Writing the F# recursive folder visitor in C# - seq vs IEnumerable

Asked 16 years, 1 month ago Modified 14 years, 8 months ago Viewed 4k times

9

I often use this recursive 'visitor' in F#

```
let rec visitor dir filter=
    seq { yield! Directory.GetFiles(dir, filter)
          for subdir in Directory.GetDirectories(dir) do yield! visitor subdir
filter}
```

Recently I've started working on implementing some F# functionality in C#, and I'm trying to reproduce this as IEnumerable, but I'm having difficulty getting any further than this:

```
static IEnumerable<string> Visitor(string root, string filter)
{
    foreach (var file in Directory.GetFiles(root, filter))
        yield return file;
    foreach (var subdir in Directory.GetDirectories(root))
        foreach (var file in Visitor(subdir, filter))
            yield return file;
}
```

What I don't understand is why I have to do a double foreach in the C# version for the recursion, but not in F#... Does the seq {} implicitly do a 'concat'?

c# f#

Share Improve this question Follow

asked Nov 21, 2008 at 11:26
Benjol
66.4k ● 55 ● 192 ● 275

4 Answers

Sorted by: Highest score (default)

14

`yield!` does a 'flatten' operation, so it integrates the sequence you passed it into the outer sequence, implicitly performing a `foreach` over each element of the sequence and `yield` on each one.

Share Improve this answer Follow

answered Nov 21, 2008 at 12:27

Sunlight
2,113 ● 13 ● 9

1 So could I do do a .SelectMany to reproduce this in C#? – Benjol Nov 21, 2008 at 12:44

1 C#'s `yield return` can only return one item at a time, so you're stuck with your foreaches, I'm afraid. – Sunlight Nov 21, 2008 at 15:59

3

There is no simple way to do this. You could workaround this by defining a C# type that can store either one value or a sequence of values - using the F# notation it would be:

```
type EnumerationResult<'a> =
  | One of 'a
  | Seq of seq<'a>
```

(translate this to C# in any way you like :-))

Now, you could write something like:

```
static IEnumerable<EnumerationResult<string>> Visitor
       (string root, string filter) {
   foreach (var file in Directory.GetFiles(root, filter))
      yield return EnumerationResult.One(file);
      foreach (var subdir in Directory.GetDirectories(root))
           yield return EnumerationResult.Seq(Visitor(subdir, filter))
   }
}
```

To use it, you'd have to write a function that flattens EnumerationResult, which could be an extension method in C# with the following signature:

```
IEnumerable<T> Flatten(this IEnumerable<EnumerationResult<T>> res);
```

Now, this is a part where it gets tricky - if you implemented this in a straighforward way, it would still contain "forach" to iterate over the nested "Seq" results. However, I believe that you could write an optimized version that wouldn't have quadratic complexity.

Ok.. I guess this is a topic for a blog post rather than something that could be fully described here :-), but hopefully, it shows an idea that you can try following!

[EDIT: But of course, you can also use naive implementation of "Flatten" that would use "SelectMany" just to make the syntax of your C# iterator code nicer]

Share Improve this answer Follow

answered Nov 21, 2008 at 16:24

Tomas Petricek
243k ● 22 ● 394 ● 563

2

In the specific case of retrieving all files under a specific directory, this overload of `Directory.GetFiles` works best:

```
static IEnumerable<string> Visitor( string root, string filter ) {
  return Directory.GetFiles( root, filter, SearchOption.AllDirectories );
}
```

In the general case of traversing a tree of enumerable objects, a nested foreach loop or equivalent is required (see also: All About Iterators).

Edit: Added an example of a function to flatten any tree into an enumeration:

```
static IEnumerable<T> Flatten<T>( T item, Func<T, IEnumerable<T>> next ) {
  yield return item;
  foreach( T child in next( item ) )
    foreach( T flattenedChild in Flatten( child, next ) )
      yield return flattenedChild;
}
```

This can be used to select all nested files, as before:

```
static IEnumerable<string> Visitor( string root, string filter ) {
  return Flatten( root, dir => Directory.GetDirectories( dir ) )
    .SelectMany( dir => Directory.GetFiles( dir, filter ) );
}
```

Share

Improve this answer

Follow

edited Nov 29, 2008 at 3:59

answered Nov 29, 2008 at 3:46

Emperor XLII
13.4k ● 11 ● 67 ● 77

1 Actually, that particular overload has a serious practical issue; namely that if *any* file or directory within the search space is invalid by virtue of having a too-long path or the user not having the appropriate permissions or any other IO exception, the entire operation is aborted and returns no results. By contrast, using a manually recursive search, there are no such

issues; you can try-catch each directory's listing individually. – Eamon Nerbonne Apr 26, 2010 at 20:50

2

In C#, I use the following code for this kind of function:

```
public static IEnumerable<DirectoryInfo> TryGetDirectories(this DirectoryInfo
dir) {
    return F.Swallow(() => dir.GetDirectories(), () => new DirectoryInfo[] {
});
}
public static IEnumerable<DirectoryInfo> DescendantDirs(this DirectoryInfo dir)
{
    return Enumerable.Repeat(dir, 1).Concat(
        from kid in dir.TryGetDirectories()
        where (kid.Attributes & FileAttributes.ReparsePoint) == 0
        from desc in kid.DescendantDirs()
        select desc);
}
```

This addresses IO errors (which inevitably happen, unfortunately), and avoids infinite loops due to symbolic links (in particular, you'll run into that searching some dirs in windows 7).

Share Improve this answer Follow

answered Apr 26, 2010 at 20:56

Eamon Nerbonne
48k ● 20 ● 104 ● 171