TABLE OF CONTENTS

Chapter 1

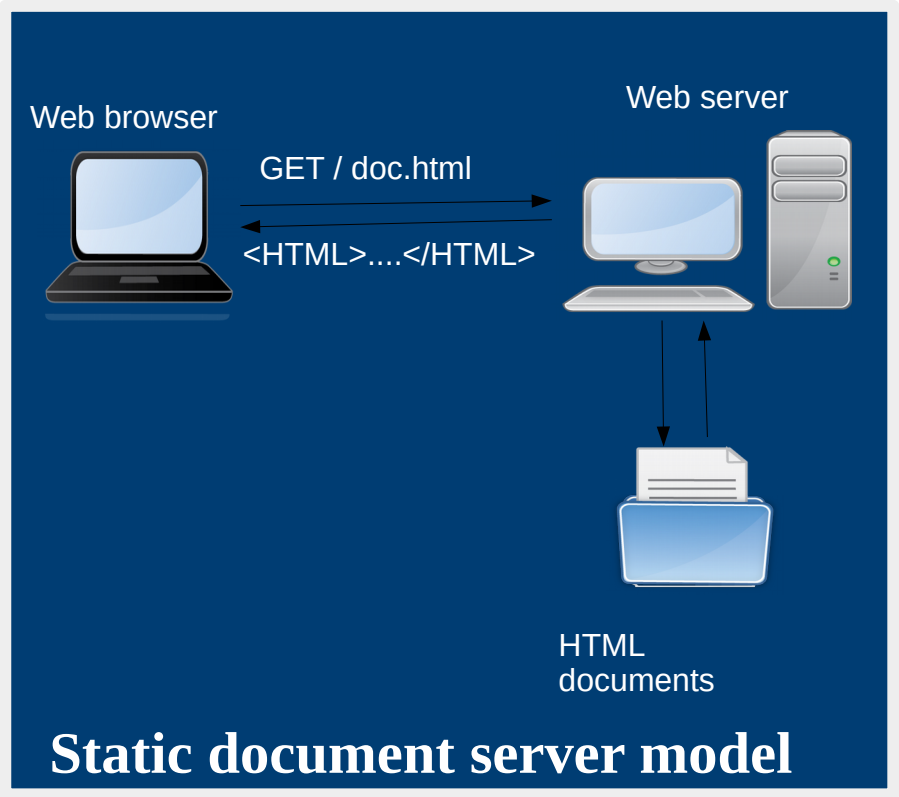
Introduction to Web Application

The world wide web was not developed as an application environment. It was just used for sending and receiving mails. But now it is mainly used for developing web applications- especially, e- commerce. In 1990, the world wide web and Hypertext Transfer Protocol grew as a result of the work done at European Laboratory for particle physics. HTTP was developed as a networking protocol by Tim Berners-Lee for distribubuting documents and developed the first Web browser. In 1991 and 1992, this system was used at CERN and other high energy physics laboratories and universities. In 1993, the usage of the Mosaic browser led to the explosion of commercial web use. In five years, more than 650,000 web servers were in use world wide with millions of users.

Growth of the Web Programming Model

i. Static document server model

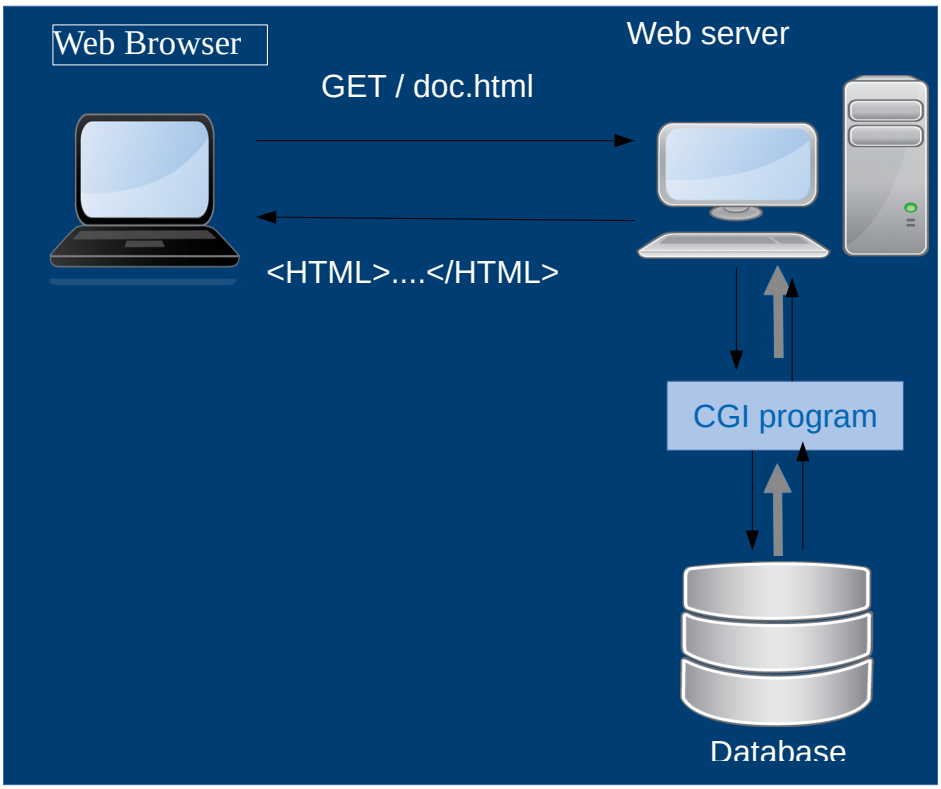
The Web server simply serving up documents on request was the first operational model. Unless a human author supplies a new version of a document, the content doesn’t change in this environment. The figure below shows the client/server interaction.



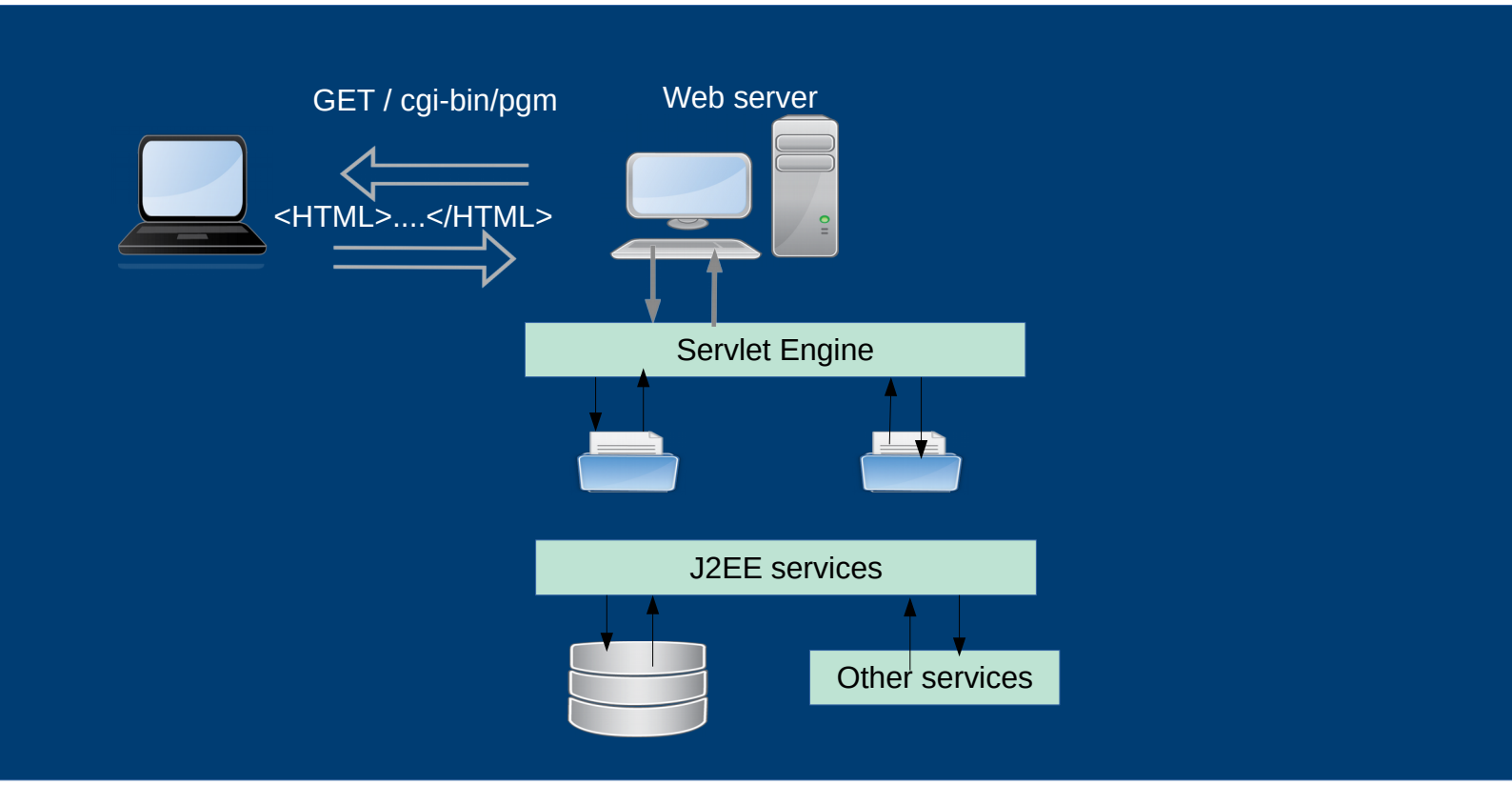
ii. Dynamic content generated by a CGI script:

While developing web applications using CGI script, we use HTTP protocol for data transfer. Other than that, we use scripting languages like pearl, ruby etc to make the client side changes. When data has to be retreived from the database, we use cgi scripting.

