ASP.NET Concepts

Top 10 ASP.NET Interview Questions and Answers

*"ASP.NET is a **web application development** framework for building web sites and web applications that follows object oriented programming approach"*

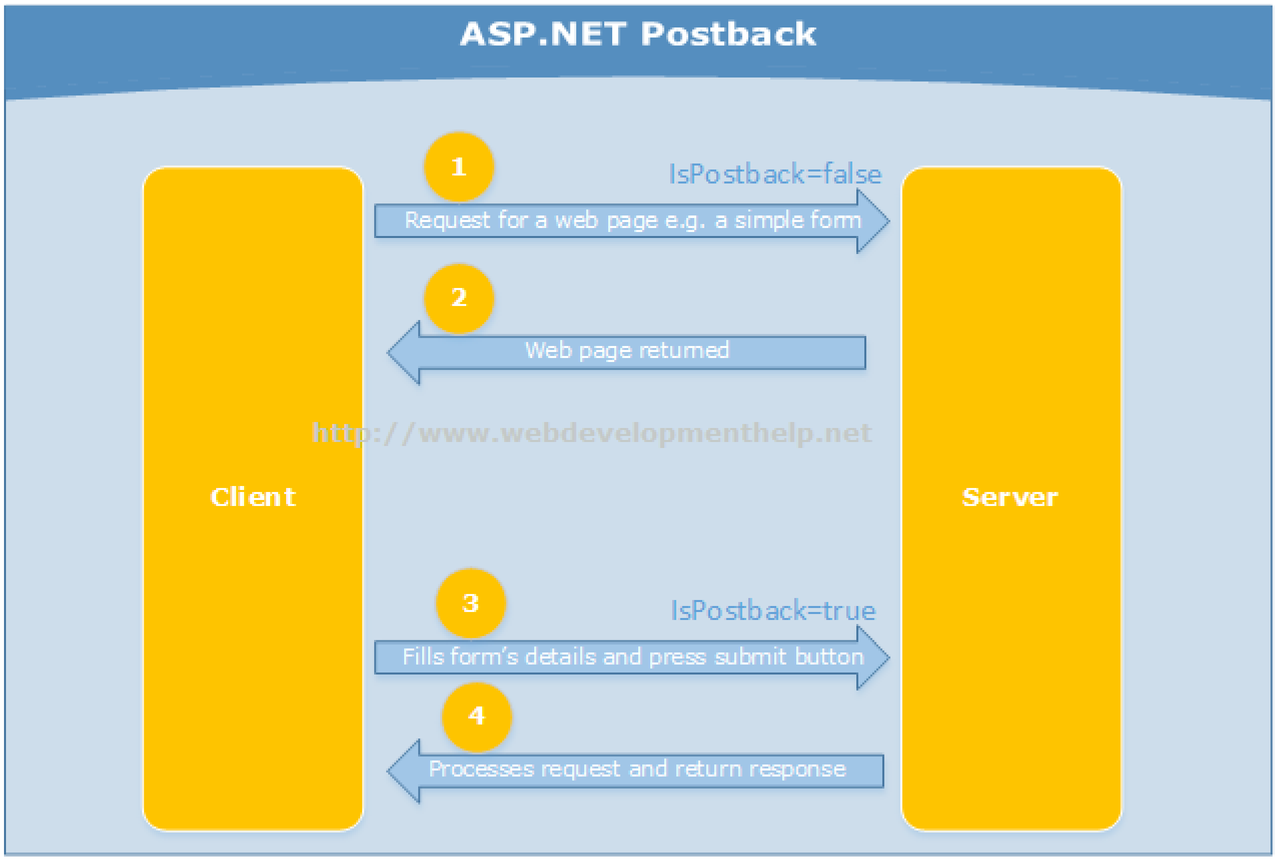
1. What is the concept of Postback in ASP.NET?

A postback is a request sent from a client to server from the same page user is already working with.
ASP.NET was introduced with a mechanism to post an HTTP POST request back to the same page. It's basically posting a complete page back to server (i.e. sending all of its data) on same page. So, the whole page is refreshed.

In below diagram, when first request is made from client to server for a web page (say a simple registration form), *IsPostback* property value will be *false*. It will be a GET request that may be initiated by:

- typing a web page URL in a browser window or
- using JavaScript ***window.open*** method
- or, clicking a hyperlink on a webpage page.

After user filled the returned form and presses a button (say submit button, or use JavaScript to submit form), an HTTP Post request is made to server on same page with data as shown in point 3. This time *IsPostback* property value will be *true*. Server processes the request and returns a response back to client.


ASP.NET Postback
1
IsPostback=false
Request for a web page e.g. a simple form
2
Web page returned
http://www.webdevelopmenthelp.net
Client
Server
3
IsPostback=true
Fills form's details and press submit button
4
Processes request and return response

Another concept related to this approach is "Callback" that is also asked sometimes during a technical interview question. Click here to understand Postback Vs Callback in ASP.NET.

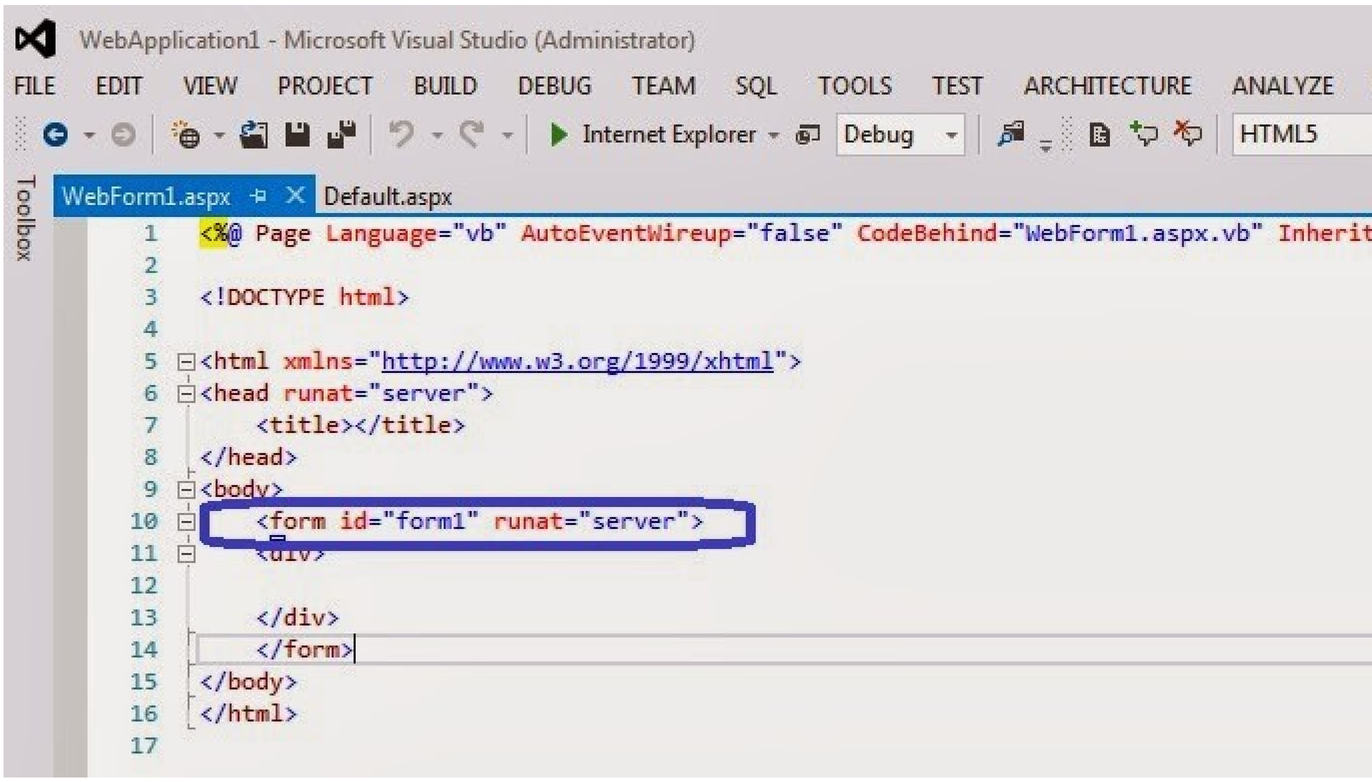
Difference between a Postback and a Callback in ASP.NET

By
*"A **postback** is a request sent from a client to server from the same page, user is already working with."*

ASP.NET was introduced with a mechanism to post an HTTP POST request back to the same page. It's basically posting a complete page back to server (i.e. sending all of its data) on same page. So, the whole page is refreshed.In order to understand how this postback mechanism works in ASP.NET, follow the simple steps:

- Add a new ASP.NET web form page to a project e.g. WebForm1.aspx.

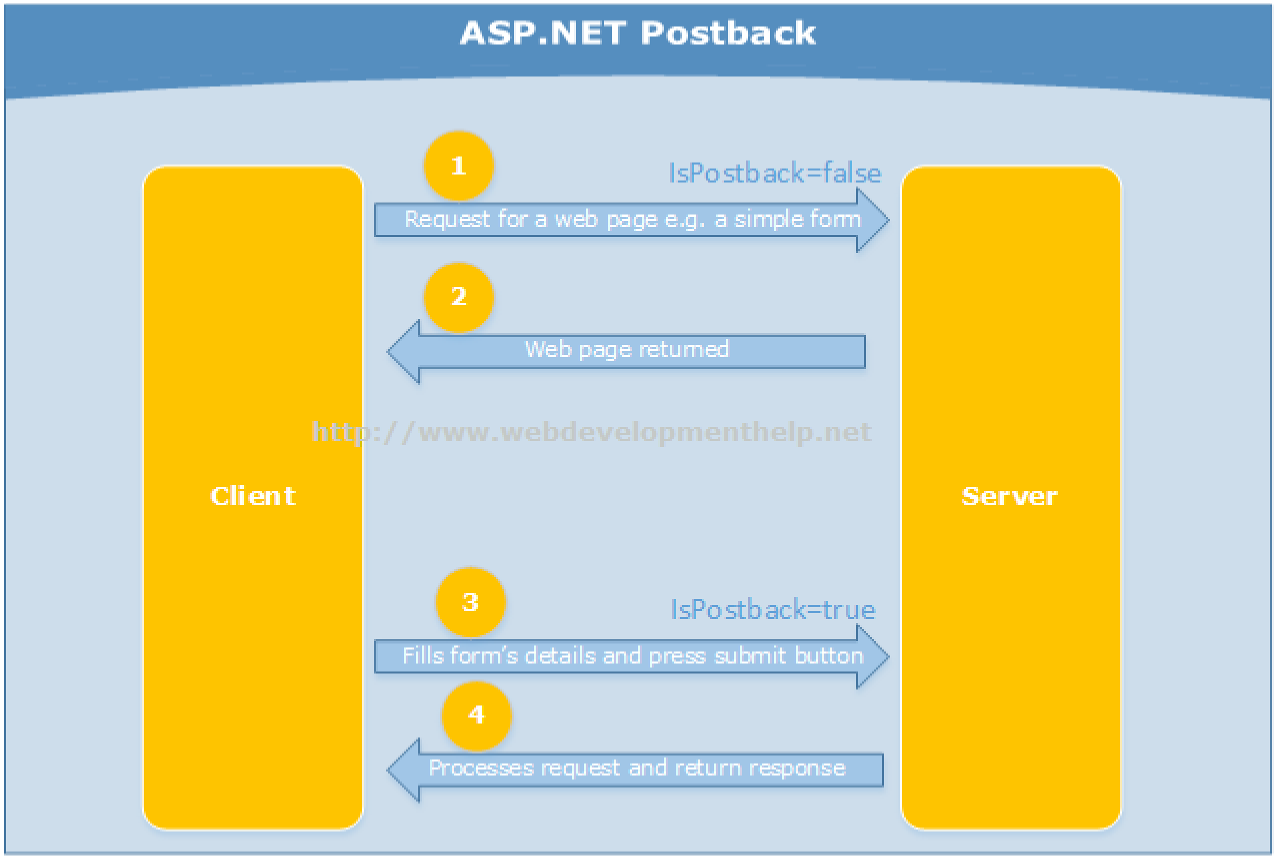
- View the page code in HTML Source view. You will find something like following screen.


WebApplication1 - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE
Internet Explorer
Debug
HTML5
Toolbox
WebForm1.aspx
Default.aspx

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="WebForm1.aspx.vb" Inherit

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>

```

- Look the form line of code.
  <form id="form1" runat="server">
  It represents a server-side implementation of form control.
- Now just run the application to see WebForm1.aspx page and view its source code. HTML source of the page will display form element as follows:
  <form method="post" action="WebForm1.aspx" id="form1">

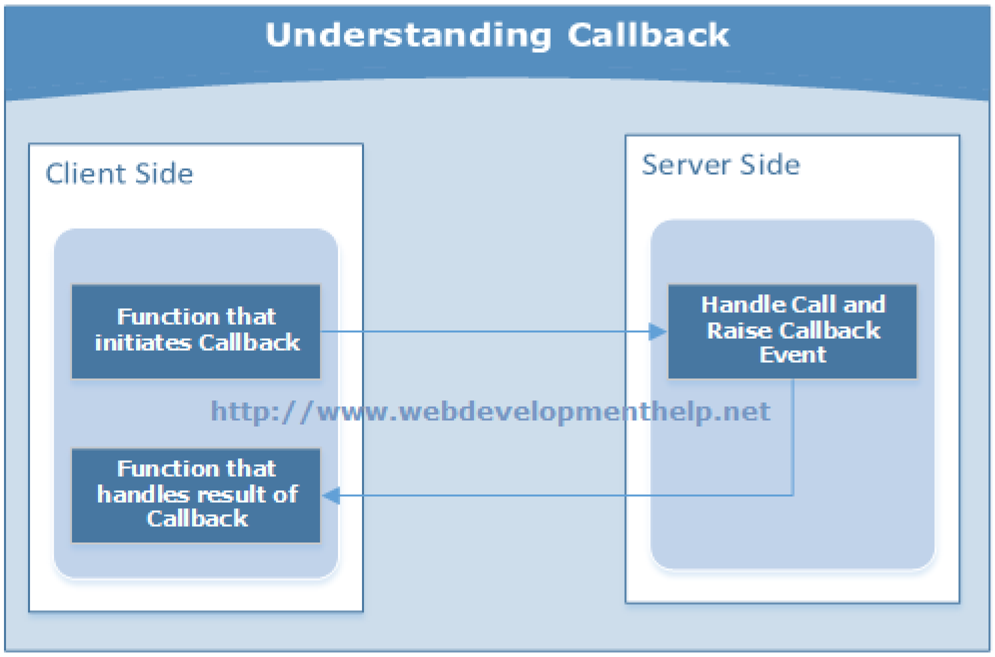
  You can see that an HTML form element generated with an HTTP method as "POST" and action="WebForm1.aspx". So, if a submit button is clicked, the page will postback to itself by default.Following figure will also give you more understanding on ASP.NET Postback.


ASP.NET Postback
1
IsPostback=false
Request for a web page e.g. a simple form
2
Web page returned
http://www.webdevelopmenthelp.net
Client
Server
3
IsPostback=true
Fills form's details and press submit button
4
Processes request and return response

More on *IsPostBack* property here.

Series of ASP.NET Interview Questions and Answers

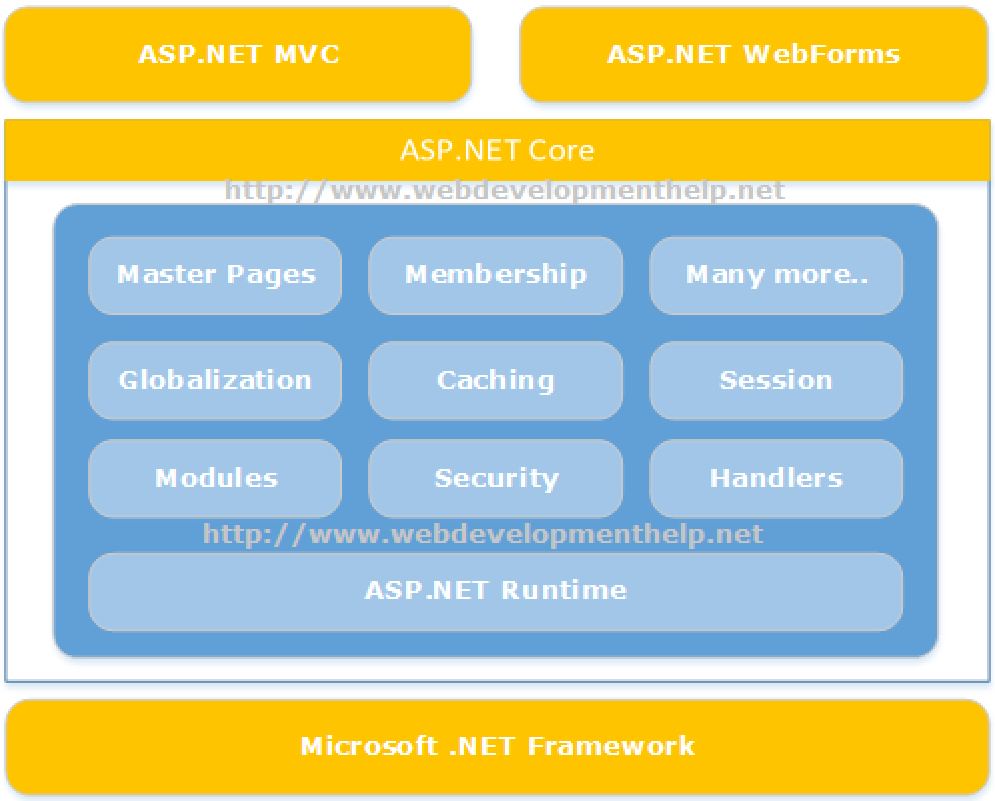
*"A **callback** is generally a call for execution of a function after another function has completed."*


Understanding Callback
Client Side
Server Side
Function that initiates Callback
Handle Call and Raise Callback Event
http://www.webdevelopmenthelp.net
Function that handles result of Callback

But if we try to differentiate it from a postback then we can say: It's a call made to the server to receive specific data instead of whole page refresh like a postback. In ASP.NET, its achieved using AJAX, that makes a call to server and updating a part of the page with specific data received.

2. Difference between ASP.NET WebForms and ASP.NET MVC?

ASP.NET Web Forms uses Page controller pattern approach for rendering layout. In this approach, every page has it's own controller i.e. code-behind file that processes the request. On the other hand, ASP.NET MVC uses Front Controller approach. In this approach a common controller for all pages, processes the requests.



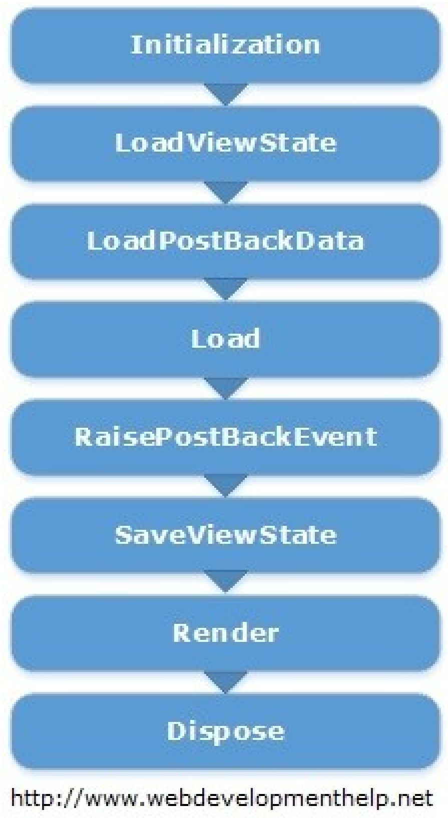
Please follow for detailed information on WebForms Vs MVC.

3. Please briefly explain ASP.NET Page life Cycle?

ASP.NET page passes through a series of steps during its life cycle. Following is the high-level explanation of life cycle stages/steps.

Initialization: Controls raise their Init event in this stage.Objects and variables are initializes for complete lifecycle of request.
LoadViewState: is a post back stage and loads the view state for the controls that enabled

its view state property.



LoadPostBackData: is also a post back stage and loads the data posted for the controls and update them.
Load: In this stage page as well as all the controls raise their Load event. Till this stage all the controls are initialized and loaded. In most of the cases, we are coding this event handler.
RaisePostBackEvent: is again a postback stage. For example, it's raise against a button click event. We can easily put our code here to perform certain actions.
SaveViewState: Finally, controls state is saved in this stage before Rendering HTML.
Render: This is the stage where HTML is generated for the page.
Dispose: Lastly, all objects associated with the request are cleaned up.

Related Articles

- Free Online Tests (ASP.NET | ASP.NET MVC | AJAX | HTML5 | jQuery)
- Postback Vs Callback
- WebResource.axd and ScriptResource.axd in ASP.NET
- ASP.NET WebForms Vs ASP.NET MVC
- Online Practice Exam: 70-486

For very detailed explanation of Page Life Cycle is explained here.

4. What is the difference between custom controls and user controls?

Custom controls are basically compiled code i.e. DLLs. These can be easily added to toolbox, so it can be easily used across multiple projects using drag and drop approach. These controls are comparatively hard to create.
But User Controls (.ascx) are just like pages (.aspx). These are comparatively easy to create but tightly couple with respect to User Interface and code. In order to use across multiple projects, we need to copy and paste to the other project as well.

A simple ASP.NET Interview Question
Please mark what’s true about Custom Controls in ASP.NET [Choose Three]?

- A. Compiled Code (i.e. DLL)
- B. Tightly couple with respect to user interface and code.
- C. A bit hard to create.
- D. Can be easily used across multiple projects.

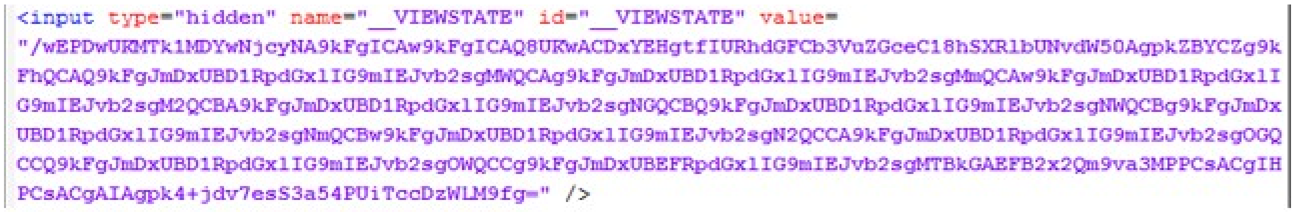
For a complete ASP.NET online test and Practice Exams, Click Here.

Correct Answer: A, C, D

5. What is the concept of view state in ASP.NET?

As in earlier question, we understood the concept of postback. So, in order to maintain the state between postbacks, ASP.NET provides a mechanism called view state. Hidden form fields are used to store the state of objects on client side and returned back to server in subsequent request (as postback occurs).

If we see the view source for an ASP.NET web page, we will find that hidden form field with Id = “__VIEWSTATE” something like the following:



