SPRING MVC(MODEL VIEW CONTROLLER)

- Spring mvc which is used to build a web application
- We will build using Servlet API
- It follows the MVC design pattern(not followed by servlet and JSP).
- MODEL→DATA
- VIEW→Presentation(UI)
- Controller→Using Servlet(Flow control)
- It is the subframework of Spring.It implements all the basic features of a core spring framework like IOC,DI.
- WHY SPRING MVC?
- Separate each role model,view,controller.
- We can use spring core functionalities.(Make application loosely coupled)
- Application development faster.

MVC(MODEL VIEW CONTROLLER) Design Pattern

- MODEL→DATA.
- VIEW→ Presents data to user.
- Controller→ control the flow(Interface between model and view).
- Way to organize the code in the application.
- JSP(View →it converts into servlet.when the page loads it slow down.
- Servlet→ implementation of logic.(controller)
- Pojo class,data→model
- Working
- Client→request→Front controller(Dispatcher Servlet)→send the request to the concern→controller(Handles)→response send to front controller(response Including data→Model,name (to which page we need to send))→front controller use view resolver(To whom we need to show→Home.jsp(Dynamic))→Response sent to client.

JPA VS Hibernate(ORM)

JPA	Hibernate
Java Persistence API (JPA) defines the management of relational data in the Java applications.	Hibernate is an Object-Relational Mapping (ORM) tool which is used to save the state of Java object into the database.
It is just a specification. **Various ORM tools implement it for data persistence.**	It is one of the most frequently used JPA implementation.
It is defined in **javax.persistence** package.	It is defined in **org.hibernate** package.
The **EntityManagerFactory** interface is used to interact with the entity manager factory for the persistence unit. Thus, it provides an entity manager.	It uses **SessionFactory** interface to create Session instances.
It uses **EntityManager** interface to create, read, and delete operations for instances of mapped entity classes. This interface interacts with the persistence context.	It uses **Session** interface to create, read, and delete operations for instances of mapped entity classes. It behaves as a runtime interface between a Java application and Hibernate.
It uses **Java Persistence Query Language** (JPQL) as an object-oriented query language to perform database operations.	It uses **Hibernate Query Language** (HQL) as an object-oriented query language to perform database operations

SPRING MVC Steps

- ❑ Configure the **dispatcher servlet** in web.xml(Front Controller)
- ❑ Create Spring Configuration file(Beans declare,IOC container)
- ❑ Configure view resolver(InternalResourceviewResolver)
- ❑ Create Controller
- ❑ Create a view to show(Page)

Sending data from controller to view

- ❑ **Model**
- ❑ addAttribute(String key,Object value) →HttpServletRequest
- ❑ Object→Any class,collection etc
- ❑ To get→request.getAttribute(“key”);

- ❑ ModelAndView
- ❑ addObject (String key,Object value)

Interceptors

- ❑ Interceptors are generally used do some processing before handing it over to the controller handler methods

HandlerInterceptor interface defined 3 methods.

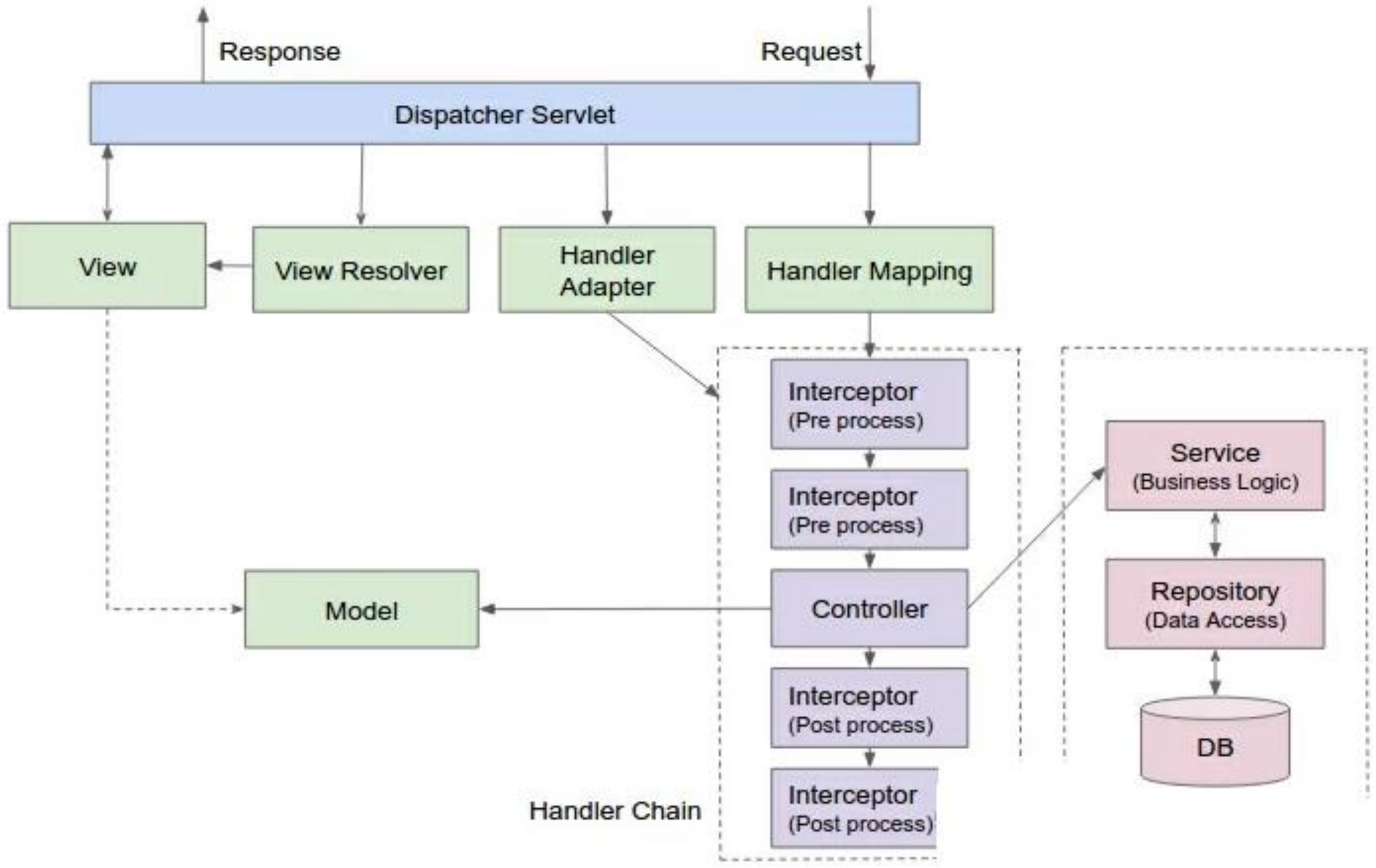
preHandle(request, response, handler) – Used to intercept the request before handed over to the handler method. Here handler is the chosen handler object to handle the request.

postHandle(request, response, handler, modelAndView) – Used to intercept the request after handler has completed request processing but **DispatcherServlet** is yet to render the view.

afterCompletion(request, response, handler, exception)

– It is called once the handler execution is complete and view is rendered as well.

Interceptors


Response
Request
Dispatcher Servlet
View
View Resolver
Handler Adapter
Handler Mapping
Interceptor (Pre process)
Interceptor (Pre process)
Service (Business Logic)
Repository (Data Access)
Model
Controller
Interceptor (Post process)
DB
Handler Chain
Interceptor (Post process)