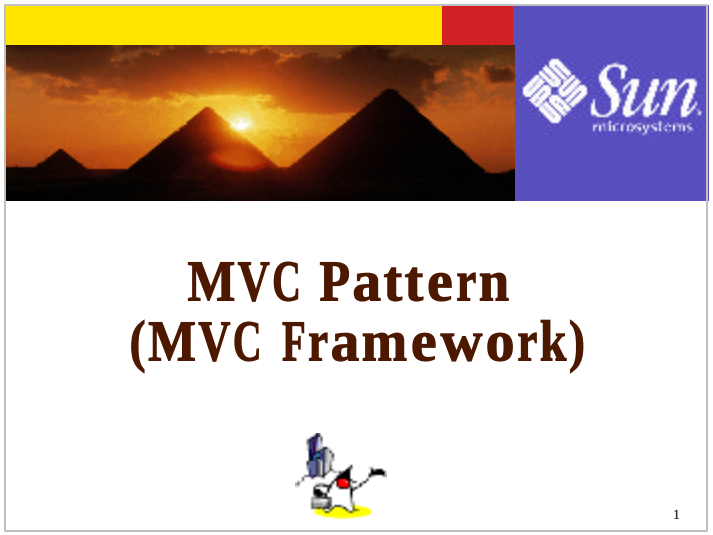


In this session, we will learn about basic concept of MVC (Model View Controller) pattern and frameworks. Understanding MVC pattern is required in order to understand Struts and JSF later on.

Sang Shin

sang.shin@sun.com
www.javapassion.com
Java™ Technology Evangelist
Sun Microsystems, Inc.

2

Disclaimer & Acknowledgments

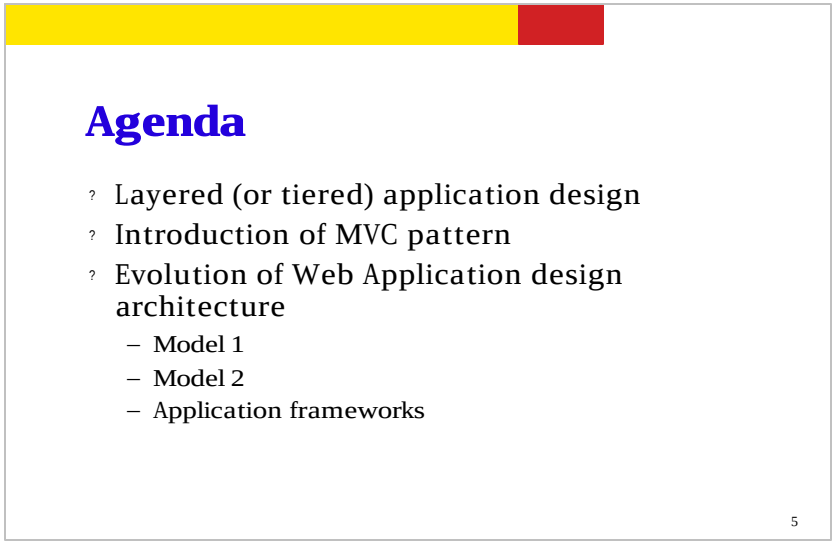
- Even though Sang Shin is a full-time employees of Sun Microsystems, the contents here are created as their own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgements
  - Many slides are borrowed from "Sevlet/JSP" codecamp material authored by Doris Chen of Sun Microsystems
  - Some slides are from JavaOne 2003 "Blueprint for Web services" presentation authored by Inderjeet Singh and Sean Brydon

3

Revision History

- 11/01/2003: version 1: created by Sang Shin
- Things to do
  - speaker notes need to be polished

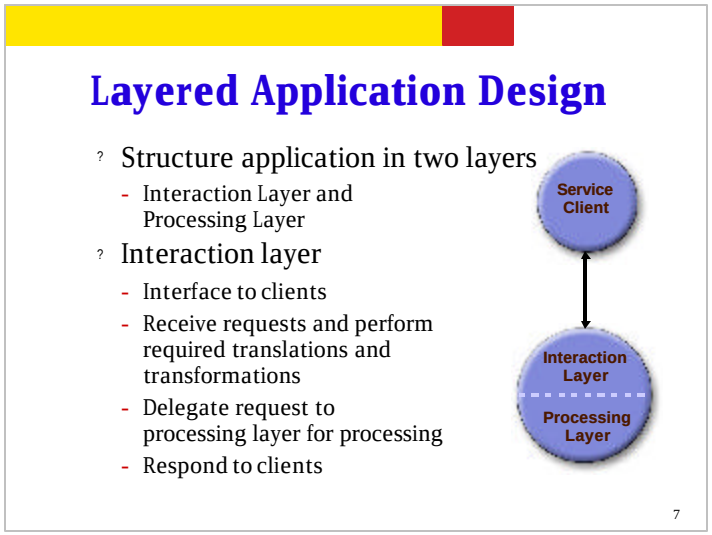
4



This is the agenda. First, we will talk about general concept behind layered (or tiered) application architecture. Then we will discuss how MVC pattern fits in with this layered architecture.

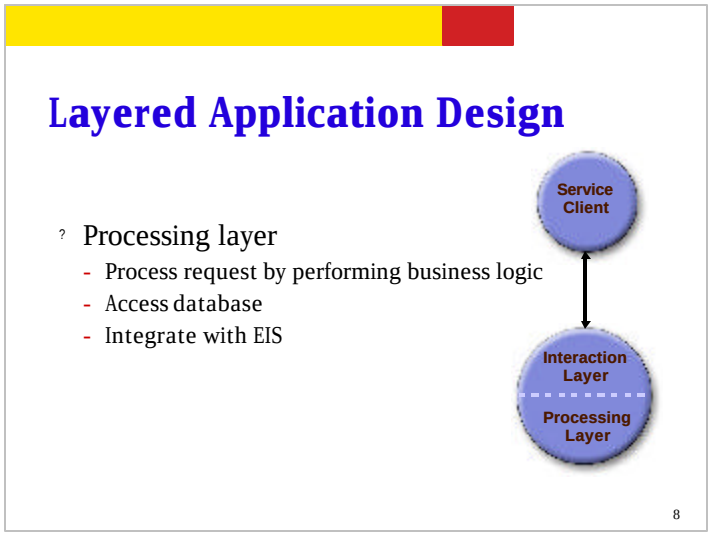
During the rest of the session, we will discuss the evolution of web application design architecture starting with Model 1, then talk about Model2 and finally application frameworks.

First, let's talk about general concept of layered application design before we talk about MVC pattern for Web applications.

