استثناها خطاهای درحال اجرا هستند. به طور معمول خطاها توسط دستور if تشخیص و در صورت برقرار بودن شرط عمل مناسب، که اغلب نمایش پیغام و اتمام برنامه بود، انجام می گرفت. راه دیگر هنگام مواجه شدن با خطا گیر انداختن آن هنگام پیش آمدن شرایط خطاست. کدی که ممکن است باعث خطا شود اجرا می شود و هر خطائی که ایجاد می شود گرفته می شود و عمل مناسب انجام می گیرد. بلاک های try-catch بر اساس این مفهوم عمل می کنند. مدیریت خطاها به طور سطحی در اینجا معرفی می شود.

مدیریت خطاها از موضوعات مهم در برنامه نویسی است. حالات متعددی وجود دارند که ممکن است ایجاد خطا کند. مثلا باز کردن فایلی که وجود ندارد یا دریافت ورودی نادرست از کاربر. کنترل این خطاها هنگام اجرای برنامه اهمیت زیادی دارد. اگر برنامه این گونه خطاها را در دست نگیرد برنامه سقط می کند و نارضایتی کاربر را باعث می شود. روش قدیمی بررسی خطا توسط if و دادن یک پیغام دوستانه به کاربر و اتمام برنامه بود. C++ راه دیگری را توسط بلاک های try-catch برای مدیریت خطاهای در حال اجرا در اختیار می گذارد.

خطاها

خطاها را به سه دسته می توان تقسیم کرد:

- خطاهای گرامری (syntax error). خطاهائی که با رعایت نکردن قواعد زبان برنامه نویسی توسط کامپایلر تشخیص داده می شوند. مثلا بجای if(x !=y) بنویسید if(x <>y). کامپایلر کد برنامه را تا وقتی خطای گرامری وجود دارد کامپایل نمی کند.
- خطاهای زمان اجرا (runtime error). هر وقفه ای که در جریان عادی اجرای برنامه پیش آید که معمولا باعث سقط برنامه می شود. مثلا باز کردن فایلی که وجود ندارد یا تقسیم بر صفر. این خطاها استثنا (exception) نامیده می شوند.
- خطاهای منطقی (login error). وقتی برنامه کامپایل و اجرا می شود اما به دلیل خطائی در منطق برنامه نتایج غلطی تولید می کند. سخت ترین نوع خطا است که معمولا به آن bug هم گفته می شود.

برای رفع خطا ابتدائی ترین کاری که می شود کرد تست کلیه برنامه و کشف و برطرف سازی خطا است. کنترل مناسب خطاها از سقط برنامه جلوگیری می کند. کامپایلر خطاهای گرامری را

مشخص می کند. مدیریت استثناها خطاهای زمان اجرا را بر عهده دارند بنابراین خطاهای منطقی بزرگترین مشکل برنامه نویس خواهد بود.

مدیریت استثناها

قبل از اینکه سیستم مدیریت استثناها معرفی شود برنامه های کامپیوتری کاملا بدون خطا نبودند. البته برای گرفتن خطاها روش هائی وجود داشت که معمول ترین آن بررسی مقداری بود که توسط توابع برگردانده می شد و نحوه اجرای آن را بیان می کرد. کدی که تابع را فراخوانی می کرد مقدار بر گشتی را چک می کرد و مطابق با آن عمل می کرد (اگر با استفاده از Windows API برنامه نوشته باشید با این روش آشنا هستید).

مفهوم کلی مدیریت استثناها ساده است. هر زمان که استثنائی شناسائی می شود یک فلگ فرضی خطا بالا می رود. سیستمی همواره مراقب این فلگ خطاست. و درصورت بالا رفتن فلگ کد مدیریت خطا فراخوانی می شود. بالا رفتن فلگ خطا را اصطلاحا گیر انداختن (throwing) خطا می نامند. وقتی خطائی به دام انداخته می شود سیستم با گرفتن خطا (catching) عکس العمل نشان می دهد. محاصره کردن بلاکی از کد حساس به خطا و مدیریت استثنا را سعی کردن (trying) به اجرای بلاک می نامند.

دلایل خوبی برای استفاده از مدیریت استثناها وجود دارد. مهمترین آن جداکردن کد مدیریت خطا از کد برنامه است. وقتی استثنائی گرفته می شود مسیری موازی اجرا طی می شود و با اجرای نرمال کد برنامه تداخلی ندارد. این باعث می شود نوشتن کد ساده تر شود چون مجبور نیستید برای هر خطائی چک داشته باشید. علاوه بر این یک استثنا نمی تواند ندیده گرفته شود و تضمین می کند که با خطا برخورد کند. و چون جلوی سقط ناخواسته برنامه را می گیرد اتکاپذیری برنامه را بالا می برد.

