

The Following information and code is compiled by Mr. Faisal Hassan (Lecturer in CCIT) for Advance Java Technologies.

Apache Struts is an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt a model-view-controller (MVC) architecture. It was originally created by Craig McClanahan and donated to the Apache Foundation in May, 2000. Formerly located under the Apache Jakarta Project and known as Jakarta Struts, it became a top-level Apache project in 2005.

In a standard Java EE web application, the client will typically submit information to the server via a web form. The information is then either handed over to a Java Servlet that processes it, interacts with a database and produces an HTML-formatted response, or it is given to a Java Server Pages (JSP) document that intermingles HTML and Java code to achieve the same result. Both approaches are often considered inadequate for large projects because they mix application logic with presentation and make maintenance difficult.

The goal of Struts is to separate the **model** (application logic that interacts with a database) from the **view** (HTML pages presented to the client) and the **controller** (instance that passes information between view and model). Struts provide the controller (a Servlet known as ActionServlet) and facilitate the writing of templates for the view or presentation layer (typically in JSP, but XML/XSLT are also supported). The web application programmer is responsible for writing the model code, and for creating a central configuration file struts-config.xml that binds together model, view and controller.

Requests from the client are sent to the controller in the form of "Actions" defined in the configuration file; if the controller receives such a request it calls the corresponding Action class that interacts with the application-specific model code. The model code returns an "ActionForward", a string telling the controller what output page to send to the client. Information is passed between model and view in the form of special JavaBeans. A powerful custom tag library allows it to read and write the content of these beans from the presentation layer without the need for any embedded Java code.

Struts are categorized as a request-based web application framework.

Components of struts

Java Server Pages (JSP) take over the role of dialogs in struts,
Java Beans take over the business logic and business processes and
Java Servlets take over the central control unit

That's why we talk about three great parts of struts.

Model (business logic / business processes – Java Beans)

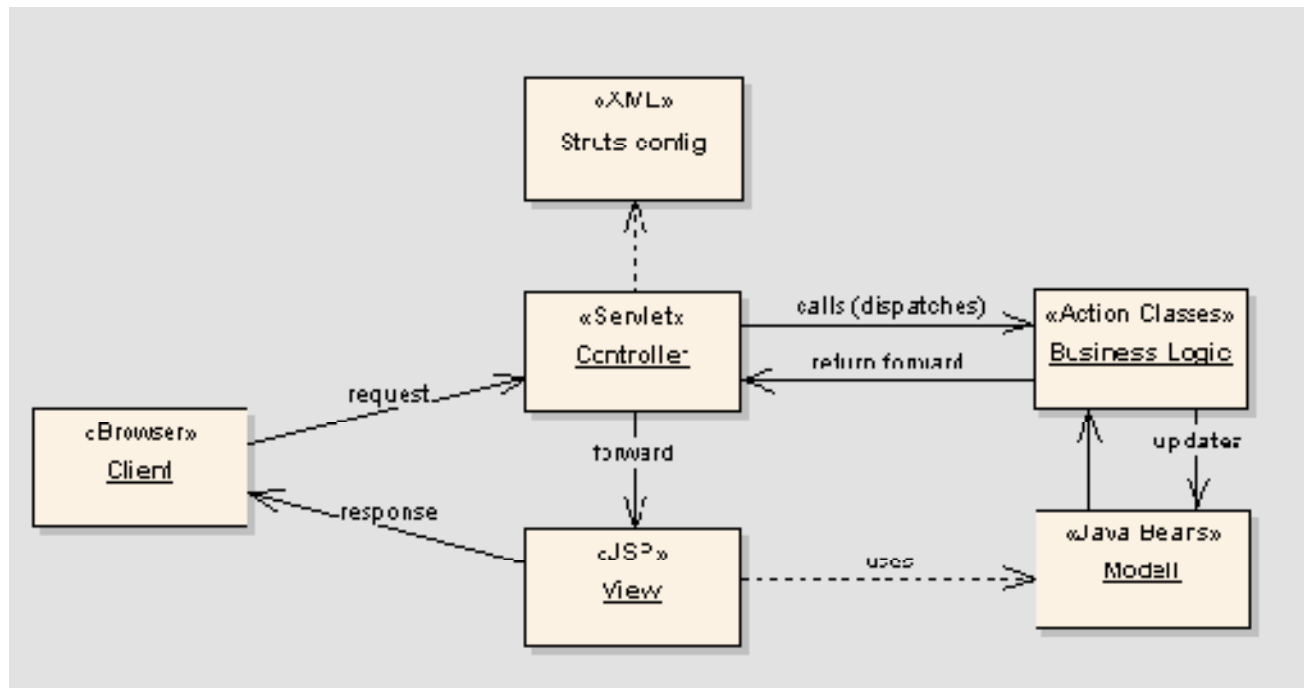
View (dialogs – Java Server Pages)

