Spring Boot Spring - xml cfgs + AutoConfiguration

=>Based on the starters (special jar files) that are added to class path the spring boot takes care of multiple common operations like making java classes as spring beans, Injecting one spring bean with another spring bean, Establishing the connection Db s/w, providing embedded server and etc..

Spring boot starters (Special dependencies (jar files))

=> These are special and advanced dependencies(jar files) in Spring boot programming having

capability to perform Autoconfiguration activities ...

=> The autoconfiguration activities are perform common things dynamically

that are required in application development and execution

->configuration certain java classes as spring beans

-> Performing dependency Injections among the spring beans

-> providing dependent and relavent jar files

-> providing embedded Db s/ws

-> providing embedded Servers

-> Supplies the necessary plugin in required for maven or gradle tool

-> Gives DB Connectivity configurations

and etc..

App

note:: Embedded DB s/w means the DB s/w that comes along with spring Boot Execution (eg :: h2) note:: Embedded Server s/w means the Server s/w that comes along with spring Boot Execution (eg:: embedded tomcat, embedded jetty)

The syntax of spring boot starter is

spring-boot-starter-* (naming convention)

nearly 1000+ starters are available..

eg: spring-boot-starter-jdbc

spring-boot-starter-web

spring-boot-starter-batch

spring-boot-starter-security

spring-boot-starter-jpa

and etc..

=> if java App uses third party apis/libraries (other than java, jee apjs,user-defined apis) then that api related both main and dependent jar files must be classpath..

added to

=> if the classes of first.jar file are using the classes of second.jar file .. then we can say first.jama ar file and second.jar files dénet jar file.

=> While working with frameworks or third party apis/libraries finding main jar files is easy.. but indentifying theier dependent jar files and transitive dependent jar files of main jar files is very difficult. (the dependents of dependent jars)

=> getting both direct and indirect dependent jar files of main jar files is called arranging transitive

dependents..

=> maven, gradle and etc.. build tools simplify this job.. if we add main jar file dependency/info

to these tools they automatically download both main jar files and direct and indirect dependent jar files. (transitive dependents also will come)

=> if we add spring boot starters info to app using maven/gradle not only we get both direct, dependent indirect jar files.. but we also get relavent jar files.

if we add spring-boot-starter-jdbc to the project then we get

=>spring core jar files (spring-context-support and thier dependents)

=> spring jdbc jar fiels (spring -jdbc-<ver>.jar, spring-tx-<ver>.jar)

=> sprinb boot jar files main jar files

=> Hikiari cp jar files

=>log4j, slf4j jar files relavent jar files

=> jacksion api jar files

and etc..

boot

note::: maven and gradle are the build tools given

to the developers to simplify the application devleopement process by integrating with IDES like Eclipse

What is the difference b/w dependent jar file and relevant jar file?

First iar

(main jar file)

oracle.iar

class-A

lots of classes

(dependent jar file) Second.iar

(dependent jar file) Third.jar

to connect to oracle DB (relavent jar file)

Class-C

class -B

Let us assume first.jar is adeed to CLASSPATH for DB interaction, So Spring boot automatically gives oracle.jar as relavent jar files by guessing that programmer may show intrest talking to oracle DB s/w .. (if not talked it will be unused)

=> if class A of first.jar are using class B, class C of Second.jar and Third.jar file s then First.jar is called Main jar file and Sencond.jar,Third.jar files are called dependent jar file s

A.jar classes are using B.jar classes B.jar classes are using C.jar classes C.jar classes are using D.jar classes

Spring boot starter (jar file) gives

=>main jar file => dependent jar files Transitive dependency =>dependent's dependent jar file (transitive dependencies) =>relevant jar files (extra jar files may be used or unused)

note: Here B.jar file is called dependent jar file to A.jar file (main jar file) B.jar, C.jar and D.jar files are called

transitive dependent jar files to A.jar file direct, indirect dependent jar files are nothing but transitive dependent jar files

Here the maven/gradle tools arranges the given starter related main jar files, dependent jar files, transitive dependent jar files and also relavent jar files

main and dependent jar files

(including transtive dependents)

note:: relevant jar files are not dependent jar files of main jar files becoz the classes of main jar files do not use the classes of relevant jar files directly or indirectly

spring starters gives main jar files + trasitive dependent jar fiels + relavent jar files. (direct and indirect dependent jar files)

=> we can collect this starters info required for maven or gralde from

mvnrepository.com or using spring initializers or using sts plugin or ...