از مزایای دیگر مدیریت استثناها این است که خطا می تواند خارج از محدوده تابع گیرانداخته شود. یعنی اگر درونی ترین تابع خطائی داشته باشد این خطا می تواند تا تابع بالائی که دارای بلاک trying است منتشر شود که به برنامه نویس اجازه می دهد کد مدیریت خطا را در هر جا مثل تابع اصلی قرار بدهد.

بلاک های try-catch

مدیریت استثنا بر اساس مفاهیمی که گفته شد توسط سه دستور try، catch و throw عمل می کند. بلاک کدی می خواهیم استثناهای آن را بگیریم با دستور try مشخص می شود. درون بلاک می توان هر خطائی با دستور throw گیر افتاده و از بین می رود. بلاک catch که محلی برای کد مدیریت خطا است بلافاصله بعد از بلاک try قرار می گیرد.

فرم کلی به صورت زیر است:

```
try
{
  ...
  ...
  throw Exception;
  ...
  ...
  }
catch( Exception e )
{
  ...
  ...
  }
```

سه نقطه درون پرانتز catch به معنی اینستکه کلیه خطاها توسط این بلاک مدیریت می شود. می توان انواع مختلفی از خطاها را هندل کرد.

وقتی استثنائی رخ می دهد کنترل به بلاک catch منتقل می شود بنابراین دستورات بعد از throw اجرا نخواهد شد. در بلاک catch کد باید به نحوی باشد که اجرای برنامه را به صورت معمول ادامه یابد یا در صورت برطرف نشدن خطا با exit یا abort خاتمه پیدا کند. اگر بلاک catch اجرای برنامه را به پایان نرساند کنترل اجرا به دستور بعد از بلاک try-catch منتقل می شود.

نکته. بین دو بلاک هیچ چیز نباید باشد.
نکته. در صورت گیر انداختن یک استثنا دستورات بعد از throw اجرا نخواهد شد.
نکته. یک استثنا را می توان به تابعی که تابع جاری را صدا زده است انداخت تا مدیریت شود .

هر دستور try می تواند با چندین دستور catch در ارتباط باشد. هر کدام برای نوع مختلفی از استثنا. نوع داده ای که در دستور catch مشخص شده است با استثنای تطبیق داده می شود

سپس بخش مربوطه به اجرا در می آید و از بقیه صرفنظر می شود. هر نوع داده ای از جمله کلاس های تعریف شده در برنامه می تواند بدام انداخته شود.

وقتی چندین خطا را پاسخ می دهید سلسله مراتب خطاها اهمیت دارد. خطاها با ترتیبی که در سلسله مراتب نشان داده شده است باید در تله انداخته شوند یعنی اگر شما run-time-error را گیر انداختید نمی توانید عبارت catch دیگری برای error file access داشته باشید چون خطاهائی که در سلسله مراتب بالاتر هستند کلیه خطاهای پایینی را در بر دارند.

Exception – the parent class for all exception class
Login-error – the parent class of a variety of logic error
invalid-argumet
out-of-range
Runtime-error – the parent of class for a variety of rumtime error
overflow-error
FileAccessError
range-error

(#Exception Handling in C)

یک خطای زمان اجراست که بدلیل شرایطی غیرنرمال در برنامه ایجاد می شود. در سی شارپ exeption کلاسی است در فضای نام سیستم. شیء ایی از نوع exception بیانگر شرایطی است که سبب رخ دادن خطا در کد شده است. سی شارپ از exception ها به صورتی بسیار شبیه به جاوا و سی پلاس پلاس استفاده می نماید.

دلایلی که باید در برنامه handling exception حتما صورت گیرد به شرح زیر است:
- قابل صرفنظر کردن نیستند و اگر کدی این موضوع را در نظر نگیرد با یک خطای زمان اجرا خاتمه پیدا خواهد کرد.
- سبب مشخص شدن خطا در یک نقطه از برنامه شده و ما را به اصلاح آن سوق می دهد.

بوسیله ی عبارات try...catch می توان مدیریت خطاها را انجام داد. کدی که احتمال دارد خطایی در آن رخ دهد درون try قرار گرفته و سپس بوسیله ی یک یا چند قطعه ی catch می توان آنرا مدیریت کرد. و اگر از این قطعات خطایابی استفاده نشود برنامه به صورتهای زیر متوقف خواهد شد :

```
class A{ static void Main() {catch {}}}
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected
```

```
class A{ static void Main() {finally {}}}
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected

class A{ static void Main() {try {}}}
TEMP.cs(6,3): error CS1524: Expected catch or finally
```

