The implementation of iterators in C# and its consequences (part 4)

devblogs.microsoft.com/oldnewthing/20080815-00

August 15, 2008

Raymond Chen

You can breathe a sigh of relief. Our long national nightmare is over: this is the end of CLR Week 2008. We wind down with a look back at iterators.

Michael Entin points out that you can use C# iterators to make asynchronous code easier to write. You can use C# iterators for more than simply iterating.

The automatic conversion of straight line code into a state machine is handy when you want an easy way to write, well, a state machine. It's one of those things that's blindingly obvious once you look at it the right way.

The transformation that the `yield return` statement induces on your function turns it from a boring function into an implicit state machine: When you execute a `yield return`, execution of your function is suspended until somebody asks your iterator the next item, at which point execution resumes at the statement after the `yield return`. This is exactly what you want when breaking a synchronous function into asynchronous pieces: Each time you would normally block on an operation, you instead perform a `yield return`, and when the operation completes, you call the `MoveNext` method, which resumes execution of the function until the next time it needs to wait for something and performs a `yield return`.

It's so simple it's magic.

Additional iterator-related reading:

- Dan Crevier investigates a problem where an enumerator's `Dispose` method was never called.
- Stuart Leeks has his own series on the `yield` keyword: Part 1, Part 2.
- Eric Lippert invites you to use your psychic debugging powers to diagnose what was originally reported as a compiler bug. You in fact already know enough to solve this. (Answer.)

Raymond Chen

Follow