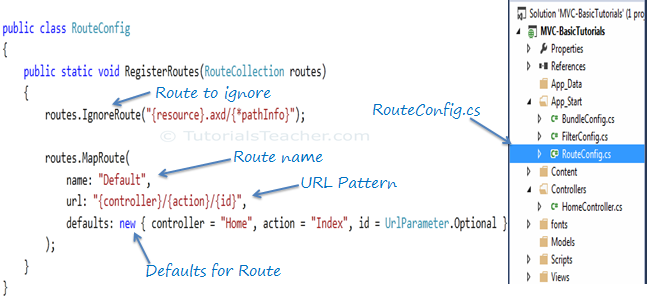
**What is Routing in ASP.NET MVC?**

Web Forms application, every URL must match with a specific .aspx file. For example, a URL http://domain/studentsinfo.aspx must match with the file studentsinfo.aspx

Routing to eliminate needs of mapping each URL with a physical file. Routing enable us to define URL pattern that maps to the request handler.

request handler in MVC, it is Controller class and Action method



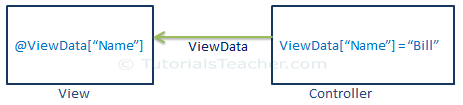
**How can we maintain sessions in MVC?**

we can maintain session in asp.net mvc three  
1-viewdata   
2-viewbag   
3-tempdata

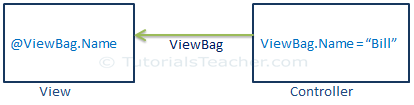
**What is the difference between tempdata, viewdata, and viewbag?**

**ViewData**

1. ViewData is used to pass data from controller to view
2. It is derived from ViewDataDictionary class
3. It is available for the current request only
4. Requires typecasting for complex data type and checks for null values to avoid error
5. If redirection occurs, then its value becomes null



**ViewBag**



1. ViewBag is also used to pass data from the controller to the respective view
2. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
3. It is also available for the current request only
4. If redirection occurs, then its value becomes null
5. Doesn’t require typecasting for complex data type

**TempData**

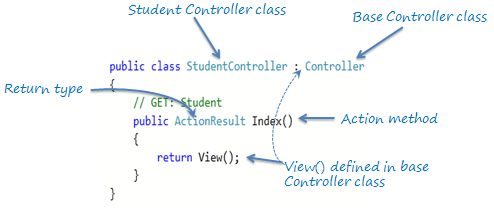
1. TempData is derived from TempDataDictionary class
2. TempData is used to pass data from the current request to the next request
3. It keeps the information for the time of an HTTP Request. This means only from one page to another. It helps to maintain the data when we move from one controller to another controller or from one action to another action
4. It requires typecasting for complex data type and checks for null values to avoid error. Generally, it is used to store only one time messages like the error messages and validation messages

**What are Action Methods in ASP.NET MVC?**

All the public methods of a Controller class are called Action methods

a method must meet the following requirements:

* The method must be public.
* The method cannot be a static method.
* The method cannot be an extension method.
* The method cannot be a constructor,getter, or setter.
* The method cannot have open generic types.
* The method is not a method of the controller base class.
* The method cannot contain ref or out parameters



ActionResult:

MVC framework includes various result classes, which can be return from an action methods.

There result classes represent different types of responses such as html, file, string, json, javascript etc

| **Result Class** | **Description** | **Base Controller method** |
| --- | --- | --- |
| ViewResult | Represents HTML and markup. | View() |
| EmptyResult | Represents No response. |  |
| ContentResult | Represents string literal. | Content() |
| FileContentResult, FilePathResult, FileStreamResult | Represents the content of a file | File() |
| JavaScriptResult | Represent a JavaScript script. | JavaScript() |
| JsonResult | Represent JSON that can be used in AJAX | Json() |
| RedirectResult | Represents a redirection to a new URL | Redirect() |
| RedirectToRouteResult | Represent another action of same or other controller | RedirectToRoute() |
| PartialViewResult | Returns HTML | PartialView() |
| HttpUnauthorizedResult | Returns HTTP 403 status |  |

**What are HTML helpers in MVC?**

The following table lists HtmlHelper methods and html control each method generates.

| **HtmlHelper** | **Strogly Typed HtmlHelpers** | **Html Control** |
| --- | --- | --- |
| Html.ActionLink |  | Anchor link |
| Html.TextBox | Html.TextBoxFor | Textbox |
| Html.TextArea | Html.TextAreaFor | TextArea |
| Html.CheckBox | Html.CheckBoxFor | Checkbox |
| Html.RadioButton | Html.RadioButtonFor | Radio button |
| Html.DropDownList | Html.DropDownListFor | Dropdown, combobox |
| Html.ListBox | Html.ListBoxFor | multi-select list box |
| Html.Hidden | Html.HiddenFor | Hidden field |
| Password | Html.PasswordFor | Password textbox |
| Html.Display | Html.DisplayFor | Html text |
| Html.Label | Html.LabelFor | Label |
| Html.Editor | Html.EditorFor | Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type. |

