

Xử lý ngoại lệ



Nội dung

- Tình huống phát sinh ngoại lệ
- Cách xử lý ngoại lệ làm việc trong C#
- Câu lệnh try-catch-finally



Tình huống

- Những lỗi phát sinh khi runtime có thể làm hư hại chương trình
- Có thể không phải lỗi lập trình
- VD: một số tình huống viết dữ liệu vào file
 - Đĩa bị đầy
 - Lỗi phần cứng
 - File bị thay đổi thành chỉ đọc
 - Không thể truy cập, truy vấn CSDL
 - ...



Cách xử lý trước đây

- Hầu hết các bước có khả năng thất bại
- Khó xác định chính xác lỗi từ những thông tin trả về của hàm thư viện
- Đoạn code phải thực hiện rồi mới biết lỗi!

GET A FILENAME

OPEN THE FILE

IF THERE IS NO ERROR OPENING THE FILE

 READ SOME DATA

 IF THERE IS NO ERROR READING THE DATA

 PROCESS THE DATA

 WRITE THE DATA

 IF THERE IS NO ERROR WRITING THE DATA

 CLOSE THE FILE

 IF THERE IS NO ERROR CLOSING FILE

 RETURN



Cách xử lý Exception

- Đoạn code xử lý ngắn gọn, dễ đọc
- Logic chương trình hợp lý hơn, những đoạn nghi ngờ có lỗi nằm trong vùng Try
- Cho phép xử lý tình huống lỗi rõ ràng và đơn giản!

TRY TO DO THESE THINGS:

GET A FILENAME

OPEN THE FILE

READ SOME DATA

PROCESS THE DATA

WRITE THE DATA

CLOSE THE FILE

RETURN

IF ERROR OPENING THE FILE THEN ...

IF ERROR READING THE DATA THEN ...

IF ERROR WRITING THE DATA THEN ...

IF ERROR CLOSING THE FILE THEN ...

