MVC STATE MANAGEMENT

ASP.NET MVC offers us three options ViewData, VieBag and TempData for passing data from controller to view and in next request. ViewData and ViewBag are almost similar and TempData performs additional responsibility. Lets discuss or get key points on those three objects:
Similarities bet

ASP.NET MVC offers us three options ViewData, VieBag and TempData for passing data from controller to view and in next request. ViewData and ViewBag are almost similar and TempData performs additional responsibility. Let's discuss or get key points on those three objects:

Similarities between ViewBag & ViewData :

1. Helps to maintain data when you move from controller to view.
2. Used to pass data from controller to corresponding view.
3. Short life means value becomes null when redirection occurs. This is because their goal is to provide a way to communicate between controllers and views. It's a communication mechanism within the server call.

Difference between ViewBag & ViewData:

1. ViewData is a dictionary of objects that is derived from ViewDataDictionary class and accessible using strings as keys.
2. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.
3. ViewData requires typecasting for complex data type and check for null values to avoid error.
4. ViewBag doesn't require typecasting for complex data type.

ViewBag & ViewData Example:

```
public ActionResult Index()
{
    ViewBag.Name = "Monjurul Habib";
    return View();
}

public ActionResult Index()
{
    ViewData["Name"] = "Monjurul Habib";
    return View();
}
```

In View:

```
@ViewBag.Name
@ViewData["Name"]
```

Example:

```
// Controller class - ControllerBase class has a property "ViewData"
public ActionResult Index()
{
  ViewData["Message"] = "Hello, World";
  return View();
}
// View page
The message is <%: ViewData["Message"] %>
```

Example:

```
// Controller class - ControllerBase class has a property "ViewBag"
public ActionResult Index()
{
  ViewBag.Message = "Hello, World";
  return View();
}
// View page
The message is @ViewBag.Message
```

TempData:

TempData is also a dictionary derived from TempDataDictionary class and stored in short lives session and it is a string key and object value. The difference is that the life cycle of the object. TempData keep the information for the time of an HTTP Request. This mean only from one page to another. This also work with a 302/303 redirection because it's in the same HTTP Request. Helps to maintain data when you move from one controller to other controller or from one action to other action. In other words when you redirect, "Tempdata" helps to maintain data between those redirects. It internally uses session variables. Temp data use during the current and subsequent request only means it is use when you are sure that next request will be redirecting to next view. It requires typecasting for complex data type and check for null values to avoid error. generally used to store only one time messages like error messages, validation messages. "*TempData*" is similar to Session data, except that "*TempData"* values are marked for deletion when they are read, and they are removed when the request has been processed.

The "*System.Web.Mvc.TempDataDictionary*" class provides "*Keep*()" and "*Peek*()" methods:

- public void *Keep*(string key) // marks the specified key in the dictionary for retention
- public Object *Peek*(string key) // returns an object without marking the key for deletion

```
public ActionResult Index()
{
  var model = new Review()
            {
                Body = "Start",
                Rating=5
            };
    TempData["ModelName"] = model;
    return RedirectToAction("About");
}
<pre><pre lang="cs">public ActionResult About()
{
    var model= TempData["ModelName"];
    return View(model);
}
```

The last mechanism is the Session which work like the ViewData, like a Dictionary that take a string for key and object for value. This one is stored into the client Cookie and can be used for a much more long time. It also need more verification to never have any confidential information. Regarding ViewData or ViewBag you should use it intelligently for application performance. Because each action goes through the whole life cycle of regular asp.net mvc request. You can use ViewData/ViewBag in your child action but be careful that you are not using it to populate the unrelated data which can pollute your controller.

View Model Object – WebFormViewEngine

You can pass any object as a model to a view using the one of the overloaded "*View*()" helper methods.

- ViewResult *View*(Object *model*)
- ViewResult *View*(string *viewName*, Object *model*)
- ViewResult *View*(string *viewName*, string *masterName*, Object *model*)

In a view, you can acsess this object through the "*Model*" property. You do not have a code-behind file for a view anymore but actually the view is a subclass of the "*System.Web.Mvc.ViewPage*" class.

- public Object *Model* { get; }

Note that the "*Model*" property is the type of "*Object*". Therefore you need to cast it before accessing data.

At first, you need to prepare data as a Model class.

```
public class Product { ... }
```

In the action method, create a data (Model) object and pass it to the View.

```
public ActionResult Index()
{
  Product p = new Product();
  p.Name = "Toy";
  return View(p);
}
```

Now in a view, you can access the data you passed from the controller.

```
Product : <%: ((Product)Model).Name %>
```

Strongly-Typed View Model Object – Razor

When you use a "*Razor*" view engine, the view derives from "*System.Web.Mvc.WebViewPage<TModel>"* class.

You can define the model type like this:

```
@model Mvc3App.Models.Product
```

Now you are free to access data in a view.

```
<h2> Product: @Model.Name </h2>
```