Course Booklet for Spring framework

Know how of Spring frameworks
By Emertxe Trainers

version1.0 (May 29, 2012)

Table of Contents

Spring Framework Overview

The Spring Framework is a lightweight solution and a potential one-stop-shop for building your enterprise-ready applications. it is an open source Java platform and it was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

The Spring Framework supports declarative transaction management, remote access to your logic through RMI or web services, and various options for persisting your data. It offers a full-featured MVC framework, and enables you to integrate AOP transparently into your software.

Spring is designed to be non-intrusive, meaning that your domain logic code generally has no dependencies on the framework itself.

Advantages of using Spring Framework

>Spring enables developers to develop enterprise-class applications using POJOs.

>Spring is organized in a modular fashion.

>Testing an application written with Spring is simple because environment-dependent code is moved into this framework.

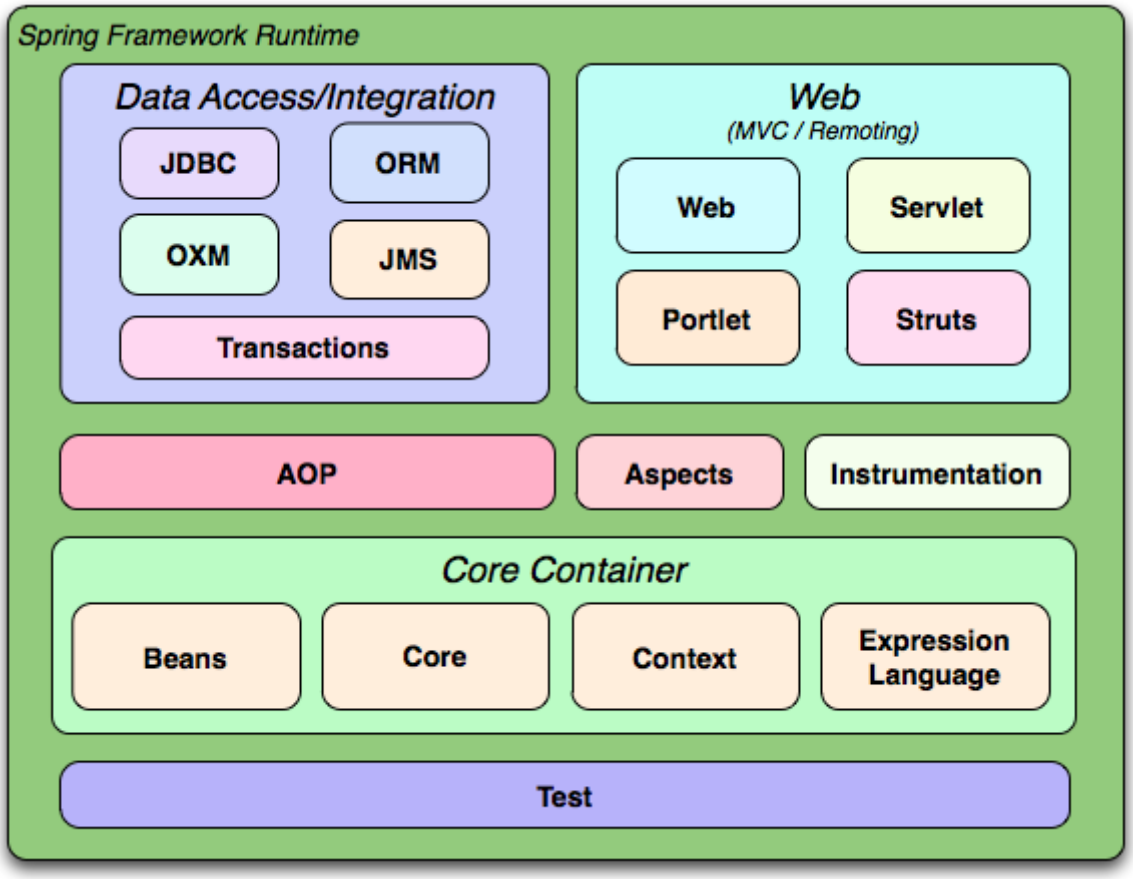
>Spring's web framework is a well-designed web MVC framework.

>Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

>Spring provides a consistent transaction management interface .

Spring Framework Architecture

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.



Modules

Core Container

The Core Container consists of the Core, Beans, Context, and Expression Language modules.

Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

Web

The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

AOP and Instrumentation

The AOP module provides aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.

The Instrumentation module provides class instrumentation support and class loader implementations to be used in certain application servers.

