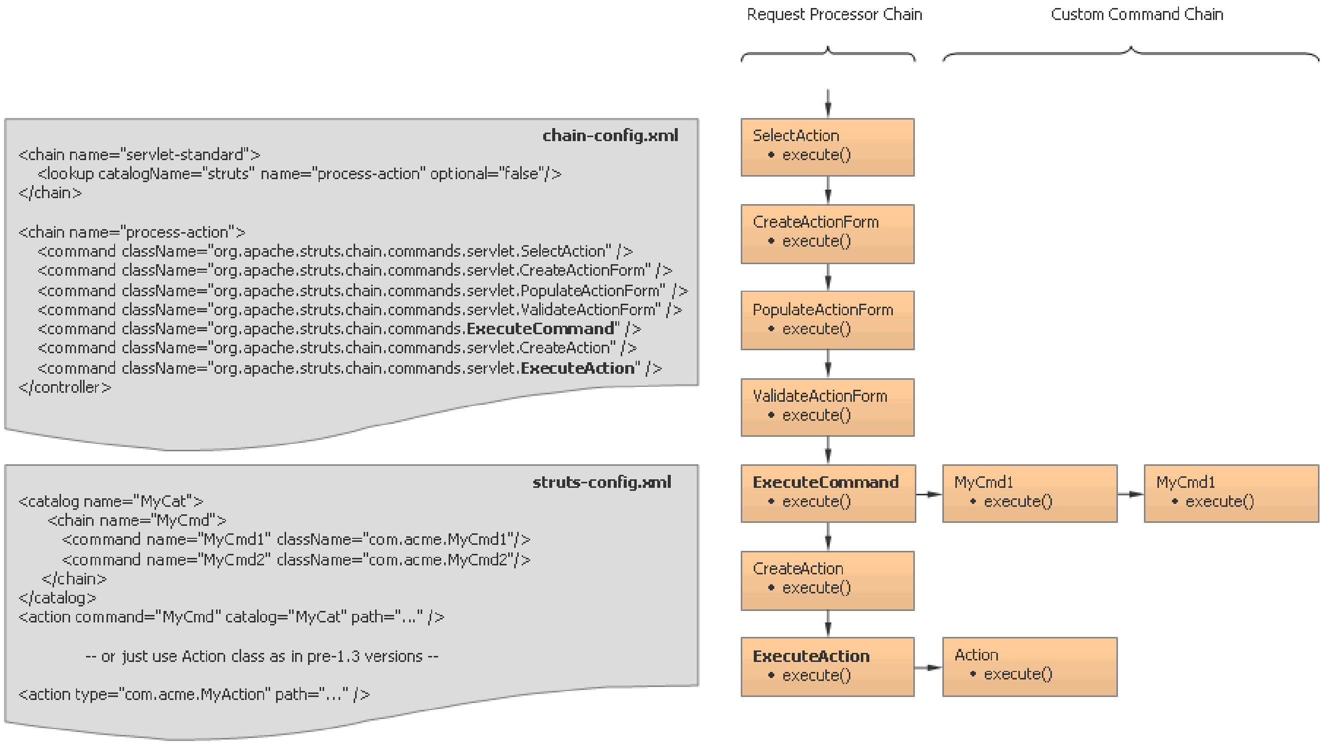
ComposableRequestProcessor

For Struts 1.3, the RequestProcessor methods has been turned into Commands. Rather than subclassing a monolithic object, it is possible not just to replace Commands. It is also possible to insert or remove Commands, if needed, to extend or streamline the request processing gauntlet.

The RequestProcessor class has been turned into ComposableRequestProcessor that invokes Command in the proper order. This order is defined with standard `chain-config.xml` file buried deep in Struts library, but can be easily overriden with custom config file if needed. To do this one need to define "chainConfig" property for ActionServlet in `web.xml` file. This propertly should contain comma-separated list of either context-relative or classloader path (s) to load commons-chain catalog definitions from. If none specified, the default Struts catalog that is provided with Struts will be used.



