Beginning ASP.NET 2.0 in C# from Novice to Professional
Apress 2006
Matthew MacDonald
فصل ۲۵
صفحه‌ی ۹۵۲ الی ۹۷۳

ترجمه: سید منصور عمرانی
mansoor.omrani@yahoo.com
تابستان ۱۳۹۱

محدودیت‌های کنترل‌های کاربر

در مواردی که می‌خواهید چندین کنترل وب را (به همراه سایر المان‌های HTML) در قالب یک واحد مجزا ترکیب کرده و منطق سطح بالاتری نیز برای مجموعه‌ی ایجاد شده تعریف کنید، کنترل‌های کاربر انعطاف‌پذیری بسیار زیادی فراهم می‌کنند. اما اگر کنترل شما بسیار ساده بوده و مثلاً فقط بخواهید قابلیت‌های یکی از کنترل‌های خود ASP.NET را دستکاری کرده یا توسعه بدهید، کنترل‌های کاربر چندان مفید نیستند.

به عنوان مثال تصور کنید می‌خواهید یک کنترل TextBox درست کنید که چند قابلیت اضافی دیگر نسبت به کنترل TextBox معمولی ASP.NET داشته باشد. به طوری که با وجود یکسان بودن شکل ظاهری آن با کنترل TextBox، چند قابلیت جدید داشته باشد. به عنوان مثال فرض کنید بخواهید متد جدیدی به آن اضافه کنید که برخی عملیات را برای پردازش یا قالب‌بندی متن درون آن را به طور خودکار انجام بدهد.

برای ایجاد چنین کنترلی با استفاده از مدل کنترل‌های کاربر باید یک کنترل کاربر جدید ایجاد کنید و درون آن یک کنترل TextBox قرار بدهید. سپس متدهایی را که نیاز دارید به آن اضافه کنید. به این مدل طراحی مدل *شمول* یا Containment گفته می‌شود، زیرا کنترل کاربر شامل یک کنترل جعبه متن است.

اما مشکل این است که این مدل اغلب به جای آن که مشکلاتی را بر طرف کند مشکلات زیادی را به بار می‌آورد. به عنوان مثال اگرچه می‌توانید به سادگی متدهای مورد نیاز خود را برای کنترل جدید تعریف کنید، اما عملاً کنترل TextBox را درون یک جعبه مخفی کرده‌اید. لذا کسی که می‌خواهد از این کنترل در یک صفحه‌ی وب استفاده کند، فقط به قابلیت‌هایی که شما برای آن تعریف کرده‌اید دسترسی خواهد داشت و دیگر به قابلیت‌های ذاتی کنترلی که درون کنترل خود قرار داده‌اید دسترسی نخواهد داشت. به عنوان مثال دیگر نمی‌توانید خصوصیت‌هایی مانند Font، ForeColor، Width و سایر خصوصیت‌های ظاهری TextBox را تغییر بدهند. لذا اگرچه تلاش کرده‌اید یک کنترل پیشرفته‌تر بسازید، اما تمام قابلیت‌های قبلی آن را به طور کامل از دسترس برنامه‌نویس خارج کرده‌اید!

البته برای حل مشکل می‌توانید خصوصیت‌هایی معادل خصوصیت‌های TextBox برای کنترل خود تعریف کنید. اما طبیعی است که این کار بسیار خسته کننده خواهد بود. زیرا کنترل‌های وب پیچیده هستند و خصوصیت‌های متعددی دارند. کپی سازی تمام خصوصیت‌ها و متدهای آنها کار بسیار زیادی خواهد برد.

در این قسمت رهیافت بهتر و منطقی‌تری برای توسعه یا بهبود دادن کنترل‌های ASP.NET معرفی کرده و به جای به‌کارگیری مدل شمول، از مدل وراثت استفاده می‌کنیم.

کنترل‌های شخصی اشتقاقی[1]

به لطف مدل شیءگرای چهارچوب کاری NET. می‌توانید با استفاده از ارث‌بری کنترل‌های پیشرفته‌ای را به روشی بسیار ساده ایجاد کنید. تمام کاری که باید انجام بدهید این است که یکی از کنترل‌های کتابخانه‌ی کلاس‌های NET. را انتخاب کنید، کلاسی از آن مشتق کنید و قابلیت‌های جدید را به کلاس مشتق شده‌ی جدید اضافه کنید. به چنین کنترلی یک کنترل اشتقاقی گفته می‌شود.

برای ایجاد یک کنترل اشتقاقی در NET. از دو روش می‌توانید استفاده کنید:

- می‌توانید کنترل شخصی را داخل خود برنامه‌ی وب ایجاد کنید. در این حالت باید فایل کلاس کنترل شخصی را درون دایرکتوری App_Code قرار بدهید تا به طور خودکار کامپایل شده و در کل برنامه‌ی وب قابل دسترس باشد.
- می‌توانید کنترل شخصی را در قالب یک پروژه‌ی جدا ایجاد کنید و آن را به شکل DLL کامپایل کنید. بدین ترتیب می‌توانید کنترل خود را از برنامه‌ی استفاده کننده جدا کرده و از هرگونه دستکاری احتمالی آن جلوگیری کنید. به علاوه می‌توانید از آن در چندین برنامه‌ی وب مختلف استفاده کنید.

در این قسمت ابتدا روش ساده‌تر (ایجاد کنترل شخصی درون پروژه‌ی وب فعلی) را نشان می‌دهیم. پس از آن به سمت ایجاد یک کتابخانه‌ی کنترل شخصی کامل و مجزا حرکت خواهیم کرد.

ایجاد یک کنترل شخصی اشتقاقی

فرض کنید بخواهیم کنترل TextBox را توسعه بدهیم و قابلیت‌های جدیدی به آن اضافه کنیم به طوری که به طور اختصاصی برای دریافت اسامی افراد استفاده شود. به لطف وراثت خواهید دید انجام این کار از آن چه تصورش را می‌کنید بسیار ساده‌تر است.

در کُد زیر کلاس جعبه متنی به اسم NameTextBox از کنترل TextBox (که کلاس آن در فضای نام System.Web.UI.WebControls قرار دارد) مشتق کرده‌ایم و دو متد به نام ()GetFirstName و ()GetLastName برای آن تعریف کرده‌ایم. این متدها متن وارد شده را بررسی کرده و طبق یکی از دو قرارداد برای وارد کردن نام و نام خانوادگی (جدا شده توسط کاراکتر فاصله مانند John White یا جدا شده توسط کاما مانند John,

[1] Derived Custom Controls

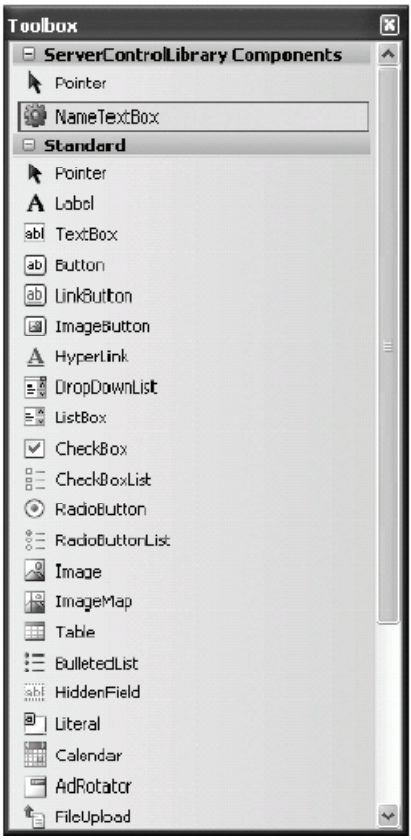
White)، نام و نام خانوادگی را از میان متن وارد شده در TextBox استخراج می‌کنند. بدین ترتیب کاربر می‌تواند نـام و نام خانوادگی خود را به هر ترتیبی که تمایل دارد وارد کند و برنامه نیز نیازی ندارد خـودش مـتن وارد شـده توسـط کاربر را بررسی کند. به جای آن کنترل NameTextBox خودش این کار را به طور خودکار انجام می‌دهد.

```
namespace apress
{
  public class NameTextBox: TextBox
  {
    private firstName;
    private lastName;

    public string GetFirstName()
    {
      UpdateNames();
      return firstName;
    }
    public string GetLastName()
    {
      UpdateNames();
      return lastName;
    }
    private void UpdateNames()
    {
      int commaPos = this.Text.IndexOf(',');
      int spacePos = this.Text.IndexOf(' ');

      string[] nameArray;
      if (commaPos != -1)
      {
        nameArray = this.Text.Split(',');
        firstName = nameArray[1];
        lastName = nameArray[0];
      }
      else if (spacePos != -1)
      {
        nameArray = this.Text.Split(' ');
        firstName = nameArray[0];
        lastName = nameArray[1];
      }
      else
      {
        // The text has no comma or space.
        // It cannot be converted to a name.
        throw new InvalidOperationException();
      }
    }
  }
}
```

این کلاس را در فولدر App_Code قرار بدهید. از آنجایی که کلاس NameTextBox از کـلاس TextBox مشـتق شده تمام اعضا (یا خصوصیت‌ها و متدهای) آن را به ارث می‌برد و لذا تمام این اعضا در دسترس برنامه‌نویسـی کـه از این کلاس استفاده می‌کند خواهد بود و از طریق کُد یا از طریق تگ آن در صفحه‌ی aspx. قابل تنظیم خواهد بود.

