**SPRING**

Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.

Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

**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 save 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**

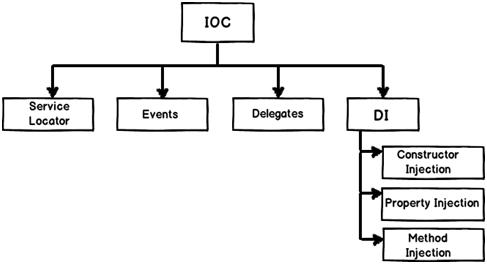
It provides declarative support for caching, validation, transactions and formatting.

**Inversion of Control (IOC) and**

**Dependency Injection**

These are the design patterns that are used to remove dependency from the programming code. They make our programming code loosely coupled which resulted to an application that is easier to maintain and test.

In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by XML file or annotation.



Inversion of Control (IoC) – means that there are other objects that create objects for you. They are giving control to the container to get an instance. So instead of creating an object using the new operator, they get the objects that they need from an external source (like xml configuration file).

Dependency Injection (DI) - is a subset of IoC; it is a pattern through which to implement IoC, where the control being inverted is the setting of object’s dependencies. The act of connecting objects with other objects, or “injecting” objects into other objects, and at the same time keeping them independent.

Dependency Injection in Spring can be done Constructors and Setter methods.

Scenario:

Consider you have an application which has a text editor component and you want to provide a spell check. Your standard code would look something like this –

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor() {

spellChecker = new SpellChecker();

}

}

What we've done here is, create a dependency between the TextEditor and the SpellChecker. In an inversion of control scenario, we would instead do something like this –

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor(SpellChecker spellChecker) {

this.spellChecker = spellChecker;

}

}

Here, the TextEditor should not worry about SpellChecker implementation. The SpellChecker will be implemented independently and will be provided to the TextEditor at the time of TextEditor instantiation.

**Tight Coupling and Loose Coupling**

**Tight Coupling** – In Tight coupling, classes and objects are dependent on one another. I must say that, tight coupling is usually bad because it reduces flexibility and re-usability of code and we are not able to achieve complete object originated programming features

- Tightly Coupled Object is an object that needs to know about other objects and are usually highly dependent on each other's interfaces. When we change one object in a tightly coupled application often it requires changes to a number of other objects. There is no problem in a small application we can easily identify the change. But in the case of a large applications these inter-dependencies are not always known by every consumer or other developers or there is many chance of future changes.

**Loose Coupling** – loose coupling means reducing dependencies of a class that use a different class directly. Loose coupling promotes greater reusability, easier maintainability

-Loose coupling is a design strategy which allows us to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component will require changes in any other component. It’s all about thinking a problem in generic manner and which intended to increase the flexibility of a system, make it more maintainable, and makes the entire framework more stable.

Non-technical explanation (real world example):

(1) Let’s say we have a ComputerTechnician. And we can tell it to formatPC(). And he does it by using a WindowsXP. Normally you wouldn’t care what OS he uses to format the PC,only that he does format PCs. Internally, he will formatPC() using his own OS which is Windows XP. *(tight coupling).*

Lets’ say a DifferentComputerTechnician has no OS. But he also formatPC() using an OS. And you give him Windows7.

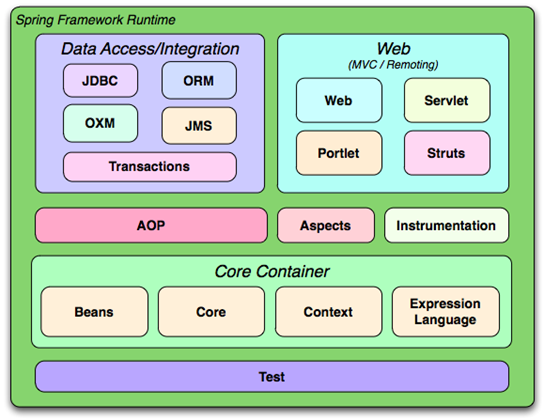
Now you've injected a dependency. The DifferentComputerTechnician is dependent to Windows7, but Windows7 did not belong to him, unlike ComputerTechnician and his WindowsXP. *(loose coupling).*

(2) The Hat is "loosely coupled" to the body. This means you can easily take the hat off without making any changes to the the person/body. When you can do that then you have "loose coupling. See below for elaboration.