بهتر است یک مثال ساده را در این زمینه مرور کنیم

```
int a, b = 0 ;
Console.WriteLine( "My program starts " );
try
{
a = 10 / b;
}
catch ( Exception e )
{
Console.WriteLine ( e ) ;
}
Console.WriteLine ( "Remaining program" );
The output of the program is:
My program starts
System.DivideByZeroException: Attempted to divide by zero.
at ConsoleApplication4.Class1.Main(String[] args) in
d:\dont delete\consoleapplication4\class1.cs:line 51
Remaining program
```

برنامه شروع به اجرا می کند. سپس وارد بلوک و یا قطعه ی try می گردد. اگر هیچ خطایی هنگام اجرای دستورات داخل آن رخ ندهد ، برنامه به خط آخر جهش خواهد کرد و کاری به قطعات catch ندارد.

اما در اینجا در اولین try عددی بر صفر تقسیم شده است بنابراین کنترل برنامه به بلوک catch منتقل می شود و صرفا نوع خطای رخ داده شده نوشته و نمایش داده می شود. سپس برنامه به کار عادی خودش ادامه می دهد.

تعدادی از کلاس های exception در سی شارپ که از کلاس System.Exception ارث برده اند به شرح زیر هستند :

- Exception Class - - Cause
- SystemException - A failed run-time check;used as a base class for other.
- AccessException - Failure to access a type member, such as a method or field.

• ArgumentException - An argument to a method was invalid.
• ArgumentNullException - A null argument was passed to a method that doesn't accept it.
• ArgumentOutOfRangeException - Argument value is out of range.
• ArithmeticException - Arithmetic over - or underflow has occurred.
• ArrayTypeMismatchException - Attempt to store the wrong type of object in an array.
• BadImageFormatException - Image is in the wrong format.
• CoreException - Base class for exceptions thrown by the runtime.
• DivideByZeroException - An attempt was made to divide by zero.
• FormatException - The format of an argument is wrong.
• IndexOutOfRangeException - An array index is out of bounds.
• InvalidCastExpression - An attempt was made to cast to an invalid class.
• InvalidOperationException - A method was called at an invalid time.
• MissingMemberException - An invalid version of a DLL was accessed.
• NotFiniteNumberException - A number is not valid.
• NotSupportedException - Indicates sthat a method is not implemented by a class.
• NullReferenceException - Attempt to use an unassigned reference.
• OutOfMemoryException - Not enough memory to continue execution.
• StackOverflowException - A stack has overflown.

در کد فوق صرفا عمومی ترین نوع از این کلاس ها که شامل تمامی این موارد می شود مورد استفاده قرار گرفت یعنی :

```
catch ( Exception e )
```

اگر نیازی به خطایابی دقیقتر باشد می توان از کلاس های فوق برای اهداف مورد نظر استفاده نمود.

مثالی دیگر: (در این مثال خطایابی دقیق تر با استفاده از کلاس های فوق و همچنین مفهوم finally نیز گنجانده شده است)

```
int a, b = 0 ;
Console.WriteLine( "My program starts" );
try
{
a = 10 / b;
}
catch ( InvalidOperationException e )
{
Console.WriteLine ( e ) ;
}
catch ( DivideByZeroException e)
{
Console.WriteLine ( e ) ;
}
finally
{
Console.WriteLine ( "finally" ) ;
}
Console.WriteLine ( "Remaining program" );
The output here is:
My program starts
System.DivideByZeroException: Attempted to divide by zero.
at ConsoleApplication4.Class1.Main(String[] args) in
d:\dont delete\consoleapplication4\class1.cs:line 51
finally
Remaining program
```

قسمت موجود در قطعه ی فاینالی همواره صرفنظر از قسمت های دیگر اجرا می شود.
به مثال زیر دقت کنید :

```
int a, b = 0 ;
Console.WriteLine( "My program starts" )
try
{
a = 10 / b;
}
finally
```

```
{
Console.WriteLine ( "finally" ) ;
}
Console.WriteLine ( "Remaining program" );
Here the output is
My program starts
Exception occurred: System.DivideByZeroException:
Attempted to divide by zero.at ConsoleApplication4.Class1.
Main(String[] args) in d:\dont delete\consoleapplication4
\class1.cs:line 51
finally
```

قسمت چاپ Remaining program اجرا نشده است.

عبارت throw :

این عبارت سبب ایجاد یک خطا در برنامه می شود.

مثال :

```
int a, b = 0 ;
Console.WriteLine( "My program starts" ) ;
try
{
a = 10 / b;
}
catch ( Exception e)
{
throw
}
finally
{
Console.WriteLine ( "finally" ) ;
}
```

در این حالت قسمت فاینالی اجرا شده و برنامه بلافاصله خاتمه پیدا می کند

Mohammad Amirkhiz
1389/05/05