نحوه‌ی استفاده از کنترل‌های اشتقاقی در Visual Studio

نحوه‌ی استفاده از کنترل‌های اشتقاقی با کنترل‌های کاربر فرق دارد و نمی‌توانید این کنترل‌ها را به همان شکل استفاده کنید. به عبارت دیگر کشیدن و انداختن فایل این کنترل‌ها از پنجره‌ی Solution Explorer بر روی صفحه تاثیری ندارد. به جای این کار ابتدا باید پروژه‌ی وب خود را از طریق منوی Build > Build Website کامپایل کنید. وقتی این کار را انجام می‌دهید Visual Studio اسمبلی پروژه‌ی وب شما را به دنبال هر گونه کلاس کنترل وبی که ممکن است در آن وجود داشته باشد می‌گردد (به بیان ساده کلاس‌هایی که به طور مستقیم یا غیر مستقیم از کلاس System.Web.UI.Control مشتق شده باشند). در صورتی که چنین کنترلی پیدا کند همان طور که در شکل ۷–۲۵ نشان داده شده آن را به طور خودکار به جعبه ابزار محیط طراحی اضافه می‌کند.



شکل ۷–۲۵. یک کنترل شخصی در جعبه ابزار Visual Studio

اکنون از طریق جعبه ابزار می‌توانید کنترل NameTextBox را در هر صفحه‌ی وبی که می‌خواهید استفاده کنید و آن را از جعبه ابزار روی صفحه بیندازید. وقتی این کار را انجام می‌دهید Visual Studio به طور خودکار یک دایرکتیو Register شبیه زیر به ابتدای صفحه اضافه می‌کند:

```
<%@ Register TagPrefix="sc1" Namespace="apress" %>
```

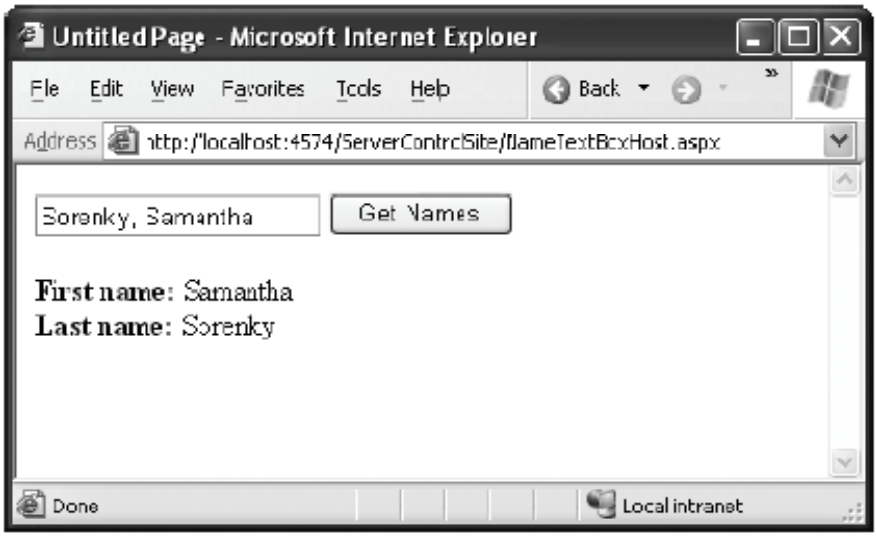
تگ کنترل NameTextBox نیز چنین خواهد بود:

```
<sc1:NameTextBox id="NameTextBox1" runat="server" />
```

در حقیقت توسط دایرکتیو Register تمام کنترل‌هایی که در فضای نام apress تعریف شده‌اند در صفحه‌ی فعلی رجیستر شده و بر اساس نام کلاس آنها و از طریق تگ پیشوندی ذکر شده در خصوصیت TagPrefix می‌توان آنها را در صفحه استفاده کرد.

در شکل ۸–۲۵ صفحه‌ی ساده‌ای نشان داده شده که در آن کنترل NameTextBox آزمایش شده است. وقتی کاربر دگمه‌ی Get Names را کلیک می‌کند، برنامه نام و نام خانوادگی را از متن وارد شده در کنترل NameTextBox از طریق متدهای ()GetFirstName و ()GetLastNameبه دست آورده و در یک برچسب نمایش می‌دهد:

```
protected void cmdGetNames_Click(Object sender, EventArgs e)
{
   lblNames.Text = "<b>First name:</b>";
   lblNames.Text += NameTextBox1.GetFirstName();
   lblNames.Text = "<br/><b>Last name:</b>";
   lblNames.Text += NameTextBox1.GetLastName();
}
```



شکل ۸–۲۵. یک کنترل اشتقاقی که از کنترل جعبه متن ارث برده است

می‌توانید خصوصیت‌های کنترل NameTextBox را از طریق کُد یا از طریق تگ کنترلی آن مقداردهی کنید. این خصوصیت‌ها می‌تواند هر یک از خصوصیت‌های اصلی کنترل TextBox یا خصوصیت‌هایی باشد که خودتان برای کنترل جدید تعریف کرده‌اید:

```
<apress:NameTextBox id="NameTextBox1" runat="server" BackColor="LightYellow"
        Font="Verdana" Text="Enter your name" />
```

تکنیک مشتق کردن کنترل‌های شخصی از کنترل‌های ASP.NET «*زیر کلاس‌سازی*»[2] نام دارد و توسط آن می‌توانید به سادگی قابلیتی را که لازم دارید به کنترل قبلی اضافه کنید بدون آن که قابلیت‌های پایه‌ای و اصلی آن را از دست بدهید. برای کنترل‌های زیر کلاس‌سازی شده می‌توان متدها، خصوصیت‌ها و رویدادهای جدیدی تعریف کرد و یا می‌توان متدها، خصوصیت‌ها یا رویدادهای کلاس پدر را سربارگذاری کرد. به عنوان مثال می‌توانید متدی مانند ReverseText() یا EncryptText() برای کنترل NameTextBox تعریف کنید که محتوای متن وارد شده را بر عکس کرده یا رمز می‌کند.

نحوه‌ی تعریف و استفاده از خصوصیت‌ها، متدها و رویدادهای شخصی برای یک زیر کلاس درست مانند کنترل‌های کاربر است.

ایجاد یک کتابخانه‌ی کنترل شخصی

بهترین کار برای سازمان‌دهی بهتر این است که *کنترل‌های شخصی*[3] را در یک پروژه‌ی Class Library جداگانه ایجاد کنید. زیرا پس از آن می‌توانید آنها را به شکل یک فایل DLL کامپایل کرده و با اضافه کردن ارجاع آنها به پروژه‌ها و سایت‌های وب مختلف از آنها در چندین پروژه‌ی مختلف استفاده کنید. همچنین می‌توانید آنها را به دیگران بدهید بدون این که سورس آنها را در اختیار دیگران قرار بدهید. این کار پشتیبانی زمان اجرای این کنترل‌ها را نیز بهبود می‌بخشد.

نکته: برای این که کار را کمی راحت‌تر کنید، کنترل‌های شخصی را به جای Class Library داخل یک پروژه‌ی Web Class Library ایجاد کنید. این پروژه‌ها شبیه پروژه‌های Class Library هستند، با این تفاوت که از قبل ارجاعاتی به اسمبلی‌های مورد نیاز مانند System.Web.Dll در آنها قرار داده شده است.

برای ایجاد یک Web Class Library حاوی کنترل‌های شخصی مراحل زیر را دنبال کنید:

۱. یک سایت جدید (Add > New Website) به صورت عادی ایجاد کنید.
۲. گزینه‌ی File > Add > New Project را برای اضافه کردن پروژه‌ی جدیدی به Solution جاری کلیک کنید.
۳. در پنجره‌ی Add New Project از میان الگوهای مختلف نوع پروژه‌ی جدید را Web Control Library انتخاب کنید.
۴. نام و محلی برای اسمبلی آن مشخص کنید. توجه کنید این محل باید از محل سایت شما جدا باشد. از آنجایی که پروژه‌ی Web Class Library و سایتی که ایجاد کرده‌اید هر دو در یک Solution قرار دارند وقتی پروژه‌ی Web Class Library را کامپایل می‌کنید فایل DLL آن به طور خودکار به دایرکتوری Bin

[2] Subclassing
[3] Custom Controls

سایت شما کپی می‌شود. لذا نیازی نیست خودتان فایل DLL را بـه طـور جداگانـه و بـه صـورت دسـتی در دایرکتوری Bin سایت خود کپی کنید.

۵. دگمه‌ی Ok را کلیک کنید تا پروژه‌ی Web Class Library ایجـاد شـود. اکنـون در پنجـره‌ی Solution Explorer دو پروژه خواهید دید: سایت خودتـان و پـروژه‌ی کتابخانـه‌ی کنتـرل شخصـی کـه اسـم آن را ServerControlLibrary گذاشته‌ایم (شکل ۹–۲۵).

۶. برای ساده‌سازی روند باز کردن پروژه‌ی Web Class Library و سایتی که کنترل‌هـای شخصـی را در آن استفاده می‌کنید، Solution جاری را به شکل یک فایـل sln. ذخیـره کنیـد. بـرای ایـن کـار از پنجـره‌ی Solution Explorer نام Solution را انتخاب کرده (اولین گزینه در بالای درختواره)، سپس گزینه‌ی File > Save [SolutionName] را کلیک کنید. می‌توانید محل دقیق ذخیره‌سازی ایـن فایـل را نیـز مشـخص کنید تا بعداً راحت‌تر بتوانید آن را پیدا کنید.

شکل ۹–۲۵. یک Solution با یک سایت وب و یک کتابخانه‌ی کنترل شخصی

نکته: به خاطر داشته باشید فایل Solution (sln.) فایل کوچکی است که به طور بسیار ساده فقط حاوی لینک‌هـایی به پروژه‌هایی است که در آن تعریف شده است. لذا توجه داشته باشید اگر فولدر پـروژه‌هـای یـک Solution را تغییـر بدهید یا آنها را به دایرکتوری دیگری منتقل کنید، Solution دیگر کار نخواهد کرد و باید یک Solution جدید بـرای آنها ایجاد کنید.

