ASP.NET MVC View Engine Comparison

Asked 10 years, 5 months ago Active 3 years, 5 months ago Viewed 105k times

339

223

I've been searching on SO & Google for a breakdown of the various View Engines available for ASP.NET MVC, but haven't found much more than simple high-level descriptions of what a view engine is.

I'm not necessarily looking for "best" or "fastest" but rather some real world comparisons of advantages / disadvantages of the major players (e.g. the default WebFormViewEngine, MvcContrib View Engines, etc.) for various situations. I think this would be really helpful in determining if switching from the default engine would be advantageous for a given project or development group.

Has anyone encountered such a comparison?

asp.net-mvc spark-view-engine viewengine razor

edited May 8 '13 at 22:16

casperOne
68.7k 15 165 233

asked Sep 20 '09 at 15:47

mckamey
16.5k 13 72 113

43 Ironic that it is "not constructive" yet has provided a lot of value to those involved who've viewed it nearly 45K times. Too bad that SO is restricting their own utility despite the needs of the community. I doubt @jeff-atwood would feel that way. – mckamey May 10 '13 at 16:06

6 Answers

430

ASP.NET MVC View Engines (Community Wiki)

Since a comprehensive list does not appear to exist, let's start one here on SO. This can be of great value to the ASP.NET MVC community if people add their experience (esp. anyone who contributed to one of these). Anything implementing `IViewEngine` (e.g. `VirtualPathProviderViewEngine`) is fair game here. Just alphabetize new View Engines (leaving WebFormViewEngine and Razor at the top), and try to be objective in comparisons.

Design Goals:

> A view engine that is used to render a Web Forms page to the response.

Pros:

- ubiquitous since it ships with ASP.NET MVC
- familiar experience for ASP.NET developers
- IntelliSense
- can choose any language with a CodeDom provider (e.g. C#, VB.NET, F#, Boo, Nemerle)
- on-demand compilation or precompiled views

Cons:

- usage is confused by existence of "classic ASP.NET" patterns which no longer apply in MVC (e.g. ViewState PostBack)
- can contribute to anti-pattern of "tag soup"
- code-block syntax and strong-typing can get in the way
- IntelliSense enforces style not always appropriate for inline code blocks
- can be noisy when designing simple templates

Example:

```
<%@ Control Inherits="System.Web.Mvc.ViewPage<IEnumerable<Product>>" %>
<% if(model.Any()) { %>
<ul>
    <% foreach(var p in model){%>
    <li><%=p.Name%></li>
    <%}%>
</ul>
<%}else{%>
    <p>No products available</p>
<%}%>
```

System.Web.Razor

Pros:

- Compact, Expressive, and Fluid
- Easy to Learn
- Is not a new language
- Has great Intellisense
- Unit Testable
- Ubiquitous, ships with ASP.NET MVC

Cons:

- Creates a slightly different problem from "tag soup" referenced above. Where the server tags actually provide structure around server and non-server code, Razor confuses HTML and server code, making pure HTML or JS development challenging (see Con Example #1) as you end up having to "escape" HTML and / or JavaScript tags under certain very common conditions.
- Poor encapsulation+reuseability: It's impractical to call a razor template as if it were a normal method - in practice razor can call code but not vice versa, which can encourage mixing of code and presentation.
- Syntax is very html-oriented; generating non-html content can be tricky. Despite this, razor's data model is essentially just string-concatenation, so syntax and nesting errors are neither statically nor dynamically detected, though VS.NET design-time help mitigates this somewhat. Maintainability and refactorability can suffer due to this.
- ~~No documented API~~, http://msdn.microsoft.com/en-us/library/system.web.razor.aspx

Con Example #1 (notice the placement of "string[]..."):

```
@{
    <h3>Team Members</h3> string[] teamMembers = {"Matt", "Joanne", "Robert"};
    foreach (var person in teamMembers)
    {
        <p>@person</p>
    }
}
```

Bellevue

Design goals:

- Don't mess with my HTML! The data binding code (Bellevue code) should be separate from HTML.
- Enforce strict Model-View separation

Brail

Design Goals:

> The Brail view engine has been ported from MonoRail to work with the Microsoft ASP.NET MVC Framework. For an introduction to Brail, see the documentation on the Castle project website.

