Day10: Distributed application introduction, MVC model introduction, Client server communication

Distributed Multitiered Applications:

The Java EE platform uses a distributed multitiered application model for enterprise applications. Application logic is divided into components according to function, and the application components that make up a Java EE application are installed on various machines, depending on the tier in the multitiered Java EE environment to which the application component belongs.

The Java EE application parts are presented in Java EE Components.

- Client-tier components run on the client machine.
- Web-tier components run on the Java EE server.
- Business-tier components run on the Java EE server.
- Enterprise information system (EIS)-tier software runs on the EIS server.

Although a Java EE application can consist of all tiers shown in Figure 1-1, Java EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three locations: client machines, the Java EE server machine, and the database or legacy machines at the back end. Three-tiered applications that run in this way extend the standard two-tiered client-and-server model by placing a multithreaded application server between the client application and back-end storage.

Java EE Clients

A Java EE client is usually either a web client or an application client.

Web Clients

A **web client** consists of two parts:

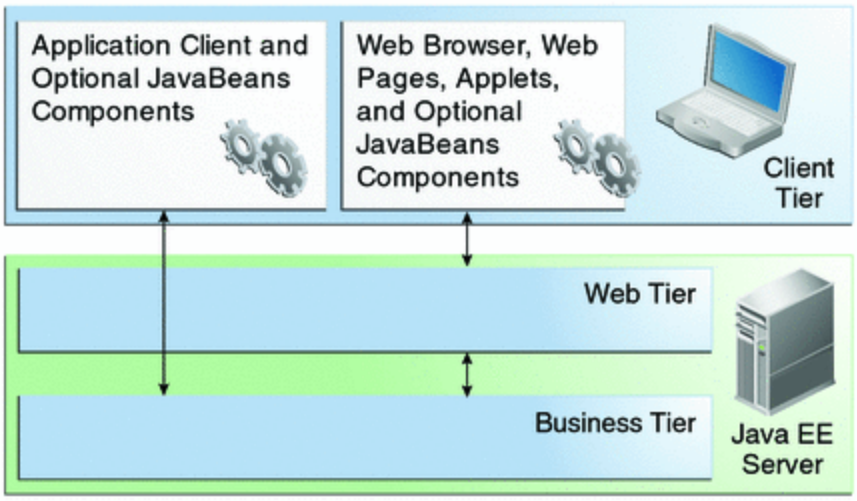
- Dynamic web pages containing various types of markup language (HTML, XML, and so on), which are generated by web components running in the web tier
- A web browser, which renders the pages received from the server

A web client is sometimes called a **thin client**. Thin clients usually do not query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, such heavyweight operations are off-loaded to enterprise beans executing on the Java EE server, where they can leverage the security, speed, services, and reliability of Java EE server-side technologies.

Application Clients

An **application client** runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language. An application client typically has a graphical user interface (GUI) created from the Swing or the Abstract Window Toolkit (AWT) API, but a command-line interface is certainly possible.

Application clients directly access enterprise beans running in the business tier. However, if application requirements warrant it, an application client can open an HTTP connection to establish communication with a servlet running in the web tier. Application clients written in languages other than Java can interact with Java EE servers, enabling the Java EE platform to interoperate with legacy systems, clients, and non-Java languages.