```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value=
"/wEPDwUKMTk1MDYwNjcyNA9kFgICAw9kFgICAQ8UKwACDxYEHgtfIURhdGFCb3VuZGceC18hSXRlbUNvdW50AgpkZBYCZg9k
FhQCAQ9kFgJmDxUBD1RpdGxlIG9mIEJvb2sgMWQCAg9kFgJmDxUBD1RpdGxlIG9mIEJvb2sgMmQCAw9kFgJmDxUBD1RpdGxlI
G9mIEJvb2sgM2QCBA9kFgJmDxUBD1RpdGxlIG9mIEJvb2sgNGQCBQ9kFgJmDxUBD1RpdGxlIG9mIEJvb2sgNWQCBg9kFgJmDx
UBD1RpdGxlIG9mIEJvb2sgNmQCBw9kFgJmDxUBD1RpdGxlIG9mIEJvb2sgN2QCCA9kFgJmDxUBD1RpdGxlIG9mIEJvb2sgOGQ
CCQ9kFgJmDxUBD1RpdGxlIG9mIEJvb2sgOWQCCg9kFgJmDxUBEFRpdGxlIG9mIEJvb2sgMTBkGAEFB2x2Qm9va3MPPCsACgIH
PCsACgAIAgpk4+jdv7esS3a54PUiTccDzWLM9fg=" />
```

6. Difference between Response.Redirect and Server.Transfer?

In case of Response.Redirect, a new request is generated from client-side for redirected page. It’s a kind of additional round trip. As new request is generated from client, so the new URL is visible to user in browser after redirection.
While in case of Server.Transfer, a request is transferred from one page to another without making a round trip from client. For the end user, URL remains the same in browser even

after transferring to another page.

*We have just released a complete article series on **[Building an ASP.NET MVC Shopping Cart]** using **MVC**, **C#**, **Entity Framework** and **SQL Server**. Keep in touch to get the knowledge that how to create Online Shopping Store in ASP.NET MVC?, along with complete **[Source Code]**.*

7. Please briefly explain the usage of Global.asax?

Global.asax is basically ASP.NET Application file. It's a place to write code for Application-level events such as Application start, Application end, Session start and end, Application error etc. raised by ASP.NET or by HTTP Modules.

There is a good list of events that are fired but following are few of the important events in Global.asax:

- *Application_Init* occurs in case of application initialization for the very first time.
- *Application_Start* fires on application start.
- *Session_Start* fires when a new user session starts
- *Application_Error* occurs in case of an unhandled exception generated from application.
- *Session_End* fires when user session ends.
- *Application_End* fires when application ends or time out.

8. What are the different types of Validation controls in ASP.NET?

In order to validate user input, ASP.NET provides validation server controls. All validation controls inherits from BaseValidator class which contains the common validation properties and methods like ControlToValidate, Enabled, IsValid, EnableClientScript, ValidationGroup,Validate() etc.

ASP.Net provides a range of validation controls:

- *RequiredFieldValidator* validates compulsory/required input.
- *RangeValidator* validates the range. Validates that input falls between the given range values.
- *CompareValidator* validates or compares the input of a control with another control value or with a fixed value.
- *RegularExpressionValidator* validates input value against a defined regular expression pattern.
- *CustomValidator* allows to customize the validation logic with respect to our application logic.
- *ValidationSummary* displays all errors on page collectively.

9. What are the types of Authentication in ASP.NET?

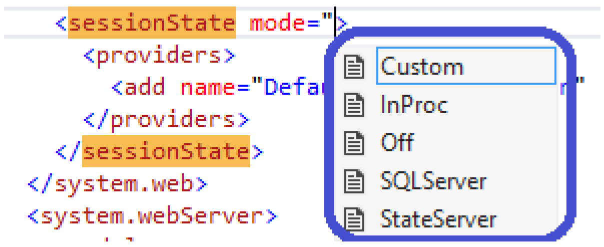
There are three types of authentication available in ASP.NET:

- *Windows Authentication:* This authentication method uses built-in windows security features to authenticate user.
- *Forms Authentication:* authenticate against a customized list of users or users in a database.
- *Passport Authentication:* validates against Microsoft Passport service which is basically a centralized authentication service.

10. What are Session state modes in ASP.NET?

ASP.NET supports different session state storage options:

- **In-Process** is the default approach. It stores session state locally on same web server memory where the application is running.
- **StateServer** mode stores session state in a process other than the one where application is running. Naturally, it has added advantages that session state is accessible from multiple web servers in a Web Farm and also session state will remain preserved even web application is restarted.
- **SQLServer** mode stores session state in SQL Server database. It has the same advantages as that of StateServer.
- **Custom** modes allows to define our custom storage provider.
- **Off** mode disables session storage.



```
<sessionState mode="">
  <providers>
    <add name="Defa
  </providers>
</sessionState>
</system.web>
<system.webServer>
```

I strongly believe that every asp.net web developer should prepare and understand the above discussed interview questions. You can follow here for Comprehensive list of ASP.NET Interview Questions.

ASP.NET Interview Questions for Beginners and Professionals – Part 1

By IMRAN ABDUL GHANI | March 30, 2014

0 Comments

This ASP.NET Tutorial is an extension to my previous tutorial "Top 10 ASP.NET Interview Questions and Answers". In previous tutorial, focus was to present you with the most important and top ASP.NET Interview Questions that are normally asked during an ASP.NET developer Interview. Here in this article, I'll try to further extend those important questions as well as add more important questions.

Basically, this is how an interviewer normally does? Interviewer asked a question about a technical concept at high level. If he gets a right answer, he further goes into details related to that particular concept and its implementation details.

For example, in previous article, we asked about the concept of *View State* in ASP.NET but in this tutorial, we will further explore the *View State* concept with more questions. But we will not repeat the questions already presented in previous post, so it's highly recommended to go through that ASP.NET Interview Questions tutorial first.

*For a comprehensive list of **ASP.NET MVC Interview Questions**, follow here.*

You can follow other Parts of this ASP.NET Interview Questions Series as:

- *ASP.NET Interview Questions – Part 1 (Controls, State Management etc.)*
- *ASP.NET Interview Questions – Part 2 (Controls, Globalization & Localization etc.)*
- *ASP.NET Interview Questions – Part 3 (Caching & Security etc.)*
- *ASP.NET Interview Questions – Part 4 (Security)*
- *ASP.NET Interview Questions – Part 5 (ASP.NET AJAX)*
- *ASP.NET Interview Questions – Part 6 (ASP.NET Web API)*

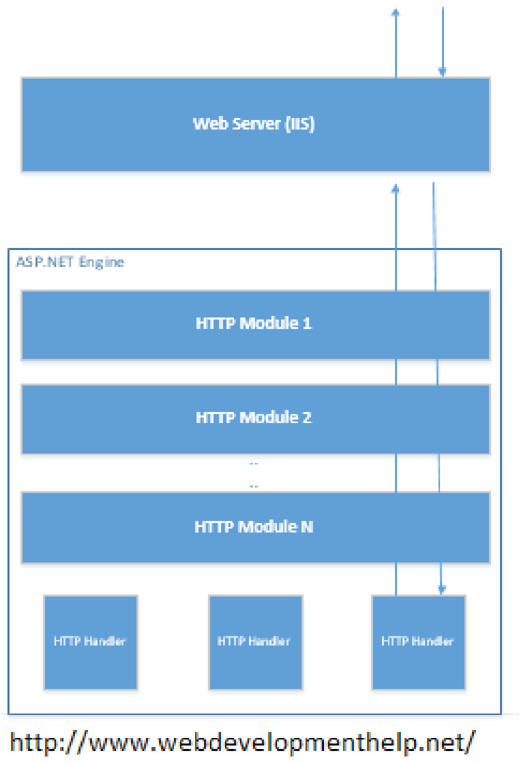
ASP.NET Interview Questions List – Part 1

What are HttpHandlers and HttpModules in ASP.NET?

In order to fully comprehend the concept of HttpHandlers and HttpModules, I have written a detailed ASP.NET Tutorial. Here I am defining both the concepts as follows:

HttpHandler: ASP.NET Engine uses HttpHandlers to handle specific requests on the basis of it's extensions. ASP.NET Page Handler handles all requests coming for (.aspx) pages. We can define our own custom HttpHandler to handle a specific request with a specific extension, say .jpeg, .gif, or .ahmad. But there will always be only one handler for a specific request.

HttpModule: ASP.NET Engine uses HttpModules to inject some specific functionality along with ASP.NET default functionality for all incoming requests regardless of its extensions. There are a number of built-in modules already available in ASP.NET HTTP Pipeline. But we can write our own custom HTTP module to perform some additional functionality (for example, URL rewriting or implementing some security mechanism) for all incoming requests.



For more details about HttpHandlers and HttpModules in ASP.NET, you can follow MSDN link here.

Back to top

What is State Management?

HTTP is a stateless protocol by nature. So, we need some mechanism to preserve state (i.e. state of a webpage, a control or an object etc.) between subsequent requests to server from one or more clients. And this mechanism is referred as **State Management**.

Back to top

What are the State Management Techniques used in ASP.NET?

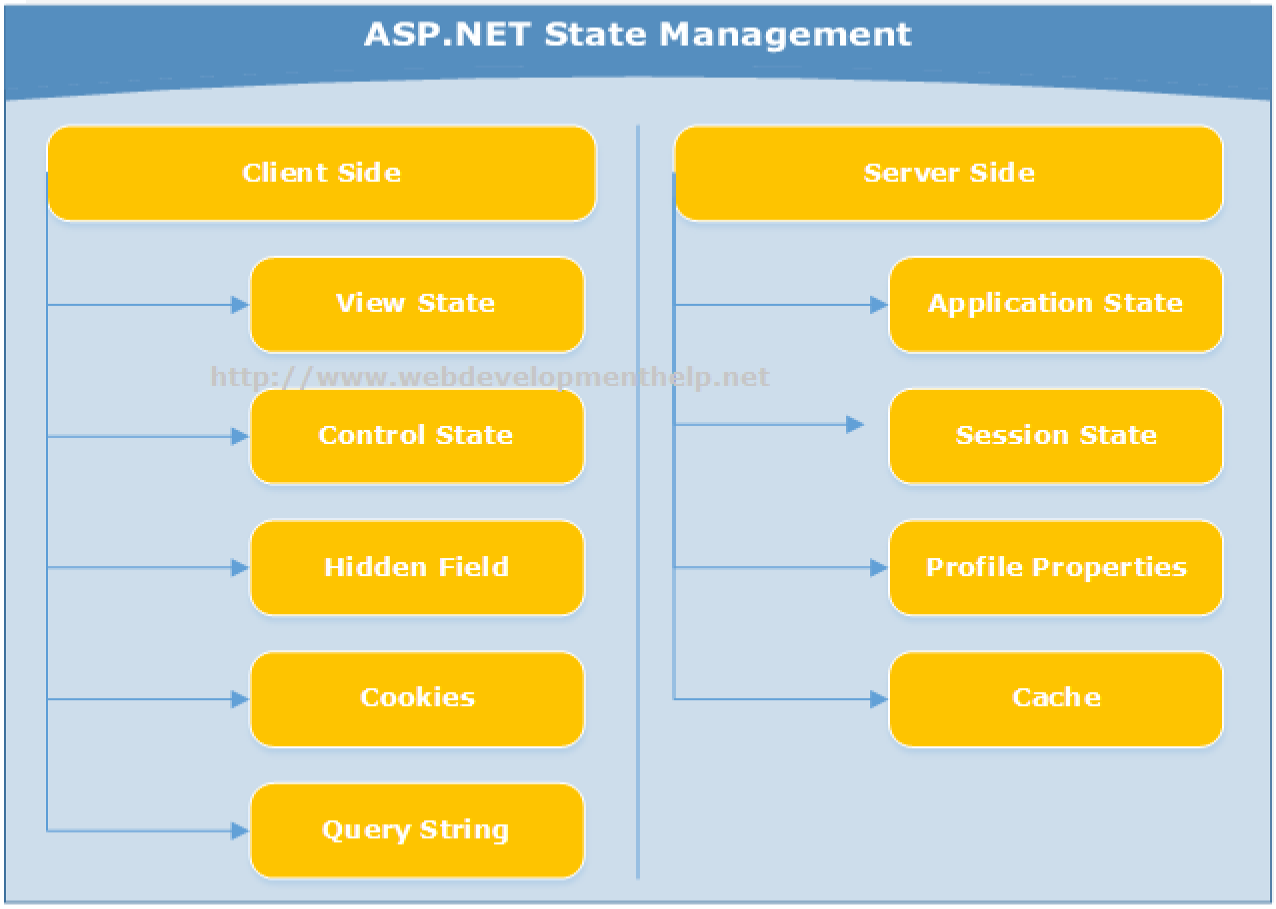
State Management techniques used in ASP.NET can be categorized in two types:

1. Client-Side State Management
   - View State
   - Control State
   - Hidden Fields
   - Cookies
   - Query String
2. Server-Side State Management
   - Application State
   - Session State
   - Profile Properties

    - Cache

Related Articles

- Free Online Tests (ASP.NET | ASP.NET MVC | HTML5 | AJAX)
- ASP.NET WebForms Vs ASP.NET MVC
- Practical Guide to ASP.NET Web API
- Your First ASP.NET MVC Application
- MVC App using Entity Framework
- Online Practice Exam: 70-486



Back to top

What is ViewState? or Explain ViewState as State Management Technique?

ViewState is one of the **Client-Side State Management** techniques that provides page-level state management, which means state is preserved between subsequent requests to same page. By using this technique, state of the page along with its controls is stored in a hidden form field i.e. "__VIEWSTATE" and this field is again available on server when page is posted back with HTTP Request.
You can find this hidden field by looking into view source of an .ASPX page as:

```
<input type="hidden" name="__VIEWSTATE"
value="wEPDwUKMTM4OTIxNTEzNA9kFgJmD2QWAgIBD2QWAgIDDxYCHgVzdHlsZQV" />
```

ViewState data is encoded in Base64 String encoded format.
Back to top

Can we Enable/Disable ViewState?

Yes, ViewState can be enabled or disable at different levels:

- Control Level
  ViewState for a specific control can be enabled or disabled by setting *EnableViewState* property as follows:

```
aControl.EnableViewState = false;
```

- Page Level
  We can enable/disable ViewState for a complete page as follows:

```
<%@ Page Language="C#" EnableViewState="false" %>
```

- Application Level
  For whole application, we can enable/disable views in configuration file as follows:

```
<pages enableViewState="false">
   ….
</pages>
```

Back to top

What is the difference between Session.Clear() and Session.Abandon() in ASP.NET?

As we understand that Session is a Collection and it stores data as Key/Value pair. So, **Session.Clear()** clears all the session values but doesn't destroy the Session. however,

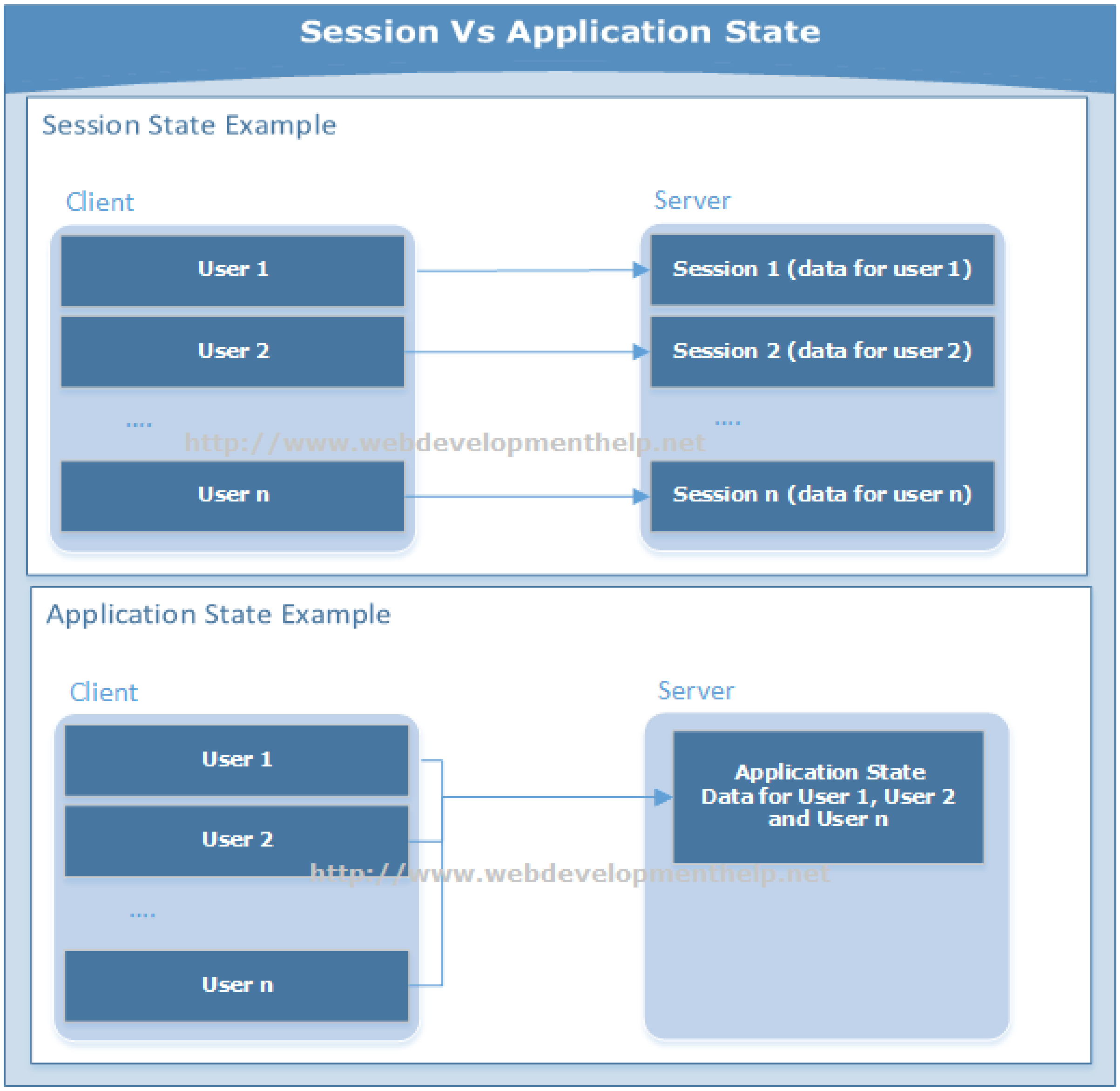
Session.Abandon() destroys the session object.

In other words, Session.Clear() is like deleting all files inside a folder (say "Root") but Session.Abandon() means deleting the "Root" folder.

Back to top

What is the difference between Application and Session State?

Application state is basically a common data repository for an application's all users and their all sessions. On the other hand, Session state is specific to a single user session. Following diagram explains the difference between two state management techniques:


Session Vs Application State
Session State Example
Client
Server
User 1
User 2
....
User n
Session 1 (data for user 1)
Session 2 (data for user 2)
....
Session n (data for user n)
Application State Example
Client
Server
User 1
User 2
....
User n
Application State Data for User 1, User 2 and User n
http://www.webdevelopmenthelp.net

So, we can store data in application state object that is common for all users of a particular application as follows:

```
//Set Value
Application[“UsersCounter”] = Convert.ToInt32(Application[“UsersCounter”]) + 1;
//Retrieve Value
lblUsersCounter.Text = Application[“UsersCounter”].ToString();
```

It’s recommended to store smaller size values in application object.

Session object can store data for a specific session of user. Storage and retrieval is also simple just as for application object.

