1. 读取表中元素提交的数据

表中数据<input type=”text” name=”input\_name”>

服务器端读取context.Request[“input\_name”] or Request[“input\_name”]

1. 响应请求，向客户端输入内容

context.Response.Write(“Hello”) or context.Response.Write(“<input type…..>”)

1. 页面响应问题

当你在Example.html里提交一个这样的表单<form action=”Handler.ashx”>

得到的响应地址是localhost:port/Handler.ashx?somedata…

1. @操作符加在C# string前的作用

例如：@"c:\Docs\Source\a.txt" // rather than "c:\\Docs\\Source\\a.txt"

@"""Ahoy!"" cried the captain." // "Ahoy!" cried the captain.

1. Response.ContentType

即使在程序中将Response.ContentType=”text/html”，响应的URL仍然是localhost:port/Handler.ashx

1. 使用File类读写html文件

string filePath = context.Server.MapPath("HtmlPage.html");  
string content = System.IO.File.ReadAllText(filePath);  
context.Response.Write(content);

The [MapPath](https://msdn.microsoft.com/en-us/library/ms524632(v=vs.90).aspx) method maps the specified relative or virtual path to the corresponding physical directory on the server.

1. 占位符@

在html中使用占位符

<form action="Handler.ashx">  
        Name:<input type="text" name="text1" value=""/><input type="submit" value="submit" />  
        <input type="hidden" name="isPostBack" value="true"/>  
        Name:<input type="text" name="text1" value="@value"/><input type="submit" value="submit" />  
    </form>  
    @msg

在服务器端替换占位符的值

string filePath = context.Server.MapPath("HtmlPage.html");

string content = System.IO.File.ReadAllText(filePath);

content=content.Replace("@value",name);

content=content.Replace("@msg",msg)；

1. 内存数据的自动增长

在html中<form action="Handler.ashx" method="post">

<input type="hidden" name="isPostBack" value="true" />

Value:<input type="text" name="text1" value="@value" /><input type="submit" value="Increse" />

</form>

服务器端代码

public void ProcessRequest (HttpContext context) {

string value = context.Request["text1"];

string isPostBack = context.Request["isPostBack"];

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

if (isPostBack=="true")

{

int temp = Convert.ToInt32(value);

temp++;

value = temp.ToString();

}

else

{

value = "0";

}

string filePath = context.Server.MapPath("HtmlPage.html");

string content = System.IO.File.ReadAllText(filePath);

content = content.Replace("@value", value);

context.Response.Write(content);

}

1. View State

<https://msdn.microsoft.com/en-us/library/ms972976.aspx>

The Role of View State

View state's purpose in life is simple: it's there to persist state across postbacks. (For an ASP.NET Web page, its state is the property values of the controls that make up its control hierarchy.) This begs the question, "What sort of state needs to be persisted?" To answer that question, let's start by looking at what state *doesn't* need to be persisted across postbacks. Recall that in the instantiation stage of the page life cycle, the control hierarchy is created and those properties that are specified in the declarative syntax are assigned. Since these declarative properties are automatically reassigned on each postback when the control hierarchy is constructed, there's no need to store these property values in the view state.

For example, imagine we have a Label Web control in the HTML portion with the following declarative syntax:

<asp:Label runat="server" Font-Name="Verdana"

Text="Hello, World!"></asp:Label>

When the control hierarchy is built in the instantiation stage, the Label's Text property will be set to "Hello, World!" and its Font property will have its Name property set to Verdana. Since these properties will be set each and every page visit during the instantiation stage, there's no need to persist this information in the view state.

What needs to be stored in the view state is any programmatic *changes* to the page's state. For example, suppose that in addition to this Label Web control, the page also contained two Button Web controls, a Change Message Button and an Empty Postback button. The Change Message Button has a Click event handler that assigns the Label's Text property to "Goodbye, Everyone!"; the Empty Postback Button just causes a postback, but doesn't execute any code. The change to the Label's Text property in the Change Message Button would need to be saved in the view state. To see how and when this change would be made, let's walk through a quick example. Assuming that the HTML portion of the page contains the following markup:

<asp:Label runat="server" ID="lblMessage"

Font-Name="Verdana" Text="Hello, World!"></asp:Label>

<br />

<asp:Button runat="server"

Text="Change Message" ID="btnSubmit"></asp:Button>

<br />

<asp:Button runat="server" Text="Empty Postback"></asp:Button>

And the code-behind class contains the following event handler for the Button's Click event:

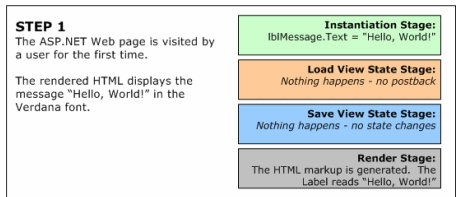
private void btnSubmit\_Click(object sender, EventArgs e)

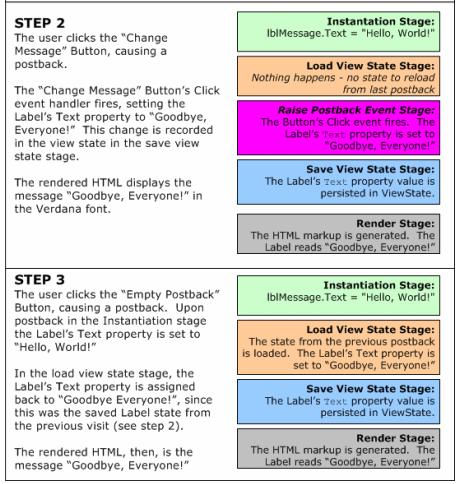
{

lblMessage.Text = "Goodbye, Everyone!";

}

Figure 4 illustrates the sequence of events that transpire, highlighting why the change to the Label's Text property needs to be stored in the view state.





To understand why saving the Label's changed Text property in the view state is vital, consider what would happen if this information were not persisted in view state. That is, imagine that in step 2's save view state stage, no view state information was persisted. If this were the case, then in step 3 the Label's Text property would be assigned to "Hello, World!" in the instantiation stage, but would not be reassigned to "Goodbye, Everyone!" in the load view state stage. Therefore, from the end user's perspective, the Label's Text property would be "Goodbye, Everyone!" in step 2, but would seemingly be reset to its original value ("Hello, World!") in step 3, after clicking the Empty Postback button.

1. ASP.NET Lifecycle

