StrutsWhyOnlyOneInstanceOfActionClass

Weigen Liang:

The current Struts framework creates only one instance for each Action class, and uses that instance to serve all the relevant requests. This makes the Action class thread safety a big issue for newer users of the Struts framework. The struts user mailing list has many, many such questions from Struts users. This is similar to servlet itself. The servlet spec has the SingleThreadModel to "try" to solve this problem, but as we know, this does not really work.

If one instance of Action object is created for EACH request, then we would not have this threading issue. This instance would be tied to a request object and a response object, and has life span of that particular request. The class would be like this:

```
public class Action {
  public Action(ActionMapping mapping, ActionForm form,
     ServletRequest request, ServletResponse response) {
     ...
  }

  public ActionForward execute() {
     ...
  }

  private ActionMapping mapping;
  private ActionForm form;
  private ServletRequest request;
  private ServletResponse response;

  // plus other instance variable for this particular request
  ...
}
```

Considering that a significant percentage of Struts users have limited understanding of thread safety, removal of such a concern from the framework would certainly help the users.

Jacob Hookom:

Actions follow the same constraints as the Servlet API such with thread safety-- see HttpServlet (not to mention that I believe they are getting rid of the single thread HttpServlet model all together with the API).

Andrew Hill:

The main argument against this is that it is inefficient and will result in more object creation - taking time and memory.

Personally I dont think that one extra object per request would be such a big deal - I mean its already created an action form instance, gone through all the effort of poulating it etc... One more new Action object probably wouldnt hurt too much... (and its pretty insignificant compared to all the tons of objects that get created and garbage collected inside one of my actions!...)

And like you point out in your example, it would make the signature a lot simpler for the execute method. Furthermore one could add extra setters for custom stuff that could be called from a custom request processor.

That said - the efficiency argument does still stand - and I cant see the struts developers being ready to make such a big break with the current design. There is also a certain truth in the argument that things are simplified for developers by following the same technique as the servlet api itself.

TedHusted:

In the beginning, object creates were expensive, and the framework was being parsimonious. Using Action this way also made it easy to refactor working servlets into an Action. But, those days are past.

Moving forward, IMHO, the framework should support a wider range of action paradigms. We should also support creating actions using scripts, rather than just Java classes. JPublish supports this directly, and there is already a Struts extension along these lines.

This sounds scary, but there's really no reason why JSPs could not be used as actions (so long as they did not contain any display code).

The use case for scripts and JSPs would be to open the framework to non-Java programmers. Right now, to build a Struts web app, there's a lot of overhead involved just in getting a Java compiler, Ant, and an editor/IDE up and running.

Another approach entirely would be to combine ActionForm and Actions into a single object that would collect and validate its own data and then execute itself. The use case here would be to permit better object orientation to complex applications.

Now, I'm not suggesting that the current approach be modified. I'm suggesting that we increase the number of alternative approaches.

I think the trick to making this sort of thing work lies in defining our use cases and designing the configuration file to suit our definitions. We do have a very good design now, but for Struts 2.0, we might look at making it more generic and allowing for a broader range of functionality.

I've posted some ideas I've been kicking around for awhile on the Wiki.

http://nagoya.apache.org/wiki/apachewiki.cgi?StrutsUseCases

http://nagoya.apache.org/wiki/apachewiki.cgi?ModelingNotes

Craig R. McClanahan:

Changing this now would be a major backwards incompatibility issue, so it will not be done in any Struts 1.x release. We can look at other approaches in a 2.x time frame.

Per-request object instantiation is cheaper in 1.4 JDKs, but it's still not free. However, lots of Struts users still run on 1.2 and 1.3 JDKs where this is still a pretty big issue.

```
> Considering that a significant percentage of Struts users have limited
> understanding of thread safety, removal of such a concern from the
> framework would certainly help the users.
```

Yet the large majority of those people seem to figure it out after a while
– especially the single most important rule (do not store per-request state information in instance variables of an Action). As others have pointed out, the Servlet API imposes exactly the same sort of issue; it wasn't invented here.

Bradley Hill:

Indeed, the multi-threaded use of Actions is much like the Servlet API, but most Struts users (not developers) at this point are probably more familiar with the JSP APIs than with working directly with "pure" Servlet apps.

Why not use the lifecycle model of a JSP tag for Actions? Pool them for performance (if it's even worth the trouble), include a reset/cleanup method to be called by the framework, and guarantee single-threaded access within the scope of a request.