```
//Set Value
Session[“ProductsCount”] = Convert.ToInt32(Session[“ProductsCount”]) + 1;
//Retrieve Value
lblProductsCounter.Text = Session[“ProductsCount”].ToString();
```

Interview Questions about Session State Modes and Session_Start/Session_End events in Global.asax are already explained here.

Back to top

What is the difference between Label Control and Literal Control?

A Label control in ASP.NET renders text inside <span> tags while a Literal Control renders just the text without any tags.
With Label controls we can easily apply styles using it's CssClass property, however, if we don't want to apply style/formatting, it's better to go for a Literal control.
Back to top

Hyperlink Vs LinkButton in ASP.NET?

A Hyperlink just redirects to a given URL identified by "NavigateURL" property. However a LinkButton which actually displays a Hyperlink style button causes a postback to the same page but it doesn't redirect to a given URL.

Validation Controls related Interview Questions are already given in previous post here.
Back to top

Hopefully, this pool of ASP.NET Interview Questions and Answers along with previous list of Top 10 will be helpful for ASP.NET Developers.

ASP.NET Interview Questions List – Part 2

HTML Server Controls Vs Web Server Controls, Please define?

HTML Server Controls are server-side mapped form of HTML elements. In order to make HTML elements programmable on server-side, **ASP.NET** framework added runat="server" attribute, so it's accessible in ASP.NET code-behind. A typical HTML Server Control is as follows:

```
    <input type="text" id="txtFirstName" runat="server" />
```

On the other hand, **Web Server Controls** are more feature-rich as compared to HTML server controls and truly designed to provide Win Apps development experience for ASP.NET Web developers. Also, it provides comparatively high level of abstraction. Apart from mapping to existing HTML elements, web server controls provide more rich and complex functionality like Calendar, Grid, Repeater, menu, tree view etc.

Back to top

So, which one you prefer to use while developing an ASP.NET Web Application?

Although both of these types of controls render HTML elements but Web Server controls being rich in functionality can render additional HTML tags that collectively fulfill control's functionality. On the other hand, HTML Server Controls are comparatively lightweight only producing the corresponding HTML element.So, preference depends totally on your circumstances. If you are migrating from a classic ASP application to ASP.NET, then only choice is using HTML server controls but if you are going to develop a new application and you wanted to provide rich functionality with ease (i.e. calendar, grid, tree view etc), web server controls are the only choice because we don't have it in HTML server controls.

Back to top

What is the role of ValidationSummary Control?

Sometimes, we have a requirement that all validation messages need to display at one location (may be top or bottom of a web form). In such scenario, ValidationSummary control displays all validation messages at one place.
DisplayMode property of this control can be used to display in different formats as follows:

- BulletList
- List
- SingleParagraph

We have already provided few questions about Validation Controls in ASP.NET in previous article.
Back to top

Globalization Vs Localization

There are situations when we need to build an application that work for multiple cultures e.g *en-US*, *ar-SA* etc, this process of designing and building applications that work for more than one cultures is called globalization. However, customizing an application for a specific culture is localization. Both globalization and localization normally go together.

Back to top

How to access information about a user's locale in ASP.NET?

User's locale information can be accessed through ***System.Web.UI.Page.Culture*** property.

Back to top

Difference between Culture and UICulture properties in ASP.NET?

CultureInfo class plays an important role for localizing our application pages. Culture is specific in localizing non-visual parts of the page like DateTime, Currency, number formatting etc. while on the other hand, ***UICulture*** is specific in localizing visual part of a webpage like Language being used to display the contents of the web page.
Back to top

Difference between Local and Global resources?

Resources for a localized ASP.NET application can be stored locally as well as globally. Local resources are specific to web page and stored in a folder ***App_LocalResources***. It can be accessible to that specific page only. Global resources are accessed by almost all application pages and stored in ***App_GlobalResources*** folder.

Back to top

What is a Neutral Culture? and how its different from a Specific Culture?

As we have seen earlier that a culture has two things i.e. Language and Country/Region. For example, en-Us represents English – United States; en-GB represents English – Great Britain; and ar-SA represents Arabic – Saudi Arabia. So, we can easily understand that first part is language while second is country/region.

A neutral culture is one that is associated only with part-1 i.e. language and not with part-2 i.e. country/region. For example, ar is the neutral name for Arabic culture while ar-SA is specific to Saudi Arabian Arabic culture.
Back to top

Difference between Response.Write() and Response.Output.Write()?

Difference between Response.Write() and Response.Output.Write() is that the later provide formatting capability through String.Format-Style which the former doesn't have.

In case of Response.Write(), HttpResponse calls directly the following method:

```
    public void Write(object obj)

   {

this._writer.Write(obj);
```

```
   }
```

Above method internally calls it TextWriter’s write() method.

However, in case of Response.Output.Writer(), HttpResponse actually get reference to TextWriter through

Reponse.Output and then after getting control TextWtiter, call other overloaded method that helps to format the string.

```
     Response.Output.Write(“This is an {0} test at {1:d}”, “amazing”,
DateTime.Now);
```

ASP.NET Interview Questions List – Part 3

What is Caching and what are the benefits of using it?

Performance has always been a concern for web based applications. So, Caching is a mechanism that improve performance for an application by storing data in memory for fast access. When the application will access data from Cache (i.e. in-memory) instead of fetching it from original data store (may be a database), it will definitely improve performance.But Caching benefits are not limited to performance only, it also improve application Scalability as well as Availability.

- Load on server is reduced when data is fetched from Cache instead of original source, thus improving scalability of an application.
- Caching normally keep serving application data even if the original source is temporarily down, thus improving availability of an application.

Back to top

Cache Management in ASP.NET?

ASP.NET provided support for Cache Management in almost all versions. In .NET Framework 3.5 and older, the support for caching was provided through classes available in *System.Web.Caching*. But this support was limited to *System.Web* meaning for ASP.NET Web Applications only. Now, with .NET Framework 4.0 and later, this support is enhance to non-Web Applications also by providing APIs in *System.Runtime.Caching*.

ASP.NET supports three types of Caching:

- Page Output Caching
- Partial Page Caching
- Data Caching

Back to top

Page Output Cache Vs Partial Page Cache Vs Application Data Cache in ASP.NET?

Page Output Cache
In case of Page Output Cache, the output of a complete web page is stored in a cache. So, when that web page is accessed again, it will be loaded from cache instead of fetching page data again from data source.

Partial Page Cache
For Partial Page Cache (also known as Page Fragment Cache), a part or fragment of a web page is stored in Cache as opposed to complete page caching for Page Output Cache. For example, caching a user control on a web page that displays product categories using Page Fragment Cache.

Data Cache
In some scenarios, we may store frequently used objects into cache using ASP.NET Cache API. So, later on, that object will be loaded from cache instead of instantiating object again and fetching data from original source for it.

Back to top

How to use Page Output Cache in ASP.NET?

Implementing Page Output Cache in ASP.NET is simple. For Page Output Caching, **@ OutputCache** directive is used on an ASP.NET page as follows:

```
<%@ OutputCache Duration="50" VaryByParam="None" %>
```

Duration value is in seconds and it tells the page that how long to cache the contents? Now, when we will access the page, it will verify that either it exists in Cache? if Yes, then verify that is it expired? If not then fetch it from Cache and render otherwise create a new instance of the page and put it back to Cache.

The other parameter of this directive is "VaryByParam". If it's value is specified to something as follows:

```
<%@ OutputCache Duration="50" VaryByParam="ProductId" %>
```

Now, Cache is dependent on the value of this parameter, If the value of parameter remains same, page will be fetched from Cache otherwise it will be refreshed again.For in-depth details on Page Output Cache, follow here.

Back to top

How to use Page Fragment or Partial Page Cache in ASP.NET?

Page Fragment Caching uses the same @ OutputCache directive with VaryByControl parameter as follows:

```
<%@ OutputCache Duration="50" VaryByParam="None" VaryByControl="ControlName"
%>
```

In this case, Cache is dependent on the value of Control specified in VaryByControl parameter. For example, content on a page are dependent on the selected values of a dropdownlist, so, VaryByControl will have the dropdownlist control name as value.
Back to top

How to use Data Cache in ASP.NET?

We have already explained the usage of Data Cache above in this series of ASP.NET Interview Questions that in particular situations, we need to store objects into cache. Adding an object to Cache and accessing it from Cache is simple.

We can use "*Add*" method to add an object to Cache as:

```
Cache.Add(key, value, dependencies, absoluteExpiration, slidingExpiration, priority,
onRemoveCallback); if (Cache["ProductKey"] == null)
Cache.Add("ProductKey",
objProduct,
null,
DateTime.Now.AddSeconds(60),
Cache.NoSlidingExpiration,
CacheItemPriority.High,
null);
```

To retrieve it back:

```
Product objProduct = (Product) Cache["ProductKey"];
```

Further, in this post and next post, we will be discussing about **ASP.NET Security Interview Questions**.

Back to top

Authentication Vs Authorization?

Authentication and Authorization are two key security related concepts that are independent but normally go together.

Authentication is a process that verifies the identity of a user. On ther hand, Authorization is the process of assigning rights/privileges to already authenticated user.

For example, when a user tries to login a web application. First of all, user identity is verified that either he/she is valid registered user of application. If his/her identity validated successfully then appropriate privileges are assigned accordingly. Different users may have different privileges on same application, for example, user1 can only view/read some records while user2 may have privileges for all CRUD (Create, Read, Update, Delete) operations on same data.

Back to top

What are the available Authentication modes in ASP.NET?

We have already explained this question in previous ASP.NET Interview Questions and Answers post, don't skip this important questions details.

Back to top

What is the difference between Windows Authentication and Forms Authentication in ASP.NET?

Windows Authentication is a way to authenticate a user against Windows accounts. Windows authentication mode is suitable for corporate users in windows environment.

In case of **Forms Authentication**, a separate list of users is maintained for authentication. For example, we can maintain the list in database and authenticate user against it.

We can set authentication mode in web.config as follows:

```
<authentication mode="Forms">
```

Back to top

What is Protected Configuration in ASP.NET?

While developing an ASP.NET application, we normally store a number of important sensitive information in our config files like encryption keys, connection strings etc. Application vulnerability increases if this sensitive information is stored as plane text. So ***Protected Configuration*** is an ASP.NET feature that enables to encrypt such sensitive information in configuration files.

ASP.NET Interview Questions List – Part 4

What is Passport Authentication?

As we have discussed previously that there are three types of authentications in ASP.NET i.e.

- Windows Authentication
- Forms Authentication
- Passport Authentication

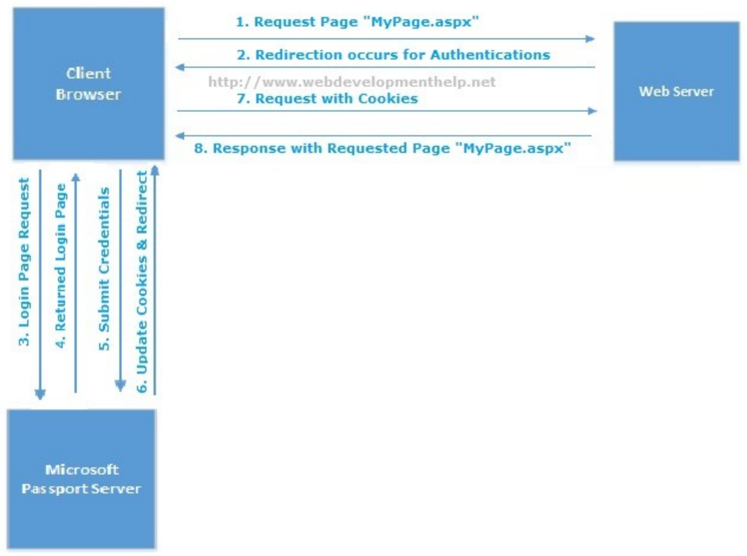
Windows and Forms Authentications are already explained.
Passport Authentication actually validates against a centralized authentication service i.e. Microsoft Passport Service. We don’t need to implement our own custom authentication mechanism if implementing .NET Passport Single Sign-In (SSI) service.

Back to top

Can you briefly explain how Passport Authentication works?

As discussed above that Passport Authentication is a central service. It just authenticate (validate the credentials), no authorization (grant or deny access to a site). So, implementing application will check for the Passport Authentication Cookie. In case of unavailability of Passport Cookie, user is redirected to passport Sign-In page. User provides the credentials on Sign-In page, if validated, Authentication Cookie is stored on client machine and redirected to the requested page.

Below picture clearly explains step by step process of Passport authentication in ASP.NET.


Client Browser
Web Server
Microsoft Passport Server
1. Request Page "MyPage.aspx"
2. Redirection occurs for Authentications
http://www.webdevelopmenthelp.net
7. Request with Cookies
8. Response with Requested Page "MyPage.aspx"
3. Login Page Request
4. Returned Login Page
5. Submit Credentials
6. Update Cookies & Redirect

Back to top

What are the advantages of using Passport Authentication?

Advantages of Passport Authentication are:

- We don't need to care of authentication mechanism our self, Passport SSI does this for us.
- Single login credentials can be used to access multiple sites. User don't need to remember separate credentials for individual site.

Back to top

What is Role-based Security?

We have discussed about authentication in above questions but another different but related concept is Authorization. Authorization is a process of granting privileges or permissions on resources to an authenticated user. So,
"*Role Based Security is a technique we use to implement authorization on the basis of user's roles within an organization. It's more granular approach to grant or revoke permissions on resources through user's roles.*"

An example of granting or revoking permissions in configuration file using windows built-in groups as follows:

```
<authorization >
   <allow roles="MyDomain1Administrators" / >   < !– Allow Admin of this domain — >
   <deny users="*"  / >                            < !– Deny anyone else. — >
</authorization >
```

Back to top

What are the different Security Controls in ASP.NET?

ASP.NET provides several security controls which are actually Web Server controls. You can find those in your Visual Studio Toolbox.

Login Control:
In almost every application we need to take user credentials on a typical login page. Login control provides the same standard functionality and reduces the effort for building it from scratch.

LoginName:
After a user successfully logged in to an application, we normally display his/her username to top right or some other place on the page. Now, this functionality is provided by LoginName control.

LoginView Control:
LoginView control displays different view for different users. Using AnonymousTemplate and LoggedInTemplate, different information can be presented to different users.

LoginStatus Control:
LoginStatus control implies whether a user is authenticated or not. For an unathenticated user, it displays a link to login page. On the other hand, for authenticated user, a logout link is displayed.

LoginRecovery Control:
Password recovery is another important functionality simplified through PasswordRecovery control. It sends an email with login credentials to registered user email.

Back to top

What is Code-Access Security (CAS)?

In one of above ASP.NET security related interview questions, we discussed about Role Based Security that restrict access to resources on the basis of user's role. CAS (**Code Access Security**) is entirely a different concept. It's .NET CLR's security system that restrict the code to perform an unwanted task by applying security policies. Using CAS (Code Access Security), we can restrict *what our code can do?* and also *what code can call our code?*

Back to top

What are the key functions of Code Access Security?

As per documentation, key functions of Code Access Security are (straight from MSDN):

- Defines permissions and permission sets that represent the right to access various system resources.

- Enables code to demand that its callers have specific permissions.
- Enables code to demand that its callers possess a digital signature, thus allowing only callers from a particular organization or site to call the protected code.
- Enforces restrictions on code at run time by comparing the granted permissions of every caller on the call stack to the permissions that callers must have.

Back to top

What .NET Tool can be used to Enable/Disable CAS?

Code Access Security Tool (Caspol.exe) can be used to turn Code Access Security ON or OFF as follows:

- caspol -security on
- caspol -security off

We can also list all code groups using following command.

- caspol -listgroups

Back to top

What is Impersonation in ASP.NET?

Impersonation is an act of a user to pretend itself to be another user. By default, ASP.NET executes application code using the same user account as that of ASP.NET process i.e. Network Service. But with impersonation enabled, it executes code with the windows identity of the user making the request.

For example, if a user 'user1' logged in and IIS is setup to run as Network Service. If 'user1' call a piece of code on another computer (may be a web service call), the other computer will see the IIS user instead of 'user1'. But we can enable impersonation to allow 'user1' to access the web service using its windows identity instead of Network Service.

Back to top

How to configure Impersonation in ASP.NET?

By default, impersonation is disabled in ASP.NET. Impersonation can be Enabled/Disabled as follows:

```
</configuration>
    <system.web>
       <identity impersonate="true"/> <! — To disable set impersonate="false" –>
     </system.web>
</configuration>
```

Impersonate a specific user account as:

```
<identity impersonate="true" userName="user" password="pwd" />
```

Back to top

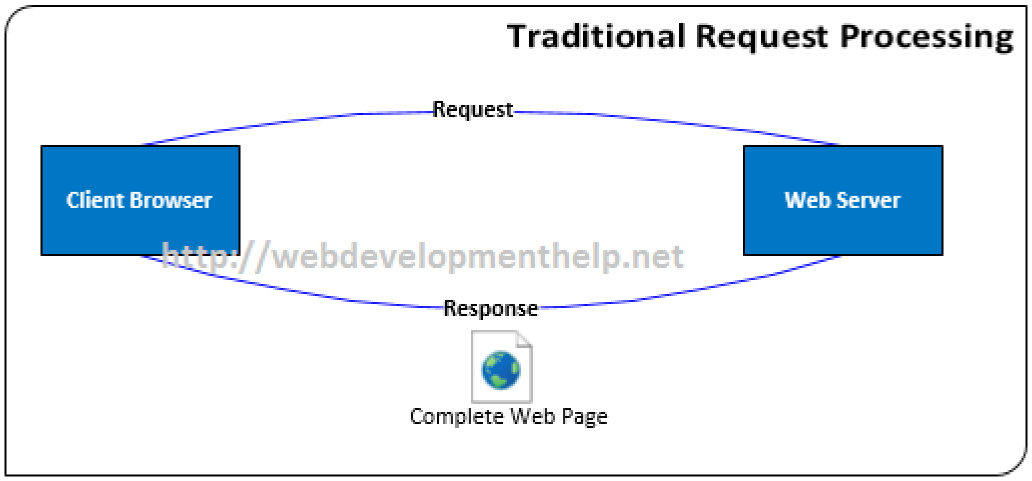
On completing Part 4 of this ASP.NET Interview Questions and Answers series, we have completed major questions on ASP.NET Security. Hopefully, this series will be beneficial in terms of preparing an ASP.NET Interview.

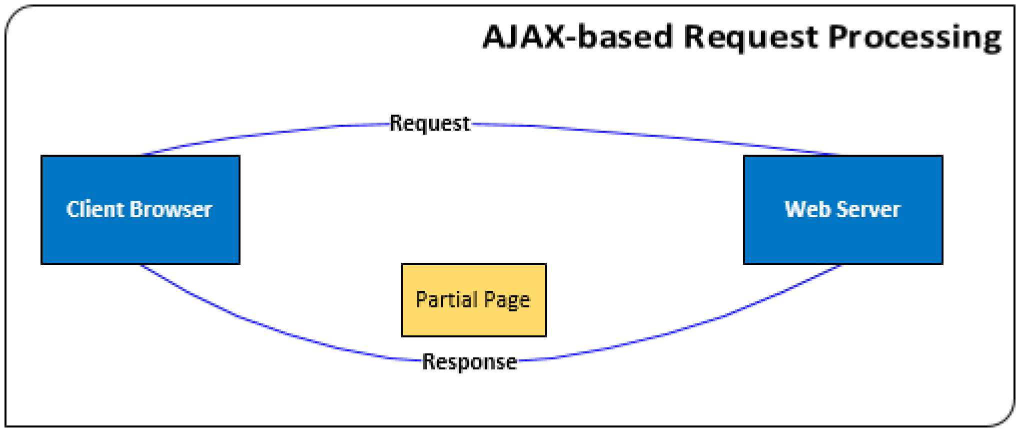
ASP.NET AJAX Interview Questions List

Define AJAX?

AJAX stands for “Asynchronous JavaScript and XML”. It’s basically a technique for creating Rich Internet Applications (RIA) that are faster as well as more interactive, using a combination of commonly used techniques as HTML/XHTML, CSS, Document Object Model (DOM), JavaScript, XML/XSLT and XMLHttpRequest object.

XMLHttpRequest object is the key basis of AJAX and makes it possible to communicate with the web server asynchronously and exchange data. As compared to a traditional request which returns a complete web page, partial web page is returned as response to an AJAX request.





[Back to top]()

::::: Free Practical Guide to ASP.NET MVC Web API (Web Edition | PDF Download) :::::

Please elaborate XMLHttpRequest Object further?

XMLHttpRequest is the core object in AJAX technology regardless of any implementation. XMLHttpRequest object is used to exchange data with a server seamlessly. Basically JavaScript uses this Object to exchange XML as well as text data between client and server. An AJAX implementation uses this object and communicate with server but it doesn't require the complete page to be refreshed.

Back to top

How to send a request to server using XMLHttpRequest Object?

We can send a request to server using HTTP GET and POST methods as follows:

```
//Simple GET Request

var xmlHttp = new XMLHttpRequest();

xmlHttp.open(“GET”, “TestFile.txt”, true);

xmlHttp.send();

//Simple POST Request

var xmlHttp = new XMLHttpRequest();

xmlHttp.open(“POST”, “TestFile.txt”, true);

xmlHttp.send();
```

Back to top

What is ASP.NET AJAX?

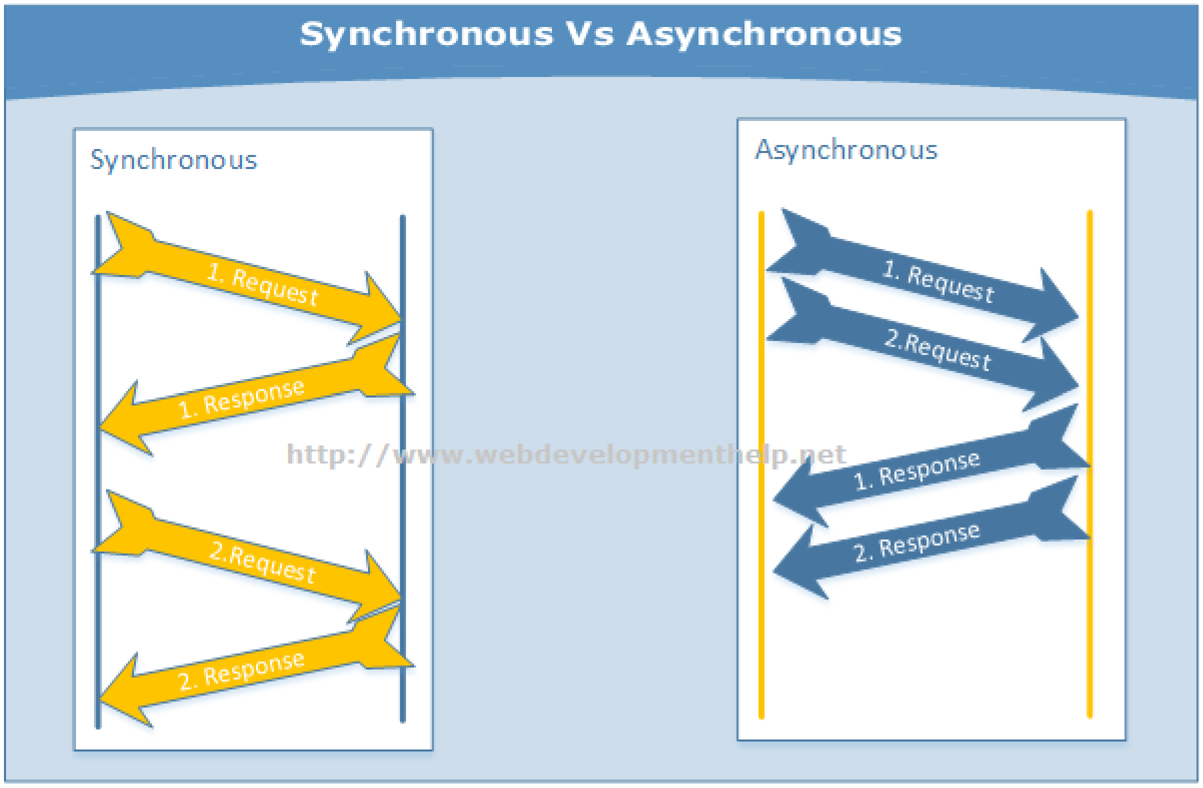
Microsoft provided an implementation of AJAX functionality known as **ASP.NET AJAX**.
As we discussed in above interview question that AJAX is a combination of various techniques, so Microsoft simplified the usage of these techniques with its own implementation. ASP.NET AJAX is a set of extensions to ASP.NET and comes with reusable AJAX controls. Using ASP.NET AJAX, we can develop applications that can update partial page instead of a complete page refresh.

Back to top

Difference between Synchronous and Asynchronous Postback?

In Synchronous postback, complete web page is sent to server and in return rendering the output (i.e. complete page), whereas in case of Asynchronous postback, partial page goes to the server and renders only partial (required) part of the page.

Normally a method making Synchronous call always waits for response to do next task (i.e. might be another call). On the other hand, for Asynchronous, no waiting required as illustrated by below figure:



Back to top

What are the basic controls in ASP.NET AJAX?

Following controls can be considered as core AJAX controls in ASP.NET.

- ScriptManager
- ScriptManagerProxy
- UpdatePanel
- UpdateProgress
- Timer

Later more controls are added to ASP.NET AJAX library e.g. Script Loader, Client Data Context, Client Data Access, jQuery Integration etc. For complete reference, look for AJAX Control Toolkit.

Back to top

What is a ScriptManager in ASP.NET AJAX?

In order to use AJAX functionality on a web page, we add a ScriptManager control to the page in most of the scenarios, because ScriptManager control register AJAX library scripts to that particular web page. We can have only one ScriptManager per page.

```
<asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
```

ScriptManager basically manages all ASP.NET AJAX resources of a web page, creates proxies for asynchronous web service call and also manages partial page updates… etc.

Back to top

ScriptManager Vs ScriptManagerProxy?

As we understand that we can have only one **ScriptManager** control on a page but we can have multiple **ScriptManagerProxy** controls.
Consider a scenario that we have ScriptManager in our MasterPage that is available for all content pages. Now, we wanted to register a web service in a particular page. So, we will not add another ScriptManager to that page instead we will add ScriptManagerProxy to it in order to avoid error.
Back to top

What is the role of UpdatePanel in ASP.NET AJAX?

UpdatePanel is the control that facilitate the partial page rendering functionality in an ASP.NET application. As discussed earlier that using ASP.NET AJAX, we can communicate with a web server asynchronously and update a part of a page without a complete page postback. In order to apply partial page update/rendering, we can add one or more UpdatePanel controls to our ASP.NET Page as follows:

```
  <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>

<asp:UpdatePanel ID="UpdatePanel1" runat="server">

          <ContentTemplate>

             <asp:Label ID="lblPanel" runat="server" Text="Update Panel
Added."></asp:Label><br />

             <asp:Button ID="btnTestButton"
                              runat="server"
                             OnClick="btnTestButton_Click"
                             Text="Test Button" />

     </ContentTemplate>

   </asp:UpdatePanel>
```

Back to top

What are the limitations of AJAX?

- AJAX on an application will not work if JavaScript is disabled.
- In some scenarios, it exposes vulnerability.
- It will always be difficult to bookmark application state.

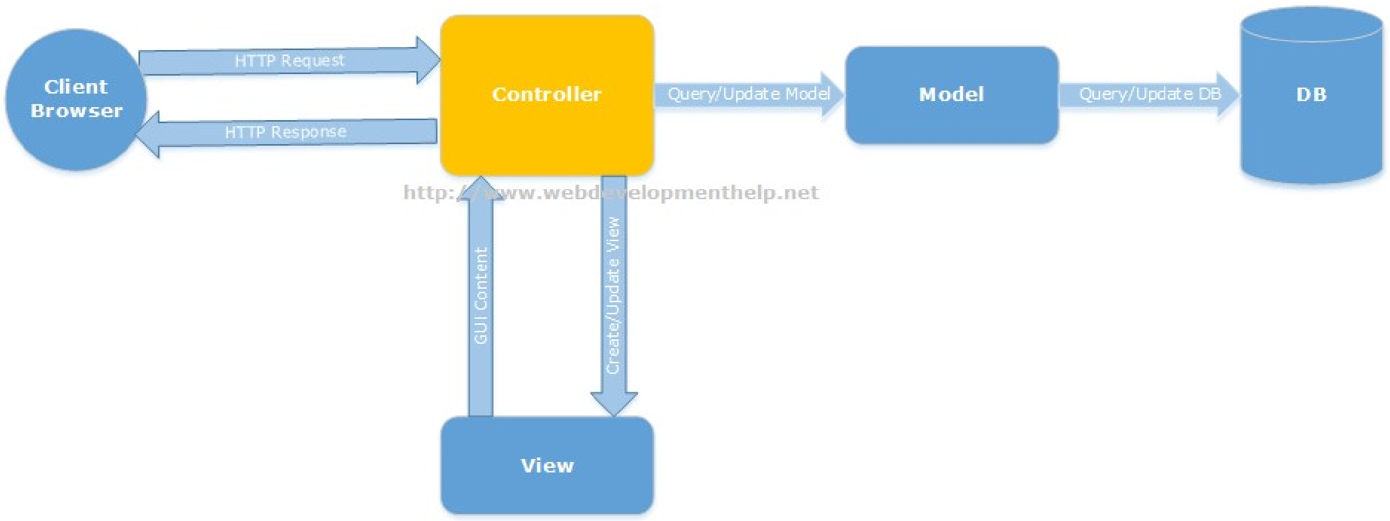
- Application behavior may be slow in some scenarios, because of different loading time of controls on a single page.

ASP.NET MVC Interview Questions List

1. Explain MVC (Model-View-Controller) in general?

MVC (Model-View-Controller) is an architectural software pattern that basically decouples various components of a web application. By using MVC pattern, we can develop applications that are more flexible to changes without affecting the other components of our application.

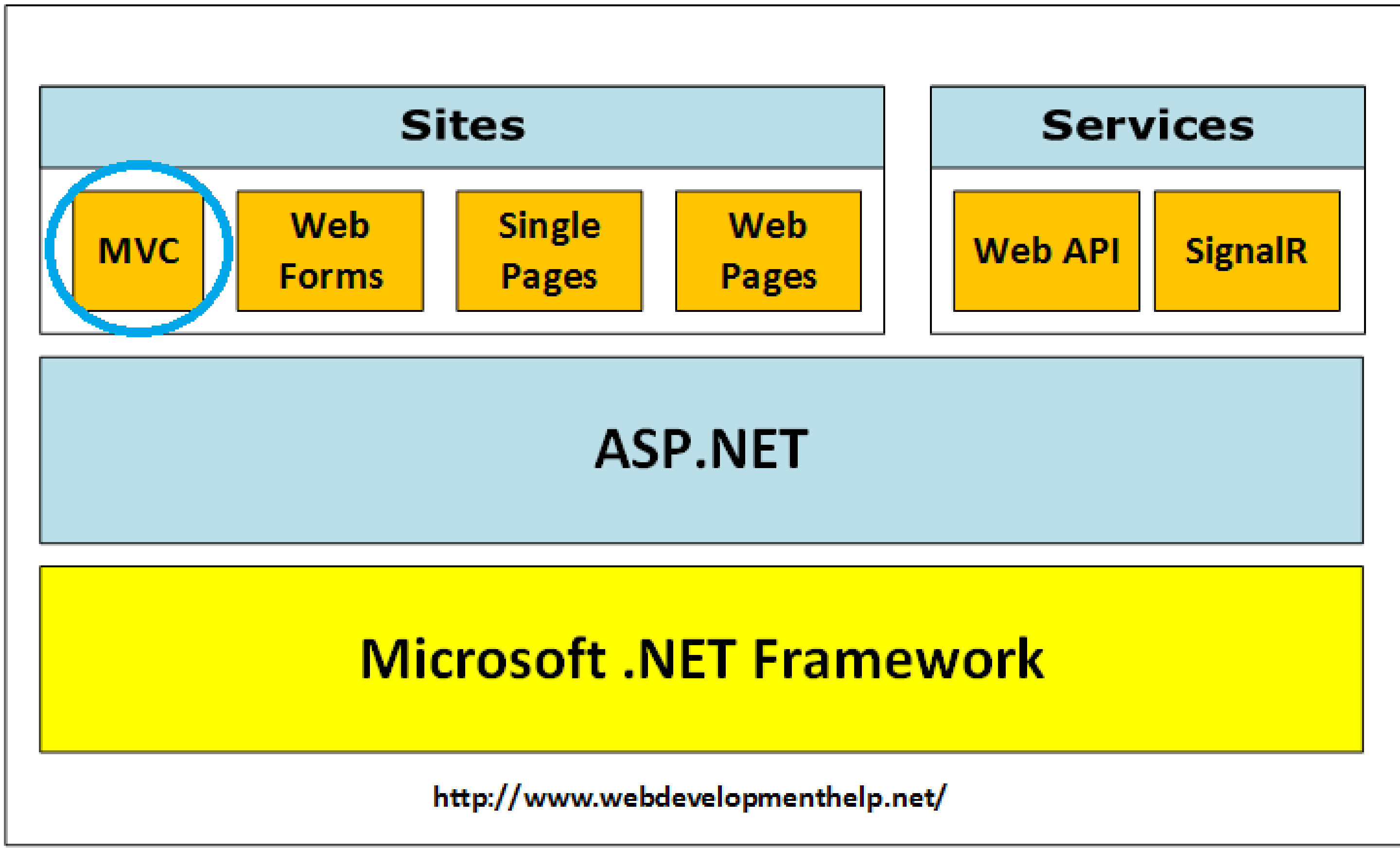
- “Model”, is basically domain data.
- “View”, is user interface to render domain data.
- “Controller”, translates user actions into appropriate operations performed on model.



Back to top

2. What is ASP.NET MVC?

ASP.NET MVC is a web development framework from Microsoft that is based on MVC (Model-View-Controller) architectural design pattern. Microsoft has streamlined the development of MVC based applications using ASP.NET MVC framework.



Back to top

3. Difference between ASP.NET MVC and ASP.NET WebForms?

ASP.NET Web Forms uses Page controller pattern approach for rendering layout, whereas ASP.NET MVC uses Front controller approach. In case of Page controller approach, every page has its own controller i.e. code-behind file that processes the request. On the other hand, in ASP.NET MVC, a common controller for all pages processes the requests. Follow the link for the difference between the ASP.NET MVC and ASP.NET WebForms. Back to top

4. What are the Core features of ASP.NET MVC?

Core features of ASP.NET MVC framework are:

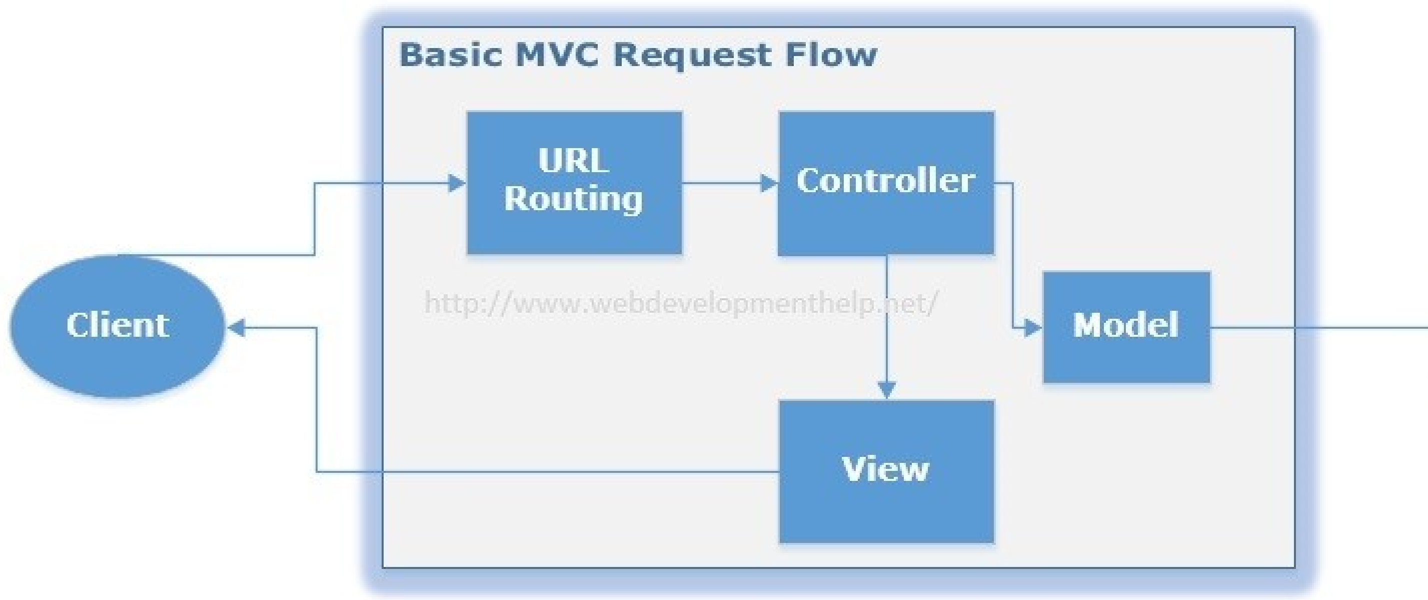
- ***Clear separation of application concerns*** (Presentation and Business Logic). It reduces complexity that makes it ideal for large scale applications where multiple teams are working.
- It's an **extensible** as well as **pluggable framework**. We can plug components and further customize them easily.
- It provides extensive support for URL Routing that helps to make friendly URLs (means friendly for human as well as Search Engines).
- It supports for **Test Driven Development (TDD)** approach. In ASP.NET WebForms, testing support is dependent on Web Server but ASP.NET MVC makes it independent of Web Server, database or any other classes.
- Support for **existing ASP.NET features** like membership and roles, authentication and authorization, provider model and caching etc.

Follow for detailed understanding of above mentioned core features.

Back to top

5. Can you please explain the request flow in ASP.NET MVC framework?

Request flow for ASP.NET MVC framework is as follows: Request hits the controller coming from client. Controller plays its role and decides which model to use in order to serve the request. Further passing that model to view which then transforms the model and generate an appropriate response that is rendered to client.



You can follow the link, in order to understand the [Complete Application Life Cycle in ASP.NET MVC](). [Back to top]()

6. What is Routing in ASP.NET MVC?

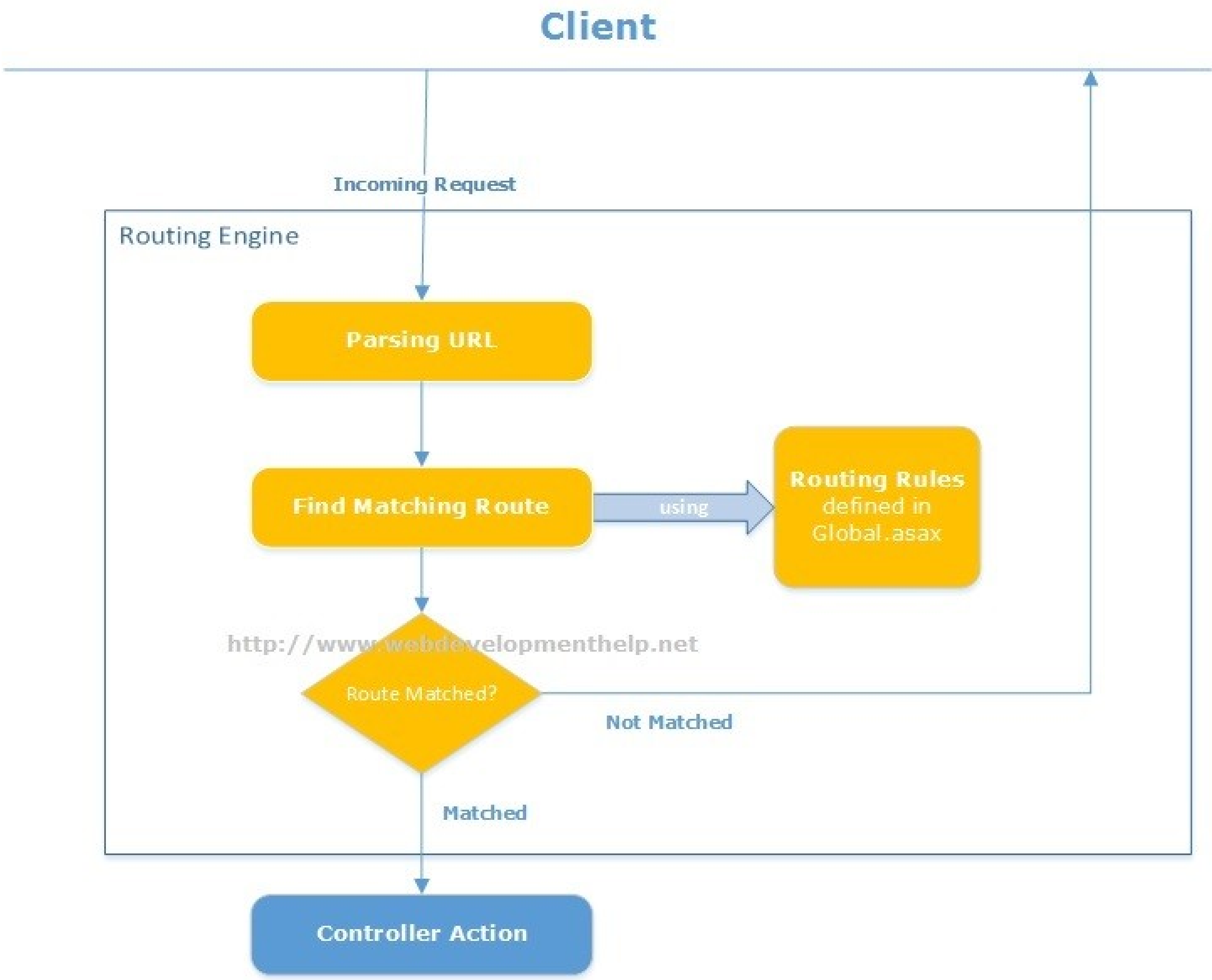
In case of a typical ASP.NET application, incoming requests are mapped to physical files such as .aspx file. On the other hand, ASP.NET MVC framework uses friendly URLs that more easily describe user's action but not mapped to physical files.

Let's see below URLs for both ASP.NET and ASP.NET MVC.