CGI is convenient, but it has a drawback. Using CGI scripting, each and every request is taken as a new process. So, it takes too much of CPU space. It is not a problem when the traffic is low, but it creates a great overhead when the traffic level increases CGI doesn’ t scale well.



iii. Dynamic web applications using servlets, JSP, and J2EE



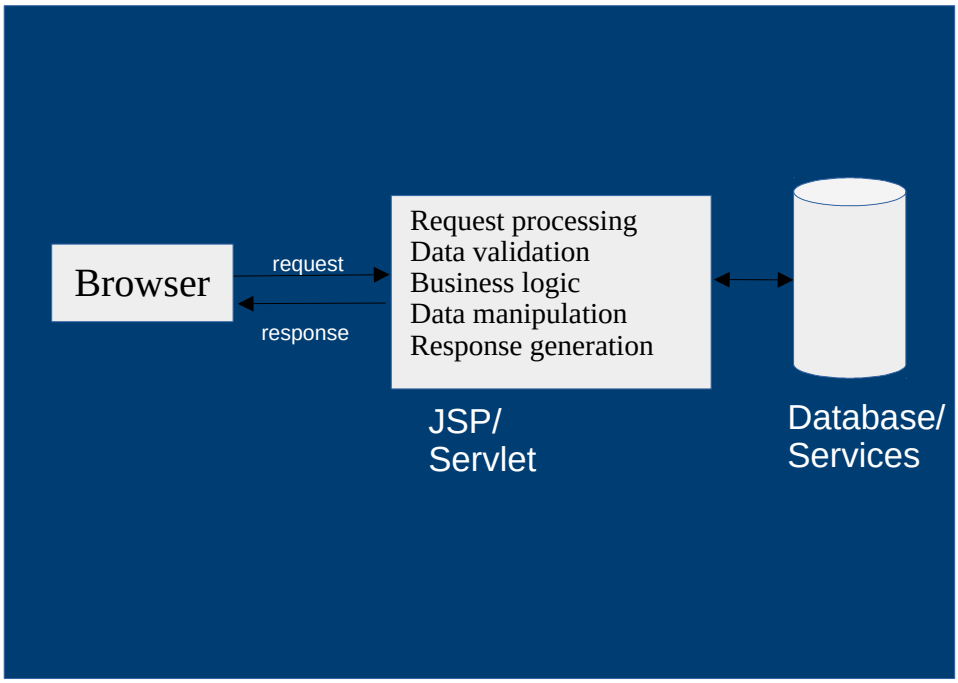
This technology was introduced in 1997. In the dynamic web applications using servlets, jsp and database, the Jsp acts as the client side user interface. The business logic and validations are moved to the servlet, so we have thin clients and fat servers. Earlier it was fat clients where validations were done.

TWO DEVELOPMENT MODELS

The web applications models are known as Model1 and Model2 and they provide different ways to develop jsp based web applications.

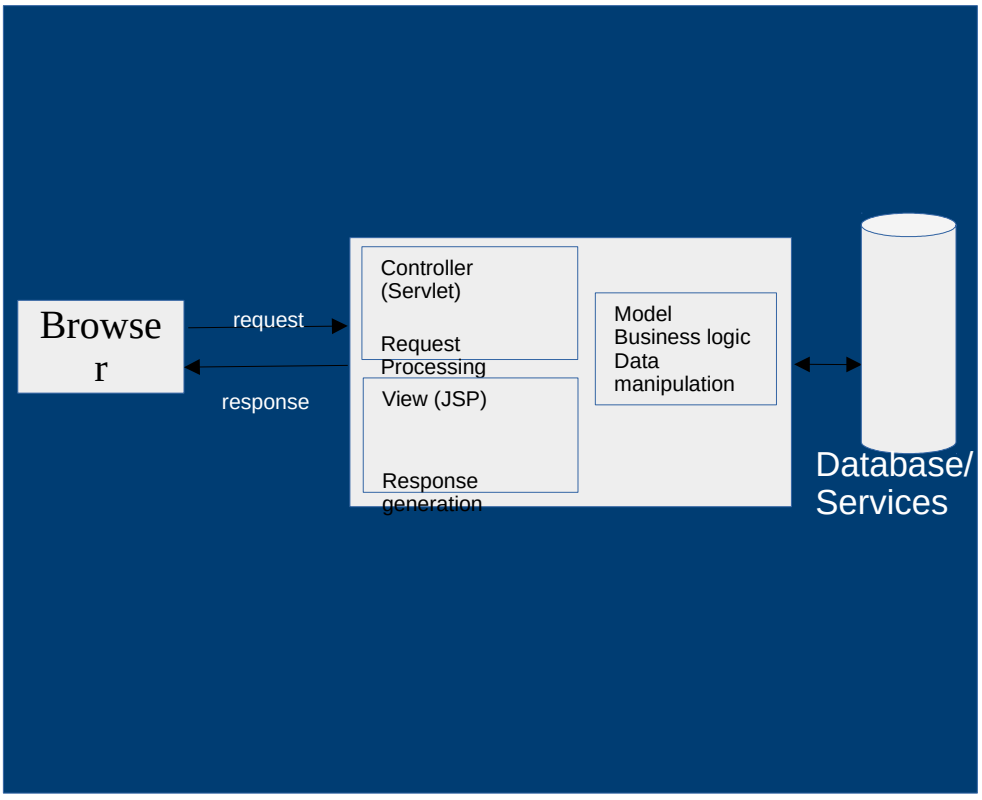
Model 1 Architecture:

In this architecture the request is made from the view page which is present in the web browser. This request is processed in the web container and the response is sent to the client machine's web browser. This Model1 architecture is simple, but when it comes to large-scale applications, the calling of functions causes congession.



Model 2 Architecture:

The Model 2 Achitecture is also known as the (Model- View- Controller) MVC Architecture. The view page is the jsp, html page etc which is the user interface. A request from the view page goes to the Controller. The controller validates the data and then calls the business logic which is written in the model page. So there is a complete separation between the presentation page and the business logic. It is the controller page which controls the flow of the program from one jsp page to the other.



Model: The model part in the mvc architecture contains the business logic and data manipulation functions. The model components can be simple java beans, web services or EJB components.

