Module 5 – Spring

Module Overview

Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly. Spring framework was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Module Objective

At the end of this module, students should be able to demonstrate appropriate knowledge, and show an understanding of the following:

- Explain in detail evolution of Spring Framework
- Understand the benefits of using Spring
- Connect business objects to persistent stores using Spring's DAO modules
- Use the Spring MVC web framework to develop flexible web applications
- Use the Spring MVC Java Based Configuration to develop flexible web application

Introduction to Spring

Why to Learn Spring?

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.

Spring framework is an open source Java platform. **It was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003**.

Spring is lightweight when it comes to size and transparency. The basic version of Spring framework is around **2MB**.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.

How to download and install Spring framework

1. For Downloading Spring Repository you need to visit https://repo.spring.io/release/org/springframework/spring/.
   In this website, you will find different spring framework releases. You have to click on latest framework release. Here you will find three files which are:
   - spring-framework-5.1.4.RELEASE-dist.zip
   - spring-framework-5.1.4.RELEASE-docs.zip
   - spring-framework-5.1.4.RELEASE-schema.zip

   Now you have to click on "spring-framework-5.1.4.RELEASE-dist.zip" so that it will download.
2. For Installing, You need to extract the "spring-framework-5.1.4.RELEASE-dist.zip" file in your C drive. Now you are able to run your application in spring framework.

Advantages of Spring Framework

1) Predefined Templates

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So, there is no need to write too much code. It hides the basic steps of these technologies.

Let's take the example of JdbcTemplate, you don't need to write the code for exception handling, creating connection, creating statement, committing transaction, closing connection etc. You need to write the code of executing query only. Thus, it saves a lot of JDBC code.

2) Loose Coupling

The Spring applications are loosely coupled because of dependency injection.

3) Easy to test

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

4) Lightweight

Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.

5) Fast Development

The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

6) Powerful abstraction

It provides powerful abstraction to JavaEE specifications such as JMS, JDBC, JPA and JTA.

7) Declarative support

It provides declarative support for caching, validation, transactions and formatting.

Features of the Spring Framework

The features of the Spring framework such as IoC, AOP, and transaction management, make it unique among the list of frameworks. Some of the most important features of the Spring framework are as follows:

- **IoC container:**
  It refers to the core container that uses the DI or IoC pattern which implicitly provides an object reference in a class during runtime. This pattern acts as an alternative to the service locator pattern. The IoC container contains assembler code that handles the configuration management of application objects.
  The Spring framework provides two packages, namely org.springframework.beans and org.springframework.context which helps in providing the functionality of the IoC container.

- **Data access framework:**
  This framework allows the developers to use persistence APIs, such as JDBC and Hibernate, for storing persistence data in database. It helps in solving various problems of the developer, such as how to interact with a database connection, how to make sure that the connection is closed, how to deal with exceptions, and how to implement transaction management It also enables the developers to easily write code to access the persistence data throughout the application.

- **Spring MVC framework:**
  Allows you to build Web applications based on MVC architecture. All the requests made by a user first go through the controller and are then dispatched to different views, that is, to different JSP pages or Servlets.

The form handling and form validating features of the Spring MVC framework can be easily integrated with all popular view technologies such as ISP, Jasper Report, FreeMarker, and Velocity.

- **Transaction management:**
  Helps in handling transaction management of an application without affecting its code. This framework provides Java Transaction API (JTA) for global transactions managed by an application server and local transactions managed by using the JDBC Hibernate, Java Data Objects (JDO), or other data access APIs. It enables the developer to model a wide range of transactions on the basis of Spring's declarative and programmatic transaction management.

- **Spring Web Service:**
  Generates Web service endpoints and definitions based on Java classes, but it is difficult to manage them in an application. To solve this problem, Spring Web Service provides layered-based approaches that are separately managed by Extensible Markup Language (XML) parsing (the technique of reading and manipulating XML). Spring provides effective mapping for transmitting incoming XML message request to an object and the developer to easily distribute XML message (object) between two machines.

- **JDBC abstraction layer:**
  Helps the users in handling errors in an easy and efficient manner. The JDBC programming code can be reduced when this abstraction layer is implemented in a Web application. This layer handles exceptions such as DriverNotFound. All SQLExceptions are translated into the DataAccessException class. Spring's data access exception is not JDBC specific and hence Data Access Objects (DAO) are not bound to JDBC only.

- **Spring TestContext framework:**
  Provides facilities of unit and integration testing for the Spring applications. Moreover, the Spring TestContext framework provides specific integration testing functionalities such as context management and caching DI of test fixtures, and transactional test management with default rollback semantics.

Evolution of Spring Framework

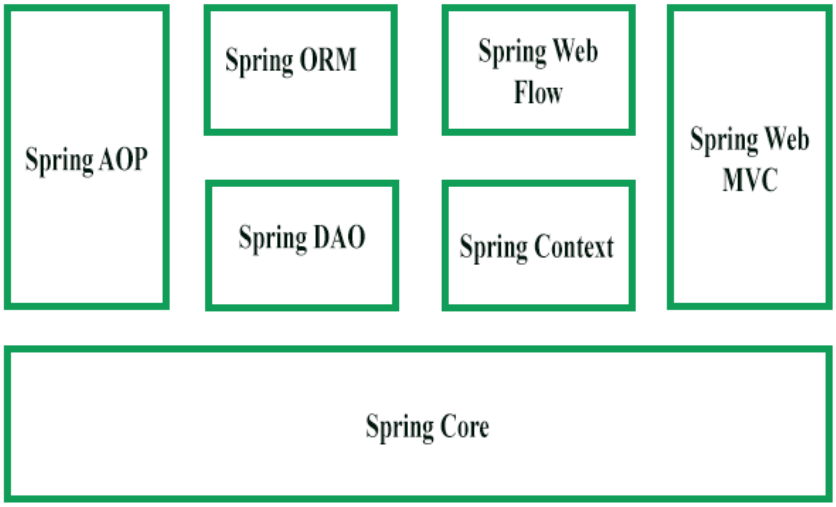
The Spring framework consists of seven modules which are shown in the Figure below.

