1. What are the major features in different versions of Spring Framework?

Features of Spring Framework

Version	Feature
Spring 2.5	This version was released in 2007. It was the first version which supported annotations.
Spring 3.0	This version was released in 2009. It made full-fledged use of improvements in Java5 and also provided support to JEE6.
Spring 4.0	This version was released in 2013. This was the first version to provide full support to Java 8.

2. What is a Spring Framework?

- Spring is a powerful open source, application framework created to reduce the complexity of enterprise application development.
- It is light-weighted and loosely coupled.
- It has layered architecture, which allows you to select the components to use, while also providing a cohesive framework for J2EE application development.
- Spring framework is also called the framework of frameworks as it provides support to various other frameworks such as Struts, Hibernate, EJB, JSF etc.

3. List the advantages of Spring Framework.

- Because of Spring Frameworks layered architecture, you can use what you need and leave which you don't.
- Spring Framework enables POJO (Plain Old Java Object) Programming which in turn enables continuous integration and testability.
- JDBC is simplified due to Dependency Injection and Inversion of Control.
- It is open-source and has no vendor lock-in.

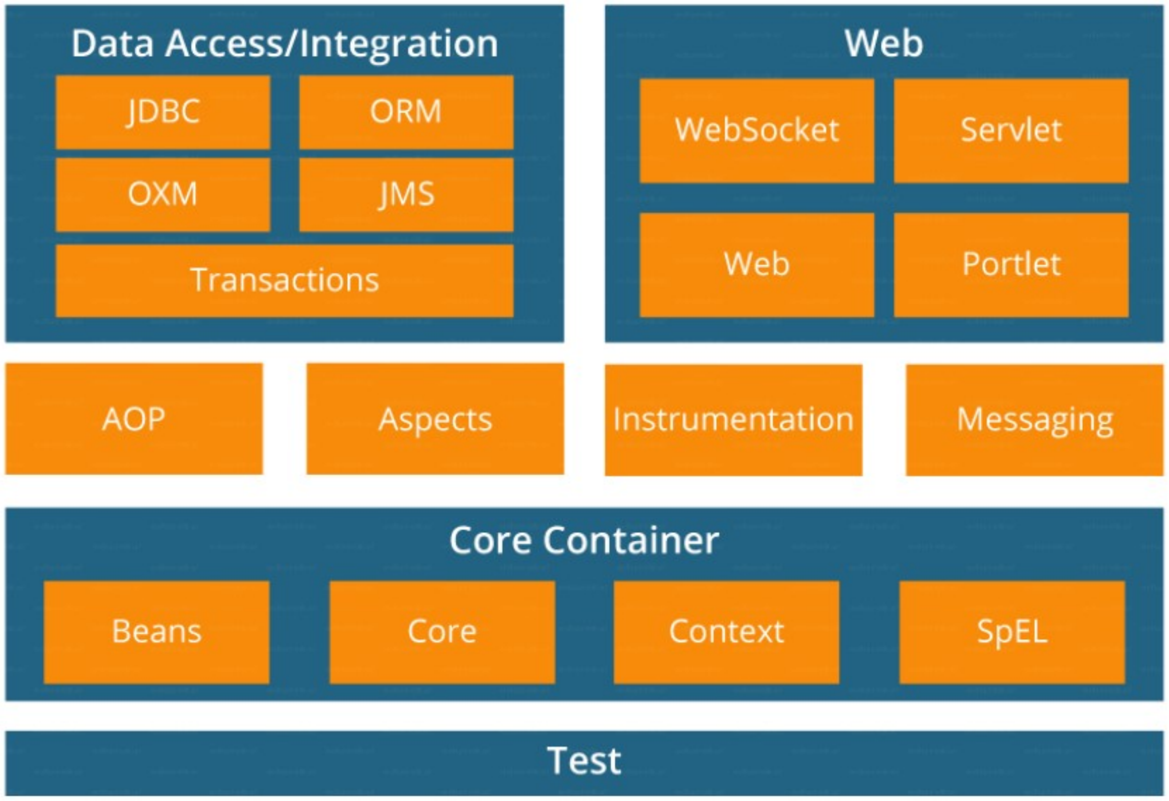
4. What are the different features of Spring Framework?

Following are some of the major features of Spring Framework :

- **Lightweight:** Spring is lightweight when it comes to size and transparency.
- **Inversion of control (IOC):** The objects give their dependencies instead of creating or looking for dependent objects. This is called Inversion Of Control.
- **Aspect oriented Programming (AOP):** Aspect oriented programming in Spring supports cohesive development by separating application business logic from system services.
- **Container:** Spring Framework creates and manages the life cycle and configuration of the application objects.
- **MVC Framework:** Spring Framework's MVC web application framework is highly configurable. Other frameworks can also be used easily instead of Spring MVC Framework.
- **Transaction Management:** Generic abstraction layer for transaction management is provided by the Spring Framework. Spring's transaction support can be also used in container less environments.
- **JDBC Exception Handling:** The JDBC abstraction layer of the Spring offers an exception hierarchy, which simplifies the error handling strategy.

5. How many modules are there in Spring Framework and what are they?

There are around 20 modules which are generalized into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation and Test.



- **Spring Core Container –** This layer is basically the core of Spring Framework. It contains the following modules :

a. Spring Core
b. Spring Bean
c. SpEL (Spring Expression Language)
d. Spring Context

- **Data Access/Integration –** This layer provides support to interact with the database. It contains the following modules :

a. JDBC (Java DataBase Connectivity)
b. ORM (Object Relational Mapping)
c. OXM (Object XML Mappers)
d. JMS (Java Messaging Service)
e. Transaction

- **Web –** This layer provides support to create web application. It contains the following modules :

a. Web
b. Web – MVC
c. Web – Socket
d. Web – Portlet

- **Aspect Oriented Programming (AOP) –** In this layer you can use Advices, Pointcuts etc., to decouple the code.
- **Instrumentation –** This layer provides support to class instrumentation and classloader implementations.
- **Test –** This layer provides support to testing with JUnit and TestNG.