Pros:

- modeled after "wrist-friendly python syntax"
- On-demand compiled views (but no precompilation available)

Cons:

- designed to be written in the language Boo

Example:

```
<html>
<head>
<title>${title}</title>
</head>
<body>
     <p>The following items are in the list:</p>
     <ul><%for element in list:    output "<li>${element}</li>"%></ul>
     <p>I hope that you would like Brail</p>
</body>
</html>
```

Hasic

Hasic uses VB.NET's XML literals instead of strings like most other view engines

Pros:

- Compile-time checking of valid XML
- Syntax colouring
- Full intellisense
- Compiled views
- Extensibility using regular CLR classes, functions, etc
- Seamless composability and manipulation since it's regular VB.NET code
- Unit testable

Cons:

- Performance: Builds the whole DOM before sending it to client.

Example:

```
Protected Overrides Function Body() As XElement
    Return _
    <body>
        <h1>Hello, World</h1>
    </body>
End Function
```

NDjango

Design Goals:

> NDjango is an implementation of the Django Template Language on the .NET platform, using the F# language.

Pros:

- NDjango release 0.9.1.0 seems to be more stable under stress than `WebFormViewEngine`
- Django Template Editor with syntax colorization, code completion, and as-you-type diagnostics (VS2010 only)

NHaml

Design Goals:

> .NET port of Rails Haml view engine. From the Haml website:
>
> Haml is a markup language that's used to cleanly and simply describe the XHTML of any web document, without the use of inline code... Haml avoids the need for explicitly coding XHTML into the template, because it is actually an abstract description of the XHTML, with some code to generate dynamic content.

Pros:

- terse structure (i.e. D.R.Y.)
- well indented
- clear structure
- C# Intellisense (for VS2008 without ReSharper)

Cons:

- an abstraction from XHTML rather than leveraging familiarity of the markup
- No Intellisense for VS2010

Example:

```
@type=IEnumerable<Product>
- if(model.Any())
  %ul
    - foreach (var p in model)
      %li= p.Name
- else
  %p No products available
```

NVelocityViewEngine (MvcContrib)

Design Goals:

Pros:

- easy to read/write
- concise view code

Cons:

- limited number of helper methods available on the view
- does not automatically have Visual Studio integration (IntelliSense, compile-time checking of views, or refactoring)

Example:

```
#foreach ($p in $viewdata.Model)
#beforeall
    <ul>
#each
    <li>$p.Name</li>
#afterall
    </ul>
#nodata
    <p>No products available</p>
#end
```

SharpTiles

Design Goals:

> SharpTiles is a partial port of JSTL combined with concept behind the Tiles framework (as of Mile stone 1).

Pros:

- familiar to Java developers
- XML-style code blocks

Cons:

Example:

```
<c:if test="${not fn:empty(Page.Tiles)}">
  <p class="note">
    <fmt:message key="page.tilesSupport"/>
  </p>
</c:if>
```

Spark View Engine

Design Goals:

> The idea is to allow the html to dominate the flow and the code to fit seamlessly.

Pros:

- Produces more readable templates
- C# Intellisense (for VS2008 without ReSharper)
- SparkSense plug-in for VS2010 (works with ReSharper)
- Provides a powerful Bindings feature to get rid of *all* code in your views and allows you to easily invent your own HTML tags

Cons:

- No clear separation of template logic from literal markup (this can be mitigated by namespace prefixes)

Example:

```
<viewdata products="IEnumerable[[Product]]"/>
<ul if="products.Any()">
    <li each="var p in products">${p.Name}</li>
</ul>
<else>
    <p>No products available</p>
</else>

<Form style="background-color:olive;">
```

```
    <ValidationMessage For="username" Message="Please type a valid username." />
</Form>
```

StringTemplate View Engine MVC

Design Goals:

- Lightweight. No page classes are created.
- Fast. Templates are written to the Response Output stream.
- Cached. Templates are cached, but utilize a FileSystemWatcher to detect file changes.
- Dynamic. Templates can be generated on the fly in code.
- Flexible. Templates can be nested to any level.
- In line with MVC principles. Promotes separation of UI and Business Logic. All data is created ahead of time, and passed down to the template.

Pros:

- familiar to StringTemplate Java developers

Cons:

- simplistic template syntax can interfere with intended output (e.g. jQuery conflict)

Wing Beats

Wing Beats is an internal DSL for creating XHTML. It is based on F# and includes an ASP.NET MVC view engine, but can also be used solely for its capability of creating XHTML.

Pros:

- Compile-time checking of valid XML
- Syntax colouring

- Compiled views
- Extensibility using regular CLR classes, functions, etc
- Seamless composability and manipulation since it's regular F# code
- Unit testable

Cons:

- You don't really write HTML but code that represents HTML in a DSL.

