View bag, View data, Tempdata

Introduction

ViewData and ViewBag are used for the same purpose --  to transfer data from controller to view.

ViewData is nothing but a dictionary of objects and it is accessible by string as key.

ViewData is a property of controller that exposes an instance of the ViewDataDictionary class.

ViewBag is very similar to ViewData.

ViewBag is a dynamic property (dynamic keyword which is introduced in .net framework 4.0).

ViewBag is able to set and get value dynamically and able to add any number of additional fields without converting it to strongly typed.

ViewBag is just a wrapper around the ViewData.

ViewData Example

//Controller Code

public ActionResult Index()

{

List<string> Student = new List<string>();

Student.Add("Jignesh");

Student.Add("Tejas");

Student.Add("Rakesh");

ViewData["Student"] = Student;

return View();

}

//page code

<ul>

<% foreach (var student in ViewData["Student"] as List<string>)

{ %>

<li><%: student%></li>

<% } %>

</ul>

C#

Copy

ViewBag Example

//Controller Code

public ActionResult Index()

{

List<string> Student = new List<string>();

Student.Add("Jignesh");

Student.Add("Tejas");

Student.Add("Rakesh");

ViewBag.Student = Student;

return View();

}

//page code

<ul>

<% foreach (var student in ViewBag.Student)

{ %>

<li><%: student%></li>

<% } %>

</ul>

C#

Copy

TempData is a dictionary which is derived from TempDataDictionary class. TempData is stored data just like live session for short time. TempData keeps data for the time of HTTP Request, which means that it holds data between two consecutive requests. TempData helps us to transfer data between controllers or between actions. TempData internally use Session variables. Note that TempData is only work during the current and subsequent request. It is generally used to store one time messages. With the help of the TempData.Keep() method we can keep value in TempData object after request completion.

TempData Example

//Controller Code

public ActionResult Index()

{

List<string> Student = new List<string>();

Student.Add("Jignesh");

Student.Add("Tejas");

Student.Add("Rakesh");

TempData["Student"] = Student;

return View();

}

//page code

<ul>

<% foreach (var student in TempData["Student"] as List<string>)

{ %>

<li><%: student%></li>

<% } %>

</ul>

C#

Copy

ViewData VS ViewBag VS TempData

|  |  |  |
| --- | --- | --- |
| **ViewData** | **View Bag** | **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 |

Conclusion

We have three options: ViewData, ViewBag and TeampData for passing data from controller to view and in next request. ViewData and ViewBag are almost similar and it helps us to transfer the data from controller to view whereas TempData also works during the current and subsequent requests.

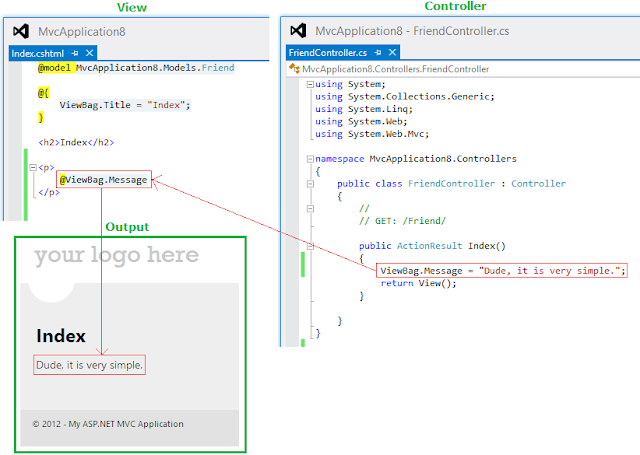
# **Various Ways to Pass Data From Controller to View in MVC**

There are various ways to pass data from a Controller to a View. I'm going to discuss how Controllers interact with Views and specifically cover ways you can pass data from a Controller to a View to render a response back to a client. So, let's get started.