These modules are:

- ✓ Spring Core
- ✓ Spring AOP
- ✓ Spring Web MVC
- ✓ Spring DAO

- ✓ Spring ORM
- ✓ Spring context
- ✓ Spring Web flow

These modules provide different platforms to develop different enterprise applications; for example, one can use Spring Web MVC module for developing MVC-based applications.



- **Spring Core Module:**

  The Spring Core module, which is the core component of the Spring framework, provides the IoC container There are two types of implementations of the Spring container, namely, bean factory and application context. Bean factory is defined using the org.springframework.beans.factory.BeanFactory interface and acts as a container for beans. The Bean factory container allows you to decouple the configuration and specification of dependencies from program logic.

  In the Spring framework, the Bean factory acts as a central IoC container that is responsible for instantiating application objects. It also configures and assembles the dependencies between these objects. There are numerous implementations of the BeanFactory interface. The XmlBeanFactory class is the most common implementation of the BeanFactory interface.

- **Spring AOP Module:**

  Similar to Object-Oriented Programming (OOP), which breaks down the applications into hierarchy of objects, AOP breaks down the programs into aspects or concerns. Spring AOP module allows you to implement concerns or aspects in a Spring application in Spring AOP, the aspects are the regular Spring beans or regular classes annotated with @Aspect annotation.

  These aspects help in transaction management and logging and failure monitoring of an application.

  For example, transaction management is required in bank operations such as transferring an amount from one account to another Spring AOP module provides a transaction management abstraction layer that can be applied to transaction APIs.

- **Spring ORM Module:**

  The Spring ORM module is used for accessing data from databases in an application. It provides APIs for manipulating databases with JDO, Hibernate, and iBatis. Spring ORM supports DAO, which provides a convenient way to build the following DAOs-based ORM solutions:
  - Simple declarative transaction management
  - Transparent exception handling
  - Thread-safe, lightweight template classes
  - DAO support classes
  - Resource management

- **Spring Web MVC Module:**

  The Web MVC module of Spring implements the MVC architecture for creating Web applications. It separates the code of model and view components of a Web application. In Spring MVC, when a request is generated from the browser, it first goes to the DispatcherServlet class (Front Controller), which dispatches the request to a controller (SimpleFormController class or AbstractWizardformController class) using a set of handler mappings.

  The controller extracts and processes the information embedded in a request and sends the result to the DispatcherServlet class in the form of the model object. Finally, the DispatcherServlet class uses ViewResolver classes to send the results to a view, which displays these results to the users.

- **Spring Web Flow Module:**

  The Spring Web Flow module is an extension of the Spring Web MVC module. Spring Web MVC framework provides form controllers, such as class SimpleFormController and AbstractWizardFormController class, to implement predefined workflow. The Spring Web Flow helps in defining XML file or Java Class that manages the workflow between different pages of a Web application. The Spring Web Flow is distributed separately and can be downloaded through http://www.springframework.org website.

  The following are the advantages of Spring Web Flow:
  - The flow between different UIs of the application is clearly provided by defining Web flow in XML file.
  - Web flow definitions help you to virtually split an application in different modules and reuse these modules in multiple situations.
  - Spring Web Flow lifecycle can be managed automatically

- **Spring Web DAO Module:**

  The DAO package in the Spring framework provides DAO support by using data access technologies such as JDBC, Hibernate, or JDO. This module introduces a JDBC abstraction layer by eliminating the need for providing tedious JDBC coding. It also provides programmatic as well as declarative transaction management classes.

  Spring DAO package supports heterogeneous Java Database Connectivity and O/R mapping, which helps Spring work with several data access technologies. For easy and quick access to database resources, the Spring

framework provides abstract DAO base classes. Multiple implementations are available for each data access technology supported by the Spring framework.
For example, in JDBC, the JdbcDaoSupport class and its methods are used to access the DataSource instance and a preconfigured JdbcTemplate instance. You need to simply extend the JdbcDaoSupport class and provide a mapping to the actual DataSource instance in an application context configuration to access a DAO-based application.

- **Spring Application Context Module:**
  The Spring Application context module is based on the Core module. Application context org.springframework.context.ApplicationContext is an interface of BeanFactory. This module derives its feature from the org.springframework.beans package and also supports functionalities such as internationalization (I18N), validation, event propagation, and resource loading.
  The Application context implements MessageSource interface and provides the messaging functionality to an application.

Now let us practise Spring program:

You need to create spring configuration file and declare all of your beans here. I am naming the configuration file as applicationContext.xml file.

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
   <bean id="springTest" class="com.java2novice.beans.SpringFirstTest" />
</beans>
```

Here is the Spring Demo class to run the spring bean:

```
package com.java2novice.test;


import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.java2novice.beans.SpringFirstTest;


public class SpringDemo {

   public static void main(String a[]){

      String confFile = "applicationContext.xml";

      ApplicationContext context = new
ClassPathXmlApplicationContext(confFile);

      SpringFirstTest sft = (SpringFirstTest) context.getBean("springTest");

      sft.testMe();

   }   }
```

Output:

```
I am listening...
```

Spring Boot

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

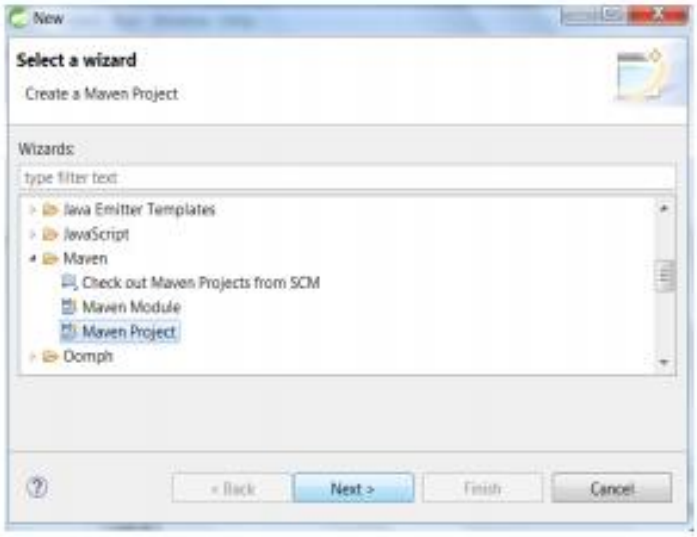
Spring STS can be downloaded from spring.io/tools

Ways to create Spring Boot project:

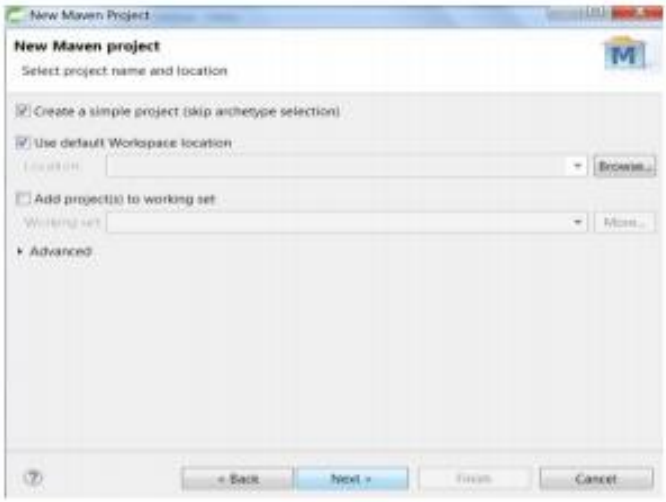
1. Using the Spring Tool Suite IDE (STS)
2. Spring Initializer
3. Spring command line interface

Steps to create Spring boot application using STE IDE:

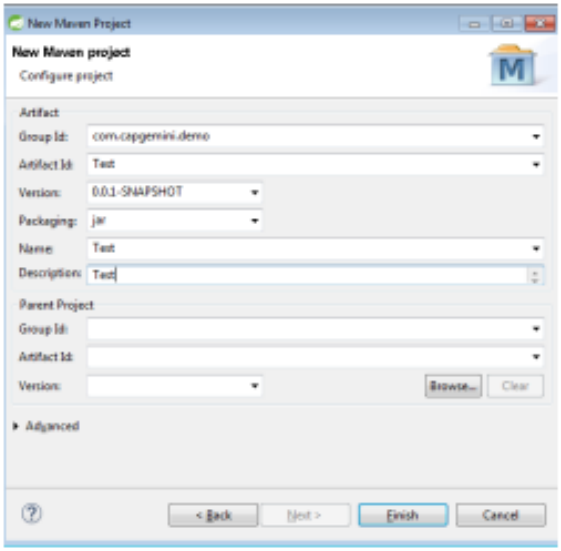
Step1: Click on menu, File →New –Other - Maven -Maven Project- Click on Next.



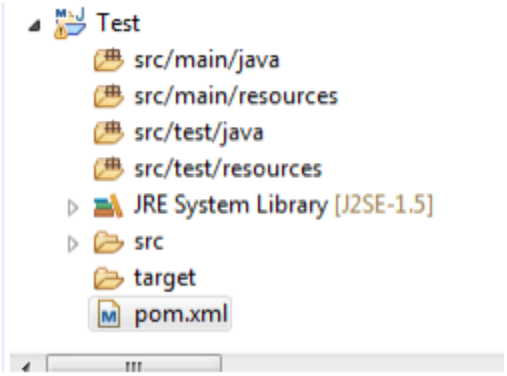
Step 2: Select the checkbox, “Create a simple project” and Click On Next



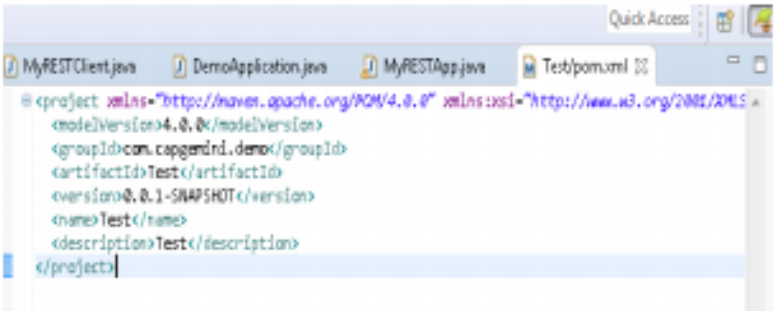
Step 3: Click on Finish. Observe the folder structure of the newly created project



Step 4: Double click on the generated pom.xml file



Default pom.xml is shown below:



Step 5: Add the following code in the pom.xml under the <description> tag

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.1.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.1.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
</parent>

Above entry will bring in all the dependency management features of Spring boot. There is no need to declare all the dependencies one by one in pom.xml

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>

```
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Above will integrate Spring MVC and auto configure the project for us. When you add the Spring boot starter web dependency in pom.xml, this brings in the Spring MVC sub framework dependency into the application.

Step 6: Without Spring Boot, these jar files are among those that you would have had to copy physically into the project.

Create a new java class having the following code

```
@SpringBootApplication
public class Client {
public static void main(String[] args) {
        SpringApplication.run(Client.class,args);
  }
}
```

Run the above program as a regular java application. There is no need to deploy this application on any external server.

SpringApplication.run(): Starts Spring, creates Spring context , applies annotations and sets up embedded container.\

Step 7: Run the application as a java application and observe the console.

