1. IHttpHandler：

1) HttpContext的常用属性和方法：

Response：

-> 属性：

1. HttpContext.Response.ContentEncoding：指定HTTP响应报文中的编码格式
2. HttpContext.Response.ContentType：指定HTTP响应报文的实体数据类型
3. HttpContext.Response.Buffer/BufferOutput：设置是否启用响应缓存模式进行输出

-> 为提高服务器的性能，默认情况下ASP.Net向浏览器输出数据的时候并不会立即输出到浏览器，而是会将数据缓存至缓冲区中，到合适的时机或者响应结束才会将缓冲区中的数据一起发送到浏览器

1. HttpContext.Response.Cookies[string name,HttpCookie value]：获取返回给客户端进行cookie设置的集合[HttpCookieCollection]，我们可以使用部分键值对集合Diconary中的一些相关属性和方法，但存在区别的是，[HttpCookieCoolection]集合访问一个还未添加过的键的时候不会抛异常，而是会返回一个null

-> 方法：

a. HttpContext.Response.Write(string writeContent)：把字符串写入至HTTP响应体

b. HttpContext.Response.BinaryWrite(byte[] buffer)：把二进制数据写入至HTTP响应体中

c. HttpContext.Response.Redirect(string url,bool endResponse)：指定重定向的URL，并根 据endResponse来确定是否立刻结束之后的代码开始一次重定向的流程

-> 结束当前页面的代码的底层实现原理是通过抛出一个[ThreadAbortException]的异 常来实现的

-> 在MVC开发中，就算是设置了endResponse为true，后续的代码还是会执行，这涉 及到MVC的工作原理的问题

-> 重定向原理：当浏览器接收到的一段响应状态码为[302]且响应头中带有[Location]字 段的响应报文的时候，浏览器则会根据[Location]字段所指定的位置再次发送一次新的 GET请求，这就是重定向的流程

d. HttpContext.Response.Flush()：立刻将缓冲区的数据发送至浏览器

e. HttpContext.Response.Clear()：清空缓冲区中等待发送至浏览器的数据

f. HttpContext.Response.End()：立即抛出一个异常，终止后续的代码执行

Request：

-> 属性：

a.HttpContext.Request.Form[string name,string value]：获取POST方式提交数据的集合[NameValueCollection]，我们可以使用部分键值对集合Diconary中的一些相关属性和方法，但存在区别的是，[NameValueCollection]集合访问一个还未添加过的键的时候不会抛异常，而是会返回一个null

b.HttpContext.Request.QueryString[string name,string value]：获取GET方式提交数据的的 集合[NameValueCollection]，我们可以使用部分键值对集合Diconary中的一些相关属 性和方法，但存在区别的是，[NameValueCollection]集合访问一个还未添加过的键的时 候不会抛异常，而是会返回一个null

c.HttpContext.Request.HttpMethod：获取当前请求的方法

d.HttpContext.Request.Files[string name,HttpPostedFile value]：获取由客户端所上传的文 件集合[HttpFileCollection]，我们可以使用部分键值对集合Diconary中的 一些相关属 性和方法，但存在区别的是，[HttpFileCollection]集合访问一个还未添加过 的键的时候 不会抛异常，而是会返回一个null

f.HttpContext.Request.Url：获取当前所请求的URL地址

g.HttpContext.Request.UrlReferrer：获取上一次所请求的URL地址（请求的来源地址）

-> 此属性可用于防盗链，但是只能用于最基本的防盗链的手段，因为其实现原理是根 据HTTP请求头部字段 [Referer] 来获取的，而请求首部字段是可以伪造的，所以要百 分之百解决防盗链的问题是一件十分困难的事情

i.HttpContext.Request.UserHostAddress：获取发送此次请求的用户的IP地址

j.HttpContext.Request.Cookies[string name,HttpCookie value]：获取从客户端中所接收到的 cookie的集合[HttpCookieCollection]，我们可以使用部分键值对集合 Diconary中的一 些相关属性和方法，但存在区别的是，[HttpCookieCoolection]集合访问一个还未添加过 的键的时候不会抛异常，而是会返回一个null