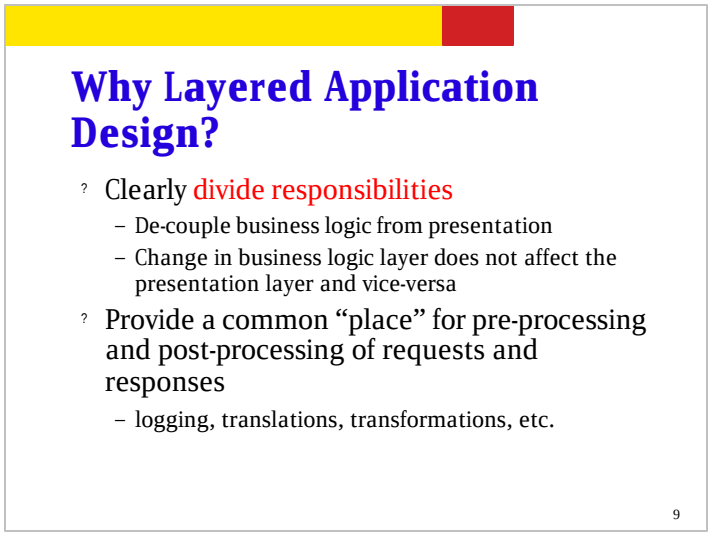


Typically a distributed application is composed of two layers - interaction layer and processing layer.

The interaction layer provides an interface to clients and receives requests and perform any required translations and transformations. Then it delegates or dispatches the request handling to process layer. It receives whatever result from the processing layer and then send response to clients.



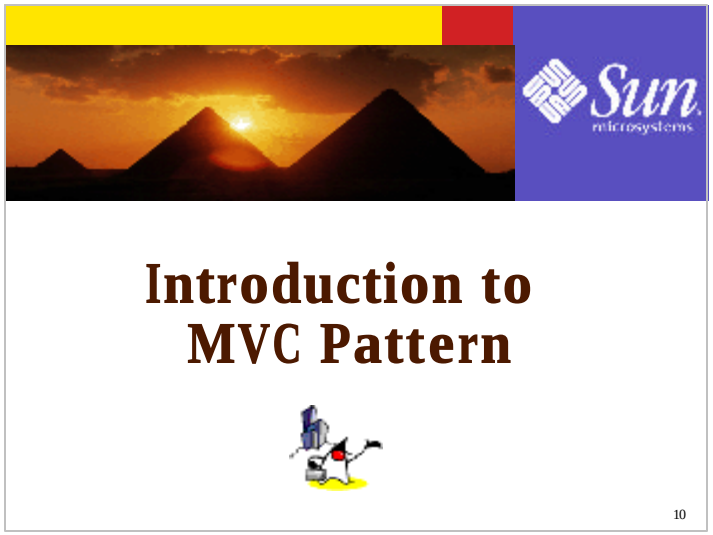
The processing layer processes the request by performing business logic. The business logic might involve accessing database or integration with Enterprise Information Systems.

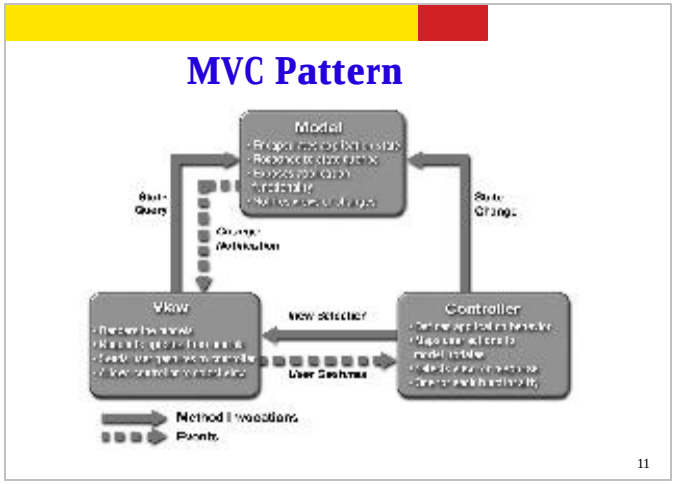


Now let's talk about why you want to use layered application design?

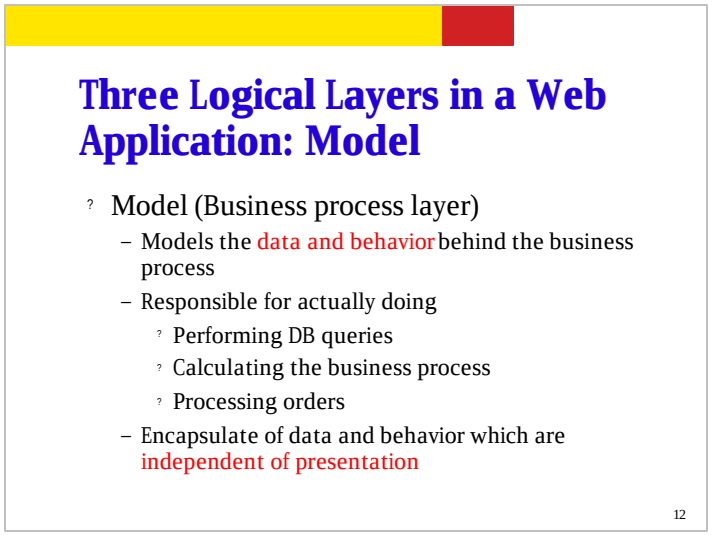
Layered design clearly divides responsibilities as mentioned in the previous two slides. For example, the business logic handling is separated from interaction with clients. This separation allows one layer to change without affecting the other.

The layered design also provides a common place where pre and post processing such as logging, translations and transformations can be performed.


Sun
microsystems
Introduction to
MVC Pattern
10

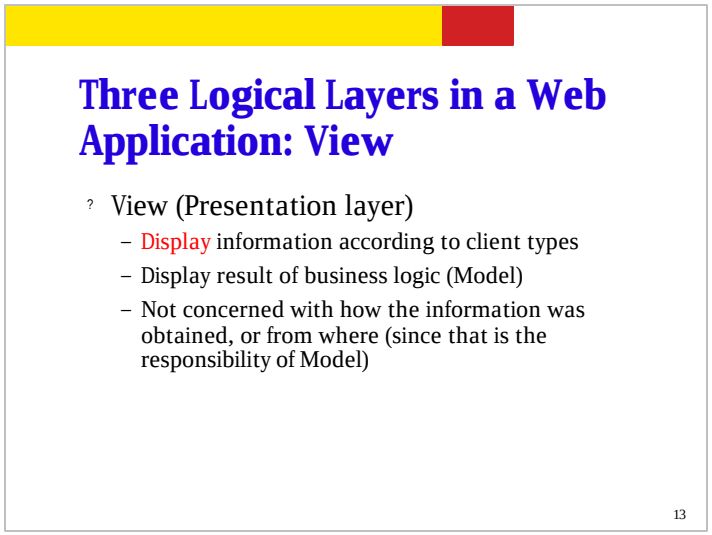


This picture shows MVC pattern, in which Model, View, and Controller components are shown and how they interact.