**How can we restrict MVC actions to be invoked only by GET or POST?**

By default, each and every action method can be invoked by any HTTP request (i.e. GET, PUT, POST, and DELETE). But you can restrict an action to be invoked only by a specific HTTP request by applying HttpGet or HttpPost or HttpPut or HttpDelete attribute.  
If you want to restrict an action method for HTTP Get request only then decorate it with HttpGet action method selector attribute as given below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [HttpGet]  public ActionResult Index()  {      //TODO:      return View();  } |

**What is filters**

where you want to execute some logic before or after an action method executes. ASP.NET MVC provides filters for this purpose

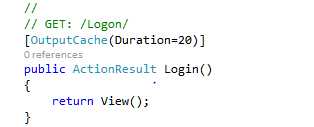
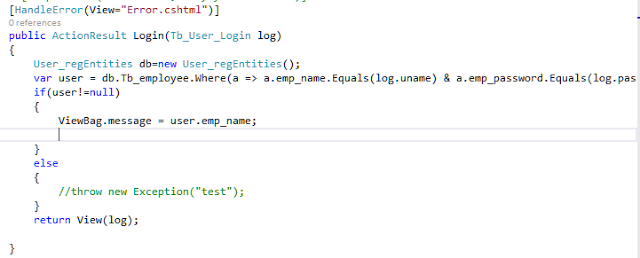
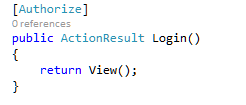
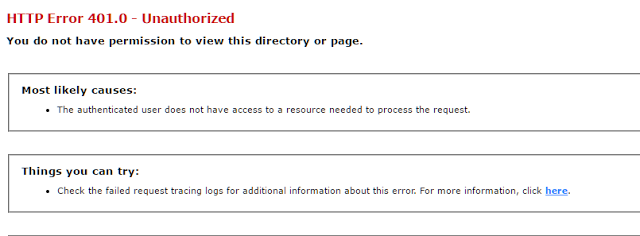
ASP.NET MVC Filter is a custom class where you can write custom logic to execute before or after an action method executes

Filters can be applied to an action method or controller

| **Filter Type** | **Description** | **Built-in Filter** | **Interface** |
| --- | --- | --- | --- |
| Authorization filters | Performs authentication and authorizes before executing action method. | [Authorize], [RequireHttps] | IAuthorizationFilter |
| Action filters | Performs some operation before and after an action method executes. |  | IActionFilter |
| Result filters | Performs some operation before or after the execution of view result. | [OutputCache] | IResultFilter |
| Exception filters | Performs some operation if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline. | [HandleError] | IExceptionFilter |

**Authorization Filters:**  
  
It is used to implement authorization and authentication for action filters  
  
**Result Filters:**Result filters contains logic that gets executed before or after a view result gets executed. E.g. if you want to change view before its get render to browser.  
  
**Exception Filters:**Exception filters are used to handle error, caused by either controller action or controller action results, we can also use it for logging the exceptions

**MVC provides following action filters:**

* **Output Cache:   
    
  **This filter caches the output of action for certain duration. E.g. below code snippet, we are decorating login action with output cache keyword, that will cache the output of login action for 20 seconds( we have given 20 seconds duration).
* **Handle Error:**It handles the error caused by action or controller, if any exception occurs it redirects the action to custom error page. e.g: here in the below code snippet handle error attribute is decorated to login action method, it will redirect to a view called "error.cshtml" when any exception will occur by this action method.   
  
* **Authorize:**  
  It is used for filtering the authorized user to access the resource. E.g. in below code snippet we have decorated an action method with Authorize attribute.  
  
* If we will try to access this action then it will give following error:  
  

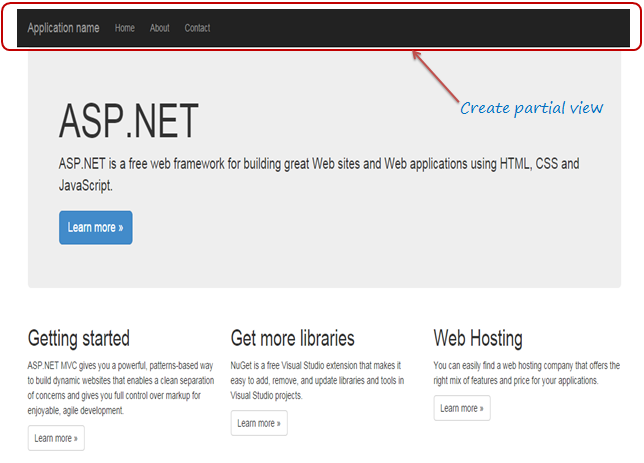
So these are the action filters

**partial views in MVC?**

Partial view is a reusable view, which can be used as a child view in multiple other views.

