ASP.NET 3.5编程（影印版）

下卷
第4版
Updated for ASP.NET 3.5
and Visual Studio 2008

Programming
ASP.NET 3.5

O'REILLY®
开明出版社

Jesse Liberty, Dan Hurwitz
& Dan Maharry 著

第4版，下卷

ASP.NET 3.5 编程（影印版）

Programming ASP.NET 3.5

Jesse Liberty, Dan Hurwitz & Dan Maharry

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权北京凤凰天下文化发展有限公司、开明出版社出版发行

开明出版社

图书在版编目（CIP）数据

ASP.NET 3.5 编程：第 4 版 =Programming ASP.NET 3.5, 4E：英文 / （美）丽波蒂（Liberty，J.），（美）赫维茨（Hurwitz，D.），（美）马哈里（Maharry，D.）著 . 影印本 .—北京：开明出版社，2009.3
ISBN 978-7-80205-735-7

Ⅰ. A… Ⅱ. ①丽…②赫…③马… Ⅲ. 主页制作—程序设计—英文 Ⅳ .TP393.092

中国版本图书馆 CIP 数据核字（2009）第 035516 号

江苏省版权局著作权合同登记
图字：10-2009-081 号

书名：ASP.NET 3.5 编程 第四版（影印版）
出版：开明出版社出版（北京海淀区西三环北路 19 号　邮编 100089）
经销：全国新华书店
印刷：北京市梦宇印务有限公司（北京市通州区张家湾镇张辛庄村）
开本：787 × 1092　1/16
印张：73 .75
字数：1239 千字
版次：2009 年 4 月　北京第 1 版
印次：2009 年 4 月　北京第 1 次印刷
定价：118.00 元（上、下册）

印刷、装订质量问题，出版社负责调换货　联系电话：（010）88817647

Table of Contents

上卷

下卷

CHAPTER 11
Validation

As you saw in previous chapters, many web applications involve user input. Sadly, however, users make mistakes: they skip required fields, they enter phone numbers with the wrong number of digits, and they send to your application all manner of incorrectly formatted data. Your database routines can choke on corrupted data, and orders can be lost, for example, if a credit card number is entered incorrectly or an address is omitted, so it is imperative to validate user input.

Traditionally, it takes a great deal of time and effort to write reliable validation code. Each field must be checked, and routines must be created for ensuring data integrity. If bad data is found, error messages must be displayed so that the user knows there is a problem and knows how to correct it.

In a given application, you may choose to validate that certain fields have a value, that the values fall within a given range, or that the data is formatted correctly. For example, when processing an order, you may need to ensure the user has input an address and phone number, the phone number has the right number of digits (and no letters), and that the Social Security number entered is in the appropriate form of nine digits separated with hyphens.

Some applications require more complex validation, in which one field is validated to be within a range established by two other fields. For example, in one field you might ask what date a customer wishes to arrive at your hotel, and in a second field you might ask for the departure date. When the user books dinner, you'll want to ensure the date is between the arrival and departure dates.

There is no limit to the complexity of the validation routines you may need to write. Credit cards have checksums built into their values, as do ISBN numbers. Zip and postal codes follow complex patterns, as do international phone numbers. You may need to validate passwords, membership numbers, dollar amounts, dates, runway choices, and launch codes.

In addition, you usually want all of this validation to happen on the client side, so you can avoid the delay of repeated round trips to the server while the user is tinkering with his input. In the past, this was solved by writing client-side JavaScript to validate the input, and then writing server-side script to handle input from browsers that don't support client-side programming. In addition, as a security check, you may want to do server-side validation even though you have client-side validation, because users can circumvent validation code by deliberately spoofing requests. Traditionally, this involved writing your validation code twice, once for the client and once for the server.

As you can see, validating user input can require a lot of hard work, but ASP.NET simplifies this process considerably by providing rich controls for this task. The validation controls allow you to specify how and where the error messages will be displayed: inline with the input controls, aggregated together in a summary report, or both. These controls can be used to validate input for both HTML and ASP.NET server controls.