## ViewBag

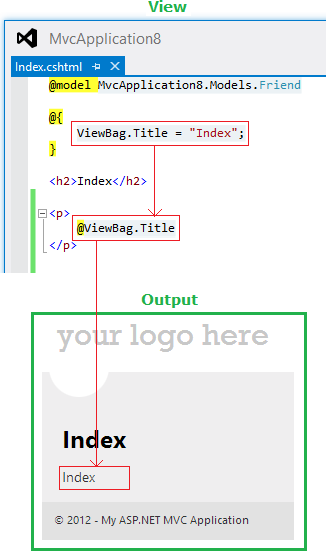
ViewBag is a very well known way to pass the data from Controller to View & even View to View. ViewBag uses the dynamic feature that was added in C# 4.0. We can say ViewBag=ViewData + Dynamic wrapper around the ViewData dictionary. Let's see how it is used.

### **Pass Data Between Controller and View**



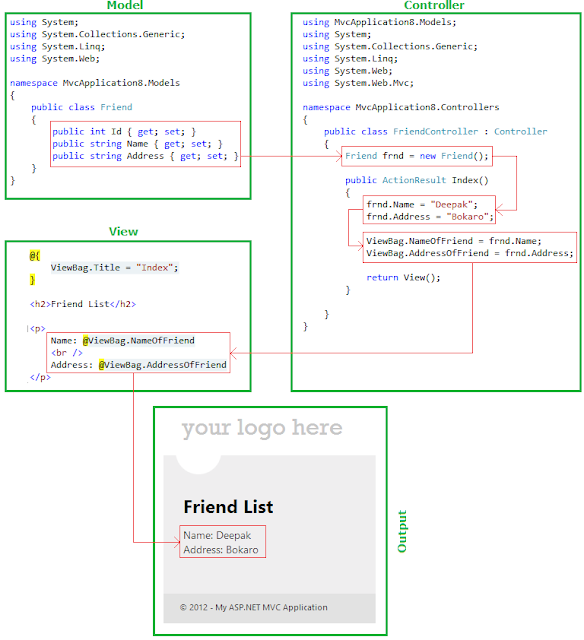
In the above image, you can see how data flows from the "Controller" to the "View", and how it looks in the browser.

### **Pass Data Between View to View**



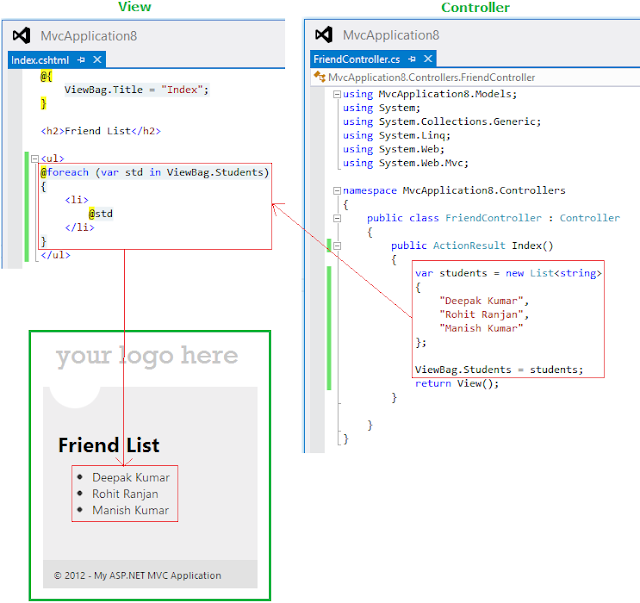
In the above image, you see how data is initialized on the "View" page itself using 'ViewBag.Title = "Index"' and then how it is getting rendered using '@ViewBag.Title'. What is "Title"? It is nothing more than a key, which has very limited availability and can be used on the same page only. So, the key naming is up to you, use any name which makes you happy.

Look at one more case, where I will take the advantage of "Model".



In the above case, we have a "Model" defined by name "Friend" that has three properties "Id", "Name" and "Address". On the "Controller", we have an object of the "Friend" class named "frnd" and then using the dot (.) operation properties are assigned then attached to these properties of the ViewBag.

