Does anyone beside me just NOT get ASP.NET MVC? [closed]

Asked 16 years ago Modified 7 years, 4 months ago Viewed 25k times

141

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 13 years ago.

I've been fiddling with ASP.NET MVC since the CTP, and I like a lot of things they did, but there are things I just don't get.

For example, I downloaded beta1, and I'm putting together a little personal site/resume/blog with it. Here is a snippet from the ViewSinglePost view:

```
 <%
        // Display the "Next and Previous" links
        if (ViewData.Model.PreviousPost != null || ViewData.Model.NextPost !=
null)
        {
            %> <div> <%

            if (ViewData.Model.PreviousPost != null)
            {
                %> <span style="float: left;"> <%
                    Response.Write(Html.ActionLink("<< " +
ViewData.Model.PreviousPost.Subject, "view", new { id =
ViewData.Model.PreviousPost.Id }));
                %> </span> <%
            }

            if (ViewData.Model.NextPost != null)
            {
                %> <span style="float: right;"> <%

Response.Write(Html.ActionLink(ViewData.Model.NextPost.Subject + " >>", "view",
new { id = ViewData.Model.NextPost.Id }));
                %> </span> <%
            }
            %>
                    <div style="clear: both;" />
                </div> <%
        }
    %>
```

Disgusting! **(Also note that the HTML there is temporary placeholder HTML, I'll make an actual design once the functionality is working)**.

Am I doing something wrong? Because I spent many dark days in classic ASP, and this tag soup reminds me strongly of it.

Everyone preaches how you can do cleaner HTML. Guess, what? 1% of all people look at the outputted HTML. To me, I don't care if Webforms messes up my indentation in the rendered HTML, as long as I have code that is easy to maintain...This is not!

So, convert me, a die hard webforms guy, why I should give up my nicely formed ASPX pages for this?

Edit: Bolded the "temp Html/css" line so people would stfu about it.

asp.net-mvc

Share Follow

edited Mar 23, 2016 at 14:13

Rob

45.7k ● 24 ● 125 ● 155

asked Dec 24, 2008 at 3:43

FlySwat

175k ● 75 ● 248 ● 314

3 man that is some ugly markup! – Steven A. Lowe Dec 24, 2008 at 3:48

40 You include CSS markup with your HTML? – Todd Smith Dec 24, 2008 at 4:13

12 Your formatting sucks. THat's not a problem inherent to MVC, it's a sign of a bad HTML programmer. Too much time spent in the observer pattern, me thinks. A little more than drag, drop, click is required here. – Chris Dec 24, 2008 at 4:59

7 Never mind MVC, calling Winforms "easy to maintain" is silly. It's easy to maintain as long as you don't need any degree of control over the output. Which means it works fine as long as your users only use MS-sanctioned web browsers. – Stack Overflow is garbage Dec 24, 2008 at 6:41

2 Am I doing something wrong? - Yes. But it isn't your fault. You need to write some nice HTML and then add the output from the Model to it. You don't need Response.Write ever! The idea behind the MVC framework is that it makes things much cleaner than .aspx, whereas your example looks more like Classic ASP. – Fenton Jul 28, 2011 at 11:24

24 Answers

Sorted by: Highest score (default)

154

Compared to Web Forms, MVC is simultaneously a lower-level approach to HTML generation with greater control over the page output *and* a higher-level, more architecturally-driven approach. Let me capture Web Forms and MVC and show why I think that the comparison favors Web Forms in many situations - as long as you don't fall into some classic Web Forms traps.

Web Forms

In the Web Forms model, your pages correspond directly to the page request from the browser. Thus, if you are directing a user to a list of Books, you'll likely have a page somewhere called "Booklist.aspx" to which you'll direct him. In that page, you'll have to provide everything needed to show that list. This includes code for pulling data, applying any business logic, and displaying the results. If there is any architectural or routing logic affecting the page, you'll have to code the architectural logic on the page as well. *Good* Web Forms development usually involves the development of a set of supporting classes in a separate (unit-testable) DLL. These class(es) will handle business logic, data access and architectural/routing decisions.

MVC

MVC takes a more "architectural" view of web application development: offering a standardized scaffold upon which to build. It also provides tools for automatically generating model, view and controller classes within the established architecture. For example, in both Ruby on Rails (just "Rails" from here on out) and ASP.NET MVC you'll always start out with a directory structure that reflects their overall model of web application architecture. To add a view, model and controller, you'll use a command like Rails's "Rails script/generate scaffold {modelname}" (ASP.NET MVC offers similar commands in the IDE). In the resulting controller class, there will be methods ("Actions") for Index (show list), Show, New and Edit and Destroy (at least in Rails, MVC is similar). By default, these "Get" Actions just bundle up the Model and route to a corresponding view/html file in the "View/{modelname}" directory (note that there are also Create, Update and Destroy actions that handle a "Post" and route back to Index or Show).

The layout of directories and files is significant in MVC. For example, in ASP.NET MVC, the *Index* method for a "Book" object will likely just have one line: "Return View();" Through the magic of MVC, this will send the Book model to the "/View/Books/Index.aspx" page where you'll find code to display Books. Rails's approach is similar although the logic is a bit more explicit and less "magic." A *View* page in an MVC app is usually simpler than a Web Forms page because they don't have to worry as much about routing, business logic or data handling.

Comparison

The advantages of MVC revolve around a clean separation of concerns and a cleaner, more HTML/CSS/AJAX/Javascript-centric model for producing your output. This enhances testability, provides a more standardized design and opens the door to a more "Web 2.0" type of web site.

However, there are some significant drawbacks as well.

First, while it is easy to get a demo site going, the overall architectural model has a significant learning curve. When they say "Convention Over Configuration" it sounds good - until you realize that you have a book's-worth of convention to learn. Furthermore, it is often a bit maddening to figure out what is going on because you *are* relying on magic rather than explicit calls. For example, that "Return View();" call above? The exact same call can be found in other Actions but *they go to different places.* If you understand the *MVC convention* then you know why this is done. However, it certainly doesn't qualify as an example of good naming or easily understandable code and it is much harder for new developers to pick up than Web Forms (this isn't just opinion: I had a summer intern learn Web Forms last year and MVC this year and the differences in productivity were pronounced - in favor of Web Forms). BTW, Rails is a bit better in this regard although Ruby on Rails features dynamically-named methods that take some serious getting-used-to as well.

