asp.net mvc related, mainly a refactor question

Asked 16 years ago Modified 16 years ago Viewed 325 times



can anyone think of a better way to do this?











```
return Url<MessageController>(c => c.Index("inbox",
1)).Redirect();
            else if (action == MemberMessageAction.ExportXml)
                return new MemberMessageDownload(Identity.ID, items,
MemberMessageDownloadType.Xml);
            else if (action == MemberMessageAction.ExportCsv)
                return new MemberMessageDownload(Identity.ID, items,
MemberMessageDownloadType.Csv);
            }
            else
            {
                return new MemberMessageDownload(Identity.ID, items,
MemberMessageDownloadType.Text);
        }
        else if (action == MemberMessageAction.Delete)
            for (int i = 0; i < messages.Length; i++)</pre>
            {
                foreach (int id in messages[i].Selected)
                {
                    MemberMessage message = MessageRepository.FetchByID(id);
                    if (message.Sender.ID == Identity.ID || message.Receiver.ID
== Identity.ID)
                    {
                        if (message.Sender.ID == Identity.ID)
                        {
                            message.SenderActive = false;
                        }
                        else
                         {
                             message.ReceiverActive = false;
                        }
                        message.Updated = DateTime.Now;
                        MessageRepository.Update(message);
                        if (message.SenderActive == false &&
message.ReceiverActive == false)
                         {
                             MessageRepository.Delete(message);
                        }
                    }
                }
            }
        }
        else
        {
            for (int i = 0; i < messages.Length; i++)</pre>
            {
                foreach (int id in messages[i].Selected)
                    MemberMessage message = MessageRepository.FetchByID(id);
                    if (message.Receiver.ID == Identity.ID)
```

```
if (action == MemberMessageAction.MarkRead)
                    {
                        message.ReceiverRead = true;
                    }
                    else
                    {
                        message.ReceiverRead = false;
                    }
                    message.Updated = DateTime.Now;
                    MessageRepository.Update(message);
                }
            }
        }
    }
    return Url<MessageController>(c => c.Index("inbox", 1)).Redirect();
}
```

asp.net-mvc

refactoring

Share
Improve this question
Follow





5 Answers

Sorted by:

I think you can also leverage the mvc framework for most of your code. Correct me if I'm wrong because I'm gonna make a few assumptions about your classes because I

Highest score (default)

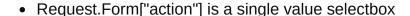


3

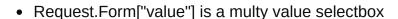
My assumptions:

can't deduct it from your post.











• action is the kind of action you want to be taken on all the messages



• message is the list of values that should go with the action

I would try to leverage the framework's functionality where possible

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult SaveMemberAction(SelectList selectedMessages,
MemberMessageAction actionType){
    //Refactors mentioned by others
}
```

If you then give your inputs in your Html the correct name (in my example that would be selectedMessages and actionType) the first few rules become unnessecary.

If the default modelBinder cannot help you, you might want to consider putting the parsing logic in a custom modelbinder. You can search SO for posts about it.

As a side note: you might want to reconsider your variable namings. "action" might be confusing with MVC's action (like in ActionResult) and MemberMessageSaveAction might look like it's a value of MemberMessageAction enum. Just a thought.

Share Improve this answer Follow

answered Dec 8, 2008 at 8:42 **Boris Callens**

93.2k • 86 • 210 • 308



The first step will be making different methods for each action.

Next is to remove the negative logic.



This results in something like this:

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult SaveAction() {
 // SNIP
 if (action == MemberMessageAction.Delete) {
    return DoDeleteAction(...);
 else if (action == MemberMessageAction.MoveToFolder) {
   return DoMoveToFolderAction(...);
 else if (action == MemberMessageAction.ExportXml) {
   return DoExportXmlAction(...);
 }
 else if (action == MemberMessageAction.ExportCsv) {
   return DoExportCsvAction(...);
 }
 else {
   return HandleUnknownAction(...);
 }
}
```

Share

edited Dec 8, 2008 at 8:00

answered Dec 8, 2008 at 7:49

Improve this answer

Toon Krijthe **53.4k** • 38 • 149 • 202

Follow

Why not use a switch together with the enum? Looks a lot cleaner, is more complete and offers a bunch of visual studio code suggestion goodness ("switch", tab, tab) - Boris Callens Dec 8, 2008 at 8:45



Turn MemberMessageAction into a class that has a Perform virtual function.

2

For your Special actions, group the common Perform code:



1

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult SaveAction()
    NameValueDeserializer value = new NameValueDeserializer();
    MemberMessageSaveAction[] messages =
(MemberMessageSaveAction[])value.Deserialize(Request.Form, "value",
typeof(MemberMessageSaveAction[]));
    MemberMessageAction action = MemberMessageAction.FromName(
        messages,
        Request.Form["action"]));
    return action.Perform();
}
class MoveToFolder : SpecialAction { /*...*/ }
class ExportXml : SpecialAction { /*...*/ }
class ExportCsv : SpecialAction { /*...*/ }
class Delete : MemberMessageAction { /*...*/ }
class MarkRead : MemberMessageAction { /*...*/ }
class MarkUnRead : MemberMessageAction { /*...*/ }
abstract class MemberMessageAction {
    protected MemberMessageSaveAction[] messages;
    public MemberMessageAction(MemberMessageSaveAction[] ms) { messages = ms; }
    public abstract ActionResult Perform();
    public static MemberMessageAction FromName(MemberMessageSaveAction[] ms,
string action) {
        // stupid code
        // return new Delete(ms);
    }
}
abstract class SpecialAction : MemberMessageAction {
    protected IList<MemberMessage> items;
    public SpecialAction(MemberMessageSaveAction[] ms) : base(ms) {
        // Build items
    }
}
```

Now you can easily factor the code.

Share edited Dec 9, 2008 at 2:08 answered Dec 8, 2008 at 7:58

Improve this answer

Follow

answered Dec 8, 2008 at 7:58

Frank Krueger

70.9k • 48 • 164 • 211



I don't like



this will make messages. Length (selected) queries to the database. I think you need to store your messages in ViewData, perform a filtering and pass them to Update() without the need to requery your database.



Share Improve this answer Follow

answered Dec 8, 2008 at 10:52





I came up with this.









1

```
[AcceptVerbs(HttpVerbs.Post)]
    public ActionResult Update(MemberMessageUpdate[] messages,
MemberMessage.Action action)
    {
        var actions = new List<MemberMessage.Action>
            MemberMessage.Action.MoveToFolder,
            MemberMessage.Action.ExportCsv,
            MemberMessage.Action.ExportText,
            MemberMessage.Action.ExportText
        };
        if (actions.Contains(action))
        {
            IList<MemberMessage> items = new List<MemberMessage>();
            for (var i = 0; i < messages.Length; i++)</pre>
            {
                if (messages[i].Selected == false)
                {
                    continue;
                }
                items.Add(MessageRepository.FetchByID(messages[i].ID));
            }
            if (action == MemberMessage.Action.MoveToFolder)
                var data = new MessageMoveViewData
                {
                    Messages = items
                };
                return View("move", data);
            }
            return new MessageDownloadResult(Identity.ID, items, action);
        }
        MessageRepository.Update(messages, action);
        return Url<MessageController>(c => c.Index(null, null, null,
null)).Redirect();
    }
```



You can make the filling of the items more terse like this: items = messages.where(m=>Selected) .Select(MessageRepository.FetchByID(m.ID) .ToList(); But make sure to check the post about the X database connections made by petar.petrov

- Boris Callens Dec 8, 2008 at 11:11