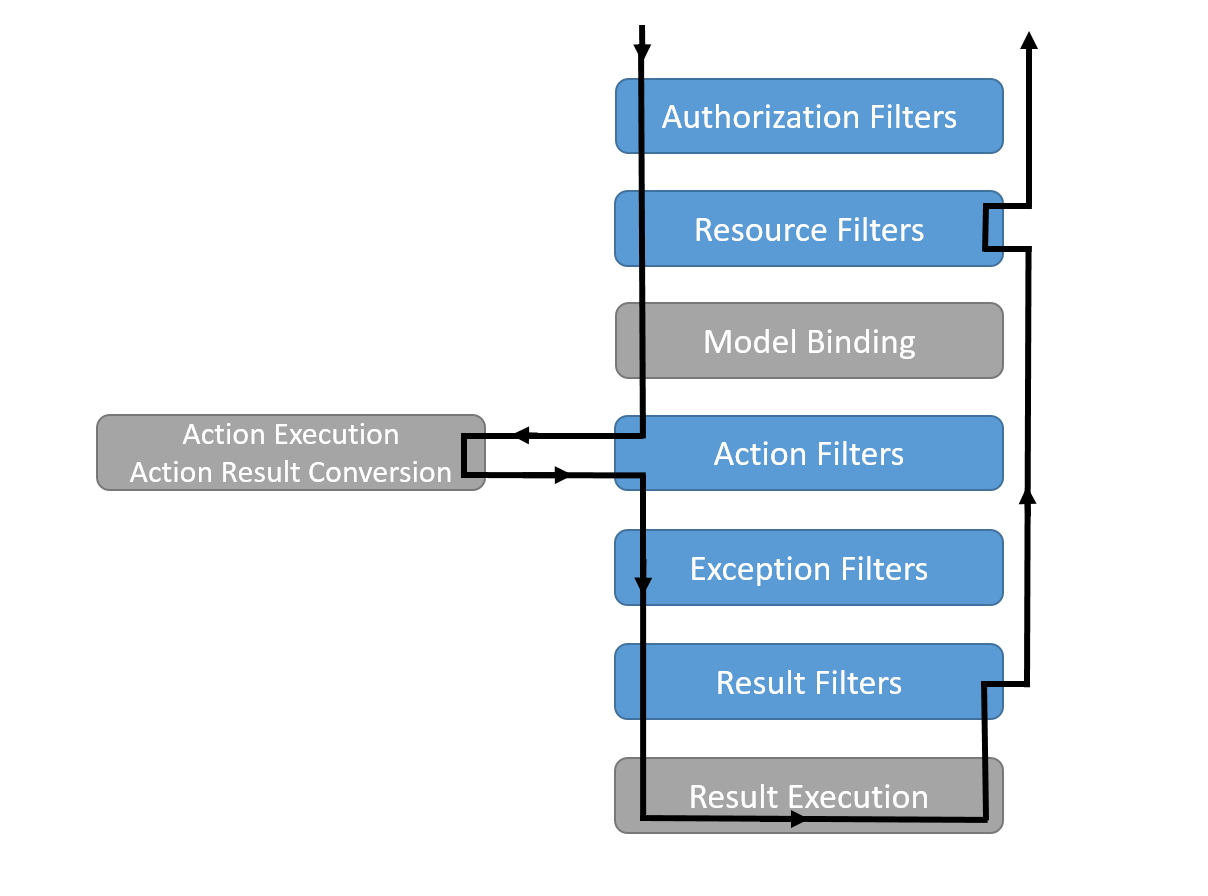
# 控制器与过滤器

Controller(控制器):用于定义和聚合操作(Action)的一个集合,在 Asp.net Core中Controller可以是任何以Controller结尾,或者继承任何以Controller结尾的可实现类.

在Asp.net MVC中Controller负责初始化请求,和实例化模型.

Controller主要确保入站数据是否有效,然后选择返回对应的视图或者API,在良好的分解项目中,控制器不会直接包含数据的访问,或者业务逻辑.而是委托对应的服务去处理.

Filters(过滤器):过滤器可以在特定的阶段执行指定的代码.



授权过滤器=>资源过滤器=>Action过滤器=>异常过滤器=>结果过滤器=>资源过滤器

所有过滤器都可以通过接口来实现同步或者异步的实现,譬如IActionFilter和IAsyncActionFilter.要注意的是,只能实现一个过滤器接口.不能既是异步又是同步的,如果一个类实现了两个接口,最后只有异步的实现会被调用.

过滤器的作用域

全局过滤器.在整个程序的Action生效,控制器使用特性过滤器,Attribute.特定的Action使用特性过滤器,Attribute

例子1，添加全局过滤器

1、添加一个类

public class TestActionFilters : IActionFilter

{

/// <summary>

/// 在管道后执行

/// </summary>

/// <param name="context"></param>

public void OnActionExecuted(ActionExecutedContext context)

{

var path = context.HttpContext.Request.Path.Value;

Console.WriteLine(path);

}

/// <summary>

/// 在执行管道前执行

/// </summary>

/// <param name="context"></param>

public void OnActionExecuting(ActionExecutingContext context)

{

var path = context.HttpContext.Request.QueryString.Value;

Console.WriteLine(path);

}

}

2、在Starup的ConfigureServices函数添加下述代码

public void ConfigureServices(IServiceCollection services)

{

//添加MVC服务,并且里面添加了一个过滤器

services.AddMvc(options=>{ options.Filters.Add(typeof(TestActionFilters)); } );

}

例子2，使用特性过滤器

1、新增一个类TestAttribute

public class TestAttribute : ActionFilterAttribute

{

private readonly string \_name;

private readonly string \_value;

public TestAttribute(string name, string value)

{

\_name = $"hello{name}";

\_value = $"{value} ";

}

public override void OnActionExecuted(ActionExecutedContext context)

{

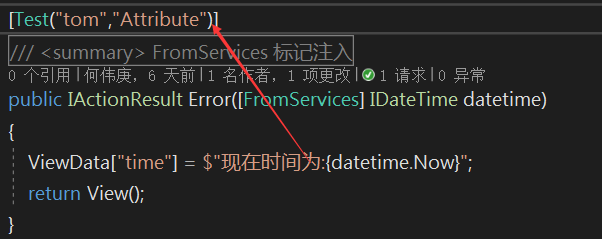
context.HttpContext.Response.Headers.Add(\_name, new string[] { \_value });

base.OnActionExecuted(context);

}

}

2、在控制器上添加特性标记



3、然后运行项目，打完收工