Second, MVC implicitly assumes that you are building a classic CRUD-style web site. The architectural decisions and especially the code generators are all built to support this type of web application. If you are building a CRUD application and want to adopt a proven architecture (or simply dislike architecture design), then you should probably consider MVC. However, if you'll be doing more than CRUD and/or you are reasonably competent with architecture then MVC may feel like a straightjacket until you really master the underlying routing model (which is considerably more complex than simply routing in a WebForms app). Even then, I've felt like I was always fighting the model and worried about unexpected outcomes.

Third, if you don't care for Linq (either because you are afraid that Linq-to-SQL is going to disappear or because you find Linq-to-Entities laughably over-produced and under powered) then you also don't want to walk this path since ASP.NET MVC scaffolding tools are build around Linq (this was the killer for me). Rails's data model is also quite clumsy compared to what you can achieve if you are experienced in SQL (and especially if you are well-versed in TSQL and stored procedures!).

Fourth, MVC proponents often point out that MVC views are closer in spirit to the HTML/CSS/AJAX model of the web. For example, "HTML Helpers" - the little code calls in your vew page that swap in content and place it into HTML controls - are much easier to integrate with Javascript than Web Forms controls. However, ASP.NET 4.0 introduces the ability to name your controls and thus largely eliminates this advantage.

Fifth, MVC purists often deride Viewstate. In some cases, they are right to do so. However, Viewstate can also be a great tool and a boon to productivity. By way of comparison, handling Viewstate is *much* easier than trying to integrate third-party web controls in an MVC app. While control integration may get easier for MVC, all of the current efforts that I've seen suffer from the need to build (somewhat grody) code to link these controls back to the view's Controller class (that is - to *work around* the MVC model).

Conclusions

I like MVC development in many ways (although I prefer Rails to ASP.NET MVC by a long shot). I also think that it is important that we don't fall into the trap of thinking that ASP.NET MVC is an "anti-pattern" of ASP.NET Web Forms. They are different but not completely alien and certainly there is room for both.

However, I prefer Web Forms development because, *for most tasks*, it is simply easier to get things done (the exception being generation of a set of CRUD forms). MVC also seems to suffer, to some extent, from an *excess of theory.* Indeed, look at the many questions asked here on SO by people who know page-oriented ASP.NET but who are trying MVC. Without exception, there is much gnashing of teeth as developers find that they can't do basic tasks without jumping through hoops or enduring a huge learning curve. *This* is what makes Web Forms superior to MVC in my book: MVC makes you pay a *real world price* in order to gain a bit more testability or, worse yet, to simply be seen as *cool* because you are using the *latest technology.*

Update: I've been criticized heavily in the comments section - some of it quite fair. Thus, I have spent several months learning Rails and ASP.NET MVC just to make sure I wasn't really missing out on the next big thing! Of course, it also helps ensure that I provide a balanced and appropriate response to the question. You should know that the above response is a major rewrite of my initial answer in case the comments seem out of synch.

While I was looking more closely into MVC I thought, for a little while, that I'd end up with a major mea culpa. In the end I concluded that, while I think we need to spend a lot more energy on Web Forms architecture and testability, MVC really doesn't answer the call for me. So, a hearty "thank you" to the folks that provided intelligent critiques of my initial answer.

As to those who saw this as a *religious battle* and who relentlessly engineered downvote floods, I don't understand why you bother (20+ down-votes within seconds of one another on multiple occasions is certainly not normal). If you are reading this answer and wondering if there is something truly "wrong" about my answer given that the score is far lower than some of the other answers, rest assured that it says more

about a few people who disagree than the general sense of the community (overall, this one has been upvoted well over 100 times).

The fact is that many developers don't care for MVC and, indeed, this is not a minority view (even within MS as the blogs seem to indicate).

Share Follow edited May 23, 2017 at 12:17 community wiki
14 revs, 2 users 92%
Mark Brittingham

32 -1, The original question is good - saying you don't understand why something is good and asking for more information is awesome. This answer, though, is very strange - you're basically stating "I don't understand either", but wrapping it in a story. – orip Dec 24, 2008 at 16:39

49 I find it interesting that your criticism of ASP.NET MVC is that it adds abstraction and overhead. and therefore is more complicated. It is precisely the opposite. Webforms adds a layer of abstraction and MVC is far more straight forward and low-level that doesn't hide you from what you are doing – Trevor de Koekkoek Dec 25, 2008 at 14:24

75 Why is this marked as "The answer" when the poster didn't even know anything about the MVC pattern when he answered? – ScottKoon Apr 23, 2009 at 18:37

12 ScottKoon - agreed, not sure why this was accepted. Seems that all he did was agree with the OP. -1 – Jared Apr 23, 2009 at 19:05

11 I take issue with your characterization of WebForms as better fitting the web paradigm. WebForms is essentially the event-driven WinForms model overlaid on the web. As such the framework -- and we developers if, heaven forbid, we attempt to do something with non-MS client-side tools -- has to jump through a lot of hoops to pretend that you're doing event-handling over the web. MVC is a very natural fit for RESTful interfaces and REST attempts to put the web protocols to the use for which they were intended. MS designed WebForms the way they did, not because it's the best fit for the web, – tvanfosson Apr 28, 2009 at 2:45

117

MVC gives you more control over your output, and with that control comes greater risk of writing poorly designed HTML, tag soup, etc...

But at the same time, you have several new options you didn't have before...

1. More control over the page and the elements within the page
2. Less "junk" in your output, like the ViewState or excessively long IDs on elements *(don't get me wrong, I like the ViewState)*
3. Better ability to do client side programming with Javascript (Web 2.0 Applications anyone?)
4. Not just just MVC, but JsonResult is slick...

Now that's not to say that you can't do any of these things with WebForms, but MVC makes it easier.

I still use WebForms for when I need to quickly create a web application since I can take advantage of server controls, etc. WebForms hides all the details of input tags and submit buttons.

Both WebForms and MVC are capable of absolute garbage if you are careless. As always, careful planning and well thought out design will result in a quality application, regardless if it is MVC or WebForms.

