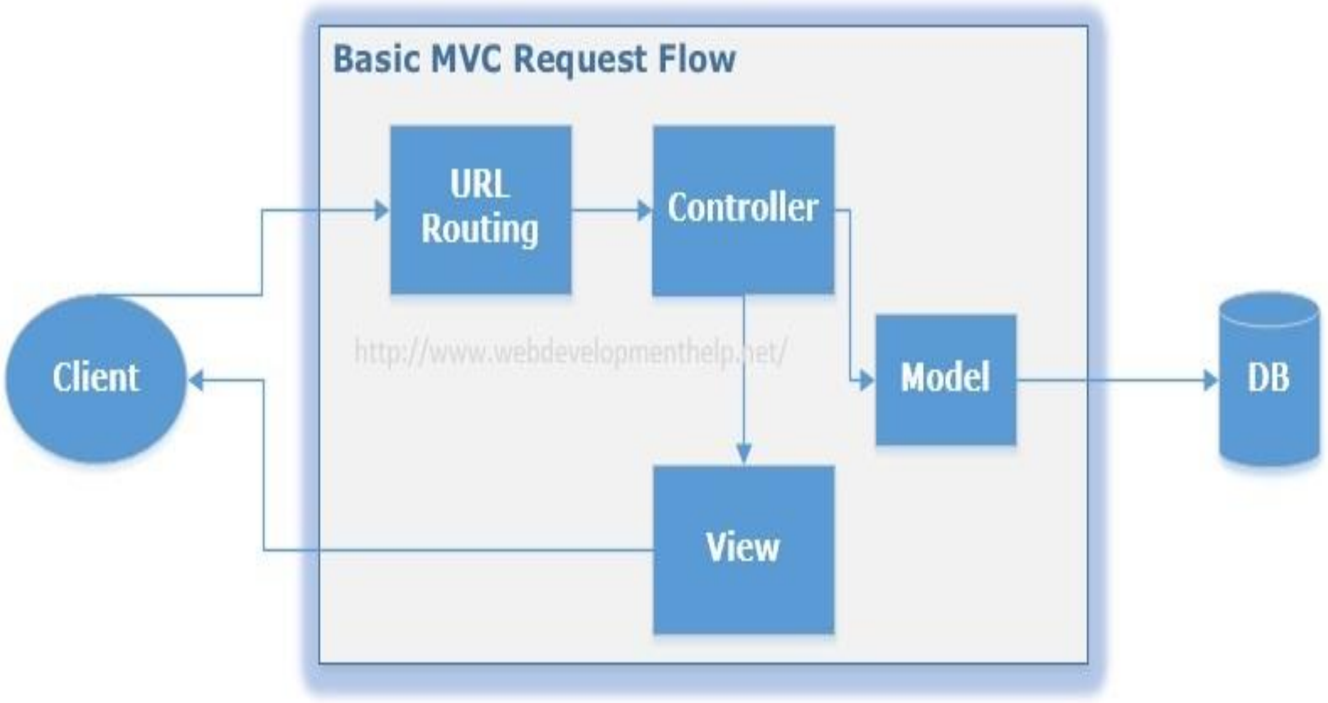
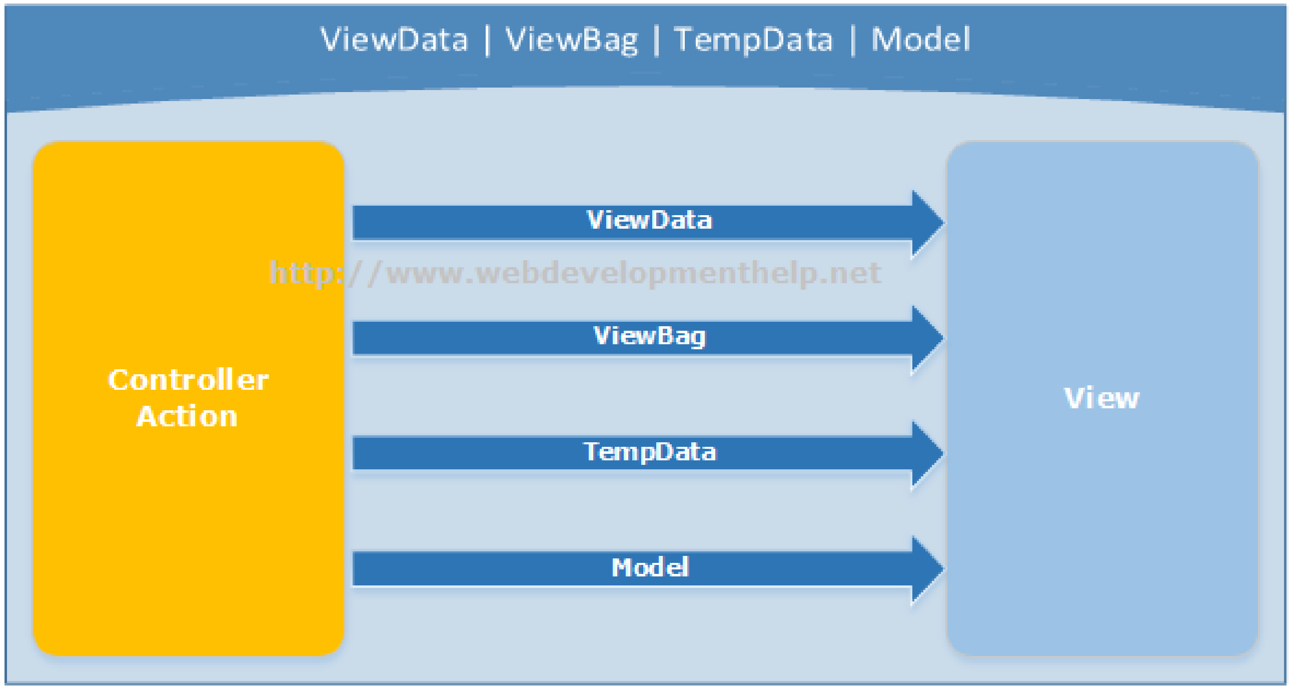
Passing Data from Controller to View

