



Headspring

Controllers In Depth

Action Filters

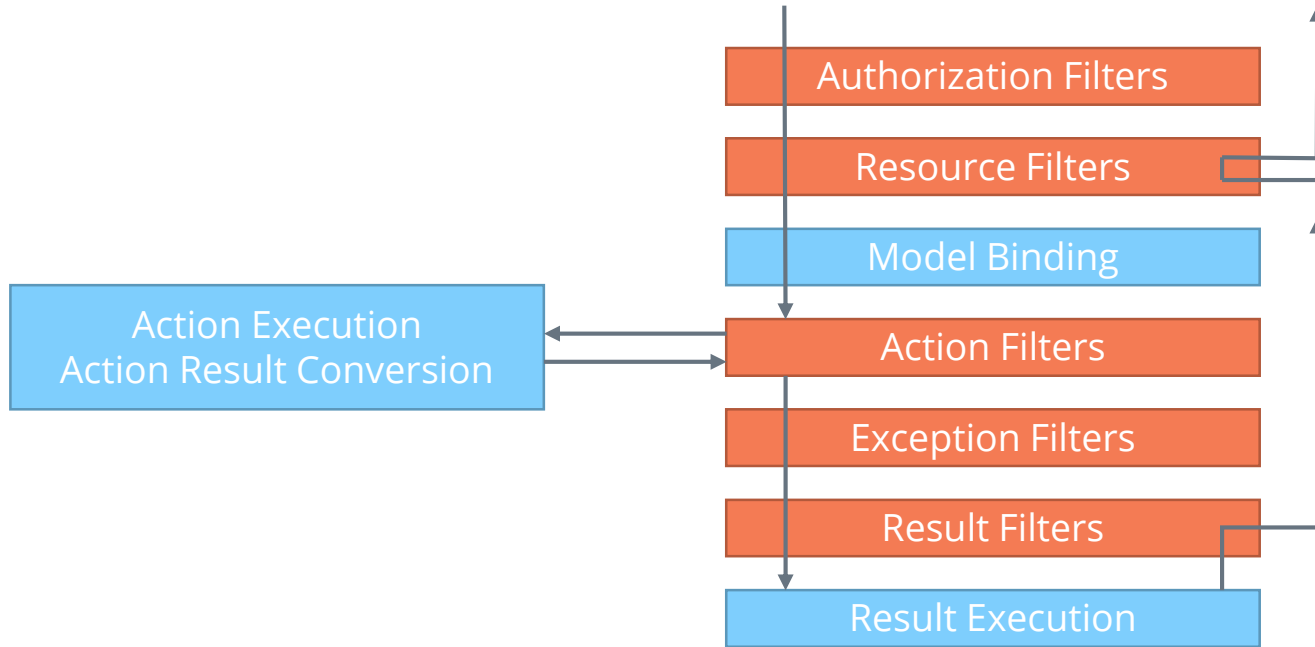
| Main Controller Duplication

- Before/after action code
- Authorization
- Filtering
- Result processing

| Duplication Antidotes

- Filters
 - Action
 - Resource
 - Authorization
 - Exceptions
 - Result
- Model Binders
- Action Results

Filter Pipeline



| Action Filters

- Execute before/after action
- Decorate action with attribute
- Sync and async version
- Attribute or interface