[Update]

If it is any consolation as well, MVC is just a new, evolving technology from Microsoft. There has been many postings that WebForms will not only remain, but continue to be developed for...

http://haacked.com

http://www.misfitgeek.com

http://rachelappel.com

... and so on...

For those concerned about the route MVC is taking, I'd suggest giving "the guys" your feedback. They appear to be listening so far!

Share Follow

edited Jan 13, 2010 at 18:48

Dave Markle

97.6k ● 20 ● 151 ● 171

answered Dec 24, 2008 at 4:01

hugoware

36.4k ● 24 ● 62 ● 71

10 I think points 1 and 2 are the crux. Webforms as an abstraction just doesn't really "work" IMHO because it isn't an abstraction that maps well on top of what is really going on. I think ASP.NET MVC is a better fitting abstraction than webforms is given my limited experiecne with MVC. – Daniel Auger Dec 24, 2008 at 5:49

3 And a lot of people use ASP.Net without touching MVC or Webforms. I've seen lots of .Net applications that completely avoid the webforms model and just do everything in the code behind page. And frankly, I think it works better. – Kibbee Dec 24, 2008 at 14:37

Thanks for the update HBoss! I'd hate to see MS abandon WebForms after spending 7 years working on them. – Mark Brittingham Dec 24, 2008 at 14:47

1 Daniel - you say Webforms doesn't "work" because it doesn't map well to the web model. I respectfully disagree: http originated as a model for retrieving *documents*. The Webforms architecture simply extends the idea of documents so they are dynamic. This works for me... – Mark Brittingham Dec 24, 2008 at 14:51

3 @mdbritt. I respect your opinion because it is true on a high level of abstraction (retrieving a document). However, for me the psuedo statefulness and postback model is where it starts to break down, especially in highly dynamic scenarios. It works great for simple things, but it unravels quickly. – Daniel Auger Dec 24, 2008 at 15:54

76

Most of the objections to ASP.NET MVC seems centered around the views, which are one of the most "optional" and modular bits in the architecture. NVelocity, NHaml, Spark, XSLT and other view engines can be easily swapped out (and it's been getting easier with every release). Many of those have MUCH more concise syntax for doing presentation logic and formatting, while still giving complete control over the emitted HTML.

Beyond that, nearly every criticism seems to come down to the <% %> tagging in the default views and how "ugly" it is. That opinion is often rooted in being used to the WebForms approach, which just moves most of the classic ASP ugliness into the code-behind file.

Even without doing code-behinds "wrong", you have things like OnItemDataBound in Repeaters, which is *just as* aesthetically ugly, if only in a different way, than "tag soup". A foreach loop can be much easier to read, even with variable embedding in the output of that loop, particularly if you come to MVC from other non-ASP.NET technologies. It takes much less Google-fu to understand the foreach loop than to figure out that the way to modify that one field in your repeater is to mess with OnItemDataBound (and the rat's nest of checking if it's the right element to be changed.

The biggest problem with ASP tag-soup-driven "spaghetti" was more about shoving things like database connections right in between the HTML.

That it happened to do so using <% %> is just a correlation with the spaghetti nature of classic ASP, not causation. If you keep your view logic to HTML/CSS/Javascript and the minimal logic necessary to do *presentation*, the rest is syntax.

When comparing a given bit of functionality to WebForms, make sure to include all of the designer-generated C#, and the code-behind C# along with the .aspx code to be sure that the MVC solution is really not, in fact, much simpler.

When combined with judicious use of partial views for repeatable bits of presentation logic, it really can be nice and elegant.

Personally, I wish much of the early tutorial content focused more on this end of things than nearly exclusively on the test-driven, inversion of control, etc. While that other stuff is what the experts object to, guys in the trenches are more likely to object to the "tag soup".

Regardless, this is a platform that is still in beta. Despite that, it's getting WAY more deployment and non-Microsoft developers building actual stuff with it than most Microsoft-beta technology. As such, the buzz tends to make it seem like it's further along than the infrastructure around it (documentation, guidance patterns, etc) is. It being genuinely usable at this point just amplifies that effect.

Share Follow edited Jun 1, 2009 at 18:38 community wiki 2 revs, 2 users 82% J Wynia

1 I did not know you could easily swap in other view engines, can you provide further info on that? – D'Arcy Rittich Dec 25, 2008 at 2:15

I don't have a link handy, but Phil Haack did an article a couple of weeks ago on his site showing how you can even run them side by side. – J Wynia Dec 25, 2008 at 18:37

1 Amen! I hate using Findcontrol. I hate it so much. – Adam Lassek Jan 2, 2009 at 20:29

3 Excellent, well rounded post. – Owen May 25, 2009 at 21:17

59

```
<% if (Model.PreviousPost || Model.NextPost) { %>
    <div class="pager">
        <% if (Model.PreviousPost) { %>
            <span><% Html.ActionLink("<< " + Model.PreviousPost.Subject,
"view")); %></span>
        <% } if (Model.NextPost) { %>
            <span><% Html.ActionLink(Model.NextPost.Subject + " >>", "view"));
%></span>
        <% } %>
    </div>
<% } %>
```

You can make another post asking how to do this without including the embeded CSS.

NOTE: ViewData.Model becomes Model in the next release.

And with the aid of a user control this would become

```
<% Html.RenderPartial("Pager", Model.PagerData) %>
```

where PagerData is initialized via an anonymous constructor in the action handler.

edit: I'm curious what your WebForm implementation would look like for this problem.

Share Follow edited Dec 24, 2008 at 18:04 answered Dec 24, 2008 at 6:59

Todd Smith
17.3k ● 11 ● 62 ● 82

4 Good post. The OP is missing out on some of the techniques such as reuse via RenderPartial that would make his code cleaner. In the OP's defense he did hint that he felt he must be doing something wrong. – Daniel Auger Dec 24, 2008 at 15:59

25

I am not sure at what point people stopped caring about their code.

HTML is the most public display of your work, there are a LOT of developers out there who use notepad, notepad++, and other plain text editors to build a lot of websites.

MVC is about getting control back from web forms, working in a stateless environment, and implementing the Model View Controller design pattern without all the extra work that normally takes place in an implementation of this pattern.