برای قرار دادن یک نمونه از کنترل شخصی بر روی یک صفحه‌ی وب، باید از همان روشی که کمـی پـیش توضـیح داده شد استفاده کنید. ابتدا Solution را کامپایل کنید (گزینه‌ی Build > Build Solution). سپس کنترل شخصی را که اکنون به جعبه ابزار اضافه شده از جعبه ابزار برداشته و در صفحه‌ی وب مورد نظر قرار بدهید. اولین باری کـه ایـن کار را انجام می‌دهید Visual Studio یک نسخه از فایل DLL کتابخانه‌ی کنترل شخصی را به دایرکتوری Bin سایت شما اضافه می‌کند. از آن لحظه به بعد، هر بار که سایت خود را کامپایل می‌کنیـد، Visual Studio بـه طـور خودکـار

بررسی می‌کند که آیا نسخه‌ی کنترل شخصی تغییر کرده است یا خیر و در صورتی که تغییر کرده باشد، فایل DLL دایرکتوری Bin سایت را با نسخه‌ی جدید موجود در پروژه‌ی Web Class Library به روز می‌کند.

در این حالت دایرکتیو Register ابتدای صفحه‌ی وبی که کنترل شخصی را در آن استفاده می‌کنید تفاوت کوچکی با دایرکتیو Register حالت قبل دارد. دایرکتیو Register در این حالت چیزی شبیه زیر خواهد بود:

```
<%@ Register TagPrefix="apress" Namespace="ServerControlLibrary" Assembly="CustomCtrls" %>
```

در این حالت نام اسمبلی کامپایل شده‌ی کنترل‌های شخصی نیز به صفحه شناسانده می‌شود. هنگامی که یک اسمبلی کنترل شخصی را بدین صورت برای یک صفحه ثبت می‌کنید، به تمام کنترل‌هایی که درون آن اسمبلی تعریف کرده‌اید دسترسی خواهید داشت. از این لحظه به بعد می‌توانید هر یک از کنترل‌های شخصی اسمبلی را با استفاده از تگ پیشوندی مشخص شده (TagPrefix)، به اضافه‌ی یک علامت کالن و سپس نام کلاس کنترل شخصی در صفحه‌ی وب مورد نظر استفاده کنید.

```
<apress:NameTextBox id="NameTextBox1" runat="server" />
```

کنترل‌های شخصی و مقادیر پیش فرض

یک دلیل رایج برای تعریف زیر کلاس از کلاس یک کنترل استاندارد ASP.NET، اضافه کردن مقادیر پیش فرض برای خصوصیت‌های آن است. به عنوان مثال می‌توانید یک کنترل تقویم شخصی از کنترل Calendar مشتق کنید و سازنده‌ای برای آن تعریف کنید که مقدار برخی از خصوصیت‌های آن را به طور خودکار مقداردهی می‌کند.

```
public class FormattedCalendar : Calendar
{
  public FormattedCalendar()
  {
    // Configure the appearance of the calendar table.
    this.CellPadding = 8;
    this.CellSpacing = 8;
    this.BackColor = Color.LightYellow;
    this.BorderStyle = BorderStyle.Groove;
    this.BorderWidth = Unit.Pixel(2);
    this.ShowGridLines = true;

    // Configure the font.
    this.Font.Name = "Verdana";
    this.Font.Size = FontUnit.XXSmall;

    // Set sothis special calendar settings.
    this.FirstDayOfWeek = FirstDayOfWeek.Monday;
    this.PrevMonthText = "<--";
    this.NextMonthText = "-->";

    // Select the current date by default.
    this.SelectedDate = DateTime.Today;
  }
}
```

حتی می‌توانید یک اداره‌گر شخصی برای برخی از رویدادهای این کلاس تعریف کرده و در سازنده‌ی کلاس، این اداره‌گرها را به رویدادهای آن وصل کنید. مثلاً برای رویداد DayRender اداره‌گری بنویسید که نحوه‌ی نمایش تاریخ‌ها را شخصی‌سازی می‌کند. بدین ترتیب کلاس تقویم شما خودش به طور خودکار تمام قالب‌بندی‌ها و پیکربندی را اداره خواهد کرد و به هیچ کُد دیگری در صفحه‌ی وبی که از این کنترل استفاده می‌کند نیازی نخواهد بود!

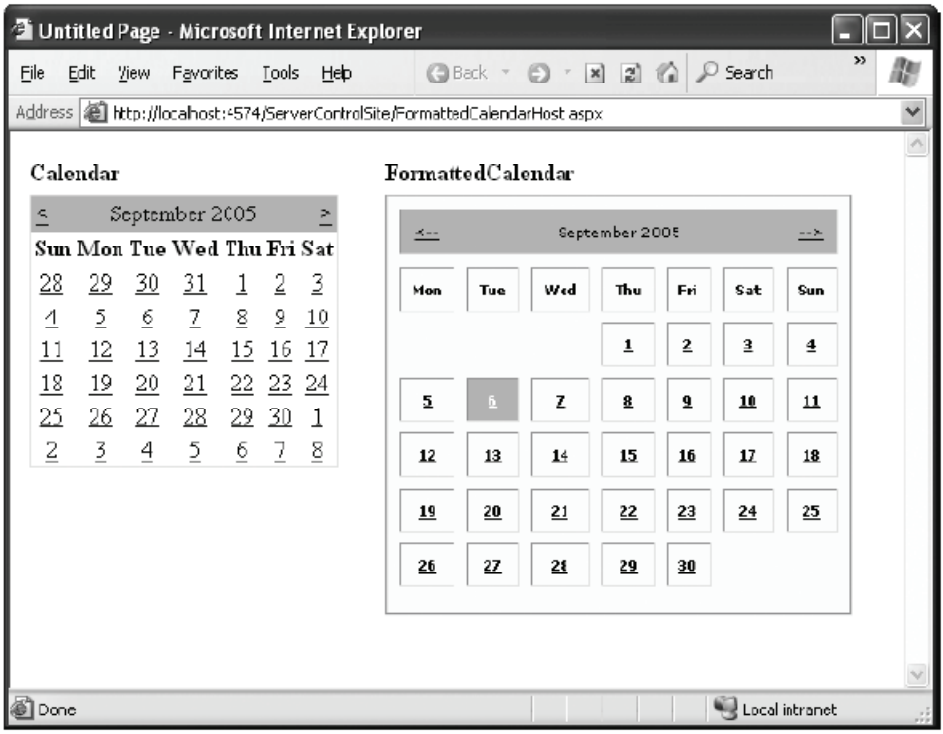
```
public class FormattedCalendar : Calendar
{
  public FormattedCalendar()
  {
    this.DayRender += new EventHandler(FormattedCalendar1_DayRender);
  }
  protected void FormattedCalendar1_DayRender(object sender, DayRenderEventArgs e)
  {
    if (e.Day.IsOtherMonth)
    {
      e.Day.IsSelectable = false;
      e.Cell.Text = "";
    }
    else
    {
      e.Cell.Font.Bold = true;
    }
  }
}
```

اما یک راه بهتر این است که متد Calendar.OnDayRender() را که به طور خودکار پیش از آتش شدن رویداد DayRender فراخوانی می‌شود ابطال یا override کنید. بدین ترتیب نیازی هم نیست اداره‌گری بنویسید و به طور دستی آن را به رویداد DayRender وصل کنید (زیرا متد OnDayRender() به طور خودکار فراخوانی می‌شود). اما در این حالت حتماً به خاطر داشته باشید که با استفاده از کلمه‌ی کلیدی base، متد کلاس پدر (Calendar) را نیز فراخوانی کنید. این کار تضمین می‌کند هر عملی که قرار بوده در خود کنترل پدر انجام شود (مانند آتش کردن رویداد) اجرا شود:

```
protected override void OnDayRender(TableCell cell, CalendarDay day)
{
  // Call the base Calendar.OnDayRender method.
  base.OnDayRender(cell, day);
  if (day.IsOtherMonth)
  {
    day.IsSelectable = false;
    cell.Text = "";
  }
  else
  {
    cell.Font.Bold = true;
  }
}
```

در شکل ۱۰–۲۵ دو کنترل تقویم در کنار یکدیگر نشان داده شده و با یکدیگر مقایسه شده‌اند: یکی کنترل عادی تقویم و دیگری کنترل FormattedCalendar.

نکته: با این که می‌توانید مقادیر پیش فرضی را در یک کلاس شخصی تعریف کنید، اما در عین حال کماکان می‌توانید مقدار خصوصیت‌های کلاس را از طریق کُد صفحه نیز تغییر بدهید و مثلاً در فایل aspx. مقادیری را برای برخی از خصوصیت‌های یک کنترل مشخص کنید. در این حالت ابتدا سازنده‌ی کلاس (موقع نمونه‌سازی از کلاس کنترل) اجرا می‌شود و پس از آن تنظیماتی که در تگ کنترلی در صفحه‌ی aspx. تعریف کرده‌اید اعمال می‌شود. همچنین علاوه بر این که می‌توانید داخل کلاس کنترل یک متد اداره‌گر رویداد شخصی تعریف کنید، اما همچنان می‌توانید در کُد صفحه‌ی وبی که از کنترل استفاده می‌کند اداره‌گری را برای رویداد کنترل تعریف کنید. در این حالت ابتدا اداره‌گر داخلی کلاس و سپس اداره‌گری که در صفحه‌ی وب تعریف کرده‌اید اجرا می‌شود.



