异常

① Message：它是一个 string 类型的只读属性，包含了异常原因的描述。例如，在程序 ExceptionPro 中，Message 属性返回的值是"输入字符串的格式不正确"。

② InnerException：它是一个 Exception 类型的只读属性，如果其值为 null，则表示该异常不是由另一个异常引发的，而是由系统内部产生的或者根据相关条件直接抛出的；如果其值不是 null，则表示当前异常是对另外一个异常的回应而被抛出的，“另外一个异常”保存在 InnerException 属性中（见例【5.3】）。

(3)“(异常类 对象名)”部分可以省略。如果省略这部分，则不管在 try 块中产生什么异常，程序都会转向执行 catch 块中的代码，但在这种情况下无法获取此异常的任何信息。

【例 5.1】 内存溢出异常的捕获和处理。

在程序执行过程中，有时需要向操作系统申请存储空间。但再大的内存空间都有可能被用完的时候，因此程序在申请较大块的存储空间时可能出现失败，这时会产生一个内存溢出的异常(OutOfMemoryException)，根据这个异常就可以决定下一步要采取什么样的动作，如中止程序的运行等。

在下面的 OutOfMemExc_Exa 程序中，申请了 20 000×30 000 个存储单元，结果超出了笔者机器的可用内存空间，因而产生了内存溢出异常。代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace OutOfMemExc_Exa
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int[,] a = new int[20000, 30000]; //申请存储空间
            }
            catch (OutOfMemoryException e)        //异常捕获与处理
            {
                Console.WriteLine("产生异常:{0}", e.Message);
            }
            Console.ReadKey();
        }
    }
}
```

执行该程序，结果如图 5.2 所示。

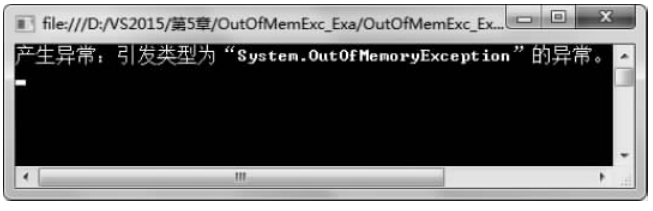

图 5.2 程序 OutOfMemExc_Exa 的运行结果

5.2.3　try-catch-catch 结构

从表 5.1 可以看到，存在多个不同的异常类。这意味着可以捕获和处理 try 块中可能出现的多个不同的异常，这就需要用到带多个 catch 块的 try-catch-catch 结构。

【例 5.2】 多个异常的捕获和处理。

下面程序 MultiExce_Pro 中，try 结构包含的两条语句在执行时都会产生异常，分别为 DivideByZeroException 异常和 OutOfMemoryException 异常。这两个异常分别由两个 catch 结构来捕获和处理。程序代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace MultiExce_Pro
{
    class Program
    {
        static void Main(string[] args)
        {
            int n, m;
            n = 30000;
            m = 30000;
            try
            {
                n = 1/(n - m);
                int[,] a = new int[n, n];
            }
            catch (OutOfMemoryException e1)
            {
                Console.WriteLine("内存溢出异常:{0}", e1.Message);
            }
            catch (DivideByZeroException e2)
            {
                Console.WriteLine("零除异常:{0}", e2.Message);
            }
            Console.ReadKey();
        }
    }
}
```

运行该程序，结果如图 5.3 所示。

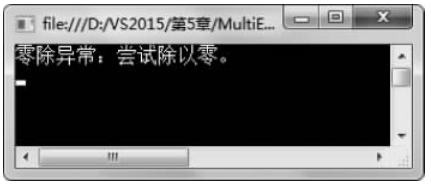

图 5.3　程序 MultiExce_Pro 的运行结果

本例中，try 块中的两条语句都能产生异常。由于第一条语句产生 DivideByZeroException 异常，程序立即转向执行“catch (DivideByZeroException e2)”部分，因此出现如图 5.3 所示的结果。如果将这两条语句的顺序对换，则运行结果将输出“内存溢出异常：