How Spring boot works?

1. The applicaton is started from the Java main class

2. Spring boot initialises Spring context that comprises the Spring app and honors autoconfig initializers, configuration and annotations which direct how to initialize and startup the spring context

3. Embedded server container is started and autoconfigured.

This removes the need for web.xml. Spring has chosen “Tomcat” as the default container.

@SpringBootApplication

A convenience annotation that wraps commonly used annotations.

Used in place of the following 3 different annotations :

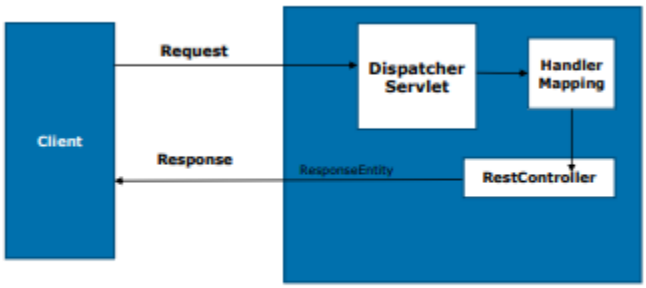
1. @configuration : Instructs that a Spring configuration class is being used instead of XML to define the components.

2. @EnableAutoconfiguraton : is a Spring boot specific annotation. Instructs that the application should auto configure the other frameworks included as dependency with Spring. This annotation told Spring boot to automatically set up so that we can use Spring controllers without doing any other integration work with MVC framework

3. @ComponentScan : Scans project for Spring components annotated with @Service, @Repository, @Component.

1. Simple Java application using Spring Boot
2. Restful web application using Spring Boot
3. Spring boot application which integrates with Spring Data JPA

Spring RESTFul Web Services



Spring5 MVC REST Workflow

Spring's annotation based MVC framework simplifies the process of creating RESTful web services. The key difference between a traditional Spring MVC controller and the RESTful web service controller is the way the HTTP response body

is created. While the traditional MVC controller relies on the View technology, the RESTful web service controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML.

Spring MVC REST Workflow

The following steps describe a typical Spring MVC REST workflow:

The client sends a request to a web service in URI form.

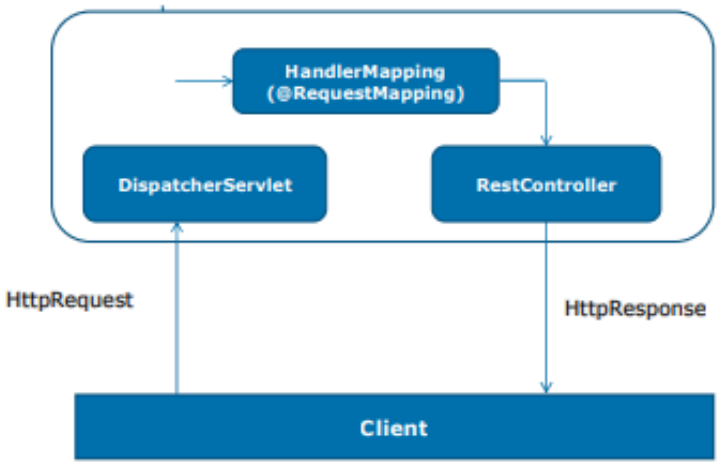
The request is intercepted by the DispatcherServlet which looks for Handler Mappings and its type:

- The Handler Mappings section defined in the application context file tells DispatcherServlet which strategy to use to find controllers based on the incoming request.
- Spring MVC supports three different types of mapping request URIs to controllers: annotation, name conventions and explicit mappings.

Requests are processed by the Controller and the response is returned to the DispatcherServlet which then dispatches to the view.

Spring has a list of HttpMessageConverters registered in the background. The responsibility of the HTTPMessageConverter is to convert the request body to a specific class and back to the response body again, depending on a predefined mime type. Every time an issued request hits @ResponseBody, Spring loops through all registered HTTPMessageConverters seeking the first that fits the given mime type and class, and then uses it for the actual conversion.

Life cycle of a Request in Spring MVC Restful



REST(Representational State Transfer) is an architectural style with which Web Services can be designed that serves resources based on the request from client. A Web Service is a unit of managed code, that can be invoked using HTTP requests. You develop the core functionality of your application, deploy it in a server and expose to the network. Once it is exposed, it can be accessed using URI's through HTTP requests from a variety of client applications. Instead of repeating the same functionality in multiple client (web, desktop and mobile) applications, you write it once and access it in all the applications.

In the above diagram, from the time that a request is received by Spring until the time that a response is returned to the client, many pieces of Spring Restful web services are involved.

The process starts when a client (typically a web browser) sends a request. It is first received by a DispatcherServlet. Like most Java-based MVC frameworks, Spring MVC uses a front-controller servlet (here DispatcherServlet) to intercept requests. This in turn delegates responsibility for a request to other components of an application for actual processing.

The Spring MVC uses a Controller component for handling the request. But a typical application may have several controllers. To determine which controller should handle the request, DispatcherServlet starts by querying one or more HandlerMappings. A HandlerMapping typically maps URL patterns to RestControllers.

Once the DispatcherServlet has a appropriate RestController selected, it dispatches the request to that Controller which performs the business logic (a well-designed RestController object delegates responsibility of business logic to one or more service objects). Upon completion of business logic, HTTPResponse is generated and sent back to the client.

HTTP Methods in REST

HTTP Method	Operation	Comment
GET	Read Operation only	Uses only for the read operation.GET should be idempotent
POST	Create new resource	Should only be used to create a new resource
PUT	Update / Replace Resource	Update an existing resource.Think of PUT method as putting a resource
DELETE	Delete Resource	To remove a given resource.DELETE operation is *idempotent*
PATCH	Partial Update / Modify	Partial update to a resource should happen through PATCH

RESTful URLs – HTTP methods

At times if the HTTP Get is used over a couple of RestController methods it has to be combined with URL patterns to create unique identifications.

