Передача информации между страницами

ASP.NET --- Основы ASP.NET --- Передача информации между страницами

Одним из самых серьезных ограничений состояния представления является его тесная связь с конкретной страницей. Когда пользователь переходит на другую страницу, эта информация утрачивается. Существует несколько решений данной проблемы и то, какое из них будет наилучшим, зависит от имеющихся требований.

Строка запроса

Наиболее распространенный подход заключается в передаче информации с помощью строки запроса в URL-адресе. Именно этот подход часто применяется в поисковых службах. Например, при выполнении поиска на веб-сайте Google пользователь перенаправляется на новый URL-адрес, который включает заданные условия поиска:

http://www.google.ru/search?q=organic+gardening

Строка запроса — это часть URL-адреса, которая находится после вопросительного знака. В данном случае она определяет единственную переменную по имени q, которая содержит строку organic+gardening.

Преимущество строки запроса в том, что она проста и не привносит никакой нагрузки на сервер. В отличие от межстраничной отправки, строка запроса может переносить одну и ту же информацию со страницы на страницу. Однако с ней связано несколько ограничений:

Информация ограничивается простыми строками, которые должны содержать символы, допустимые для URL-адресов.

Информация визуально доступна пользователю и любому другому работающему в Интернете человеку.

Предприимчивый пользователь может решить изменить строку запроса и предоставить новые значения, которые программа никак не ожидает получить, и от которых она не защищена.

Многие браузеры накладывают ограничения на длину URL-адреса (которая обычно не должна превышать 1-2 Кбайт). Поэтому в строку запроса нельзя помещать большой объем информации, к тому же все равно придется выполнять проверку на предмет совместимости с большинством браузеров.

Но все равно добавление информации в строку запроса является полезным приемом. Он особенно подходит для приложений баз данных, в которых пользователю отображается список элементов, соответствующих записям в базе данных (например, это может быть список товаров). Пользователь выбирает элемент, после чего он перенаправляется на другую страницу, содержащую детальную информацию о выбранном элементе.

Один из наиболее простых способов реализовать такое проектное решение — заставить первую страницу отправлять идентификатор элемента второй странице. Вторая страница затем ищет этот элемент в базе данных и отображает детальную информацию о нем. Эта технология часто применяется на сайтах электронной коммерции, например, Amazon.com.

Чтобы сохранить информацию в строке запроса, ее необходимо поместить туда самостоятельно. К сожалению, способа сделать это с помощью коллекций не существует. Как правило, это означает, что придется использовать специальный элемент управления HyperLink или оператор Response.Redirect(), подобный показанному ниже:

// Перейти на страницу page.aspx. Отправить единственный аргумент

// строки запроса по имени recordID и установить его в 10

int recordID = 10;

Response.Redirect("page.aspx?recordID=" + recordID.ToString());

Для отправки множества параметров они должны разделяться с помощью амперсанда (&):

Response.Redirect("page.aspx?recordID=10&mode=full");

Получающей странице легче работать со строкой запроса. Она может извлекать значения из словарной коллекции QueryString, предоставляемой встроенным объектом Request, как показано ниже:

string ID = Request.QueryString["recordID"];

Если строка запроса не содержит параметр recordID или если она содержит этот параметр, но без значения, строка идентификатора устанавливается в null. Обратите внимание, что информация всегда извлекается в виде строки, которая затем может быть преобразована в другой простой тип данных. Значения в коллекции QueryString индексируются по имени переменной.

К сожалению, ASP.NET не предоставляет механизма для автоматической проверки или шифрования данных строки запроса. Он мог бы работать в основном так же, как механизм защиты состояния представления. Без этой функциональности строка запроса очень уязвима.

URL-кодирование

Одной из возможных проблем со строкой запроса является присутствие в ней символов, которые применять в URL-адресе не допускается. Список символов, которые разрешено использовать в URL-адресе, намного короче списка символов, которые можно применять в HTML-документе. В целом, в URL-адресе могут использоваться только буквы, цифры или специальные символы (вроде $-\_.+!\*'(),). Некоторые браузеры допускают наличие в URL-адресе определенных дополнительных специальных символов (самым нестрогим в этом плане является Internet Explorer), но многие — все-таки нет.

Более того, некоторые символы имеют специальное значение. Например, символ & служит для разделения параметров в строке запроса, символ + является альтернативным способом для представления пробела, а символ # применяется для указания на конкретную закладку на веб-странице. Попытка отправить в строке запроса значения, включающие любой из этих символов, обязательно приведет к потере каких-нибудь данных.