引发类型为'System.OutOfMemoryException'的异常"信息。

多个 catch 块在出现顺序上有何要求呢？这要分两种情况来讨论：①catch 后面的异常类之间没有继承关系(如 DivideByZeroException 和 System.OutOfMemoryException)，这时 catch 块的位置不分先后，即在前、在后都不影响程序的运行结果。例如，例【5.2】中的 catch 结构就属于这种情况。② catch 后面的异常类之间存在继承关系(如 DivideByZeroException 类继承了 ArithmeticException 类、所有异常类都继承了 Exception 类)，这时派生类所在的 catch 块必须放在基类所在的 catch 块的前面，因为前者的捕获范围小，后者的捕获范围大。例如，下面代码中的两个 catch 块的顺序是不能颠倒的，否则无法通过编译检查。

```
int n = 1, m = 1;
try
{
    n = 1/(n - m);
}
catch (DivideByZeroException e)                    //派生类所在的 catch 块
{
    Console.WriteLine("产生异常:{0}", e.Message);
}
catch (ArithmeticException ee)                     //基类所在 catch 块
{
    Console.WriteLine("产生异常:{0}", ee.Message);
}
```

同样，由于 Exception 类是所有其他异常类的基类，因此 Exception 类所在的 catch 块必须是最后面的 catch 块，它可以捕获任意类型的异常。

显然，如果不想具体区分是哪一种类型的异常，也不想利用 Exception 派生类更强大、更具针对性的处理能力，可以利用 Exception 类"笼统"地捕获所有的异常。这使代码变得简洁，保证所有的异常都能被捕获，而不会出现遗漏。

5.2.4 try-catch-finally 结构

程序在运行过程中一旦出现异常会立即转向执行相应 catch 块中的语句，执行完后接着执行 try-catch 结构后面的语句。这意味着在出现异常时程序并不是按照既定的顺序执行，而是跳转执行。为维持系统的有效性和稳定性，必须保证有相应的代码能够"弥补"被跨越代码的工作，主要是完成必要的清理工作(如关闭文件、释放内存等)。这种保证机制可以由带 finally 的 try-catch-finally 结构来实现。

try-catch-finally 结构的格式如下：

```
try
{
    //可能产生异常的代码
}
catch [(异常类 对象名)]
{
    //处理异常的代码
```

```
}
finally
{
    //完成清理工作的代码
}
```

【说明】

(1) 根据需要,可以在这种结构中带 1 个或多个 catch 块。

(2) 不管在 try 块中是否产生异常,finally 块中的代码都会被执行。也就是说,不管 catch 块是否被执行,finally 块都会被执行。哪怕是在执行 catch 块中遇到 return 语句,也会执行 finally 块中的语句。

例如,下列代码在执行时会产生一个零除异常,当产生异常时程序会转向执行 catch 块中的语句。

```
int n = 1, m = 1;
try
{
    n = 1/(n - m);
}
catch (Exception e)
{
    Console.WriteLine("产生异常:{0}", e.Message);
    return;
    Console.WriteLine("紧跟在 return 后面…");     //因有 return 语句,故该语句没被执行
}
finally
{
    Console.WriteLine("finally 块…");      //总是被执行(即使在 catch 块中遇到 return 语句)
    Console.ReadKey();                     //让程序"暂停"下来,以观察效果
}
```

因在 catch 块中遇到 return 语句,故下列语句没被执行。

```
Console.WriteLine("try - catch - finally 结构后面的部分…");
```

该代码段执行后输出的结果如图 5.4 所示。

图 5.4　检验 finally 块的作用

由这个结果可以看到,虽然 catch 块包含了一条 return 语句,且执行该 return 语句时也会立即结束当前函数的执行,但在结束之前仍然会执行 finally 块。这说明,只要程序进入 try-catch-finally 结构,就会执行 finally 块。

5.3 异常的抛出及自定义异常

一般来说,异常在被捕获后应进行相应的处理。但是有的代码是为其他代码提供调用服务的,在这种代码中可能难以决定应该对捕获的异常进行何种处理,这时最好将捕获的异常向调用代码抛出,由调用代码捕获后再进行相应的处理。抛出异常有两种方式:一种是将捕获的异常原封不动地直接抛出;另一种是先利用捕获的异常来创建新的异常(在创建过程中可以进行一些必要的处理),然后将新建的异常抛出。