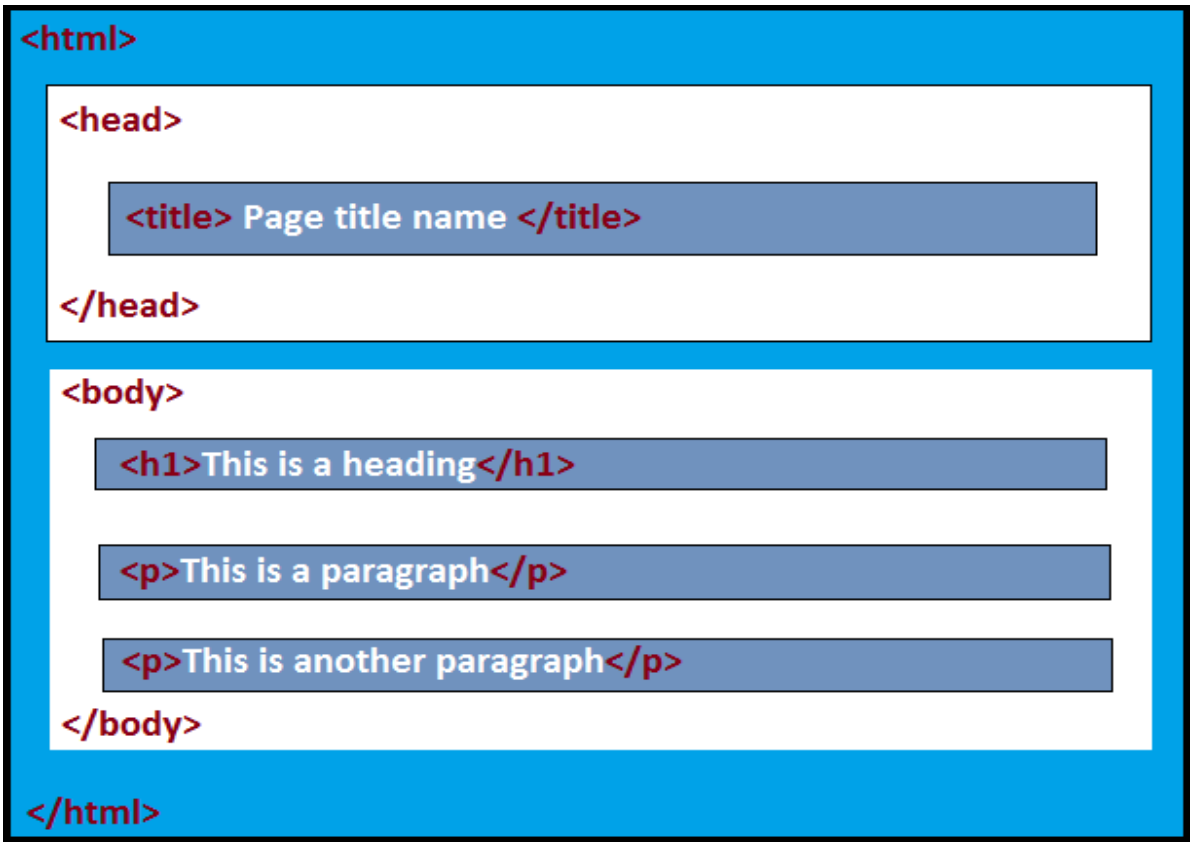
View: The view part in the mvc architecture contains user interface which can be opened in the browser. These view pages can be html, jsp, jsf pages etc.

Controller: The controller part in the mvc architecture contains the servlet in this case, which contains the request and response parameters. Every request has a process, the process is called using the model object. After processing, the response is sent to the client browser as per the instruction of the controller.

Chapter 2

Introduction to HTML

HTML stands for HyperText Markup Language. It is the standard markup language used for displaying data on the browser. It gives the structure to HTML data and have predefined tags used for displaying data in a particular format. HTML has different tags like <html></html>,   etc. The browser displays the data according to the tags used.



HTML page Structure

HTML Browsers:

There are a number of HTML browsers. These browsers can be used to display the html data. Some of the browsers most frequently used are: Chrome, Edge, Firefox, Opera etc.

HTML Elements:

The HTML tags used for displaying data in different formats is known as the html elements. Eg:

-    is for displaying data as a paragraph.
- <input type =”submit”/> is used for displaying a button.

HTML Attributes:

All HTML elements can hold additional details about that element, such elements are known as HTML attributes. Eg:

- In the tag element <input type = “text” name =”sid” value =”s001”/>
  Here, the textbox element has a name attribite which refers to the name of the textbox, it is “sid”. It has a value attribute which displays the value on the textbox.

HTML Tables:

HTML tables are used to display data in a tabular format. <table> </table> is used to display the table data. Inside the table, rows are displayed as <tr> </tr> and collumns as <td> </td>.

Eg:

<table>

<tr>

<td>Number <td>

<td>Square<td>

<tr>

</table>

This will display a table with one row and **number** and **square** as the two contents in the two collumns.

HTML Form elements:

The HTML <form> element defines a form that is used to collect user input. Form elements are different types of input elements like textfields, checkboxes, radio buttons etc.

HTML Form Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element
<fieldset>	Groups related elements in a form
<select>	Defines a drop-down list
<option>	Defines an option in a drop-down list
<button>	Defines a clickable button

HTML Form attributes:

The HTML Form attributes are:

- Form Handler
- The Action Attribute
- The Method Attribute

The Form Handler is used to process the input data. It contains scripts which are written to process the input data.

The Action Attribute is used to send the input data to a particular page. If any html or any other view page or any other page is not mapped then it returns to the same page from where the request was sent.

The Method Attribute specifies the HTTP method (GET and POST) methods. The Get method sends the data through the url and the Post method sends the data through the body data hence data is safe.

Chapter 3

Java Server Pages

Java Server Pages (JSP) is a template for web pages where Java code can be embedded into the HTML tags. JSP's are server side components known as a JSP container which translates the JSP code into its equivalent java servlet code.

The advantages of servlets:

- They have better performance and scalability than CGI Script because they are multithreaded and persistance in memmory.
- No special client setup is required.
- They have built in support for HTTP sessions which makes application programming possible.
- They have complete access to Java technology like network awareness, threads and database connectivity without the limitations of client side applets.

Advantages of JSP:

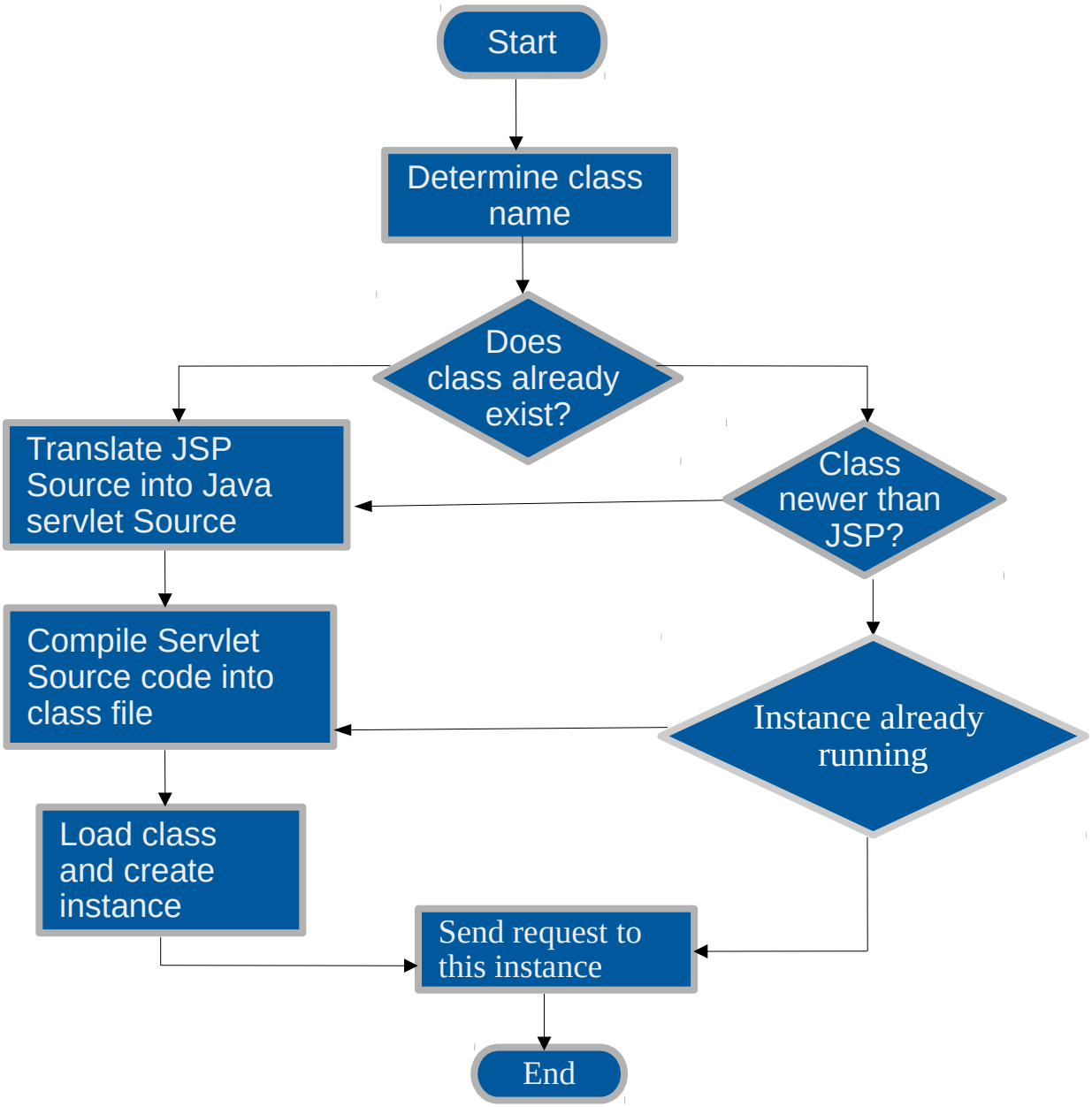
They are automatically recompiled when necessary. They exist in the ordinary web server document space, so addressing JSP pages is simpler than addressing servlets. JSP pages are HTML like, so they have greater compatibility with web development tools.