```
//ASP.NET  approach - Pointing to physical files (Student.aspx)
//Displaying all students
http://locahost:XXXX/Student.aspx

//Displaying a student by Id = 5
http://locahost:XXXX/Student.aspx?Id=5

```

```
//ASP.NET MVC approach - Pointing to Controller i.e. Student
//Displaying all students
http://locahost:XXXX/Student

//Displaying student by Id = 5
http://locahost:XXXX/Student/5
```

ASP.NET MVC framework uses a routing engine, that maps URLs to controller classes. We can define routing rules for the engine, so that it can map incoming request URLs to appropriate controller.



Practically, when a user types a URL in a browser window for an ASP.NET MVC application and presses “go” button, routing engine uses routing rules that are defined in Global.asax file in order to parse the URL and find out the path of corresponding controller. You can find complete details of ASP.NET MVC Routing here.

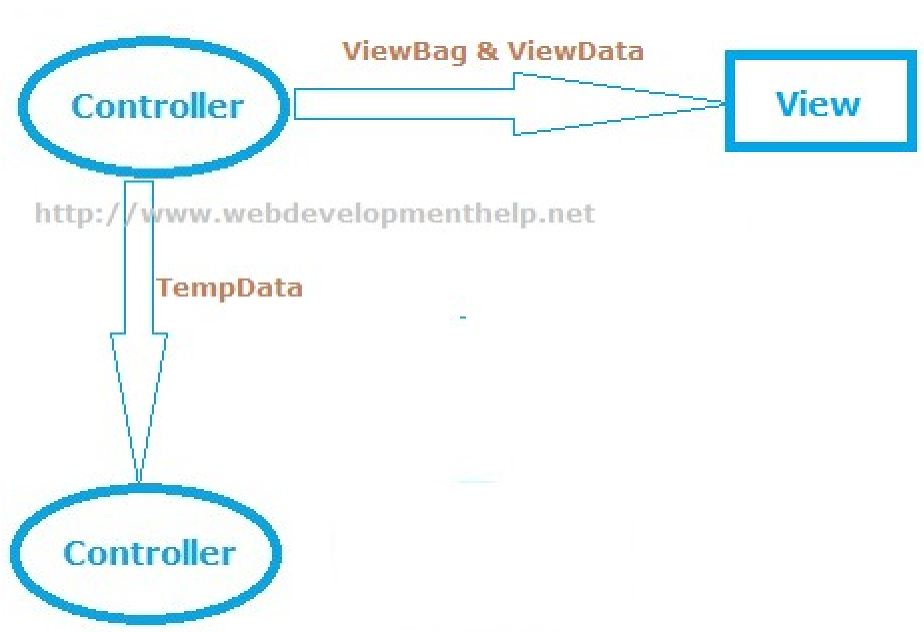
Back to top

7. What is the difference between ViewData, ViewBag and TempData?

In order to pass data from controller to view and in next subsequent request, ASP.NET MVC framework provides different options i.e. ViewData, ViewBag and TempData. Both **ViewBag and ViewData** are used to to communicate between controller and corresponding view. But this communication is only for server call, it becomes null if redirect occurs.

So, in short, its a mechanism to maintain state between controller and corresponding view. **ViewData** is a dictionary object while **ViewBag** is a dynamic property (a new C# 4.0

feature). ViewData being a dictionary object is accessible using strings as keys and also requires typecasting for complex types. On the other hand, ViewBag doesn't have typecasting and null checks. **TempData** is also a dictionary object that stays for the time of an HTTP Request. So, Tempdata can be used to maintain data between redirects i.e from one controller to the other controller.



You can easily find detailed examples for [implementation of ViewBag, ViewData and TempData]() here.
[Back to top]()

So are you comfortable with ASP.NET MVC Now? Let's test your ASP.NET MVC Skill.

Please choose the correct statements about ViewBag and ViewData [Choose Two]?

- A. ViewData is a Dictionary object and it requires typecasting while getting data.
- B. ViewData is a Dynamic Property so it doesn't require typecasting for getting data.
- C. ViewBag is a Dictionary object and it requires typecasting while getting data.
- D. ViewBag is a Dynamic Property so it doesn't require typecasting for getting data.

Correct Answer: A, D

If you want to further test your ASP.NET MVC skill, Take a Complete **FREE Online Test** or MCSD Practice Exam: 70-486 (**Developing ASP.NET MVC Web Applications**). Simply [Click Here]().

8. What are Action Methods in ASP.NET MVC?

As I already explained about request flow in ASP.NET MVC framework that request coming from client hits controller first. Actually MVC application determines the corresponding controller by using routing rules defined in Global.asax. And controllers have specific methods for each user actions. Each request coming to controller is for a specific Action Method. The following code sample, "ShowBook" is an example of an Action Method.

```
public ViewResult ShowBook(int id)
{
var computerBook = db.Books.Where(p => P.BookID == id).First();
return View(computerBook);
}
```

Action methods perform certain operation using *Model* and return result back to *View*. As in above example, ***ShowBook*** is an action method that takes an Id as input, fetch specific book data and returns back to *View* as *ViewResult*. In ASP.NET MVC, we have many built-in *ActionResults* type:

- ViewResult
- PartialViewResult
- RedirectResult
- RedirectToRouteResult
- ContentResult
- JsonResult
- EmptyResult
- and many more….

For a complete list of available *ActionResults* types with *Helper methods*, please click here. **Important Note:** All public methods of a Controller in ASP.NET MVC framework are considered to be Action Methods by default. If we want our controller to have a Non Action Method, we need to explicitly mark it with *NonAction* attribute as follows:

```
[NonAction]
public void MyNonActionMethod()
{
```

```
4 .....

5 }
```

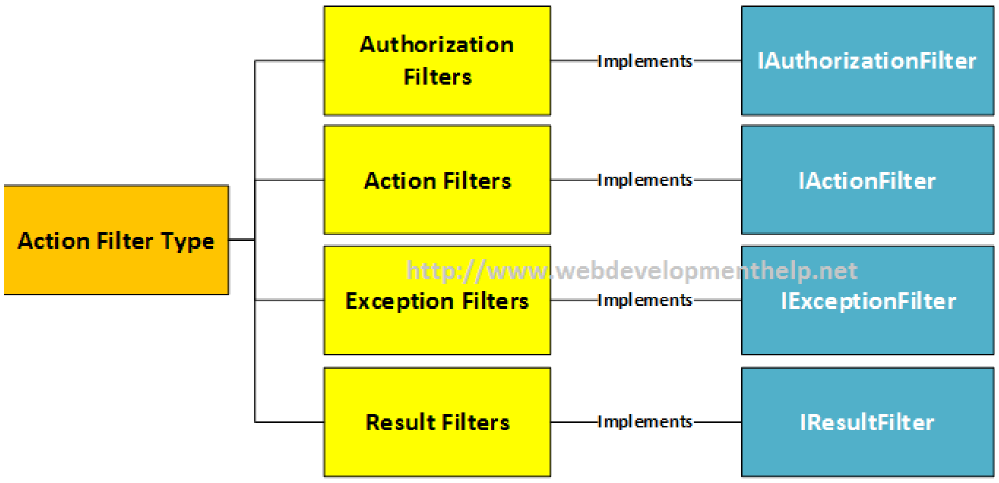
Back to top

9.Explain the role of Model in ASP.NET MVC?

One of the core feature of ASP.NET MVC is that it separates the input and UI logic from business logic. Role of Model in ASP.NET MVC is to contain all application logic including validation, business and data access logic except view i.e. input and controller i.e UI logic. Model is normally responsible for accessing data from some persistent medium like database and manipulate it, so you can expect that interviewer can ask questions on database access topics here along with ASP.NET MVC Interview Questions. Back to top

10.What are Action Filters in ASP.NET MVC?

If we need to apply some specific logic before or after action methods, we use action filters. We can apply these action filters to a controller or a specific controller action. Action filters are basically custom classes that provide a mean for adding pre-action or post-action behavior to controller actions.



For example,

- Authorize filter can be used to restrict access to a specific user or a role.
- OutputCache filter can cache the output of a controller action for a specific duration.
- and more…

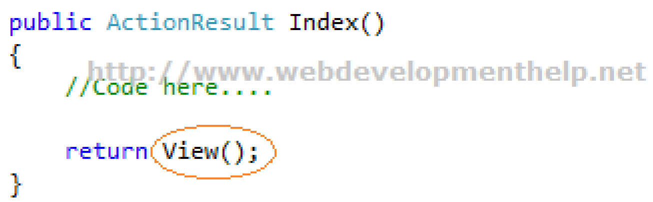
What is a ViewEngine in ASP.NET MVC?

“View Engine in ASP.NET MVC is used to translate our views to HTML and then render to browser.” There are few View Engines available for ASP.NET MVC but commonly used View

Engines are Razor, Web Forms/ASPX, NHaml and Spark etc. Most of the developers are familiar with Web Forms View Engine (ASPX) and Razor View Engine.

- Web Form View Engine was with ASP.NET MVC since beginning.
- Razor View Engine was introduced later in MVC3.
- NHaml is an open source view engine since 2007.
- Spark is also an open source since 2008.

Note: For a detailed comparison between Web Form View Engine and Razor View Engine,



```
public ActionResult Index()
{
    //Code here....

    return View();
}
```

please follow here.
Back to top

What is the difference between Razor View Engine and ASPX View Engine?

I have written a separate detailed blog post to understand the Difference between ASPX View Engine and Razor View Engine. You can follow the link to get detailed step by step description here. Most important differences are listed below:

ASPX View Engine	Razor View Engine
*WebForms View Engine uses namespace “**System.Web.Mvc.WebFormViewEngine**”.*	***“System.Web.Razor”*** is the namespace for Razor View Engine.
Comparatively fast.	A little bit slower than ASPX View Engine.
Nothing like Test Driven Development	Good support for Test Driven Development.
Syntax for ASPX View Engine is inherited from Web Forms pages as: ***<%= employee.FullName %>***	Syntax for Razor View Engine is comparatively less and

	clean. ***@employee.FullName***

Back to top

Is it possible to remove the unused ViewEngine in ASP.NET MVC project?

Yes, it’s possible. As we know that by default two view engines are installed.

- WebForm ViewEngine
- Razor ViewEngine

If we are 100% sure that we are only going to use say ‘Razor ViewEngine’, and WebForm ViewEngine will remain idle/unused all the time, then we can remove the unused View engine by following the below simple steps:

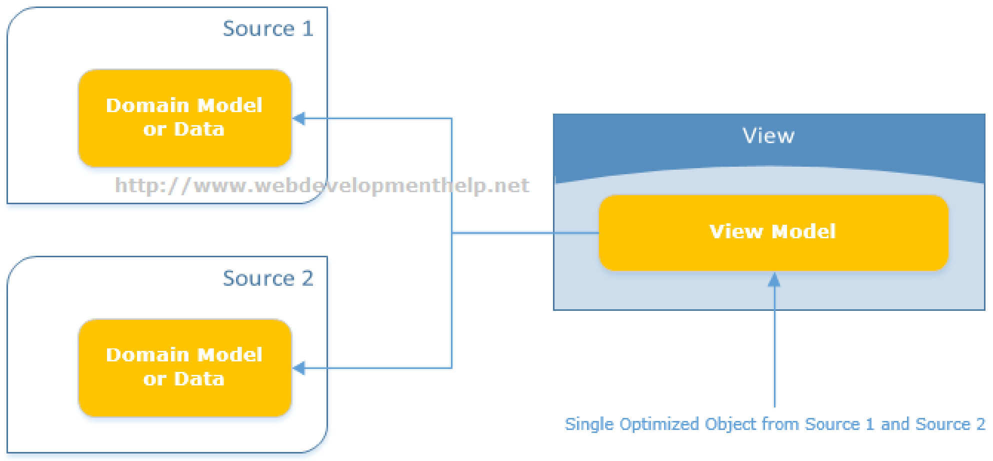
1. In our Gloabal.asax file, clear all the ViewEngines on Application_Start method.
2. Add the specific one i.e Razor ViewEngine to our ViewEngine collection.

