

# Ngoại lệ (Exception) và xử lý ngoại lệ trong C#

Hướng dẫn tự học lập trình C# toàn tập > Ngoại lệ (Exception) và xử lý ngoại lệ trong C#

*Ngoại lệ* (exception) trong C# là những tình huống mà chương trình không thể thực hiện được lệnh theo yêu cầu.

Ví dụ, khi thực hiện phép chia, nếu mẫu số vô tình nhận giá trị 0, phép chia không thể thực hiện được. Khi người dùng yêu cầu truy xuất một file nhưng lại cung cấp sai đường dẫn khiến không thể thực hiện thao tác truy xuất. Khi gặp những tình huống này, chương trình không biết phải làm gì tiếp theo.

Trong lập trình, những tình huống tương tự xảy ra rất nhiều và được gọi chung là ngoại lệ (exception). C# cung cấp giải pháp cho những tình huống tương tự, gọi là phát ra exception và xử lý exception.

## NỘI DUNG CỦA BÀI [ Ấn ]

1. Exception (ngoại lệ) trong C#
  - 1.1. Khái niệm exception trong C#
  - 1.2. Cơ chế xử lý exception trong C#
2. Kỹ thuật xử lý ngoại lệ (Exception Handling) trong C#
  - 2.1. Exception và chế độ Debug trong C#
  - 2.2. Xử lý ngoại lệ ở chế độ Release
  - 2.3. Cấu trúc try – catch
3. Kết luận

## Exception (ngoại lệ) trong C#

### Khái niệm exception trong C#

*Ngoại lệ* (exception) là những tình huống mà chương trình không thể thực hiện được lệnh theo yêu cầu.

C# (và các ngôn ngữ lập trình khác) cung cấp các công cụ đặc biệt để sử dụng trong những tình huống tương tự, bao gồm: *thông báo ngoại lệ*, *bắt và xử lý ngoại lệ*.

Lệnh biểu diễn với từ khóa *throw* ở trên là *thông báo ngoại lệ* (hay còn gọi là thông báo lỗi).

Trong C# (và .NET framework), cách thức đơn giản nhất để phát ra thông báo ngoại lệ là sử dụng lớp Exception với từ khóa *throw* theo cấu trúc:

1. **throw new** Exception("thông tin về lỗi");

Cấu trúc này khởi tạo một object của lớp Exception và gửi object này cho cơ chế thông báo lỗi của .NET framework. Khi gọi lệnh *throw*, luồng điều khiển của chương trình sẽ thay đổi.

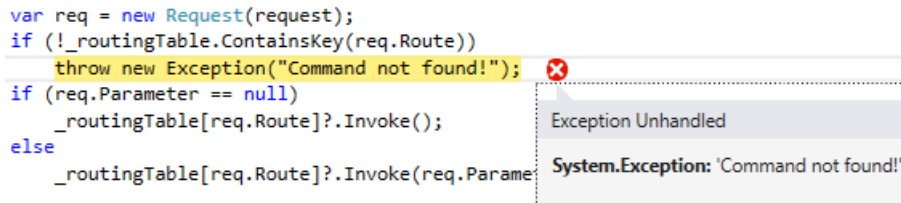
Exception là lớp mô tả ngoại lệ cơ bản nhất trong .NET. Khi học đến phần kế thừa chúng ta có thêm khả năng để tạo ra các lớp thông báo ngoại lệ riêng của mình. Các lớp thông báo

ngoại lệ do người dùng định nghĩa có khả năng đóng gói thêm nhiều thông tin khác giúp ích cho quá trình dò lỗi.

## Cơ chế xử lý exception trong C#

Khi một ngoại lệ được phát ra ở một vị trí bất kỳ trong chương trình, việc thực thi của chương trình sẽ dừng lại. Nếu chương trình đang chạy ở chế độ Debug, trình soạn thảo code sẽ được mở ra và đoạn code bị lỗi sẽ được đánh dấu giúp cho người lập trình xác định vị trí và nguyên nhân gây lỗi.