-> 方法：

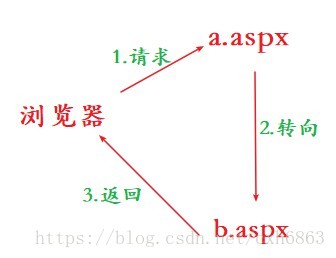
a. HttpContext.Request.MapPath(string virtualPath)：根据虚拟路径virtualPath去获取绝对 路径（在ASP.NET中，对文件或文件夹操作一定要获取物理路径）

Server：

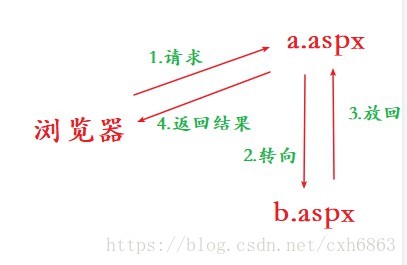
-> 属性：

-> 方法：

1. HttpContext.Server.MapPath(string virtualPath)：根据虚拟路径virtualPath去获取绝对路径（在ASP.NET中，对文件或文件夹操作一定要获取物理路径）
2. HttpContext.Server.Transfer(string virtualPath)：停止当前网页文件在调用该函数后的内容输出，保留在调用该函数前所执行的结果，跳转至虚拟路径virtualPath所指定的网页文件并读取里面的内容，最后，所跳转的网页文件结合上一个网页所保留的结果合并发送回给浏览器进行内容的渲染解析



1. HttpContext.Server.Execute(string virtualPath)：跳转并执行虚拟目录virtualPath所指定的网页文件并读取里面的内容，并把结果返回给调用该函数的网页文件，最后调用该函数的网页文件再把合并两个网页的执行结果一起发送回给浏览器进行渲染解析



1. HttpContext.Server.HtmlEncode(string encodeValue)：将标准的HTML字符串encodeValue进行转义符的编码处理，返回一种不会被浏览器当成是HTML标签处理的一种普通格式的字符串(可作为跨站脚本攻击XSS的一种防御手段)
2. HttpContext.Server.HtmlDecode(string decodeValue)：将已经被转义符编码处理过的字符串decodeValue解码回一种标准格式的HTML字符串并返回
3. HttpContext.Server.UrlEncode(string urlEncode)：将字符串urlEncode中所存在的中文字符进行编码处理并返回(在低版本的浏览器当中，URL是不允许出现中文的，当然现在高版本的浏览器当中已经杜绝了此类问题)
4. HttpContext.Server.UrlDecode(string urlDecode)：将中文字符已经被编码过的字符串urlDecode进行解码处理并返回

Session[string name,object value]

-> 获取操作Session集合[HttpSessionState]，我们可以使用部分键值对集合Diconary中的一些相关属性和方法，但存在区别的是，[HttpCookieCoolection]集合访问一个还未添加过的键的时候不会抛异常，而是会返回一个null

-> ASP.NET中操作Session必须要显式/隐式的继承自 [System.Web.SessionState.IRequiresSessionState] 这个接口，否则该属性的值为null

-> 关于Session的更多简介参考另一篇文档

-> 属性：

1. HttpContext.Session.SessionID：获取当前会话的SessionID
2. HttpContext.Session.IsNewSession：获取当前SessionID是否是一个新的SessionID，也 就是检验这是否是一次新的会话
3. HttpContext.Session.TimeOut：获取或设置当前SessionID存储在Session池中的过期时 间

-> 方法：

1. HttpContext.Session.add(string name,object value)：在当前SessionID的Session池中添加一对键值数据(不推荐做法)

2) HttpPostedFile

-> 属性：

1. HttpPostedFile.ContentType：获取当前文件的文件的类型
2. HttpPostedFile.ContentLength：获取当前文件的文件大小
3. HttpPostedFile.FileName：获取当前文件的文件名
4. HttpPostedFile.InputStream：获取当前文件的流

