**ASP.NET MVC过滤器**

　　MVC过滤器是加在 Controller 或 Action 上的一种 Attribute，通过过滤器，MVC 网站在处理用户请求时，可以处理一些附加的操作，如：用户权限验证、系统日志、异常处理、缓存等。MVC 中包含Authorization filter、Action filter、Result filter、Exception filter 四种过滤器。

　　APS.NET MVC中的每一个请求，都会分配给相应的控制器和对应的行为方法去处理，而在这些处理的前前后后如果想再加一些额外的逻辑处理。这时候就用到了过滤器。

**一、MVC支持的过滤器类型有四种：**

　　Authorization(授权)

　　Action（行为）

　　Result（结果）

　　Exception（异常）

|  |  |  |
| --- | --- | --- |
| **滤器类型** | **接口** | **描述** |
| Authorization | IAuthorizationFilter | 此类型（或过滤器）用于限制进入控制器或控制器的某个行为方法 |
| Exception | IExceptionFilter | 用于指定一个行为，这个被指定的行为处理某个行为方法或某个控制器里面抛出的异常 |
| Action | IActionFilter | 用于进入行为之前或之后的处理 |
| Result | IResultFilter | 用于返回结果的之前或之后的处理 |

　　但是默认实现它们的过滤器只有三种，分别是Authorize（授权），ActionFilter，HandleError（错误处理）；各种信息如下表所示

**二、使用Filter的方式**

　　1、通过特性的方式使用Filter

　　2、通过重写Controller方法来使用Filter

**三、Authorize过滤器**

**方法一：使用特性的方式**

　　定义一个类，继承与AuthorizeAttribute，并重写OnAuthorization方法

[复制代码](javascript:void(0);)

1 using System;

2 using System.Collections.Generic;

3 using System.Linq;

4 using System.Web;

5 using System.Web.Mvc;

6

7 namespace MvcApplication2.Filters

8 {

9 public class LoginAttribute:AuthorizeAttribute

10 {

11 public override void OnAuthorization(AuthorizationContext filterContext)

12 {

13 //如果Session中不存在该值，那么表示没有登录

14 if (filterContext.HttpContext.Session["User"] == null)

15 {

16 filterContext.Result = new RedirectResult("/Account/Login");

17 }

18 }

19 }

20 }

[复制代码](javascript:void(0);)

　　这里判断如果，用户没有登录，那么需要返回到登录界面。

　　虽然定义了一个过滤器，但是还需使用过滤器

**使用过滤器的方式也有几种方式：**

1、在方法前加上授权特性

[复制代码](javascript:void(0);)

1 using System;

2 using System.Collections.Generic;

3 using System.Linq;

4 using System.Web;

5 using System.Web.Mvc;

6 using MvcApplication2.Filters;

7

8 namespace MvcApplication2.Controllers

9 {

10 public class HomeController : Controller

11 {

12 //在方法上加入授权特性

13 [Login]

14 public ActionResult Index()

15 {

16 ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

17

18 return View();

19 }

20

21 public ActionResult About()

22 {

23 ViewBag.Message = "Your app description page.";

24

25 return View();

26 }

27

28 public ActionResult Contact()

29 {

30 ViewBag.Message = "Your contact page.";

31

32 return View();

33 }

34 }

35 }

[复制代码](javascript:void(0);)

 　　这种方式，只能对有加上该授权特性的方法进行登录检查，其它没有加的方法无法进行过滤。在上面代码中，只能对该controller中index的action进行过滤，另外的About和Contact无法过滤。

2、在类上加特性，那么表示对该类下所有方法加上该特性

[复制代码](javascript:void(0);)

1 using System.Web;

2 using System.Web.Mvc;

3 using MvcApplication2.Filters;

4

5 namespace MvcApplication2.Controllers

6 {

7 [Login]

8 public class HomeController : Controller

9 {

10 public ActionResult Index()

11 {

12 ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

13

14 return View();

15 }

16

17 public ActionResult About()