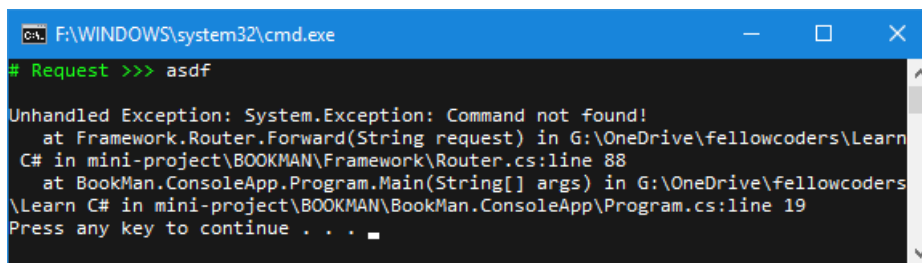
Hình dưới đây minh họa tình huống lỗi khi người dùng nhập vào một lệnh chưa tồn tại.



Giao diện Visual Studio khi xảy ra ngoại lệ

Nếu chương trình chạy ở chế độ Release, chương trình sẽ bị dừng lại và cơ chế xử lý ngoại lệ mặc định của .NET framework sẽ được kích hoạt để hiển thị lỗi. Chương trình được dịch ở chế độ này sẽ không chạy được ở chế độ Debug nữa.

Nếu chương trình console chạy ở chế độ Release mà gặp lỗi, thông báo lỗi sẽ được hiển thị như dưới đây.



Thông báo ngoại lệ ở giao diện console

Đây là cơ chế bắt và xử lý lỗi mặc định của .NET framework đối với ứng dụng console. Đối với ứng dụng windows form, giao diện bắt và xử lý lỗi có khác biệt.

Tuy nhiên, cơ chế thông báo lỗi mặc định của .NET framework tương đối không thân thiện với người dùng.

.NET cung cấp cho các chương trình tính năng *bắt và xử lý ngoại lệ* để tự mình xác định xem khi xảy ra lỗi (ngoại lệ) thì sẽ làm gì.

## Kỹ thuật xử lý ngoại lệ (Exception Handling) trong C#

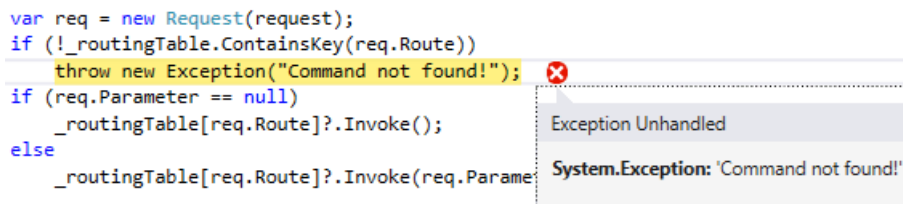
Chúng ta đã nhắc đến khái niệm ngoại lệ (exception) và xem xét cách thức đơn giản nhất để phát thông báo ngoại lệ bằng lệnh `throw` và lớp `Exception`. Exception là một cơ chế

rất mạnh trong .NET giúp phát hiện lỗi logic trong chương trình ở giai đoạn Runtime.

## Exception và chế độ Debug trong C#

Khi chạy chương trình ở chế độ *debug*, nếu phát sinh ngoại lệ, Visual Studio sẽ mở file mã nguồn ở đúng vị trí lỗi cùng với thông báo cụ thể. Qua đó, chúng ta có thể xác định nguồn gốc của lỗi và đưa ra cách giải quyết.

Hình dưới đây minh họa tình huống lỗi khi người dùng nhập vào một lệnh chưa tồn tại.