If you want control, clean code, and to use MVC design patterns, this is for you, if you don't like working with markup, don't care about how malformed your markup gets, then use ASP.Net Web Forms.

If you don't like either, you are definitely going to be doing just about as much work in the markup.

EDIT I Should also state that Web Forms and MVC have their place, I was in no way stating that one was better than the other, only that each MVC has the strength of regaining control over the markup.

Share Follow

edited Dec 24, 2008 at 4:16

answered Dec 24, 2008 at 4:05

Tom Anderson
10.8k ● 3 ● 48 ● 64

6 I don't care about outputted markup (to a point), I care about maintainable code. The tradeoff here from Webforms is not worth it IMHO. – FlySwat Dec 24, 2008 at 4:08

1 To elaborate, I've never had an issue with good markup from Webforms, sure you have a ViewState field, but if you are careful with your design, you don't get broken HTML. – FlySwat Dec 24, 2008 at 4:08

2 When you see a 2mb Viewstate, have to do tons of control searching to find a control on a Web Form due to the naming malforming of the actual elements, something this light is welcomed. I have always been anal about my code, anyone can View Source it, and I have OCD and want my house clean. – Tom Anderson Dec 24, 2008 at 4:09

6 You only get a 2mb viewstate if you don't know what you are doing. – FlySwat Dec 24, 2008 at 4:10

3 You also get tag soup when you don't know what you're doing and throwing code together... – hugoware Dec 24, 2008 at 4:12

15

I think you are missing some things. First, there's no need for the Response.Write, you can use the `<%= %>` tags. Second, you can write your own HtmlHelper extensions to do common actions. Third, a little bit of formatting helps a lot. Fourth, all of this would probably be stuck in a user control to be shared between several different views and thus the overall mark up in the main view is cleaner.

I'll grant you that the mark up is still not as neat as one would like, but it could be cleaned up considerably through the use of some temporary variables.

Now, that's not so bad and it would be even better if I didn't have to format it for SO.

```
<%
   var PreviousPost = ViewData.Model.PreviousPost;
   var NextPost = ViewData.Model.NextPost;

   // Display the "Next and Previous" links
   if (PreviousPost != null || NextPost != null)
   {
 %>

<div>

       <%= PreviousPost == null
               ? string.Empty
               : Html.ActionLinkSpan("<< " + PreviousPost.Subject,
                                "view",
                                new { id = PreviousPost.Id },
                                new { style = "float: left;" } ) %>
         <%= NextPost == null
               ? string.Empty
               : Html.ActionLinkSpan( NextPost.Subject + " >>",
                                  "view",
                                   new { id = NextPost.Id },
                                   new { style = "float: right;" } ) %>

 <div style="clear: both;" />
 </div>

 <% } %>
```

Share Follow

edited Dec 24, 2008 at 4:38

answered Dec 24, 2008 at 4:32

tvanfosson

532k ● 102 ● 699 ● 798

2 Doesn't it still seem just so strange, considering I could have just set the visibility of a <asp:hyperlink> in Webforms? – FlySwat Dec 24, 2008 at 4:41

2 No, because even webforms wouldn't be that simple. You'd probably have to set some properties on the hyperlink in the codebehind because they're dynamic, you'd still need the DIV/span code, but you wouldn't be able to roll that into a method. And none of it would be testable. – tvanfosson Dec 24, 2008 at 5:12

3 I've done enough webforms that the pain of dealing with databound controls and making them do what I want client-side, coupled with the ability to increase the amount of tested code at least 2-fold has won me over. I've also done enough in Rails that this doesn't seem odd at all. – tvanfosson Dec 24, 2008 at 5:13

1 I'd have to set NavigateUrl, and Visibility. It would be 2 lines in the page_load event. – FlySwat Dec 24, 2008 at 5:13

2 I think its the fact that you were doing this in rails. The last time I did this was classic ASP, which has a lot of other reasons to hate it. Perhaps I just need to get over my initial gag reflex revulsion of <% %> tags. – FlySwat Dec 24, 2008 at 5:14

14

Please have a look at Rob Conery's post [I Spose I'll Just Say It: You Should Learn MVC]

Share Follow

edited Jan 19, 2015 at 18:10

JohnFx

34.9k ● 18 ● 107 ● 166

answered Apr 23, 2009 at 4:51

Shiju

1,313 ● 2 ● 12 ● 14

13

The big deal with MVC is that it's a conceptual framework that has been around a long time, and it has proven itself as a productive, powerful way to build both web applications and workstation applications that scale horizontally and vertically. It goes directly back to the Alto and Smalltalk. Microsoft is late to the party. What we have now with ASP.NET MVC is really primitive, because there's so much catching up to do; but damn, they're pouring out new releases fast and furiously.

What was the big deal with Ruby on Rails? Rails is MVC. Developers have been converting because, by word of mouth, it's become the way for programmers to be productive.

It's a huge deal; MVC and the implicit endorsement of jQuery are tipping points for Microsoft accepting that platform-neutral is critical. And what's neutral about it, is that unlike Web Forms, Microsoft can't lock you in conceptually. You can take all your C# code and reimplement in another language entirely (say PHP or java - you name it) because it's the MVC concept that's portable, not the code itself. (And think how huge it is that you can take your design and implement it as a workstation app with little code change, and no design change. Try that with Web Forms.)

Microsoft has decided that Web Forms will not be the next VB6.

Share Follow

edited Dec 24, 2008 at 4:53

answered Dec 24, 2008 at 4:46

dkretz

37.6k ● 13 ● 83 ● 140

1 Adding to that: There are a number of view-engines available that plug into MVC, including the default WebForms as well as a port of RoR's HAML (which bans angle brackets, especially when they include percent-signs). – yfeldblum Dec 24, 2008 at 5:02

Interesting way of stating it... Good points – hugoware Dec 24, 2008 at 15:23

HAML FTW, especially if your only objection to MVC is the <% %> 's – Peter Recore Jan 13, 2010 at 19:21

9

The two main advantages of the ASP.NET MVC framework over web forms are:

1. **Testability** - The UI and events in web forms are next to impossible to test. With ASP.NET MVC, unit testing controller actions and the views they render is easy. This comes with a cost in up-front development cost, but studies have shown that this pays off in the long run when it comes time to refactor and maintain the app.
2. **Better control over rendered HTML** - You state that you don't care about the rendered HTML because nobody looks at it. That's a valid complaint if that were the only reason to have properly formatted HTML. There are numerous reasons for wanting properly formatted HTML including: SEO, the ability to use id selectors more often (in css and javascript), smaller page footprints due to lack of viewstate and ridiculously long ids (ctl00_etcetcetc).