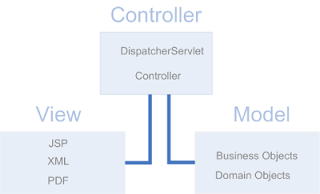
Think of your skin. It's stuck to your body. It fits like a glove. But what if you wanted to change your skin colour from say white to black? Can you imagine just how painful it would be to peel off your skin, dye it, and then to paste it back on etc? Changing your skin is difficult because it is tightly coupled to your body. You just can't make changes easily. You would have to fundamentally redesign a human being in order to make this possible.

**SPRING MODULES**

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.



**SPRING MVC**

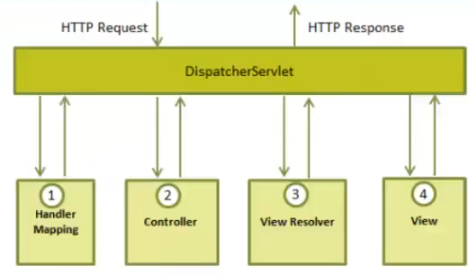


In Spring MVC DispatcherServlet and Controllers act as the Controller and they receive the requests and decide how the request will be served and also decide which view is used. Business objects and domain objects are the Model and the business objects are invoked by the controllers and the all data are in domain objects. The domain objects will be passed to some JSP which are the View part. The JSP will render the data in the domain objects.

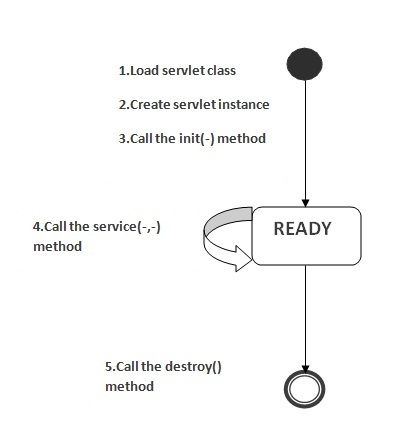
**DisptacherServlet**

DispatcherServlet is the front controller in Spring MVC.   It is a HttpServlet that will receive the request and return the response.  DispatcherServlet is the key player in SpringMVC.  From the below brief work flow of SpringMVC you can see DispatcherServlet is the driving force that make the request served with the right response sent back to the client.    

1. Receive the request from client
2. Consult HandlerMapping to decide which controller processes the request
3. Dispatch the request to the controller
4. Controller processes the request and returns the logical view name and model back to DispatcherServlet
5. Consult ViewResolver for appropriate View for the logical view name from Controller
6. Pass the model to View implementation for rendering
7. View renders the model and returns the result to DispatcherServlet
8. Return the rendered result from view to the client



**Life Cycle of a Servlet**



**Servlet class is loaded**

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

**Servlet instance is created**

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

**init() method is invoked**

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.



**service() method is invoked**

The service() method is the main method to perform the actual task (client requests). The servlet container calls the service() method to handle requests coming from the client and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, etc.) and calls doGet, doPost, etc. methods as appropriate.



**destroy() method is invoked**

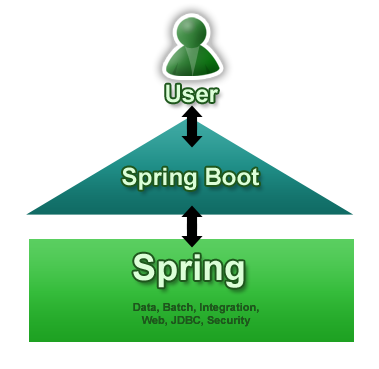
The destroy() method is called only once at the end of the life cycle of a servlet. This method gives you an opportunity to clean up any resource for example memory, thread etc.

After the destroy() method is called, the servlet object is marked for garbage collection.