| Common Action Filters usage

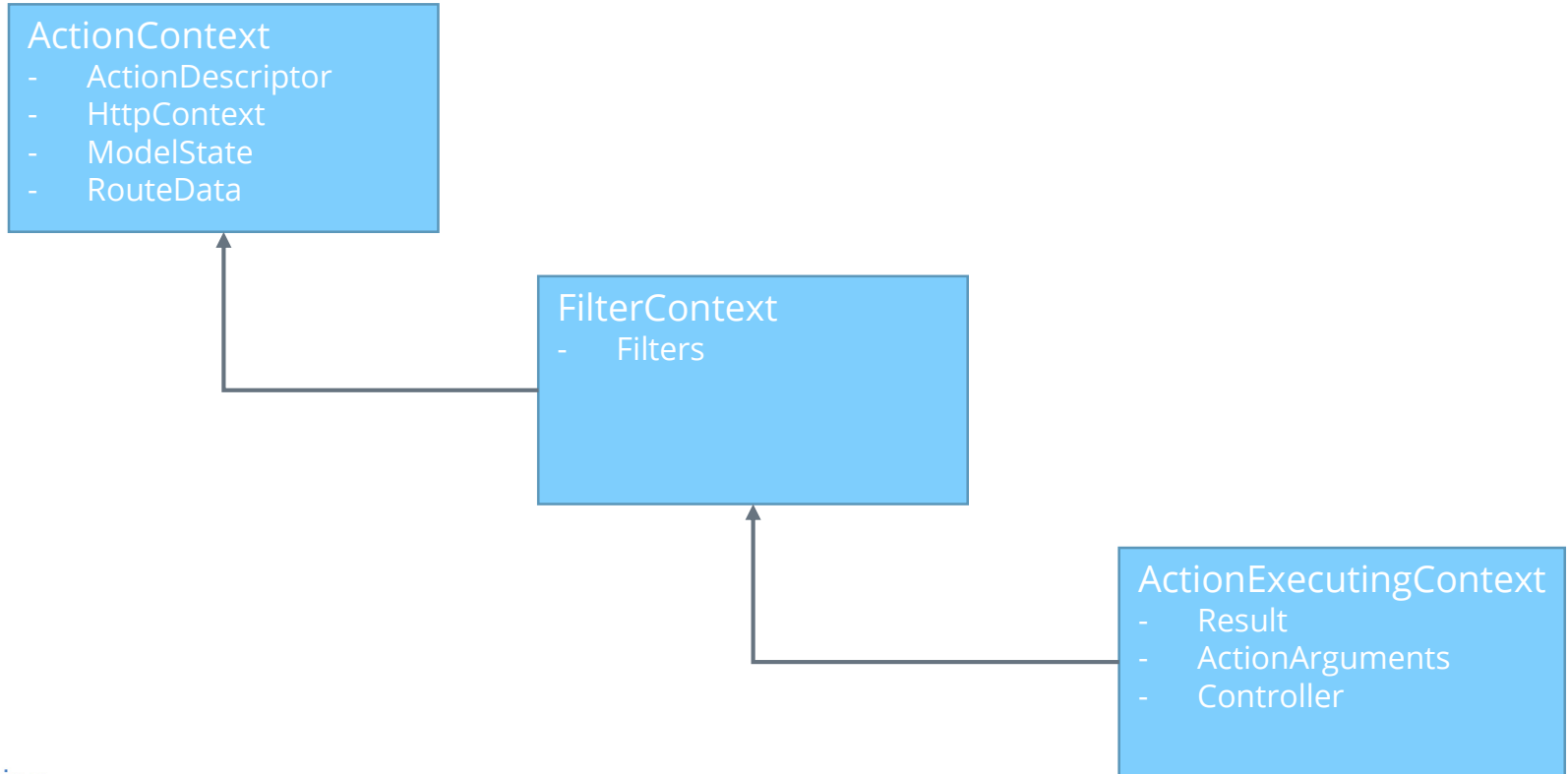
- Populate common ViewData
- Pre/Post process data
- Common execution paths
- Built-in:
 - ResponseCacheFilter

| IActionFilter example

```
public class MyActionFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
        // Before action executes
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        // After action executes
    }
}
```


ActionExecutingContext



| IAsyncActionFilter example

```
public class MyAsyncActionFilter : IAsyncActionFilter
{
    public async Task OnActionExecutionAsync(
        ActionExecutingContext context,
        ActionExecutionDelegate next)
    {
        // Before action executes

        ➡ await next();

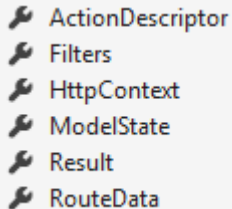
        // After action executes
    }
}
```