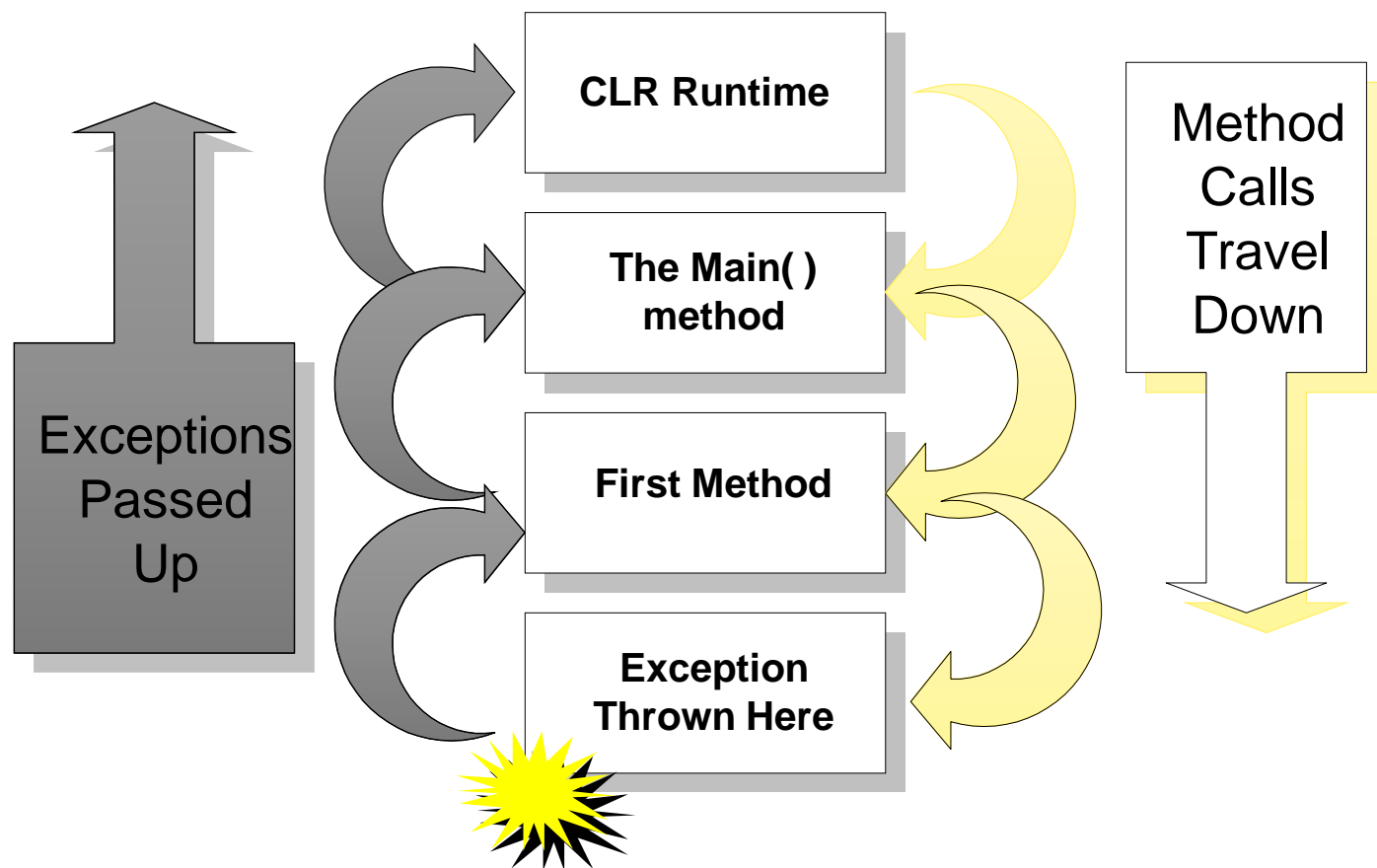


Các đối tượng Exception

- Trong C#, khi runtime error xuất hiện
 - CLR sẽ xác định lỗi và phát sinh ra đối tượng Exception
 - Đối tượng Exception này được ném trở lại stack chờ cho một phương thức bắt lỗi đó.
 - Nếu Exception này không được chương trình “catch” thì CLR sẽ in ra thông điệp lỗi



Mô hình gọi – xử lý Exception





Sử dụng try-catch

- Dùng try-catch để xử lý ngoại lệ
 - Đặt code có khả năng dẫn đến ngoại lệ vào khối “try”
 - Cung cấp các khối “catch” theo sau “try”
 - Có thể cung cấp tất cả catch cho các lỗi nếu muốn xử lý, bằng cách sử dụng các lớp exception thích hợp
 - Nếu không cung cấp “catch” cho một ngoại lệ, thì exception này được lan truyền lên trên.



Cú pháp try-catch

```
try
```

```
{
```

```
    RiskyBussiness();
```

```
}
```

```
catch (SomeException e )
```

```
{
```

```
    // Handle code
```

```
}
```

Code có khả năng
dẫn đến lỗi

Tham số exception
được catch

Đoạn xử lý với tình
huống có lỗi



Khối try

- Bao gồm các phần
 - Từ khóa try
 - Theo sau khối “{...}”
 - Khối “{...}” bắt buộc phải có, khác với “{...}” trong if hay for
- Bên trong khối try
 - Đặt bất cứ câu lệnh nào có khả năng phát sinh ra ngoại lệ



Khối catch

- Đặt một hay nhiều ngay sau khối try
 - Không có lệnh nào chen giữa hai khối catch của một khối try
- Cú pháp khối catch như sau

```
catch (Exception-class [var1])  
{  
    // xử lý ngoại lệ 1  
}  
catch (Exception-class [var2])  
{  
    // xử lý ngoại lệ 2  
}
```



Ví dụ try-catch

```
namespace NguyenHaGiang
{
    class Program
    {
        static void Main(string[] args)
        {
            int x=0;
            int div = 100 / x;
            Console.Write(div);
        }
    }
}
```



Chương trình bị terminate

```
namespace NguyenHaGiang
{
    class Program
    {
        static void Main(string[] args)
        {
            int x=0;
            int div=0;
            try
            {
                div = 100 / x;
                Console.Write("This line is not executed!");
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Exception occurred");
            }
            Console.WriteLine("Result is {0}",div);
        }
    }
}
```