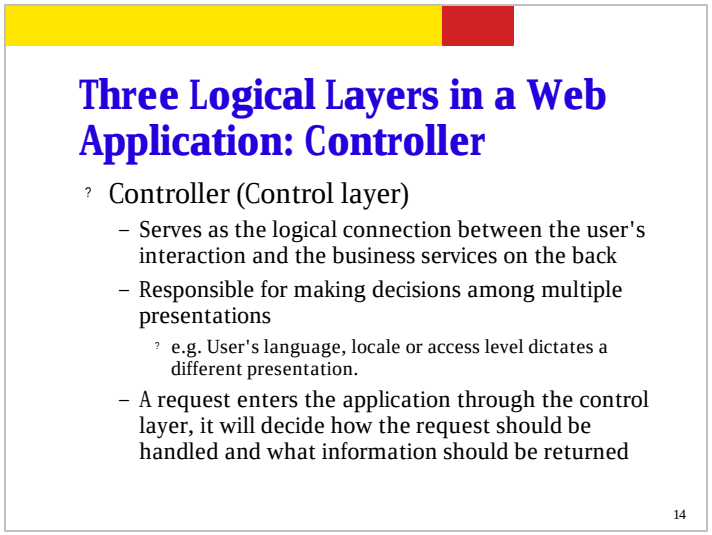


So the MVC pattern is basically layered architecture for a Web application. The Model provides business process layer. It models the data and behavior behind the business process. As part of performing business process, the model layer might perform database access, calculating business process, or processing orders for example.

One important point to remember is that Model encapsulate data and behavior independent of how they are presented. That is, the presentation could change depending on the client type but the Model should not change.



The View represents presentation layer. The view handles displaying information according to client type. The view displays result of business logic processing, that is Model. Since View is independent of Model, the view is not concerned with how and where the information in Model was obtained.



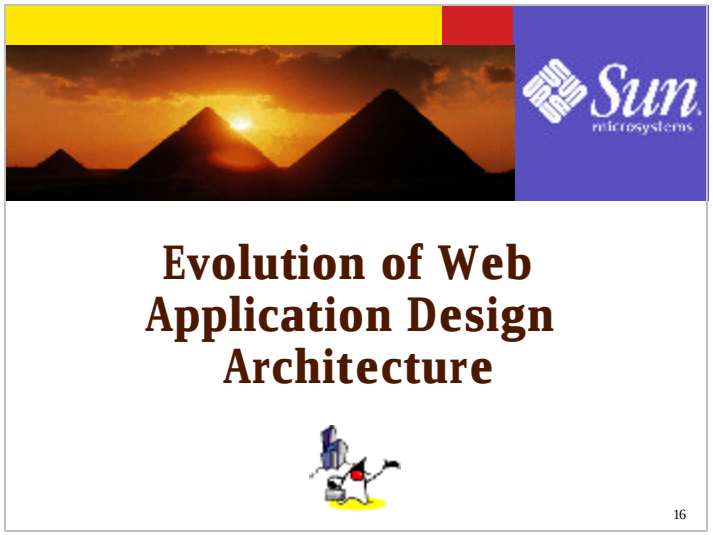
(read the slide)

Web Applications

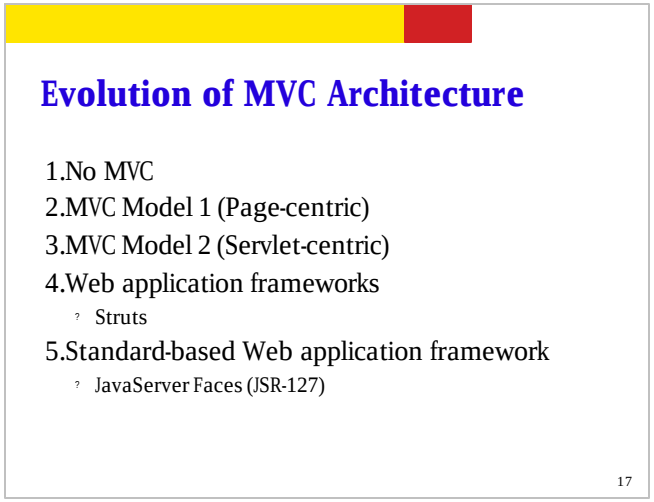
- It is often advantageous to treat each layer as an independent portion of your application
- Do not confuse logical separation of responsibilities with actual separation of components
- Some or of the layers can be combined into single components to reduce application complexity

15

(read the slide)



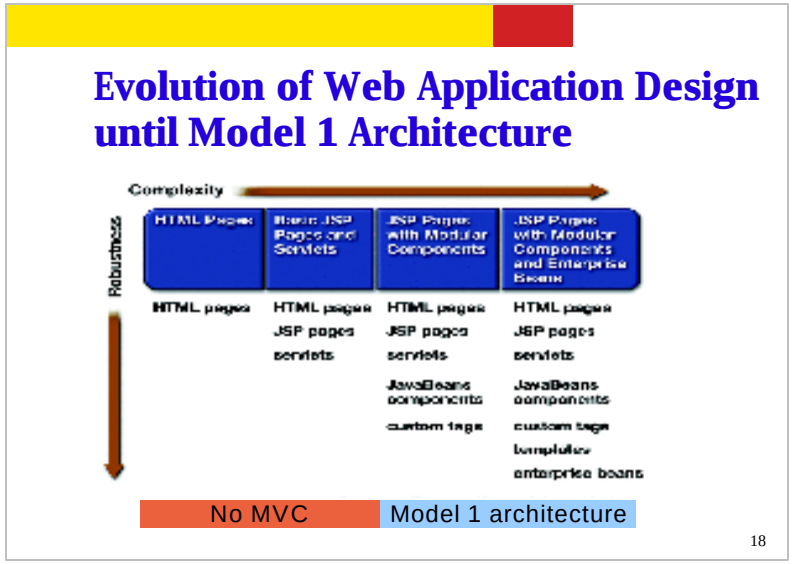
Now let's talk about the evolution of web application design architecture.



Now when we talk about Web application framework, we are basically talking about the evolution of MVC architecture, which stands for Model, View, and Controller.

So in the beginning, we used no MVC. Then we had JSP Model1 and Model 2 architecture. And people came up with so called Web application frameworks such as Apache Strut based on Model 2 architecture. And finally we are at the phase there will be a standard based Web application framework.

So let's talk about these in a bit more detail.

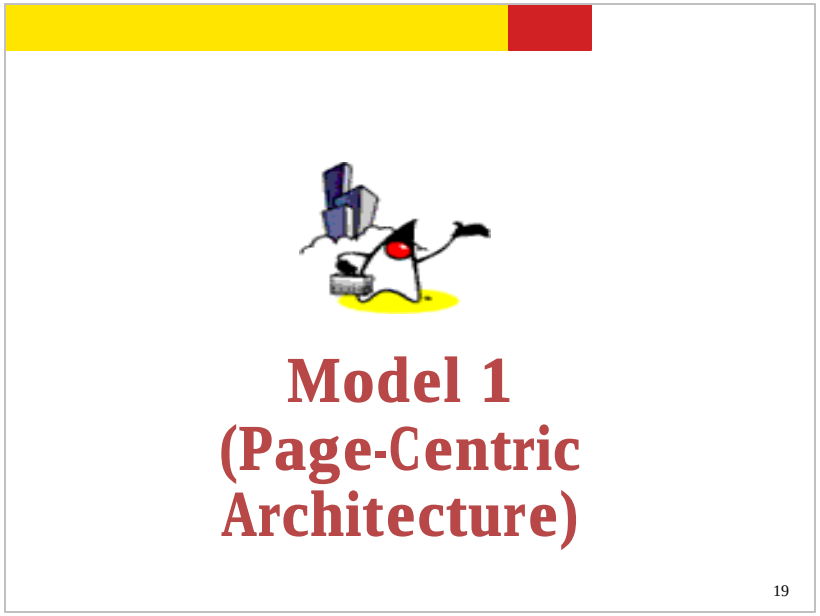


This picture shows the evolution of web application starting from a simplest one which then evolves into more sophisticated and more robust design.