Test

The Test module supports the testing of Spring components with JUnit or TestNG frameworks.

Spring Environment Setup

Step 1 - Setup Java Development Kit(JDK)

Step 2 - Install Apache Common Logging API

You can download the latest version of Apache Commons Logging API from http://commons.apache.org/logging/.

Step 3 - Setup Eclipse IDE

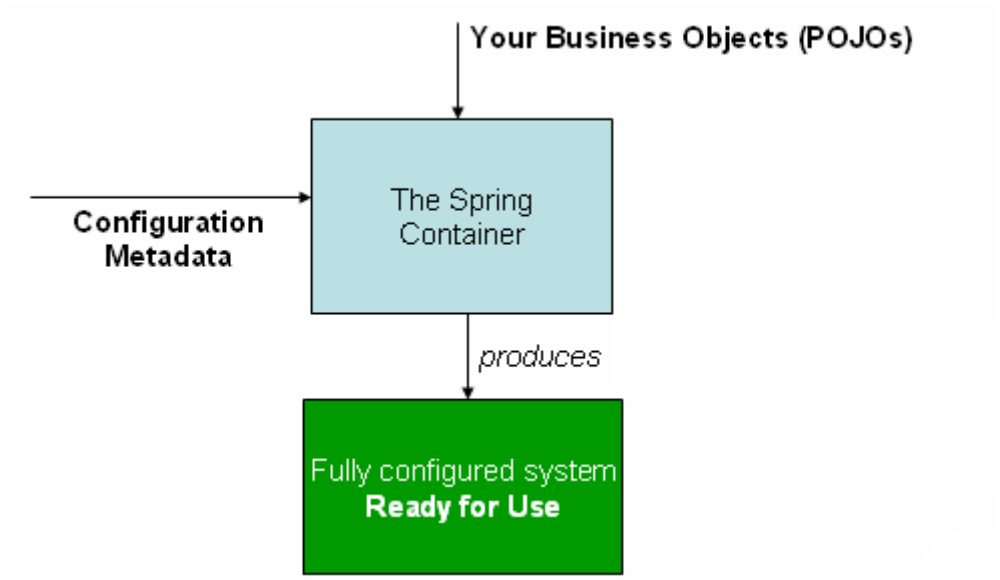
Step 4 - Setup Spring Framework Libraries

Download the latest version of Spring from http://www.springsource.org/download.

Spring Sample Example

Spring IoC Containers

IoC is also known as dependency injection(DI). It is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This process is fundamentally the inverse, hence the name Inversion of Control(IoC) .



The Spring IoC container

Configuration metadata

Sample snippet

```
<?xml version="1.0" encoding="UTF-8"?>
<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="ref id" class="path for class">
        <property name="field name" value="Value for the field"/>
   </bean>
</beans>
```

Spring Bean Definition

A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container,

The bean definition contains the information called configuration metadata which is needed for the container to know the followings:

>How to create a bean

>Bean's life cycle details

>Bean's dependencies

Bean Scopes

The Spring Framework supports following five scopes :

>singleton
If scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition.

>prototype
This scopes a single bean definition to have any number of object instances.

>request
This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

>session
This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

>global-session
This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Bean Life Cycle

When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required. Though, there is lists of the activities that take place behind the scenes between the time of bean instantiation and its destruction.

>Initialization callbacks
>Destruction callbacks

Bean Post Processors

The BeanPostProcessor interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic etc. You can also implement some custom logic after the Spring container finishes instantiating, configuring and initializing a bean by plugging in one or more BeanPostProcessor implementations.

The BeanPostProcessors operate on bean(or object) instances which means that the Spring IoC container instantiates a bean instance and then BeanPostProcessor interfaces do their work.

An ApplicationContext automatically detects any beans that are defined with implementation of theBeanPostProcessor interface and registers these beans as post-processors, to be then called appropriately by the container upon bean creation.

Bean Definition Inheritance

A bean definition can contain a lot of configuration information, including constructor arguments, property values, and container-specific information such as initialization method, static factory method name and so on.

A child bean definition inherits configuration data from a parent definition. The child definition can override some values or add others as needed.

Bean Definition Template

You can create a Bean definition template which can be used by other child bean definitions without putting much effort.

Sample code snippet:

```
 <bean id="beanTeamplate" abstract="true">
   <property name="field name" value="value of field"/>

</bean>
```

Spring Dependency Injection

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while doing unit testing. Dependency Injection(or sometime called wiring) helps in gluing these classes together and same time keeping them independent.

Constructor-based dependency injection :

Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.