-> 方法：

1. HttpPostedFile.SaveAs(string filePath)：将当前文件保存至指定目录filePath中
2. HttpCookie

-> 关于Cookie的更多简介，参考另一篇文档

-> 属性：

1. HttpCookie.Name：设置当前cookie的键名
2. HttpCookie.Value：设置当前cookie的值
3. HttpCookie.Values：获取多值cookie的集合



1. HttpCookie.Expires：设置当前cookie的过期时间，当设置了过期，浏览器在进行cookie 存储的时候会把cookie存储的用户磁盘的响应临时目录下，反之浏览器在进行cookie 存储的时候会保存至浏览器所占用的内存当中
2. HttpCookie.Path：设置当前cookie所发送的虚拟路径，可以控制浏览器在访问当前 站点资源的时候，当前cookie只能在我们所指定虚拟路径下进行发送，默认为“/”（意 思是当前站点目录下的所有资源请求，浏览器都会把当前cookie放在HTTP 请求报 文中发送给服务器）
3. HttpCookie.Domain：设置当前cookie相关联的域，实现cookie的跨域

-> 主域中所设置的cookie是可以被子域所引用的，所以我们不需要手动的设置当前 cookie相关联的域，然而在子域所创建的cookie无法被主域所引用的，这时候我们只需 要在子域中设置当前cookie相关联的域为主域的时候，子域中所创建的cookie也能够 被主域所引用了，需要注意的是，当我们在主域中创建cookie并设置其相关联的域不 是主域的时候，虽然响应报文中也包含了我们刚刚所设置的cookie，但是却无法被创建

->请求

->服务器

->内核模式

-> HTTP.SYS

->用户模式

-> IIS

->应用程序池

->w3wp.exe工作者进程

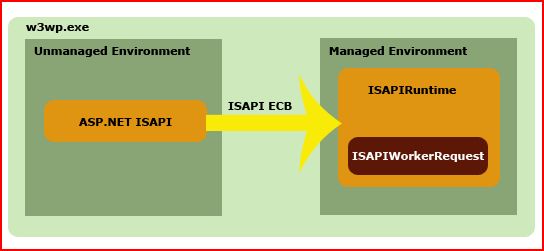
->初始化的时候加载ISAPI Extention

->**初始化CLR**加载.NET Framework运行时

->非托管环境

-> aspnet\_isapi.dll(非托管环境)

->先经过系列系统内部COM级别的调用(比如获得Server Variable的数据，获得通过Post Mehod传回Server的数据等等)，然后会通过一个非托管指针epc(执行控制模块Execution Control Block，是ASP.NET Process Model中托管环境和非托管环境数据交互的唯一通道)去调用一个运行在托管环境中并实现System.Web.Hosting.IISAPIRuntime接口的类ISAPIRuntime的ProcessRequest(InPtr epc)函数(把处在非托管环境中的交互input至托管环境中)，在该函数中，会通过函数的epc Siganature，也就是刚刚说到的那个非托管指针，来创建一个ISAPIWorkerRequest对象，并且把该对象传入到目标函数[HttpRuntime](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Web:2.0.0.0:b03f5f7f11d50a3a/System.Web.HttpRuntime" \o "System.Web.HttpRuntime" \t "https://www.cnblogs.com/artech/archive/2007/09/13/_blank).[ProcessRequestNoDemand](http://www.aisto.com/roeder/dotnet/Default.aspx?Target=code://System.Web:2.0.0.0:b03f5f7f11d50a3a/System.Web.HttpRuntime/ProcessRequestNoDemand(System.Web.HttpWorkerRequest)" \o "void System.Web.HttpRuntime.ProcessRequestNoDemand(HttpWorkerRequest wr);" \t "https://www.cnblogs.com/artech/archive/2007/09/13/_blank)中，当调用该函数起，ASP.NET也就真正意义上的把当前流程带入到 ASP.NET Http Runtime Pipeline中，加载HTTP运行时



->托管环境

->调用AppManagerAppDomainFactory的create() 函数为Web Application 构建 Application Domain

-> **CLR初始化完成**

->把当前流程带入到 ASP.NET Http Runtime Pipeline，加载HTTP运行时

->