Dependency Injection

In Spring, objects define their dependencies and do not worry about how they will get those dependencies. It is the responsibility of Spring to provide the required dependencies for creating objects.

Types of dependency injection, setter and constructor

Inversion of Control is a principle by which the control of objects or portions of a program is transferred to a container or framework.

IoC enables a framework to take control of the flow of a program and make calls to our custom code.

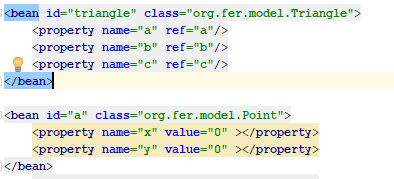
Application Context

Bean Factory (instantiate, keeps track and manage the life cycle of the beans) with additional functionality like event notification and AOP





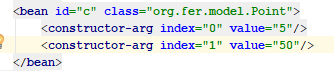
Define a bean in xml file for spring to instantiate it



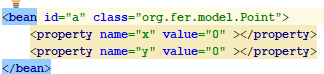
Property tag allows to set a value for the property or use another bean as value with the ref attribute

Constructor injection

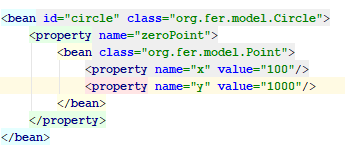
Will make the injection through the constructor instead than the setter. Can use index to specify the position of each arg. Can also specify type in case each arg would be of a different type.



Setter injection, will use the property tag and will set the value through the setter



Inner bean, will be available only inside another bean



Alias

Allows to create an alias for a bean and reference it inside config file or even get it from the spring context. Ex b existing bean, secondPoint its alias



Idref, if you want to restrict a bean to use only id, not aliases or names

Initializing collections on the bean



Autowiring

byName, if the properties of the bean match then they are linked together

byType, only works if there are only one type of the autowired bean

constructor

default

no (default autowiring off)



Spring Scopes

Singleton, will return always the same instance (default). At container initialization these are instantiated

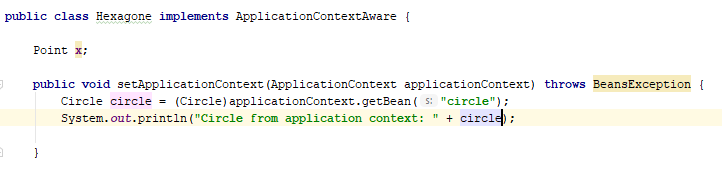
Prototype, will return a new instance of the class on each get bean

Request, session, globalsession also available (web aware context bean scopes)



Application Context Aware Interface

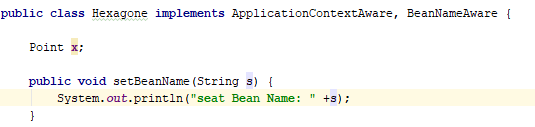
Classes implementing this interface will have access to the application context which will be injected by Spring.



Bean Name Aware Interface

Allows to get the name of the bean in the class.

Ex. Bean declared in spring.xml as hexagone of class Hexagone, in the class Hexagone will have the method setBeanName which will receive the name of the bean (hexagone).

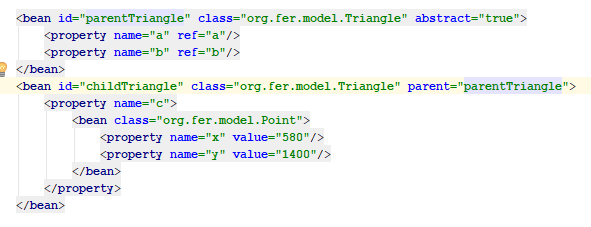


Bean Definition Inheritance

Allows beans to inherit from another bean and share its properties using the attribute parent and the parent’s bean name

Abstract

If set to true, allows a bean to not being instantiated, will just be use as a template for other beans



Inheritance with collections

If a bean inherits from another and they have a collection, then the child would override the values of the parent if the value is specified, or you can use the option merge which will merge the values from parent and child



Initializing Bean Interface

It gives a method afterPropertiesSet which will execute after the bean has bean instantiated

Disposable Bean Interface

Offers a method destroy() which will be executed before the bean gets destroyed.

AbstractApplicationContext

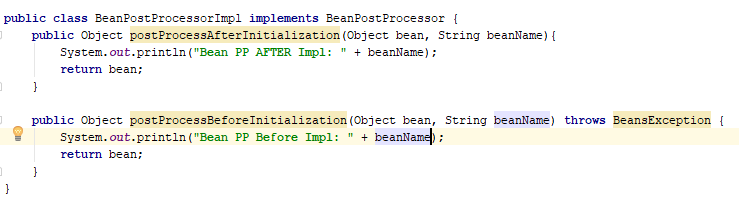
Application context which allows us to use the method registerShutdownHook to identify when the program ends and the beans will be destroyed.

Init-method, destroy-method

Allows the same functionality without implementing the interfaces, declare on the bean the init and destroy method for the bean, If want it for all the beans you can declare it on the beans tag with the attribute default-init-method and default-destroy-method

Bean Post Processor

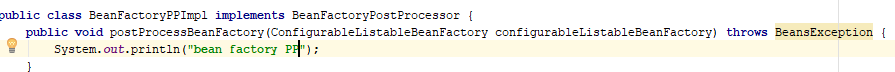
It executes before and or after any bean has been initialized (for all beans) (bean needs to be added to spring xml





Bean Factory Post Processor interface

Has a method which will run after the bean factory itself has been instantiated



@Required

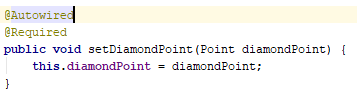
Annotation which force a dependency to be fulfill in order to initialize

(needs to add <context:component-scan base-package="org.fer"/> for it to work)

@Autowired

Spring helps to autowire dependencies, first for type then for name

Add the annotation to the setter

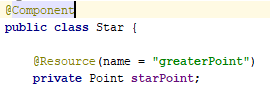


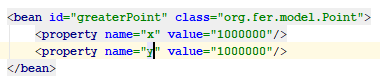
@Qualified

You can add it and a qualifier tag on spring xml for it to be use as bean name

@Resource

Allows to inject a value into a variable



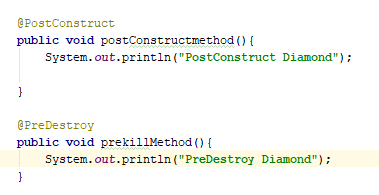


@PostConstruct

You can annotate a method with this and will be executed automatically after it has ben created

@PreDestroy

You can annotate a method with this and it will be executed automatically before the bean is destroyed.



Stereotypes

@Component, allows to define a class as a bean so its not necessary to add it to the spring.xml

@Service

@Controller

@Repository

AOP

Model of programming that helps with cross cutting concerns

Do not make reference to it on the classes and can be applied before or after the method

TARGET METHOD

ASPECT

ASPECT

Aspect

Class with annotations

Advice

Methods inside the aspect which will run

@Before

This will be the moment where the aspect will be applied, before the method with the signature sent is executed this will execute first.

Pointcut

Says where you want the advice to apply

@Before

@After

@AfterReturning

@AfterThrowing

@Around

Questions

What is spring

Java EE framework which core concepts are dependency injection and aop, to achieve loose coupling between different components by implementing dependency injection

We can implement DI in Spring with Spring xml based or annotations

Aspect: Aspect is a class that implements cross-cutting concerns, such as transaction management. Aspects can be a normal class configured and then configured in Spring Bean configuration file or we can use Spring AspectJ support to declare a class as Aspect using @Aspect annotation.

Advice: Advice is the action taken for a particular join point. In terms of programming, they are methods that gets executed when a specific join point with matching pointcut is reached in the application. You can think of Advices as Spring interceptors or Servlet Filters.

Pointcut: Pointcut are regular expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points. Spring framework uses the AspectJ pointcut expression language for determining the join points where advice methods will be applied.

Join Point: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

Advice Arguments: We can pass arguments in the advice methods. We can use args() expression in the pointcut to be applied to any method that matches the argument pattern. If we use this, then we need to use the same name in the advice method from where argument type is determined.

Any normal java class that is initialized by Spring IoC container is called Spring Bean.

