عنوان: **محدود کردن درخواست های Asp.Net Web Api بر اساس Client IP**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۶/۲۸ ۱۷:۱۵
آدرس: www.dotnettips.info
گروه‌ها: ASP.NET Web API, Self-Hosting, IIS-Hosting, OWIN

در بسیاری از سناریوها این موضوع مطرح می‌شود که سرویس‌های طراحی شده بر اساس Asp.Net Web Api، فقط به یک سری آی پی‌های مشخص سرویس دهند. برای مثال اگر Ip کلاینت در لیست کلاینت‌های دارای لایسنس خریداری شده بود، امکان استفاده از سرویس میسر باشد؛ در غیر این صورت خیر. بسته به نوع پیاده سازی سرویس‌های Web api، پیاده سازی این بخش کمی متفاوت خواهد شد. در طی این پست این موضوع را برای سه حالت IIs Host و SelfHost و Owin Host بررسی می‌کنیم.
در اینجا قصد داریم حالتی را پیاده سازی نماییم که اگر درخواست جاری از سوی کلاینتی بود که Ip آن در لیست Ipهای غیر مجاز قرار داشت، ادامه‌ی عملیات متوقف شود.

IIS Hosting:

حالت پیش فرض استفاده از سرویس‌های Web Api همین گزینه است؛ وابستگی مستقیم به System.Web . در مورد مزایا و معایب آن بحث نمی‌کنیم اما اگر این روش را انتخاب کردید تکه کد زیر این کار را برای ما انجام می‌دهد:

```
if (request.Properties.ContainsKey["MS_HttpContext"])
{
    var ctx = request.Properties["MS_HttpContext"] as HttpContextWrapper;
    if (ctx != null)
    {
        var ip = ctx.Request.UserHostAddress;
    }
}
```

برای بدست آوردن شی HttpContext می‌توان آن را از لیست Propertiesهای درخواست جاری به دست آورد. حال کد بالا را در قالب یک Extension Method در خواهیم آورد؛ به صورت زیر:

```
public static class HttpRequestMessageExtensions
{
    private const string HttpContext = "MS_HttpContext";

    public static string GetClientIpAddress(this HttpRequestMessage request)
    {
        if (request.Properties.ContainsKey(HttpContext))
        {
            dynamic ctx = request.Properties[HttpContext];
            if (ctx != null)
            {
                return ctx.Request.UserHostAddress;
            }
        }
        return null;
    }
}
```

Self Hosting:

در حالت Self Host می‌توان عملیات بالا را با استفاده از خاصیت RemoteEndpointMessageProperty انجام داد که تقریبا شبیه به حالت Web Host است. مقدار این خاصیت نیز در شی جاری *HttpRequestMessage* وجود دارد. فقط باید به صورت زیر آن را واکشی نماییم:

```
if (request.Properties.ContainsKey[RemoteEndpointMessageProperty.Name])
{
    var remote = request.Properties[RemoteEndpointMessageProperty.Name] as
RemoteEndpointMessageProperty;
```

```
    if (remote != null)
    {
        var ip = remote.Address;
    }
}
```

خاصیت RemoteEndpointMessageProperty به تمامی درخواست‌ها وارده در سرویس‌های WCF چه در حالت استفاده از Http و چه در حالت Tcp اضافه می‌شود و در اسمبلی System.ServiceModel نیز می‌باشد. از آنجا که Web Api از هسته‌ی WCF استفاده می‌کند (WCF Core) در نتیجه می‌توان از این روش استفاده نمود. فقط باید اسمبلی System.ServiceModel را به پروژه‌ی خود اضافه نمایید.