Now, these reasons don't really make ASP.NET MVC any better or worse than web forms in a black-and-white sort of way. ASP.NET MVC has its strengths and weaknesses, just like web forms. However, the majority of complains about ASP.NET MVC seem to stem from a lack of understanding on how to use it rather than actual flaws in the framework. The reason your code doesn't feel right or look right might be because you have several years of web forms experience under your belt and only 1-2 months of ASP.NET MVC experience.

The problem here isn't so much that ASP.NET MVC rocks or sucks, it's that it's new and there's very little agreement as to how to use it correctly. ASP.NET MVC offers much more fine-grained control over what's occurring in your app. That can make certain tasks easier or harder depending on how you approach them.

Share Follow

answered Dec 24, 2008 at 19:01

Kevin Pang
41.4k • 38 • 122 • 173

8

Hey, I've been struggling with switching to MVC as well. I am absolutely not a fan of classic ASP and MVC rendering reminds me a lot of those days. However, the more I use MVC, the more it grows on me. I am a webforms guy (as many are) and spent the past several years getting used to working with datagrids, etc. With MVC that is taken away. HTML Helper classes are the answer.

Just recently I spent 2 days trying to figure out the best way to add paging to a "grid" in MVC. Now, with webforms I could whip this out in no time. But I will say this... once I had the paging helper classes built for MVC, it became extremely simple to implement. To me, even easier than webforms.

That being said, I think that MVC will be much more developer friendly when there are a consistent set of HTML Helpers out there. I think we are going to start seeing a ton of HTML helper classes pop up on the web in the near future.

Share Follow

answered Dec 24, 2008 at 5:32

Papa Burgundy
6,467 ● 6 ● 44 ● 48

I would like to see your paging implementation because I have no idea how I'm going to tackle that one yet :) – dtc Dec 24, 2008 at 5:36

Here's where I got the paging helpers from. You can download the sample project here: blogs.taiga.nl/martijn/archive/2008/08/27/… – Papa Burgundy Dec 24, 2008 at 5:46

As with anything, there is a learning curve. You might be pleasantly surprised at how much better certain area's work. I won't lie though - some of the luxuries like Server Controls and ViewState are *certainly* missed. I've typed in <asp:TextB... and then realized... Whoops!! – hugoware Dec 24, 2008 at 13:21

Here is an honest question. If you need HTML "Helper" classes, haven't you really kind of violated the MVC model? Aren't you leaving behind the simplicity and elegance with which it is sold? If I'm wrong (and I may be!) please tell me why. – Mark Brittingham Dec 24, 2008 at 15:00

1 I don't think that you have to "switch" to MVC. I still use WebForms depending on the project. – hugoware Dec 24, 2008 at 15:21

It's funny because that's what I said when I first saw webforms.

8

Share Follow

answered Dec 24, 2008 at 15:49

greg
259 ● 1 ● 1

Seriously, it really was. WebForms, without knowing the context of the times that brought it to us, makes little sense. It doesn't embrace the web but tries to hide it, and it is a very poor abstraction. – Jason Bunting Jul 21, 2010 at 20:43

6

I'll admit that I don't get asp.net MVC yet. I'm trying to use it in a side project I'm doing but it's going pretty slow.

Besides not being able to do things that were so easy to do in web forms I notice the tag soup. It really does seem to take a step backwards from that perspective. I keep hoping that as I learn it will get better.

So far I've noticed that the top reason to use MVC is to gain full control over your HTML. I also read that asp.net MVC is able to serve up more pages faster than web forms and probably related to this, the individual page size is smaller than an average web forms page.

I really don't care what my HTML looks like as long as it works in the major browsers, but I do care how fast my pages load and how much bandwidth they take up.

Share Follow

answered Dec 24, 2008 at 5:25

dtc

10.3k ● 17 ● 82 ● 105

6

While I fully agree that that is ugly markup, I think using the ugly view syntax to write off ASP.NET MVC as a whole is not fair. The view syntax has gotten the least attention from Microsoft, and I am fully expecting something to be done about it soon.

Other answers have discussed the benefits of MVC as a whole, so I will focus on the view syntax:

The encouragement to use Html.ActionLink and other methods that generate HTML is a step in the wrong direction. This smacks of server controls, and, to me, is solving a problem that doesn't exist. If we are going to generate tags from code, then why bother using HTML at all? We can just use DOM or some other model and build up our content in the controller. Ok, that sounds bad, doesn't it? Oh yes, separation of concerns, that is why we have a view.

I think the correct direction is to make the view syntax as much like HTML as possible. Remember, a well designed MVC should not only give you separation of code from content, it should let you streamline your production by having people who are expert in layout work on the views (even though they do not know ASP.NET), and then later as a developer you can step in and make the view mockup actually dynamic. This can only be done if if the view syntax looks very much like HTML, so that the layout folks can use DreamWeaver or whatever the current popular layout tool is. You might be building dozens of sites at once, and need to scale in this way for efficiency of production. Let me give an example of how I could see the view "language" working:

```
<span mvc:if="ViewData.Model.ShowPrevious" style="float: left;">
    <a mvc:inner="ViewData.Model.PreviousPost.Subject"
href="view/{ViewData.Model.PreviousPost.Id}">sample previous subject</a>
</span>
<span mvc:if="ViewData.Model.ShowNext" style="float: left;">
    <a mvc:inner="ViewData.Model.NextPost.Subject"
href="view/{ViewData.Model.NextPost.Id}">sample next subject</a>
</span>
<div mvc:if="ViewData.Model.ShowNextOrPrevious" style="clear: both;" />
```

This has several advantages:

- looks better
- more concise
- no funky context switching betwen HTML and <% %> tags
- easy to understand keywords that are self-explanatory (even a non-programmer could do this - good for parallelization)
- as much logic moved back into controller (or model) as possible
- no generated HTML - again, this makes it very easy for someone to come in and know where to style something, without having to mess around with Html. methods
- the code has sample text in it that renders when you load the view as plain HTML in a browser (again, good for layout people)

So, what exactly does this syntax do?

mvc:inner="" - whatever is in the quotes gets evaluated and the inner HTML of the tag gets replaced with the resulting string. (Our sample text gets replaced)

mvc:outer="" - whatever is in the quotes gets evaluated and the outer HTML of the tag gets replaced with the resulting string. (Again, sample text gets replaced.)