==for maven===

(best)

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-jdbc --> <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

<version>3.2.4</version>

</dependency>

add this to pom.xml

of maven prpject

(But they will be required in overall application development)

=>DataSource obj represents jdbc con pool having set of readily available jdbc con objects before actually being used.

eg: hikaricp, proxool, c3p0 and etc...

represents

hikari cp (jdbc con pool)

=== for gradle===

// https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-jdbc implementation 'org.springframework.boot:spring-boot-starter-jdbc:3.2.4'

=> There are multiple JVM based languages

java, scala, groovy, kotlin and etc..

note :: all these languages have their own programming syntax and compilers.. but they

give common byte code (.class file) which can be executed the common JRE /JVM

compile

.java

.class

compile

.groovy

->.class.

JRE/JVM

output comes here

.scala.compile

.class

DataSource obj

add this to build.gradle

of gradle project

note: All operations on the jdbc con pool will be done by DataSource obj like creating new con object, accessing one con obj, deleting the existing con obj and etc...

Spring boot latest version ::

3.4.2 (jan 2025)

STS (Spring Tool Suite) is an IDE that is used by developers for developing spring/spring boot Projects.. but we can add all these facilities to Eclipse IDE by adding STS plugin to Eclipse IDE

in

.kotlin

compile

--> .class

Spring boot Features

=> we can do coding for spring boot App in multiple jvm based languages like java, groovy, koltin .. since all are habituated with .. java is recomanded.

=> SPRING Boot and its module gives new annotations by combining multiple relavent annotations

to reduce the burden on the programmer

@RestController = @Controller + @ResponseBody

@SpringBootApplication = @Configuration + @EnableAutoConfiguration + @ComponentScan

=> we can develop and test spring boot Apps with out installing external DB s/w and webServer/Application server s/w becoz the spring boot app execution give Embedded Db s/w and embedded servers based on the starter we add.

Why do we need starters (special jar files) Spring boot app? ans) ans1) Based on the starters that are added to the CLASSPATH of the Project the Spring boot takes care of lots of AutoConfiguration activity ans2) Based on the starters that are added to the CLASSPATH of the Project the build tools like maven /gradle starts giving main jar files, dependent jar files and the relavent jar files

=> Spring boot starters reduces no.of jar files that should be added by programmer to the project.. the each starter gives main jar files, dependent jar files and relavent jar files.

=> Converting existing spring projects to spring boot projects is very difficult and not recomanded..

So use spring boot for new projects (green field projects) .. not for migration (not for brown filed projects)

hyd is brownfield capital

chandigadh is green field capital

AP's Amaravathi is green field capital

Spring boot = spring - xml cfgs + autocomfiguration

Important components of spring Boot

AUTO

CONFIGURE

A

CORE

CLI

x

ACTUATOR

STARTERS

TOOLS

CLI:: Command Line Interface

Spring Boot Auto Configure

Module to auto configure a wide range of Spring projects. It will detect availability of certain frameworks (Spring Batch, Spring Data JPA, Hibernate, JDBC). When detected it will try to auto configure that framework with some sensible defaults, which in general can be overridden by configuration in an application.properties/yml file.

Spring Boot Core

The base for other modules, but it also provides some functionality that can be used on its own, eg. using command line arguments and YAML files as Spring Environment property sources and automatically binding environment properties to Spring bean properties (with validation).

Spring Boot CLI

A command line interface, based on ruby, to start/stop spring boot created applications.

Spring Boot Actuator

This project, when added, will enable certain enterprise features (Security, Metrics, Default Error pages) to your application. As the auto configure module it uses auto detection to detect certain frameworks/features of your application. For an example, you can see all the REST Services defined in a web application using Actuator.

Spring Boot Starters

Different quick start projects to include as a dependency in your maven or gradle build file. It will have the needed dependencies for that type of application. Currently there are many starter projects (We will learn about few of them in the next section) and many more are expected to be added.

Spring Boot Tools

The Maven and Gradle build tool as well as the custom Spring Boot Loader (used in the single executable jar/war) is included in this project.