1. In the above slide first URL pattern demonstrates : HTTP Get method to fetch all country details. Similarly if details for a particular country need to be fetched then the country id can be appended in URL and extracted via the @PathVariable
   http://localhost:9090/SpringRESTWebServices/rest/countries/3 -> With this URL country details are fetched for country Id = 3
2. The second URL pattern demonstrates : HTTP Post method to create a new country
3. The third URL pattern demonstrates : HTTP Delete method to delete an existing country

Note: As HTML supports only Get and Post methods for the method attribute in the form tag; we also need to map the HTTP PUT(update) and HTTP DELETE (delete) methods to update and delete the resources respectively.

For this Spring provides us with a Filter-mapping which is to be given in web.xml file:

```
<filter>
        <filter-name>httpMethodFilter</filter-name>
        <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>httpMethodFilter</filter-name>
        <servlet-name>dispatcher</servlet-name>
```

</filter-mapping>

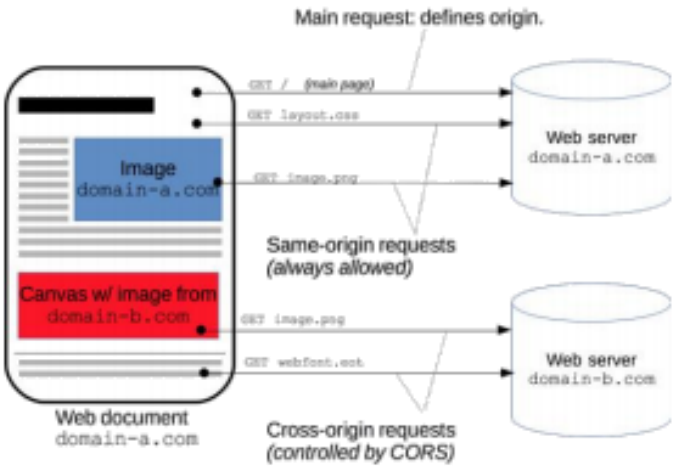
By using this Spring in-built filter the different methods of HTTP specification will be mapped to their actual HTTP implementations.

Here the filter will be intercepted for all the requests coming to DispatcherServlet. Also in the JSP pages for updating and deleting a country need to pass a HTML hidden parameter : For example

<input type="hidden" name="_method" value="delete"/> to pass the “real” HTTP method to Spring Rest Controller.

Cross-Origin Resource Sharing (CORS)

The CORS mechanism supports secure cross-domain requests and data transfers between browsers and web servers. Modern browsers use CORS in an API container such as XMLHttpRequest or Fetch to help mitigate the risks of cross-origin HTTP requests.



• CORS (Cross-origin resource sharing) allows a webpage to request additional resources into browser from other domains e.g. fonts, CSS or static images from CDNs.

• Helps in serving web content from multiple domains into browsers who usually have the same-origin security policy.

• Spring CORS support in Spring MVC application at method level and global level.

• @CrossOrigin allows all origins, all headers, the HTTP methods specified in the @RequestMapping annotation and a maxAge of 30 minutes.

Attributes:

• Origins - List of allowed origins. It's value is placed in the Access-Control-Allow-Origin header of both the pre-flight response and the actual response.

– * – means that all origins are allowed.

– If undefined, all origins are allowed.

• allowedHeaders - List of request headers that can be used during the actual request. Value is used in preflight's response header Access-Control-Allow-Headers.

– * – means that all headers requested by the client are allowed.

– If undefined, all requested headers are allowed.

• methods - List of supported HTTP request methods. If undefined, methods defined by RequestMapping annotation are used.

• exposedHeaders - List of response headers that the browser will allow the client to access. Value is set in actual response header Access-Control-Expose-Headers.

– If undefined, an empty exposed header list is used.

• allowCredentials - It determine whether browser should include any cookies associated with the request.

– false – cookies should not included.

– "" (empty string) – means undefined.

– true – pre-flight response will include the header Access-Control-Allow-Credentials with value set to true.

– If undefined, credentials are allowed.

• maxAge - maximum age (in seconds) of the cache duration for pre-flight responses. Value is set in header Access-Control-Max-Age.

– If undefined, max age is set to 1800 seconds (30 minutes).

REST Testing

Spring RestTemplate:

▪ Spring RestTemplate class is part of spring-web, introduced in Spring 3.

▪ We can use RestTemplate to test HTTP based restful web services, it doesn't support HTTPS protocol.

▪ RestTemplate class provides overloaded methods for different HTTP methods, such as GET, POST, PUT, DELETE etc.

Spring RestTemplate Methods

Get:

getForObject, getForEntity

Post:

postForObject(String url, Object request, Class responseType, String…

uriVariables) postForLocation(String url, Object request, String… urlVariables),

Put:

put(String url, Object request, String…urlVariables)

Delete:

delete()

Head:

headForHeaders(String url, String… urlVariables)