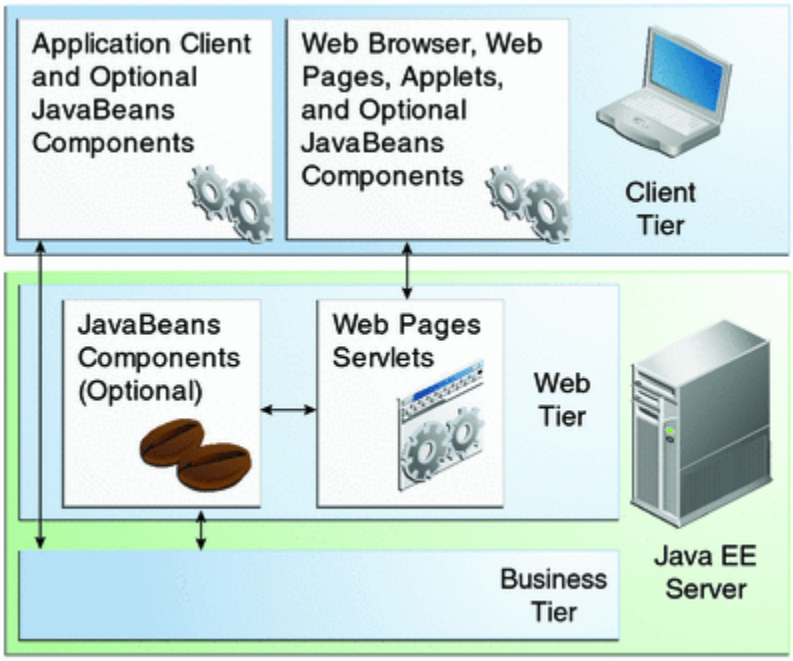


Web Components:

Java EE web components are either servlets or web pages created using JavaServer Faces technology and/or JSP technology (JSP pages). **Servlets** are Java programming language classes that dynamically process requests and construct responses. **JSP pages** are text-based documents that execute as servlets but allow a more natural approach to creating static content. **JavaServer Faces technology** builds on servlets and JSP technology and provides a user interface component framework for web applications.

Static HTML pages and applets are bundled with web components during application assembly but are not considered web components by the Java EE specification. Server-side utility classes can also be bundled with web components and, like HTML pages, are not considered web components.

As shown in below image, the web tier, like the client tier, might include a JavaBeans component to manage the user input and send that input to enterprise beans running in the business tier for processing.



A Web component is nothing more than a software component that services an incoming HTTP request and provides some kind of (hopefully valid) response.

Most if not all Java Web frameworks are built upon the core Java servlets technology. A servlet is a persistent piece of code that receives an abstraction of an HTTP request and gives an HTTP response.

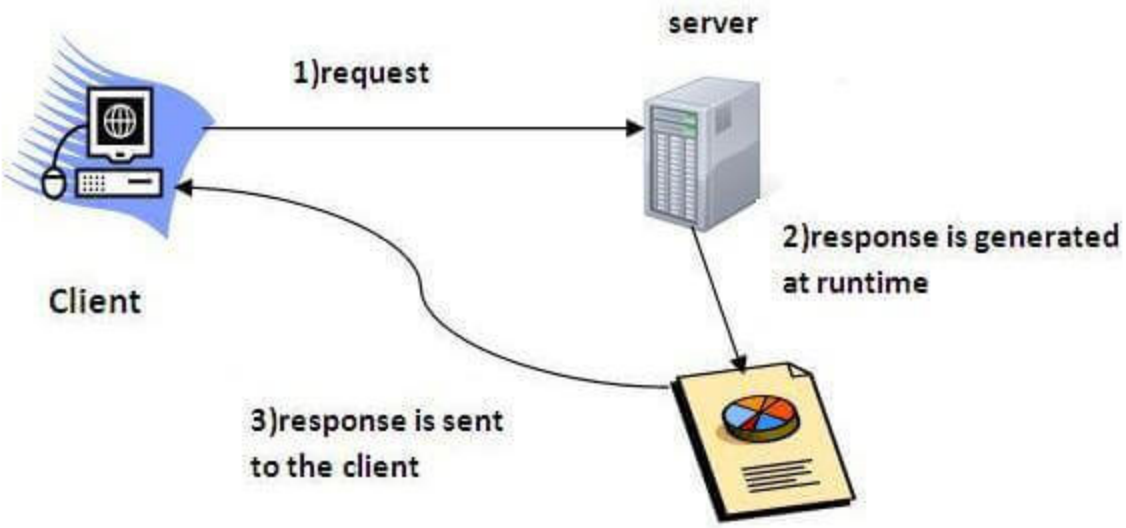
What is a web application?