Based on the starters and jar files thaboot

are added to the CLASSPATH the spring f/w

perform multiple common operations internally

and dynamically based on the AutoConfiguratuion feature

yml/yaml file is alternate file properties file which can be used to provide inputs Spring boot framework
yml/yaml :: yet another markup language

note: Spring boot autoconfigure component takes care of common activities of the projects based on the starters and jar files that are added to the CLASSPATH eg:: Establishing the connection with DB s/w

note:: Spring boot core takes care of basic activities of the Application startup and execution

eg:: recognizing the properties /yml files and reading their content

Spring boot CLI (Command Line Interface) provides set of commands to use from the command promot to create, test, build,run and pack the spring boot Applications

note:: The dependency (jar) added to maven or gradle project will give us only main and dependent jar files (including transitive dependencies) note: The spring boot starters added to maven or gradle project will give us main jar files, dependent jar files and relavent jar files.. there are 1000+ starters in spring boot

spring-boot-starter-<<*> (syntax)

eg:: spring-boot-starter-jdbc

spring-boot-starter-web

spring-boot-starter-jpa and etc..

Spring Boot Starters

Spring Boot incorporates many starters packages (in Maven & Gradle) which you can include in

order to add appropriate support in your project. At a high-level there are so far 6 types of starters packages available. You can find all of them in Spring Boot official WebSite.

Spring Boot Actuators are the non-functional readymade features/services that will be added every Project to enable better monitoring and maintaianance of the Project eg:: Actuator giving threads info Actuator giving heap info Actuator giving processes info Actuator giving health info

Functional features --> related to b.logic of the Project eg:: patient registration, doctor registration non-functional features ---> not related to b.logic but required in the project eg: health info, threads info and etc..

=>The STarters added to the Project are responsible to make the Springfamework to perform the AutoConfiguration activity

CLOUD

WEB

The multiple microservices of the MicroService Architecture Project

DATA STORE

SPRING CORE NETFLIX OSS

...

MESSAGE

BATCH

will communicate with each other using Netflix supplied third party tools

to

Netflix is not just OTT platform.. it is a big software vendor providing lotstools related microservice archicteture based application development are given by Netflix

duction-to-spring-b...

Starter

Spring boot App development Procedure

Cloud

(1) create the Basic Project

start.spring.io

STS

(or)

Structure

Data store

Message

(or)

(2) Select the build Tool

MAVEN

GRADLE

Batch

Over the TOP (OTT)

Purpose

To develope MicroService

Architecture Apps

Neflix OSS

STS :: Spring Tool Suite

(Exclusive IDE for the spring/spring boot Apps)

note:: Good IDES for Spring/spring boot App development

a) Eclise IDE with STS Plugin (best) (1)

b) STS IDE (3)

To interact with Db s/w (both SQL and NoSQL)

For Messages based interaction between

the Apps

For Batch processing

(processing the huge amount of

data at a time)

To develop the web applications and Distributed Apps

Gives the tools required for the MicroServices based Apps interaction

STARTER

(3) Select the Starters

Web

SpringBost

Application

(4) Develop the code

JAR

(5) Pack and release the code

CODE

TOMCAT

for execution

c) IntelliJ IDE (2)

Every Spring boot Application contains one important annotation that is @SpringBootApplication which is combination of 3 annotations

a) @Configuration/@SpringBootConfiguration :: makes the java class as the

spring bean cum configuration class

note:: A plugin is a patch software that added to the existing software to provide additional functionalities

to the existing software

where other java classes can be configured

as spring beans.. It is alternate to spring bean cfg file (xml file)

b) @EnableAutoConfiguration:: This annotation makes the spring boot framework to

c) @ComponentScan

::

perform AutoConfiguration i.e based on the starters

that added it will give lots common things including

the processing certain pre-defined classes as spring beans and doing injection activities on them

Takes all the java classes marked with stereo type annotations (like @Component, @Service and etc..) from current class pkg and its sub pkgs to make them as spring beans automatically..
eg:n@ComponentScan(base Packages="com.nt"

note:: @SpringBootApplication is class level Annotation..

Example

@SpringBootApplication

--

publi class App1Config{

public <RT> <method>(){ useful to make

@Bean

}

}

...

bean id

@Bean(name="dt") public Date createDate(){ return new Date();

This class can be used to provide instructions/inputs to spring boot framework

In Spring Boot Apps, we generally take main class having @SpringBootApplication annotation in the root packages like "com.nt" .. So that all the classes of that root package and its sub packages can be scanned

for the java classes having stereo type annotation to make those classes as the spring beans

By adding STS plugin to eclipse IDE, we can use all features of STS IDE in Eclipse IDE Itself

=> To Configure user-defined classes as the spring beans we need to use Stereo type annotations like @Component, @Service and etc... becoz we can add these annotation on the top of user-defined class definitations

=> To Cfg pre-defined classes the as the Spring beans we need to use @Bean methods in @Configuration class becoz we can not edit the source code of the pre-defined classes to add the stereo type annotations

This method returned Date class obj will become spring bean automatically having the bean id "dt".