Advanced Filters

Authorization Filters

- Execute BEFORE any other filter
- Implement `IAuthorizationFilter` or `IAsyncAuthorizationFilter`

```
public class MyAuthFilter : IAuthorizationFilter
{
    public void OnAuthorization(AuthorizationFilterContext context)
    {
        // Before all other items
    }
}
```



- 🔧 ActionDescriptor
- 🔧 Filters
- 🔧 HttpContext
- 🔧 ModelState
- 🔧 Result
- 🔧 RouteData

| Common Authorization Filters usage

- NOT just security
- Halt execution of action
- Provide alternate result
- Built-in Filters
 - AuthorizeFilter
 - CorsAuthorizationFilter
 - ValidateAntiforgeryTokenAuthorizationFilter
 - RequireHttpsFilter

Exception Filters

- Execute in the Catch block of the main “action invocation” block
- Catches exceptions from filters and action method

```
public class MyExceptionFilter : IExceptionHandler
{
    public void OnException(ExceptionContext context)
    {
        // Custom exception handling/logging
    }
}
```

- ✎ ActionDescriptor
- ✎ Exception
- ✎ ExceptionDispatchInfo
- ✎ ExceptionHandled
- ✎ Filters
- ✎ HttpContext
- ✎ ModelState
- ✎ Result
- ✎ RouteData

| Common Exception Filter Usage

- Handle general errors
- Handle specific exceptions
- Provide alternate result

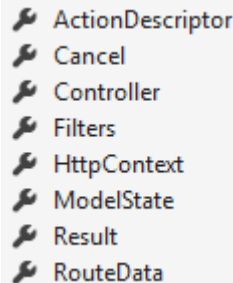
```
public class MyExceptionFilterAttribute : ExceptionFilterAttribute
{
    public override void OnException(ExceptionContext context)
    {
        if (context.Exception.GetType() == typeof(InvalidOperationException))
            context.Result = new ViewResult { ViewName = "InvalidOperation" };
    }
}
```

Result Filters

- Execute before/after ActionResult.ExecuteResult
- Implement IResultFilter or IAsyncResultFilter or inherit ResultFilterAttribute

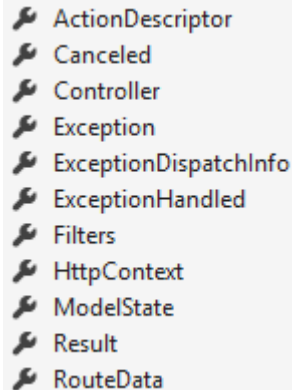
```
public class MyResultFilter : IResultFilter
{
    public void OnResultExecuting(ResultExecutingContext context)
    {
        // Before action result executes
    }

    public void OnResultExecuted(ResultExecutedContext context)
    {
        // After action result executes
    }
}
```



A list of properties available to the IResultFilter interface, each preceded by a wrench icon. The properties are: ActionDescriptor, Cancel, Controller, Filters, HttpContext, ModelState, Result, and RouteData.

- ✎ ActionDescriptor
- ✎ Cancel
- ✎ Controller
- ✎ Filters
- ✎ HttpContext
- ✎ ModelState
- ✎ Result
- ✎ RouteData



A list of properties available to the IAsyncResultFilter interface, each preceded by a wrench icon. The properties are: ActionDescriptor, Canceled, Controller, Exception, ExceptionDispatchInfo, ExceptionHandled, Filters, HttpContext, ModelState, Result, and RouteData.

- ✎ ActionDescriptor
- ✎ Canceled
- ✎ Controller
- ✎ Exception
- ✎ ExceptionDispatchInfo
- ✎ ExceptionHandled
- ✎ Filters
- ✎ HttpContext
- ✎ ModelState
- ✎ Result
- ✎ RouteData

| Common Result Filter Usage

- Caching
- Logging
- Anything surrounding view/formatter execution

```
public class MyAsyncResultFilter : IAsyncResultFilter
{
    public async Task OnResultExecutionAsync(ResultExecutingContext context,
        ResultExecutionDelegate next)
    {
        // Before action result execution

        await next();








        // After action result execution
    }
}
```