{} - this is used for inserting output inside of attributes, similar to <%= %>

mvc:if="" - insde the qoutes is the boolean expression to be evaulated. The close of the if is where the HTML tag gets closed.

mvc:else

mcv:elseif="" - ...

mvc:foreach

answered Dec 24, 2008 at 13:08

D'Arcy Rittich
171k 41 297 285

1 You could fairly easily implement exactly what you're describing as your own view engine and ditch the WebForms solution entirely. – J Wynia Dec 24, 2008 at 15:58

1 i was going to make a note that there are other view engines available, and also that I think a cf-style markup would work well, which is what you are showing here. – Tracker1 Dec 24, 2008 at 21:47

Thanks for the info, was not aware there were other view engines. – D'Arcy Rittich Dec 25, 2008 at 2:16

Genshi is a popular Python templating engine with a similar approach, you could look at it for more inspiration: genshi.edgewall.org/wiki/... – orip Dec 25, 2008 at 7:44

But nobody liked XSLs so how do you expect this to get adopted? – BobbyShaftoe Dec 30, 2008 at 4:18

4

Now, I can only speak for myself here:

IMO, if you're a die-hard (anything) then conversion isn't for you. If you love WebForms that's because you can accomplish what you need to, and that's the goal of any too.

Webforms does a good job of abstracting HTML from the *developer*. If that's your goal, stick with Web Forms. You have all the wonderful "click and drag" functionality that has made desktop development what it is today. There are many included controls (plus a wealth of third party controls) that can bring about different functionality. You can drag a "grid" that's directly associated with a DataSource from your database; it comes with built in inline-editing, paging, etc.

As of right now, ASP.NET MVC is very limited in terms of third party controls. So again, if you like Rapid Application Development, where a lot of functionality is wired up for you, you should not try to get converted.

With all of this said, this is where ASP.NET shines: - TDD. Code is not testable, nuff said.

- Separation of concerns. That is the backbone of the MVC pattern. I am fully aware that you can accomplish this in Web Forms. However, I like imposed structure. It was too easy in Web Forms to mix design and logic. ASP.NET MVC makes it easier for different members of a team to work on different sections of the application.
- Coming from somewhere else: My background is CakePHP and Ruby on Rails. So, it is clear where my bias lies. It's simply about what you're comfortable with.

- Learning Curve: To expand on the last point; I hated the idea of "templating" to change the functionality of different elements. I didn't like the fact that a lot of the design was accomplished in the code behind file. It was one more thing to learn, when I was already intimately familiar with HTML and CSS. If I want to do something in an "element" on the page, I stick in a "div" or "span", slap an ID on it and off I go. In Web Forms I would have to go research how to do this.
- Current State of Web "Design": Javascript libraries like jQuery are becoming more commonplace. The way that Web Forms mangles your IDs just makes implementation (outside of Visual Studio) more difficult.
- More Separation (Design): Because a lot of the design is wired into your controls, it would be very difficult for an outside designer (without Web Forms) knowledge to work on a Web Forms project. Web Forms was built to be the end all and be all.

Again, much of these reasons stem from my unfamiliarity with Web Forms. A Web Forms project (if designed right) can have "most" of the benefits of ASP.NET MVC. But that's the caveat: "If designed right". And that's just something I don't know how to do in Web Forms.

If you're doing stellar work in Web Forms, you're efficient and it works for you, that's where you should stay.

Basically, do a quick review on both (try to find a unbiased source [good luck]) with a list of pros and cons, evaluate which one fit your goals and pick based on that.

Bottom line, pick the path of least resistance and most benefit to meet your goals. Web Forms is a very mature framework and will only get better in the future. ASP.NET MVC is simply another alternative (to draw in Ruby on Rails and CakePHP developers such as myself :P)

answered Sep 5, 2009 at 18:02

Kevin

1,743 ● 2 ● 17 ● 16

3

Java EE's JSPs looked like this when they were first proposed - ugly scriptlet code.

Then they offered up tag libraries to make them more HTML tag-ish. The problem was that anybody could write a tag library. Some of the results were disastrous, because people embedded a lot of logic (and even style) into tag libraries that generated HTML.

I think the best solution is the JSP Standard Tag Library (JSTL). It's "standard", HTML tag-ish, and helps prevent people from putting logic into pages.

An added benefit is that it preserves that line of demarcation between web designers and developers. The good sites that I see are designed by people with an aesthetic sense and designing for usability. They lay out pages and CSS and pass them on to developers, who add in the dynamic data bits. Speaking as a developer who lacks these gifts, I think we give something important away when we ask developers to write web pages from soup to nuts. Flex and Silverlight will suffer from the same problem, because it's unlikely that designers will know JavaScript and AJAX well.

If .NET had a path similar to JSTL, I'd advise that they look into it.