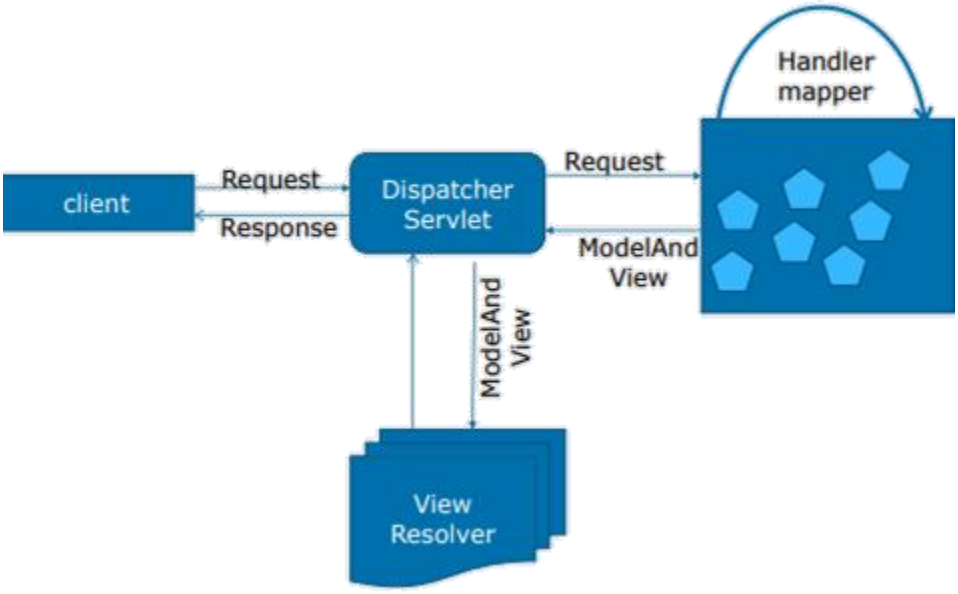
Options:

optionsForAllow(String url, String… urlVariables

Spring MVC Framework

The Spring MVC is a web framework built within the Spring Framework. There are a number of challenges while creating a web-based application - like state management, workflow and validation, which are addressed by Spring's web framework. This framework can be used to automatically populate your model objects from incoming request parameters while providing validation and error handling as well. The entire framework is modular, with each set of components having specific roles and completely decoupled from the rest of the framework. This allows you to develop the front end of your web application in a very pluggable manner.

MVC provides out-of-the-box implementations of workflow typical to web applications. It's highly flexible, allowing you to use a variety of different view technologies. It also enables you to fully integrate with your Spring based, middle-tier logic through the use of dependency injection. You can use Spring web MVC to make services that are created with other parts of Spring available to your users, by implementing web interfaces.



Life cycle of a request in Spring MVC:

From the time that a request is received by Spring until the time that a response is returned to the client, many pieces of Spring MVC framework are involved.

The process starts when a client (typically a web browser) sends a request. It is first received by a DispatcherServlet. Like most Java-based MVC frameworks, Spring MVC uses a front-controller servlet (here DispatcherServlet) to intercept requests. This in turn delegates responsibility for a request to other components of an application for actual processing.

The Spring MVC uses a Controller component for handling the request. But a typical application may have several controllers. To determine which controller should handle the request, DispatcherServlet starts by querying one or more HandlerMappings. A HandlerMapping typically maps URL patterns to Controller objects.

Once the DispatcherServlet has a Controller object, it dispatches the request to the Controller which performs the business logic (a well-designed Controller object delegate's responsibility of business logic to one or more service objects). Upon completion of business logic, the Controller returns a ModelAndView object to the DispatcherServlet. The ModelAndView object contains both the model data and logical name of view. The DispatcherServlet queries a ViewResolver with this logical name to help find the actual JSP. Finally the DispatcherServlet dispatches the request to the View object, which is responsible for rendering a response back to the client.

Configuring DispatcherServlet in web.xml

```
<servlet>
   <servlet-name>basicspring</servlet-name>
   <servlet-class> org.springframework.web.servlet.DispatcherServlet
   </servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>basicspring</servlet-name>
   <url-pattern>*.obj</url-pattern>
 </servlet-mapping>
```

The servlet-name given to the servlet is significant

The dispatcher servlet is at the heart of the Spring MVC and functions as Spring MVC's front controller. Like any other servlet, it must be configured in web.xml file. Place the declaration (in the first listing above) in the web.xml file. The servlet-name given to the servlet is significant. By default, when DispatcherServlet is loaded, it will load the Spring application context from an xml file whose name is based on the name of the servlet. In the above example, the servlet-name is basicspring and so the DispatcherServlet will try to load the application context from a file called basicspring-servlet.xml.

Next, you must indicate which URL's will be handled by DispatcherServlet. Add the following tag (the second xml listing) to web.xml to let DispatcherServlet handle all url's that end in .obj. The URL pattern is arbitrary and could be any extension.

DispatcherServlet is now configured and ready to dispatch requests to the web layer of your application

WebApplicationContext

```
<listener>
  <listener-class>
        org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
      /WEB-INF/basicspring-service.xml
     /WEB-INF/basicspring-data.xml
   </param-value>
</context-param>
```

We have seen that DispatcherServlet will load the Spring application context from a single XML file whose name is -servlet.xml. But you can also split the application context across multiple XML files. Ideally splitting it into logical pieces across application layers can make maintenance easier by keeping each of the Spring configuration files focused on a single layer of the application. To ensure that all the configuration files are loaded, you will need to configure a

context loader in your web.xml file. A context loader loads context configuration files in addition to the one that DispatcherServlet loads. The most commonly used context loader is a servlet listener called ContextLoaderListener that is configured in web.xml.

With ContextLoaderListener configured, we need to tell it the location of the Spring configuration file(s) to load. If not specified, context loader will look for a Spring configuration file at /WEB-INF/applicationContext.xml. We will specify one or more Spring configuration file(s) by setting the contextConfigLocation parameter in the servlet context as seen in the second listing above. The context loader will use contextConfigLocation to load the two context configuration files – one each for the service and data layer.

Annotation-based controller configuration

@Controller: Indicates that an annotated class is a "Controller"

Example:

```
@Controller
Public class LoginController{}
```

@RequestMapping: @RequestMapping can be applied to the controller class as well as methods. It maps web requests onto specific handler classes and/or handler methods.

