第5章 chapter 5

异常处理与程序发布


课程练习

异常处理是提高程序可靠性的重要手段。当程序调试成功生成解决方案，就可以将.NET Framework、源代码以及项目所需的其他资源等打包，生成安装包，进行程序的发布。

本章主要内容如下。

(1) .NET 异常处理机制和.NET 类库中的主要异常类。

(2) 结构化异常处理语句及自定义异常处理方法。

(3) 在 Visual Studio 2022 集成环境下，使用调试工具进行程序调试的方法。

5.1 错误、异常与调试的概念

程序设计是一个不断完善的过程，在这个过程中不可避免地产生一些可预知和不可预知的错误。程序错误按照发生机理可分为语法错误、语义错误和逻辑错误。表 5-1 对这 3 种错误进行了归纳。

表 5-1 C#错误类型

错误类型	描　　述	示　　例
语法错误	有两种情形：一种，在 IDE 环境中编写代码时，输入错误语法，此时能够被 IDE 检查出来，在错误地方以红色波浪线标出；另一种，C#语言编译器在编译时将错误信息显示在“错误列表”窗口	(1) 语句结束时少了分号 (2) int x=12.3; (3) 变量未定义 (4) 数组越界 (5) 空值错误
语义错误	在编译阶段不能被发现，而在程序运行阶段抛出异常	(1) 除数为 0 (2) 加载图片时文件不存在
逻辑错误	程序未达到期望结果	

程序中存在的这些错误会引发异常。异常就是程序运行期间发生的错误及其他的意外行为。在应用程序中发现并排除错误的过程被称为调试。调试是程序设计的一个重要组成部分，在软件界有一句口头语，就是“程序其实都是调试出来的!”。调试是帮助程序设计人员查找和排除代码错误的有效手段。

5.2 异常处理

5.2.1 异常类

C#语言采用面向对象的方法来处理异常，其异常处理机制可以简单地描述为以下几个步骤。

（1）C#程序在执行过程中一旦出现异常，会自动产生一个异常类对象。该异常对象被提交给C#运行时系统（即第1章介绍的JIT），这个过程被称为抛出异常。此外，在C#中，抛出异常也可以用throw语句强制产生。

（2）C#运行时系统接收到异常对象后，会寻找能处理该异常的方法并把当前异常对象交给其处理，这一过程被称为捕获异常。

（3）当C#运行时系统找不到可以捕获异常的方法时，运行时系统将终止，相应的C#程序也将退出。

System.Exception类是所有异常类的基类。Exception类有两个派生子类：SystemException和ApplicationException。SystemException类是系统定义的各种异常，而ApplicationException类则是用户定义的各种异常的基类。Exception类有一个经常用到的Message属性，该属性为字符串类型，提供了错误描述的文本。表5-2给出了系统已经定义的一些常用异常类。

表5-2 系统定义的常用异常类

异常类	描述
Exception	所有异常对象的基类
SystemException	系统定义的各种异常的基类
ApplicationException	用户定义的异常的基类
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运行时引发
DivideByZeroException	被零除时引发的异常
FileNotFoundExeption	文件不存在引发的异常
OverflowException	算术操作溢出引发的异常
IOException	发生I/O错误时引发的异常
InvalidOperationException	当对象处于无效状态时，由方法引发
ArgumentException	所有参数异常的基类
ArgumentNullException	在不允许参数为空的情况下，由方法引发
ArgumentOutOfRangeException	当参数不在一个给定范围之内时，由方法引发

5.2.2 异常处理语句

1. try-catch 语句结构

当一个异常被抛出时，应该有专门的语句来接收和处理被抛出的异常对象。在 C# 中，异常对象是依靠以 catch 语句为标志的异常处理语句来捕获和处理的。其格式如下：

```
try
{
    …//可能产生异常的代码
}
catch(异常类  异常对象标识符)
{
    …//异常处理代码
}
[其他 catch]
finally
{
    …//无论是否产生异常总要执行代码
}
```

