القسم الثاني– التقنيات من جهة المخدم Server Side

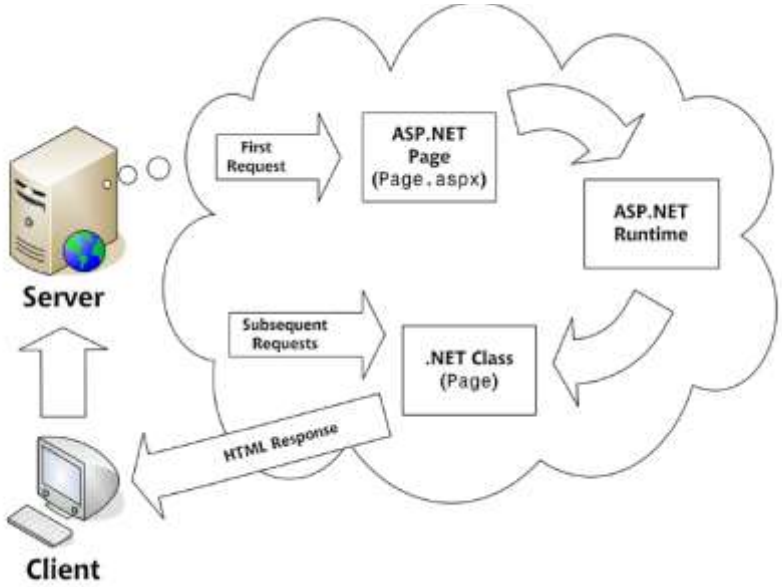
الفصل الثامن– دورة حياة وثيقة الـ ASP.NET

Life Cycle of ASP.NET Page

مقدمة

عندما يقوم الزبون Client بطلب صـــفحة ASP.NET، يقوم المخدم بإنشـــاء منتســـخ Instance من صف الصفحة. ويقوم هذا المنتسخ بإنشاء صفحة HTML التي تمثل الجواب الذي يرسل إلى الزبون.

وعندما يعيد الزبون التفاعل مع الصـــفحة بضـــغط زر إرســـال مثلاً، يقوم المخدم بإنشـــاء صـفحة الجواب من دون أن يعيد عملية الترجمة Compilation. لهذا السبب تكون الإستجابة في الطلب الأول دائماً أطول من الطلبات اللاحقة.



دورة حياة وثيقة ASP.NET Life Cycle

يكون لكل وثيقة ASP.NET نوعين من الطلب request:

- الطلب الأول initial request والذي ينتج عن طلب الوثيقة وإظهار نماذجها للمستخدم.
- الطلب اللاحق postback request والذي يكون بعد تغيير قيم عناصر النموذج من قبل المستخدم. يُدعى هذا الطلب بالطلب اللاحق postback لأن قيم عناصر النموذج تُعاد إلى الوثيقة على المخدم.

لتوضيح تسلسل الأحداث لوثيقة ASP.NET تحوي نموذج لنأخذ المثال التالي:

```
<!-- ex3.aspx
A simple example of an ASP.NET document with HTML controls.
It uses textboxes to get the name and age of the client,
which are then displayed.
-->
<%@ Page language="c#" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
    <head> <title> Ex3 </title>
    </head>
    <body>
      <form runat = "server">
        <p>
          Your Name:
          <input type = "text" id = "name" runat = "server" />
          <br />
          Your age:
          <input type = "text" id = "age" runat = "server" />
          <br />
          <input type = "submit" value = "submit"  />
       <br />
       <% if (IsPostBack) { %>
          Hello <%= name.Value %> <br />
          You are <%= age.Value %> years old <br />
          <% } %>
       </p>
     </form>
  </body>
</html>
```

لاحظ أن كل من النموذج وعناصر التحكم يجب أن تحوي القيمة server للواصفة runat.

