[**Web APi之过滤器执行过程原理解析【二】（十一）**](http://www.cnblogs.com/CreateMyself/p/4855419.html)

**http://www.cnblogs.com/CreateMyself/p/4855419.html**

**前言**

上一节我们详细讲解了过滤器的创建过程以及粗略的介绍了五种过滤器，用此五种过滤器对实现对执行Action方法各个时期的拦截非常重要。这一节我们简单将讲述在Action方法上、控制器上、全局上以及授权上的自定义特性的执行过程。

**APiController**

之前有讲到该APiController，也就稍微介绍了，这节我们来详细此Web API控制器的基类：

[复制代码](javascript:void(0);)

1 public abstract class ApiController : IHttpController, IDisposable

2 {

3 // Fields

4 private HttpConfiguration \_configuration;

5 private HttpControllerContext \_controllerContext;

6 private bool \_disposed;

7 private ModelStateDictionary \_modelState;

8 private HttpRequestMessage \_request;

9 private UrlHelper \_urlHelper;

10

11 // Methods

12 protected ApiController();

13 public void Dispose();

14 protected virtual void Dispose(bool disposing);

15 public virtual Task<HttpResponseMessage> ExecuteAsync(HttpControllerContext controllerContext, CancellationToken cancellationToken);

16 protected virtual void Initialize(HttpControllerContext controllerContext);

17 internal static Func<Task<HttpResponseMessage>> InvokeActionWithActionFilters(HttpActionContext actionContext, CancellationToken cancellationToken, IEnumerable<IActionFilter> filters, Func<Task<HttpResponseMessage>> innerAction);

18 internal static Func<Task<HttpResponseMessage>> InvokeActionWithAuthorizationFilters(HttpActionContext actionContext, CancellationToken cancellationToken, IEnumerable<IAuthorizationFilter> filters, Func<Task<HttpResponseMessage>> innerAction);

19 internal static Task<HttpResponseMessage> InvokeActionWithExceptionFilters(Task<HttpResponseMessage> actionTask, HttpActionContext actionContext, CancellationToken cancellationToken, IEnumerable<IExceptionFilter> filters);

20

21 // Properties

22 public HttpConfiguration Configuration { get; set; }

23 public HttpControllerContext ControllerContext { get; set; }

24 public ModelStateDictionary ModelState { get; }

25 public HttpRequestMessage Request { get; set; }

26 public UrlHelper Url { get; set; }

27 public IPrincipal User { get; }

28

29 // Nested Types

30 private class FilterGrouping

31 {

32 // Fields

33 private List<IActionFilter> \_actionFilters;

34 private List<IAuthorizationFilter> \_authorizationFilters;

35 private List<IExceptionFilter> \_exceptionFilters;

36

37 // Methods

38 public FilterGrouping(IEnumerable<FilterInfo> filters);

39 private static void Categorize<T>(IFilter filter, List<T> list) where T: class;

40

41 // Properties

42 public IEnumerable<IActionFilter> ActionFilters { get; }

43 public IEnumerable<IAuthorizationFilter> AuthorizationFilters { get; }

44 public IEnumerable<IExceptionFilter> ExceptionFilters { get; }

45 }

46 }

[复制代码](javascript:void(0);)

我们首先来看看此类中的一个私有类 FilterGrouping ，顾名思义是对过滤器分组，我们查看其构造函数看看：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public FilterGrouping(IEnumerable<FilterInfo> filters)  {      this.\_actionFilters = new List<IActionFilter>();      this.\_authorizationFilters = new List<IAuthorizationFilter>();      this.\_exceptionFilters = new List<IExceptionFilter>();      foreach (FilterInfo info in filters)      {          IFilter instance = info.Instance;          Categorize<IActionFilter>(instance, this.\_actionFilters);          Categorize<IAuthorizationFilter>(instance, this.\_authorizationFilters);          Categorize<IExceptionFilter>(instance, this.\_exceptionFilters);      }  } |

我们仅仅只需 \_actionFilters 为例，其余一样，我们再来看看 Categorize 方法：

[复制代码](javascript:void(0);)

1 private static void Categorize<T>(IFilter filter, List<T> list) where T: class

2 {

3 T item = filter as T;

4 if (item != null)

5 {

6 list.Add(item);