ترکیب حالت‌های قبلی:

اگر می‌خواهید کدهای نوشته شده شما وابستگی به نوع هاست پروژه نداشته باشد، یا به معنای دیگر، در هر دو حالت به درستی کار کند می‌توانید به روش زیر حالت‌های قبلی را با هم ترکیب کنید.
»در این صورت دیگر نیازی به اضافه کردن اسمبلی System.ServiceModel نیست.

```
public static class HttpRequestMessageExtensions
{
    private const string HttpContext = "MS_HttpContext";
    private const string RemoteEndpointMessage =
"System.ServiceModel.Channels.RemoteEndpointMessageProperty";

    public static string GetClientIpAddress(this HttpRequestMessage request)
    {
        if (request.Properties.ContainsKey(HttpContext))
        {
            dynamic ctx = request.Properties[HttpContext];
            if (ctx != null)
            {
                return ctx.Request.UserHostAddress;
            }
        }

        if (request.Properties.ContainsKey(RemoteEndpointMessage))
        {
            dynamic remoteEndpoint = request.Properties[RemoteEndpointMessage];
            if (remoteEndpoint != null)
            {
                return remoteEndpoint.Address;
            }
        }

        return null;
    }
}
```

مرحله بعدی طراحی یک DelegatingHandler جهت استفاده از IP به دست آمده است .

```
public class MyHandler : DelegatingHandler
{
 private readonly HashSet<string> deniedIps;

    protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
CancellationToken cancellationToken)
    {

      if (deniedIps.Contains(request.GetClientIpAddress()))
        {
            return Task.FromResult( new HttpResponseMessage( HttpStatusCode.Unauthorized ) );
        }

        return base.SendAsync(request, cancellationToken);
    }
}
```

Owin :

زمانی که از Owin برای هاست سرویس‌های Web Api خود استفاده می‌کنید کمی روال انجام کار متفاوت خواهد شد. در این مورد نیز می‌توانید از DelegatingHandlerها استفاده کنید. معرفی DelegatingHandler طراحی شده به Asp.Net PipeLine به صورت زیر خواهد بود:

```
public class Startup
    {
        public void Configuration( IAppBuilder appBuilder )
        {
            var config = new HttpConfiguration();

            var routeHandler = HttpClientFactory.CreatePipeline( new HttpControllerDispatcher( config
), new DelegatingHandler[]
            {
                new MyHandler(),
            } );

            config.Routes.MapHttpRoute(
                name: "Default",
                routeTemplate: "{controller}/{action}",
                defaults: null,
                constraints: null,
                handler: routeHandler
            );

            config.EnsureInitialized();

            appBuilder.UseWebApi( config );
        }
    }
```

اما نکته ای را که باید به آن دقت داشت، این است که یکی از مزایای استفاده از Owin، یکپارچه سازی عملیات هاستینگ قسمت‌های مختلف برنامه است. برای مثال ممکن است قصد داشته باشید که بخش هایی که با Asp.Net SignalR نیز پیاده سازی شده‌اند، قابلیت استفاده از کدهای بالا را داشته باشند. در این صورت بهتر است کل عملیات بالا در قالب یک Owin Middleware عمل نماید تا تمام قسمت‌های هاست شده‌ی برنامه از کدهای بالا استفاده نمایند؛ به صورت زیر:

```
public class IpMiddleware : OwinMiddleware
{
    private readonly HashSet<string> _deniedIps;

    public IpMiddleware(OwinMiddleware next, HashSet<string> deniedIps) :
        base(next)
    {
        _deniedIps = deniedIps;
    }

    public override async Task Invoke(OwinRequest request, OwinResponse response)
    {
        var ipAddress = (string)request.Environment["server.RemoteIpAddress"];

        if (_deniedIps.Contains(ipAddress))
        {
            response.StatusCode = 403;
            return;
        }

        await Next.Invoke(request, response);
    }
}
```

برای نوشتن یک Owin Middleware کافیست کلاس مورد نظر از کلاس OwinMiddleware ارث ببرد و متد Invoke را Override کنید. لیست Ipهای غیر مجاز، از طریق سازنده در اختیار Middleware قرار می‌گیرد. اگر درخواست مجاز بود از طریق دستور Next.Invoke(request,response) کنترل برنامه به مرحله بعدی منتقل می‌شود در غیر صورت عملیات با کد 403 متوقف می‌شود. در نهایت برای معرفی این Middleware طراحی شده به Application، مراحل زیر را انجام دهید.

```
public class Startup
    {
        public void Configuration( IAppBuilder appBuilder )
```

```
        {
            var config = new HttpConfiguration();
            var deniedIps = new HashSet<string> {"192.168.0.100", "192.168.0.101"};



            app.Use(typeof(IpMiddleware), deniedIps);
             appBuilder.UseWebApi( config );
        }
    }
```