2. try-catch 语句执行流程

图 5-1 描述了异常处理语句流程。

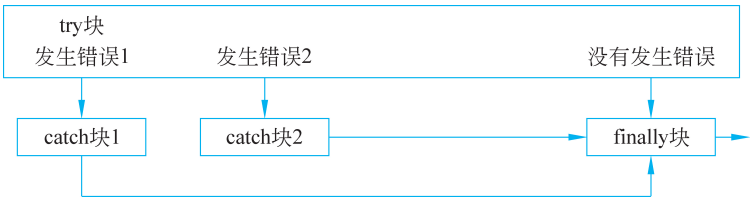

图 5-1 异常处理语句流程

(1) 将可能抛出异常的程序代码放在 try 块中，把异常处理代码放在 catch 块中，把无论是否产生异常总需要执行的代码放在 finally 块中。

(2) 在上述格式中，try 块不能省略，catch 块和 finally 块可以省略，但不能同时省略。

(3) catch 块可以不止一个，每一个 catch 块对应一种异常处理代码。

(4) 上述异常处理语句执行流程为先执行 try 块，如果发生异常则转入对应的 catch 块中，如果没有产生异常或没有对应的 catch 块，则执行 finally 块。

【实例 5-1】 异常处理实例。输入两个数，输出其相除的结果。源代码如表 5-3 所示。

表 5-3　实例 5-1 源代码

行号	源　代　码
01	using System;
02	namespace 实例 5_1{
03	class ConsoleApp{
04	static void Main(string[] args) {
05	try
06	{
07	Console.Write("请输入第 1 个数：");
08	double x=double.Parse(Console.ReadLine()); //数据类型转换
09	Console.Write("请输入第 2 个数:");
10	double y=double.Parse(Console.ReadLine());
11	Console.WriteLine("这两个数的商是：{0}", x / y);
12	}
13	catch (FormatException) //捕捉数据类型转换异常
14	{
15	Console.WriteLine("必须输入数字");
16	}
17	catch (DivideByZeroException) //捕获除数为零异常
18	{
19	Console.WriteLine("第 2 个数不能为零");
20	}
21	catch (Exception e) //最后捕捉其他未知类型异常
22	{
23	Console.WriteLine("其他错误：{0}", e.Message);
24	}
25	finally
26	{
27	Console.WriteLine("按任意键退出");
28	Console.ReadLine();
29	}
30	}
31	}
32	}

5.2.3　自定义异常

系统定义的异常主要用来处理系统可以预见的运行错误。对于某个应用程序特有的运行错误，则需要程序设计人员自行创建用户自定义的异常类。自定义异常的创建可以概括为以下步骤。

（1）声明一新的异常类。该类以 ApplicationException 类或其他已经存在的异常类（包括用户异常类）为父类。

（2）为新的异常类定义属性和方法，或重载父类的属性和方法，使这些属性和方法能够体现该类所对应的错误信息。

（3）对于用户自定义的异常，是不可能依靠系统自动抛出的，需要在程序中使用

throw 关键字将其抛出。

下面的实例 5-2 演示了自定义异常处理。该实例只是在实例 5-1 的基础上增加了一个自定义异常类 outofBoundException，该类的父类为 ApplicationException。该类同时声明了一个带参数的构造函数。在 ConsoleApp 类中，通过 throw 关键字，抛出自定义异常。