XsltViewEngine (MvcContrib)

Design Goals:

> Builds views from familiar XSLT

Pros:

- widely ubiquitous
- familiar template language for XML developers
- XML-based
- time-tested
- Syntax and element nesting errors can be statically detected.

Cons:

- functional language style makes flow control difficult
- XSLT 2.0 is (probably?) not supported. (XSLT 1.0 is much less practical).

edited May 23 '17 at 12:18

community wiki
46 revs, 19 users 52%
mckamey

Sep 22 '09 at 19:31

7 Down voting because of the Cons section of Brail. Having Boo as the language is certainly not a con. – Owen Mar 28 '10 at 16:42

48 @Owen: Yes it is. You have to look at this from the perspective of a C# developer. You don't want to learn yet another language just to use a templating engine. (naturally if you already know Boo, that's cool, but for the majority of C# developers, this is an addition hurdle to overcome) – Christian Klauser Jun 1 '10 at 17:09

5 Razor is in there. It should be updated to alphabetize Razor. – mckamey Jul 22 '10 at 20:08

3 Boo is a Pro, not a Con. You are already "outside" C#, the terser the template can come the better. C# was not meant to be used in a "templating" context, it is somewhat expressive but not "wrist friendly". On the other hand, BOO was created with that in mind and as such it lends itself much better to be used in a templating context. – Loudenvier May 16 '11 at 15:44

17

My current choice is Razor. It is very clean and easy to read and keeps the view pages very easy to maintain. There is also intellisense support which is really great. ALos, when used with web helpers it is really powerful too.

To provide a simple sample:

```
@Model namespace.model
<!Doctype html>
<html>
<head>
<title>Test Razor</title>
</head>
<body>
<ul class="mainList">
@foreach(var x in ViewData.model)
{
<li>@x.PropertyName</li>
}
</ul>
</body>
```

And there you have it. That is very clean and easy to read. Granted, that's a simple example but even on complex pages and forms it is still very easy to read and understand.

As for the cons? Well so far (I'm new to this) when using some of the helpers for forms there is a lack of support for adding a CSS class reference which is a little annoying.

answered Nov 13 '11 at 19:21

nathj07
587 1 6 22

1 Doh! Just noticed how old this discussion is. Oh well, maybe someone will find it and it will still prove useful. – nathj07 Nov 13 '11 at 19:22

10

I know this doesn't really answer your question, but different View Engines have different purposes. The Spark View Engine, for example, aims to rid your views of "tag soup" by trying to make everything fluent and readable.

Your best bet would be to just look at some implementations. If it looks appealing to the intent of your solution, try it out. You can mix and match view engines in MVC, so it shouldn't be an issue if you decide to not go with a specific engine.

answered Sep 20 '09 at 15:49

MunkiPhD
3,508 26 50

Thanks for the answer. I was literally starting what you suggested when I figured "someone has had to have done a summary already." I'm hoping for some aggregation of these types of design goals and shortcomings. "The Spark View Engine... aims to rid your views of "tag soup" by trying to make everything fluent and readable." That implies a reason for building it as well as a shortcoming of the default view engine. One more bullet in the list. – mckamey Sep 20 '09 at 15:56

8

Check this SharpDOM . This is a c# 4.0 internal dsl for generating html and also asp.net mvc view engine.

answered Apr 30 '10 at 21:10

Anton Shelin
2,450 19 17

Sounds like the only reasonable way to build views. – Stephan Eggermont Jul 7 '10 at 8:22

can you add it to the general wiki'd answer? – Mauricio Scheffer Aug 31 '10 at 13:59

5

I like ndjango. It is very easy to use and very flexible. You can easily extend view functionality with custom tags and filters. I think that "greatly tied to F#" is rather advantage than disadvantage.

answered Feb 25 '10 at 19:48

rdovhan
51 1 1

4

I think this list should also include samples of each view engine so users can get a flavour of each without having to visit every website.

Pictures say a thousand words and markup samples are like screenshots for view engines :) So here's one from my favourite Spark View Engine

```
<viewdata products="IEnumerable[[Product]]"/>
<ul if="products.Any()">
  <li each="var p in products">${p.Name}</li>
</ul>
<else>
  <p>No products available</p>
</else>
```

edited Oct 12 '10 at 21:01

bzlm
8,775 4 55 82

answered Feb 1 '10 at 17:51

mythz
128k 25 225 356

4 looks too much like coldfusion. I'm not a big fan of mixing code into markup like this. It becomes difficult to maintain. – Agile Jedi Dec 30 '10 at 4:55