```
ViewEngines.Engines.Clear();

ViewEngines.Engines.Add(new RazorViewEngine());
```

This will slightly improve the performance of our ASP.NET MVC application. The same approach we can use to add our own custom ViewEngine.

What is a ViewModel in ASP.NET MVC?

A ViewModel basically acts as a single model object for multiple domain models, facilitating to have only one optimized object for View to render. Below diagram clearly express the idea of ViewModel in ASP.NET MVC:


Source 1
Domain Model or Data
http://www.webdevelopmenthelp.net
Source 2
Domain Model or Data
View
View Model
Single Optimized Object from Source 1 and Source 2

There are multiple scenarios where using ViewModel becomes obvious choice. For example:

- Parent-Child View Scenario
- Reports where often aggregated data required
- Model object having lookup data
- Dashboards displaying data from multiple sources

Back to top

What are ASP.NET MVC HtmlHelpers?

ASP.NET MVC HtmlHelpers fulfills almost the same purpose as that of ASP.NET Web From Controls. For imlementation point of view, HtmlHelper basically is a method that returns a string (i.e. an HTML string to render HTML tags). So, in ASP.NET MVC we have HtmlHelpers for links, Images and for Html form elements etc. as follows:

```
@Html.ActionLink(“WebDev Consulting Company Profile”, “CompanyInfo”)

will render:

<a href=""/Site/CompanyInfo"">WebDev Consulting Company Profile</a>
```

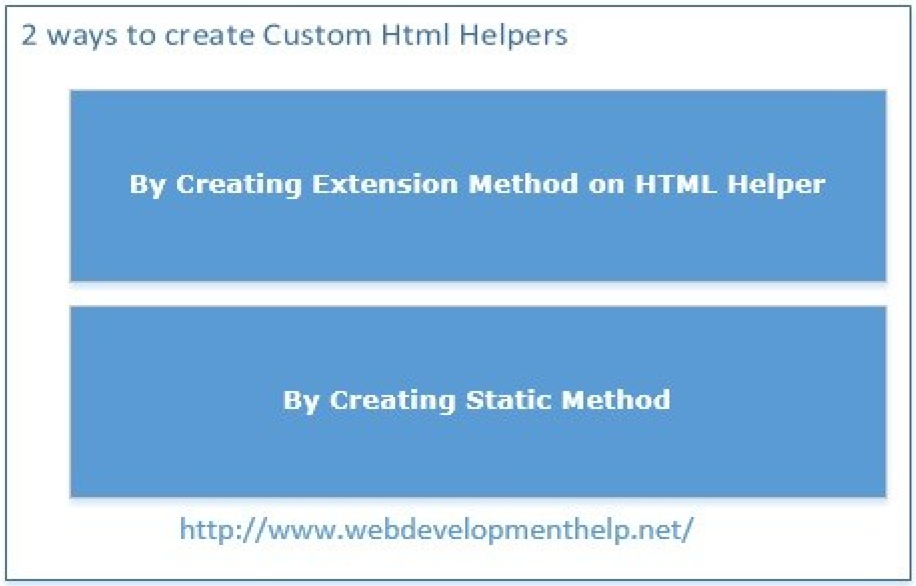
and

```
@Html.TextBox("strEmployeeName")

renders:

<input id=""strEmployeeName"" name=""strEmployeeName"" type=""text"" value="""" />
```

For a complete reference to Standard ASP.NET MVC Html Helpers, Click Here.

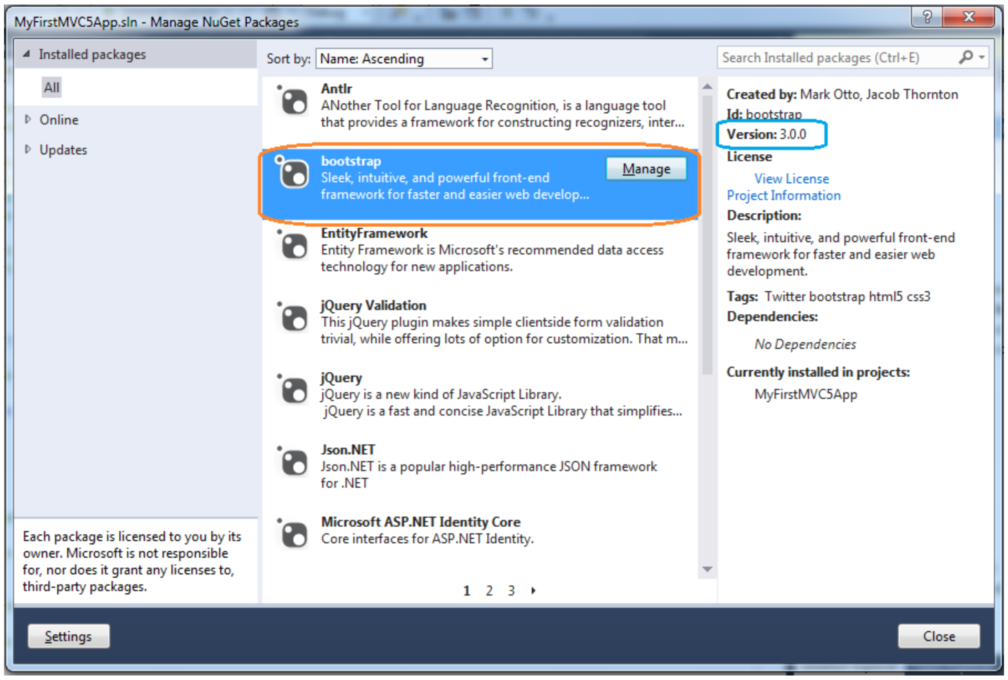
***Note:** Html Helpers are comparatively lightweight because these don't have ViewState and event model as for ASP.NET Web Form Controls.*

We can also create our Custom Html Helpers to fulfill specific application requirements. There are 2 ways to create custom HtmlHelpers as follows:


2 ways to create Custom Html Helpers
By Creating Extension Method on HTML Helper
By Creating Static Method
http://www.webdevelopmenthelp.net/
Back to top

What is Bootstrap in MVC5?

Bootstrap (a front-end framework) is an open source collection of tools that contains HTML and CSS-based design templates along with Javascript to create a responsive design for web applications. Bootstrap provides a base collection including layouts, base CSS, JavaScript widgets, customizable components and plugins. Project Template in ASP.NET MVC5 is now using bootstrap that enhances look and feel with easy customization. Bootstrap version 3 is added to ASP.NET MVC5 template as shown below:



Antlr
ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, inter...

bootstrap
Sleek, intuitive, and powerful front-end framework for faster and easier web develop...

EntityFramework
Entity Framework is Microsoft's recommended data access technology for new applications.

jQuery Validation
This jQuery plugin makes simple clientside form validation trivial, while offering lots of option for customization. That m...

jQuery
jQuery is a new kind of JavaScript Library.
jQuery is a fast and concise JavaScript Library that simplifies...

Json.NET
Json.NET is a popular high-performance JSON framework for .NET

Microsoft ASP.NET Identity Core
Core interfaces for ASP.NET Identity.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

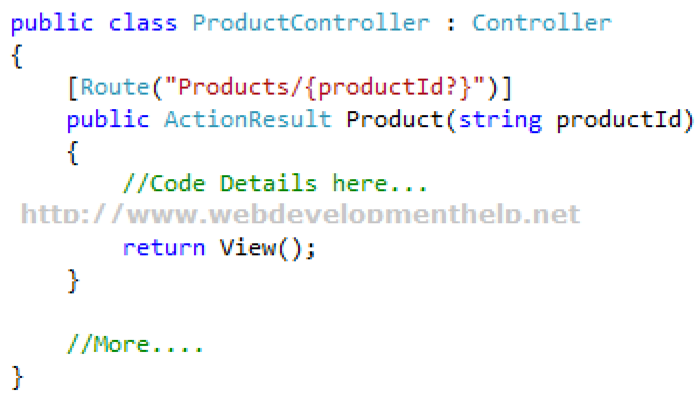
Created by: Mark Otto, Jacob Thornton
Id: bootstrap
Version: 3.0.0
License
View License
Project Information
Description:
Sleek, intuitive, and powerful front-end framework for faster and easier web development.
Tags: Twitter bootstrap html5 css3
Dependencies:
No Dependencies
Currently installed in projects:
MyFirstMVC5App

***Note**: You can follow here to learn **step by step how we can use any theme in Bootstraps to our ASP.NET MVC web application**. If you are further interested to learn Bootstrap in much detailed level, a very comprehensive course is available that teaches you **bootstrap by building 10 professional projects**.*

Back to top

Kindly explain Attribute Routing in ASP.NET MVC5?

We already have discussed about Routing in Question#6 that in ASP.NET MVC, we use friendly URLs that are mapped to controller's actions instead of physical files as in case of ASP.NET WebForms. Now in ASP.NET MVC5, we can use attributes to define routes giving better control over the URIs. We can easily define routes with Controller's action methods as follows:



```
public class ProductController : Controller
{
    [Route("Products/{productId?}")]
    public ActionResult Product(string productId)
    {
        //Code Details here...

        return View();
    }

    //More....
}
```

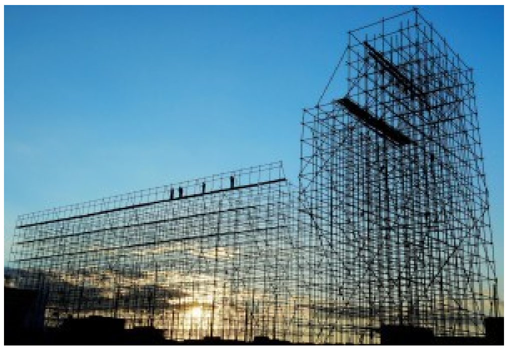
***Note**: Remember that conventional routing approach is not discarded, it's still there and fully functional. Also, we can use both routing techniques in a same application. Click here for more details on Attribute Routing.*

Back to top

What is Scaffolding in ASP.NET MVC? and what are the advantages of using it?

We (developers) spent most of our time writing code for CRUD operations that is connecting to a database and performing operations like Create, Retrieve, Update and Delete. Microsoft introduces a very powerful feature called Scaffolding that does the job of writing CRUD operations code for us.

Scaffolding is basically a Code Generation framework. Scaffolding Engine generates basic controllers as well as views for the models using Micrsoft's T4 template. Scaffolding blends with Entity Framework and creates the instance for the mapped entity model and generates code of all CRUD Operations. As a result we get the basic structure for a tedious and repeatative task. You can find a detailed Web Development Tutorial with implementation on ASP.NET MVC Scaffolding here.



Following are the few **advantages of Scaffolding**:

- RAD approach for data-driven web applications.
- Minimal effort to improve the Views.

- Data Validation based on database schema.
- Easily created filters for foreign key or boolean fields.

Back to top

_Another Question about Handling Controller for Real World Scenario.

Consider you are working as MVC Developer with WebDevTutorials. Our ASP.NET MVC application has a controller named EmployeeController. We want our EmployeeController to have a public non action method. What we will do to achieve this?

- A. We can't make a public method as non action in ASP.NET MVC.
- B. Make return type void to a public method.
- C. Add NonActionAttribute attribute to a public method.
- D. Add VoidAttribute attribute to a public method.

*Want to improve you ASP.NET MVC skill, Take a Complete **FREE Online ASP.NET MVC Test** or MCSD Practice Exam: 70-486 (**Developing ASP.NET MVC Web Applications**). Simply Click Here.*

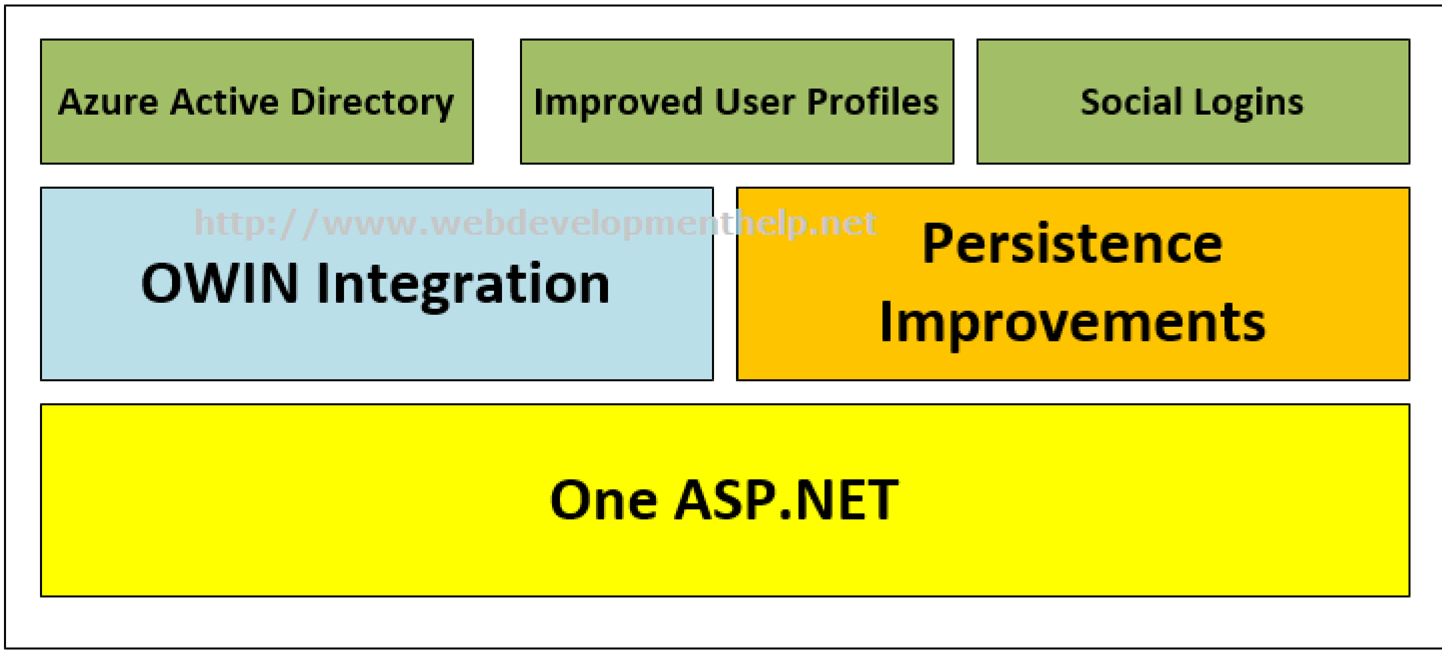
Correct Answer: C

Briefly explain ASP.NET Identity?

Microsoft introduces **ASP.NET Identity** as a system to manage access in ASP.NET application on premises and also in the cloud. There were issues with **Membership Provider Model** especially when we want to implement more advanced security features in our applications, so ASP.NET Identity gets away from the membership provider model. If we look into the history of membership, its like follows:

- ASP.NET 2.0 Membership (VS 2005)
  - Forms Authentication
  - Sql Server Based Membership
- ASP.NET Simple Membership (VS 2010)
  - Easy to customize profile
  - ASP.NET Web Pages
- ASP.NET Universal Providers (VS2012)
  - Support Sql Azure

ASP.NET Identity is much improved system to manage access to our application and services.



For more information on ASP.NET Identity, please follow here. Back to top

Is it possible in ASP.NET MVC Project to boost speed by caching the static content folder(s)?

Yes, we can easily boost speed for static content folder(s) by applying caching on specific content folder. It's a simple 2 step process as follows:

1. Place the web.config inside the content folder, and
2. Set the cacheControlMaxAge property to specific value you want.

It's an older approach, we have been using since long with apply folder specific settings.

Complete code for applying caching to boost speed for static content folders as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
      <staticContent>
          <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="3.00:00:00" />
```

```
    </staticContent>
  </system.webServer>
</configuration>
```

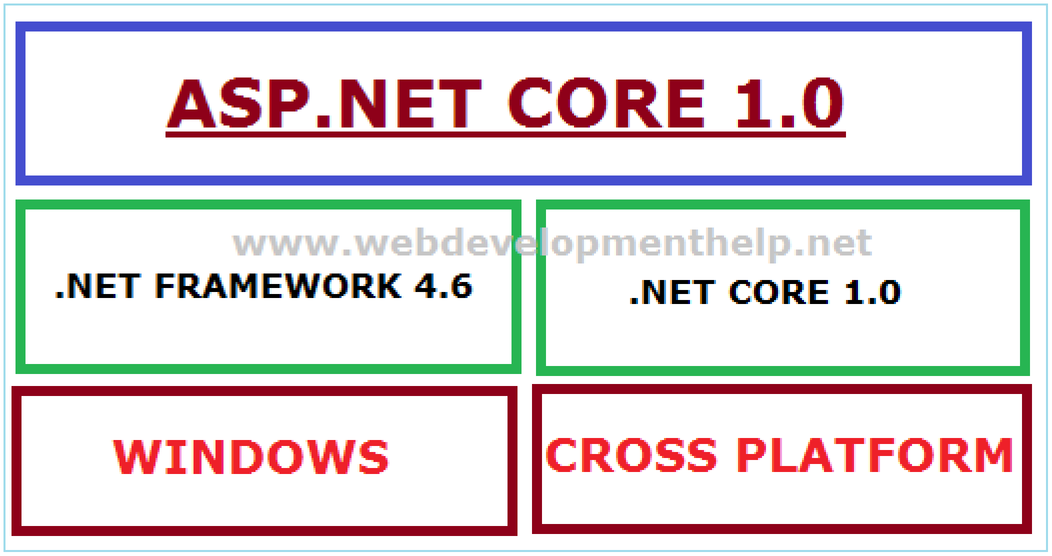
This will inform the client to cache the static content for 3 days in that folder as well as the subfolders under it.

ASP.NET CORE Interview Questions – ASP.NET Core 1.0 – MVC6

What is ASP.NET Core?

ASP.NET Core 1.0 also known as ASP.NET 5 – MVC6 is a re-write version of ASP.NET 4.6 with following key features:

- ASP.NET Core is Open-Source.
- ASP.NET Core is Cross-Platform framework means can run on Linux or MacOS – a huge enhancement.
- Ideal for building web applications, Iot apps and mobile backends.
- Cloud ready configuration.
- ASP.NET Core 1.0 is smaller in size as compared to ASP.NET 4.6 framework as it runs on stripped down version of .NET Framework i.e. .NET Core 1.0



At a broader view, ASP.NET 5 is now renamed as ASP.NET Core 1.0; .NET Framework 5 will be known as .NET Core 1.0 and Entity Framework 7 will be renamed to Entity Framework Core 1.0.
Back to top

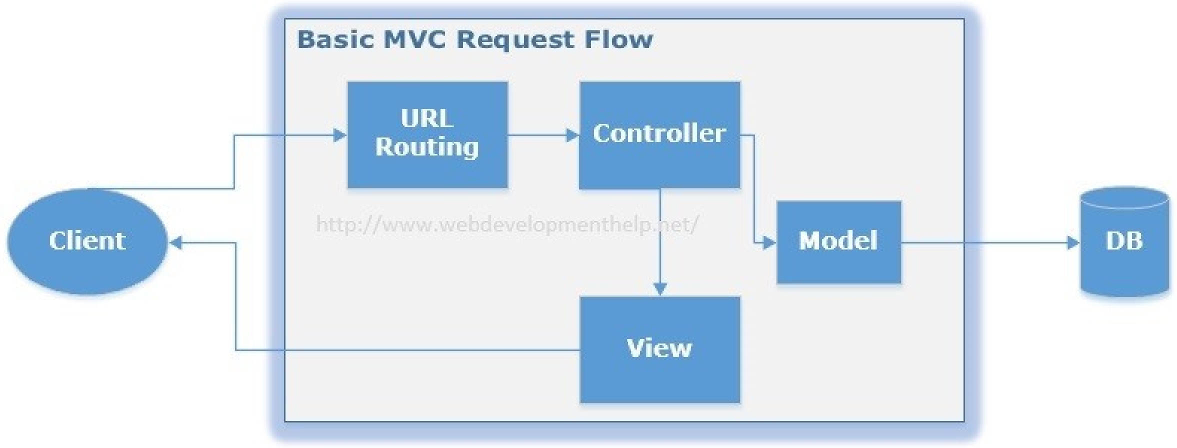
ASP.NET MVC is an amazing framework for developing applications. Above mentioned **ASP.NET MVC interview questions** must be prepared before appearing for a MVC interview.

ASP.NET MVC Passing data from Controller to View

By IMRAN ABDUL GHANI | June 10, 2014

5 Comments

ASP.NET MVC is a framework that facilitates building web applications based on MVC (Model-View-Controller) design pattern. Request coming from client reaches the *Controller* through URL Rewriting Module. *Controller* decides which model to use in order to fulfill the request. Further passing the *Model* data to *View* which then transforms the *Model* data and renders response to client as shown in following basic level request flow diagram.



*You can access to Complete ASP.NET MVC Tutorial Series on building **Online MVC Shopping Cart Application** with **source code** here.*

In this **ASP.NET MVC Tutorial**, we will discuss and implement different options to pass data from ASP.NET MVC *Controller* to *View*.

Let's begin by answering a simple ASP.NET MVC Question

You are working as ASP.NET MVC developer at WebDevTutorials and developing a library that supports multiple ASP.NET MVC web applications on a shared server. This library provides implementations of security algorithms. If a problem with any of the security algorithms is discovered, a newer version of the library must be created and deployed. Application downtime during the update must be minimized. As a responsible developer, you need to ensure that the new version of the library will be used by all applications as soon as possible. What should you do?