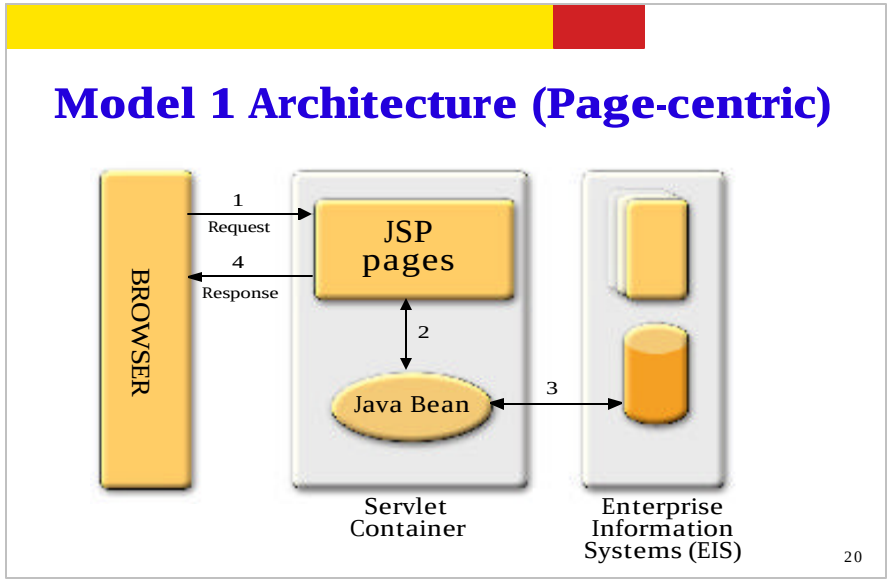
So in the first phase of the evolution, just static HTML pages were used to display static information. Then in the subsequent evolution phase, dynamic contents generation technologies such as CGI initially, then servlet and JSP are introduced to generate and display dynamic contents along with static contents.

When you are using JSP pages, you can use only the basic features that come with it. Or you can leverage more sophisticated features such as component-based dynamic contents generation, for example, leveraging JavaBeans or custom tags, which provide more reusable, more maintainable, more flexible development and deployment options.

Then in a more sophisticated environment, people use so-called template based design or eventually they might want to delegate their business logic processing to EJB tier.



Let's see Model 1 architecture first. The model 1 architecture is page-centric architecture.



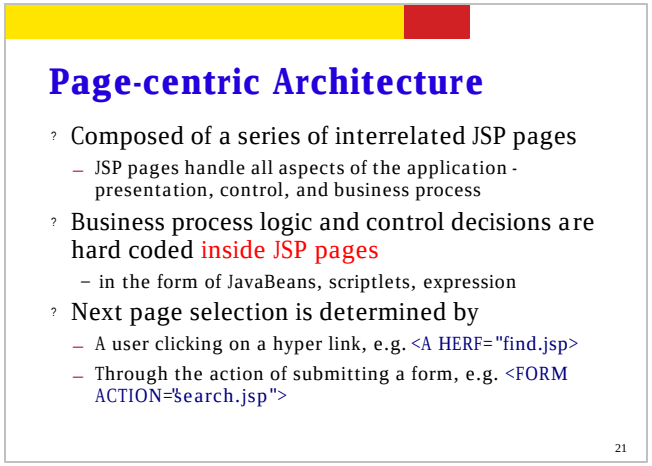
The literature on Web-tier technology in the J2EE platform frequently uses the terms "Model 1" and "Model 2" without explanation. This terminology stems from early drafts of the JSP specification, which described two basic usage patterns for JSP pages. While the terms have disappeared from the specification document, they remain in common use.

Model 1 and Model 2 simply refer to the absence or presence (respectively) of a controller servlet that dispatches requests from the client tier and selects views.

A Model 1 architecture consists of a Web browser directly accessing Web-tier JSP pages. The JSP pages access Web-tier JavaBeans that represent the application model. And the next view to display (JSP page, servlet, HTML page, and so on) is determined either by hyperlinks selected in the source document or by request parameters.

In a Model 1 architecture, view selection is decentralized, because the current page being displayed determines the next page to display. In addition, each JSP page or servlet processes its own inputs (parameters from GET or POST). And this is hard to maintain, for example, if you have to change the view selection, then several JSP pages need to be changed.

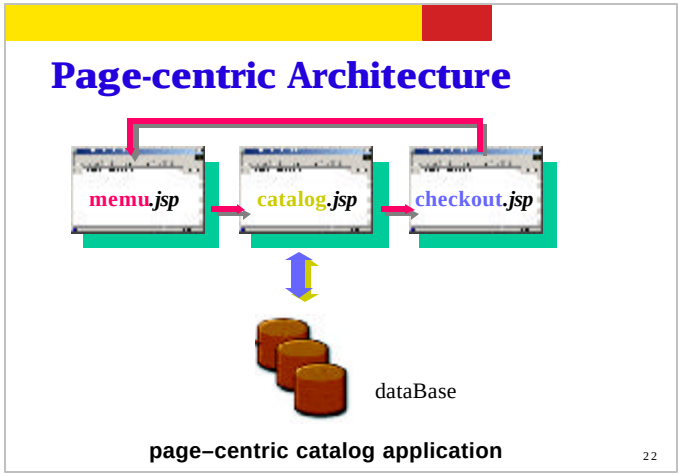
In some Model 1 architectures, choosing the next page to display occurs in scriptlet code, but this usage is considered poor form.



In page-centric architecture, the application is composed of a series of interrelated JSP pages. And these JSP pages handle all aspects of application including presentation, and control and the business process.

Now an issue is that, in page centric architecture, the business process logic and control decisions are hard-coded inside JSP pages in the form of JavaBeans, scriptlets and expressions. Here JSP is performing the role of model.

Also in page-centric architecture, the page selection is determined inside JSP pages either user clicking on a hyperlink or action forms. Here JSP pages are playing the role of view selection which is typically handled by controller.