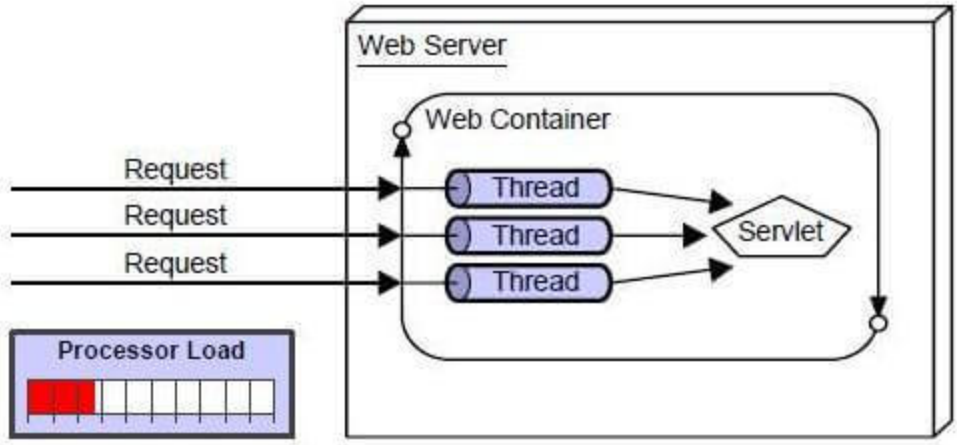
A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.

Servlet is a web component that is deployed on the server to create a dynamic web page.





JSP technology:

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

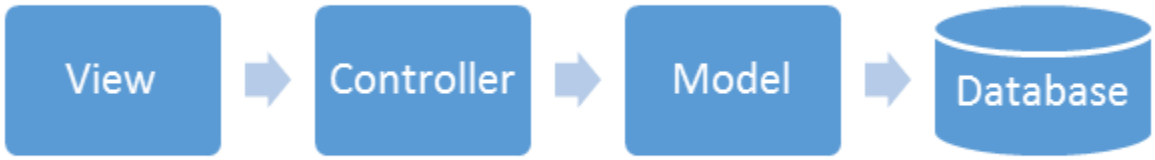
JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

Traditional MVC Architecture in Java

MVC is an architecture that separates business logic, presentation and data. In MVC,

- M stands for Model
- V stands for View
- C stands for controller.

MVC is a systematic way to use the application where the flow starts from the view layer, where the request is raised and processed in controller layer and sent to model layer to insert data and get back the success or failure message. The MVC Architecture diagram is represented below:



Model Layer

- This is the data layer which consists of the business logic of the system.
- It consists of all the data of the application

- It also represents the state of the application.
- It consists of classes which have the connection to the database.
- The controller connects with model and fetches the data and sends to the view layer.
- The model connects with the database as well and stores the data into a database which is connected to it.

View Layer

- This is a presentation layer.
- It consists of HTML, JSP, etc. into it.
- It normally presents the UI of the application.
- It is used to display the data which is fetched from the controller which in turn fetching data from model layer classes.
- This view layer shows the data on UI of the application.

Controller Layer

- It acts as an interface between View and Model.
- It intercepts all the requests which are coming from the view layer.
- It receives the requests from the view layer and processes the requests and does the necessary validation for the request.
- This requests is further sent to model layer for data processing, and once the request is processed, it sends back to the controller with required information and displayed accordingly by the view.