شکل ۱۰-۲۵. یک کنترل تقویم زیر کلاس‌سازی شده

تغییر دادن نحوه‌ی پردازش و تولید خروجی یک کنترل

در کنترل‌های اشتقاقی با پیروی از چند راهبرد کوچک، حتی می‌توانید نحوه‌ی تولید خروجی HTML را نیز تغییر بدهید. این راهبردها چنین هستند:

- یکی از متدهایی را که برای پردازش و تولید خروجی به کار می‌رود ابطال (override) کنید – مانند ()Render، ()RenderContents، ()RenderBeginTag و نظایر آن. توجه کنید برای ابطال موفقیت‌آمیز این متدها، باید همان سطح دسترسی متد اصلی را برای متد ابطال شده مشخص کنید (مانند public یا

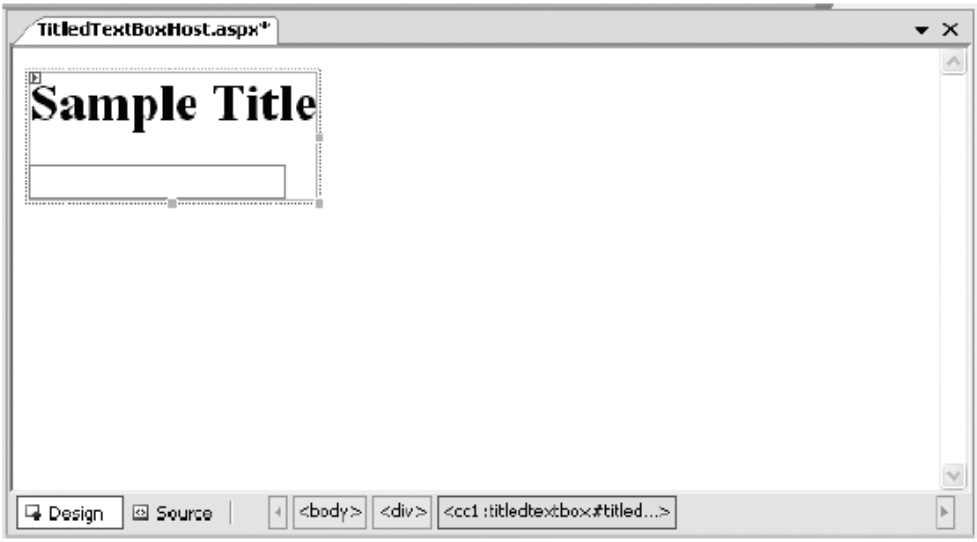
protected). در Visual Studio اگر سطح دسترسی متد ابطال شده را به چیز دیگری متفاوت با سطح دسترسی متد اصلی تعریف می‌کنید، Visual Studio به شما اخطار می‌دهد و بدین ترتیب می‌توانید متوجه شوید که آیا سطح دسترسی را درست تعریف کرده‌اید یا خیر. در صورتی که از محیط برنامه‌نویسی دیگری استفاده می‌کنید، برای کسب اطلاع از سطح دسترسی متدهای پردازش و تولید خروجی، به مرجع راهنمای MSDN مراجعه کنید.

- با استفاده از کلمه‌ی کلیدی base متد اصلی کلاس پدر را فراخوانی کنید. این کار بدین دلیل لازم است که متد اصلی ممکن است پاک‌سازی‌هایی انجام بدهد که ممکن است لازم باشد و شما از آن خبر نداشته باشیدو یا ممکن است رویدادی را آتش کند که قرار است صفحه‌ی وب آن را اداره کند.
- کُد مورد نظر خود را در متد ابطالی که نوشته‌اید بنویسید و هر گونه کُد HTML که مایل هستید به خروجی کنترل اضافه کنید. معمولاً بهتر است قابلیت جدیدی را به متد ابطال شده‌ی اصلی و خروجی‌ای که تولید می‌کند اضافه کنید نه این که متد خودتان را به طور کامل جایگزین آن کنید. متدهای پردازش و تولید خروجی معمولاً پارامتری از جنس HtmlWriter دارند که برای نوشتن در خروجی باید از آن استفاده کنید. این کلاس متد مفیدی به اسم ()Write دارد که برای ارسال محتوا به خروجی صفحه به کار می‌رود.

در کلاس زیر نمونه‌ای از یک کنترل TextBox شخصی نشان داده‌ایم و خصوصیتی به اسم Title برای آن تعریف کرده‌ایم. در این کلاس با ابطال متد ()Render مقدار خصوصیت Title را به شکل تگ <h1> به خروجی می‌فرستیم و پس از آن با فراخوانی ()base.Render، اجازه می‌دهیم کنترل اصلی، خروجی‌اش را تولید کند.

```
public class TitledTextBox: TextBox
{
  private string title;
  public string Title
  {
    get { return title; }
    set { title = value; }
  }
  protected override void Render(HtmlTextWriter writer)
  {
    // Add new Html.
    writer.Write("<h1>" + title + "</h1>");

    // Call the base method ( so that the text box is rendered).
    base.Render(writer);
  }
}
```

شکل ۱۱–۲۵ کنترل TitledTextBox را در محیط طراحی Visual Studio نشان می‌دهد.



شکل ۱۱-۲۵. یک جعبه متن زیر کلاس سازی شده به همراه یک عنوان

به خاطر داشته بشید که ASP.NET خروجی یک صفحه را با پردازش تک تک کنترل‌های درون آن تولید می‌کند و با فراخوانی متد ()Render از هر کنترل می‌خواهد که خودش خروجی‌اش را تولید کند. سپس تمام خروجی تولید شده را جمع‌آوری کرده و آن را به صورت یک صفحه‌ی کامل به سمت مرورگر ارسال می‌کند.

تنها محدودیت این روش این است که توانایی مدل طراحی بر اساس کنترل‌های سرویس‌دهنده را از دست خواهید داد. زیرا خودتان باید تگ خروجی را به صورت رشته ایجاد کرده و ویژگی‌هایی را به طور دستی به آن اضافه کنید (در حالی که در روش برنامه‌نویسی بر اساس کنترل‌های سرویس‌دهنده می‌توانید مثلاً از یک کنترل Label استفاده کنید و کنترل مزبور خودش خروجی‌اش را تولید می‌کند.

در واقع شما فقط خصوصیت‌هایش را تنظیم می‌کنید و دیگر درگیر تولید رشته‌ی HTML نمی‌شوید). این در حالی است که اگر یک کنترل کاربر یا یک *کنترل مرکب* (نوع به خصوصی از کنترل‌های شخصی که کمی جلوتر در همین فصل توضیح می‌دهیم) تعریف کنید، می‌توانید المان <h1> را توسط یک کنترل Label تولید کنید و هر تغییری را به خصوصیت‌های ظاهری آن اعمال کنید.

می‌توانید با استفاده از تکنیکی مشابه، خصوصیت‌هایی را به تگ HTML جعبه متن اضافه کنید. به عنوان مثال ممکن است بخواهید یک تکه کُد جاوا اسکریپت به خصوصیت OnBlur تگ HTML نهایی جعبه متن اضافه کنید تا وقتی تمرکز از این کنترل خارج می‌شود یک پیغام به کاربر نشان داده شود. هیچ یک از خصوصیت‌های کنترل معمولی TextBox چنین امکانی را فراهم نمی‌کند، اما می‌توانید این خصوصیت را به طور دستی توسط متد ()AddAttributesToRender به کنترل TextBox اضافه کنید. کُدی که نیاز دارید چنین است:

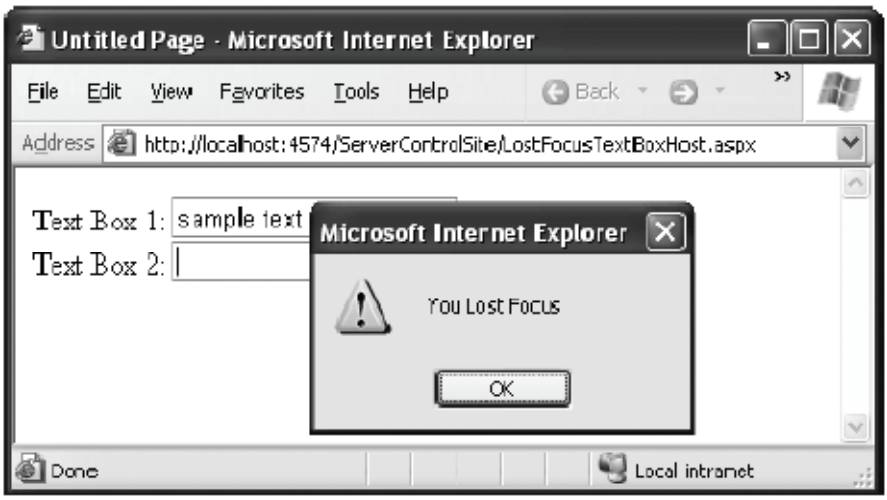
```
public class LostFocusTextBox: TextBox
{
   protected override void AddAttributesToRender(HtmlTextWriter writer)
   {
      base.AddAttributesToRender(writer);
      writer.AddAttribute("OnBlur", "javascript:alert('You lost focus')");
   }
}
```

خروجی HTML تولید شده برای این کنترل چیزی شبیه زیر خواهد بود:

```
<input type="text" id="LostFocusTextBox1" OnBlur=" javascript:alert('You lost focus')" />
```

شکل ۱۲–۲۵ خارج شدن تمرکز از چنین جعبه متنی را نشان می‌دهد. حتی می‌توانید به صورتی که در زیر نشـان داده شده خصوصیتی تعریف کنید که برنامه‌نویس توسط آن بتواند متن پیغامی را که در حالت از دست رفتن تمرکز نمـایش داده می‌شود مشخص کند.

```
public class LostFocusTextBox : TextBox
{
   private string alert;
   public string Alert
   {
      get
      { return alert; }
      set
      { alert = value; }
   }
   protected override void AddAttributesToRender(HtmlTextWriter writer)
   {
      base.AddAttributesToRender(writer);
      writer.AddAttribute("OnBlur",  "javascript:alert('" + alert + "')");
   }
}
```