Example:

```
@Controller
@RequestMapping(“loginController”)
Public class LoginController{
@RequestMapping(“loadForm”)
public String loadData(){}
}
```

URL need to be used for making the request for the above mentioned controller is

http://localhost:8080/projectname/loginController/loadForm.ob

Building a basic Spring MVC application

▪ InternalResourceViewresolver:

Resolves logical view names into View objects that are rendered using template file resources

▪ BeanNameViewResolver:

Looks up implementations of the View interface as beans in the Spring context, assuming that the bean name is the logical view name

▪ ResourceBundleViewResolver

Uses a resource bundle that maps logical view names to implementations of the View interface

▪ XmlViewResolver

Resolves View beans from an XML file that is defined separately from the application context definition files

So far, we have seen how model objects are passed to the view through the ModelAndView object. In Spring MVC, a view is a bean that renders results to the user. The view most likely is a JSP. But you could also use other view technologies like Velocity and FreeMarker templates or even views that produce PDF and MS-Excel documents.

View resolvers resolve the view name given by the ModelAndView object to a View bean. Spring provides a number of useful view resolvers, some of which are shown in the table above. See Spring docs for more.

InternalResourceViewResolver resolves a logical view name by affixing a prefix and a suffix to the view name returned by the ModelAndView object. It then loads a View object with the path of the resultant JSP. By default, the view object is an InternalResourceView, which simply dispatches the request to the JSP to perform the actual rendering. But, if the JSP uses JSTL tags, then you may replace InternalResourceView with JstlView as seen in the code demos earlier.

```
<bean id="viewResolver" class=
        "org.springframework.web.servlet.view.InternalResourceViewResolver ">
        <property name="viewClass">
        <value>org.springframework.web.servlet.view.JstlView </value>
        </property>
        <property name="prefix"><value>/</value></property>
        <property name="suffix"><value>.jsp</value></property>
</bean>
```

Spring MVC annotations -Validating input with Bean Validation

Before an object can be processed further, it is essential to ensure that all the data in the object is valid and complete. Faulty form input must be rejected. For example, username must not contain spaces, password must be minimum 6 characters long, email must be correct etc.

The @Valid annotation (part of the JavaBean validation specification) tells Spring that the User object should be validated as it's bound to the form input. If anything goes wrong while validating the User object, the validation error will be carried to the processForm() method via the BindingResult that's passed in on the second parameter. If the BindingResult's hasErrors() method returns true, then that means that validation failed.

How do we declare validation rules?

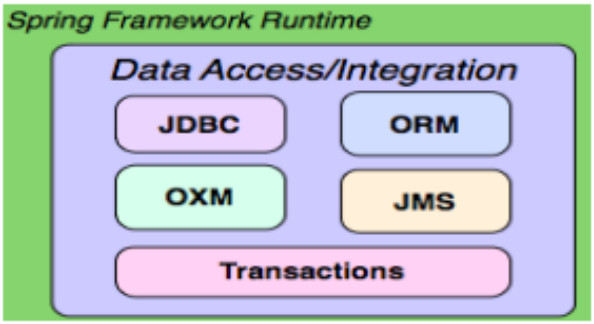
JSR-303 defines some annotations that can be placed on properties to specify validation rules. The code above shows the properties of the User class that are annotated with validation annotations.

@Size annotation validates that the fields meet criteria on their length.

@Pattern annotation along with a regular expression ensures that the value given to the email property fits the format of an email address and that the username is only made up of alphanumeric characters with no spaces. Notice how we've set the message attribute with the message to be displayed in the form when validation fails. With these annotations, when a user submits a registration form to AddUserFormController's processForm() method, the values in the User object's fields will be validated. If any of those rules are violated, then the handler method will send the user back to the form.

Spring JPA Integration

Why Spring JPA Integration?



The client application that accesses the database using JPA has to depend on the JPA APIs like PersistenceContext, EntityManagerFactory, EntityManager and Transaction. These objects will continue to get scattered across the code throughout the Application.

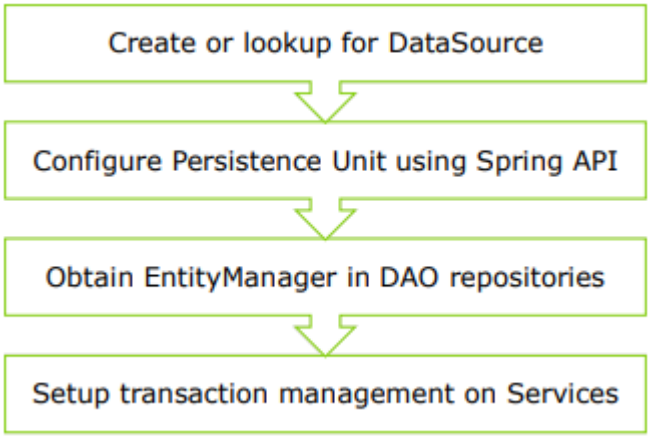
Moreover, the Application code has to manually maintain and manage these objects. In the case of Spring, the business objects can be highly configurable with the help of IOC Container. Which means that now it is possible to use the JPA objects as Spring Beans and they can enjoy all the facilities that Spring provides.

The EntityManager instance is managed automatically, and can be injected into data access object (DAO) beans; so there is no need to manage it manually in application code.

Another advantage of integrating JPA in Spring is that Spring manages starting, committing, and rolling back transactions automatically on your behalf. You will never use it directly and you'll only configure an implementation.

The JPA defines a modest exception hierarchy starting at PersistenceException, but even it is missing some key features like an exception to indicate that a unique key violation occurred. Spring Framework solve this problem by defining a thorough hierarchy of persistence exceptions.

Implementation of JPA Integration



Integration work starts by defining or obtaining data source, which contains information about database url, username and password etc.

Once we obtain datasource, we need configure JPA related beans using spring configuration file. It is used in place of persistence.xml. Though, we are using Spring configuration, optionally we may keep persistence.xml just to hold the persistence unit reference. This file may be required in future for advance configuration like caching etc.

After configuration, we can work on entities and DAO repositories to define our database operations.

Finally, we need to instruct Spring to handle all JPA transactions, this is done by annotating and marking DAO operations with various transaction demarcation policies.

