**[](https://www.c-sharpcorner.com/technologies/asp-dot-net-programming)Basics Of ASP.NET MVC In Depth**

**Thursday, March 29, 2018**

**Synopsis**

1. Introduction
2. Definition
3. Advantages
4. Disadvantages
5. Flow of MVC
6. Routing
7. Simple MVC Form

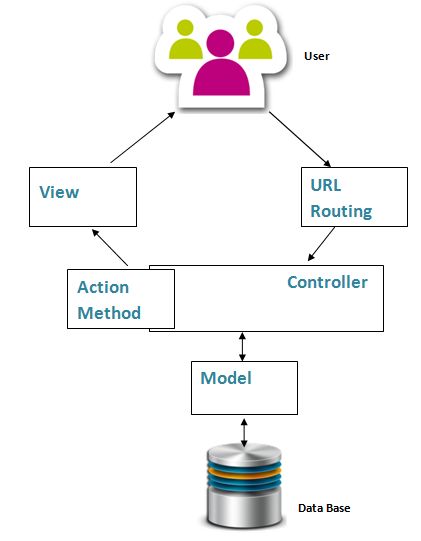
**Introduction**  
MVC is a one of the famous design patterns for developing web applications, websites and web pages. It is much easier for developers compared to traditional [ASP.NET](http://www.c-sharpcorner.com/technologies/asp-dot-net-programming). MVC has a lot of advantages. MVC is the latest trend in the word. This article explains about the basic structure of MVC and how to handle the MVC Applications

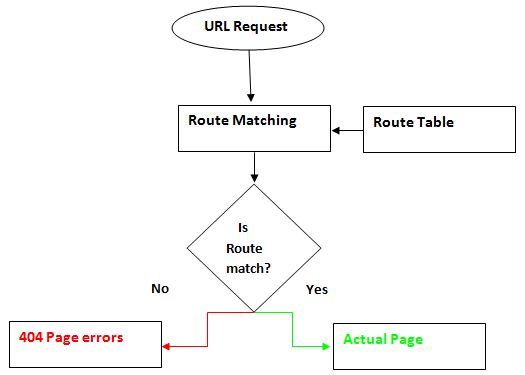
**Definition**  
Model View Controller (MVC) is a design pattern or methodology. It is used to effectively develop web applications. MVC is a not complex one, we can learn it easily.  
  
**Advantages**   
  
There are many advantages available in MVC.

* A separation of concern is a main advantage of MVC. It means we can divide into three part of the application like Model, View and Controller.
* Handles the code easily because of separation of concern.
* In the same time we can split many developers' work at one time. It will not affect one developer's work to another developer's work.
* It supports TTD (test-driven development). We can create an application with unit test. We can write one test case.
* Latest version of MVC supports default responsive web site and mobile templates.
* We can create our own view engine. Its syntax is very easy compared to traditional view engines.

**Disadvantages**

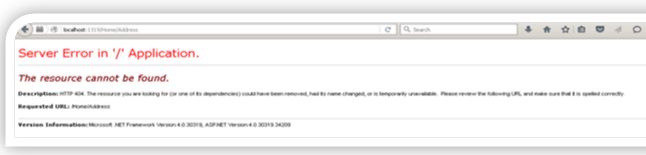
* Cannot see design page preview like .aspx page. Every time we want to run it, then we see the design.
* Understanding flow of application is very hard. It is a little bit complex to implement and not suitable for small level applications.
* Its deployment is a little bit hard.

**Basic Work Flow of MVC**  
MVC work flow is different compared with ASP.NET. First understand MVC work flow,  then you need to know every one. The below diagram explains the work flow of MVC.  
  