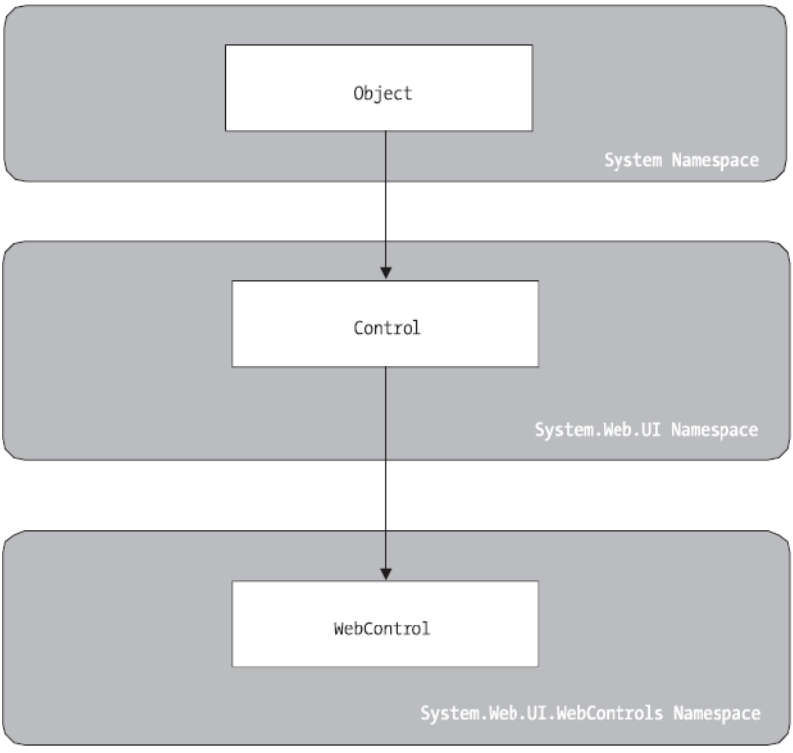


شکل ۱۲–۲۵. کنترل LostFocusTextBox

ایجاد یک کنترل کاملاً جدید (غیر اشتقاقی)

وقتی کمی خروجی کنترل TextBox را تغییر می‌دهید ممکن است این به ذهنتان برسد که آیا اساساً می‌توانید خودتان یک کنترل بسازید بدون آن که مجبور باشید یکی از کنترل‌های از پیش آماده‌ی ASP.NET را توسعه بدهید؟ پاسخ مثبت است و چنین کاری نیز بسیار ساده است. تمام کاری که باید انجام بدهید این است که کلاس کنترل خود را از کلاس System.Web.UI.WebControls.WebControl که کلاس پدر تمام کلاس‌های کنترل‌های ASP.NET است مشتق کنید. در شکل ۱۳–۲۵ درختواره‌ی وراثت و اشتقاق کنترل‌های وب در ASP.NET نشان داده شده است.

نکته: از نظر فنی می‌توانید کنترل خود را از کلاس System.Web.UI.Control نیز مشتق کنید اما چنین کنترل‌هایی قابلیت‌های کمتری خواهند داشت. اختلاف اصلی بین کلاس WebControl و Control این است که کلاس WebControl مجموعه‌ای پایه‌ای از خصوصیت‌هایی برای قالب‌بندی ظاهر کنترل نظیر Font، BackColor و ForeColor دارد. هنگام پردازش یک کنترل وب و تولید خروجی HTML آن، این خصوصیت‌ها به طور خودکار به خصوصیت‌های مناسبی در HTML تبدیل می‌شوند و همچنین به طور خودکار در View State نیز حفظ می‌شوند. اگر کلاس کنترل خود را از کلاس پایه‌ای‌تر Control مشتق کنید، باید خودتان خصوصیت‌های قالب‌بندی آن را تعریف کنید و آنها را به شکل ویژگی‌های HTML با استفاده از کلاس HtmlTextWriter در خروجی بنویسید. همچنین خودتان باید آنها را در View State قرار بدهید و در برگشت صفحه آنها را از View State بخوانید.



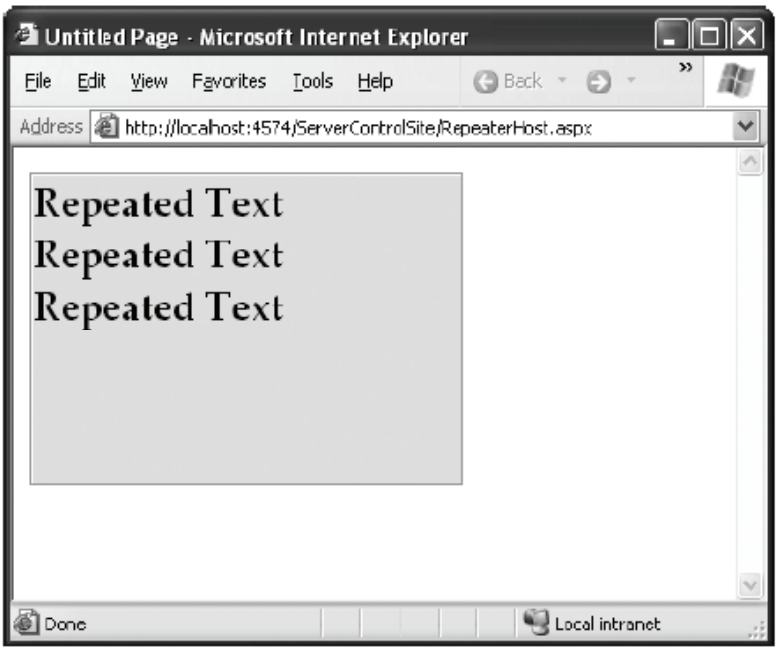
شکل ۱۳–۲۵. سلسله مراتب وراثت در کنترل‌های ASP.NET

البته ایجاد یک کنترل شخصی از ابتدا کمی بیشتر از تعریف خصوصیت‌های قالب‌بندی و سربارگذاری متد ()Render یا ()RenderContents (برای تولید خروجی HTML مورد نیاز) کار می‌برد. متد ()RenderContents پس از متد ()Render اجرا می‌شود که بدین معنی است که خصوصیت‌های قالب‌بندی قبلاً اعمال شده‌اند. در این قسمت مثالی از یک کنترل شخصی نشان می‌دهیم.

کُد کنترل جدید بسیار به کُد کنترل‌های جعبه متن مثال‌های قبل شباهت دارد. در این مثال یک نوع کنترل تکرارگر می‌سازیم که به طور ساده یک رشته‌ی تک خطی را چند بار تکرار می‌کند. تعداد دفعات نمایش یا تکرار رشته از طریق خصوصیتی به نام RepeatTimes مشخص می‌شود.

```
public class ConfigurableRepeater: WebControl
{
   private int repeatTimes = 3;
   private string text = "Text";

   public int RepeatTimes
   {
      get { return repeatTimes; }
      set { repeatTimes = value; }
   }
   public string Text
   {
      get { return text; }
      set { text = value; }
   }
   protected override void RenderContents(HtmlTextWriter writer)
   {
      base.RenderContents(writer);
      for (int i = 0; i < repeatTimes; i++)
      {
         writer.Write(text + "<br/>");
      }
   }
}
```

از آنجایی که کلاس این کنترل را به جای Control از WebControl مشتق کرده‌ایم و از آنجایی که خروجی را از طریق متد ()RenderContents تولید می‌کنیم، برنامه‌نویسی که از این کنترل استفاده می‌کند می‌تواند خصوصیت‌های قالب‌بندی مختلفی را برای این کنترل تنظیم کند. در شکل ۱۴–۲۵ یک کنترل ConfigurableRepeater که قالب‌بندی نیز شده نشان داده شده است. اگر می‌خواهید عنوان یا هر جزء دیگری برای این کنترل تعریف کنید که نمی‌خواهید قالب‌بندی تنظیم شده برای این کنترل به آن اعمال نشود تولید آن را در متد ()Render انجام بدهید.

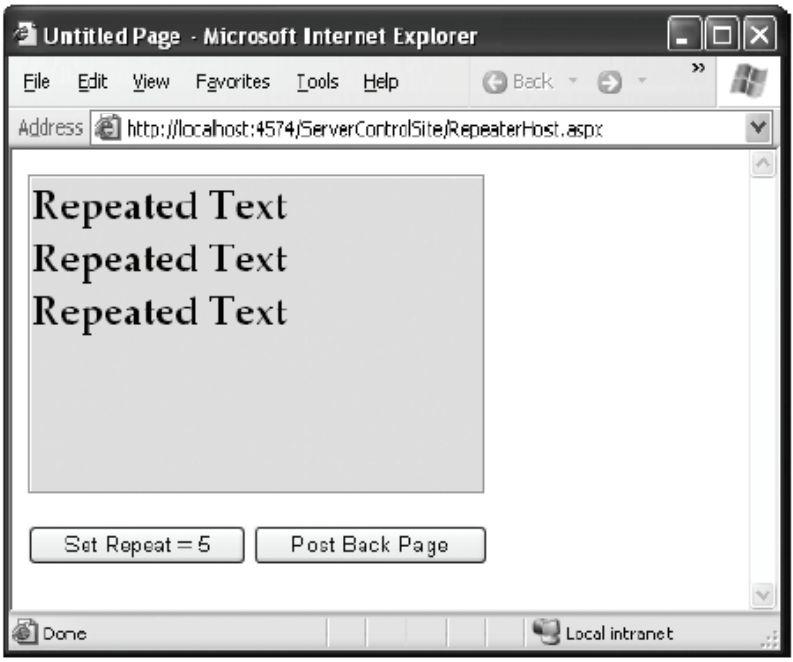


شکل ۱۴-۲۵. قالب‌بندی یک کنترل شخصی

حفظ وضعیت

کنترل تکرارگر فعلی ما خصوصیتی به نام EnableViewState دارد، اما به آن پایبند نیست، یعنی وقتی صفحه رفت و برگشت می‌کند محتوای خودش را حفظ نمی‌کند. می‌توانید این مساله را با ایجاد یک صفحه‌ی وب آزمایشی با دو دگمه امتحان کنید (شکل ۱۵-۲۵). یک دگمه تعداد دفعات تکرار متن را روی عدد ۵ تنظیم می‌کند، در حالی که دگمه‌ی بعدی فقط باعث پس فرستاده شدن صفحه می‌شود. خواهید دید که هر بار که دگمه‌ی Post Back را کلیک می‌کنید، خصوصیت RepeatTimes به مقدار پیش فرض آن (۳) برگردانده می‌شود. اگر خصوصیت Text را نیز از طریق کُد تغییر بدهید خواهید دید این خصوصیت نیز به مقدار پیش فرضی که برای آن در کلاس ConfigurableRepeater تعریف شده بر می‌گردد.