除了系统提供的异常类以外,用户也可以通过继承已有的相关异常类来定义新的、满足特定需要的异常类,这就是用户自定义异常。

5.3.1 抛出异常

抛出异常有两种方式。一种是直接抛出,也称为异常重发,格式如下:

```
throw e;                                    //e 为已捕获的异常的名称
```

另一种是先利用捕获的异常来创建新的异常,然后将之抛出,格式如下:

```
throw new 异常类名([参数列表]);
```

当然,也可以写成:

```
异常类名 异常对象名 = new 异常类名([参数列表]);
throw 异常对象名;
```

【例 5.3】 抛出异常的例子。

考虑下面有关抛出异常的 ThrowException 程序。

```
using System;
namespace ThrowException
{
    class testException
    {
        int n, m;
        public void g()
        {
            n = 10;
            m = 10;
            try
            {
                n = 1/(n - m);
            }
            catch (Exception e)
            {
                throw new Exception("这是在方法 g()产生的异常:" + e.Message, e);
                //throw e;                  //异常重发
            }
```

```
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            testException te = new testException();
            try
            {
                te.g();
            }
            catch (Exception ex)
            {
                Console.WriteLine("当前捕获的异常:" + ex.Message);
                Console.WriteLine("内部的异常(原异常):" + ex.InnerException.Message);
            }
            Console.ReadKey();
        }
    }
}
```

该程序首先增加了一个类——testException 类，在该类定义的方法 g()中捕获一个零除异常，并利用该异常的 Message 属性值以及该异常本身作为参数来创建一个新的异常并将之抛出：

```
throw new Exception("这是在方法 g()产生的异常: " + e.Message, e);
```

该语句也可以写成

```
Exception ee = new Exception("这是在方法 g()产生的异常: " + e.Message, e);
throw ee;
```

这种抛出方法的优点是程序员可以利用捕获到的异常的相关信息构造满足调用者需要的新异常，同时也可以将原异常“嵌入”到新的异常中而被同时抛出。如果用下列语句对异常进行重发，则程序员不能对要抛出的异常进行任何修改(只能原样抛出)：

```
throw e;                                    //异常重发
```

而且由于抛出的是最初的异常，故其 InnerException 属性值为 null，这会导致下列语句出现错误：

```
Console.WriteLine("内部的异常(原异常): " + ex.InnerException.Message);
```

程序 ThrowException 的运行结果如图 5.5 所示。该结果可以印证上面的分析。

图 5.5　程序 ThrowException 的运行结果

5.3.2 用户自定义异常

在实际应用中,系统提供的异常类也许不能很好地满足大家的需要,这时程序员可以根据需要定义自己的异常类,但定义的异常类必须继承已有的异常类。

【例 5.4】 定义和使用用户自定义异常。

在程序 MyExceptionClass 中先定义了一个学生类——student 类,该类包含两个私有变量成员:name 和 score,分别表示学生姓名和成绩,且 name 的长度不超过 8 个字节,score 的范围为[0,100];另外还包含一个方法成员 setInfo,用于设置 name 和 score。然后自定义一个异常类 UserException,当对 name 所赋的值的长度超过 8 个字节或者对 score 所赋的值不在[0,100]范围内时都抛出此自定义异常。程序 MyExceptionClass 的代码如下:

```
using System;
namespace MyExceptionClass
{
    class UserException : Exception          //定义用户的异常类
    {
                                             //重载 Exception 类的构造函数
        public UserException() { }
        public UserException(string ms):base(ms) { }
        public UserException(string ms,Exception inner) : base(ms,inner) { }
    }
    class student                            //定义学生类
    {
        private string name;                 //姓名,长度不超过 8 个字节
        private double score;                //成绩,范围为[0,100]
        public void setInfo(string name, double score)
        {
            if (name.Length > 8)
            {
                throw (new UserException("姓名长度超过了 8 个字节!"));
            }
            if (score < 0 || score > 100)
            {
                throw (new UserException("非法的分数!"));
            }
            this.name = name;
            this.score = score;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            student s = new student();
            try
            {
                s.setInfo("张三", 958.5);
            }
            catch (Exception e)
            {
```

```
                Console.WriteLine("产生异常:{0}", e.Message);
            }
            Console.ReadKey();
        }
    }
}
```