Setter-based dependency injection :

Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

Injecting Inner Beans

Inner beans are beans that are defined within the scope of another bean. Thus, a <bean/> element inside the <property/> or <constructor-arg/> elements is called *inner bean*.

Sample snippet:

```
<bean id="outerBean" class="...">
    <property name="target">
      <bean id="innerBean" class="..."/>
    </property>
</bean>
```

Injecting Collection

If you want to pass plural values like Java Collection types List, Set, Map, and Properties. To handle the situation, Spring offers four types of collection configuration elements which are as follows:

<list>
This helps in wiring ie injecting a list of values, allowing duplicates.
<set>
This helps in wiring a set of values but without any duplicates.
<map>
This can be used to inject a collection of name-value pairs where name and value can be of any type.
<props>
This can be used to inject a collection of name-value pairs where the name and value are both Strings.

Developing an application by injecting Collection:

Beans Auto-Wiring

The Spring container can **autowire** relationships between collaborating beans without using <constructor-arg> and <property> elements which helps cut down on the amount of XML configuration you write for a big Spring based application.

Autowiring Modes

There are following autowiring modes which can be used to instruct Spring container to use autowiring for dependency injection. You use the autowire attribute of the <bean/> element to specify autowire mode for a bean definition.

>no
>byName
>byType
>constructor
>autodetect

Annotation Based Configuration :

From Spring 2.5 it became possible to configure the dependency injection using annotations. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method or field declaration.

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider to have following configuration file in case you want to use any annotation in your Spring application.

code snippet:

```
<?xml version="1.0" encoding="UTF-8"?>
  <beans xmlns="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:context="http://www.springframework.org/schema/context"
          xsi::schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context-3.0.xsd">
                  <context:annotation-config/>
                  <!-- bean definitions go here -->
</beans>
```

Java Based Configuration

Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.

@Configuration & @Bean Annotations

Annotating a class with the @Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions. The @Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

Event Handling in Spring

Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. So if a bean implements the ApplicationListener, then every time an ApplicationEvent gets published to the ApplicationContext, that bean is notified.

Listening to Context Events

To listen a context event, a bean should implement the ApplicationListener interface which has just one method onApplicationEvent(). So let us write an example to see how the events propagates and how you can put your code to do required task based on certain events.

AOP with Spring Framework

Aspect Oriented Programming entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects like logging, auditing, declarative transactions, security, and caching etc.

AOP Terminologies

Types of Advice

Custom Aspects Implementation

@AspectJ Based AOP with Spring

@AspectJ refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.

Declaring an aspect

Declaring a pointcut

Declaring advices

AOP sample application

Spring JDBC Framework

While working with database using plain old JDBC, it becomes cumbersome to write unnecessary code to handle exceptions, opening and closing database connections etc. But
Spring JDBC Framework takes care of all the low-level details starting from opening the connection, prepare and execute the SQL statement, process exceptions, handle transactions and finally close the connection.

So what you have do is just define connection parameters and specify the SQL statement to be executed and do the required work for each iteration while fetching data from the database.

Spring JDBC provides several approaches and correspondingly different classes to interface with the database. I'm going to take classic and the most popular approach which makes use of JdbcTemplate class of the framework. This is the central framework class that manages all the database communication and exception handling.

JdbcTemplate Class

DAO support

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate, JPA or JDO in a consistent way. This allows one to switch between the aforementioned persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

An sample Application using jdbc spring framework

Spring Transaction Management

A database transaction is a sequence of actions that are treated as a single unit of work. These actions should either complete entirely or take no effect at all. Transaction management is an important part of and RDBMS oriented enterprise applications to ensure data integrity and consistency. The concept of transactions can be described with following four key properties described as ACID:

1. **Atomicity**: A transaction should be treated as a single unit of operation which means either the entire sequence of operations is successful or unsuccessful.
2. **Consistency**: This represents the consistency of the referential integrity of the database, unique primary keys in tables etc.
3. **Isolation**: There may be many transactions processing with the same data set at the same time, each transaction should be isolated from others to prevent data corruption.
4. **Durability**: Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

A real RDBMS database system will guarantee all the four properties for each transaction. The simplistic view of a transaction issued to the database using SQL is as follows:

>Begin the transaction using begin transaction command.
>Perform various deleted, update or insert operations using SQL queries.
>If all the operation are successful then perform commit otherwise rollback all the operations.

Spring framework provides an abstract layer on top of different underlying transaction management APIs. The Spring's transaction support aims to provide an alternative to EJB transactions by adding transaction capabilities to POJOs. Spring supports both programmatic and declarative transaction management. EJBs requires an application server, but Spring transaction management can be implemented without a need of application server.