Spring bean life cycle

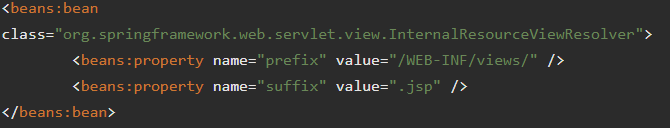
Spring Beans are initialized by Spring Container and all the dependencies are also injected. When context is destroyed, it also destroys all the initialized beans.

Get ServletContext and Servlet Config in a Spring Bean

1 Implement ServletContextAware and ServletConfigAware interfaces

2 @Autowired to a bean variable of type Servlet Context and Servlet Config (only in servlet container environment)

ViewResolver implementations are used to resolve the view pages by name. Usually we configure it in the spring bean configuration file



So if a controller handler method returns “home”, view resolver will use view page located at /WEB-INF/views/home.jsp.

Handle Exceptions on Spring MVC

@ExceptionHandler

@ControllerAdvice annotation that we can use with any class to define our global exception handler.

@Controller – for controller classes in Spring MVC project.

@RequestMapping – for configuring URI mapping in controller handler methods. This is a very important annotation, so you should go through Spring MVC RequestMapping Annotation Examples

@ResponseBody – for sending Object as response, usually for sending XML or JSON data as response.

@PathVariable – for mapping dynamic values from the URI to handler method arguments.

@Autowired – for autowiring dependencies in spring beans.

@Qualifier – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.

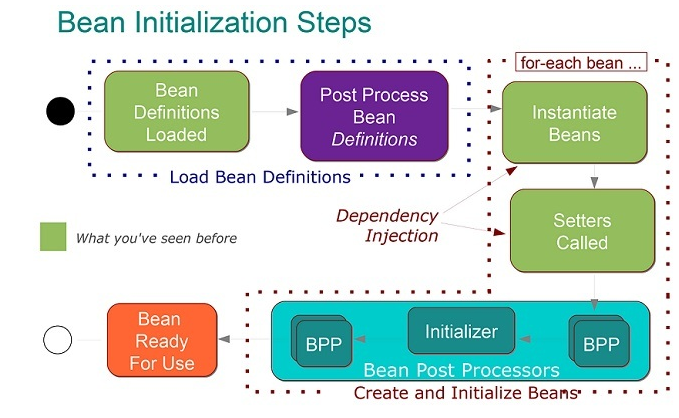
@Service – for service classes.

@Scope – for configuring scope of the spring bean.

@Configuration, @ComponentScan and @Bean – for java based configurations.

AspectJ annotations for configuring aspects and advices, @Aspect, @Before, @After, @Around, @Pointcut etc.

The recommended approach is to use constructor arguments for mandatory dependencies and setters for optional ones. Constructor injection allows injecting values to immutable fields and makes testing easier.



Spring Boot is a project that provides a pre-configured set of frameworks to reduce boilerplate configuration so that you can have a Spring application up and running with the smallest amount of code.

The Spring Web MVC framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.

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

Constructor Injection

Setter Injection

Interface Injection

ViewResolver enables a web application to select its view (such as JSP) dynamically.

(i) The browser sends a request to DispatcherServlet

(ii) DispatcherServlet knows the HanderMapping and can find the appropriate controllers

(iii) Controllers execute the request and put the data in the model and return back the view name to the DispatcherServlet.

(iv) DispatcherServlet uses the view name and ViewResolver to map to the view.

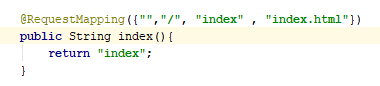
@Controller vs @RestController

with the @RestController you get the @ResponseBody annotation automatically, which means you don't need to separately annotate your handler methods with the @ResponseBody annotation.

The @EnableWebMvc annotation is required to enable Spring MVC when Java configuration is used to configure Spring MVC instead of XML



Spring MVC

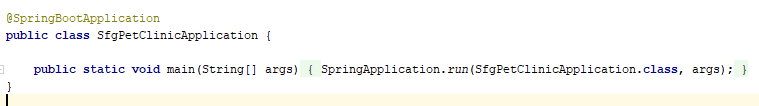


@RequestMapping

is used to map web requests to Spring Controller methods.

The HTTP method parameter has no default – so if we don’t specify a value, it’s going to map to any HTTP request.

Entry point of SpringBoot Spring MVC app



Spring MVC Execution Flow

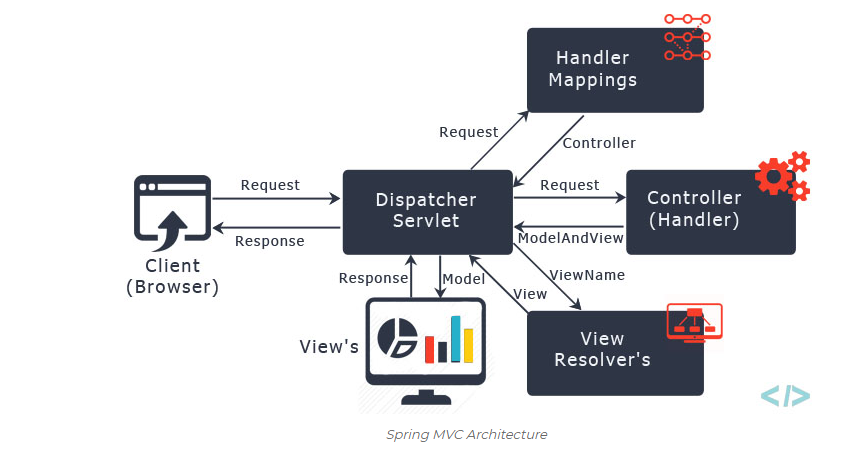
Step 1: First request will be received by DispatcherServlet.

Step 2: DispatcherServlet will take the help of HandlerMapping and get to know the Controller class name associated with the given request.

Step 3: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModelAndView object (contains Model data and View name) back to the DispatcherServlet.

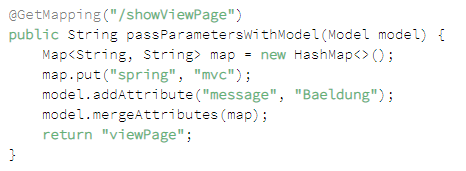
Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page.

Step 5: Finally DispatcherServlet will pass the Model object to the View page to display the result.



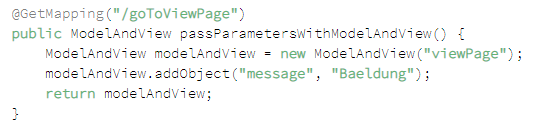
Model Object

To provide a view with usable data, we simply add this data to its Model object.



ModelAndView

Set the view and add data to the model to use on the web page.



@GetMapping

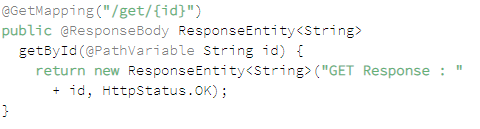
@PostMapping

@PutMapping

@DeleteMapping

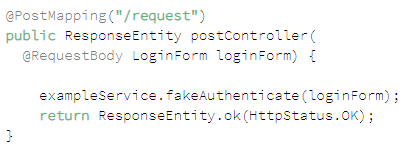
@PatchMapping

From the naming convention we can see that each annotation is meant to handle respective incoming request method type, i.e. @GetMapping is used to handle GET type of request method

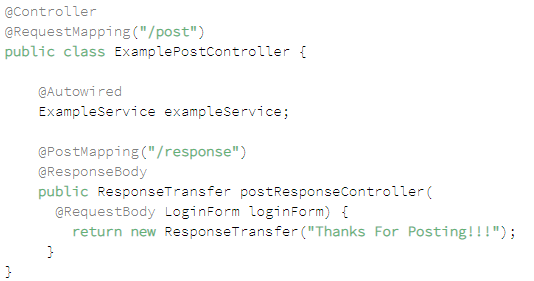


@RequestBody annotation maps the HttpRequest body to a transfer or domain object, enabling automatic deserialization of the inbound HttpRequest body onto a Java object.

Spring automatically deserializes the JSON into a Java type assuming an appropriate one is specified

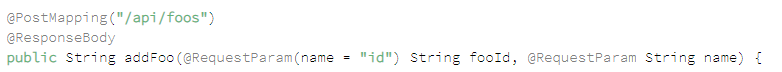


The @ResponseBody annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object.



@RequestParam to extract query parameters, form parameters and even files from the request.





@PathVariables extract values from the URI path

