ASP.NET MVC - ViewBag

We have learned in the previous section that the model object is used to send data in a razor view. However, there may be some scenario where you want to send a small amount of temporary data to the view. So for this reason, MVC framework includes ViewBag.

ViewBag can be useful when you want to transfer temporary data (which is not included in model) from the controller to the view. The viewBag is a dynamic type property of ControllerBase class which is the base class of all the controllers.

The following figure illustrates the ViewBag.

ViewBag

In the above figure, it attaches Name property to ViewBag with the dot notation and assigns a string value "Bill" to it in the controller. This can be accessed in the view like @ViewBag.Name. (@ is razor syntax to access the server side variable.)

You can assign a primitive or a complex type object as a value to ViewBag property.

You can assign any number of properties and values to ViewBag. If you assign the same property name multiple times to ViewBag, then it will only consider last value assigned to the property.

Note:

ViewBag only transfers data from controller to view, not visa-versa. ViewBag values will be null if redirection occurs.

The following example demonstrates how to transfer data from controller to view using ViewBag.

Example: Set ViewBag in Action method

```
namespace MVC_BasicTutorials.Controllers
{
    public class StudentController : Controller
    {
```

```
        IList<Student> studentList = new List<Student>() {
                    new Student(){ StudentID=1, StudentName="Steve", Age = 21 },
                    new Student(){ StudentID=2, StudentName="Bill", Age = 25 },
                    new Student(){ StudentID=3, StudentName="Ram", Age = 20 },
                    new Student(){ StudentID=4, StudentName="Ron", Age = 31 },
                    new Student(){ StudentID=5, StudentName="Rob", Age = 19 }
                };
        // GET: Student
        public ActionResult Index()
        {
            ViewBag.TotalStudents = studentList.Count();

            return View();
        }

    }
}
```

In the above example, we want to display the total number of students in a view for the demo. So, we have attached the TotalStudents property to the ViewBag and assigned the student count using `studentList.Count()`.

Now, in the Index.cshtml view, you can access ViewBag.TotalStudents property and display all the student info as shown below.

Example: Acess ViewBag in a View

```
<label>Total Students:</label>  @ViewBag.TotalStudents
```

Output:

```
Total Students: 5
```

ViewBag doesn't require typecasting while retriving values from it.

Internally, ViewBag is a wrapper around ViewData. It will throw a runtime exception, if the ViewBag property name matches with the key of ViewData.

Points to Remember :

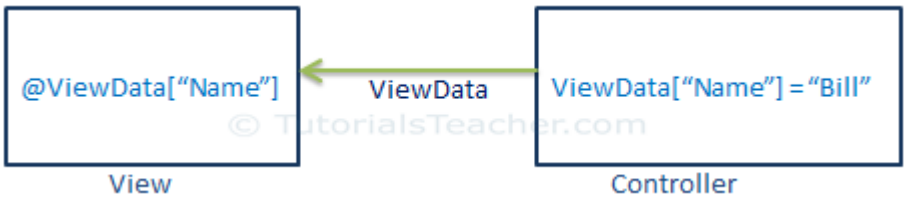
1. ViewBag transfers data from the controller to the view, ideally temporary data which in not included in a model.
2. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
3. You can assign any number of propertes and values to ViewBag
4. The ViewBag's life only lasts during the current http request. ViewBag values will be null if redirection occurs.
5. ViewBag is actually a wrapper around ViewData.

ASP.NET MVC - ViewData

ViewData is similar to ViewBag. It is useful in transferring data from Controller to View.

ViewData is a dictionary which can contain key-value pairs where each key must be string.

The following figure illustrates the ViewData.



ViewData

Note:

ViewData only transfers data from controller to view, not vice-versa. It is valid only during the current request.
The following example demonstrates how to transfer data from controller to view using ViewData.

Example: ViewData in Action method

```
public ActionResult Index()
{
    IList<Student> studentList = new List<Student>();
    studentList.Add(new Student(){ StudentName = "Bill" });
    studentList.Add(new Student(){ StudentName = "Steve" });
    studentList.Add(new Student(){ StudentName = "Ram" });

    ViewData["students"] = studentList;

    return View();
}
```

In the above example, we have added a student list with the key "students" in the ViewData dictionary. So now, the student list can be accessed in a view as shown below.

Example: Access ViewData in a Razor View

```
<ul>
@foreach (var std in ViewData["students"] as IList<Student>)
{
    <li>
        @std.StudentName
    </li>
}
</ul>
```

Please notice that we must cast ViewData values to the appropriate data type.

You can also add a KeyValuePair into ViewData as shown below.