Few Miscellaneous modules are given below:

- **Messaging –** This module provides support for STOMP. It also supports an annotation programming model that is used for routing and processing STOMP messages from WebSocket clients.
- **Aspects –** This module provides support to integration with AspectJ.

6. What is a Spring configuration file?

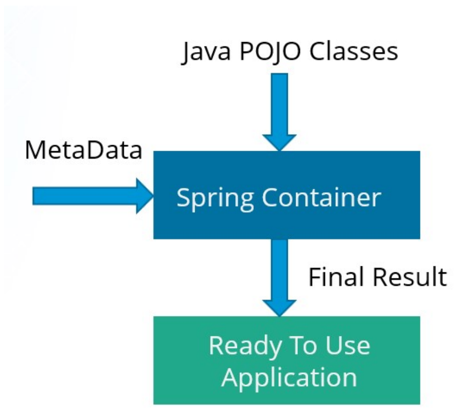
A Spring configuration file is an XML file. This file mainly contains the classes information. It describes how those classes are configured as well as introduced to each other. The XML configuration files, however, are verbose and more clean. If it's not planned and written correctly, it becomes very difficult to manage in big projects.

7. What are the various ways of using Spring Framework?

Spring Framework can be used in various ways. They are listed as follows:

1. As a Full-fledged Spring web application.
2. As a third-party web framework, using Spring Frameworks middle-tier.
3. For remote usage.
4. As Enterprise Java Bean which can wrap existing POJOs (Plain Old Java Objects).

9. What is Spring IOC Container?



At the core of the Spring Framework, lies the Spring container.

The container creates the object, wires them together, configures them and manages their complete life cycle.

The Spring container makes use of Dependency Injection to manage the components that make up an application.

The container receives instructions for which objects to instantiate, configure, and assemble by reading the configuration metadata provided. This metadata can be provided either by XML, Java annotations or Java code.

10. What do you mean by Dependency Injection?

In Dependency Injection, you do not have to create your objects but have to describe how they should be created. You don't connect your components and services together in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

11. In how many ways can Dependency Injection be done?

In general, dependency injection can be done in three ways, namely :

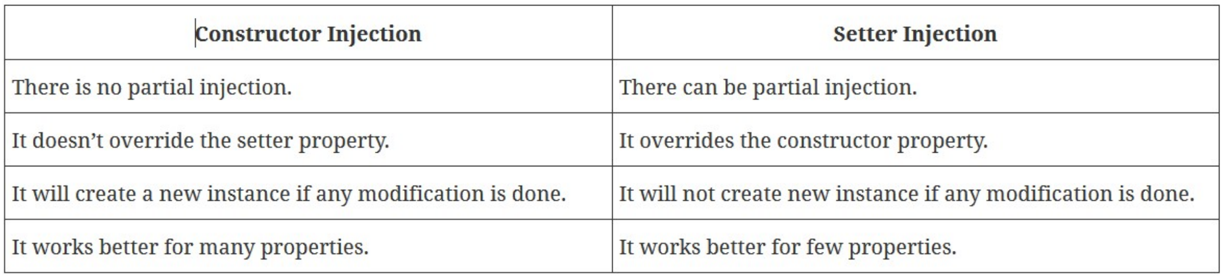
- Constructor Injection
- Setter Injection
- Interface Injection

In Spring Framework, only constructor and setter injections are used.

REFER to Class Notes

12. Differentiate between constructor injection and setter injection.

Constructor Injection vs Setter Injection



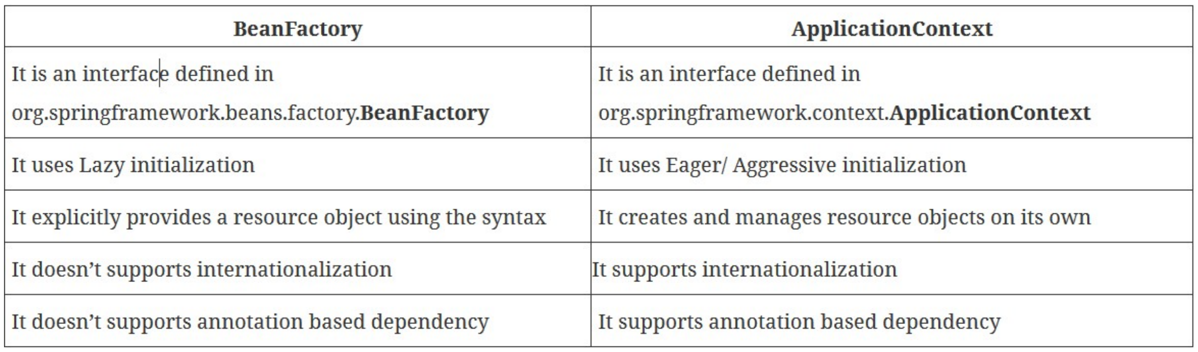
Constructor Injection	Setter Injection
There is no partial injection.	There can be partial injection.
It doesn't override the setter property.	It overrides the constructor property.
It will create a new instance if any modification is done.	It will not create new instance if any modification is done.
It works better for many properties.	It works better for few properties.

13. How many types of IOC containers are there in spring?

a. **BeanFactory**: BeanFactory is like a factory class that contains a collection of beans. It instantiates the bean whenever asked for by clients.
b. **ApplicationContext**: The ApplicationContext interface is built on top of the BeanFactory interface. It provides some extra functionality on top BeanFactory.

