**异 常**

**C#为我们提供了一套完美的方案，让我们在程序主线之外处理异常，不但使程序更加健壮、更加容错，而且保持了程序结构的清晰。**

**12.2 捕获异常——try-catch结构**

**常见异常：**  
 SystemException 其他用户可处理的异常的基本类   
 ArgumentException 方法的参数是非法的   
 ArgumentNullException 一个空参数传递给方法，该方法不能接受该参数   
 ArgumentOutOfRangeException 参数值超出范围   
 ArithmeticException 出现算术上溢或者下溢   
 ArrayTypeMismatchException 试图在数组中存储错误类型的对象   
 BadImageFormatException 图形的格式错误   
 DivideByZeroException 除零异常   
 DllNotFoundException 找不到引用的DLL   
 FormatException 参数格式错误   
 IndexOutOfRangeException 数组索引超出范围   
 InvalidCastException 使用无效的类   
 InvalidOperationException 方法的调用时间错误   
 MethodAccessException 试图访问思友或者受保护的方法   
 MissingMemberException 访问一个无效版本的DLL   
 NotFiniteNumberException 对象不是一个有效的成员   
 NotSupportedException 调用的方法在类中没有实现   
 NullReferenceException 试图使用一个未分配的引用   
 OutOfMemoryException 内存空间不够   
 PlatformNotSupportedException 平台不支持某个特定属性时抛出该错误   
 StackOverflowException 堆栈溢出

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运行时引发。  
 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。

ArgumentException：所有参数异常的基类。  
 ArgumentNullException：在参数为空（不允许）的情况下，由方法引发。  
 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。  
 SqlException：封装了SQL操作异常。

static void DiviseZero()

{

int d = *Convert*.*ToInt32*(*Console*.*ReadLine*());

*Console*.*WriteLine*("100 / {0} = {1}", d, 100 / d);

*Console*.*ReadKey*();

}

static void TryException()

{

try

{

int d = *Convert*.*ToInt32*(*Console*.*ReadLine*());

*Console*.*WriteLine*("100 / {0} = {1}", d, 100 / d);

}

catch(*DivideByZeroException*)

{

*Console*.*WriteLine*("除数为0异常！");

}

catch (*FormatException*)

{

*Console*.*WriteLine*("输入串无法转换为数字！");

}

*Console*.*ReadKey*();

}

**12.3 收尾工作——try-catch-finally结构**

在某些问题中，不论是否出现异常，都要进行一些收尾工作，这些收尾工作常放在

finally 块中。

程序经常动态申请资源，比如打开某个文件，并从中读写数据。操作系统通常不允许

多个程序同时读写同一个文件，所以读写完文件后，应及时关闭文件（即释放资源），以

便其它程序使用。如果不关闭文件，就会发生资源泄露。这些释放资源的代码通常放在

finally 块，不管是否发生异常，资源均被释放。

**12.4 抛出异常——throw 语句**

static void ThrowExtheption()

{

int d = 0;

try

{

d = *Convert*.*ToInt32*(*Console*.*ReadLine*());

if (d >= 10 || d <= 0)

{

throw new *ArgumentOutOfRangeException*();

}

}

catch (*ArgumentOutOfRangeException*)

{

*Console*.*WriteLine*("参数超出范围");

}

finally

{

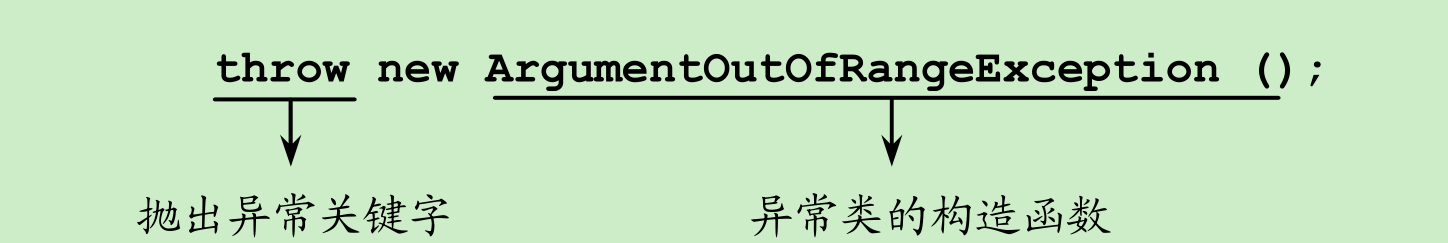
*Console*.*WriteLine*("100 / {0} = {1}", d, 100 / d);