chain-config.xml

```
<chain name="servlet-standard">
   <lookup catalogName="struts" name="process-action" optional="false"/>
</chain>

<chain name="process-action">
   <command className="org.apache.struts.chain.commands.servlet.SelectAction" />
   <command className="org.apache.struts.chain.commands.servlet.CreateActionForm" />
   <command className="org.apache.struts.chain.commands.servlet.PopulateActionForm" />
   <command className="org.apache.struts.chain.commands.servlet.ValidateActionForm" />
   <command className="org.apache.struts.chain.commands.ExecuteCommand" />
   <command className="org.apache.struts.chain.commands.servlet.CreateAction" />
   <command className="org.apache.struts.chain.commands.servlet.ExecuteAction" />
</controller>
```

struts-config.xml

```
<catalog name="MyCat">
     <chain name="MyCmd">
        <command name="MyCmd1" className="com.acme.MyCmd1"/>
        <command name="MyCmd2" className="com.acme.MyCmd2"/>
    </chain>
</catalog>
<action command="MyCmd" catalog="MyCat" path="..." />

           -- or just use Action class as in pre-1.3 versions --

<action type="com.acme.MyAction" path="..." />
```

Refactoring the RequestProcessor for Chain of Responsibility (CoR) is about extending the request processing cycle using Commons Chain. It's been a straight-line refactoring since the beginning. I don't believe anyone is trying to follow a particular pattern. We are just trying to provide a flexible approach to extend the request processing cycle.

Meanwhile, users are forever trying to "chain" Actions. Since most teams do not have a coherent business application framework of their own, they tend to piggyback one on Struts. The input and outputs to their business logic become embedded in Actions, and so they want to chain a "copy" and a "delete" to do a "move".

Another use of Commons Chain is that people can use it as the base of their own business application framework (and stop misusing Struts Actions).

The idea behind CoR and Struts Chain/Commons Chain is to solve both of these problems:

- A flexible processing layer for business applications
- An extensible request processor for Struts

The refactoring of the request processor is a proof-of-concept for CoR. The request processing gauntlet is the "business logic" of the Struts framework. It's also a hoary example of some very nasty business logic. IMHO, if we can do the Struts request processor in CoR, we can do anything in CoR

todo

Beginning with Struts 1.3.0, the original RequestProcessor class has been extended in a way that substantially overhauls the way in which a Struts module processes an HTTP Request. Before this version of Struts, the RequestProcessor class implemented a variation of the Template pattern, in that it articulated a primary "process" method which called various protected methods in a reliable order. Users who wanted to alter the request processing lifecycle were advised to extend RequestProcessor and override one or more of those methods.

This led to a very inflexible environment, where it was very difficult to synthesize a request processing chain from library code, because of the familiar problems with "single inheritance" in Java. With the introduction of the ComposableRequestProcessor class, every request is handed to a composable chain of commands, each of which can perform a small bit of the request processing.

This is done by overriding the central "process" method, which means that earlier RequestProcessor classes which relied on that method's implementation to call other methods in a predictable fashion will no longer work. It should be extremely easy to extract that kind of behavior and put it into classes implementing the ActionCommand interface, while documenting for users how they should modify their chain configurations to insert these new commands in the right position in the chain so that any necessary context has been established.

A discussion on StrutsUpgradeNotes12to13 brought up the point that perhaps the responsibility for initializing the chains should be based in the ComposableRequestProcessor, where currently it is done by the ActionServlet. This generally makes sense, but a few things must be considered. First and more trivial, the initialization process relies on some simple resource-loading code which is difficult to extract from the ActionServlet. (Difficult, but certainly not impossible.) More fundamentally, though, the fact that one Struts application may have more than one ComposableRequestProcessor must be considered. The current chain initialization process leverages code from Commons Chain which populates a "global" (to a ClassLoader) CatalogFactory based on static methods. Multiple overlapping initialization processes might cause unexpected effects.

It would probably be worth revisiting that initialization strategy anyway, because it does not cooperate well with a DependencyInjection approach.
It might be better to have the chain assembled by a mechanism which could also inject dependencies like business service objects into various commands. This deserves more consideration.

Other notes on the ComposableRequestProcessor are welcomed!

Ted Husted made a pretty good case for a composable chain of processors on the mailing list. http://marc.theaimsgroup.com/?l=struts-dev&m=105472470724762&w=2

Other links:

http://www.onjava.com/pub/a/onjava/2005/03/02/commonchains.html

http://www.onjava.com/pub/a/onjava/2005/03/02/commonchains2.html

http://www.infonoia.com/en/content.jsp?d=inf.05.06&pr=1