**Spring Boot**

* Spring Boot makes building Spring applications easier by requiring absolutely no xml configuration. It takes most of the work out of configuring Spring-based applications
* The goal of Spring Boot is to provide a set of tools for quickly building Spring applications that are easy to configure.
* Spring is a very popular Java-based framework for building web and enterprise applications. Unlike many other frameworks, which focus on only one area, Spring framework provides a wide verity of features addressing the modern business needs
* You can use Spring Boot to create stand-alone Java applications that can be started using java -jar or more traditional WAR deployments. We also provide a command line tool that runs spring scripts.
* If one has to start a new spring project we have to add build path or add maven dependencies, configure application server, add spring configuration . So a lot of effort is required to start a new spring project as we have to currently do everything from scratch. Spring Boot is the solution to this problem. Spring boot has been built on top of existing spring framework. Using spring boot we avoid all the boilerplate code and configurations that we had to do previously. Spring boot thus helps us use the existing Spring functionalities more robustly and with minimum efforts.



**Opionated and Non-opinionated**

Opinionated software means that there is basically one way (the right way) to do things which is nice and easy and trying to do it differently will be difficult.

Most of the framework conventions are enforced by the framework, and it might refuse to work properly (if at all) if you attempt to bypass them. This ensures that all developers follow strict conventions and use the same tools

Non-opinionated software, on the other hand, leaves lots of flexibility to the user (developer). It doesn't proscribe one method of solving a problem, but provides flexible tools that can be used to solve the problem in many ways.

But that freedom often comes at the expense of giving more room for developers to introduce messy code, or having to go through the hassle of integrating framework components, or having less uniform support in the community

**The primary goals of Spring Boot are:**

* To provide a radically faster and widely accessible ‘getting started’ experience for all Spring development
* To be opinionated out of the box, but get out of the way quickly as requirements start to diverge from the defaults
* To provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration)

**Features of Spring Boot**

Auto-Configuration - No need to manually configure dispatcher servlet, static resource mappings, property source loader, message converters etc.

Dependency Management - The different versions of commonly used libraries are pre-selected and grouped in different starter POMs that we can include in your project. By selecting one Spring Boot version we are implicitly selecting dozens of dependencies that we would have to otherwise select and harmonize ourself.

Advanced Externalized Configuration - There is a large list of bean properties that can be configured through application.properties file without touching java or xml config.

Production support- We get health checking, application and jvm metrics, jmx via http and a few more things for free.

Runnable Jars - We can package your application as a runnable jar with embedded tomcat included so it presents a self-contained deployment unit

**Benefits of Spring Boot:**

* It makes it easier to develop Spring-based applications with Java or Groovy. Also, it reduces Developer’s effort with the “Opinionated Defaults Configuration” approach.
* It minimizes writing multiple boilerplate codes, XML configuration and annotation, ultimately enhancing productivity while reducing lots of development time.
* It makes it easier to integrate the Spring Boot Application with the Spring Ecosystem that majorly includes Spring ORM, Spring JDBC, Spring Security, Spring Data and many other things.
* It tests the web applications easily with the help of different Embedded HTTP servers that includes Tomcat, Jetty and many others.
* It offers Command Line Interface (CLI) tool for developing and testing Spring Boot.
* It offers a number of plugins for developing and testing Spring Boot Applications easily using Maven/Gradle- the build tools.
* It offers a number of plugins for working with embedded and in-memory databases easily.

<https://www.ibm.com/developerworks/library/j-spring-boot-basics-perry/>

Enter Spring Boot. The Spring Boot website says it much more succinctly than I can:

“ 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.”

Basically, this means you can quickly get a Spring application up and running with very little configuration. What little configuration there is comes in the form of annotations, so no XML.

**Spring Security**

Spring Security is a framework that provides security services to spring-based Java web application.

It focuses on providing both authentication and authorization

**Areas for application securities**

Authentication: Process of checking the identity of the user.

Authorization: Process of deciding whether a user is allowed to perform an activity within the application.