14. Differentiate between BeanFactory and ApplicationContext.

BeanFactory vs ApplicationContext



BeanFactory	ApplicationContext
It is an interface defined in org.springframework.beans.factory.**BeanFactory**	It is an interface defined in org.springframework.context.**ApplicationContext**
It uses Lazy initialization	It uses Eager/ Aggressive initialization
It explicitly provides a resource object using the syntax	It creates and manages resource objects on its own
It doesn't supports internationalization	It supports internationalization
It doesn't supports annotation based dependency	It supports annotation based dependency

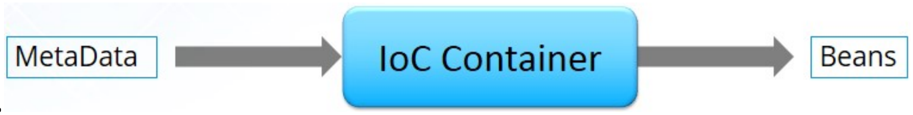
15. the benefits of IoC.

Some of the benefits of IoC are:

- It will minimize the amount of code in your application.
- It will make your application easy to test because it doesn't require any singletons or JNDI lookup mechanisms in your unit test cases.
- It promotes loose coupling with minimal effort and least intrusive mechanism.
- It supports eager instantiation and lazy loading of the services.

16. Explain Spring Beans?

- They are the objects that form the backbone of the user's application.
- Beans are managed by the Spring IoC container.
- They are instantiated, configured, wired and managed by a Spring IoC container
- Beans are created with the configuration metadata that the users supply to the container.



17. How configuration metadata is provided to the Spring container?

Configuration metadata can be provided to Spring container in following ways:

- **XML-Based configuration:** In Spring Framework, the dependencies and the services needed by beans are specified in configuration files which are in

XML format. These configuration files usually contain a lot of bean definitions and application specific configuration options. They generally start with a bean tag. For example:

```
<bean id="studentbean" class="com.adam.app.StudentBean">
 <property name="name" value="adam"></property>

•       </bean>
```

- **Annotation-Based configuration**: Instead of using XML to describe a bean wiring, you can configure the bean into the component class itself by using annotations on the relevant class, method, or field declaration. By default, annotation wiring is not turned on in the Spring container. So, you need to enable it in your Spring configuration file before using it. For example:

```
    • <beans>
    • <context:annotation-config/>
    • <!-- bean definitions go here -->
    • </beans>
```

- **Java-based configuration:** The key features in Spring Framework's new Java-configuration support are @Configuration annotated classes and @Bean annotated methods.

1. @Bean annotation plays the same role as the <bean/> element.

2.@Configuration classes allows to define inter-bean dependencies by simply calling other @Bean methods in the same class.

```
@Configuration
```

```
public class StudentConfig
{
@Bean
public StudentBean myStudent()
{ return new StudentBean(); }
}
```

18. How many bean scopes are supported by Spring?

The Spring Framework supports five scopes. They are:

- **Singleton:** This provides scope for the bean definition to single instance per Spring IoC container.
- **Prototype:** This provides scope for a single bean definition to have any number of object instances.
- **Request:** This provides scope for a bean definition to an HTTP-request.
- **Session:** This provides scope for a bean definition to an HTTP-session.
- **Global-session:** This provides scope for a bean definition to an Global HTTP-session.

The last three are available only if the users use a web-aware ApplicationContext.

Define Bean Wiring.

When beans are combined together within the Spring container, it's called wiring or bean wiring. The Spring container needs to know what beans are needed and how the container should use dependency injection to tie the beans together, while wiring beans.

What do you understand by auto wiring and name the different modes of it?

The Spring container is able to autowire relationships between the collaborating beans. That is, it is possible to let Spring resolve collaborators for your bean automatically by inspecting the contents of the BeanFactory. Different modes of bean auto-wiring are:

a. **no:** This is default setting which means no autowiring. Explicit bean reference should be used for wiring.
b. **byName:** It injects the object dependency according to name of the bean. It matches and wires its properties with the beans defined by the same names in the XML file.
c. **byType:** It injects the object dependency according to type. It matches and wires a property if its type matches with exactly one of the beans name in XML file.
d. **constructor:** It injects the dependency by calling the constructor of the class. It has a large number of parameters.
e. **autodetect:** First the container tries to wire using autowire by *constructor*, if it can't then it tries to autowire by *byType*.

What do you mean by Annotation-based container configuration?

Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class itself by using annotations on the relevant class, method, or field declaration. It acts as an alternative to XML setups. For example:

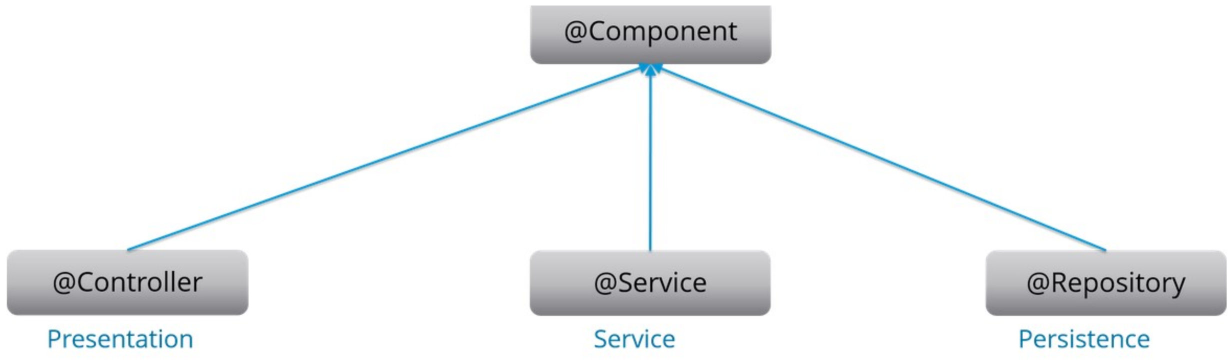
```
@Configuration
public class AnnotationConfig
{
@Bean
public MyDemo myDemo()
 { return new MyDemoImpl(); }
}
```

How annotation wiring can be turned on in Spring?

