# The implementation of iterators in C# and its consequences (part 3)

devblogs.microsoft.com/oldnewthing/20080814-00

August 14, 2008

Raymond Chen

I mentioned that there was an exception to the general statement that the conversion of an iterator into traditional C# code is something you could have done yourself. That's true, and it was also a pun, because the exception is exception handling.

If you have a `try … finally` block in your iterator, the language executes the `finally` block under the following conditions:

- After the last statement of the `try` block is executed. (No surprise here.)
- When an exception propagates out of the `try` block. (No surprise here either.)
- When execution leaves the `try` block via `yield break`.
- When the iterator is `Dispose`d and the iterator body was trapped inside a `try` block at the time.

That last case can occur if somebody decides to abandon the enumerator before it is finished.

```
IEnumerable<int> CountTo10()
{
 try {
  for (int i = 1; i <= 10; i++) {
   yield return i;
  }
 } finally {
  System.Console.WriteLine("finally");
 }
}


foreach (int i in CountTo10()) {
 System.Console.WriteLine(i);
 if (i == 5) break;
}
```

This code fragment prints "1 2 3 4 5 finally".

If you think about it, this behavior is completely natural. You want the `finally` block to execute when the `try` block is finished executing, either by normal or abnormal means. Although control leaves the `try` block during the `yield return`, it comes back when the caller asks for the next item from the enumerator, so execution of the `try` block isn't finished yet. The `try` is finished executing after the last statement completes, an exception is thrown past it, or execution is abandoned when the enumerator is prematurely destroyed.

And this is exactly what you want when you use the `finally` block to clean up resources used by the `try` block.

Now, technically, you *can* write this yourself without using iterators, but it's pretty ugly. You'll need more internal state variables to keep track of whether the `try` block is still active and whether the exit of the `try` block is temporary (due to `yield return`) or permanent. It's a real pain in the neck, however, so you probably are better off letting the compiler do the work for you.

Raymond Chen

**Follow**