执行该程序,在运行过程中由于试图对 score 赋值 958.5,导致下列异常被创建和抛出:

```
new UserException("非法的分数!")
```

运行结果如图 5.6 所示。

图 5.6　程序 MyExceptionClass 的运行结果

5.4　习题

一、填空题和改错题

1. 常用的异常处理的关键字包括_____、_____、_____、_____。

2. 对于下列的代码段:

```
int n, m;
int[] a = new int[5];
n = 10; m = 10;
try
{
    for (int i = 0; i <= a.Length; i++) a[i] = i;
    n = 1/(n - m);
}
catch (DivideByZeroException e1)
{
    Console.WriteLine("产生零除异常!");
}
catch (IndexOutOfRangeException e2)
{
    Console.WriteLine("产生数组访问越界异常!");
}
```

执行后,输出的结果是_____。

3. 下列代码段中试图用 try-catch-catch 结构捕获和处理异常,其中有的地方是错误的,请将错误的地方纠正过来。

```
int m;
int[] a = new int[5];
```

```
try
{
    m = int.Parse("2000$");
    for (int i = 0; i <= a.Length; i++) a[i] = i;
}
catch (Exception e1)
{
    Console.WriteLine("产生异常:{0}", e1.Message);
}
catch (IndexOutOfRangeException e2)
{
    Console.WriteLine("产生异常:{0}", e2.Message);
}
```

4. 对于下列的代码段:

```
int m,n;
n = 10; m = 10;
try
{
    n = 1/(n - m);
}
catch (Exception e)
{
    Console.WriteLine("产生零除异常!");
    return;
}
finally
{
    Console.WriteLine("在执行 finally 块中的语句…"); Console.ReadKey();
}
Console.WriteLine("在执行 try - catch - finally 结构后面的语句…");
```

执行后,输出的结果是_____。

5. 对于下面程序:

```
using System;
namespace ThrowException
{
    class testException
    {
        public void g()
        {
            try
            {
                int n = Convert.ToInt16("200$");
            }
            catch (Exception e)
            {
                throw new Exception("抛出新的异常!");
            }
```

```
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            testException te = new testException();
            try
            {
                te.g();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.InnerException.Message);
            }
            Console.ReadKey();
        }
    }
}
```

上述程序中有的地方在运行时会产生没有被捕获的异常，应该如何纠正保证程序的稳定性？为什么？

6. 对于下面定义的类 A：

```
class A
{
    public void g()
    {
        try
        {
            int n = Convert.ToInt16("200$");
        }
        catch (Exception e)
        {
        }
    }
}
```

执行下列语句时是否会出现异常？为什么？

```
A a = new A();
a.g();
```

7. 阅读下列程序，请将其运行结果写出来。

```
using System;
namespace ThrowException
{
    class myException : Exception
    {
        public myException(string ms) : base(ms) { }
        public myException(string ms, Exception e) : base(ms,e) { }
```

```
    }
    class A
    {
        public void f()
        {
            int n, m = 0;
            if (m == 0) throw new myException("在函数 f()中抛出的零除异常!");
            else n = 1/m;
        }
    }
    class B
    {
        public void g()
        {
            try
            {
                A a = new A();
                a.f();
            }
            catch (myException e)
            {
                throw new myException("在函数 g()中抛出的异常!",e);
            }
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            B b = new B();
            try
            {
                b.g();
            }
            catch (myException e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine(e.InnerException.Message);
            }
        }
    }
}
```

运行后输出的结果是：_____。

二、上机题

1. 编写一个能够进行加、减、乘、除的计算器程序(窗体应用程序),并能够处理可能产生的异常。

2. 编写一个控制台应用程序,将一组数据写到一个文本文件中,并保证即使在写数据的过程中产生异常而退出程序时也能将已打开的文本文件关闭。