ASP.NET: Everything on a page, or break into controls?

Asked 15 years, 9 months ago Modified 15 years, 8 months ago

Viewed 646 times

3

When you're making a page that is visually broken down into specific regions, do you break those regions down into controls, or do you put everything on the page and in one code-behind?

If regions are separated into controls, how do you facilitate communication among the controls? By communication I mean simply exchanging data on the server side.

Think of a page with a couple boxes above a rather complicated GridView. One box dictates what the GridView displays, the other box allows you to do various things with the GridView. Both boxes need to communicate with the grid, but not with each other.

The code-behind for this would be at least a couple thousand lines.

Any resources on making these types of design decisions would be helpful.

Thanks

asp.net architecture controls

Share

Improve this question

Follow

edited Apr 6, 2009 at 16:22

Mihai Limbășan

67.2k ● 4 ● 50 ● 59

asked Mar 16, 2009 at 18:55

MStodd

4,746 ● 3 ● 32 ● 50

7 Answers

Sorted by: Highest score (default)

4

For my first asp.net app, I used the separate into controls approach. However, I no longer do that and I now take the approach of just putting everything into one page. I would only use controls if you need to repeat the same part of the page and functionality across multiple pages. Reuse is really what controls are meant for.

If you are going to take the multiple controls approach, you can use properties, methods, and/or events on the controls to allow them to communicate with each other or with controls on the page.

Share Improve this answer

Follow

answered Mar 16, 2009 at 19:02

DCNYAM

12.1k ● 8 ● 56 ● 71

There are a few reasons you may use user controls, for example:

- **Modular code**: instead of one bloated page class, you can partition the code into several user controls, each with less code. Thus, if your page has too much code to manage, splitting it up into user controls could reduce the size of each class.
- **Re-Use**: if one control is used across several pages, or has several instances on one page, its much easier to build it once and then re-use it several times. Thus, if you have common UI components that are re-used in many places, user controls can help you.

Of course there is a price to pay, each control has to expose properties that allow its parent containers to control it, and it has to expose events to notify its parents of some event occurring. Parent controls can also expose similar events and properties, but this adds a dependency on the parent control in the user control which may complicate re-use of the control in other containers.

If you can break down many parts of your page into logical components that have clearly defined separation of responsibility, then they are ideal candidates for user controls. Its just like when you are defining classes, and yat some point a class may become too complex and you decide to partition it into several smaller classes, only here there is a higher cost for each new class you define.

In my experience, user controls are great, just don't break down your controls to so many levels of depth to the point where you've created more complexity than the original bloated page. Try to find a happy medium where controls do just enough, not too much, and you don't have too many controls.

Here are some tutorials on user control communication
http://www.codeproject.com/KB/user-controls/Page_UserControl.aspx
http://aspalliance.com/1461_Page_and_User_Control_Communication

Share Improve this answer

Follow

answered Mar 16, 2009 at 19:14

TJB
13.5k ● 4 ● 36 ● 47

1

Usercontrols facilitate reusability, and they incidentally break up functionality into smaller parts which can make the codebase more easy to manage. You can also do this by carefully planning out your code on one monolithic page as well, using regions and other organizational techniques.

In the past I've done both, and both can work. I find the best method of doing control-to-page communication is with events - i.e. given a page with BoxA (grid display,) BoxB (grid options) and Grid (a usercontrol with the gridview in it,) BoxA could define an event "DisplaySettingsChanged," which it would throw during a postback whenever its settings have changed. The

hosting page would create event subscriptions between the different components, for example catching DisplaySettingsChanged and calling Grid.Refresh or whatever.

For all the extra infrastructure, if you're sure you don't want to re-use any of these components anywhere, I would probably put it in one monolithic page with extra care taken to keep everything organized and readable.

Share Improve this answer

Follow

answered Mar 16, 2009 at 19:03

James Orr

5,105 ● 7 ● 43 ● 63

0

the controls should expose events and page should handle the same and update the grid view accordingly.

