IOC Container

* Find Beans
* Wire Dependencies
* Manage Lifecycle of the Bean

Application Context

* Bean Factory++
  + Spring’s AOP features
  + I18n capabilities
  + WebApplicationContext for web applications

Dependency injection: We use Spring Framework to instantiate beans and wire dependencies. Spring injects dependencies with @Component, @Autowired. Springs finds the beans and autowired.

Inversion of Control: Spring finds the beans and wire them, then injects the dependencies in the classes.

@Component: Spring should manage the bean. Generic component.

@Autowired: Spring should find the matching bean and wire the dependency in.

@Repository: encapsulating storage, retrieval, and search behavior typically from a relational database. DTO, usado en data layer. Obtener datos de la BD

@Service: Business Service Facade. Usado en la business layer, transaction management

@Controller: Controller in MVC pattern. Se usa en web, en el patron mvc. Web layer

**Scopes**

* Singleton: one instance per Spring Context. Default scope.
* Prototype: new bean whenever requested
* Request: one bean per HTTP request. Web-aware Spring Application Context.
* Session: One bean per HTTP session. Web-aware Spring Application Context.
* Globalization: used in the context of portlet. Portlet is not used.

Spring is not thread safe.

The Gang of Four defines Singleton as having one and only one instance per ClassLoader.

The default injection in Spring is by constructor and setters. Spring use the set or the constructor to do the wired. @Autowired on top of the method.

Spring uses reflection to wire when use: this is considered setter injection

@Autowired

DataService dataService;

Constructor injection for mandatory dependencies. Helps to create immutable beans.

Setter injection for optional dependencies.

**Application Context**

* XML
* Java: use @Configuration annotation in the top. With Spring Boot, we are slowly moving towards complete Java Configuration.

**Autowiring**

* byType
* byName
* constructor: similar to byType, but through constructor. Finds the matching constructor by the type.

NoSuchBeanDefinitionException: when you don’t add @Component in the class of the bean to wire.

@Componet missing

Or @ComponentScan not defined properly

NoUniqueBeanDefinitionException: because there are 2 or more beans with the @Component. To fix yu could name the bean with the same name, then spring do wire by name.

Add @Primary

Add @Qualifier in the class and in the bean definition: @Qualifier(“mainAlgorithm”)

**CDI (Context Dependency Injection)**

CDI: API for use dependency injection. Spring supports CDI.

* Java EE Dependency Injection Standard (JSR-330)
* Spring Supports most annotations
  + @Inject (@Autowired)
  + Named (@Component & @Qualifier)
  + @Singleton (Defines a scope of Singleton)

**Spring Features in different versions**

* Spring 2.5 made annotation-driven configuration possible.
* Spring 3.0 made great use of the Java 5 improvements in language.
* Spring 4: first version to fully support Java 8. The minimum version is 6.
  + Introduced @RestController annotation
  + Spring 4.1 supports JCache (JSR-107) annotations
* Spring 5.0
  + Functional web framework
  + Support for Jigsaw (Java Modularity)
  + Support for reactive programming
  + Support for Kotlin

BOM: build of Material dependencies. Declares the dependency versions for all the spring modules. In <dependencyManagement> add the <dependency> spring-framework-bom

**Design Patters in Spring**

* Front Controller – Dispatcher Servlet.
  + All requests for the browser pass for the dispatcher server and then it identify the right controller.
* Prototype – Beans
* Dependency Injection
* Factory Pattern – Bean Factory & Application Context. Implement the factory patters to manage the beans and application context.
* Template Method
  + Org.springframework.web.servlet.mvc.AbstractController
  + Provides an abstract controller to manage all the handleRequest

Annotation most important

* @Component, @Service, @Repository, @Controller
* @Autowired
* @Primary
* @Qualifier
* @Configuration

**Spring MVC**

* Model 1 architecture
  + Browser 🡪 JSP 🡪 Model
  + JSP has all the logic to manage the entire application. First pass through the browser.
  + Difficult to maintain.
* Model 2 architecture
  + Browser 🡪 Servlet 🡪 View 🡪 Model
  + Pass for a servlet after the browser. The servlet redirects to the view.
* Front controller pattern
  + Browser 🡪 Front Controller 🡪 Servlet 🡪 View 🡪 Model

@RequestMapping allows to map the url and the type of the request.