7 }

8 }

[复制代码](javascript:void(0);)

从这里我们可以得知：

**当我们在HttpActionDescriptor初始化创建了封装了Filter对象的FilterInfo的集合列表，此时然后利用此类中的三个属性类型：IActionFilter、IAuthorizationFilter、以及IExceptionFilter进行过滤器分组得到对应过滤器集合列表**

**执行过程原理解析**

下面我们通过例子来看看之执行过程，我们自定义以下五个过滤器

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | /// <summary>     /// 全局的行为过滤器     /// </summary>     public class CustomConfigurationActionFilterAttribute : FilterAttribute, IActionFilter     {         public Task<HttpResponseMessage> ExecuteActionFilterAsync(System.Web.Http.Controllers.HttpActionContext actionContext, System.Threading.CancellationToken cancellationToken, Func<Task<HttpResponseMessage>> continuation)         {             Console.WriteLine(this.GetType().Name);             return continuation();         }     }       /// <summary>     /// 控制器级行为过滤器     /// </summary>     public class CustomControllerActionFilterAttribute : FilterAttribute, IActionFilter     {         public Task<HttpResponseMessage> ExecuteActionFilterAsync(System.Web.Http.Controllers.HttpActionContext actionContext, System.Threading.CancellationToken cancellationToken, Func<Task<HttpResponseMessage>> continuation)         {             Console.WriteLine(this.GetType().Name);             return continuation();         }     }       /// <summary>     /// 控制器方法级行为过滤器     /// </summary>     public class CustomActionFilterAttribute : FilterAttribute, IActionFilter     {         public Task<HttpResponseMessage> ExecuteActionFilterAsync(HttpActionContext actionContext, System.Threading.CancellationToken cancellationToken, Func<Task<HttpResponseMessage>> continuation)         {             Console.WriteLine(this.GetType().Name);             return continuation();         }     }       /// <summary>     /// 控制器级授权访问过滤器     /// </summary>     public class CustomControllerAuthorizationFilterAttribute : FilterAttribute, IAuthorizationFilter     {         public Task<HttpResponseMessage> ExecuteAuthorizationFilterAsync(HttpActionContext actionContext, System.Threading.CancellationToken cancellationToken, Func<Task<HttpResponseMessage>> continuation)         {             Console.WriteLine(this.GetType().Name);             return continuation();         }     }       /// <summary>     /// 控制器方法级授权访问过滤器     /// </summary>     public class CustomControllerActionAuthorizationFilterAttribute : FilterAttribute, IAuthorizationFilter     {         public Task<HttpResponseMessage> ExecuteAuthorizationFilterAsync(HttpActionContext actionContext, System.Threading.CancellationToken cancellationToken, Func<Task<HttpResponseMessage>> continuation)         {             Console.WriteLine(this.GetType().Name);             return continuation();         }     } |

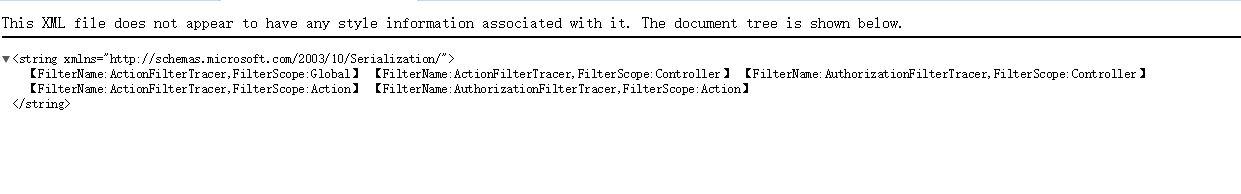
接下来就是实现过滤器，配置文件中配置全局过滤器

|  |  |
| --- | --- |
| 1 | config.Filters.Add(new CustomConfigurationActionFilterAttribute()); |

控制器及方法上过滤器

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | [CustomControllerAuthorizationFilter]  [CustomControllerActionFilter]  public class ProductController : ApiController  {      [CustomActionFilter]      [CustomControllerActionAuthorizationFilter]      public string GetFilter()      {          var sb = new StringBuilder();            var actionSelector = this.Configuration.Services.GetActionSelector();          var actionDesciptor = actionSelector.SelectAction(this.ControllerContext);          foreach (var filterInfo in actionDesciptor.GetFilterPipeline())          {              sb.AppendLine("【FilterName:" + filterInfo.Instance.GetType().Name + ",FilterScope:" + filterInfo.Scope.ToString() + "】");          }          return sb.ToString();      }  } |