**Chương trình kết thúc
bình thường**



Sử dụng Finally

- Khi một exception được ném ra
 - Luồng thực thi sẽ nhảy vào khối catch xử lý nó.
 - Một số đoạn code giải phóng tài nguyên có thể bị bỏ qua

Open File

Read Data // ngoại lệ được phát sinh

Close File // đoạn code này bị bỏ qua, dù file chưa đóng

- Khối try-catch có phần option là **finally**
 - Luôn luôn được gọi
 - Sử dụng để dọn dẹp các tài nguyên đang nắm giữ



VD có try-catch-finally

```
namespace NguyenHaGiang
{
    class Program
    {
        static void Main(string[] args)
        {
            int x=0;
            int div=0;
            try
            {
                div = 100 / x;
                Console.WriteLine("This line is not executed!");
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Exception occurred");
            }
            finally // always executed
            {
                Console.WriteLine("Finally block!");
            }
            Console.WriteLine("Result is {0}",div);
        }
    }
}
```

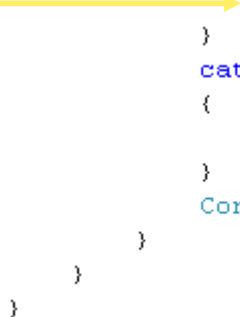
Luôn thực thi dù có hay không có ngoại lệ!



Lệnh throw

- Cho phép ném ra một ngoại lệ
 - Cú pháp: **throw** exception-object

```
namespace NguyenHaGiang
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                throw new DivideByZeroException("Invalid Division");
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Exception!");
            }
            Console.WriteLine("Last statement");
        }
    }
}
```



Phát sinh ra ngoại lệ



Lớp Exception

- Có 2 loại ngoại lệ
 - Ngoại lệ phát sinh bởi chương trình
 - Ngoại lệ được tạo bởi CLR
- Lớp System.Exception là lớp cơ sở cho tất cả lớp trong C#
- 2 lớp kế thừa từ lớp này:
 - ApplicationException: thường làm lớp cơ bản cho lớp ngoại lệ phát sinh từ ứng dụng
 - SystemException: do CLR phát sinh



Lớp Exception (2)

Một số lớp Exception thường dùng

- System.OutOfMemoryException
- System.NullReferenceException
- System.InvalidCastException
- System.ArrayTypeMismatchException
- System.IndexOutOfRangeException
- System.ArithmeticException
- System.DivideByZeroException
- System.OverflowException



Tự tạo lớp exception

```
namespace NguyenHaGiang
{
    class MyException : Exception
    {
        public MyException(string str)
        {
            Console.WriteLine("User defined exception");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                throw new MyException("Giang");
            }
            catch (MyException e)
            {
                Console.WriteLine("Exception caught here:"+e.ToString());
            }
            Console.WriteLine("Last statement");
            Console.ReadLine();
        }
    }
}
```



VD truy xuất file

```
namespace NguyenHaGiang
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream outStream = null;
            FileStream inStream = null;
            try {
                outStream = File.OpenWrite("hutech1.txt");
                inStream = File.OpenRead("hutech2.txt");
            }
            catch (Exception ex) {
                Console.WriteLine(ex.ToString());
            }
            finally {
                if (outStream != null) {
                    outStream.Close();
                    Console.WriteLine("outStream closed.");
                }
                if (inStream != null) {
                    inStream.Close();
                    Console.WriteLine("inStream closed.");
                }
            }
        }
    }
}
```



Tóm tắt

- Exception làm cho chương trình chặt chẽ hơn. Tránh terminate chương trình đột ngột vì những lỗi runtime
- Tập các lớp Exception đa dạng, bao hàm các vấn đề phát sinh lúc runtime
- Lớp Exception là lớp cơ sở cho các lớp xử lý ngoại lệ trong .NET
- Dễ dàng định nghĩa lớp ngoại lệ, và phát sinh ngoại lệ trong chương trình