=>SpringApplication.run(-,-) method is having two args

arg1---> current class nothing but @SpringBootApplication class as the @Configuration class

arg2 ---> String args[] --> nothing but command line args we can be bind to spring beean properties

Spring boot First application Flow of execution

========

=>SpringApplication.run(-,-) (static method of SpringApplication class) performs bootstraping(starting) of spring boot app by performing multiple activities including the ApplicationContext IOC container creation by taking given class as the configuration class.

In Spring,Spring boot we have two IOC cintainers

a) BeanFactory b)ApplicationContext (best)

=> In standalone Spring Boot Apps, the SpringApplication.run(-,-) method uses the following statement to create the IOC container

>

(internal

statement)

AnnotationConfigApplicationContext ctx=

new AnnotationConfigApplicationContext(<Given configuration class>);

The first arg of SpringApplication.run(-,-) will be taken here as the Configuration class

[@SpringBootApplication class becomes the @Configuration class]

//SeasonFinder.java (Target class) package com.nt.sbeans;

import java.time.LocalDate;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component;

BootProj01-DependencyInjection [boot]

src/main/java (for placing source code) #com.nt [#8]

› BootProj01 DependencyInjectionApplication.java

#com.nt.sbeans

SeasonFinder.java

src/main/resources (for placing properties files/yml files)

application.properties

src/test/java (for placing junit code)

JRE System Library [JavaSE-17] (represents JRE)

Maven Dependencies (holds all maven downloaded jar files)

<

src

>

(source folder)

target (output folder)

HELP.md

mvnw

mvnw.cmd

helper files in Maven setup

(#10)

@Component("sf")

[#8]

public class SeasonFinder {

@Autowired // Field Injection

private LocalDate date; //field [#11]

[#12]

//b.method

(19)

public String findSeasonName() {

//get current month of the year

int month=date.getMonthValue(); // 1-12

//find the season name

if(month>=3 && month<=6) {

pom.xml (File to give instructions to maven tool)

}

return "Summer Season";

else if(month>=7 && month<=10) {

return "Rainy Season";

>

>

>

}

else {

}

} }

Main class /Starter Class

package com.nt======

import java.time.LocalDate;

return "Winter Season";

(20)

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.annotation.Bean;

import com.nt.sbeans.SeasonFinder;

(6) reads this annotation

@SpringBootApplication

(5) Loads the this class as the @Configuration class

public class BootProj01DependencyInjection Application object---> makes the object as the spring bean

@Bean(name="dt")

[#9]

public LocalDate createLD() {

(#10) }

return LocalDate.now();

(2)

public static void main(String[] args) {

having class name as the default bean id

Dependent spring bean Configuration here

(1) Run the application

(IOC container creation) (14) //get IOC container by bootstapping the application(3) --> Bootstrapping of the Application ApplicationContext ctx-SpringApplication.run(BootProj01DependencyInjectionApplication.class, args); //get Target spring bean class obj

(17) SeasonFinder finder-ctx.getBean("sf",SeasonFinder.class);

//invoke b.methods

(15)

(21) String result=finder.findSeasonName(); (18)

System.out.println("Season name::"+result); (22) //close the container

((ConfigurableApplicationContext) ctx).close();

(24)

}

} (end of main(-)

method and

the application)

(23)--> When the IOC container

is closed.. all objs will be

(4) Takes the given main class as the Configuration class

vanished including the Container AND ITS Spring Beans

(9) IOC container searches for the @Bean methods in @Configuration class

BootProj01DependencyInjection Application class object (#5) (class name self acts the default bean id ::

bootProj01DependencyInjection Application)

(7) The @EnableAutoConfiguration annotation of @SpringBootApplication checks are there any special starters (jar files) added to CLASSPATH .. Since not found, So no AutoConfiguration activities will take place

(8)@ComponentScan annotation of @SpringBootApplication scans current package (com.nt) and its sub packages to get all the classes names that are annoted with stereo type annotations like @Component.. In that process finds

and finds only one @Bean method in the @SpringBootApplication class (MAIN CLASS)

com.nt.sbeans. Season Finder class and it make as spring bean

@SpringBootApplication = @Configuration+

@ComponentScan + @EnableAutoConfiguration

(10) IOC container searches for Singleton scope (default scope) spring beans

and pre-instantiation (early/eager object creation) [Season Finder and

=> com.nt.sbeans.Season Finder @Bean(name="dt")

public LocalDate createLD(){

java.time.LocalDate]

Since no scope is specified the

default scope is singleton scope

...

SeasonFinder

class bj(sf) (#10)

LocalDate class (dt)

(#10)

Spring Beans are ready

date:

@Autowired

based Injection

(#12)

ton

note:: As Part of pre-instantiation the single scope stereo type annotation classes will be instantiated (object creation) automatically and the @Bean methods of @Configuration class will be executed automatically

11)@ComponentScan annotation activaties the code related to @Autowired Annotation to search and get all the HAS-A properties where @Autowired annotation is applied .. In this process it finds only in Season Finder spring bean class

In SeasonFinder class.