  
**Figure 1:** MVC Work Flow

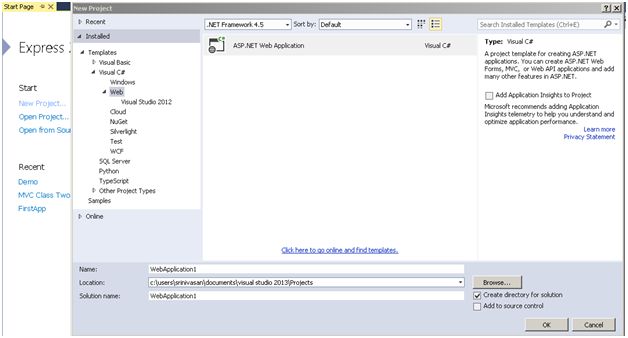
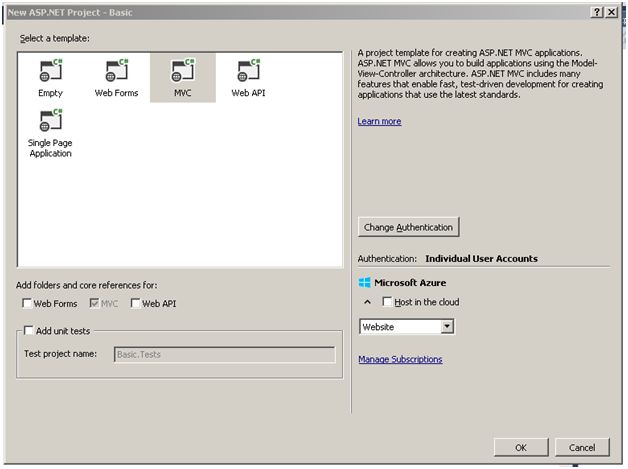
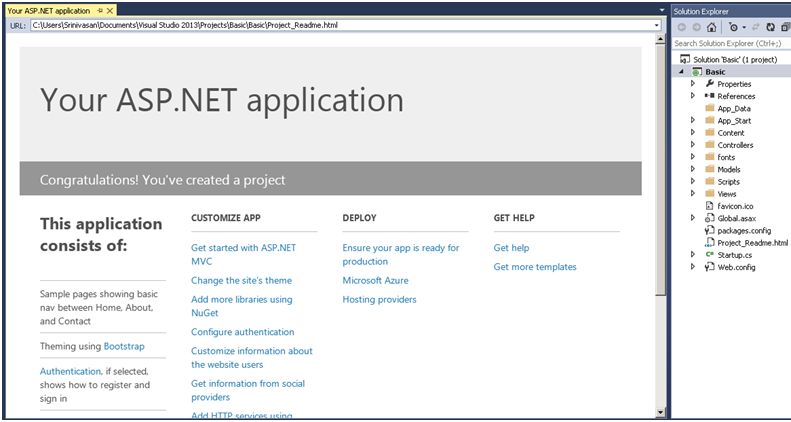
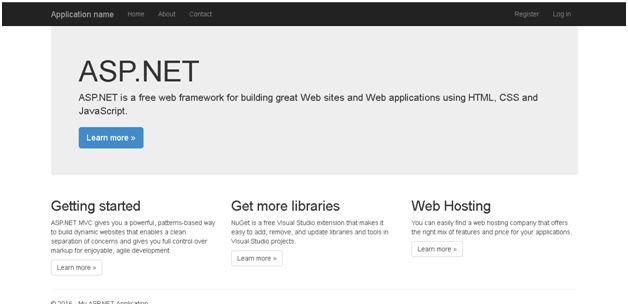
**Routing**  
Routing finds the available URL Request from which one user is given the URL. Routing is a pattern matching system with the help of a routing table. Routing table contains all available routes based on controller and action. If a URL is entered, the routing engine compares the given URL with the route table. If the given URL is  not available it shows an error, otherwise it renders corresponding page.  
  
  
**Figure 2:**Routing  
  
**Route Table**  
Routing table contains all possible ways of the URL based on controller and action methods. Whenever you enter the URL in the browser the routing engine matches it with the route table. The below screen shot shows how many action methods are available in one Controller.In “**Home**” the controller contains three action methods, so three different ways of URL requests are possible. I have different controller and action possible ways of URL request increases. All URL requests are based on controllers and actions.

  
**Figure 3:**Action Results  
  
Possible method of URL Request below:

* http://localhost:1319/Home/index
* http://localhost:1319/Home/About
* http://localhost:1319/Home/Contact

If any other URL request is given it shows an error like the below screen shots.  
  
  
**Figure 4:**Error Page

**Simple MVC Application**  
Follow the below steps for opening a simple MVC default application. You can learn many concepts from MVC default application. MVC 5 is available in Visual Studio 2013 and above.