How the JSP Works:

A JSP page exists in three forms.

1. JSP Source Code:
   This is the form, the developer actually writes. It exists in a text file with an extension of .jsp and has a combination of HTML template code, java codes, jsp directives and actions that describes how to generate a web page to generate a particular request.
2. Java Source Code:
   The JSP Souce code is then converted into its corresponding Servlet code and this is present as the .java file. This source code is present in the work area.
3. Compiled code:
   The servlet code is then converted into its corresponding byte codes.These byte codes are the .class files which are ready to be loaded and executed.

Logic used by a JSP container to manage JSP translation



JSP Scripting Elements

A .jsp file can contain JSP elements, fixed template data or combination of the two.

Three types of JSP elements exist:

- Directives
- Scripting elements including expressions, scriplets and declarations.
- Actions

Directives:

These are instructions which tell the JSP container what code should be generated. They have the general form

<%@ directive-name [attribute="value" attribute="value" ...] %>

Three standard directives are available in all JSP environments. They are:

- page
- include
- taglib

The page directive:

The page directive is used to specify the attributes for the JSP page as a whole.

The syntax of the page directive is:

<%@ page [attribute="value" attribute="value" ...] %>

contentType, language, session, import etc are some frequently used page attributes.

The include directive:

The include directive is used to include another files contents into the existing .jsp page at translation time.

The syntax of the include directive is:
<%@ include file="filename" %>

Here the file can be any html file, jsp file etc.

The taglib directive:

The taglib directive is used to add certain tag libraries to the jsp page this makes certain actions possible.

The syntax of the taglib directive is:
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>

We can use tag libraries like JSTL, Spring taglibrary, Struts taglibrary etc.

Scripting Elements:

We use different scripting elements to provide various codes of java in the jsp page. Some of the scripting elements are:

- Comment tag
- Expression tag
- Scriplet tag
- Declarative tag

Comment tag:

JSP provides two types of comment tags. One type of comment tag which is visible only in the code and hidden in the browser. The other type of comment tag lets the comment to be visible in the output of the browser data.

The former type has the syntax
<%- - This is a hidden JSP comment - -%>

The latter looks like this:
<!- - This is included in the generated HTML - ->

Expression tag:

JSP provides the Expression tag to display the variable values or result of a function values in the browser.

The syntax of the Expression tag is:
<%= exp %>

Scriplet tag:

The scriplet tag is used in the jsp page to write the java code. In a JSP page, we have html elements. In between those html elements we can embed the java code to retrieve the required data. We use scriplet tags for this purpose.

The syntax of the Scriplet tag is:
<% %>

Declarative tag:

The declarative tag is used to declare variables and functions inside the jsp page. All other tags come inside the _jspService method, but the declaration tag can be used outside the _jspService() and called inside it. ie; it can be declared outside the <body> </body> tag.

The syntax of the declarative tag is:
<%! %>

Implicit Objects:

These variables are implicitly available inside scriptlets and expressions (but not declarations). They can be used just like any other variable, but need not be declared first.

Some Implicit objects are:

- request
- response
- pageContext
- session
- application
- out
- config
- page

Standard Actions:

Standard Actions are high level Java Objects that are used to create, modify or retrieve other objects. These objects can be javabean objects etc.

Some of the Standard Actions are:

<jsp:useBean>

<jsp:setProperty>

<jsp:getProperty>

<jsp:include>

<jsp:forward>

<jsp:param>

<jsp:plugin>

Chapter 4
Servlets

Servlets were introduced in the year 1997 and since then web technology immensly started using servlets for server side programming and application portal. They have many advantages compared to the earlier web technologies. They are:

- **Performance:** While preparing web applications using CGI technology, each request is taken as process and hence heavy. While using servlets, each request is taken as a thread and hence increases the performance of the application.
- **Simplicity:** With the servlet technology, the client applications could be run on the browsers and so no external technologies are needed.
- **HTTP Sessions:** Though each request is an object of the HTTP request object, the request object does not store the value for more than one request, but there is an HTTP session class whose object is the session object and this session object can hold data which is accessible througout that session.
- **Access to Java technology:** Servlets use the java technology, so they can access all the features of java such as threading, network access and database connectivity.

Servlet LifeCycle:

Servlets work on the request and response model which is managed by a servlet engine. The servlet engine works with the following life cycle methods:

- **init:** When the servlet recieves a request, it first loads the servlet. After loading the servlet before servicing any requests, it first calls the initialization method.

  public void init(ServletConfig config) throws ServletException

  This method is method is called just before the servlet is placed into service. The init() method is used to initialize any variable, to establish any database connectivity or to initialize any particular task.

- **Service:** After the init() method works completely, the service() method is called. It contains the doGet() and doPost() methods. The request is processed by either of the two methods depending on the **method attribute** of the form tag in the jsp or html page from where the request is produced. The default method value is “GET” and it will call the doGet() method if the method attribute is not mentioned. These methods have two parameters request and response which are the objects of HttpServletRequest and HttpServletResponse classes respectively.
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException;

- **destroy:** The servlet engine can unload a servlet at any time. Before unloading, it can call the destroy() method which destroys all the objects

serve that were initialized during the init() method. We can't call the servlet engine on its own. The servlet engine does it on its own.

Steps to create a Servlet and JSP example

1. Create a JSP page which is the User Interface from which the request is going to be generated.
2. The request from the JSP page is sent to the correspondingly mapped servlet.
3. The servlet has to be created. The servlet is a java file which processes the request from the JSP page. The request is processed using the doGet() or the doPost() method depending on the value given to the method attribute of the jsp page from where the request is produced. Both these methods have two parameters; the request and response objects which are the objects of HttpServletRequest and HttpServletResponse interfaces respectively.
4. The data written inside the method is again java code.
5. The data is processed and the result is returned through the RequestDispatcher object to the next JSP psge.

To understand the situation bettter we can look into a program using JSP User Interface, Servlet processing technology and databse connectivity

Chapter 5
The Spring Framework

Introduction to Spring Framework

The Spring Framework was developed by Rod Johnson in the year 2003. It was used for developing JavaEE applications in an easier way.Spring is not just a framework for developing web applications, it is a framework for developing desktop application, web application, mobile application and also a framework for developing Android applications.Spring is a lightweight Framework. It is lightweight because of the use of POJO (Plain Old Java Objects) for the development of java applications. When compared to simple web and ejb applications, spring applications are lightweight.It also provides solution to various technical problems which the earlier frameworks had.

Spring Framework can be thought of as a framework of frameworks. This is because the Spring framework provides support for various frameworks like Struts, Hibernate, Tapestry, EJB, JSF etc. Spring can be integrated with all these frameworks and projects can be written.

Spring framework also comprises of many modules like IOC, AOP, DAO, Context, ORM, Web MVC etc. Each of these modules are used for different purposes. We will cover each of these modules chapter wise in the later on.

Why Spring – Its Advantages:

Compared to the earlier technologies, the Spring Framework makes the working of the web application more easier. Even there are many advantages in its structural working. So, we can see what the advantages of the framework is:

- **Predefined Templates:** The spring framework provides templates for JDBC, Hibernate, JPA etc. These templates contains predefined methods. We just need to configure them in our spring project and use the functions in the code. We need not write lengthy codes for connectivity etc. That is what is the advantage of predefined templates, it simply reduces the code to be written for database connectivity and transactions.

- **Loose Coupling:** The spring framework is loosely coupled because of Dependency Injection. Using Dependency injection, let a class be dependant on another class. Example: say Student class has an Address class. Now, the Address can be assigned to the Student class through the default contructor or pass it as a parameter of the contructor of the Student class. When it is passed as a parameter of the Student class, we call it dependency injection because even if a change occurs on the Address class, it will automaically get updated when the Student object is created. This is dependency injection.

- **Easy testing:** Dependency injection will make testing of application developed in Spring very easy. Spring doesn't need any servers to run the program.

- **Light Weight:** Spring framework is a lightweight framework. It is light weight because of the implementation of POJO (Plain Old Java Objects) for the creation of java applications. In the earlier frameworks we used to prepare heavy weight components to prepare web applications, but now we just use simple java class files in Spring framework.