}

pre-defined java clases as spring beans.

@Bean Methods will be executed automatically to make the

method returned objs as the spring beans

Thumb rule to remember while working with spring boot for Application developement

=====

a) Configure user-defined classes as spring beans using stereo type annotations like @Component,@Service and etc..

b) Configure pre-defined classes as spring beans using @Bean methods in

@SpringBootApplication class if they are not coming as spring beans through

AutoConfiguration based Spring boot starters that added.

note:: we can give inputs or can override defaults of AutoConfiguration

with the the support of application.properties/yml file entries.

c) In the main(-) method of @SpringBootApplication class get the

IOC container by calling SpringApplication.run(-,-) method and use

that container to get spring bean class obj ref by calling ctx.getBean(-,-) method

// get IOC container

ApplicationContext ctx-SpringApplication.run(<cApp1Config.class

//get Spring bean class obj ref

Date d=ctx.getBean("dt",Date.class);

, args);

main class name

=>we are getting Spring bean class

yml/yaml :: yet another markup language

(or)

yml/yaml :: yaint markup language

yml/yaml :: yamiling markup language

First App development

(@Component)

(target class)

WishMessageGenerator

(generates the wish

message based on the

bean id

object from the IOC container

by giving its Bean id

@Bean method in @Configuration class (dependent class) java.util.Date (or) java.time.LocalDateTime (java 8)

(best)

current hour of the day) WishMessageGenerator uses java.util.Date or java.time.LocalDateTime class

for getting system date and time to collect the current hour of the day which can be used to generate the wish message. So "WishMessageGenerator" is called target class and "java.util.Date or java.timeLocalDateTime" is called dependent class. =>The Spring Bean that uses other spring bean services is called target/main spring bean

=> The spring bean that acts helper class to other spring bean is called dependent spring bean

eg1: Student(target) course Material (dependent) --->Student uses the course material eg2: Crickter(target) CricketBat (dependent) --->Cricketer uses the CricketBat

eg3: Flipkart(target) DTDC (dependent) ---> Flipkart uses DTDC as the Courier Service

What is the difference between stereo type annotations like@Component,@SErvice

This method performs

(a) Boot straps (starts) the Spring boot Application

(b) creates the IOC container by taking the given current

and @ComponentScan Annotation?

class as the Configuration class (by loading and creating object for configuration class)

(c) Based on the starters that are added, and inputs that are given in the application.properties file performs the AutoConfiguration activities

(d) Performs the Componescanning to get stereo type annotation classes and to make them as the spring beans

(e) Automatically calls @ Bean methods to make the method returned objects as the spring beans

(f) returns ApplicationContext obj representing the IOC container

and etc...

Ans) @ComponentScan takes the given package name and makes the IOC container going to

that given package and sub packages to search and get list of stereo type annotation applied java classes

So the same IOC container makes those classes as the spring beans becoz of the stereo type annotations annotation that are appiled

=>Scanning packages for identifying the stereo types annotations is done by @ComponentScan annotation

=> but making the java classes as spring beans is done by Stereo type annotations

A.java

package com.nt.p1; @Component("a1")

public class A{

}

AppConfig.java

B.java

package com.nt;

package com.nt.p2; @Component("b1") public class B{

@Configuration @ComponentScan (base Package="com.nt") @EnableAutoConfiguration public class AppConfig{

3

These Annotations

with

can be replaced single

@SpringBootApplication

}

}

=> Making the Java class as the spring bean is nothing but making the IOC container

(spring container) loading the java class, creating object, managing the object

and destroying the object [Total life cycle of spring bean will be taken care by the IOC container]

To simplify the Spring boot App development, we need to install STS plugin to Eclipse IDE

Procedure is

(By default it will search for stereo type annotations

Help menu ---> eclipse market place --> search for "Spring Tool Suite" --> go ---> select

Spring Tool Suite 4 ---> install ----> next ---> select all check boxes ---> next ----> Accept terms and conditions --->

restart the IDE .

note: Plugin is a patch software that provides additional functionalities to the existing software

In Eclipse IDE, we can install two types of Plugins

a) Eclipse IDE's supplied plugins

(Use help menu --> install new software option)

|

eg: Lombok api p2 plugin gui builder Plugin and etc...

b) Third Party Plugins for Eclipse IDE

(use help menu ----> eclipse market place option)

eg:: STS plugin, BuildShip plugin and etc...

in the current package and its sub packages)