It eliminates duplicate coding by reusing same partial view in multiple places.

You can use the partial view in the layout view, as well as other content views



**What is Bundling and Minification in MVC?**

Bundling and minification techniques were introduced in MVC 4 to improve request load time.

Bundling allow us to load the bunch of static files from the server into one http request

Bundling technique in MVC 4 allows us to load more than one JavaScript file, MyJavaScriptFile-1.js and MyJavaScriptFile-2.js in one request

Minification technique optimizes script or css file size by removing unnecessary white space and comments and shortening variable names to one character

**Data annotations**

**System.ComponentModel.DataAnnotations** includes the following attributes that impacts the nullability or size of the column.

* Key
* Timestamp
* ConcurrencyCheck
* Required
* MinLength
* MaxLength
* StringLength

We have added a student Controller and added Post action method to add a new student. In this Post Action method, we will apply and test the data annotation validation.  
  
**Student Controller**

1. **using** System.Web.Mvc;
2. **using** DataAnnotationsValidations.Models;
4. **namespace** DataAnnotationsValidations.Controllers
5. {
6. **public** **class** StudentController: Controller
7. {
8. // GET: Student
9. **public** ActionResult Index()
10. {
11. **return** View();
12. }
14. // GET: Student/Create
15. **public** ActionResult Create()
16. {
17. **return** View();
18. }
20. // POST: Student/Create
21. [HttpPost]
22. **public** ActionResult Create(StudentModel student)
23. {
24. **try**
25. {
26. **if** (ModelState.IsValid)
27. {
29. **return** RedirectToAction("Index");
30. }
31. **return** View();
32. } **catch**
33. {
34. **return** View();
35. }
36. }
38. }
39. }

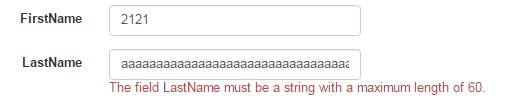
We have added a student view to create Action Method, when we run that view  it will look like the one, shown below:

**Required**  
We will force the student to give their first name and last name, we can decorate the *FirstName*and *LastName*properties of the student Model with *required*attributes.

1. [Required]
2. **public** **string** FirstName
3. {
4. **get**;
5. **set**;
6. }
7. [Required]
8. **public** **string** LastName
9. {
10. **get**;
11. **set**;
12. }

When we run the page without giving the *FirstName*and *LastName*, we will get error message, as shown below:  
  
  
  
These attributes raise a validation error, if either property value is null or empty. Like all the built in validation attributes the required attributes deliver both client side and server side validation.  
  
With these attributes in place, if someone tries to submit the page without proving the *FirstName*and *LastName*, they will see the default error, shown above.   
  
Attributes based on validation ensures that our client and server side validation rules are kept in synchronization because they have been declared in one place.  
  
**StringLength**  
We are forcing the user to enter his name but what happens if he enters a name with enormous length? For student *LastName*, we will set *maximum 60*characters that can be entered. Hence, for doing that, we must decorate *LastName*with *StringLength*attributes.

1. [Required]
2. [StringLength(60)]
3. **public** **string** LastName
4. {
5. **get**;
6. **set**;
7. }

When we run the page after entering *LastName*more than *60*characters, we will get the error message, as shown below:  
  
  
  
We notice that we can attack multiple validation attributes on the single properties. *MinimumLength*is an optional named parameter we can use to specify the minimum length for a string.

1. [Required]
2. [StringLength(60, MinimumLength = 4)]
3. **public** **string** LastName
4. {
5. **get**;
6. **set**;
7. }

Here, last name requires a minimum of four character words.  
  
**RegularExpression**  
Regular expressions are an efficient means to enforce the shape and contains a string value. Suppose, we need to validate the Email ID of the student without sending the Email and Regular expression attributes will do it for us in the way, shown below:

1. [RegularExpression(@ "\A(?:[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?)\Z")]
2. **public** **string** EmailId
3. {
4. **get**;
5. **set**;
6. }

When a user tries to enter an invalid Email ID and submit the page, he will get validation error, as shown below:  
  
RegularExpression   
  
We will see how to display user friendly error message in a later part of this article.