**Authentication Models**

Spring Security framework supports wide range of authentication models. These models either provided by third parties or framework itself. Some of them are

1. X.509 client certificate exchange

2. LDAP Authentication

3. OpenID authentication

4. Java Open Source Single Sign On

**Spring Batch**

Spring Batch is a framework for developing batch processing applications.

Batch processing is a processing mode which involves execution of series of automated complex jobs. Batch process jobs can run without any end-user interaction or can be scheduled to start up on their own as resources permit.

This framework provides functions for

• Bulk processing

• Including logging and tracing

• Transaction management

• Job processing statistics

• Job restart

• Skip and Resource management

**Spring Social**

Spring Social supports integration with Software-as-a-Service (SaaS) providers like Facebook, Twitter, LinkedIn, GitHub, etc.

In SaaS, a service provider hosts the application at its data center and a customer accesses it via Internet.

The benefit of this approach is that clients do not need to manage the software installation, and that they also do not need expensive hardware. One of the problems is that the data is also on the site of the company running the software - that is "in the cloud". And getting the data may be difficult.

For example, Gmail is a type of a SaaS mail provider because you don't have to manage any service yourself and it’s all done by the vendor (Google in this example).

**Spring AOP**

AOP (Aspect Oriented Programming) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns (e.g. logging, transaction management, data validation).

It does so by adding additional behavior to existing code without modification of the code itself. Instead, we can declare this new code and these new behaviors separately.

**Scenario**

In a class, there are 5 methods that start in m, 2 methods that start in n, and 3 methods that start in p.

Understanding Scenario: I have to maintain log and send notification after calling methods that starts from m.

Problem without AOP: We can call methods (that maintains log and sends notification) from the methods starting with m. In such scenario, we need to write the code in all the 5 methods.

But, if client says in future, I don't have to send notification, you need to change all the methods. It leads to the maintenance problem.

Solution with AOP: We don't have to call methods from the method. Now we can define the additional concern like maintaining log, sending notification etc. in the method of a class. Its entry is given in the xml file.

In future, if client says to remove the notifier functionality, we need to change only in the xml file. So, maintenance is easy in AOP.

**AOP Core Concepts**

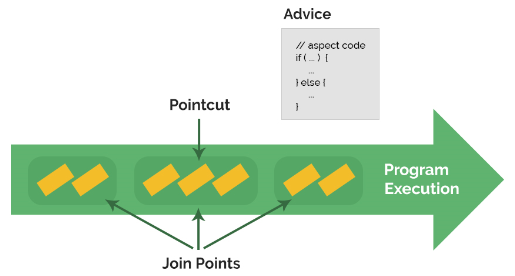
Advise defines what needs to be apply. Different types of advice include “around,” “before” and “after” advice.

Joinpoint is where an Advice is apply. A Joinpoint is a point during the execution of a program

Pointcut is a combination of different Jointpoints.

A Pointcut is a predicate that helps match an Advice to be applied by an Aspect at a particular JoinPoint.

Aspect is applying an Advice at Pointcuts.



**AOP Advice Types**

Before Advice: it executes before a join point.

After Returning Advice: it executes after a joint point completes normally.

After Throwing Advice: it executes if method exits by throwing an exception.

After (finally) Advice: it executes after a join point regardless of join point exit whether normally or exceptional return.

Around Advice: It executes before and after a join point.

**Spring Transaction**

Transactions are basically units of work (ie changes to something) that are managed as a single operation that can be either committed or rolled back.

**Spring supports 2 types of transaction management:**

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

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.

**Transactions without XML**

@Configuration

@EnableTransactionManagement

public class PersistenceJPAConfig{

@Bean

public LocalContainerEntityManagerFactoryBean

entityManagerFactoryBean(){

//...

}

@Bean

public PlatformTransactionManager transactionManager(){

JpaTransactionManager transactionManager

= new JpaTransactionManager();

transactionManager.setEntityManagerFactory(

entityManagerFactoryBean().getObject() );

return transactionManager;

}

}

**Transactions with XML**

<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="myEmf" />

</bean>

<tx:annotation-driven transaction-manager="txManager" />

**@Transactional Annotation**

@Service

@Transactional

public class FooService {

//...

