Asp.net有三大核心对象：**HttpContext, HttpRequest, HttpResponse**。

除此之外，还有二个对象虽然称不上核心，但仍然比较重要：**HttpRuntime，HttpServerUtility**

**HttpRuntime**

HttpRuntime公开了一个静态方法 UnloadAppDomain() ，这个方法可以让我们用代码重新启动网站。通常用于用户通过程序界面修改了一个比较重要的参数，这时需要重启程序了。

HttpRuntime还公开了一个大家都熟知的静态属性 Cache 。可能有些人认为他/她在使用Page.Cache或者HttpContext.Cache，事实上后二个属性都是HttpRuntime.Cache的【快捷方式】。HttpRuntime.Cache是个非常强大的东西，主要用于缓存一些数据对象，提高程序性能。

可能还有人会担心往Cache里放入太多的东西会不会影响性能，因此有人还想到控制缓存数量的办法。我只想说：缓存容器决定一个对象的保存位置是使用Hash算法的，并不会因为缓存项变多而影响性能，更有趣的是Asp.net的Cache的容器还并非只有一个，它能随着CPU的数量而调整，看这个架式，应该在设计Cache时还想到了高并发访问的性能问题。如果这时你还在统计缓存数量并手工释放某些缓存项，我只能说您在写损害性能的代码。

**HttpServerUtility , HttpUtility**

HttpServerUtility的实例通常以Server的属性公开，但它的提供一些Encode, Decode方法其实调用的是HttpUtility类的静态方法。

HttpUtility公开了一些静态方法，如：

　　HtmlEncode()，应该是使用频率比较高的方法，用于防止注入攻击，它负责安全地生成一段HTML代码。

　　有时我们还需要生成一个URL，那么UrlEncode()方法就能派上用场了，因为URL中并不能包含所有字符，所以要做相应的编码。

　　HttpUtility还有一个方法HtmlAttributeEncode()，它也是用于防止注入攻击，安全地输出一个HTML属性。

HttpUtility还提供了另一个方法：JavaScriptStringEncode()，也是为了防止注入攻击，安全地在服务端输出一段JS代码。

HttpServerUtility除了公开了比较常用的Encode, Decode方法外，还公开了一个非常有用的方法：Execute()，是的，它非常有用，尤其是您需要在服务端获取一个页面或者用户控件的HTML输出时。

**HttpRequest**

在HttpRequest包含的所有输入数据中，有我们经常使用的QueryString, Form, Cookie，它还允许我们访问一些HTTP请求头、浏览器的相关信息、请求映射的相关文件路径、URL详细信息、请求的方法、请求是否已经过身份验证，是否为SSL等等。

HttpRequest的公开属性绝大部分都是比较重要的，这里就简单地列举一下吧。

// 获取服务器上 ASP.NET 应用程序的虚拟应用程序根路径。  
public string ApplicationPath { get; }  
  
// 获取应用程序根的虚拟路径，并通过对应用程序根使用波形符 (~) 表示法（例如，以“~/page.aspx”的形式）使该路径成为相对路径。  
public string AppRelativeCurrentExecutionFilePath { get; }  
  
// 获取或设置有关正在请求的客户端的浏览器功能的信息。  
public HttpBrowserCapabilities Browser { get; set; }  
  
// 获取客户端发送的 cookie 的集合。  
public HttpCookieCollection Cookies { get; }  
  
// 获取当前请求的虚拟路径。  
public string FilePath { get; }  
  
// 获取采用多部分 MIME 格式的由客户端上载的文件的集合。  
public HttpFileCollection Files { get; }  
  
// 获取或设置在读取当前输入流时要使用的筛选器。  
public Stream Filter { get; set; }  
  
// 获取窗体变量集合。  
public NameValueCollection Form { get; }  
  
// 获取 HTTP 头集合。  
public NameValueCollection Headers { get; }  
  
// 获取客户端使用的 HTTP 数据传输方法（如 GET、POST 或 HEAD）。  
public string HttpMethod { get; }  
  
// 获取传入的 HTTP 实体主体的内容。  
public Stream InputStream { get; }  
  
// 获取一个值，该值指示是否验证了请求。  
public bool IsAuthenticated { get; }  
  
// 获取当前请求的虚拟路径。  
public string Path { get; }  
  
// 获取 HTTP 查询字符串变量集合。  
public NameValueCollection QueryString { get; }  
  
// 获取当前请求的原始 URL。  
public string RawUrl { get; }  
  
// 获取有关当前请求的 URL 的信息。  
public Uri Url { get; }  
  
// 从 QueryString、Form、Cookies 或 ServerVariables 集合中获取指定的对象。  
public string this[string key] { get; }  
  
// 将指定的虚拟路径映射到物理路径。  
// 参数:  virtualPath:  当前请求的虚拟路径（绝对路径或相对路径）。  
// 返回结果:  由 virtualPath 指定的服务器物理路径。  
public string MapPath(string virtualPath);

下面我来说说一些不被人注意的细节。

　　HttpRequest的QueryString, Form属性的类型都是NameValueCollection

以下代码演示了这个特殊的现象：

protected void Page\_Load(object sender, EventArgs e)  
{  
    string[] allkeys = Request.QueryString.AllKeys;  
    if( allkeys.Length == 0 )  
        Response.Redirect(  
            Request.RawUrl + "?aa=1&bb=2&cc=3&aa=" + HttpUtility.UrlEncode("5,6,7"), true); //重定向并终止当前页面的执行  
  