Если есть подозрение, что сохраняемые в строке запроса данные могут включать недопустимые для URL-адреса символы, необходимо применить URL-кодирование. Когда используется URL-кодирование, специальные символы заменяются последовательностями преобразованных символов, начинающимися со знака процента (%), за которым следует двузначное шестнадцатеричное представление. Единственным исключением является символ пробела, который может заменяться как последовательностью символов %20, так и знаком +.

С помощью методов класса HttpServerUtility можно обеспечить автоматическое кодирование данных. Например, ниже показано, как закодировать строку произвольных данных для использования в строке запроса. Этот код просто заменяет все недопустимые символы последовательностями преобразованных символов:

string productName = "Video & Audio company";

Response.Redirect("page.aspx?product=" + Server.UrlEncode(productName));

Для возврата строке с закодированным URL первоначального значения можно использовать метод HttpServerUtility.UrlDecode(). Однако в случае строки запроса выполнять этот шаг нет необходимости, поскольку ASP.NET автоматически декодирует значения при получении к ним доступа через коллекцию Request.QueryString.

Обычно безопаснее вызывать метод UrlDecode() во второй раз, потому что декодирование данных, которые уже были декодированы, не вызывает проблем. Единственным исключением является наличие значения, которое обоснованно включает знак +. В таком случае вызов метода UrlDecode() приведет к преобразованию этого знака в символ пробела.

**Межстраничная обратная отправка**

Вы уже знаете, каким образом страницы ASP.NET выполняют обратную отправку самим себе. При такой обратной отправке текущее содержимое всех элементов управления страницы (вместе с содержимым скрытого поля состояния представления) отправляется в форме для этой же самой страницы.

Для передачи информации с одной страницы на другую можно использовать точно такой же механизм, но только отправлять данные другой странице. Этот прием выглядит довольно просто в теории, но на практике может иметь много "подводных камней". В случае несоблюдения мер предосторожности, все может закончиться получением страниц, слишком тесно связанных между собой и крайне неудобных для расширения и отладки.

Инфраструктурой, которая отвечает за поддержку обратной отправки данных между страницами, является свойство PostBackUrl. Это свойство определено интерфейсом IButtonControl и присутствует в кнопочных элементах управления, таких как ImageButton, LinkButton и Button. Чтобы использовать межстраничную обратную отправку, необходимо присвоить свойству PostBackUrl в качестве значения имя другой веб-формы. Когда пользователь щелкнет на кнопке, страница будет отправлена по этому новому URL-адресу вместе со всеми значениями, которые на текущий момент содержатся в ее элементах управления.

В показанном ниже фрагменте кода определяется форма с двумя текстовыми полями и кнопкой, которая отправляет данные странице по имени CrossPage2.aspx:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="CrossPage1.aspx.cs" Inherits="CrossPage1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head id="Head1" runat="server">

<title>CrossPage1</title>

</head>

<body>

<form id="form1" runat="server">

<div>

Введите данные:

<asp:TextBox runat="server" ID="txtFirstName"></asp:TextBox>

<asp:TextBox runat="server" ID="txtLastName"></asp:TextBox>

<asp:Button runat="server" ID="cmdSubmit" PostBackUrl="CrossPage2.aspx" Text="Отправить" />

</div>

</form>

</body>

</html>

Страница CrossPage2.aspx может взаимодействовать с объектами страницы CrossPagel.aspx, используя свойство PreviousPage. Например:

protected void Page\_Load(object sender, EventArgs e)

{

if (PreviousPage != null)

{

Label1.Text = "Заголовок предыдущей страницы " + PreviousPage.Header.Title;

}

}

Обратите внимание, что перед попыткой доступа к объекту PreviousPage эта страница проверяет ссылку на предмет null. Если объект PreviousPage не существует, межстраничная обратная отправка не выполняется.

Чтобы эта система работала, ASP.NET применяет интересный трюк. Когда вторая страница впервые пытается получить доступ к объекту Page.PreviousPage, среда ASP.NET должна создать объект предыдущей страницы. Для этого ASP.NET фактически запускает цикл обработки страницы, но прерывает его прямо перед началом этапа PreRender. По ходу дела ASP.NET также создает объект-заместитель HttpResponse, который молча перехватывает и игнорирует все поступающие с предыдущей страницы команды Response.Write().

Однако происходят еще некоторые интересные побочные эффекты. Например, срабатывают все события предыдущей страницы, включая Page.Load, Page.Init и даже Button.Click для кнопки, которая запустила обратную отправку (если она определена). Срабатывание этих событий является обязательным, т.к. они необходимы для корректной инициализации страницы.