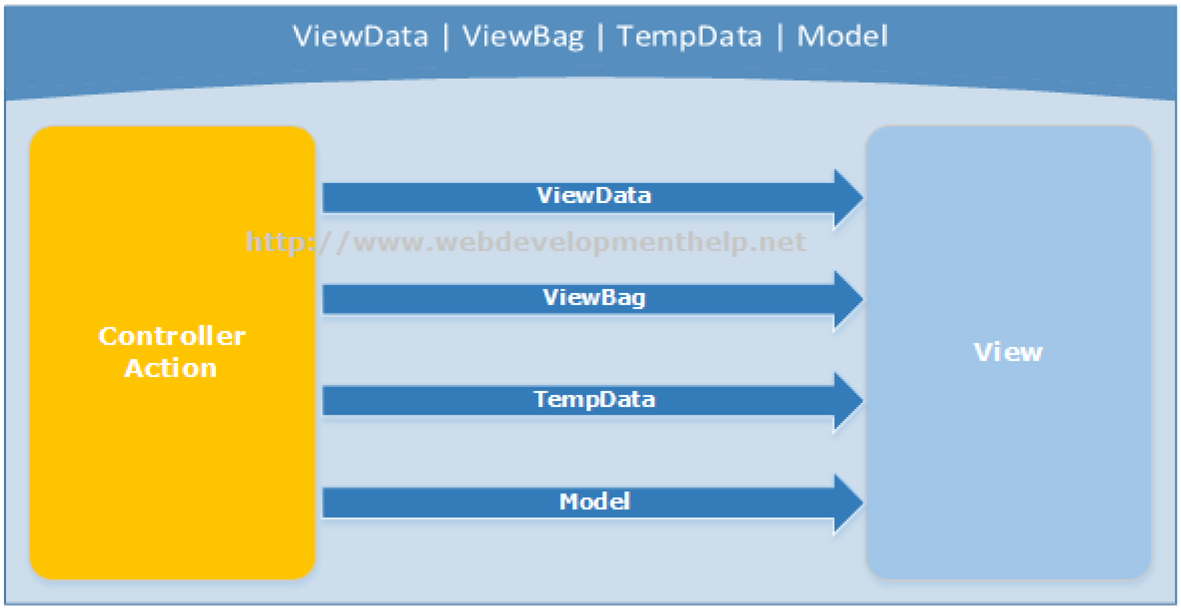
- A. Build the web applications and include the security assembly as an embedded resource. When an update is needed, copy the new assembly to the bin directory for the application.
- B. Install the security assembly in the Global Assembly Cache (GAC). When an update is needed, update the assembly in the GAC.
- C. Build the security assembly as a netmodule in a shared location. Use the assembly linker to merge the netmodule into the assemblies for the application. When an update is needed, update the netmodule in the shared location.
- D. Sign all assemblies in each application with the same key used to sign the security assembly. When an update is needed, create a new key pair and re-sign all assemblies.

For a complete ASP.NET MVC online test and Practice Exams, Click Here.

Correct Answers: B

Following are the different available options to pass data from a *Controller* to *View* in ASP.NET MVC along with Introduction and Background:

- Introduction and Background
- ViewBag
- ViewData
- TempData (TempData Vs Session)
- Model


ViewData | ViewBag | TempData | Model
Controller Action
ViewData
http://www.webdevelopmenthelp.net
ViewBag
TempData
Model
View

Let's discuss all these options step by step in this series.

Introduction

If we want to maintain state between a *Controller* and corresponding *View*– ***ViewData*** and ***ViewBag*** are the available options but both of these options are limited to a single server call (meaning it's value will be null if a redirect occurs). But if we need to maintain state from one *Controller* to another (redirect case), then ***TempData*** is the other available option.

It’s common that initially it might be a bit difficult for a ASP.NET WebForms developer to digest above flow and need for options to pass data from *Controller* to *View.* Because in WebForms approach, Controller and View are tightly coupled to each other. Please follow the link for a detailed comparison of the differences between ASP.NET WebForms and ASP.NET MVC here.

For the purpose of implementation, we will take earlier ASP.NET MVC tutorial on this blog as base and implement with different options. If you haven’t gone through the article, please read “Building your first ASP.NET MVC application in 4 simple steps” first.

ViewBag Example

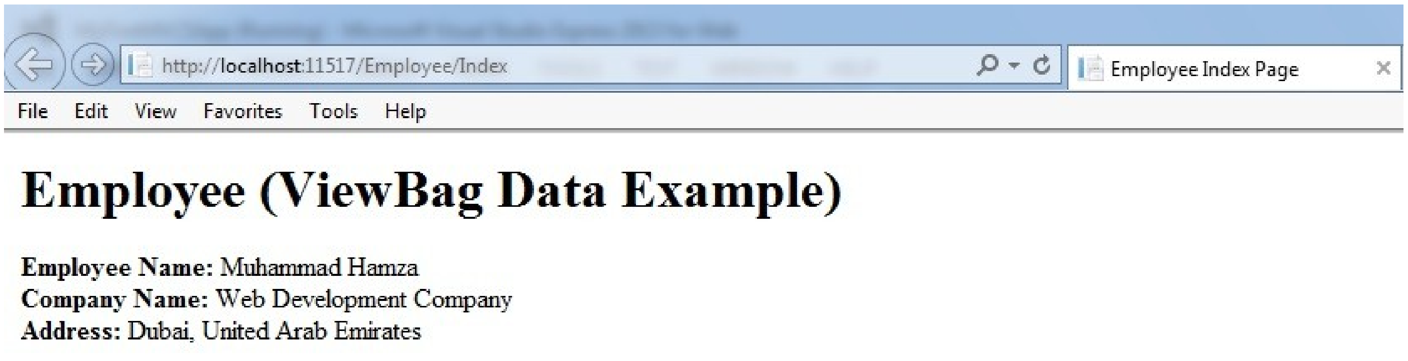
As we discussed earlier that *ViewBag* and *ViewData* serves the same purpose but *ViewBag* is basically a dynamic property (a new C# 4.0 feature) having advantage that it doesn’t have typecasting and null checks.
So, In order to pass data from Controller to View using ViewBag, we will modify our EmployeeController code as follows:

```
public class EmployeeController : Controller
{
         // GET: /Employee/
        public ActionResult Index()
       {
                  ViewBag.EmployeeName = “Muhammad Hamza”;
                  ViewBag.Company = “Web Development Company”;
                  ViewBag.Address = “Dubai, United Arab Emirates”;

                  return View();
       }
 }
```

And to get Employee details passed from Controller using ViewBag, View code will be as follows:

```
 <body>
   <div>
        <h1>Employee (ViewBag Data Example)</h1>
        <div>
                 <b>Employee Name:</b> @ViewBag.EmployeeName<br />
                <b>Company Name:</b> @ViewBag.Company<br />
                <b>Address:</b> @ViewBag.Address<br />
        </div>
   </div>
 </body>
```

In order to see the above changes in action run the solution, we will find the following output.


http://localhost:11517/Employee/Index
Employee Index Page
File Edit View Favorites Tools Help

Employee (ViewBag Data Example)

Employee Name: Muhammad Hamza
Company Name: Web Development Company
Address: Dubai, United Arab Emirates

Top Courses on ASP.NET MVC and related

- **Learn ASP NET MVC 5 step by step** [Maruti Makwana, Corporate Trainer] 28 Lectures, 2.5 Hours Video, Intermediate Level
  Very easy to learn video series on Asp.Net MVC 5 Specially for those who are familiar with Asp.Net Web forms.
- **AngularJS for ASP.NET MVC Developers** [Brett Romero] 10 Lectures, 1 hour video, Intermediate Level
  The Fastest Way For .NET Developers To Add AngularJS To Their Resume
- **ASP.NET with Entity Framework from Scratch**
  [Manzoor Ahmad, MCPD | MCT] 77 Lectures, 10 hour video, All Level
  Latest approach of web application development
- **Comprehensive ASP.NET MVC** [3D BUZZ] 34 lectures, 14 Hours Video, All Levels
  From zero knowledge of ASP.NET to deploying a complete project to production.

ViewData Example

As compared to ViewBag, ViewData is a dictionary object which requires typecasting as well as null checks. Same above implementation using ViewData can be achieved as follows:

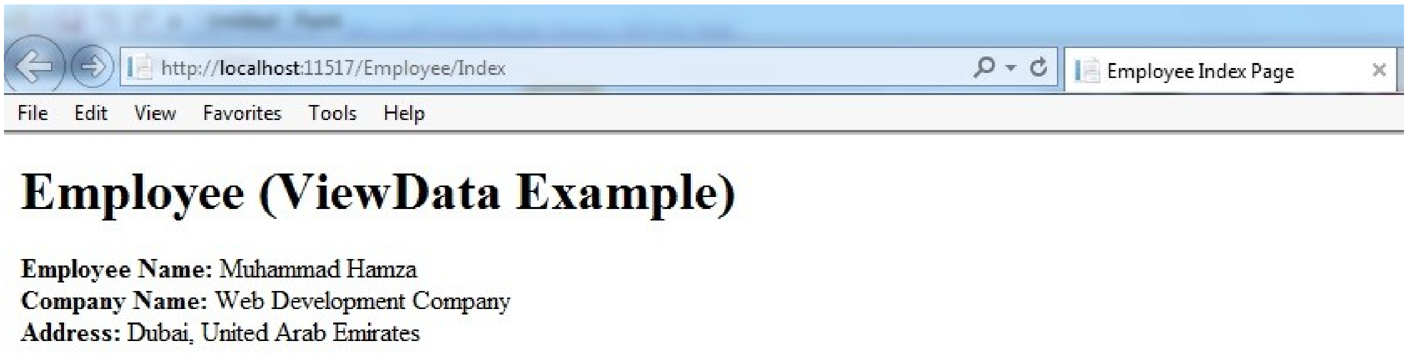
```
public class EmployeeController : Controller
{
        // GET: /Employee/
       public ActionResult Index()
      {
                ViewData["EmployeeName"] = "Muhammad Hamza";
                ViewData["Company"] = "Web Development Company";
                ViewData["Address"] = "Dubai, United Arab Emirates";

                return View();
      }
 }
```

And to get Employee details passed from Controller using ViewBag, View code will be as follows:

```
 <body>
   <div>
       <h1>Employee (ViewBag Data Example)</h1>
       <div>
              <b>Employee Name:</b> @ViewData["EmployeeName"]<br />
```

```
                <b>Company Name:</b> @ViewData["Company"]<br />
                <b>Address:</b> @ViewData["Address"]<br />
        </div>
   </div>
</body>
```

Run the application to view the following output.


http://localhost:11517/Employee/Index
Employee Index Page
File Edit View Favorites Tools Help

Employee (ViewData Example)

Employee Name: Muhammad Hamza
Company Name: Web Development Company
Address: Dubai, United Arab Emirates

Hopefully, this ASP.NET MVC Tutorial will provide reader with a better understanding of passing data from *Controller* to *View* in ASP.NET MVC using ViewBag and ViewData. Please follow the link for detailed understanding about Using TempData in ASP.NET MVC.

ASP.NET MVC TempData & Session

By IMRAN ABDUL GHANI | June 11, 2014

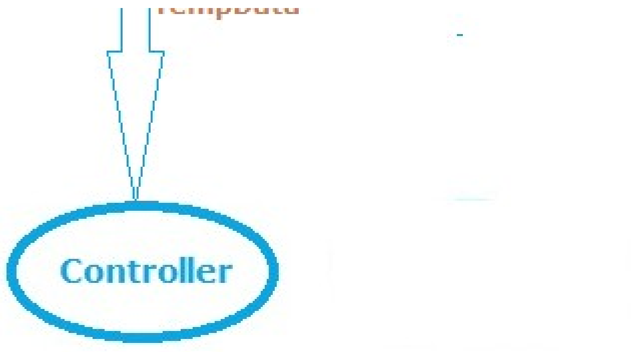
0 Comments

In previous ASP.NET MVC Tutorial, we discussed about different available options for passing data from Controller to View in ASP.NET MVC. We implemented passing data using *ViewBag* and *ViewData* in ASP.NET MVC application. Now, in this part of the tutorial, we are going to discuss about the third available option i.e. *TempData*.

- ASP.NET MVC Online Shopping Store Application using C#, Entity Framework, SQL Server
- FREE Online Tests (C# | ASP.NET MVC | ASP.NET | AJAX | jQuery)

ASP.NET MVC TempData Tutorial

TempData in ASP.NET MVC is basically a dictionary object derived from TempDataDictionary. TempData stays for a subsequent HTTP Request as opposed to other options (ViewBag and ViewData) those stay only for current request. So, TempdData can be used to maintain data between controller actions as well as redirects.



Note: *Just like ViewData, typecasting and null checks required for TempData also in order to avoid errors.*

Let’s see how we can use TempData in a practical scenario to pass data from one controller action to another.

```
//Controller Action 1 (TemporaryEmployee)
 public ActionResult TemporaryEmployee()
{
               Employee employee = new Employee
               {
                      EmpID = "121",
                      EmpFirstName = "Imran",
                EmpLastName = "Ghani"
            };

            TempData["Employee"] = employee;
            return RedirectToAction("PermanentEmployee");
}

 //Controller Action 2(PermanentEmployee)

 public ActionResult PermanentEmployee()
{
           Employee employee = TempData["Employee"] as Employee;
           return View(employee);
 }
```

As in above example, we store an employee object in TempData in Controller Action 1 (i.e. TemporaryEmployee) and retrieve it in another Controller Action 2 (i.e. PermanentEmployee). But If we try to do the same using ViewBag or ViewData, we will get null in Controller Action 2 because only TempData object maintains data between controller actions.

An important thing about ***TempData*** is that it stores contents in Session object. Then one may raise a question that "*What's the difference between TempData in ASP.NET MVC and Session?*" or "*Why we have TempData dictionary object?, Why can't we use the Session object directly?*"

ASP.NET MVC TempData Vs Sessions

Although ASP.NET MVC TempData stores it's content in Session state but it gets destroyed earlier than a session object. TempData gets destroyed immediately after it's used in subsequent HTTP request, so no explicit action required. If we use a Session object for this purpose, we would have to destroy the object explicitly.

It's best fit for scenarios when:

- we need to preserve data from current to subsequent request.
- passing error message to an error page.

With the above discussion on using TempData in ASP.NET MVC, we have successfully covered different options for passing data from Controller to View in ASP.NET MVC. Hopefully reader will have a better understanding of using ViewBag, ViewData and TempData in ASP.NET MVC.
Previous: ViewBag and ViewData in ASP.NET MVC

Using ASP.NET MVC TempData and Session to pass values across Requests

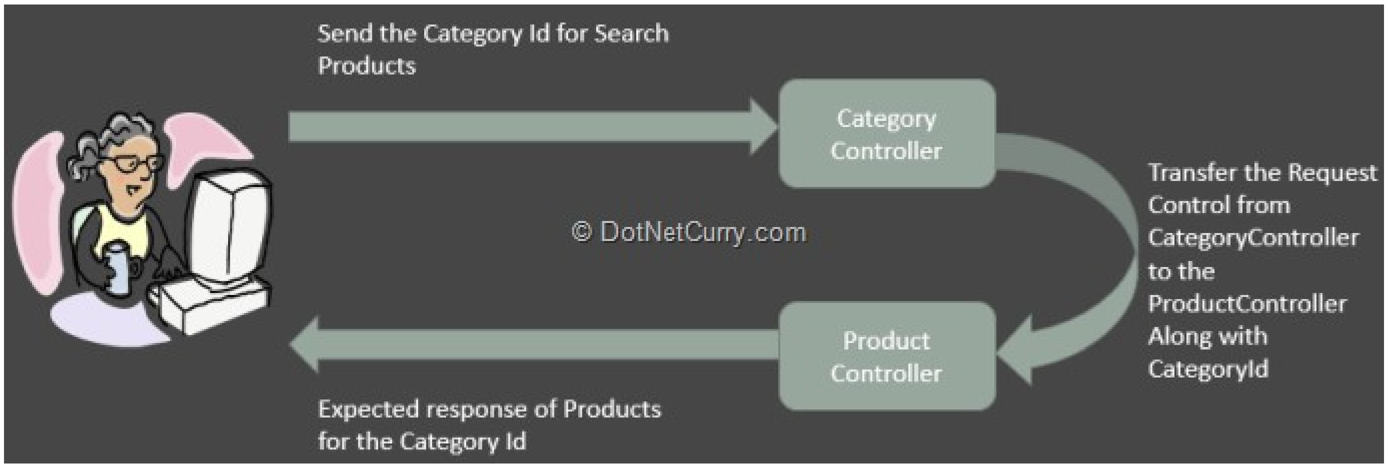
Posted by: Mahesh Sabnis , on 1/19/2015, in **Category** ASP.NET MVC

Views: 117310

Abstract: In ASP.NET MVC business applications, you may need to maintain the state of the data across requests. This article demonstrates how to use TempData and Session to achieve this requirement.

ASP.NET has numerous useful features and one of it is state management. ASP.NET Web Form developers have the habit of using Session objects for passing values across pages. Since in ASP.NET, everything is controlled by *Page,* the use of various state management techniques like ViewState, Cookies, Sessions, etc. play an important role here. But unlike ASP.NET Web Forms, in ASP.NET MVC the *Controller* decides what values are accepted from a view and which view to send the response to. The *Controller* in MVC is the major component for request processing and has its own lifecycle. Consider a scenario where one controller has an action method which returns the action method for another controller. During this controller transition, value entered from one controller needs to be used by another controller for some processing. In this case, it is mandatory for the developer to make use of some mechanism to maintain the state of the value across various controllers.

In MVC the following image explains the scenario:


Send the Category Id for Search Products
Category Controller
© DotNetCurry.com
Transfer the Request Control from CategoryController to the ProductController Along with CategoryId
Product Controller
Expected response of Products for the Category Id

Here the question is how to implement this? Since ASP.NET MVC builds on the top of the ASP.NET framework, we have access to the **Session** objects derived from **HttpSessionBase.** Like ASP.NET WebForms, in MVC the*Session* property is provided by *HttpContext* class. But MVC has its own *TempData* object which is used to pass data across controllers. This is derived from *TempDataDictionary***.** The *TempData* has a very short life and sets itself to *null* when the target view is fully loaded. Here are some features of TempData and Session objects.

TempData:

- is the property of the ControllerBase class

- is used to pass data from one controller to other
- its life is ended when the target view is loaded completely
- it is recommended to type-cast the value when TempData value is used for processing
- Mostly used to store Validation Error Messages, etc.

The Life time of the TempData can be managed using the *Keep()* method.

Session:

- is used to pass data across controllers in MVC Application.
- Session data never expires.

Like TempData the value must be type-casted when it is used for processing.

Implementating TempData

Step 1: Download the free Visual Studio 2013 Community Edition (the article uses VS2013 Ultimate with Update 4) and create a new empty MVC application of the name *MVC5_Sessions*. In this project in App_Data folder add a new Sql Server database of the name *Application*. In this database add the following tables:

Category

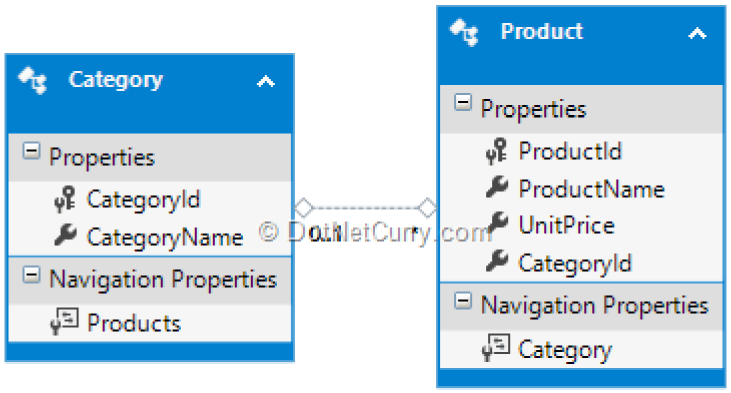
```
CREATE TABLE [dbo].[Category]

(

    CategoryId int primary key Identity,

    CategoryName varchar(100) Not Null

)
```

Product

```
CREATE TABLE [dbo].[Product]

(

    ProductId INT NOT NULL PRIMARY KEY identity,

    ProductName varchar(100) Not Null,

    UnitPrice decimal Not Null,
    CategoryId int References Category (CategoryId)
)
```

Add some sample data in these tables.

