What is the use of profile in Spring boot:-

When we are developing a application, we are dealing on diff environment like Dev, QA, Production. So configuration properties for diff environment is diff. for Dev environment they need H2 or for QA dept they need Oracle DB. To make this code easy and clean, Spring has provision of profiles, to help separate the configuration for each environment. So we can make diff application properties for diff environment application-dev.properties, application-prod.properties. The default application.properties point to the currently active profile using spring profile active so that correct configuration is picked up.So based on our requrements we can active any profile at a time with the help of this command **spring.profiles.active=dev**

@SpringBootApplication:-

This *@SpringBootApplication* annotation is a convenience annotation that is equivalent to declaring [**@Configuration**](https://www.javaguides.net/2018/09/spring-configuration-annotation-with-example.html), [**@EnableAutoConfiguration**](https://www.javaguides.net/2018/09/spring-boot-enableautoconfiguration-annotation-with-example.html), and [**@ComponentScan**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/ComponentScan.html).

**@Configuration**

This annotation indicates that a configuration class declares one or more [**@Bean**](https://www.javaguides.net/2023/01/spring-boot-bean-annotation-example.html) methods and bean definations. These classes are processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

The main application class is also a bean,as it’s annotated with @Configuration, which is a @Component

**@ComponentScan**

This annotation tells Spring boot to scan all the current package and its sub packeges in order to identify annotated classes and configure them as Spring beans.

# @EnableAutoConfiguration Annotation Overview

this enables spring boot’s auto configurations mechanism. Auto configuration refers to create and configure beans automatically in the spring container.

Since the *spring-boot-starter-web* dependency added to classpath leads to configuring Tomcat and Spring MVC, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly.

@RequestMapping vs @GetMapping:-

@RequestMapping can be used with GET,POST,UPDATE,PUT mapping, @GetMapping is giving clarity that it is a GetMapping not PostMapping.

@Component is helps to marks beans as Spring managed [component. @Component](mailto:component.@Component) is a generic stereo type annotation.

@Service, @Repository, @Controller this all are specialization of @Component annotation.

Under @Component, @Repository, @Service, @Controller will be there

**@Service:**We specify a class with @Service to indicate that they’re holding the business logic.

**@Repository:**We specify a class with @Repository to indicate that they’re dealing with **CRUD operations**

**@Controller:**We specify a class with @Controller to indicate that they’re front controllers and responsible to handle user requests and return the appropriate response.

@RestController

Classes annotated with stereotype annotation are automatically registerd as Spring manage beans during the component scanning process.

Other non –stereotype annotation or custom annotations generally don’t trigger automatic bean registration. Like @RequestMapping , @Autowired, So if you want to create bean of these no-stereotype annotation then you have to configure beans through configuration metadata with the help of XML or JAVA file

@Target(ElementType.***TYPE***)

@Retention(RetentionPolicy.***RUNTIME***)

@Documented

@Inherited

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan(excludeFilters = { @Filter(type = FilterType.***CUSTOM***, classes = TypeExcludeFilter.**class**),

@Filter(type = FilterType.***CUSTOM***, classes = AutoConfigurationExcludeFilter.**class**) })

**public** **@interface** SpringBootApplication {

@Target(ElementType.***TYPE***) : means in which place this annotation will be used class or fields or constructors etc. here TYPE means class.

@Retention(RetentionPolicy.***RUNTIME***) : this annotation is used to specify to what level anootation will be available

[1.@Retention(RetentionPolicy.***RUNTIME***)](mailto:1.@Retention(RetentionPolicy.RUNTIME))

2.@Retention(RetentionPolicy.***SOURCE***)

[3.@Retention(RetentionPolicy.***CLASS***)](mailto:3.@Retention(RetentionPolicy.CLASS))

@Inherited : By default, annotations are interface and they are not inherited to subclasses. The @Inherited annotation marks the annotation to be inherited to subclasses.

=> Autowiring :-

-> Autowiring is the feature of Spring Framework by which we can achieve "DI automatically"

=> Dependency Injection (DI) :-

-> Dependency Injection is a design pattern that is used to implement IoC principal

-> Dependency Injection main functionality is to "inject" one object into another object

What will Happen if we replace @Controller with @Component?

This means @Component and @Controller are the same with respect to bean creation and dependency injection but as @Controller is a specialize from of @Controller so that all @Controller has all the functionalities of @Component also it has its own extra specific functionalities to handle HTTP request and response that’s why.