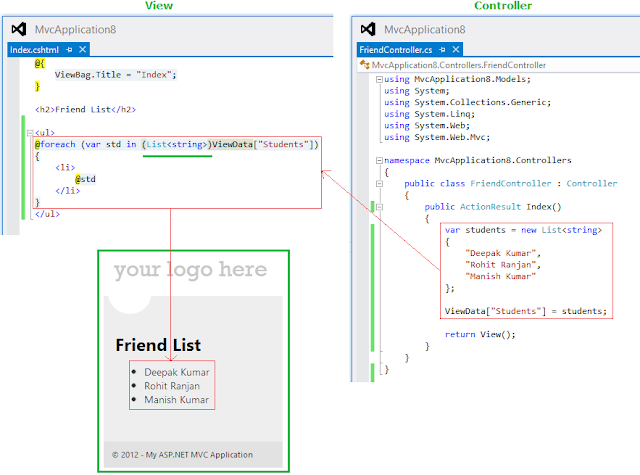
Look at one more example, in which a list of students is passed using ViewBag.



So, in this way we can pass the list of students. I hope this is clear to you.

## ViewData

ViewBag and ViewData serves the same purpose in allowing developers to pass data from controllers to views. When you put objects in either one, those objects become accessible in the view. Let's look at one example:

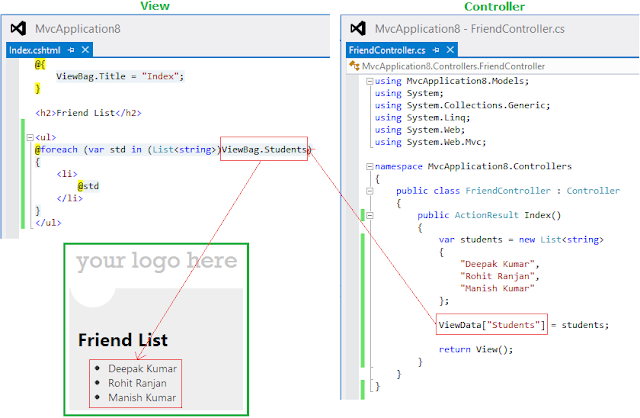


In the above image, everything is normal instead something in foreach loop. ViewData is typed as a dictionary containing "objects", we need to cast ViewData["Students"] to a List<string> or an IEnumerable<string> in order to use the foreach statement on it. Such as in:

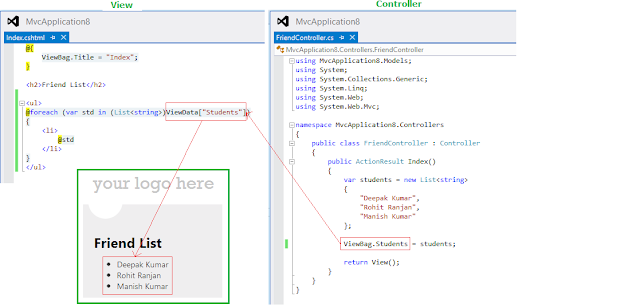
1. @foreach (**var** std **in** (List<string>)ViewData["Students"])
2. //OR
3. @foreach (**var** std **in** (IEnumerable<string>)ViewData["Students"])

Now look at one more beauty of MVC, you can put data into the ViewBag and access it from ViewData or put data in the ViewData and access it from the ViewBag, here you have all the freedom.

