**What is Spring?**

It is a lightweight, loosely coupled and integrated framework for developing enterprise applications in Java.

**DI**: 3rd party takes the responsibility of creating objects and injecting to the actual classes.

**IOC:** Creation of Object will happen in reverse order.

Ex: Class A has instance of Class B. Class B has an instance of Class C.

**In IOC:** C is created first and then B object. C instance is injected into B. In the same way happens in Class A and instance of B.

IOC is a design Pattern. Removes object creation dependency from the programming code.

Loosely coupled code.

It reduces the code. It avoids boiler plate code.

Maintaining the code will be easy.

Spring manages the life cycle of object using BeanFactory.

**Role of IOC?**

* Create the instance
* Configure the instance
* Assemble the dependencies

Benefits:

* Testing is easy. We can mock the Heavy resources (3rd Party calls, rest calls, DB calls) if required.
* Development fast
* Resources (memory etc.,) is well organized.

**Spring Container:** In any major applications, we will have 2 servers. One is Web Server(Apache) and another is Application Server (JBOSS etc.,). But if you use Spring, it has its own container, which will manage pooling and etc., Here we no need of external Container like JBOSS etc.,

What are the benefits of Spring Framework?

* **Lightweight:** Spring is lightweight when it comes to size and transparency. (Each has separate module. If you want Spring data module you can add etc.)
* **Inversion of Control:** Loose coupling is achieved in Spring, with the Inversion of Control Technique. The objects give their dependencies instead of creating or looking for dependent objects.
* **Aspect oriented:** Spring supports Aspect oriented programming and separates application business logic from system services.
* **Container:** Spring contains and manages the life cycle and configuration of Application objects.
* **MVC Framework:**
* **Transaction Management:**
* **Exception Handling:**

**Bean Definition:**

* Class
* Name
* Scope
* Constructor-arg
* Properties
* Autowiring-mode
* Lazy-initialization mode

**Bean scopes:**

|  |  |
| --- | --- |
| Scope | Description |
| Singleton | This scope the bean definition to a single instance per Spring IoC container (default). |
| Prototype | This scope a single bean definition to have any number of object instances |
| Request | This scope a bean definition to an HTTP request. |
| Session | This scope a bean definition to HTTP session. |
| Global-session | This scope a bean definition to a global HTTP session. |

**Bean Life cycle:**

To define setup and teardown for a bean, we simply declare the <bean> with init-method and /or destroy-method parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, destroy-method specifies a method that is just called before a bean is removed from the container.

**What is bean wiring?**

The process of injecting spring bean dependencies while initializing it called Spring bean wiring.

**What is autowiring in Spring?**

Autowire enables the programmer to inject the bean automatically. We don’t need to write explicit injection logic.

**@Autowired:** By default it uses **byType.**

@Autowired

**private** MasterPublicationsDao masterPublicationsDao;

@Repository("masterPublicationsDao")

**public** **class** MasterPublicationsDaoImpl **implements** MasterPublicationsDao {

If there are 2 classes implementing MasterPublicationsDao, then Spring will be into ambiguous, then it doesn’t know which one to be Autowired.

In these scenarios use **@Qualifier** with class name.

|  |  |  |
| --- | --- | --- |
| No. | Mode | Description |
| 1 | No | This is default mode, it means, autowiring is not enabled |
| 2 | byname | Injects the bean based on the property name. It uses setter method. |
| 3 | byType | Injects the bean based on the property type. It uses setter method. |
| 4 | Constructor | It injects the bean using constructor. |

**Servlet:**

The Java component that accepts the web request is called as Servlet.

**Difference between ApplicationContext.xml and Spring-dispatcher Servlet:**

ApplicationContext.xml defines the beans that are shared among all the servlets. If your application is having more than one servlet, then defining the common resources in the applicationContext.xml would make sense.

Spring-servlet.xml defines the beans that are related to only that servlet. Here it is the dispatcher servlet.