@Autowired

private LocalDate date; (HAS-A property)

13) IOC container keeps the singleton scope spring bean calsses objs refs

12) @Autowired Annoation gets the HAS-A property type (LocalDate) and seaches for the spring bean whose class name is LocalDate.. So makes the IOC container to inject LocalDate class obj(dependent class obj) to

Season Finder(target class) class obj's HAS -Aproperty "date" (This is nothing but field Injection)

in the internal cache of the IOC container

sf

(16?)

dt

bootProj1Dependency

InjectionApplication

(bean ids) (keys)

SeasonFinder class obj ref

LocalDate class obj ref

BootProj01DependencyInjection Application class obj ref

(bean class obj refs)

IOC container Internal cache

(values)

14) SpringApplication.run(-,-) method returns ApplicationContext object representing the IOC container

note:: using interface ref variable that is pointing to impl class obj, we can call only the common methods of the both.. if u want to call direct methods of impl class we need go for type casting.. like ctx.close() method call

What is difference b/w spring and spring Boot?

JEE = Java Enterprise Edition (old name)

JEE = Jakarta Enterprise Edition (new name)

Spring

Spring Boot

also

as

a) It is called Java EE framework or Application Framework

b) provides abstraction on Java, JEE technologies

and simplifies their App development

c) Avoids boilerplate code related to

JAVA,JEE Technologies based application development

more

(But still continues some boilerplate code)

d) The Main feature of spring framework is Dependency Management (both Dependency Injection and Dependency lookup)

e) supports Xml driven cfgs to provide inputs/cfgs to IOC container

(f) Allows to develop Apps using 3 types of cfgs

a) xml driven cfgs

xml

b) Annotation driven cfgs

(inputs)

c) 100% Code driven /Java Config approach cfgs

g) Programmer creates IOC contaner

explicitly (except in spring MVC Applications)

(h) Does not give Embedder Server .. Sorun spring

based web applications we need to arrange server explicitly (Go for extenal servers)

(i) does not give any InMemory Databases

(go for Extenal DB s/ws)

j) Supports good amount of Loose coupling

using spring bean cfg file (xml file) (Changing one dependent with another depedent for target class with out touching the java source code)

just

a) It is called Spring Boot Framework

b) provides abstraction on spring framework

and simplies spring Apps development

c) avoids spring framework related boilterplate code..

note: the code that repeates in multiple parts of the App /project either with no changes or with minor changes is called boilerplate code (commonly repeating logics are called boilerplate code)

d) The main feature of spring boot framework is AutoConfiguration (giving commons things automatically)

directly

e) Does not support xml driven cfgs.. Spring boot avoids or minimize xml driven cfgs..

Spring boot framework is self intelligent

note:: Dependency Management arraging the dependent class obj to target class obj by taking their life cycle

[if want to use certain features of spring for which annotations are not there like method replacer, bean aliasing, inner beans, tiles,bean inheritance, collection merging and etc.. we need to link spring bean cfg file (xml file) to @SpringBootApplicaiton calss using @ImportResource)

(f) supports only one style of cfgs that is through Annotations directly giving auto configuration inputs through application.properites/yml) note:: can add special cfgs using xml file through @ImportResource

(g) Programmer does not create IOC container rather he gets it by calling SpringApplication.run(-,-) according to Applicaiton type.

(h) gives Tomcat,Jetty and etc. server as Embedded servers.. in web application... (only for testing) (Allows to work with both Embedded and extenal servers) Embedded/ HSQL

i) gives InMemory Database like H2' Tonly for testing) Supports both Embedded and Externa

wname)

j) Loose coupling bit less becoz all cfgs takes place

through annotations which is java code..

(To get perfect loose coupling here we need to work with xml file support)

k) We need to add dependencies (jar files) manually 1) Spring boot gives starters (kind of dependencies)

and directly (using maven /gradle we can

get dependent jar files when we add main jar files to add

i.e still we need relevant jar files explicitlty) (No starters support)

L) Suitable for new projects development,

which provides main jar files,dependent jar files and relavent jar files..

(Here also we use maven/gradle to add starters)

L) Not suitable for Migration projects and also not

to enhace existing projects and to migrate project from one version of spring to anther version of spring (Supports both brown field and Green field project development)

M) Bit light weight compare to spring boot becoz No AutoConfiguration support (Light weight in terms of Memory utilization and CPU time consumption)

N) Spring framework is suitable for

devleoping standalone Apps and

small scale and medium scale web applicaitons..

Exclusive

(0) No support for MicroServices architecture based application development..