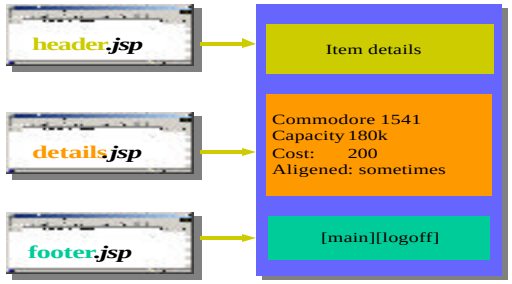
This picture shows what I mentioned in the previous slide. In page-centric architecture, the next page selection is determined by each JSP page. So the page selection is distributed. (On the other hand, servlet-centric approach, which we will learn later, the page selection will be handled by a single servlet which is a centralized approach in terms of page selection.) The same thing can be said about input handling. Here each page has to handle input handling such as translation or transformation. So controller function is also distributed among different pages.

Page-centric: Simple Application

- One page might display a menu of options, another might provide a form for selecting items from the catalog,and another would be to complete shopping process
  - This doesn't mean we lose separation of presentation and content
  - Still use the dynamic nature of JSP and its support for JavaBeans component to factor out business logic from presentation
  - The pages are tightly coupled:
    - Need to sync up request parameters
    - Be aware of each other's URLs

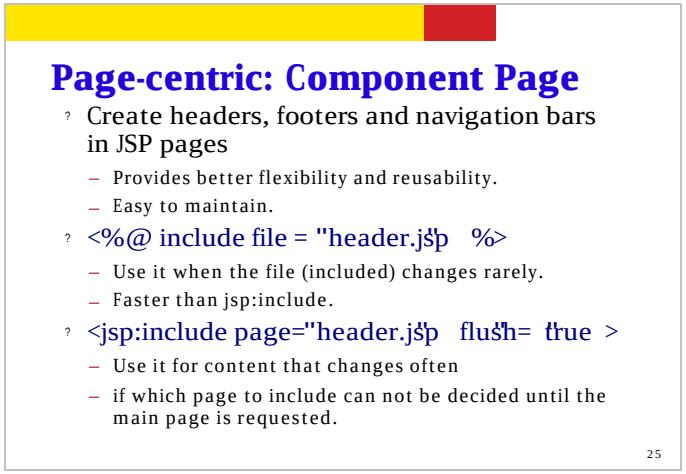
23

Page-centric: Component Page Diagram

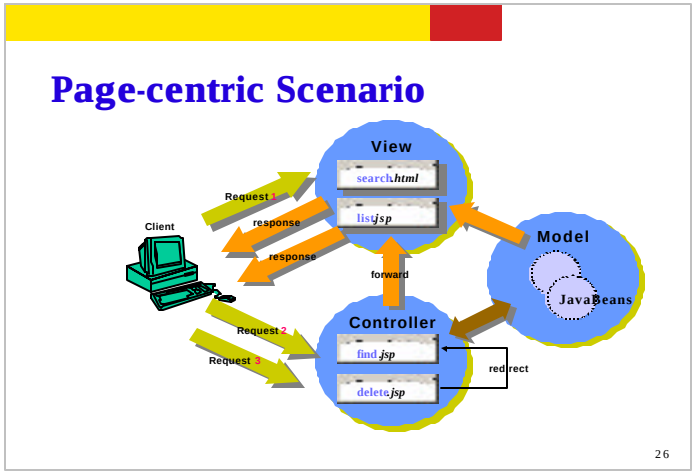


Component design

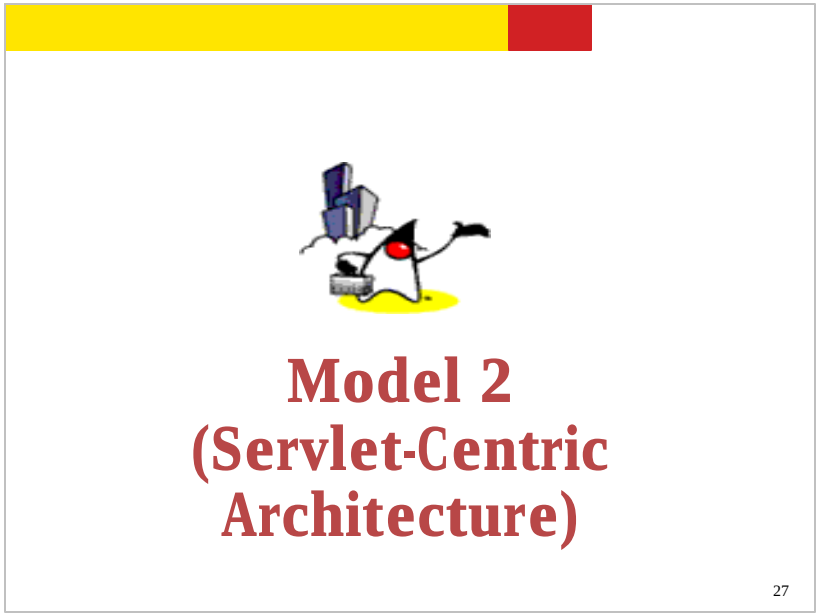
24



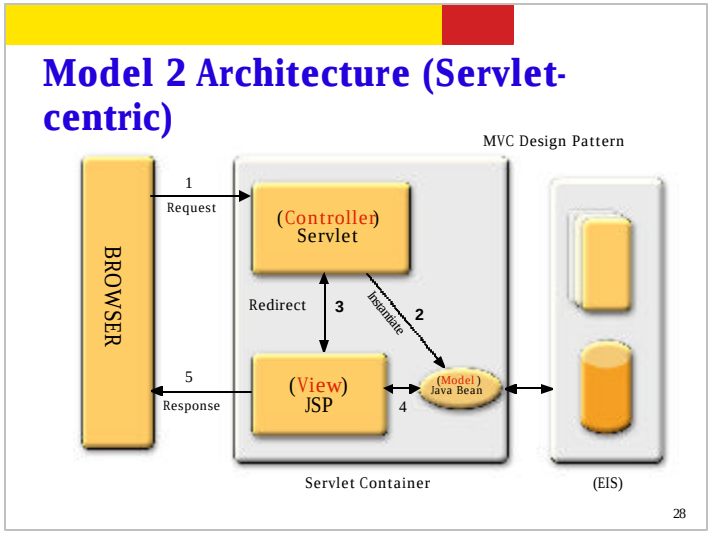
Now if you decide to use page centric approach, here are a few suggestions.



In page centric scenario, the role of controller is played by JSP pages.



Now let's talk about model 2 architecture which we call servlet-centric architecture.

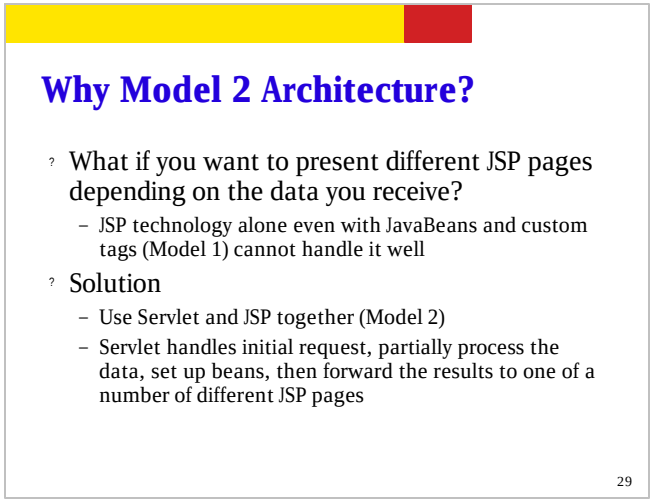


A Model 2 architecture introduces a controller servlet between the browser and the JSP pages.