By default, Annotation wiring is not turned on in the Spring container. Thus, to use annotation based wiring we must enable it in our Spring configuration file by configuring **<context:annotation-config/>** element. For example:

```
<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">
<context:annotation-config/>
<beans ………… />
</beans>
```

What's the difference between @Component, @Controller, @Repository & @Service annotations in Spring?



@Component: This marks a java class as a bean. It is a generic stereotype for any Spring-managed component. The component-scanning mechanism of spring now can pick it up and pull it into the application context.

@Controller: This marks a class as a Spring Web MVC controller. Beans marked with it are automatically imported into the Dependency Injection container.

@Service: This annotation is a specialization of the component annotation. It doesn't provide any additional behavior over the @Component annotation. You can use @Service over @Component in service-layer classes as it specifies intent in a better way.

@Repository: This annotation is a specialization of the @Component annotation with similar use and functionality. It provides additional benefits specifically for DAOs. It imports the DAOs into the DI container and makes the unchecked exceptions eligible for translation into Spring DataAccessException.

What do you understand by @Autowired annotation?

The **@Autowired** annotation provides more accurate control over where and how autowiring should be done. This annotation is used to autowire bean on the setter methods, constructor, a property or methods with arbitrary names or multiple arguments. By default, it is a type driven injection.

```
public class Employee
{
private String name;
@Autowired
public void setName(String name)
{this.name=name; }
public string getName()
{ return name; }
}
```

What do you understand by @Qualifier annotation?

When you create more than one bean of the same type and want to wire only one of them with a property you can use the **@Qualifier** annotation along with **@Autowired** to remove the ambiguity by specifying which exact bean should be wired.

For example, here we have two classes, Employee and EmpAccount respectively. In EmpAccount, using @Qualifier its specified that bean with id emp1 must be wired.

```
public class Employee
{
```

```
private String name;
@Autowired
public void setName(String name)
{ this.name=name; }
public string getName()
{ return name; }
}
```

```
public class EmpAccount
{
private Employee emp;
@Autowired
@Qualifier(emp1)
public void showName()
{
System.out.println("Employee name : "+emp.getName);
}
}
```

What do you understand by @RequestMapping annotation?

@RequestMapping annotation is used for mapping a particular HTTP request method to a specific class/ method in controller that will be handling the respective request. This annotation can be applied at both levels:

- **Class level** : Maps the URL of the request
- **Method level**: Maps the URL as well as HTTP request method

Describe Spring DAO support?

The Data Access Object (DAO) support in Spring makes it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the persistence technologies easily. It also allows you to code without worrying about catching exceptions that are specific to each of these technology.

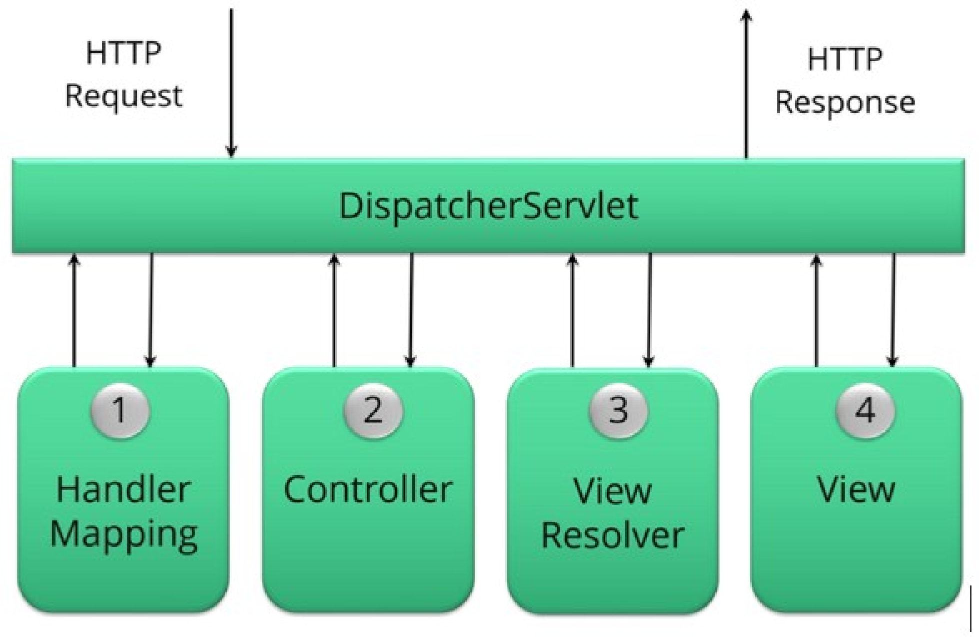
MVC (Model-View-Controller) – Spring Interview Questions

What do you mean by Spring MVC framework?

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

Describe DispatcherServlet.

The DispatcherServlet is the core of Spring Web MVC framework. It handles all the HTTP requests and responses. The DispatcherServlet receives the entry of handler mapping from the configuration file and forwards the request to the controller. The controller then returns an object of Model And View. The DispatcherServlet checks the entry of view resolver in the configuration file and calls the specified view component.

