The Model Layer

What is Model?

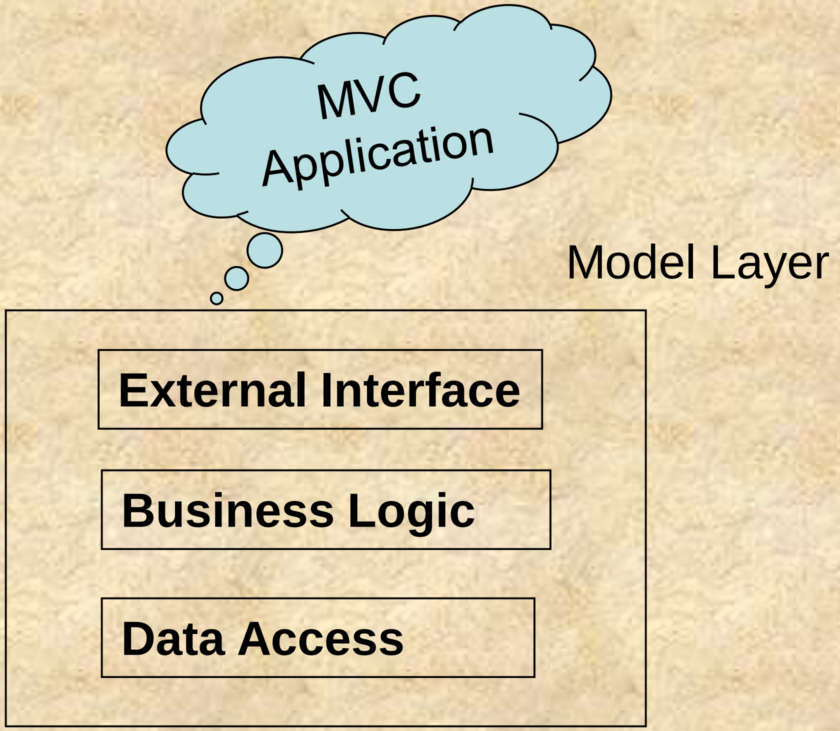
In MVC the Model layer is typically the largest and more important core piece. The Model is designed to house the business logic and data access code. For example, it computes logics and maintains the database transactions. The view and controller interacts with the model and provides user interface to it.

The MVC architecture dictates that, the Model layer should be self contained and function independently from the view and control layer. The core application code can be used over and over again with multiple user interface.

The Model Layer

Model Layer Breakdown:

Typically the Model layer of designed MVC architecture can be broken down into three conceptual sub layers.



The Model Layer

Struts and the Model:

The Struts layer does not provide any specific features to developing Model layer of your application. Struts gives your application flexibility to use any approach for building model layer code. It might be Object Relational Mapping (ORM), Enterprise JavaBeans(EJB), Java Data Objects (JDO), Data Access Object (DAO) pattern, Struts will accommodate.

Your model code will be accessed from the sub classes of the Struts ***Action*** object that are part of the controller layer of the Struts framework.

The View Layer

In MVC application, the view layer provides an interface to your application, be it for users with browser or for web services.

Basically the view layer is the conduit for getting data in and out of the application. The view layer simply concentrate on the interface.

Keeping the model and view layers separate from one another allows an application's interface to change independent to the model layer and vice versa. This separation also allows to have multiple interfaces (or views).

The View Layer

Struts and View Layer:

Struts provides rich set of functionality and features for developing the View layer of MVC applications. It can be HTML/JSP or XML or Swings or whatever your application needs. Struts HTML/JSP view layer support can be broken down into the following major components:

1) JSP Pages
2) Form Beans
3) JSP tag Libraries
4) Resource bundles

The Controller Layer

The Controller layer of MVC is responsible for creating the abstraction between the Model and View layers. It acts as a liaison (link) between model and view layer. The controller layer serves as a central point of access to the application.

All the requests to MVC web application flow through the controller. It provides security, caching, logging and so on.

The View Layer

Struts and Control layer:

Struts provides a robust controller layer implementation that has been designed from the ground up to be extensible.

Its core is the Controller servlet, ***ActionServlet***, which is responsible for initializing a struts application's configuration from the struts configuration file and for receiving all incoming requests to the application.

Upon receiving a request, ActionServlet delegates its processing to the struts request processing engine.

The View Layer

The ActionServlet class:

The class org.apache.struts.action.ActionServlet is called the ActionServlet. In the Jakarta Struts Framework this class plays the role of controller. All the requests to the server goes through the controller. Controller is responsible for handling all the requests. The ActionServlet is the main controller class that receives all incoming HTTP requests for the application. ActionServlet is responsible for initializing the struts framework for your application. Like any other servlet, ActionServlet must be configured in your application's web application deployment descriptor: web.xml.There are two ways that ActionServlet can be configured to receive requests in web.xml

The ActionServlet class: First, using path mapping:

```
<servlet>
<servlet-name>action</servlet-name>
 <servlet-class>org.apache.struts.action.ActionServlet </servlet-class>
<init-param><param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value></init-
param>
 <load-on-startup>1</load-on-startup>
 </servlet>
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>/do/*</url-pattern>
</servlet-mapping>
```

The ActionServlet class: Second, using extension mapping:

```
<servlet>
<servlet-name>action</servlet-name>
 <servlet-class>org.apache.struts.action.ActionServlet </servlet-class>
<init-param><param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value></init-
param>
 <load-on-startup>1</load-on-startup>
 </servlet>
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
```

The Request Processing Engine:

Struts uses its request processing engine to perform the processing for all requests received by the ActionServlet. The request processing engine takes each request and breaks its processing down into several small tasks.

Jakarta Commons Chain based request Processing:

The RequestProcessor class based processing originally used in struts provides a nice abstraction for each part of the request processing cycle.

The common chain lib. Provides an implementation of the chain of responsibility pattern. The chain represents the entire computations as a series of commands.

The RequestProcessor class based Processing:

The RequestProcessor Class is the actual place where the request processing takes place in a Struts controller environment.

When the request object first reaches the actionservlet class then it invokes the process method of the underlying RequestProcessor Class.

processor has most of the following responsibilities:

Determine path, Handle Locale, Process content and encoding type,

Process cache headers, Pre-processing hook, Determine mapping, Determine roles, Process and validate actionForm, Return a response

The Action Class:

The goal of an Action class is to process a request, via its execute method, and return an ActionForward object that identifies where control should be forwarded (e.g. a JSP, Tile definition, Velocity template, or another Action) to provide the appropriate response. In the *MVC/Model 2* design pattern, a typical Action class will often implement logic like the following in its execute method:

1)Validate the current state of the user's session

2) If validation is not complete, validate the form bean properties as needed.

3) Perform the processing required to deal with this request

4) Update the server-side objects that will be used to create the next page of the user interface.

5) Return an appropriate ActionForward object that identifies the presentation page