18 {

19 ViewBag.Message = "Your app description page.";

20

21 return View();

22 }

23

24 public ActionResult Contact()

25 {

26 ViewBag.Message = "Your contact page.";

27

28 return View();

29 }

30 }

31 }

[复制代码](javascript:void(0);)

　　这种方式虽然不用对每个方法都添加特性，但是也只是对在类上加上了该特性的Controller有作用，其它没有加的会无效

3、注册全局过滤器

　　在实际项目中，一个项目往往会有很多的Controller和Action那么，如果向上面的两种方式来出来都不够理性，通常我们使用全局过滤器。使用方式

[复制代码](javascript:void(0);)

1 using System.Web;

2 using System.Web.Mvc;

3

4 namespace MvcApplication2

5 {

6 public class FilterConfig

7 {

8 public static void RegisterGlobalFilters(GlobalFilterCollection filters)

9 {

10 //在此处添加需要注册的全局过滤器

11 filters.Add(new MvcApplication2.Filters.LoginAttribute());

12 }

13 }

14 }

[复制代码](javascript:void(0);)

　　这样就不用再每个类上添加Login过滤器了，默认对每个Controller的每个Action都使用该特性。但是有个问题，就是连登陆页面也被过滤掉了。在这里使用AllowAnonymous都无效

 　  可以在自定义Filter时，通过获取当前访问路径，判断是否是需要排除的路径，如果是，则跳过

[复制代码](javascript:void(0);)

1 using System;

2 using System.Collections.Generic;

3 using System.Linq;

4 using System.Web;

5 using System.Web.Mvc;

6

7 namespace MvcApplication2.Filters

8 {

9 public class LoginAttribute:AuthorizeAttribute

10 {

11 public override void OnAuthorization(AuthorizationContext filterContext)

12 {

13 //获取当前相对项目的更目录的路径

14 string path = filterContext.HttpContext.Request.CurrentExecutionFilePath;

15 if(!path.StartsWith("/Account/", StringComparison.CurrentCultureIgnoreCase))

16 {

17 //如果Session中不存在该值，那么表示没有登录

18 if (filterContext.HttpContext.Session["User"] == null)

19 {

20 filterContext.Result = new RedirectResult("/Account/Login");

21 }

22 }

23

24 }

25 }

26 }

[复制代码](javascript:void(0);)

　　这里，以/Account/开始的请求都不会执行该授权过滤器。

**方法二：使用重写Controller方法的方式**

1 protected override void OnAuthorization(AuthorizationContext filterContext)

2 {

3 //自定义授权验证代码

4 }

　　这种方式相当于在类上使用授权特性，对于该Controller下的所有方法都有效

**四、异常过滤器**

　　需要继承与HandleErrorAttribute，并重写OnException方法

[复制代码](javascript:void(0);)

1 using System;

2 using System.Collections.Generic;

3 using System.Linq;

4 using System.Web;

5 using System.Web.Mvc;

6

7 namespace MvcApplication2.Filters

8 {

9 public class MyExceptionAttribute:HandleErrorAttribute

10 {

11 public override void OnException(ExceptionContext filterContext)

12 {

13 base.OnException(filterContext);

14

15 //填写验证信息，记录错误日志

16

17 //跳转到错误页面

18 filterContext.Result = new RedirectResult("/Error");

19 }

20 }

21 }

[复制代码](javascript:void(0);)

　　这里base.OnException(filterContext)不能省略，否则无法捕获到异常

　　同时在web.config文件中需要配置，启用自定义异常。    <customErrors mode="On"></customErrors>

**五、行为和结果过滤器**

　　这两种过滤器都是继承于ActionFilterAttribute类

　　行为过滤器：需要选择重写OnActionExecuting或者OnActionExecuted，至于是哪个方法看需求，也可以哪个方法都重写

　　结果过滤器：需要选择重写OnResultExecuting或者OnResultExecuted，至于是哪个方法看需求，也可以哪个方法都重写