All the datas comes in the model

ModelMap: can only store data

ModelAndView: can store objects and data.

Basic spring o spring MVC: The dispatcher server is configurated in the xml config, the path and the url pattern.

Spring Boot: automatically configures the servlet.

Form Backing Object: backend representation, the server side representation of a form. The form is in a jsp.

Java Validation API allows to validate the field in a form. Helps to validate data, size, min, max… is a Hibernate validation.

BindigResult: the result of the validation on the controller.

@PathVariable: send the value of the variable in the url.

@ModelAttribute

* Indicates the purpos of that method is to add one or more model attributes
* Invoked before @RequestMapping methods.
* Used to fill the model with commonly needed attributes.
  + Drop down values for form.

@SessionAttributes: List the name of model attributes which should be transparently stored in the session or some conversational.

@InitBinder: specifies the format used in this method.

@ControllerAdvice: when want to implement logic which is common to multiple controllers.

@ExceptionHandler(value = Exception.class)

Anything that happens in the application of exception or any of its subtypes would be handled by this method. When only want to handle exceptions for one controller, this method is in the specific controller.

Spring MVC is popular for:

* Clear separation of concerns
  + Dispatcher Servlet
  + View Resolver
  + View
  + Model

**Spring Boot**

* Spring based applications have a lot of configurations.
* When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars (for delivering static content) among other things.
* World is moving towards Microservices.
* We do not have a lot of time to set up 100 microservices.

Spring Boot Goals

* Quick start to spring
* Be opinionated
* Non functional features
* No code generation

Spring Boot Features

* Auto configuration. If you add some jar, spring does all the configurations.
* Spring Boot Starter Projects. It provides a quick start of many type of projects.
* Spring Boot Actuator. Helps to monitoring the application.
* Embedded server. You can add into the jar the server, then when you do not need to installer the server when you want to run the application.

Spring

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

Problems Spring Solves

* Duplication/Plumbing Code. With the autowired and the exceptiohandle. Dependency injection by bringing in loose coupling and also makes the application testable.
* Good integration with other frameworks.

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

* Eliminates all configuration needed by Spring and Spring MVC and auto configures it
  + No need for @ComponentScan. Default Component Scan.
  + No need to configure DispatcherServlert
  + No need to configure a Data Source, Entity, Manager Factory o Transaction Manager.
* Spring Boot looks at
  + Frameworks available on the CLASSPATH
  + Existing configuration for the application.
  + Bases on these, Spring Boot provides auto configuration.

@SpringBootApplication: this tells this spring boot application is going to create a spring context.

@EnableAutoConfiguration: it says at enable auto configuration to auto configuration is turned on.

@ComponentScan: enables scan components in the package.

Auto Configuration: 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.

Embedded Server

* Server is embedded as part of the deployable – jar.
* Removes the need to have the server pre-installed on the deployment environment.
* Default is Tomcat.
* Spring Boot also supports Jetty and UnderTow.

Starter Projects: Starters are a set of convenient dependency descriptors that you can include in your application (pom). 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.

Starters

* Spring-boot-starter-web-services – SOAP WebServices
* Spring-boot-starter-we – Web & RESTful applications
* Spring-boot-starter-test – Unit, Integration Testing
* Spring-boot-starter-jdbc – Traditional JDBC
* Spring-boot-starter-hateoas – HATEOHAS features
* Spring-boot-starter-security – Authentication and Authorization using Spring Security.
* Spring-boot-starter-data-jpa – Spring Data JPA with Hibernate
* Spring-boot-starter-cache – Enabling Spring Framework’s caching support.
* Spring-boot-starter-data-rest – Expose Simple REST Services using Spring Data REST.

Spring Initializer is a tool to bootstrap your Spring Boot projects. It is a place where you can create your spring project very easily.