You add validation controls to your ASP.NET document as you would add any other control. Within the declaration of the validation control, you specify which other control is being validated. You may freely combine the various validation controls, and you may even write your own custom validation controls, as you'll see later in this chapter.

With up-level browsers that support DHTML, such as Internet Explorer 4 and later, .NET validation is done on the client side, avoiding the necessity of a round trip to the server. With down-level browsers or browsers with scripting turned off, your code is unchanged, but the code sent to the client ensures validation at the server.

> Even when client-side validation is done, the values are also validated on the server side as a security measure.

Because client-side validation will prevent your server-side code from ever running if the control is invalid, sometimes you may want to force server-side validation. In that case, add a ClientTarget attribute to the @Page directive:

```
<% @Page Language="C#" AutoEventWireup="true"
   CodeFile="Default.aspx.cs" Inherits="Default_aspx"
   ClientTarget="downlevel"
%>
```

This directive will cause the validation to occur on the server even if your browser would have supported DHTML and client-side validation.

Sometimes you don't want any validation to occur, such as when a Cancel button is clicked. To specify this, many server controls, such as Button, ImageButton, LinkButton, ListControl, and TextBox, have a CausesValidation property, which dictates whether validation is performed on the page when the control's default event is raised.

If CausesValidation is set to true, which is the default value, the postback will not occur if any control on the page fails validation. If CausesValidation is set to false, however, no validation will occur when that button is used to post the page.

ASP.NET supports the following validation controls:

`RequiredFieldValidator`
: Ensures the user does not skip over your input control. A `RequiredFieldValidator` can be tied to a `TextBox` to force input into the `TextBox`. With selection controls, such as a `DropDownList` or `RadioButtons`, the `RequiredFieldValidator` ensures the user makes a selection other than the default value you specify. The `RequiredFieldValidator` does not examine the validity of the data, but only ensures that some data is entered or chosen.

`RangeValidator`
: Ensures the value entered is within a specified lower and upper boundary. You can check the range within a pair of numbers (greater than 10 and less than 100), a pair of characters (greater than D and less than K), or a pair of dates (after 1/1/09 and before 2/28/09).

`CompareValidator`
: Compares the user's entry against another value. It can compare against a constant you specify at design time, or against a property value of another control. It can also compare against a database value.

`RegularExpressionValidator`
: One of the most powerful validators, `RegularExpressionValidator` compares the user's entry with a regular expression you provide. You can use this validator to check for valid Social Security numbers, phone numbers, passwords, and so on.

`CustomValidator`
: For use if none of the previous controls meets your needs. `CustomValidator` checks the user's entry against whatever algorithm you provide in a custom method.

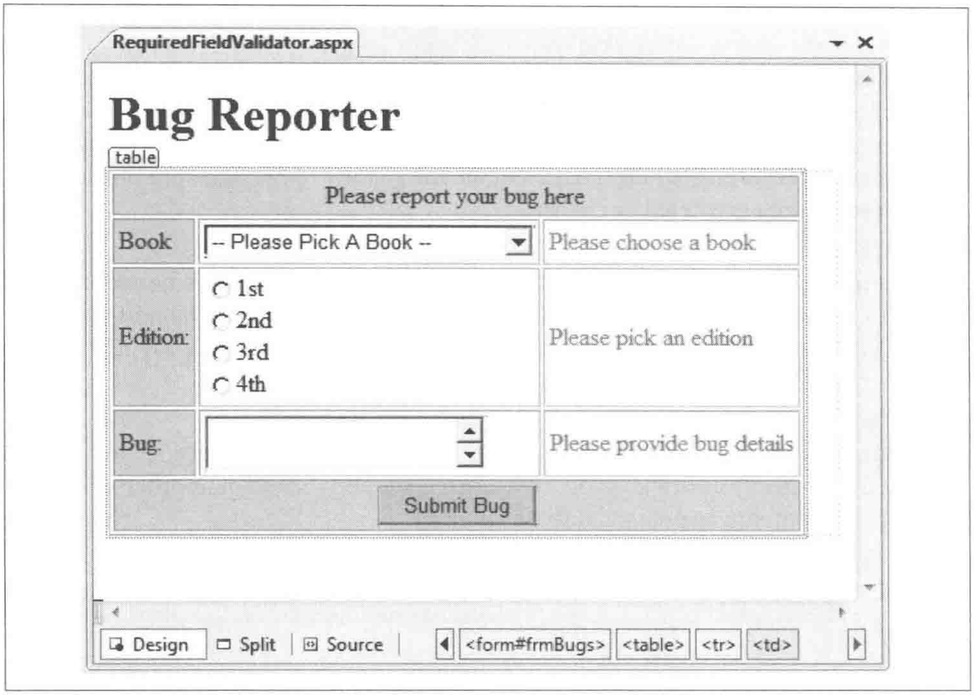
In the remainder of this chapter, we'll examine how to use each of these controls to validate data in ASP.NET applications. You'll also see how some of the extender controls that come with the ASP.NET AJAX Control Toolkit aid in guiding a user's input toward the desired entry.