Introduction



Passing data Options



Passing data Options

Following are the different available options to pass data from a *Controller* to *View* in ASP.NET MVC

- ViewBag
- ViewData
- TempData (TempData Vs Session)
- Model

Passing data Options

- Maintain state between a *Controller* and corresponding *View*
  ViewData and ***ViewBag*** options are limited to a single server call (meaning it's value will be null if a redirect occurs).
- Maintain state from one *Controller* to another (redirect case)
  TempData is the other available option

ViewBag

- *ViewBag* is basically a dynamic property (a new C# 4.0 feature) having advantage that it doesn’t have typecasting and null checks.

```
public class EmployeeController : Controller
{
        // GET: /Employee/
        public ActionResult Index()
    {
                ViewBag.EmployeeName = “John Nguyen”;
                ViewBag.Company = “Web Design Company”;
                ViewBag.Address = “USA”;

            return View();
    }
}
```

ViewBag

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>
```

URL: http://localhost:2345/Employee/index

ViewData

ViewData is a dictionary object which requires typecasting as well as null checks.

```
public class EmployeeController : Controller
 {
        // GET: /Employee/
        public ActionResult Index()
       {
                ViewData[“EmployeeName”] = “John Nguyen”;
                ViewData[“Company”] = “Web Design Company”;
                ViewData[“Address”] = “USA”;

                return View();
       }
 }
```

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

ViewData

```
<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>
```

TempData

- 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.
- TempdData can be used to maintain data between controller actions as well as redirects.
- *Just like ViewData, typecasting and null checks required for TempData also in order to avoid errors.*

TempData

- 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 = “John”,
                    EmpLastName = “Nguyen”
             };

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


 //Controller Action 2(PermanentEmployee)

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

TempData

- 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.
- Although ASP.NET MVC TempData stores its content in Session state but it gets destroyed earlier than a session object.
- It’s best fit for scenarios when:
  - we need to preserve data from current to subsequent request.
  - passing error message to an error page

Model

- Create a Model class named “Product” in the Models folder of solution.
- This class will be used as a Model class for passing data to Views.
- Right click the Models folder and choose Add.. -> Class. Name the new class “Product” and make sure you make it public.

Model

```
namespace MvcPassDataToView.Models
{
    public class Product
    {
        public long Id { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public decimal price { get; set; }
    }
}
```

Model

- Create an Index action of the ProductsController controller.
- Test the different ways we can use to pass data from controller to the View
- In the Index action of your controller create a Product object and assign it to a ViewData dictionary object as follows.

Model

```
public ActionResult Index()
        {
            Product cap = new Product
            {
                Id = 1,
                Name = "Golf cap",
                Description = "A nice cap to play golf",
                price = 10m
            };

            ViewData["MyProduct"] = cap;

            return View();
        }
```

Model

Change the Index View file contents as follows:

```
@using MvcPassDataToView.Models
@{
   ViewBag.Title = "Index";
}

<h2>Index</h2>

@{
   var prod = (Product)ViewData["MyProduct"];
}

<h1>Product</h1>
<h3>ID: @prod.Id</h3>
<h3>Name: @prod.Name </h3>
<h3>Description: @prod.Description</h3>
<h3>Price: @prod.price</h3>
```

Model

- An appropriate way to pass data from Controller to View is to pass the Model class object to the View.
- To demonstrate how this works, change the Index action contents as follows:

```
public ActionResult Index()
        {
            Product cap = new Product
            {
                Id = 1,
                Name = "Golf cap",
                Description = "A nice cap to play golf",
                price = 10m
            };

            return View(cap);
        }
```

Model

- To use that “cap” Product object, you need to tell the Index View that the Binding object is of class “Product”.
- In the Index View file add the following code at the top of the page.
  @model MvcPassDataToView.Models.Product

Now you can access the cap object’s properties using Razor syntax such as:

@Model."property_name"

Model

The Index View’s contents should look like this.

```
@using MvcPassDataToView.Models
@model MvcPassDataToView.Models.Product

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<h1>Product</h1>
<h3>ID: @Model.Id</h3>
<h3>Name: @Model.Name </h3>
<h3>Description: @Model.Description</h3>
<h3>Price: @Model.price</h3>
```

Model

The last way to pass data is the TempData object. This object is used to pass data from one request to the next and only.

For example, when you successfully add a new product you want to display a message “A new product was added!”.

To demonstrate this, add a new action named “TestTempData” in the ProductsController.

Model

```
public ActionResult TestTempData()
        {
            TempData["data"] = "This wont be displayed in the next request! Test it by refreshing the page!";

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

- The above code, assigns a string to the TempData[“data”] object and then invokes the Index action of the same controller. This will result the Index View to be displayed again.
- Add the following line at the end of the Index View <h2>@TempData["data"]</h2>

Model

Try to refresh the page. You will get a System.NullReferenceException. On purpose, we didn’t check if the TempData[“data”] object is null in the Index page (something you should always do) just to show you that this kind of data is accessible only between two requests!