### **Pass Data ViewData to ViewBag**



### **Pass Data ViewBag to ViewData**

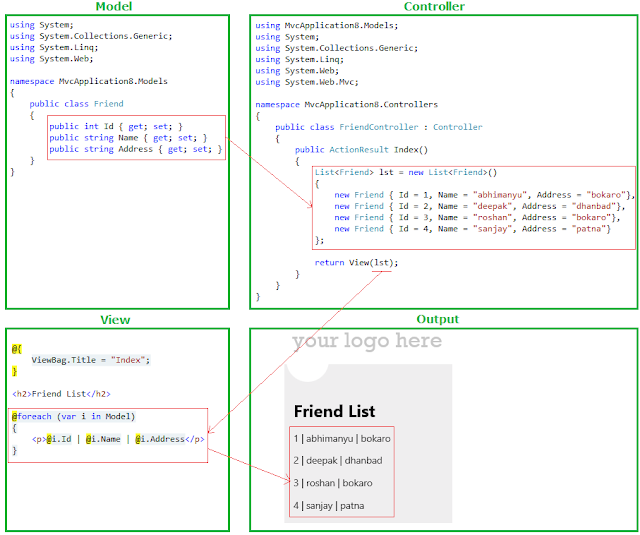


So these two (ViewBag and ViewData) things seem to work almost exactly the same. Then, what's the difference? The difference is only how you access the data. ViewBag is actually just a wrapper around the ViewData object, and its whole purpose is to let you use access the data dynamically instead of using magic <strings> conversion, you can realize it by the above examples. Some people prefer one style over the other. You can pick whichever makes you happy. In fact, because they're the same data just with two different ways of accessing it, you can use them interchangeably like ViewData to ViewBag or ViewBag to ViewData. It is not recommended, however, that you actually use them interchangeably, since it will will confuse others.

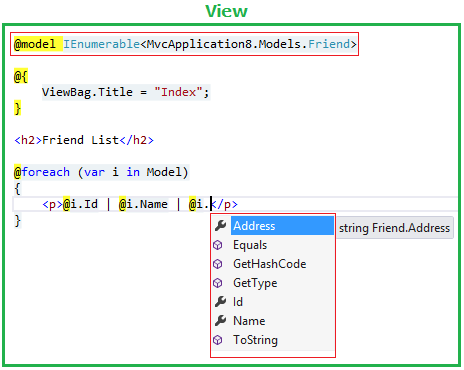
Now, so far we have looked into some ViewBag and ViewData, which is really very useful but we can pass the data using a model also and this will provide you full intellisense features.

## ViewModel

Using ViewModel we can also pass the data from the Controller to View; let's look at the image.



In the above image, we have multiple people in a list form being passed as a View, simple. I will add one more thing here, you are not going to get intellisence support or a strongly typed view page here, to get it do it this way. Just add a reference of the model by using the IEnumerable interface and you are done.



## TempData

TempData is meant to be a very short-lived instance, and you should only use it during the current and the subsequent requests only. Since TempData works this way, you need to know for sure what the next request will be, and redirecting to another view is the only time you can guarantee this. You can use TempData to pass error messages or something similar.

**Example1**: Using TempData like a ViewData and ViewBag.

1. **public** **class** FriendController : Controller
2. {
3. //
4. // GET: /Friend/
5. **public** ActionResult Index()
6. {
7. ViewData["VDFriend"] = "Deepak K Gupta";
8. ViewBag.VBFriend = "Deepak K Gupta";
9. TempData["TDFriend"] = "Deepak K Gupta";
10. **return** View();
11. }
12. }

And on "View",

1. <p>Using ViewData: @ViewData["VDFriend"]</p>
2. <p>Using ViewBag: @ViewBag.VBFriend</p>
3. <p>Using TempData: @TempData["TDFriend"] </p>

This is a simple example, but we are not using the real advantage of TempData, let's look at one more example.

**Example2**: Using TempData to get data after redirect also.

1. **public** **class** FriendController : Controller
2. {
3. //
4. // GET: /Friend/
6. **public** ActionResult Index()
7. {
8. ViewData["VDFriend"] = "Deepak K Gupta";
9. ViewBag.VBFriend = "Deepak K Gupta";
10. TempData["TDFriend"] = "Deepak K Gupta";
12. **return** **new** RedirectResult(@"~\Friend\AnotherPage\");
13. }
15. **public** ActionResult AnotherPage()
16. {
17. **return** View();
18. }
20. }

And on "View":

1. <p>Using ViewData: @ViewData["VDFriend"]</p>
2. <p>Using ViewBag: @ViewBag.VBFriend</p>
3. <p>Using TempData: @TempData["TDFriend"]</p>

As in the above "FriendController", I'm redirecting the view, in other words the Index() view will be redirected to the AnotherPage() view instantly and now on another view after one redirect we won't be able to get data from ViewData or ViewBag but TempData will work here.

A screenshot of a phone

Description automatically generated

Please read this blog also: [http://weblogs.asp.net/scottgu/archive/2007/12/06/asp-net-mvc-framework-part-3-passing-viewdata-from-controllers-to-views.aspx](https://weblogs.asp.net/scottgu/asp-net-mvc-framework-part-3-passing-viewdata-from-controllers-to-views).