Трассировочные сообщения, в отличие от сообщений Response, не игнорируются, а это значит, что при межстраничной обратной отправке можно просматривать трассировочную информацию с обеих страниц.

Получение специфической для страницы информации

В рассмотренном выше примере извлекавшаяся с предыдущей страницы информация ограничивалась только членами класса Page. Если нужно получить более специфические детали, такие как значения элементов управления, понадобится привести ссылку PreviousPage к соответствующему типу.

Ниже показан пример того, как это должно делаться. Сначала производится проверка, является ли объект PreviousPage экземпляром ожидаемого источника (CrossPage1):

CrossPage1 prevPage = PreviousPage as CrossPage1;

if (prevPage != null)

{

// Прочитать необходимую информацию с предыдущей страницы

}

В беспроектном веб-сайте Visual Studio может пометить такой код как ошибку, указывая на отсутствие информации о типе класса исходной страницы (в данном примере это CrossPage1). Однако после компиляции веб-сайта эта ошибка исчезает.

Эту проблему можно решить и другим способом. Вместо того чтобы вручную приводить ссылку к соответствующему типу, можно добавить в код страницы директиву элемента управления PreviousPageType, которая обозначает ожидаемый тип страницы, инициирующей межстраничную обратную отправку, например:

<%@ PreviousPageType VirtualPath="CrossPage1.aspx" %>

Однако такой подход является менее гибким, потому что предполагает использование единственного типа, не позволяя иметь дело с ситуациями, когда необходимо, чтобы межстраничную обратную отправку инициировало более одной страницы. Поэтому подход, предполагающий приведение к соответствующему типу, более предпочтителен.

Поскольку свойство PostBackUrl позволяет указывать только на одну страницу, может показаться, что межстраничную обратную отправку допускается устанавливать только между двумя страницами. Однако это не так — эта модель очень легко расширяется с помощью разнообразных технологий. Например, можно изменить свойство PostBackUrl программно, чтобы выбрать другую целевую страницу. Или наоборот сделать так, чтобы целевая страница в межстраничной обратной отправке данных могла проверять значение свойства PreviousPage и определять, относится ли оно к одному из нескольких классов. Затем в зависимости от того, какая страница инициировала межстраничную отправку, можно решать разные задачи.

После приведения предыдущей страницы к соответствующему типу получать напрямую доступ к значениям ее элементов управления по-прежнему не получится. Это связано с тем, что элементы управления объявлены как защищенные члены. Решить эту проблему можно, добавив в класс страницы свойства, являющиеся оболочками для переменных элементов управления, например:

public TextBox FirstNameTextBox

{

get { return txtFirstName; }

}

public TextBox LastNameTextBox

{

get { return txtLastName; }

}

Однако, как правило, такой подход не является наилучшим. Проблема в том, что он раскрывает слишком много деталей, тем самым предоставляя целевой странице возможность считывать свойства всех элементов управления. Если позже возникнет необходимость изменить страницу так, чтобы в ней использовались другие элементы управления, сохранить эти свойства будет очень трудно и поэтому, скорее всего, придется просто переписать код обеих страниц.

Более удачный вариант — определить специфические ограниченные методы или свойства, извлекающие только необходимую информацию. Например:

public string FullName

{

get { return txtFirstName.Text + " " + txtLastName.Text; }

}

При таком подходе отношения между страницами являются хорошо документированными и не представляют сложности для понимания. В случае изменения элементов управления на исходной странице интерфейс для общедоступных методов или свойств, скорее всего, удастся оставить тот же. Например, в случае изменения элементов управления, отвечающих за получение информации об имени в предыдущем примере, все равно понадобилось бы вносить поправки в код свойства FullName. Однако после внесения всех этих поправок в CrossPage1.aspx страницу CrossPage2.aspx изменять вообще не нужно.

В некоторых случаях вместо межстраничной обратной отправки лучше использовать разновидность элемента управления, имитирующего множество страниц или множество шагов, например, отдельные элементы управления Panel, Multiview или Wizard. Такие элементы управления позволяют обеспечивать для пользователя во многом точно такой же результат и упрощают модель кодирования.

Выполнение межстраничной отправки в любом обработчике событий

Как рассказывалось в предыдущем разделе, межстраничная отправка доступна только с элементами управления, которые реализуют интерфейс IButtonControl. Однако существует и обходной путь. Этот путь подразумевает использование перегруженной версии метода Server.Transfer() для переключения на новую ASP.NET-страницу, не затрагивая информацию состояния представления. Все, что требуется — это добавить в данный метод булевский параметр preserveForm и установить для него значение true, как показано ниже:

Server.Transfer("CrossPage2.aspx", true);

Такой подход дает возможность использовать межстраничную отправку в любом месте кода веб-страницы. Как и любой вызов метода Server.Transfer(), такой вызов приводит к переадресации на стороне сервера. Это означает, что дополнительного полного обхода для переадресации клиента не делается. Единственный недостаток в том, что в окне браузера пользователя остается отображаться URL-адрес исходной страницы несмотря на то, что был совершен переход на другую страницу.