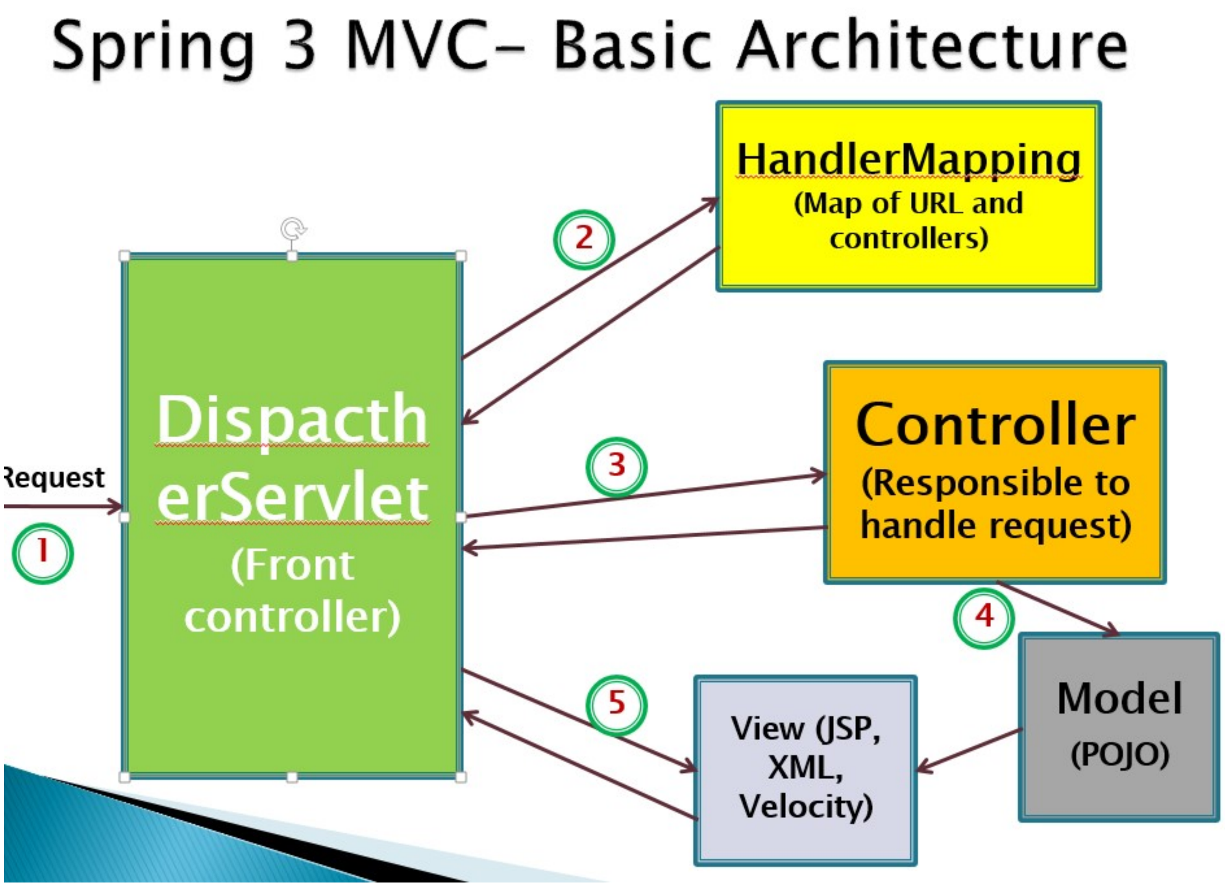


Explain WebApplicationContext.

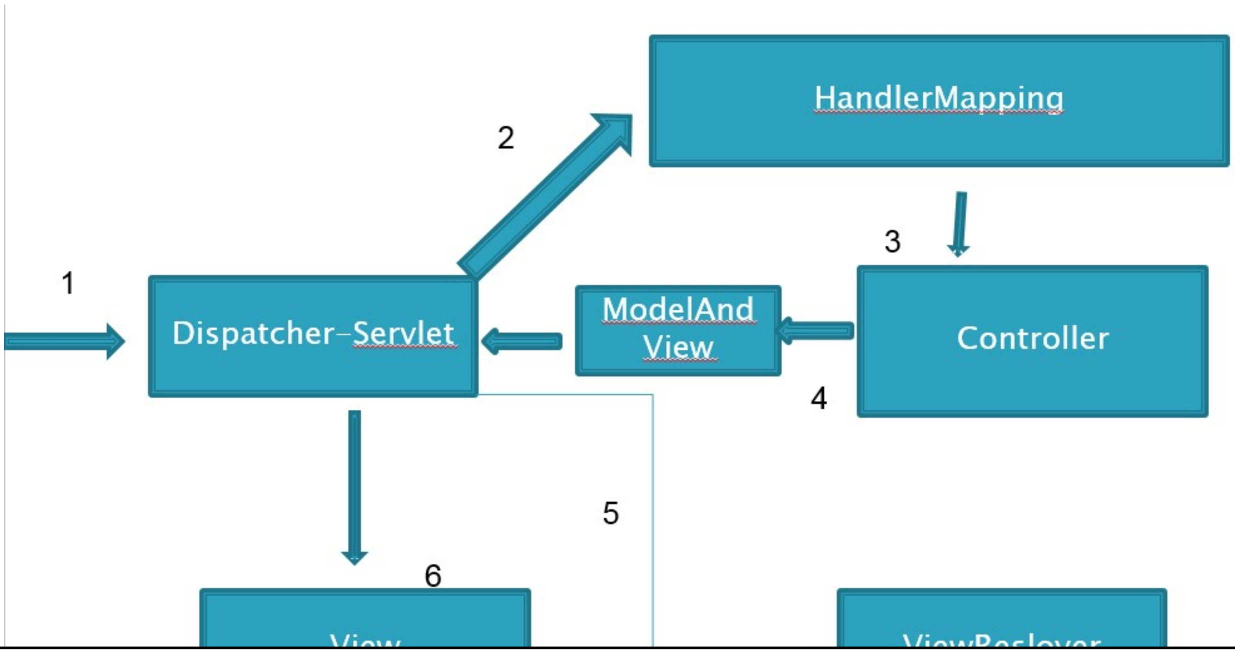
The WebApplicationContext is an extension of the plain ApplicationContext. It has some extra features that are necessary for web applications. It differs from a normal ApplicationContext in terms of its capability of resolving themes and in deciding which servlet it is associated with.

In Spring MVC framework, what is controller?

Controllers provide access to the application behavior. These behaviors are generally defined through a service interface. Controllers interpret the user input and transform it into a model which is represented to the user by the view. In Spring, controller is implemented in a very abstract way. It also enables you to create a wide variety of controllers.


