**Spring MVC**

The Spring DispatcherServlet uses special beans to process requests and render the appropriate views.

These beans are part of Spring MVC.

You can choose which special beans to use by simply configuring one or more of them in the WebApplicationContext.

Spring’s MVC framework uses IoC to provide a clean separation of controller logic from business objects.

DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.

The model (the M in MVC) is a Map interface, which allows for the complete abstraction of the view technology.

DispatcherServlet is an expression of the "Front Controller" design pattern (this is a pattern that Spring Web MVC shares with many other leading web frameworks).

Spring’s web MVC framework is request-driven, designed around a central Servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications.

**SPRING MVC FLOW**

web.xml

loads the DispatcherServlet

context-param

contextConfigLocation

put the applicationcontext xml file name

One thing to note here is the name of servlet in <servlet-name> tag in web.xml.

Once the DispatcherServlet is initialized, it will looks for a file name [servlet-name]-servlet.xml in WEB-INF folder of web application.

[servlet-name]-servlet.xml configuration file

we have defined a tag <context:component-scan>.

This will allow Spring to load all the components from package com.controller and all its child packages.

viewResolver is configured. (InternalResourceViewResolver) (UrlBasedViewResolver)

where prefix points to jsp folder

and suffix is .jsp file to look up

When Spring scans the package, it will recognize this bean as being a Controller bean for processing requests.

The @RequestMapping annotation tells Spring that this Controller should process all requests beginning with /welcome in the URL path. That includes /welcome/\* and /welcome.html.

**Spring Interceptors**

Use *HandlerInterceptor* interface in your spring mvc application to pre-handle and post-handle web requests that are handled by Spring MVC controllers.

These handlers are mostly used to manipulate the model attributes returned/submitted they are passed to the views/controllers

A handler interceptor can be registered for particular URL mappings, so it only intercepts requests mapped to certain URLs.

Each handler interceptor must implement the HandlerInterceptor interface, which contains three callback methods for you to implement: preHandle(), postHandle() and afterCompletion().

Testing Spring MVC

MockMvc

**Exception Handling**

1.Using HTTP Status Codes

@ResponseStatus(value=HttpStatus.NOT\_FOUND, reason="No such Order") // 404  
public class OrderNotFoundException extends RuntimeException {  
 // ...  
}

2.Controller Based Exception Handling  
Using @ExceptionHandler

1.

a. Write a custom exception say postal code not found exception

annotate using ResponseStatus (HTTPStatus and reason)

b. throw the exception in ‘search Postalcode’ method setting the model in the controller class

2.

ExceptionHandler parameter (custom expections) // still have to write custom exceptions

Return to appropriate error models (jsps)

// Specify the name of a specific view that will be used to display the error:

@ExceptionHandler({SQLException.class,DataAccessException.class})

public String databaseError() {

// Nothing to do. Returns the logical view name of an error page, passed to

// the view-resolver(s) in usual way.

// Note that the exception is \_not\_ available to this view (it is not added to

// the model) but see "Extending ExceptionHandlerExceptionResolver" below.

return "databaseError";

}