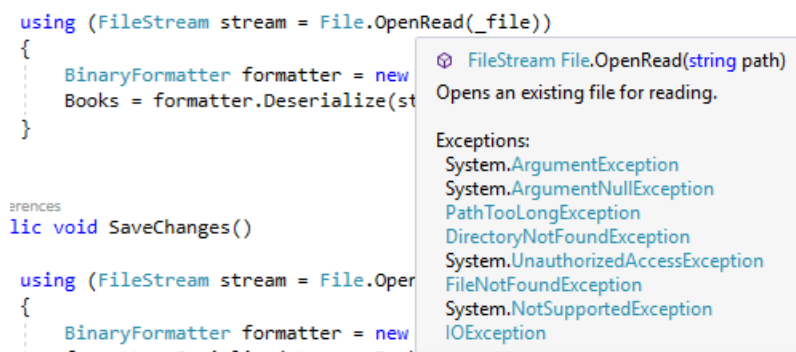
Ngoại lệ ở chế độ chạy debug

Đây là cơ chế bắt và xử lý lỗi ở chế độ Debug. Chương trình chúng ta viết từ đầu dự án đến giờ đều dịch và chạy ở chế độ Debug.

.NET framework cung cấp nhiều lớp hỗ trợ thông báo lỗi kế thừa từ lớp Exception với các thông tin chi tiết hơn về lỗi có thể gặp phải. Nếu một phương thức nào đó có khả năng phát sinh lỗi, Visual Studio sẽ hiển thị danh sách các loại lỗi có thể gặp phải.

Ví dụ, đối với phương thức OpenRead của lớp File có thể phát sinh 8 loại Exception khác nhau như lỗi vào ra (IOException), lỗi không tìm thấy file (FileNotFoundException), lỗi không tìm thấy thư mục (DirectoryNotFoundException), v.v..

Danh sách các loại ngoại lệ này được thể hiện bằng các class khác nhau sẽ được sử dụng nếu tình huống lỗi tương ứng phát sinh. Khi đặt con trỏ chuột lên tên phương thức này chúng ta sẽ xem được danh sách các lớp chứa thông tin về ngoại lệ của phương thức:



Các ngoại lệ có thể phát sinh khi sử dụng phương thức OpenRead

Thông tin này có nghĩa là, nếu phát sinh bất kỳ exception nào trong 8 loại exception có thể xảy ra khi thực hiện phương thức này, lệnh throw sẽ được gọi cùng với một object của loại exception tương ứng.

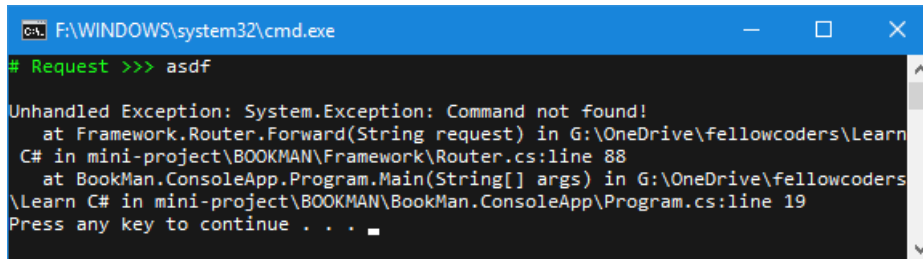
Ví dụ, nếu cung cấp một file không tồn tại, phương thức OpenRead không thể làm gì được và sẽ phát lệnh throw new FileNotFoundException(), tương tự như cách chúng ta tự phát ra

thông báo ngoại lệ ở trên.

## Xử lý ngoại lệ ở chế độ Release

Một chương trình trước khi đem triển khai cho người dùng cuối phải được dịch ở chế độ Release. Chương trình được dịch ở chế độ này sẽ không chạy được ở chế độ Debug nữa.

Nếu chương trình chạy ở chế độ Release mà gặp lỗi, thông báo lỗi sẽ được hiển thị như dưới đây.