Advantages of MVC Architecture

The advantages of MVC are:

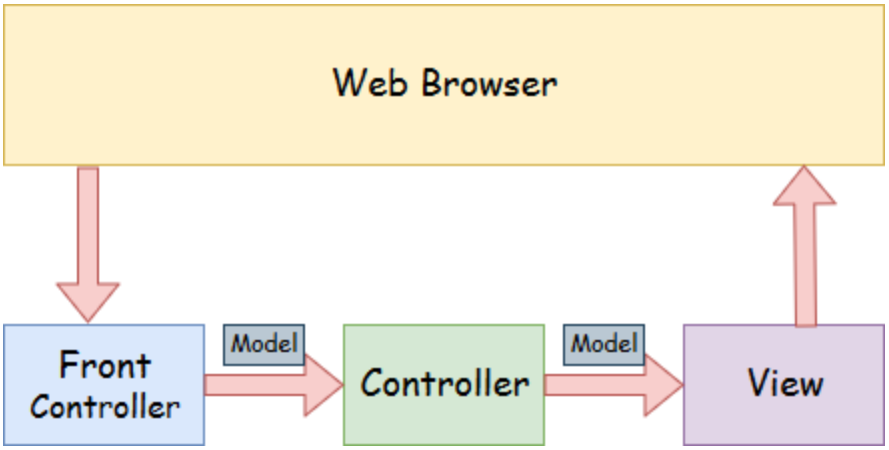
- Easy to maintain
- Easy to extend
- Easy to test

- Navigation control is centralized

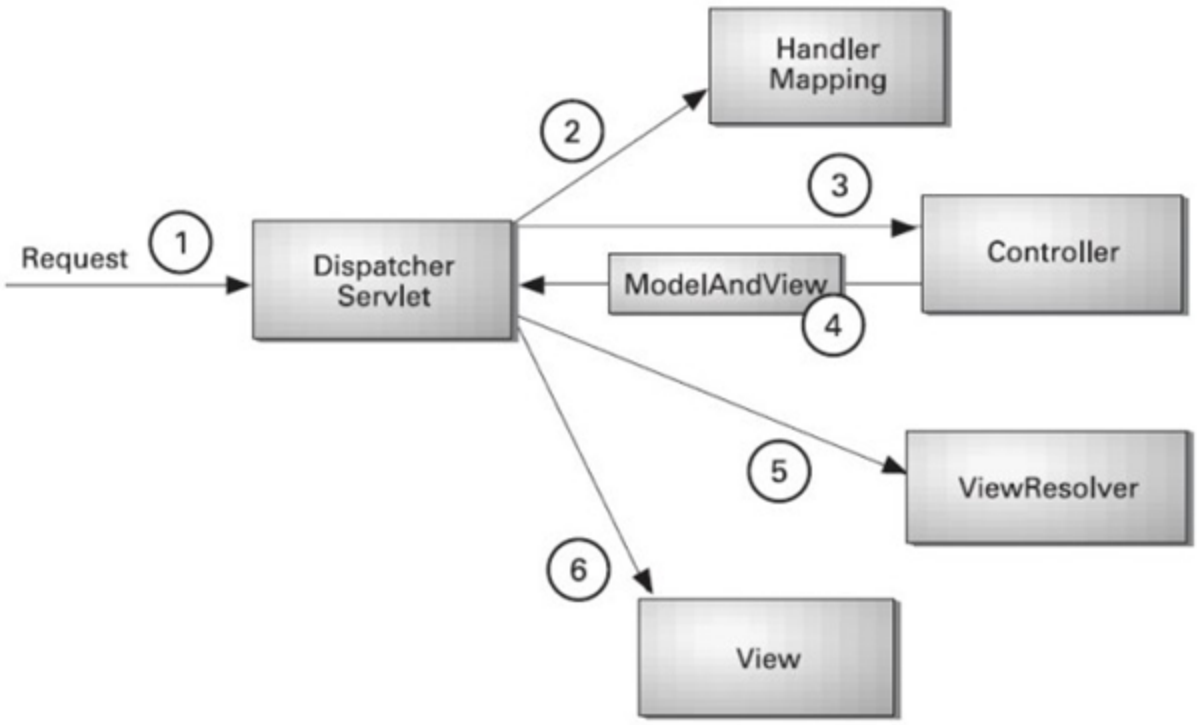
Spring MVC:

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.



- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
- **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.



- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.