It would be an equally familiar and easy to explain by analogy model for most users and avoid the issues surrounding thread safety which, if in practice are not actually so complicated, many find intimidating and unintuitive.

David Graham:

Indeed, the multi-threaded use of Actions is much like the Servlet API, but most Struts users (not developers) at this point are probably more familiar with the JSP APIs than with working directly with "pure" Servlet apps.

```
> Why not use the lifecycle model of a JSP tag for Actions?
```

Because this is much more difficult to understand than how Struts currently is. Look at the number of bugs reported against custom tags that deal with lifecycle issues for proof.

```
> Pool them for
> performance (if it's even worth the trouble), include a reset/cleanup method to
> be called by the framework, and guarantee single-threaded access within the
> scope of a request.
```

IMO, this is not worth the trouble. Singleton Actions are fast, easy to code, and easy for the Struts team to maintain the implementation.

```
> It would be an equally familiar and easy to explain by analogy model for most
> users and avoid the issues surrounding thread safety which, if in practice are
> not actually so complicated, many find intimidating and unintuitive.
```

Coding Struts Actions should be done by programmers. Java programmers should know (or at least be willing to learn) threading rules. It's extremely simple to explain to people that they shouldn't use instance variables in their actions.

Craig R. McClanahan:

```
> Indeed, the multi-threaded use of Actions is much like the Servlet API,
> but most Struts users (not developers) at this point are probably more
> familiar with the JSP APIs than with working directly with "pure"
> Servlet apps.
```

None of this is relevant for 1.x anyway, but I would counter this with the observation that people who don't understand the fundamentals of the APIs they are using are going to run into trouble no matter how "simple" we try to make them.

Case in point - sessions can be accessed from multiple threads simultaneously, and it's ridiculously easy to cause this to happen. It doesn't matter whether you are used to the JSP or the Servlet APIs for accessing session attributes – if you don't understand this, your app is going to fail in mysterious ways.

There are lots more similar issues that servlet and JSP inherit simply because they run on top of a stateless protocol (HTTP). You cannot write robust web-based applications without understanding them – papering over the problems with simpler apis makes webapp development more approachable, but it cannot hide the fundamental ugliness of HTTP.

```
> Why not use the lifecycle model of a JSP tag for Actions?  Pool them for
> performance (if it's even worth the trouble), include a reset/cleanup method to
> be called by the framework, and guarantee single-threaded access within the
> scope of a request.
```

JSP 1.2 taught the world that the lifecycle model of custom tag instances was horribly hard to use, and even more difficult to program for than the thread-safe single instance model.

It is so bad that, in JSP 2.0, a new API (SimpleTag) was introduced that (among other things) dispenses with all the pooling complexity and just creates instances every time.

```
> It would be an equally familiar and easy to explain by analogy model for
> most users and avoid the issues surrounding thread safety which, if in
> practice are not actually so complicated, many find intimidating and
> unintuitive.
>
```

Strong advice from having lived through tags and pooling – don't go there
-).

Leonardo Quijano:

And why is so expensive to create an action in each request? How many objects are created on each request for a typical web app? And what's expensive about an Action? AFAIK, an Action is a object with a lot of static constants and one reference to the ActionServlet... a String could be more expensive than that!

As for the user-implemented subclasses, they are lightweight too, because we currently can't use instance variables - you can't just say "be careful with thread safety" and expect programmers to behave, life is not that way. If we could have those variables, that's the same with every other object... a heavy one is a bad one.

I have to admit that, besides of the thread safety issue, I don't see much gain in this change (and as a matter of fact I'd prefer an interface - it just makes testing so easier...!). But it *is* a safer design...

Brendan:

The advantages to the "action instance per request" approach are:

- The current approach means that users must be specifically cautioned about

the issues for *actions*.

- Instance variables can be used by both the framework and users to shorten

parameter lists etc.

The disadvantage to this new alternative approach is that it is different from the current struts model.

If a mechanism can be invented so that new isolated actions are clearly distinguished from singleton actions to provide backward compatablity, for humans and source, then the change can be made.

I personally like instance variables, and I like not having to worry about shared variables when I don't have too.

David Graham:

You can still use instance variables but you have to do it in a thread safe way. If you have many instance variables then you have too much logic in the actions. I have coded many (maybe all) of my actions without using one instance variable.