Share Follow

answered Dec 24, 2008 at 16:16

duffymo

308k ● 46 ● 374 ● 565

> +1 - more if I could give it. Yeah... Java has been down this road long ago. The weird part is that Java has also seen some MVC style frameworks that bolt on components as well - like JSF and Tapestry. MVC is about the only sane architecture for a web application IMHO. I wouldn't let beef with the framework prevent you from getting a deeper understanding of it. The framework will evolve. – cwash Oct 31, 2009 at 1:17

3

Just thought I'd share how this sample looks with the shiny new Razor view engine which is default since ASP .NET MVC 3.

```
@{
    var prevPost = ViewData.Model.PreviousPost;
    var nextPost = ViewData.Model.NextPost;
}

@if (prevPost != null || nextPost != null) {
    <div>
        @if (prevPost != null) {
            <span style="float: left;">
                @Html.ActionLink("<< " + prevPost.Subject, "view", new { id =
prevPost.Id })
            </span>
        }
        @if (nextPost != null) {
            <span style="float: left;">
                @Html.ActionLink(nextPost.Subject + " >>", "view", new { id =
nextPost.Id })
            </span>
        }
        <div style="clear: both;" />
    </div>
}
```

Any problem with that?
Also, you shouldn't actually inline your CSS styles, should you? And why do you

check for nullity in three places instead of just two? An extra `div` rarely hurts. This is how I'd do it:

```
<div>
    @if (prevPost != null) {
        @Html.ActionLink("<< " + prevPost.Subject, "view",
            new { id = prevPost.Id, @class = "prev-link" })
    }
    @if (nextPost != null) {
        @Html.ActionLink(nextPost.Subject + " >>", "view",
            new { id = nextPost.Id, @class = "next-link" })
    }
    <div class="clear" />
</div>
```

Share Follow

edited Jul 28, 2011 at 9:22

answered Jul 28, 2011 at 9:17

Dan Abramov

268k ● 86 ● 416 ● 517

2

I can't speak directly to the ASP.NET MVC project, but generally speaking MVC frameworks have come to dominate **web** application development because

1. They offer a formal separation between Database Operations,"Business Logic", and Presentation
2. They offer enough flexibility in the view to allow developers to tweak their HTML/CSS/Javascript to work in multiple browsers, **and future versions of those browsers**

It's this last bit that's the important one. With Webforms, and similar technologies, you're relying on your vendor to write your HTML/CSS/Javascript for you. It's powerful stuff, but you have no guarantee that the current version of Webforms is going to work with the next generation of web browsers.

That's the power of the view. You get full control over your application's HTML. The downside is, **you** need to be disciplined enough to keep the logic in your views to a minimum, and keep the template code as simple as you possibly can.

So, that's the trade-off. If Webforms are working for you and MVC isn't, then **keep using Webforms**

Share Follow

answered Dec 25, 2008 at 0:37

Alana Storm

166k ● 95 ● 415 ● 618

1 "you're relying on your vendor to write your HTML/CSS/Javascript for you" Thats nonsense. Does everyone use the prebuilt controls? We never have. – FlySwat Dec 25, 2008 at 1:38

1 Point 1 is nonsense as well. Every app I've architected has been strongly separated, with just the code behind being binding logic for the view. The event handlers in teh codebehind simply called the appropriate methods in the business layer. – FlySwat Dec 25, 2008 at 1:39

1 Like I said Jon, if Webforms is working for you and your shop has the time to develop their own controls, then keep doing it. – Alana Storm Dec 25, 2008 at 2:19

1 @FlySwat - you've NEVER used a DropDownList or DataGrid ? – Simon_Weaver Jan 20, 2009 at 4:51

2

Most of my frustration with it is just not knowing how to do things "properly". Since it was released as a preview we've all had a bit of time to look at things, but they've been changing quite a bit. At first I was really frustrated but the framework seems workable enough to enable me to create extremely complex UI's (read: Javascript) pretty quickly. I understand that you can do this with Webforms as I was doing it before but it was a huge pain in the butt trying to get everything to render correctly. A lot of times I would end up using a repeater to control the exact output and would end up with a lot of the spaghetti mess that you have above as well.

In an effort to not have the same mess, I've been using a combination of having domain, service layers, and a dto to transfer to the view. Put that together with spark, a view engine and it gets really nice. It takes a bit to setup but once you get it going, I've seen a big step up in the complexity of my applications visually, but its so stinking simple to do code wise.

I wouldn't probably do it exactly like this but here's your example:

```
<if condition="myDtp.ShowPager">
  <div>
    <first />
    <last />
    <div style="clear: both;" />
  </div>
</if>
```

That's pretty maintainable, testable, and tastes yummy in my code soup.

The takeaway is that clean code is really possible, but it's a big ass shift from the way we were doing things. I don't think everyone groks it yet though. I know I'm still figuring it out...

Share Follow

answered Jan 26, 2009 at 22:11

rball
6,955 ● 7 ● 50 ● 78

2

Steve Sanderson's recently published book 'Pro ASP.NET MVC' [1] [2] from Apress suggests another alternative -- creating a custom HtmlHelper extension. His sample (from Chapter 4 on page 110) uses the example of a paged list, but it can easily be adapted for your situation.

```
public static string PostNavigator(this HtmlHelper html, Post previous, Post
next, Func<int, string> pageUrl)
{
    StringBuilder result = new StringBuilder();

    if (previous != null || next != null)
    {
        result.Append("<div>");

        if (previous != null)
        {
            result.Append(@"<span class=""left"">");
            TagBuilder link = new TagBuilder("a");
            link.MergeAttribute("href", pageUrl(previous.Id));
            link.InnerHtml = String.Format("<< {0}",
html.Encode(previous.Subject));
            result.Append(link.ToString());
            result.Append("</span>");
        }

        if (next != null)
        {
            if (previous != null)
            {
                result.Append(" ");
            }

            result.Append(@"<span class=""right"">");
            TagBuilder link = new TagBuilder("a");
            link.MergeAttribute("href", pageUrl(next.Id));
            link.InnerHtml = String.Format("{0} >>",
html.Encode(next.Subject));
            result.Append(link.ToString());
            result.Append("</span>");
        }

        result.Append("</div>");
    }

    return result.ToString();
}
```

You would call it in your view with code something like this:

```
<%= Html.PostNavigator(ViewData.Model.PreviousPost, ViewData.Model.NextPost, id
=> Url.Action("View", new { postId = id })) %>
```

[1] http://blog.codeville.net/2009/04/29/now-published-pro-aspnet-mvc-framework-apress/

[2] http://books.google.com/books?id=Xb3a1xTSfZgC

Share Follow

edited May 4, 2009 at 23:09

answered May 4, 2009 at 23:02

GuyIncognito

2 Wow, thats horrible markup in code...Bleck. – FlySwat May 4, 2009 at 23:13

If a 'PostNavigator' is a reusable widget (or WebControl if you will) with markup that doesn't change, and that is styled by CSS rules -- how is that such a horrible solution? – GuyIncognito May 5, 2009 at 13:34