The controller centralizes the logic for dispatching requests to the next view based on the request URL, input parameters, and application state. The controller also handles view selection, which de-couples JSP pages and servlets from one another.

Model 2 applications are easier to maintain and extend, because views do not refer to each other directly. The Model 2 controller servlet provides a single point of control for security and logging, and often encapsulates incoming data into a form usable by the back-end MVC model.

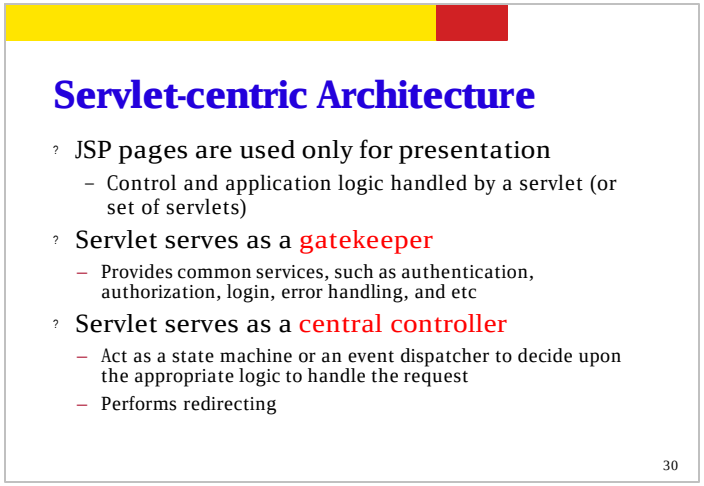
For these reasons, the Model 2 architecture is recommended for most web applications.



I mentioned in previous slide, under Model 1 architecture, a view selection is done by each JSP page. And this poses a maintenance problem.

Now there is another limitation of using Model 1 architecture. In many cases, you want to select JSP pages depending on the data you received from the client. This means there has to be some software entity that handles the processing of the data and then selection of the view. And JSP is not really a good place that you can put this programming logic.

So what is the solution? Model 2 architecture. In Model 2 architecture, both servlet and JSP are used together. In other words, Servlet handles initial request, partially process the data, then forward the results to different JSP pages.

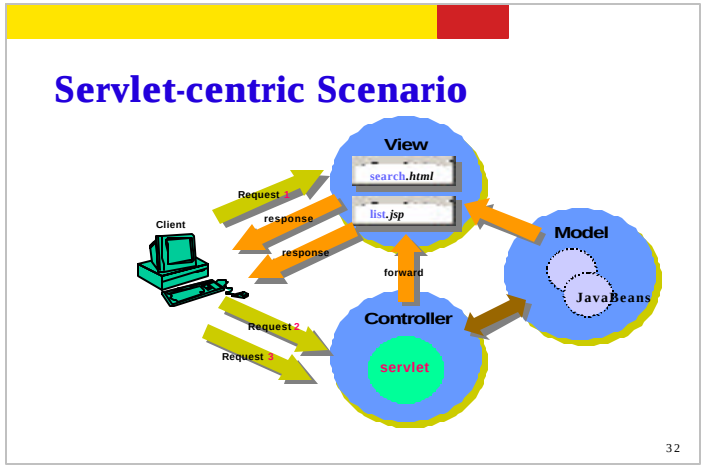


In servlet-centric architecture, JSP pages are used only for presentation and control and business process are handled by a servlet or a set of servlets.

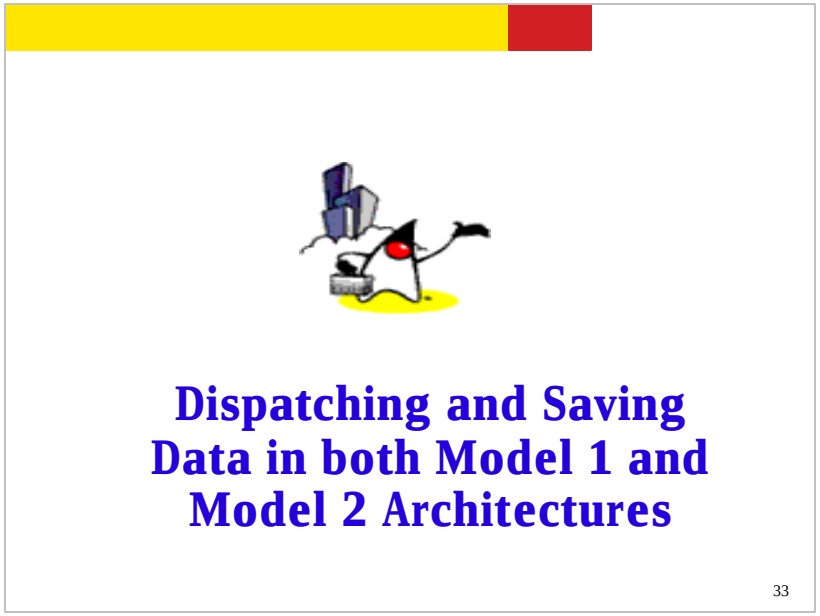
So here, this servlet serves as a gatekeeper providing common services such as authentication, logging, translation, and transformation for all incoming requests.

This servlet also serves as a central controller acting as a state machine or event dispatcher to decide upon appropriate logic to handle the request. It also performs the redirecting the request.

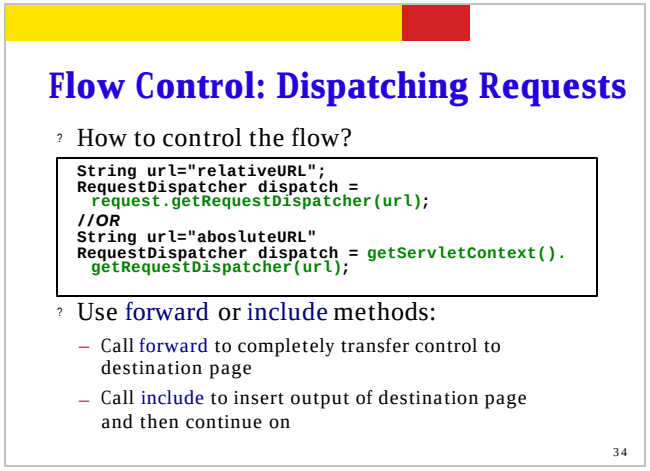
One question people ask is who many servlets are desired to build servlet-centric architecture. The answer is of course “it depends”. It depends on the granularity of your application. For a simple application, you could have a single master servlet that handles everything or you could have multiple servlets each of which handles different business function. Or you could build a combination of these two approach in which you have a master servlet that handles common functions such as common login for all business functions and then delegate to child servlets for further processing.