A screenshot of a Windows command prompt window titled "F:\WINDOWS\system32\cmd.exe". The prompt shows a command "# Request >>> asdf" being entered. Below the command, an unhandled exception is displayed: "Unhandled Exception: System.Exception: Command not found!". The stack trace shows the exception was thrown in "G:\OneDrive\fellwcoders\Learn C# in mini-project\BOOKMAN\Framework\Router.cs:line 88" and caught in "G:\OneDrive\fellwcoders\Learn C# in mini-project\BOOKMAN\ConsoleApp\Program.cs:line 19". The prompt ends with "Press any key to continue . . .".

```
F:\WINDOWS\system32\cmd.exe
# Request >>> asdf

Unhandled Exception: System.Exception: Command not found!
   at Framework.Router.Forward(String request) in G:\OneDrive\fellwcoders\Learn C# in mini-project\BOOKMAN\Framework\Router.cs:line 88
   at BookMan.ConsoleApp.Program.Main(String[] args) in G:\OneDrive\fellwcoders\Learn C# in mini-project\BOOKMAN\ConsoleApp\Program.cs:line 19
Press any key to continue . . .
```

Đây là cơ chế bắt và xử lý lỗi mặc định của .NET framework đối với ứng dụng console.

Đối với ứng dụng windows form, giao diện bắt và xử lý lỗi có khác biệt.

Như chúng ta thấy, cơ chế bắt và xử lý lỗi mặc định của .NET framework tương đối không thân thiện với người dùng.

.NET cũng cung cấp cho các chương trình tính năng *bắt và xử lý ngoại lệ* (Exception Handling) để tự mình xác định hoạt động của chương trình khi xảy ra lỗi (ngoại lệ), tránh phải sử dụng cơ chế bắt và xử lý lỗi mặc định.

## Cấu trúc try – catch

Cơ chế bắt và xử lý ngoại lệ sử dụng cấu trúc cú pháp sau:

```
try { <code có khả năng gây lỗi viết trong block này> }
catch(<loại lỗi 1> object1) { <hành động khi xảy ra lỗi> }
catch(<loại lỗi 2> object2) { <hành động khi xảy ra lỗi> }
... // có thể kết hợp nhiều khối catch nữa ở đây
finally{ <hành động sẽ thực hiện cả khi có lỗi hay không có lỗi> }
```

Cấu trúc này có 3 khối code:

Khối "try": chứa các đoạn code có khả năng gây lỗi;

Các khối "catch": dùng để bắt từng loại lỗi cụ thể.

Khi xảy ra lỗi, khối này sẽ bắt object của lớp exception tương ứng (mà ở phần phát ngoại lệ tạo ra cùng lệnh throw) và thực thi đoạn code tương ứng. Trong đoạn code này có thể sử dụng các object chứa thông tin ngoại lệ mà khối catch này bắt được.

Chúng ta đã biết các lớp ngoại lệ kế thừa nhau tạo thành một cấu trúc phân cấp với lớp Exception ở gốc. Nếu khối catch được chỉ định bắt loại lỗi cha, nó đồng thời bắt tất cả các loại lỗi con kế thừa từ lớp cha đó. Nếu chúng ta bắt lỗi thuộc loại cao nhất là Exception thì cũng đồng thời bắt tất cả các loại lỗi có thể phát sinh trong chương trình.

Khối "finally": không bắt buộc. Nếu có mặt khối này, dù có xảy ra ngoại lệ hay không thì các lệnh trong khối code này đều sẽ được thực hiện.

## Kết luận

---

Có một số điểm cần lưu ý khi làm việc với ngoại lệ:

- Phân biệt giữa **phát ra** ngoại lệ với **bắt/xử lý** ngoại lệ;
- Không nên bắt ngoại lệ ở các class cấp thấp (tức là class được sử dụng bởi class khác). Việc bắt ngoại lệ nên đặt ở class cấp cao nhất. Như đối với ứng dụng console, đó là ở phương thức Main.
- Ở các class cấp thấp chỉ phát ra ngoại lệ, các class cấp cao sẽ bắt và xử lý.

+ Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.

+ Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.

+ Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.

Cảm ơn bạn!