}

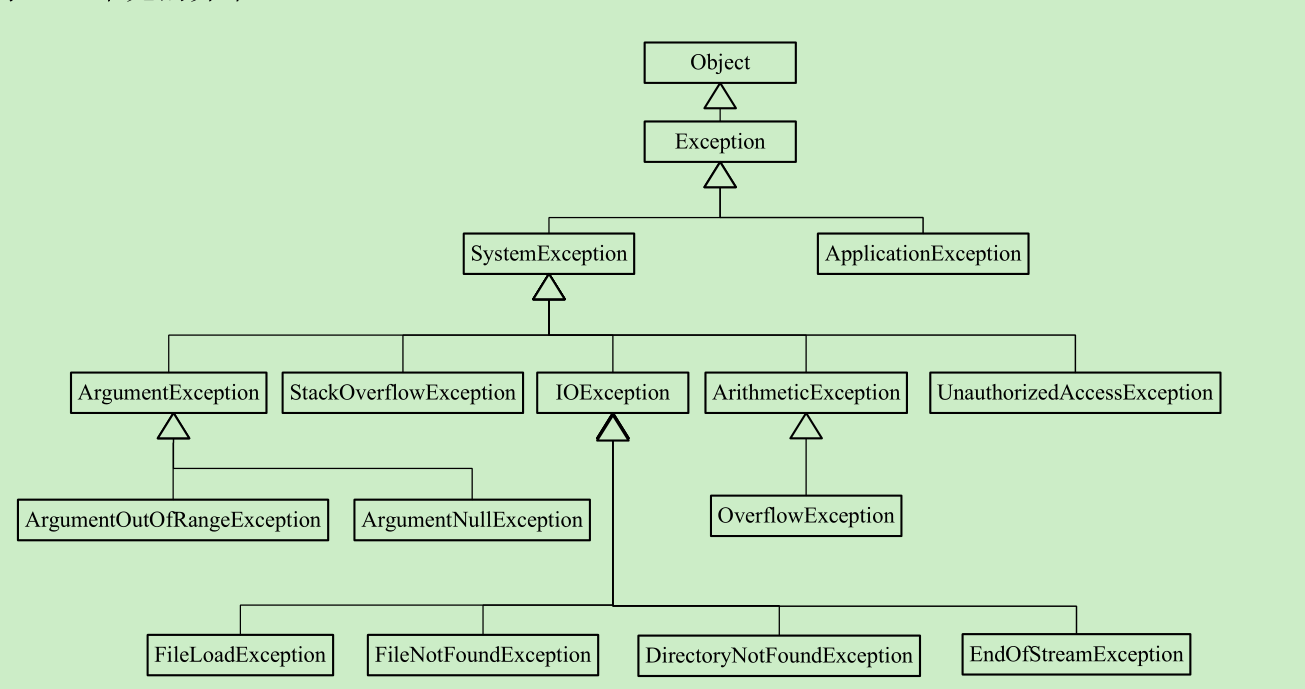
*Console*.*ReadKey*();

}

在 C#中使用 thow 语句抛出异常，其一般格式为：



**12.5 .N ET 中的异常类**



由于派生类对象属于基类，所以当有多个 catch 块时，要按从具体到一般的顺序由上

往下排列，基类必须放在最后，否则捕捉派生类异常的 catch 块永远没机会被执行。

Exception 类有两个非常重要的派生类——SystemException 和 ApplicationException。.NET 中预定义的异常类都派生于 SystemException 类，而用户自定义的异常类都应派生于ApplicationException 类。

**12.6 异常类的属性**

Exception 类和其他一般类一样，有几个公有属性，通过这些属性可以非常方便的了解

异常信息。其中比较重要的两个属性是 Message 和 StackTrace。属性 Message 用于描述异常的原因，属性 StackTrace 用于描述异常的堆栈信息，即发生异常位置。

static void ExceptionProperty()

{

int d = 0;

try

{

d = *Convert*.*ToInt32*(*Console*.*ReadLine*());

if (d >= 10 || d <= 0)

{

throw new *ArgumentOutOfRangeException*();

}

}

catch (*Exception* a)

{

*Console*.*WriteLine*("message:{0}",a.*Message*);

*Console*.*WriteLine*("stacktrace:{0}", a.*StackTrace*);

}

finally

{

*Console*.*WriteLine*("100 / {0} = {1}", d, 100 / d);

}

*Console*.*ReadKey*();

}

**12.7 自定义异常**

必要时我们也可以针对程序中的问题创建新的异常类。用户定义的异常类都直接或间接继承于 ApplicationException 类。

我们知道负数不能开平方，下面我们定义一个名为NegativeNumberExeption的异常类。

class NegativeException:*ApplicationException*

{

public NegativeException() : base("对负数进行非法操作") { }

public NegativeException(string message) : base(message) { }

}

static void SelfDefine()

{

int d = 0;

try

{

d = *Convert*.*ToInt32*(*Console*.*ReadLine*());

if (d < 0)

{

throw new NegativeException("非法负数：" + d);

}

}

catch (NegativeException n)

{

*Console*.*WriteLine*("message:{0}", n.*Message*);

*Console*.*WriteLine*("stack:{0}", n.*StackTrace*);

}

finally

{

*Console*.*ReadKey*();

}

}

运行结果:

