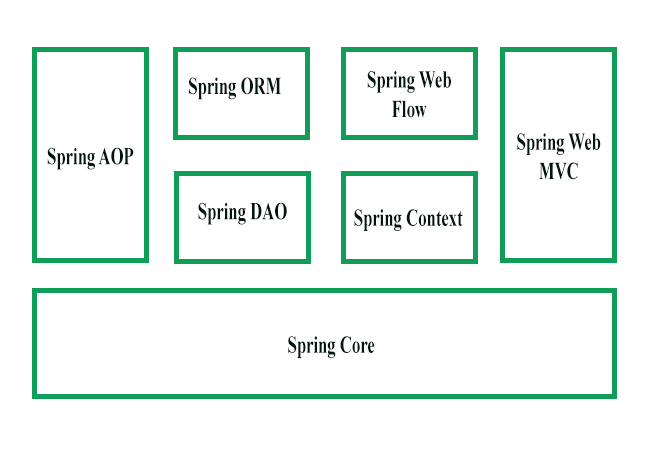
Spring Core

* What is spring framework. Why it is required.
* Spring framework modules
* 
* Software setup and installation
* **Inversion of control container:**
  + Creating Object for us,
  + Managing our objects,
  + Helping our application to be configurable,
  + Managing dependencies

Spring Container

Application

Meta data in XML form

Java POJO classes

* **Type of IOC container**: BeanFactory and ApplicationContext
  + What is a bean
  + What is bean factory
  + What is application context

Example of BeanFactory:

*Resource resource = new ClassPathResource(“application.xml”);*

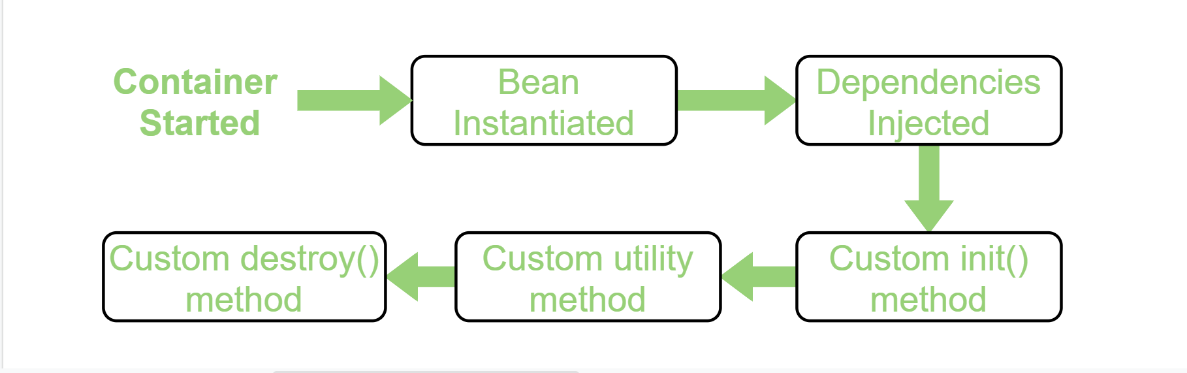
*BeanFactory factory = new XmlBeanFactory(Resource object);*

*Student student = (Student)factory.getBean(“studentId”);*

To create a bean in xml file:

*<bean id=”studentId” class=”Student”></bean>*

* **Dependency Injection**
  + **Setter injection**
    - In this, the DI will be injected with the help of setter and/or getter methods.
    - We use <property> tag in bean config file.
    - *<property name=”variableName”><value>Value</value></property>*
    - We can inject :
      * Primitive and string based value
      * Dependent object
      * Collection values
  + **Constructor injection**
    - We can inject the dependency by constructor. The **<constructor-arg>** subelement of **<bean>** is used for constructor injection.
    - Example:
      * *<constructor-arg value="10" type="int"></constructor-arg>*
* **Beans Life Cycle**
  + The bean life cycle refers to when & how the bean is instantiated, what action it performs until it lives, and when & how it is destroyed.



* + **Singleton scope**
  + **Prototype scope**
* **Autowiring**
  + Enables to inject the object dependency implicitly. It internally uses setter or constructor injection.
  + Can't be used to inject primitive and string values. It works with reference only.