Application.properties: It is a properties files for the application. Logs level, security. It is used to configure your spring boot application. The file is the central place where you can configure a lot of things relates to spring boot.

Externalize configuration: application.properties is the fundamental way to externalize configuration

@ConfiguratiionProperties(“basic”) to create a custom configuration

Profile: allows to have different configuration for different environments.

* Application-dev.properties
* Application-qa.properties
* Application-stage.properties
* Application.properties

Setting a profile

* Using -Dspring.profiles.active=prod in VM Arguments
* In application.properties, spring.profiles.active=prod

Profile

* Based on the active profile, appropriate configuration is picked up.
* Used to configure Resources – Databases, Queues, External Services

@Profile(“dev”) : to customize the bean for a specific profile

Spring Boot Actuator: is an application which you can use to monitor your application spring boot. Actuator is basically like a manager console; it helps you to manage your application.

CommandLinerRunner: interface used to indicate that a bean should run when it is contained within a SpringApplication. This is something you can implement the command line and implement a bean.

**Spring JDBC**

* Executes SQL querys, you need to create a jdbcTemplate
  + jdbcTemplate.update(“update todo set user=?”, todo.getUser());
* reduce the number of lines of code
* reduce complexity in doing JDBC
* removes the need to do all exception handling
* it can be implemented the RowMapper to maps the columns of a table and inserts values.c
* you do not need to manage the resource connections (execute, close)

**JPA**

JPA defines the mapping from your Java object to a table. JPA provides a lot of useful annotations where you can define the relationship between your classes and tables. We create the mapping, and the API creates the queries using annotations.

@Entity: becomes the bean into an entity that can be manage by JPA.

@Table(name = “Todo”): add when the name of the class is different to the name of the table

@GeneratedValue(strategy = GenerationType.IDENTITY): autogenerates the value of the column in the table.

@PersistenceContext is used to manage all the defined entities. It is used for an object EntityManager

@OneToOne: relation one to one object. The own table should be indicated in the map

@OneToOne (fetch = FetchType.LAZY, mappedBy = “´passport”)

Private Student student;

The default fetch is eager, you need to add the FetchType.LAZY to avoid an infinite cycle because the student property is eager.

@OneToMany (mappedBy = “project”)

Private List<Task> tasks;

@OneToOne

Private Project project;

The Data Source is defined in the configuration file, it is necessary add and configure the bean, jdbcUrl, user and password. The values are in the application.properties

In the persistence.xml file is defined what is the persistence provider that we want to make use.

The Entity Manager Factory is the one which create the entity managers. It needs the persistence.xml

Transaction Manager allows to manage the transaction, you can roll back all the transactions if one fails. To configure the Transaction Manager, you need the entityManagerFactory and the data source.

In the repository you can write the code to manage the entity

**Spring Data**

Common Abstractions to store and retrieve data from data stores.

* Independiente of type of data store.

**Spring Data**

Extends Spring Data for connecting to JPA. It helps us to use spring data kind of abstractions to connect to a JPA database.

The interface Repository must extends CrudRepository<primaryKey, type>

The CrudRepository has methods to save, find, delete…

PagingAndSortingRepository provides short and page fuctionalities.

**Unit Testing**

Spring does test very easy.

The tests are independent of the database.

**Moking**

Helps us to define what each method should return to on the service data.

DataService dataServiceMock = mock (DataService.class);

**Mokito**

Does easier the test because use annotation to create the mocks

@RunWith (MockitoJUnitRunner.class): says that this class will use Mockito annotations

@Mock

DataService dataServiceMock;

@InjectMocks

SomeBussinesImpl bussinesImpl;

@Test

Public void test…

**MockMvc**

Is used in the web layer, in the controllers.

@RunWith (SpringRunner.class): this is in the controller to test

@WebMvcTest (value = SurveyController.class, secure=false)

public class SurveyControllerTest {

@Autoeired

private MockMvc mockMvc;

@MockBean

private SurveyService surveyService;

JSONAssert: to assert that the design is as expected.

@SpringBootTest: we would want to launch of the real server.

