Self-Study

1. What is filter in asp.net mvc and what is it used for ?

In ASP.NET MVC, a filter is a custom class that allows you to run code before or after specific stages in the execution of a controller action method. Filters are used to add cross-cutting concerns like authentication, authorization, logging, error handling, or caching, without cluttering the action methods themselves. They can be applied declaratively using attributes or programmatically.

What is a Filter Used For?

- To execute logic before or after an action method runs.
- To control access through authentication or authorization.
- To handle exceptions and provide error management.
- To enable logging and auditing of user activities.
- To manage caching of action results for performance.

Types of Filters

- Authorization filters: Run first and check user permissions ([Authorize] attribute).
- Action filters: Run before and after an action method.
- Result filters: Run before and after the action result executes.
- Exception filters: Handle exceptions thrown during action execution.

Simple Example of a Custom Action Filter

```
using System;
using System.Web.Mvc;

public class CustomLogFilterAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext
```

```
filterContext)
    {
        // Code to execute before the action executes
        System.Diagnostics.Debug.WriteLine("Action is about to
execute: " + filterContext.ActionDescriptor.ActionName);
    }

    public override void OnActionExecuted(ActionExecutedContext
filterContext)
    {
        // Code to execute after the action executes
        System.Diagnostics.Debug.WriteLine("Action executed: " +
filterContext.ActionDescriptor.ActionName);
    }
}

// Usage on a controller or action method
[CustomLogFilter]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

This filter logs messages before and after the action method runs.

Summary

Filters in ASP.NET MVC provide a clean way to inject behavior at key points of controller action processing, improving modularity and reusability by separating cross-cutting concerns like security, logging, and error handling.

2. What does the status code 405 mean ?

The HTTP status code 405 Method Not Allowed means that the server understands the HTTP request method being used (such as GET, POST, PUT, DELETE), but the target resource does not support that specific method for the requested URL. Essentially, the client made a valid request with a method that the server or resource does not allow.

Key points:

- The server recognizes the method but refuses to fulfill it for that resource.
- The server should respond with an `Allow` header listing HTTP methods that are permitted for the resource.
- Common scenarios include using POST on a URL that only supports GET, or trying to DELETE a static file.

Example

- Trying to submit a form with POST to an endpoint that only supports GET.
- Accessing an API endpoint with PUT when only GET and POST are allowed.

Summary

- Status Code: 405
- Meaning: Method Not Allowed
- Cause: The HTTP method used is not supported by the resource on the server.
- Server Response: Must include an `Allow` header with supported methods.

This status code helps inform the client that it should change the HTTP method to a supported one for successful interaction with the resource.

3. What do Request and Response Header and Body consist of?

HTTP Request consists of:

1. Request Header
    - Metadata about the request, such as:
        - Host (target server)
        - User-Agent (client software)
        - Accept (content types client can process)
        - Authorization (credentials)
        - Cookies, caching directives, language preferences, etc.
2. Request Body (Optional)
    - Contains the data sent from client to server.
    - Typically used in POST, PUT, PATCH requests for sending form data, JSON, XML, or file uploads.
    - Not present in GET or DELETE requests usually.

HTTP Response consists of:

1. Response Header
    - Metadata describing the response, including:
        - Status code (e.g., 200 OK, 404 Not Found)
        - Content-Type (format of the returned data)
        - Content-Length (size of the data)
        - Server (information about the server)
        - Cache-Control, Set-Cookie, Date, and others.
2. Response Body (Optional)

- The actual data returned from the server, such as HTML, JSON, images, or files.
- May be empty in responses like 204 No Content or 304 Not Modified.

This structure enables effective communication between clients and servers through HTTP messages.