[IHttpHandlerFactory](https://msdn.microsoft.com/zh-cn/library/system.web.ihttphandlerfactory(v=vs.100).aspx) 接口用于创建和管理处理请求的 HTTP 处理程序。因此，可以创建一个实现 [IHttpHandlerFactory](https://msdn.microsoft.com/zh-cn/library/system.web.ihttphandlerfactory(v=vs.100).aspx) 接口的类，然后将该类用作 HTTP 处理程序。

在本演练中，将创建一个 HTTP 处理程序工厂，该处理程序工厂为以扩展名 .sample 标识的资源创建两个处理程序。一个处理程序在 HTTP **GET** 请求期间处理资源，另一个处理程序处理 HTTP **POST** 请求。

创建自定义 HTTP 处理程序工厂

首先，创建一个处理程序工厂类。

### 创建 HTTP 处理程序工厂类

如果 ASP.NET 网站还没有 App\_Code 文件夹，请在该站点的根目录下创建一个这样的文件夹。

在 App\_Code 目录中，创建一个名为 HelloWorldHandler 的类。

将下面的代码添加到类文件中。

using System;using System.Web;

class HandlerFactory : IHttpHandlerFactory

{

public IHttpHandler GetHandler(HttpContext context,

string requestType, String url, String pathTranslated)

{

IHttpHandler handlerToReturn;

if ("get" == context.Request.RequestType.ToLower())

{

handlerToReturn = new HelloWorldHandler();

}

else if ("post" == context.Request.RequestType.ToLower())

{

handlerToReturn = new HelloWorldAsyncHandler();

}

else

{

handlerToReturn = null;

}

return handlerToReturn;

}

public void ReleaseHandler(IHttpHandler handler)

{

}

public bool IsReusable

{

get

{

return false;

}

}

}

创建自定义 HTTP 处理程序

### 创建 HelloWorldHandler 和 HelloWorldAsyncHandler 类

在网站的 App\_Code 目录中，创建一个名为 HelloWorldHandler 的类。

using System.Web;

public class HelloWorldHandler : IHttpHandler

{

public HelloWorldHandler()

{

}

public void ProcessRequest(HttpContext context)

{

HttpRequest Request = context.Request;

HttpResponse Response = context.Response;

// This handler is called whenever a file ending

// in .sample is requested. A file with that extension

// does not need to exist.

Response.Write("<html>");

Response.Write("<body>");

Response.Write("<h1>Hello from a synchronous custom HTTP handler.</h1>");

Response.Write("</body>");

Response.Write("</html>");

}

public bool IsReusable

{

// To enable pooling, return true here.

// This keeps the handler in memory.

get { return false; }

}

}

在网站的 App\_Code 目录中，创建一个名为 HelloWorldAsyncHandler 的类。

using System;using System.Web;using System.Threading;

class HelloWorldAsyncHandler : IHttpAsyncHandler

{

public bool IsReusable { get { return false; } }

public HelloWorldAsyncHandler()

{

}

public IAsyncResult BeginProcessRequest(HttpContext context, AsyncCallback cb, Object extraData)

{

context.Response.Write("<p>Begin IsThreadPoolThread is " + Thread.CurrentThread.IsThreadPoolThread + "</p>\r\n");

AsynchOperation asynch = new AsynchOperation(cb, context, extraData);

asynch.StartAsyncWork();

return asynch;

}

public void EndProcessRequest(IAsyncResult result)

{

}

public void ProcessRequest(HttpContext context)

{

throw new InvalidOperationException();

}

}

class AsynchOperation : IAsyncResult

{

private bool \_completed;

private Object \_state;

private AsyncCallback \_callback;

private HttpContext \_context;

bool IAsyncResult.IsCompleted { get { return \_completed; } }

WaitHandle IAsyncResult.AsyncWaitHandle { get { return null; } }

Object IAsyncResult.AsyncState { get { return \_state; } }

bool IAsyncResult.CompletedSynchronously { get { return false; } }

public AsynchOperation(AsyncCallback callback, HttpContext context, Object state)

{

\_callback = callback;

\_context = context;

\_state = state;

\_completed = false;

}

public void StartAsyncWork()

{

ThreadPool.QueueUserWorkItem(new WaitCallback(StartAsyncTask), null);

}

private void StartAsyncTask(Object workItemState)

{

\_context.Response.Write("<p>Completion IsThreadPoolThread is " + Thread.CurrentThread.IsThreadPoolThread + "</p>\r\n");

\_context.Response.Write("Hello World from Async Handler!");

\_completed = true;

\_callback(this);

}

}

### 集成模式下的 IIS 7.0 中注册处理程序工厂

<configuration>

<system.webServer>

<handlers> <add verb="GET,POST" path="\*.sample" name="HandlerFactory" type="HandlerFactory"/> </handlers>

</system.webServer>

</configuration>

配置元素通过类名注册自定义处理程序工厂，并将 .sample 文件扩展名映射到该处理程序。