Spring 3 MVC- Basic Architecture
HandlerMapping
(Map of URL and controllers)
DispactherServlet
(Front controller)
Request
1
2
3
4
5
Controller
(Responsible to handle request)
Model
(POJO)
View (JSP, XML, Velocity)

OR



Following is the Request process lifecycle of Spring MVC:

1)The client sends a request to web container in the form of http request.

2)This incoming request is intercepted by **Front controller**(DispatcherServle and it will then tries to find out appropriate **Handler Mappings**.

3)With the help of Handler Mappings, the DispatcherServlet will dispatch t request to appropriate Controller.

4)The Controller tries to process the request and returns the Model and Vi object in form of **ModelAndView** instance to the Front Controller.

5)The Front Controller then tries to resolve the View (which can be JSP, Freemarker, Velocity etc) by consulting the **View Resolver** object.

6)The selected view is then rendered back to client.

For Practical example refer to class notes

OUT OF SCOPE:

Spring AOP

Spring JDBC

Spring Boot

What is Spring Boot?
First of all Spring Boot is not a framework, it is a way to ease to create stand-alone application with minimal or zero configurations. It is approach to develop spring based application with very less configuration. It provides defaults for code and annotation configuration to quick start new spring projects within no time. It leverages existing spring projects as well as Third party projects to develop production ready applications. It provides a set of Starter Pom's or gradle build files which one can use to add required dependencies and also facilitate auto configuration.

Spring Boot automatically configures required classes depending on the libraries on its classpath. Suppose your application want to interact with DB, if there are Spring Data libraries on class path then it automatically sets up connection to DB along with the Data Source class.

What are the advantages of using Spring Boot?

- It is very easy to develop Spring Based applications with Java or Groovy.
- It reduces lots of development time and increases productivity.
- It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
- It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
- It follows "Opinionated Defaults Configuration" Approach to reduce Developer effort
- It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
- It provides CLI (Command Line Interface) tool to develop and test Spring Boot (Java or Groovy) Applications from command prompt very easily and quickly.
- It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle
- It provides lots of plugins to work with embedded and in-memory Databases very easily.

Spring Boot vs Spring MVC vs Spring - How do they compare?

Spring Framework

Most important feature of Spring Framework is Dependency Injection. At the core of all Spring Modules is Dependency Injection or IOC Inversion of Control.

When DI or IOC is used properly, we can develop loosely coupled applications. And loosely coupled applications can be easily unit tested.

Spring MVC

Spring MVC Framework provides decoupled way of developing web applications. With simple concepts like Dispatcher Servlet, ModelAndView and View Resolver, it makes it easy to develop web applications.

Spring Boot

The problem with Spring and Spring MVC is the amount of configuration that is needed.

```
<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
      <value>/WEB-INF/views/</value>
    </property>
    <property name="suffix">
      <value>.jsp</value>
    </property>
</bean>

<mvc:resources mapping="/webjars/**" location="/webjars/"/>
```

Spring Boot solves this problem through a combination of Auto Configuration and Starter Projects. Spring Boot also provide a few non functional features to make building production ready applications faster.

For complete answer with code examples refer - Spring Boot vs Spring vs Spring MVC

What is Auto Configuration?

The problem with Spring and Spring MVC is the amount of configuration that is needed.

```
<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
      <value>/WEB-INF/views/</value>
    </property>
    <property name="suffix">
      <value>.jsp</value>
    </property>
</bean>

<mvc:resources mapping="/webjars/**" location="/webjars/"/>
```

Can we bring more intelligence into this? When a spring mvc jar is added into an application, can we auto configure some beans automatically?

Spring Boot looks at a) Frameworks available on the CLASSPATH b) Existing configuration for the application. Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks. This is called Auto Configuration.

For complete answer with code examples refer Auto Configuration.

What are Spring Boot Starter Projects?

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy paste loads of dependency descriptors.

For example, if you want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.

Can you explain more about Starters with an example?

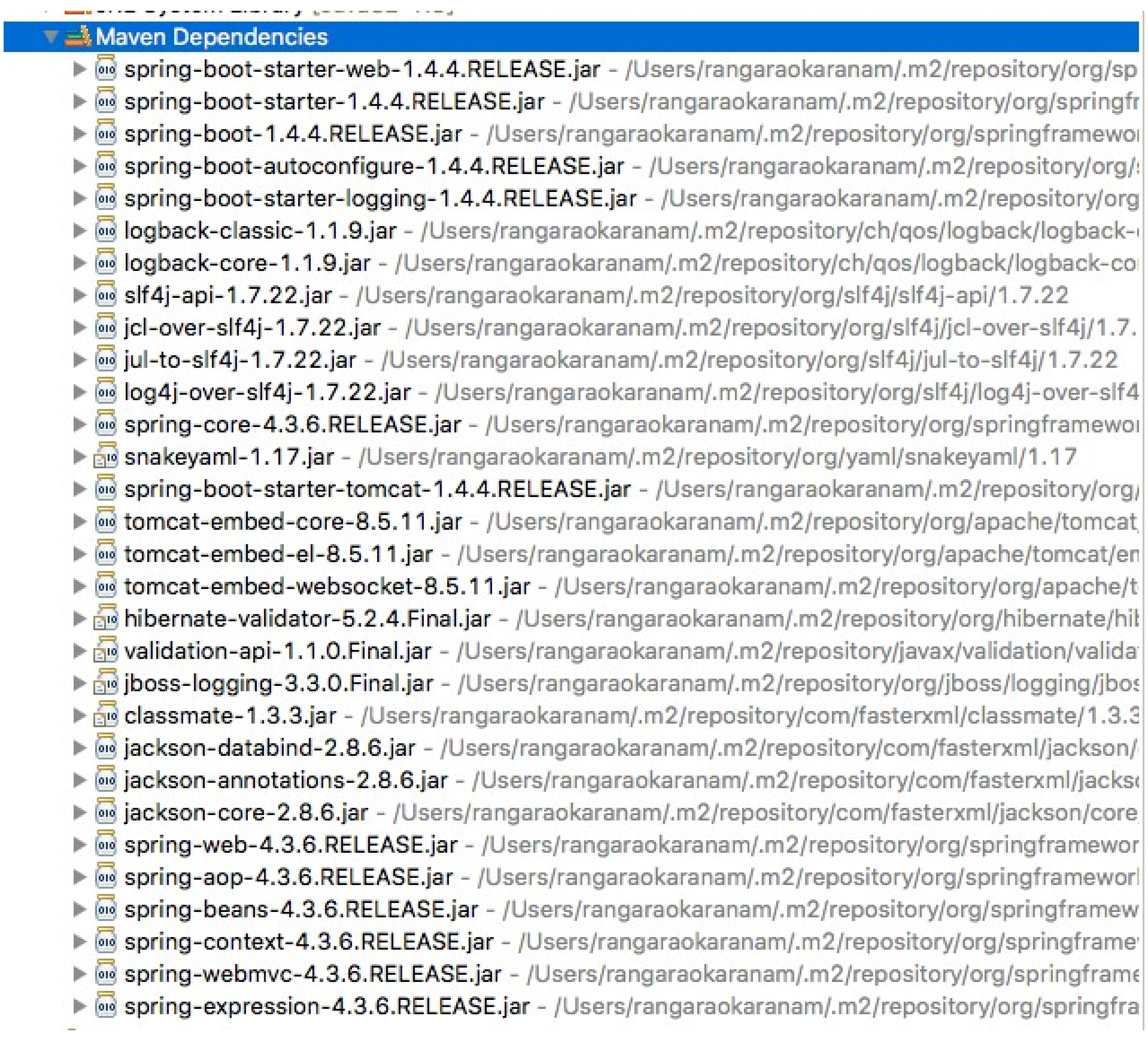
Let's consider an example starter - Spring Boot Starter Web.

