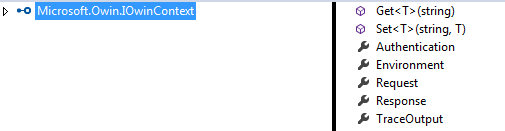
Middleware是什么

如果把HTTP交互理解为一次答题活动,那么Request是问题,Response就是答案,Server是课堂,Middleware就是参与者,注意我这里用的是参与而不是解答,因为我们允许有些Middleware不给出答案.

Middleware有什么资源

要参与答题活动就必须有知识,也就是资源.在OWIN规则中,所有Middleware只能获得并影响一个资源,这个就是OWIN Context

我们来看下这个里面有什么东西.



Authentication : 获取可在当前请求上使用的身份验证(Identity)中间件功能。通过这个属性可以非常便捷在任何Middleware中访问当前的Identity信息.当然至少一个Identity中间件需要被加载,否则这个属性中的内容没有意义.

Environment: 获取已包装的 OWIN 环境。它本质是一个数据字典,一个Middleware利用Key放入一个信息,而另外一个Middleware根据Key拿出来使用.

Request: 获取可公开特定于请求的属性的包装。

Response : 获取可公开特定于响应的属性的包装。

Middleware到底做什么,怎么做

简单来说,Middleware可以做什么怎么做可以归结为以下几点:

获得OWIN Context和它内部封装的所有信息.

从Request中获取请求的所有信息.

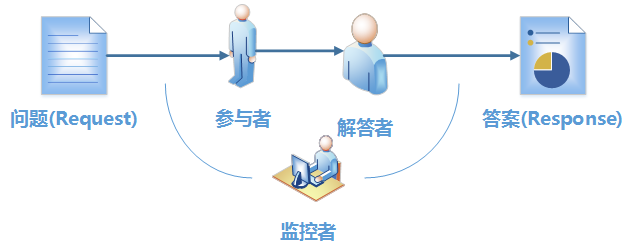
从Environment中获取其他Middleware共享的资源,以便于和其他Middleware交互,或者使用其他Middleware提供的功能.

从Authentication中获取当前的身份验证信息和结果.

通过Response给出,影响,甚至改变Server即将发出的”答案”

Middleware有哪几种类型

根据他们的参与方式,我把他们分成3种情况:



解答者: 了解问题的内容(Request),给出最终答案(Response)

参与者: 了解问题的内容,给出一定的资源(Environment)供其他参与者使用,比较典型的有Session和Identity.

监控者: 在其他参与者开始处理或者处理完毕的时候对当前的Context中的信息进行处理,它也一般不参与解答,有可能在答案中加入一些附加信息. 比较典型的有Logging, Diagnostics.

如何创建Middleware

创建一个的Middleware分以下几个步骤:

引入Microsoft.Owin包

建立一个类

使这个类继承Microsoft.Owin.OwinMiddleware

实现这个类的构造函数

覆盖并实现父类的Invoke函数

一个最为典型的实现如下

[IMG_258](https://www.cnblogs.com/zergcom/p/javascript:void(0);)

using Microsoft.Owin;

using System.Threading.Tasks;

/// <summary>

/// Middleware类必须继承Microsoft.Owin.OwinMiddleware

/// </summary>

public class SampleMiddleware : OwinMiddleware

{

public SampleMiddleware(OwinMiddleware next)

: base(next)

{

//构造函数 }

public override Task Invoke(IOwinContext context)

{

//中间件的实现代码

return Next.Invoke(context);

}

}

[IMG_259](https://www.cnblogs.com/zergcom/p/javascript:void(0);)

绝大部分Middleware需要预设一些属性,这些属性可以通过改造构造函数来实现:

[IMG_260](https://www.cnblogs.com/zergcom/p/javascript:void(0);)

object m\_Options;

public SampleMiddleware(OwinMiddleware next,object options)

: base(next)

{

//引入参数类,并可以再类中使用

m\_Options = options;

}

[IMG_261](https://www.cnblogs.com/zergcom/p/javascript:void(0);)

当然类似的options参数可以有多个.

一个简单的Middleware范例

剩下的工作,就是在Invoke函数中实现当前Middleware的功能,这里给出一个非常简单的实现,来做出一个最简单的功能: 输入结尾为\tick的URL,返回一个纯文本的Response,里面包含当前服务器时间的Tick信息.

public override Task Invoke(IOwinContext context)

{

PathString tickPath = new PathString("/tick");

//判断Request路径为/tick开头

if (context.Request.Path.StartsWithSegments(tickPath))

{

string content = DateTime.Now.Ticks.ToString();

//输出答案--当前的Tick数字

context.Response.ContentType = "text/plain";

context.Response.ContentLength = content.Length;

context.Response.StatusCode = 200;

context.Response.Expires = DateTimeOffset.Now;

context.Response.Write(content);

//解答者告诉Server解答已经完毕,后续Middleware不需要处理

return Task.FromResult(0);

}

else

//如果不是/tick路径,那么交付后续Middleware处理

return Next.Invoke(context);

}

这里提几个要点:

PathString是Miscrosoft.Owin下一个类,封装了URL处理的一些功能.

Task.FromResult(0) 表示一个空的Task,说明该Middleware在某些情况下不再触发后续的Middleware运行—也就是”到此为止”.

Next.Invoke(context)是一个非常标准的实现,把上下文交付下一个Middleware继续处理—相当于”交出接力棒”.

如何使用Middleware

private static void Startup(Owin.IAppBuilder app)

{

//加载Sample Middleware

Console.WriteLine("Sample Middleware loaded...");

app.Use<SampleMiddleware>();

}