**How to configure multiple view resolvers?**

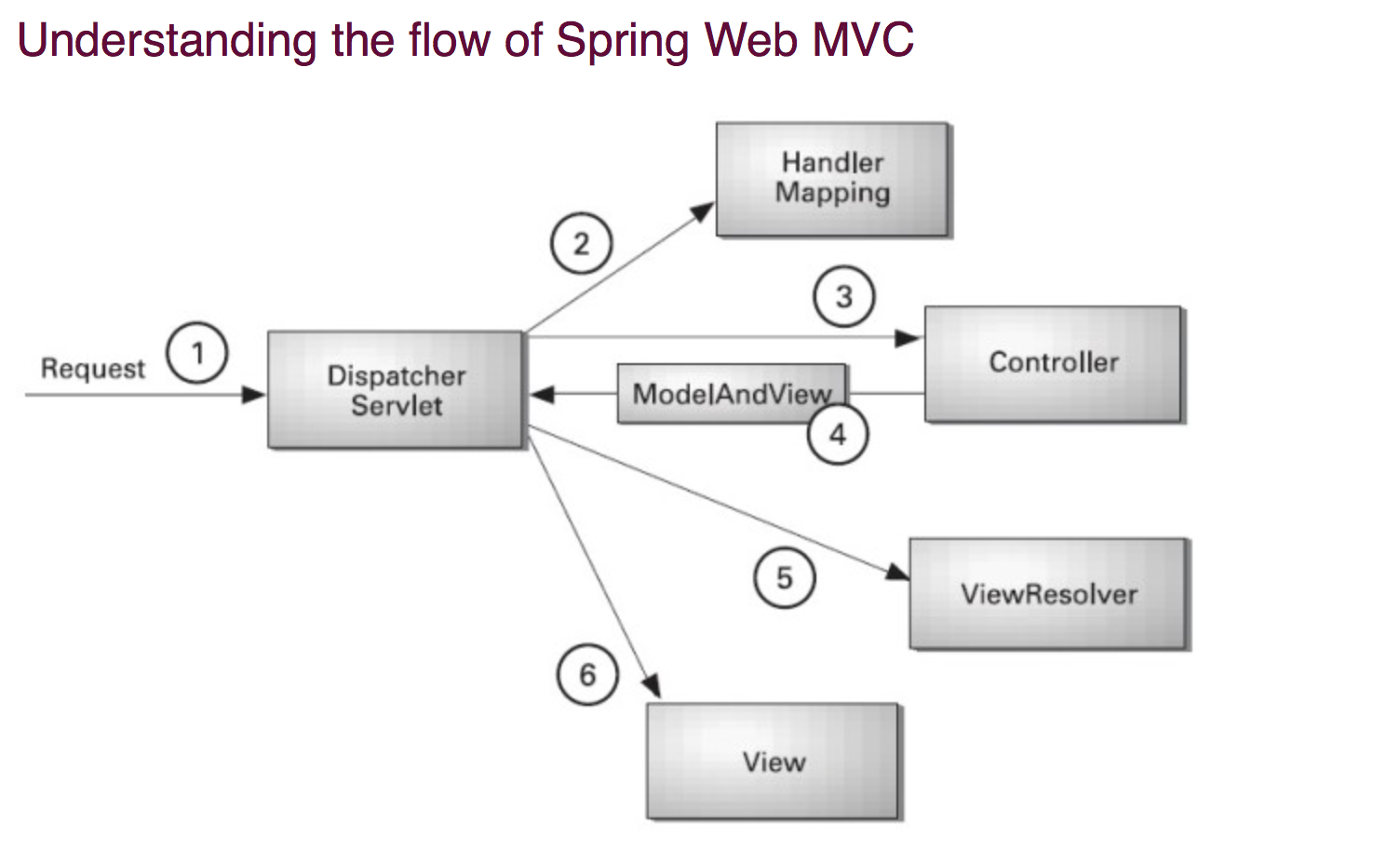
If multiple view resolver strategies are applied, you have to declare the priority through “**order**” property, where the **lower order value has a higher priority**, for example

|  |
| --- |
| <beans ...>  <bean class="org.springframework.web.servlet.view.XmlViewResolver">  <property name="location">  <value>/WEB-INF/spring-views.xml</value>  </property>  <property name="order" value="0" />  </bean>  <bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">  <property name="basename" value="spring-views" />  <property name="order" value="1" />  </bean>    <bean id="viewResolver"  class="org.springframework.web.servlet.view.InternalResourceViewResolver" >  <property name="prefix">  <value>/WEB-INF/pages/</value>  </property>  <property name="suffix">  <value>.jsp</value>  </property>  <property name="order" value="2" />  </bean>  </beans> |