Example: Add KeyValuePair in ViewData

```
public ActionResult Index()
{
    ViewData.Add("Id", 1);
    ViewData.Add(new KeyValuePair<string, object>("Name", "Bill"));
    ViewData.Add(new KeyValuePair<string, object>("Age", 20));

    return View();
}
```

ViewData and ViewBag both use the same dictionary internally. So you cannot have ViewData Key matches with the property name of ViewBag, otherwise it will throw a runtime exception.

Example: ViewBag and ViewData

```
public ActionResult Index()
{
    ViewBag.Id = 1;

    ViewData.Add("Id", 1); // throw runtime exception as it already has "Id" key
    ViewData.Add(new KeyValuePair<string, object>("Name", "Bill"));
    ViewData.Add(new KeyValuePair<string, object>("Age", 20));

    return View();
}
```

Points to Remember :

1. ViewData transfers data from the Controller to View, not vice-versa.
2. ViewData is derived from ViewDataDictionary which is a dictionary type.
3. ViewData's life only lasts during current http request. ViewData values will be cleared if redirection occurs.
4. ViewData value must be type cast before use.
5. ViewBag internally inserts data into ViewData dictionary. So the key of ViewData and property of ViewBag must **NOT** match.

ASP.NET MVC - TempData

TempData in ASP.NET MVC can be used to store temporary data which can be used in the subsequent request. TempData will be cleared out after the completion of a subsequent request.

TempData is useful when you want to transfer non-sensitive data from one action method to another action method of the same or a different controller

as well as redirects. It is dictionary type which is derived from TempDataDictionary.

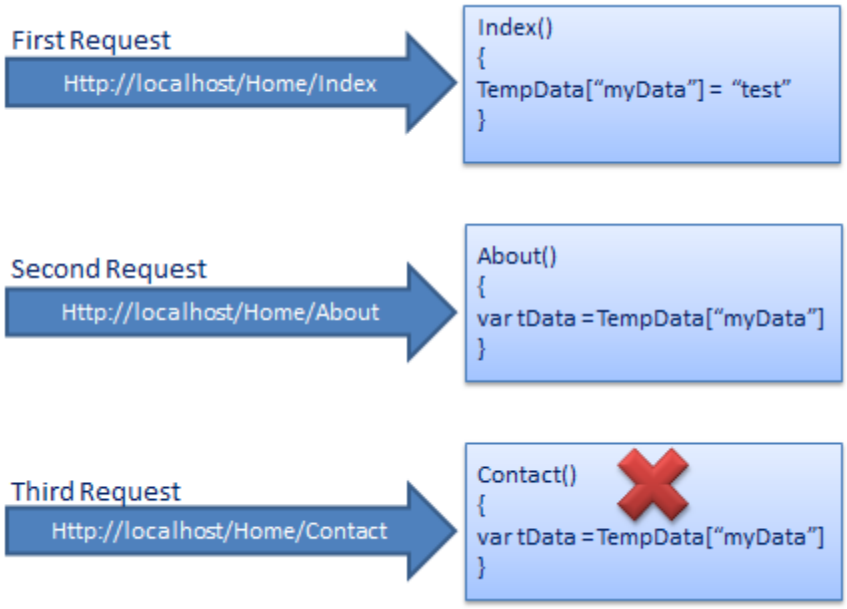
You can add a key-value pair in TempData as shown in the below example.

Example: TempData

```
public class HomeController : Controller
{
    // GET: Student
    public HomeController()
    {

    }
    public ActionResult Index()
    {
        TempData["name"] = "Test data";
        TempData["age"] = 30;

        return View();
    }

    public ActionResult About()
    {
        string userName;
        int userAge;

        if(TempData.ContainsKey("name"))
            userName = TempData["name"].ToString();

        if(TempData.ContainsKey("age"))
            userAge = int.Parse(TempData["age"].ToString());

        // do something with userName or userAge here

        return View();
    }
}
```

In the above example, we have added data into TempData and accessed the same data using a key inside another action method. Please notice that we have converted values into the appropriate type.

The following figure illustrates TempData.



TempData

TempData internally uses session to store the data. So the data must be serialized if you decide you to switch away from the default Session-State Mode, and use State Server Mode or SQL Server Mode.

As you can see in the above example, we add test data in TempData in the first request and in the second subsequent request we access test data from TempData which we stored in the first request. However, you can't get the same data in the third request because TempData will be cleared out after second request.

Call `TempData.Keep()` to retain TempData values in a third consecutive request.