Implementing both singleton and instance-per-request actions is not worth the trouble.

Brandon Goodin:

Something else to consider...

Within the strut-config, the action element has the set-property capability. But, it doesn't really set the property of the action. instead it sets the property of a custom ActionMapping class which your Action uses. Two classes for the job of one. This is especially confusing to new users. Setting default parameters via the config works great for several scenarios. I use the custom ActionMapping for several general use Actions. If we were able to get away from the Singelton action it would allow for the elimination of the custom ActionMapping class just to set properties for a specific action to use.

I think the key point in all this discussion is productivity. Granted, any java developer should understand the general constructs of an MVC framework (or Model 2 framework, for all you achronymbaphobiacs) . But, as time marches on the efficiency we gain through technological maturity should be taken advantage of to reduce the learning curve.

Of course... this is a Struts 2.0 thing and not 1.x.

Margaritis Jason:

Kinda new to struts, but wouldn't it be trivial to implement this on top of the current framework by having your Action instantiate a new Object (could be an Action) and go from there? But i don't really see how instance variables are going to shorten parameter lists.

Edgar Dollin:

Since everyone is weighing in on this, I'll add my two cents.

First off, the only 'real' problems people have with struts (excluding bugs) is the mindset of the first application. The problems that I see are

1) Initial form load 2) Getting used to the tags 3) Getting used to reset / validation / validator 4) Triing to force their business logic into action forms.

By then if they have had a problem with thread safety, they have already worked through it during the initial confusion stage anyway.

In my opinion, not worth the effort.

Jason Miller:

I'll throw in one thought, as well.

Having multiple instances of Actions doesn't really allow much anyway, since ALL parts of a web app have to be reentrant. By shifting the thread safety issues down one level, I don't see much being gained. Particularly in light of the fact that Actions should be fairly lightweight, anyway.

David Graham wrote:

> Why should we reduce dependency on the Servlet API? Servlets are the Java
> standard web application technology. Struts is a web tier MVC framework
> enabling fast application development.

Craig R. McClanahan:

One good reason is the upcoming Portlet API Spec (JSR-168), which is in Community Review in the JCP process right now – so public review is probably not far down the road. I would like to make it a strong goal for Struts 2.0 that one be able to leverage Struts in either a servlet or portlet environment, with minimal-to-no changes in your business logic and pages.

As you'll see when Portlet goes public, this is going to require some decoupling from the direct linkage to servlet apis.

Vic Cekvenich:

In order to wrap action around non servlet signatures (execute(req,resp, formbean, mapping) I am using an event object. This lets same signature work on tileaction, and other action. It works fine, the event object contains the req,resp, formbean,mapping so any signature will work, and simpler to dispatch.

Also, A while back a group wanted "Struts" for SOAP, not sure what happened to them but...

I have started going into XML/RPC world, looking for Rich UI View. I followed X-Forms around and.... I re-discovered action script (based on ECMA script as Javascipt) that has good plug in support for browsers, since I want UI to use browser side processing for UI and ofload processing from server.

Sounds hard?

We'll "my" beans are list backed, so easy to write a base bean reflection (find getters) that generates W3 XML Doc using JDOM. So all my baseBeans have getXMLDOC() that returns a list of rows in XML. (this should realy go in Beantuils) Already done, tested in CVS.

Step 2: ActionSciprt/openSWF (yes, Flash, but no splash screen) knows XML RPC, and the new Fire Fly (data connection kit) has $0 run time costs.

So today in basicPortal cvs I have flash calling bean.populate(); bean.getXMLDoc(); and displaying in Fire Fly. (what a name) There are hundreds of Flash GUI controls out there 3rd party. Fire Fly has live preview.

It's not portlet API, more XML/RPC, but now with $0 run time costs, under apache license I can have GUI that executes on client and looks like this:

http://examples.macromedia.com/petmarket/flashstore.html

(try to order something, looks nice).

So the formbean can have getXMLDoc(), called by FireFly. This is done. My next move is to have Validator.xml emit action script! Moving Struts to Fire Fly will be apache license, $0 run time costs (yes you may have to buy the IDE from Macromedia but..... swf is open source, and editing action script is just text editor).