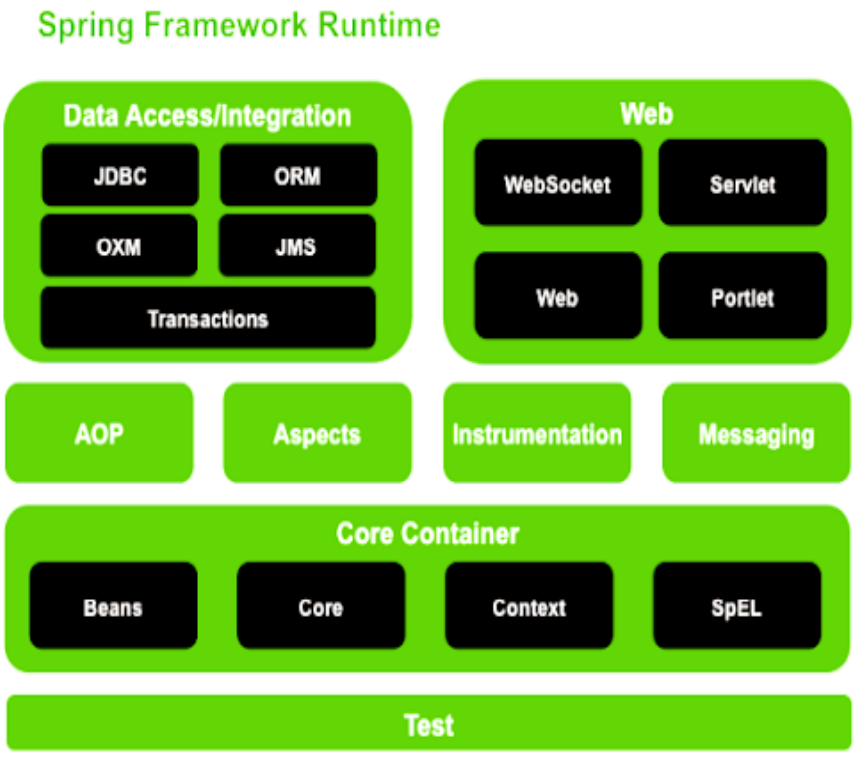
- **Fast Development:** The development part is made easier and faster in the Spring framework by the use of injection. Objects can be injected into the framework and this makes value assignation faster.

- **Powerful Abstraction:** It provides abstraction to JavaEE application specification like JMS, JDBC, JPA, JTA etc. All these features have predefined API's in the spring framework and hence provides abstraction.

- **Declarative Support:** It also provides certain predefined declarations like for validation, caching, transactions and formatting there are predefined classes and methods for supporting these declarations.

Spring Modules:

The spring framework comprises of many modules. Each module is used for preparing a particular kind of application.The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



Chapter 6

Spring IOC

Introduction to Spring IOC

The IOC container is responsible to instantiate, configure and assemble the objects. The information is passed to the IOC container through the XML files and they work accordingly. The main taks performed by the IOC container are:

- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects

There are two types of IOC containers. They are:

i. Bean Factory
ii. ApplicationContext

The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

Dependency Injection in Spring:

Dependency Injection is a design pattern that removes the dependency from the programming code. Here the data is passed externally through an xml file. So it makes the applications easy to manage and test. Dependency injection also makes our programming code loosely coupled.

```
class Student{

Address address;

Student(Address address){
this.address=address;
}
public void setAddress(Address address){
this.address=address;
}

}
```

Here instance of the Student class is used to assign values and these values can be passed through xml file. There are two ways to assign values to the variable

- by constructor
- by getter and setter method

To understand the situation bettter we can continue with programs of Dependency Injection using constructor and getter and setter method.

Chapter 7

Spring AOP

Aspect Oriented Programming complements the Object Oriented Programming on the concept of modularity. In the OOP's environment, it is the class which causes modularity while in Aspect Oriented Programming, it is Aspect which causes modularity.

The programming logic is divided into distinct parts known as concerns in AOP. It increases modularity by cross-cutting concerns. A cross-cutting concern is a concern that can affect the entire application, so it has to be centralised in one location in the code as possible such as transaction management, authentication, logging, security etc.

Where is AOP used?

AOP is mostly used in cases where we have to provide declarative enterprise services such as declarative transaction management. It also allows users to implement custom aspects. It can also be used for Performance Monitoring. By using AOP you can easily monitor the execution time of any method. So before calculating start time and after calculating end time, that logic should be in Aspect class

AOP Concepts:

- **Join point:** It is any point in our program such as method execution, exception handling, field access etc. Spring supports only method execution join point.

- **Advice:** It represents the action taken by an aspect at a particular join point. We have different types of advices:
  - Before Advice: It executes before a join point.
  - After Returning Advice: It executes after a join point completes normally.
  - After Throwing Advice: It executes if a method throws an exception and then exits.
  - After (finally) Advice: It executes after a join point irrespective of if join pont exits normally or if an exception is thrown by the method.
  - Around Advice: It executes before and after a join point.

- **Pointcut:** It is an expression language of AOP that matches join point.

- **Introduction:** It is introduction of additional method and fields for a type. It allows you to introduce new interface to any adviced object.

- **Target Object:** It is the object that is being advised by one or more aspects. It is also known as proxied object as the Spring AOP is implemented using runtime proxies.

- **Aspect:** It is a class which contains advices, join points etc.

- **Interceptor:** It is an Aspect which contains only one advice.

- **AOP Proxy:** It is used to implement aspect contracts, created by AOP frameworks.

- **Weaving:** It is the process of linking aspect with other application types or objects to create an advised object Weaving can be done at compile time, load time or run time. Spring AOP performs weaving at runtime.

Chapter 8

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. Spring Framework uses DispatcherServlet to implement MVC in spring framework. Here DispatcherServlet is a class that receives the incoming request and maps it to the right resource such as controllers, models and views.

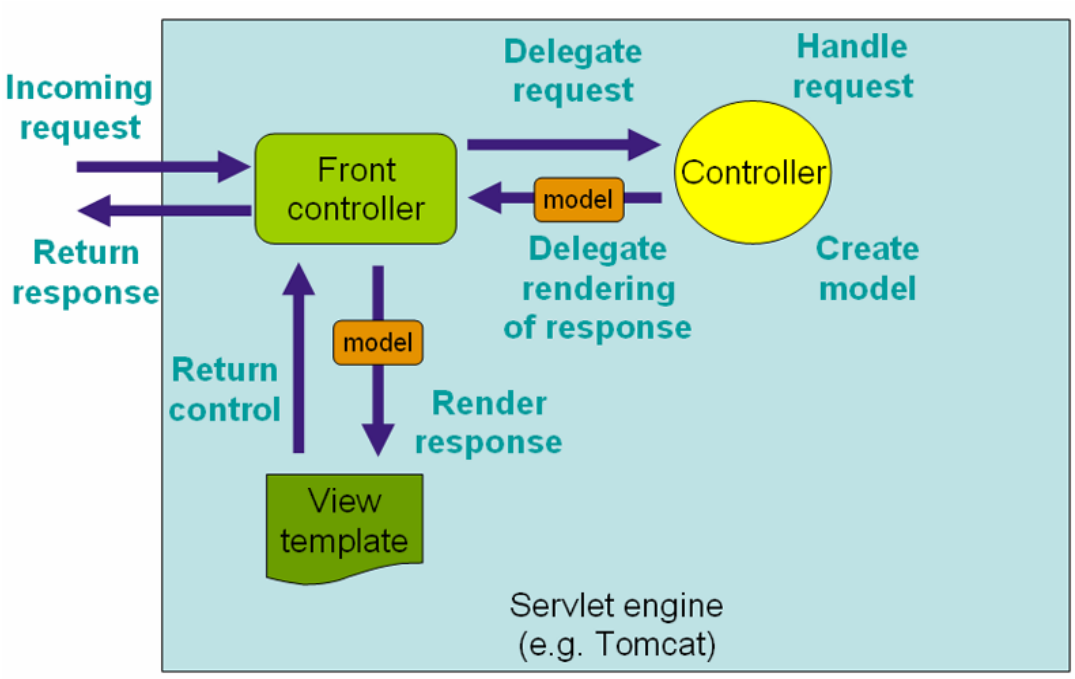
The DispatcherServlet dispatches requests to handlers with configured handler mappings, view resolution, locale, theme resolution and support for uploading files. The default handler is based on the @Controller and @RequestMapping annotations which offers a wide range of flexible handling methods.

Spring Web Model – View - Controller

- **Model –** A model contains the data of the application. A data can be 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 –** The view represents the given data in a particular format. JSP+ JSTL is ued to represent a view page. Spring also supports JSF, Thymeleaf and FreeMarker.

- **FrontController -** In SpringMVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the SpringMVC application.

Understanding the flow of Spring Web MVC



The flow of the program is in this order:

View Page: The request for a particular action is started from the view page. From here it moves to the suitable controller.