【实例 5-2】 自定义异常处理实例。源代码如表 5-4 所示。

表 5-4 实例 5-2 源代码

行号	源 代 码
01	using System;
02	namespace 实例 5_2{
03	class outofBoundException: ApplicationException //声明新的异常类
04	{
05	public outofBoundException(string msg)
06	: base(msg) //声明带参数的构造函数，并向基类传递参数
07	{}
08	}
09	class ConsoleApp {
10	static void Main(string[] args) {
11	try {
12	Console.Write("请输入第 1 个数: ");
13	double x=double.Parse(Console.ReadLine());
14	Console.Write("请输入第 2 个数");
15	double y=double.Parse(Console.ReadLine());
16	if (x<0 || y<0)
17	{
18	//抛出异常，调用自定义异常类的有参构造函数
19	throw new outofBoundException("不允许为负数!");
20	}
21	Console.WriteLine("这两个数的商是: {0}", x / y);
22	}
23	catch (FormatException) //捕捉数据类型转换异常
24	{
25	Console.WriteLine("必须输入数字");
26	}
27	catch (DivideByZeroException) //捕获除数为零异常
28	{
29	Console.WriteLine("第 2 个数不能为零");
30	}
31	catch (outofBoundException e)
32	{
33	Console.WriteLine(e.Message);
34	}
35	catch (Exception e) //最后捕获其他类型的异常
36	{
37	Console.WriteLine(e.Message);
38	}
39	finally

续表

行号	源 代 码

40	{
	Console.WriteLine("按任意键退出…");
	Console.ReadLine();
	}

41	
42	
43	
44	}
45	}
46	}

5.3 程序调试


5.3

在应用程序中发现并排除错误的过程叫作调试。Visual Studio 2022 集成开发环境提供了丰富的调试手段，可以方便地跟踪程序的运行，解决程序错误，并进行适当的错误处理。它提供了几种调试工具来帮助分析应用程序的执行过程，这些调试工具对于发现错误来源很有用，也可以使用这些工具来检验应用程序的改变，或者了解应用程序的工作过程。

5.3.1 控制应用程序的执行过程

在调试应用程序的过程中，可以充分控制应用程序的执行过程，包括以不同方式启动调试过程、中断应用程序的执行、步进执行程序、运行到指定位置以及终止应用程序的执行等。

开始调试程序时，可以通过“调试”菜单选择如何启动应用程序，主要包括以下几种方式。

（1）开始执行：在这种方式下，应用程序开始执行并一直执行下去直到遇到断点或者程序结束。一般设置了断点时才使用这种方式启动应用程序，否则程序一直处于执行过程，用户无法对其进行调试。

（2）逐语句：在逐语句方式下，应用程序开始执行第1条语句然后中断。当有函数调用时，执行过程会进入被调用函数的内部。

（3）逐过程：这种方式和逐语句类似，但是它不会进入被调用程序的内部，而是把函数调用当作一条语句执行。

在调试过程中，可以随时使用“调试”菜单中的“停止调试”命令终止调试过程。此外，还可以在调试程序运行的过程中通过“调试”菜单中的“全部中断”命令随时中断程序的执行而进入中断状态，因为调试器的许多功能只有在中断状态下才能使用。处于中断状态的程序，可以随时通过“继续”命令恢复执行状态。

5.3.2 附加到进程

Visual Studio 2022 调试器能够附加在集成开发环境外部运行的进程上，可以使用这

种功能来调试正在运行的应用程序、同时调试多个程序、调试远程机器上的程序以及在应用程序崩溃时自动启动调试器。一旦附加到进程上，就可以使用调试器提供的各种功能控制程序的运行和查看进程的状态。