يقوم ASP.NET ضمنياً بتخزين حالة كل عنصر تحكم لمنتسخ صف الوثيقة document class instance قبل أن يُعيد المخدم خرج المنتسخ إلى الزبون. تُخزن هذه المعلومات في عنصر تحكم مخفي على الصفحة يُدعى ViewState:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
```

عندما تُعاد الوثيقة إلى المخدم تُستخدم بيانات ViewState ضمنياً لإعطاء قيم ابتدائية للمنتسخ الجديد عند إعادة توليده. تسمح ViewState إذاً بالمحافظة على الحالة (قيم عناصر النموذج) بين الطلبات المتتالية لنفس الصفحة.

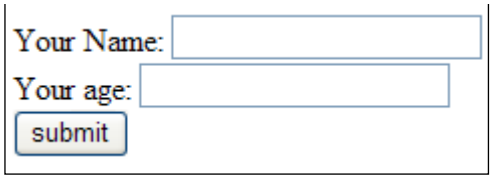
تُبين القائمة التالية الأمور التي تحدث إذا طُلبت الوثيقة السابقة، فقُدمت للمتصفح، ثم قام المستخدم بتعبئة صناديق النص فيها، ثم أعادها للمخدم، ثم أُعيدت مرة أخرى للمتصفح:

1- يطلب المستخدم الوثيقة ex3.aspx.
2- تتمّ ترجمة الوثيقة المطلوبة في المخدم فيُنشئ صف الوثيقة ويُستدعى باني الصف.
3- يتمّ إعطاء قيم ابتدائية لحالة عناصر التحكم باستخدام ViewState (في الطلب الأول لاتحوي ViewState أية بيانات).
4- تُستخدم بيانات النموذج للطلب لإسناد حالة عناصر التحكم لمنتسخ صف الوثيقة (لايوجد بيانات نموذج في الطلب الأول).
5- تُسجل الحالة الحالية للمنتسخ في الحقل المخفي ViewState.
6- يُنفذ المنتسخ وتُعاد النتيجة إلى الزبون.
7- يُمسح كل من صف الوثيقة والمنتسخ من المخدم.
8- يقوم المستخدم بالتفاعل مع النموذج في الوثيقة (تغيير قيم بعض عناصر النموذج).
9- يقوم المستخدم بطلب إعادة postback إلى المخدم (عند النقر على زر الإرسال submit مثلاً).
10- تتمّ ترجمة الوثيقة المطلوبة في المخدم فيُنشئ صف الوثيقة ويُستدعى باني الصف.
11- يتمّ إسناد حالة عناصر التحكم في المنتسخ باستخدام ViewState.
12- تُستخدم بيانات نموذج الطالب لإسناد قيم لحالة عناصر التحكم لمنتسخ صف الوثيقة.

13- تُسجل الحالة الحالية للصف في ViewState.

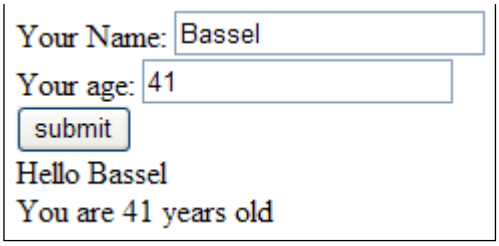
14- يتمّ تنفيذ المنتسخ وتُعاد النتيجة إلى الزبون. ويُمسح الصف والمنتسخ من المخدم.

تكون الوثيقة المنشأة من صف الوثيقة المترجم من الوثيقة ex3.aspx كمايلي:




```
<html xmlns="http://www.w3.org/1999/xhtml" >
   <head> <title> Ex3 </title>
   </head>
   <body>
  <form name="ctl00" method="post"
        action="ex3.aspx" id="ctl00">
<div>
<input type="hidden"
name="__VIEWSTATE"
id="__VIEWSTATE"
value="/wEPDwULLTEzNTg2NzExNzZkZPMc7/+Qlyt5as8R30c8Ya5ME5fW" />
</div>
<p>
          Your Name:
          <input name="name" type="text" id="name" />
          <br />
          Your age:
          <input name="age" type="text" id="age" />
          <br />
          <input type = "submit" value = "submit"  />
        <br />
        </p>
<div>
                              <input type="hidden"
name="__EVENTVALIDATION"
 id="__EVENTVALIDATION"
value="/wEWAwKL1uvQCgL7uPQdAtCCr6oGMaO8C59NSjo1aHXJ1cGerJ0RLFk=" />
</div></form>
  </body>
</html>
```

وبعد أن تُعبأ من قبل المستخدم وترسل للمخدم (بالنقر على submit) كما يلي:




```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Ex3 </title>
</head>
<body>
    <form name="ctl00" method="post"
       action="ex3.aspx" id="ctl00">
        <div>
            <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
             value="/wEPDwULLTEzNTg2NzExNzZkZPMc7/+Qlyt5as8R30c8Ya5ME5fW"
/>
        </div>
        <p>
            Your Name:
           <input name="name" type="text" id="name" value="Bassel" />
            <br />
            Your age:
            <input name="age" type="text" id="age" value="40" />
            <br />
            <input type="submit" value="submit" />
            <br />
            Hello Bassel
            <br />
            You are 40 years old
            <br />
        </p>
        <div>
            <input          type="hidden"          name="__EVENTVALIDATION"
id="__EVENTVALIDATION"

value="/wEWAwKL1uvQCgL7uPQdAtCCr6oGMaO8C59NSjo1aHXJ1cGerJ0RLFk=" />
        </div>
    </form>
```

```
    </body></html>
```

تختلف هذه الوثيقة عن النسخة الأصلية من ex3.aspx في ثلاثة مواضع:

1- تحوي عنصر التحكم المخفي ViewState والذي يحوي بيانات النموذج بشكل مشفر.
2- يكون للنموذج اسم ومعرف (ct100).
3- تمّ استبدال كتلة الإعادة البرمجية بخرجها.

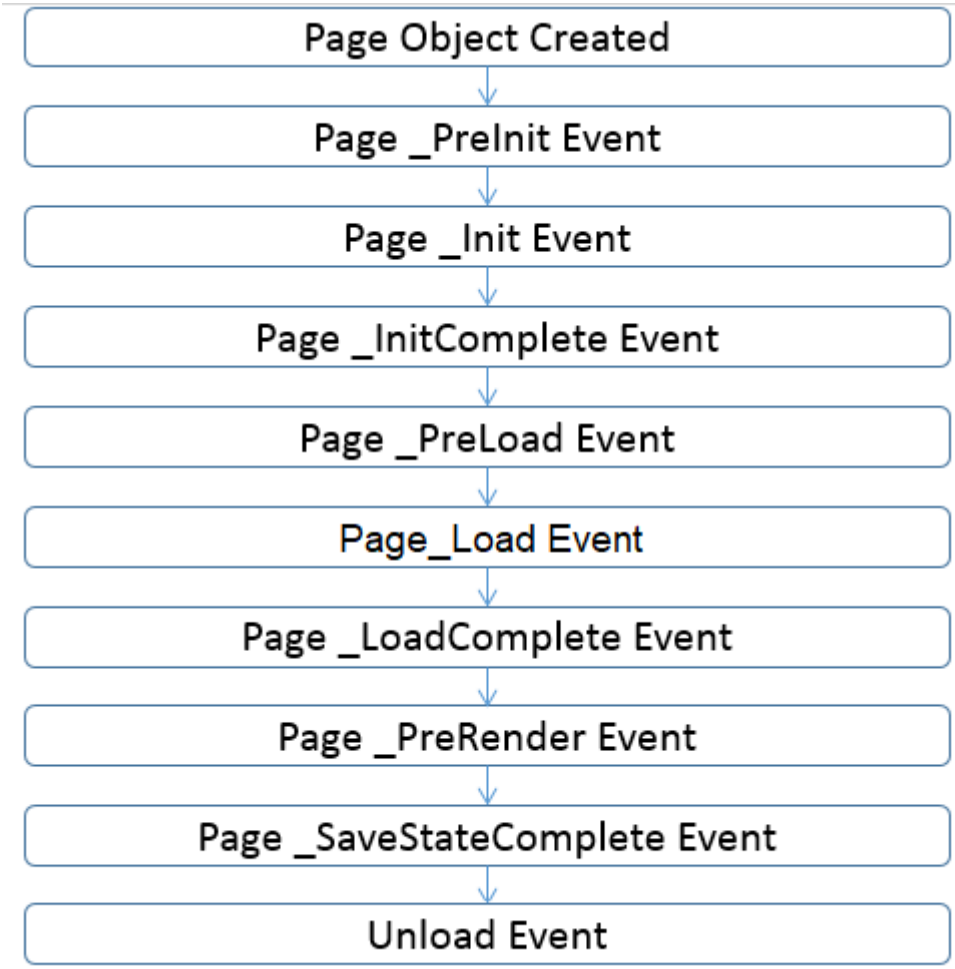
يُمكن طلب الإرجاع (أي ارسال قيم عناصر النموذج إلى المخدم) postback من قبل المستخدم بعدة طرق. فبالطبع، يحدث الإرجاع عند نقر المستخدم على زر الإرسال submit. كما أنه يحدث عند النقر على أي زر أمر. كذلك يُمكن للمبرمج تحديد طلب الإرجاع لمختلف عناصر التحكم كصناديق الخيار مثلاً وذلك بإسناد القيمة true إلى الخاصية AutoPostBack، وعندها يتمّ الإرجاع عند تغيير قيمة العنصر.

أحداث الصفحة Page-Level Events

يُنشئ الصف عدة أحداث أهمها الأحداث التالية:

- الحدث Init والذي يُرفع مباشرةً بعد إنشاء منتسخ صف الوثيقة.
- الحدث Load والذي يُرفع بعد إسناد حالة المنتسخ من بيانات النموذج و ViewState.
- الحدث PreRender والذي يُرفع قبل تنفيذ المنتسخ لبناء الوثيقة الجواب للزبون.
- الحدث Unload والذي يُرفع قبل مسح المنتسخ.

ويوضح الشكل التالي بقية الأحداث وتسلسل تنفيذها:



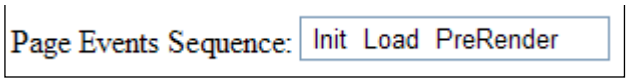
يكفي لكتابة معالجات هذه الأحداث التصريح عن الإجرائيات باستخدام أسمائها المعرفة مسبقاً والمسجلة ضمنياً عند إنشاء صف الوثيقة.

يُبين المثال التالي التصريح عن الأحداث الأربع السابقة وتسلسل وقوعها:

```
<%@ Page Language="c#" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        T.Value = T.Value + " Load ";
    }
    protected void Page_Unload(object sender, EventArgs e)
    {
        T.Value = T.Value + " Unload ";
    }
    protected void Page_PreRender(object sender, EventArgs e)
    {
        T.Value = T.Value + " PreRender ";
    }
```

```
    protected void Page_Init(object sender, EventArgs e)
    {
        T.Value = T.Value + " Init ";
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Page Events </title>
</head>
<body>
    <form runat="server" id="MyForm">
        Page Events Sequence:
           <input id="T" runat="server" />
    </form>
</body>
</html>
```

حيث يكون الخرج:



أحداث عناصر التحكم Control Events

يوجد طريقتين لإنشاء وتسجيل معالج حدث لعنصر HTML. نُسند في الطريقة الأولى اسم إجرائية معالج الحدث إلى الواصفة OnServerClick أو الواصفة OnServerChange. يكون لإجرائية معالج الحدث البروتوكول التالي:

- تُعيد void.
- يوجد لها معاملي دخل الأول من النوع object، والثاني من النوع System.EventArgs.

(وهو البروتوكول للتفويض EventHandler (delegate) والذي يُعرّف المنهج القياسي لمعالجة الأحداث في لغة التنفيذ الموحدة CLR).

يُبين المثال التالي الطريقة الأولى لتسجيل معالج الحدث.

```
protected void TextHandler (object sender, EventArgs e)
{
. . .
}
<input type="text" id="Name" OnServerChange="TextHandler" Runat="server"  />
```

(و هي الطريقة الافتراضية المستخدمة)

تعتمد الطريقة الثانية على كتابة وتسجيل الحدث باستخدام المنهج القياسي في CLR، والذي يستخدم التفويض لتسجيل معالج الحدث. حيث نتبع الخطوات الثلاث التالية:

- كتابة معالج الحدث والذي هو عبارة عن وظيفة تُعيد void، ولها معاملي دخل من النوع object و EventArgs.
- إنشاء منتسخ جديد من النمط EventHandler باستخدام new وتمرير اسم الوظيفة السابقة للباني.
- تسجيل منتسخ التفويض السابق إلى خاصية حدث عنصر التحكم بإضافته إلى المنتسخات المسجلة سابقاً.

يتمّ عادة دمج الخطوتين السابقتين في تعليمة واحدة توضع في معالج الحدث Page_Init كما يُبين المثال التالي:

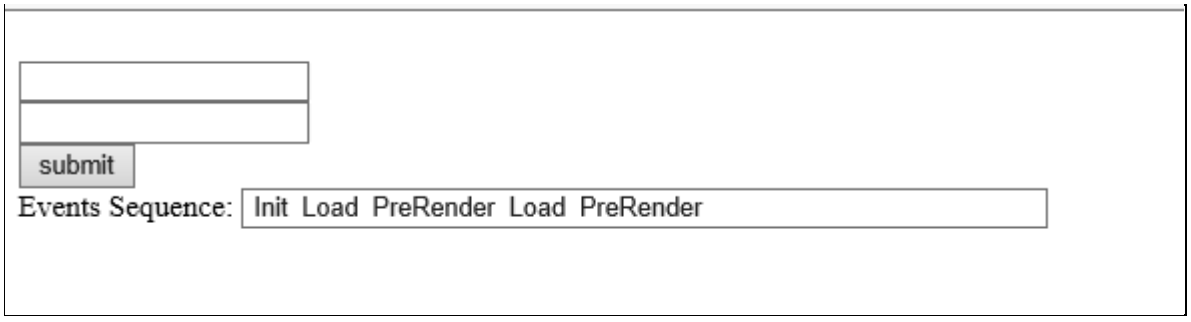
```
protected void TextboxHandler(object sender, EventArgs e)
{
. . .
}