}

**Spring Data JPA**

Simplifies data access from relational and NoSQL data stores.

**Advantage of Spring JpaTemplate**

You don't need to write the before and after code for persisting, updating, deleting or searching object such as creating Persistence instance, creating EntityManagerFactory instance, creating EntityTransaction instance, creating EntityManager instance, commiting EntityTransaction instance and closing EntityManager.

So, it save a lot of code.

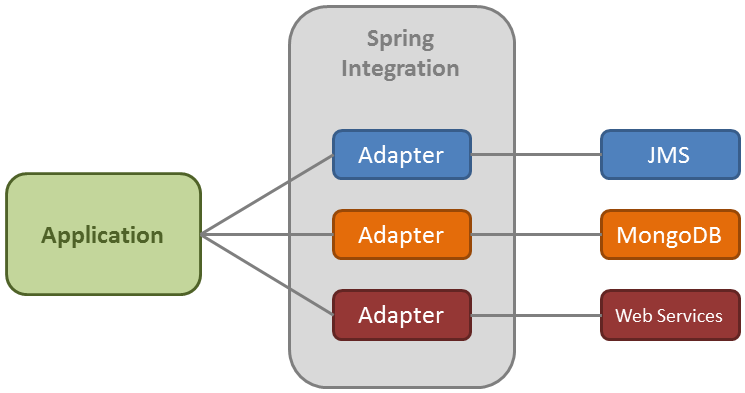
**Spring jdbc**

**Spring Integration**

Spring Integration provides a lot of powerful components that can greatly enhance the interconnectivity of systems and processes within an enterprise architecture.

Spring Integration abstracts message sources and destinations and uses message passing and message manipulation to integrate various components within the application environment. Applications built with Spring Integration are able to send messages between components, either across a message bus to another server in your environment or even to another class within the same virtual machine.

Spring Integration code is deployable as a standalone project within JavaSE as well as part of something larger in a JavaEE environment.



**Spring Integration Components**

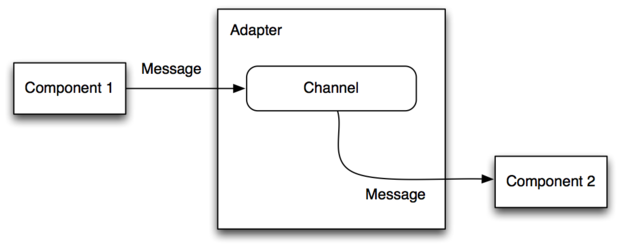
Message is the unit of data transfer within a Spring Integration context.

Channel is the basic plumbing in an integration architecture. It’s the pipe by which messages are relayed from one system to another. Example is Point-to-Point (P2P) Channel

Bridge is used to connect two message channels or adapters if for any reason they can’t connect directly.

Service Activator is any POJO that defines the @ServiceActivator annotation on a given method. This allows us to execute any method on our POJO when a message is received from an inbound channel, and it allows us to write messages to an outward channel.

Adapter is an enterprise integration pattern-based component that allows one to “plug-in” to a system or data source. It is almost literally an adapter as we know it from plugging into a wall socket or electronic device.



**Spring Boot Admin Server**

Spring Boot Admin is a web application, used for managing and monitoring Spring Boot applications. Each application is considered as a client and registers to the admin server.

**Spring OAuth**

Spring Security OAuth provides support for using Spring Security with OAuth (1a) and OAuth2 using standard Spring and Spring Security programming models and configuration idioms.

OAuth2.0 is an open authorization protocol, which allows accessing the resources of the resource owner by enabling the client applications on HTTP services such as Facebook, GitHub, etc. It allows sharing of resources stored on one site to another site without using their credentials. It uses username and password tokens instead.

**Spring Statemachine**

The Spring Statemachine project aims to provide a common infrastructure to work with state machine concepts in Spring applications.

State Machine is any device that stores the status of something at a given time and can operate on input to change the status and/or cause an action or output to take place for any given change. A computer is basically a state machine