Интересно то, что существует способ, позволяющий различать, когда межстраничная обратная отправка инициируется непосредственно через кнопку, а когда — через метод Server.Transfer(). Хотя доступ к объекту Page.PreviousPage возможен в обоих случаях, в ситуации, когда используется метод ServerTransfer(), свойство Page.PreviousPage.IsCrossPagePostBack имеет значение false. Ниже показан код, демонстрирующий, как работает эта логика:

if (PreviousPage == null)

{

// Страница была запрошена (или отправлена обратно) напрямую

}

else if (PreviousPage.IsCrossPagePostBack)

{

// Межстраничная обратная отправка была инициирована через кнопку

}

else

{

// Передача данных без изменения состояния была

// инициирована через метод Server.Transfer()

}

Межстраничная обратная отправка и проверка достоверности

При использовании технологии межстраничной обратной отправки с элементами управления проверкой достоверности возникает несколько сложностей. Как объяснялось ранее, когда применяются элементы управления проверкой достоверности, необходимо проверять свойство Page.IsValid, которое показывает, корректные ли данные ввел пользователь. Хотя возможность отправки пользователями недействительных страниц обратно серверу обычно исключается (за счет написания специального клиентского сценария JavaScript), это не всегда так.

Например, браузер клиента может просто не поддерживать JavaScript, или же злоумышленник может отыскать способ "обхода" всех выполняющихся на стороне клиента проверок достоверности.

Когда проверка достоверности используется в сценарии с межстраничной обратной отправкой, всегда существует вероятность возникновения каких-нибудь проблем. Давайте посмотрим, что происходит, когда применяется межстраничная обратная отправка, а исходная страница включает элементы управления проверкой достоверности. На рисунке ниже показан пример страницы, которая содержит элемент управления RequiredFieldValidator, требующий ввода в текстовом поле:

Обе кнопки на этой странице имеют свойство CausesValidation, установленное в true. В результате, если вы щелкнете на кнопке Межстраничная обратная отправка, клиентский код проверки достоверности в браузере не позволит это сделать. Вместо этого появится сообщение об ошибке.

Можно также посмотреть, что происходит, когда клиентский сценарий не поддерживается браузером, установив свойство RequiredFieldValidator.EnableClientScript в false. (После улучшения кода его следует снова установить в true.) Если теперь щелкнуть на одной из кнопок, страница будет отправлена обратно серверу и появится новая страница.

Чтобы исключить такое поведение, прежде чем выполнять любое другое действие, на целевой странице необходимо проверить достоверность исходной страницы с помощью свойства Page.IsValid. Это стандартный метод защиты, используемый в любой веб-форме, которая реализует проверку достоверности. Разница состоит лишь в том, что если данные на странице являются недействительными, просто ничего не делать будет недостаточно. Здесь придется выполнить еще один дополнительный шаг: вернуть пользователя на исходную страницу. Вот какой для этого понадобится код в целевой странице:

// Этот код находится в целевой странице

protected void Page\_Load(object sender, EventArgs e)

{

// Проверить достоверность предыдущей страницы

if (PreviousPage != null)

{

if (!PreviousPage.IsValid)

{

// Отобразить сообщение об ошибке

}

else

{

if (PreviousPage is CrossPage1)

Label1.Text = (PreviousPage as CrossPage1).FullName;

}

}

}

Приведенный код можно улучшить. Сейчас, когда пользователь возвращается на исходную страницу, сообщение об ошибке не появляется, потому что страница просто запрашивается повторно (а не отправляется обратно). Это можно исправить, установив флаг, который позволит исходной странице узнать, что данная страница была отклонена целевой страницей. Ниже показан фрагмент кода, который добавляет такой флаг в строку запроса:

if (!PreviousPage.IsValid)

{

Response.Redirect(Request.UrlReferrer.AbsolutePath + "?err=true");

}

Теперь исходная страница должна просто проверить наличие этого значения в строке запроса и соответствующим образом выполнить проверку достоверности. Для любых недопустимых данных отображаются сообщения об ошибках:

// Этот код находится в исходной странице

protected void Page\_Load(object sender, EventArgs e)

{

if (Request.QueryString["err"] != null)

Page.Validate();

}

Эту страницу можно продолжать улучшать и дальше. Например, если пользователь находится на середине процесса заполнения детализированной формы, повторный запрос страницы будет не очень хорошей идей, поскольку он очистит все элементы управления вводом, и пользователю придется начать все сначала, вместо этого лучше добавить в поток ответа небольшой JavaScript-код, выполняющий возврат на исходную страницу с помощью средства возврата самого браузера.