The RequiredFieldValidator

The `RequiredFieldValidator` ensures the user provides a valid value for your control. To demonstrate, create for the chapter a new website called *C11_Validation* and then add a new web page called *RequiredFieldValidator.aspx*. You'll create a simple form nominally for reporting bugs, as shown in Figure 11-1.

Validator controls will be added to the form such that when the user clicks the Submit Bug button, the page is validated to ensure that each field has been modified. If not, the offending field is marked with an error message in red. See Figure 11-2.

To help with layout, add an HTML table to the page with five rows and three columns. (You could equally put each row of the table in a `<div>` or `<p>` tag and lay things out with CSS, but that would just complicate the code here.) In the first row,



Bug Reporter

Please report your bug here		
Book	-- Please Pick A Book --	Please choose a book
Edition:	○ 1st ○ 2nd ○ 3rd ○ 4th	Please pick an edition
Bug:		Please provide bug details
Submit Bug		

Figure 11-1. RequiredFieldValidator.aspx design

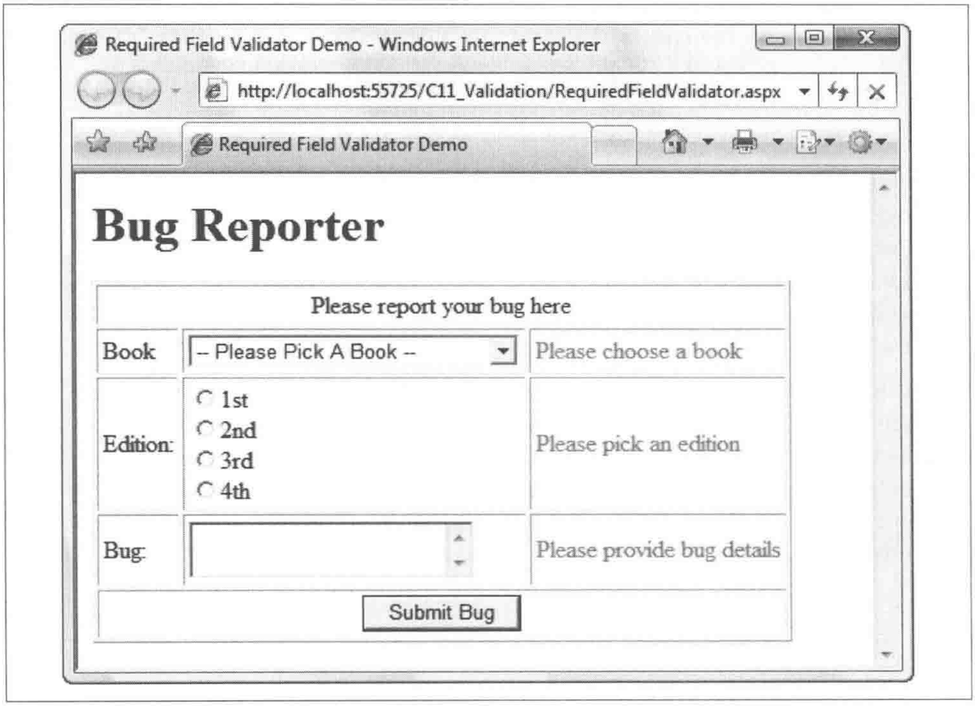
add a `Label` control called `lblMsg`. Later on, you'll use the `Page_Load` method to change its `Text` property depending on whether the page is valid or not.

```
<%@ Page Language="C#" AutoEventWireup="true"
   CodeFile="RequiredFieldValidator.aspx.cs" Inherits="RequiredFieldValidator" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
   <title>Required Field Validator Demo</title>
</head>

<body>
   <h1>Bug Reporter</h1>
   <form runat="server" id="frmBugs">
   <table>
      <tr>
         <td colspan="3" align="center">
            <asp:Label ID="lblMsg" Text="Please report your bug here"
             runat="server" />
         </td>
      </tr>
```