1. Open Microsoft Visual Studio then choose New Project then show the screen as below.  
     
     
   **Figure 5:**New Project Window
2. Click Web and select ASP.NET Web Application. Give specified name in Name Colum; if you need to choose which location your project wants to store then click Ok button.
3. After clicking Ok then select a template need like Empty, Web Forms, MVC , Web API and Single Page Application.  
     
     
   **Figure 6:**Project Template Window
4. Select MVC as in the above image. If you need unit testing for your application select the add unit tests check box.
5. Directly host our application in cloud using Hosting in Cloud option marked in red. If you want to host in the cloud you need to register and get a cloud account.
6. Finally click the ok button then it takes a few minutes to open the application. After opening we can see what the necessary folders on the right side are in the coding page of MVC.  
   **Figure 7:**Project Folders
7. App\_Data, App\_Start, Content, Fonts, Model, Script, Views and Web.Config are available in Solution Explore.
8. In default application you do not need to do any coding, it has everything,  just run it, you can see the output shown below.  
     
     
   **Figure 8:** Final Application Output

**Conclusion**  
MVC support responsive websites with the help of bootstrap. It is the default available in MVC 4 and above.

Monday, April 2, 2018

What are **ViewData**, **ViewBag**, and **TempData** in ASP.NET MVC?

**Explanation 1 – ( Simple with Examples )**

To start with, ViewData, ViewBag, and TempData all three are objects in ASP.NET MVC that are used to carry or pass data in different scenarios. You may have a requirement to pass data in the following cases:

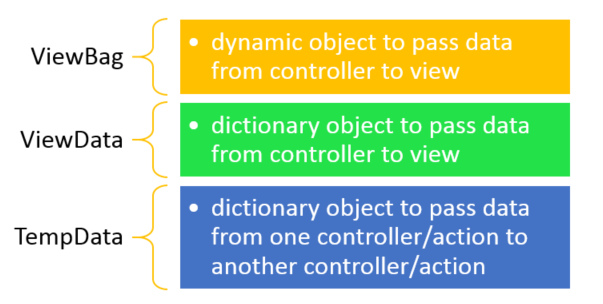
·        Pass data from the controller to view;

·        Pass data from one controller to another controller;

·        Pass data from one action to another action;

·        Pass data between subsequent HTTP requests

At a higher level, we can depict the use of ViewData, ViewBag, and TempData as shown in the image below:



**Passing Data from Controller to View**

Let us consider a scenario where you’re passing data from the controller to view. Usually, we pass complex data to the view using the model. Here let’s say we have a strongly typed View which is using the data model List as shown in the listing below:

public ActionResult Index()

        {

            List<Product> p = new List<Product>() {

               new Product { Id = 1, Name = "Pen", Price = 300 },

               new Product { Id = 2, Name = "Pencil", Price = 100 }

            };

            return View(p);

        }

On the View, data is displayed by rendering the model as shown in the listing below:

<table class="table">

    <tr>

        <th>

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

        th>

        <th>

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

        th>

        <th>th>

    tr>

@foreach (var item in Model) {

    <tr>

        <td>

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

        td>

        <td>

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

        td>

        <td>

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

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

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

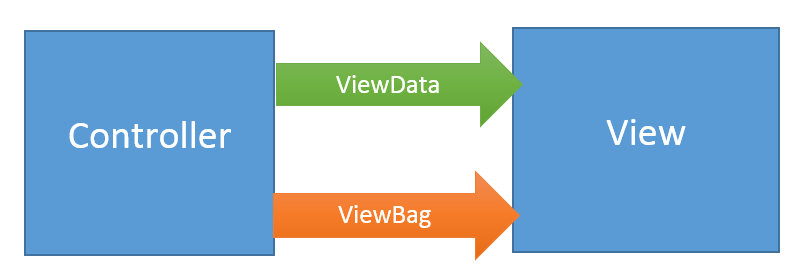
        td>

    tr>

}

table>

Now we have a requirement to pass data (other than a model) to the view from the controller. There are two possible ways data can be passed.



Let us assume that we want to pass a simple string to the view besides the Product data model.