Этот пример демонстрирует, что межстраничные обратные отправки часто оказываются сложнее, чем поначалу ожидают разработчики. Неумелое их использование может приводить к получению тесно связанных между собой страниц с трудно обнаруживаемыми зависимостями, которые затрудняют изменение страниц в будущем. Поэтому прежде чем применять их в качестве метода для передачи информации, обязательно взвесьте все "за" и "против".

Alexandr Erohin ✯ alexerohinzzz@gmail.com © 2011 – 2017

Cookie

ASP.NET --- Основы ASP.NET --- Cookie

Специальные cookie-наборы — это еще один способ сохранения информации для последующего использования. Cookie-наборы представляют собой небольшие файлы, которые создаются на жестком диске клиента (или, если они являются временными, то в памяти веб-браузера). Одно из преимуществ cookie-наборов состоит в том, что они работают прозрачно, и пользователь даже не подозревает о том, что должна быть сохранена какая-то информация.

Они также могут применяться любой страницей в приложении и сохраняться между посещениями, что подразумевает действительно долгосрочное хранение. Они обладают некоторыми недостатками, которые присущи строкам запросов. В частности, они могут хранить только простую строковую информацию, и пользователь очень легко может получить к ним доступ и просмотреть их, отыскав и открыв соответствующий файл. Эти факторы делают их неподходящим вариантом, когда требуется сохранить сложную или секретную информацию либо просто большой объем данных.

Некоторые пользователи отключают cookie-наборы в своих браузерах, что приводит к появлению проблем при работе с веб-приложениями, которые требуют их. Однако в целом cookie-наборы являются распространенным явлением, т.к. применяются на очень многих сайтах.

Cookie-наборы просты в использовании. Оба объекта Request и Response (доступные через свойства объекта Page) предоставляют коллекцию Cookies. Важно запомнить, что извлекать cookie-наборы следует из объекта Request, а устанавливать — через объект Response.

Чтобы установить cookie-набор, просто создайте новый объект System.Web.HttpCookie. Затем заполните его строковой информацией (используя уже знакомый шаблон словаря) и присоедините его к текущему веб-ответу, как показано ниже:

// Создать объект cookie-набора

HttpCookie cookie = new HttpCookie("My localhost cookie");

// Установить значения в нем

cookie["Language"] = "Russian";

cookie["Country"] = "ru-ru";

// Добавить куки в ответ

Response.Cookies.Add(cookie);

Cookie-набор, добавленный таким способом, будет сохраняться до тех пор, пока пользователь не закроет окно браузера, и отправляться вместе с каждым запросом. Чтобы создать cookie-набор с более долгим сроком существования (который сохраняется вместе с временными файлами Интернета на жестком диске пользователя), установите для него дату истечения срока действия, как показано ниже:

// Этот cookie-набор будет оставаться

// действительным в течение одного года

cookie.Expires = DateTime.Now.AddYears(1);

Теперь можно увидеть информацию о добавленном cookie-наборе в браузере:

Извлекаются cookie-наборы по именам с помощью коллекции Request.Cookies:

HttpCookie cookieReq = Request.Cookies["My localhost cookie"];

// Проверить, удалось ли обнаружить cookie-набор с таким именем.

// Это хорошая мера предосторожности, потому что

// пользователь мог отключить поддержку cookie-наборов,

// в случае чего cookie-набор не существует

string language;

if (cookieReq != null)

{

language = cookieReq["Language"];

}

Единственный способ удалить cookie-набор — заменить его cookie-набором с истекшим сроком хранения. В следующем фрагменте кода показано, как это делать:

HttpCookie cookie = new HttpCookie("My localhost cookie");

cookie.Expires = DateTime.Now.AddDays(-1);

Response.Cookies.Add(cookie);

Вы заметите, что некоторые другие средства ASP.NET тоже используют cookie-наборы. Двумя примерами могут служить состояние сеанса (которое позволяет временно сохранять специфическую информацию о пользователе в памяти сервера) и безопасность с помощью форм (она дает возможность ограничить доступ к определенным разделам веб-сайта, заставляя пользователей получать к ним доступ через страницу входа).