要附加到进程，执行以下步骤。

(1) 首先由“调试”→“附加到进程”菜单项打开“附加到进程”对话框，如图 5-2 所示。

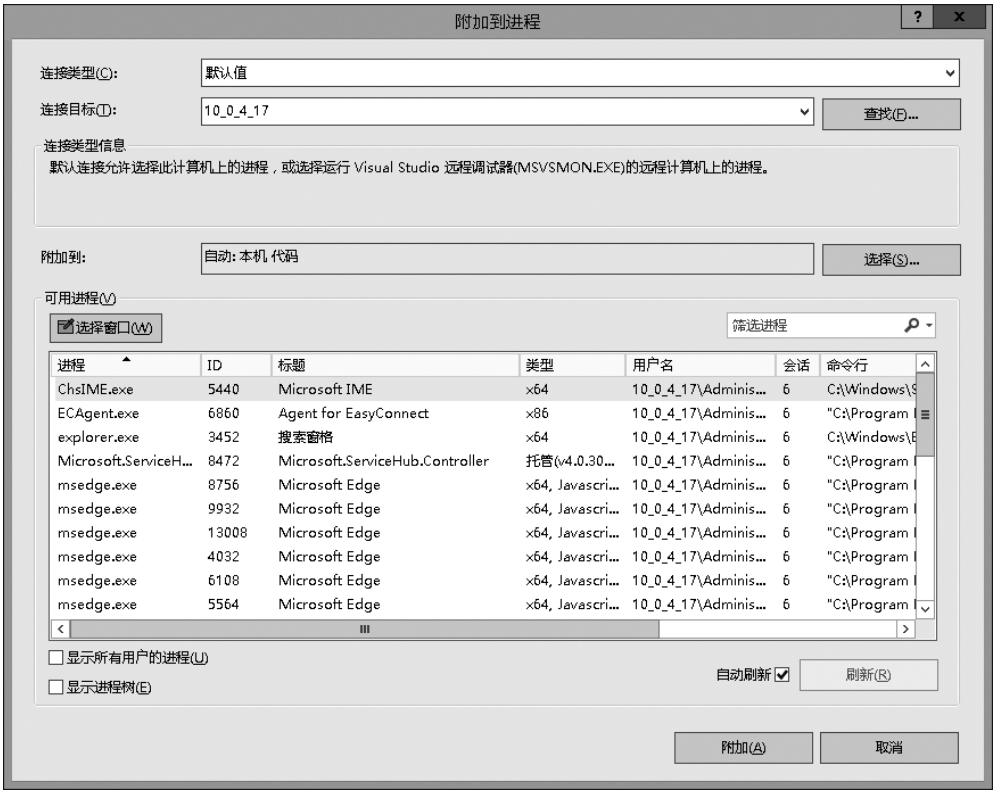

图 5-2 “附加到进程”对话框

(2) 在该对话框的“可用进程”列表框中选择想要附加的进程(如果想要附加的进程位于远程机器上，则需要通过该对话框上面的“连接类型”和“连接目标”下拉列表框选择一个远程计算机，单击“刷新”按钮可以刷新“可用进程”列表)，选择好后单击“附加”按钮即可完成选择。

5.3.3 断点

在调试应用程序时要经常使用断点来中断程序的执行使之进入中断状态，进而使用调试器提供的强大功能来查看变量的值、寄存器和内存的使用情况及函数调用情况等应用程序状态信息。每当调试器遇到一个断点时，会中断程序的执行而转入中断模式。Visual Studio 2022 调试器支持 4 种类型的断点。

(1) 函数断点：这种类型的断点标识的位置是特定函数中的偏移位置。

（2）文件断点：这种类型的断点标识的位置是特定文件中的偏移地址。

（3）地址断点：这种类型的断点标识的位置是内存地址。

（4）数据断点：这种类型的断点标识某个变量并且每当它所标识的变量发生变化时就会中断程序的执行。

1. 插入断点与取消断点

可以使用两种方式插入新的断点：使用菜单命令和使用指示器边距。最简单的方式，就是单击打算插入断点的代码行左边的指示器边距，此时指示器边距内会出现中断图标。注意，使用指示器边距插入的断点是文件类型的断点，而且只能插入这种类型的断点。

当使用菜单方式时，首先在“调试”→“新建断点”菜单项下选择“函数断点”或“数据断点”。图 5-3 为“新建函数断点”对话框。然后，输入或选择所需的参数类型并输入必要的内容。最后，单击“确定”按钮，完成新断点的插入。

图 5-3 “新建函数断点”对话框

在代码编辑器内只能看到源代码中的函数断点和文件断点，在指示器边距内会显示这两种断点的图标。要想查看所有类型的断点和它们的详细信息，需要使用“断点”窗口。可以通过“调试”→“窗口”→“断点”菜单命令打开断点窗口。可以使用该窗口中的工具栏执行插入新断点、删除已有断点、查看断点属性以及指定该窗口中显示的列等操作。