@LocalServerPort: helps to get the port where the server is running

TestRestTemplate: to execute the rest request

@RunWith (SpringRunner.class)

@SpringBootTest (classes = Application.class, webEnvironment = SpringBootTest.WebEnviroment.RANDOM\_PORT)

public class SurveyControllerIT{

@LocalServerPort

private int port;

**AOP (Aspect Oriented Programming)**

You can say if want to execute some code before or after something. @Before, @After, @Around.

@Before (value = “execution(\* HiByeService.\*(..))”) – intercepts all the methods called in the class

Advice: is what code you want to execute. Que quiere que haga

Pointcut: helps to identify what kind of method you would want to intercept. Que interceptar

Aspect: is a combination of the advice plus the poitcut

Jointpoint: is the specific result of one execution. It is basically a specific execution point of the aspect.

Weaving: is just the process of making sure that these methods are getting called at the right instances.

Weaver: is the AOP framework.

Advice Types

* Before advice
* After returning advice
* After throwing advice
* After (finally) advice – Always executed
* Around advice – Most powerful – Performance Logging

AspectJ vs Spring AOP

* AspectJ is full fledged AOP framework
* Advise object not managed by the Spring container
* Advise join points other than simple method executions
  + (for example, field get or set join points=

**Web Service**

3 Keys

* Designed for machine-to-machine (or application-to-application) interaction
* Should be interoperable – Not platform dependent
* Should allow communication over a network

**SOAP Web Service**

Sends a SOAP XML Request and gets a SOAP XML Response.

SOAP Envelope:

* SOAP-ENV: Envelope
* SOAP-ENV: Header (optional)
* SOAP-ENV: Body

The transport of the SOAP request is HTTP or can be over the same queue.

WSDL: Web Service Definition Language. Defines the structure of the requests, types, messages, portType, binding, service.

**Contract**

Service Definition specifying. The contract specifies what is the format of the request, XML, JSON or any other kind of format.

* Format of Request
* Format of Response
* Request/Response Structure
* Transport used
* Endpoint details

With Spring Web Service we define an XSD first.

XSD: XML Specification Document.

JAXB: Java API for XML Binding. Defines how you can do XML to Java, and Java to XML.

Endpoint: accepts a request and does all the business logic needed (other service does this) and converts the response back into XML.

MessageDispatcherServlet: Receives the message request and then maps to the the appropriate endpoint method.

SOAP Fault: is the structure in which we can send a response back if there are any error.

**RESTful Web Services**

REST: Representational State Transfer. REST is a style architecture for distributed hypermedia systems.

A resource has an URI (Uniform Resource Identifier)

REST

* Data Exchange Format
  + No Restriction.JSON is popular
* Transport
  + Only HTTP
* Service Definition
  + No Standard WADL/Swagger/…

@GetMapping: shortcut of @RequestMapping

**HATEOAS**

* Hypermedia as The Engine of Application State
* Example
  + When requested for details of a facebook post, we return
    - Link for actions to like, unlike or comment on the post

Returns what actions can be performed, hear are the other things that consumer of his resource might be interested in. (links)

**Swagger**

OpenAPI Specification (formerly called the Swagger Specification)

The specification creates the RESTful contract for API, detailing all of its resources and operations in a human and machine readable format for easy development, discovery, and integration.

Accept header: HTTP Header used for Content Negotiation. Allows to choose the type of response, it can be json, xml…

Representation of a Resource: can be XML, JSON (format used to represent the resource)

Content Negotiation: is a negotiation which happens between the server and the client. The client sends the request in one format and then the server sends the response back in the same format.

Response Status

* 200 – SUCCESS
* 201 – CREATED
* 404 – RESOURCE NOT FOUND
* 400 – BAD REQUEST
* 401 – UNAUTHORIZED
* 500 – SERVER ERRROR

Versioning Options

* URI Versioning …/v1/person
* Request Param Versioning …/person/param?version=1
* Header Versioning …/person/header headers=[X-API-VERSION=1]
* MIME Time or Accept Header Versioning …/person//produces produces=[application/vnd.company.app-v1+json]