Controller (central control unit - Java Servlets)

The **model** represents the actual state of the application. Two kinds of java beans are used. There are java beans which contains the data of a form or data to display (ex. The books of a library) and java beans which includes the functionality of the application or call the business logic (when a user borrows a book).

The **view component** is responsible for the presentation of the data. The java server pages Contain HTML, XML and Java Script, like a normal HTML site. Furthermore you can use java code. Struts provide tag libraries, a summary of functions, which can be use to prepare the data for displaying.

The last component is the controller. The controller manage the request of the web browser, which a user called by an address (URL). But also forward to an action which are execute and which dialogs will be used to display the information.



Struts configuration

Struts will be configured with various configuration files. The following files are very important.

- web.xml
- struts-config.xml
- Struts-Tags
- Properties

web.xml

With the web.xml you configure the web server for the struts application. In this file you can set where the web server find the struts-config.xml and some other global properties.

struts-config.xml

The controller calls the business logic or a view with a name. The allocation of the names to the action classes or JSP Files(Views) will be set in the struts-config.xml. The advantage is that you can change the definition of the Workflows (Action Mapping), without using the long class names everytime. If you change a class name, you only have to change the name in the struts-config.xml. You don't update any other part of your application.

Packages and the Java Namespace

A *package* is a named collection of classes (and possibly subpackages). Packages serve to group related classes and define a namespace for the classes they contain.

The Java platform includes packages with names that begin with `java`, `javax`, and `org.omg`. (Sun also defines standard extensions to the Java platform in packages whose names begin with `javax`.) The most fundamental classes of the language are in the package `java.lang`. Various utility classes are in `java.util`. Classes for input and output are in `java.io`, and classes for networking are in `java.net`. Some of these packages contain subpackages. For example, `java.lang` contains two more specialized packages, named `java.lang.reflect` and `java.lang.ref`, and `java.util` contains a subpackage, `java.util.zip`, that contains classes for working with compressed ZIP archives.

In Java, the idea of a namespace is embodied in Java packages. All code belongs to a package, although that package need not be explicitly named. Code from other packages is accessed by prefixing the package name before the appropriate identifier, for example class `String` in package `java.lang` can be referred to as `java.lang.String` (this is known as the fully qualified class name). Like C++, Java offers a construct that makes it unnecessary to type the package name (`import`). However, certain features (such as reflection) require the programmer to use the fully qualified name.

Unlike C++, namespaces in Java are not hierarchical as far as the syntax of the language is concerned. However, packages are named in a hierarchical manner. For example, all packages beginning with `java` are a part of the Java platform—the package `java.lang` contains classes core to the language, and `java.lang.reflect` contains core classes specifically relating to reflection.

Globally Unique Package Names

One of the important functions of packages is to partition the Java namespace and prevent name collisions between classes. It is only their package names that keep the `java.util.List` and `java.awt.List` classes distinct, for example. In order for this to work, however, package names must themselves be distinct. As the developer of Java, Sun controls all package names that begin with `java`, `javax`, and `sun`.

For the rest of us, Sun proposes a package-naming scheme, which, if followed correctly, guarantees globally unique package names. The scheme is to use your Internet domain name, with its elements reversed, as the prefix for all your package names. My web site is *davidflanagan.com*, so all my Java packages begin with `com.davidflanagan`. It is up to me to decide how to partition the namespace below `com.davidflanagan`, but since I own that domain name, no other person or organization who is playing by the rules can define a package with the same name as any of mine.

Java Server Faces (JSF) is an application framework for creating Web-based user interfaces. If you are familiar with Struts (a popular open source JSP-based Web application framework) and Swing (the standard Java user interface framework for desktop applications), think of Java Server Faces as a combination of those two frameworks. Like Struts, JSF provides Web application lifecycle management through a controller Servlet; and like Swing, JSF provides a rich component model complete with event handling and component rendering.