Configuration page: The controller is configured using the @ComponentScan annotation in the configuration page. The Spring MVC configuration can be done either using the **xml mapping** or **annotation method**. To maintain the flow of control of springMVC applications, It has to be mapped in the configuration file.

Controller page: The controller page is annotated with the @controller annotation. The controller method is mapped with a url value. This method then calls the the model dao method to perform execution.

Dispatching Data: The Model object in the method parameter of the controller can set the atribute or call the DAO methods and set the value using Model.setAttribute() method and return to the jsp page by mentioning the jsp page name as string value of the return keyword.

Jsp Page: The Jsp page uses Expression Language or Expression tags to display the value which is mapped in the corresponding controller mapping.The jsp page again is mapped using @ViewResolver annotation in the configuration page.

We can understand the situation better by writing a small program showing the flow of the SpringMVC Application.

SpringMVC Forms:

In a Spring web application, there are chances that we enter data in a form. In such a situation, we use jsp form tag. We can also use Spring form tag libraries. So, these form tag libraries make the transfer of the data between the form tags in the jsp page possible.

SpringMVC CRUD Operations:

Using the SpringMVC Forms, we can insert, update, retrieve and delete data.
To perform such actions, we have to first connect the spring web application to the required database. We can connect to any type of database. The different databases are: MySQL, Oracle, SQL Server, SQLlite, MongoDB etc.

We can understand the situation better by writing a small program showing the CRUD operations in SpringMVC Application.

SpringMVC Annotations:

SpringMVC provides different types of Annotations.

- **@Service, @Component and @Repository**, they are all used to represent bean class. They are used to annotate on our class according to the meaning and context of the application.

- **@Autowired** is used to inject value to a variable.

- **@Configuration** is used to represent a class as a configuration class.

- **@ComponentScan** is used to tell the compiler to scan through all the classes in a particular package.

- **@RequestMapping** is used to map the controller method with a url name to tell the compiler which view page to access from a particular view page.

- **@PropertySource** is used to assign the source of a property variable.

- **@EnableTransactionManagement** is used to provide the required transaction environment related files to enable transactions in the application.

- **@Transactional** is used to enable transactions in an application.

- **@Controller** is used to tell the Spring compiler that a class is a controller.

- **@EnableWebMVC** is used to enable web applications and enable redirect Attribute in the controller class.

- **@InitBinder** is used to bind a targeted bean class to its corresponding validation class.

- **@ModelAttribute** is used to bind the form values.

- **@Validated** is used to represent a form validator.

- **@RequestParam** is used to bind the request parameter to the method parameter of the controller class.

- **@PathVariable** is used to bind a method argument to the value of the URI template variable.

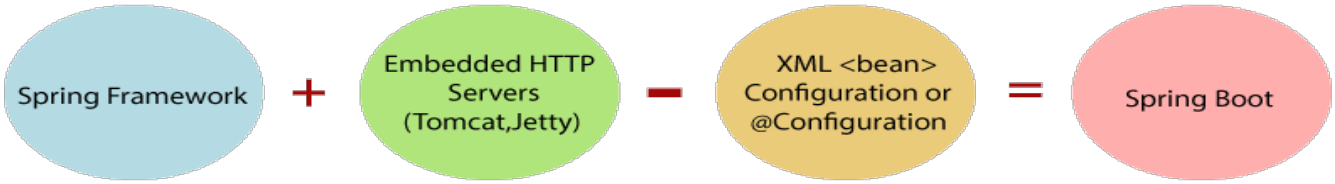
- **@ResponseBody** will automatically convert the return value to a string and write it to the HTTPResponse.

Chapter 9

Spring Boot

Using Spring Boot, we can easily create stand – alone, production-grade Spring based Applications that can just run. Using Spring Boot framework, we take an optionated view of the Spring platform and third-party libraries so you can get started with minimum confusion. Most Spring Boot applications need minimal Spring configuration.

Spring Boot provides the RAD (Rapid Application Development) feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration. Spring is a combination of Spring Framework and Embedded Servers.



www.javatpoint.com

In Spring Boot there is no need for XML configuration. To decrease the effort of the developer it uses convention over configuration software design paradigm. To develop Spring Boot Java applications, we can use Spring STS IDE or Spring Initializr.

Features of Spring Boot Framework:

- Stand-alone Spring applications can be created using Spring Boot Framework.
- Using Spring Boot Framework Tomcat, Jetty or Undertow are embedded directly (no need to deploy WAR files).
- The Spring Boot Framework provides opinionated 'starter' dependencies to simplify your build configuration.
- The Spring Boot Framework automatically configures Spring and 3rd party libraries whenever possible.
- The Spring Boot Framework provides production-ready features such as metrics, health checks, and externalized configuration.
- The Spring Boot Framework has absolutely no code generation and no requirement for XML configuration.

Spring vs Spring Boot:

Spring Framework:

- For building Java EE applications, Spring is one of the most widely used Frameworks.
- An elaborate programming and configuration model is provided by the Spring Framework.
- The Spring Framework helps developers make work more productive by simplifying the Java EE development.
- The Spring Framework can be used at any kind of deployment platform.

- Spring Framework focuses on various areas of an application and provides a wide range of features.
- Dependency injection is one of the major features of the Spring framework.
- By allowing us to develop loosely coupled applications, the Spring Framework helps make things simpler.

Spring Boot:

- Spring Boot shortens the code length and provides the easiest way to develop a web application, while the Spring framework focuses on providing flexibility.
- Spring Boot shortens the time involved in developing an application by using annotation configuration and default codes.
- With less than or almost zero-configuration, the Spring Framework helps create a stand-alone application.
- Depending on the requirement a class is automatically configured.This special feature of Spring Boot is known as Autoconfiguration.

Benefits of the Spring Boot:

We don’t have to deploy WAR files in the Spring Boot Framework.

- Stand-alone applications are created using Spring Boot Framework.
- Tomcat, Jetty, or Undertow can be embeded directly in the Spring Boot Framework.

- The Spring Boot Framework doesn’t require XML configuration.
- LOC is the reduced by using the Spring Boot Framework.
- Production ready features are provided by the Spring Boot Framework.
- It is easier to launch an application in the Spring Boot Framework.
- The Spring Boot Framework provides easier customization and management.

Benefits of the Spring Framework:

- In the development of an application, the Spring framework can be used for all layers of implementation.
- It is a very lightweight framework because of its POJO model.
- Loose coupling and easy testability is provided by the Spring Framework.
- Declarative programming is supported by the Spring Framework.
- The formation of singleton and factory classes can be eliminated, this is the capability of the Spring Framework.
- Both XML and annotation configurations are supported by the Spring Framework.
- Middleware services are provided by the Spring Framework.

The added advantages that comes with Spring Boot are of great value to the developers as they offer completion of projects with very little efforts. To all the problems that arise from the Spring framework, Spring Boot is the solution.

Spring Boot Application Properties:

Common application properties can be used to configure your application. Various properties can be specified inside the application.properties file, inside the application.yml file, or as command line switches. We can divide the Application properties into 16 major groups. Apart from this we can create our own properties.

Application Property class division:

These are some of the major Application property class divisions:

Core properties	**Cache properties**
Mail properties	**JSON properties**
Data properties	**Transaction properties**
Data migration properties	**Integration properties**

Web properties	**Templating properties**
Server properties	**Security properties**
RSocket properties	**Actuator properties**
Devtools properties	**Testing properties**

Building REST services with Spring:

REST has quickly become the standard for building web services on the web because they’re easy to build and easy to consume. There’s much to learn how REST fits in the world of microservices, but - for this tutorial - let’s just look at building RESTful services. REST enhancess the precepts of the web, including its architecture, benefits, and everything else.

Building on top of HTTP, REST APIs provide the means to build flexible APIs that can:

- Support backward compatibility
- Evolvable APIs
- Scaleable services

- Securable services
- A spectrum of stateless to stateful services

RestController in the controller class indicates that the data returned by each method will be written straight into the response body instead of rendering a template.