protected void Page_Init(object sender, EventArgs e)
{
Name.ServerChange += New EventHandler(TextBoxHandler);

}
. . .
<input type="text" id="Name" runat="server" />
```

يُبين المثال التالي استخدام الطريقتين السابقتين وتسلسل وقوع الأحداث:

```
<%@ Page Language="c#" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        T.Value = T.Value + " Load ";
    }
    protected void Page_Unload(object sender, EventArgs e)
    {
        T.Value = T.Value + " Unload ";
```

```
    }
    protected void Page_PreRender(object sender, EventArgs e)
    {
        T.Value = T.Value + " PreRender ";
    }
    protected void Page_Init(object sender, EventArgs e)
    {
        T.Value = T.Value + " Init ";
        T2.ServerChange += new EventHandler(T2Handler);
    }

    protected void T1Handler(object sender, EventArgs e)
    {
        T.Value = T.Value + " T1 Handler ";
        T1.Style.Add(HtmlTextWriterStyle.BackgroundColor,"red");

    }
    protected void T2Handler(object sender, EventArgs e)
    {
        T.Value = T.Value + " T2 Handler ";
        T2.Style.Add(HtmlTextWriterStyle.BackgroundColor, "blue");

    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Control Events </title>
</head>
<body>
    <form runat="server" id="MyForm">
        <br />
        <input type="text" id="T1" onserverchange="T1Handler" runat="server" />
        <br />

        <input type="text" id="T2" runat="server" />
        <br />
        <input type="submit" value="submit" />
        <br />
        Events Sequence:
           <input id="T" runat="server" style="width: 424px" />
    </form>
</body>
</html>
```

حيث يظهر بعد النقر على الزر submit:



ملف أحداث التطبيق Global.asax

يتوضع هذا الملف في المجلد الاساسي (root level) ويستخدم لمعالجة عدة أحداث للتطبيق أهمها الأحداث التالية:

Application_Start(): يتم تنفيذ هذه الإجرائية عند طلب وإنشاء التطبيق أول مرة.

Application_End(): يتم تنفيذ هذه الإجرائية عند انتهاء أو ايقاف التطبيق.

Session_Start(): يتم تنفيذ هذه الإجرائية عند دخول مستخدم جديد.

Session_End(): يتم تنفيذ هذه الإجرائية عند انتهاء جلسة المستخدم.

Application_BeginRequest(): يتم تنفيذ هذه الإجرائية عند كل طلب.

Application_EndRequest(): يتم تنفيذ هذه الإجرائية عند انتهاء الطلب.

مثال: عند إنشاء ملف أحداث التطبيق في Visual Studio .Net تظهر معالجات الأحداث التالية:

```
<%@ Application Language="C#" %>

<script runat="server">

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup

    }

    void Application_End(object sender, EventArgs e)
    {
        //  Code that runs on application shutdown

    }

    void Application_Error(object sender, EventArgs e)
    {
        // Code that runs when an unhandled error occurs

    }

    void Session_Start(object sender, EventArgs e)
    {
        // Code that runs when a new session is started

    }

    void Session_End(object sender, EventArgs e)
```

```
    {
        // Code that runs when a session ends.
        // Note: The Session_End event is raised only when the sessionstate mode
        // is set to InProc in the Web.config file. If session mode is set to StateServer
        // or SQLServer, the event is not raised.

    }
    void Application_BeginRequest(object sender, EventArgs e)
    {

    }
    void      Application_EndRequest(object sender, EventArgs e)
    {

    }


</script>
```