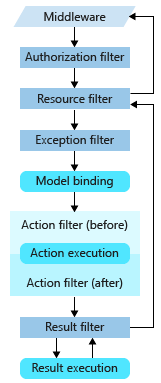
https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.1

* [授权筛选器](https://docs.microsoft.com/zh-cn/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.1#authorization-filters)Authorization filters
* [资源筛选器](https://docs.microsoft.com/zh-cn/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.1#resource-filters)Resource filters
* [操作筛选器](https://docs.microsoft.com/zh-cn/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.1#action-filters)Action filters
* [异常筛选器](https://docs.microsoft.com/zh-cn/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.1#exception-filters)Exception filters
* [结果筛选器](https://docs.microsoft.com/zh-cn/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.1#result-filters)Result filters

下图展示了这些筛选器类型在筛选器管道中的交互方式。



筛选器属性：[可以设置标头，局部过滤]

* ActionFilterAttribute
* ExceptionFilterAttribute
* ResultFilterAttribute
* FormatFilterAttribute
* ServiceFilterAttribute
* TypeFilterAttribute

Filter作用域设置:

全局：

例如：

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc(options =>

{

options.Filters.Add(new AddHeaderAttribute("GlobalAddHeader",

"Result filter added to MvcOptions.Filters")); // an instance

options.Filters.Add(typeof(SampleActionFilter)); // by type

options.Filters.Add(new SampleGlobalActionFilter()); // an instance

});

services.AddScoped<AddHeaderFilterWithDi>();

}

局部：

继承xxAttribute类，用标头的方式

Filter 执行顺序：

* : The *before* code of filters applied globally
  + The *before* code of filters applied to controllers
    - The *before* code of filters applied to action methods
    - The *after* code of filters applied to action methods
  + The *after* code of filters applied to controllers
* The *after* code of filters applied globally

改变默认顺序：

例：

[MyFilter(Name = "Controller Level Attribute", Order=1)]

Order越小越早执行，before code 和after code 的顺序不能变



设置短路：

Result属性，例如：

namespace FiltersSample.Filters

{

public class ShortCircuitingResourceFilterAttribute : Attribute,

IResourceFilter

{

public void OnResourceExecuting(ResourceExecutingContext context)

{

context.Result = new ContentResult()

{

Content = "Resource unavailable - header should not be set"

};

}

public void OnResourceExecuted(ResourceExecutedContext context)

{

}

}

}

如果短路了，就不会执行接下来还有的Filter

------这部分还不清楚---------

ServiceFilterAttribute

TypeFilterAttribute

------这部分还不清楚---------

**Resource filters**

* 实现 IResourceFilter 或 IAsyncResourceFilter 接口，

## Action filters

* Implement either the IActionFilter or IAsyncActionFilter interface.

例子：

public class SampleActionFilter : IActionFilter

{

public void OnActionExecuting(ActionExecutingContext context)

{

// do something before the action executes

}

public void OnActionExecuted(ActionExecutedContext context)

{

// do something after the action executes

}

}

[ActionExecutingContext](https://docs.microsoft.com/zh-cn/dotnet/api/microsoft.aspnetcore.mvc.filters.actionexecutingcontext) 提供以下属性：

* ActionArguments：用于处理对操作的输入。
* Controller：用于处理控制器实例。
* Result：设置此属性会使操作方法和后续操作筛选器的执行短路。 引发异常也会阻止操作方法和后续筛选器的执行，但会被视为失败，而不是一个成功的结果。

OnActionExecuted 方法在操作方法之后运行，可通过 ActionExecutedContext.Result 属性查看和处理操作结果。 如果操作执行已被另一个筛选器设置短路，则 ActionExecutedContext.Canceled 设置为 true。 如果操作或后续操作筛选器引发了异常，则 ActionExecutedContext.Exception 设置为非 NULL 值。 将 ActionExecutedContext.Exception 设置为 null：

## Exception filters

Exception filters implement either the IExceptionFilter or IAsyncExceptionFilter interface

* 没有之前和之后的事件。
* 实现 OnException 或 OnExceptionAsync。
* 处理控制器创建、[模型绑定](https://docs.microsoft.com/zh-cn/aspnet/core/mvc/models/model-binding?view=aspnetcore-2.1)、操作筛选器或操作方法中发生的未经处理的异常。
* 请不要捕获资源筛选器、结果筛选器或 MVC 结果执行中发生的异常。

若要处理异常，请将 ExceptionContext.ExceptionHandled 属性设置为 true，或编写响应。 这将停止传播异常。 异常筛选器无法将异常转变为“成功”。 只有操作筛选器才能执行该转变。

## Result filters

* Implement either the IResultFilter or IAsyncResultFilter interface.