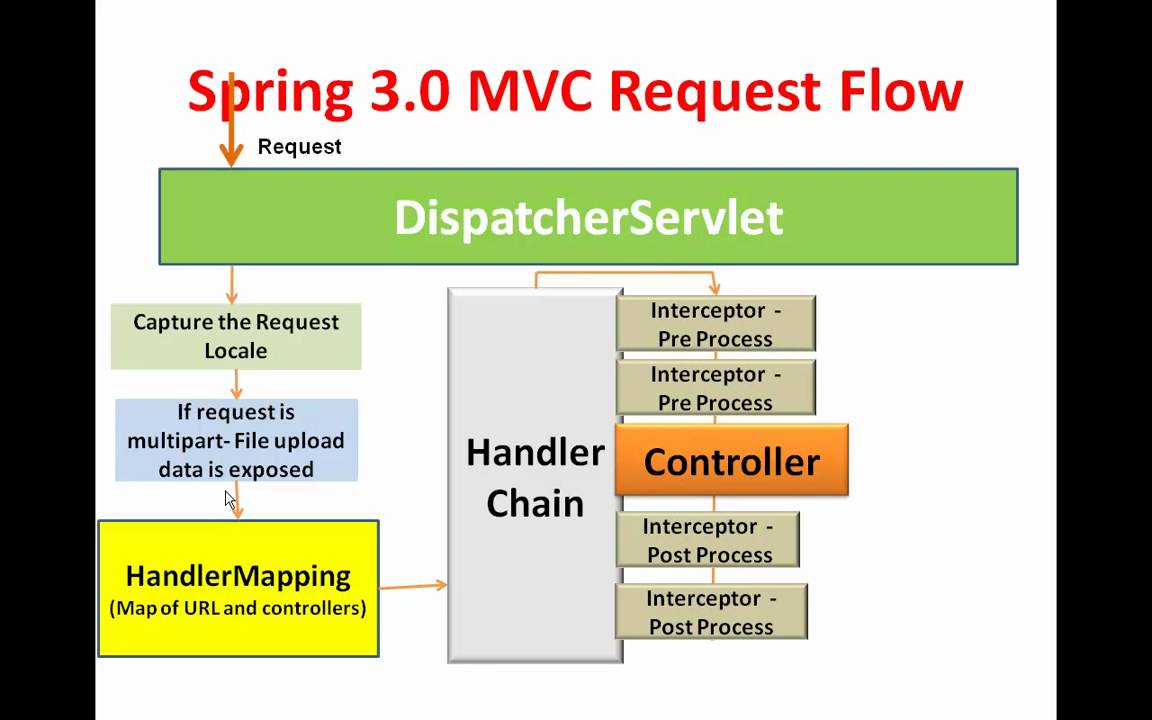
Now, if a view name is returned, the view resolving strategy works in the following order :

XmlViewResolver --> ResourceBundleViewResolver --> InternalResourceViewResolver

**Spring MVC architecture:**

****

As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller. The DispatcherServlet gets entry of handler mapping from the xml file and forwards the request to the controller. The controller returns an object of ModelAndView. The DispatcherServlet checks the entry of view resolver in the xml file and invokes the specified view component.

****

**Spring - 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.

You can define a parent bean definition as a template and other child beans can inherit required configuration from the parent bean.