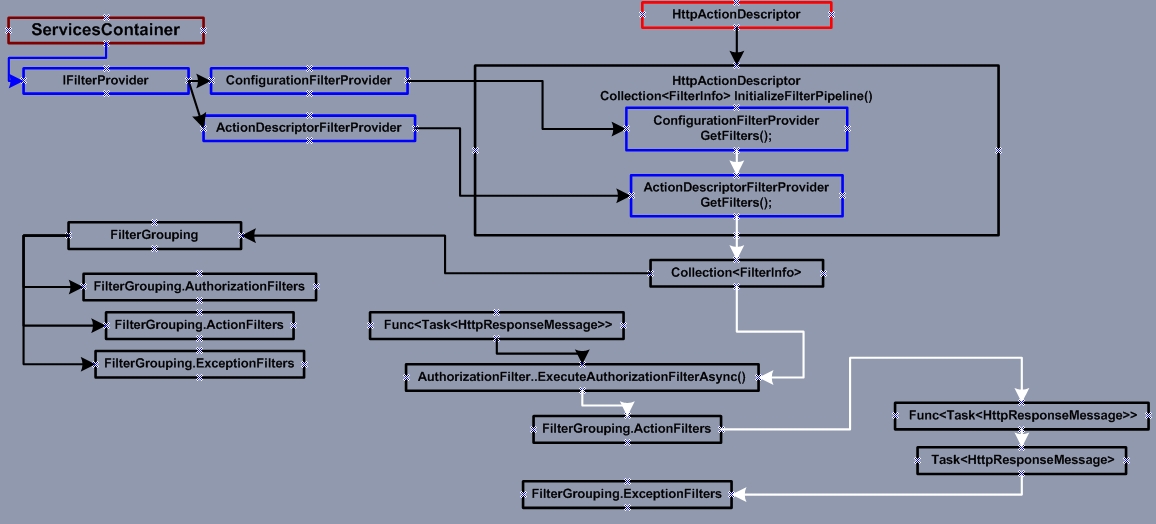
最后来查看其结果：

看到这里是不是有点疑惑怎么按照Global->Controller->Action来进行排序，如果你看过前面文章就会知道这是过滤器管道按照FilterScope来生成的，实际上在服务器端生成的顺序为 CustomControllerAuthorizationFilterAttribute 、 CustomControllerActionAuthorizationFilterAttribute 、 CustomConfigurationActionFilterAttribute 、 CustomControllerActionFilterAttribute 以及 CustomActionFilterAttribute 由此我们得出结论：

**授权过滤器不管任何的FilterScope都是优于行为过滤器，而在同一种类型的过滤器中是根据FilterScope来确定执行顺序的。**

**总结**

有关更多深入的内容就不再探讨，本想多写一点，但是状态不佳加上更多内容比较复杂以免说不太明白云里雾里，想想还是算了，就这样了，下面还是给出其一张执行的详细示意图，来源【[过滤器执行过程](http://jinyuan.blog.51cto.com/8854733/1552724)】



**接下来将通过实例详细讲解Web API中的认证（Authentication）以及授权（Authorization），敬请期待。。。。。。**



**为了方便大家在移动端也能看到我分享的博文，现已注册个人公众号，扫描上方二维码即可，欢迎大家关注，有时间会及时分享相关技术博文。**感谢花时间阅读此篇文章,如果您觉得这篇文章你学到了东西也是为了犒劳下博主的码字不易不妨**打赏**一下吧，让楼主能喝上一杯咖啡，在此谢过了！  
如果您觉得阅读本文对您有帮助，请点一下“**推荐**”按钮，您的**“推荐”**将是我最大的写作动力！  
本文版权归作者和博客园共有，来源网址：[http://www.cnblogs.com/CreateMyself)/](http://www.cnblogs.com/CreateMyself)欢迎各位转载，但是未经作者本人同意，转载文章之后**必须在文章页面明显位置给出作者和原文连接**，否则保留追究法律责任的权利。

**年轻不拼搏？你怎么知道你是人物还是废物！**