Required Field Validator Demo - Windows Internet Explorer

http://localhost:55725/C11_Validation/RequiredFieldValidator.aspx

Bug Reporter

Please report your bug here		
Book	-- Please Pick A Book --	Please choose a book
Edition:	○ 1st ○ 2nd ○ 3rd ○ 4th	Please pick an edition
Bug:		Please provide bug details

Figure 11-2. Required field validation errors reported when page is submitted

The second row contains a DropDownList control (ddlBooks) and a RequiredFieldValidator control (rfvBooks). Besides the mandatory runat and ID, its most important properties are ControlToValidate, which identifies the control that it should validate, and ErrorMessage, which contains the text the control will display if a value has not been selected in the list, or if the value selected is the same as the one given in the InitialValue property.

rfvBooks also has a Display property, which is set to Static. This tells ASP.NET to allocate room on the page for the validator regardless of whether there is a message to display. If this property is set to Dynamic, space will not be allocated until (and unless) an error message is displayed.

```
<tr>
   <td>Book</td>
   <td>
      <asp:DropDownList ID="ddlBooks" runat="server">
         <asp:ListItem>-- Please Pick A Book --</asp:ListItem>
         <asp:ListItem>Programming ASP.NET</asp:ListItem>
         <asp:ListItem>Learning ASP.NET With AJAX</asp:ListItem>
         <asp:ListItem>Programming C# 2008</asp:ListItem>
         <asp:ListItem>Programming Visual Basic 2008</asp:ListItem>
      </asp:DropDownList>
   </td>
```

```
            <td>
               <asp:RequiredFieldValidator ID="rfvBooks" ControlToValidate="ddlBooks"
                  Display="Static" InitialValue="-- Please Pick A Book --"
                  runat="server" ErrorMessage="Please choose a book" />
            </td>
         </tr>
```

Dynamic allocation is powerful, but it can cause your controls to bounce around on the page when the message is displayed. For example, if you set all the validation controls to Dynamic on this page, no space will be allocated for them. If you then click the Submit Bug button and one of the controls is not validated, the validators' error text will display, widening the table and relocating the Submit Bug button and Label that are centered across the whole table.

The third row in the table contains a RadioButtonList control (rblEdition) and another RequiredFieldValidator (rfvEdition) set to validate it. In this case, rfvEdition does not have its InitialValue property set. Because the control is a radio button list, the validator knows the user is required to pick one of the buttons; if any button is chosen, the validation will be satisfied. If no button is chosen, the validator's error message will be displayed.

```
         <tr>
            <td>
               Edition:
            </td>
            <td>
               <asp:RadioButtonList ID="rblEdition" RepeatLayout="Flow"
                runat="server">
                  <asp:ListItem>1st</asp:ListItem>
                  <asp:ListItem>2nd</asp:ListItem>
                  <asp:ListItem>3rd</asp:ListItem>
                  <asp:ListItem>4th</asp:ListItem>
               </asp:RadioButtonList>
            </td>
            <td>
               <asp:RequiredFieldValidator ID="rfvEdition"
                  ControlToValidate="rblEdition"
                  Display="Static" runat="server"
                  ErrorMessage="Please pick an edition" />
            </td>
         </tr>
```