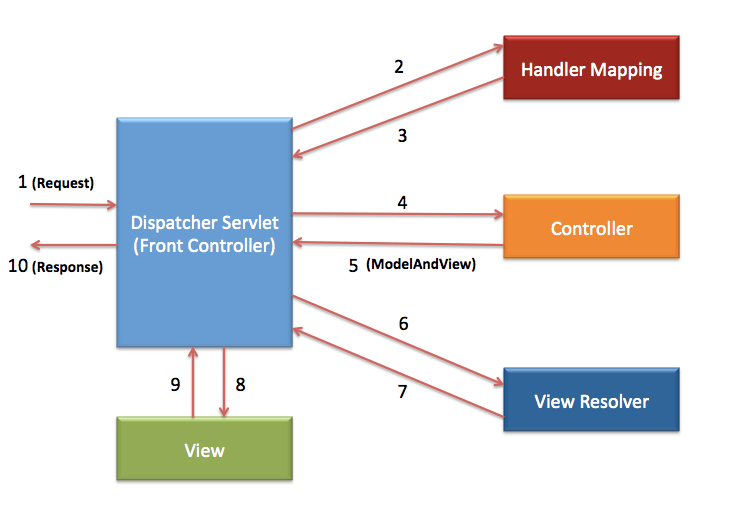
|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object dependency according to name of the bean. |
| 3) | byType | The byType mode injects the object dependency according to type. So property name and bean name can be different. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters. |

**Example:** *<bean id="studentCons" class="Student" autowire="byName"></bean>*

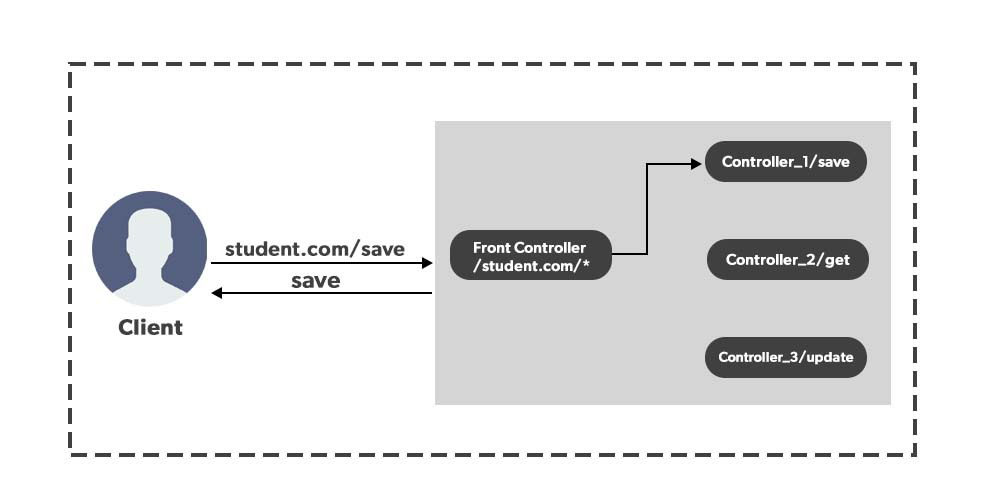
Spring MVC

* A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection**.**
  + Model**:** contains the data of the application.
  + View:generates the model data and also generates the HTML output.
  + Controller: contains the business logic of the application.

**Work flow of spring MVC**

.

**What is dispatcher servlet**



*DispatcherServlet handles an incoming HttpRequest, delegates the request, and processes that request according to the configured HandlerAdapter interfaces that have been implemented within the Spring application along with accompanying annotations specifying handlers, controller endpoints, and response objects.*

**Controllers:** A controller is responsible for handling user requests and coordinating the flow of information within a web application, ensuring that the right actions are taken and the appropriate responses are given.

**Different types of controller are @Controller and @RestController.**

When a request is made to the application, the **@Controller** class receives the request, processes it, and returns a view (HTML page) to be displayed in the user's web browser.

When a request is made to the application, the **@RestController** class handles the request, processes data, and returns the data directly in the response, which is usually in a structured format like JSON.

In Spring Web MVC, **view** resolution and rendering play a crucial role in processing user requests and generating the appropriate response to be sent back to the client (usually a web browser). The process involves converting the model data into a view (HTML page) and then rendering it as the final response.

**ModelAndView** is a class in Spring Web MVC that represents a model and view combination, used to carry data from a controller to a view for rendering in a web application.

**@RequestMapping**

It defines which URL paths should be handled by which methods in the controller. When a request is made to the application with a specific URL path, Spring will route the request to the appropriate controller method based on the **@RequestMapping** configuration.

It can be applied at both class level and method level.

*@RequestMapping(“/api”);*

Additionally, the **@RequestMapping** annotation can also specify the HTTP method (GET, POST, PUT, DELETE, etc.) using the **method** attribute to restrict the type of requests that the method will handle.

**Steps of creating a spring MVC project:**

* Create a maven project and add necessary “spring web” dependencies.
* Create the controller class
* Provide the entry of controller in the web.xml file
* Define the bean in the separate XML file
* Display the message in the JSP page
* Start the server (Tomcat server)