Local vs. Global Transactions

Programmatic vs. Declarative

Spring supports two types of transaction management:

1. **Programmatic transaction management:** This means that you have manage the transaction with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.

2. **Declarative transaction management**: This means you separate transaction management from the business code. You only use annotations or XML based configuration to manage the transactions.

Spring Transaction Abstractions

Declarative Transaction Management

Declarative transaction management approach allows you to manage the transaction with the help of configuration instead of hard coding in your source code. This means that you can separate transaction management from the business code. You only use annotations or XML based configuration to manage the transactions. The bean configuration will specify the methods to be transactional

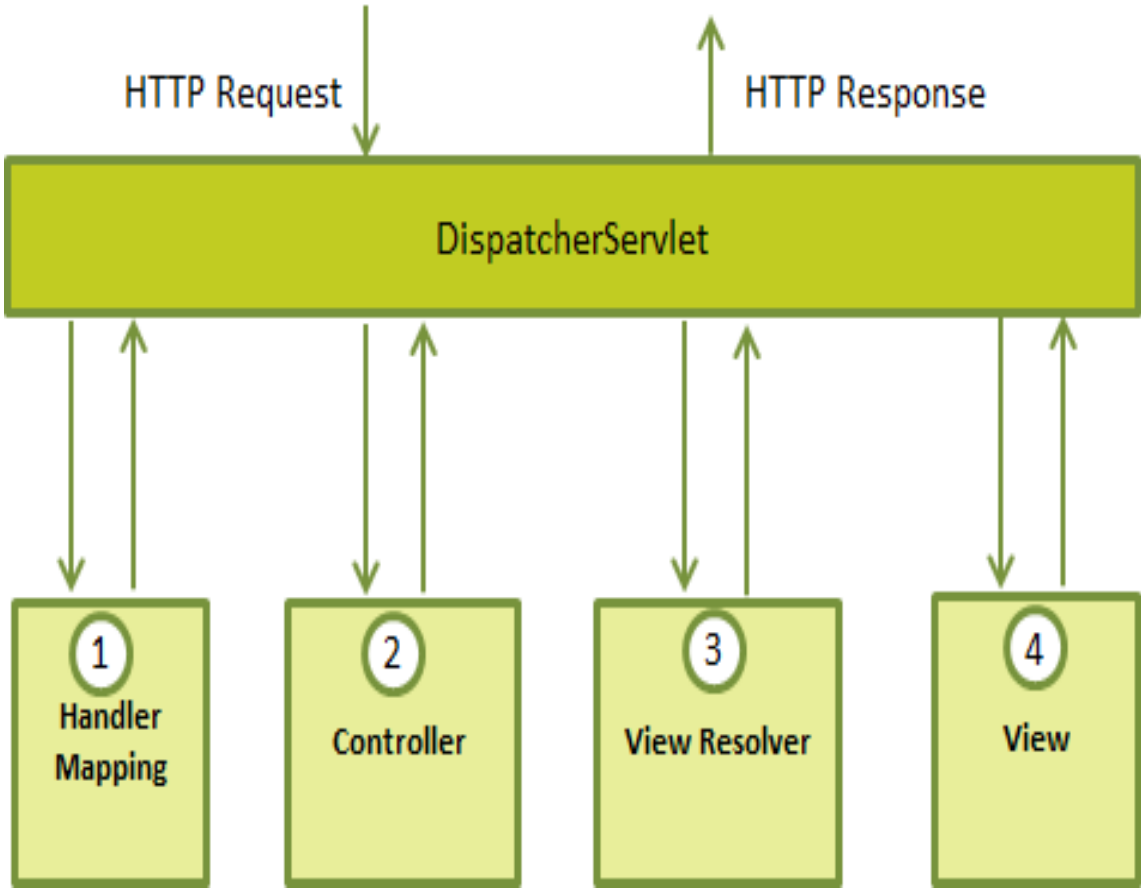
Spring Web MVC Framework

The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

> The Model encapsulates the application data and in general they will consist of POJO.
> The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
> The Controller is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

The DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC DispatcherServlet is illustrated in the following diagram:



Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet:

1. After recieving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller.
2. The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.
3. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.
4. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

Creating JSP Views

Spring MVC supports many types of views for different presentation technologies. These include - JSPs, HTML, PDF, Excel worksheets, XML, Velocity templates, XSLT, JSON, Atom and RSS feeds, JasperReports etc. But most commonly we use JSP templates written with JSTL.