    StringBuilder sb = new StringBuilder();  
    foreach( string key in allkeys )  
        sb.AppendFormat("{0} = {1}<br />",   
            HttpUtility.HtmlEncode(key), //字符串编码为html 格式文本

HttpUtility.HtmlEncode(Request.QueryString[key]));  
  
    this.labResult.Text = sb.ToString(); //设置labResult控件的文本  
}

HttpRequest有一个Cookies属性

请看如下代码：

protected void Page\_Load(object sender, EventArgs e)  
{  
    string key = "Key1";  
  
    HttpCookie c = new HttpCookie(key, DateTime.Now.ToString());  
    Response.Cookies.Add(c);  
  
  
    HttpCookie cookie = Request.Cookies[key];  
}

HttpRequest还有二个用于方便获取HTTP数据的属性Params，Item ，后者是个默认的索引器。

为了更清楚地演示这们的差别，请看以下示例代码：

<body>      
    <p>Item结果：<%= this.ItemValue %></p>  
    <p>Params结果：<%= this.ParamsValue %></p>  
      
    <hr />  
      
    <form action="<%= Request.RawUrl %>" method="post">  
        <input type="text" name="name" value="123" />  
        <input type="submit" value="提交" />  
    </form>  
</body>

public partial class ShowItem : System.Web.UI.Page  
{  
    protected string ItemValue;  
    protected string ParamsValue;  
  
    protected void Page\_Load(object sender, EventArgs e)  
    {  
        string[] allkeys = Request.QueryString.AllKeys;  
        if( allkeys.Length == 0 )  
            Response.Redirect("ShowItem.aspx?name=abc", true);  
  
  
        ItemValue = Request["name"]; //Item  
        ParamsValue = Request.Params["name"];        //Params  
    }  
}

**HttpResponse**

　　我们处理HTTP请求的最终目的只有一个：向客户端返回结果。而所有需要向客户端返回的操作都要调用HttpResponse的方法。它提供的功能集中在操作HTTP响应部分，如：响应流，响应头。

我把一些认为很重要的成员简单列举了一下：

// 获取网页的缓存策略（过期时间、保密性、变化子句）。  
public HttpCachePolicy Cache { get; }  
  
// 获取或设置输出流的 HTTP MIME 类型。默认值为“text/html”。  
public string ContentType { get; set; }  
  
// 获取响应 Cookie 集合。  
public HttpCookieCollection Cookies { get; }  
  
// 获取或设置一个包装筛选器对象，该对象用于在传输之前修改 HTTP 实体主体。  
public Stream Filter { get; set; }  
  
// 启用到输出 Http 内容主体的二进制输出。  
public Stream OutputStream { get; }  
  
// 获取或设置返回给客户端的输出的 HTTP 状态代码。默认值为 200 (OK)。  
public int StatusCode { get; set; }  
  
// 将 HTTP 头添加到输出流。  
public void AppendHeader(string name, string value);  
  
// 将当前所有缓冲的输出发送到客户端，停止该页的执行，并引发EndRequest事件。  
public void End();  
  
// 将客户端重定向到新的 URL。指定新的 URL 并指定当前页的执行是否应终止。  
public void Redirect(string url, bool endResponse);  
  
// 将指定的文件直接写入 HTTP 响应输出流，而不在内存中缓冲该文件。  
public void TransmitFile(string filename);  
  
// 将 System.Object 写入 HTTP 响应流。  
public void Write(object obj);

比如：Response.StatusCode = 500;

**HttpContext**

HttpRequest, HttpResponse分别控制了输入输出,它们在哪里创建的呢，哪里保存有它们最原始的引用呢?答案当然是：HttpContext 。而且我前面提到另二个实力不错的选手(HttpServerUtility和Cache)，也都是它的手下。因此，任何事情，找到它就算是有办法了。

虽然HttpContext也公开了一些属性和方法，但我认为最重要的还是上面提到的那些对象的引用。

　　这里再补充二个上面没提到的实例属性：User, Items

User属性保存于当前请求的用户身份信息。如果判断当前请求的用户是不是已经过身份认证，可以访问：Request.IsAuthenticated这个实例属性。

Items属性，如果希望在一次请求的过程中保存一些临时数据，那么，这个属性是最理想的存放容器了。它会在下次请求重新创建，因此，不同的请求之间，数据不会被共享。

HttpContext有个静态属性Current，因为这个属性，在Asp.net的世界里，您可以在任何地方访问Request, Response, Server, Cache，还能在任何地方将一些与请求有关的临时数据保存起来，这绝对是个非常强大的功能。

很多人都说WebForms框架搞得代码没法测试，通常也是的确如此。

　　我看到很多人在页面的CodeFile中写了一大堆的控件操作代码，还混有很多调用业务逻辑的代码，甚至在类库项目中还中访问QueryString, Cookie。再加上诸如ViewState, Session这类【有状态】的东西大量使用，这样的代码是很难测试。

　　换个视角，看看MVC框架为什么说可测试性会好很多，理由很简单，你很少会需要使用HttpRequest, HttpRespons，从Controller开始，您需要的数据已经给您准备好了，直接用就可以了。但MVC框架并不能保证写的代码就一定能方便的测试，比如：您继续使用HttpContext.Current.XXXXX而不使用那些HttpXxxxxBase对象。