**Passing data using ViewBag**

We can pass data using the ViewBag as shown in the listing below:

public ActionResult Index()

        {

            ViewBag.data1 = "I am ViewBag data";

            return View(p);

        }

 On the view, ViewBag data can be read as the property of the ViewBag as shown in the listing below:

<h2>@ViewBag.data1h2>

**Passing data using ViewData**

We can pass data using the ViewData as shown in the listing below:

 public ActionResult Index()

        {

            ViewData["data1"] = "I am ViewBag data";

            return View(p);

        }

On the view, ViewData data can be read as the string value pair of the ViewData as shown in the listing below:

<h2>@ViewData["data1"]h2>

Let us examine the differences between ViewData and ViewBag. ViewBag is a dynamic property which is based on the dynamic type, whereas ViewData is a dictionary object. We can read data from ViewBag as a property and from ViewData as a key-value pair. Some bullet points about both are as follows:

**ViewData**

        It’s a property of type ViewDataDictionary class.

        Data can be passed in the form of a key-value pair.

        To read the complex type data on the view, typecasting is required.

        To avoid the exception, null checking is required.

        Life of ViewData is restricted to the current request and becomes Null on redirection.

        ViewData is a property of the ControllerBase class

**ViewBag**

        It’s a property of the dynamic type.

        Data is passed as a property of the object.

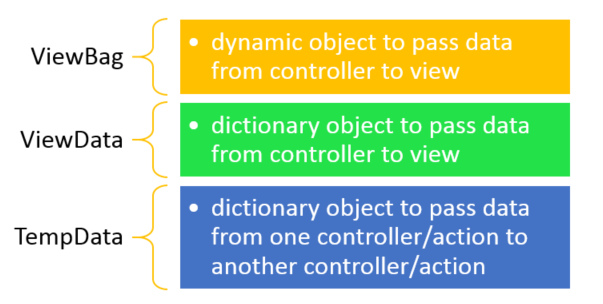
        There is no need of typecasting to read the data.

        There is no need for null checking.

        Life of ViewBag is restricted to the current request and becomes Null on redirection.

        ViewBag is a property of ControllerBase class.

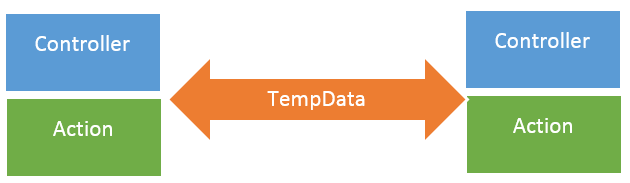
In the ControllerBase class, both are defined as property as shown in the image below:



We can summarize ViewBag and ViewData as objects that are used to pass data from the controller to view in a single cycle. The value assigned in ViewBag and ViewData get nullified in the next HTPP request or navigating to another view.

**TempData**

One of the major attributes of both ViewData and ViewBag are that their lifecycle is limited to one HTTP request. On redirection, they lose the data. We may have another scenario to pass data from one HTTP request to the next HTTP request; for example, passing data from one controller to another controller or one action to other action. TempData is used to pass data from one request to the next request.



 Let us say that we want to navigate to Read action from Index action and while navigating, pass data to the Read action from the Index action.  So in the Index action, we can assign a value to TempData as shown in the listing below:

  public ActionResult Index()

        {

          TempData["data1"] = "I am from different action";

          return RedirectToAction("Read");

        }

 We can read TempData as a key-value pair. In the Read action, TempData can be read as shown in the listing below:

public string Read()

        {

            string str;

            str = TempData["data1"].ToString();

            return str;

        }

 Like ViewData, TempData is also a dictionary object and to read the data, typecasting and null checking is required. Keep in mind that TempData can persist data only to the subsequent HTTP request. When you are very sure about the redirection, then use TempData to pass the data.

Some points about **TempData** are as follows:

        TempData is used to pass data from one HTTP request to next HTTP request.

o   In other words, TempData is used to pass data from one controller to another controller or action to another action.

        TempData is a property of BaseController class.

        TempData stores data in a session object

        TempData is a property of ControllerBase class

        To read data,  Typecasting and null checking are required.

        Type of TempData is TempDataDictionary.

        TempData works with HTTP redirection like HTTP 302/303 status code

**Summary**

ViewData, ViewBag, and TempData are used to pass data between controller, action, and views. To pass data from the controller to view, either ViewData or ViewBag can be used. To pass data from one controller to another controller, TempData can be used.

**Explanation 2 – ( Different Approach )**

ViewData, ViewBag and TempData are used for transferring data and objects from the Controller to the View or from one Controller to another in ASP.NET MVC.