RESTful Services:

For JSON output, Spring HATEOAS, a Spring project aimed at helping write hypermedia-driven outputs. To upgrade your service to being RESTful, add this to your build:

Adding Spring HATEOAS to pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
```

This tiny library will give us the constructs to define a RESTful service and then render it in an acceptable format for client consumption.

A critical ingredient to any RESTful service is adding links to relevant operations. To make your controller more RESTful, add links like this:

Getting a single item resource

```
@GetMapping("/employees/{id}")
EntityModel<Employee> one(@PathVariable Long id) {
```

```
  Employee employee = repository.findById(id) //
      .orElseThrow(() -> new EmployeeNotFoundException(id));

  return EntityModel.of(employee, //
      linkTo(methodOn(EmployeeController.class).one(id)).withSelfRel(),
      linkTo(methodOn(EmployeeController.class).all()).withRel("employees"));
}
```

Restful Controllers:

Throughout we have been engaged in various tactics to build REST API. As it turns out, REST isn't just about pretty URIs and returning JSON instead of XML.

- This concludes on how to build RESTful services using Spring.
- nonrest - Simple Spring MVC app with no hypermedia
- rest - Spring MVC + Spring HATEOAS app with HAL representations of each resource
- evolution - REST app where a field is evolved but old data is retained for backward compatibility
- links - REST app where conditional links are used to signal valid state changes to clients

RestController vs Controller:

Controller: The Spring MVC traditional work flow relies on the view technology, the RESTful webservice controller returns object and the Object data is sent directly to the HTTP response as JSON/XML.

RestController: In the traditional SpringMVC where @Controller is used we can implement RESTful webservice by returning the output directly from the controller using @ResponseBody.

Using @RestController is similar to using @Controller + @ResponseBody. It is implemented from the Spring 4.0 version onwards.

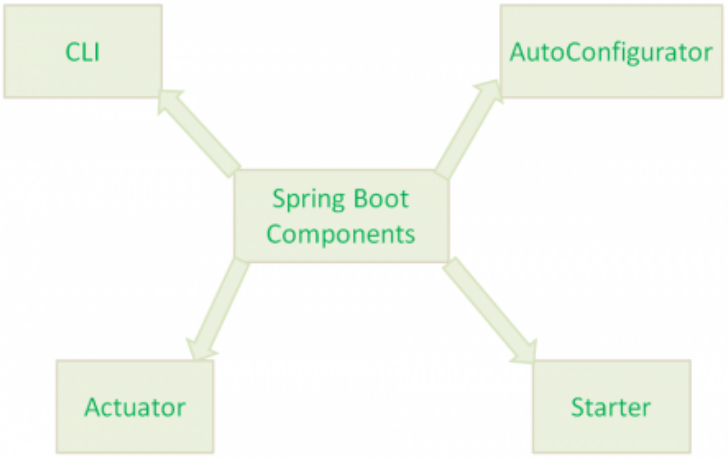
Spring Boot Service Layer:

- In a SpringMVC Architecture, we have the Model, View and Controller components to control the flow of data.
- In the Spring DAO model, the entire code for persistance is written in the DAO class.
- By the introduction of the Service layer, loose coupling is implemented. It handles the business requirements on the topmost layer.
- The service layer calls the DAO methods after validations if any.
- The main importance of the service layer is that the controller doesn't have to call the service method. It just has to forward to the view page.

- The service is a named component and they are called in the view page.

- So we can conclude that the service layer provides separation of concern, security and loose coupling in the application.

Key Components of Spring Boot Framework:



The Spring Boot Framework has mainly four Components.

- **Spring Boot Starters:** Spring Boot Starters is one of the major components of Spring Boot Framework. The main responsibility of Spring Boot Starter is to combine a group of common or related dependencies into single dependency.
- **Spring Boot AutoConfigurator:** The main responsibility of Spring Boot AutoConfigurator is to reduce the Spring Configuration. If we develop Spring applications in Spring Boot, then We dont need to define single XML configuration and almost no or minimal Annotation configuration. If we use @SpringBootApplication annotation at class level, then Spring Boot AutoConfigurator will automatically add all required annotations to Java Class ByteCode.
- **Spring Boot CLI:**Spring Boot CLI(Command Line Interface) is a Spring Boot software to run and test Spring Boot applications from command

prompt. When we run Spring Boot applications using CLI, then it internally uses Spring Boot Starter and Spring Boot AutoConfigurate components to resolve all dependencies and execute the application.

- **Spring Boot Actuator:** When we run our Spring Boot Web Application using CLI, Spring Boot Actuator automatically provides hostname as “localhost” and default port number as “8080”. We can access this application using “https://localhost:8080/” end point.

Spring Boot Utils:

The Utils class in Java Spring Boot Applications is used for Session related activities like:

- Accessing a particular Session attribute.
- Creating a particular Session attribute.
- Removing a particular Session attribute.
- Storing a particular session attribute.
- Accessing the Stored session attribute.

We can understand the situation better with a Spring Boot example

Hudson - Continuous Integration Server:

Hudson is a continuous integration (CI) tool written in Java, which runs in a servlet container, such as Apache Tomcat or the GlassFish application server. It supports SCM tools including CVS, Subversion, Git and Clearcase and can execute Apache Ant and Apache Maven based projects, as well as arbitrary shell scripts and Windows batch commands.

- Hudson is "best in its class" is because of its simplicity to implement.
- A new user can get an instance up and running without any expertise.
- An experienced user can leverage its tremendous capabilities.

Jenkins Overview:

Jenkins is a free and open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands.

- The Jenkins project was originally named Hudson.

- Oracle's Hudson is no longer maintained and was announced as obsolete in February 2017.

- Jenkins replaced Hudson since February 8, 2017 in Eclipse.

- In March 2018 Jenkins X software project for Kubernetes was publicly presented, with support for different cloud providers including AWS EKS among others.

Apache Tomcat:

- The Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies is an open source implementation of the Apache Tomcat® software.

- Under the Java Community Process, the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket specifications are developed.

- Released under the Apache License version 2, the Apache Tomcat software is developed in an open and participatory environment.

- From around the world, the Apache Tomcat project is intended to be a collaboration of the best-of-breed developers.

- Across a diverse range of industries and organizations, the Apache Tomcat software powers numerous large-scale, mission-critical web applications.

- Apache Tomcat, Tomcat, Apache, the Apache feather, and the Apache Tomcat project logo are trademarks of the Apache Software Foundation.

Deploying to the cloud:

- For most popular cloud PaaS (platform-as-a-service) providers, Spring Boot's executable jars are ready-made.

- These providers need some intermediary layer that adapts your application to the cloud's notion of a running process so they tend to require that you "bring your own container"; they manage application processes (not Java applications specifically).

- The buildpack wraps your deployed code in whatever is needed to start your application: it might be a JDK and a call to java, it might be an embedded web server, or it might be a full-fledged application server. Two popular cloud providers, Heroku and Cloud Foundry, employ a "buildpack" approach.

- A buildpack is pluggable, but ideally you should be able to get by with as few customizations to it as possible. This reduces the footprint of functionality that is not under your control. It minimizes divergence between development and production environments.

- Ideally, your application, like a Spring Boot executable jar, has everything that it needs to run packaged within it.

Chapter 10

Jackson

- The Jackson Project, earlier known as the standard JSON library for Java (or JVM platform in general), or, as the "best JSON parser for Java." Or simply as "JSON for Java." More than that, Jackson is a suite of data-processing tools for Java (and the JVM platform), including the flagship streaming JSON parser / generator library, matching data-binding library (POJOs to and from JSON) .

- Jackson JSON Java Parser is very popular and used in Spring framework too. Java JSON Processing API is not very user friendly and doesn't provide features for automatic transformation from Json to Java object and vice versa. Luckily we have some alternative APIs that we can use for JSON processing.

- **jackson-databind jar** depends on **jackson-core and jackson-annotations libraries,** so if you are adding them directly to build path, make sure you add all three otherwise you will get runtime error.

Jackson Annotations:

Annotations (jackson-annotations) contains standard Jackson annotations.

- @JsonProperty (also indicates that property is to be included) is used to indicate external property name, name used in data format (JSON or one of other supported data formats)
- @JsonAutoDetect: class annotation used for overriding property introspection definitions.
- @JsonIgnore: simple annotation to use for ignoring specified properties.
- @JsonIgnoreProperties: per-class annotation to list properties to ignore, or to indicate that any unknown properties are to be ignored.
- @JsonIgnoreType: per-class annotation to indicate that all properties of annotated type are to be ignored.
- @JsonInclude: annotation used to define if certain "non-values" (nulls or empty values) should not be included when serializing; can be used on per-property basis as well as default for a class (to be used for all properties of a class).
- @JsonPropertyDescription: Annotation used to define a human readable description for a logical property.

Deserialization and Serialization details

- @JsonFormat: general annotation that has per-type behavior; can be used for example to specify format to use when serializing Date/Time values.
    - lenient (boolean)
    - locale (String)
    - pattern (String)
    - shape (Shape)
    - timezone (String)

    - with (JsonFormat.Feature[])
    - without (JsonFormat.Feature[])

- @JsonUnwrapped: property annotation used to define that value should be "unwrapped" when serialized (and wrapped again when deserializing), resulting in flattening of data structure, compared to POJO structure.

- @JsonView: property annotation used for defining View(s) in which property will be included for serialization, deserialization.

Deserialization details:

- @JacksonInject: annotation to indicate that property should get its value via "injection", and not from data (JSON).

- @JsonAnySetter: annotation used for defining a two-argument method as "any setter", used for deserializing values of otherwise unmapped JSON properties

- @JsonCreator: annotation used for indicating that a constructor or static factory method should be used for creating value instances during deserialization.