Resource Filter

- Executes just after authorization
- Wraps most of the work of a request
- Implement `IResourceFilter` or `IAsyncResourceFilter`

```
public class MyResourceFilter : IResourceFilter
{
    public void OnResourceExecuting(ResourceExecutingContext context)
    {
        // Before execution pipeline
    }

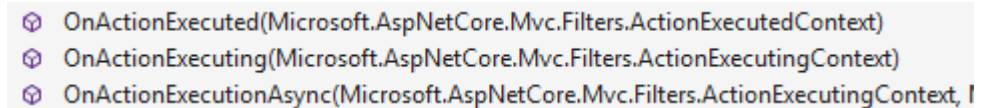
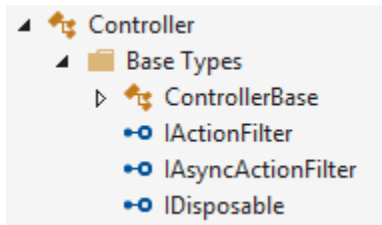
    public void OnResourceExecuted(ResourceExecutedContext context)
    {
        // After execution pipeline
    }
}
```

 `ActionDescriptor`
 `Filters`
 `HttpContext`
 `ModelState`
 `Result`
 `RouteData`
 `ValueProviderFactories`

 `ActionDescriptor`
 `Canceled`
 `Exception`
 `ExceptionDispatchInfo`
 `ExceptionHandled`
 `Filters`
 `HttpContext`
 `ModelState`
 `Result`
 `RouteData`

Controller and Filters

- Controller implements all action filter interfaces
- Override in application controllers



| Controlling Filter Order

- Attribute Filter helpers
- IOrderedFilter

```
[MyExceptionFilter(Order = 4)]
public class HomeController : Controller
{

    public abstract class ExceptionFilterAttribute : Attribute,
        IAsyncExceptionFilter, IExceptionFilter,
        IOrderedFilter
    {
        public int Order { get; set; }
    }
}
```

| Filter application level

- Global
- Controller
- Action

```
services.AddMvc(opt =>
{
    opt.Filters.Add(typeof(MyActionFilter));
    opt.Filters.Add(typeof(MyAsyncResultFilter));
});
```

| Dependency Injection

- Instantiated by container or directly

```
services.AddMvc(opt =>
{
    // Instantiated by built-in container
    opt.Filters.Add(typeof(MyActionFilter));
    // Single instance used for entire lifetime
    opt.Filters.Add(new MyAsyncResultFilter());
});
```

| Dependency Injection with attributes

- Use ServiceFilter to direct container to build
- For DI-enabled filters at controller/action level

```
[ServiceFilter(typeof(MyExceptionFilter))]  
public class HomeController : Controller  
{  
    [ServiceFilter(typeof(MyAsyncActionFilter))]  
    public IActionResult Index()  
    {  
        return View();  
    }  
}
```

| Dependencies inside filters

- Create constructor
- Capture dependency in field

```
public class MyAsyncActionFilter : IAsyncActionFilter
{
    private readonly MyDbContext _dbContext;

    public MyAsyncActionFilter(MyDbContext dbContext)
    {
        _dbContext = dbContext;
    }
}
```


Dependency Injection

| New in ASP.NET MVC Core

- Built-in from the start
- Covers all available extension points
- Controllers, filters, model binders, etc.
- Can be replaced with your own
 - But consider using the built-in one
 - Support for 3rd party containers is limited at the moment
- Supports constructor injection

| Configuring services

```
public interface IOrderTotalCalculator
{
    int Calculate(Order order);
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IOrderTotalCalculator, WcfOrderCalculator>();
}
```

| Lifecycles

- Singleton
 - Once per application
- Scoped
 - Once per HTTP request
- Transient
 - Once per instance request

| Registering services

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IAppConfig, DbAppConfig>();

    services.AddTransient<IOrderTotalCalculator, WcfOrderCalculator>();

    services.AddScoped<IRepository, EFRepository>();
}
```

| Using Dependencies

```
public class HomeController : Controller
{
    private readonly IRepository _repository;

    public HomeController(IRepository repository)
    {
        _repository = repository;
    }
}
```