可以使用“调试”菜单或快捷菜单中的“禁用断点”命令来禁用或启用断点。当不再需要断点时，可以使用“调试”菜单中的“清除所有断点”命令来删除文件中的所有断点。

2. 设置断点的属性

默认情况下，调试器遇到断点时总会中断程序的执行。但是，也可以设置断点的属性来改变这种默认行为，指定在满足一定的条件时才发生中断。在断点上右击，弹出如图 5-4 所示的快捷菜单，选择其中的菜单命令设置断点的属性。

图 5-4 断点选项菜单

(1) 条件：中断条件是一个表达式，每次到达该断点时都会计算该表达式的值，而计算的结果决定了该次到达断点是否是一次有效的点击。如果点击有效并且满足点击次数属性，则调试器就会中断程序的执行。

(2) 操作：打开断点设置对话框，可设置筛选器、条件表达式、命中次数、显示信息、命中后是否移除断点等。

(3) 命中次数：无论选择条件或操作均可以设置命中次数。所谓命中次数，对于位置断点(包括函数断点、文件断点和地址断点)来说就是执行到指定位置的次数；而对于数据断点来说，则是变量的值发生改变的次数。这个属性决定了在中断执行前要发生多少次点击。

5.3.4 查看程序的状态

Visual Studio 2022 调试器提供了许多工具可以在中断模式下查看应用程序的状态。当程序处于中断模式时，把鼠标移到当前执行范围内某个变量上会以工具提示的方式显示该变量的值。除此之外，还可以打开“调试”→“窗口”菜单，使用以下工具。

(1) 局部变量窗口：这个窗口显示当前上下文中的局部变量。默认的上下文就是包含当前执行位置的函数，因此，局部变量窗口中显示的就是这个函数内的局部变量。但是可以通过调试位置工具栏来改变“局部变量”窗口显示的上下文。

(2) 自动窗口：该窗口中显示当前语句和前一条语句中的变量(所谓当前语句就是当前执行位置上的语句)。结构或数组类型的变量在该窗口中以树的形式显示。

(3) 监视窗口：Visual Studio 2022 提供了 4 个监视窗口，可以在监视窗口中计算变量和表达式的值，并可随着程序的执行观察它们的变化。

(4) 内存窗口：可以使用内存窗口查看内存中的数据，通常这些数据量很大，不适合在监视窗口查看。可以滚动显示内存窗口中的内容并可以同时打开 4 个内存窗口。另外，还可以指定内存窗口显示的起始地址。

(5) 反汇编窗口：该窗口显示了被调试程序的反汇编代码。当调试没有源代码的程序时，就只能查看它的反汇编代码。

(6) 寄存器窗口：在寄存器窗口中动态显示寄存器的内容，新改变的寄存器使用红色显示。还可以在寄存器窗口中改变寄存器的值。

(7) 调用堆栈窗口：可以使用调用堆栈窗口查看当前堆栈中的函数调用情况。该窗口显示每个函数的名字和编写它们所采用的语言。可以直接从调用堆栈窗口中跳转到函数的源代码中，同时还可以查看函数的反汇编代码。

本章小结

异常处理是提高程序可靠性的一个重要手段。C#语言中内置了结构化的异常处理语句，包括 try-catch 语句、try-catch-finally 语句、try-finally 语句和 throw 语句，这样就能够在程序中以统一的方式来处理各类不同的异常。另外，.NET 类库定义了丰富的异

常类，开发人员还可以从中派生出自己的异常类型。这二者共同组成了C#程序的异常处理模型，能够对异常进行实时、高效、层次化的管理。

此外，Visual Studio 2022集成环境也提供了丰富的调试工具处理程序运行时的错误。利用单元测试可以在与系统其他部分隔离的情况下进行测试。

习　题

简答题

（1）C#程序的错误类型有哪些？

（2）简述.NET异常处理机制。

（3）在异常处理结构中只使用一个catch语句时，能否捕获所有的异常？又能否判断出各种异常的类型？

（4）Visual Studio 2022调试器支持哪几种类型的断点？