If you want to develop a web application or an application to expose restful services, Spring Boot Start Web is the starter to pick. Lets create a quick project with Spring Boot Starter Web using Spring Initializr.

Dependency for Spring Boot Starter Web

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

Following screenshot shows the different dependencies that are added in to our application


```
▼ Maven Dependencies
  ▶ spring-boot-starter-web-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/sp
  ▶ spring-boot-starter-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springfr
  ▶ spring-boot-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewo
  ▶ spring-boot-autoconfigure-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/
  ▶ spring-boot-starter-logging-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org
  ▶ logback-classic-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-
  ▶ logback-core-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-co
  ▶ slf4j-api-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/slf4j-api/1.7.22
  ▶ jcl-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jcl-over-slf4j/1.7.
  ▶ jul-to-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jul-to-slf4j/1.7.22
  ▶ log4j-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/log4j-over-slf4
  ▶ spring-core-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewo
  ▶ snakeyaml-1.17.jar - /Users/rangaraokaranam/.m2/repository/org/yaml/snakeyaml/1.17
  ▶ spring-boot-starter-tomcat-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/
  ▶ tomcat-embed-core-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat
  ▶ tomcat-embed-el-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat/en
  ▶ tomcat-embed-websocket-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/t
  ▶ hibernate-validator-5.2.4.Final.jar - /Users/rangaraokaranam/.m2/repository/org/hibernate/hil
  ▶ validation-api-1.1.0.Final.jar - /Users/rangaraokaranam/.m2/repository/javax/validation/valida
  ▶ jboss-logging-3.3.0.Final.jar - /Users/rangaraokaranam/.m2/repository/org/jboss/logging/jbos
  ▶ classmate-1.3.3.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/classmate/1.3.3
  ▶ jackson-databind-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/
  ▶ jackson-annotations-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jacks
  ▶ jackson-core-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/core
  ▶ spring-web-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewor
  ▶ spring-aop-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewor
  ▶ spring-beans-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframew
  ▶ spring-context-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframe
  ▶ spring-webmvc-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframe
  ▶ spring-expression-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springfra
```

Dependencies can be classified into:

- Spring - core, beans, context, aop
- Web MVC - (Spring MVC)
- Jackson - for JSON Binding
- Validation - Hibernate Validator, Validation API
- Embedded Servlet Container - Tomcat
- Logging - logback, slf4j

Any typical web application would use all these dependencies. Spring Boot Starter Web comes pre packaged with these.

As a developer, I would not need to worry about either these dependencies or their compatible versions.

What are the other Starter Project Options that Spring Boot provides?

Spring Boot also provides other starter projects including the typical dependencies to develop specific type of applications

- spring-boot-starter-web-services - SOAP Web Services
- spring-boot-starter-web - Web & RESTful applications
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-hateoas - Add HATEOAS features to your services
- spring-boot-starter-security - Authentication and Authorization using Spring Security
- spring-boot-starter-data-jpa - Spring Data JPA with Hibernate
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data REST

How does Spring enable creating production ready applications in quick time?

Spring Boot aims to enable production ready applications in quick time. Spring Boot provides a few non functional features out of the box like caching, logging, monitoring and embedded servers.

- spring-boot-starter-actuator - To use advanced features like monitoring & tracing to your application out of the box
- spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat - To pick your specific choice of Embedded Servlet Container
- spring-boot-starter-logging - For Logging using logback
- spring-boot-starter-cache - Enabling Spring Framework's caching support

What is the minimum baseline Java Version for Spring Boot 2 and Spring 5?

