Struts

Table of contents

Note: Just in case it's not clear, this page is **not** an 'official' view/position of the Wicket-Developers and at best, represents the author's (whosoever that may have been) point of view.

Comparing Wicket and Struts

Overview

Wicket is an advanced component based web application framework. Its primary benefits are:

- Clean separation of concerns between HTML and Java
- Object-oriented component model
- Automated state management
- High productivity
- Low learning curve
- Abstraction away from Servlet API and HTTP protocol details
- No XML configuration files
- Easy to build reusable components

Struts is a Model2 MVC web application framework. It is based around action classes that handle HTTP requests. Configuration is by XML files. Struts is usually used in conjunction with another technology (usually JSP and custom tag libraries).

The remainder of this article addresses the main differences between the two frameworks and explains why Wicket is so much better than Struts (in our opinion).

General Philosophy

The philosophy adopted by Struts is based around intercepting each HTTP request to a web application and directing it to a specific Action class that handles the request. Each action class then returns a result that informs struts of what to do next. This could be forwarding or redirecting to another action or passing control to a JSP page in order to output HTML. This approach is now rather outdated, for the following two main reasons:

- It is not particularly object-oriented. Yes, each Action class defines an abstraction, but the abstraction is determined by the HTTP procotol request mechanism rather than by an object-oriented analysis.
- Unless you output HTML directly from your Java code (bad practice) you still need to learn another major technology (usually JSP plus custom tag libraries) in order to produce output HTML pages. JSP pages with custom tags are notoriously difficult to maintain and edit, especially if you want you graphic design team to do this work.

Wicket's approach is very different. It adopts a very clean object-oriented and component based approach (much more like Swing). Each page in Wicket is a collection of components (built using the Composite design pattern). Pages and components take care of rendering themselves, either directly or via an associated markup file. When HTTP requests arrive these are converted into events that are invoked on individual components. Wicket therefore solves the two main problems with the struts philosophy as follows:

- Wicket is fully object-oriented. You work with hierarchies of components and design your application as such. There is no need to bend your oo design to fit with the request-response nature of the HTTP protocol.
- Wicket markup files are as close to pure HTML as it is possible to get. Where Wicket does introduce content into the markup it is very clean and complies with all XHTML standards (via the wicket namespace). Anyone who knows HTML can work with and edit Wicket markup files regardless of whether they know anything other than HTML or not.

HTTP Request Handling

In Struts, an HTTP request is received. Struts looks up the request path in its config file and finds the associated Action class. If configured it then extracts the request parameters into an ActionForm bean and performs some validation. The HTTP request, response and the ActionForm are then passed to the Action class. From this point on it is up to the action developer to control every aspect of the application. They must manually work with the HTTP session, they manipulate attributes in the HTTP request and the HTTP session, they must set up all of the information that they want to be available when the action is complete and so on. Finally they must return the appropriate ActionForward so that struts knows what to do next. If this ActionForward indicates that control should be passed to a JSP then the developer must also write the JSP, using all of the Struts custom tag libraries. This is a huge amount of work and VERY error prone due to the fact that names must be synchronized across THREE locations (the Struts XML config file, the Java Action class and the JSP custom tags). The whole thing is convoluted, error prone and inefficient.

In Wicket, an HTTP request is received. Wicket identifies which page the request is for and which component on the page the request is associated with. If the request is to a form then Wicket automatically extracts the request parameters, validates them, does type conversions and sets the model values on each form component in turn. It then converts the request to an event of the appropriate type and invokes the appropriate event listener on the target component. The event handling code runs, doing whatever business logic is necessary. The event handler can then choose which page to go to next. The page is instantiated (if it does not already exist) and is told to render itself. The render process visits each component in turn and asks it to render itself. Each component may render its HTML directly or may utilise an HTML markup file. The only tie up of names is that each component name in the Java component model must be mapped to an HTML element in the markup file (via a wicket:id attribute).

Reasons why the Wicket approach is much more powerful:

- Wicket components know how to handle their own events. Therefore, once you place a component on a page and write an event handler (usually a handful of lines), that's it. If a page has 20 different components that can cause events you still need do no more than add them to the page. In struts you would either have to create 20 different Action classes or one class with a 20 way if/else if/else/switch statement plus do all the XML config!
- Wicket allows you to think of your application in terms of reusable components and events raised against these components. No more corrupting your object-oriented design to fit with the request/response nature of the HTTP protocol.
- With Wicket you write less stuff. The Java code is generally shorter than the equivalent Struts Action classes, because the reusable components have so much logic build in to them. Pure HTML with Wicket tags is much shorter and simpler than JSP with custom tag libraries. There are no XML config files in Wicket.

If you have programmed with Windows API and then with Visual Basic or Borland Delphi, the following comparison may be helpful. Programming with Struts is similar to program with Windows API: receiving raw message, decoding it and doing whatever you need to do. System does not expect any return value except acknowledgment that message was received. Windows API is procedural and is based on idea of message cycle.

On the other hand, Delphi hides Windows message cycle inside the TApplication class, and allows to build other classes around it. Raw system message is initially received by Delphi built-in classes, it is parsed and its recipient is determined. Then message is converted to an event and delivered to a proper object.

Just like in Windows application, Wicket application has resource files, which are property files for text and HTML templates for GUI. So, in a way, Wicket for web development is what Delphi for desktop development.

Servlet API & HTTP Protocol Abstraction

Struts does not hide the details of the Servlet API and the HTTP protocol. To use Struts you must be happy working with the HttpServletRequest, HttpServletResponse and HttpSession classes as well as the Struts Action, ActionForm, ActionMapping and ActionForward classes. You must also build your application around request/response principles as dictated by the HTTP protocol. This is an inherent weakness of all Model2 MVC web frameworks.

Wicket hides all the details of the Servlet API and the HTTP protocol. For many applications you will never touch any of these details. Even for very complex applications you may only need to very occasionally use one of the Wicket protocol abstraction classes, but even this is very rare. With Wicket you are working with real Java component classes, POJO business models and pure HTML markup files.

State Management

Struts pretty much requires that you take full control over state management. This is fine if you are building a massive, highly scalable, clustered application - you want granular control over everything in your HttpSession. However, for a small or medium sized application it means you have to write a lot of extra code for no reason whatsoever. Thus your application is more complex and takes longer to write.

Wicket adopts an alternative approach. By default the Wicket framework does pretty much ALL state management for you (by remembering the Page and Component states). For small and medium applications the amount of state management code you need to write yourself will be close to zero. However, Wicket also exposes a number of APIs that allow you to take control of the standard state management and implement your own alternatives if necessary. Thus, large applications can still take full control of state management, but this is not imposed on smaller applications. In fact, even when writing a large Wicket application it is common to start out with Wicket managing all of the state and then implementing custom state management solutions as part of performance tuning.

Configuration

Struts requires an XML file that defines all of the request and response mappings, all of the ActionForm objects and so on. This file can get very BIG and complex to maintain. Newer versions of Struts allow this file to be broken into separate modules. However, all this does is result in many slightly smaller but still just as complex XML files.

Wicket has NO CONFIGURATION FILES. There is a simple application configuration class that can be initialised programatically (or via Servlet init parameters) and that is it. All details of how HTTP requests map to component events, how components output their HTML and so on is contained within the Wicket application logic. Most Wicket applications can be deployed with just a handful of entries in the web.xml file - nothing more.

WARNING Unfortunately Wicket doesn't suppress such a configuration; it is still done *but as code*, which is the same and can become just as complex as Struts' XML file(s).