Back full circle, Today I am doing sample app that does not use Servlet but uses formbean and a single JSP that just has flash on it. I would be happy to share and welcome contributions. I am writing my 2nd book, 20 chapters, one chapter is Rich GUI to form beans and is a demo I do when I teach Advanced Struts.

There was a thread that said growth will be in the "Soap" like services and I agree with them, to that end my beans will be callable via SOAP. (good things "my" beans have a DAO helper that has nothing to do with action). So basicPortal in futre will demo how to go from Struts JSP to Struts Flash, under Apache license.

Just FYI, there is open source implemenations code out there already.

Brendan Johnston:

The struts designers clearly do not agree that actions should be single use. They are not single use now.

This shows that Struts designers have different tastes to me. I have yet to see one advantage of singleton actions.

What advantage of singletons am I not getting?

I have responded to the other comments. But this seems like a useless distraction if Singleton actions have zero advantages.

All parts of a web application do not have to be reentrant. ActionForms do not have to be reentrant. In general re-entrant code is very hard to test and therefore likely to be buggy.
I would suggest that all part of a web application that are written by applications prgrammers should not be re-entrant.

To avoid the problem Craig mentioned with session, one solution is to insert a standard object whenever you create a new session,
and in your request processor / wib action server / some filter, synchronize on that object. I think I might implement that.

All the actions I have written have no instance variables. That's because instance variables would be useless/dangerous because they are shared by all requests.

However OO provides instance variables and they are useful in my designs. When I look at actions, I see functional decomposition, not object oriented designs.

Instantiating another object to get around this sometimes does not work well with subclassed actions.

For an example of a shorter parameter list, see the wiki posted by Ted. http://nagoya.apache.org/wiki/apachewiki.cgi?WhyOnlyOneInstanceOfActionClass

Based on road map I don't agree/understand why this is a struts 2.0 issue. There is no need to break backward compatability.

David Graham:

Using a Singleton is faster than creating a new object for each request, especially for high traffic applications. If you need instance variables then you likely have too much logic in your action that should go into a separate layer.

There is nothing preventing you from using instance variables in actions but you must do it in a thread safe manner. The easiest solution is to just not use them.

It's trivial to override RequestProcessor.processActionCreate to return a new Action for each request so Struts doesn't even have to support this by default.

—

Weigen:

I disagree withthe downplay of thread safety issue. IMHO in a web application thread safety is more critical than in many other multi-user environments due to the fact that system load is horribly unpredictable for web applications. If thread safety is not kept in one's mind ALL THE TIME when one designs and codes the web application, I guarantee that when the web application has to process hundreds of orders per minute, the architects/programmers will be drown in boss and customers' angry voice/emails of "why I am get charged for things I did not order". If one does not pay enough attention to threading issue during design/development, one usually will find it out when the system is tested/used under load near the end of the development circle. And that "small" bug somewhere will cost one many, many frustrating days to track down, since everyone in the team swears that "when I tested it on my machine, it worked". Granted for a small and less demanding web application, this may not be as prominent as drawing up pretty pages.

Couple of days ago I brought up the thread safety issue after I was kind of surprised by the large numbers of struts users grappling this very issue on the struts user mailing list. Since I have to live with threading issue all the time, and have to educate my team constantly about the issue. I'm in sympathy with them. If a framework can relieve some of that burden from its users, it's well worth the effort.

David Graham:

Coding thread safe Struts actions is extremely easy, just don't use instance variables (there's probably something I forgot but this is the biggest pitfall). If you want to make it even easier, override RequestProcessor.processActionCreate to return a new Action instance for each request.

Craig McClanahan:

> The struts designers clearly do not agree that actions
> should be single use. They are not single use now.

The real key is whether your business logic is embedded in actions or not. If it is, I would suggest that you are making a mistake.

If you want per-request instantiation of business logic objects (so that you can use instance variables), the appropriate design is to do that inside an Action. Properly designed, these business objects should not have any linkage to servlet or Struts APIs – they should be configured simply by property beans setters or parameters to the constructor. You can easily provide per-request instantiation as a service in your Action as well if you want.

If the single-instance nature of Action gets in your way, that is a pretty good clue that you aren't factoring your application correctly.

> This shows that Struts designers have different tastes to me.
> I have yet to see one advantage of singleton actions.
> What advantage of singletons am I not getting?

Struts gives you mapping of logical paths to instances of an Action class where the caller doesn't have to know the name of the Java class, or worry about instantiating it. Nothing you couldn't do for yourself with singletons, but you would need to replicate this logic.