To complete the example, the fourth row contains a multiline TextBox (txtBug) and a third RequiredFieldValidator (rfvBug) set to monitor it. Again, a user has either added some text to the TextBox or not, so no InitialValue property needs to be set on rfvBug.

The fifth and final row in the table contains a simple Button control, which when clicked automatically causes the page to initiate validation on the browser according to the rules set out in the three validation controls you've added to the page.

```
        <tr>
           <td>Bug: </td>
           <td>
              <asp:TextBox ID="txtBug" runat="server" TextMode="MultiLine" />
           </td>
           <td>
              <asp:RequiredFieldValidator ID="rfvBug"
                 ControlToValidate="txtBug" Display="Static"
                 runat="server" ErrorMessage="Please provide bug details" />
           </td>
        </tr>
        <tr>
           <td colspan="3" align="center">
              <asp:Button ID="btnSubmit" Text="Submit Bug" runat="server" />
           </td>
        </tr>
     </table>
     </form>
  </body>
  </html>
  <!-- end of RequiredFieldValidator.aspx -->
```

If you run the page now and click the Submit Bug button without making any changes on the form, each control being validated is checked and error messages are displayed, as shown previously in Figure 11-2.

Once validation has occurred, ASP.NET then posts back to the server when any server-side validation occurs. If that validation also passes, the Page's IsValid property is set to true or false depending on whether the value in every control being monitored is valid.

To demonstrate, add to the code-behind page in *RequiredFieldValidator.aspx* a handler for the Submit Bug button's Click event, and then add the highlighted code in Example 11-1.

Example 11-1. RequiredFieldValidator.aspx.cs in full

```
using System;
using System.Web.UI;

public partial class RequiredFieldValidator : Page
{
    protected void btnSubmit_Click(object sender, EventArgs e)
    {
       if (Page.IsValid)
       {
          lblMsg.Text = "Page is valid";
       }
       else
       {
          lblMsg.Text = "Some of the fields still have no value";
       }
    }
}
```

If you run the page again, notice that when the Submit Bug button is clicked, the page is posted to the server only if all client-side validation routines return true. If you view the source for the page, you'll see the following code, which is injected into the page and enforces this behavior:

```
<script type="text/javascript">
//<![CDATA[
function WebForm_OnSubmit() {
if (typeof(ValidatorOnSubmit) == "function" &&
       ValidatorOnSubmit() == false) return false;
return true;
}
//]]>
</script>
```

> The text that AJAX Control Toolkit extender controls, such as the WatermarkExtender and MaskedEditExtender, add into a TextBox does not affect whether the RequiredFieldValidator works. The watermark or mask does not count as a value having been entered.

The Summary Control

You can decide how and where validation errors are reported. You are not required to place validator controls alongside the control they are validating, although it does help to identify which text box or list control has been filled out incorrectly. For forms of any size, though, a good strategy to help a user identify her mistakes is to summarize all the validation failures with a ValidationSummary control. This control can place a summary of the errors in a bulleted list, a simple list, or a paragraph that appears on the web page or in a pop-up message box.

