[ASP.NET Web API身份验证和授权](http://www.cnblogs.com/youring2/archive/2013/03/09/2950992.html)

英语原文地址：<http://www.asp.net/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api>

本文是作者所理解和翻译的内容。

这篇文章包括两部分：身份验证和授权。

* 身份验证用来确定一个用户的身份。例如，Alice用她的用户名和密码登陆系统，服务器用她的用户名和密码来确定她的身份。
* 授权是判断一个用户是否允许执行某一操作。例如，Alice有获取资源的许可，但不能创建资源。

**身份验证**

Web API有两种方式进行身份验证：在宿主程序中的身份验证和使用 HTTP Message Handlers进行身份验证。

如果你的Web API运行在IIS中，那么身份验证程序就是HTTP Modules，可以使用内置的asp.net身份验证模块进行身份验证，也可以自己写一个身份验证模块完成自定义身份验证。

当在宿主程序中进行身份验证时，宿主程序会创建一个principal对象，这个对象的类实现了[IPrincipal](http://msdn.microsoft.com/en-us/library/System.Security.Principal.IPrincipal.aspx)接口，用来代表当前代码运行的安全上下文。宿主通过设置**Thread.CurrentPrincipal** 将主体附加到当前进程。principal包含一个关联用户信息的**Identity** 对象，如果用户验证通过，**Identity.IsAuthenticated** 属性返回**true**；对于匿名请求，**IsAuthenticated** 返回**false**。关于更多的principals信息，参见[Role-Based Security](http://msdn.microsoft.com/en-us/library/shz8h065.aspx)。

**使用HTTP Message Handlers 进行身份验证**

你可以在 [HTTP message handler](http://www.asp.net/web-api/overview/working-with-http/http-message-handlers)中创建身份验证逻辑来代替使用宿主身份验证机制，在这种机制中，message handler检验Http请求并设置principal。

该在何时使用HTTP Message Handler进行身份验证呢？这里给出了一个这种的参考：

* 一个HTTP Module观察所有通过ASP.NET管道的请求；一个Message Handler只观察被路由处理的Web API请求。
* 你可以对每个路由设置一个Message Handler，这个方便了针对特殊的路由设置特殊的身份验证方案。
* HTTP Module是IIS特有的；Message Handler则与宿主无关，因此你可以同时在web-hosting和self-hosting中使用。
* HTTP Module参与IIS的登陆、审核等处理。
* HTTP Module在管道中更早的执行，如果你使用Message Handler进行身份验证，在handler执行前principal 没有准备好。此外，在response离开Message Handler后，principal 会被恢复到之前的principal 。

综上所述，如果你不需要支持self-hosting，HTTP Module回事更好的选择。如果你需要支持self-hosting，那就考虑使用Message Handler吧。

**设置Principal**

如果你的程序完成了一些自定义的身份验证逻辑，那么你必须设置爱两个地方的principal：

* **Thread.CurrentPrincipal**. 这个属性是在.NET中设置线程 principal的标准途径。
* **HttpContext.Current.User**. 这个属性是ASP.NET 专用的。

下面的代码展示了如何设置principal：

private void SetPrincipal(IPrincipal principal)

{

Thread.CurrentPrincipal = principal;

if (HttpContext.Current != null)

{

HttpContext.Current.User = principal;

}

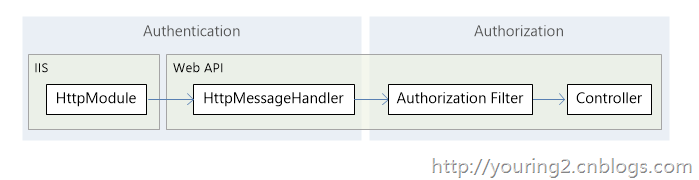
}

对于web-hosting，你必须在这两个地方食指principal，否则安全上下文可能会变得不一致。对于self-hosting，HttpContext.Current是null。为了确定你的代码与宿主无关，所以在赋值之前检查一下HttpContext.Current，就像上面的代码那样。