این مشکل یک راه حل ساده دارد. کافی است مقدار خصوصیت‌های RepeatTimes و Text را در View State ثبت کنیم. در زیر نسخه‌ی بازنویسی شده‌ی کنترل ConfigurableRepeater ذکر شده که مقدار خصوصیت‌ها را با استفاده از View State در صفحه حفظ می‌کند و موقع برگشت صفحه آنها را از View State خوانده و احیاء می‌کند.



شکل ۱۵-۲۵. آزمایش حفظ وضعیت در خصوص یک کنترل شخصی

```
public class ConfigurableRepeater: WebControl
{
   public ConfigurableRepeater()
   {
      ViewState["RepeatTimes"] = 3;
      ViewState["Text"] = "";
   }
   public int RepeatTimes
   {
      get
      { return (int)ViewState["RepeatTimes"]; }
      set
      { ViewState["RepeatTimes"] = value; }
   }
   public string Text
   {
      get
      { return (string)ViewState["Text"]; }
      set
      { ViewState["Text"] = value; }
   }
   protected override void RenderContents(HtmlTextWriter writer)
   {
      base.RenderContents(writer);
      for (int i = 0; i < RepeatTimes; i++)
      {
         writer.Write(Text + "<br>");
      }
   }
}
```

کاری که کُد این کلاس انجام می‌دهد درست مانند کلاس قبلی است، با این تفاوت که در اینجا برای مقداردهی اولیه‌ی خصوصیت‌ها از سازنده‌ی بدون آرگومان کلاس استفاده شده است. در این تکنیک باید دقت کنید در روال‌های get خصوصیت‌ها بعد از خواندن اطلاعاتی که در ViewState ذخیره کرده‌اید و قبل از تحویل آنها به برنامه‌نویس باید آنها را به طور دستی به نوع داده‌ای که برای هر خصوصیت در نظر گرفته بودید تبدیل کنید. توجه کنید اگرچه کُد ذخیره‌سازی یک مقدار در ViewState بسیار به کُد ذخیره‌سازی مقادیر در ViewState یک صفحه‌ی وب شباهت دارد، اما این دو ViewState با یکدیگر تفاوت دارد. چنین تفکیکی در ASP.NET به منظور امنیت و صحت بیشتر انجام شده تا برنامه‌نویسان صفحات وب نتوانند به طور مستقیم به ViewState مربوط به کنترل‌ها دسترسی پیدا کنند.

اگر خصوصیت EnableViewState کنترل شما با false مقداردهی شود، دیگر کلکسیون ViewState برای کنترل فعلی حفظ نمی‌شود. البته کُد شما باز هم کار می‌کند، اما از مقادیری که سعی می‌کنید در ViewState ذخیره کنید صرفنظر می‌شود. لذا به عنوان مثال کنترل ConfigurableRepeater مانند قبل (بدون ViewState) عمل می‌کند و هر بار که صفحه پس فرستاده می‌شود بعد از نمونه‌سازی از این کنترل، مقدار خصوصیت‌های آن با مقدار پیش فرض آنها مقداردهی می‌شود.

ممکن است چنین چیزی جالب به نظر نرسد، اما همان چیزی است که می‌خواهیم. زیرا به عنوان مثال برنامه‌نویس یک صفحه ممکن است بخواهد هر بار که صفحه پس فرستاده می‌شود برخی از خصوصیت‌ها را مقداردهی کند. در این حالت دلیلی برای استفاده از ViewState برای حفظ این مقادیر وجود ندارد. چیزی که باید اطمینان حاصل کنید این است که کنترل شما بدون ViewState نیز درست کار خواهد کرد، حتی اگر قرار نباشد مقدار خصوصیت‌ها در پس فرستاده شدن صفحه حفظ شود.

نکته: در برخی سناریوهای ساخت برخی کنترل‌های پیشرفته ممکن است بخواهید مجموعه‌ی حداقلی از اطلاعات وضعیتی را حفظ کنید تنها به این منظور که کنترل شما کار کند. در چنین سناریوهایی می‌توانید از قابلیتی که ASP.NET برای حفظ وضعیت کنترل‌ها فراهم می‌کند استفاده کنید. ایده‌ی کلی این است که متد ()SaveControlState را ابطال می‌کنید تا اطلاعاتی را که لازم دارید ذخیره کنید. سپس برای بازیابی آنها از متد ()LoadControlState استفاده می‌کنید. در این حالت حتی اگر خصوصیت EnableViewState نیز false باشد اطلاعات باز هم در ViewState حفظ می‌شود. اما مایکروسافت توصیه می‌کند از این قابلیت با احتیاط استفاده کنید و تنها زمانی که واقعاً به چنین چیزی نیاز دارید از آن استفاده کنید. برای کسب اطلاعات بیشتر درباره‌ی این قابلیت به مستندات متدهای ()SaveControlState و ()LoadControlState در راهنمای MSDN مراجعه کنید.

پشتیبانی زمان طراحی[4]

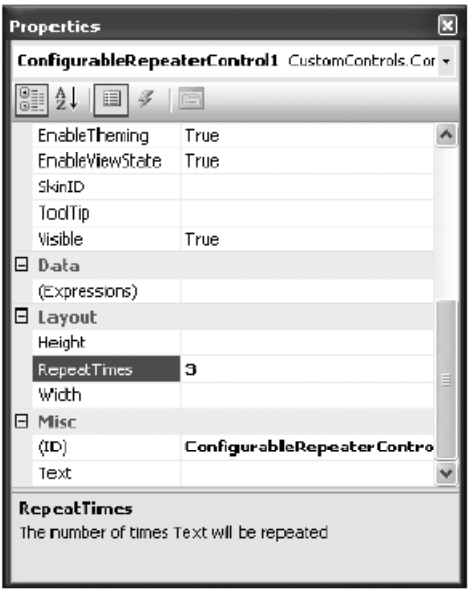
در Visual Studio خصوصیت‌هایی که اضافه بر خصوصیت‌های کنترل پدر برای یک کنترل شخصی تعریف می‌کنید به طور خودکار در پنجره‌ی Properties تحت گروه Misc نمایش داده می‌شود. اما می‌توانید با استفاده از ویژگی‌هایی در فضای نام System.ComponentModel، رفتار محیط طراحی Visual Studio را در خصوص تعامل با کنترل شخصی تغییر بدهید. ابتدا فضای نام System.ComponentModel را به کُد کلاس خود import کنید:

```
using System.ComponentModel;
```

به عنوان مثال در زیر با اعمال ویژگی [Category] خصوصیت RepeatTimes همیشه تحت گروه Layout در پنجره‌ی Properties نمایش داده می‌شود.

```
[Category("Layout")]
public int RepeatTimes
{
   get { return repeatTimes; }
   set { repeatTimes = value; }
}
```

توجه کنید ویژگی‌ها همیشه بین براکت باز و بسته قرار می‌گیرند و همیشه باید در همان خطی که قرار است به آن ارجاع کنند ذکر شوند (در اینجا تعریف خصوصیت RepeatTimes) یا توسط کاراکتر زیر خط از آن جدا شوند (اگر خاطرتان باشد ویژگی‌ها در خصوص سرویس‌های وب نیز استفاده می‌شوند مانند ویژگی WebMethod و WebService).



شکل ۱۶-۲۵. شخصی‌سازی پشتیبانی زمان طراحی محیط Visual Studio از کنترل‌های شخصی

[4] Design-Time Support

با استفاده از کاما نیز می‌توانید به هر تعداد ویژگی که بخواهید بین کاراکترهای براکت تعریف کنید. در مثال زیر از ویژگی Category و Description استفاده کرده‌ایم. شکل ۱۶–۲۵ نتیجه‌ی اعمال این ویژگی‌ها را به خصوصیت RepeatTimes در پنجره‌ی Properties در Visual Studio نشان می‌دهد.

```
[Description("The number of times Text will be repeated"), Category("Layout")]
```

در جدول ۱–۲۵ لیستی از ویژگی‌های مختلفی که برای شخصی‌سازی محیط طراحی Visual Studio در خصوص کنترل‌های شخصی می‌توانید استفاده کنید ذکر شده است.