When you use XML-based configuration metadata, you indicate a child bean definition by using the **parent** attribute, specifying the parent bean as the values of the attribute.

|  |
| --- |
| <?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="helloWorld" class="com.tutorialspoint.HelloWorld">  <property name="message1" value="Hello World!"/>  <property name="message2" value="Hello Second World!"/>  </bean>  <bean id="helloIndia" class="com.tutorialspoint.HelloIndia" parent="helloWorld">  <property name="message1" value="Hello India!"/>  <property name="message3" value="Namaste India!"/>  </bean>  </beans> |

**Spring @Component, @Repository, @Service and @Controller:**

**What is the difference between BeanFactory and ApplicationContext?**

BeanFactory is a basic container whereas ApplicationContext is the advanced container. ApplicationContext extends the BeanFactory interface. ApplicationContext provides more facitilies than BeanFactory such as integration with AOP, message resource handling etc .,

2 ways to create ApplicationContext:

* ClassPathXmlApplicationContext
* FileSystemXmlApplicationContext

Lazy loading

**BeanFactory:** Itis a Factory pattern which is based on IOC design principles. It is used to make a clear separation between application configuration and dependency from actual code.

Eager initializing.

It is used to create the beans.

3 ways to create BeanFactory:

* ClassPathResource
* FileSystemResource
* ClassPathXmlApplicationContext

**Difference between constructor injection and setter injection?**

|  |  |  |
| --- | --- | --- |
| No. | Constructor Injection | Setter Injection |
| 1 | No Partial Injection | Partial Injection |
| 2 | Doesn’t override setter | Overrides the constructor property if both are defined |
| 3 | Create new instance if any modification occurs | Doesn’t create any new instance if you change the property value. |
| 4 | Better for too many properties | Better for few properties. |

**What are the Transaction management supports providing by Spring?**

1. **Programmatic Transaction Management:** Should be used for few transactions operations.
2. **Declarative Transaction Management:** Should be used for many transactions operations.

**What are the advantages of JdbcTemplate in spring?**

**Less code:** By using JdbcTemplate class, you don’t need to create connection, start transaction, commit transaction and close connection to execute different queries. You can execute the query directly.

It will also convert SQL exceptions into User readable.

**What are classes for spring JDBC API?**

* JdbcTemplate
* SimpleJdbcTemplate
* NamedParameterJdbcTemplate
* SimpleJdbcInsert
* SimpleJdbcCall

**How can you fetch records by spring JdbcTemplate?**

* ResultSetExtractor
* RowMapper

**What is the advantage of NamedParameterJdbcTemplate?**

NamedParameterJdbcTemplate class is used to pass value to the named parameter. A named parameter is better than ? (question mark in prepared

statement)

**What is the advantage of SimpleJdbcTemplate?**

It supports var-args and autoboxing.

**What is the front controller of Spring MVC?**

DispatcherServlet class works as the front controller in the Spring MVC.

**What does @Controller annotation?**

It marks the class as controller class. It is applied on the class.

**What does @RequestMapping annotation?**

It maps request with the method. It is applied on method.

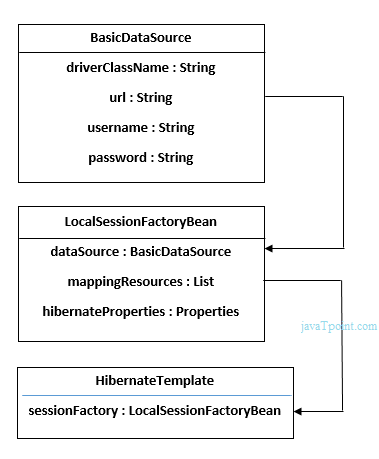
**What is ViewResolver class?**

The view resolver class invokes the view component to be invoked for the request. It defines prefix and suffix properties to resolve the view component.

**Which ViewResolver class is widely used?**

The org.springframework.web.servelt.view.InternalResourceViewResolver class is widely used.

**Spring Hibernate:**



**If use annotations in Spring-Hibernate:**

1. DriverManagerDataSource
2. AnnotationSessionFactoryBean- sessionFactory
3. TransactionManager

**Q) If an applicationContextXML has 2 bean definitions for same class, when getting the bean with byType: how you will identify?**

Primary=”true”

Autowire-candidate=”false”.

These 2 will help in getting unique result. Else it will through error saying “found 2 beans”.