- @JsonSetter: alternative to @JsonProperty, for marking that specified method is a "setter-method"

- @JsonEnumDefaultValue (added in 2.8): annotation used for defining a default value when deserializing unknown Enum values. Requires config READ_UNKNOWN_ENUM_VALUES_USING_DEFAULT_VALUE feature to be enabled.

Serialization details

- @JsonAnyGetter: annotation used to define a getter as "any getter", which returns a java.util.Map, contents of which will be serialized as additional properties for JSON Object, along with regular properties that the Object may have.

- @JsonGetter: alternative to @JsonProperty, for marking that specified method is a "getter-method"

- @JsonPropertyOrder: annotation for specifying order in which properties are serialized

- @JsonRawValue: per-property marker that can be used to specify that the value of property is to be included in serialization "exactly" as is, with no escaping or decoration -- useful for embedding pre-serialized JSON (or whatever data format is being used) in output

- @JsonValue: per-property marker to indicate that the POJO should serialization is to be done using value of the property, often a java.lang.String (like annotation toString() method)

- @JsonRootName: class annotation used to indicate name of "wrapper" entry used for root value, if root-wrapping is enabled

Type handling:

- @JsonSubTypes: class annotation used to indicate sub-types of annotated type; necessary when deserializing polymorphic types using logical type names (and not class names)

- @JsonTypeId: property annotation used to indicate that the property value should be used as the Type Id for object, instead of using class name or external type name.

- @JsonTypeInfo: class/property annotation used to indicate details of what type information is included in serialization, as well as how.

- @JsonTypeName: class annotation used to define logical type name to use for annotated class; type name can be used as Type Id (depending on settings of @JsonTypeInfo)

Object references, identity:

- @JsonManagedReference, @JsonBackReference: pair of annotations used to indicate and handle parent/child relationships expressed with pair of matching properties

- @JsonIdentityInfo: class/property annotation used to indicate that Object Identity is to be used when serializing/deserializing values, such that multiple references to a single Java Object can be properly deserialized. This can be used to properly deal with cyclic object graphs and directed-acyclic graphs.

Chapter 11

Template Engines for Spring

The Spring web is built around the SpringMVC (Model-View-Controller) pattern which brings separation of concern in an application. This allows the use of various view technologies from JSP to a variety of template engines. We're going to take a look at the main template engines that can be used with Spring like Thymeleaf and Velocity.

Spring Boot Thymeleaf:

Thymeleaf is a Java template engine which can process HTML, XML, text, JavaScript or CSS files. Unlike other template engines, Thymeleaf allows using templates as prototypes, meaning they can be viewed as static files. Thymeleaf templates are very similar in syntax to HTML templates.

Some of the features that are available when using Thymeleaf in a Spring application are:

- support for defining forms behavior
- binding form inputs to data models
- validation for form inputs
- displaying values from message sources
- rendering template fragments

Spring Boot Velocity:

Velocity is a template engine from the Apache Software Foundation that can work with normal text files, SQL, XML, Java code and many other types. Here we're going to focus on utilizing Velocity with a typical Spring MVC web application.

To create a Spring Boot Velocity:

- The newest versions of both velocity and velocity-tools have to be installed
- Here we are telling Spring where to look for annotated bean definitions:
- <context:component-scan base package="org.techmindz.mvc.velocity.*" />
- We are indicating we are going to use annotation-driven configuration in our project with the following line:
- <context:annotation-config />
- By creating “velocityConfig” and “viewResolver” beans we are telling VelocityConfigurer where to look for templates, and VelocityLayoutViewResolver where to find views and layouts.

Chapter 12

Apache Commons Library

Apache Commons is an Apache project focused on all aspects of reusable Java components.

The Apache Commons project is composed of three parts:

- The Commons Proper - A repository of reusable Java components.
- The Commons Sandbox - A workspace for Java component development.
- The Commons Dormant - A repository of components that are currently inactive.

The Commons Proper :

Commons Proper is dedicated to one principal goal: creating and maintaining reusable Java components. The Commons Proper is a place for collaboration and sharing, where developers from throughout the Apache community can work together on projects to be shared by the Apache projects and Apache users.

The Commons Sandbox:

The Commons project also contains a workspace that is open to all Apache committers. It's a place to try out new ideas and prepare for inclusion into the Commons portion of the project or into another Apache project. Users are free

to experiment with the components developed in the sandbox, but sandbox components will not necessarily be maintained, particularly in their current state.

The Commons Dormant:

These are Commons components that have been deemed inactive since they have seen little recent development activity. If you wish to use any of these components, you must build them yourselves. It is best to assume that these components will not be released in the near future.

Chapter 13

Validation in Frameworks

Any application needs validations. Particularly in web applications, the user interface has to have many conditions to be checked. These conditions can be checked as validations.

Client side validations:

In a web application we can have both client side validation and server side validation. The client side validations are done on the view pages and the drawback of client side validations is that it can be manipulated. So we use server side validations for security concern.

Server side validations:

In Spring Boot, we have annotations for validations and these annotations are used in the spring bean file, Now the fields are validated and mapped to the databese also. Now, on validation, we can see that a validation check happens.

Custom Validation in Spring Boot:

MessageSource is a powerful feature available in Spring applications. This helps application developers handle various complex scenarios with writing much extra code, such as environment-specific configuration, internationalization or configurable values. One more scenario could be modifying the default validation messages to more user-friendly/custom messages. Here we've added validation constraints that verify if an email is not provided at all, or provided, but not following the standard email address style.
To show custom and locale-specific message, we can provide a placeholder as mentioned for the @NotEmpty annotation. The email.notempty property will be resolved from a properties files by the MessageSource configuration. Finally, create a properties file in the src/main/resources directory with the name provided in the basename.

Chapter 14

Logging in Spring Boot

- A Spring Boot module has to be created at first.

- Now create the only class file, LoggingController.

- It allows us to forget about the majority of the configuration settings, many of which it opinionatedly autotunes hence the Spring Boot is a very helpful framework.

- The only mandatory dependency in the case of logging is Apache Commons Logging.

- Only when using Spring 4.x (Spring Boot 1.x), we need to import it. Since in Spring 5 (Spring Boot 2.x) it's provided by Spring Framework’s spring-jcl module.

- We need not have to worry about importing spring-jcl at all if we're using a Spring Boot Starter (which almost always we are) because every starter, like our spring-boot-starter-web, depends on spring-boot-starter-logging, which already pulls in spring-jcl for us.

- Logback is used for logging by default when using starters.

- Spring Boot pre-configures it with patterns and ANSI colors to make the standard output more readable.

- Let's now run the application and visit the http://localhost:8080/page, and see what happens in the console:

Logback Configuration Logging

How can we include a Logback configuration with a different color and logging pattern, with separate specifications for console and file output with a decent rolling policy to avoid generating huge log files.

We should go towards a solution which allows handling our logging settings alone at first, instead of polluting the application.properties file as it is commonly used for many other application settings.

The Spring Boot will automatically load it over the default configuration, when a file in the classpath has one of the following names:

logback-spring.xml
logback.xml
logback-spring.groovy
logback.groovy

Spring recommends using the -spring variant over the plain ones whenever possible.

Log4j2 Configuration Logging

- At first, place in the classpath a file named like one of the following:
  - log4j2-spring.xml
  - log4j2.xml

- Without further modifications, we'll print through Log4j2 (over SLF4J).

- The output is quite different from the Logback one — this shows that we're fully using Log4j2 now.

- Log4j2 allows us to use also a YAML or JSON configurations in addition to the XML configuration,.

- Without passing through SLF4J ,we can also use Log4j2 natively.

Logging with Lombok

- The boilerplate code can be annoying, and we can avoid it using various annotations introduced by Lombok.In the examples, we've seen so far, we've had to declare an instance of a logger from our logging framework.

@Slf4j and @CommonsLog

- Lombok's @Slf4j and @CommonsLog annotations can be added to the right logger instance into our class: org.slf4j.Logger for SLF4J and org.apache.commons.logging.Log for Apache Commons Logging.

- Log4j2 implementation will be used for logging with Log4j2-Configuration Logging. Just as with Zero-Configuration Logging, the application will use underlying logging implementation Logback for logging.

- When we replace the annotation @Slf4j with @CommonsLog, we get the same behavior .

@Log4j2

We can use the annotation @Log4j2 to use Log4j2 directly. Hence, we make a simple change to LombokLoggingController to use @Log4j2 instead of @Slf4j or @CommonsLog.