جدول ۱–۲۵. ویژگی‌های قابل استفاده برای شخصی‌سازی محیط طراحی Visual Studio در رابطه با خصوصیت‌های کنترل‌های شخصی

ویژگی	توضیح
[Browsable(true|false)]	اگر false باشد خصوصیت فعلی در پنجره‌ی Properties نشان داده نخواهد شد (البته مادامی که برای این خصوصیت روال set تعریف کرده باشید، برنامه‌نویس کماکان می‌تواند آن را از طریق کُد یا از طریق خصوصیت‌های تگ کنترلی، مقداردهی کند).
[Category(" ")]	نام گروهی که این خصوصیت تحت آن در پنجره‌ی Properties نمایش داده خواهد شد.
[Description(" ")]	توضیحی که برای خصوصیت در پایین پنجره‌ی Properties نمایش داده خواهد شد.
[DefaultValue()]	مقدار پیش فرضی که برای این خصوصیت در پنجره‌ی Properties نمایش داده خواهد شد.
[ParenthesizePropertyName(true|false)]	اگر true باشد Visual Studio در پنجره‌ی Properties دور خصوصیت مزبور پرانتز قرار می‌دهد (مانند خصوصیت ID).
[ToolboxData(" ")]	توسط این ویژگی می‌توان مشخص کرد وقتی نمونه‌ای از این کنترل با استفاده از جعبه ابزار در یک صفحه ایجاد می‌شود، تگ کنترلی آن چه قالبی داشته باشد. در حالت کلی معمولاً تگ کنترل‌های شخصی به صورت <TagPrefix:ClassName> می‌باشد (مانند <apress:NameTextBox>). اما می‌توانید این قاعده را طوری تغییر بدهید که تگی که برای کنترل استفاده می‌شود با نام کلاس آن کنترل فرق داشته باشد. برای این کار از ویژگی ToolboxData استفاده کنید و برای آن رشته‌ای به صورت "<{0}:NameTextBox runat=server></{0}:NameTextBox>" مشخص کنید که در این رشته قسمت {0} معادل پیشوند تگ کنترلی بوده و مابقی آن چه که ذکر می‌شود به همان صورت به هنگام ایجاد یک نمونه از کنترل، در صفحه‌ی aspx. نوشته خواهد شد.

ایجاد یک کنترل مرکب[5]

تا اینجا دیدید کنترل‌های کاربر عموماً برای مجتمع‌سازی مجموعه‌ای از کنترل‌ها به همراه سطح بالاتری از منطق کسب و کار برنامه هستند. از آن سو توسط کنترل‌های شخصی نیز می‌توانید کنترل‌های از پیش آماده را دستکاری کنید و خروجی HTML آنها را تغییر بدهید. همان طور که در عمل خواهید دید به طور کلی کنترل‌های کاربر را سریعتر می‌توان ایجاد کرد و برنامه‌نویسی آنها نیز ساده‌تر است. اما از آن سو کنترل‌های شخصی قابلیت‌های سطح پایین متنوعی فراهم می‌کنند که هنوز در این فصل به آنها نپرداخته‌ایم، مانند قابلیت مقیدسازی و الگوها.

یک تکنیک دیگر که هنوز با آن آشنا نشده‌اید کنترل‌های مرکب است. نوعی از کنترل‌های شخصی که از ترکیب چند کنترل به دست می‌آیند. این کنترل‌ها بیشتر به کنترل‌های کاربر شبیه هستند، زیرا تا حدی واسط کاربری و خروجی HTML خود را بر اساس کنترل‌های دیگر می‌سازند و تولید می‌کنند. به عنوان مثال ممکن است به این نتیجه برسید که به واسط کاربری پیچیده‌ای که بر اساس یک جدول HTML بنا شده نیاز دارید. به جای این که کل بلاک HTML را به صورت رشته و به طور دستی در متد ()Render در خروجی بنویسید (و سعی کنید آن را بر اساس خصوصیت‌های سطح بالاتری که برای کنترل تعریف کرده‌اید قابل تنظیم کنید)، می‌توانید به صورت پویا یک کنترل جدول ایجاد کنید. چنین روشی در کنترل‌های سرویس‌دهنده‌ی ASP.NET کاملاً رایج است – در واقع کنترل‌های پیشرفته‌ای مانند کنترل تقویم (Calendar) و شبکه‌ی داده (GridView) خودشان برای تولید واسط کاربرشان بر اساس کنترل‌های ساده‌تری مانند Table بنا شده‌اند.

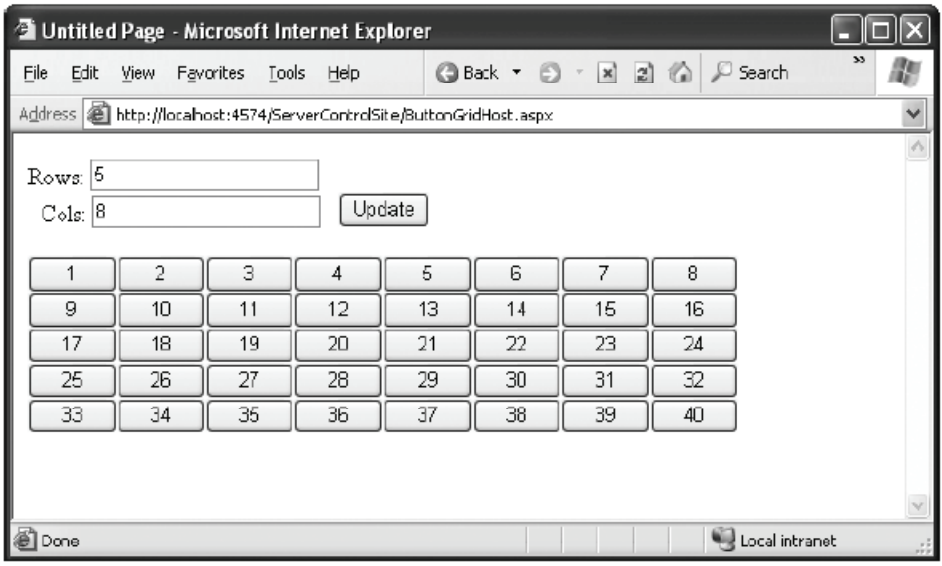
ساخت کنترل‌های مرکب بسیار ساده است. تمام کاری که باید انجام بدهید این است که کنترل خود را از کلاسی مانند Control یا WebControl مشتق کنید، متد ()CreateChildControls آن را سربارگذاری کنید، کنترل‌هایی را که نیاز دارید ایجاد کنید و آنها را بر اساس ترتیب و چیدمانی که لازم دارید به کلکسیون ()Controls در کنترل شخصی اضافه کنید.

در مثال زیر یک کنترل شخصی تعریف شده که شبکه‌ای از دگمه‌ها را نمایش می‌دهد و تعداد سطرها و ستون‌های شبکه را توسط خصوصیت‌های Rows و Cols از کاربر دریافت می‌کند. در شکل ۱۷–۲۵ نیز یک صفحه‌ی ساده برای آزمایش این کنترل نشان داده شده است.

```
public class ButtonGrid: Control
{
  public ButtonGrid()
  {
    Rows = 2;
    Cols = 2;
  }
  public int Cols
  {
    get { return (int)ViewState["Cols"]; }
    set { ViewState["Cols"] = value; }
  }
  public int Rows
  {
```

[5] Composite Control

```
    get { return (int)ViewState["Rows"]; }
    set { ViewState["Rows"] = value; }
  }
  protected override void CreateChildControls()
  {
    int count = 0;
    for (int row = 0; row < (int)ViewState["Rows"]; row++)
    {
      for (int col = 0; col < (int)ViewState["Cols"]; col++)
      {
        count++;

        // Create and configure a button.
        Button ctrlB = new Button();
        ctrlB.Width = Unit.Pixel(60);
        ctrlB.Text = count.ToString();

        // Add the Button
        this.Controls.Add(ctrlB);
      }

      // Add a line break.
      LiteralControl ctrlL = new LiteralControl("<br/>");
      this.Controls.Add(ctrlL);
    }
  }
}
```



شکل ۱۷-۲۵. یک کنترل مرکب که از کنترل‌های دیگر استفاده می‌کند

رویدادها و پس فرستاده شدن کنترل‌های شخصی

آتش کردن رویدادهای شخصی در کنترل‌های مرکب درست به همان سادگی کنترل‌های کاربر است. تمام کاری که باید انجام بدهید این است که رویداد شخصی (و هر کلاس لازم EventArgs دیگر) را تعریف کنید و سپس با فراخوانی متغیری که برای رویداد تعریف کرده‌اید، آن را آتش کنید (البته قبل از آن با مطابق دادن این متغیر با مقدار null باید بررسی کنید که آیا کاربر برای رویدادی که برای کنترل خود تعریف کرده‌اید اداره‌گری نوشته و آن را اداره کرده است یا خیر).

با این حال برای این که رویداد شما آتش شود ابتدا کُد شما باید اجرا شود و برای این که کُد شما اجرا شود صفحه باید به سمت سرور پس فرستاده شود. لذا در درجه‌ی اول باید کاری کنیم که صفحه پس فرستاده شود و در درجه‌ی دوم باید این مساله را متوجه شویم (که این خودمان بوده‌ایم که صفحه را پس فرستاده‌ایم، نه یک کنترل دیگر).

در حالت کلی کنترل‌های وب در یک صفحه با فراخوانی یک تابع جاوا اسکریپتی به اسم ()__doPostBack صفحه را پس می‌فرستند. این تابع در فصل ۷ معرفی شد و دو پارامتر دارد: ارجاعی به شیء جاوااسکریپتی کنترلی که می‌خواهد صفحه را پس بفرستد و رشته‌ای که حاوی اطلاعاتی اضافی در خصوص این رویداد است. ساده‌ترین راه برای فراخوانی تابع جاوااسکریپتی ()__doPostBack استفاده از ویژگی OnClick در المان HTML کنترل مورد نظر است (در زبان HTML المان <a>، <img>، <input> همگی خصوصیت OnClick دارند). لذا برای پس فرستادن صفحه باید این ویژگی را باید در کُد تولید خروجی کنترل خود به المان آن اضافه کنیم.

اما به جای نوشتن دستور فراخوانی تابع ()__doPostBack به صورت دستی و نسبت دادن آن به ویژگی OnClick می‌توانیم از متد ()Page.GetPostBackEventReference استفاده کنیم (تمام کنترل‌ها خصوصیتی به اسم Page دارند که ارجاعی به صفحه‌ی جاری که کنترل در آن به کار رفته بر می‌گرداند).