@GuyIncognito: Agreed, this seems analogous to a WebControl and a clean solution. At some point, you have to generate a bunch of HTML strings. An extension method like this is a good solution. – gbc Oct 6, 2009 at 1:04

1

I was hoping to see a post from Phil Haack, but it wasnt here so I just cut and paste it from http://blog.wekeroad.com/blog/i-spose-ill-just-say-it-you-should-learn-mvc/ in the comments section

> Haacked - April 23, 2009 - Rob, you're a riot! :) I do find it funny when people write spaghetti code in MVC and then say "look! Spaghetti!" Hey, I can write spaghetti code in Web Forms too! I can write it in rails, PHP, Java, Javascript, but not Lisp. But only because I can't yet write anything in Lisp. And when I do write spaghetti code I don't look at my plate glumly expecting to see macaroni. The point people often make when comparing it to classic ASP is that with classic ASP people tended to mix concerns. Pages would have view logic with user input handling mixed in with model code and business logic all mixed up in one. That's what the spaghetti was about! Mixing concerns all in one big mess. With ASP.NET MVC, if you follow the pattern, you're less likely to do it. Yeah, you still might have a bit of code in your view, but hopefully that's all view code. The pattern encourages you to not put your user interaction code in there. Put it in the controller. Put your model code in a model class. There. No spaghetti. It's O-Toro Sushi now. :)

Share Follow

answered Jan 20, 2010 at 17:02

Sameer Alibhai

3,158 ● 4 ● 36 ● 36

1

Me too; I would spend my time on Silverlight rather than ASP.NET MVC. I have tried MVC 1.0 and have had a look at the latest 2.0 Beta 1 release a few day ago.

I (can)'t feel that how ASP.NET MVC is better than webform. The selling point of MVC are:

1. Unit (test)
2. Separate the design (view) and logic (controller + model)
3. No viewstate and better element id management
4. RESTful URL and more ...

But webform, by using code-behind.Theme, Skin, and Resource are already perfect to separate the design and logic.

Viewstate: client id management is coming on ASP.NET 4.0. I am only concerned about unit tests, but unit tests are not enough being a reason to turn me to ASP.NET MVC from webform.

Maybe I can say: ASP.NET MVC is not bad, but webform is already enough.

Share Follow

edited Aug 9, 2017 at 18:42

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Aug 8, 2009 at 4:00

Cheung

15.5k ● 21 ● 66 ● 95

-1

I've been using MVC for since Preview 3 came out and while it is still has it's growing pains it helps quite a bit in the area of team development. Try having three people work on a single webforms page and then you'll understand the concept of banging your head on the wall. I can work with a developer who understands the design elements on the view page and our resident Linq god in making the model work while I put everything together in the controller. I've never been able to develop in a realm where the separation of concerns was so easy to achieve.

Biggest selling point of ASP.Net MVC - StackOverflow runs on the MVC stack.

That being said... your code is so much prettier than what I normally do with the view page. Also, one of the guys I work with is working on wrapping the HTML helper into a tag library. Instead of <%=Html.RandomFunctionHere() %> it works like

< hh:randomfunction / >

I just hope the MVC team gets around to it at some point because I'm sure they'd do a better job of it.

Share Follow edited Dec 28, 2008 at 4:35 answered Dec 24, 2008 at 6:53

thaBadDawg
5,220 ● 6 ● 36 ● 44

1 "...one of the guys I work with is working on wrapping the HTML helper into a tag library. Instead of <%=Html.RandomFunctionHere() %> it works like <hh:randomfunction />" That's just it - a call to the method "RandomFunctionHere()" is just that - a method call. It is not markup, and abstracting that fact into something that *looks* like a tag seems to me to be missing the point. If I am a developer looking at that code, I would rather see it as a method than some custom tag... – Jason Bunting Apr 24, 2009 at 16:27

-2

Use a Facade Pattern to wrap all the logic for all the data you need to display and then use it as your generic in your view and then.... no more spaghetti code. http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

If you need more help cgardner2020@gmail.com

Share Follow answered Dec 25, 2008 at 11:27

Charles Gardner
32 ● 3

-17

The implementation ASP.NET MVC is horrible. The product plain sucks. I've seen several demos of it and I'm ashamed of MSFT... I'm sure the guys who wrote it are smarter than me, but it's almost as if they don't know what the Model-View-Controller is.

The only people I know who are trying to implement MVC are people who like to try new things from MSFT. In the demos I've seen, the presenter had an apologetic tone...

I'm a big fan of MSFT, but I have to say that I see no reason to bother with their implementation of MVC. Either use Web Forms or use jquery for web development, don't choose this abomination.

EDIT:

The intent of the Model-View-Controller architectural design pattern is to separate your domain from the presentation from the business rules. I've used the pattern successfully in every Web Forms project I've implemented. The ASP.NET MVC product does not lend itself well to the actual architectural design pattern.

Share Follow

edited Aug 30, 2012 at 18:41

Buu

50.3k ● 5 ● 69 ● 85

answered Dec 24, 2008 at 3:57

mson

7,824 ● 6 ● 42 ● 70

3 Depending on the site, MVC is a very powerful tool out of the box without a lot of extra legwork. For me, I am using just to get control BACK of my markup, the abomination that web forms can do to a simple website's markup is just... crazy. – Tom Anderson Dec 24, 2008 at 4:02

I'd wager to say that MVC is actually quite good since it is being fit into the mold of ASP.NET, which is geared to favor WebForms... – hugoware Dec 24, 2008 at 4:03

@tom - if you have to preface a positive comment about something with... Depending on the site... you are already standing on thin ice. I'll agree that if the requirement is to develop a site using ASP.NET MVC, that it may be a good choice, but for anything else, it looks like a bad choice. – mson Dec 24, 2008 at 4:07

4 I like, and prefer, chocolate to vanilla. Anyone want to argue with me about it? – Jason Bunting Apr 24, 2009 at 16:25

4 @Jason CHOCOLATE IS HORRIBLE. The product just plain SUCKS.... yru downvoting me? – user1228 Jul 16, 2010 at 20:24