The first step towards integration is making DataSource available to application. There are different ways to do this. For example, if you need to simply test something quickly, use Spring's DriverManagerDataSource to create a DataSource on demand.

However, this creates a simple DataSource that returns single use Connections. Because it does not provide connection pooling, it really should never be used in a production environment. For production environment, we can define DataSource on server and look that DataSource up application.

The EntityManagerFactory is what we use to start up JPA inside our application. Without Spring, we use persistence.xml file to configure this object. While working with Spring, we don't need configuration via persistence.xml, instead we can define all JPA related objects as spring beans.

Spring allows two ways to define this object, either application managed or container managed (used throughout this integration).

In addition to EntityManagerFactory, Spring provides TransactionManager to handle all JPA related transactions, which is also configured inside spring configuration file.

Spring offers three different options to configure EntityManagerFactory in a project:

1. LocalEntityManagerFactoryBean
2. EntityManagerFactory lookup over JNDI
3. LocalContainerEntityManagerFactoryBean

LocalEntityManagerFactoryBean is the most basic and limited one. It is mainly used for testing purposes and standalone environments. It reads JPA configuration from /META-INF/persistence .xml, doesn't allow you to use a Spring-managed DataSource instance, and doesn't support distributed transaction management.

Use *EntityManagerFactory* lookup over JNDI if the run time is Java EE 5 Server.

LocalContainerEntityManagerFactoryBean is the most powerful and flexible JPA configuration approach Spring offers. It gives full control overEntityManagerFactory configuration, and it's suitable for environments where fine-grained control is required. It enables you to work with a SpringmanagedDataSource, lets you selectively load entity classes in your project's classpath, and so on. It works both in application servers and standalone environments.

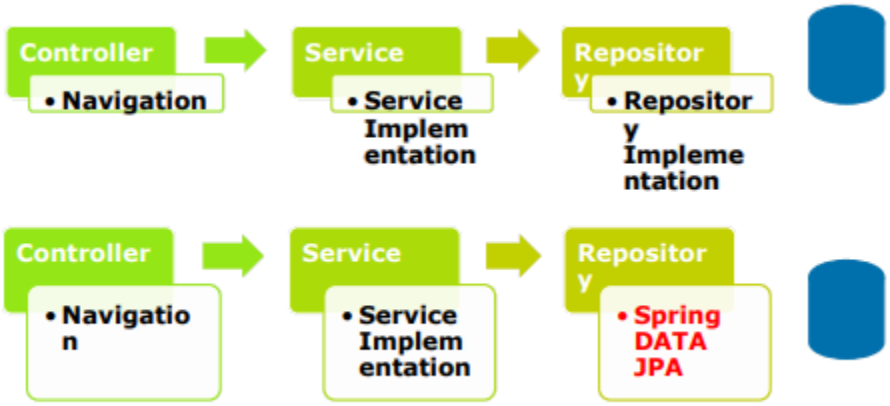
Configuration of TransactionManager:

Spring provides a class org.springframework.orm.jpa.JpaTransactionManager to work with JPA specific transactions. Slide demonstrate how to configure this class in spring configuration file. It is required to inject entitymanagerfactory created in earlier step in this class using setter injection.

Basically all of JPA interactions will be wrapped in transaction by this TransactionManager and we are instructing spring to create a bean of this class for application transaction management.

Once this transaction manager is made available, instead of opening and closing transactions manually, we can instruct Spring to handle transaction using annotations. To do so, we must use injecting the transaction manager instance.

What is Spring Data JPA



Spring Data supports a variety of data access methodologies, including JPA, JdbcTemplate, NoSQL, and more. Its primary subproject, Spring Data Commons, provides a core toolset that all other subprojects use to create repositories. The Spring Data JPA subproject provides support for repositories implemented against the Java Persistence API.

Spring Data JPA, a separate Spring project but dependent on Spring Framework, can write your repositories on behalf of developers.

Spring Data JPA is another layer of abstraction for the support of persistence layer in spring context. The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores. We just have to write repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

As a first step we define a domain class-specific repository interface. The interface must extend JpaRepository and be typed to the domain class and an ID type. JpaRepository is a child inteface of following:

1. CrudRepository<T,ID>,
2. PagingAndSortingRepository<T,ID>,
3. QueryByExampleExecutor<T>,
4. Repository<T,ID>

The CrudRepository provides sophisticated CRUD functionality for the entity class that is being managed. One can add additional query methods to this repository interface. Below listed are few methods of CrudRepository:

I. count() returns a long representing the total number of unfiltered entities extending T.
II. delete(T) and delete(ID) delete the single, specified entity and deleteAll() deletes every entity of that type.
III. exists(ID) returns a boolean indicating whether the entity of this type with the given key exists.
IV. findAll() returns all entities of type T, whereas findOne(ID) retrieves a single entity of type T given its key.
V. save(T) saves the given entity (insert or update).

Exception Handling in Spring REST

Exception handling at controller level

@ExceptionHandler along with @ResponseStatus to map the exception to the custom method in controller which can handle all exception in that controller.In @ExceptionHandler annotation we can include the Exception classes which we need to handle for this controller.

Exception handling at application level

The new annotation allows the multiple scattered @ExceptionHandler from before to be consolidated into a single, global error handling component. The actual mechanism is extremely simple but also very flexible:

it allows full control over the body of the response as well as the status code

it allows mapping of several exceptions to the same method, to be handled together

it makes good use of the newer RESTful ResposeEntity response

```
public class ErrorInfo {
private String url;
private String message;
public ErrorInfo(String url, String message) {
this.url = url;
this.message = message;
}
public String getUrl() {
return url;
}
public void setUrl(String url) {
this.url = url;
}
public String getMessage() {
```

```
return message;
}
public void setMessage(String message) {
this.message = message;
}
}
```