(p) No built-in properties file support i.e every properties file must be configuredd explicitly (q) NO support for yml files

(r) we can not inject the entries of properties /yml file to spring bean properties throguh bulk Injection process i.e we need to go for 1 to 1 injection using @Value annotation

(s) Gives multiple independent annotations to use

in spring programming

(t) Gives less productivity

The server that is installed by programmers manually is called_Extenal server...

eg:: GlassFish, Tomcat, Widlfy and etc.. note:: Tomcat is popular as the Embeded and external server

Embedded DB s/w InMemory DB s/w

Supports the starters

suitable to convert spring projects to spring boot projects But very much suitable hew projects development

from scrrach level. (Supports only green field project development)

M) Bit Heavy weight compare to spring becoz

of AutoConfiguration many unneccesary objects

will be created.. even unneccessary jars may add

In this we use

Spring-boot-starter-JDBC gives the fllowing objs as spring beans =>HikariDataSource

=>JdbcTemplate

one or two >Named ParameterJdbcTemplate

objects

=>DataSourceTransaction Manager and etc..

comes through autoconfiguration

N) Spring Boot is good to

develop Large scale web applications,

Distributed Apps and MicorService

Architecture based applications

[Micro Service Archeicture says develope

different modules as diffrent projects and intergrate

them using different thrid party libraries]

(0) supports more exclusively..

in

(p) Gives built properties file called application.properties with implicit configuration.

(q) Gives great support for yml files

r) Supports the bulk injection

in the same env.. using @Configuration Properties annotation

(s) Gives advanced annotations which internally combines multiple related spring annotations
eg@SpringBootApplication = @Configuration +

(t) Gives more productivity

yml files are alternte files to proeprties files

(or)

yaml/yml :: Yet another markup language

Yain't markup language

yamling markup markup language

@EnableAutoConfiguration + @ComponentScan

(Doing more work in less time with good accuracy)

(or)

JDBC

Developer

develops

Spring Boot App

using

Spring Boot Framework

uses

Spring Framework

uses

JNDI

JMS

many more

JAva - JEE Technologies

JAVA Programming Language

Microservices App = web services App ++

(Restfull webServices App ++)

JNDI:: Java Naming and Directory Interface

JMS :: Java Messaging Service

JPA :: Java Persistence API Jakarta persistence API

=>JPA is s/w specification that provides set of rules and guidelines for the software vendor companies to create ORM softwares

What is the relation b/w webServices and MicroServices?

=>WebServices is given to develop Distributed Apps (Application to Application .two

WebServices can implemented in approaches

interaction)

a) SOAP based webservices (Legacy -- Almost outdated)

-> Jax-rpc, jax-ws are Technologies to develop SOAP based webservices

SOAP : Simple Object Access Protocol

(data)

HEre the media of communication b/w Client App and Server App is only Xml

(data)

Here the media fo

Communication can be the

-> Apache CFX, AXIS are frameworks to develop SOAP based webservices.

by Restfull webServices (Popular - Hot Cake) (REST :: Representational State Transfer)

-> Jax-RS (metro) is technology to develop Restful webServices

-> Jersy, Spring Rest, Rest easy, Restlet and etc.. are frameworks to develop Restful webServices spring boot rest

=>Mircroservices are built on the top of Restful webService to develop different services/modules as different projects (each project is called one microservice) later these projects (microservices) can be integrated for single project or multiple projects..

=>Ulsing Spring, spring Boot we can develop

standalone Apps, web applications, distributed Apps and etc..

(single App) (browser --->App)

(App to App)

Enterprise Apps = web applications + distributed Apps

note:: we can develop these enterprise apps either having regular architecture nothing but developing different services as different modules of single project(Monilithic architecture) ..

or we can develop using micro service Architecture nothing but developing different services as different projects.

E-Commerce App (Monolithic Arch)

E-Commerice App

(which is becoming outdated)

note:: In web services env.. the Distributed Apps are interoperable i.e the client

and server app (or) Consumer and Producer Apps can be there either in same language/ technology/framework or can be there in two different

languages/technologies/frameworks

xml:: extensible markup language (tag based) (fading out) json:: Java Script Object Notation (key-value based) (Industry standard)

xml or can be the json

xml:: extensible markup language

json:: java script object notation

xml, json are the global formats to define the global data

for exchanging data b/w any two compitable or incompitalbe apps Compare to Xml, the JSON format

(MicroServices arch)

jar2/war2

 # Payment MS (proj2) (jar3/war3) jar4/war4) #Promotions MS # Inventory MS

Sales

(Proj#1) Payment

(jar1/war1)

 # Sales MS

Modile

Module

Inventory Module

(war/jar file)

(proj1)

Promotions

(These modules are different packages of

with one E-commece App]

E-commerice App So these fightly coupled

(proj 3) (proj4) [These Micro services are indepedent projects,So they can be he

either

in One E-commerce App or in

Multiple E-commerice Apps]

Spring boot App = 100% Code driven cfgs Approach + AutoConfiguration

is good for data exchange

Monolithic arch= Monolith arch

Micro services Project /app = multiple restful services/projects+ third party tools for integration

+ Design Patterns