کنترل شبکه‌ی دگمه‌های مثال قبل را در نظر بگیرید. تا اینجا تمام دگمه‌ها ایجاد شده‌اند، اما هنوز راهی برای دریافت رویداد آنها نداریم. برای حل مشکل یک ویژگی OnClick به المان خروجی دگمه‌ها (که المان <input> یا <button> خواهد بود) اضافه می‌کنیم و آن را با رشته‌ای که ()Page.GetPostBackEventReference بر می‌گرداند تنظیم می‌کنیم. این متد دو پارامتر دارد: شیء جاری و یک رشته‌ی اختیاری. مقدار برگشتی آن نیز رشته‌ای حاوی یک فراخوانی به تابع ()__doPostBack با آرگومان‌های مناسب است. خطوط انجام این کار در کُد زیر پر رنگ شده است:

```
protected override void CreateChildControls()
{
   int k = 0;
   for (int i = 0; i < Rows; i++)
   {
      for (int j = 0; j < Cols; j++)
      {
         k++;

         // Create and configure a button.
         Button ctrlB = new Button();
         ctrlB.Width = Unit.Pixel(60);
```

```
        ctrlB.Text = k.ToString();

        // Set the OnClick attribute with a reference to
        // __doPostBack. When clicked, ctrlB cause a
        // postback to the current control and return
        // the assigned text of ctrlB.
        ctrlB.Attributes["OnClick"] = Page.ClientScript.GetPostBackEventReference(this, ctrlB.Text);

        // Add the Button
        this.Controls.Add(ctrlB);
      }

      // Add a line break.
      LiteralControl ctrlL = new LiteralControl("<br/>");
      this.Controls.Add(ctrlL);
    }
}
```

از آن سو بعد از برگشت صفحه برای این که کنترل ما بتواند از این مساله مطلع شود باید به صورتی که در زیر نشان داده شده در کلاس آن واسط IPostBackEventHandler را نیز پیاده‌سازی کنیم:

```
public class ButtonGrid: CompositeControl, IPostBackEventHandler
{
  // (Control code goes here).
}
```

واسط IPostBackEventHandler متدی به اسم ()RaisePostBackEvent دارد که باید آن را برای کلاس کنترل‌مان پیاده‌سازی کنیم. در حالتی که یک کنترل باعث پس فرستاده شدن صفحه شود ASP.NET به طور خودکار این متد را (برای تمام کنترل‌هایی که واسط IPostBackEventHandler را پیاده‌سازی کرده‌اند) فراخوانی می‌کند و هرگونه اطلاعات اضافی را که به متد ()Page.GetPostBackEventReference داده بودید به آن می‌دهد. در مثال کنترل ButtonGrid، برای این اطلاعات اضافی از متن دگمه‌ی کلیک شده استفاده کردیم.

```
public virtual void RaisePostBack(string eventArgument)
{
  // (Respond to postback here).
}
```

بعد از دریافت اطلاعاتی که صفحه پس فرستاده می‌توانید کنترل خود را تغییر بدهید یا حتی رویداد دیگری برای صفحه آتش کنید. بگذارید این کار را برای کنترل ButtonGrid انجام بدهیم. برای این کنترل رویداد جدیدی به اسم GridClick تعریف می‌کنیم که موقع کلیک شدن دکمه‌هایش رخ می‌دهد و نام دکمه را به شیء EventArgs در پارامتر دوم اداره‌گر رویداد پاس می‌دهد. بدین ترتیب کسی که از این کنترل استفاده می‌کند با اداره کردن رویداد GridClick می‌تواند متوجه شود کدام دکمه کلیک شده است و کاری مطابق نیاز خود انجام بدهد. برای بهبود دادن کنترل ButtonGrid با این قابلیت‌ها باید یک کلاس EventArgs، یک نماینده و یک رویداد تعریف کنیم. داریم:

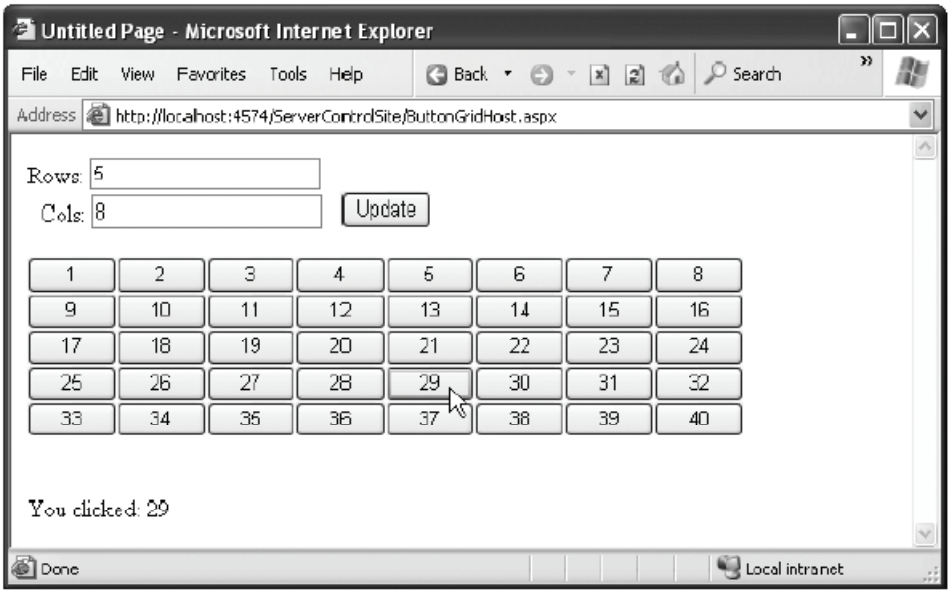
```
public class GridClickEventArgs: EventArgs
{
   public string ButtonName;
   public GridClickEventArgs(string buttonName)
   {
      ButtonName = buttonName;
   }
}
public delegate void GridClickEventHandler(Object sender, GridClickEventArgs e);

public event GridClickEventHandler GridClick;
```

رویداد GridClick را باید داخل متد RaisePostBackEvent() آتش کنیم. همان طور که گفته شد در برگشت صفحه ASP.NET به طور خودکار این متد را برای تمام کنترل‌هایی کـه واسـط IPostBackEventHandler را پیـاده‌سـازی کرده‌اند فراخوانی می‌کند. کُد کامل کنترل ButtonGrid به صورت زیر خواهد بود:

```
public class ButtonGrid : Control,IPostBackEventHandler
{
   public event GridClickEventHandler GridClick;

   public ButtonGrid()
   {
      Rows = 2;
      Cols = 2;
   }
   public int Cols
   {
      get { return (int)ViewState["Cols"]; }
      set { ViewState["Cols"] = value; }
   }

   public int Rows
   {
      get { return (int)ViewState["Rows"]; }
      set { ViewState["Rows"] = value; }
   }

   public void RaisePostBackEvent(string eventArgument)
   {
      // This is a workaround for a bug that makes
      // the eventArgument null (when it should contain the
      // button text).
      string evtArg = null;

      if (eventArgument == null)
      {
         if (Page.Request.Form["__EVENTTARGET"] == this.UniqueID)
         {
            evtArg = Page.Request.Form["__EVENTARGUMENT"];
         }
      }
      if (GridClick != null)
      {
         GridClick(this, new GridClickEventArgs(evtArg));
      }
   }
```

```
  protected override void CreateChildControls()
  {
    base.CreateChildControls();
    int k = 0;
    for (int i = 0; i < Rows; i++)
    {
      for (int j = 0; j < Cols; j++)
      {
        k++;

        // Create and configure a button.
        Button ctrlB = new Button();
        ctrlB.Width = Unit.Pixel(60);
        ctrlB.Text = k.ToString();
        ctrlB.Attributes["onClick"] =
        Page.ClientScript.GetPostBackEventReference(this, ctrlB.Text);

        // Add the button.
        this.Controls.Add(ctrlB);
      }
      // Add a line break.
      LiteralControl ctrlL = new LiteralControl("<br />");
      this.Controls.Add(ctrlL);
    }
  }
  protected override void OnInit(EventArgs e)
  {
    Page.RegisterRequiresRaiseEvent(this);
    base.OnInit(e);
  }
}
```

در شکل ۱۸–۲۵ صفحه‌ای نشان داده شده که توسط آن می‌توانید کنترل ButtonGrid را آزمایش کنید. این صفحه رویداد GridClick را اداره کرده و پیامی نشان می‌دهد که اعلام می‌کند کدام دگمه کلیک شده است. همچنین این صفحه به کاربر اجازه می‌دهد تعداد سطرها و ستون‌های شبکه‌ی دگمه‌ها را مشخص کند.



شکل ۱۸-۲۵. اداره‌ی رویدادهای کنترل‌های شخصی

کُد صفحه‌ی وب این مثال نیز چنین است:

```
public partial class ButtonGridHost : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
   }
   protected void cmdUpdate_Click(object sender, EventArgs e)
   {
      ButtonGrid1.Rows = Int32.Parse(txtRows.Text);
      ButtonGrid1.Cols = Int32.Parse(txtCols.Text);
   }
   protected void ButtonGrid1_GridClick(object sender, CustomControls.GridClickEventArgs e)
   {
      lblInfo.Text = "You clicked: " + e.ButtonName;
   }
}
```

نکته: یکی از نکات جالب کامپایل کنترل‌های شخصی در قالب یک اسمبلی جدا این است که Visual Studio در محیط طراحی خود به خوبی از آنها پشتیبانی می‌کند. نه تنها می‌توانید کنترل‌ها را در این حالت به جعبه ابزار اضافه کنید، بلکه از طریق پنجره‌ی Properties می‌توانید برای هر یک از رویدادهای شخصی که برای آنها تعریف کرده‌اید (مانند ButtonGrid.GridClick)، اداره‌گر رویداد ایجاد کنید. اما متاسفانه این کار در خصوص کنترل‌های کاربر امکان‌پذیر نیست.