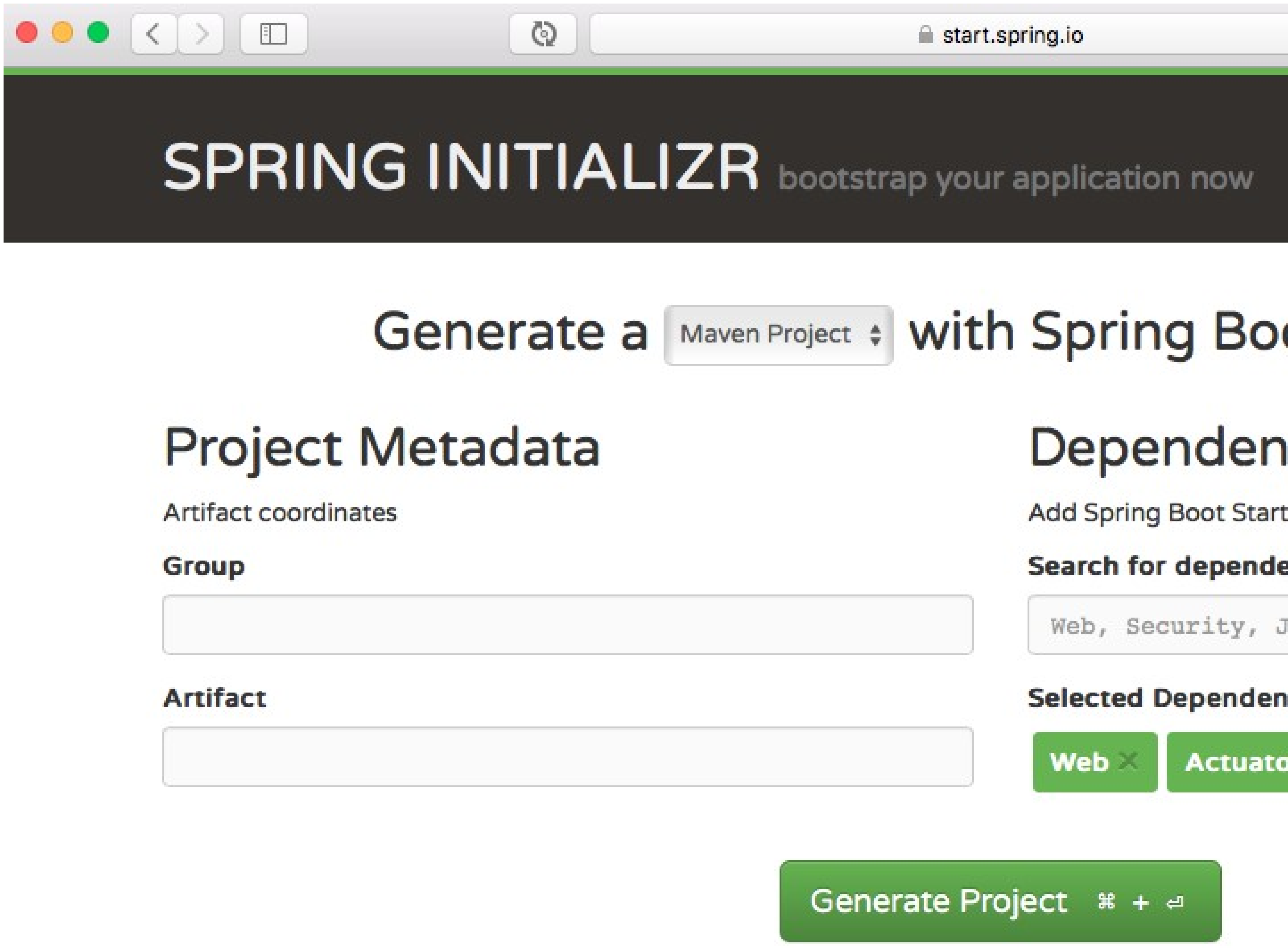
Spring Boot 2.0 requires Java 8 or later. Java 6 and 7 are no longer supported.

Recommended Reading

- https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0.0-M1-Release-Notes

What is the easiest approach to create a Spring Boot Project?

Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.



As shown in the image above, following steps have to be done

- Launch Spring Initializr and choose the following
  - Choose com.in28minutes.springboot as Group
  - Choose student-services as Artifact
  - Choose following dependencies
    - Web
    - Actuator
    - DevTools
- Click Generate Project.
- Import the project into Eclipse. File -> Import -> Existing Maven Project.

Is Spring Initializr the only way to create Spring Boot Projects?

No.

Spring Initializr makes it easy to create Spring Boot Projects. But you can setup a maven project and add the right dependencies to start off.

In our Spring course, we use 2 approaches to create projects.

- The first one is start.spring.io.
- The other one - setting up a project manually is used in the Section titled - “Basic Web Application”

Setting up a maven project manually

Here are the important steps:

- In Eclipse, Use File -> New Maven Project to create a new project.
- Add dependencies.
- Add the maven plugins!
- Add the Spring Boot Application class

Why do we need spring-boot-maven-plugin?

spring-boot-maven-plugin provides a few commands which enable you to package the code as a jar or run the application

- spring-boot:run runs your Spring Boot application.
- spring-boot:repackage repackages your jar/war to be executable.
- spring-boot:start and spring-boot:stop to manage the lifecycle of your Spring Boot application (i.e. for integration tests).
- spring-boot:build-info generates build information that can be used by the Actuator.

How can I enable auto reload of my application with Spring Boot?

Use Spring Boot Developer Tools.

Adding Spring Boot Developer Tools to your project is very simple.

Add this dependency to your Spring Boot Project pom.xml

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-devtools</artifactId>
   <scope>runtime</scope>
</dependency>
```

Restart the application.

What and Why Embedded Servers?

Think about what you would need to be able to deploy your application (typically) on a virtual machine.

- Step 1 : Install Java
- Step 2 : Install the Web/Application Server (Tomcat/Websphere/Weblogic etc)
- Step 3 : Deploy the application war

What if we want to simplify this?

How about making the server a part of the application?

You would just need a virtual machine with Java installed and you would be able to directly deploy the application on the virtual machine.

This idea is the genesis for Embedded Servers.

When we create an application deployable, we would embed the server (for example, tomcat) inside the deployable.

For example, for a Spring Boot Application, you can generate an application jar which contains Embedded Tomcat. You can run a web application as a normal Java application!

Embedded server is when our deployable unit contains the binaries for the server (example, tomcat.jar).