ASP.NET MVC 3: Razor's @: and <text> syntax

Thursday, December 16, 2010 .NET ASP.NET MVC

This is another in a series of posts I'm doing that cover some of the new ASP.NET MVC 3 features:

- New @model keyword in Razor (Oct 19th)
- Layouts with Razor (Oct 22nd)
- Server-Side Comments with Razor (Nov 12th)
- Razor's @: and <text> syntax (today)

In today's post I'm going to discuss two useful syntactical features of the new Razor view-engine – the @: and <text> syntax support.

Fluid Coding with Razor

ASP.NET MVC 3 ships with a new view-engine option called "Razor" (in addition to the existing .aspx view engine). You can learn more about Razor, why we are introducing it, and the syntax it supports from my Introducing Razor blog post.

Razor minimizes the number of characters and keystrokes required when writing a view template, and enables a fast, fluid coding workflow. Unlike most template syntaxes, you do not need to interrupt your coding to explicitly denote the start and end of server blocks within your HTML. The Razor parser is smart enough to infer this from your code. This enables a compact and expressive syntax which is clean, fast and fun to type.

For example, the Razor snippet below can be used to iterate a list of products:

```
<h2>Products</h2>

<ul>
  @foreach (var p in products) {
     <li>@p.ProductName</li>
  }
</ul>
```

When run, it generates output like:

```
<h2>Products</h2>

<ul>
      <li>Chai</li>
      <li>Chang</li>
      <li>Guarana Fantastica</li>
      <li>Sasquatch Ale</li>
      <li>Steeleye Stout</li>
      <li>Cote de Blaye</li>
      <li>Chartreuse verte</li>
      <li>Ipoh Coffee</li>
      <li>Laughing Lumberjack Lager</li>
      <li>Outback Lager</li>
      <li>Rhonbrau Klosterbier</li>
      <li>Lakkalikoori</li>
</ul>
```

One of the techniques that Razor uses to implicitly identify when a code block ends is to look for tag/element content to denote the beginning of a content region. For example, in the code snippet above Razor automatically treated the inner block within our foreach loop as an HTML content block because it saw the opening <li> tag sequence and knew that it couldn't be valid C#.

This particular technique – using tags to identify content blocks within code – is one of the key ingredients that makes Razor so clean and productive with scenarios involving HTML creation.

Using @: to explicitly indicate the start of content

Not all content container blocks start with a tag element tag, though, and there are scenarios where the Razor parser can't *implicitly* detect a content block.

Razor addresses this by enabling you to *explicitly* indicate the beginning of a line of content by using the @: character sequence within a code block. The @: sequence indicates that the line of content that follows should be treated as a content block:

```
@if (someCondition) {
    @: Some content not surrounded by a tag element...
}
```

As a more practical example, the below snippet demonstrates how we could output a "(Out of Stock!)" message next to our product name if the product is out of stock:

```
<h2>Products</h2>

<ul>
  @foreach (var p in products) {
     <li>
         @p.ProductName

         @if (p.UnitsInStock == 0) {
            @: (Out of stock!)
         }
     </li>
  }
</ul>
```

Because I am not wrapping the (Out of Stock!) message in an HTML tag element, Razor can't implicitly determine that the content within the @if block is the start of a content block. We are using the @: character sequence to explicitly indicate that this line within our code block should be treated as content.

Using Code Nuggets within @: content blocks

In addition to outputting static content, you can also have code nuggets embedded within a content block that is initiated using a @: character sequence.

For example, we have two @: sequences in the code snippet below:

```
<ul>
  @foreach (var p in products) {
     <li>
         @p.ProductName

         @if (p.UnitsInStock == 0) {
            @: (Out of stock!)
         }
         else if (p.UnitsInStock < 4) {
            @: (Only @p.UnitsInStock left!)
         }
     </li>
  }
</ul>
```

Notice how within the second @: sequence we are emitting the number of units left within the content block (e.g. - "(Only 3 left!"). We are doing this by embedding a @p.UnitsInStock code nugget within the line of content.

Multiple Lines of Content

Razor makes it easy to have multiple lines of content wrapped in an HTML element. For example, below the inner content of our @if container is wrapped in an HTML <p> element – which will cause Razor to treat it as content:

```
@if (p.UnitsInStock == 0) {
    <p>
        Line one of content
        Line two of content
        Date is: @DateTime.Now
        Line four of content
    </p>
}
```

For scenarios where the multiple lines of content are not wrapped by an outer HTML element, you can use multiple @: sequences:

```
@if (p.UnitsInStock == 0) {
    @: Line one of content
    @: Line two of content
    @: Line three of content
}
```

Alternatively, Razor also allows you to use a <text> element to explicitly identify content:

```
@if (p.UnitsInStock == 0) {
    <text>
        This is a multi-line block of content.
        The text tag wraps us and will be removed by the
        Razor parser. We can still have code
        nuggets too: @DateTime.Now
        Isn't that neat?
    </text>
}
```

The <text> tag is an element that is treated specially by Razor. It causes Razor to interpret the inner contents of the <text> block as content, and to not render the containing <text> tag element (meaning only the inner contents of the <text> element will be rendered – the tag itself will not). This makes it convenient when you want to render multi-line content blocks that are not wrapped by an HTML element.

The <text> element can also optionally be used to denote single-lines of content, if you prefer it to the more concise @: sequence:

```
<ul>
  @foreach (var p in products) {
    <li>
        @p.ProductName

        @if (p.UnitsInStock == 0) {
            <text>(Out of stock!)</text>
        }
    </li>
  }
</ul>
```

The above code will render the same output as the @: version we looked at earlier. Razor will automatically omit the <text> wrapping element from the output and just render the content

within it.

Summary

Razor enables a clean and concise templating syntax that enables a very fluid coding workflow. Razor’s smart detection of <tag> elements to identify the beginning of content regions is one of the reasons that the Razor approach works so well with HTML generation scenarios, and it enables you to avoid having to explicitly mark the beginning/ending of content regions in about 95% of if/else and foreach scenarios.

Razor’s @: and <text> syntax can then be used for scenarios where you want to avoid using an HTML element within a code container block, and need to more explicitly denote a content region.

Hope this helps,

Scott

P.S. In addition to blogging, I am also now using Twitter for quick updates and to share links. Follow me at: twitter.com/scottgu