**Difference between Razor View Engine and ASPX View Engine**

**View Engine is responsible for rendering the view into html form to the browser.** By default, Asp.net MVC support Web Form(ASPX) and Razor View Engine. There are many third party view engines (like Spark, Nhaml etc.) that are also available for Asp.net MVC.

|  |  |
| --- | --- |
| **Razor View Engine** | **Web Form View Engine** |
| Razor Engine is an advanced view engine that was introduced with MVC3. This is **not a new language** but it is a **new markup syntax**. | Web Form Engine is the default view engine for the Asp.net MVC that is included with Asp.net MVC from the beginning. |
| The namespace for Razor Engine is System.Web.Razor. | The namespace for Webform Engine is System.Web.Mvc.WebFormViewEngine. |
| **The file extensions** used with Razor Engine are different from Web Form Engine. It has .**cshtml** (Razor with C#) or .vbhtml (Razor with VB) extension for views, partial views, editor templates and for layout pages. | The file extensions used with Web Form Engine are also like Asp.net Web Forms. It has .aspx extension for views, .ascx extension for partial views & editor templates and .master extension for layout/master pages. |
| Razor has **new and advance syntax** that are compact, expressive and reduces typing. | Web Form Engine has the same syntax like Asp.net Web Forms uses for .aspx pages. |
| Razor syntax are easy to learn and much clean than Web Form syntax. Razor uses **@ symbol** to make the code like as:   1. **@Html.ActionLink("SignUp", "SignUp")** | Web Form syntax are borrowed from Asp.net Web Forms syntax that are mixed with html and sometimes make a view messy. Webform uses delimiters to make the code like as:  <% if(model.Any()) { %>  <ul><% foreach(var p in model){%>  <li><%=p.Name%>  </li>%}%>  </ul><%}else{%>  <p>No products available</p>  <%}%> |
| By default, **Razor Engine prevents XSS attacks(Cross-Site Scripting Attacks)** means it encodes the script or html tags like <,> before rendering to view. | Web Form Engine does not prevent XSS attacks means any script saved in the database will be fired while rendering the page |
| **Razor Engine is little bit slow as compared to Webform Engine.** | Web Form Engine is faster than Razor Engine. |
| Razor Engine, doesn't support design mode in visual studio means you cannot see your page look and feel. | Web Form engine support design mode in visual studio means you can see your page look and feel without running the application. |
| Razor Engine support TDD (Test Driven Development) since it is not depend on System.Web.UI.Page class. | Web Form Engine doesn't support TDD (Test Driven Development) since it depend on System.Web.UI.Page class which makes the testing complex. |

[**System.Web.Mvc.WebFormViewEngine**](http://www.codeplex.com/aspnet)

**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](http://msdn.microsoft.com/en-us/library/ms229863.aspx) 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

[**System.Web.Razor**](http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx)

**Design Goals:**

A view engine for ASP.Net Web development platform, optimized around HTML generation via a code-focused templating approach.

**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.