Example: TempData.Keep()

```
public class HomeController : Controller
{
    public HomeController()
    {

    }

    public ActionResult Index()
    {
        TempData["myData"] = "Test data";
        return View();
    }

    public ActionResult About()
    {
        string data;
```

```
        if(TempData["myData"] != null)
            data = TempData["myData"] as string;

        TempData.Keep();

        return View();
    }

    public ActionResult Contact()
    {
        string data;

        if(TempData["myData"] != null)
            data = TempData["myData"] as string;

        return View();
    }
}
```

Points to Remember :

1. TempData can be used to store data between two consecutive requests. TempData values will be retained during redirection.
2. TemData is a TempDataDictionary type.
3. TempData internaly use Session to store the data. So think of it as a short lived session.
4. TempData value must be type cast before use. Check for null values to avoid runtime error.
5. TempData can be used to store only one time messages like error messages, validation messages.
6. Call TempData.Keep() to keep all the values of TempData in a third request.

ASP.NET MVC - TempData

TempData in ASP.NET MVC can be used to store temporary data which can be used in the subsequent request. TempData will be cleared out after the completion of a subsequent request.

TempData is useful when you want to transfer non-sensitive data from one action method to another action method of the same or a different controller as well as redirects. It is dictionary type which is derived from TempDataDictionary.

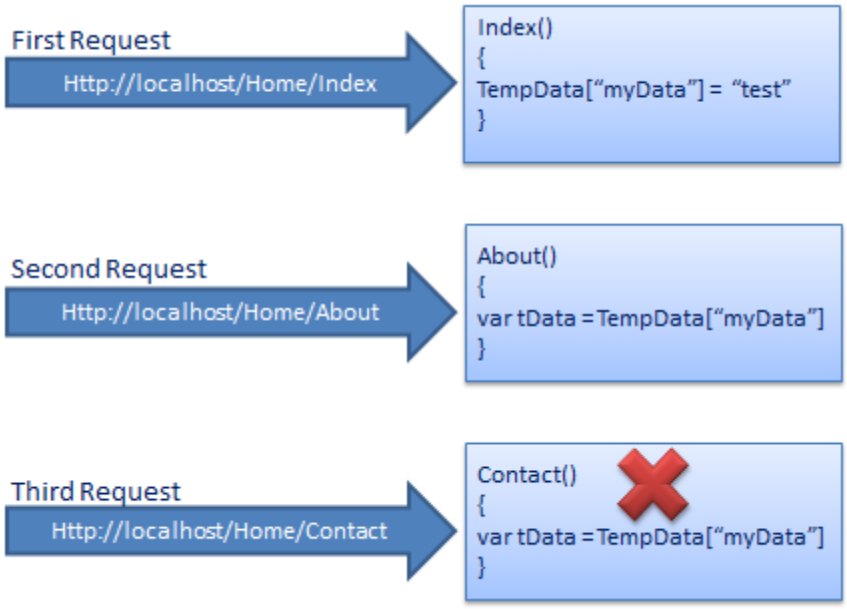
You can add a key-value pair in TempData as shown in the below example.

Example: TempData

```
public class HomeController : Controller
{
    // GET: Student
    public HomeController()
    {

    }
    public ActionResult Index()
    {
        TempData["name"] = "Test data";
        TempData["age"] = 30;

        return View();
    }

    public ActionResult About()
    {
        string userName;
        int userAge;

        if(TempData.ContainsKey("name"))
            userName = TempData["name"].ToString();

        if(TempData.ContainsKey("age"))
            userAge = int.Parse(TempData["age"].ToString());

        // do something with userName or userAge here

        return View();
    }
}
```

In the above example, we have added data into TempData and accessed the same data using a key inside another action method. Please notice that we have converted values into the appropriate type.

The following figure illustrates TempData.



TempData

TempData internally uses session to store the data. So the data must be serialized if you decide you to switch away from the default Session-State Mode, and use State Server Mode or SQL Server Mode.

As you can see in the above example, we add test data in TempData in the first request and in the second subsequent request we access test data from TempData which we stored in the first request. However, you can't get the same data in the third request because TempData will be cleared out after second request.

Call `TempData.Keep()` to retain TempData values in a third consecutive request.

Example: TempData.Keep()

```
public class HomeController : Controller
{
    public HomeController()
    {

    }

    public ActionResult Index()
    {
        TempData["myData"] = "Test data";
        return View();
    }

    public ActionResult About()
    {
        string data;
```

```
        if(TempData["myData"] != null)
            data = TempData["myData"] as string;

        TempData.Keep();

        return View();
    }

    public ActionResult Contact()
    {
        string data;

        if(TempData["myData"] != null)
            data = TempData["myData"] as string;

        return View();
    }
}
```

Points to Remember :

1. TempData can be used to store data between two consecutive requests. TempData values will be retained during redirection.
2. TemData is a TempDataDictionary type.
3. TempData internaly use Session to store the data. So think of it as a short lived session.
4. TempData value must be type cast before use. Check for null values to avoid runtime error.
5. TempData can be used to store only one time messages like error messages, validation messages.
6. Call TempData.Keep() to keep all the values of TempData in a third request.