This picture shows servlet-centric scenario. Here the role of controller is played by a single (or a set of) servlets.



Now let's see some techniques you can use for dispatching and saving data in model 1 and model 2 architectures.



So in a servlet, how do you perform the delegation, that is dispatching requests? (This is something we already learned when we talk about advanced servlet in “J2EE programming” class.)

You can call “forward” to completely transfer control to destination page. Or you can call “include” to insert output of the destination page and then continue on.

Storing Data in a JavaBean in Request

- Store data that servlet looked up and that JSP will use in this request
- Sevlet: store data

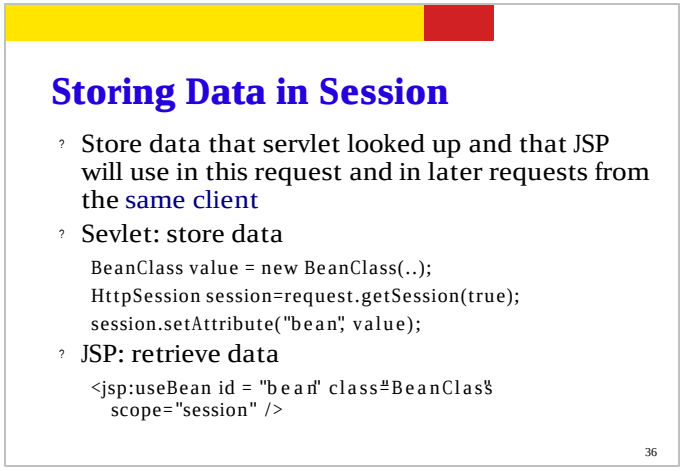
```
BeanClass value = new BeanClass(..)
request.setAtrribute("bean" , value)
```

- JSP: retrieve data

```
<jsp: useBean id="bean" class="BeanClass"
      scope="request"/>
```

35

This slide shows how servlet stores data in a JavaBean and then saves it as a Request scope variable from which JSP page will later retrieves data.



This slide shows how servlet stores data in a JavaBean and then saves it as a session scope variable from which JSP page will later retrieves data.

Storing Data in Servlet Context

- Store data that servlet looked up and that JSP will use in this request and in later requests from the any client
- Sevlet: store data

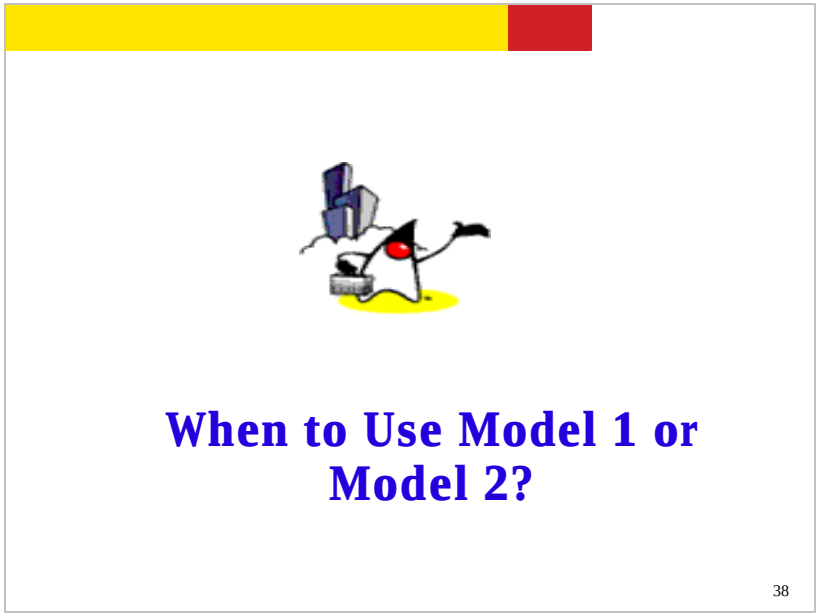
```
BeanClass value = mew BeanClass(..);
getServletContext().setAttribute("bean, value);
```

- JSP: retrieve data

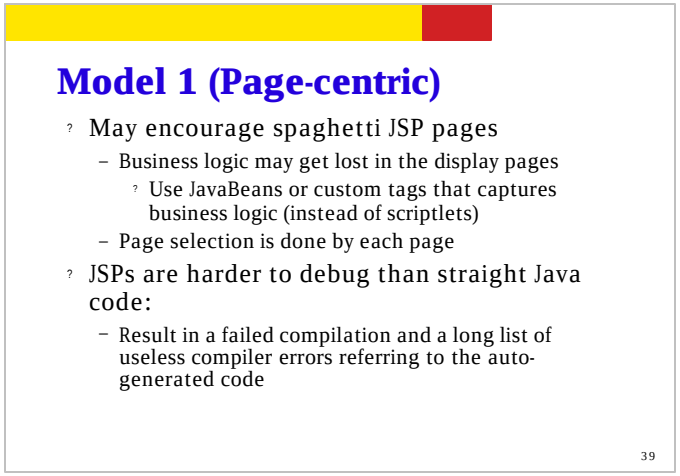
```
<jsp:useBean id = "bean" class="BeanClass"
  scope="application" />
```

37

This slide shows how servlet stores data in a JavaBean and then saves it as an application context scope variable from which JSP page will later retrieves data.



Even though most relatively sophisticated Web applications are expected to use Model 2 architecture, let's compare them here anyway.



One issue with Model 1 architecture is that because the page selection is done by each JSP page and business logic processing is done within JSP pages, it may encourage spaghetti JSP pages. The recommended approach if you are going to use page-centric approach is to capture business logic in the form of JavaBeans or custom tags.

In general, JSP pages are easier to write than servlets but it is harder to debug since the actual code is generated by the JSP compiler.

Model 2 (Servlet-centric)

- Loosens the coupling between the pages and improves the abstraction between presentation and application logic
  - Use JSPs for pure data display and input collection activities
  - Most of the business logic can be debugged through the servlet before passed to JavaBeans and JSP

40

Best Practice Guideline

- Factor out the business logic into business objects and complex display logic into view objects
  - Improves reusability, maintainability, unit testing and regression testing.

41

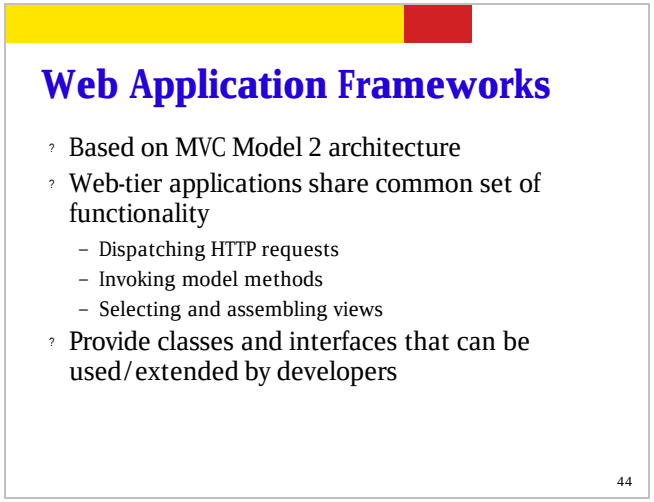
How Do I Decide?