### **ViewData**

1. ViewData is derived from the ViewDataDictionary class and is basically a Dictionary object, i.e. it has keys and values where keys are strings and Values are objects.
2. Data is stored as an Object inViewData.
3. While retrieving data, the data needs to be Type Cast to its original type, as it will be stored as an object. ViewData also requires NULL checks while retrieving data.
4. ViewData is used for passing a value from the Controller to the View.
5. ViewData is available only for Current Requests. It will be destroyed upon redirection.

#### ****Example :**** In the below example, a string value is set in the ViewData object in the Controller and it is then displayed in the View.

**Controller**

public class FirstController: Controller {

// GET: First

public ActionResult Index() {

ViewData["Message"] = "Hello Sultan!";

return View();

}

}

**View**

<html>

<head>

<meta name="viewport" content="-width"/>

<title>Index</title>

</head>

<body>

<div>

@ViewData["Message"]

</div>

</body>

</html>

### **ViewBag**

1. ViewBag is a Wrapper built around ViewData.
2. ViewBag is a dynamic property and it makes use of C# 4.0's dynamic features.
3. While retrieving data, there is no need for Type Casting the data.
4. ViewBag is used for passing a value from the Controller to the View.
5. ViewBagis available only for Current Requests. It will be destroyed upon redirection.

#### ****Example****

In the below example, a string value is set in the ViewBag object in the Controller and it is then displayed in the View.

**Controller**

public class FirstController: Controller {

// GET: First

public ActionResult Index() {

ViewBag.Message = "Hello Sultan!";

return View();

}

}

**View**

<html>

<head>

<meta name="viewport" content="-width"/>

<title>Index</title>

</head>

<body>

<div>

@ViewBag.Message

</div>

</body>

</html>

### **TempData**

1. TempData is derived from the TempDataDictionary class and is basically a Dictionary object, i.e. keys and values where keys are string while values will be objects.
2. Data is stored as an object in TempData.
3. While retrieving, the data it needs to be Type Casted to its original type as the data is stored as objects and it also requires NULL checks while retrieving data.
4. TempData can be used for passing a value from the Controller to the View and also from the Controller to another Controller.
5. TempData is available for current and subsequent requests. It will not be destroyed on redirection.

#### ****Example****

In the below example, a string value is set in the TempData object in the Controller and it is redirected to another Controller and, finally, it is displayed in the View.

**First Controller**

public class FirstController: Controller {

// GET: First

public ActionResult Index() {

TempData["Message"] = "Hello Sultan!";

return new RedirectResult(@ "~\Second\");

}

}

**Second Controller**

public class SecondController: Controller {

// GET: Second

public ActionResult Index() {

return View();

}

}

**View of Second Controller**

<html>

<head>

<meta name="viewport" content="-width"/>

<title>Index</title>

</head>

<body>

<div>

@TempData["Message"];

</div>

</body>

</html>

**Explanation – 3 ( Differences )**

**ViewData** VS **ViewBag** VS **TempData**

|  |  |  |
| --- | --- | --- |
| **ViewData** | **ViewBag** | **TempData** |
| It is Key-Value Dictionary collection | It is a type object | It is Key-Value Dictionary collection |
| ViewData is a dictionary object and it is property of ControllerBase class | ViewBag is Dynamic property of ControllerBase class. | TempData is a dictionary object and it is property of controllerBase class. |
| ViewData is Faster than ViewBag | ViewBag is slower than ViewData | NA |
| ViewData is introduced in MVC 1.0 and available in MVC 1.0 and above | ViewBag is introduced in MVC 3.0 and available in MVC 3.0 and above | TempData is also introduced in MVC1.0 and available in MVC 1.0 and above. |
| ViewData also works with .net framework 3.5 and above | ViewBag only works with .net framework 4.0 and above | TempData also works with .net framework 3.5 and above |
| Type Conversion code is required while enumerating | In depth, ViewBag is used dynamic, so there is no need to type conversion while enumerating. | Type Conversion code is required while enumerating |
| Its value becomes null if redirection has occurred. | Same as ViewData | TempData is used to pass data between two consecutive requests. |
| It lies only during the current request. | Same as ViewData | TempData only works during the current and subsequent request. |