The breaking of control is purely based on need. if you want to reuse break them into controls. If its like a scrap project, there is no need for specific controls.

Share Improve this answer

Follow

answered Mar 16, 2009 at 19:01

Ramesh

13.3k ● 3 ● 54 ● 88

There are 2 routes that you can take -

Route 1: Get OK from User/Stakeholder/Management/UAT, extract components from the page for better manageability, reuse etc.

Route 2: Think reusable components right from the beginning. Test your components/user controls/ custom controls/ web parts. Mix and match your components on web page and send for approval.

For most times you should not have any problem going Route 1, specially if your requirements keep changing and you dont have much time. You will get an understanding of UI requirements and analyze if you break the page into components, will the components require to communicate directly with each other or thru a DB etc.

Route 2 is where you are sure if you have enough requirements to indicate that certain UI/Behavior can be reused, will require independent testing etc.

At times its not very easy to directly jump to route 2. If you have the domain experience, then sure.

Now in ASP.net world you even have to make a choice between Custom and User controls. Thats another decision you might have to make.

Hope this helps

Happy Coding!!

Share Improve this answer

Follow

answered Mar 16, 2009 at 19:11

Perpetualcoder

13.6k ● 9 ● 66 ● 101

0

I would only break into separate controls if the code is going to be reused. Otherwise it is just a waste of time. I also try to create server controls if the html isn't easily managed in design mode, like code that generates breadcrumbs or navigation dynamically. A login form might be a good example of a user control.

Share Improve this answer

Follow

answered Mar 16, 2009 at 19:18

Shawn

19.8k ● 20 ● 100 ● 153

0

I've worked on projects at both extremes, from all functionality piled into one page to all functions having a dedicated usercontrol. At the very least, I recommend doing the following:

- Site-wide design belongs in your masterpages
- Functionality which appears on two or more pages should be encapsulated in a user control. (A login control, for example, might appear in several locations.)
- Other functions is perfectly fine to embed in your aspx page.

The tricky part is getting communication right, which can be difficult when you have deeply nested layers of controls which need to communicate with one another, or when you have a control on a master page which needs to notify controls on a child page (if they exist). If you need communication between two or more levels of controls, use the following model:

```
                ProxyNotifier
            /                   \
   Control1                       Control2
```

In other words, Control1 and Control2 both have access to a ProxyNotifier instance. Control1 signals ProxyNotifer, and ProxyNotifer passes the signal to Control2.

A very simple implementation would look like this: (**CAUTION: untested code, not intended for production use**):

```
public static class ProxyNotifer
{
    // NOTE: This delegate should NOT be held in a sta
    // its session specific
    private static Action<string> NotifyEvent
    {
        get
        {
            if (!Session.ContainsKey["ProxyNotifyEvent
            {
                Session["ProxyNotifyEvent"] = null;
            }
            return Session["ProxyNotifyEvent"] as Acti
        }
    }
```

```
    public static void Subscribe(Action<string> handle
    {
        NotifyEvent += handler;
    }

    public static void Unsubscribe(Action<string> hand
    {
        NotifyEvent -= handler;
    }

    public static void Invoke(string msg)
    {
        if (NotifyEvent != null) { NotifyEvent(msg); }
    }
}


public class Control1 : UserControl
{
    void SomethingHappened()
    {
        ProxyNotifyer.Invoke("Hello world!");
    }
}


public class Control2 : UserControl
{
    public Control2()
    {
        ProxyNotifer.Subscribe(Respond);
    }

    // ALWAYS unregister your event when your control
    // otherwise weirdness happens
    public void Dispose()
    {
        ProxyNotifier.Unsubscript(Respond);
        base.Dispose();
    }

    public void Respond(string msg)
    {
        lblMsg.Text = msg;
```

```
    }
}
```

Now Control1 and Control2 can communicate no matter where they are on the page.

Share Improve this answer Follow

edited Mar 16, 2009 at 19:25

answered Mar 16, 2009 at 19:20

Juliet

81.4k ● 46 ● 199 ● 229