- Use page-centric
  - If the application is simple enough that links from page to page.
- Use servlet-centric
  - Each link or button click requires a great deal of processing and decision-making about what should be displayed next.
- "How mapping between requests and responses are done" can help you to decide
  - Each request maps to one and only one response
    - No need for controller.
  - Each request spawns a great deal of logic and a variety of different views can result
    - A servlet is ideal

42

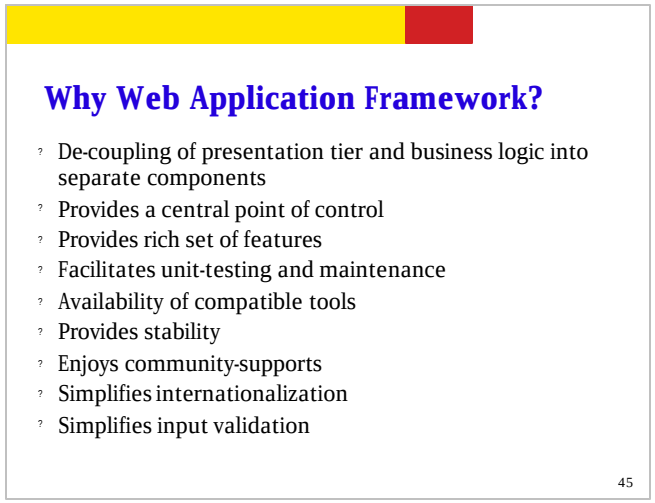
Web Application Frameworks

43

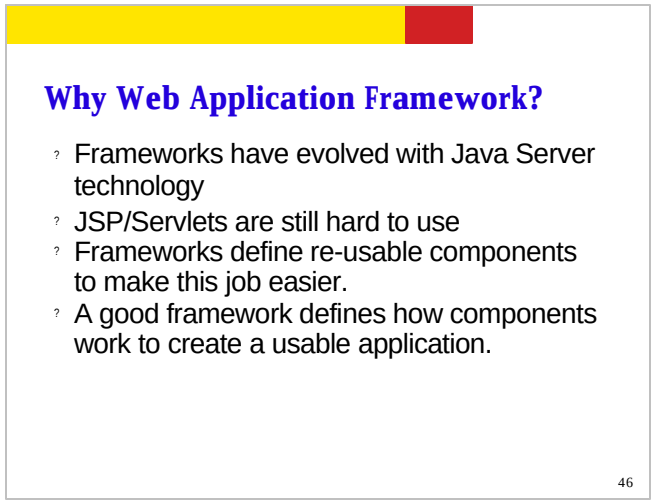


Now as people are gaining more experience, they found that most Model 2 architecture based web applications share a common set of functionality. For example, they all do receive and dispatch HTTP requests, invoking model methods, selecting and assembling views.

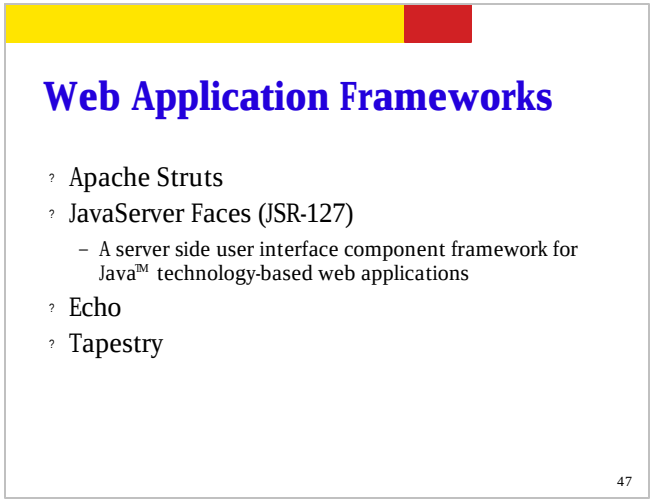
Well, if everybody is doing the same thing over and over every time they write Model 2 based application, then people thought why don't we create a common framework that support these set of functionality so only thing you have to do is basically using or extending the frameworks using common interface and classes.



In a more concrete terms, these are the benefits of using a common Web application frameworks. Framework decouples presentation tier from business logic, which would be useful for maintenance and reusability of the code. It provides a central point of control. Popular frameworks also come with other extra features



(read the slide)



Now one of the most popular Web application framework is Apache Struts. Sun ONE Application framework has also popular.

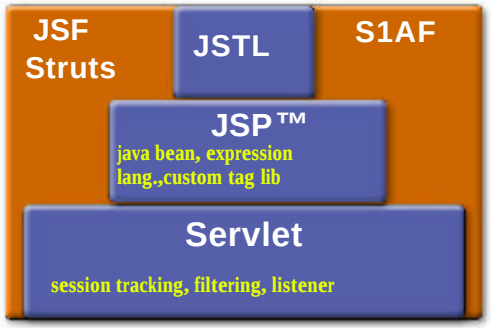
Now Java community is almost done with JavaServer Faces (JSR-127) and this will unify all existing web application frameworks such as Struts and Sun ONE application framework into a single standard web application framework.

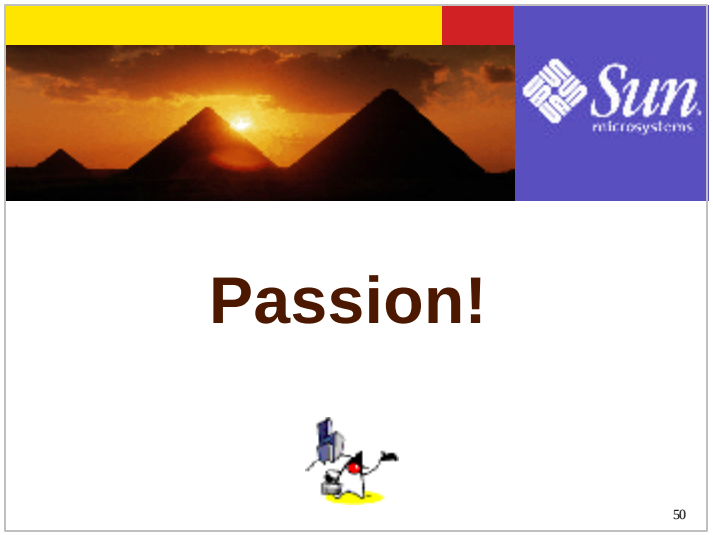
Struts and JSF

- Struts is a very popular web application framework
  - Includes tag library for HTML presentation (overlapping area with JSF)
- Struts-Faces Integration Library
  - Use JSF component model for new UI components
  - A few Struts-specific components / renderers
  - Replace use of Struts HTML tags
  - Maintain investment in Struts business logic

48

How It Fits Together




Sun
microsystems
Passion!
50