To demonstrate, add to the website a new page called *ValidationSummary.aspx* and copy the contents of *RequiredFieldValidator.aspx* into it. Add a ValidationSummary control at the bottom of the page, between the closing </table> and </form> tags.

```
...
      <tr>
         <td colspan="3" align="center">
            <asp:Button ID="btnSubmit" Text="Submit Bug" runat="server" />
         </td>
      </tr>
   </table>
   <asp:ValidationSummary ID="ValidationSummary1" runat="server"
      DisplayMode="BulletList"
      HeaderText="The following errors were found: "
      ShowSummary="true" />
   </form>
</body>
</html>
```

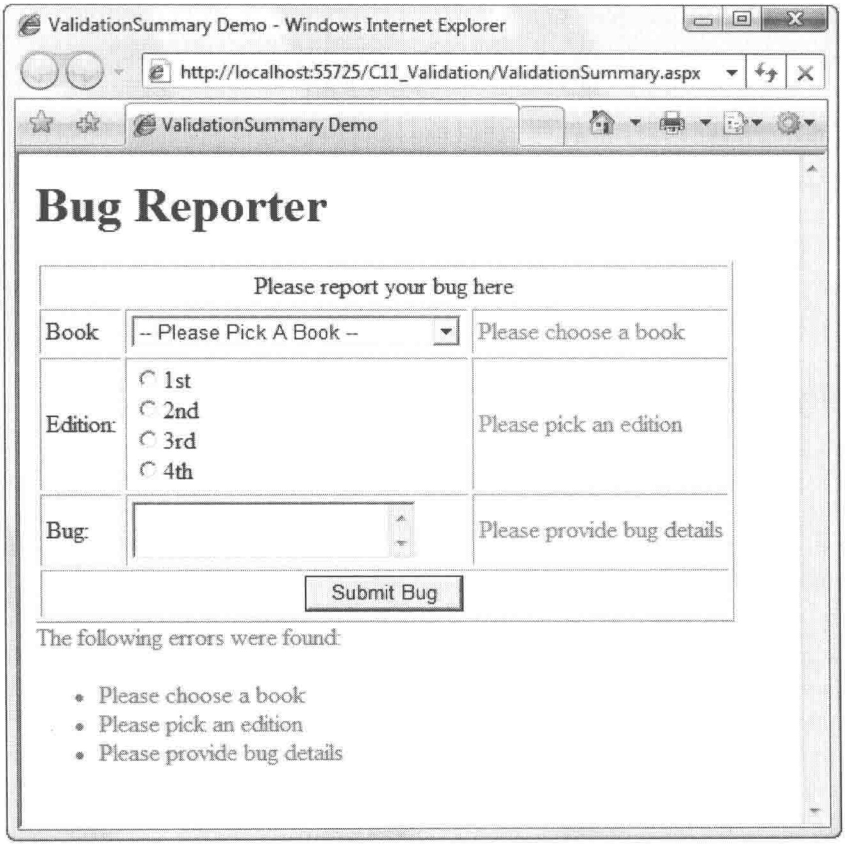
Three properties are set on the ValidationSummary control besides the mandatory runat and ID:

`DisplayMode`
: Sets the way in which those errors are shown in the summary. Possible values are `BulletList` (shown in Figure 11-3), `List`, and `SingleParagraph`.

`HeaderText`
: Sets the header that will be displayed if there are any errors to report.

`ShowSummary`
: Indicates that the errors should be shown in the body of the HTML document. It has a sister property, `ShowMessageBox`, which will display the errors in a pop-up message box if set to `true`. You can set both to `true` if desired.

Now save and run the page. If you click the Submit Bug button, the text in the validator controls' `ErrorMessage` attributes will be displayed in the summary if this control reports a validation error, as shown in Figure 11-3.


ValidationSummary Demo - Windows Internet Explorer
http://localhost:55725/C11_Validation/ValidationSummary.aspx
ValidationSummary Demo
Submit Bug

Bug Reporter

Please report your bug here		
Book	-- Please Pick A Book --	Please choose a book
Edition:	○ 1st ○ 2nd ○ 3rd ○ 4th	Please pick an edition
Bug:		Please provide bug details

The following errors were found:

- Please choose a book
- Please pick an edition
- Please provide bug details

Figure 11-3. ValidationSummary.aspx

Figure 11-4 shows the `ValidationSummary` with `ShowMessageBox` set to `true`.