Step 2: In the Models folder add a new ADO.NET Entity Data Model. In the wizard select the Application database we added in Step 1. Select all tables from the database. After completing the wizard, the following mapping will be displayed:



Using TempData to pass values across controllers

Step 3: In the Controllers add a new Empty MVC Controller of the name *CategoryController* .In this controller, we already have an *Index* method along with the add *SerachProducts* action method with HttpGet and HttpPost as below:

```
using System.Linq;
using System.Web.Mvc;


using MVC5_Sessions.Models;


namespace MVC5_Sessions.Controllers
{
public class CategoryController : Controller
{
    ApplicationEntities ctx;
    public CategoryController()
    {
        ctx = new ApplicationEntities();
    }

    // GET: Category
    public ActionResult Index()
    {
        var cats = ctx.Categories.ToList();
        return View(cats);
    }

    public ActionResult SearchProducts()
    {
        return View();
    }


    [HttpPost]
    public ActionResult SearchProducts(Category c)
    {
       TempData["Category"] = c;
        return RedirectToAction("Index","Product");
    }
}
}
```

The above code has the *SearchProducts* action method for [HttpPost]. This method declares the TempData object with key as ‘Category’. This object is set to the Category object passed to the *SearchProducts* action method. The SearchProducts action method returns the Index method of the ProductController using the below statement:

```
return RedirectToAction("Index","Product");
```

Step 3: Scaffold Index.cshtml and SearchProducts.cshtml from the Index and Search Products action method. Change the SearchProducts.cshtml to the following:

```
@model MVC5_Sessions.Models.Category


@{

    ViewBag.Title = "SearchProducts";

}


<h2>SearchProducts</h2>




@using (Html.BeginForm())

{

    @Html.EditorFor(cat=>cat.CategoryId);


         <div class="form-group">

            <div class="col-md-offset-2 col-md-10">

                <input type="submit" value="Search" class="btn btn-default" />

            </div>

        </div>

}
```

The above view has a textbox which is bound with the CategoryId of the Category Model class.

Step 4: In the Controllers folder, add a new Empty MVC controller of the name ProductController and add the following code in it:

```
using System.Linq;

using System.Web.Mvc;

using MVC5_Sessions.Models;
```

```
namespace MVC5_Sessions.Controllers
{
    public class ProductController : Controller
    {

        ApplicationEntities ctx;
        public ProductController()
        {
            ctx = new ApplicationEntities();
        }

        // GET: Product
        /// <summary>
        /// Type Case the TempData to Category
        /// </summary>
        /// <returns></returns>
        public ActionResult Index()
        {
            if (TempData["Category"] != null)
            {
                var Cat = (Category)TempData["Category"];

                var Products = from p in ctx.Products
                               where p.CategoryId == Cat.CategoryId
                               select p;



                return View(Products.ToList());
            }
            else
            {
                return View(ctx.Products.ToList());
            }
        }

        [HttpPost]
        public ActionResult Index(int id=0)
        {
            var Data = TempData["Category"] as Category;

            return RedirectToAction("Index");
        }
    }
}
```

The above code has Index action methods for HttpGet and HttpPost. The HttpGet Index method accepts the TempData[“Category”] object and type casts it to Category. The code further queries the Products based upon the CategoryId and returns Products under the category.

Step 5: Scaffold the Index.cshtml from the Index action method and make the following changes in it:

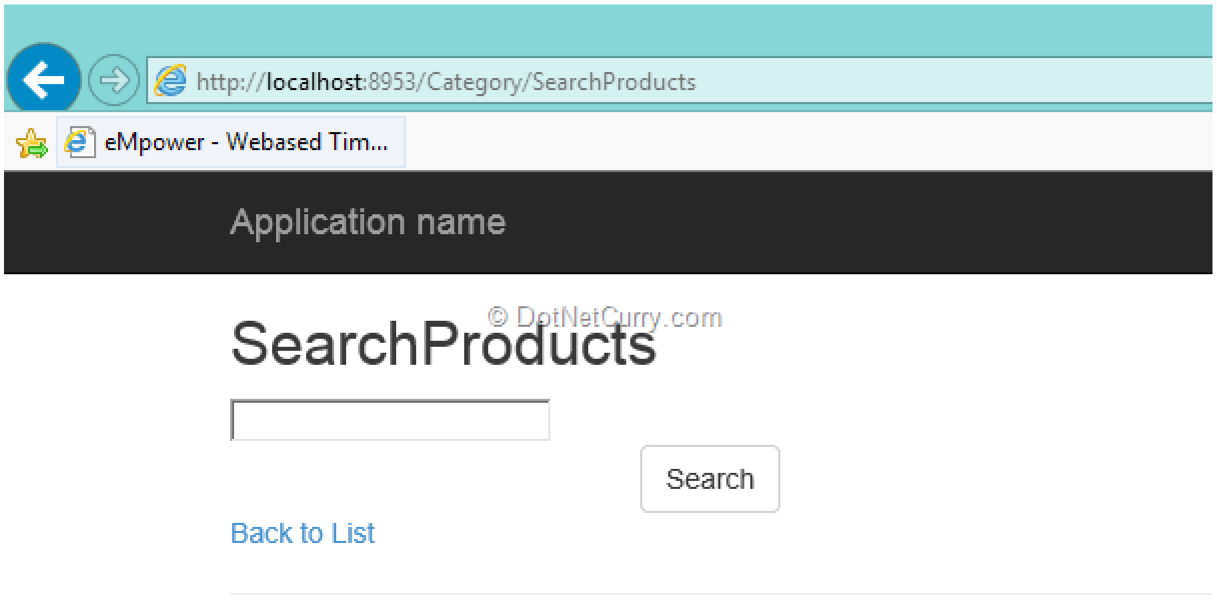
```
@model IEnumerable<MVC5_Sessions.Models.Product>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

```
<p>

    @Html.ActionLink("Create New", "Create")

</p>

<table class="table">

    <tr>

        <th>

            @Html.DisplayNameFor(model => model.ProductId)

        </th>

        <th>

            @Html.DisplayNameFor(model => model.ProductName)

        </th>

        <th>

            @Html.DisplayNameFor(model => model.UnitPrice)

        </th>

        <th>

            @Html.DisplayNameFor(model => model.Category.CategoryName)

        </th>

        <th></th>

    </tr>


@foreach (var item in Model) {

    <tr>

        <td>

            @Html.DisplayFor(modelItem => item.ProductId)

        </td>

        <td>

            @Html.DisplayFor(modelItem => item.ProductName)

        </td>

        <td>

            @Html.DisplayFor(modelItem => item.UnitPrice)
        </td>

        <td>

            @Html.DisplayFor(modelItem => item.Category.CategoryName)

        </td>

        <td>

            @Html.ActionLink("Edit", "Edit", new { id=item.ProductId }) |

            @Html.ActionLink("Details", "Details", new { id=item.ProductId }) |

            @Html.ActionLink("Delete", "Delete", new { id=item.ProductId })

        </td>

    </tr>

}


</table>


@using(Html.BeginForm())

{

    <input type="submit" value="Post" />

}
```

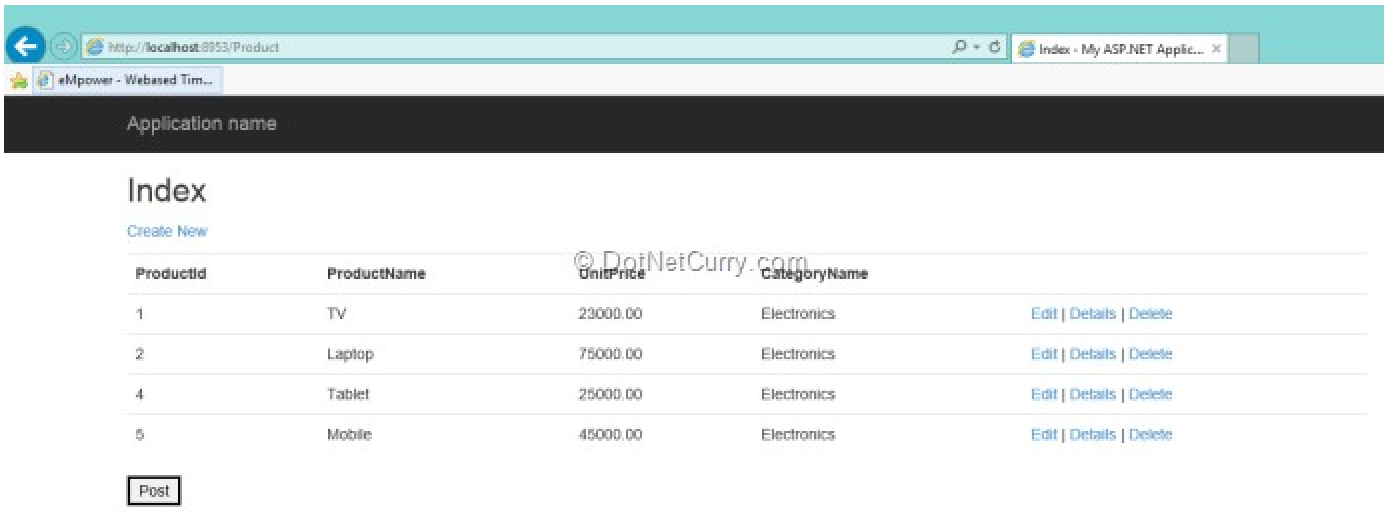
When the submit button is clicked, it posts to the Index action method with HttpPost. Apply breakpoint on the HttpPost Index action method to check the value in the TempData[“Category”] when the end-user posts to the Index view.

Run the Application and navigate to the SearchProducts action of the Category Controller: http://<server>/Category/SearchProducts

The following view will be displayed:



Enter CategoryId in the TextBox and click on the ‘Search’ button. The Product Index view will be displayed:



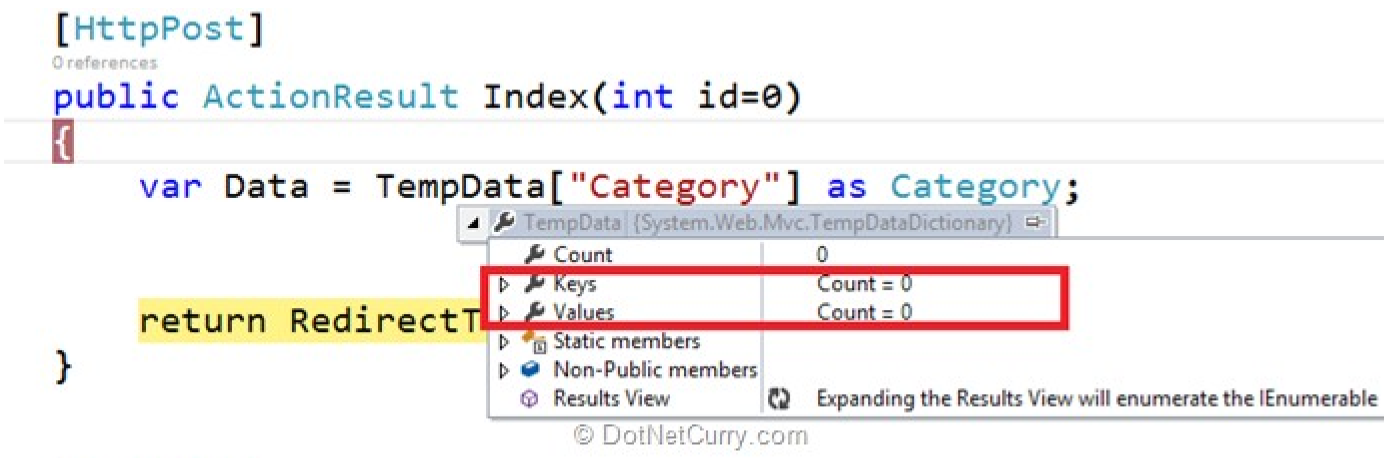
Index

Create New

ProductId	ProductName	UnitPrice	CategoryName	
1	TV	23000.00	Electronics	Edit | Details | Delete
2	Laptop	75000.00	Electronics	Edit | Details | Delete
4	Tablet	25000.00	Electronics	Edit | Details | Delete
5	Mobile	45000.00	Electronics	Edit | Details | Delete

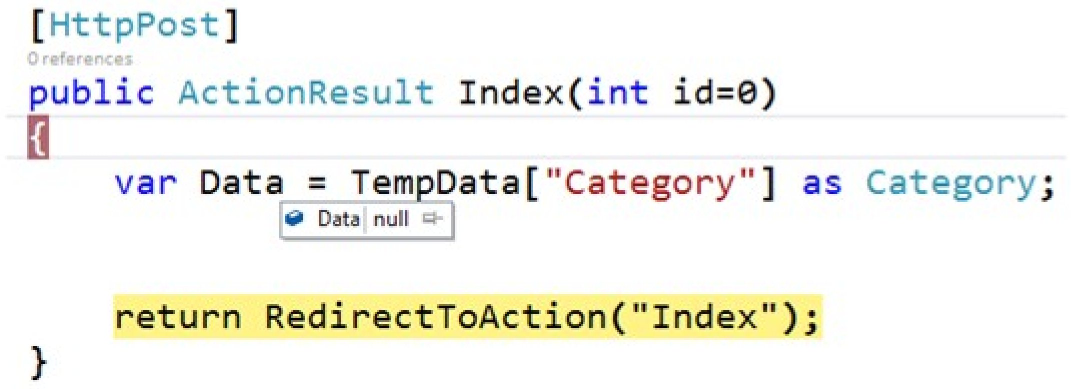
Post

Click on the ‘Post’ button which will enter in the debug-mode to the HttpPost Index action method. We have already applied the breakpoint on this Index method. Here check the value of the TempData[“Category”]. It will display the values in the TempData[“Category”] as zero (0) and the data will be null as shown here:



```
[HttpPost]
0 references
public ActionResult Index(int id=0)
{
    var Data = TempData["Category"] as Category;
    ◢ TempData {System.Web.Mvc.TempDataDictionary}
        Count              0
      ▷ Keys               Count = 0
      ▷ Values             Count = 0
      ▷ Static members
      ▷ Non-Public members
        Results View       Expanding the Results View will enumerate the IEnumerable

    return RedirectT
}
```

© DotNetCurry.com



```
[HttpPost]
0 references
public ActionResult Index(int id=0)
{
    var Data = TempData["Category"] as Category;
          Data  null

    return RedirectToAction("Index");
}
```

(Note: These are two different images showing debug from LHS and RHS of this expression)

The value from the TempData["Category"] is null because the TempData gets killed when the response is completely loaded.

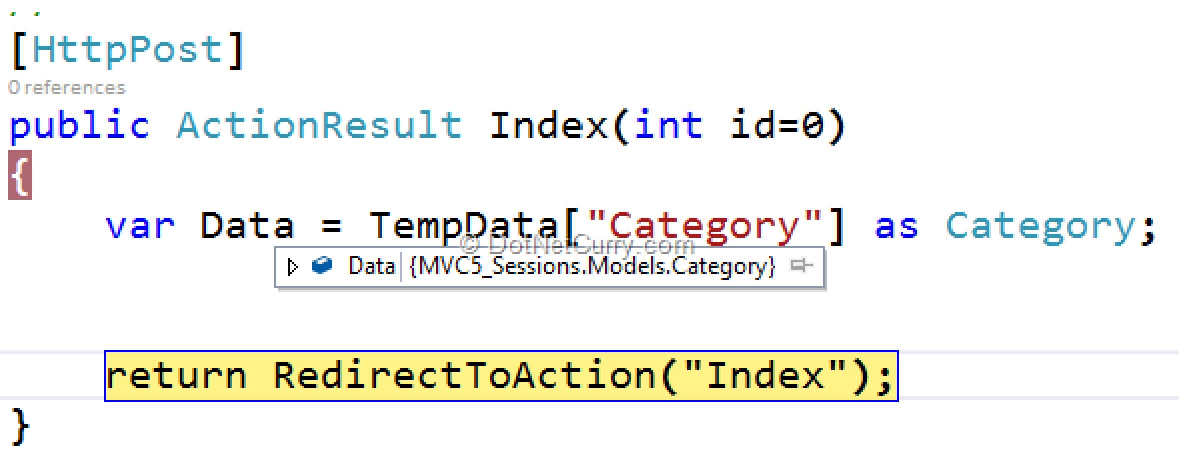
Step 6: To maintain the life of the TempData["Category"], make the following change in the HttpGet Index Action method of the ProductController:

```
public ActionResult Index()

{

    if (TempData["Category"] != null)

            {

        var Cat = (Category)TempData["Category"];

        var Products = from p in ctx.Products

                       where p.CategoryId == Cat.CategoryId

                       select p;


        TempData.Keep("Category");
```

```
            return View(Products.ToList());
        }
        else
        {
            return View(ctx.Products.ToList());
        }
    }
```

The TempData.Keep(“Category”) will maintain the state of the data now even when the response view is loaded.

Step 7: Repeat Step 5. You will see the TempData[“Category”] is maintained as shown here:



```
[HttpPost]
0 references
public ActionResult Index(int id=0)
{
    var Data = TempData["Category"] as Category;
               Data {MVC5_Sessions.Models.Category}

    return RedirectToAction("Index");
}
```

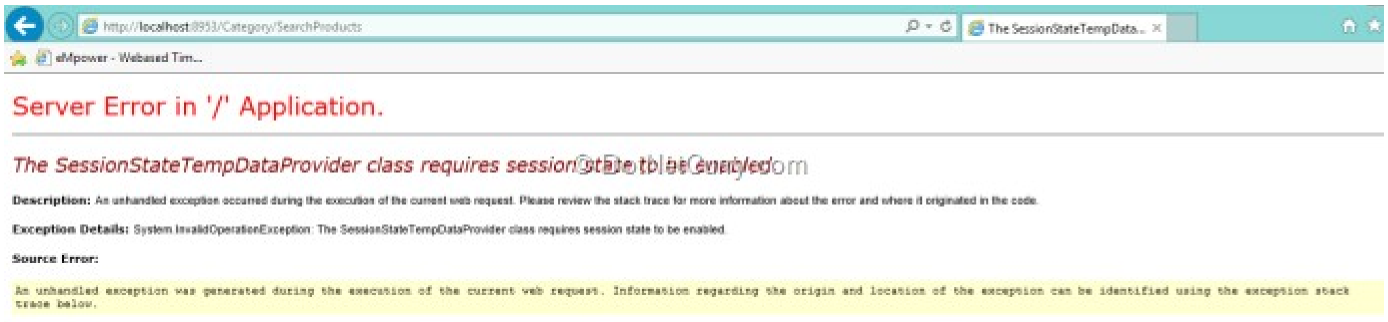
That’s it, in MVC we can make use of TempData to maintain the state of the values across the requests using which data can be passed across controllers.

Something very Important about TempData and Session

Since TempData makes use of the Session State behavior, it must be *enabled* on the controller using TempData. By default it is always enabled, however using code, the session state can be disabled on the controller. In the CategoryController class, add the following attribute:

```
[SessionState(System.Web.SessionState.SessionStateBehavior.Disabled)]
public class CategoryController : Controller
{
```

The above code disables the session state on request processing. Run the application and navigate to the Category/SearchProducts View, enter CategoryId in the TextBox, and click on the 'Search' button. The following page gets displayed:



http://localhost:8953/Category/SearchProducts
The SessionStateTempData...
eMpower - Webased Tim...

Server Error in '/' Application.

The SessionStateTempDataProvider class requires session state to be enabled.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.InvalidOperationException: The SessionStateTempDataProvider class requires session state to be enabled.

Source Error:

```
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack
trace below.
```

The above error clearly shows that 'SessionStateTempDataProvider' requires the Session State to be enabled.

The Same demo can be implemented using Session["Category"] object to pass data across controllers.

Conclusion: While developing business applications using ASP.NET MVC, we may need to maintain the state of the data across requests. TempData is your friend in such scenarios.

http://www.dotnetcurry.com/aspnet-mvc/1074/aspnet-mvc-pass-values-temp-data-session-request