> I have responded to the other comments.
> But this seems like a useless distraction
> if Singleton actions have zero advantages.
> All parts of a web application do not have to be reentrant.
> ActionForms do not have to be reentrant.

Absolutely not true if the form bean is in session scope.

> In general re-entrant code is very hard to test and therefore likely to be
> buggy.
> I would suggest that all part of a web application
> that are written by applications prgrammers should not be re-entrant.

For business logic developers, you can make a good case for this –
indeed, I would go further and say these classes should not depend on struts apis either.

> To avoid the problem Craig mentioned with session,
> one solution is to insert a standard object whenever you create a new
> session,
> and in your request processor / wib action server / some filter, synchronize
> on that object. I think I might implement that.

Feel free – but I wish you'd also look at the benefits of decoupling a little bit first. Think of a Struts action as an adapter between the web tier and the business logic tier, not as a container for business logic.

> All the actions I have written have no instance variables.
> That's because instance variables would be useless/dangerous
> because they are shared by all requests.
>
> However OO provides instance variables and they are useful in my designs.
> When I look at actions, I see functional decomposition,
> not object oriented designs.
>

So use 'em ... just not in Actions 🙂.

> Instantiating another object to get around this sometimes does not
> work well with subclassed actions.
>
> For an example of a shorter parameter list, see the wiki posted by Ted.
> http://nagoya.apache.org/wiki/apachewiki.cgi?WhyOnlyOneInstanceOfActionClass
>
> Based on road map I don't agree/understand why this is a struts 2.0 issue.
> There is no need to break backward compatability.

Changing Action itself would break backwards compatibility for people who, based on the promise that there is one instance of an Action, do expensive one-time setup operations when the Action instance is created.

But the right answer to getting what you want in Struts 1.x is to provide your own Action that implements per-request instantiation of your business objects. You don't need the framework to provide that for you.

Note that it's certainly feasible to add this service into a later Struts 1.x release – but it will NOT be done by making Action into a per-request-instantiated object.

Phil Steitz:

> Using a Singleton is faster than creating a new object for each request,
> especially for high traffic applications.

Memory and speed of garbage collection can also be an issue when creating excessive objects per request in high volume applications.

Struts is doing us a *big* favor by maintaining singletons so we don't have to worry about wasting resources on what should be reusable object instances (since as has been repeated several times on this thread, Actions should just be stateless adaptors that delegate to business objects). The fact that object creation and garbage collection are faster/more efficient in newer jvm's does not mean that we should needlessly waste resources.

Hajime Ohtoshi wrote:
> The real key is whether your business logic is embedded in actions or not.
> If it is, I would suggest that you are making a mistake.
> Then, we had better define business logic bean as framework.
> How do you think about it?

Ted Husted:

I think that's a good idea, but it may be another framework entirely.

I started to do some work on a generic Business Request/Business Service framework. What you end up with something that looks just like Struts but without the web semantics. (Good patterns are good patterns!)

Ideally, I believe there should be a business layer Controller framework, and then Struts becomes a web specialization of that framework. This would drop right in with what we've been doing with things like the Commons Validator. Both the business layer Controller and the Struts Controller could share the same set of validations. Ditto for MessageResources and the rest of it.

But, the problem for me, is that I'm just writing web applications now, and my clients are not interested in anything but web applications. I like the idea of layered architectures, and leaving my options open, but it's hard to justify re-writing Struts on the business layer when I don't have a use case of my own = 😞

At this point, I'm defining a business interfaces and then implementing the interfaces as Actions and ActionForms. So, instead of having the Action instantiate a business object, I have a subclass that instantiates the business method, and the Action just "calls itself".

Since the business method doesn't use web semantics, I could easily refactor it into a separate object if need be, but until then, I save the extra machinery.

With the ActionForms, there's the old issue of String-based properties. But I decided not to care. If they put letters in a numeric field, and it ends up blank, no one really notices.

(Though, I'd really like to follow up on the idea of using the request as a data-entry buffer. If the ActionForm property is null, the page could check for a request parameter.)

The one side effect is that my unit tests for the business interfaces have to import and instantiate Actions and ActionForms. This isn't a real problem, but it complicates the builds a bit. The alternative would be to do a business implementation of the interfaces too, but, without a usecase, that would just be busy work.
