

Strongly typed, Loosely typed



- Strongly typed
- Loosely typed
 - ✓ ViewBag
 - ✓ ViewData
 - ✓ TempData

Section 1

STRONGLY TYPED


- Strongly Typed is a concept used to refer to a programming language that enforces strict restrictions on intermixing of values with different data types.
- In other words, Strongly Typed allows to work with data type clearly

- Strongly Typed Views are views that use strongly typed for rendering specific types of model objects.
 - ✓ View of Product should render product object.
 - ✓ View of list students should render list of students
 - ✓

- ✓ **Automatic scaffolding:** Creates view with skeleton based on selected Template and Model.
- ✓ **IntelliSense support :** Visual Studio able to display IntelliSense using the Model.
- ✓ **Compile time type checking:** The compiler is able to detect problems with data type and we will get compiler error rather than a runtime error.

- With Strongly Typed Views, controller sends model object to view

```
DbShopContext dbShopContext = new DbShopContext();  
0 references  
public ActionResult Details()  
{  
    var product = dbShopContext.Products.Find(1);  
    return View(product);  
}
```

 (local variable) Product product

- In Strongly Typed Views, view uses model object strongly

```
@model Product

<div>
  <dl class="dl-horizontal">
    <dt>
      @Html.DisplayNameFor(model => model.ProductName)
    </dt>

    <dd>
      @Html.DisplayFor(model => model.ProductName)
    </dd>
  </dl>
</div>
```


DEMO

STRONGLY TYPED VIEWS

Section 2

LOOSELY TYPED

Loosely typed

- ViewBag
- ViewData
- TempData

- Is a dynamic data.
- Data only alive for one request.
- Value(s) get cleared if redirection occurs.
- Type casting is not required.

From Controller:

```
public ActionResult Index()
{
    //// ViewBag is a dynamic, we can set any value into any property
    ViewBag.Message = "Passing data from Controller to View using ViewBag";
    return View();
}
```

To View:

```
<div class="alert alert-info">@ViewBag.Message</div>
```

Result:

Passing data from Controller to View using ViewBag

- A dictionary object that is derived from **ViewDataDictionary** class.
- We can get and set values with key and values.
- Data only alive for one request.
- ViewData values get cleared if redirection occurs.
- ViewData value must be type cast before use.

From Controller:

```
public ActionResult Index()
{
    //// ViewData is a dictionary object, get and set values with key and values
    ViewData["Message"] = "Passing data from Controller to View using ViewData";
    return View();
}
```

To View:

```
<div class="alert alert-info">@ViewData["Message"]</div>
```

Result:

Passing data from Controller to View using ViewData

ViewBag vs. ViewData

- We can use both ViewData and ViewBag to pass the data from a Controller to a View.
- Both ViewData and ViewBag are used to create loosely typed views in MVC.
- Both the ViewData keys and ViewBag dynamic properties are resolved only at runtime.

ViewBag vs. ViewData

- ViewBag is a dynamic property
- We use the dynamic properties to store and retrieve data
- ViewBag doesn't require any typecasting for the complex data type
- ViewData is a dictionary object
- We use the string as the key to store and retrieve the data
- ViewData requires typecasting for complex data type, checks for null values

- Used when we need to persist the data between actions redirection.
- Is a dictionary object that is derived from **TempDataDictionary** class.
- TempData internally use Session to store the data.
- TempData's life is very short and lies only till the redirected view is fully loaded.
- In TempData type casting is required.

- TempData is used to store only one time messages like error messages etc...
- TempData often used to send data from action to action
- TempData can be used to send data from action to view
- TempData value will become null once the subsequent request is completed by default.
- To retain the TempData value in the third consecutive request, call **TempData.Keep()** method

- Demo 1: Send data from action to action
- Demo 2: Send data from action to view
- Demo 3: Retain the TempData value in the third consecutive request

Thank you