HOW MENY LAYERS OUR APPLICATION HAVE?

Controller , Businness , DAO layer

What will Happen if we replace @Repository with @Component?

We must denote @annotations based on layers.

@Repository not only helping in annotation based configuration but also catch Platform-specific exception and re-write them as one of Spring’s unified unchecked exception.

Difference between Spring and Spring boot:-

|  |  |
| --- | --- |
| Spring | Spring boot |
| Spring is an open-source lightweight framework widely used to develop enterprise applications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. |
| To run the Spring application, we need to set the server explicitly. | Spring Boot provides embedded servers such as Tomcat and Jetty etc. |
| To run the Spring application, a deployment descriptor is required. | There is no requirement for a deployment descriptor. |
| It doesn’t provide support for the in-memory database. | It provides support for the in-memory database such as H2. |
| Developers have to define dependencies manually in the pom.xml file. | pom.xml file internally handles the required dependencies. |
| In Spring to work with dependency we need to add jar files. | In spring boot we don’t need to download jar files and add them in classpath also we don’t need to add dependency in pom.xml files just we have to add spring boot starters dependency. |
|  |  |

What is REST?

Representational State Transfer is an architractural style for providing some guidinglines to communicate between client and server.

Representational means format means in which format data is going either it is JSON or XML.

State means data

Transfer means transfer data between parties.

Guidelines:-

Client – Server architechture.

Stateless – server me koi vi data store nehi hona chaiye , sirf data ko process karke veg dena hai.

Cacheble – for same request we can store data in cache for better performance

Layer architecture – for scalableity

JAVA VALIDATION:-

JSR380 – it is a specification

Hibernate Validator – it’s a implementation of JSR380.

MethodValidNotArgumentException

What is the plugins in Spring boot?

Plugin can be define as a extention of spring boot which is use to do specific task like compiling code ,test with JUNIT, creating jar/war/ear files and documentation of the projets etc. It provides Spring Boot support in Apache Maven

Example - Spring-boot-maven-plugin

Handeler methods:

RequestMapping

GetMapping

Path veriable passed on the parameter of the handeler methods.

@RequestMapping(“/show/{id})

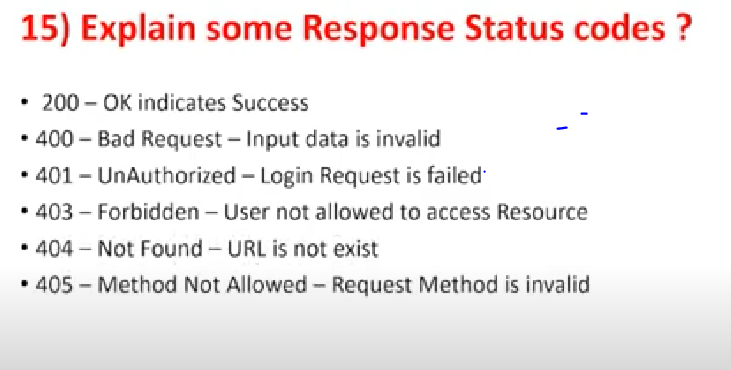
@RequestBody vs @ResponseBody

@RequestBody:-

The @RequestBody annotation, used as a handeller method parameter,binds the HTTP request body to a transfer or domain object.

Spring automatically deserialize incoming HTTP request to the java object using Http Message Converters.

@ResponseBody:-

The @ResponseBody annotation is used to serialize the returned object automatically in JSON and bind it with Http response body.

What is View Resolver:-

View Resolver in responsible to rendering of models in web browser.

CUSTOM ANNOTATION FOR VALIDATION:-

@Documented

@Retention(RUNTIME)

@Target({ TYPE, FIELD })

@Constraint(validatedBy = LoanApplicationValidator.class)

public @interface ValidatePurposeType {

}

public class LoanApplicationValidator implements ConstraintValidator<ValidatePurposeType,String> {

@Override

public boolean isValid(String value, ConstraintValidatorContext context) {

return false;

}

}

CUSTOM GENERATOR:-

* To create custom generator we should implement identifierGenerator interface
* In that interface we have generate() method… we should write logic in that method to generate PK column value according to our requirement.

Steps to develop Data JPA application with Custom Generator:-

* Create sequesnce in DB for suffix value generation.
* Create custom generator class by implementing Identifire Generator interface.
* Configure Custom Generator in Entity.