**授权**

授权发生在管道的后面一些，更接近于controller。它让你在授权访问资源是可以进行更细小粒度的操作。

* *Authorization filters* 在controller Action之前运行。如果请求未授权，filter返回一个错误的response，action不会被调用。
* 在Controller Action内部，你可以通过ApiController.User属性访问当前principal。例如你可能要根据用户来过滤一个资源列表，只返回属于该用户的资源。



**使用[Authorize]属性**

Web API 提供了一个内置的授权过滤器：[AuthorizeAttribute](http://msdn.microsoft.com/en-us/library/system.web.http.authorizeattribute.aspx)。这个过滤器检查用户是否被授权，如果没有，则返回 http status code 401(Unauthorized)，不会调用Action。

你可以添加将过滤器添加到globally，或Controller级别，或Action级别。

globally：要为每一个Web API请求添加约束，则在全局的过滤器列表中添加AuthorizeAttribute。

public static void Register(HttpConfiguration config)

{

config.Filters.Add(new AuthorizeAttribute());

}

Controller：要约束一个特殊的controller的访问，则在controller上添加AuthorizeAttribute。

[Authorize]

public class ValuesController : ApiController

{

public HttpResponseMessage Get(int id) { ... }

public HttpResponseMessage Post() { ... }

}

Action：要约束一个特殊Action的访问，在Action上添加AuthorizeAttribute。

public class ValuesController : ApiController

{

public HttpResponseMessage Get() { ... }

// Require authorization for a specific action.

[Authorize]

public HttpResponseMessage Post() { ... }

}

另外，你还可以约束一个controller访问的同时允许匿名访问特殊的Action，这需要使用[AllowAnonymous]属性。在下面的示例中，Post方法被约束了，而Get方法允许被匿名访问：

[Authorize]

public class ValuesController : ApiController

{

[AllowAnonymous]

public HttpResponseMessage Get() { ... }

public HttpResponseMessage Post() { ... }

}

在上面的例子中，过滤器允许任何被验证的用户访问受约束的方法，只有匿名用户被拒绝。

你还可以限制一些特殊用户或特殊角色的访问：

// Restrict by user:

[Authorize(Users = "Alice,Bob")]

public class ValuesController : ApiController

{

}

// Restrict by role:

[Authorize(Roles = "Administrators")]

public class ValuesController : ApiController

{

}

注意：Web API中的AuthorizeAttribute过滤器是在命名空间System.Web.Http中的，在MVC Controller中有一个相似的过滤器，定义在System.Web.Mvc中，他俩不能兼容使用。

**自定义 Authorization Filters**

一个自定义authorization filter衍生自一下几个类型：

* **AuthorizeAttribute**. 扩展这个类来完成基于当前用户和角色的授权逻辑。
* **AuthorizationFilterAttribute**. 扩展这个类完成同步的授权逻辑，这种方式必须要基于当前用户和角色。
* **IAuthorizationFilter**. 实现这个接口来完成异步的授权逻辑。例如，如果你的授权逻辑使用了异步的IO或网络调用（如果你的授权逻辑是CPU-Bound，那就和简单的衍生自AuthorizationFilterAttribute一样，因为你不需要写一个异步方法）

The following diagram shows the class hierarchy for the **AuthorizeAttribute** class.

下面的图片展示了AuthorizeAttribute的类层级：



**在Controller Action中授权**

有时候，你可能允许一个请求继续下去，但根据principal改变其行为。例如，你返回的信息会依赖用户角色发生改变。在Controller的方法中，你可以通过ApiController.user属性得到当前的principal。

public HttpResponseMessage Get()

{

if (User.IsInRole("Administrators"))

{

// ...

}

}

ps. 勉强的翻译了一下这篇文章的内容，就当是学习的笔记吧。

如果认为此文对您有帮助，别忘了支持一下哦！

作者：[齐飞](http://youring2.cnblogs.com/)

来源：<http://youring2.cnblogs.com/>

声明：本博客原创文字只代表本人工作中在某一时间内总结的观点或结论，与本人所在单位没有直接利益关系。非商业，未授权，贴子请以现状保留，转载时必须保留此段声明，且在文章页面明显位置给出原文连接。